Pier Luigi Farri
Giovanni Piotti
Sandro Sbroggiò

a SCUOLA *con* VISUAL BASIC 2010 EXPRESS

http://vbscuola.it
II edizione riveduta e corretta, novembre 2011.

Ringraziamento.

Gli autori desiderano ringraziare **Ambra, Giada e Claudio Gucchierato**, che si sono sobbarcati un paziente e prezioso lavoro di revisione e di controllo del testo del manuale e dei materiali allegati.

Avvisi e istruzioni.

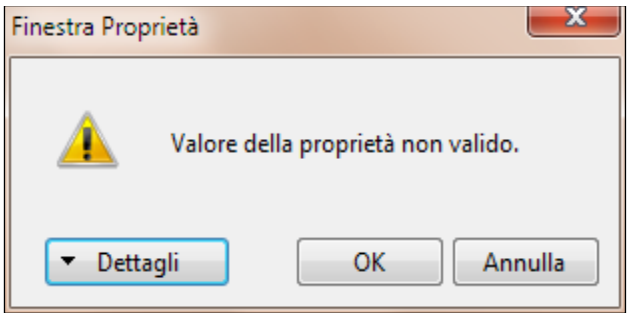
1
La sigla **VB** è usata nel testo come abbreviazione del nome completo del linguaggio di programmazione **Microsoft® Visual Basic® 2010 Express** che fa parte della suite di strumenti per programmatori denominata **Visual Studio® Express.**
Alla data di pubblicazione di questo manuale è possibile scaricare e installare **VB** gratuitamente, in versione completa, in lingua italiana, da questo indirizzo:

http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express-iso

2
All'indirizzo http://vbscuola.it/VB2010/Materiali_VB2010.exe è disponibile un file che contiene materiali di supporto necessari per lo studio di questo manuale.
Si tratta di un file eseguibile che, mandato in esecuzione, crea nella cartella **Documenti** alcune cartelle e sottocartelle con i materiali usati nel manuale.
Al termine dell'esecuzione le lettrici e i lettori troveranno dunque nella cartella **Documenti / A scuola con VB 2010** le sottocartelle contenenti:

- i materiali necessari per eseguire gli esercizi del manuale (immagini, file di testo, file audio)
- i listati degli esercizi e
- i sorgenti delle due applicazioni complete illustrate nel manuale.

3
Nelle prime fasi dei loro esercizi, le lettrici e i lettori si imbatteranno facilmente in questo messaggio di errore:



Esso compare - tra l'altro - quando si tenta di modificare le proprietà dei controlli mentre il programma è in esecuzione. Per procedere è necessario fermare l'esecuzione del programma cliccando l'icona **Termina debug**.

4

Nel testo del manuale, le parti scritte con un bordo rosso su fondo grigio sono esercizi; le parti scritte con bordo verde sono listati o righe di codice che la lettrice o il lettore possono copiare e incollare nella Finestra del Codice del programma al quale stanno lavorando:

questo è il testo di un esercizio

```
questo è il testo di un listato
```

Nel caso di problemi nelle operazioni di copia e incolla, nella cartella **Documenti \ a scuola con VB 2010 \ Esercizi** sono disponibili i testi di tutti i listati degli esercizi.

Pier Luigi Farri
Giovanni Piotti
Sandro Sbroggiò

a SCUOLA *con* VISUAL BASIC 2010 EXPRESS

Manuale di programmazione dei computer
con il linguaggio Microsoft® Visual Basic® 2010 Express
per insegnanti di scuole dell'infanzia, primarie, medie.

http://vbscuola.it
II edizione riveduta e corretta, novembre 2011.

Sommario.

Indice degli esercizi.

Indice delle tabelle.

Indice delle figure.

INTRODUZIONE.

1: Impariamo a programmare i computer.

Il computer è uno strumento per l'informatica (informazione automatica) come il televisore è uno strumento per la televisione (visione a distanza); il loro uso ha molti aspetti in comune.
Come gli spettatori davanti a un televisore si siedono per guardare programmi preparati da altri, così gli utenti dei computer usano i computer per fare cose con programmi creati da altri.
Spettatori e utenti possono agire e reagire in vari modi, e in particolare possono cambiare canale o spegnere il televisore, oppure possono cambiare il programma o spegnere il computer, ma rimangono comunque vincolati alle capacità e alle scelte dei programmatori, di coloro che creano spettacoli per la televisione o programmi per il computer.
Guardare o utilizzare programmi preparati da altri non è certamente una cosa negativa in sé, anzi: questi possono essere strumenti utili per arricchire e stimolare le conoscenze e capacità degli utenti, ma è evidente che l'attività di creazione di programmi si pone su un gradino superiore, nella scala dell'impiego dell'intelligenza umana.
Nelle scuole, negli anni dell'obbligo scolastico, gli alunni imparano nella migliore delle ipotesi a usare il computer come utenti, e rimangono relegati in questo ruolo subalterno che impedisce loro di sedere al posto di guida, di fare progetti in prima persona, di dare corpo alle loro idee.
Lungi dall'essere un'attività tediosa o superflua, imparare a programmare i computer costituisce la via maestra per avvicinarsi a queste macchine in modo attivo, usandole come strumenti per l'espressione della propria creatività; l'apprendimento della programmazione dei computer è dunque l'esperienza più significativa che possa essere fatta a scuola con queste macchine, dagli insegnanti e dai loro alunni, già a partire dagli anni della scuola dell'infanzia.

2: A chi si rivolge il manuale.

Questo manuale si propone come uno strumento concreto, operativo e immediato, per gli insegnanti delle scuole dell'infanzia, primarie e medie che intendono imparare a programmare i computer con il linguaggio VB, per creare programmi didattici e/o per realizzare esperienze di programmazione con i loro alunni.
Il manuale si rivolge a chi non ha un minimo di esperienza nell'uso dei computer e non ha alcuna precedente esperienza di programmazione.
Al termine delle lezioni e degli esercizi le lettrici e i lettori saranno in grado di realizzare e di distribuire applicazioni o programmi perfettamente funzionanti quali:

- giochi con l'impiego di immagini e suoni;
- documentazioni multimediali di esperienze;
- programmi didattici;
- materiali di supporto dell'attività didattica (ad esempio strumenti di gestione di dati e/o strumenti di preparazione, correzione e valutazione di prove standardizzate).

Nella presentazione del linguaggio VB sono state sviluppate quelle parti che più si prestano a un utilizzo immediato da parte di un lettore/programmatore alle prime armi, interessato alla realizzazione delle applicazioni elencate sopra.
Non compaiono nel manuale gli elementi di VB che si rivolgono a programmatori già esperti; vi è sviluppata ampiamente, tuttavia, la programmazione di alcune funzioni multimediali (audio, grafica, animazioni), importanti per l'ambiente scolastico cui il manuale è rivolto.
Il manuale può anche essere considerato, da parte di un programmatore alle prime armi, come uno strumento di base, da utilizzare come piattaforma per accedere successivamente agli aspetti più complessi di un linguaggio di programmazione che ormai si colloca sullo stesso piano dei linguaggi professionali.

L'esposizione del materiale in questo manuale segue un metodo particolare: la lettrice o il lettore è messo in grado sin dall'inizio di compiere esperienze concrete di programmazione nell'ambiente di VB.
Trattandosi di una guida rivolta soprattutto all'ambiente scolastico, possiamo anche citare il *tâtonnement* di **C.Freinet**[1]: il manuale fornisce per ogni argomento rapidi quadri di riferimento, all'interno dei quali la lettrice o il lettore sono stimolati a provare ed esplorare forme sempre più complesse di programmazione, con esperienze dirette, sin dalle prime pagine.
Il manuale è stato progettato per essere utilizzato anche come strumento per corsi di aggiornamento tenuti da esperti o per corsi autogestiti da gruppi di insegnanti. In questo caso, al termine del corso può essere riconosciuto ai partecipanti che abbiano

[1] C.Fréinet (1896-1966), pedagogo e educatore francese. Con le sue tecniche didattiche propugnò la trasformazione della esperienza scolastica in un'esperienza di vita in cui i bambini sviluppano, sperimentano e sistematizzano le loro conoscenze in modo attivo, in un quadro di relazione aperta con l'ambiente socio-culturale in cui vivono.

completato lo studio delle lezioni e l'esecuzione degli esercizi un impegno complessivo di 80 ore.
Il manuale può essere utilizzato dai docenti come guida per realizzare progetti didattici per l'insegnamento della programmazione agli alunni. L'ordine degli argomenti può essere rispettato; sarà cura dei docenti adeguare l'esposizione alle capacità di comprensione degli alunni.

3: Come contattare gli autori.

Gli autori **Pier Luigi Farri**, **Giovanni Piotti** e **Sandro Sbroggiò** sono, a vario titolo, esperti di programmazione e/o di didattica, collaboratori del sito **http://vbscuola.it**.
E' possibile inviare ai loro indirizzi di posta elettronica segnalazioni di errori o imprecisioni, richieste di chiarimenti, consigli, suggerimenti e materiali:

plfarri@vbscuola.it
giovanni.piotti@vbscuola.it
sandro.sbroggio@vbscuola.it

4: Requisiti per la lettrice o il lettore.

Per lo studio del manuale sono richieste alla lettrice o al lettore normali capacità di movimento in ambiente Windows:

- aprire, chiudere, spostare, ingrandire e ridurre finestre;
- usare l'editing di un elaboratore di testi (formattare un testo, scegliere il tipo e le dimensioni dei caratteri, tagliare, copiare e incollare parti di testo, inserire e dimensionare immagini);
- usare il mouse per selezionare e aprire files, spostare oggetti sullo schermo, selezionare parole, righe o paragrafi di testo, rispondere a finestre di dialogo premendo i pulsanti Sì, No, Annulla, ecc.);
- utilizzare i menu e i menu a icone;
- trovare file in collocazioni diverse;
- rinominare file;
- gestire le cartelle (creare nuove cartelle, spostarle, rinominarle);
- salvare file in collocazioni diverse;
- spostare file da una cartella all'altra.

È necessario inoltre che la lettrice o il lettore conosca il funzionamento di queste periferiche del computer:

- lettore CD- ROM;

- microfono, casse acustiche o altoparlanti;
- stampante.

5: Requisiti per il sistema.

VB è un software impegnativo per il computer e richiede una macchina con requisiti minimi di una certa consistenza:

- sistema operativo Windows XP (aggiornato all'ultimo Service Pack) o Windows 7;
- processore a 1.6 GHz o più veloce;
- memoria ram di 1024 MB o più;
- risoluzione video di 1024 x 768 pixel o maggiore.

6: Come procurarsi il linguaggio VB.

Il linguaggio VB è di proprietà della casa di software Microsoft®, basata negli Stati Uniti; esso fa parte di un gruppo di strumenti di programmazione denominata Visual Studio® 2010 Express, che comprende tre linguaggi di programmazione

- Microsoft Visual Basic® 2010 Express;
- Microsoft C#® 2010 Express;
- Microsoft C++® 2010 Express;

e il software per creare siti internet

- Microsoft Web Developer® 2010 Express.

Alla data di pubblicazione di questo manuale VB può essere installato gratuitamente in versione completa, in lingua italiana[2], collegandosi a questo indirizzo:

http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-basic-express

Da questa pagina si avvia l'installazione a distanza (per scegliere la lingua italiana cliccare il menu a tendina che si apre a destra nella pagina):

[2] "*in lingua italiana*" significa che si troveranno in italiano tutti gli elementi che fungono da *involucro* a VB, cioè l'interfaccia utente con i suoi menu, le finestre, le barre degli strumenti.
Il linguaggio vero e proprio è unico, identico per i programmatori di tutto il mondo, ed è costituito da circa 200 termini di uso comune nella lingua inglese.

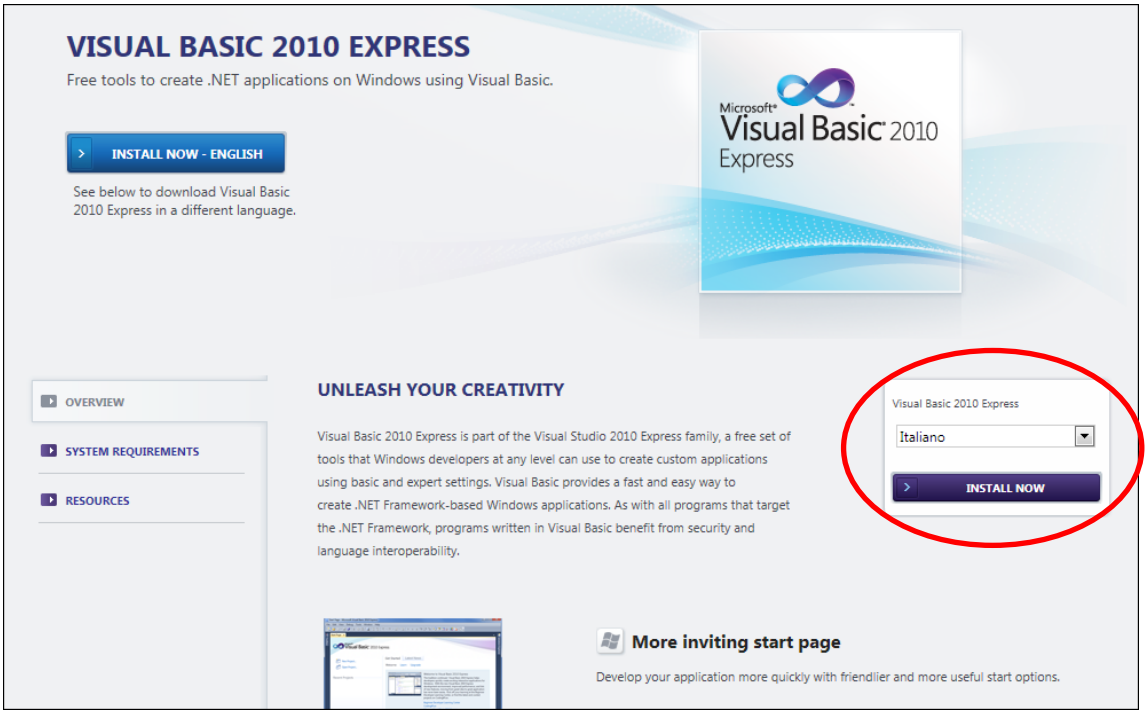


Figura 1: La pagina per l'installazione a distanza di VB, dal sito Microsoft.com.

In alternativa all'installazione a distanza (che non consente di avere una copia di sicurezza del software installato), da questa pagina è possibile scaricare un file in formato .iso (cioè un file *immagine* di DVD-ROM) dal quale si può ricavare un DVD-ROM da utilizzare per installare VB, e da conservare per ripetere l'installazione, in caso di necessità:

http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express-iso

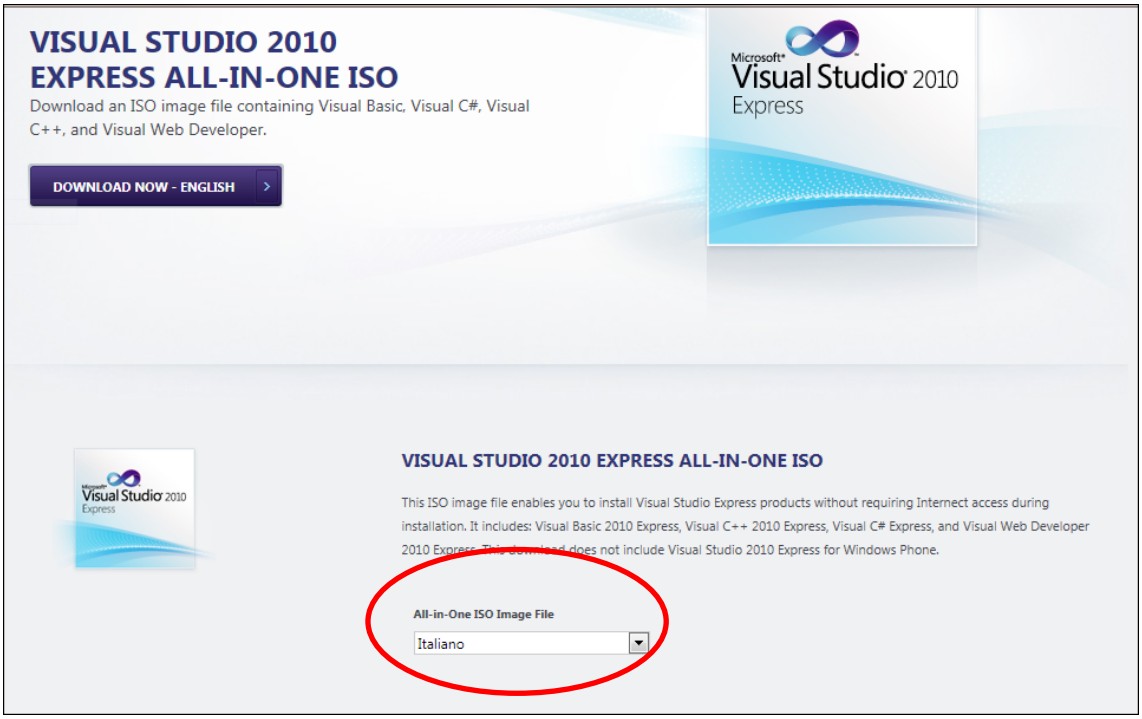


Figura 2: La pagina con il download di VB dal sito Microsoft.com.

Prima di scaricare il file in formato .iso, selezionare la lingua italiana cliccando il menu a tendina che compare in basso nella pagina.
Il DVD-ROM così creato comprende tutti gli strumenti della famiglia Visual Studio 2010 Express: Visual Basic 2010 Express, Visual C++ 2010 Express, Visual C# Express, e Visual Web Developer 2010 Express.

7: L'installazione di VB.

Prima di procedere alla installazione di VB - a distanza o da DVD- ROM - è necessario disinstallare ogni versione *beta* dello stesso linguaggio eventualmente presente sul computer; è consigliabile - ma non necessario - disinstallare le versioni precedenti di VB (VB 2005, VB 2008).
Non è necessario disinstallare Visual Basic 6 o le versioni precedenti a Visual Basic 6 eventualmente presenti sul computer, in quanto queste versioni sono un ambiente di progettazione diverso e possono continuare a funzionare anche in presenza di VB 2010.
Apriamo qui un inciso indirizzato a chi conosce il linguaggio di programmazione Visual Basic 6: **VB 2010 non è un aggiornamento di VB6**, per cui le conoscenze già acquisite con VB6 non possono essere trasferite automaticamente in VB 2010. Certamente la pratica della programmazione con VB6 è di grande aiuto a chi inizia lo studio di VB 2010, ma VB 2010 è un ambiente di progettazione diverso, che opera con nuove forme

di linguaggio e con concetti nuovi, basandosi sull'archivio di software **Microsoft.NET FrameWork 4,** che ai tempi di VB6 non esisteva.
L'archivio **Microsoft.NET FrameWork 4** è un po' *il pozzo di San Patrizio* al quale attingono tutti e tre i linguaggi della suite Visual Studio: Visual Basic, C# e C++. Nella piattaforma .NET i tre linguaggi hanno una radice comune che li colloca tutti e tre sullo stesso piano, cosa che non accadeva con il linguaggio Visual Basic che sino alla versione 6 era considerato quasi un *linguaggio giocattolo,* parente povero del linguaggio C.

Nella fase di installazione, si abbia cura di verificare che sia installato anche il software **Microsoft SQL Server 2008 Express SP1**: si tratta di uno strumento per creare e gestire archivi di dati in VB la cui installazione è facoltativa:

Figura 3: L'installazione di Microsoft SQL Server 2008 Express SP1.

Al termine della installazione, al primo utilizzo di VB, l'utente viene informato che ha 30 giorni di tempo per registrare il prodotto.
Per effettuare la registrazione è richiesto il possesso di un ***Windows Live ID***, cioè di un indirizzo email valido del tipo

tizio@hotmail.com
caio@msn.com
sempronio@live.com

Se non si dispone di uno di questi indirizzi lo si può creare contestualmente alla richiesta della registrazione di VB.
L'eventuale richiesta del Windows Live ID e la registrazione di VB sono gratuite.
Una volta completata la registrazione, l'utente può disporre di VB gratuitamente, a tempo indeterminato.

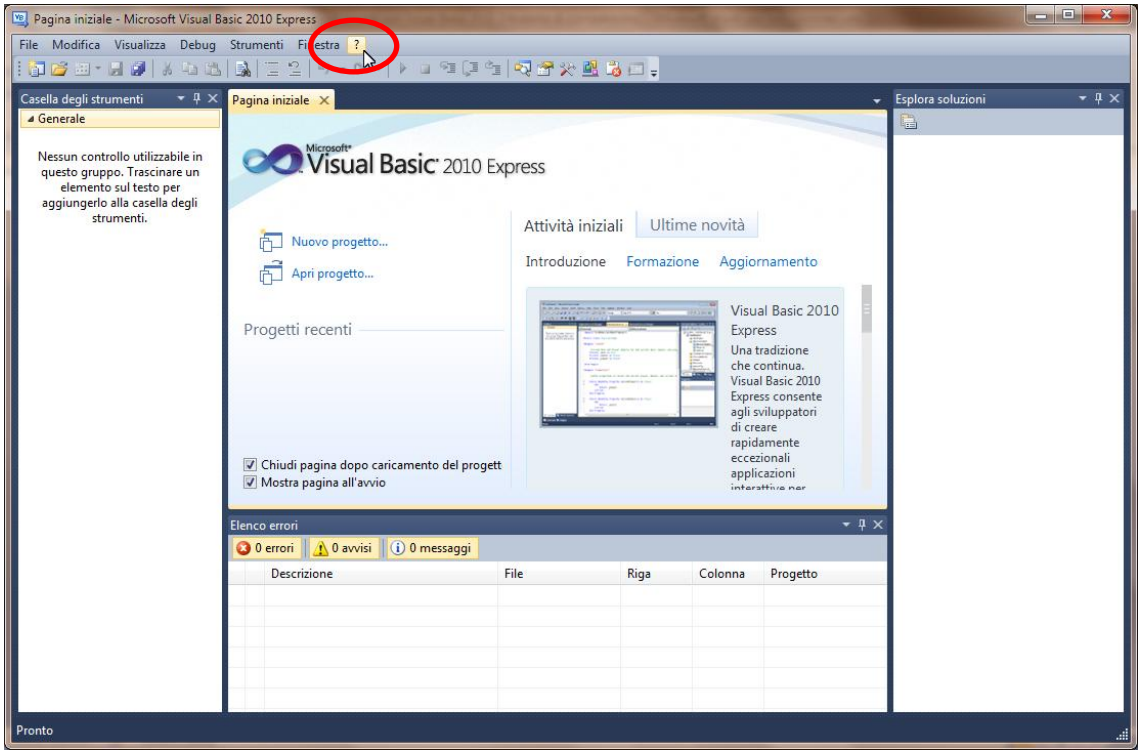


Figura 4: Il menu per procedere alla registrazione di VB.

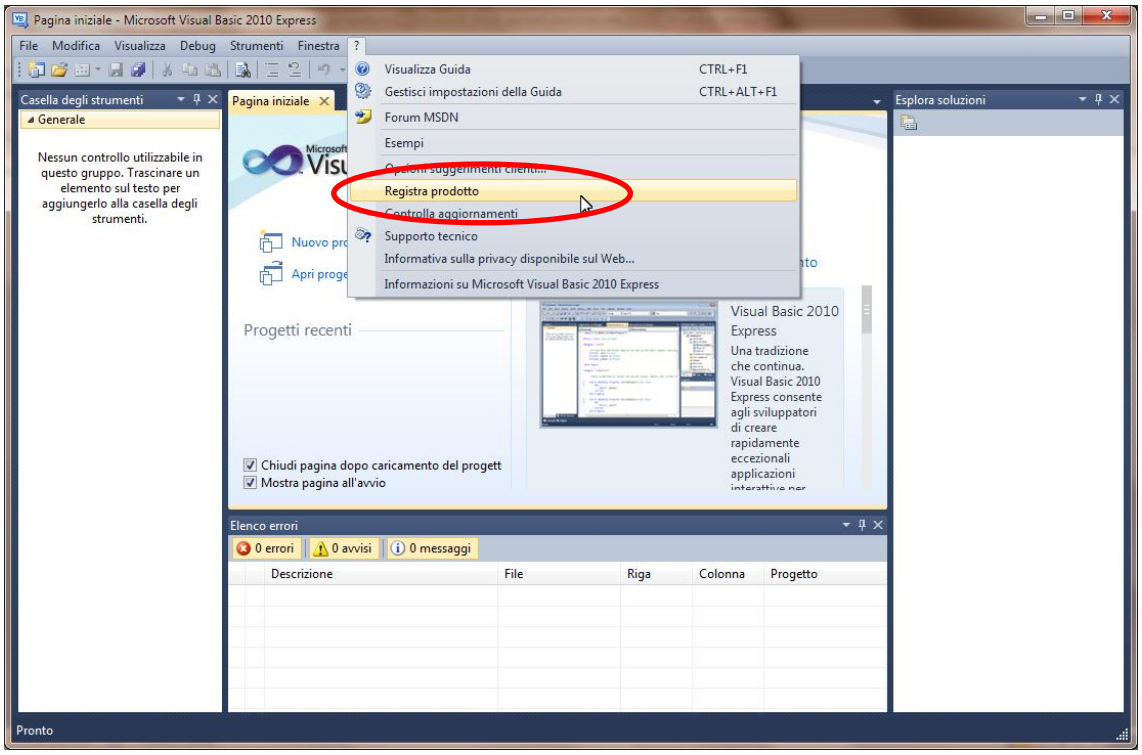


Figura 5: L'avvio della procedura di registrazione di VB.

Al termine della installazione, nella cartella "**Documenti**" del computer troviamo la nuova cartella "**Visual Studio 2010**". La cartella "**Visual Studio 2010**" contiene a sua volta alcune sottocartelle, tra cui la sottocartella "**Projects**" dove VB salverà i nostri futuri lavori, in modo automatico, ognuno in una sottocartella diversa,.

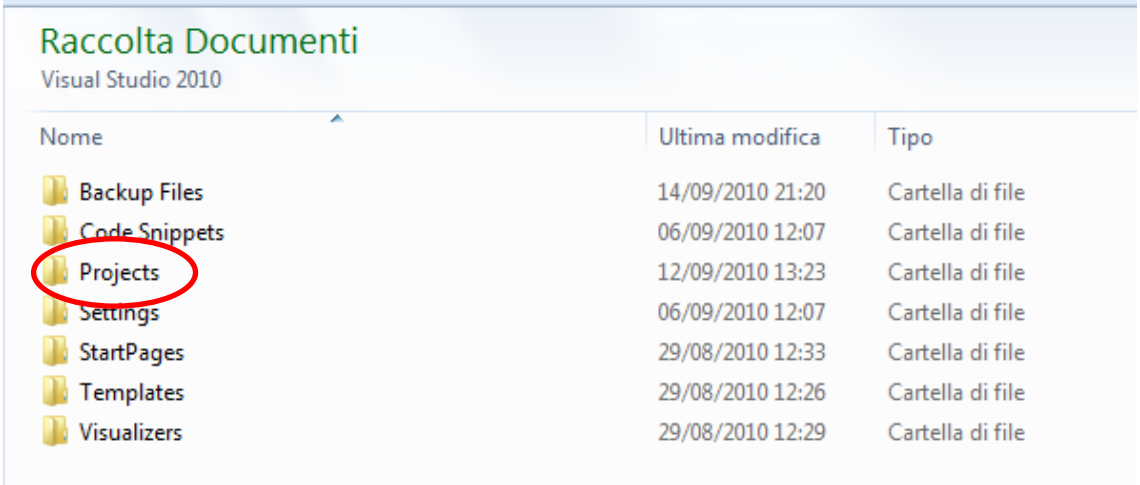


Figura 6: Il contenuto della cartella "Documenti/Visual Studio 2010" dopo l'installazione di VB.

8: Materiali di supporto.

All'indirizzo http://vbscuola.it/VB2010/Materiali_VB2010.exe è disponibile un file che contiene materiali di supporto necessari per lo studio di questo manuale.
Si tratta di un file eseguibile che, mandato in esecuzione, crea nella cartella **Documenti** alcune cartelle e sottocartelle con i materiali usati nel manuale.
Al termine dell'esecuzione le lettrici e i lettori troveranno dunque nella cartella **Documenti / A scuola con VB 2010** le sottocartelle contenenti:

- i materiali necessari per eseguire gli esercizi del manuale (immagini, file di testo, file audio)
- i listati degli esercizi e
- i sorgenti delle due applicazioni complete illustrate nel manuale.

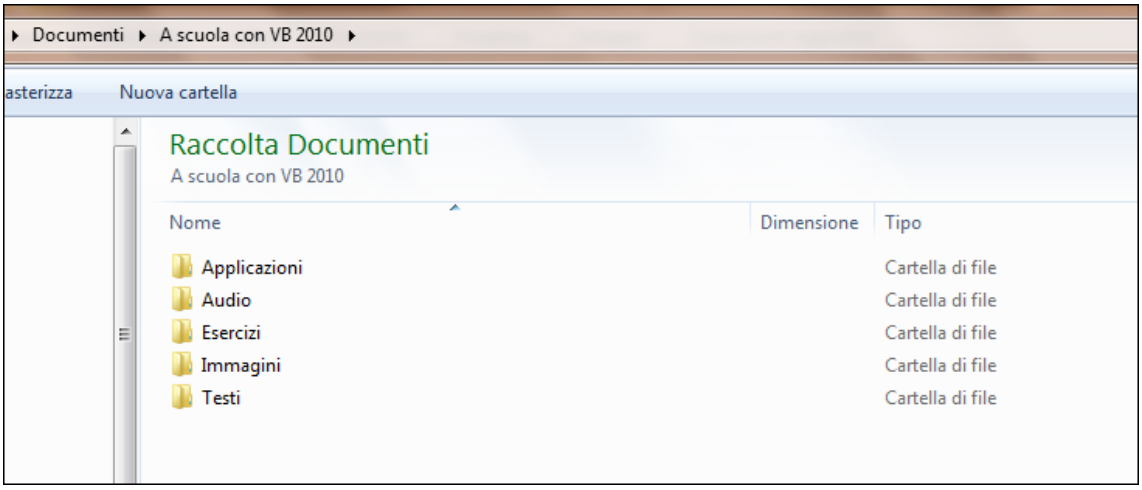


Figura 7: Il contenuto della cartella Documenti / A scuola con VB, con i materiali di supporto allo studio del manuale.

“

PARTE I: LA PROGRAMMAZIONE: POTENZIALITÁ FORMATIVE, EVOLUZIONE STORICA, CARATTERISTICHE ODIERNE.

“[…] la scuola deve insegnare a programmare, e non soltanto a utilizzare i programmi. Ho incominciato a fare avvicinare i miei studenti al computer nel 1983. Allora bisognava programmare, e dunque pensare con la logica del computer. I miei studenti di allora oggi inventano software. Mentre quelli che sono venuti dopo, con ambienti operativi certamente più amichevoli (come Windows), si limitano spesso a rispondere sì o no, e a cliccare sulle icone. Mangiano, ma non sanno che cosa c'è nel cibo […]
Se si insegna a un bambino a programmare in qualche linguaggio informatico, questo esercizio logico lo renderà padrone e non schiavo del computer".

U. Eco (intervista al quotidiano “La Repubblica”, 8 gennaio 2000).

Capitolo 1: PERCHE' IMPARARE A PROGRAMMARE I COMPUTER?

Un computer è una macchina che esegue delle istruzioni. Queste istruzioni sono scritte dai programmatori e sono contenute nei programmi (o applicazioni[3]) che si trovano sul disco fisso del computer o in una *rete* di macchine di cui il computer fa parte.
Un programma (o applicazione) consiste in una serie di istruzioni impartite al computer per fargli eseguire determinati compiti.
Alcuni programmi sono relativamente semplici come, ad esempio, la calcolatrice che compare nel menu Start / Programmi / Accessori di Windows. Per quanto possano apparire complesse le operazioni svolte da questa calcolatrice, la sua realizzazione è piuttosto semplice.
Altri programmi sono estremamente complessi e possono essere costruiti solo da équipes di programmatori:

- gli elaboratori di testo (ad esempio Microsoft Word);
- i fogli elettronici (ad esempio Microsoft Excel);
- i data base (ad esempio Microsoft Access);
- i programmi per la gestione della grafica (ad esempio Corel Draw, Adobe Photo Shop);
- i programmi per la navigazione in Internet e per la gestione della posta elettronica.

Esistono in commercio, oppure sono disponibili in rete, migliaia di applicazioni o di programmi in grado di soddisfare le esigenze di tutti gli utenti e di tutte le tasche.
A che serve dunque imparare a programmare?

9: Personalizzare l'uso del computer.

Per quanto molteplici e diversi siano i programmi disponibili, nessuno di essi è confezionato su misura per soddisfare le esigenze e le abitudini di ogni singolo utente. Destinati al maggior numero possibile di utenti, questi programmi cercano di

[3] I termini *programma* e *applicazione* sono sinonimi. Alcuni tendono a definire *applicazione* un *progetto* più complesso, che comprende al suo interno diversi *programmi* finiti a sé stanti, ma questa distinzione è poco chiara e non è entrata nell'uso comune.
Non vanno invece confusi i termini *programma* e *linguaggio di programmazione*. Un linguaggio di programmazione è uno strumento che consente di realizzare programmi: VB è un **linguaggio di programmazione**, un software sulle tabelline, invece, è un **programma** realizzato con VB o con un altro linguaggio di programmazione.

abbracciare le esigenze di un utente *standard i*mmaginario, racchiudendo le possibilità di personalizzazione nelle opzioni secondarie.
Chi ha provato a utilizzare i programmi per la gestione delle finanze personali se ne è certamente reso conto: poiché ognuno ha un modo strettamente personale di vedere e di gestire la propria situazione finanziaria (senza considerare che le situazioni finanziarie sono diverse da persona a persona), è pressoché impossibile trovare un programma che corrisponda perfettamente alle proprie esigenze.
A scuola, si può presentare l'esigenza di documentare una esperienza, oppure di realizzare un progetto didattico informatico insieme agli alunni, o ancora di creare una storia multimediale, o un gioco inventato dagli alunni, con immagini e suoni creati da loro… In questi casi nessun programma esistente può venire in aiuto: per realizzare questi progetti è necessario **fare da sé**, imparando almeno gli elementi basilari della programmazione del computer.

10: Educare la precisione e il rigore logico.

L'apprendimento della programmazione ha una forte valenza formativa per tutti (bambini e adulti, alunni e insegnanti), paragonabile per molti aspetti a quella tradizionalmente attribuita allo studio del latino.
Per ideare e creare un programma che funzioni senza problemi in tutto il suo percorso, è necessario impartire al computer delle istruzioni precise, in cui non vi siano

- errori di sintassi (altrimenti il programma arrivato a quel punto si blocca segnalando l'errore);
- errori di logica, imprecisioni o ambiguità (altrimenti il programma può produrre risultati diversi da quelli voluti dal programmatore).

Gli **errori di sintassi** sono gli errori nei quali il programmatore può incappare nella fase di scrittura dei comandi: si tratta in genere di errori di ortografia nella scrittura del codice o nella sintassi delle istruzioni.
La forma dei comandi che vengono impartiti al computer all'interno di un programma è rigidissima, in quanto per essere riconosciuti dal computer questi comandi possono essere scelti esclusivamente tra quelli previsti dal linguaggio di programmazione, e debbono essere scritti esattamente come richiesto dal linguaggio di programmazione. Un errore di scrittura, commesso perché si è battuto malamente un tasto o perché non si conosce bene il comando che si vuole impartire, genera un errore di sintassi e il blocco del programma: il computer termina il programma nel punto in cui riscontra l'errore e segnala l'errore al programmatore.
Ad esempio, sono errori di sintassi:

- scrivere la proprietà **Label1.Txt** invece di **Label1.Text**;
- scrivere un testo senza le due virgolette: **Label1.Text = "Garibaldi**, invece di **Label1.Text = "Garibaldi"**;

- sbagliare i rimandi da una parte all'altra del programma (ad esempio ordinare al computer di andare ad eseguire la procedura **Stampatesto** che invece in altra parte del programma è stata chiamata **Stampa_testo**).

Gli errori di sintassi sono captati e segnalati immediatamente da **IntelliSense**, uno strumento incorporato in **VB** con il compito di suggerire al programmatore il completamento di parole e la correzione di errori di battitura.
Ecco, ad esempio, come **IntelliSense** segnalerebbe al programmatore i tre tipi di errori di sintassi che abbiamo visto sopra:

```
    Private Sub Form1_Load() Handles MyBase.Load

        Label.txt = "CIAO"

        Label1.Text = "Garibaldi"

        GoTo StampaTesto

Stampa_Testo:

    End Sub
```

Figura 8: La segnalazione di errori di sintassi da parte di IntelliSense.

Notiamo che il primo e il terzo errore sono sottolineati con una linea ondulata. Portando il mouse su queste linee, il programmatore può leggere informazioni sugli errori e suggerimenti per la loro correzione.
Il secondo errore, invece, è corretto automaticamente da **IntelliSense**, che completa il testo *"Garibaldi* aggiungendo la virgoletta che manca.

Anche l'omissione o il cambiamento di semplici segni di interpunzione possono causare effetti del tutto indesiderati nell'esecuzione di un programma.
Ecco un esempio.
In questo programma compare un **Form** (contenitore) denominato **Form1**. All'interno del **Form1** compaiono sei controlli **Label** (etichette) denominati rispettivamente **Label1**, **Label2**, **Label3**, **Label4**, **Label5**, **Label6**:

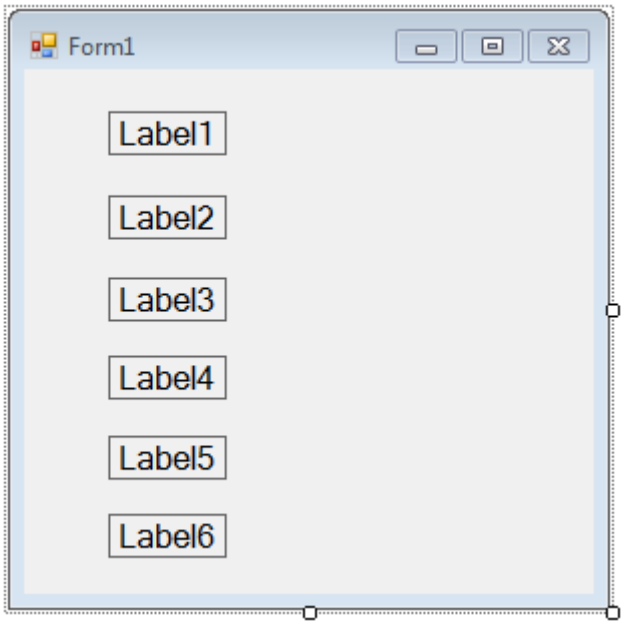


Figura 9: Programma di esempio in fase di progettazione.

All'avvio del programma, cioè al caricamento nella memoria del computer del Form1, vengono eseguite queste sei linee di codice, apparentemente piuttosto simili, finalizzate a scrivere una parola all'interno di ogni Label:

```
    Private Sub Form1_Load() Handles MyBase.Load

        Label1.Text = "CIAO"
        Label2.Text = "'CIAO"
        'Label3.Text = "CIAO"
        Label4.Text = CIAO
        Label5.Text = "CIA"
        Label6.Text = CIA

    End Sub
```

Mandando in esecuzione il programma si ottiene questo risultato:

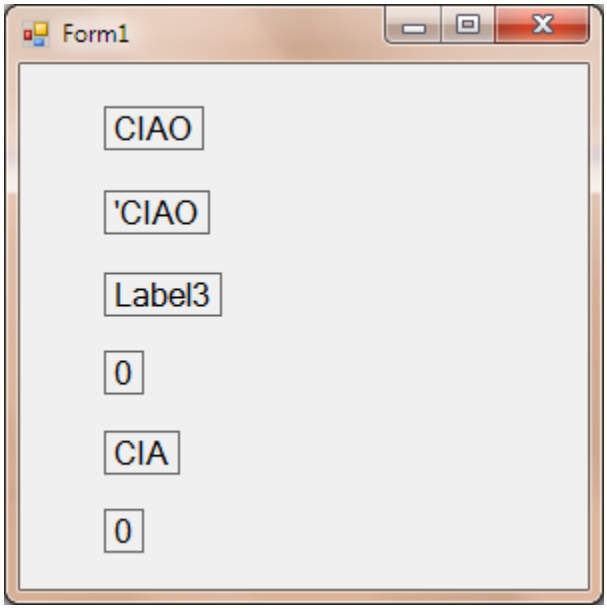


Figura 10: Programma di esempio in fase di esecuzione.

Vediamo come VB ha eseguito i comandi:

- Nella **Label1** è visualizzata correttamente la parola "CIAO" che era stata scritta tra due virgolette.
- Nella **Label2** è visualizzata la parola "'CIAO" con l'apostrofo iniziale perché tutto quanto è scritto tra virgolette, apostrofo compreso, viene considerato dal programma come **testo**.
- Nella **Label3** non è visualizzata alcuna parola perché l'apostrofo posto all'inizio della riga indica al programma che tutto quanto segue sono *annotazioni* del programmatore di cui VB non deve tenere conto.
- La parola CIAO assegnata alla **Label4**, senza virgolette, è interpretata da VB come una segnaposto che sta al posto di un numero. In questo caso, non trovando alcun numero assegnato a CIAO, VB scrive il numero 0.
- Nella **Label5** è visualizzata la parola "CIA". Come per la Label2, VB interpreta questa parola scritta tra virgolette come **testo** e la riporta tale quale, ovviamente senza chiedersi se essa abbia un senso oppure no.
- Come per la Label4, la parola CIA assegnata alla **Label6**, senza virgolette, è interpretata da VB come un segnaposto che sta al posto di un numero. Anche in questo caso, non trovando alcun numero assegnato a CIA, VB scrive il numero 0.

Se gli errori di sintassi sono intercettati da **IntelliSense** e segnalati al programmatore già durante la fase di scrittura del codice di un programma, per cui è facile individuarli e correggerli, tutt'altro discorso va fatto per gli errori di logica.
Gli **errori di logica**, o errori **semantici**, sono errori che si annidano nello schema di svolgimento di un programma: non ne inficiano il corretto funzionamento ma producono effetti e risultati non previsti dal programmatore.
Se i comandi sono stati scritti in modo ineccepibile dal punto di vista formale, il programma li esegue dal primo all'ultimo senza bloccarsi e senza segnalare problemi,

ma se nel percorso vi sono errori di logica il risultato finale non sarà quello voluto dal programmatore.
Questi errori, dunque, non si annidano nella scrittura del programma, che **IntelliSense** può vedere e correggere, ma nel progetto ideato dal programmatore, quando questi ha messo a punto gli obiettivi del suo programma e i percorsi idonei a conseguire quegli obiettivi.
Per questo motivo, gli errori di logica sfuggono a **IntelliSense**; la loro individuazione e correzione richiedono un paziente lavoro di analisi e di revisione del lavoro del programmatore.

Ecco un esempio di errore di logica: un programmatore vuole progettare un programma che, dopo avere confrontato tra loro tre numeri X, Y, Z, dica qual è il maggiore tra essi.
Il programmatore assegna al programma questo percorso:

1. Assegna un numero casuale da 1 a 100 alla variabile X;
2. Assegna un numero casuale da 1 a 100 alla variabile Y;
3. Assegna un numero casuale da 1 a 100 alla variabile Z;
4. Confronta il numero X con il numero Y:
5. Se il numero X è maggiore di Y, allora X è il maggiore dei tre numeri;
6. Se il numero X è minore di Y, allora Y è il maggiore dei tre numeri;
7. Fine del programma.

Uno schema di questo tipo, se scritto senza errori di sintassi, viene eseguito senza problemi dal programma, dall'inizio alla fine, ma l'esito finale può essere corretto o sbagliato, in quanto il percorso non prevede nessun confronto di X e Y con la variabile Z.
Il percorso logicamente corretto è invece questo:

1. Assegna un numero casuale da 1 a 100 alla variabile X;
2. Assegna un numero casuale da 1 a 100 alla variabile Y;
3. Assegna un numero casuale da 1 a 100 alla variabile Z;
4. Confronta il numero X con il numero Y:
5. Se il numero X è maggiore di Y, allora:
    a. Confronta X e Z:
    b. Se X è maggiore di Z, allora X è il maggiore tra i tre numeri;
    c. Se X è minore di Z, allora Z è il maggiore tra i tre numeri;
6. Se il numero X è minore di Y, allora:
7. Confronta Y e Z:
8. Se Y è maggiore di Z, allora Y è il maggiore tra i tre numeri;
9. Se Y è minore di Z, allora Z è il maggiore tra i tre numeri.
10. Fine del programma.

Nell'esercizio seguente, proponiamo alla lettrice o al lettore la ricerca dell'errore di logica che si annida nello schema di un programma sul gioco della tombola.

Esercizio 1: Gioco della Tombola.

Un programmatore ha messo a punto lo schema di un gioco elettronico che deve simulare - sullo schermo del monitor - l'estrazione dei 90 numeri del gioco della tombola.
Si dà per scontato che i giocatori hanno a disposizione le tradizionali cartelle di cartoncino, sulle quali segnano a mano i numeri già usciti.

Il programma fa queste cose:
- estrae, uno alla volta, i numeri da 1 a 90;
- mostra, a richiesta, tutti i numeri già usciti.

Il programmatore ha inserito, sullo schermo, quattro grandi pulsanti[4] sui quali è possibile fare *clic* con il mouse:
- sul primo pulsante è scritto: "ESTRAI UN NUMERO"
- sul secondo pulsante è scritto: "MOSTRA LA TABELLA"
- sul terzo pulsante è scritto: "INIZIA UNA NUOVA PARTITA"
- sul quarto pulsante è scritto: "TERMINA IL GIOCO"

Lo schema di programmazione del gioco è diviso in quattro parti (una parte per ognuno dei quattro pulsanti):

1. Se viene premuto il pulsante "ESTRAI UN NUMERO": sorteggia un numero a caso da 1 a 90 e mostralo sul monitor.
2. Se viene premuto il pulsante "MOSTRA LA TABELLA": mostra una tabella con tutti i numeri già usciti.
3. Se viene premuto il pulsante "INIZIA UNA NUOVA PARTITA": ripulisci la tabella dei numeri usciti.
4. Se viene premuto il pulsante "TERMINA IL GIOCO": termina il programma.

Il computer esegue questo schema senza problemi ma, a causa di un errore logico, il funzionamento del programma non è per niente soddisfacente.
Alla lettrice o al lettore il compito di trovare questo errore.[5]

11: Sviluppare il pensiero progettuale.

Dall'esercizio precedente si intuisce come le diverse fasi di elaborazione di un programma sollecitino lo sviluppo di capacità intellettuali fondamentali:
- capacità di definire problemi e di porsi degli obiettivi;

[4] Questi pulsanti sono quattro *controlli*, e più precisamente quattro Button.
[5] La soluzione si trova alla fine di questo capitolo.

- capacità di formulare ipotesi e di elaborare percorsi per risolvere i problemi e per raggiungere gli obiettivi;
- capacità di tenere sotto controllo dei processi basati su elementi e componenti variabili;
- capacità di verificare i risultati raggiunti.

Grazie al quadro fortemente motivante (il fascino del computer e del progresso tecnologico, il piacere estetico e l'aspetto ludico della multimedialità) le acquisizioni di queste capacità avvengono in modo rapido e forniscono una spinta formidabile al consolidamento di nuovi abiti mentali improntati alla precisione formale, al rigore logico, alla essenzialità delle procedure, con ricadute positive su ogni campo di attività dell'individuo.
Siamo dunque di fronte a un potenziale educativo di grande rilevanza, che non può essere ignorato dagli insegnanti:

- perché è in perfetta sintonia con i loro obiettivi formativi;
- perché è difficilmente riscontrabile in modo così accessibile, completo e organico in altre attività didattiche.

"Si tratta di conoscenze importanti sia in sé, in quanto la scrittura di un "vero" programma (intendo un programma scritto veramente e non montato con pochi pezzi prefabbricati) aiuta lo sviluppo delle capacità logiche, sia come strumento utilizzabile per raggiungere vari obiettivi scientifici. La possibilità di usare un linguaggio di programmazione, anche elementare, permette infatti di esplorare della "fenomenologia matematica" altrimenti inaccessibile e di simulare fenomeni di varia natura.[6]"

Tutto questo senza dimenticare che, in definitiva, la motivazione più forte per imparare a programmare i computer consiste nel fatto che ***creare programmi è una attività piacevole,*** perché dalla realizzazione di un programma provengono quella soddisfazione e quel senso di appagamento che accompagnano le nostre imprese, quando sentiamo che stiamo realizzando una cosa che porterà il marchio della nostra personalità.

12: Evitare la segregazione digitale.

Da alcuni anni si parla a livello internazionale di un *digital divide* (segregazione digitale) che incombe come nuova discriminante per la stratificazione sociale, in base alla possibilità ed alla capacità di accedere alle nuove tecnologie.
Nel nostro paese il problema è stato avvertito da tempo dal CENSIS:

[6] L.Russo, *Segmenti e bastoncini, Dove sta andando la scuola?*, Feltrinelli, Milano, 1998, pag. 49.

"Nella società dell'informazione e della globalizzazione la pratica del computer e la conoscenza delle lingue, dopo i tradizionali "leggere, scrivere e far di conto", costituiscono i saperi che preservano da nuove, ma forse anche più pericolose, forme di marginalizzazione. (…)
Queste conoscenze e competenze rappresentano nuovi saperi di cittadinanza, saperi cioè che determinano la possibilità di integrarsi e partecipare alle dinamiche di una società dove, negli ultimi anni, si è diffuso capillarmente l'utilizzo delle tecnologie informatiche (si pensi a internet, al moltiplicarsi degli sportelli informatici on line, al telelavoro, ecc.) e che si è aperta a una dimensione internazionale nelle sue relazioni economiche, politiche, istituzionali e monetarie."[7]

Lo studio di VB può essere uno strumento di prevenzione contro l'insorgere di questa segregazione digitale. Segregazione che non è determinata tanto dalla **capacità** di accesso all'uso del computer, quanto dalla **qualità** di questo accesso: la nuova barriera sociale che si intravvede non sarà tanto tra chi saprà usare il computer e chi non ne sarà capace ma tra chi saprà usare il computer in modo attivo e chi invece non saprà andare oltre l'approccio in modo passivo, per svolgere attività preordinate da altri.
Apriamo una parentesi per chiarire questo punto.
L'approccio all'informatica ed all'uso del computer, da parte degli adulti così come da parte dei bambini, non ha un valore intrinseco uniforme.
Come ci sono modi diversi di leggere, per cui la lettura ha livelli diversi di qualità e di utilità per la formazione delle persone, così ci sono modi diversi di usare i computer, graduabili **in relazione al livello di impegno attivo** da essi richiesto.

Vediamo un esempio. Immaginiamo di vedere un autobus; intorno a questo autobus ci sono un gruppo di persone:
- alcune di queste persone hanno costruito le parti fondamentali della macchina;
- alcune di queste persone hanno costruito gli strumenti perchè l'autobus possa essere usato dal guidatore e dai passeggeri (volante, cambio, freni, poltrone, impianto di condizionamento dell'aria, …);
- alcune di queste persone guidano l'autobus;
- alcune di queste persone salgono sull'autobus perché hanno una méta precisa da raggiungere;
- alcune di queste persone salgono sull'autobus perché desiderano *fare un giro*.

Sono dunque cinque gruppi di persone che hanno a che fare con l'autobus in modi diversi.
Attorno a un computer, possiamo immaginare gli stessi gruppi di persone:
- i fabbricanti delle macchine, coloro che costruiscono e progettano le attrezzature hardware, ricercando e mettendo alla prova componenti sempre più veloci, potenti e sofisticati;
- i creatori dei sistemi operativi e dei linguaggi di programmazione che rendono utilizzabile la macchina computer;

[7] CENSIS, XXXI Rapporto sulla situazione sociale del paese, Franco Angeli editore, 1997, pag. 124.

- i programmatori di applicazioni, che utilizzano gli strumenti prodotti ai livelli precedenti (le macchine con i sistemi operativi e i linguaggi di programmazione) per costruire applicazioni per scrivere, disegnare, salvare e ricercare dati, navigare in internet, …
- coloro che sanno utilizzare in modo attivo le applicazioni prodotte ai livelli precedenti e, all'interno di queste, riescono anche a ottenere risultati originali (gli utenti di un elaboratore di testi usano in modo passivo le opportunità offerte del programma di elaborazione dei testi, ma il loro lavoro di elaborazione di parole e immagini è un lavoro attivo che può esprimere alti livelli di creatività);
- coloro che utilizzano applicazioni prodotte da altri con l'unico intento di intrattenere l'utente o di semplificarne al massimo l'interazione con il computer. Queste applicazioni relegano l'utente in un ruolo passivo di esecuzione: si tratta, ad esempio, di video-giochi iterativi, di CD-ROM di consultazione, la cui *navigazione* non va oltre uno sfarfalleggiamento privo di obiettivi e di reale partecipazione da parte dell'utente.

Questi cinque gruppi di persone hanno un approccio diverso al computer, un approccio che va da un livello massimo a un livello minimo di impegno attivo.
I primi due gradi della scala sono riservati ad équipes di ricercatori professionisti, inseriti in organizzazioni economiche in grado di supportare i costi della loro attività, in una prospettiva di vendite del loro prodotto su scala mondiale.
Alle lettrici e ai lettori di questo manuale proponiamo un obiettivo più limitato, ma ugualmente ambizioso: assestarsi al terzo grado della nostra scala, acquisendo

- le conoscenze di base per capire come funziona la programmazione dei computer,
- le capacità necessarie per **imparare** a realizzare semplici applicazioni e
- le capacità necessarie per **insegnare** a realizzare semplici applicazioni.

L'errore logico nell'esercizio a pag. 52, sul gioco della Tombola, si trova nella progettazione degli effetti del primo pulsante: "ESTRAI UN NUMERO".
Non è sufficiente assegnare al programma il compito di estrarre un numero da 1 a 90.
E' necessario inserire le istruzioni perché il computer controlli se il numero estratto è già stato sorteggiato in precedenza, per evitare che lo stesso numero venga sorteggiato più volte.
Le istruzioni corrette e complete sono queste:

1. *estrai un numero **X** da 1 a 90;*
2. *se il numero **X** è già stato estratto in precedenza, allora:*
3. *torna al punto 1 ed estrai un altro numero;*
4. *se il numero **X** non è già stato estratto in precedenza, allora:*
5. *mostra **X** sul monitor e*

6. *memorizza **X** tra i numeri già estratti.*

Capitolo 2: I LINGUAGGI PER LA PROGRAMMAZIONE DEI COMPUTER.

I computer operano solo in base al cosiddetto linguaggio macchina (o **codice nativo**).
Nel linguaggio macchina le istruzioni sono scritte in questo modo:

0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1

Figura 11: Rappresentazione grafica di una sequenza di bit.

Ogni casella rappresenta una unità minima di memoria del computer, nella quale è contenuta una unità minima di informazione: un **bit**. Bit sta per **BI**nary digi**T** (numero binario). Un **bit** può contenere solo due numeri: il numero 0 o il numero 1.
Un bit può essere immaginato come una *lampadina* che può essere spenta (numero 0) o accesa (numero 1).
L'accendersi o lo spegnersi di queste lampadine sono le uniche istruzioni che i computer *capiscono* e interpretano.
Un programmatore che voglia parlare direttamente al computer deve quindi conoscere questo linguaggio particolare, formato solo da due segni, per impartire al computer ogni tipo di istruzione, come ad esempio:

- riconoscere che è stato premuto sulla tastiera il tasto corrispondente a una lettera (ad esempio la lettera "A") e, di conseguenza,
- scrivere la lettera A sullo schermo del monitor.

13: Le unità di memoria del computer.

I **bit** sono raggruppati in matrici di 8 **bit** (nella figura precedente ogni riga rappresenta una matrice di 8 **bit**).
Una matrice di 8 **bit** si chiama **byte** (pronuncia: bàit):

0	1	0	1	0	1	0	1

Il **byte** è l'elemento fondamentale per la programmazione del computer. A seconda della diversa forma (0 o 1) che assumono le 8 lampadine che lo compongono, un **byte** può assumere 256 forme diverse, corrispondenti a 2^8 disposizioni di 1 o 0. Un programmatore di computer lavora quindi combinando tra di loro 256 elementi di programmazione.
Si tratta, in pratica, di un alfabeto composto da 256 segni, numerati da 0 a 255, con i numeri scritti in base 2.

Il **byte** è anche l'unità di misura della memoria del computer. I suoi multipli sono:
- il **Kb** (Kilobyte = 2^{10} = 1.024 byte)
- il **Mb** (Megabyte = 2^{20} = 1.024 Kb)
- il **Gb** (Gigabyte = 2^{30} = 1.024 Mb).

Alcune centinaia di Kb possono contenere:
- un libro (e-book) di alcune centinaia di pagine;
- un'immagine di dimensioni ridotte;
- un suono.

Un Mb può contenere:
- un libro (e-book) di 1.000 pagine, molto elaborato;
- un'immagine di grandi dimensioni salvata in alta definizione;
- una breve musica salvata in formato .mp3.

VB occupa circa 600 Mb sul disco fisso del computer.
La memoria RAM[8] di un normale computer sul quale si possa usare **VB** deve disporre di almeno 1 o 2 Gb (cioè 1.000 o 2.000 Mb).
Il disco fisso di un personal computer moderno contiene alcune centinaia di Gb.

[8] La memoria RAM (*Random Access Memory*, memoria ad accesso casuale) è la memoria che il computer utilizza nello svolgimento della sua attività quando è acceso (per l'elaborazione di testi, di giochi, di suoni, di immagini); per questo motivo è chiamata anche memoria di lavoro. Si differenzia dalla memoria ROM (*Read Only Memory*) che è invece costituita dalle informazioni salvate su supporti come il disco fisso del computer o i CD-ROM. Il contenuto della memoria RAM, a differenza della memoria ROM, si perde con lo spegnimento del computer.

14: I linguaggi compilatori.

Il linguaggio macchina, composto di bit e di byte, è un linguaggio detto di primo livello, perché in rapporto diretto con le componenti fisiche della macchina-computer.
E' superfluo segnalare la difficoltà di comunicare con il computer tramite questo linguaggio composto solo da 0 e da 1.
A partire dagli anni '60, con la diffusione dei computer e il progressivo ampliamento del numero dei programmatori, sono emerse la possibilità e la convenienza di creare altri tipi di linguaggi, comprensibili dal computer ma più vicini al linguaggio parlato dagli uomini e quindi di uso più diretto.
Sono così nati e si sono sviluppati nel tempo livelli successivi di linguaggi che utilizzano parole di uso corrente (in inglese): grazie a questi linguaggi, il programmatore scrive le istruzioni non con sequenze di 0 e di 1, ma con parole di facile comprensione; il linguaggio si fa carico della loro traduzione in linguaggio macchina.
Ecco alcuni esempi di queste parole:

Open	*(Apri)*
Close	*(Chiudi)*
Save	*(Salva)*
Print	*(Scrivi)*
Load	*(Carica)*
End	*(Termina)*
Do	*(Fa)*
If	*(Se)*
Then	*(Allora)*
And	*(E)*
Or	*(Oppure)*
Else	*(Altrimenti)*
Select	*(Seleziona)*
While	*(Finché)*
Try	*(Prova)*

Tabella 1: Esempi di comandi in linguaggio VB.

Programmare utilizzando queste parole è evidentemente più facile che scrivere una serie di 1 e di 0. Il computer, tuttavia, non è in grado di interpretare alcuna di queste parole: il linguaggio di programmazione si pone come intermediario tra il programmatore e il computer e traduce le parole inglesi (cioè le istruzioni) scritte dal programmatore. Questa attività di intermediazione compiuta dai linguaggi di alto livello si chiama **compilazione**; i linguaggi di questo tipo sono chiamati appunto **linguaggi compilatori**.
VB fa parte della famiglia dei linguaggi di ultima generazione: sono i linguaggi detti di quinto livello.
Questi linguaggi mettono a disposizione del programmatore un **ambiente di lavoro** sofisticato, nel quale la parte grafica ha ormai un rilievo pari a quello del linguaggio vero e proprio. In questo **ambiente** di progettazione si trovano tutti gli accorgimenti possibili per liberare il programmatore dalle operazioni di routine, nella scrittura delle istruzioni.
VB, ad esempio, fornisce al programmatore questi strumenti di supporto al suo lavoro:

- un ambiente grafico in cui è possibile manipolare con estrema facilità gli elementi da inserire in un programma, le loro proprietà e i loro comportamenti;
- uno strumento (**IntelliSense**) per scrivere in modo corretto le istruzioni, con la documentazione istantanea, passo per passo, su quello che sta scrivendo;
- la funzione di auto-completamento della scrittura dei nomi delle variabili eliminando l'esigenza di scriverli in modo completo e dunque il rischio di commettere errori;
- strumenti che visualizzano i rimandi interni tra le diverse parti di un progetto;
- punti di interruzione e di controllo che consentono di padroneggiare il fluire di un programma e di localizzare eventuali errori di sintassi.

Una volta terminato il lavoro di progettazione, il linguaggio di alto livello si incarica dunque di compilare il programma in linguaggio macchina (cioè in una serie di numeri 0 e 1) e di renderlo distribuibile agli utenti finali ed eseguibile sui loro computer; gli utenti finali utilizzeranno il programma finito probabilmente senza nemmeno sapere con quale linguaggio è stato creato.

15: Il linguaggio BASIC.

Il linguaggio BASIC[9] fu elaborato negli anni '60 da un gruppo di insegnanti dell'Università di Dartmouth (Canada) a uso dei loro studenti, per consentire

[9] Beginner's All-purpose Symbolic Instruction Code (Linguaggio di Istruzioni Simbolico e Multifunzionale per il Principiante).

l'apprendimento della programmazione anche agli studenti che non seguivano i corsi di informatica.
Negli anni '70 il BASIC fu adottato come linguaggio di programmazione per quello che è considerato il primo personal computer: il MITS Altair 8800; successivamente, con la proliferazione dei personal computer, il linguaggio si diffuse e si ramificò in vari *dialetti*, a seconda delle macchine alle quali era destinato.
Si ebbero così il Commodore BASIC, il Super BASIC, l'Amiga BASIC, il GW BASIC, il Quick BASIC, il Turbo BASIC, il Simon's BASIC, e decine di altre versioni.
Tutti questi *dialetti* erano costruiti sul linguaggio originale BASIC e lo incorporavano, per cui era relativamente molto semplice per un programmatore passare da un dialetto all'altro[10].
Nel 1991 comparve la prima versione di Visual Basic: un evento rivoluzionario nella storia della programmazione, perché questo linguaggio - in connubio con il nuovo sistema operativo Windows - rendeva la creazione di programmi estremamente semplice ed alla portata di tutti.
Negli anni '90 Visual Basic ebbe nuovi sviluppi e nuove versioni sino al 1998, con la pubblicazione di Visual Basic 6; il successo mondiale del sistema operativo Windows andò di pari passo con l'affermarsi di Visual Basic come il linguaggio di programmazione di gran lunga più diffuso al mondo.
Nel 2002 la casa proprietaria, Microsoft Corporation, ha deciso di concludere questa esperienza, giunta alla versione 6, incanalando Visual Basic su un binario nuovo che pone questo linguaggio sullo stesso piano di altri linguaggi professionali, e che, nelle intenzioni dei proprietari, lo mette in condizione di raccogliere la sfida del futuro, evolvendosi con l'evolversi della tecnologia informatica.
Con il salto di qualità avvenuto nel 2002, VB si avvale dello stesso archivio di software (denominato FrameWork .NET) sul quale poggiano linguaggi ritenuti un tempo di livello superiore, garantendogli la stessa potenza e affidabilità.
Nell'ultimo decennio si sono susseguite queste edizioni di Visual Basic .NET:

- Visual Basic 2003, collegato alla FrameWork 1.1 (Visual Basic 7);
- Visual Basic 2005, connesso alla FrameWork 2 (Visual Basic 8);
- Visual Basic 2008, connesso alla FrameWork 3.5 (Visual Basic 9);
- Visual Basic 2010, connesso alla FrameWork 4 (Visual Basic 10).

Per quanto le successive edizioni di VB.NET abbiano conservato anche la numerazione progressiva delle versioni di Visual Basic, per marcare una certa linea di continuità con quelle (rispettivamente: versione 7, versione 8, versione 9 e versione 10), il passaggio a VB.NET rappresenta uno stacco netto rispetto al passato, e richiede l'acquisizione di nuovi termini e di nuovi concetti di programmazione.

[10] I comandi del linguaggio BASIC originale sono ancora in gran parte utilizzabili, a 50 anni di distanza, in VB 2010.

16: Dai linguaggi procedurali ai linguaggi orientati agli eventi e agli oggetti.

Sin dalla sua prima introduzione, Visual Basic ha fatto compiere al linguaggio BASIC[11] un salto di qualità, introducendo due elementi innovativi:

- un *ambiente* visivo, in tutto simile a quello dell'ambiente Windows, in cui il programmatore può vedere e prelevare gli strumenti che ritiene utili per il suo programma;
- il passaggio dalla modalità di progettazione *testuale* alla progettazione orientata agli *eventi* ed agli *oggetti*.

Sino alla nascita di Visual Basic, il programmatore scriveva i suoi programmi, o, meglio, scriveva una serie di comandi che il programma doveva eseguire riga per riga, seguendo pedissequamente le indicazioni del programmatore. L'utente del programma non aveva molte possibilità di intervento, non poteva intervenire sul suo ciclo di svolgimento e doveva adeguarsi all'ordine predeterminato dal programmatore.
Era un tipo di programmazione adatto ai primi tempi dell'informatica, atto a creare programmi finalizzati semplicemente all'elaborazione di dati.
Questo tipo di programmazione, con lo sviluppo della tecnologia informatica, e in particolare con l'affermazione dei sistemi operativi basati sulle cosiddette *finestre*, ha ceduto il passo alla possibilità di creare programmi più complessi, in grado di rispondere ai comportamenti dell'utente e in grado di gestire elementi grafici e sonori.

Gli sviluppi della progettazione hanno dunque aperto possibilità di creare progetti di novo tipo:

- orientati agli eventi;
- orientati agli oggetti.

I programmi orientati agli eventi sono interattivi, aperti all'utente, alle sue curiosità, ai suoi desideri, alle sue esigenze. Offrono all'utente una serie di opzioni e di possibilità diverse che sono state certamente predisposte dal programmatore, ma che si attivano come e quando l'utente decide di esplorare e sfruttare le risorse del programma che sta usando.
L'utente esercita le sue scelte producendo **eventi** quali, ad esempio:

- il *clic* del mouse su un pulsante;
- la pressione di un tasto sulla tastiera;
- la scrittura di una parola[12].

[11] Il termine che si legge più frequentemente oggi, invece di linguaggio, è *ambiente di programmazione*. Il termine *ambiente* sottolinea il fatto che il programmatore opera come se fosse all'interno di una *stanza dei bottoni*, con tutti gli strumenti a sua disposizione, visibili e a portata di mouse. L'uso del termine *linguaggio* è invece circoscritto al complesso di termini utilizzati per la scrittura delle istruzioni (il codice del programma).

[12] I programmi orientati agli eventi registrano anche eventi non causati dall'utente quali, ad esempio, il trascorrere del tempo. E' possibile dare istruzioni a un programma affinché, trascorso un determinato periodo di tempo (evento registrato dal computer) succeda una determinata cosa: ad esempio, in un gioco che simuli una partita a scacchi, alla scadenza di 30 minuti di attesa

Un esempio di applicazione orientata agli eventi è l'elaboratore di testi Microsoft Word: di fronte alla schermata iniziale di questo programma l'utente può decidere di aprire un file già esistente, di iniziarne uno nuovo, di scrivere un testo, di cambiare il font di caratteri, di ingrandirli o ridurli, …
Non c'è alcun ordine predefinito in queste operazioni: è l'utente a decidere di volta in volta cosa preferisce fare, il programma si adegua in modo elastico alle sue indicazioni.
Il programmatore (nel caso di Microsoft Word: l'équipe di programmatori) ha cercato di prevedere tutte le mosse dell'utente finale e di fornire al computer istruzioni adeguate per ognuna di queste mosse.
Nei programmi basati su eventi, dunque, il ruolo del programmatore viene reso più arduo e più stimolante perché egli deve essere in grado di mettersi nei panni del futuro utente e di predisporre soluzioni per ognuna delle azioni che questi deciderà di intraprendere all'interno del programma.

L'orientamento agli oggetti riguarda la possibilità, offerta al programmatore, di inserire nel programma oggetti preconfezionati (costruiti da altri programmatori) estremamente efficaci e che tuttavia richiedono un minimo impegno di programmazione.
Immaginiamo un programmatore che voglia inserire in un suo programma un pulsante che, una volta premuto con il mouse, simuli un vero pulsante, cioè si abbassi sotto la pressione del mouse e poi torni alla sua posizione originaria.
Con un linguaggio testuale la cosa è fattibile solo da un programmatore esperto, con un notevole impegno nella scrittura delle linee di istruzioni.
Con un linguaggio orientato agli oggetti la cosa diventa semplicissima: basta prendere l'oggetto **pulsante** e collocarlo dove si desidera che questo appaia nel quadro grafico del programma. Non occorre scrivere alcuna riga di istruzioni per farlo funzionare: l'oggetto **pulsante** è preconfezionato e contiene già in sé le istruzioni necessarie per simulare il pulsante che si abbassa sotto il *clic* del mouse e poi torna alla sua posizione originaria[13].
Al programmatore, sollevato dal compito di progettare il pulsante, rimane solo il compito di progettare cosa succederà quando l'utente del programma premerà questo pulsante.

17: La programmazione orientata agli oggetti.

Il concetto di orientamento agli oggetti, basilare nella odierna attività di progettazione, richiede qualche approfondimento.

(evento) durante i quali l'utente non ha fatto alcuna mossa, può apparire il messaggio "*Il tuo tempo è scaduto*".
[13] In VB questo tipo di pulsante preconfezionato è il controllo Button, illustrato più avanti.

In termini informatici, un **oggetto** è un *elemento di programmazione **finito** e **indipendente** dagli altri elementi del progetto, con una sua autonomia di funzionamento, utilizzabile nel quadro di un progetto più complesso*.
In sostanza, un oggetto è un componente integrato in un sistema, nel quale l'oggetto opera per consentire la funzionalità di tutto il sistema.

Vediamo un esempio tratto dalla vita quotidiana: abbiamo in casa una torcia elettrica, e ne conosciamo perfettamente il funzionamento. Sappiamo che al suo interno si trovano una batteria e una lampadina. Non sappiamo come funziona la batteria e non sappiamo come funziona la lampadina**, ma sappiamo come funziona la torcia**. In caso di bisogno, pur non sapendo come funzionano, siamo in grado di cambiare sia la batteria, sia la lampadina.
Molti oggetti di uso comune nella nostra vita sono di facile uso proprio perché per il loro uso e la loro manutenzione non è richiesta la conoscenza del funzionamento dei loro componenti:

- possiamo utilizzare un robot da cucina, scambiandone i pezzi secondo le esigenze, senza conoscere il funzionamento interno di alcuno di questi pezzi;
- possiamo guidare un'automobile senza conoscere il funzionamento del suo motore, del sistema rotante, delle parti elettriche;
- possiamo usare un telefono senza conoscere come funzionano la tastiera, la batteria, il display;
- possiamo usare un personal computer senza avere mai visto i suoi componenti interni (la CPU, l'hard disk, la scheda video, la scheda audio, …).

Sono attività alle quali abbiamo fatto l'abitudine e che ci sembrano banali; esse costituiscono tuttavia esempi concreti di un fenomeno chiamato **di astrazione**, per cui la complessità di un sistema viene **celata** all'utente, fornendogli una semplice **interfaccia** che gli consente di fare funzionare il sistema stesso.
Trasferiamo l'esempio della torcia che abbiamo visto poc'anzi in termini informatici: il programmatore mette insieme un programma utilizzando alcuni elementi precostituiti (la batteria, la lampadina), pur senza preoccuparsi di conoscere **come** questi elementi funzionano, badando piuttosto a **cosa** producono, al fine di farli *interagire* per conseguire gli obiettivi del programma.

Ecco un esempio relativo a un progetto creato con VB.
Vogliamo collocare in un programma una decina di pulsanti che, cliccati dall'utente, produrranno determinati effetti.
Con la programmazione a oggetti, non abbiamo bisogno di scrivere alcuna riga di programmazione per creare i dieci pulsanti: è sufficiente prendere l'oggetto **Button** (pulsante) e inserirlo dieci volte nel nostro programma, nelle posizioni che ci sembrano più opportune.
L'oggetto **Button** è un oggetto visibile che fa parte dell'insieme degli strumenti di VB. E' un oggetto che ha delle proprietà (cioè delle qualità, come l'altezza, la larghezza, il colore, la posizione…), che è capace di captare determinati eventi (il *clic* del mouse, il

doppio clic, il passaggio del mouse…) ed è capace di rispondere a questi eventi compiendo determinate azioni.
Questi dieci pulsanti avranno dunque un funzionamento autonomo piuttosto complesso, per ottenere il quale il programmatore non deve progettare alcunché: l'unico compito che gli rimane, una volta collocati i pulsanti all'interno del programma, è definire cosa succederà quando questi pulsanti verranno cliccati.

In sostanza:
- si ha *orientamento agli oggetti* quando in un sistema (programma) si mettono insieme delle unità di programmazione più semplici, indipendenti, capaci di funzionamento autonomo;
- queste unità di programmazione sono chiamate **oggetti**; tutta la progettazione orientata agli oggetti ruota attorno ad essi;
- per utilizzare uno di questi oggetti in un progetto, è sufficiente sapere **cosa** questo oggetto fa o produce, e non **come** lo fa o lo produce.

Abbiamo visto che il programmatore che inserisce un oggetto nel suo progetto non deve preoccuparsi della programmazione che sta dietro il funzionamento dell'oggetto stesso, ma ovviamente questa programmazione esiste: il suo nome tecnico è **Classe**.
Una **Classe** non è l'oggetto, ma è il complesso di dati e di istruzioni che dicono come un oggetto deve essere costruito (quali saranno le sue **proprietà**), quali **eventi** deve sapere riconoscere e cosa deve sapere fare (quali saranno le sue **azioni**).
La **Classe** è dunque una specie di *scatola nera* che definisce la forma di un oggetto e ne guida il comportamento.

Vediamo un esempio: la planimetria di un appartamento in costruzione prevede come deve essere fatto questo appartamento: quante stanze, quante porte, quante finestre, quali dimensioni deve avere. La planimetria ovviamente non è l'appartamento, ma è il complesso di istruzioni sulla base delle quali è possibile costruire l'appartamento.
La stessa relazione mappa/appartamento esiste tra la classe e l'oggetto.
Quando un oggetto è collocato all'interno di un progetto, gli viene subito assegnato un nome che lo distingue all'interno del programma, in modo automatico da VB oppure a scelta del programmatore. Si dice allora che la sua classe è stata **instanziata** (cioè è stata attuata, si è dato corso alle istruzioni in essa contenute, che hanno portato alla costruzione dell'oggetto).

18: La scelta di Visual Basic 2010 Express.

Abbiamo visto che nel pacchetto di strumenti di programmazione denominato Visual Studio® 2010 Express, sono presenti tre linguaggi di programmazione:
- Microsoft® Visual Basic® 2010 Express;
- Microsoft® C#® 2010 Express;

- Microsoft® C++® 2010 Express.

Oltre a questi, disponibili gratuitamente in questo pacchetto Express, esistono oggi decine di linguaggi di programmazione moderni ed efficienti, orientati agli eventi ed agli oggetti.
Abbiamo scelto VB per questi motivi:

- VB è uno dei linguaggi più semplici da imparare. E' un ambiente di progettazione intuitivo, che accompagna e guida il programmatore verso la creazione di applicazioni di alto livello, cercando tuttavia di sgombrare il cammino del programmatore da tutto quanto può ostacolare la realizzazione delle sue idee. Nonostante il passare del tempo VB mantiene un'aura del tutto particolare, di freschezza e di semplicità d'uso, che non si ritrova in altri linguaggi, che gli ha guadagnato la fedeltà di milioni di programmatori nel mondo.
- La conoscenza di VB è una conoscenza informatica capitalizzabile: essa non diventerà obsoleta nel giro di pochi anni ma, al contrario, potrà sempre tornare utile in futuro. VB ha alle sue spalle una storia lunga ormai 50 anni (in campo informatico si tratta di un lasso di tempo enorme); oggi è ampiamente diffuso a livello mondiale e avrà certamente un futuro in quanto VB è stato e rimane uno dei linguaggi di bandiera di Microsoft Corporation. Accusato un tempo di essere un *linguaggio giocattolo*, oggi VB sta alla pari dei linguaggi fratelli, C# e C++ in quanto ha la loro stessa potenza e affidabilità e affonda le sue radici nel medesimo archivio di software Framework 4.

PARTE II - L'AMBIENTE DI PROGETTAZIONE DI VB.

Conclusa la trattazione di alcuni argomenti di carattere generale, inizia qui la parte operativa del manuale.
Prendiamo le mosse dall'analisi dell'ambiente di progettazione con il quale VB si presenta al programmatore.

Capitolo 3: VISITA GUIDATA E PRIMO PROGETTO.

La creazione di un programma in VB si divide in due fasi:

- la preparazione dell'interfaccia[14] grafica, cioè la disposizione dei diversi elementi che l'utente vedrà sul suo monitor durante l'esecuzione del programma;
- la scrittura del codice, cioè la scrittura delle istruzioni che il programma dovrà eseguire quando accadranno determinati eventi (ad esempio: quando l'utente premerà con il mouse un pulsante o quando premerà un determinato tasto sulla tastiera).

Iniziamo qui l'esplorazione della parte **visiva** della programmazione. Diciamo, con un gioco di parole, che per ora ci occuperemo più di **Visual** che di **Basic**, e questa parte sarà per molti la più piacevole tra le due.

Durante l'esplorazione guidata - o durante le sue escursioni autonome - la lettrice o il lettore si imbatteranno certamente in **cose di cui in questo testo non si parla**. Ciò è dovuto al fatto che l'ambiente di progettazione di VB è ormai uno strumento molto complesso, che può essere utilizzato proficuamente sia da programmatori alle prime armi, sia da programmatori professionisti. La lettrice o il lettore troveranno dunque in queste pagine solo ciò che abbiamo ritenuto indispensabile a fornirgli un primo orientamento nel campo della progettazione e a metterlo in grado di produrre in poco tempo applicazioni valide e funzionanti.

L'esplorazione personale è tuttavia una attività utile al completamento e al consolidamento delle conoscenze: lungi dallo sconsigliarla, la raccomandiamo vivamente alle lettrici e ai lettori.

19: Analisi dell'ambiente di progettazione di VB.

Dal pulsante di **Avvio**, in basso a sinistra, facciamo *clic* su **Programmi** e quindi su **Microsoft Visual Studio 2010** e su **Microsoft Visual Basic 2010 Express**.

[14] Per interfaccia qui si intende il complesso degli elementi che appaiono sullo schermo, con la funzione di facilitare la comunicazione tra l'utente del programma e le istruzioni contenute nel codice del programma, perché il computer esegua ciò che l'utente vuole. L'utente opera solo sull'interfaccia grafica, senza vedere il codice scritto con le istruzioni retrostanti. L'interfaccia si assume il compito di captare le informazioni relative a ciò che l'utente fa o vuole e di passarle al codice del programma.

Compare così la pagina iniziale dell'ambiente di progettazione di VB. In termini tecnici, questo è l'**IDE** di VB (**Integrated Development Environment,** ambiente integrato di sviluppo).
Ci troviamo nella **Pagina iniziale**, come si legge nella intestazione della pagina.
Vediamo alcuni degli elementi presenti in questa finestra scorrendo i numeri da 1 a 6, in rosso, visibili nella figura seguente.

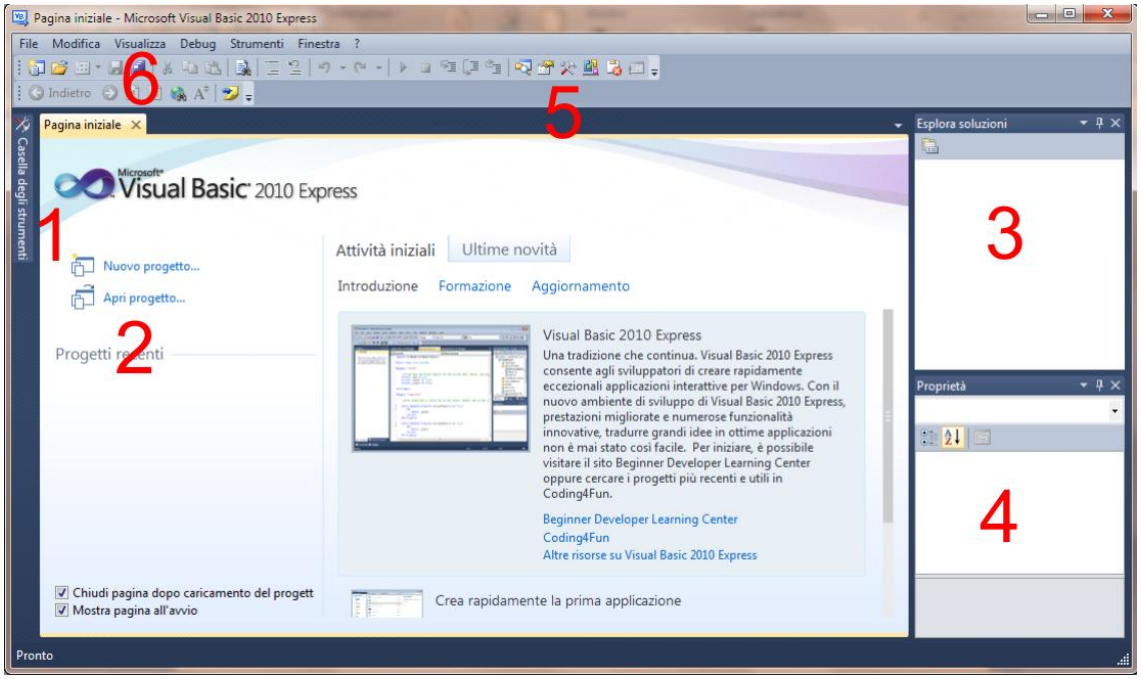


Figura 12: La pagina di apertura dell'ambiente di progettazione di VB.

All'estrema sinistra, al n. 1 troviamo la **Casella degli strumenti**. Si tratta di una casella a scomparsa, che si apre quando ci passiamo sopra con il mouse e si chiude quando il mouse si allontana. Premendo l'icona con la puntina, è possibile fissare la casella in modo che la **Casella degli strumenti** rimanga sempre visibile, ma rimandiamo queste operazioni di personalizzazione a quando avremo maggiore familiarità con tutto l'ambiente di progettazione. La casella per ora è vuota, non contiene nessuno strumento, in quanto non abbiamo ancora iniziato alcun progetto.

Procedendo verso destra, al n. 2 troviamo due link dai quali è possibile partire per creare un **nuovo progetto** o **aprire un progetto** creato in precedenza. Nello spazio sottostante, notiamo la dicitura **Progetti recenti**: qui comparirà l'elenco degli ultimi progetti avviati, quando ne avremo qualcuno. Per ora lo spazio è vuoto.

All'estrema destra dell'ambiente di lavoro, ai nn. 3 e 4 troviamo le due finestre **Esplora soluzioni** e **Finestra Proprietà.** Nella finestra **Esplora soluzioni** sono visibili tutte le

parti che costituiscono il progetto in corso di lavorazione. Da questa finestra è possibile accedere a ognuna di queste parti, per operare su ognuna di esse singolarmente[15].
La **Finestra Proprietà** è una finestra di uso molto frequente durante la creazione di un programma. In essa compaiono le proprietà di ogni singolo controllo (oggetti grafici come i pulsanti, i riquadri di testo, i menu, i riquadri di immagini) che fanno parte del progetto. La **Casella degli Strumenti** di VB contiene 21 controlli di uso comune e molti altri oggetti: ognuno di questi oggetti possiede decine di proprietà (nome, lunghezza, larghezza, colore, posizione, testo, …). Tutte le proprietà di ogni oggetto inserito nel progetto sono gestibili dalla **Finestra Proprietà**.

In alto, al n. 5, nella striscia dei pulsanti troviamo tre icone che consentono di aprire rispettivamente la **Casella degli strumenti**, la finestra **Esplora soluzioni** e la **Finestra Proprietà**.

In alto, al n. 6, tra i menu, facciamo un *clic* sul menu **Visualizza**. Nel menu che si apre a tendina, facciamo un *clic* su **Altre finestre**. Qui troviamo un'altra possibilità di aprire la **Casella degli strumenti**, la finestra **Esplora soluzioni** e **la Finestra Proprietà**. Notiamo che si accede alla **Finestra Proprietà** anche cliccando il tasto **F4**.

[15] Vediamo di fare un po' di chiarezza su questi termini usati nell'ambiente di progettazione di VB 2010:

- un **progetto** è un insieme di file creati dal programmatore, finalizzati a funzionare insieme per raggiungere un determinato scopo;
- quando un progetto è finito in tutte le sue parti, esso viene compilato per crearne un software eseguibile direttamente e distribuibile: a questo punto siamo in presenza di un **programma**. Un programma è dunque un progetto finito, reso eseguibile su tutti i computer mediante la **compilazione**;
- un gruppo di progetti diversi, finalizzati a fare parte della medesima applicazione, costituiscono una **soluzione**. Ad esempio, un progetto per la creazione di un programma di *labirinti* può essere distinto in due progetti autonomi e diversi: un progetto per *eseguire* labirinti al computer e un progetto per *creare* labirinti personali. I due progetti sono cose diverse e producono risultati diversi, possono tuttavia essere correlati e possono fare parte di una unica soluzione, finalizzata a produrre l'applicazione *Labirinti*.

In questo manuale ci occuperemo solo di **progetti** e di **programmi**. Nella finestra **Esplora soluzioni** vedremo dunque sempre e solo le parti che compongono **un singolo progetto**.

Per prendere confidenza con l'ambiente di progettazione e le sue finestre, avviamo la creazione di un **Nuovo progetto**:

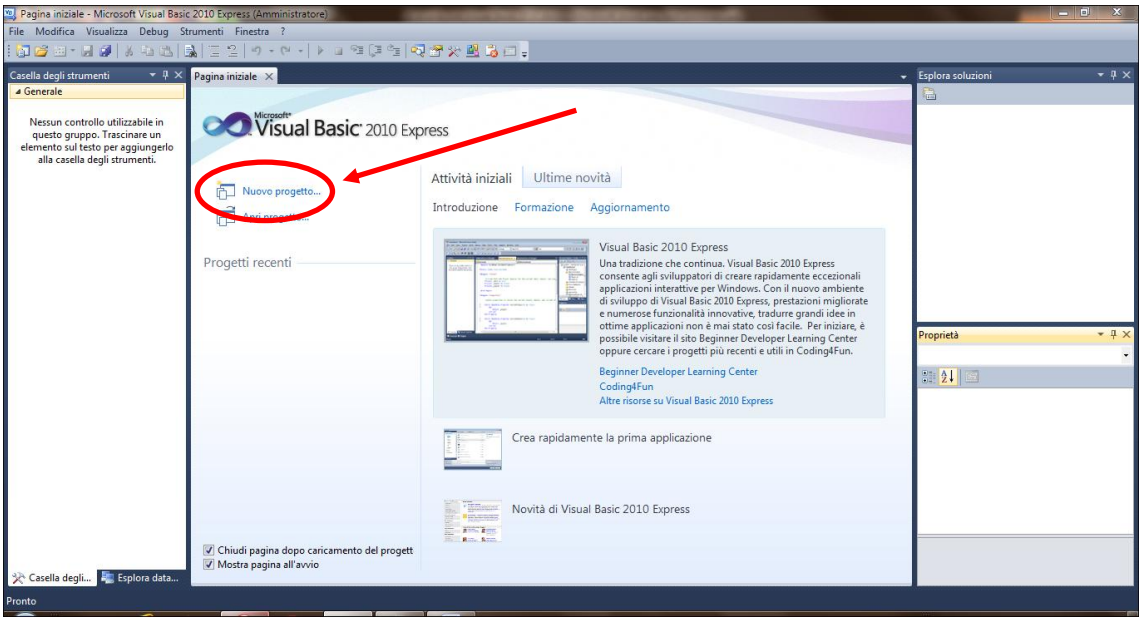


Figura 13: Avvio di un nuovo progetto.

Compare la finestra **Nuovo progetto**, visibile nella prossima figura, in cui ci viene chiesto di specificare quale tipo di progetto intendiamo avviare.

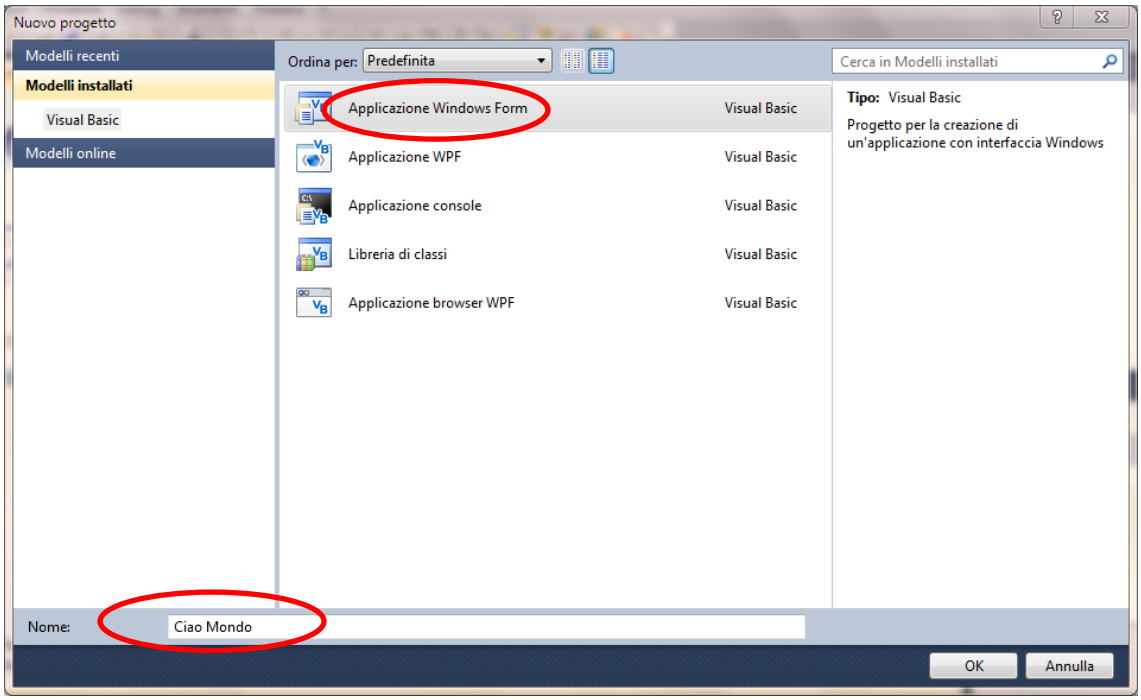


Figura 14: La finestra di scelta e denominazione del nuovo progetto.

Come si vede nella colonna centrale di questa finestra, con VB si possono creare progetti di diverso tipo: programmi destinati a internet, programmi destinati a operare dietro le quinte e a rimanere invisibili, oppure classi di oggetti da utilizzare in altri programmi.
Una **Applicazione Windows Form**, prima opzione della lista, è il tipo di programma famigliare a ognuno di noi: è un programma che opera usando una finestra sul monitor e oggetti ormai famigliari come pulsanti, testi, immagini e audio.
In questo manuale creeremo solo applicazioni di questo tipo.
Non clicchiamo subito la voce **Applicazione Windows Form**. Prima, nella linea in basso, scriviamo il nome del progetto che intendiamo realizzare: lo chiamiamo **Ciao Mondo**, e vedremo tra poco perché.
Ora possiamo fare *clic* su OK. In questo modo passiamo al linguaggio VB l'informazione che vogliamo creare una nuova **Applicazione Windows Form** che si chiamerà **Ciao Mondo.** Sarà un programma che una volta finito funzionerà autonomamente e potrà essere eseguito su qualsiasi computer con sistema operativo Windows, a prescindere dal fatto che su questo computer sia installato o meno il linguaggio VB.

Eccoci dunque nell'ambiente di progettazione vero e proprio, pronto a ricevere le nostre istruzioni.
Tradizione vuole che un programmatore alle prese con un nuovo linguaggio di programmazione realizzi come primo progetto il programma **Hello World**, cioè un progetto che faccia comparire sul monitor la scritta "Hello World". Seguiamo dunque la tradizione, per scaramanzia, con questo progetto **Ciao Mondo** che abbiamo appena avviato.

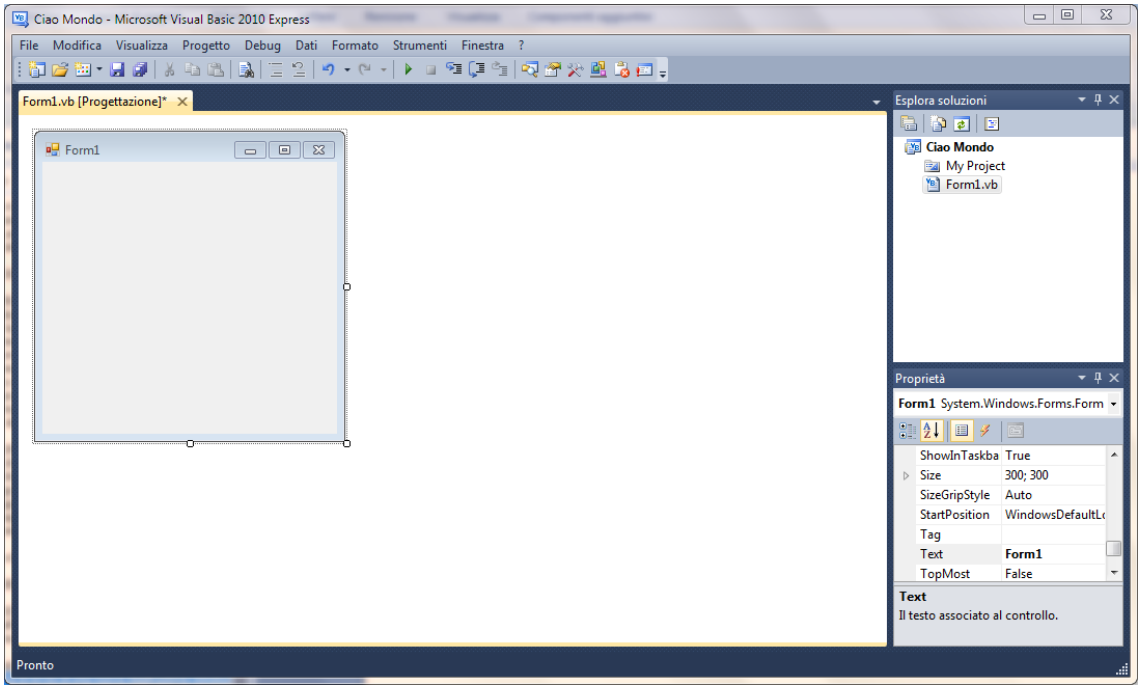


Figura 15: La pagina di avvio del progetto "Ciao Mondo".

In alto, nell'intestazione della pagina di progettazione si vedono il titolo e il tipo del nuovo progetto.
Nella parte centrale compare un oggetto **Form** (= modulo), in bianco, vuoto, denominato **Form1.**
Il nome proprio **Form1** gli è stato assegnato in modo automatico da VB; il programmatore può cambiare questo nome, ed è auspicabile che lo faccia quando lavora a un progetto con molti Form, in modo da distinguere facilmente ogni singolo Form e le sue funzioni.
Notiamo che il riquadro che contiene il **Form1** porta l'intestazione **Form1.vb [Progettazione]**.

Questo **Form1 è un oggetto** che appartiene alla categoria dei **Form** (= modulo), è il primo oggetto del nostro progetto, ed è destinato a contenere altri oggetti.
Il **Form1** è la finestra che l'utente vedrà quando il programma sarà in esecuzione, è l'interfaccia tra il programma e l'utente: questi vedrà solo l'interfaccia e opererà con gli oggetti che avremo collocato nell'interfaccia.
In quanto oggetto, il **Form1** possiede delle **proprietà**, che il programmatore può modificare, e risponde a determinati **eventi** con dei comportamenti (**azioni**) che il programmatore può attivare o meno, secondo le esigenze del progetto al quale sta lavorando.
Tutto quanto è riferibile all'oggetto Form1 (**proprietà**, capacità di rispondere a **eventi**, capacità di compiere **azioni**) è retto da una programmazione, creata da altri programmatori, contenuta in una **Classe**. Con l'inserimento del **Form1** nel progetto, la relativa **Classe** è stata istanziata, cioè le è stata data realizzazione concreta.
Tornando a un esempio usato in precedenza, possiamo dire che la planimetria dell'appartamento è stata concretizzata: questo **Form** è come un appartamento creato sui dati contenuti nella planimetria.
Notiamo che il **Form1** ha una riga di intestazione con una icona di VB e il titolo **Form1**.
A destra, in questa riga di intestazione compaiono le tre caselle che nelle finestre di Windows servono a:

- chiudere temporaneamente una Finestra e ridurla a *pop- down*[16] sulla barra delle applicazioni;
- ridurre la grandezza di una Finestra o massimizzarla a grandezza schermo;
- terminare un programma.

All'estrema sinistra dell'ambiente di progettazione si intravvede, seminascosta, la **Casella degli strumenti**.
La apriamo, passandoci sopra con il mouse, e notiamo che stavolta la Casella contiene numerosi strumenti: gli strumenti di uso più comune sono 21 e compaiono nella parte superiore della casella. Più in basso, ci sono altri gruppi di strumenti, in genere finalizzati a usi particolari, raggruppati per categorie.
Tutti questi strumenti sono oggetti che si possono prendere e collocare in un progetto, con questa distinzione:

[16] Una finestra ridotta *pop-down* compare nella striscia delle applicazioni solo con l'icona del programma.

- alcuni oggetti rimarranno **visibili** all'utente del nostro programma: questi oggetti sono chiamati **controlli**;
- altri oggetti, invece, pur operando all'interno di un programma, rimarranno **invisibili** all'utente finale (ad esempio: un **Timer**): questi oggetti sono chiamati **componenti**.

Dunque: tutti gli **strumenti** di questa casella sono **oggetti**. Gli oggetti visibili sono **controlli**, gli oggetti non visibili sono **componenti**.

Scorriamo le icone dei 21 controlli comuni e cerchiamo di capire quale può essere la funzione di ognuno di essi. Passando con il mouse su un oggetto (ad esempio, il primo della serie: il **Button**) leggiamo una breve descrizione dell'oggetto e delle sue funzioni. Per il **Button**, ad esempio, possiamo leggere che esso *"Genera un evento quando l'utente fa clic su di esso*"; ci viene detto inoltre che il Button è **.NET component**, cioè è un oggetto che proviene dall'archivio di software FrameWork .NET 4.

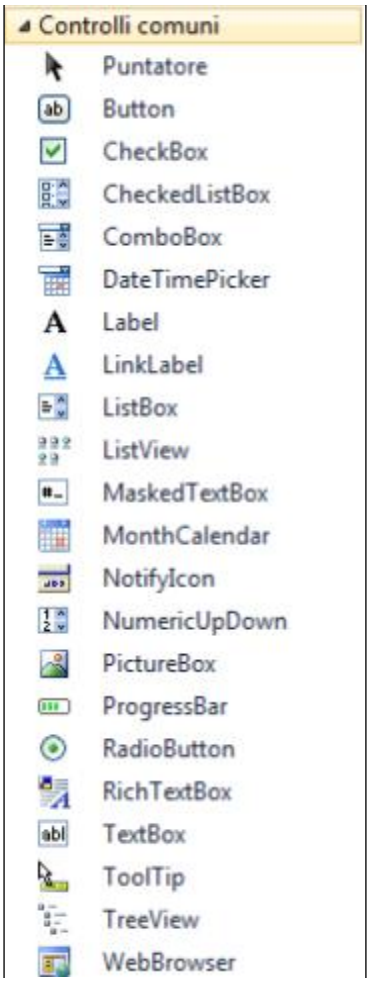


Figura 16: I controlli comuni nella Casella degli Strumenti.

A destra dell'ambiente di progettazione vediamo due finestre.
In alto, vediamo la finestra **Esplora soluzioni**, nella quale compare una visione panoramica del nostro progetto e delle sue parti.

Sotto la finestra **Esplora soluzioni** vediamo la **Finestra Proprietà**.
La **Finestra Proprietà** contiene, per ogni oggetto presente nel progetto (Form e oggetti della Casella degli Strumenti), l'elenco delle loro qualità: qual è il loro nome proprio, come si devono presentare all'utente, che aspetto devono avere, dove e come si devono collocare sul monitor, ecc.

L'ambiente di progettazione di VB presenta dunque un buon numero di finestre di lavoro. Per fortuna, non sono tutte necessarie **contemporaneamente**. Nel corso del lavoro di progettazione è possibile **spostarle** sullo schermo, **ingrandirle**, **ridurle** o **chiuderle**. Quando serviranno potranno essere riaperte di volta in volta tramite i menu o le icone che VB mette a disposizione in abbondanza a questo scopo.

L'unico oggetto presente nel nostro progetto **Ciao Mondo** per ora è il **Form1**.
Facendo un *clic* sulla immagine del **Form1**, vediamo comparire nella **Finestra Proprietà** le proprietà di questo **Form**.
Possiamo scegliere tra due modalità di visualizzazione: **proprietà raggruppate per categoria** o **proprietà elencate in ordine alfabetico** (cliccando le prime due icone che si vedono nella finestra si passa da una modalità all'altra).

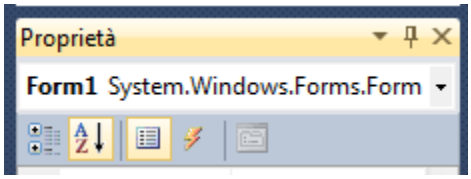


Figura 17: Le icone della Finestra Proprietà.

Scegliamo l'ordine alfabetico e scorriamo l'elenco, leggendo quali sono le caratteristiche del form (e le modifiche che possiamo apportarvi).
L'elenco è piuttosto lungo, e sarà così per ogni oggetto, ma non bisogna lasciarsi impressionare: nei progetti più comuni è necessario definire solo alcune di queste proprietà; la maggior parte di esse possono essere lasciate tranquillamente così come VB le definisce di *default*.
Nel progetto **Ciao Mondo**, modificheremo solo una delle proprietà del **Form1**: il **testo**.
Cerchiamo la proprietà **Text** (in questo caso Text significa intitolazione del Form) e, nello spazio a destra della voce **Text**, al posto di **Form1** scriviamo **Ciao Mondo**.

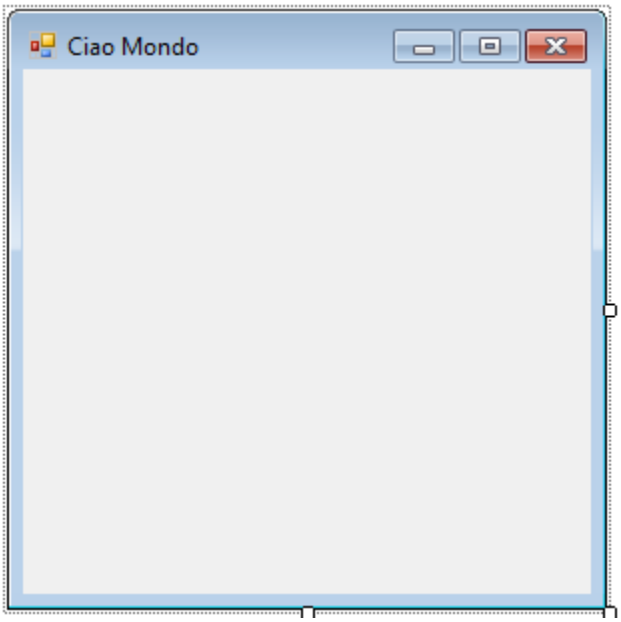


Figura 18: Il form con il testo "Ciao Mondo".

Notiamo che nella **Finestra Proprietà** vi sono altre icone: cliccando l'icona con il lampo si passa dalle proprietà del Form1 all'elenco degli eventi che il Form1 è in grado di riconoscere e di gestire secondo le istruzioni contenute nella sua Classe.
Vi troviamo molti eventi familiari a un utente di computer: il *clic* e il doppio *clic* del mouse, la pressione di un tasto, il trascinamento del mouse…; si tratta di un elenco collocato qui a scopo di documentazione, ma qui poco utile a fini pratici.
Cliccando l'icona a sinistra del lampo torniamo a vedere l'elenco delle proprietà del Form1.

Continuiamo il progetto **Ciao Mondo** inserendovi due controlli: un pulsante **Button** e una etichetta **Label**.
Funzionamento del programma: cliccando il pulsante Button, nell'etichetta Label comparirà la scritta "Ciao Mondo".

Torniamo ad aprire la Casella degli Strumenti. Per comodità, possiamo bloccarla in modo che rimanga sempre visibile sul monitor: per fare questo facciamo *clic* sull'icona con la freccina in basso, e sul comando **Àncora**.

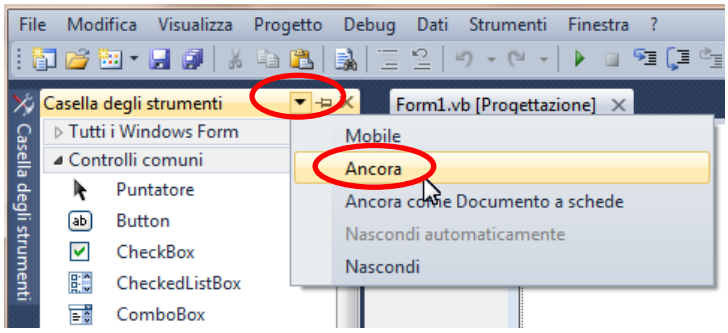


Figura 19: Il comando per bloccare la Casella degli Strumenti.

Facciamo un doppio *clic* sull'oggetto **Button** (il primo della lista, dopo il Puntatore): vediamo che un pulsante compare all'interno del nostro Form1, dove è possibile cliccarlo per collocarlo dove preferiamo. In alternativa:

- facciamo un solo *clic* sull'oggetto **Button**: in questo modo abbiamo selezionato l'oggetto;
- spostiamo il puntatore del mouse dentro il Form e notiamo che l'oggetto Button lo segue;
- quando il puntatore del mouse si trova nel Form nella posizione in cui desideriamo collocare il pulsante Button, facciamo un nuovo *clic* con il mouse.

A sinistra, in alto, seguendo l'uno o l'altro metodo, collochiamo invece un oggetto **Label**, come nella figura seguente.

Figura 20: I due oggetti Button1 e Label1 nel progetto "Ciao Mondo".

Con queste operazioni abbiamo terminato la creazione dell'interfaccia grafica del progetto; mancano ora le istruzioni per fare funzionare i due oggetti contenuti nel Form. A questo scopo, dobbiamo inserire nel progetto queste due istruzioni:

- quando avverti il *clic* del mouse sull'oggetto Button1
- cambia il testo della Label1 in "Ciao Mondo".

Queste istruzioni si scrivono in una nuova finestra, che non abbiamo ancora visto: è la **Finestra del Codice**.
Vi sono vari modi per accedervi; per ora adottiamo il modo più diretto: facciamo un doppio *clic* sull'oggetto Button1.
Abbiamo così accesso alla **Finestra del Codice**, che è di importanza fondamentale perché è qui che vengono scritte le istruzioni per il programma (cioè cosa deve fare il computer quando accade un evento nel programma in esecuzione).

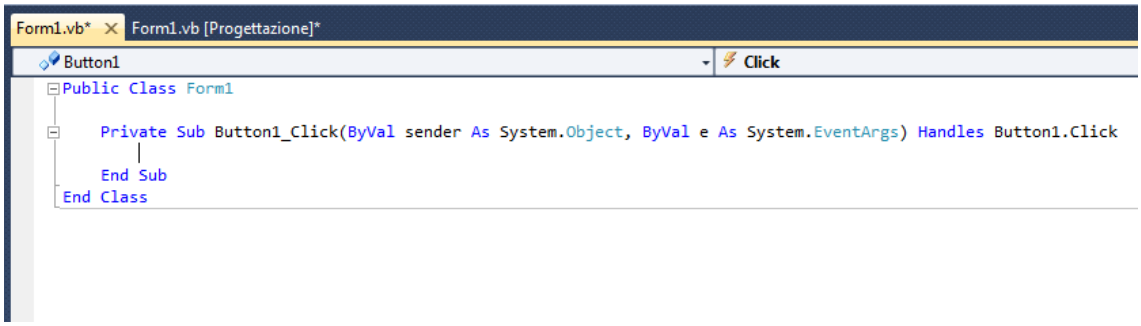


Figura 21: La Finestra del Codice.

Notiamo le due scritte che si vedono nella linea della intestazione di questa finestra:

- Form1.vb*[17] e
- Form1.vb [Progettazione]*

Si tratta dei titoli di due schede diverse:

- una scheda in cui si vede il codice del progetto;
- una scheda in cui si vede l'interfaccia grafica del progetto.

Siccome siamo arrivati ad aprire la Finestra del Codice cliccando il Button1, vi troviamo già delle righe di istruzioni relative al Button1, e in particolare all'evento del *clic* del mouse sul Button1:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    End Sub
```

[17] L'asterisco indica che il progetto non è ancora stato salvato. Esso scompare automaticamente dopo il salvataggio.

```
End Class
```

Torneremo su questa procedura tra poco.
Ora, nella scheda intitolata **Form1.vb**, cioè nella **Finestra del Codice**, sotto la linea della intestazione notiamo due menu che si aprono a tendina:

- Nel menu di **sinistra** vediamo l'elenco degli **oggetti** presenti nel Form1, e ritroviamo qui gli oggetti Button1 e Label1.

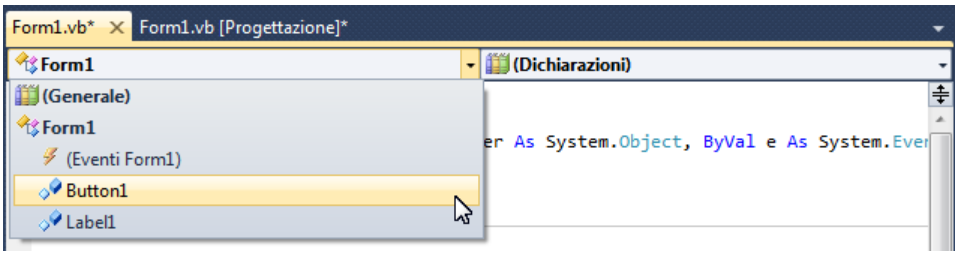


Figura 22: I menu a tendina nella Finestra del Codice.

- Nel menu di **destra** vediamo l'elenco degli **eventi** che ognuno di questi oggetti è in grado di riconoscere e di gestire.

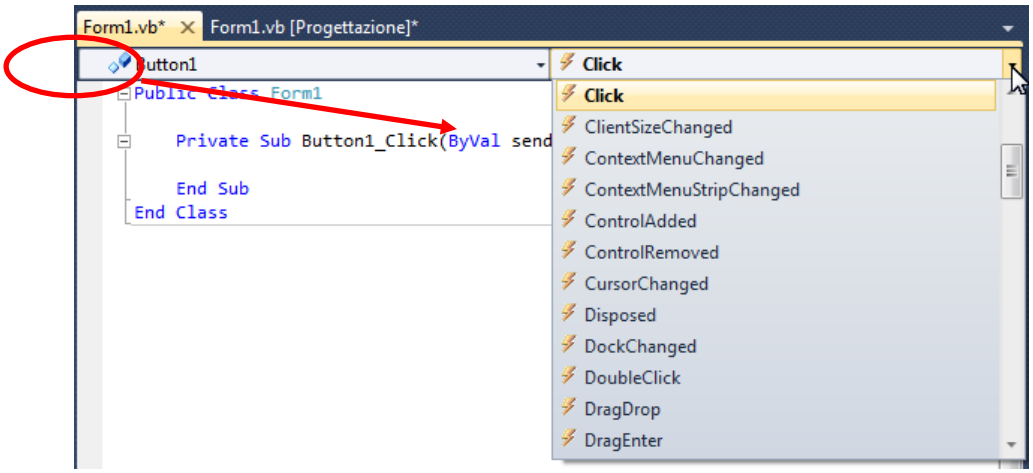


Figura 23: Gli eventi connessi all'oggetto Button1.

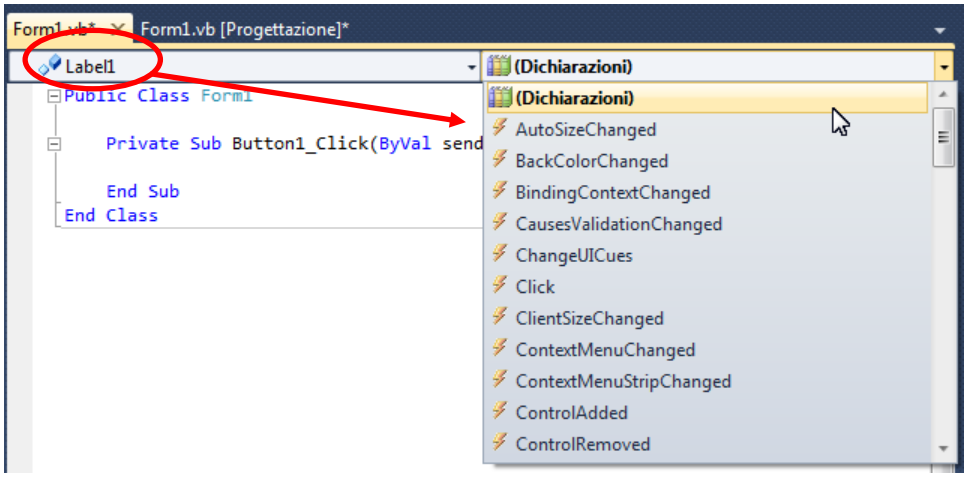


Figura 24: Gli eventi connessi all'oggetto Label1.

Nel menu di sinistra facciamo un *clic* su Button1 e nel menu di destra facciano un *clic* sull'evento **DoubleClick**. Notiamo che nella Finestra del Codice viene scritta in modo automatico una **procedura** (un elenco di istruzioni) finalizzata a gestire l'evento del doppio *clic* del mouse sul pulsante Button1.
Nel menu di sinistra facciamo ora un *clic* sul controllo Label1 e nel menu di destra facciamo un *clic* sull'evento **MouseMove**. Notiamo che nella Finestra del Codice viene scritta in modo automatico una **procedura** finalizzata a gestire l'evento del passaggio del mouse sull'etichetta Label1.
Abbiamo ora un codice formato da tre procedure:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

    End Sub

    Private Sub Button1_DoubleClick(sender As Object, e As System.EventArgs)
Handles Button1.DoubleClick

    End Sub

    Private Sub Label1_MouseMove(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Label1.MouseMove

    End Sub

End Class
```

Notiamo che ogni procedura inizia con le parole Private Sub… e termina con la parole End Sub. Tra **Private Sub**… e **End Sub** vanno scritti i comandi che il programmatore vuole fare eseguire al computer; per ora non vi è alcun comando, le tre procedure sono vuote e non producono effetti.

Notiamo che le tre procedure si collocano tra le righe **Public Class Form1**… e **End Class**. Queste righe segnalano che ci troviamo di fronte a una istanziazione della Classe Form; cancellando queste due righe, il Form1 scompare dal progetto.

Riprendiamo lo schema delle istruzioni di questo pogramma:

- quando avverti il *clic* del mouse sull'oggetto Button1
- cambia il testo della Label1 in "Ciao Mondo".

Le due procedure `Button1.DoubleClick e Label1.MouseMove` riguardano eventi che non saranno trattati in questo programma, per cui possiamo eliminarle dal codice, lasciando solo la prima procedura: `Button1.Click`.

Inseriamo in questa procedura, dedicata all'evento *clic* del mouse sul Button1, le istruzioni affinché nella Label1 compaia la scritta "Ciao Mondo":

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Label1.Text = "Ciao Mondo"

    End Sub
```

Scriviamo la nuova riga lentamente, cliccando un tasto alla volta sulla tastiera per vedere come VB ci viene in aiuto nella scrittura di queste istruzioni.
Notiamo che dopo avere battuto i tasti "la" VB suggerisce l'oggetto Label1.
Dopo avere scritto Label1 scriviamo un punto ".".
Questo punto scritto subito dopo l'oggetto Label1 indica a VB che ciò che viene dopo il punto definisce una delle proprietà o una delle azioni della Label1.
VB stesso mostra al programmatore tutte le proprietà e tutte le azioni dell'oggetto[18].

[18] Questa attività continua di supporto al lavoro del programmatore da parte di VB è data da **IntelliSense**, lo strumento di supporto di cui abbiamo già avuto modo di parlare, che in base a ciò che il programmatore sta facendo in quel momento si incarica di suggerirgli le azioni possibili. Abituarsi a utilizzare il supporto di **IntelliSense** è importante non solo perché allevia notevolmente il lavoro di scrittura del codice, ma soprattutto perché elimina il rischio di compiere errori nella scrittura del codice.

Scorriamo questo elenco e giungiamo alla proprietà **Text**.

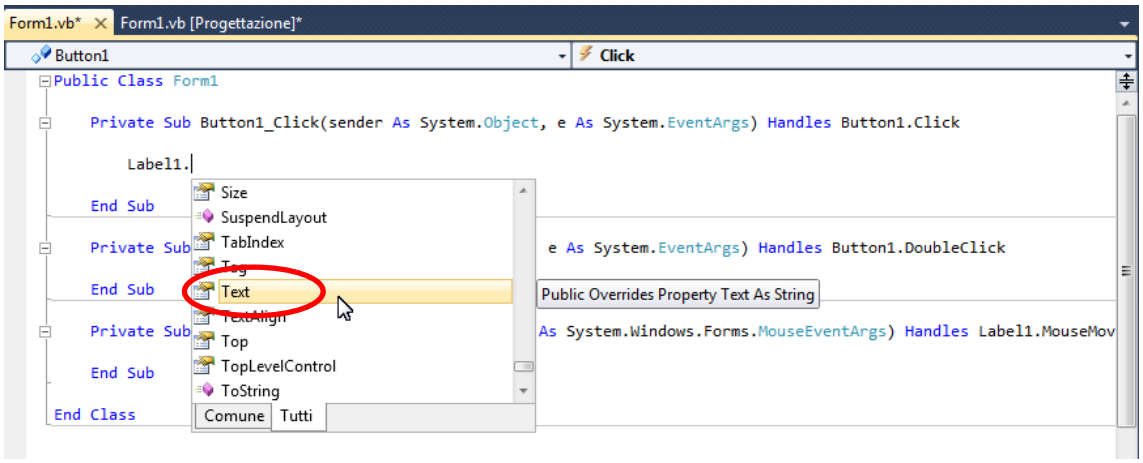


Figura 25: Impostazione della proprietà Text della Label1.

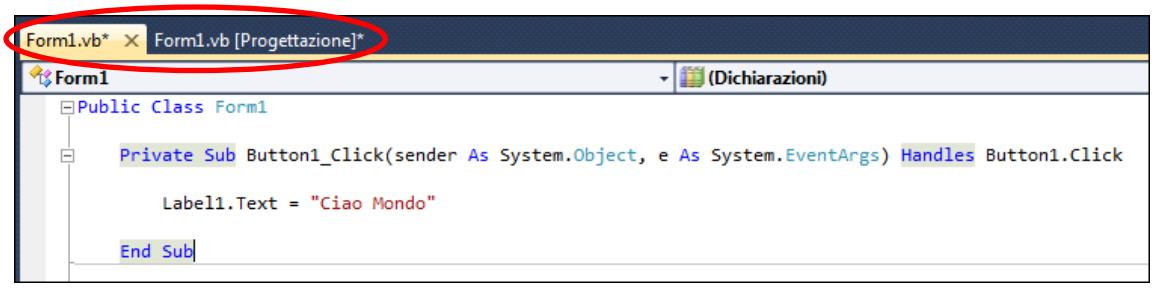
Facciamo un doppio *clic* del mouse sulla proprietà Text, per inserirla nella nostra procedura, oppure premiamo sulla tastiera il tasto TAB e completiamo la linea di istruzioni con la scritta "Ciao Mondo":

```
Label1.Text = "Ciao Mondo"
```

Notiamo ancora come **IntelliSense** continua ad assistere il programmatore:

- se scriviamo **Ciao Mondo**" senza le virgolette, VB segnala che c'è qualcosa che non va;
- se invece scriviamo **"Ciao Mondo** senza le virgolette finali, VB si incarica di aggiungerle per chiudere il testo.

Con questo, il progetto è finito. Prima di provarne l'esecuzione, consolidiamo l'abitudine di passare dalla finestra del codice alla finestra di progettazione grafica cliccando le intestazioni che si notano in alto nelle due finestre:



Oppure proviamo lo stesso passaggio cliccando le due icone che si trovano a destra nella striscia dei pulsanti della finestra **Esplora soluzioni**:

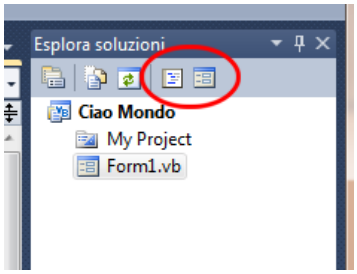


Figura 26: Il passaggio dalla finestra del codice alla finestra di progettazione.

Oppure, ancora, cliccando il menu **Visualizza**, nella barra di VB: nel menu a che si apre a tendina, le prime due righe consentono il passaggio dalla finestra del codice alla finestra della progettazione grafica e suggeriscono, per effettuare queste operazioni l'uso dei tasti **F7** e **MAIUSC+F7**.

Mandiamo in esecuzione il nostro progetto cliccando il tasto **F5**, oppure cliccando l'icona con la freccina verde nella striscia dei pulsanti, oppure ancora cliccando, in alto nella striscia dei menu, il menu **Debug / Avvia debug**.

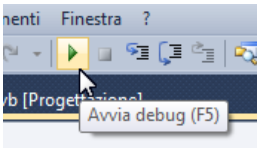


Figura 27: L'icona Avvia debug (avvio della esecuzione provvisoria di un progetto).

Una volta lanciato il programma, apparentemente non accade nulla. Trattandosi di programmazione orientata agli eventi, il programma è in attesa che si verifichi un evento che sia connesso a qualche istruzione scritta nella Finestra del Codice, che gli dica cosa fare di fronte a questo evento.
Notiamo che il programma riconosce già e gestisce in modo automatico - senza che noi abbiamo scritta una riga di istruzioni - l'evento del passaggio del mouse sopra il Button1.
Proviamo a passare con il mouse sopra il Button1: il programma riconosce l'evento *passaggio del mouse* e lo gestisce facendo *cambiare colore* al pulsante; il tutto in modo automatico. Questo avviene perché nell'oggetto Button1 sono già incorporate le istruzioni per rispondere al passaggio del mouse con il cambiamento del colore, senza che il programmatore debba ri-programmare queste funzioni ogni volta di nuovo.
Il pulsante Button1 ovviamente sa riconoscere anche l'evento *clic* del mouse, e per gestire questo evento ha a sua disposizione le istruzioni che abbiamo inserito nella procedura Button1_Click.
Se non abbiamo commesso errori, facendo un *clic* sul pulsante Button1 otteniamo l'effetto di cambiare la scritta nella Label1, come nell'immagine seguente.

Figura 28: Il progetto "Ciao Mondo" in esecuzione.

L'operazione di avvio dell'esecuzione del progetto si chiama in VB "**debug**". Il termine è pessimistico, perché tradotto alla lettera vuol dire "*elimina gli errori*".
Nel nostro caso non dovremmo avere problemi. Se ci sono errori, questi vengono segnalati nella finestra **Elenco errori**, sotto la **Finestra del Codice**.
L'operazione inversa al **debug** è **termina debug**, che termina il progetto in esecuzione.

- Si può fermare il progetto in esecuzione come si fermano tutti i programmi, cliccando la x su campo rosso in alto a destra.
- Si può cliccare l'icona **termina debug** (il quadrato a destra della freccina, nella linea delle icone).
- Si può cliccare il menu **Debug / Termina debug**.

Preoccupiamoci ora di salvare il progetto finito.
Clicchiamo l'icona **Salva tutto**, nella striscia dei pulsanti, oppure nella striscia dei menu clicchiamo il menu **File** (in alto a sinistra) / **Salva tutto**.

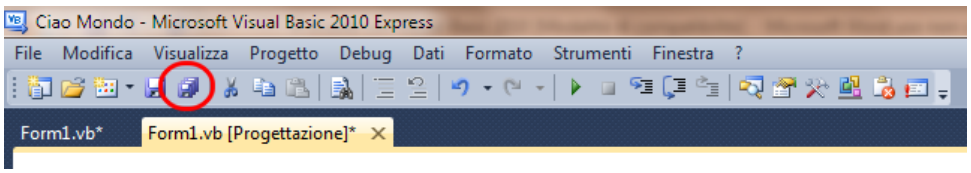
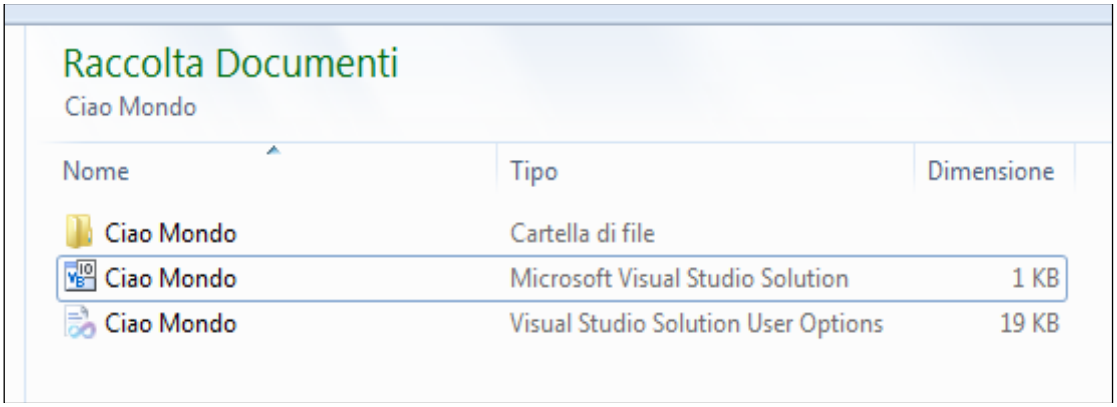


Figura 29: L'icona "Salva tutto" nella stiscia dei menu a icone.

Accettiamo le modalità di salvataggio automatico che ci vengono proposte. Il nostro progetto viene così salvato nella cartella **Documenti | Visual Studio 2010 \ Projects 1\ Ciao Mondo**.

Andiamo a controllare questa cartella: vediamo che essa contiene il file **Ciao Mondo.sln**, che è il file con il contenuto della finestra Esplora soluzioni e una sottocartella denominata ancora Ciao Mondo.



Nella sottocartella **Ciao Mondo** troviamo i file del nostro progetto, e in particolare il file **Ciao Mondo.vbproj** che è il file principale del progetto.
Assieme a questo file principale, VB ha salvato numerosi altri file e sottocartelle in cui ha memorizzato tutti i dati relativi a questo progetto, che rimangono disponibili per eventuali aggiornamenti del progetto.

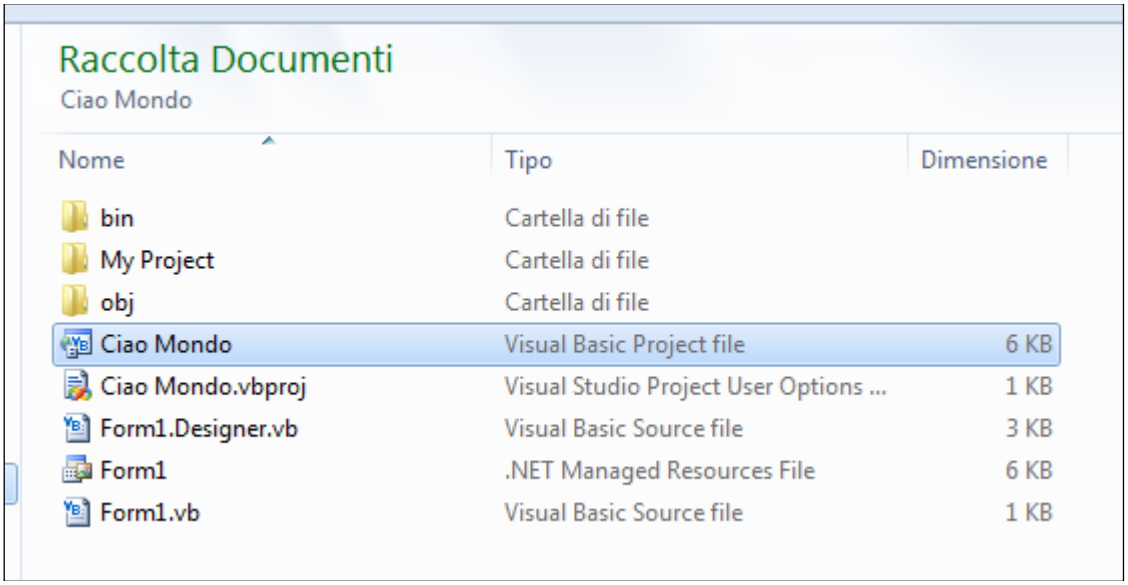


Figura 30: Le cartelle e i file di salvataggio automatico del progetto "Ciao Mondo".

Prima di abbandonare il progetto **Ciao Mondo** proviamo ad apportarvi alcune modifiche. Lo faremo con il prossimo esercizio.

Attenzione: se il progetto è in fase di *debug* (cioè se il progetto è in esecuzione) non è possibile modificarne il codice. Per tornare alla fase di progettazione è necessario cliccare l'icona **Termina debug.**

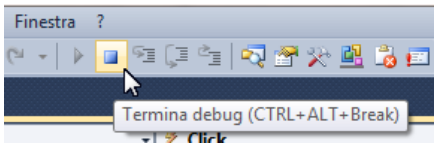


Figura 31: L'icona Termina debug.

Con un *clic* su questa icona si torna a visualizzare la **Finestra di Progettazione** (cioè la finestra dell'interfaccia grafica, in cui compaiono gli oggetti).

Esercizio 2: Modifiche e integrazioni al progetto "Ciao Mondo".

Obiettivi di questo esercizio:

- Modificare il colore dello sfondo del Form1 e raddoppiarne la larghezza.
- Cambiare il testo del Button1.
- Cambiare il colore dei caratteri della Label1 e ingrandirne i caratteri.
- Inserire una funzione che consenta all'utente del programma di scrivere un nome.
- Fare comparire questo nome nella Label1.

Procediamo punto per punto.

Per modificare il colore dello sfondo del Form1 e raddoppiarne la larghezza:

- Facciamo un *clic* sul Form.
- Vediamo le sue proprietà, nella Finestra Proprietà.
- Cerchiamo la proprietà BackColor (colore dello sfondo).
- Scegliamo il colore blu nella scheda "Personalizzato".
- Cerchiamo la proprietà Size (dimensioni): le dimensioni che vediamo sono espresse in pixel: la prima cifra indica la larghezza, la seconda cifra indica l'altezza.
- Scriviamo 600 al posto della prima cifra e vediamo che la larghezza del Form si raddoppia.

Per cambiare il testo del Button1:

- Facciamo un *clic* sul pulsante Button1.
- Vediamo le sue proprietà, nella Finestra Proprietà.
- Cerchiamo la proprietà Text e al posto di Button1 scriviamo CLICCAMI.

Per cambiare il colore dei caratteri della Label1 e ingrandirne i caratteri:

- Facciamo un *clic* sul pulsante Label1.
- Vediamo le sue proprietà, nella Finestra Proprietà.

- Cerchiamo la sua proprietà Font[19] e scegliamo il carattere Micosoft Sans Serif, stile grassetto, a 24 punti.
- Cerchiamo la sua proprietà ForeColor (colore in primo piano, immediatamente sotto la proprietà Font) e scegliamo il colore giallo nella scheda "personalizzato".

Con questo abbiamo completato le modifiche *grafiche* da apportare al progetto.
Ora mancano le istruzioni necessarie per conseguire questi due ultimi obiettivi:

- Inserire una funzione che consenta all'utente del programma di scrivere un nome.
- Fare comparire questo nome nella Label1.

Queste istruzioni ovviamente dobbiamo scriverle nella **Finestra del Codice**.
In particolare, dobbiamo inserire una Funzione[20] mediante la quale il programma sia in grado di recepire un nome scritto dall'utente, per poi visualizzarlo come testo della Label1.
Dove prima era solo l'istruzione `Label1.Text = "Ciao Mondo"`, ora scriviamo tre righe di codice.

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Dim NOME As String
        NOME = InputBox("COME TI CHIAMI?", "CIAO MONDO", "Anonimo", 100, 100)
        Label1.Text = "Ciao " & NOME

    End Sub
```

Analizziamo le nuove righe una a una.

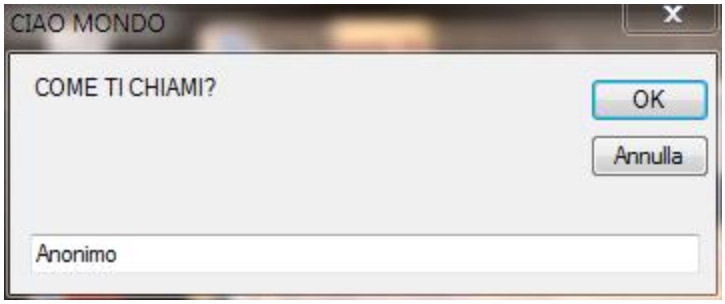
Nella prima riga (`Dim NOME As String`) viene istruito il computer perché crei (`Dim` = dimensiona) nella sua memoria una *casella* che conterrà la variabile `NOME`; si precisa che in questa casella verrà inserito un testo, e non un numero (`As String`).
La seconda riga ha la funzione di recepire il nome scritto dall'utente mediante un `InputBox`, cioè una casella per l'immissione di dati, che viene creata da VB in modo automatico.
Nella terza riga viene istruito il computer affinché nella Label1 scriva la parola `"Ciao "` e vi aggiunga (`&`) il contenuto della variabile `NOME`. Dunque, se `NOME` sarà uguale a "Tizio" il computer scriverà "Ciao Tizio".

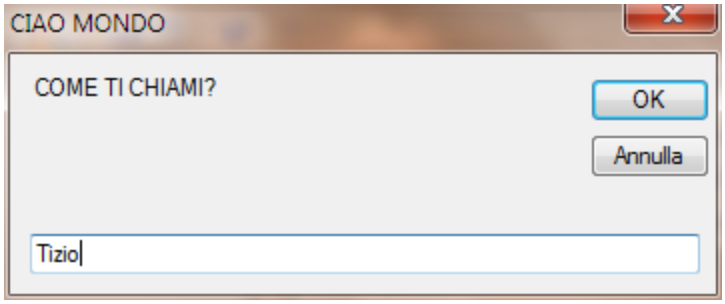
[19] Facendo *clic* con il mouse sulla casella a destra della proprietà Font appare un pulsante con tre puntini. I tre puntini stanno ad indicare che per questa proprietà vi sono molte possibilità di scelta. Facciamo un *clic* sui puntini: nella Finestra che si apre impostiamo il carattere desiderato come si fa normalmente all'interno di un elaboratore di testi.
[20] Le istruzioni scritte nella Finestra del Codice possono essere **procedure** o **funzioni**.
Le **procedure** contengono istruzioni perché il programma risponda a determinati eventi con determinate azioni.
Le **funzioni**, invece, contengono solo elaborazioni di dati, i cui risultati vengono utilizzati nelle procedure. Vi sono funzioni che elaborano numeri (cioè elaborano variabili numeriche) e funzioni che elaborano *stringhe* (cioè elaborano variabili di testo).

Vediamo in dettaglio la seconda riga.
In linguaggio corrente, essa dice: la variabile `NOME` è costituita da quanto sarà scritto dall'utente all'interno di un `InputBox`. Le caratteristiche di questo `InputBox` sono scritte tra le parentesi e sono separate l'una dall'altra dalle virgole:

- questo `InputBox` mostrerà la domanda `"COME TI CHIAMI?",`
- e il titolo `"CIAO MONDO"`,
- mostrerà la parola `"Anonimo",` come risposta di default (automatica) in attesa del testo scritto dall'utente;
- l'`InputBox` andrà a collocarsi a 100 pixel di distanza dal bordo sinistro dello schermo
- e a 100 pixel di distanza dal bordo superiore dello schermo (`100,100`).

Notiamo che, mentre impostiamo I parametri dell'`InputBox`, IntelliSense suggerisce passo per passo la parte che manca per completare e concludere correttamente le istruzioni.
Mandando in fase di debug il nostro progetto, il risultato dovrebbe essere questo:





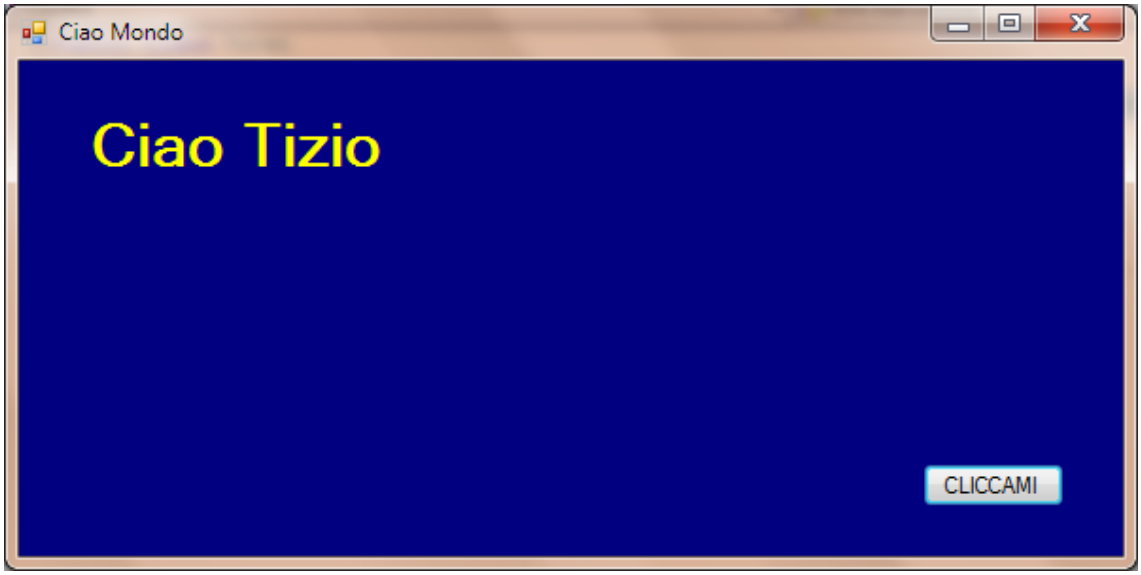


Figura 32: Il progetto "Ciao Mondo" in esecuzione dopo le modifiche.

Salviamo il tutto cliccando l'icona **Salva tutto**.
Nel prossimo esercizio vedremo di studiare il codice che abbiamo scritto in questo progetto, per iniziare da sùbito a familiarizzare con alcuni concetti basilari di VB.

Esercizio 3: Analisi del codice del progetto "Ciao Mondo".

Obiettivo di questo esercizio è esaminare il codice del progetto "Ciao Mondo" per comprendere il significato delle parti che VB vi inserisce in modo automatico.

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Dim NOME As String
        NOME = InputBox("COME TI CHIAMI?", "CIAO MONDO", "Anonimo", 100, 100)
        Label1.Text = "Ciao " & NOME

    End Sub

End Class
```

Abbiamo già visto che la procedura di nome `Button1_Click` inizia con `Private Sub Button1_Click` e termina con `End Sub`.
Abbiamo visto pure che questa procedura è inserita nella Classe che istanzia l'oggetto Form1, i cui limiti vanno da `Public Class Form1` a `End Class`.
Notiamo che mentre la Classe è definita `Public`, la procedura è definita `Private`: questo significa che altri Form, se ve ne fossero, potrebbero interagire con la Classe *pubblica*

di questo Form, ma non potrebbero interagire con la procedura *privata* `Button1_Click`, che produce effetti solo all'interno del Form1.
Ora analizziamo in dettaglio la riga di apertura della procedura `Button1_Click`:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

Button1_Click è il nome proprio di questa procedura. Possiamo cambiarlo a nostro piacere, magari con un nome più familiare e più facilmente comprensibile, ad esempio ClicSulPulsante, e il risultato non cambia:

```
Private Sub ClicSulPulsante(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

Tutto quanto è scritto tra le parentesi servirebbe, se ne avessimo bisogno, a fornire alla procedura `ClicSulPulsante` delle informazioni particolari sul *clic* del mouse che si è appena verificato.
Ad esempio:

- su quale oggetto è stato effettuato il clic?
- con quale pulsante del mouse è stato fatto[21].

Ora, non avendo bisogno di alcuna di queste informazioni particolari, possiamo tranquillamente svuotare il contenuto delle parentesi: la procedura funzionerà ugualmente, senza problemi:

```
Private Sub ClicSulPulsante() Handles Button1.Click
```

Abbiamo qui, dunque, una procedura che si chiama ClicSulPulsante, che non richiede informazioni particolari, che gestisce (Handles) l'evento del *clic* con il mouse sul pulsante Button1 (Button1.Click).
Possiamo allungare a piacere la serie degli eventi gestiti da questa procedura: per ottenere questo è sufficiente aggiungere i nuovi eventi dopo Button1.Click.
Ad esempio, se sul Form1 ci fossero altri tre pulsanti Button, di nome Button2, Button3 e Button4, la procedura ClicSulPulsante sarebbe in grado di gestire l'evento *clic* su tutti questi pulsanti, con queste aggiunte:

```
Private Sub ClicSulPulsante() Handles Button1.Click, Button2.Click,
Button3.Click, Button4.Click
```

Se vogliamo che nel nostro progetto, la procedura `ClicSulPulsante` sia attivata **anche** da un *clic* sulla Label1, aggiungiamo l'evento `Label1.Click`:

```
Private Sub ClicSulPulsante() Handles Button1.Click, Label1.Click
```

[21] All'interno delle parentesi, in questa procedura, sono contenuti due parametri:

- il parametro **sender** (= mittente) dice qual è l'oggetto dal quale è partito l'evento;
- il parametro **e** (= evento) fornisce dettagli, che vedremo più avanti, sull'evento che si è verificato.

Se vogliamo che la procedura sia attivata **anche** da un *clic* sul Form1 aggiungiamo l'evento Me.Click:

```
Private Sub ClicSulPulsante() Handles Button1.Click, Label1.Click, Me.Click
```
[22]

In questo modo abbiamo indicato al computer che il contenuto della procedura ClicSulPulsante deve essere eseguito quando si concretizza uno di questi tre eventi:

- un *clic* sull'oggetto Button1;
- un *clic* sull'oggetto Label1;
- un *clic* sull'oggetto Form1 (Me).

Ecco il codice completo del progetto:

```
Public Class Form1

    Private Sub ClicSulPulsante() Handles Button1.Click, Label1.Click, Me.Click
        Dim NOME As String
        NOME = InputBox("COME TI CHIAMI?", "CIAO MONDO", "Anonimo", 100, 100)
        Label1.Text = "Ciao " & NOME
    End Sub

End Class
```

Mandiamolo in esecuzione (debug) e vediamo che l'`InputBox` si attiva non più solo con un *clic* sul pulsante Button1, ma anche con un *clic* sulla Label1 e un *clic* sul Form1.

Lasciamo qui il progetto Ciao Mondo e vediamo di analizzare più dettagliatamente l'ambiente di progettazione di VB.

20: La striscia dei menu.

Nella linea sotto l'intestazione dell'ambiente di progettazione di VB compare la striscia dei menu: **File, Modifica, Visualizza, Progetto, Debug, Dati, Strumenti, Finestra, ?**.

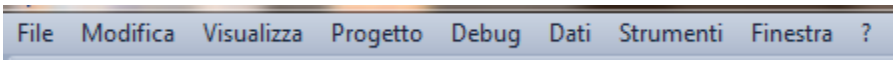


Figura 33: La striscia dei menu.

Ogni menu si apre con un *clic* del mouse, a tendina, mostrando una serie di comandi. Non tutti i comandi sono sempre disponibili (cioè eseguibili). I comandi momentaneamente disattivati appaiono sbiaditi (scritti in grigio). Nella fase di avvio della creazione di un nuovo progetto, ad esempio, nel menu **Modifica** sono disattivati

[22] Il termine Me si riferisce alla Classe dentro la quale ci troviamo a operare. Siccome in questo caso ci troviamo all'interno della classe Form1, il Me si riferisce al Form1.

tutti i comandi che riguardano le modifiche da apportare alla scrittura del codice, ovviamente perché ancora non è stata scritta alcuna linea di codice da modificare.
Vediamo ora, per alcuni di questi menu a tendina, quali sono i comandi di uso più frequente.

Il menu File.

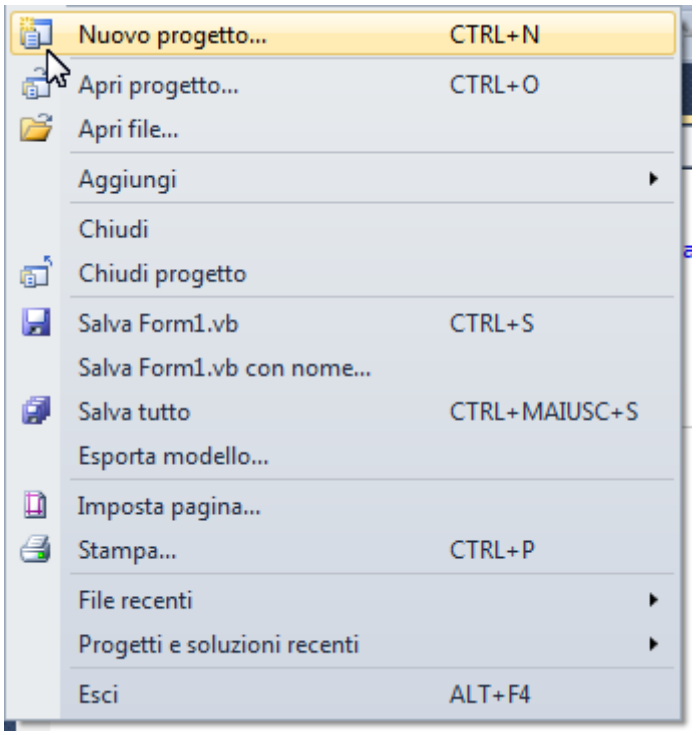


Figura 34: Il menu "File".

- **Salva tutto**: salva il progetto, con tutti gli oggetti e il codice che ne fanno parte, assegnandogli un nome, nella collocazione che si desidera.
  Se non si fornisce una indicazione diversa, VB salva il progetto nella cartella **Documenti / Visual Studio 2010 / Projects / (sottocartella con il nome del programma)**.

Il menu Modifica.

Figura 35: Il menu "Modifica".

- Questo menu contiene i comandi Taglia, Copia e Incolla, che agiscono sul testo del codice e su alcuni elementi dell'interfaccia del progetto. Si tratta dei comandi abituali nei word processor (elaboratori di testo, programmi per la scrittura): dopo avere evidenziato una parte del testo, è possibile tagliarla, copiarla e incollarla altrove.
  Come negli elaboratori di testo, gli stessi comandi possono essere eseguiti cliccando le icone corrispondenti, nella striscia dei pulsanti, oppure premendo i tasti
  CTRL (Control) **+ X** (= taglia)
  CTRL + C (= copia)
  CTRL + V (= incolla)

Il menu Visualizza.

Figura 36: Il menu "Visualizza".

- **Codice**: visualizza la finestra con il codice (le righe) delle istruzioni scritte dal programmatore. Si accede direttamente alla **Finestra del Codice** anche premendo il tasto **F7**.
- **Finestra di Pogettazione**: visualizza la parte grafica del progetto al quale si sta lavorando (il form o i form e gli elementi grafici già collocati in essi dal programmatore). Si accede direttamente alla **Finestra del Codice** anche premendo i tasti **MAIUSC + F7**.
- **Altre finestre**: da questo comando si accede a un sottomenu che consente di passare rapidamente da una finestra all'altra. In particolare, si possono visualizzare da qui la **Finestra Proprietà** (che si apre anche premendo il tasto **F4**), la finestra **Esplora Soluzioni** e la **Casella degli Strumenti**.

Il menu Progetto.

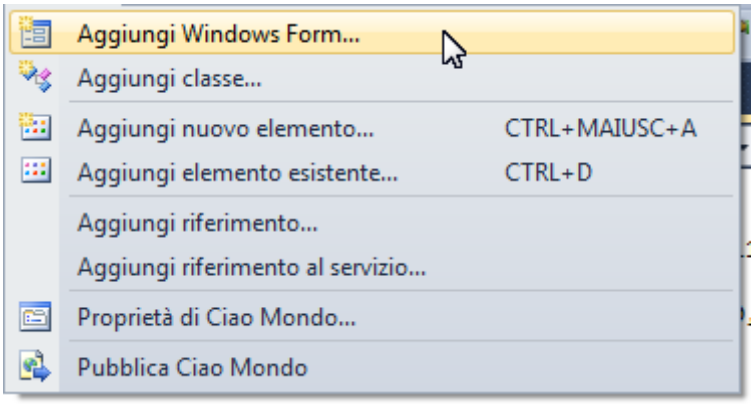


Figura 37: Il menu "Progetto"

- **Aggiungi Windows Form…**, **Aggiungi Classe…**: con questi comandi è possibile inserire altri Form nel progetto, o altre Classi destinate a gestire altri oggetti.
- **Proprietà di…**: questo comando dà accesso a una serie di schede riguardanti le caratteristiche del progetto al quale il programmatore sta lavorando:
  La prima scheda (**Applicazione**) contiene il quadro delle scelte iniziali del programmatore.
  La seconda scheda (**Compilazione**) gestisce le modalità di compilazione del progetto finito, gestisce cioè la creazione di un programma in formato .exe, finito, distribuibile ed eseguibile su altri computer.

Il menu Debug.

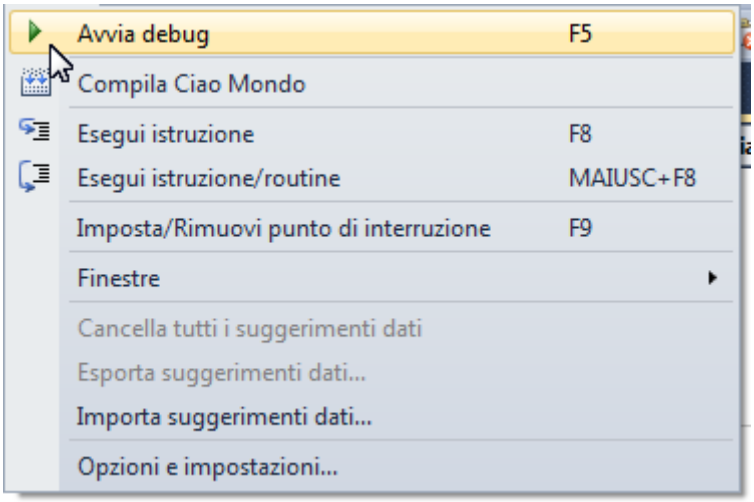


Figura 38: Il menu "Debug".

Questo menu contiene alcuni comandi utili per localizzare e correggere eventuali errori nel programma[23]; sono comandi utilizzati dal programmatore quando vuole provare gli effetti del suo lavoro, cioè quando vuole passare dalla fase di progettazione (*Design time*) alla fase di esecuzione (*Run time*) del programma.

- **Avvia debug** è il comando che manda in esecuzione provvisoria il progetto al quale il programmatore sta lavorando, segnalandogli eventuali errori presenti nel codice (nelle istruzioni). **Avvia debug** si attiva direttamente, senza passare per questo menu, premendo il tasto **F5.**
  Quando il **debug** è avviato, in questo menu compare il comando **Termina debug**, che termina l'esecuzione provvisoria del programma e restituisce il comando delle operazioni al programmatore.
- **Compila…** è il comando che conclude il lavoro del programmatore, creando un programma con estensione .exe, eseguibile direttamente su ogni computer al di fuori dell'ambiente VB[24].

Il menu Formato.

[23] **Bug** significa *pulce*, quindi **debug** equivale a *spulciare*, ad eliminare ospiti indesiderati (gli errori) dal proprio lavoro di progettazione.
[24] Una volta terminato un progetto, VB unisce le parti scritte dal programmatore e le parti create automaticamente da VB e **compila** il programma. L'operazione di compilazione è in sostanza una operazione di traduzione: il progetto, sino a ora scritto in modo leggibile dal programmatore, viene convertito nella lingua comprensibile dal computer che, come sappiamo, è una lingua fondata sui due simboli 0 e 1.

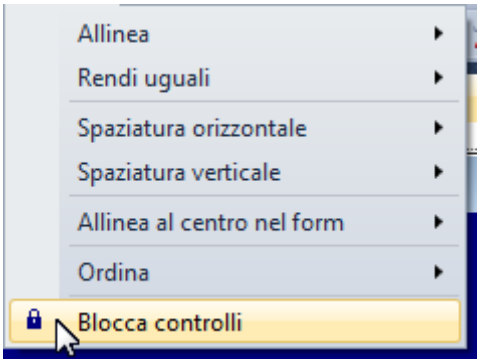


Figura 39: Il menu "Formato"

Questo menu compare solo quando il programmatore ha collocato sul Form dei controlli, cioè degli oggetti visibili (pulsanti, etichette, riquadri di testo, riquadri per immagini…). Esso mette a diposizione alcuni comandi di uso semplice ed efficace per allineare gli elementi grafici presenti in un form e creare un'interfaccia più ordinata e gradevole.
I comandi di questo menu non agiscono su tutti i controlli presenti in un form, ma agiscono solo sui controlli selezionati dal programmatore[25].

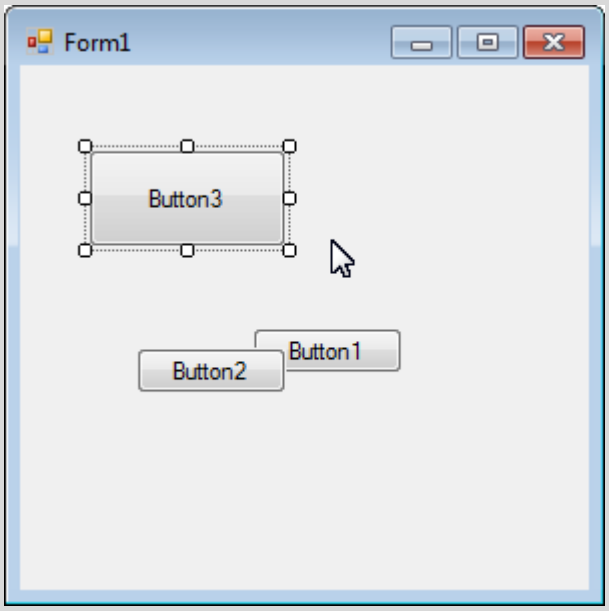
- **Allinea**: allinea o di incolonna un gruppo di controlli.
- **Rendi uguali**: rende uguali i controlli selezionati: uguali solo in larghezza, uguali solo in altezza, oppure uguali in entrambe le dimensoni.
- **Spaziatura orizzontale**: rende uguali le distanze in orizzontale tra i controlli selezionati.
- **Spaziatura verticale**: rende uguali le distanze in verticale tra i controlli selezionati.
- **Allinea al centro del form**: colloca al centro del form un singolo controllo selezionato, oppure il gruppo di controlli selezionati.
- **Ordina**: stabilisce quale controllo, tra quelli selezionati, deve comparire in primo piano, nel caso i controlli selezionati siano più di uno, e parzialmente o totalmente sovrapposti uno all'altro.
- **Blocca controlli**: blocca nel form tutti gli oggetti che il programmatore vi ha inserito, in modo che questi non corra il rischio di modificarli o spostarli accidentalmente. Quando gli oggetti sono già stati bloccati, il comando svolge la funzione inversa, cioè consente di sbloccare I controlli per modificarne l'aspetto, le dimensioni e la collocazione.

Considerata l'utilità pratica di questi comandi, li vedremo all'opera nel prossimo esercizio.

[25] Per selezionare un gruppo di controlli esistono due metodi:
- utilizzare l'oggetto **Puntatore** e tracciare un rettangolo attorno ai controlli da selezionare;
- fare un *clic* con il tasto sinistro del mouse su tutti i controlli da selezionare, tenendo premuto il tasto MAIUSC.

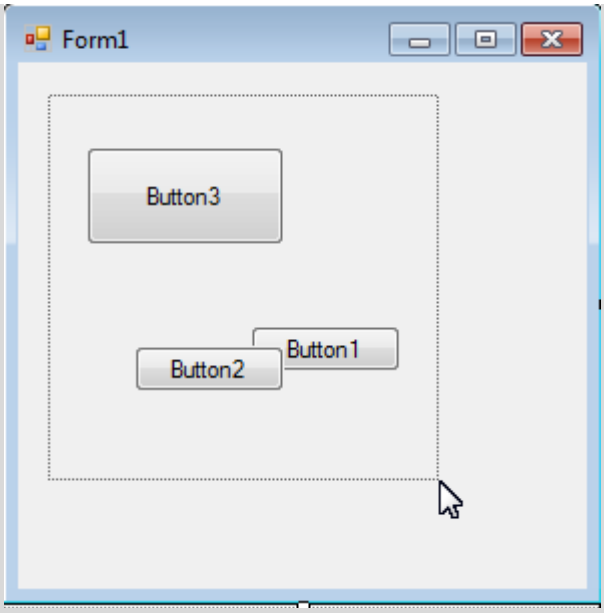
Esercizio 4: I comandi del menu Formato.

Apriamo un nuovo progetto di VB, senza preoccuparci di dargli un nome particolare, perché non lo salveremo.
Trasciniamo nel Form1 un pulsante Button1.
Nel menu Formato, clicchiamo i comandi **Allinea al centro nel form** / **Orizzontalmente** e / **Verticalmente**.
Vediamo l'effetto che questi due comandi hanno sulla posizione del Button1.
Trasciniamo sul Form1 un altro pulsante, il Button2.
Muoviamo il Button2 attorno al Button1 e vediamo come VB ci mostra, con delle linee di colore blu o di colore viola, quando i due pulsanti sono incolonnati o allineati correttamente.
Collochiamo il Button2 sopra il Button1, in modo che il Button2 copra in parte il Button1. Nel menu Formato, clicchiamo i comandi **Ordina** / **Porta in primo piano** e / **Porta in secondo piano**. Vediamo l'effetto che questi due comandi hanno sulla visualizzazione dei due pulsanti.
Trasciniamo sul Form1 un altro pulsante, il Button3 e, agendo sulle maniglie di dimensionamento (cioè sugli 8 quadretti che delimitano il Button3), modifichiamo le sue dimensioni più o meno come nella figura seguente.

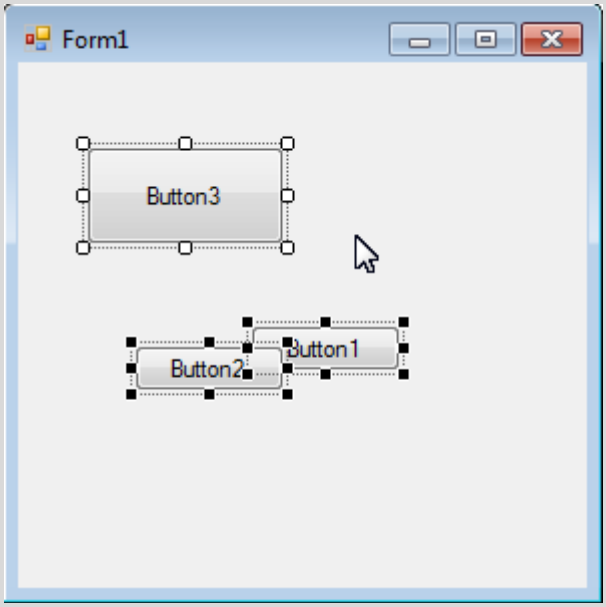


Selezioniamo i tre pulsanti sul Form1 tracciando attorno ad essi un rettangolo, con lo strumento **Puntatore**, come nella figura seguente[26].

[26] Metodo alternativo per selezionare un gruppo di controlli: fare un *clic* sul primo controllo che si desidera selezionare, poi premere il tasto MAIUSC e, tenendo premuto questo tasto, fare un *clic* su tutti gli altri controlli che si desidera selezionare. I controlli che rientrano nella selezione si distinguono dagli altri grazie alla visualizzazione delle maniglie di dimensionamento.



Dopo avere tracciato il rettangolo, rilasciamo il Puntatore e notiamo che i tre pulsanti selezionati mostrano le maniglie di dimensionamento. Le maniglie di dimensionamento sono di colore nero per tutti i controlli selezionati, salvo il primo controllo selezionato, o quello che VB considera come punto di riferimento, che ha le maniglie di dimensionamento di colore bianco:



Ora facciamo un *clic* sui comandi **Formato** / **Allinea** / **A sinistra** / **All'asse verticale** / **A destra** e vediamo l'effetto che questi comandi hanno sulla posizione dei tre pulsanti.

Facciamo un *clic* sui comandi **Formato** / **Rendi uguali** / **Larghezza** / **Altezza** / **Entrambi** e vediamo l'effetto che questi comandi hanno sulle dimensioni dei tre pulsanti.

Facciamo un *clic* sui comandi **Formato** / **Spaziatura orizzontale** / **Spaziatura verticale** e vediamo l'effetto che questi comandi hanno sulla posizione dei tre pulsanti.

Passiamo ora a esaminare i comandi degli ultimi menu.

Il menu Finestra.

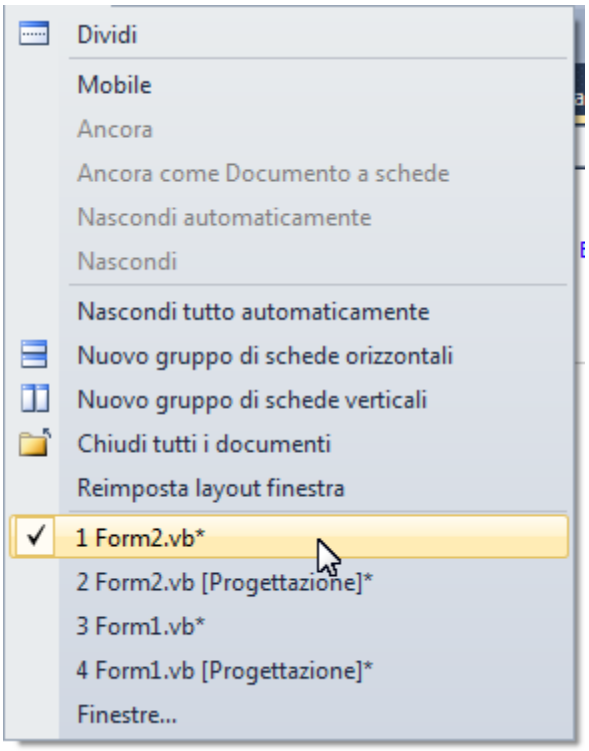


Figura 40: Il menu "Finestra".

Nel menu **Finestra** troviamo, in basso, i comandi per passare da un Form all'altro, e dalle Finestre di Progettazione alle Finestre del Codice, portando in primo piano sul desktop la finestra sulla quale vogliamo lavorare.
Importante, in questo menu, è il comando **Reimposta layout finestra**; esso si rivela utile quando, dopo avere modificato l'ambiente di progettazione di VB spostando o rimuovendo finestre, si desidera tornare all'impostazione iniziale.

Il menu "?".

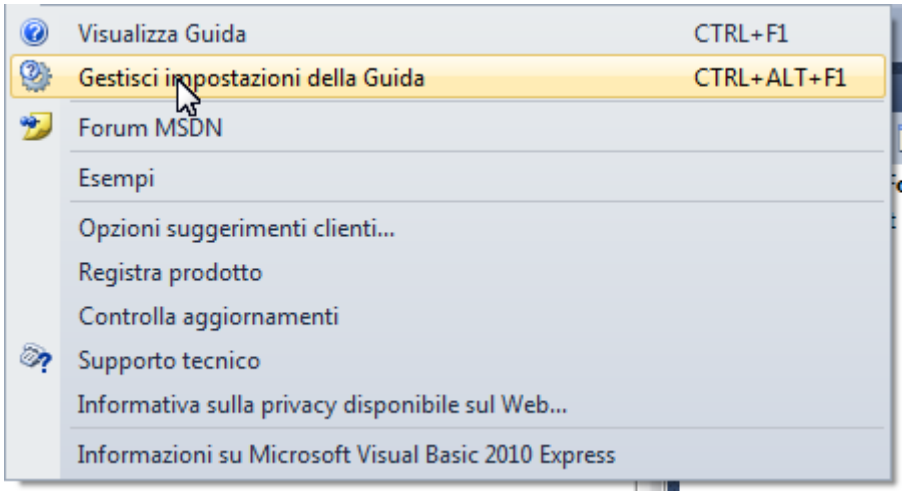
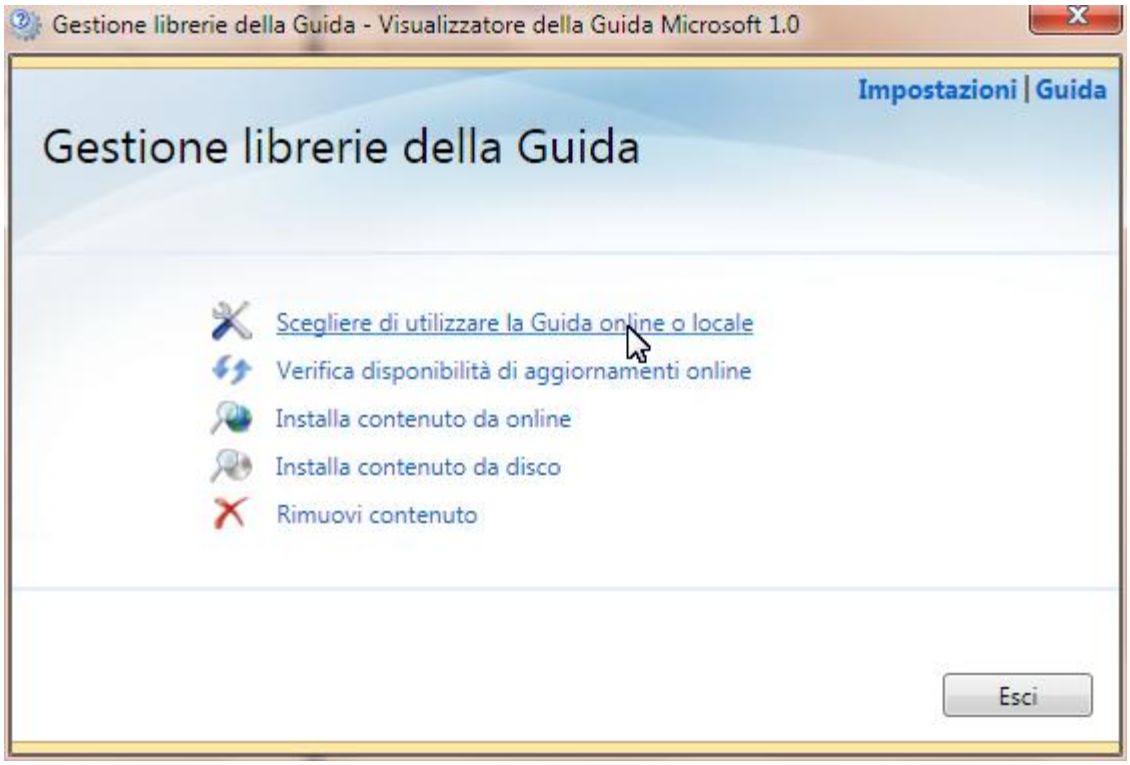


Figura 41: Il menu di documentazione "?".

L'ultimo menu, contrassegnato dal punto di domanda "**?**" è il menu della documentazione: esso dà accesso alla Guida di VB che contiene istruzioni, suggerimenti, spiegazioni tecniche consultabili durante il lavoro di programmazione.
Cliccando il comando **Gestisci impostazioni della Guida** è possibile scegliere tra la consultazione della guida online (in questo caso è necessario tenere attiva una connessione a internet) o la consultazione locale della guida installata con VB.



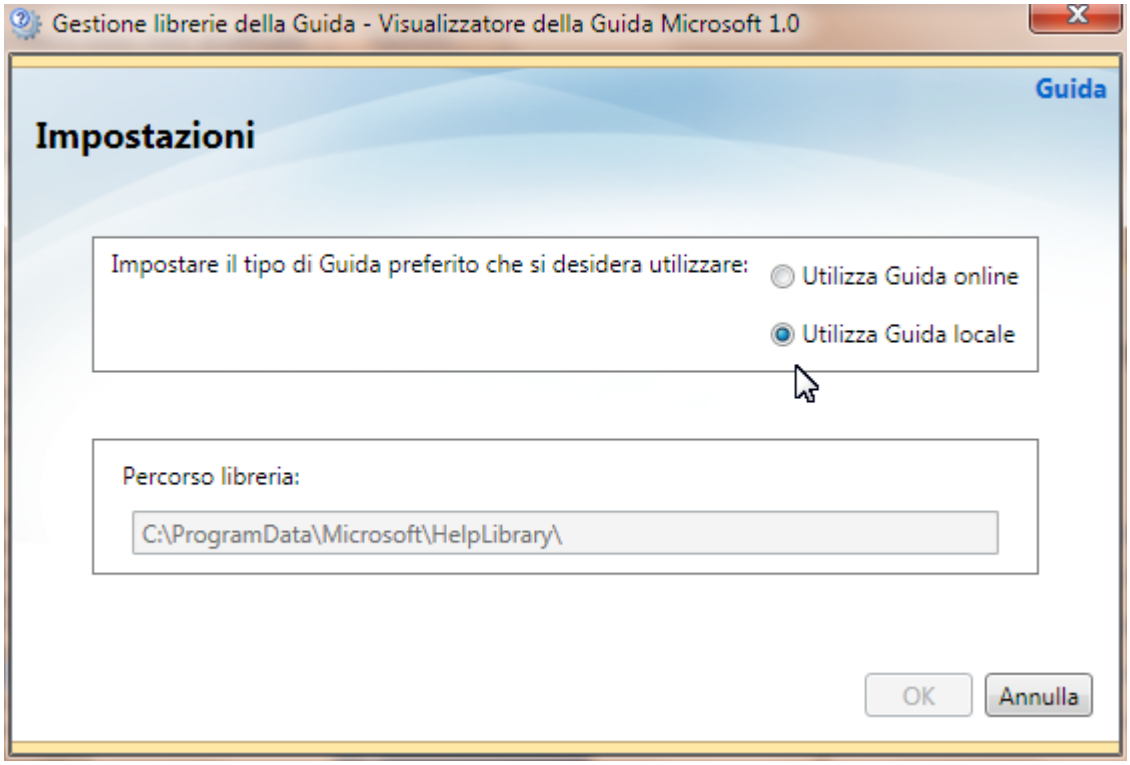


Figura 42: Impostazione della Guida di VB.

21: La striscia standard dei pulsanti.

Sotto la striscia dei menu troviamo la striscia standard dei comandi a icone.

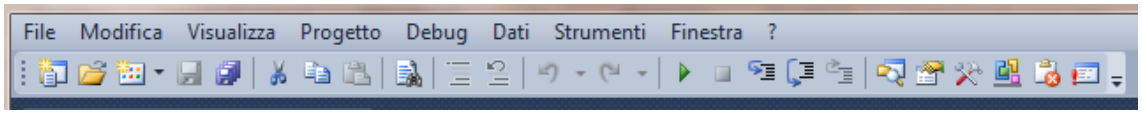
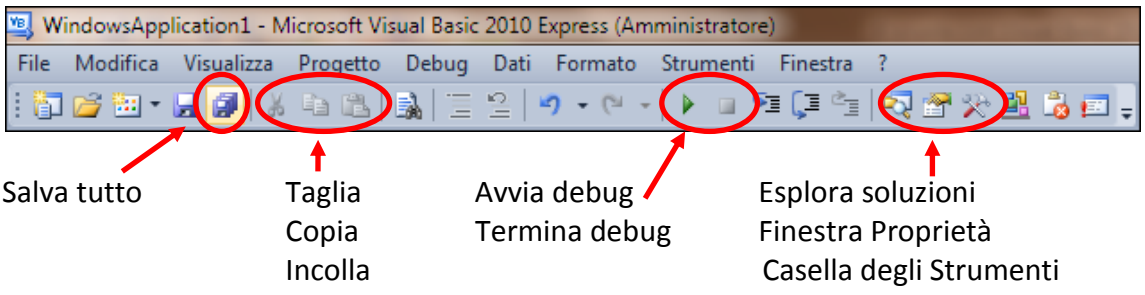


Figura 43: La striscia standard dei pulsanti.

Premendo queste icone si attivano gli stessi comandi che abbiamo visto nella striscia dei menu. Nella figura seguente sono evidenziati in rosso quelli di uso più frequente.



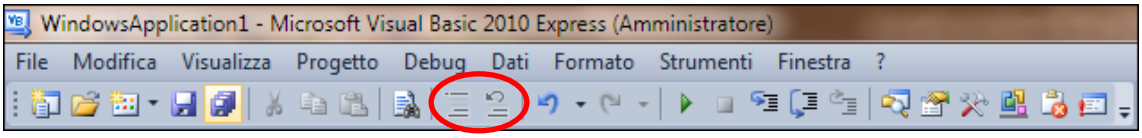


Figura 44: I comandi principali nella striscia standard dei pulsanti.

Notiamo in questa striscia la presenza di due icone particolari, che servono a **commentare** o **decommentare** le righe di istruzioni scritte nella finestra del codice.
Nel testo del codice di un programma, le righe precedute da un apostrofo sono righe di **commento** inserite dal programmatore per suo uso personale e sono ignorate dal programma.
Ecco un esempio:

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' queste righe precedute dall'apostrofo
        ' sono righe di commento a uso del porgrammatore

        Label1.Text = " Ciao Mondo"

    End Sub
```

Se il programmatore, per qualche ragione, ha l'esigenza di neutralizzare ampie porzioni di codice, in modo che il programma le ignori, può selezionare il codice in questione e **commentarlo** cliccando la prima delle due icone (**commenta**):

```
    'Private Sub Button1_Click(sender As System.Object, e As ystem.EventArgs)
Handles Button1.Click

    '    ' queste righe precedute dall'apostrofo
    '    ' sono righe di commento a uso del porgrammatore

    '    Label1.Text = " Ciao Mondo"

    'End Sub
```

In questo modo, tutta la procedura verrà ignorata dal programma, in fase di esecuzione.
Selezionando il testo e cliccando la seconda icona (**decommenta**) si ottiene ovviamente l'effetto contrario.

Questa che vediamo è la striscia **standard** dei pulsanti.
Vi sono altre strisce, il cui elenco è visibile facendo *clic* con il tasto **destro** del mouse a destra della striscia standard.
Queste strisce di pulsanti possono essere attivate di volte in volta, secondo le esigenze.
L'immagine seguente mostra l'attivazione della striscia dei pulsanti di **Layout** (visualizzazione), nella quale sono disponibili, uno per ogni pulsante, tutti i comandi del menu **Formato**.

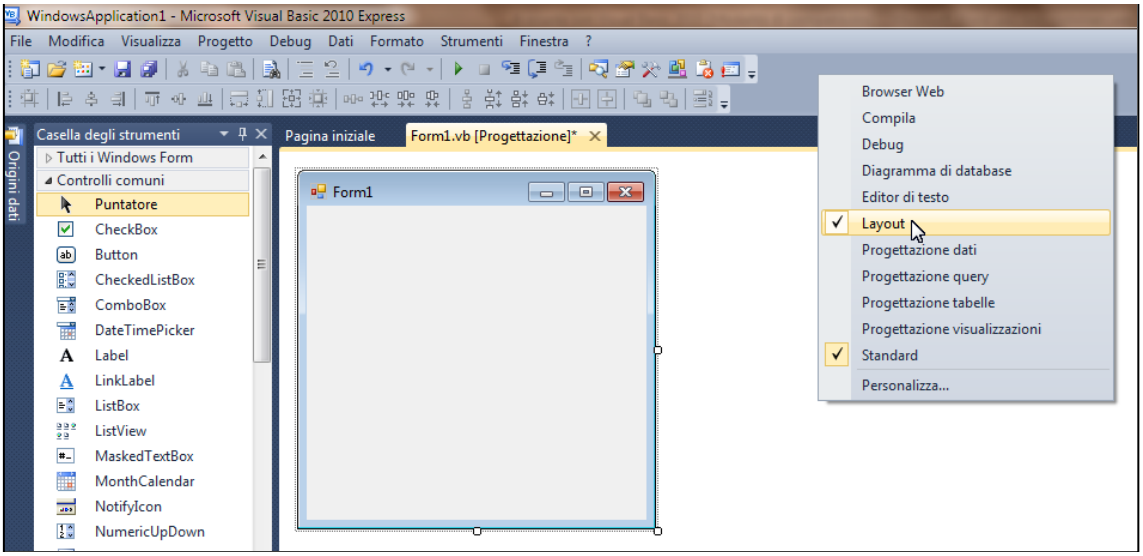


Figura 45: L'attivazione della striscia dei pulsanti di Layout.

22: I Menu di scelta rapida.

Abbiamo già avuto modo di notare in questo capitolo che VB offre molte possibilità per impartire lo stesso comando e per ottenere lo stesso risultato utilizzando i Menu nella seconda linea dell'ambiente di lavoro e utilizzando le icone presenti nella striscia dei pulsanti e in alcune delle finestre di VB;
Ma la gamma delle possibilità non si esaurisce qui, perché facendo un *clic* con il tasto **destro** del mouse su qualsiasi oggetto, nella **Finestra di Progettazione** grafica, si accede a un menu di scelta rapida che - tra altri comandi - consente di visualizzare direttamente la **Finestra del Codice** e la **Finestra Proprietà**.

Ricordiamo anche il quadro dei comandi collegati ai tasti funzione:

F4	Apre la Finestra Proprietà
F5	Avvia l'esecuzione provvisoria (**debug**) del progetto
F7	Apre la Finestra del Codice
MAIUSCOLO + F7	Chiude la Finestra del Codice

Tabella 2: I comandi collegati ai tasti funzione.

Capitolo 4: IL FORM: PROPRIETA', EVENTI E AZIONI.

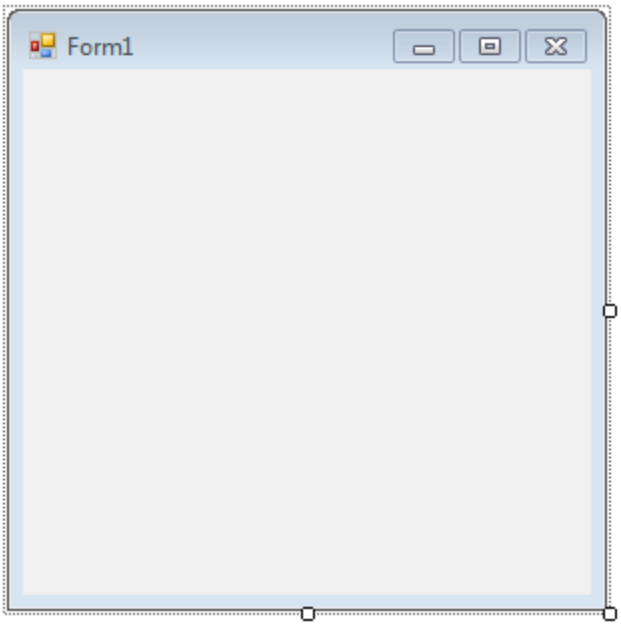


Figura 46: Un form vuoto all'apertura di un nuovo progetto.

Il form (= *modulo da riempire)* è l'oggetto principale di una nuova applicazione, il contenitore nel quale il programmatore collocherà gli elementi necessari al funzionamento del programma.
Quando il programma andrà in esecuzione, il form e il suo contenuto diventeranno **una finestra di Windows** del tutto simile alle altre finestre che popolano lo schermo di un monitor.
L'utente vedrà questa finestra e opererà in essa, ovviamente senza vedere il codice sottostante, cioè le istruzioni scritte dal programmatore che gli consentono di usare la finestra e gli oggetti in essa contenuti.

23: Proprietà, eventi e azioni.

Il form è destinato a contenere oggetti; esso stesso è un oggetto.
Come tutti gli oggetti è dotato di qualità (**proprietà**), è capace di riconoscere **eventi** ed è istruito a svolgere determinate **azioni** in risposta, ad esempio, agli eventi creati dall'utente del programma.
Il programmatore ha la facoltà di modificare le proprietà del form secondo le esigenze del suo programma, ed ha la facoltà di fare in modo che il form riconosca alcuni eventi e altri no, ed esegua alcune azioni e altre no.

Proprietà.

Le **proprietà** sono le **qualità** dell'oggetto, che lo definiscono e lo distinguono dagli altri oggetti simili che fanno parte del programma.
Sono proprietà di un oggetto:

- il suo nome proprio;
- la sua posizione sullo schermo;
- la sua altezza e la sua grandezza;
- il colore dello sfondo;
- l'immagine che eventualmente appare sullo sfondo o all'interno del form;
- le caratteristiche delle scritte che eventualmente appaiono sul form (colore del testo, tipo di caratteri, grandezza dei caratteri, stile dei caratteri);
- il fatto che esso sia visibile o no, sia attivo o no, sia spostabile sullo schermo oppure no, sia riducibile a icona, ecc.

Le proprietà degli oggetti vengono definite in fase di programmazione dal programmatore, utilizzando la **Finestra Proprietà**, ma possono essere modificate anche durante la fase di esecuzione del programma, se accadono determinati eventi, secondo le istruzioni scritte dal programmatore nella **Finestra del Codice**.
Per esempio, un form può avere il colore rosso assegnato come colore di fondo all'inizio di un programma, ma può cambiare questo colore più volte nel corso dell'esecuzione, oppure può diventare invisibile, può diventare inattivo, può diventare più grande o più piccolo, può cambiare stile e grandezza delle scritte, ecc.

Eventi.

Gli **eventi** sono *fatti che succedono* nel corso di svolgimento di un programma, generalmente in conseguenza di azioni compiute dall'utente del programma.
L'oggetto viene istruito dal programmatore a riconoscere alcuni di questi fatti, e a rispondere compiendo determinate azioni.

Un form, ad esempio, riconosce questi eventi:
- **Click** (l'utente ha fatto un *clic* con il tasto sinistro del mouse);
- **DoubleClick** (l'utente ha fatto un doppio *clic* con il mouse);
- **MouseMove** (l'utente ha fatto passare il mouse sopra il form senza premere alcun pulsante);
- **DragOver** (l'utente ha trascinato il mouse sopra il form con il tasto sinistro del mouse premuto)
- **KeyDown** (l'utente ha premuto un tasto sulla tastiera).

In risposta agli eventi, il form può cambiare le sue proprietà, oppure può svolgere le **azioni** che è stato programmato a compiere.
Esempi:
- Se avverti il *clic* del mouse sopra di te, fai diventare il tuo sfondo di colore giallo.
- Se avverti il *clic* del mouse sopra di te, diventa invisibile.

Nei due esempi, l'evento è identico (il *clic* del mouse).
Nel primo esempio, al *clic* del mouse segue il cambiamento di una proprietà (è la proprietà **BackColor** che assume il colore giallo).
Nel secondo esempio, al *clic* del mouse segue una **azione** da parte del form (è l'azione **Hide**: il form si rende invisibile).

Azioni.

Ogni oggetto è stato programmato per compiere un determinato numero di **azioni**, che possono essere o meno attivate dal programmatore, secondo le sue esigenze.
Un form, ad esempio, può compiere queste azioni:
- **Beep** (esegue un bip sonoro in risposta a un evento);
- **Close** (chiude il programma);
- **Hide** (il form si nasconde durante l'esecuzione del programma);
- **Show** (il form si mostra in un momento determinato, durante l'esecuzione del programma);
- **Refresh** (il form aggiorna il suo contenuto grafico tenendo conto delle ultime variazioni sopravvenute).

24: Le proprietà del form nella fase di progettazione.

Apriamo un nuovo progetto e, seguendo le istruzioni per l'avvio del progetto "Ciao Mondo", creiamo una nuova **Applicazione Windows Form.**
Le diamo il nome **Studio del form**, perché la useremo per studiare proprietà, eventi e azioni del form e per esercitarci al loro uso:

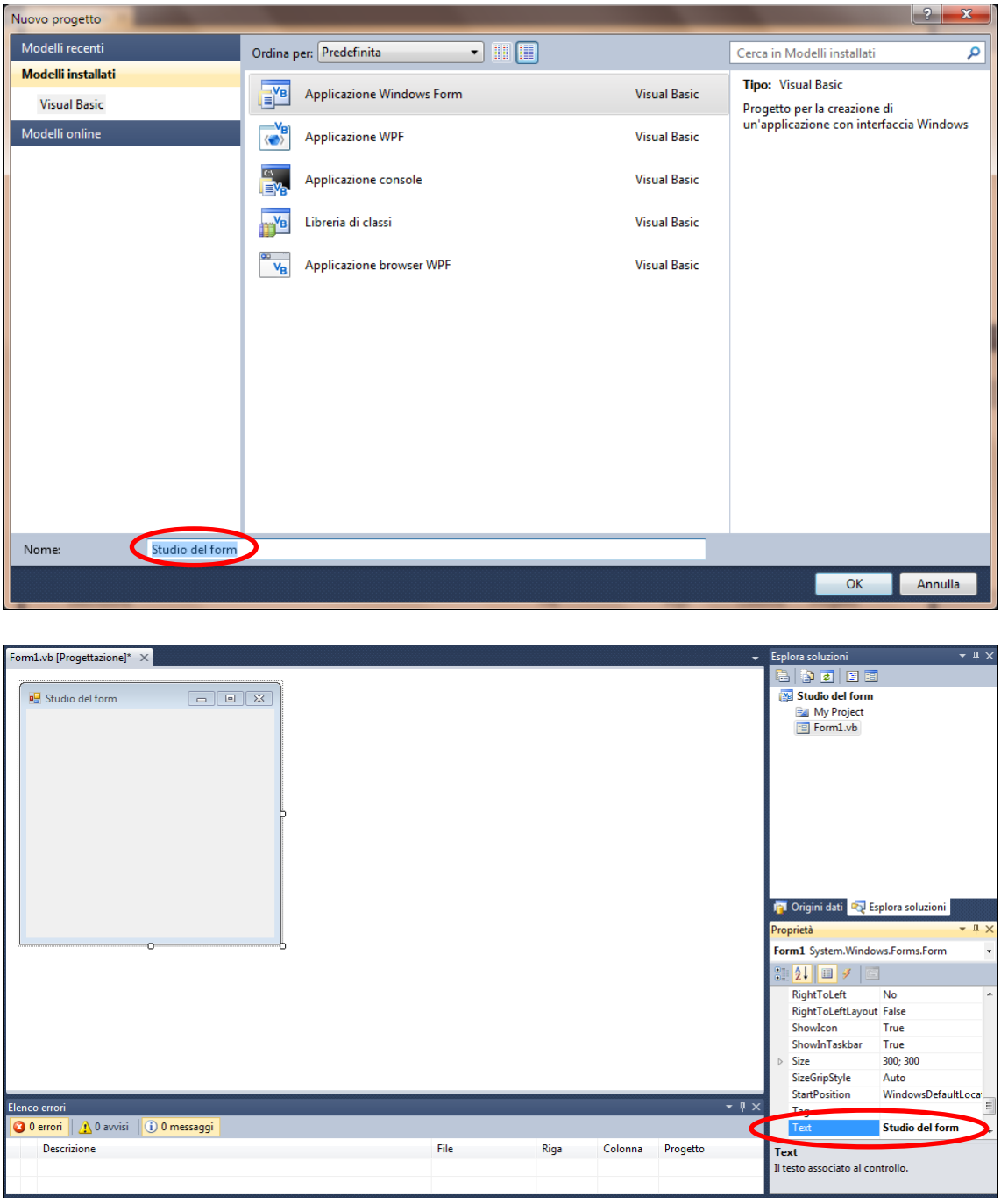


Figura 47- L'ambiente di progettazione all'avvio del progetto "Studio del form".

Facciamo un *clic* sul Form1: nella Finestra Proprietà compare automaticamente l'elenco delle proprietà del form. Si tratta di un elenco piuttosto lungo, che può essere scorso con la barra di scorrimento verticale.
Notiamo che la Finestra Proprietà si apre in modo automatico all'altezza della proprietà **Text**, che per ora riporta la dicitura **Form1**.
La proprietà Text è **l'intestazione** del Form1: ciò che è scritto nella proprietà Text compare come **titolo** dello stesso form, nella barra in alto.
Siccome useremo questo Form1 per studiarne proprietà eventi e azioni, gli assegniamo il titolo **Studio del form** e lo scriviamo nella proprietà **Text**.

Ora modifichiamo la proprietà **Name**, cioè il nome proprio, di questo form.
Questa è la prima proprietà della lista (la sua posizione al di fuori dell'ordine alfabetico ne segnala l'importanza).
Il nome proprio che appare di default (Form1) può essere modificato come si desidera.
Sostituiamo **Form1** con il nome **frmStudioForm**[27]:

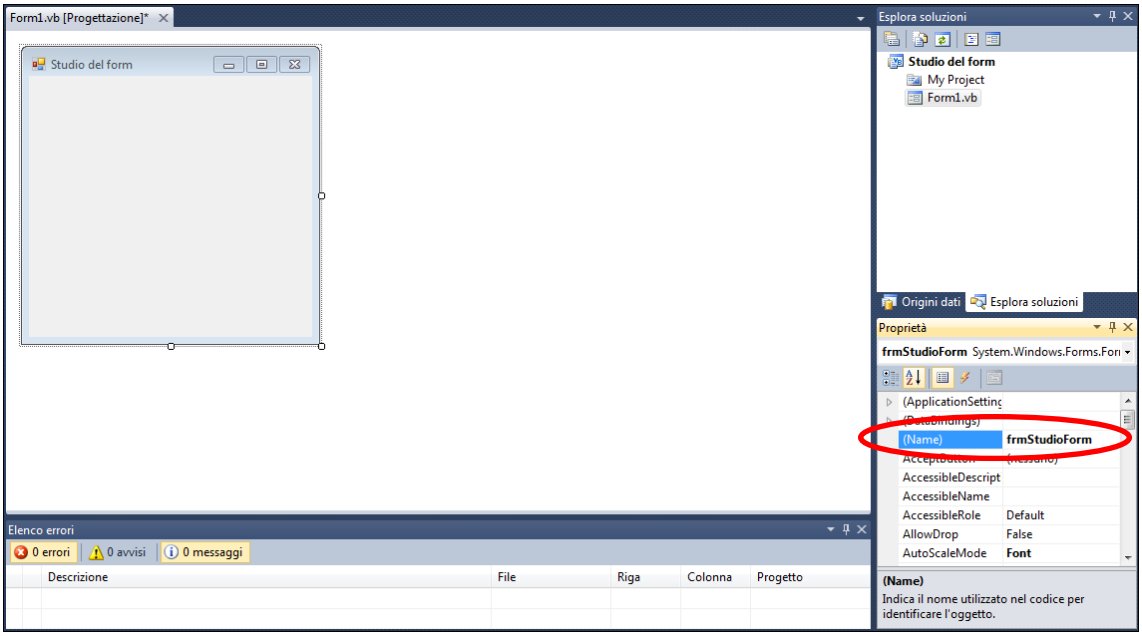


Figura 48: Modifica della proprietà Name del Form1.

La dicitura Form1, eliminata dalla proprietà **Text** e dalla proprietà **Name**, rimane visibile ancora nella finestra Esplora Soluzioni, dove troviamo il file Form1.vb.
Se non intervengono modifiche, dunque, VB salverà automaticamente il contenuto di questo form al quale stiamo lavorando con il nome Form1.vb[28].

[27] E' buona abitudine, quando si assegna un nome proprio a un oggetto, usare le prime tre consonanti del nome comune dell'oggetto, scritte in minuscolo, aggiungendo poi il nome proprio che si vuole dare all'oggetto. Esempio: a un form si può assegnare il nome proprio frmStudioForm, come in questo caso; a un pulsante Button si può assegnare il nome proprio btnEsci; a una Label (etichetta) si può assegnare il nome proprio lblDidascalia.

Per modificare il nome di questo file, facciamo *clic* con il tasto destro del mouse sul nome del file. Nel menu di scelta rapida che compare, facciamo un *clic* su **Rinomina** e rinominiamo il file **frmStudioForm.vb** (attenzione a non modificare l'estensione del file .vb, altrimenti il file non sarà più riconoscibile da VB).

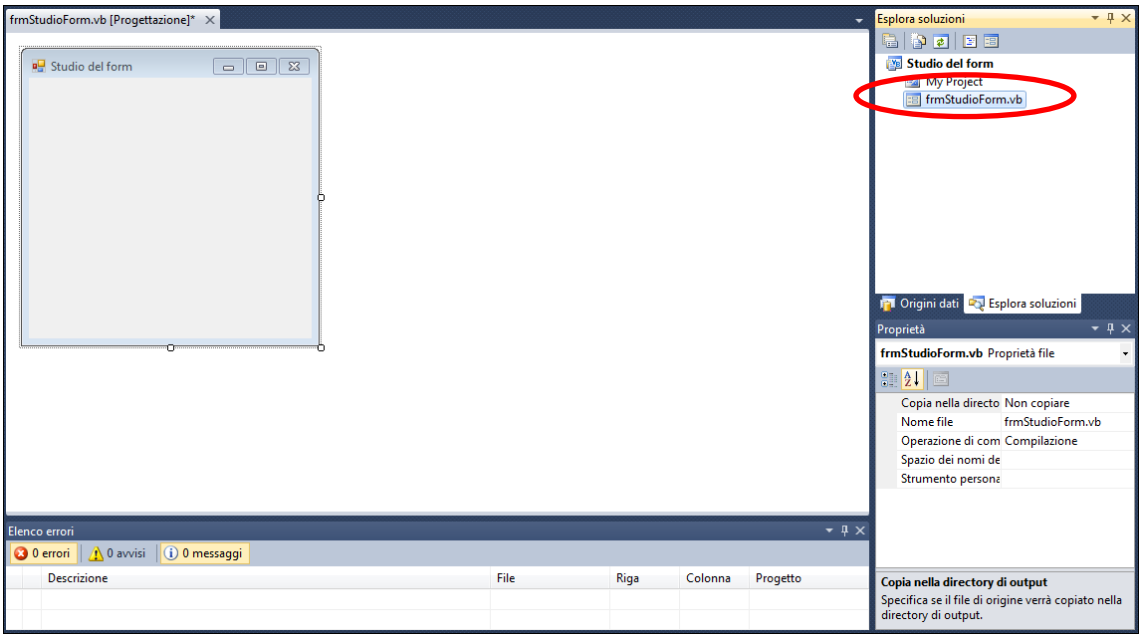


Figura 49: Il file rinominato frmStudioForm.vb nella finestra Esplora Soluzioni.

Facciamo *clic* sull'icona **Salva tutto**, nella striscia dei pulsanti.
VB salva il lavoro che abbiamo fatto sino a questo momento in modo automatico: possiamo vedere i file creati da VB nella cartella Documenti / Visual Studio 2010 / Projects / Studio del form.

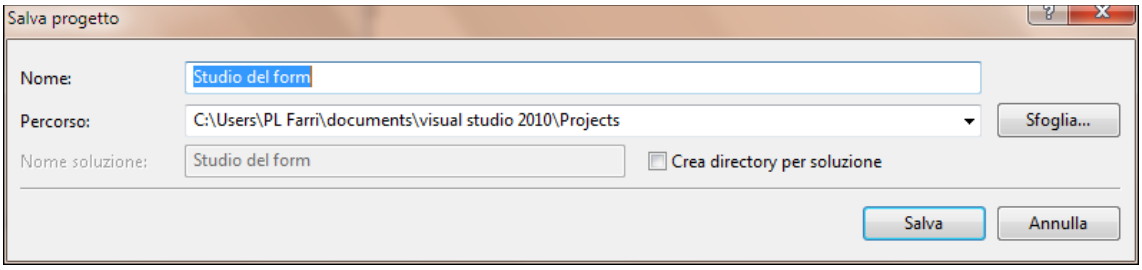


Figura 50: La procedura di salvataggio del progetto "Studio del form".

Facciamo un *clic* sul form **frmStudioForm** in modo da riattivarlo, per visualizzare di nuovo le sue proprietà.
Scorriamo verso il basso la lista delle sue proprietà.

---

[28] Il progetto **Studio del form** e tutti i file che lo compongono vengono salvati da VB nella cartella Documenti / Visual Studio 2010 / Projects / Studio del form.

Notiamo che cliccando ognuna di queste proprietà compare nello spazio grigio in basso una breve descrizione delle sue caratteristiche.

Procediamo, seguendo l'ordine alfabetico, a modificare altre proprietà.

La proprietà **BackColor** si riferisce al colore di sfondo del form. Modificando questa proprietà si modifica il colore dello sfondo del form.

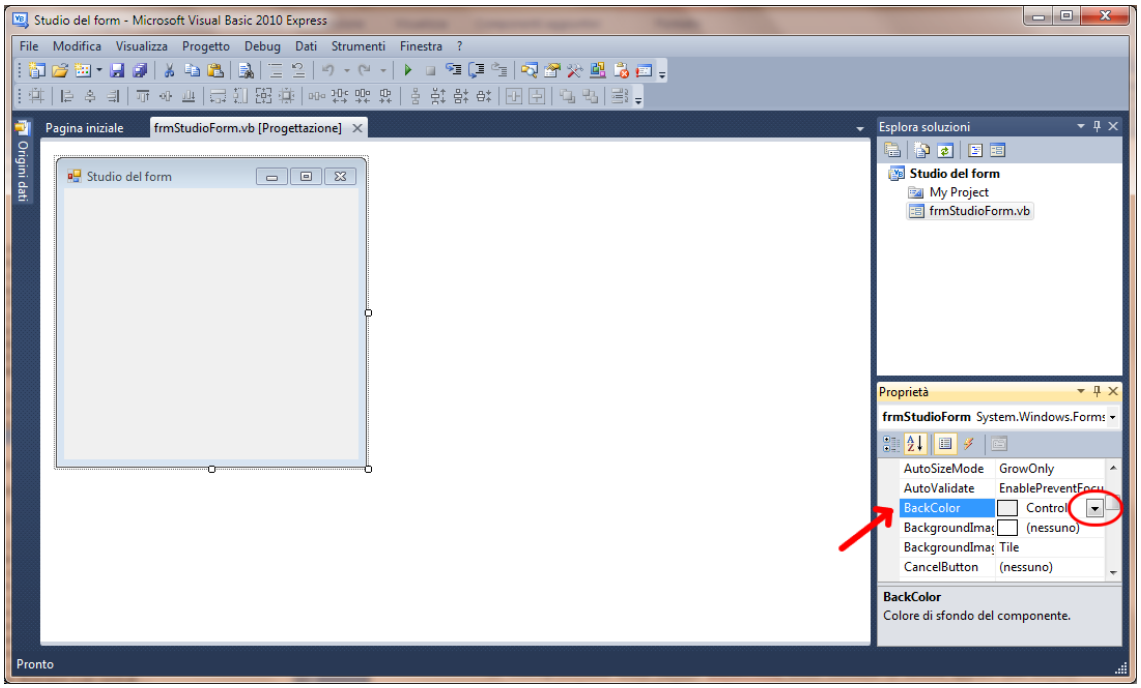


Figura 51: Impostazione della proprietà BackColor.

In corrispondenza della proprietà **BackColor**, nella colonna di destra appare una freccia nera orientata in basso, a indicare che qui sono disponibili delle opzioni tra le quali il programmatore può scegliere quella che preferisce. Apriamo il menu di queste opzioni e scegliamo, nella scheda "Personalizzato", il colore **giallo**. Osserviamo l'effetto di questo cambiamento nel form, poi proviamo con altri colori, ma **alla fine lasciamo come colore di sfondo il colore giallo.**

Nella riga successiva alla proprietà BackColor, troviamo le proprietà

- **BackGroundImage** (immagine di sfondo) e
- **BackGroundImageLayout** (modalità di presentazione della immagine di fondo).

Premesso che per visualizzare un'immagine è preferibile usare il controllo **PictureBox** (= casella immagine), vediamo com'è possibile visualizzare un'immagine nel form.
Facciamo *clic* con il mouse sulla proprietà **BackGroundImage** e notiamo che nella casella a destra appare un pulsante con tre puntini. I tre puntini indicano che per questa proprietà vi sono molte possibilità di scelta.
Facciamo un *clic* sul pulsante con i tre puntini: compare la finestra **Seleziona risorsa**, mediante la quale è possibile selezionare l'immagine che vogliamo inserire nel form.

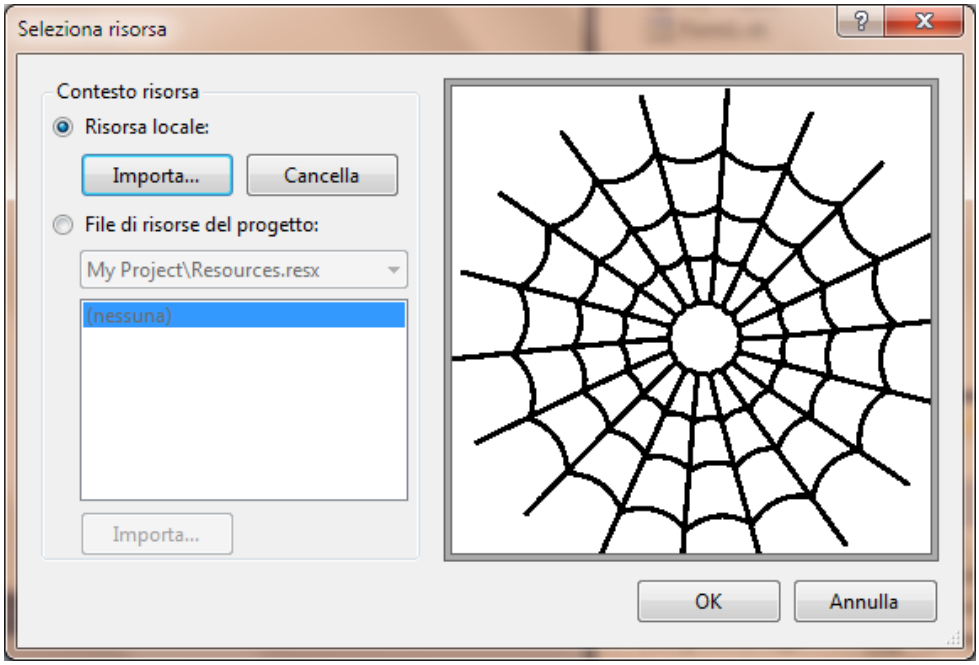


Figura 52: La Finestra di selezione di una risorsa da inserire in un progetto.

In questa finestra facciamo un *clic* sulla opzione **Risorsa locale** e sul pulsante **Importa…** Andiamo alla ricerca della immagine **Ragnatela**, che troviamo nella cartella Documenti \ A scuola con VB \ Immagini e, dopo averla selezionata, facciamo un *clic* sul pulsante OK. Ora mandiamo in esecuzione il programma cliccando l'icona **Debug**. Vediamo l'immagine della ragnatela che copre tutto il form, per cui il colore giallo dello sfondo rimane invisibile.

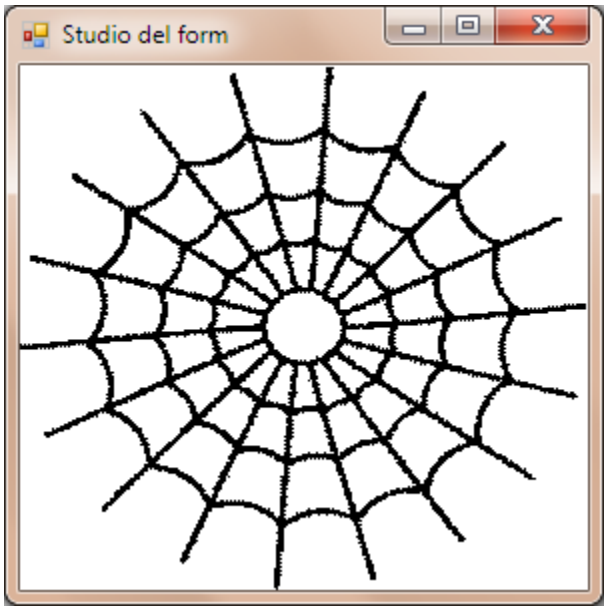


Figura 53: Il progetto "Studio del form" in esecuzione.

Ora facciamo un *clic* sul pulsante per massimizzare la finestra del nostro programma in esecuzione:

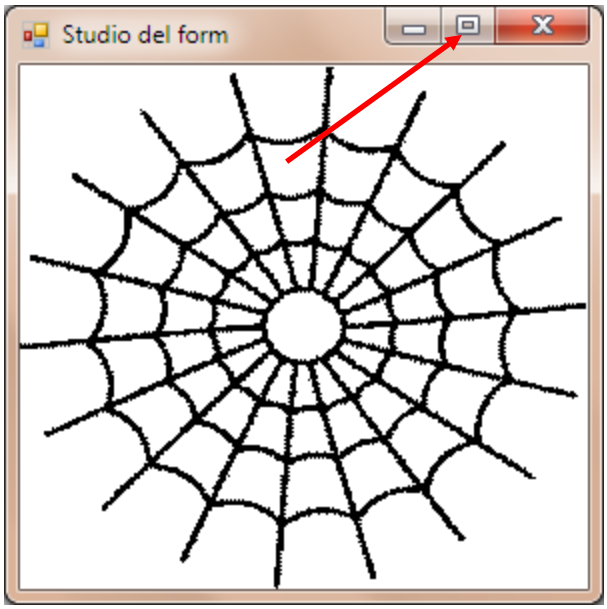


Figura 54: Il pulsante pe massimizzare le dimensioni del form.

Vediamo che l'immagine della ragnatela viene ripetuta più volte, sino a coprire tutto il form, come se fosse la mattonella di un pavimento.

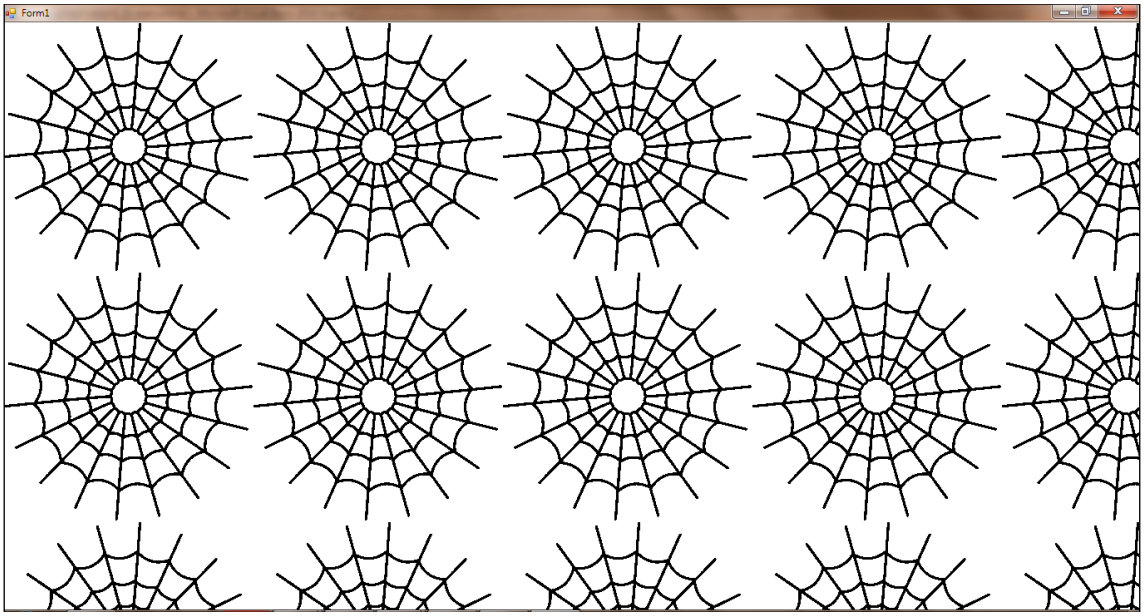
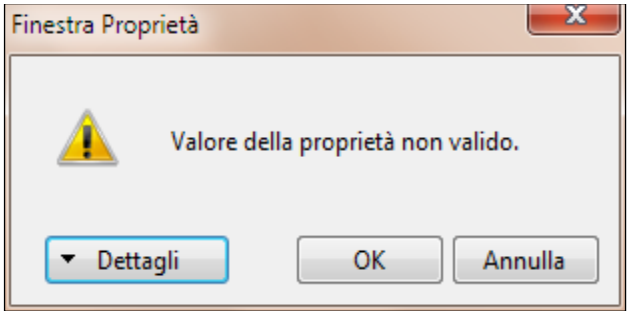


Figura 55: La proprietà BackGroundImageLayout = Tile.

Questo effetto grafico è dovuto alla proprietà **BackGroundImageLayout**, che in questo momento è impostata sulla opzione **Tile** (= mattonella).

Fermiamo il progetto in esecuzione cliccando l'icona **Termina debug** e cambiamo la proprietà **BackGroundImageLayout** impostandola sulla opzione **Center** (= centra l'immagine nel form).

Attenzione: quando si tenta di modificare una proprietà di un oggetto, mentre il programma è in fase di esecuzione, compare questo messaggio di errore:



Per procedere è necessario fermare l'esecuzione del programma cliccando l'icona **Termina debug**.

Mandiamo nuovamente in esecuzione il nostro progetto con l'impostazione della proprietà **BackGroundImageLayout = Center**, facciamo un *clic* sul pulsante per massimizzare la finestra e vediamo l'immagine della ragnatela centrata nel form:

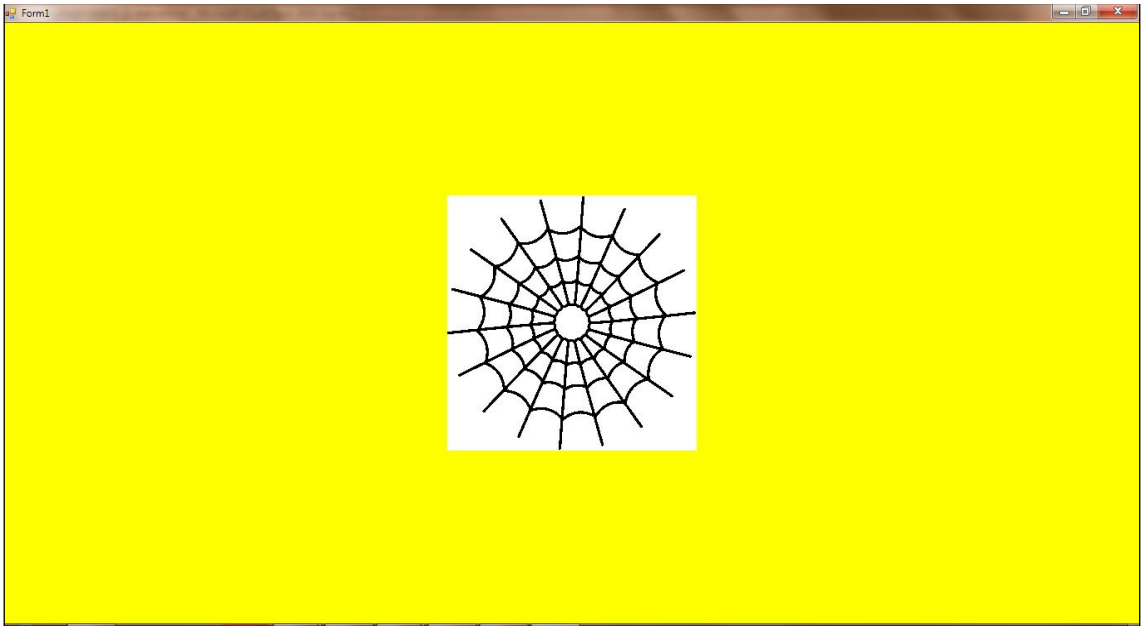


Figura 56: La proprietà BackGroundImageLayout = Center.

Fermiamo l'esecuzione del programma, e rimandiamolo in esecuzione dopo avere impostato la proprietà **BackGroundImageLayout = Stretch**. Ripetiamo l'operazione con

la proprietà **BackGroundImageLayout = Zoom** e vediamo i diversi effetti grafici che si possono ottenere cambiando la proprietà **BackGroundImageLayout.**

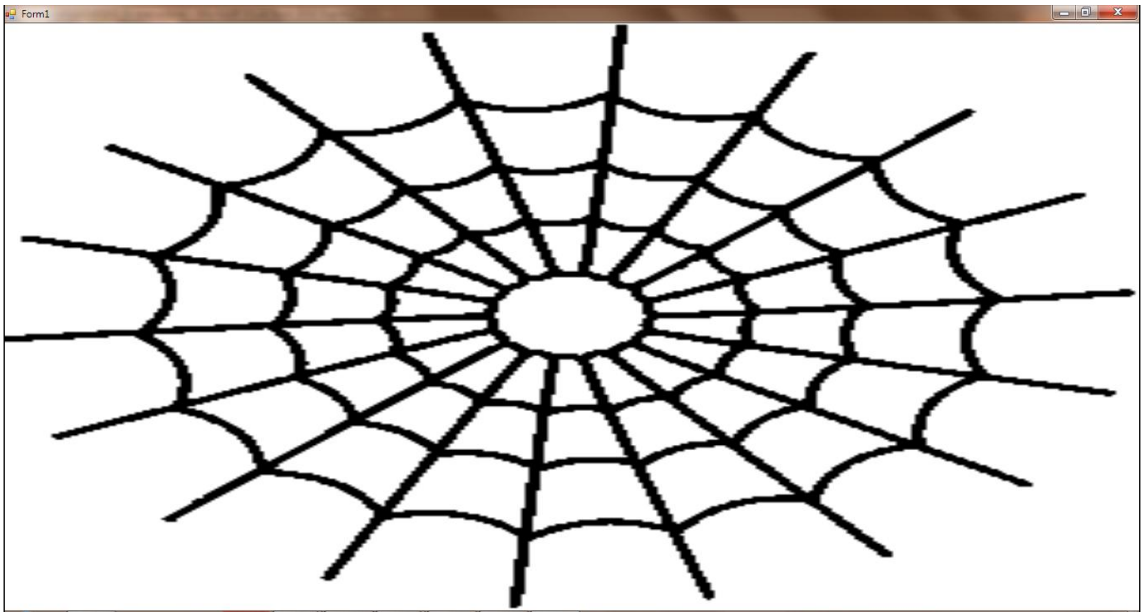


Figura 57: La proprietà BackGroundImageLayout = Stretch.

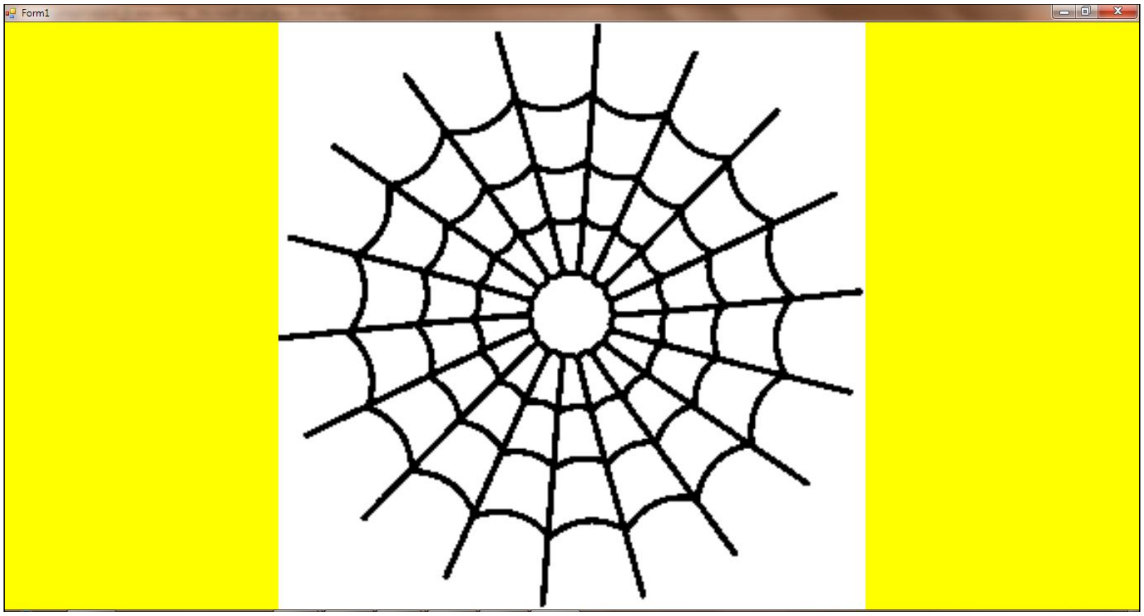
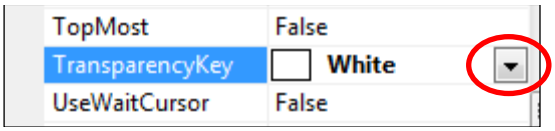


Figura 58: La proprietà BackGroundImageLayout = Zoom.

Fermiamo l'esecuzione del programma, e nell'elenco delle proprietà del form cerchiamo la proprietà **TransparencyKey**. Mediante la proprietà **TransparencyKey è** possibile scegliere un colore - presente nel form - che nella fase di esecuzione del programma rimarrà **trasparente.**

Scegliamo come colore trasparente il colore bianco, cliccando la freccina nella casella a destra della proprietà **TransparencyKey.**



Mandiamo in esecuzione il programma e spostiamo il form sul monitor, trascinandolo con il mouse. Notiamo che la ragnatela rimane visibile, ma il colore bianco tra i fili della ragnatela è trasparente e lascia intravedere gli oggetti sottostanti.

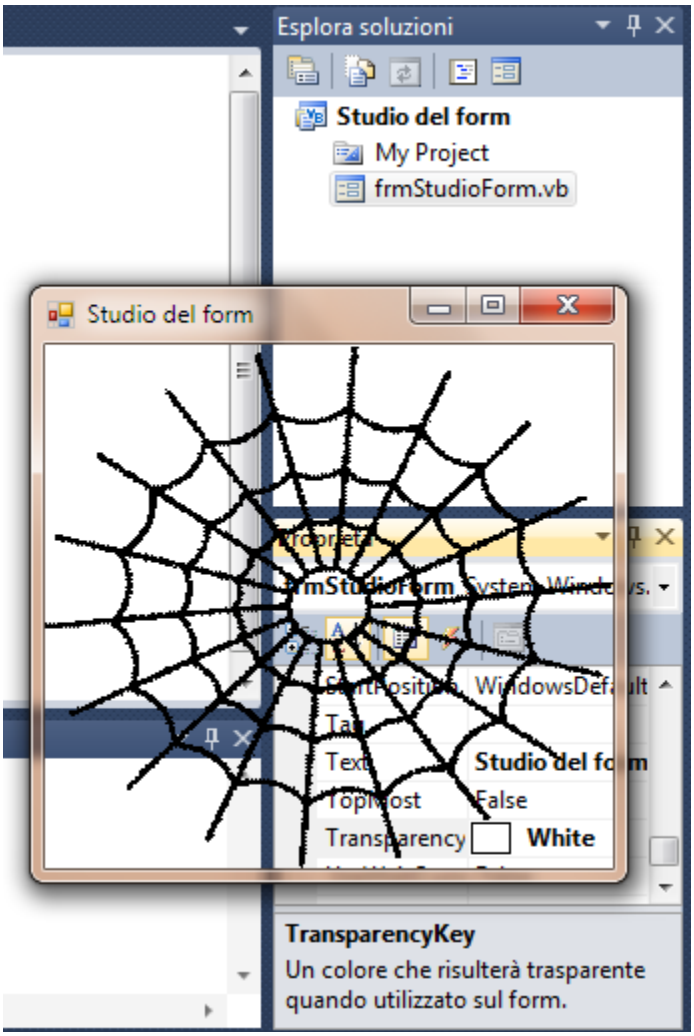
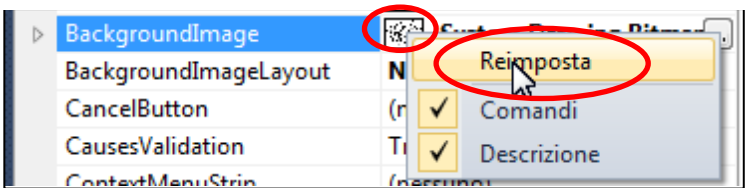


Figura 59: La proprietà TransparencyKey = White.

Vedremo ora un altro effetto della proprietà **TransparencyKey**.
Eliminiamo dal form l'immagine della ragnatela.
Per togliere l'immagine bisogna tornare alla proprietà **BackgroundImage** e possiamo agire in tre modi diversi:

- clic con il tasto **destro** del mouse sul rettangolino con l'icona della immagine della ragnatela, e *clic* sul comando **Reimposta**;



- doppio *clic* sulla scritta **System.Drawing.Bitmap**; quando la scritta è tutta evidenziata in blu è possibile cancellarla premendo il tasto CANC sulla tastiera;
- clic sul pulsante con i tre puntini; nella finestra **Seleziona risorsa** facciamo un *clic* sul pulsante **Cancella**:

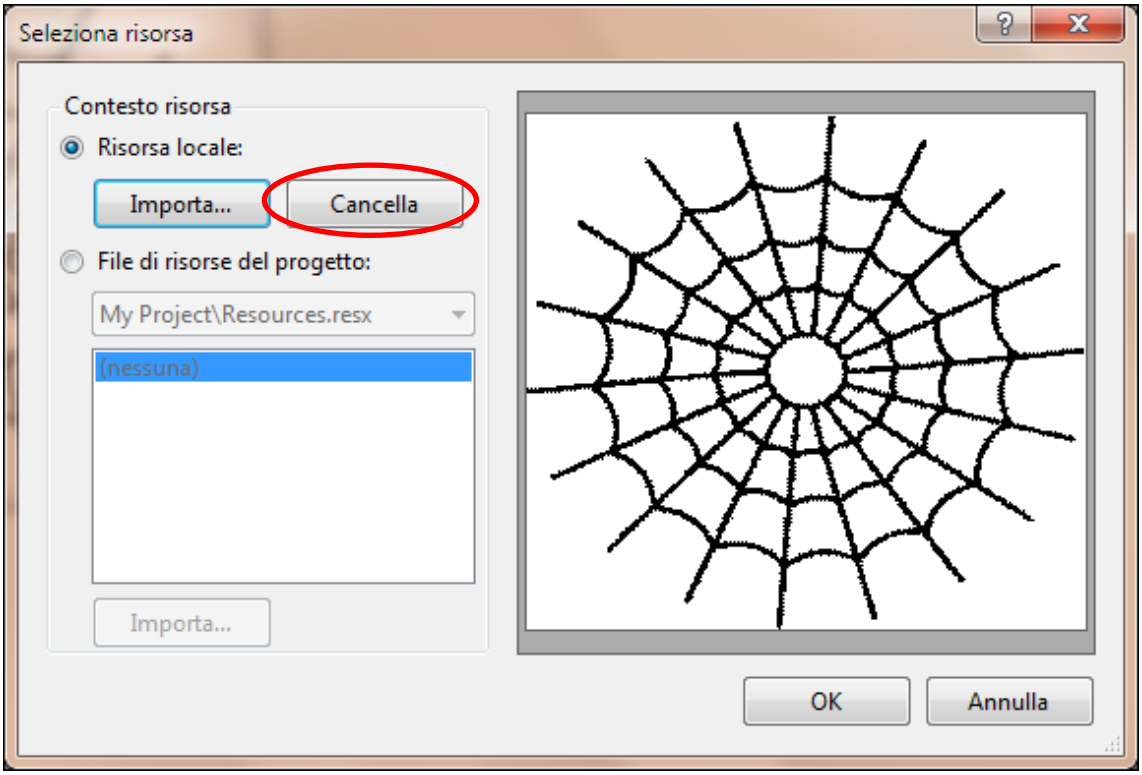
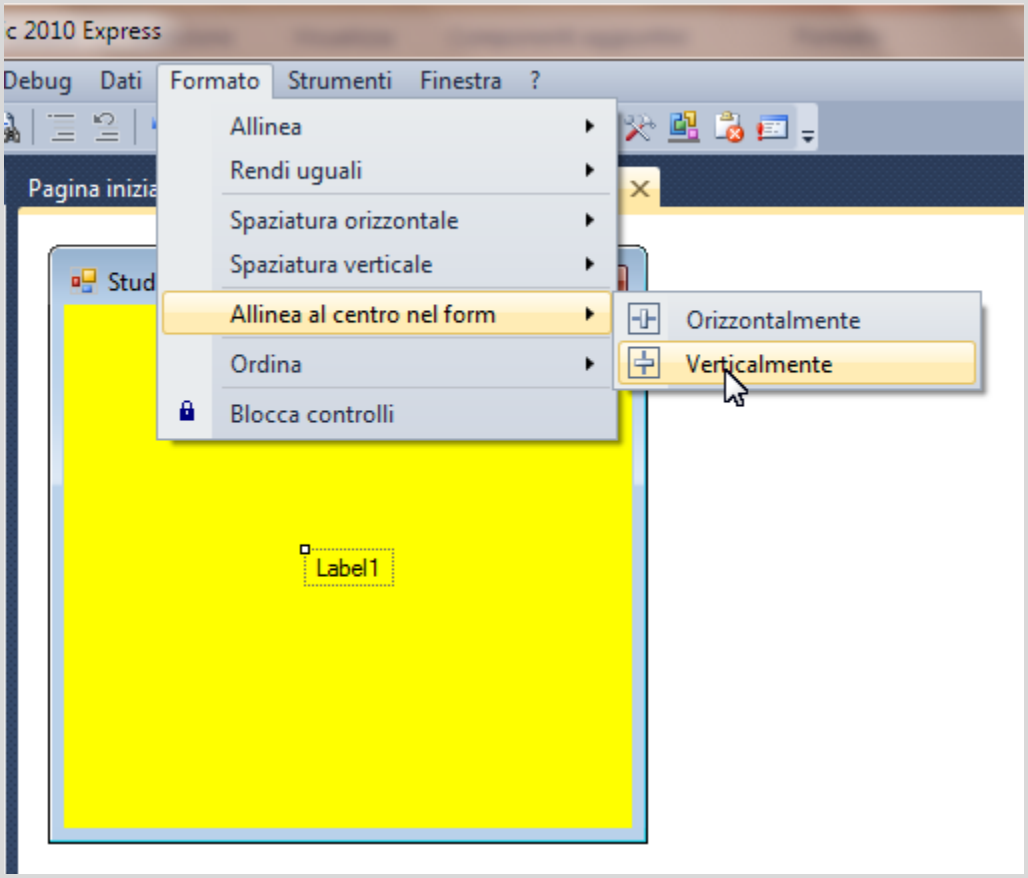


Figura 60: Cancellazione di un'immagine dalla finestra Selezione risorsa.

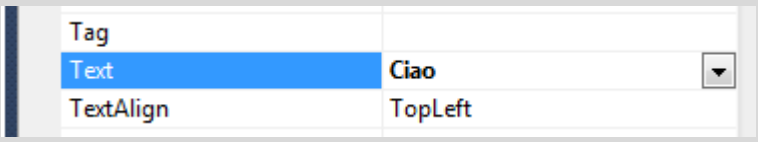
In questo modo torniamo a visualizzare il form senza alcun'immagine, con il colore giallo di sfondo (ricordiamo che la proprietà **Backcolor** è impostata sul colore **giallo**).
Il prossimo esercizio è dedicato ad alcuni effetti grafici che si possono ottenere impostando la proprietà **TransparencyKey** del form.

Esercizio 5: La proprietà TransparencyKey del form.

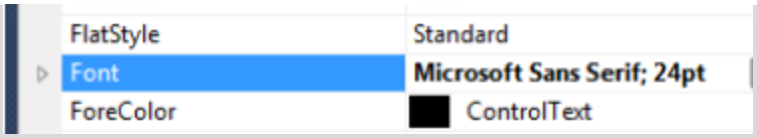
Apriamo la **Casella degli Strumenti** e selezioniamo un controllo **Label** (etichetta di testo).
Collochiamo la **Label** al centro del form utilizzando il menu **Formato / Allinea al centro nel form**.



Facciamo un *clic* sul controllo **Label1** e andiamo a modificare alcune sue proprietà.
Nella proprietà Text del controllo Label1 scriviamo **Ciao**:



Nella proprietà **Font** (carattere) del controllo Label1 scegliamo il Font Microsoft Sans Serif a 24 punti di grandezza: nella casella a destra della proprietà Font compare un pulsante con tre puntini. I tre puntini stanno a indicare che per questa proprietà vi sono molte possibilità di scelta. Facciamo un *clic* su questi puntini e nella Finestra che si apre scegliamo il carattere **Microsoft Sans Serif** a **24** punti di grandezza, procedendo come si fa all'interno di un normale elaboratore di testi.



Ora il form **frmStudioFrm** si presenta così:

Facciamo un *clic* sul form, all'esterno della Label1.
Vediamo visualizzate di nuovo le proprietà del form. Impostiamo la proprietà TransparencyKey del form sul colore giallo.

Mandiamo in esecuzione il progetto e, trascinando il form sul monitor con il mouse, vediamo che il colore giallo dello sfondo del form è diventato trasparente: rimane visibile solo la scritta **Ciao** della **Label1**.

Procediamo ora con l'analisi e la modifica di altre proprietà del form **frmStudioForm**. Fermiamo l'esecuzione del programma (clic sull'icona **Termina debug**), ed eliminiamo la scelta del colore nella proprietà **TransparencyKey**: *clic* con il tasto **destro** del mouse sul rettangolino giallo, e *clic* sul comando **Reimposta**.

Sino a ora abbiamo visto cosa sono e come si possono modificare le proprietà:

- **Text**
- **Name**
- **BackColor**
- **BackgroundImage**
- **BackgroundImageLayout**
- **TransparencyKey**

Notiamo che le proprietà che abbiamo modificato compaiono scritte in **grassetto** nella **Finestra Proprietà**.

Scorriamo l'elenco e giungiamo alla proprietà **FormBorderStyle**.
Questa proprietà ha una importanza particolare, perché determina il modo con il quale la barra del titolo di questa finestra si presenterà all'utente; in particolare, determina la presenza o meno in essa dei tre pulsanti che normalmente sono disponibili per ridurla, ampliarla o chiuderla.
La proprietà **FormBorderStyle** presenta queste opzioni:

None	Nessun bordo. La finestra si presenterà all'utente del programma senza alcun bordo e senza la barra in alto. Scegliendo questa opzione mancherà nel form anche il pulsante per chiudere la finestra.
FixedSingle	La finestra si presenterà all'utente del programma con la barra del titolo completa di tutti i pulsanti; icona, titolo, pulsante di riduzione, pulsante di ampliamento, pulsante di chiusura. L'utente non avrà tuttavia la possibilità di dimensionare la finestra utilizzando il mouse.
Fixed3D	Idem come la modalità precedente **FixedSingle**, con l'accentuazione dell'effetto visivo tridimensionale.
FixedDialog	Idem come la modalità precedente **FixedSingle**, ma sulla barra del titolo non comparirà l'icona.
Sizable	Idem come la modalità **FixedSingle**, ma in questo caso l'utente avrà anche la possibilità di modificare le dimensioni della finestra, trascinandone i bordi con il mouse. Questa è l'opzione che lascia il maggior numero di possibilità di azione all'utente.
FixedToolWindow	La finestra si presenterà all'utente del programma come una finestra di lavoro: nella barra del titolo avrà solo il titolo della finestra e il pulsante per chiuderla.
SizableToolWindow	La finestra si presenterà all'utente del programma come una finestra di lavoro, come con la proprietà precedente: nella barra del titolo si vedranno solo il titolo della finestra e il pulsante per chiuderla, ma l'utente avrà la possibilità di ridimensionare la finestra trascinandone i bordi con il mouse.

Tabella 3: Le opzioni della proprietà BorderStyle del form.

Proviamo le diverse possibilità, cambiando la proprietà e mandando ogni volta in esecuzione il programma.
Al termine, lasciamo la proprietà **FormBorderStyle** impostata su **FixedSingle**.

Poco sotto la proprietà **FormBorderStyle** troviamo la proprietà **Icon** (icona).
L'icona del form è la piccola immagine che appare sulla linea del titolo del form.
La proprietà **Icon** è già impostata da VB con una icona standard, utilizzata per tutti i form. Per cambiare l'icona standard bisogna cliccare il pulsante con i tre puntini, a destra nella riga delle proprietà Icon, e andare a scegliere una icona diversa.
Questa icona è un elemento grafico che potrà essere cliccato dall'utente del programma (solo in fase di esecuzione del progetto, non in fase di progettazione) per accedere a un menu di scelta rapida, con opzioni per visualizzare il form o chiudere il programma:

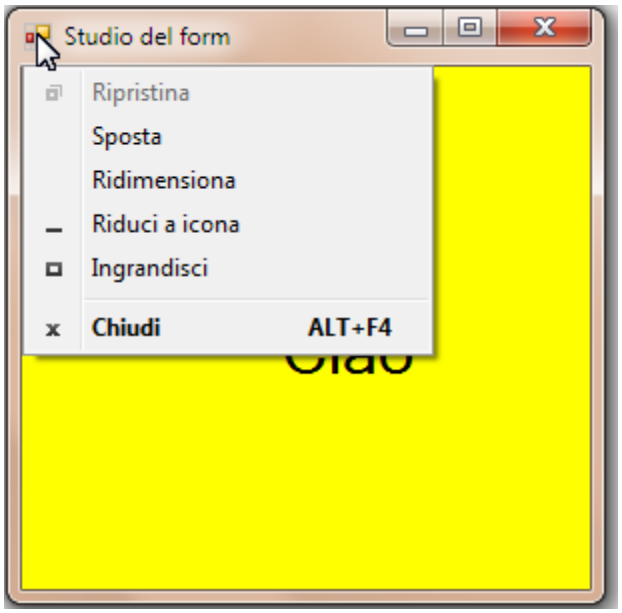


Figura 61: Il menu a scelta rapida che si apre dall'icona del form in fase di esecuzione.

Analizziamo ora le proprietà **Size** e **StartPosition** del **frmStudioForm**.

- La proprietà **Size** imposta le dimensioni del form. Sino a questo momento il form **frmStudioForm** ha avuto una forma quadrata, di 300 pixel di larghezza e 300 pixel di altezza.
  La forma classica delle finestre dei programmi è rettangolare, con un rapporto larghezza:altezza pari a 4:3.
  Modificando i numeri che compaiono a destra della proprietà **Size**, diamo al form una forma rettangolare: **400 pixel di larghezza per 300 pixel di altezza**. VB accetta la scrittura di questi numeri solo se scritti in sequenza, separati da un punto e virgola.
- La proprietà **StartPosition** indica la posizione in cui si collocherà questa finestra quando il programma sarà in esecuzione. Scegliamo la posizione a centro schermo, come nell'immagine seguente e mandiamo in esecuzione il programma per vederne l'effetto.

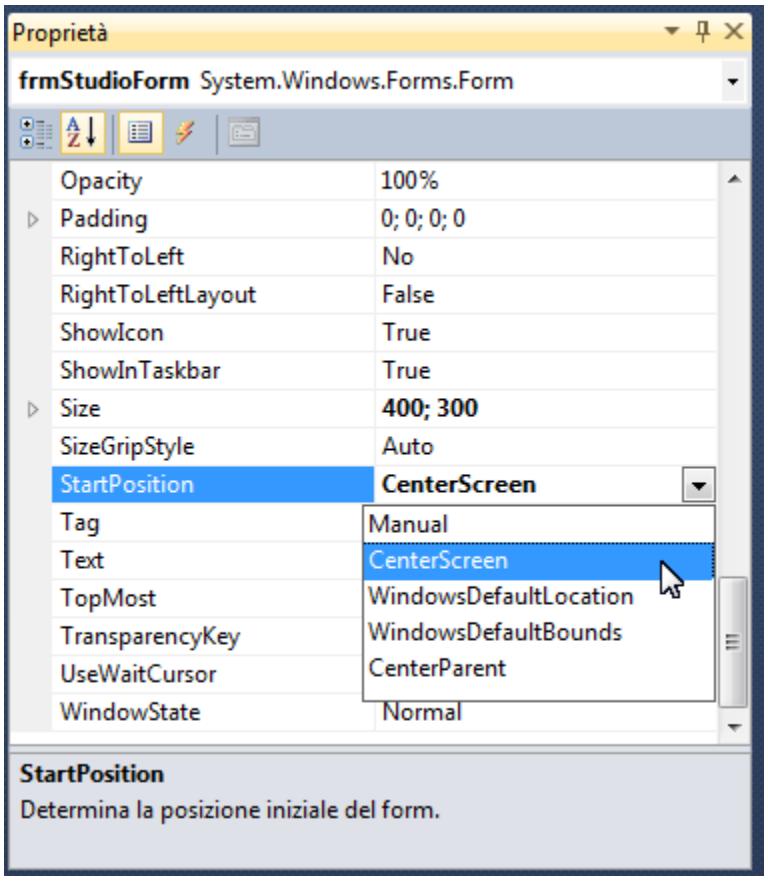


Figura 62: Le proprietà Size e StartPosition del form.

Fermiamo l'esecuzione del programma e modifichiamo queste proprietà del **frmStudioForm**:

- **StartPosition** = **Manual** (questa impostazione vuol dire che il form si collocherà nello schermo nella posizione voluta dal programmatore).
- **Location** = **30;30**. La proprietà **Location** indica il punto in cui si collocherà l'angolo superiore sinistro del frmStudioFormButton in fase di esecuzione. L'impostazione **30;30** significa: 30 pixel di distanza dal bordo sinistro dello schermo, 30 pixel di distanza dal bordo superiore dello schermo.

Ancora una volta, mandiamo in esecuzione il programma per vedere l'effetto della impostazione di queste proprietà

- La proprietà **WindowState** (ultima nella lista delle proprietà) si riferisce alla grandezza che la finestra avrà quando il programma sarà in esecuzione. Le possibilità sono tre: **Normal, Minimized**, **Maximized**.

Proviamo gli effetti del cambiamento di queste proprietà; al termine fermiamo l'esecuzione del programma (**Termina debug**) e lasciamo la proprietà **WindowState** impostata su **Normal**.

25: Il pixel.

Il **pixel** è l'unità di misura utilizzata per definire la posizione e le dimensioni e di un form o di un oggetto; corrisponde a un punto sullo schermo, è l'elemento più piccolo in cui si può suddividere lo schermo (il termine pixel è una contrazione di *picture element,* elemento dell'immagine).
Quando si parla di risoluzione dello schermo si parla in termini di pixel. Uno schermo con risoluzione 1024 x 768, ad esempio, è largo 1024 pixel e alto 768 pixel, per cui contiene 786.432 pixel.
Il **frmStudioForm** con il quale abbiamo avuto a che fare sino a ora è largo 400 pixel e alto 300 pixel; si collocherà a 30 pixel dal lato sinistro dello schermo e a 30 pixel dal lato superiore del monitor.
Facciamo un *clic* sulla Label1 e vediamo, nella barra blu in basso nella finestra di progettazione, a destra, i dati relativi alla sua posizione e alle sue dimensioni espressi in pixel:

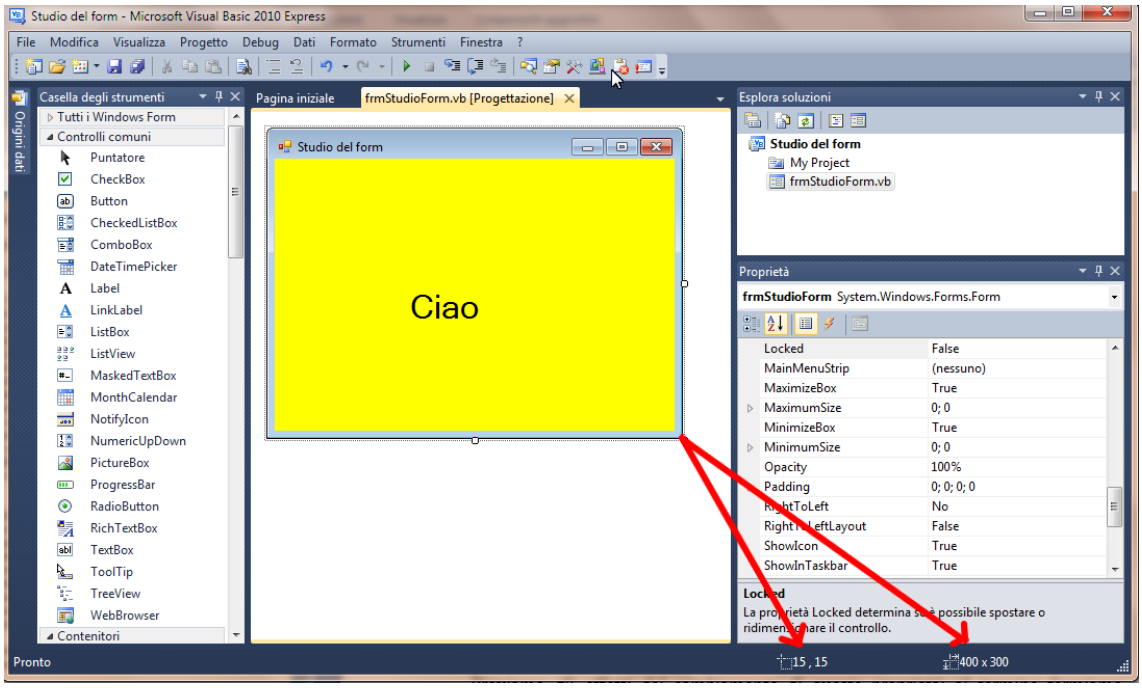


Figura 63: Posizione e dimensioni della Label1 espressi in pixel.

Concludiamo qui l'analisi delle proprietà del form.
Ne abbiamo visto le proprietà principali, ne vedremo altre più avanti, nel manuale, quando si presenterà l'occasione del loro uso concreto.
Abbiamo familiarizzato con la tecnica della modifica delle impostazioni di queste proprietà in fase di progettazione. Vedremo nel paragrafo seguente come è possibile cambiare le proprietà del form quando il programma è in esecuzione, cioè quando non si può più disporre della ***Finestra Proprietà****.*

26: Le proprietà del form nella fase di esecuzione.

Le proprietà di un form e di ogni altro oggetto possono essere modificate anche mentre il programma è in esecuzione.
Nella fase di esecuzione, non potendo più intervenire sulla **Finestra della Proprietà**, le modifiche delle proprietà degli oggetti si ottengono scrivendo apposite istruzioni nella **Finestra del Codice**.
Ecco, ad esempio, un'istruzione che, durante l'esecuzione di un programma, assegna al form, come proprietà **BackColor** (colore di fondo) il colore blu:

```
Me.BackColor = Color.Blue
```

La sintassi delle istruzioni per impostare una proprietà si compone di cinque elementi:

```
Me . BackColor = Color.Blue
↑  ↑    ↑      ↑    ↑
1  2    3      4    5
```

1. il nome dell'oggetto;
2. un punto (".") per separare il nome dell'oggetto dalla proprietà che si vuole modificare;
3. il nome della proprietà che si vuole modificare;
4. il simbolo **=**;
5. la nuova impostazione della proprietà.

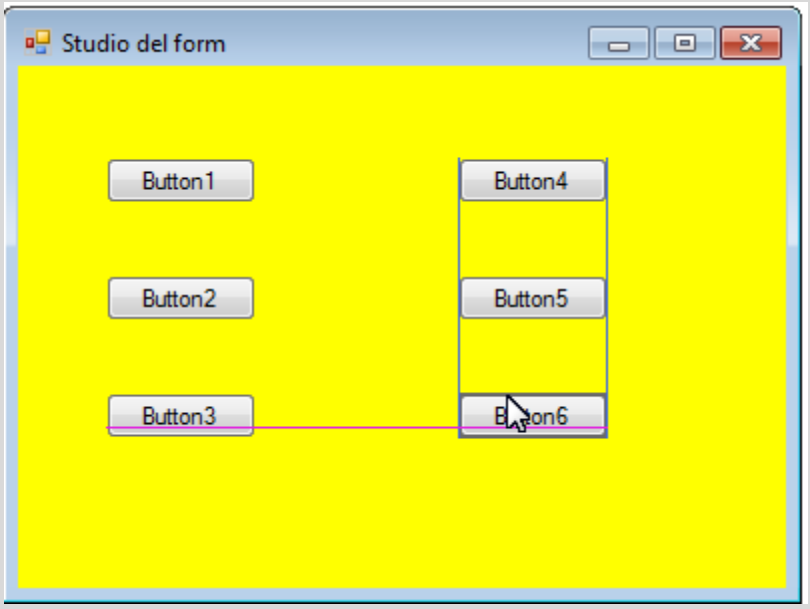
Nella riga di esempio precedente:
1. **Me** è il nome dell'oggetto da modificare (nel nostro caso **Me** si riferisce al **frmStudioForm**);
2. Un **punto** separa l'oggetto **Me** dalla proprietà **BackColor** che si vuole modificare;
3. **BackColor** è la proprietà che il programmatore intende modificare;
4. Il simbolo **=** mette in corrispondenza la proprietà con la sua nuova impostazione.
5. **Color.Blue** è la nuova impostazione della proprietà.

Nel prossimo esercizio vedremo le istruzioni per modificare le proprietà **BackColor**, **Location** e **Size** del form durante l'esecuzione di un progetto.

Esercizio 6: Modifica delle proprietà in fase di esecuzione.

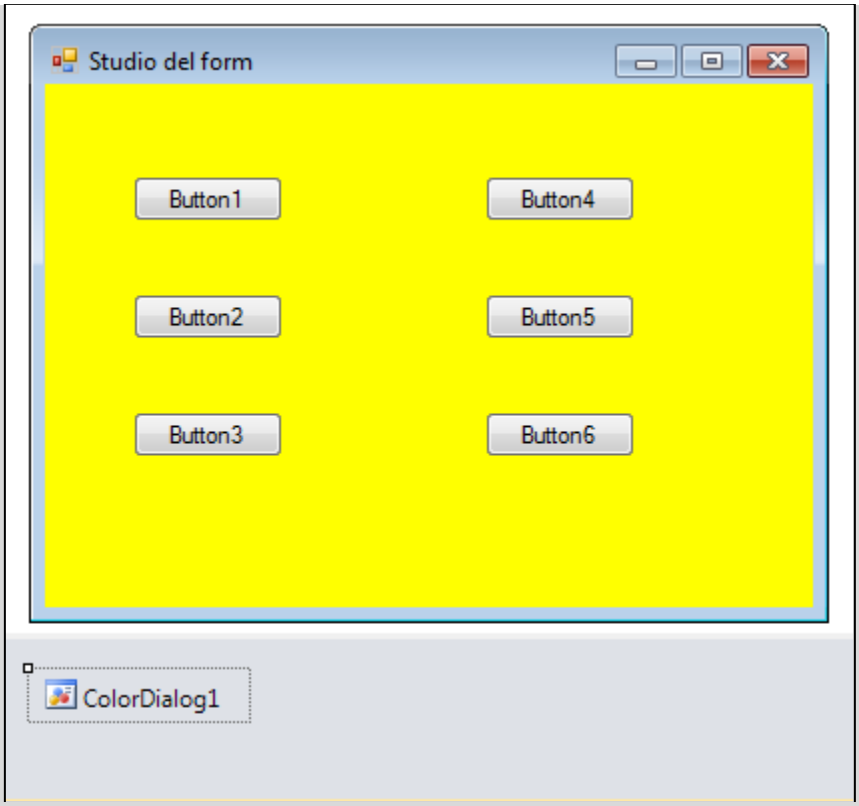
Riprendiamo il progetto **Studio del form** al quale abbiamo lavorato in questo capitolo.

Facciamo un *clic* sulla Label1 e poi un *clic* sull'icona con le forbici, per eliminare la label dal progetto[29].
Visualizziamo la **Casella degli Strumenti** e collochiamo sei pulsanti **Button** nel frmStudioForm.
Mentre sistemiamo questi sei pulsanti, notiamo come VB ci viene in aiuto con le linee guida di colore blu per il loro incolonnamento e di colore viola per il loro allineamento.



Ancora nella **Casella degli Strumenti**, andiamo al gruppo di componenti denominato **Finestre di dialogo** e facciamo un doppio *clic* sul componente **ColorDialog**.
Il **ColorDialog** è un componente (oggetto non visibile all'utente del programma) che viene utilizzato per consentire all'utente del programma di scegliere un colore.
Il **frmStudioForm** ora dovrebbe apparire come in questa immagine:

[29] Per eliminare un controllo si può anche procedere in questo modo:
1. *clic* sul controllo con il mouse.
2. Premere il tasto CANC sulla tastiera.



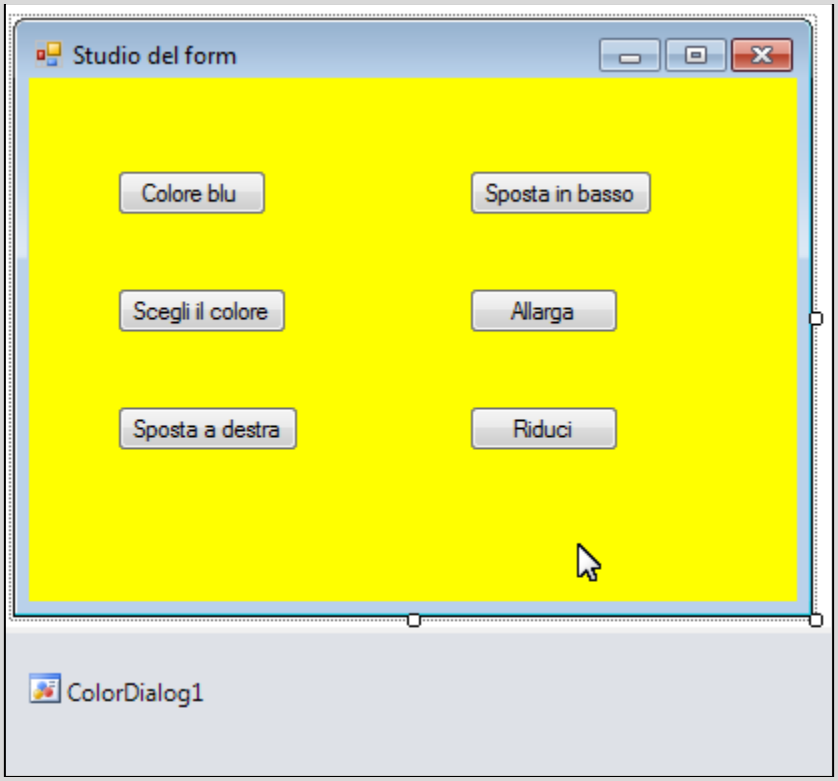
Notiamo che il **ColorDialog** è andato a collocarsi al di fuori del form (in basso), questa posizione segnala che il componente fa parte del progetto ma non è visibile nel form. Ora modifichiamo le proprietà **Text, Name** e **Autosize** dei sei pulsanti **Button** come nella tabella seguente:

Nome assegnato da VB	**Proprietà Text**	**Proprietà Name**	**Proprietà Autosize**[30]	**Funzione del pulsante**
Button1	Colore blu	btnBlu	False	Cliccando questo pulsante il colore di fondo del form diventa blu.
Button2	Scegli il colore	btnSceltaColore	True	Cliccando questo pulsante l'utente del programma può scegliere come colore di fondo per il form il colore che desidera.

[30] La proprietà **Autosize** specifica se il controllo deve adeguare le dimensioni al suo contenuto oppure no. Nel nostro caso è necessario adeguare le dimensioni dei pulsanti 2, 3 e 4, altrimenti il loro testo rimarrebbe parzialmente nascosto.

Button3	Sposta a destra	btnDestra	True	Cliccando questo pulsante il form si sposta verso destra, sul monitor, di 8 pixel.
Button4	Sposta in basso	btnBasso	True	Cliccando questo pulsante il form si sposta verso il basso, sul monitor, di 8 pixel.
Button5	Allarga	btnAllarga	False	Cliccando questo pulsante il form si allarga di 8 pixel.
Button6	Riduci	btnRiduci	False	Cliccando questo pulsante il form si allarga di 8 pixel.

Ecco come compare ora il form con i sei pulsanti:



Iniziamo a scrivere le istruzioni relative a ognuno dei sei pulsanti.
Facciamo un doppio *clic* sul pulsante **Colore blu** (**btnBlu**). Accediamo così alla **Finestra del Codice**, dove troviamo già impostata la procedura per gestire l'evento del *clic* del mouse su questo pulsante. La procedura si compone per ora di due righe:

```
    Private Sub btnBlu_Click(sender As System.Object, e As System.EventArgs)
Handles btnBlu.Click
```

```
    End Sub
```

La prima riga contiene la **dichiarazione** dell'evento che qui viene gestito, con i suoi **parametri tra parentesi**. La seconda riga, **End Sub**, chiude la procedura.
Nello spazio tra le due righe scriviamo una istruzione:

```
    Private Sub btnBlu_Click(sender As System.Object, e As System.EventArgs)
Handles btnBlu.Click

        Me.BackColor = Color.Blue

    End Sub
```

Trovandoci dentro il **frmStudioForm**, la parola `Me` si riferisce a questo form.
Scriviamo la riga in questione cercando di sfruttare l'aiuto offerto da **IntelliSense** nella scrittura del codice.
Dovendo scrivere questa riga di istruzioni

```
        Me.BackColor = Color.Blue
```

Scriviamo solo la prima parola: `Me` e aggiungiamo il punto.
Appena aggiunto il punto, vediamo che si apre un meno a tendina con l'elenco di tutti gli oggetti, le proprietà e le azioni dell'oggetto `Me` (cioè del form).
Scorriamo l'elenco e facciamo un doppio *clic* sulla proprietà `BackColor`.
Ora premiamo sulla tastiera il tasto **=** : a questo punto **IntelliSense** ci mostra l'elenco di tutti i colori che possono essere assegnati alla proprietà `BackColor`.
Scorriamo l'elenco dei colori, facciamo un doppio *clic* sul colore `Color.Blue` e la riga di istruzioni è completata.
Mandiamo in esecuzione il programma e notiamo che cliccando il primo pulsante, con il testo **Colore blu**, il colore di sfondo del form diventa blu.

Fermiamo l'esecuzione del programma, torniamo a visualizzare la Finestra di Progettazione grafica e facciamo un doppio *clic* sul secondo pulsante **Scegli il colore**.
Vediamo che la Finestra del Codice è già impostata una procedura per gestire l'evento del *clic* del mouse sul **btnScegliColore**.
Inseriamo nella nuova procdedura queste due righe di istruzioni:

```
        ColorDialog1.ShowDialog()
        Me.BackColor = ColorDialog1.Color
```

La prima riga di istruzioni dice al programma di aprire l'oggetto di selezione dei colori (**ColorDialog1**); la seconda riga dice di impostare come colore di fondo del frmStudioForm il colore scelto dall'utente.
Procediamo in questo modo con gli altri quattro pulsanti, scrivendo queste istruzioni.
Per il **btnDestra** inseriamo nella pocedura questa riga di istruzioni:

```
        Me.Left += 8
```

L'istruzione dice: al *clic* del mouse su questo pulsante, la distanza del form dal bordo sinistro dello schermo è pari alla distanza attuale + 8 pixel (cioè sposta il form a destra di 8 pixel).
Per il **btnBasso**:

```
        Me.Top += 8
```

L'istruzione dice: al *clic* del mouse su questo pulsante, la distanza del form dal bordo alto dello schermo è pari alla distanza attuale + 8 pixel (cioè sposta il forma in basso di 8 pixel).
Per il **btnAllarga** le righe di istruzioni da inserire sono due:

```
        Me.Width += 8
        Me.Height += 8
```

Traduzione: aumenta la larghezza del form di 8 pixel, aumenta l'altezza del form di 8 pixel.
Anche per il *btnRiduci* le righe di istruzioni da inserire sono due:

```
        Me.Width -= 8
        Me.Height -= 8
```

Traduzione: diminuisci la larghezza del form di 8 pixel, diminuisci l'altezza del form di 8 pixel.
Il Codice completo delle istruzioni che abbiamo scritto è questo[31]:

```
Public Class frmStudioForm

    Private Sub btnBlu_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBlu.Click

        Me.BackColor = Color.Blue

    End Sub

    Private Sub btnSceltaColore_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnSceltaColore.Click

        ColorDialog1.ShowDialog()
        Me.BackColor = ColorDialog1.Color

    End Sub

    Private Sub btnDestra_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDestra.Click

        Me.Left += 8

    End Sub
```

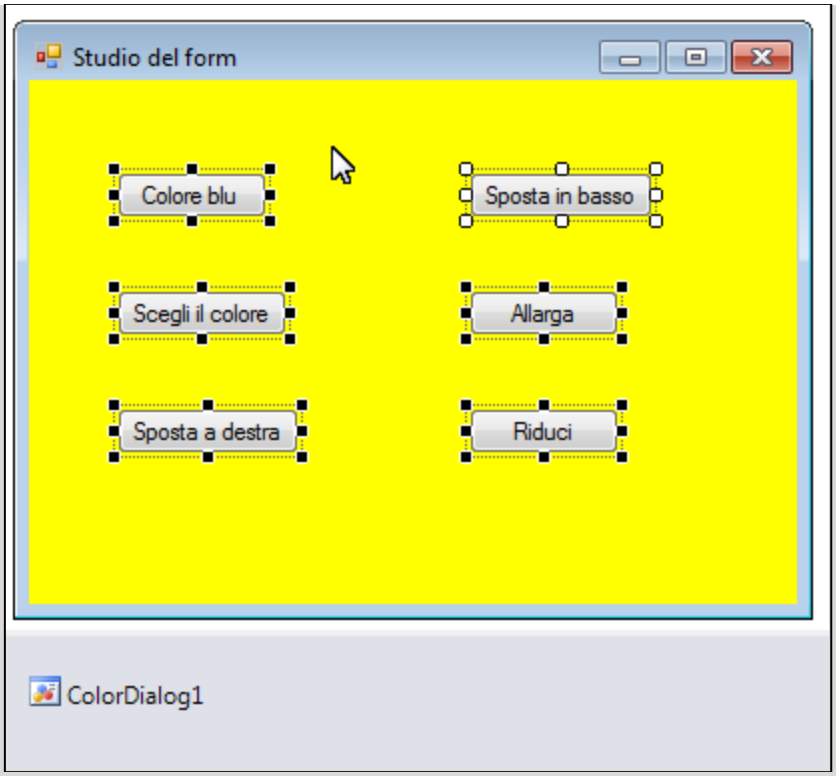
[31] I listati degli esercizi sono disponibili nella cartella **Documenti / A scuola con VB 2010 / Esercizi**. Possono essere utilizzati per il confronto con il lavoro eseguito dal lettore, oppure possono essere copiati e incollati nella Finestra del Codice.

```
    Private Sub btnBasso_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnBasso.Click

        Me.Top += 8

    End Sub
```

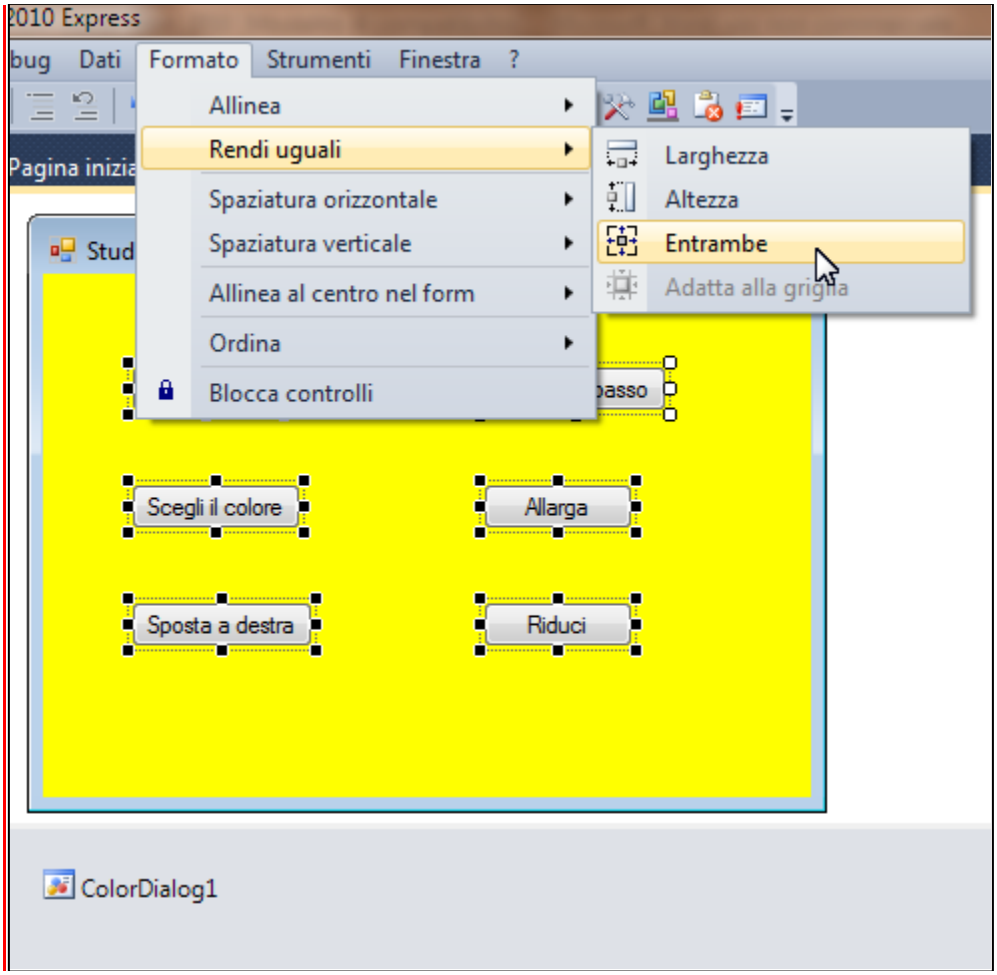
```
    Private Sub btnAllarga_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAllarga.Click

        Me.Width += 8
        Me.Height += 8

    End Sub
```

```
    Private Sub btnRiduci_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRiduci.Click

        Me.Width -= 8
        Me.Height -= 8

    End Sub

End Class
```

Mandiamo in esecuzione il programma e vediamo gli effetti prodotti dai *clic* sui vari pulsanti.
Per rendere più elegante l'interfaccia, possiamo rendere uguali le dimensioni dei sei pulsanti.
Fermiamo l'esecuzione del programma cliccando l'icona Termina debug.
Ora selezioniamo il pulsante più lungo (il pulsante **Sposta in basso**) con un *clic* del mouse, poi, tenendo premuto il tasto MAIUSC, facciamo un *clic* sugli altri pulsanti, selezionandoli uno a uno.
Terminata la selezione, rilasciamo il tasto MAIUSC. Notiamo che i sei pulsanti selezionati compaiono con le loro maniglie di dimensionamento evidenziate.
Il pulsante **Sposta in** basso, che è stato selezionato per primo, compare con le maniglie evidenziate in bianco, gli altri hanno le maniglie evidenziate in nero.
Il pulsante **Sposta in** basso sarà dunque il punto di riferimento per l'operazione che stiamo per compiere.



Nel menu Formato facciamo un *clic* sul comando **Formato / Rendi uguali / Entrambe (le dimensioni).**



27: Eventi e azioni del form.

Abbiamo visto alcune delle **proprietà** principali dell'oggetto form, e come è possibile impostarle in fase di progettazione e in fase di esecuzione di un progetto.
Vediamo ora quali sono i principali **eventi** che il form è in grado di riconoscere e le principali **azioni** che il form è in grado di intraprendere.
Nell'ambiente di progettazione di VB, l'elenco degli **eventi** di un oggetto è indicato dalla icona con il lampo.
Nel progetto **Studio del form**, facciamo *clic* sul form per visualizzarne le proprietà. Nella **Finestra Proprietà** vediamo l'icona con il lampo, che visualizza l'elenco degli eventi del form.

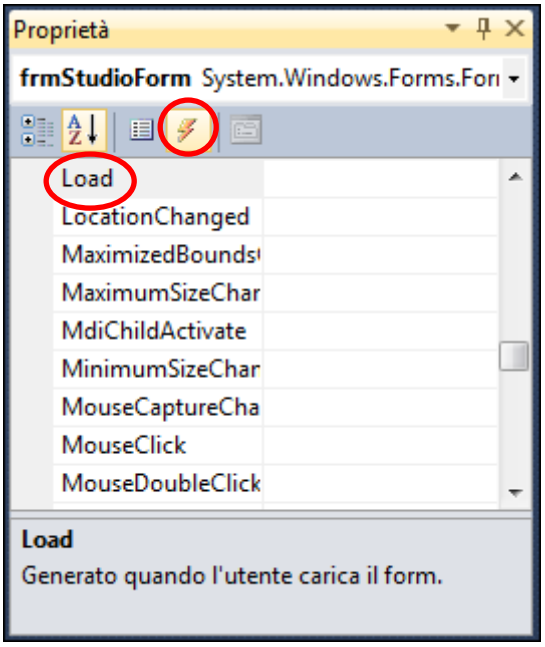


Figura 64: Elenco degli eventi riconosciuti dal form.

Notiamo che, mentre l'elenco delle proprietà si apre sulla proprietà **Text**, l'elenco degli eventi si apre sull'evento **Load** (caricamento) che è il primo evento che può capitare a un form: essere caricato nella memoria del computer.
Scorriamo l'elenco degli eventi del form e notiamo che alcuni di essi ci sono già familiari (come il *clic* e il doppio *clic* del mouse).
L'elenco di questi eventi è accessibile anche dalla Finestra del Codice.
Apriamo la Finestra del Codice e notiamo, nella parte superiore, due menu che si aprono a tendina:

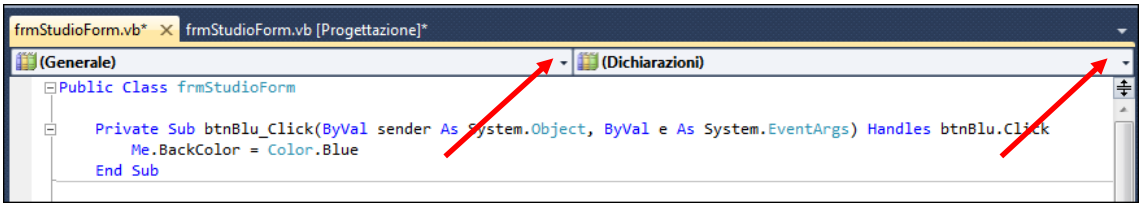


Figura 65: I menu a tendina nella Finestra del Codice.

Nel menu a sinistra vediamo l'elenco degli oggetti presenti nel form. Nel menu a destra vediamo l'elenco degli eventi che ogni oggetto è abilitato a riconoscere.
Facciamo un *clic* sugli eventi del **frmStudioForm** e vediamo che nel menu a destra compare l'elenco degli eventi del form.

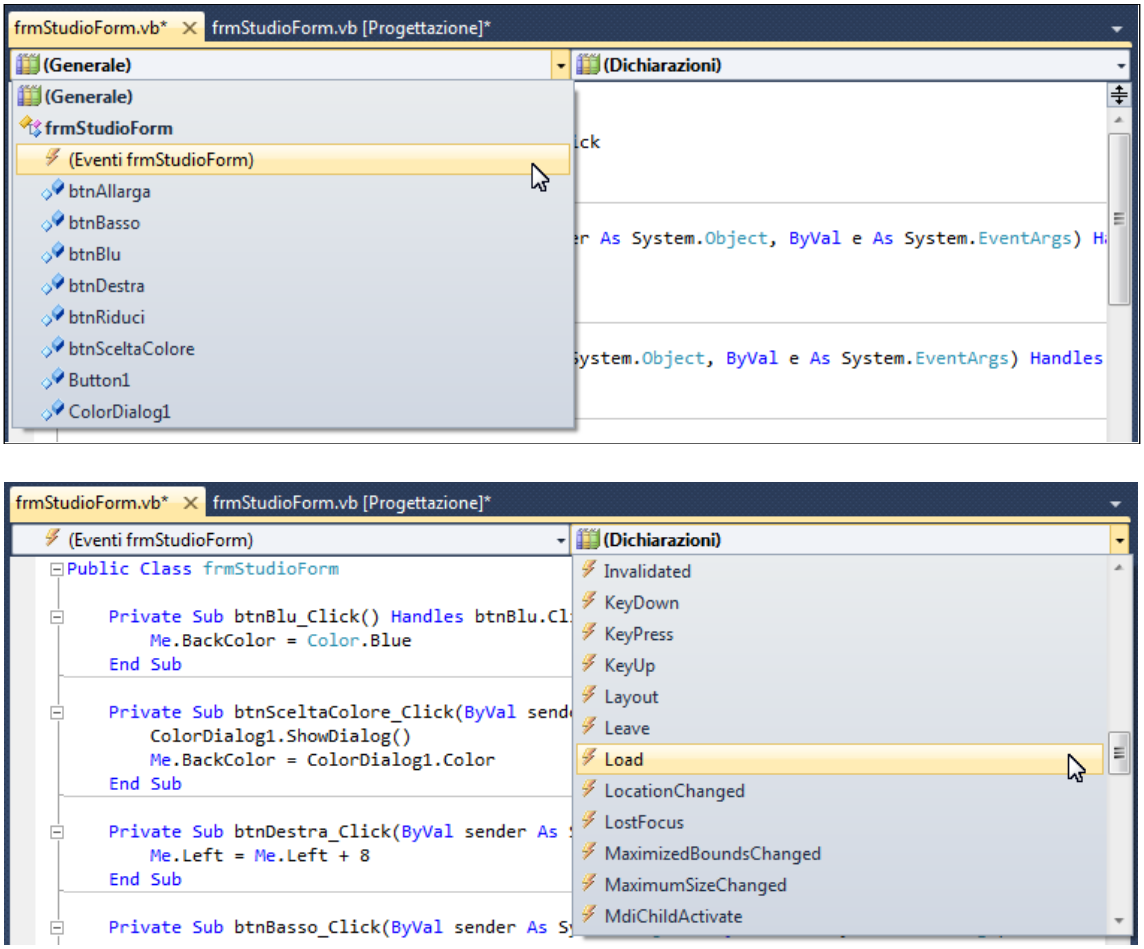


Figura 66: Elenco degli eventi riconosciuti dal form.

Facciamo un doppio *clic* sull'evento **Load** (caricamento).
Vediamo che la Finestra del Codice si predispone a ricevere un'istruzione da eseguire quando accade l'evento **Load** del form (cioè quando il frmStudioForm viene caricato nella memoria del computer, all'avvio del programma).
Con questa istruzione, nella fase di apertura, il programma esegue un bip sonoro:

```
Private Sub frmStudioForm_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

        Beep()

    End Sub
```

Ora, nell'elenco degli eventi, facciamo un doppio *clic* sull'evento opposto, la chiusura del form (**FormClosed**) e anche in questo caso facciamo eseguire un bip al programma:

```
    Private Sub frmStudioForm_FormClosed(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosedEventArgs) Handles Me.FormClosed

        Beep()
    End Sub
```

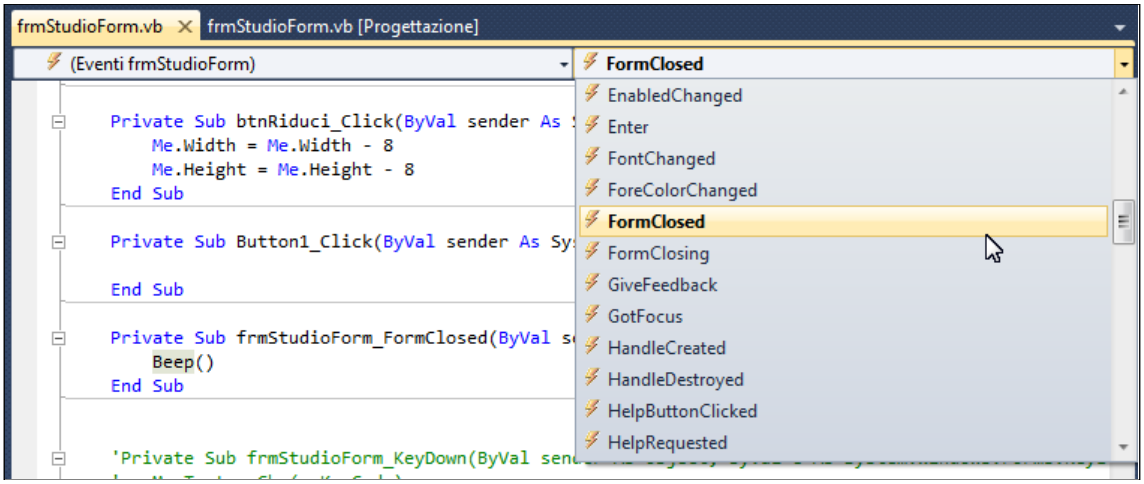


Figura 67: La scelta dell'evento FormClosed.

Mandiamo in esecuzione il progetto e ascoltiamo i due bip, all'avvio e alla chiusura del programma.
Il `Beep` che compare nelle due procedure denominate `frmStudioForm_Load` e `frmStudioForm_FormClosed` non è né una proprietà né un evento, ma è una delle **azioni** che il form è in grado di compiere.
Altre azioni che il form è in grado di compiere sono, ad esempio:

- **Close** (chiude il form);
- **Hide** (il form si nasconde durante l'esecuzione del programma);
- **Show** (il form si mostra in un momento determinato, durante l'esecuzione del programma);
- **Refresh** (il form aggiorna il suo contenuto grafico tenendo conto delle ultime variazioni sopravvenute).

Vediamo come funziona l'azione **Close**, per la chiusura del form.
Torniamo alla Finestra del Codice e inseriamo l'azione Close nella procedura btnBlu_Click:

```
Private Sub btnBlu_Click() Handles btnBlu.Click
        Me.BackColor = Color.Blue
    End Sub
```

Nella riga centrale di questa procedura, invece dell'impostazione del colore dello sfondo scriviamo questa azione:

```
        Me.Close()
```

Mandiamo in esecuzione il progetto: all'evento del *clic* sul pulsante **btnBlu** ora corrisponde la chiusura del form e del programma[32].

[32] In questo caso il programma termina con la chiusura del form, perchè è composto solo da questo form. In programmi più complessi, il comando da preferire per terminare un programma è **Application.Exit**.

Durante la scrittura del comando Me.Close notiamo che, appena scriviamo il punto, **Intellisense** si attiva per suggerirci un elenco di possibilità. In questo elenco, le azioni del form sono contrassegnate da una icona viola.

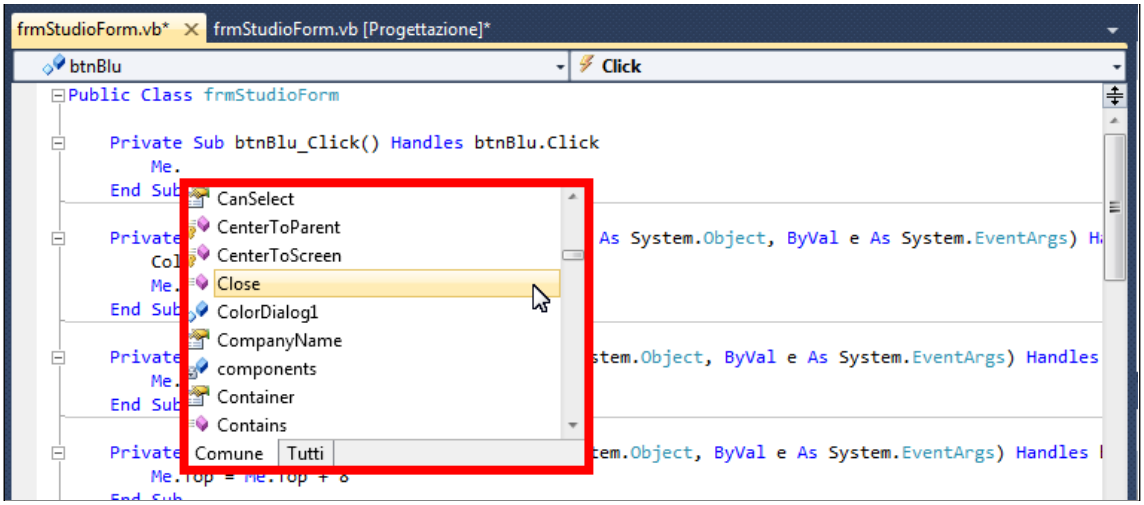
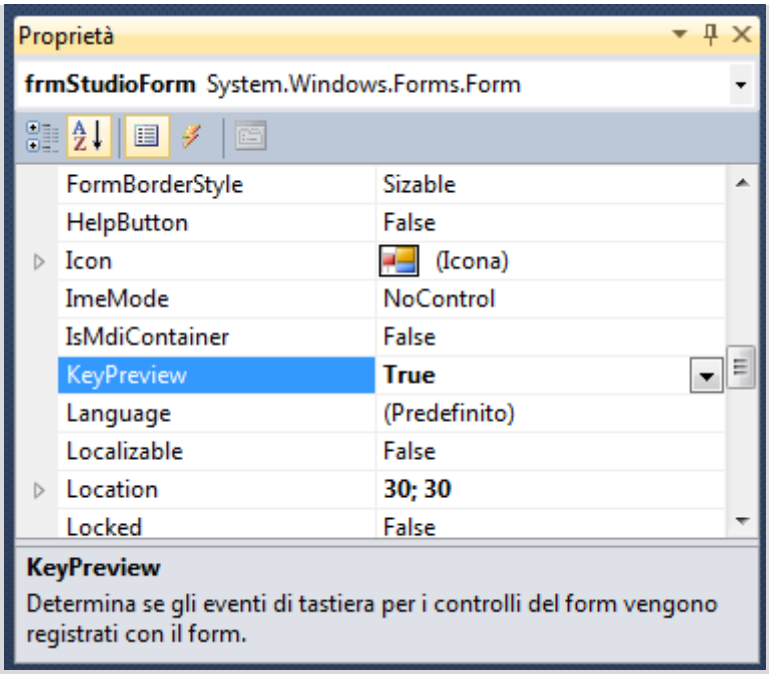


Figura 68: La scelta dell'evento FormClosed.

Nel prossimo esercizio vedremo come gestire un altro evento piuttosto comune: la pressione di un tasto sulla tastiera.

Esercizio 7: Gestione dell'evento KeyCode del form.

In questo esercizio vedremo come gestire l'evento **KeyCode** (pressione di un tasto sulla tastiera) nel **frmStudioForm**.
Vogliamo che, alla pressione di un tasto sulla tastiera, la lettera dell'alfabeto corrispondente a questo tasto compaia nella barra del titolo del form.
Per ottenere questo risultato, dobbiamo innanzitutto modificare la proprietà **KeyPreview** del form impostandola come **True**. In questo modo abilitiamo il form a recepire la pressione dei tasti sulla tastiera.



Fatto questo, andiamo a scrivere, nella Finestra del Codice, cosa il form deve fare quando avverte l'evento della pressione di un tasto sulla tastiera.
Nell'elenco degli eventi del frmStudioForm facciamo un *clic* sull'evento KeyDown e, nella procedura che si apre, scriviamo una riga di istruzioni:

```
    Private Sub frmStudioForm_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown

        Me.Text = Chr(e.KeyCode)

    End Sub
```

La procedura contiene queste istruzioni:
Prima riga:

- Quando accade l'evento della pressione di un tasto sulla tastiera…

Seconda riga:

- nel form `Me` modifica la proprietà `Text` (titolo del form) come segue:
- scrivi il carattere (`Chr`) che corrisponde al codice del tasto premuto sulla tastiera.

Terza riga:

- termina la procedura.

Mandiamo in esecuzione il progetto e vediamo, nella barra del titolo della finestra, l'effetto della pressione dei tasti sulla tastiera.

Prima di chiudere questo esercizio, torniamo a leggere attentamente la riga di apertura della procedura.
Vi troviamo innanzitutto il nome proprio della procedura: **frmStudioForm_KeyDown**.
Potremmo darle un altro nome (ad esempio **Gestione_Tasti**) e il funzionamento del programma non cambierebbe.

Al termine della riga leggiamo che questa procedura frmStudioForm_KeyDown gestisce (`Handles`) l'evento `Me.KeyDown`, la pressione di un tasto sulla tastiera.
Nella parte centrale della procedura, tra parentesi, vediamo la registrazione dei **parametri** (cioè i dettagli, i particolari) dell'evento `Me.KeyDown`.
Il primo parametro (`sender`) registra qual è l'oggetto che ha avvertito l'evento; è un dettaglio che in questo caso non interessa.
Il secondo parametro (`e`) invece è importante perché esso registra il codice corrispondente al tasto che è stato premuto. E' grazie a questo parametro che il pogramma riesce a risalire alla lettera dell'alfabeto corrispondente al tasto premuto.

28: Inserimento di altri form in un progetto.

Un progetto può essere composto di un solo form, come il progetto al quale abbiamo lavorato in questo capitolo; in questo caso abbiamo un'interfaccia con un singolo documento (**SDI**: *Single Document Interface*).
Un progetto può anche essere composto di molti form, in relazione paritaria tra di loro oppure in relazione di dipendenza uno dall'altro. In questo caso avremo un'interfaccia a documenti multipli (**MDI**: *Multiple Document Interface*).
I form aggiuntivi s'inseriscono nel progetto facendo *clic* sul menu **Progetto / Aggiungi nuovi elementi / Windows Form.**
VB assegna automaticamente ai nuovi form i nomi Form2, Form3, ecc.
In fase di esecuzione, questi form si alterneranno alla vista dell'utente, a seconda delle istruzioni che il programmatore avrà scritto per il programma e a seconda di come l'utente userà il programma.
Vediamo un esempio.
Torniamo ad aprire il progetto **Studio del form**.
Aggiungiamo al progetto un altro form (clic sul menu **Progetto / Aggiungi Windows Form**).

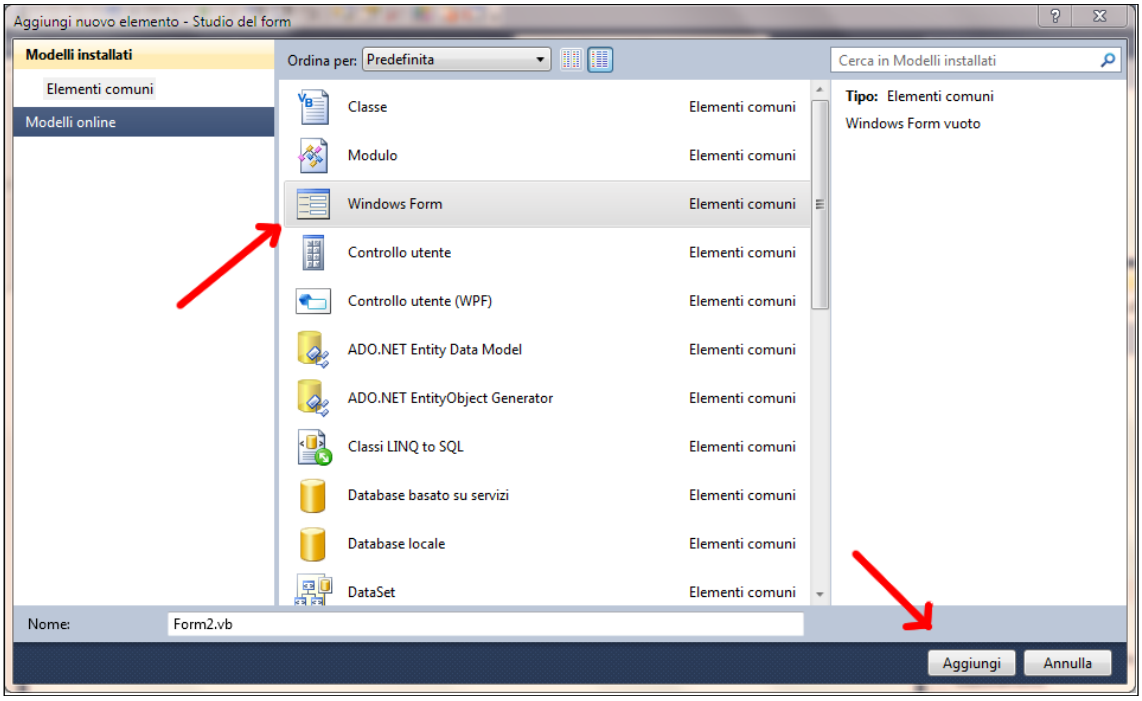


Figura 69: L'aggiunta di un nuovo form al progetto.

Il nuovo form viene aggiunto al progetto con il nome **Form1** (lo vediamo sopra al **frmStudioForm** nella finestra **Esplora Soluzioni**).
Facciamo un clic, nel **frmStudioForm**, sul pulsante con il testo **Colore blu**, per visualizzare le sue proprietà nella **Finestra Proprietà**, e cambiamo la proprietà **Text** da **Colore Blu** a "**Mostra il Form1"**.
Senza cambiare altre proprietà, facciamo un doppio *clic* sul **btnBlu** e torniamo a vedere la **Finestra del Codice**.
Qui modifichiamo la procedura che gestisce l'evento *clic* sul **btnBlu**.
Dove prima era l'istruzione per cambiare il colore dello sfondo

```
Me.BackColor = Color.Blue
```

Scriviamo l'istruzione per mostrare il Form1:

```
Form1.Show
```

Mandiamo in esecuzione il progetto e vediamo il nuovo effetto del *clic* sul pulsante **btnBlu**.

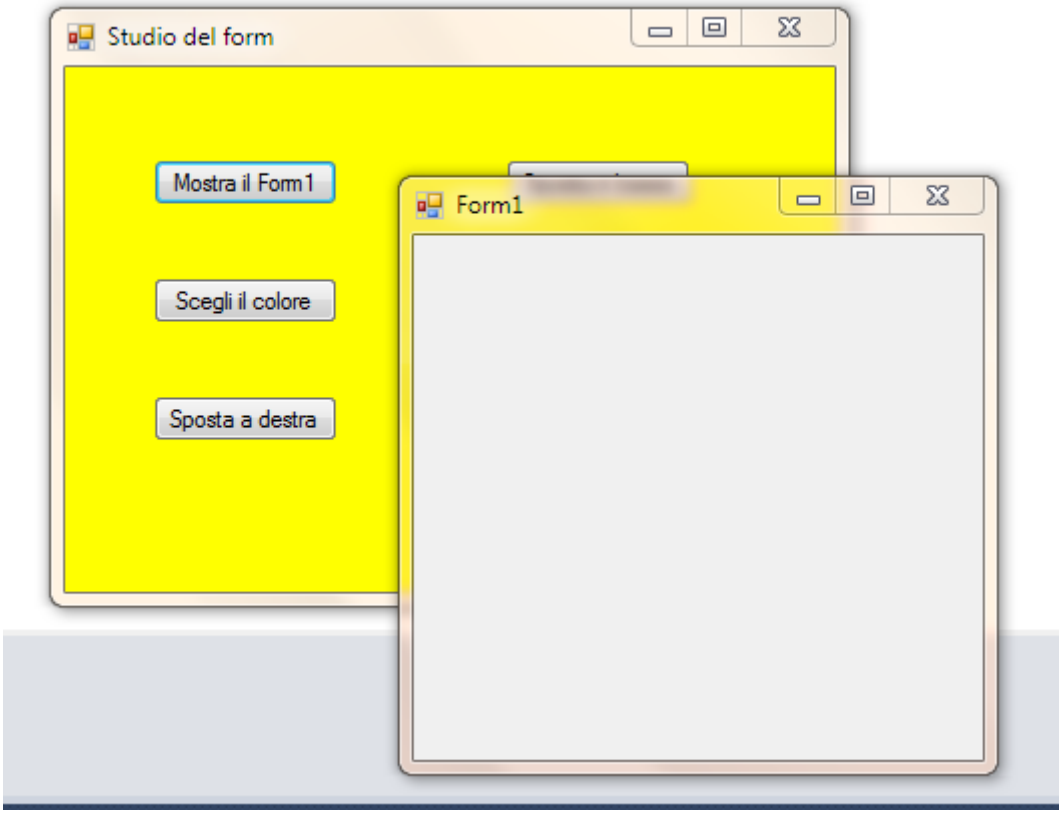


Figura 70: La visualizzazione di un form aggiuntivo durante l'esecuzione di un progetto.

Capitolo 5: CONTROLLI E COMPONENTI: GENERALITA'.

La **Casella degli Strumenti** contiene un'ampia serie di oggetti che il programmatore può inserire nel suo progetto per rendere l'interfaccia più facilmente fruibile da parte dell'utente e perché tengano sotto controllo ciò che avviene durante l'esecuzione del programma.
Alcuni di essi svolgono la funzione di sensori, pronti a cogliere le mosse dell'utente e ad eseguire le istruzioni relative, previste dal programmatore.
Altri svolgono la funzione di contenitori di controlli e di dati, di menu, di gestori di finestre per comunicare con l'utente…

Questi oggetti possono essere visibili all'utente del programma, come il Button (in questo caso di parla di **controlli**), oppure non visibili all'utente, come il Timer (in questo caso si parla di **componenti**).

Tutti questi oggetti portano con sé - create da altri programmatori - capacità e funzioni sofisticate: ogni oggetto è dotato di proprietà ed è in grado di recepire eventi e di compiere azioni determinate.
Il programmatore li utilizza come pacchetti preconfezionati da inserire nel suo lavoro, evitando così di scrivere tutte le istruzioni necessarie per farli funzionare.
Abbiamo già visto, ad esempio, che il pulsante Button *sembra abbassarsi* sotto la pressione del mouse: le istruzioni per ottenere questo effetto sono incorporate nel controllo e non c'è bisogno di scriverle ogni volta: è sufficiente prendere un oggetto Button, collocarlo nel form nella posizione desiderata, dargli le dimensioni volute e il gioco è fatto!
Naturalmente, le istruzioni perché il controllo Button si comporti in questo modo esistono, anche se non si vedono: sono state scritte da un altro programmatore e sono state incorporate nell'oggetto a beneficio di ogni altro programmatore.

VB li presenta, nella **Casella degli Strumenti**, dapprima tutti insieme (**Tutti i Windows Form**) e poi ripartiti per gruppi, a seconda delle loro funzioni:

- **Controlli comuni**;
- **Contenitori** (sono controlli destinati a contenere altri controlli);
- **Menu e barre degli strumenti** (controlli per agevolare l'uso del programma da parte dell'utente);
- **Dati** (controlli collegati alla gestione di archivi di dati);
- **Componenti** (oggetti non visibili che operano in un programma *dietro le quinte*);
- **Stampa** (controlli finalizzati alla gestione delle operazioni di stampa);

- **Finestre di dialogo** (controlli che creano finestre di dialogo con l'utente, per recepire le sue preferenze);
- **Interoperabilità WPF** (Windows Presentation Foundation è una tecnica di programmazione avanzata di cui non tratteremo in questo manuale);
- **Visual Basic PowerPacks** (controlli che provengono dall'ambiente di progettazione di Visual Basic 6).

Ognuno di questi gruppi di controlli si apre con lo strumento **Puntatore**, che tuttavia non è un controllo né un componente collocabile sul form: si tratta piuttosto di uno strumento di **deselezione**, che il programmatore può utilizzare quando intende annullare la selezione di un controllo avvenuta in precedenza.
Ecco un esempio. Apriamo un nuovo progetto, senza dargli alcun nome.
Nella Casella degli Strumenti, facciamo un *clic* con il mouse sul controllo Button.
In questo modo il mouse ha ***selezionato*** il controllo Button.
Ora, spostandoci con il mouse sopra il form in fase di progettazione, si nota che il mouse ***trascina*** con sé un controllo Button, in attesa di collocarlo nel form.
A questo punto supponiamo di avere cambiato idea: non ci serve più un controllo Button ma ci serve invece un controllo Label; il controllo Button tuttavia è ancora ***legato*** al mouse. Cliccando l'icona del **Puntatore**, si annulla la selezione del controllo Button e si libera il mouse per effettuare una nuova selezione.

29: Manipolazione dei controlli.

Per collocare un controllo nel form

si può agire in due modi:

- Fare un doppio *clic* sul controllo, nella Casella degli Strumenti. In questo caso il controllo in questione si colloca nell'angolo a sinistra in alto nel form, da dove è possibile trascinarlo nella posizione desiderata.
- Fare un *clic* sul controllo e portarsi all'interno del form nella posizione desiderata (il cursore del mouse prende l'icona del controllo cliccato). Qui, con un altro *clic* del mouse, il controllo viene rilasciato e prende forma.

Per dimensionare un controllo nel form

Bisogna fare un *clic* con il mouse all'interno del controllo. In questo modo appaiono le otto maniglie di dimensionamento. Agendo con il tasto sinistro del mouse su queste maniglie è possibile dare al controllo le dimensioni desiderate.

Per spostare un controllo nel form

bisogna fare un *clic* con il mouse all'interno del controllo per visualizzare le otto maniglie di dimensionamento. Tenendo premuto il tasto sinistro del mouse, trascinando il mouse nel form è possibile spostare il controllo dove si desidera.

Per fare apparire le maniglie di dimensionamento

bisogna fare un *clic* con il mouse all'interno del controllo.

Per fare scomparire le maniglie di dimensionamento

bisogna fare un *clic* con il mouse nell'area esterna al controllo.

Per selezionare un gruppo di controlli

bisogna tracciare attorno ad essi un'area rettangolare con il **Puntatore**, come nella figura seguente:

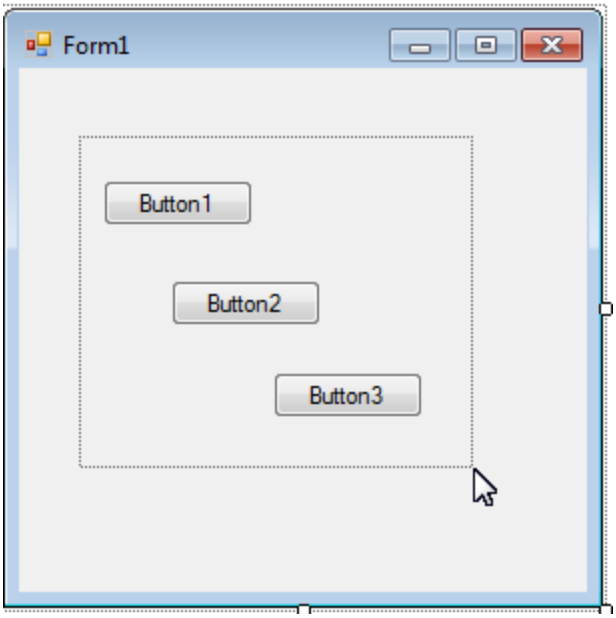


Figura 71: Tre pulsanti selezionati tracciando un rettangolo con il Puntatore.

Rilasciando il puntatore, tutti i controlli che abbiamo racchiuso nella stessa area appaiono selezionati, cioè con le maniglie di dimensionamento attivate[33]:

[33] Il pulsante con le maniglie di colore bianco sarà il pulsante di riferimento, nel caso si debbano compiere operazioni di allineamento, incolonnamento o uguaglianza delle dimensioni dei tre pulsanti.

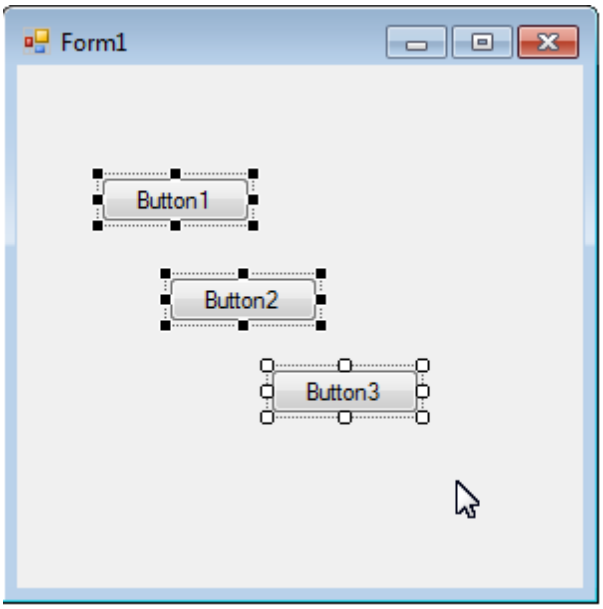


Figura 72: Tre pulsanti selezionati, con le maniglie di dimensionamento attivate.

Un **modo alternativo** per selezionare un gruppo di controlli è questo: fare un *clic* con il mouse sul primo controllo da selezionare poi, tenendo premuto il tasto MAIUSC, fare un *clic* con il mouse sopra tutti gli altri controlli che interessano. Terminata la selezione, si può lasciare il tasto MAIUSC: i controlli selezionati appaiono evidenziati con le maniglie di dimensionamento, come se attorno ad essi fosse stata tracciata l'area rettangolare con il Puntatore.
La seconda modalità presenta questi vantaggi:

- consente di selezionare anche controlli che nel form occupano un'area non omogenea, non circoscrivibile con il Puntatore, che crea solo aree di selezione rettangolari;
- consente di scegliere il controllo che farà da riferimento all'interno del gruppo selezionato (cioè il controllo con le maniglie di dimensionamento di colore bianco). Con questa modalità, il controllo di riferimento è il controllo selezionato per primo, mentre nella selezione con il Puntatore il controllo di riferimento è il controllo creato per ultimo, tra quelli selezionati.

Per spostare insieme più controlli

bisogna prima selezionare i controlli che interessano con uno dei due metodi illustrati in precedenza, in modo che compaiano le loro maniglie di dimensionamento; poi, fare un *clic* con il mouse all'interno di uno dei controlli selezionati e, **tenendo premuto il tasto sinistro del mouse**, spostare il mouse nel form.

Tutti i controlli selezionati seguono il movimento del mouse.

Per allineare o per incolonnare più controlli

bisogna prima selezionare i controlli che interessano con uno dei due metodi illustrati in precedenza, in modo che compaiano le loro maniglie di dimensionamento.
Fatto questo, tutti i controlli selezionati si allineano o si incolonnano facendo un *clic* con il mouse sul menu **Formato** e poi sul comando **Allinea**; sono disponibili diverse opzioni di allineamento o di incolonnamento. Il controllo di riferimento, la cui posizione funge da riferimento per tutti gli altri, è il controllo con le maniglie di dimensionamento di colore bianco.

Per dare le stesse dimensioni a più controlli

bisogna raggruppare i controlli che interessano e selezionarli con una delle due modalità scritte in precedenza, in modo che compaiano le loro maniglie di dimensionamento. Fatto questo, si fanno assumere a tutti i controlli selezionati le stesse dimensioni facendo un *clic* con il mouse sul menu **Formato** e poi sul comando **Rendi uguale**; è possibile rendere uguali l'altezza, la larghezza o entrambe le dimensioni. Il controllo di riferimento, le cui dimensioni fungono da riferimento per tutti gli altri, è il controllo con le maniglie di dimensionamenmto di colore bianco.

Per centrare un controllo nel form

bisogna selezionare il controllo in modo che compaiano le sue maniglie di dimensionamento, poi bisogna fare un *clic* sul menu **Formato** e sul comando **Centra nel form**; la centratura può essere fatta in senso **orizzontale** o in senso **verticale**.

Le operazioni di allineamento, incolonnamento, centratura, ridimensionamento possono essere effettuate, oltre che dal menu **Formato**, anche dalla striscia dei pulsanti di **Layout**:

Figura 73: La striscia dei pulsanti di Layout (visualizzazione).

Questa striscia di pulsanti non è visibile normalmente nell'ambiente di progettazione di VB. Per renderla visibile è necessario fare un *clic* con il **tasto destro** del mouse nell'area blu a fianco della striscia standard, come nell'immagine seguente.

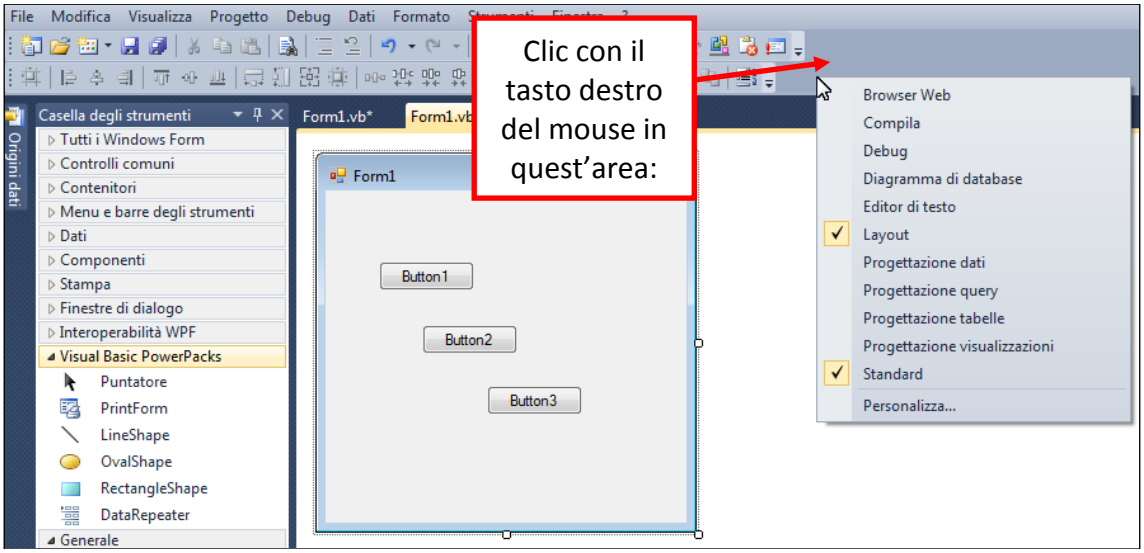


Figura 74: Visualizzazione della striscia dei pulsanti di Layout.

Per duplicare o replicare più volte un controllo

bisogna fare un *clic* all'interno del controllo per fare apparire le maniglie di dimensionamento e quindi:

- premere i tasti CTRL+C oppure fare un *clic* sull'icona **Copia**;
- premere i tasti CTRL+V oppure fare un *clic* sull'icona **Incolla**.

Il controllo replicato viene collocato al centro del form e assume tutte le proprietà del controllo originale.

Per cancellare un controllo dal form

bisogna fare un *clic* all'interno del controllo per fare apparire le maniglie di dimensionamento e quindi premere il tasto **CANC,** o i tasti CTRL+X, o l'icona **Taglia**.
E' possibile cancellare simultaneamente un gruppo di controlli dopo averli selezionati con uno dei metodi già visti:

- racchiudendoli in un'area creata con il puntatore;
- selezionandoli con il *clic* del mouse, tenendo premuto il tasto MAIUSC.

Una cancellazione effettuata per errore può essere recuperata cliccando - nella striscia dei pulsanti - l'icona **Annulla**, con la freccia rivolta a sinistra.

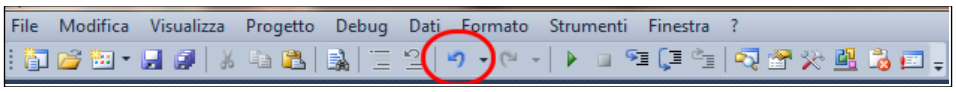


Figura 75: Il pulsante per annullare operazioni sbagliate.

Se un controllo viene cancellato a programmazione ormai avanzata, le procedure di istruzioni eventualmente già scritte, connesse a questo controllo, non vengono cancellate assieme al controllo ma rimangono nel codice. Sarà cura del programmatore rimuovere anche queste, che altrimenti rimarranno nel programma come un peso morto, senza produrre alcun effetto.

Per vedere le proprietà di un controllo

bisogna fare un *clic* sul controllo con il tasto sinistro del mouse: questa azione visualizza la Finestra Proprietà.
In alternativa, fare un *clic* con il tasto destro del mouse sul controllo e, nella finestra che si apre, fare un *clic* sull'ultima voce: Proprietà.

Per vedere l'elenco degli eventi ai quali può rispondere un controllo

nella Finestra Proprietà del controllo, fare un *clic* con il mouse sul pulsante con l'icona del lampo.
In alternativa, aprire la Finestra del Codice, fare un *clic* sul controllo che interessa nel menu a tendina in alto a sinistra e leggere l'elenco degli eventi nel menu a destra:

Figura 76: I due menu a tendina nella Finestra del Codice.

30: Le principali proprietà dei controlli.

Tutti i controlli disponibili in VB fanno parte di un'unica Classe, la Classe "Control", che si trova all'interno dell'archivio Framework 4.
Questo significa che tutti i controlli hanno una base di programmazione condivisa, uguale per tutti, alla quale ogni controllo aggiunge gli elementi di programmazione specifici necessari al suo particolare funzionamento.
Molti tra i controlli contenuti nella Casella degli Strumenti hanno dunque in comune un gruppo di qualità (**proprietà**) fondamentali, che possono essere impostate dal programmatore in fase di progettazione o possono essere modificate con apposite istruzioni scritte nel codice, in fase di esecuzione del progetto.
Vediamo in questo paragrafo alcune di queste proprietà *trasversali*, comuni a molti controlli, elencate in ordine alfabetico.

Vedremo poi, nei capitoli successivi, le proprietà specifiche di ogni singolo controllo.

Name

La proprietà **Name** contiene il nome proprio del controllo.
E' opportuno, per procedere con chiarezza nel lavoro di progettazione, assegnare ai controlli dei nomi propri che inizino con le prime tre consonanti del loro nome comune, scritte in minuscolo.
Ad esempio, il nome proprio di un pulsante Button dovrebbe iniziare con "btn"; il nome di una Label (etichetta) dovrebbe iniziare con "lbl", il nome di un ListBox con "lst", e così via.

AllowDrop

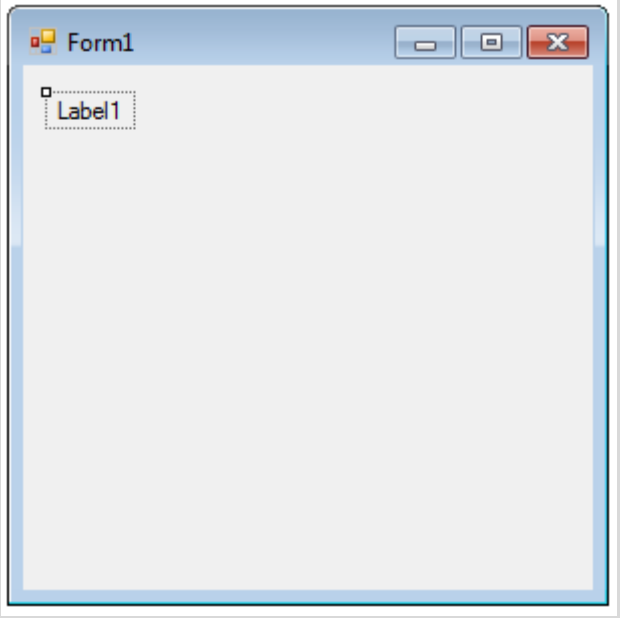
La proprietà **AllowDrop** indica se il controllo può accettare elementi (ad esempio: un'immagine) trascinati al suo interno dall'utente del programma, utilizzando il mouse.
Questa proprietà è normalmente impostata come False.
Se si prevede di dovere trascinare e rilasciare sul controllo delle immagini, ad esempio, è necessario che questa proprietà sia impostata su True.

Anchor

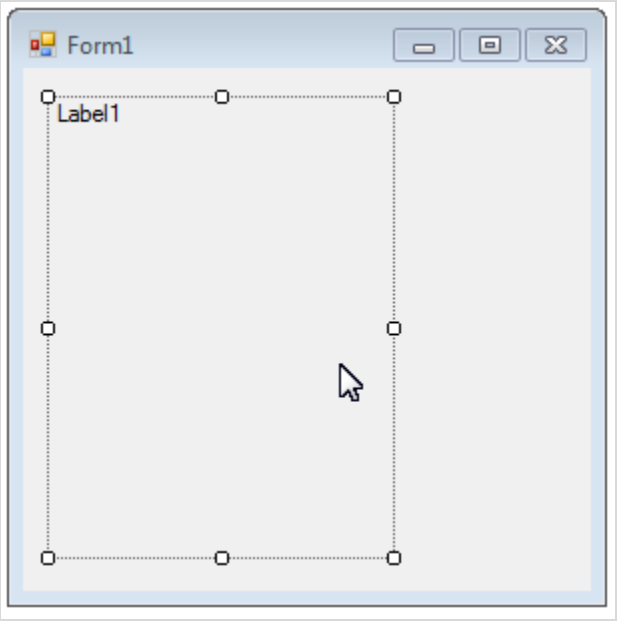
La proprietà **Anchor** indica se le distanze del controllo dai margini del form sono bloccate (cioè fisse) oppure no.
L'effetto di questa proprietà si vede, nella fase di esecuzione, quando l'utente procede ad allargare o ad allungare un form. Se i controlli contenuti nel form sono ancorati, la loro distanza dai margini del form rimane inalterata. In caso contrario, la loro distanza dai margini del form varierà in proporzione al variare delle dimensioni del form.
Vediamo un esempio di applicazione di questa proprietà nel prossimo esercizio.

Esercizio 8: La proprietà Anchor.

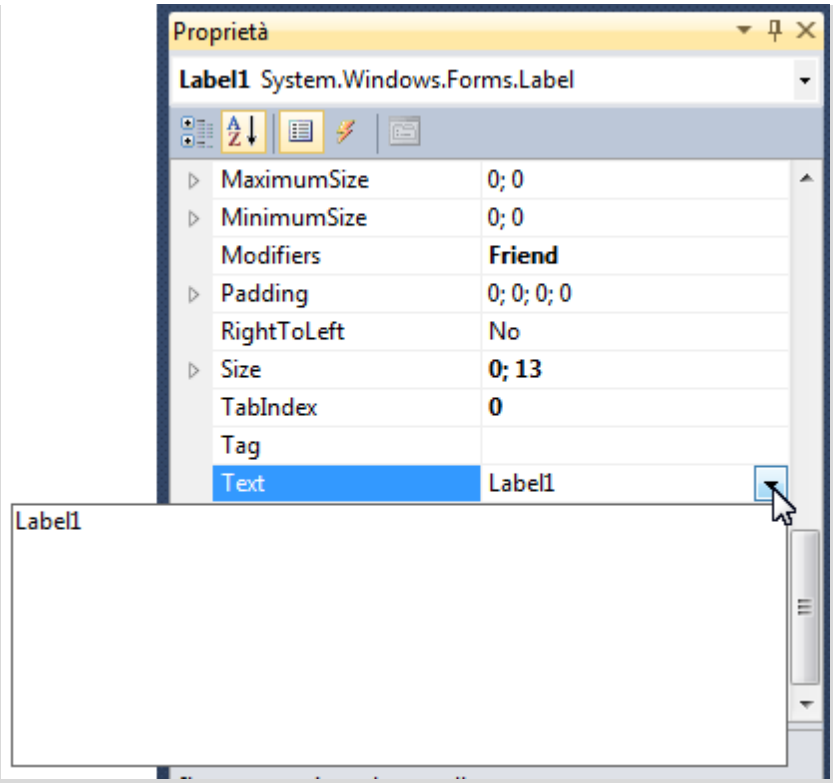
Apriamo un nuovo progetto e inseriamo nel Form1 un controllo Label (contenitore di testo) in alto a sinistra:



Facciamo un *clic* su questa Label1 e, nella Finestra Proprietà, impostiamo la sua proprietà **AutoSize = False**, in modo da visualizzare le maniglie di dimensionamento di questo controllo:

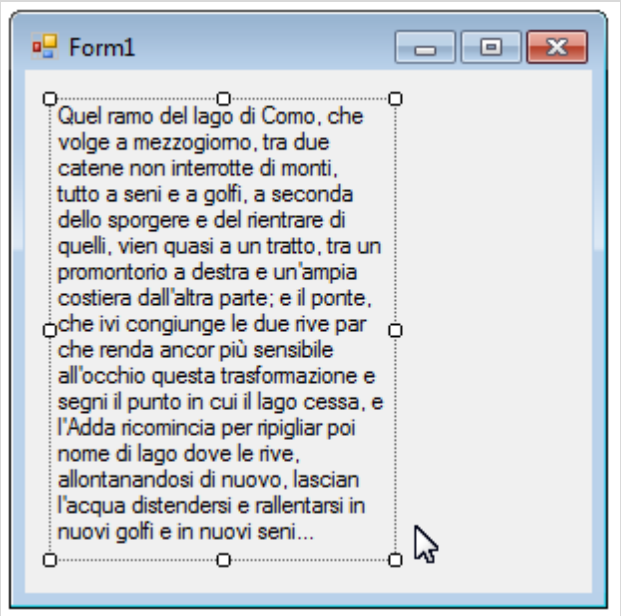


Facciamo un *clic* sul pulsante con la freccina nera che si trova a destra nella riga della proprietà **Text** della Label1:

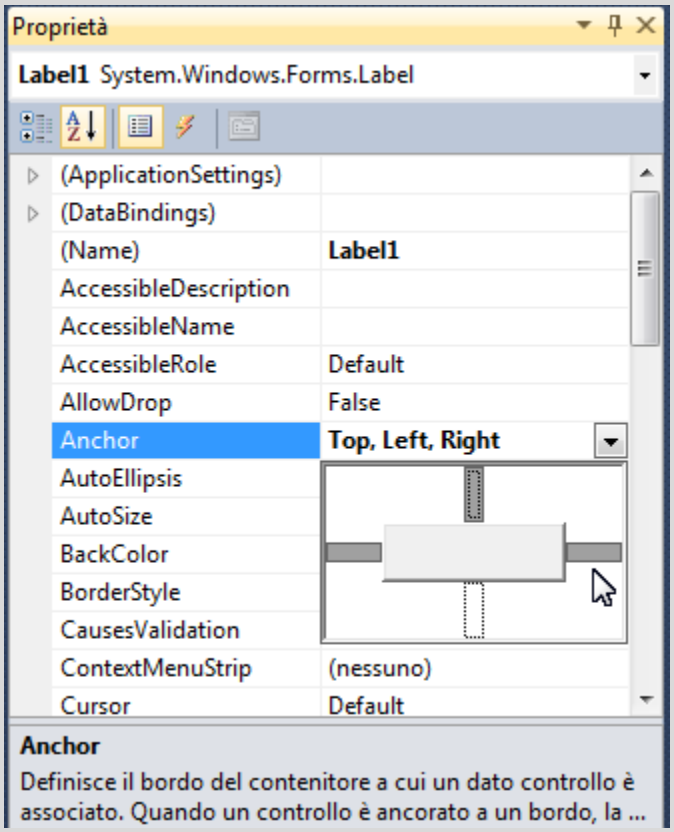


Nel riquadro che si apre, incolliamo questo testo:

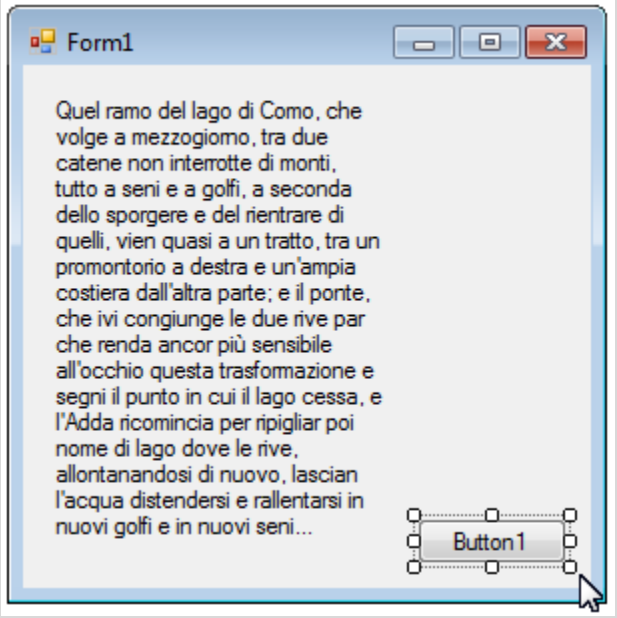
Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien quasi a un tratto, tra un promontorio a destra e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive par che renda ancor più sensibile all'occhio questa trasformazione e segni il punto in cui il lago cessa, e l'Adda ricomincia per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni...



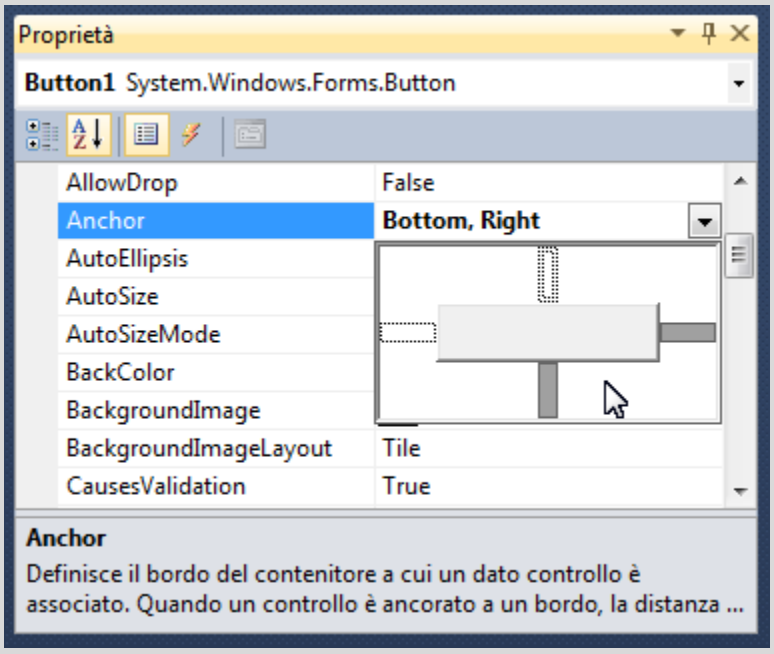
Facciamo un *clic* sulla proprietà **Anchor** di questa Label1 e impostiamo la proprietà in modo da ancorare la Label a sinistra, in alto e a destra:



Con queste impostazioni di *ancoraggio*, durante l'esecuzione del programma, se cambieranno le dimensioni del Form1, le distanze della Label1 dai bordi sinistro, superiore e destro del Form1 rimarranno immutate (in pratica, la Label1 sarà costretta ad allargarsi seguendo lo spostamento del margine destro del Form1).
Ora inseriamo nel Form1 un controllo Button1 in basso a destra:

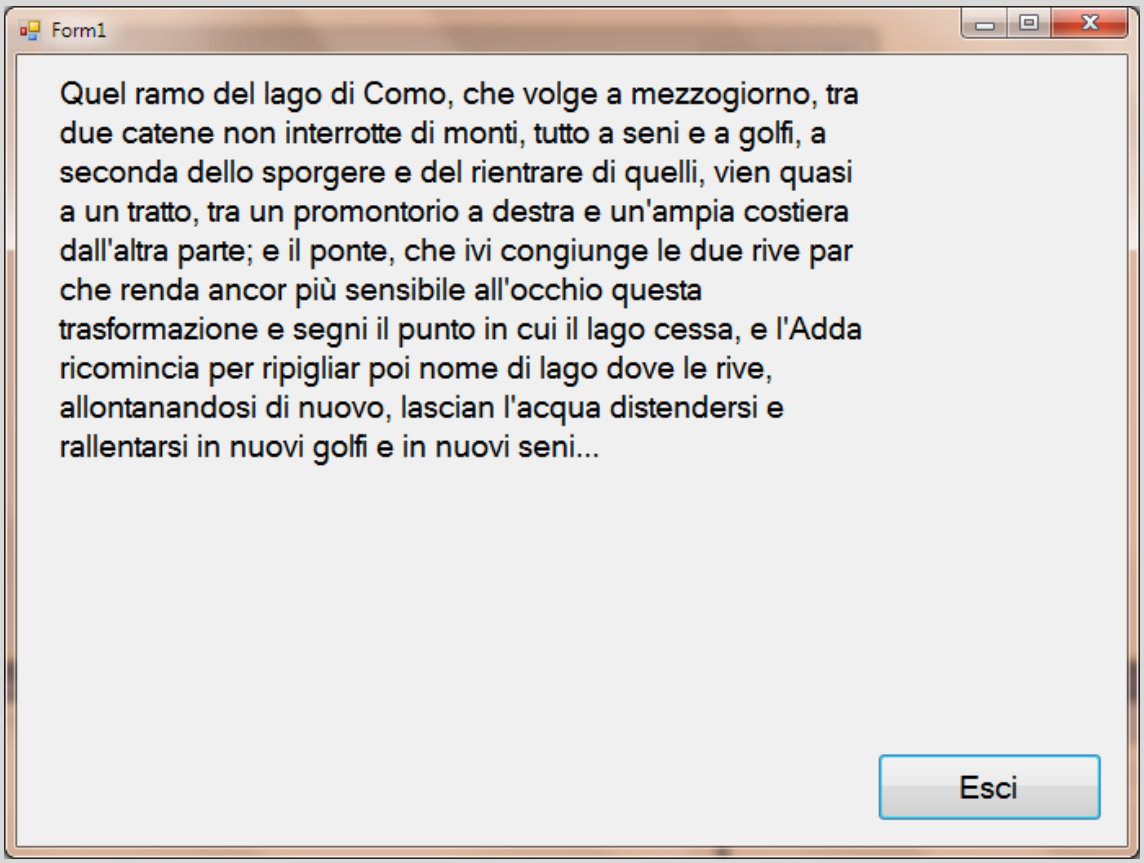


Facciamo un *clic* sul pulsante appena inserito e nella Finestra Proprietà, impostiamo la sua proprietà **Text = Esci** e la proprietà **Anchor** come in questa immagine:



In questo modo ancoriamo il Button1 al margine destro ed al margine inferiore del form. Questo vuol dire che, in fase di esecuzione del progetto, se cambieranno le dimensioni del Form1, le distanze del Button1 dal margine destro e dal margine inferiore del form rimarranno immutate.
Ora facciamo un *clic* sul Form1 e nella Finestra Proprietà impostiamo la sua proprietà **Font = Microsoft Sans Serif a 14 punti**.
Siccome la proprietà **AutoScaleMode** del Form1 è impostata su **Font** (vale a dire che il Form1 è predisposto per adattarsi alla larghezza del set di caratteri scelti dal

programmatore), notiamo che il Form1 e tutto il suo contenuto si adattano alla nuova grandezza del form.
Mandiamo in esecuzione il progetto e proviamo a modificare le dimensioni del form. Notiamo che la Label1 rimane sempre nell'angolo in alto a sinistra del form, ma segue a destra l'ampliamento del form, modificando la sua larghezza per mantenere costante la sua distanza dal margine destro del form; notiamo inoltre che il Button1 rimane sempre nell'angolo basso a destra del form, e le sue distanze dai margini del form non cambiano con il mutare delle dimensioni del form.



Fermiamo l'esecuzione del programma con un *clic* sul pulsante **Termina debug**.
Completiamo il programma scrivendo, nella Finestra del Codice, una procedura affinché il programma termini quando l'utente preme il pulsante Button1 (**Esci**).
Facciamo un doppio *clic* con il mouse sul pulsante Esci e accediamo alla Finestra Proprietà.
Qui troviamo, già impostata, la procedura che gestisce l'evento del *clic* del mouse sul pulsante Button1:

```
 Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    End Sub
```

Nella riga centrale, vuota, scriviamo il comando **End**, che farà terminare il programma quando l'utente cliccherà questo pulsante:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        End

    End Sub
```

AutoEllipsis

La proprietà **AutoEllipsis**, se impostata = True (vero), fa comparire nel testo di un controllo i tre puntini di sospensione, quando il testo è troppo lungo per essere visualizzato interamente.
Ad esempio, se assegniamo a un Button1 il testo "Questo è il pulsante Button1" e ne impostiamo la proprietà AutoEllipsis = True (vero), vediamo comparire all'interno del Button1 solo la parola "Questo …" con i tre puntini di sospensione, perché il testo è troppo lungo per essere visualizzato per intero.
Il testo che compare abbreviato all'interno del Button1 compare comunque per intero nel **ToolTipText** dello stesso Button1, cioè nel piccolo riquadro che viene visualizzato temporaneamente quando il mouse passa sopra il pulsante:

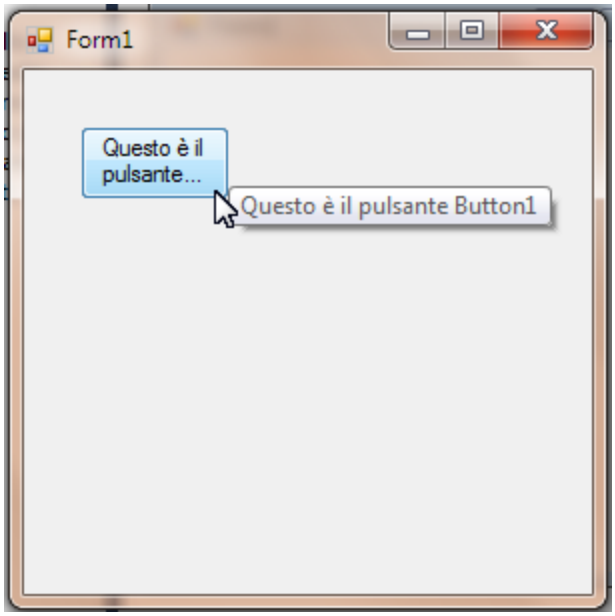


Figura 77: La proprietà AutoEllipsis.

AutoSize

La proprietà **AutoSize** indica se il controllo si deve adeguare o meno al suo contenuto.
Nel caso del Button1 con il testo "Questo è il pulsante Button1", se la proprietà AutoSize è impostata su True (vero), le dimensioni del Button1 si allargano sino a mostrare tutto il testo.

Con questa impostazione, però, il programmatore non ha più la facoltà di modificare le dimensioni del controllo a suo piacere e, per questo motivo, nel controllo in questione non compaiono più le maniglie di dimensionamento.

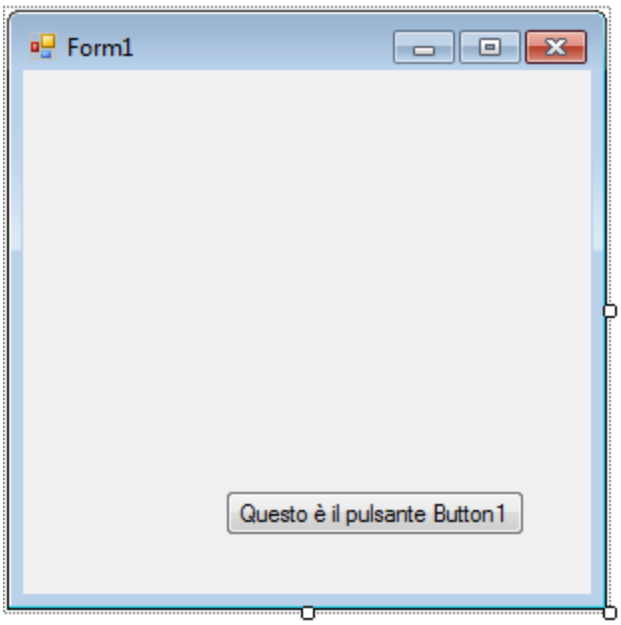


Figura 78: La proprietà AutoSize.

AutoSizeMode

La proprietà **AutoSizeMode** indica se la proprietà AutoSize (adattamento al contenuto del controllo) si applica solo in una direzione (allargare il controllo) oppure in due direzioni (allargare o ridurre il controllo secondo le dimensioni del testo).
La proprietà AutoSizeMode dunque è attiva solo se la proprietà AutoSize è impostata come True (vero) e può essere impostata a sua volta in due modalità:

- **GrowOnly**: il controllo si adatta al suo contenuto solo allargandosi;
- **GrowAndShrink**: il controllo si adatta al contenuto allargandosi o restringendosi.

Supponiamo di avere un controllo Button1 il cui testo sia la lettera “x”. Se la proprietà AutoSize è impostata su True e se la proprietà AutoSizeMode è impostata su GrowAndShrink, il risultato è questo:

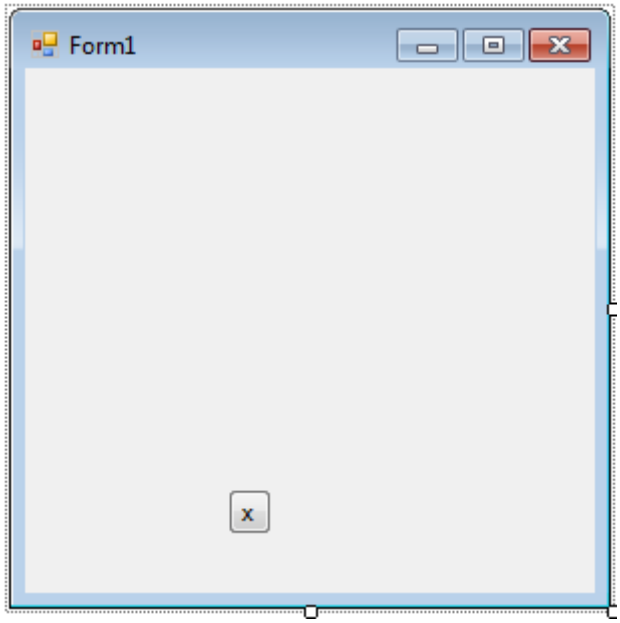


Figura 79: La proprietà AutoSizeMode impostata su GrowAndShrink.

BackColor

La proprietà **Backcolor** imposta il colore dello sfondo del controllo.

BackgroundImage

La proprietà **BackgroundImage** imposta la scelta di un'immagine da utilizzare come sfondo per il controllo.

BackgroundImageLayout

La proprietà **BackgroundImageLayout** indica come deve essere impostata, all'interno del controllo, la visualizzazione della immagine di sfondo: a mattonelle, al centro del form, adattata alle misure del form, ingrandita.
Le proprietà BackColor, BackGroudImage e BackgroundImageLayout sono illustrate nel paragrafo 24: Le proprietà del form nella fase di progettazione. 107.

Cursor

La proprietà **Cursor** imposta la scelta di un cursore per il controllo. Il cursore scelto è visualizzato, nella fase di esecuzione del programma, quando il mouse passa sopra il controllo.

Dock

La proprietà **Dock** (ormeggia) indica che il controllo va a occupare un intero lato del form (come una nave si ormeggia di fianco a un molo in un porto): in alto, in basso, a sinistra o a destra.
Le dimensioni del controllo ormeggiato cambieranno, in fase di esecuzione del programma, assieme alle eventuali modifiche delle dimensioni del form.
Le impostazioni della proprietà Dock sono:

- **Top** (il controllo è ormeggiato in alto);
- **Right** (il controllo è ormeggiato sul lato destro del form);
- **Bottom** (il controllo è ormeggiato in basso);
- **Left** (il controllo è ormeggiato sul lato sinistro del form);
- **Fill** (il controllo è ormeggiato al centro e occuperà tutto lo spazio del form);
- **None** (il controllo non è ormeggiato).

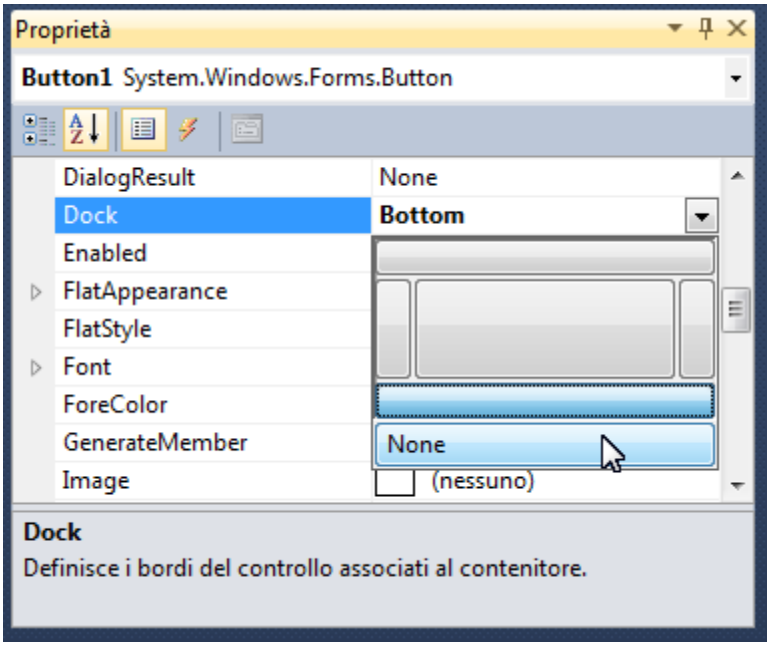


Figura 80: L'impostazione della proprietà Dock.

DoubleBuffered

La proprietà **DoubleBuffered**, se impostata **= True**, impone l'uso di un doppio spazio di memoria per l'effettuazione di operazioni grafiche. Il tempo richiesto per l'esecuzione di queste operazioni viene così ampliato, ma le operazioni grafiche risultano più stabili all'occhio umano; in particolare, viene eliminato l'effetto dello sfarfallìo di un'immagine, quando questa subisce modifiche in rapida sequenza.

La proprietà **DoubleBuffered** sarà ampiamente utilizzata, in questo manuale, soprattutto per i form, nella parte dedicata alla grafica.

Enabled

La proprietà **Enabled** indica se un controllo è attivo (**Enabled = True**, il controllo è in funzione e dunque risponde agli eventi) oppure no (**Enabled =** False).
Un controllo la cui proprietà Enabled è impostata come False rimane visibile sul form in modo sfumato, e non risponde ai movimenti del mouse o alla pressione dei tasti sulla tastiera, anche se per questi eventi sono scritte apposite procedure nel codice.
Durante l'esecuzione di un progetto, a seconda delle istruzioni scritte dal programmatore, un controllo può passare temporaneamente dalla impostazione **Enabled = True** alla impostazione **Enabled =** False e viceversa.

FlatAppearance

FlatStyle

La proprietà **FlatStyle** determina l'aspetto che un controllo assume al passaggio del mouse.
La proprietà **FlatStyle** può essere impostata in quattro modi diversi, tra i quali è particolarmente interessante l'impostazione **Flat**.
Con la proprietà **FlatStyle** impostata come **Flat**, infatti, è possibile impostare la proprietà **FlatAppearance**, che consente di scegliere i colori di primo piano, dello sfondo e del contorno di un controllo impostato come **Flat**.
Il controllo si presenta in questo modo *piatto*, ma il gioco dei colori può creare degli effetti grafici gradevoli e utili in particolare nei programmi per bambini.

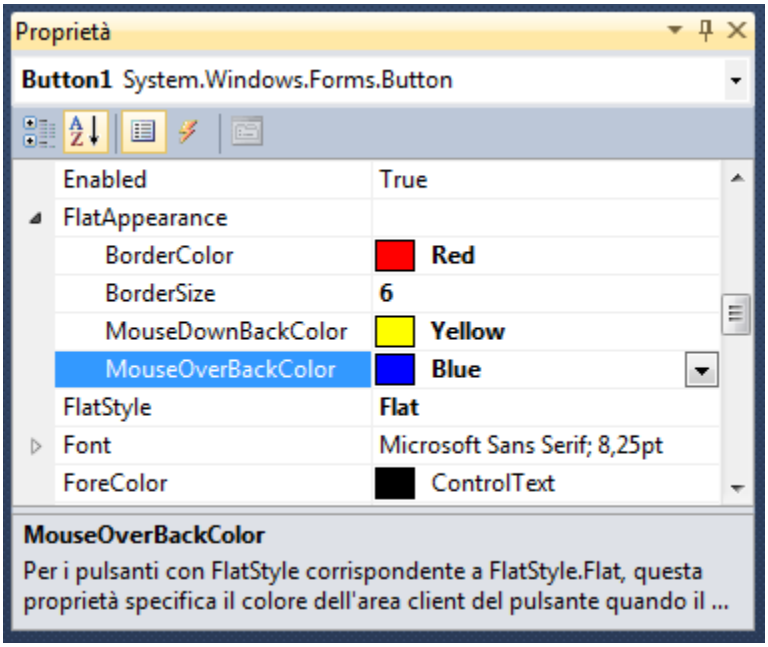


Figura 81: L'impostazione delle proprietà FlatStyle e FlatAppearance.

Font

La proprietà **Font** determina il tipo e le caratteristiche del carattere di scrittura di eventuali testi all'interno del controllo.
Nella casella a destra della proprietà **Font** compare un pulsante con tre puntini. Cliccando questo pulsante si apre la finestra per la selezione del tipo di carattere, nella quale si opera come in un programma di elaborazione di testi.
I controlli presenti in un form assumono tutti il tipo e le caratteristiche del font scelto per il form, a meno che vi siano impostazioni diverse per ogni controllo.
Se la proprietà **AutoScaleMode** di un form è impostata **= Font**, le dimensioni del form e di tutti i controlli in esso contenuti cambiano con il variare delle dimensioni del Font scelto.

ForeColor

La proprietà **ForeColor** imposta il colore del testo che compare in primo piano all'interno del controllo.

Image

ImageAlign

La proprietà **Image**, simile alla proprietà BackgroundImage, consente di scegliere un'immagine da visualizzare all'interno di un controllo.
La proprietà **ImageAlign** (allineamento della immagine) consente di determinare la posizione in cui si collocherà un'immagine scelta in precedenza con la proprietà Image.
Sono disponibili nove collocazioni diverse:

- in alto a sinistra (**TopLeft**);
- in alto al centro (**TopCenter**);
- in alto a destra (**TopRight**);
- al centro, a sinistra (**MiddleLeft**);
- al centro dell'oggetto (**MiddleCenter**);
- al centro, a destra (**MiddleRight**);
- in basso a sinistra (**BottomLeft**);
- in basso al centro (**BottomCenter**);
- in basso a destra (**BottomRight**).

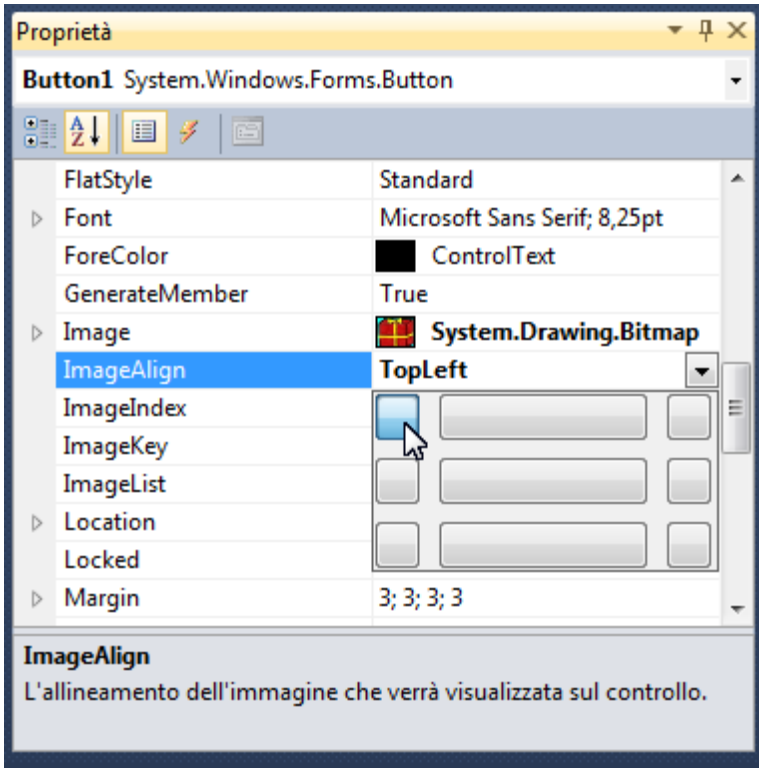


Figura 82: L'impostazione della proprietà ImageAlign.

ImageIndex

ImageKey

ImageList

Queste tre proprietà consentono di assegnare a un controllo un'immagine prelevata da un componente **ImageList**.
Il componente **ImageList** è un contenitore di immagini inserito in un programma, dal quale si possono prelevare le immagini secondo le esigenze del programmatore.
Ne vedremo caratteristiche e modalità di funzionamento più avanti, nella categoria dei componenti (oggetti non visibili).
In particolare:

- la proprietà ImageList **collega** il controllo a una lista di immagini già presente nel programma;
- la proprietà ImageKey indica il **nome** dell'immagine da prelevare nella lista di immagini;
- la proprietà ImageIndex indica il **numero** dell'immagine da prelevare nella lista di immagini.

Supponiamo di avere in un form alcuni pulsanti **Button**. Nello stesso form abbiamo alcune immagini contenute nel componente **ImageList1**.
In questa situazione è possibile impostare queste proprietà dei pulsanti **Button**:

Controllo	Proprietà	Impostazione
Button1	**ImageList**	**ImageList1**
	ImageKey	**Nome della prima immagine**
	ImageIndex	**0**
Button2	**ImageList**	**ImageList1**
	ImageKey	**Nome della seconda immagine**
	ImageIndex	**1**
ecc.		

Tabella 4: Modalità di impostazione delle proprietà ImageIndex, ImageKey e ImageList.

In pratica:

- la proprietà ImageList connette un controllo a un componente ImageList già presente nel form;

- la proprietà ImageKey consente di scegliere un'immagine, tra le immagini contenute in un componente ImageList, riferendosi a questa immagine con il suo nome proprio;
- la proprietà ImageIndex consente di scegliere un'immagine, tra le immagini contenute in un componente ImageList, riferendosi a questa immagine con il suo numero progressivo.

Locked

La proprietà **Locked** blocca un controllo nella sua posizione e nelle sue dimensioni, in modo che non possa più essere modificato dal programmatore, accidentalmente o per errore.
Lo stesso effetto si ottiene cliccando, nella barra dei menu, il menu **Formato / Blocca controlli**; cliccando il menu si ottiene il risultato di bloccare contemporaneamente tutti i controlli presenti nel form (la proprietà **Locked** di tutti i controlli viene automaticamente impostata **= True**).
Un controllo con la proprietà **Locked = True**, quando viene cliccato dal programmatore, in fase di programmazione, mostra un lucchetto nel suo angolo superiore sinistro:

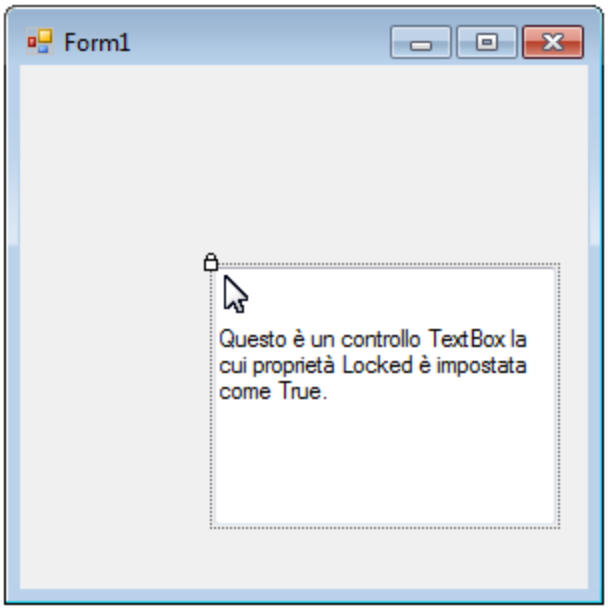


Figura 83: Un TextBox con la proprietà Locked = True.

Location

Size

Le proprietà **Location** e **Size** determinano la posizione e le dimensioni di un controllo.

Con la proprietà **Location** si impostano le **coordinate** di un **punto** che corrisponde all'angolo superiore sinistro del controllo.
Nell'ordine, le due coordinate del punto sono la sua distanza (X) dal margine sinistro del form e la sua distanza (Y) dal margine superiore del form.
Con la proprietà **Size** si impostano, nell'ordine, la larghezza (**Width**) e l'altezza (**Height**) di un controllo.
E' da notare che gli stessi dati compaiono sempre, per un riscontro più immediato, nella barra blu in basso a destra dell'ambiente di progettazione:

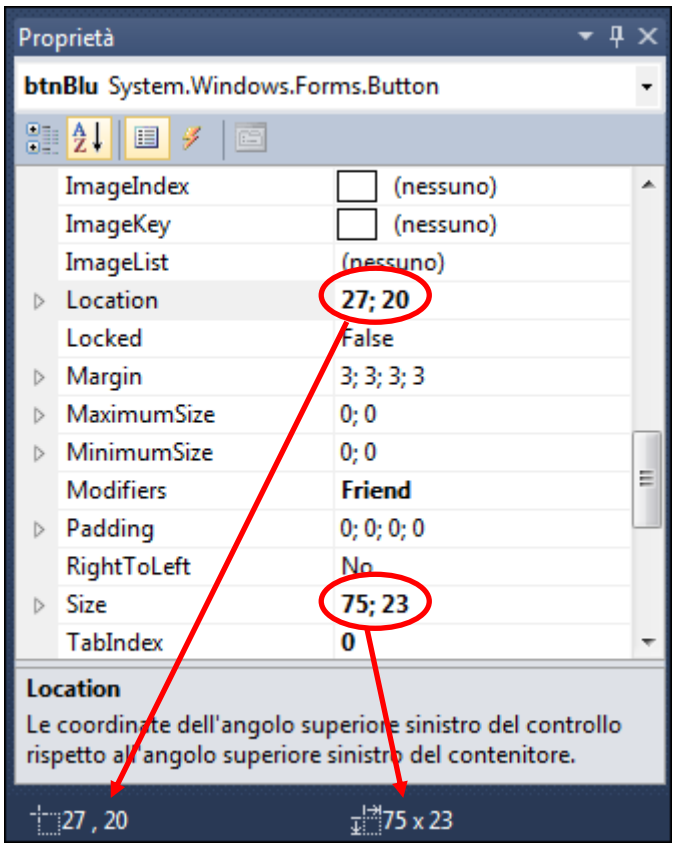


Figura 84: I dati delle proprietà Location e Size.

Queste due proprietà richiedono una sintassi particolare, se vengono impostate non da questa Finestra Proprietà ma all'interno del codice del programma.
Se è necessario modificare la posizione e le dimensioni di un pulsante Button1 nella fase di esecuzione di un programma, ad esempio, non è possibile scrivere
Button1.Location = 177;206
Button1.Size = 75;23
ma è necessario adottare questa sintassi:

```
Button1.Location = New Point(177, 206)
Button1.Size = New Size(75, 23)
```

Oppure questa:

```
Button1.Bounds = New Rectangle(177, 206, 75, 23)
```

In altri termini, per posizionare e dimensionare un controllo con comandi scritti nel codice bisogna creare

- un nuovo oggetto Point, con i due parametri X e Y scritti tra parentesi;
- un nuovo oggetto Size, con i due parametri Larghezza e Altezza scritti tra parentesi;

Oppure

- un nuovo oggetto Rectangle, con quattro parametri scritti tra parentesi: X, Y, Larghezza, Altezza.

TabIndex

La proprietà **TabIndex** riporta un numero progressivo che viene assegnato automaticamente da VB ai controlli, partendo da 0, man mano che questi vengono inseriti nel form.
Il programmatore ha la possibilità di cambiare l'ordine TabIndex dei controlli, secondo le sue esigenze.
Per visualizzare la proprietà TabIndex di tutti i controlli presenti in un form è necessario visualizzare prima la barra degli strumenti di **Layout** (visualizzazione).
Facciamo un *clic* sul menu **Visualizza / Barre degli strumenti / Layout**.
In questa barra, facciamo *clic* sull'ultimo pulsante a destra: questo pulsante comanda la visualizzazione della proprietà TabIndex. La proprietà viene visualizzata sull'angolo superiore a sinistra di ogni controllo, in un quadratino azzurro:

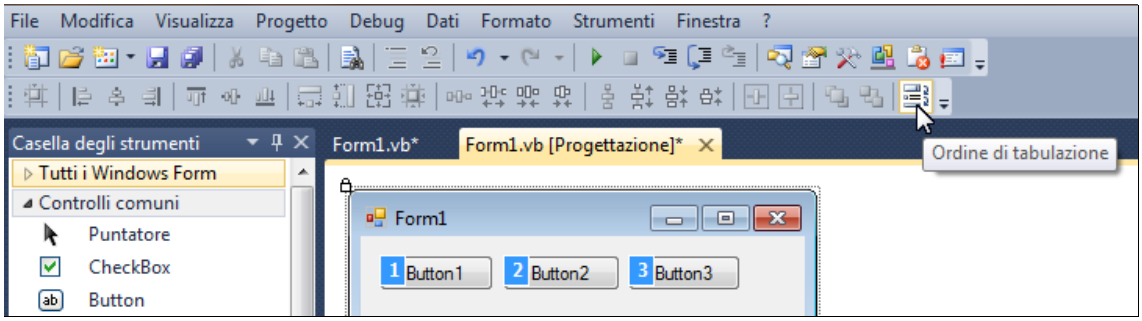


Figura 85: La barra a icone di Layout e la visualizzazione delle proprietà TabIndex.

Tag

La proprietà **Tag** è come un cartellino - a disposizione del programmatore - che rimane legato al controllo durante tutto lo svolgimento di un programma.
Questo cartellino può essere riempito dal programmatore con elementi vari (un numero, una lettera, una sigla, una parola, un testo) che accompagneranno il controllo nelle varie fasi di esecuzione del programma e potranno servire, ad esempio:

- come segno di riconoscimento del controllo stesso;
- come una didascalia che illustra il contenuto del controllo;
- come informazioni o liste di informazioni da passare da un oggetto all'altro.

Text

TextAlign

TextImageRelation

La proprietà **Text** contiene il testo da riportare all'interno di un controllo.
Questo testo può essere composto di una sola parola o di poche parole per il form o per controlli come il pulsante **Button**; può essere un testo molto lungo per i controlli dedicati alla gestione di testi come il controllo **Label** (etichette), il **TextBox** (contenitore di testo) e il **RichTextBox** (contenitore di testo potenziato con opzioni avanzate di formattazione).
La proprietà **TextAlign** indica dove va collocato il testo all'interno del controllo, oppure indica se, all'interno del contenitore il testo deve essere allineato a sinistra, al centro o a destra.
La proprietà **TextImageRelation** indica in che rapporto grafico debbono stare un testo e un'immagine, se sono entrambi presenti in un controllo. Vi sono quattro possibilità:

- **Overlay** (il testo compare in sovrimpressione sull'immagine);
- **ImageAboveText** (l'immagine compare al di sopra del testo);
- **TextAboveImage** (il testo compare al di sopra dell'immagine);
- **TextBeforeImage** (il testo compare prima dell'immagine, a sinistra dell'immagine).

Visible

La proprietà **Visible** determina se un controllo sarà visibile o no durante l'esecuzione del programma.
Un controllo la cui proprietà Visible è impostata = False rimane invisibile sul form e ovviamente non risponde ad alcun evento.
Durante l'esecuzione di un programma, a seconda delle istruzioni scritte dal programmatore, un controllo può passare temporaneamente dalla impostazione Visible = True alla impostazione Visible = False e viceversa.

31: Impostare le proprietà di un controllo dalla Finestra del Codice.

Le proprietà dei controlli possono essere impostate dal programmatore in fase di progettazione, dalla Finestra Proprietà, come abbiamo visto nel paragrafo precedente.
Anche nella fase di esecuzione di un programma, tuttavia, si può verificare l'opportunità di modificare le impostazioni delle proprietà di un controllo. Queste modifiche delle proprietà vengono scritte nel codice del programma con righe di comandi simili a questa:

```
        Button1.BackColor = Color.Yellow
        Button1.Location = New Point(100, 100)
```

La sintassi di questi comandi è questa:

- prima viene il nome del controllo interessato alla modifica della proprietà;
- segue un punto di separazione;
- poi viene il nome della proprietà da modificare;
- segue il simbolo = ;
- infine viene specificata dal programmatore la nuova impostazione della proprietà.

La modifica delle proprietà di un oggetto durante la fase di esecuzione di un programma è generalmente collegata a un evento: quando si realizza un determinato evento, si attiva la modifica della proprietà.
Ad esempio, quando si realizza l'evento di un *clic* con il mouse su un pulsante **Button**, può cambiare il colore di fondo del form. Le istruzioni relative andranno scritte in questo modo:

```
    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Me.BackColor = Color.Red

    End Sub
```

32: I principali eventi riconosciuti dai controlli.

I controlli contenuti nella Casella degli Strumenti non condividono solo un gruppo di qualità (proprietà) fondamentali: condividono anche un gruppo di eventi fondamentali, vale a dire che sono progettati in modo da segnalare un certo numero di eventi che si possono verificare durante l'esecuzione di un programma.
In questo paragrafo vedremo alcuni di questi eventi comuni a molti controlli, elencati in ordine alfabetico.

BackColorChanged

Viene segnalato da un controllo quando si cambia il suo colore di fondo, durante l'esecuzione di un programma.

Click

DoubleClick

Il *clic* e il doppio *clic* del mouse sono gli eventi che un controllo segnala quando l'utente preme un pulsante del mouse mentre il cursore è *sopra* il controllo. Sono tra gli eventi più frequenti all'interno di un programma.

DragDrop

DragOver

Sono gli eventi che un controllo segnala quando l'utente passa sopra di esso trascinando con il mouse un altro controllo (DragOver) e rilasciando il secondo controllo sopra il primo (DragDrop).

FontChanged

Viene segnalato da un controllo quando è modificato il suo set di caratteri.

ForeColorChanged

Viene segnalato da un controllo quando si cambia il suo colore in primo piano, durante l'esecuzione di un programma.

Load

Unload

Gli eventi **Load** e **Unload** sono gli eventi principali nella vita di un form.
L'evento **Load** (carica) è generato automaticamente all'avvio di un programma, quando viene *caricato* e visualizzato il primo form.
Lo stesso evento **Load** può essere causato dal programmatore quando, ad esempio, in un programma è necessario *caricare* e visualizzare un secondo form.
L'evento contrario è l'evento **Unload** (scarica), che viene generato alla chiusura di un form. Generalmente all'evento Unload è associata la visualizzazione di un messaggio rivolto all'utente di questo tenore: "*Vuoi uscire dal programma? Sei sicuro?*".

KeyDown

KeyPress

Sono due eventi segnalati da un controllo quando l'utente preme un tasto sulla tastiera.

MouseEnter

MouseMove

MouseLeave

Sono tre eventi causati dai movimenti del mouse: il primo (**MouseEnter**) si verifica quando il cursore del mouse entra nell'area del controllo, il secondo (**MouseMove**) si verifica quando l'utente muove il cursore del mouse entro l'area del controllo, il terzo (**MouseLeave**) si verifica quando il cursore del mouse esce dall'area del controllo.

Paint

Viene segnalato da un controllo quando si disegna qualcosa al suo interno.

TextChanged

Viene segnalato da un controllo quando si modifica il testo al suo interno.

Tick

E' un evento causato dal componente **Timer**, se questo è presente nel programma, e non dipende da una azione dell'utente.
L'evento **Tick** si genera ogni volta che il Timer *batte un colpo* (ad esempio: a ogni secondo; l'intervallo di tempo tra un Tick e l'altro dipende dall'impostazione della proprietà **Interval** del Timer).

33: La gestione di un evento nella fase di esecuzione di un programma.

La gestione degli eventi è compito specifico del programmatore; a lui spetta scrivere i comandi o le istruzioni che il programma dovrà eseguire quando si verificherà uno degli eventi che abbiamo visto elencati sopra.
La gestione di un evento è **scritta** in parti del codice del programma che si chiamano **procedure**.
Ogni procedura è composta di almeno tre righe:

- La prima riga inizia con la parola Sub e contiene il nome proprio della procedura (VB assegna un nome proprio in modo automatico, il programmatore può cambiare questo nome secondo i suoi gusti).
- La seconda riga (o lo spazio delle righe intermedie, che possono essere anche decine) contiene il comando o i comandi che il programma deve eseguire al verificarsi dell'evento.
- La terza riga conclude la procedura con le parole End Sub.

Ecco l'esempio di una procedura che è attivata dall'evento *clic del m*ouse su un pulsante Button1:

```
    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Me.BackColor = Color.Red

    End Sub
```

Notiamo, nella prima riga, che alla procedura è assegnato in modo automatico il nome Button1_Clik. E' possibile cambiare questo nome a piacere e il risultato non cambia:

```
    Private Sub ClicSulPulsante(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Me.BackColor = Color.Red

    End Sub
```

In questo esempio, il nome **Button1_Click** è stato modificato in **ClicSulPulsante**. La lettrice o il lettore troveranno le informazioni necessarie per la comprensione di tutti i termini della procedura (in particolare della prima riga) al paragrafo 68: Le procedure.

Capitolo 6: I CONTROLLI COMUNI.

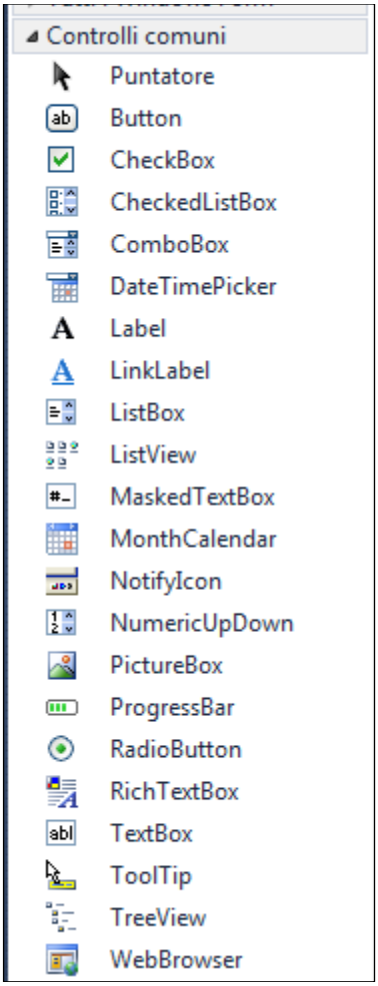


Figura 86: Il gruppo dei controlli comuni nella Casella degli Strumenti.

34: Il controllo Button.

Il controllo Button è un pulsante che risponde alle azioni del mouse: un *clic*, un doppio *clic*, un *clic* con il tasto destro del mouse, il passaggio del mouse sopra il pulsante sono tutti **eventi** che il Button riconosce e ai quali risponde eseguendo le eventuali istruzioni (**procedure**) scritte dal programmatore.

Il Button può mostrare una parola (ad esempio: "ESCI"), una frase (ad esempio: "Cliccami per cambiare colore"), può mostrare un'immagine, o tutte queste cose assieme. Può cambiare il colore dello sfondo e il colore delle parole in primo piano.

35: I controlli CheckBox e RadioButton.

I controlli CheckBox (= casella di scelta di una opzione) e RadioButton (= manopola dell'apparecchio radio) consentono all'utente del programma di effettuare delle scelte, tra le opzioni previste dal programmatore, utilizzando il mouse.
I controlli RadioButton sono a scelta esclusiva: consentono **una sola scelta**. Se l'utente cambia idea e seleziona un'altra opzione, l'opzione precedente viene automaticamente deselezionata.
I controlli **CheckBox** invece consentono all'utente di selezionare una, più di una, o anche tutte le opzioni disponibili.
All'interno di questi controlli, per illustrare le diverse opzioni, possono essere inseriti parole, numeri, una frase, un'immagine, o tutte queste cose insieme.
La proprietà caratteristica di questi controlli è la proprietà **Checked**: Se questa è impostata **= True**, sul controllo in questione appare il segno di spunta.
Il controllo CheckBox dispone della proprietà **ThreeState**. Se questa proprietà è impostata **= True**, il controllo in questione avrà tre modi di selezione, invece dei due abituali (alle due opzioni **spuntato** / **non** spuntato si aggiunge una opzione intermedia **indeterminata**);
I due controlli dispongono della proprietà **Appearance**, mediante la quale è possibile modificare la loro forma, passando dalla forma standard alla forma a pulsante. Anche nella forma a pulsante rimane il carattere esclusivo del *clic* su un RadioButton: selezionando una opzione si deselezionano le altre.
Nell'immagine seguente vediamo a sinistra un gruppo di CheckBox che mostrano le tre modalità di selezione, a destra un gruppo di tre RadioButton con la forma standard e un gruppo di tre RadioButton con la forma a pulsante.

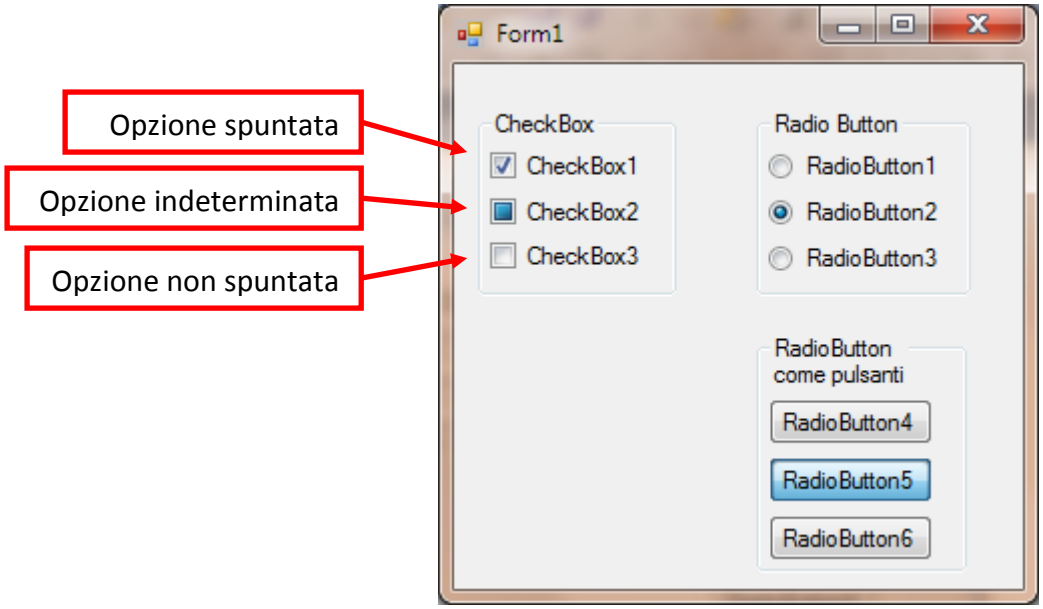


Figura 87: Gruppi di CheckBox e di RadioButton in un form.

36: I controlli CheckedListBox, ListBox e ComboBox.

I controlli **CheckedListBox** (contenitore di un elenco con il segno di spunta), **ListBox** (contenitore di un elenco), e **ComboBox** (contenitore combinato, composto di un elenco di opzioni e di una casella di testo) si utilizzano per presentare all'utente del programma una lista di items da selezionare; l'utente può selezionare uno o più item tra quelli presenti nella lista.
Gli item della lista possono essere presentati in ordine alfabetico (proprietà **Sorted** = **True**).
Se l'elenco degli item è più lungo della dimensione altezza del controllo, questo mostra automaticamente, a destra, una barra per lo scorrimento delle voci.
Il controllo **CheckedListBox** (contenitore di un elenco con il segno di spunta) consente la selezione di più item e presenta un quadratino, a sinistra degli item, in cui compare il segno di spunta per gli item selezionati.
Per rendere immediata la selezione da parte dell'utente, è utile impostare la proprietà **CheckOnClick** di questo controllo **= True**.
Il controllo **ListBox** (contenitore di un elenco) non presenta il quadratino con il segno di spunta e consente la selezione di un solo item (l'item selezionato viene evidenziato con il sottofondo blu).
Il controllo **ComboBox** (contenitore combinato, composto di un elenco di opzioni e di una casella di testo) è costituito da un contenitore con un elenco di item come il controllo ListBox, ma presenta, in testa alla lista, un contenitore di testo a disposizione dell'utente del programma. Questi può selezionare una voce già presente nella lista,

oppure può aggiungere e selezionare una voce nuova, scrivendola nel contenitore di testo.
A differenza degli altri due controlli, il controllo ComboBox dispone della proprietà **Text**, nella quale si può scrivere una parola o una frase che comparirà in testa alla lista degli item (ad esempio: *Scegli una di queste opzioni:*).

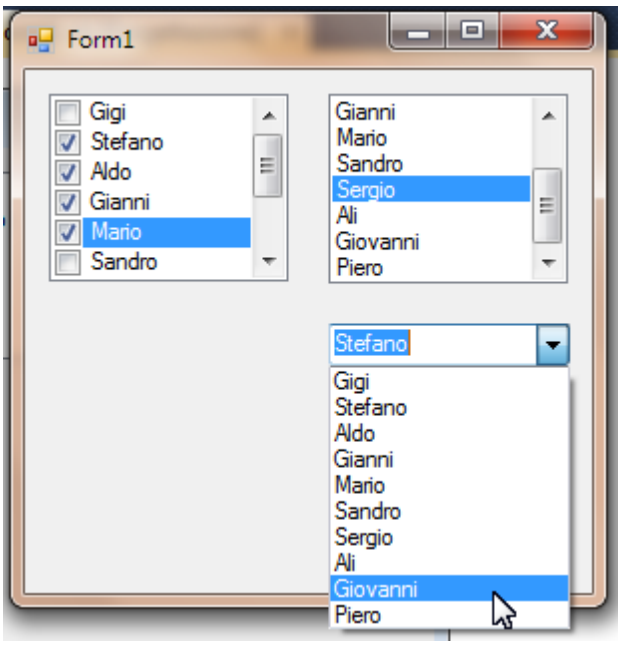


Figura 88: I controlli CheckedListBox, ListBox e ComboBox.

ListView

Anche il controllo **ListView**, come i controlli **CheckedListBox**, **ListBox** e **ComboBox**, consente di presentare all'utente del programma una serie di item da selezionare. **ListView** è dedicato alla visualizzazione di cataloghi o di tabulati, con più colonne, nei quali l'utente può selezionare una intera riga.

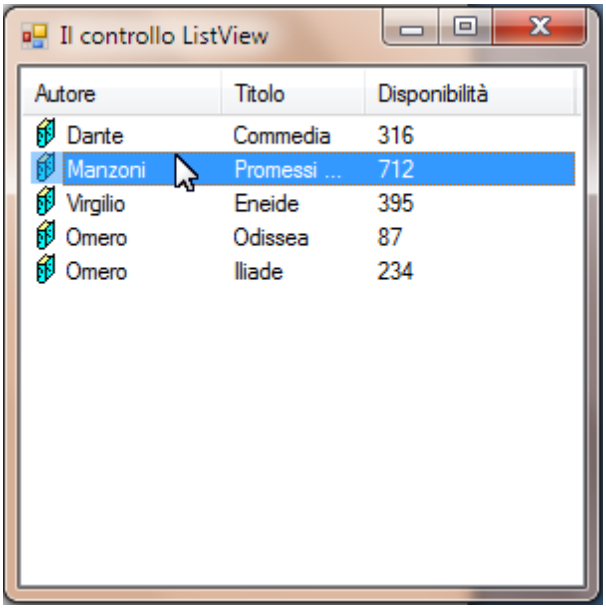
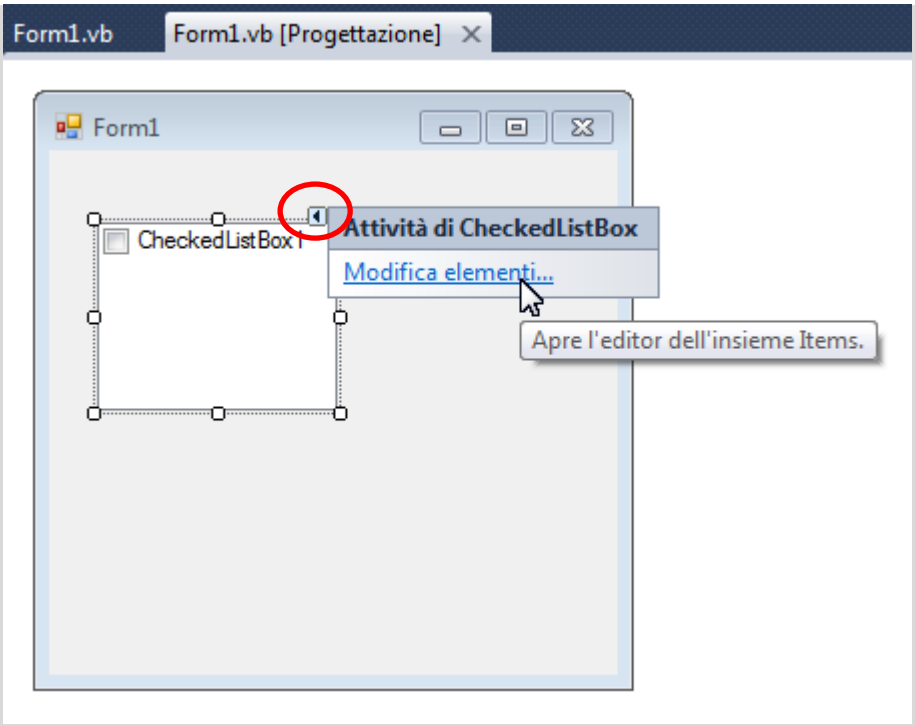


Figura 89: Il controllo ListView.

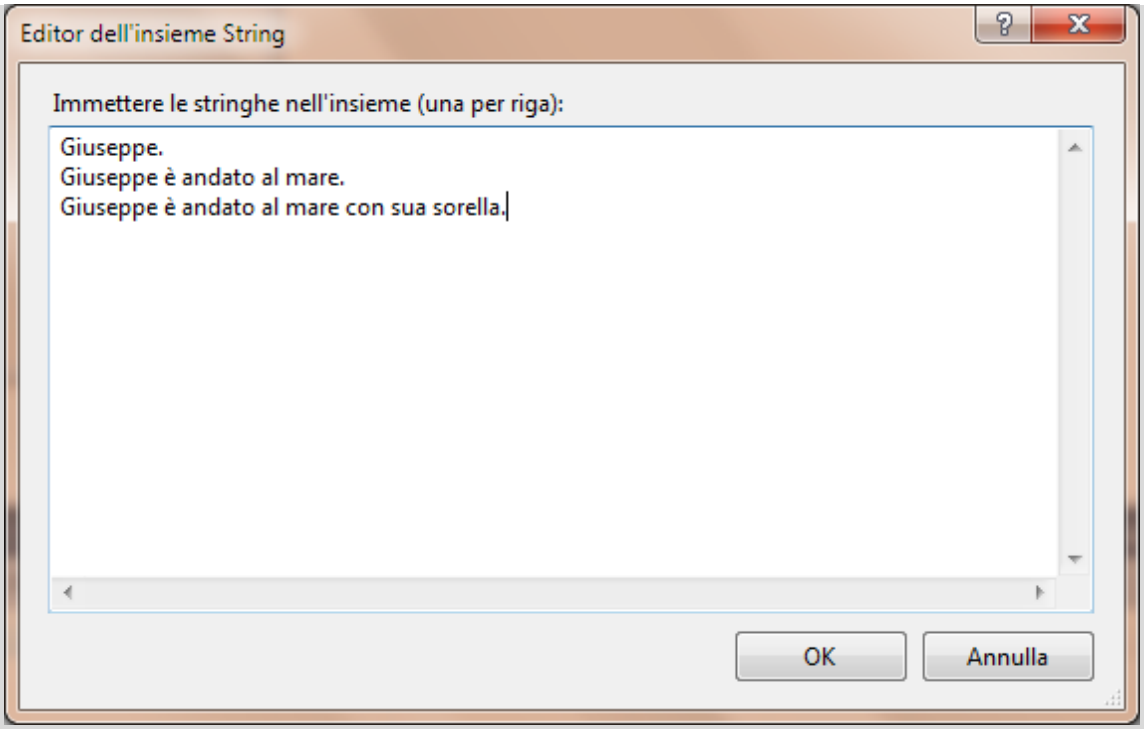
Esercizio 9: Inserimento di un elenco di item in un controllo CheckedListBox.

In questo esercizio vedremo come inserire un elenco di item in un controllo **CheckedListBox** (le modalità sono le stesse anche per i controlli **ListBox** e **ComboBox**). Apriamo un nuovo progetto e collochiamo nel Form un controllo **CheckedListBox**. Notiamo sull'angolo superiore destro del controllo una freccina nera mediante la quale si accede a un menu di scelta rapida:

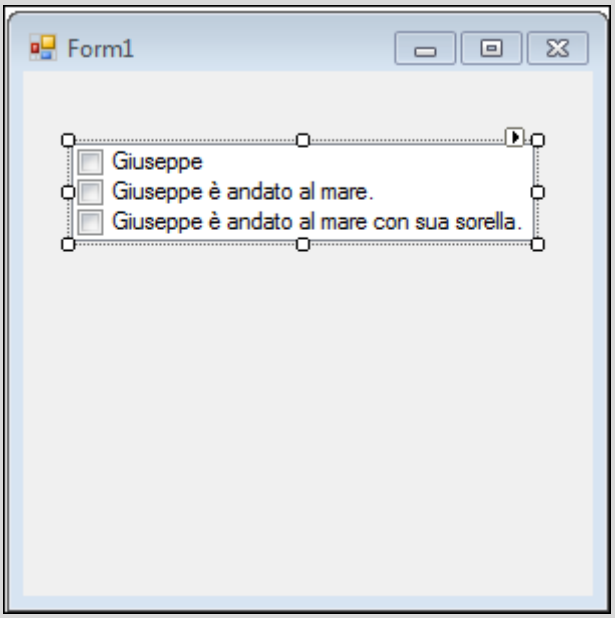


In questo menu, facciamo un *clic* con il mouse su **Modifica elementi**.
Questo ci porta alla finestra dell'**Editor di stringhe**[34], dove possiamo scrivere le voci che vogliamo fare comparire nel CheckedListBox, andando a capo dopo ogni voce.

[34] Una **stringa** è una *striscia* di testo che VB considera come un blocco unico, o una sola parola, qualsiasi sia il suo contenuto. Ad esempio, VB considera queste tre *strisce* di testo come tre blocchi unitari di testo e li tratta come se fossero tre parole:
- "Giuseppe"
- "Giuseppe è andato al mare."
- "Giuseppe è andato al mare con sua sorella."



Terminato di scrivere l'elenco delle voci, facciamo un *clic* su OK.
Ecco come compare il controllo CheckedListBox, dopo averlo dimensionato con il mouse, con i tre item che vi abbiamo inserito:



Vi è un altro modo per visualizzare l'Editor di stringhe: nel Form1, facciamo un *clic* sul controllo **ClickedListBox** e andiamo a scorrere l'elenco delle sue proprietà nella Finestra Proprietà.
Con un *clic* sulla proprietà **Items** possiamo visualizzare l'Editor di stringhe.

Come abbiamo detto all'inizio di questo esercizio, l'immissione di item tramite l'Editor di stringhe funziona allo stesso modo anche per i controlli simili al **CheckedListBox**: il **ListBox** e il **ComboBox**.

37: I controlli DateTimePicker e MonthCalendar.

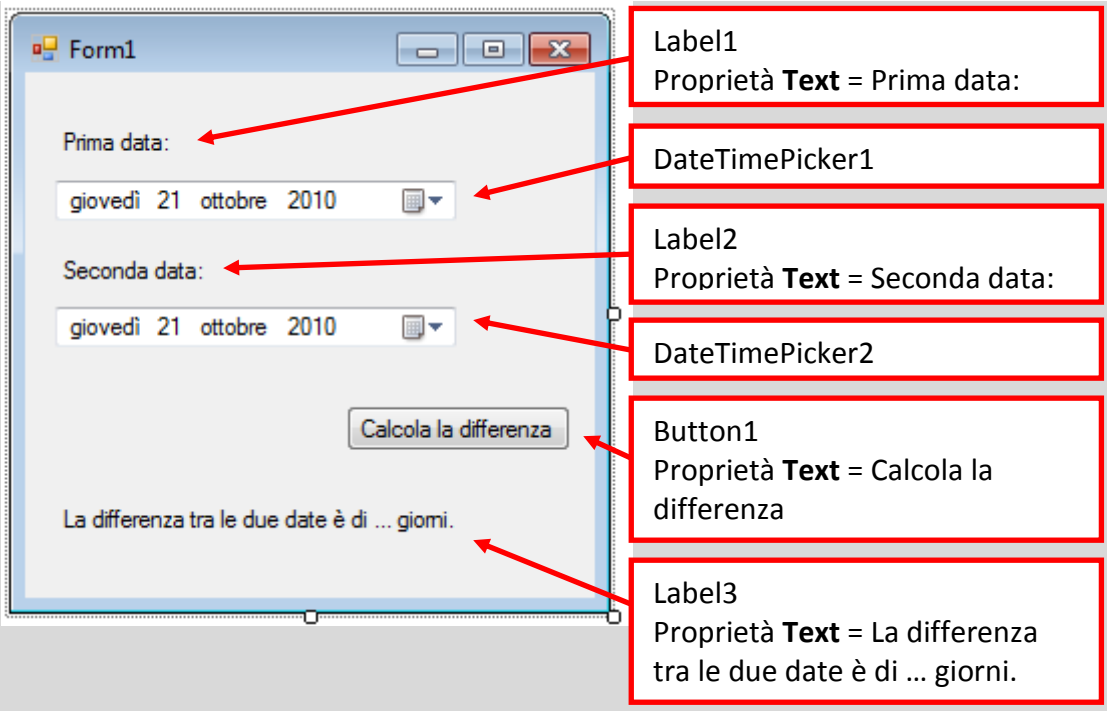
DateTimePicker (= selezione di una data) e **MonthCalendar** (= calendario mensile) sono due controlli connessi alla consultazione del calendario.
Ogni controllo, come sappiamo, è un oggetto con una propria programmazione - preparata da altri - che il programmatore può prendere e collocare nei suoi programmi evitando una notevole mole di lavoro.
Con controlli come **DateTimePicker** e **MonthCalendar** si può apprezzare pienamente il valore della programmazione a oggetti, perché questi due controlli sono due veri e propri programmi completi, capaci di operazioni complesse, che il programmatore può inserire nei suoi programmi e utilizzare senza scrivere una sola riga di codice.
Il controllo **MonthCalendar** visualizza un calendario *sfogliabile* mese per mese con giorni, mesi ed anni dal 1753 al 9998.
Il controllo **DateTimePicker** consente di visualizzare lo stesso calendario, con la possibilità di selezionare una data all'interno di esso. Il controllo si dispone automaticamente nel form come un menu a tendina che l'utente può aprire e scorrere per selezionare una data nel calendario.

Esercizio 10: Calcolo della differenza tra due date.

In questo esercizio utilizzeremo due controlli **DateTimePicker** per fare selezionare all'utente del programma due date diverse.
Il programma si incaricherà di calcolare i giorni di differenza tra le due date.
Apriamo un nuovo progetto e collochiamo sul Form1:

- due controlli DateTimePicker,
- un controllo Button e
- tre controlli Label

come in questa immagine:



Ora facciamo un doppio *clic* sul Button1 per accedere alla Finestra del Codice; scriviamo nella finestra questa procedura[35]:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim Differenza As Integer

        Differenza = DateDiff(DateInterval.DayOfYear, DateTimePicker1.Value,
DateTimePicker2.Value)

        Label1.Text = "La differenza tra le due date è di " & Differenza & "
giorni."

    End Sub

End Class
```

La procedura gestisce l'evento del *clic* del mouse sul Button1. Vediamo cosa dicono le tre righe centrali.

```
        Dim Differenza As Integer
```

[35] E' possibile copiare il codice e incollarlo direttamente nella Finestra del Codice. Ricordiamo anche che i listati degli esercizi sono disponibili nella cartella **Documenti / A scuola con VB 2010 / Esercizi**. Possono essere utilizzati per il confronto con il lavoro eseguito dal lettore, oppure possono essere copiati e incollati nella Finestra del Codice.

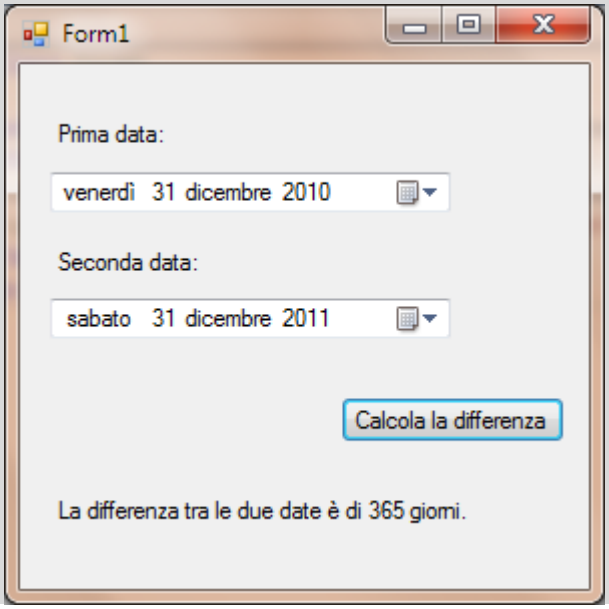
La prima riga istruisce il computer affinché riservi una casella di memoria a una variabile denominata Differenza. La variabile è destinata a ricevere solo valori numerici interi (As Integer).

```
Differenza = DateDiff(DateInterval.DayOfYear, DateTimePicker1.Value,
DateTimePicker2.Value)
```

La seconda riga effettua il calcolo della differenza (`DateDiff`) tra i valori (`Value`) contenuti nei due controlli DateTimePicker. Sembra una scrittura piuttosto complessa, ma se la scriviamo facendo attenzione ai suggerimenti di **Intellisense** l'operazione si semplifica notevolmente.

```
Label1.Text = "La differenza tra le due date è di " & Differenza & " giorni."
```

La terza riga assegna alla Label1 un testo con la frase `"La differenza tra le due date è di "`, alla quale va aggiunto il valore numerico della differenza tra le due date (`Differenza`), al quale va aggiunta la parola `" giorni."`
Ora mandiamo in esecuzione il programma per verificarne il funzionamento.



38: Il controllo Label.

Il controllo **Label** (etichetta) è un riquadro destinato a contenere testi di varia natura e lunghezza come, ad esempio, didascalie, titoli, note, spiegazioni, istruzioni.
Il testo da visualizzare nella label viene scritto dal programmatore nella proprietà **Text** (il contenuto della proprietà Text può essere modificato a piacere, nel corso di esecuzione del programma, secondo le indicazioni del programmatore).

Il controllo si presenta con la proprietà **Autosize = True** e si adatta automaticamente al testo in esso contenuto ed alla larghezza dei caratteri del testo. Per questo motivo, al momento dell'inserimento nel form il controllo si presenta senza le maniglie di dimensionamento, maniglie di dimensionamento che compaiono invece impostando la proprietà **AutoSize = False**.
Il testo contenuto in un controllo Label può essere solo **letto** dall'utente del programma; questi non ha nessuna possibilità di scrivere all'interno di una Label per modificarne il contenuto.
Il controllo può anche visualizzare al suo interno un'immagine, in aggiunta al testo, impostando la proprietà **Image**, ma in questo caso - a differenza di quanto avviene per il testo – le sue dimensioni non si adattano automaticamente a quelle della immagine.
Siccome la larghezza del controllo si adatta alla lunghezza del testo, è possibile che un testo troppo lungo faccia finire una parte della label oltre i limiti del form.
Per evitare questo inconveniente è opportuno impostare la proprietà **Autosize** della label **= False**: in questo modo il programmatore può dare alla label le dimensioni adatte a visualizzare tutto il testo; il testo contenuto nella label va a capo automaticamente, senza dividere le parole in sillabe, quando raggiunge il limite destro del controllo.
Se il programmatore vuole dividere le linee del testo in modo che questo vada a capo in punti particolari, deve indicare il punto di separazione delle linee nel riquadro della proprietà **Text** premendo il tasto ENTER sulla tastiera per ogni interruzione di riga.
Ad esempio, nella figura seguente si vede la proprietà Text di una label con una lista di nomi. Se il programmatore li scrive uno di seguito all'altro nella proprietà Text, questi compaiono scritti allo stesso modo all'interno della Label. Se il programmatore va a capo dopo ogni nome, la stessa cosa avviene all'interno della label.

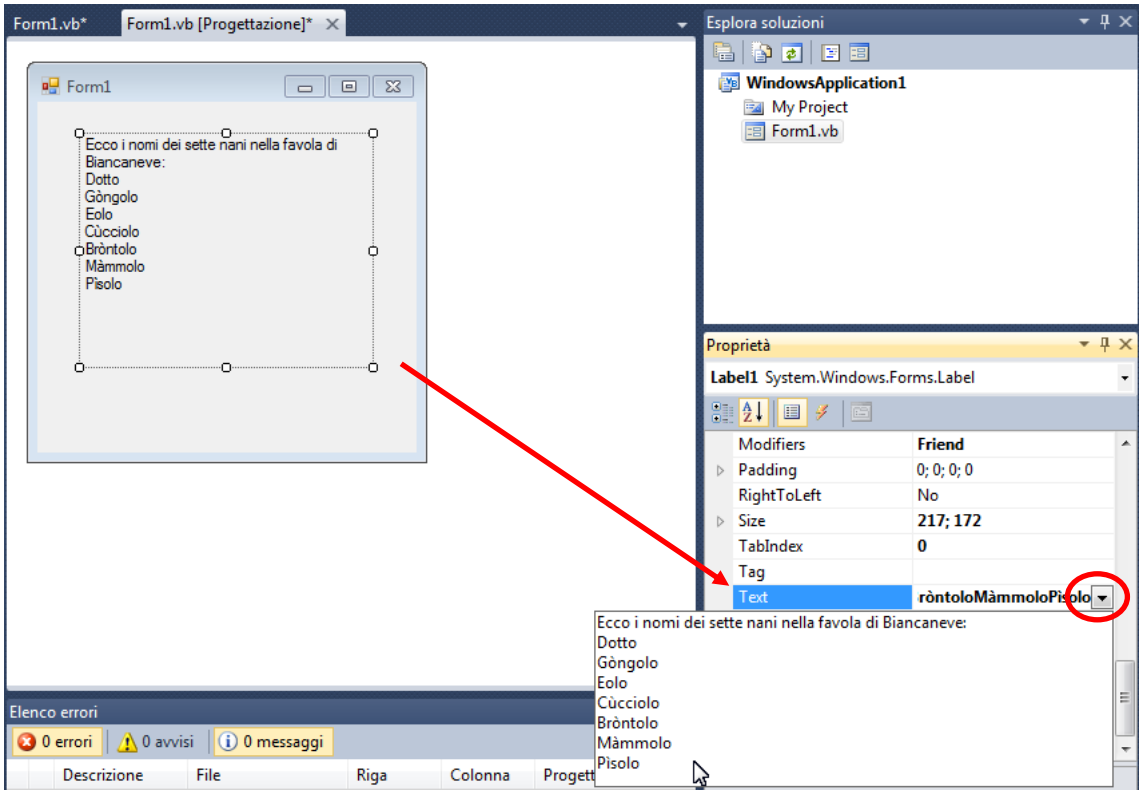
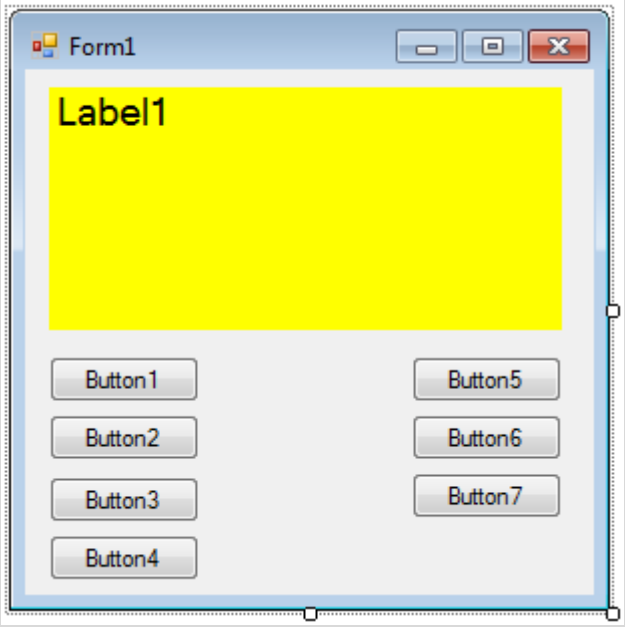


Figura 90: L'inserimento di un testo *spezzato* in un controllo Label.

Il testo contenuto in una label può essere modificato o scritto *ex novo* nel corso della esecuzione di un programma, come vedremo nel prossimo esercizio.

Esercizio 11: Impostare la proprietà Text di un controllo Label dalla Finestra del Codice.

Apriamo un nuovo progetto. Inseriamo nel Form1 sette pulsanti Button e una Label come in questa immagine:



Impostiamo queste proprietà del controllo **Label1**:

- **AutoSize = False**
- **BackColor = Yellow**
- **Font = Microsoft Sans Serif a 14 punti**

Obiettivo dell'esercizio è far sì che, premendo i diversi pulsanti, sia visualizzato all'interno della Label1 un testo formattato in modi diversi. In particolare, i primi quattro pulsanti modificheranno la proprietà **Text** della Label, mentre gli altri tre pulsanti ne modificheranno la proprietà **TextAlign**, allineando il testo sulla sinistra, al centro o sulla destra del controllo.
Facciamo un doppio *clic* sul **Button1**. Accediamo così alla Finestra Proprietà dove troviamo già impostata la procedura per gestire l'evento del *clic* del mouse su questo pulsante:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    End Sub
```

Aggiungiamo nella riga centrale, per ora vuota, un comando per visualizzare una frase all'interno della Label1:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Label1.Text = "Questo testo è scritto su una sola linea e va a capo in
modo automatico."

    End Sub
```

Notiamo che il testo da visualizzare nella Label1 è scritto tra virgolette e assume un colore diverso dalle altre parti del codice.

Torniamo nella Finestra di Progettazione, facciamo un doppio *clic* sul **Button2** e completiamo la procedura relativa all'evento del *clic* del mouse su questo pulsante:

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

        Label1.Text = "Questo testo va a capo" & vbCrLf & "tre volte" & vbCrLf &
"in modo forzato" & vbCrLf & "usando l'istruzione vbCrLf"

    End Sub
```

Nella riga centrale, il testo da visualizzare nella Label1 è interrotto più volte dal simbolo di unione **&** e dalla sigla **vbCrLf**. La sigla indica che in quel punto il testo dovrà andare a capo, iniziando una nuova linea[36].
Notiamo che ogni spezzone di testo è scritto tra virgolette e che i vari spezzoni sono sempre collegati tra di loro dal simbolo di unione &.
Continuiamo con la procedura relativa al pulsante Button3:

```
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click

        Label1.Text = "Questo testo è composto di tre frasi. " & vbCrLf & _
        "Questa è la seconda frase." & vbCrLf & _
        "Questa è la terza frase."

    End Sub
```

Anche in questo caso troviamo un testo spezzato in più parti; ogni parte è collegata alla parte precedente con i simboli di unione **&**; nel mezzo delle unioni troviamo la sigla **vbCrLf** che comanda l'inizio di una nuova riga. A differenza della procedura precedente, però, in questo caso il programmatore non ha scritto tutta l'istruzione su una sola riga ma, per comodità di lettura, è andato a capo più volte utilizzando il trattino **_**. Questo trattino, posto al termine di una riga di istruzioni, indica a VB che l'istruzione, non finita, continua alla riga successiva. Notiamo che ogni trattino di continuazione è preceduto da uno spazio e che l'ultima riga non ha il trattino di continuazione.
La procedura da completare per il **Button4** è questa:

```
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click

        Label1.Text &= vbCrLf & "Questa è una frase aggiunta."

    End Sub
```

Questa procedura comanda un effetto nuovo rispetto a quanto abbiamo visto sino a ora.

[36] vbCrLf è l'acronimo **di visual basic Carriage return Line feed** (= ritorno del carrello e inizio di una nuova linea).

Mentre con i *clic* sui pulsanti 1, 2 e 3 abbiamo scritto un testo nella Label1 cancellando il testo già presente, questa procedura aggiunge altro testo al testo eventualmente già presente nella Label1.
Notiamo che essa assegna alla Label1 il testo già presente nella Label1 e vi aggiunge la frase `"Questa è una frase aggiunta."`.
I simboli utilizzati per questa operazione sono due: +=.

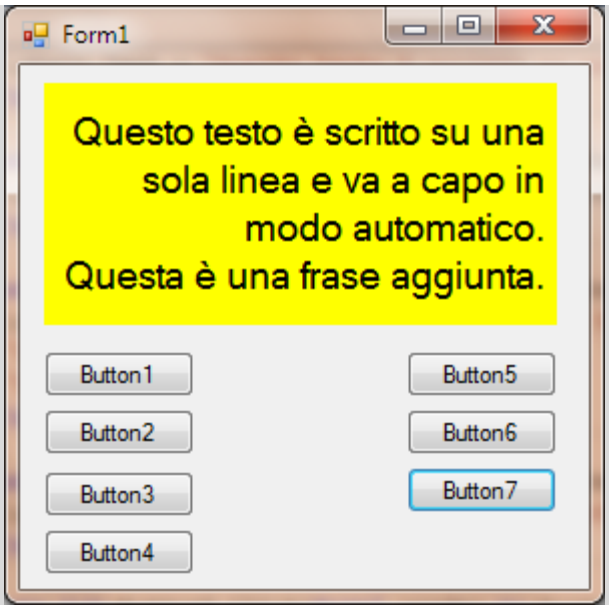
Le procedure relative agli ultimi tre pulsanti comandano l'allineamento del testo della Label1, rispettivamente, allineato sulla sinistra della Label, centrato, oppure allineato sulla destra:

```
    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click

        Label1.TextAlign = ContentAlignment.MiddleLeft

    End Sub
```

```
    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click

        Label1.TextAlign = ContentAlignment.MiddleCenter

    End Sub
```

```
    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click

        Label1.TextAlign = ContentAlignment.MiddleRight

    End Sub
```

Nell'immagine seguente vediamo un esempio di funzionamento del programma. In questo sono stati premuti i pulsanti Button1, Button4 e Button7.



LinkLabel

Il controllo **LinkLabel** visualizza una label alla quale è associato un link a un indirizzo di un sito web o a un indirizzo di posta elettronica.
Perché il *clic* sul link abbia un seguito e apra effettivamente la pagina richiesta, tuttavia, è necessario scrivere nel codice del programma una procedura di questo tipo:

```
    Private Sub LinkLabel1_LinkClicked(sender As System.Object, e As
System.Windows.Forms.LinkLabelLinkClickedEventArgs) Handles
LinkLabel1.LinkClicked

        System.Diagnostics.Process.Start("http://vbscuola.it")

    End Sub
```

39: Il controllo NumericUpDown.

Il controllo **NumericUpDown** visualizza una casella di testo con un numero che può essere aumentato o diminuito dall'utente, facendo un *clic* sulle due freccine presenti all'interno del controllo, oppure premendo i tasti con le frecce "su" e "giù" sulla tastiera.
L'utente può anche fare un *clic* sul controllo e scrivervi direttamente un numero.
La visualizzazione del numero all'interno del controllo può essere adattata a esigenze diverse, impostando queste proprietà del controllo:

DecimalPlaces	Indica il numero delle cifre che dovranno comparire dopo la virgola.
Increment	Indica il valore dell'aumento o della diminuzione del numero a ogni *clic* del mouse.
Maximum	Indica il valore massimo oltre il quale il controllo non potrà andare.
Minimum	Indica il valore minimo oltre il quale il controllo non potrà andare (accetta anche i numeri negativi).
TextAlign	Indica se il numero deve essere allineato a destra o a sinistra.
ThousandsSeparator	Indica se deve essere visualizzato o meno il punto di separazione delle migliaia.

Tabella 5: Proprietà del controllo NumericUpDown.

Ecco l'esempio di un controllo NumericUpDown con queste proprietà impostate come segue:

- DecimalPlaces = 2
- Font = Microsoft Sans Serif; 18pt
- Maximum = 10000
- TextAlign = Right
- ThousandsSeparator = True

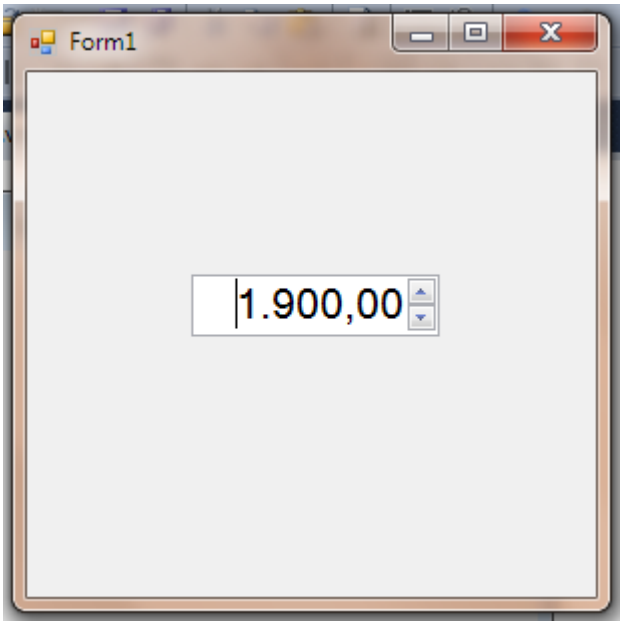


Figura 91: Il controllo NumericDown in azione.

40: Il controllo PictureBox.

Il controllo **PictureBox** (contenitore d'immagini) inserisce nel form un riquadro finalizzato a visualizzare immagini.
In questo riquadro si possono anche tracciare linee, rettangoli, ellissi e altre forme.
La proprietà fondamentale del controllo è la proprietà **Image**, che determina l'immagine da visualizzare; essa può essere scelta, modificata o sostituita in fase di esecuzione di un programma (per queste operazioni rinviamo la lettrice o il lettore alla parte del manuale dedicata alla grafica).

La proprietà **SizeMode** determina come questa immagine deve essere visualizzata. Sono disponibili cinque possibilità di visualizzazione:

- **Normal** (l'immagine è visualizzata nelle sue dimensioni normali, partendo dall'angolo superiore sinistro del controllo PictureBox);

- **StretchMode** (l'immagine si adatta alle dimensioni del controllo PictureBox, eventualmente distorcendo il proprio rapporto larghezza/altezza per adattarsi al rapporto larghezza/altezza del PictureBox);
- **AutoSize** (il controllo PictureBox si adatta alle dimensioni dell'immagine in esso contenuta);
- **CenterImage** (l'immagine si colloca al centro del PictureBox);
- **Zoom** (l'immagine si adatta alle dimensioni del controllo PictureBox, ma mantiene il proprio rapporto larghezza/altezza).

Il controllo PictureBox può avere anche un'immagine di sfondo, scelta impostando la proprietà **BackGroundImage**.
Se è stata impostata un'immagine di sfondo, l'immagine principale compare in primo piano e l'immagine sullo sfondo è visualizzata, se lo spazio lo consente, solo nella parte non coperta dalla immagine principale.
La figura seguente mostra due controlli PictureBox in uno stesso form. La proprietà **SizeMode** del PictureBox di sinistra è impostata come **StretchImage**, la proprietà **SizeMode** del secondo è impostata come **Zoom**:

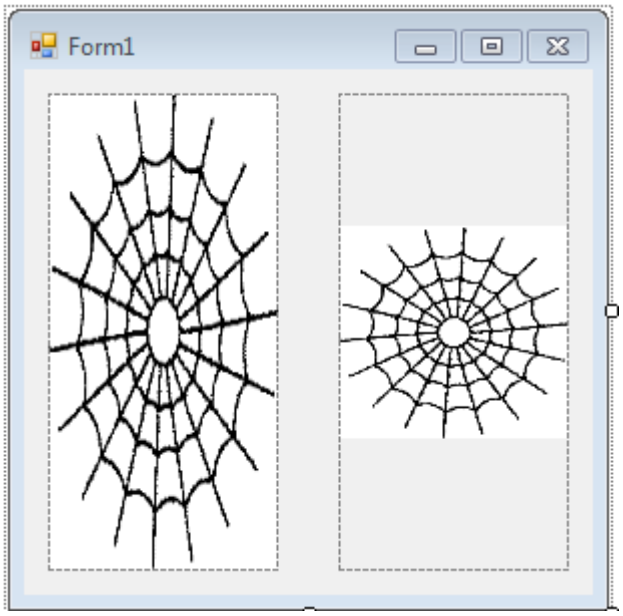
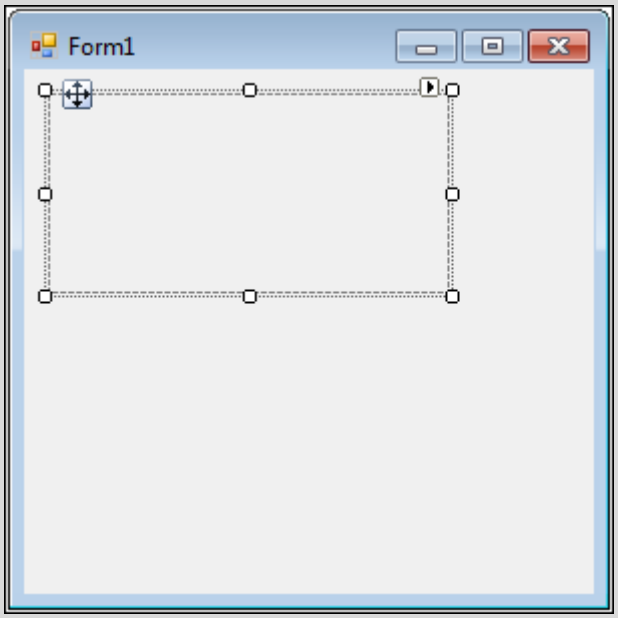


Figura 92: Il controllo PictureBox.

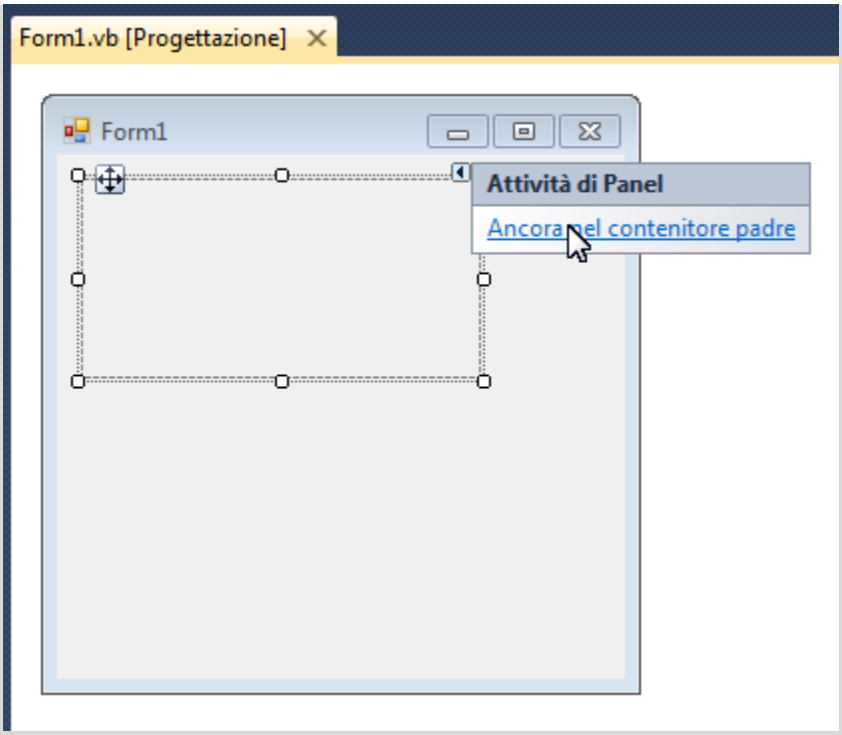
L'esercizio seguente mostra come visualizzare un'immagine troppo grande, le cui dimensioni eccedano quelle del form contenitore, senza ridurla, visualizzando delle barre di scorrimento per fare scorrere l'immagine in senso orizzontale o in senso verticale.

Esercizio 12: Un controllo PictureBox con le barre di scorrimento.

In questo esercizio vedremo come visualizzare un'immagine di grandi dimensioni all'interno di un PictureBox, senza ridurla, con delle barre di scorrimento orizzontale e verticale.
Per avere le barre di scorrimento, che non fanno parte delle proprietà del controllo PictureBox, ricorriamo a un accorgimento: inseriamo il controllo PictureBox in un controllo **Panel**, che è un controllo contenitore di cui parleremo più avanti nel manuale.
Apriamo un nuovo progetto e inseriamo nel form un controllo **Panel** come in questa immagine:



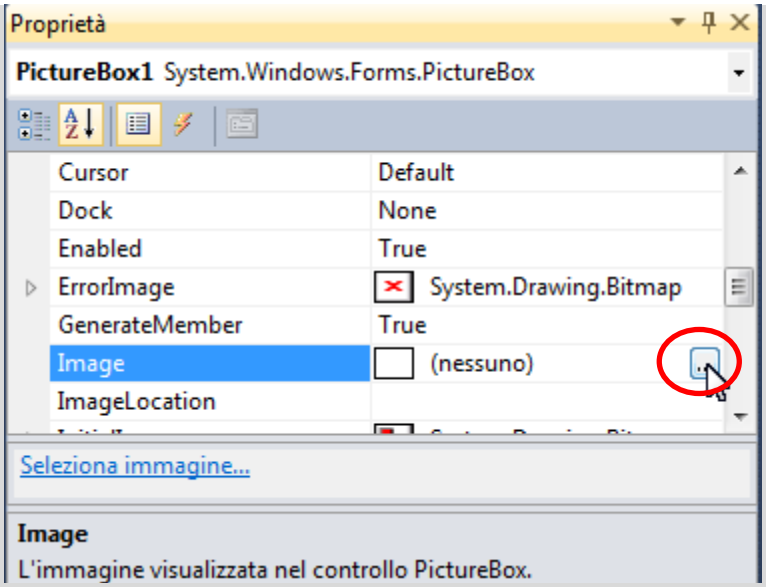
Facciamo un *clic* sulla freccina nera in alto a destra del controllo, e poi un *clic* su Ancora nel contenitore padre:



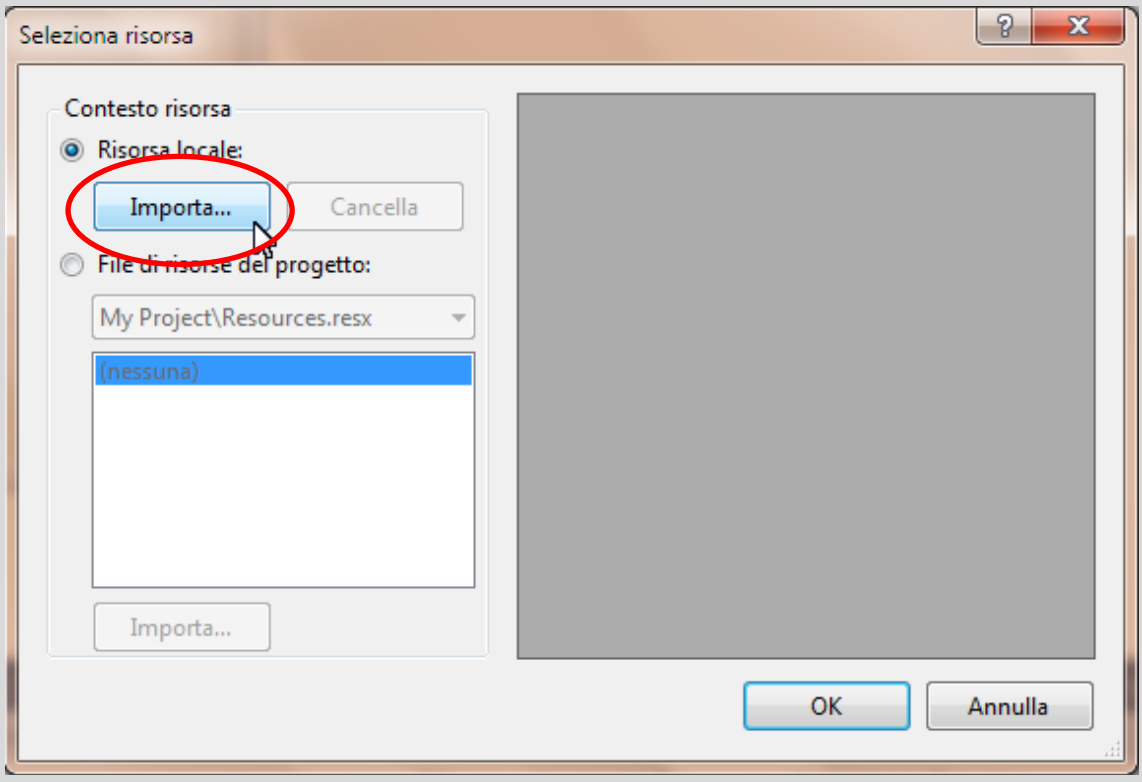
Questo fa sì che il controllo Panel si adatti alle dimensioni del Form, occupandone tutto lo spazio.
Impostiamo la proprietà **Autoscroll** del controllo Panel **= True**. Con questa impostazione il controllo mostrerà automaticamente le barre di scorrimento orizzontale e verticale, quando necessario.
Ora inseriamo nel controllo Panel un controllo **PictureBox**, con queste proprietà:

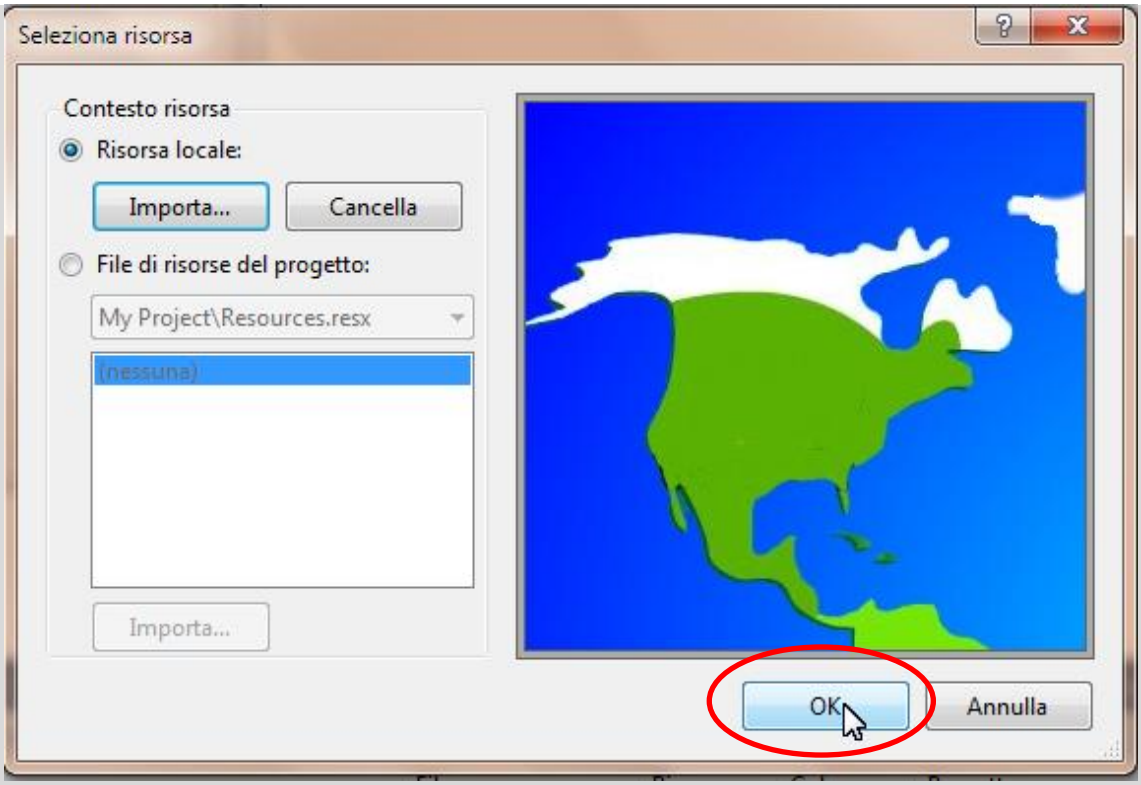
- **Location = 0; 0**
- **Size = 800; 600**

Le dimensioni di 800 x 600 pixel, assegnate al controllo PictureBox, eccedono le dimensioni del controllo contenitore, per cui questo mostrerà le barre di scorrimento del suo contenuto.
Ora visualizziamo nel controllo PictureBox l'immagine **Planisfero.jpg**, che si trova nella cartella **Documenti \ A scuola con VB \ Immagini**.
Per visualizzare questa immagine facciamo *clic* sulla proprietà **Image** del controllo PictureBox:

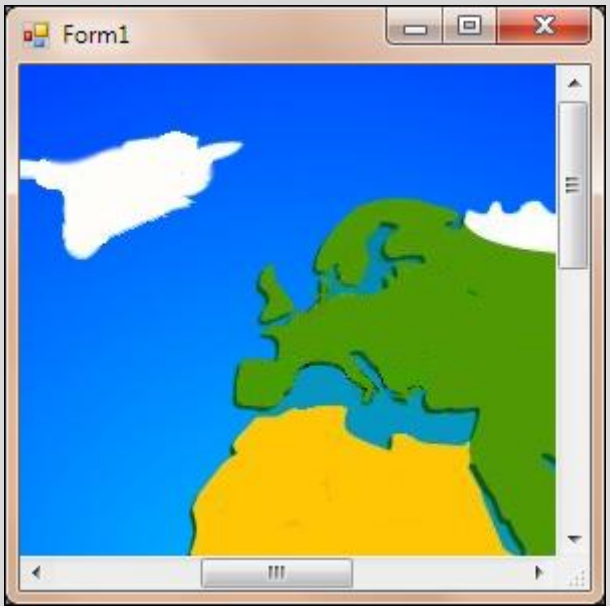


Facciamo *clic* sui tre puntini a destra e scegliamo l'immagine desiderata, importandola dalle risorse locali del computer:





Facciamo *clic* sul pulsante OK. L'immagine seguente mostra il programma in esecuzione, con le barre di scorrimento per visualizzare l'immagine del planisfero:



41: Il controllo ProgressBar.

Il controllo **ProgressBar** (barra di progressione) mostra graficamente, in una striscia colorata, il tempo che l'utente del programma ha a disposizione per effettuare una determinata operazione.
Man mano che il tempo trascorre, la barra si colora da sinistra a destra, sino a risultare tutta colorata allo scadere del tempo disponibile.
La proprietà principale di questo controllo è la proprietà **Value** (quantità) che determina dove si trova il punto tra la parte della barra già colorata (cioè il tempo trascorso) e la parte restante.
La proprietà **Value** parte da un livello minimo (proprietà **Minimum**) definito dal programmatore e aumenta via via con il passare del tempo sino ad arrivare al livello massimo (proprietà **Maximum**) stabilito dal programmatore. Quando la proprietà **Value** è uguale al valore della proprietà **Minimum**, la ProgressBar non è colorata. Viceversa, quando la proprietà **Value** è uguale al valore della proprietà **Maximum** la ProgressBar appare pienamente colorata: questo significa che il tempo è scaduto.
VB imposta automaticamente le proprietà Minimum e Maximum rispettivamente pari a 0 e a 100; mantenendo questi valori, la proprietà Value della ProgressBar indica anche la percentuale di tempo trascorso (esempio: se la barra è arrivata al valore 45, questo significa che è trascorso il 45% del tempo disponibile).
Vi sono tre modalità di visualizzazione della ProgressBar:

- **Blocks** (il riempimento colorato avanza a scatti, in questo caso la proprietà **Step** determina la grandezza dell'avanzamento a ogni scatto);
- **Continuous** (il riempimento colorato avanza in modo continuo);
- **Marquee** (il colore interno alla barra rimane fluttuante: l'utente non ha modo di vedere quanto tempo è trascorso né quanto tempo rimane.

Un controllo ProgressBar può anche essere utilizzato per misurare la progressione di altre quantità, non solo il trascorrere del tempo.
Ad esempio, si può usare una ProgressBar per visualizzare il numero dei tentativi compiuti dall'utente in una determinata situazione. Supponiamo che in un gioco l'utente abbia a disposizione al massimo 10 tentativi: in questo caso, il numero dei tentativi effettuati può essere visualizzato da una ProgressBar, impostata con i valori minimo e massimo da 0 a 10; a ogni tentativo questa ProgressBar aumenta il suo valore di una unità.
L'uso principale della **ProgressBar** è tuttavia la rappresentazione grafica del trascorrere del tempo, per cui questo controllo si trova abitualmente connesso a un componente **Timer**, come vedremo più avanti.

42: I controlli RichTextBox e TextBox.

I controlli **RichTextBox** (casella di testo potenziata) e **TextBox** (casella di testo) inseriscono nel form dei riquadri in cui l'utente del programma può leggere o scrivere dei testi più o meno lunghi (anche una sola parola, una frase, un testo, una storia di più pagine…).
Il **RichTextBox** è un contenitore più ricco di funzioni perché consente di variare la formattazione di parti del testo (in un RichtTextBox è possibile colorare, evidenziare, sottolineare il testo, scrivere in corsivo, cambiare il formato di singole parole o frasi).
Il **TextBox** invece è un contenitore più semplice, con impostazioni di formattazione ridotte, che vengono applicate in modo uniforme a tutto il testo contenuto nel controllo. A differenza del RichtTextBox, il TextBox si dispone automaticamente per ricevere o mostrare un testo su una sola riga. Qualora servano più righe, è necessario impostarne la proprietà **Multiline** = **True**.
In entrambi i controlli il testo da visualizzare è inserito dal programmatore impostando la proprietà **Text**.
Il RichtTextBox visualizza automaticamente una barra verticale per fare scorrere il testo, se questo eccede le dimensioni del controllo.
Nella figura seguente vediamo due caselle di testo: la prima, in alto, è un controllo RichTextBox con tre pulsanti per consentire all'utente di formattare parti del testo (Grassetto, Colore rosso, Evidenziazione); la seconda casella di testo, in basso, è un controllo TextBox normale. Entrambi i controlli mostrano la barra per lo scorrimento del testo in senso verticale.

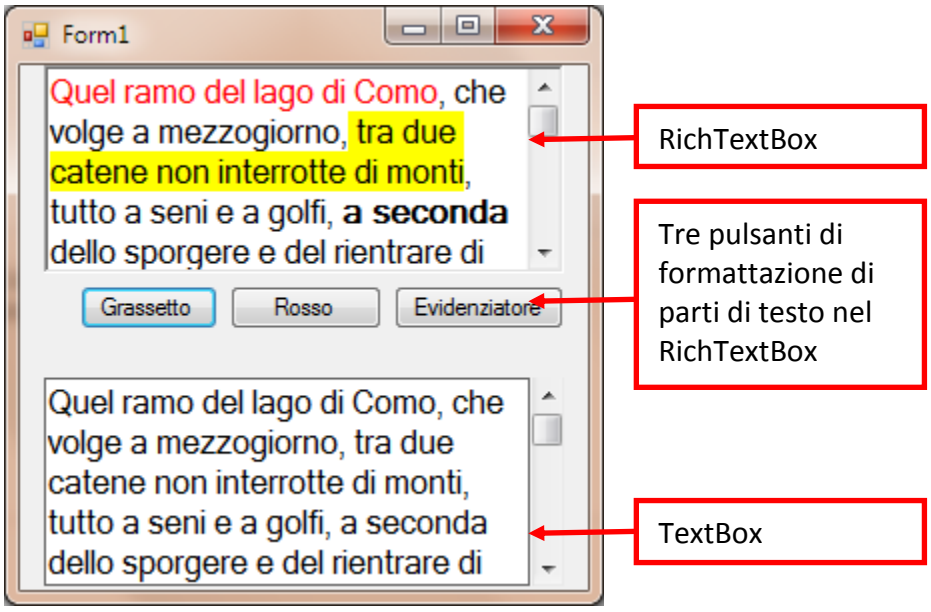


Figura 93: I controlli RichtTextBox e TextBox.

Il testo scritto dal programmatore nella proprietà **Text** compare all'avvio del programma nella casella di testo evidenziato su uno sfondo blu, come se fosse stato

selezionato con il mouse. Quando l'utente inizia a scrivere, con la pressione di qualsiasi tasto ottiene l'effetto di cancellare il testo preesistente:

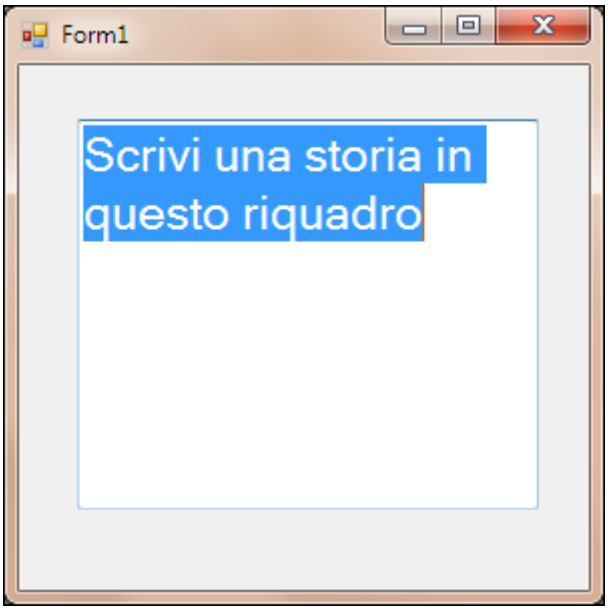


Figura 94: Un controllo TextBox con il testo di default evidenziato in blu.

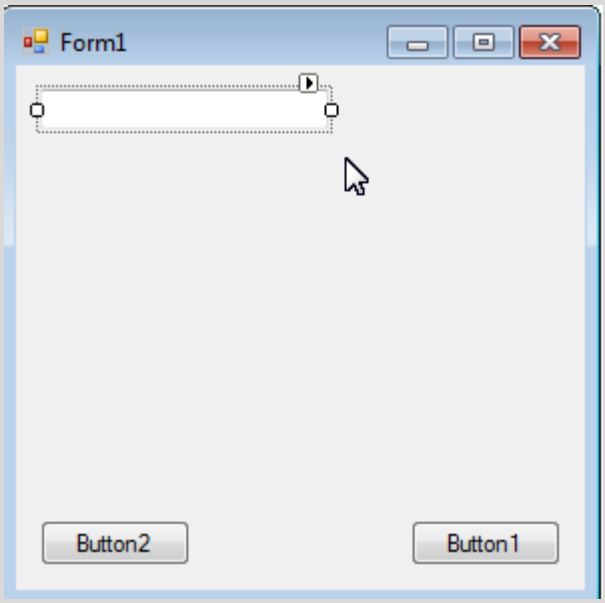
Se si desidera che l'utente non possa modificare il testo che compare in una casella di testo è necessario impostare la sua proprietà **ReadOnly** (solo lettura) = **True**.
Con questa impostazione l'utente non potrà fare aggiunte o modifiche al testo che compare nel RichTextBox o nel TextBox: l'utente potrà solo leggerlo, evidenziarlo con il trascinamento del mouse, copiarlo ed eventualmente esportarlo verso un programma di elaborazione dei testi.
Una casella di testo con la proprietà **ReadOnly = False** ha lo sfondo di colore bianco, altrimenti lo sfondo è di colore grigio.

Il testo da visualizzare in un RichtTextBox o in un TextBox può essere assegnato a una variabile di tipo **String** (stringa di testo); questa variabile può subire tutte le elaborazioni che il programmatore desidera e il risultato finale può essere visualizzato nella casella di testo.
Ne vedremo un esempio nel prossimo esercizio.

Esercizio 13: Elaborazione di una variabile di tipo String da visualizzare in un TextBox.

In questo esercizio vedremo l'impostazione di alcune proprietà di un TextBox dal codice del programma, e l'elaborazione di un testo come una variabile da visualizzare in questo TextBox.

Apriamo un nuovo progetto e inseriamo nel form un controllo **TextBox**, in alto a sinistra, e due pulsanti **Button** come in questa immagine:



Non impostiamo alcuna proprietà del TextBox, perché lo faremo dal codice del programma.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea una variabile di nome Testo, valida per tutte le procedure di
questo codice:
    Dim Testo As String
```

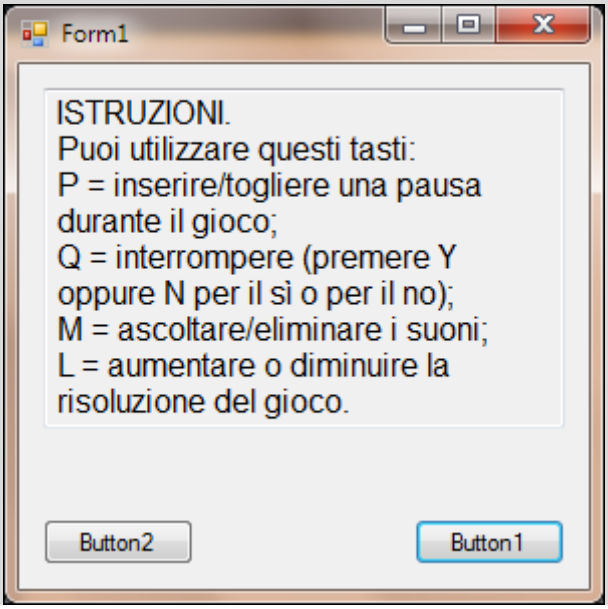
```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' Definisce le proprietà del controllo TextBox1:
        TextBox1.Multiline = True
        TextBox1.Size = New Size(260, 170)
        TextBox1.ReadOnly = True
        TextBox1.Font = New Font("Arial", 12)

    End Sub
```

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' Elabora il contenuto della variabile Testo e al termine lo
visualizza nel TextBox1:

        Testo = "ISTRUZIONI." & vbCrLf
        Testo += "Puoi utilizzare questi tasti:" & vbCrLf
        Testo += "P = inserire/togliere una pausa durante il gioco;" & vbCrLf
        Testo += "Q = interrompere (premere Y oppure N per il sì o per il
no);" & vbCrLf
```

```
        Testo += "M = ascoltare/eliminare i suoni;" & vbCrLf
        Testo += "L = aumentare o diminuire la risoluzione del gioco."

        TextBox1.Text = Testo

    End Sub
```

```
    Private Sub Button2_Click_1(sender As System.Object, e As
System.EventArgs) Handles Button2.Click

        ' Elabora il contenuto della variabile Testo e al termine lo
visualizza nel TextBox1:
        Testo = "Per leggere le istruzioni, premi il pulsante Button1."
        TextBox1.Text = Testo

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Per lo studio delle proprietà e delle azioni del controllo TextBox rinviamo la lettrice o il lettore al paragrafo 171: Stampare un testo. 745.

MaskedTextBox

Il controllo **MaskedTextBox** (casella di testo con schema obbligato) è un contenitore di testo che presenta all'utente uno **schema vincolante**, da usare e da rispettare per l'immissione di dati.

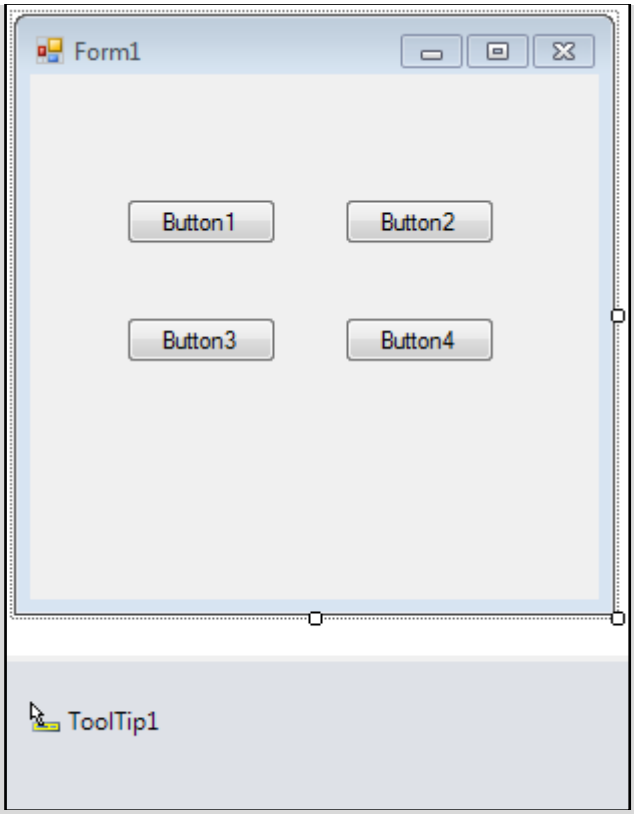
Lo schema vincolante indica all'utente quali caratteri sono ammessi nel testo, e in quali posizioni; l'utente, quando scrive in questo contenitore, deve attenersi a tali indicazioni, altrimenti il programma non accetta i dati immessi.
Esempio: in un programma in cui l'utente deve scrivere una data, può essere inserito un MaskedTextBox che accetti la data solo se questa è scritta nel formato "**gg/mm/aaaa**" (due caselle per indicare il giorno, due caselle per il mese, quattro caselle per l'anno).
In questo caso, il programma accetta una data scritta come 25/12/1995 e rifiuta una data scritta come 25/12/95.
Un esempio dell'uso del controllo MaskedTextBox è visibile nel paragrafo dedicato al componente **ErrorProvider**, a pag. 268.

43: Il componente ToolTip.

Il componente **ToolTip** (= strumento di suggerimento) consente di gestire le scritte di aiuto che compaiono temporaneamente quando il mouse passa sopra un controllo.
Queste scritte, che rimangono visibili pochi secondi, contengono di solito spiegazioni o istruzioni relative al controllo sul quale il mouse sta passando. Ne vedremo l'uso nel prossimo esercizio.

Esercizio 14: Il componente ToolTip.

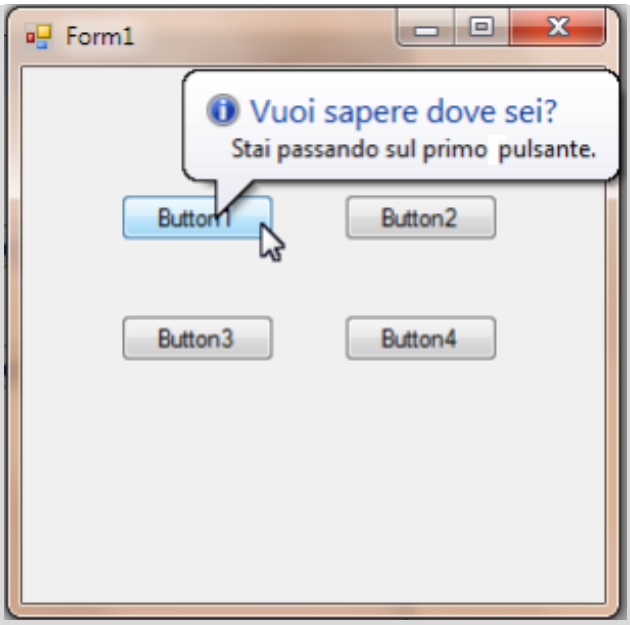
In questo esercizio vedremo come inserire il componente ToolTip in un progetto e come impostarne la grafica e i contenuti.
Apriamo un nuovo progetto e inseriamo nel Form1 quattro pulsanti Button e il componente ToolTip, come in questa immagine:



Notiamo che il componente ToolTip si colloca all'esterno del Form1 perché, anche se è inserito nel gruppo dei controlli comuni, esso in realtà è un oggetto che rimarrà non visibile all'utente del programma.
Dopo avere inserito nel progetto il componente ToolTip, facciamo un *clic* sul primo pulsante Button e andiamo a vedere le sue proprietà.
Notiamo che in questo elenco è presente ora una proprietà nuova: la proprietà **ToolTip su Tooltip1**. Nello spazio vuoto di queta proprietà, scriviamo il breve testo di aiuto che vogliamo fare apparire quando il mouse passerà sopra il Button1.
Scriviamo, ad esempio: *Stai passando sul primo pulsante*.
Procediamo allo stesso modo con gli altri tre pulsanti.
Adesso facciamo un *clic* sul componente ToolTip, fuori dal form e andiamo a impostare alcune sue proprietà:

- **AutomaticDelay = 100** (è il ritardo, in millisecondi, dopo il qual viene visualizzato il ToolTip sui controlli);
- **IsBalloon = True** (le scritte compariranno all'interno di un fumetto);
- **ToolTipIcon = Info** (assieme al ToolTip verrà visualizzata una icona con la lettera "i" di informazioni);
- **ToolTipTitle = *Vuoi sapere dove sei?*** (è il titolo che comparirà nei fumetti sopra i ToolTip).

Ora mandiamo in esecuzione il progetto e osserviamo la visualizzazione dei ToolTip sui quattro pulsanti.



L'impostazione dei quattro ToolTip sui quattro pulsanti può essere fatta anche dalla Finestra del Codice, senza dovere andare a modificare le proprietà di tutti i controlli, scrivendo queste istruzioni nella Finestra del Codice:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

        ' al caricamento in memoria del Form1, cioè all'avvio del programma,
        ' si impostano le proprietà dei ToolTip:

        ToolTip1.ToolTipIcon = ToolTipIcon.Warning
        ToolTip1.ToolTipTitle = "Che pulsante è questo?"
        ToolTip1.SetToolTip(Button1, "Questo è il primo pulsante")
        ToolTip1.SetToolTip(Button2, "Questo è il secondo pulsante")
        ToolTip1.SetToolTip(Button3, "Questo è il terzo pulsante")
        ToolTip1.SetToolTip(Button4, "Questo è il quarto pulsante")

    End Sub

End Class
```

Quando si registra l'evento Me.Load, cioè quando quando il Form1 viene caricato nella memoria del computer, all'avvio del programma, le istruzioni scritte assegnano al componente ToolTip, nell'ordine:

- l'icona,
- il titolo e
- la frase per ognuno dei quattro pulsanti presenti nel form.

44: Altri controlli comuni.

NotifyIcon

NotifyIcon è un componente, non visibile dall'utente del programma, che serve a visualizzare una icona nella barra degli strumenti di Windows (la barra che compare nella parte inferiore del monitor), quando il programma è in esecuzione.

TreeView

Il controllo TreeView (visualizzazione ad albero) visualizza una serie di dati disponendoli gerarchicamente in una scala grafica, come in questa immagine:

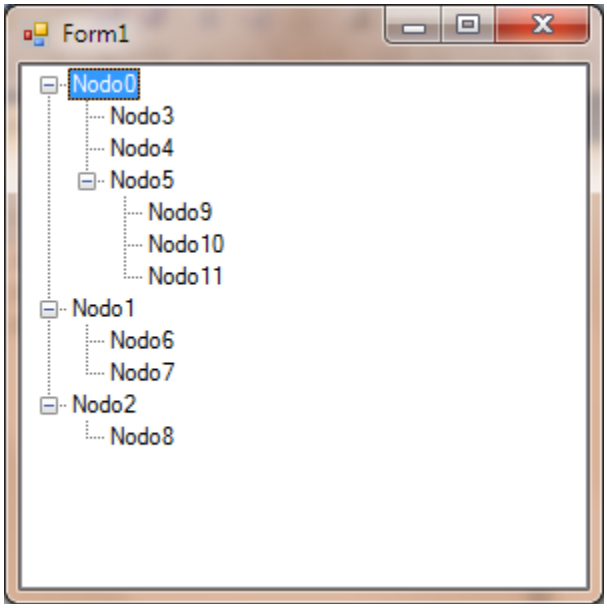


Figura 95: Il controllo TreeView in esecuzione.

WebBrowser

Il controllo WebBrowser visualizza il contenuto di siti internet.
Il controllo si collega agli indirizzi scritti dall'utente e visualizza siti e pagine con le stesse modalità di navigazione di un normale *browser*.
Può essere utilizzato anche per leggere le pagine in formato .htm o .html di una guida di un programma, o di un manuale allegato ad un programma.

45: Tutti i controlli disponibili per i form di Windows.

Vi sono alcuni controlli che non compaiono nel gruppo dei Controlli comuni esaminati in questo capitolo, né in alcun altro gruppo di controlli, ma compaiono solo nell'elenco **Tutti i (controlli per i) Windows Form**.
Ne analizzeremo le caratteristiche in questo paragrafo.
Per visualizzarli nella Casella degli Strumenti, è necessario cliccare l'opzione di visualizzazione **Tutti i Windows Form**:

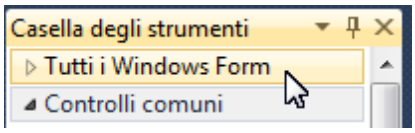


Figura 96: Visualizzazione di tutti i controlli disponibili per i form di Windows.

HScrollBar

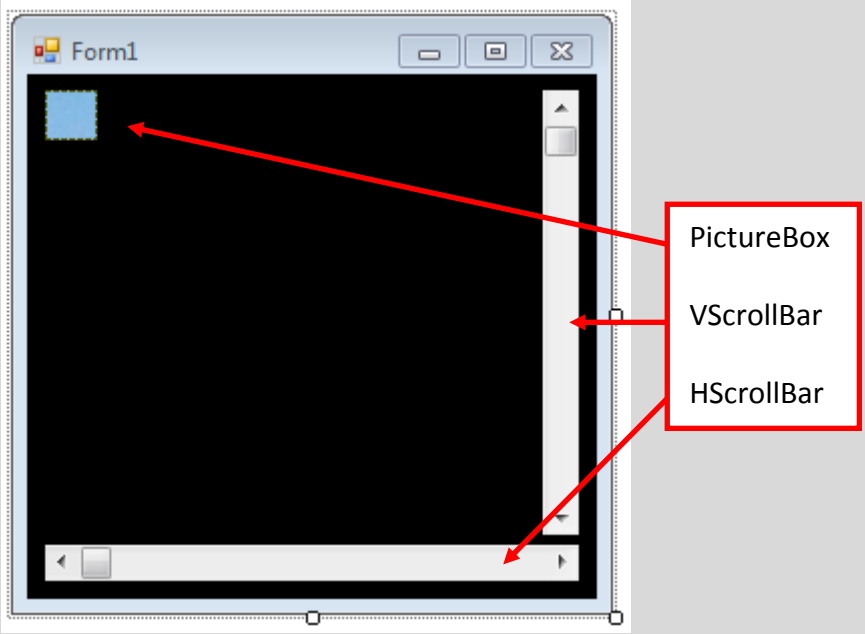
VScrollBar

I controlli **HScrollBar** (barra di scorrimento orizzontale) e **VScrollBar** (barra di scorrimento verticale) sono di uso molto comune; consentono all'utente del programma di scorrere un documento o un oggetto in direzione orizzontale o verticale, cliccando i pulsanti con le freccine che si trovano alle estremità dei controlli oppure trascinando la manopola scorrevole che si trova lungo le barre stesse, tra le due estremità.
Sono utilizzati generalmente come barre di scorrimento per contenitori di testi o di immagini; si trovano già inseriti nei controlli come il **ListBox** o il **TextBox**, e nei controlli **Panel**, dove le barre di scorrimento vengono visualizzate automaticamente quando se ne presenta la necessità.
Inserite in un programma come oggetti autonomi, queste due barre possono anche essere utilizzate come strumenti per fare scegliere all'utente un valore numerico, in una scala numerica tra un valore minimo e un valore massimo definiti dal programmatore.
Ne vedremo un esempio nel prossimo esercizio.

Esercizio 15: I controlli HScrollBar e VScrollBar.

Apriamo un nuovo progetto.

Facciamo un *clic* sul Form1 e, nella pagina delle proprietà, Impostiamo la sua proprietà **BackColor = Black**
Inseriamo nel form un controllo PictureBox, un controllo HScrollBar e un controllo VScrollBar, come in questa immagine:



Impostiamo alcune proprietà di questi tre oggetti:

PictureBox	Size	26;26
	SizeMode	Normal
	Image	Documenti / A scuola con VB 2010 / Immagini / Pinguino.jpg
VScroll1	Minimum	10
	Maximum	230
HScroll1	Minimum	10
	Maximum	230

Abbiamo gà visto come visualizzare un'immagine in un controllo PictureBox.
In questo caso l'immagine da inserire è l'immagine Pinguino.jpg, che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini**.
Il controllo PictureBox è ridotto a un quadrato di 26 pixel per lato, per cui l'immagine in esso contenuta (il pinguino) non è visibile.
L'obiettivo del programma è questo: usando le due barre di scorrimento, l'utente potrà allargare il riquadro dell'immagine in orizzontale e in verticale.
Le proprietà **Minimum** e **Maximum** delle due barre di scorrimento vanno da 10 a 230.

Il formato dell'immagine del pinguino è di 230 x 230 pixel, dunque, quando le due barre di scorrimento saranno sul loro valore massimo, l'immagine potrà essere visualizzata per intero.
Nella Finestra del Codice dobbiamo scrivere le istruzioni necessarie affinché il programma modifichi le dimensioni del controllo PictureBox seguendo il variare dei valori delle due barre di scorrimento.
Facciamo un doppio *clic* sulla barra di scorrimento orizzontale ed accediamo alla Finestra del Codice.
Qui troviamo una procedura già impostata per gestire l'evento dello scorrimento (**Scroll**) della barra **HScroll1**. Completiamo la procedura indicando che allo scorrimento della barra orizzontale, la larghezza del controllo PictureBox deve assumere il valore indicato dalla barra:

```
    Private Sub HScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ScrollEventArgs) Handles HScrollBar1.Scroll

        PictureBox1.Width = HScrollBar1.Value

    End Sub
```

Ripetiamo la cosa per la barra di scorrimento verticale; in questo caso è necessario indicare che allo scorrimento della barra verticale, l'altezza del controllo PictureBox deve assumere il valore indicato dalla barra:

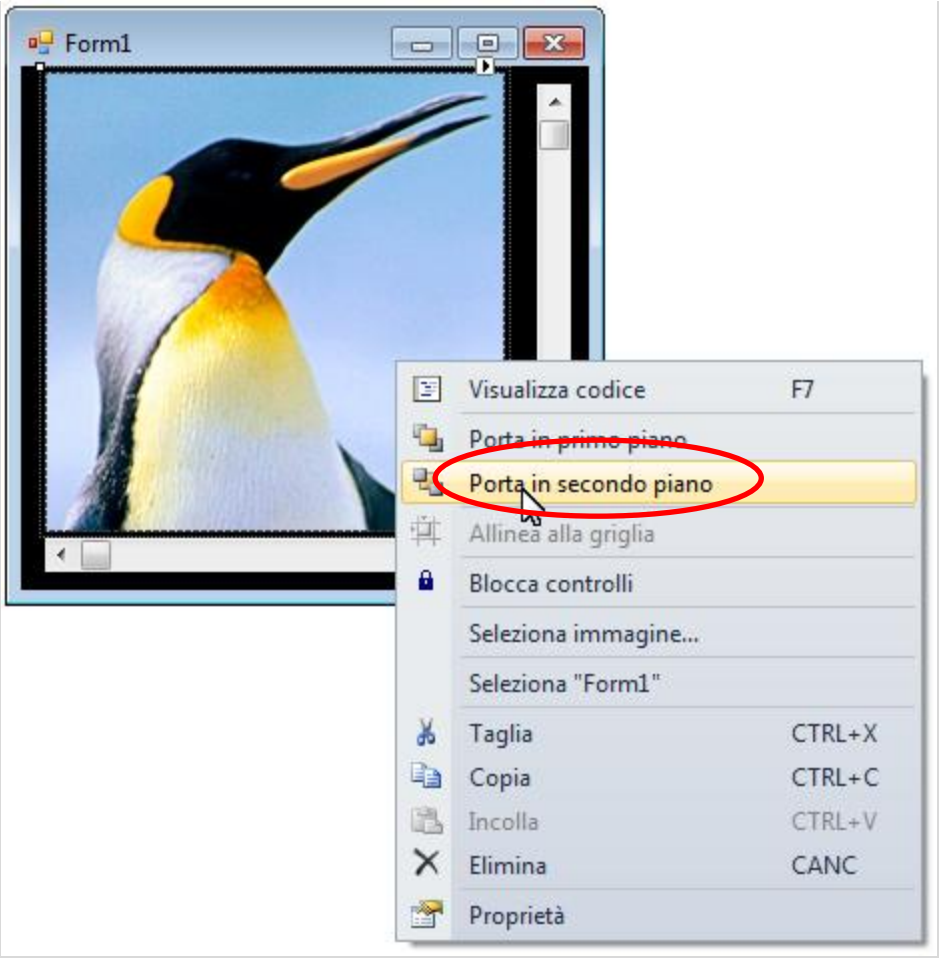
```
    Private Sub vScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ScrollEventArgs) Handles VScrollBar1.Scroll

        PictureBox1.Height = VScrollBar1.Value

    End Sub
```

Il programma è ora predisposto per gestire lo scorrimento dei pulsanti delle due barre HScrollBar1.Scroll e VScrollBar1.Scroll. Al verificarsi di uno di essi, il programma imposta la larghezza e l'altezza del controllo PictureBox sui valori correnti delle barre di scorrimento.
Nell'immagine seguente vediamo il programma in esecuzione.

Fermiamo l'esecuzione del programma e torniamo alla fase di progettazione.
Apportiamo al programma queste varianti:

- lo scorrimento della barra orizzontale farà scorrere il controllo PictureBox verso destra;
- lo scorrimento della barra verticale farà scorrere il controllo PictureBox verso il basso.

Facciamo un *clic* sul controllo PictureBox1 e impostiamo la sua proprietà **SizeMode = AutoSize.**
Facciamo un *clic* con il tasto **destro** del mouse sul controllo PictureBox1 e nel menu che si apre facciamo un *clic* sulla opzione **Porta in secondo piano**:



Ora torniamo nella Finestra del Codice e scriviamo le nuove istruzioni all'interno delle due procedure, per ottenere lo scorrimento del controllo PictureBox verso destra e verso il basso:

```
Private Sub HScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ScrollEventArgs) Handles HScrollBar1.Scroll

        PictureBox1.Left = -HScrollBar1.Value

    End Sub
```

```
    Private Sub VScrollBar1_Scroll(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ScrollEventArgs) Handles VScrollBar1.Scroll

        PictureBox1.Top = -VScrollBar1.Value

    End Sub
```

In fase di esecuzione, notiamo che il controllo PictureBox nei suoi spostamenti passa dietro le due barre di scorrimento; ciò è dovuto alla scelta dell'opzione **Porta in secondo piano**, fatta in precedenza.

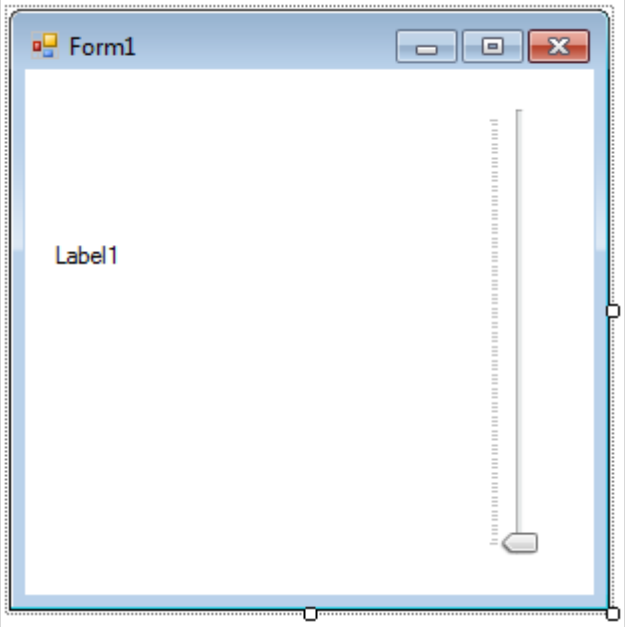
Il segno "-" posto davanti ai valori delle due barre di scorrimento fa sì che l'immagine scorra nella stessa direzione delle barre, altrimenti si ha un movimento in direzione inversa.

TrackBar

Il controllo **TrackBar** (barra su binario) consente all'utente di trascinare un puntatore lungo un *binario* obbligato, per scorrere i valori numerici presenti lungo questo binario e per scegliere il valore sul quale la barra si arresta.
E' un controllo molto simile alle barre di scorrimento, con una diversa veste grafica: questo non ha le frecce che compaiono ai due estremi delle barre e ha una diversa tacca di scorrimento.
Ciò che lo caratterizza rispetto alle barre di scorrimento è la possibilità di essere usato come strumento per l'indicazione o la scelta di dati numerici da parte dell'utente del programma.
La proprietà **Orientation** imposta la visualizzazione del controllo in **orizzontale** o in **verticale**.
La proprietà **TickFrequency** determina il numero delle tacche che verranno visualizzate lungo il *binario* del controllo.
La proprietà **TickStyle** determina l'orientamento delle tacche e della manopola di scorrimento: queste possono essere rivolte vero l'alto o verso il basso se il binario è in posizione orizzontale, verso destra o verso sinistra se il binario è in posizione verticale.
Nel prossimo esercizio vedremo un esempio di uso di questo controllo per modificare l'altezza dei caratteri in un controllo Label, secondo la scelta dell'utente.

Esercizio 16: Il controllo TrackBar e la proprietà FontSize in un controllo Label.

In questo esercizio utilizzeremo il controllo TrackBar per modificare l'altezza dei caratteri del testo di un controllo Label (etichetta): durante l'esecuzione del programma, l'utente potrà scorrere la manopola del TrackBar da 10 a 100, modificando in corrispondenza l'altezza del testo di una label.
Apriamo un nuovo progetto.
Inseriamo nel form un controllo Label e un controllo TrackBar, disponendoli come nell'immagine seguente:



Per disporre il **TrackBar** in verticale, con la manopola rivolta a sinistra, è necessario impostare queste sue proprietà:

- **Orientation = Vertical**
- **TickStyle = TopLeft**

Impostiamo queste altre proprietà del controllo TrackBar:

- **Maximum 100**
- **Minimum 10**

Con queste impostazioni dei valori minimo e massimo, l'utente potrà fare scorrere la TrackBar entro i numeri da 10 a 100; il programma cambierà l'altezza dei caratteri del testo visualizzato nella Label1 in corrispondenza dell'azione dell'utente sulla TrackBar.
Facciamo un doppio *clic* sul TrackBar1 e accediamo alla Finestra del Codice.
Vi troviamo già impostata la procedura per gestire l'evento dello scorrimento della tacca del TrackBar1:

```
    Private Sub TrackBar1_Scroll_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TrackBar1.Scroll

    End Sub
```

Nella riga centrale, vuota, scriviamo questo comando:

```
    Private Sub TrackBar1_Scroll_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TrackBar1.Scroll

        Label1.Font = New Font("Verdana", TrackBar1.Value)

    End Sub
```

Traduzione della procedura:

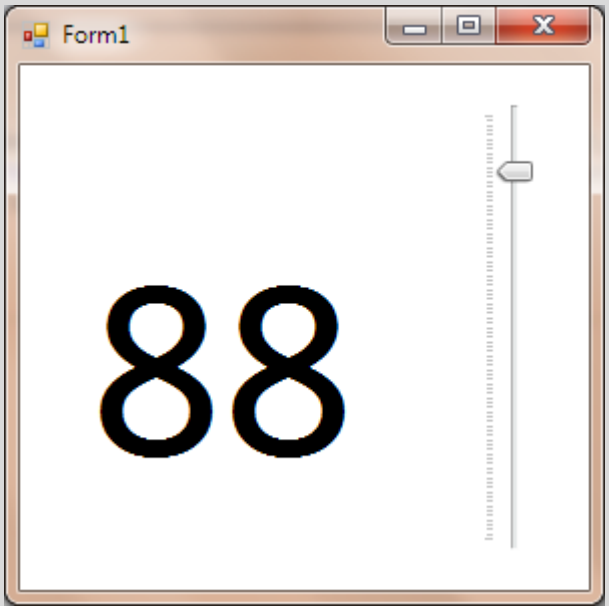
- quando si verifica l'evento dello scorrimento della tacca della TrackBar1 (TrackBar1.Scroll)

- assegna al controllo Label1 un nuovo font che, mantenendo tutte le caratteristiche del font già presente nella Label1 (Label1.Font.Style), assuma come grandezza il valore indicato dalla TrackBar1.

Possiamo aggiungere un altro comando, per visualizzare nella Label1 il valore numerico indicato dalla TrackBar1:

```
Public Class Form1

    Private Sub TrackBar1_Scroll_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TrackBar1.Scroll

        Label1.Font = New Font("Verdana", TrackBar1.Value)
        Label1.Text = TrackBar1.Value

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



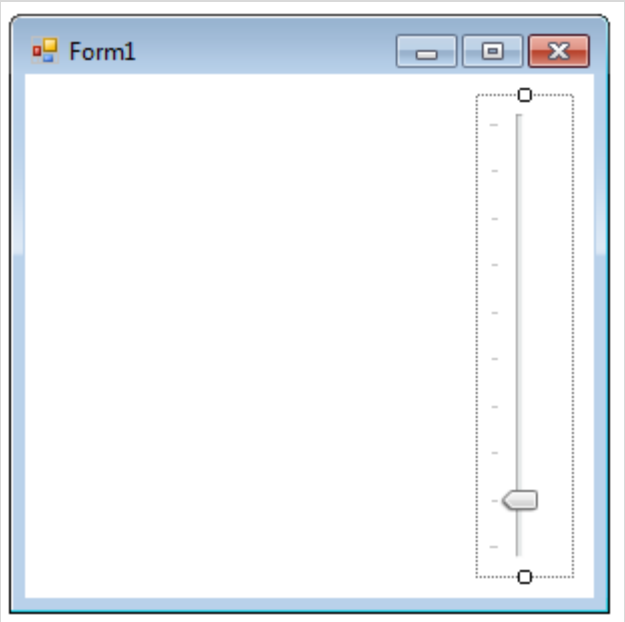
Nel prossimo esercizio vedremo un altro uso del controllo TrackBar.
In questo esercizio il valore numerico espresso dalla Trackbar sarà impostato tra 0 e 9 e verrà utilizzato per mostrare una serie di immagini, numerate da 0 a 9, prelevate da un componente **ImageList**[37], contenitore di immagini.

[37] Il funzionamento del questo componente ImageList è spiegato più avanti nel manuale, a pag. 276.

Esercizio 17: Il controllo TrackBar e il componente ImageList.

In questo esercizio utilizzeremo il controllo TrackBar per realizzare un *misuratore dell'umore*: l'utente del programma, facendo scorrere la manopola del TrackBar lungo una scala di valori da 0 a 9, potrà visualizzare l'immagine corrispondente al suo umore del momento.
Le 10 immagini (da 0 a 9) corrispondenti ai diversi stati d'umore verranno immagazzinate in un componente ImageList.
Apriamo un nuovo progetto.
Inseriamo nel Form1 un controllo TrackBar. Per disporlo come appare nella figura seguente è necessario impostare queste sue proprietà:

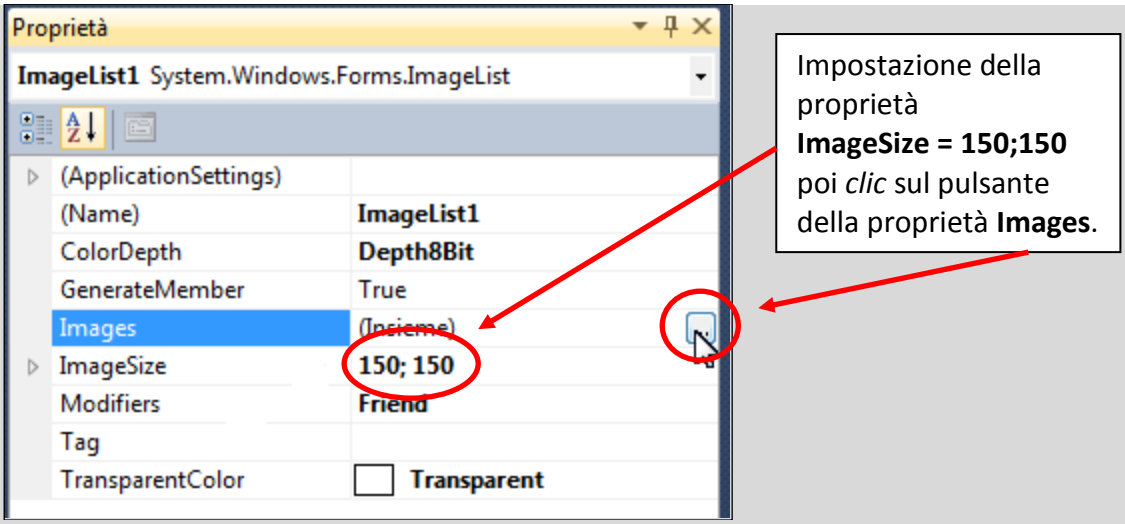
- **Orientation = Vertical**
- **TickStyle = TopLeft**



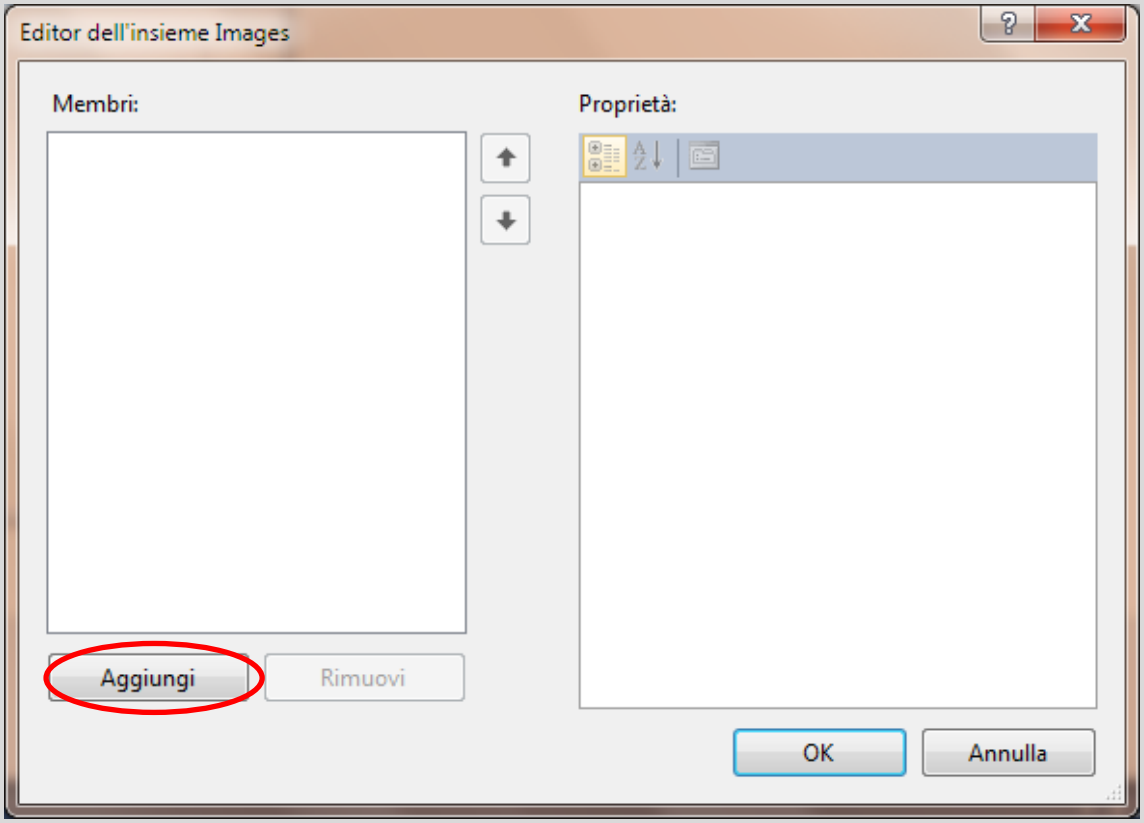
Siccome avremo a disposizione 10 immagini, numerate da 0 a 9, impostiamo di conseguenza le proprietà Minimum e Maximum del TrackBar:

- **Maximum = 9**
- **Minimum = 0**

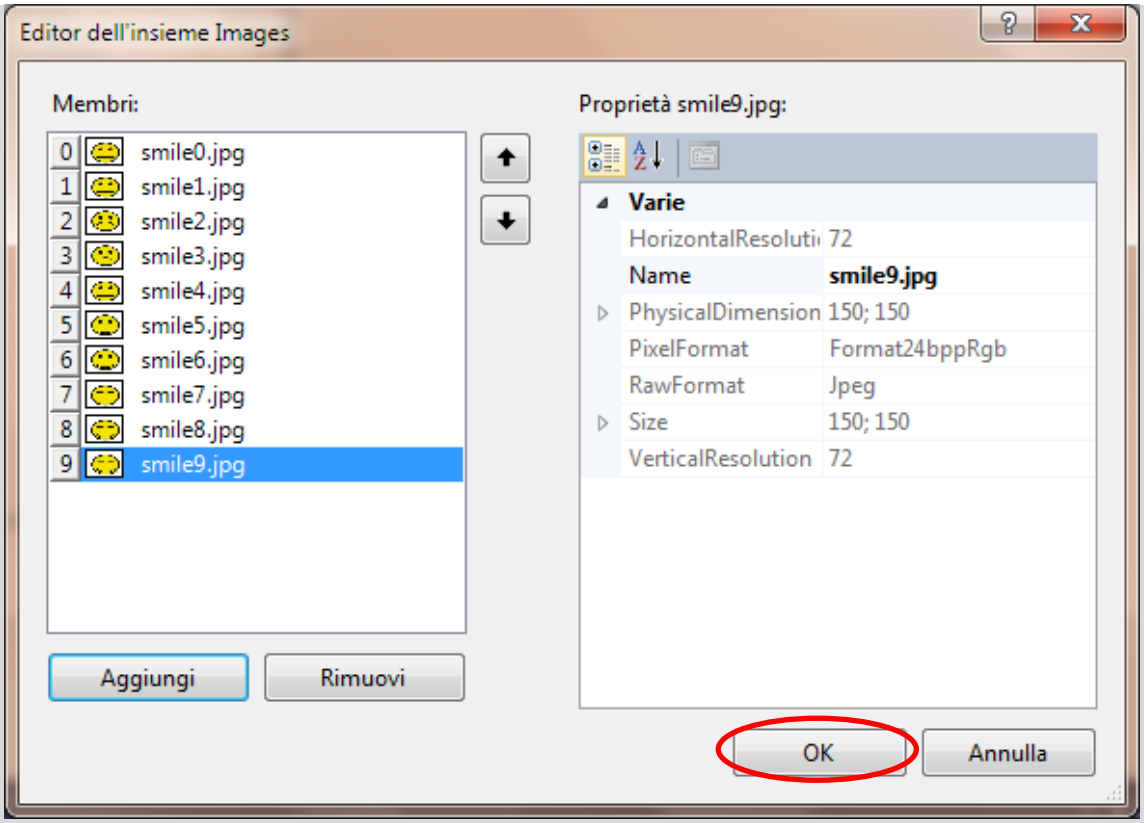
Ora inseriamo nel progetto un componente **ImageList**.
Trattandosi di un componente (oggetto che rimarrà invisibile all'utente del programma), l'ImageList va a collocarsi fuori dell'area del Form1, nella parte inferiore della Finestra di Progettazione.
Facciamo un *clic* sul componente ImgageList, accediamo alla Finestra Proprietà e impostiamo la sua proprietà **ImageSize = 150;150** (sono queste le dimensioni delle immagini che vi inseriremo).
Facciamo poi un *clic* sul pulsante che si trova nella riga della proprietà **Images**:



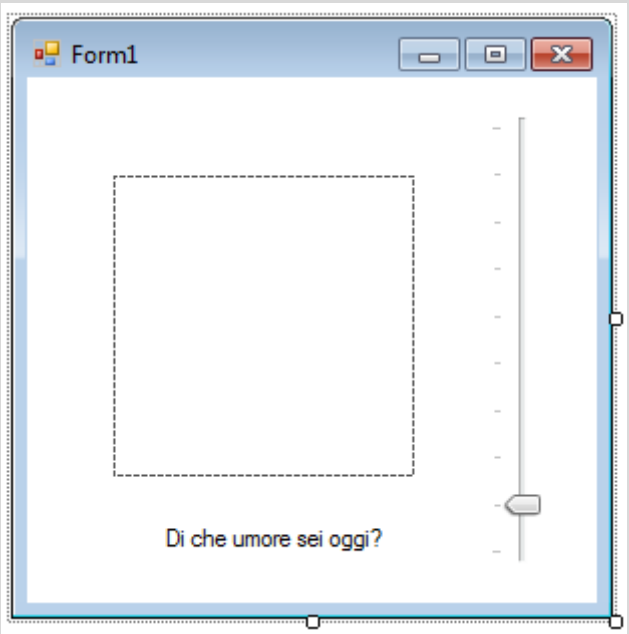
Facendo un *clic* su questo pulsante con tre puntini accediamo alla finestra dell'**Editor dell'insieme Images**; in questa finestra dobbiamo aggiungere le dieci immagini che faranno parte di questo misuratore dell'umore.



Facciamo un *clic* sul pulsante **Aggiungi** e andiamo a selezionare le dieci immagini numerate con il nome **smile** che si trovano nella cartella **Documenti / A Scuola con VB / Immagini / Smile**.



Notiamo che alle immagini inserite viene assegnato, nella prima colonna a sinistra, un numero d'ordine che parte da 0 e arriva a 9.
Facciamo un *clic* sul pulsante OK.
Dopo avere caricato la lista delle immagini che compariranno nel programma, inseriamo nel form un controllo **PictureBox** e un controllo **Label** come in questa immagine:



Il controllo **PictureBox** ha la proprietà **Size = 150;150** (è lo stesso formato delle immagini inserite nell'ImageList).
Il controllo **Label** ha la proprietà **Text = Di che umore sei oggi?**

Con questo, il lavoro di creazione dell'interfaccia è finito.
Nella Finestra del Codice, dobbiamo scrivere una procedura affinché a ogni scorrimento del controllo TrackBar il programma registri la posizione della TrackBar, che va da da 0 a 9, e mostri nel controllo PictureBox l'immagine che si trova nel componente ImageList con il numero corrispondente.
Facciamo un doppio *clic* sul controllo TrackBar: accediamo così alla Finestra del Codice, dove troviamo già impostata la procedura per gestire l'evento del suo scorrimento:

```
Private Sub TrackBar1_Scroll_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TrackBar1.Scroll

    End Sub
```

Nella riga centrale, vuota, scriviamo il comando che ci interessa:

```
    Private Sub TrackBar1_Scroll_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TrackBar1.Scroll

        PictureBox1.Image = ImageList1.Images(TrackBar1.Value)

    End Sub
```

Traduzione della procedura:

- quando si verifica l'evento dello scorrimento della TrackBar1 (TrackBar1.Scroll)
- assegna al controllo PictureBox1 l'immagine che si trova nella lista ImageList, il cui numero corrisponde alla posizione del TrackBar (TrackBar1.Value).

L'immagine seguente mostra il programma in esecuzione.

DomainUpDown

Il controllo **DomainUpDown** visualizza - uno alla volta - gli item di una lista che l'utente può scorrere cliccando i pulsanti con le frecce "su" e "giù" sul lato destro del controllo. L'utente del programma ha la possibilità di selezionare con un *clic* del mouse l'item di volta in volta visualizzato.
Con la proprietà **InterceptArrowKeys = True**, il controllo riconosce anche la pressione dei tasti con le frecce "su" e "giù" sulla tastiera.
Con la proprietà **Sorted** = **True**, gli item vegono visualizzati secondo l'ordine alfabetico.
Se la proprietà **Wrap** = **True**, dopo avere visualizzato l'ultimo item dell'elenco il controllo passa automaticamente a visualizzare il primo e, viceversa, dal primo item dell'elenco passa automaticamente all'ultimo.
La figura seguente mostra un programma in esecuzione con un controllo DomainUpDown che visualizza i giorni della settimana. L'utente ha la possibilità di scorrere l'elenco dei giorni e di scegliere un giorno, cliccandolo con il mouse.

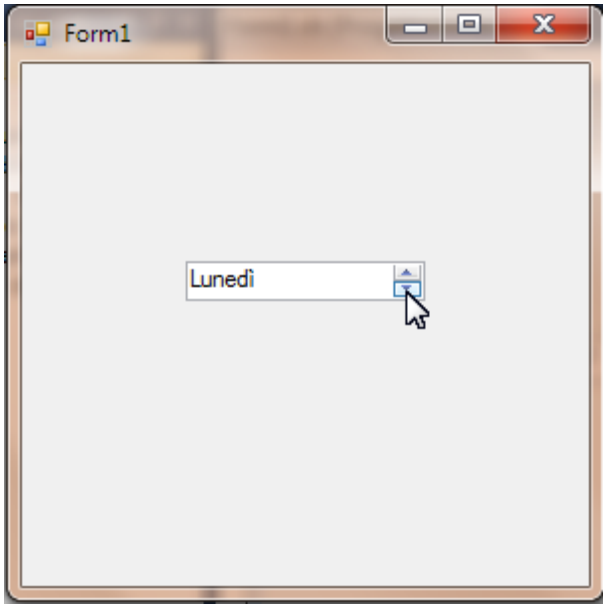


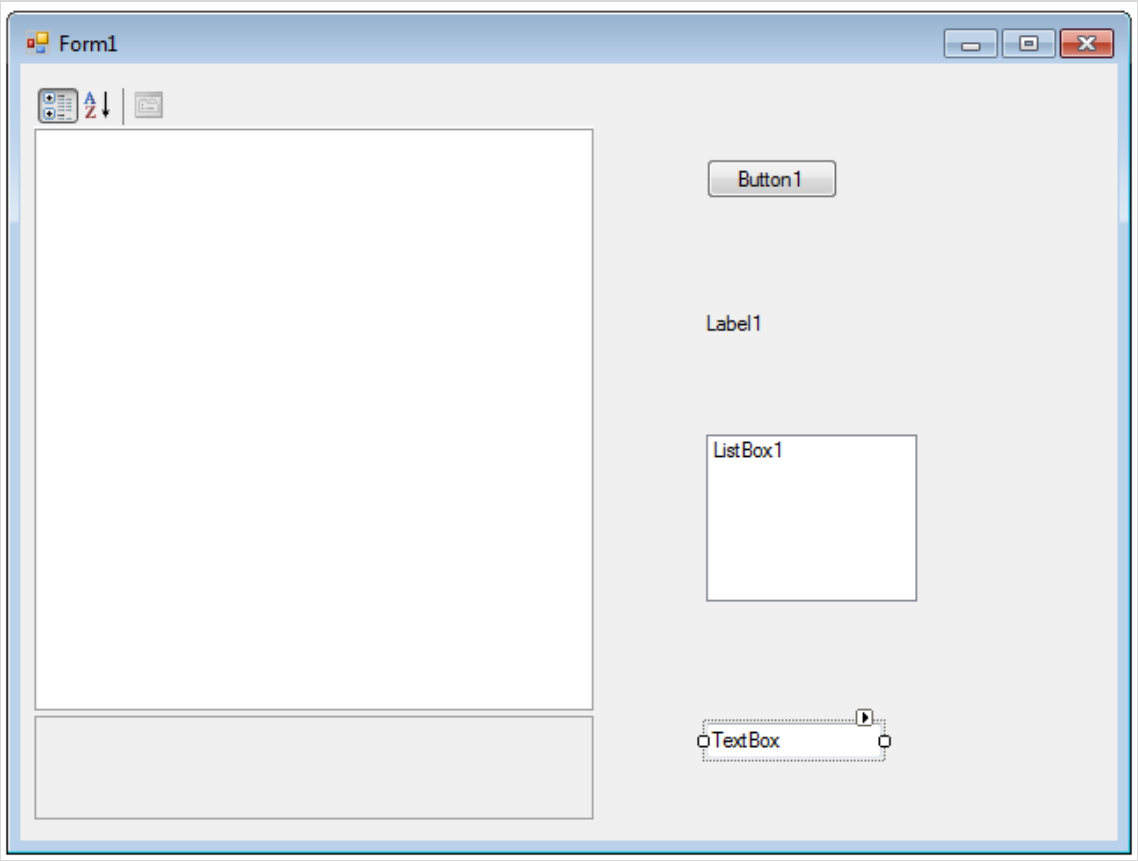
Figura 97: Il controllo DomainUpDown.

PropertyGrid

Il controllo **PropertyGrid** (griglia delle proprietà) visualizza la scheda delle proprietà di un oggetto presente nel form, esattamente come fa la Finestra Proprietà nell'ambiente di progettazione.
L'utente del programma può agire in questa scheda e modificare le proprietà degli oggetti anche quando il programma è in esecuzione.

Esercizio 18: Il controllo PropertyGrid.

In questo esercizio vedremo un esempio del funzionamento del controllo PropertyGrid.
Apriamo un nuovo progetto.
Facciamo un *clic* sul Form1 e, nella scheda delle sue proprietà, impostiamo la proprietà **Size = 640;480**.
Ora nella Casella degli Strumenti facciamo un *clic* sul gruppo di controlli **Tutti i Windows Form**.
Qui prendiamo un controllo **PropertyGrid** e lo inseriamo nel Form; inseriamo poi nel Form1 un controllo **Button1**, un controllo **Label1**, un controllo **ListBox1** e un controllo **TextBox1**, come in questa immagine:



Nell'area del controllo **PropertyGrid**, a sinistra del form, verranno visualizzate a uso dell'utente le proprietà dei quattro controlli presenti nel Form1, e le proprietà dello stesso Form1, esattamente come avviene per il programmatore nella Finestra Proprietà, nella fase di progettazione.
Nella Finestra del Codice scriviamo le istruzioni necessarie perché, a un *clic* su un controllo presente nel form, il programma colleghi questo controllo alla **PropertyGrid** e ne visualizzi le proprietà.

Partiamo dal pulsante Button1. Facciamo un doppio *clic* sul Button1 e accediamo alla Finestra del Codice, che si presenta già impostata per gestire l'evento di un *clic* sul Button1:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    End Sub
```

Nella riga centrale della procedura scriviamo il collegamento tra il Button1 e la PropertyGrid in questo modo:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        PropertyGrid1.SelectedObject = sender

    End Sub
```

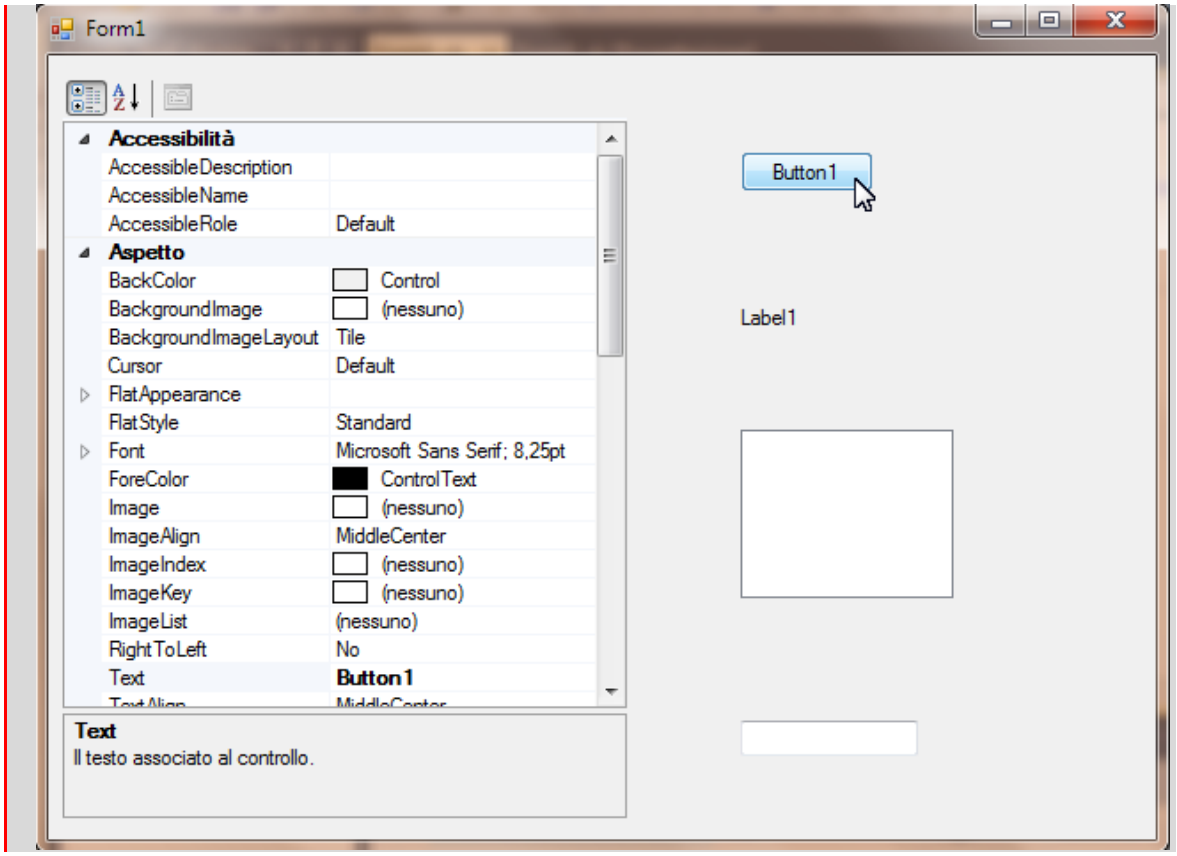
La riga aggiunta dice che l'oggetto selezionato per la PropertyGrid è l'oggetto mittente (sender) dell'evento Click, cioè l'oggetto che ha avvertito il *clic* del mouse: nel nostro caso, il pulsante Button1.
Avremmo potuto scrivere

```
        PropertyGrid1.SelectedObject = Button1
```

e il risultato sarebbe stato lo stesso. L'uso del parametro sender, però, ci consente di utilizzare la stessa riga centrale della procedura per gestire il *clic* del mouse su tutti gli oggetti presenti nel Form1:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click, Label1.Click, ListBox1.Click,
TextBox1.Click, Me.Click

        PropertyGrid1.SelectedObject = sender

    End Sub
```

La procedura, ora completa, istruisce il programma affinché a un *clic* del mouse su uno degli oggetti Button1, Label1, Listbox1, TextBox1 e Form1, esso colleghi l'oggetto mittente dell'evento alla PropertyGrid e ne mostri le proprietà.
Mandiamo in esecuzione il programma e visualizziamo l'elenco delle proprietà dei diversi oggetti facendo un *clic* con il mouse su ognuno di essi.
Notiamo che l'elenco di queste proprietà è un elenco **attivo** e che l'utente può modificare mediante esse le impostazioni dell'oggetto cliccato.


Form1
Accessibilità
AccessibleDescription
AccessibleName
AccessibleRole Default
Aspetto
BackColor Control
BackgroundImage (nessuno)
BackgroundImageLayout Tile
Cursor Default
FlatAppearance
FlatStyle Standard
Font Microsoft Sans Serif; 8,25pt
ForeColor ControlText
Image (nessuno)
ImageAlign MiddleCenter
ImageIndex (nessuno)
ImageKey (nessuno)
ImageList (nessuno)
RightToLeft No
Text Button1
Text
Il testo associato al controllo.
Button1
Label1

Capitolo 7: I CONTROLLI CONTENITORI DI ALTRI CONTROLLI.

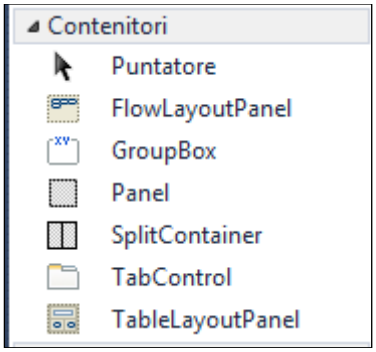


Figura 98: Il gruppo dei controlli contenitori di altri controlli, nella Casella degli Strumenti.

46: I controlli FlowLayoutPanel, Panel e GroupBox.

I controlli **FlowLayoutPanel** (riquadro di visualizzazione scorrevole), **Panel** (riquadro) e **GroupBox** (casella di raggruppamento) sono contenitori di altri controlli; essi svolgono una funzione identica, che consiste nel mostrare simultaneamente all'utente del programma una serie di opzioni, racchiudendo queste opzioni in un riquadro che le distingue e le separa dal resto dell'interfaccia.
Dal punto di vista grafico, il controllo **FlowLayoutPanel** è il più dinamico in quanto, grazie alle proprietà **WrapContents** e **FlowDirection**, consente di visualizzare il suo contenuto su una o più linee, in senso orizzontale o verticale.
Con la proprietà **WrapContents** = **True**, i controlli contenuti, se eccedono la larghezza o l'altezza del **FlowLayoutPanel**, vengono mandati a capo alla fine di ogni riga.
Con la proprietà **WrapContents** = **False**, i controlli contenuti vengono visualizzati su una sola linea o su una sola colonna e, se queste eccedono la larghezza o l'altezza del **FlowLayoutPanel**, il controllo visualizza automaticamente le barre di scorrimento necessarie.
Con la proprietà **FlowDirection** = **LeftToRight** oppure = **RightToLeft**, i controlli contenuti nel **FlowLayoutPanel** sono visualizzati in senso orizzontale, in una o più righe.
Con la proprietà **FlowDirection** = **TopDown** o = **BottomUp**, i controlli contenuti sono visualizzati in senso verticale, in una o più colonne.

Nell'immagine seguente vediamo due controlli FlowLayoutPanel in un form.

- Il primo **FlowLayoutPanel** contiene un gruppo di pulsanti Button disposti su una sola linea, in orizzontale, con una barra di scorrimento per visualizzare i pulsanti che si trovano oltre la larghezza del FlowLayoutPanel.
- Il secondo **FlowLayoutPanel** contiene un gruppo di pulsanti Button disposti su una sola colonna, in verticale, con una barra di scorrimento per visualizzare i pulsanti che si trovano oltre l'altezza del FlowLayoutPanel.

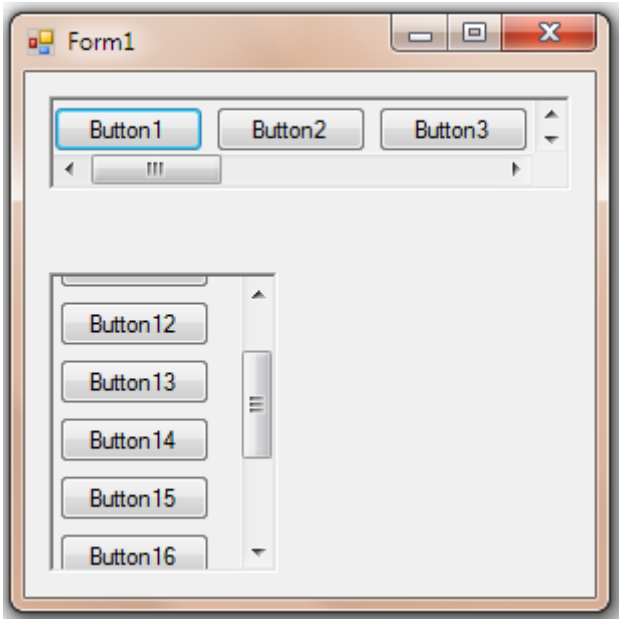


Figura 99: Il controllo FlowLayoutPanel.

I controlli **Panel** e **GroupBox**, dal canto loro, consentono al programmatore di fissare con precisione la posizione degli oggetti in essi contenuti; il controllo **GroupBox** consente inoltre di dare un titolo per il raggruppamento di oggetti.
In questa tabella si può vedere il confronto delle caratteristiche dei tre contenitori:

	FlowLayoutPanel	**Panel**	**GroupBox**
Ha la proprietà **Text** per impostare un titolo con un contorno:	**no**	**no**	**sì**
Ha la proprietà **AutoScroll** per visualizzare con una barra di scorrimento i controlli che escono dai suoi limiti:	**sì**	**sì**	**no**
Ha la proprietà **BorderStyle** per impostare il contorno:	**sì**	**sì**	**no**
Ha la proprietà **FlowDirection** per impostare la visualizzazione del contenuto in orizzontale o in verticale:	**sì**	**no**	**no**

Ha la proprietà **WrapContents** per sistemare automaticamente il contenuto su una sola linea o una sola colonna:	**sì**	**no**	**no**

Tabella 6: Confronto delle caratteristiche dei controlli FlowLayoutPanel, Panel e GroupBox.

Per sistemare un controllo all'interno di uno di questi contenitori è necessario collocare nel form **prima** il contenitore e **poi** al suo interno il controllo che vi si vuole inserire.
Ad esempio, se abbiamo già un pulsante Button1 nel form e vogliamo inserirlo in un GroupBox, non è possibile tracciare il GroupBox e trascinarvi dentro il pulsante Button1 già esistente. L'operazione di inserimento effettuata in questo modo dal punto di vista grafico ***sembra*** riuscita in quanto il pulsante compare graficamente all'interno box, ma ***funzionalmente*** esso non fa parte del box, come si intuisce se si trascina il pulsante a cavallo del bordo del box. Viceversa, un pulsante, disegnato dopo il GroupBox e inserito in esso, ne fa parte graficamente e funzionalmente e non può uscire dai limiti del contenitore[38]:

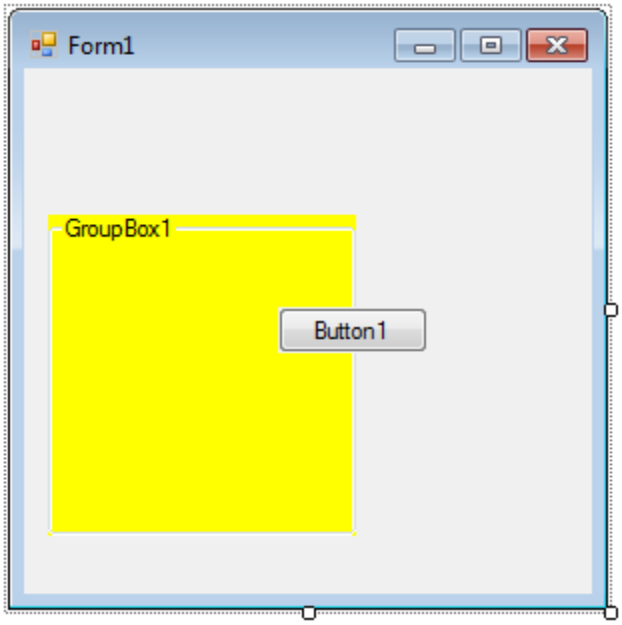


Figura 100: Un pulsante Button1 esterno al GroupBox.

[38] Un altro metodo per verificare se un controllo è inserito o meno in un contenitore è questo: spostando il contenitore sul form con il mouse, si spostano con esso tutti i controlli che vi sono contenuti. Un controllo che non segue lo spostamento del contenitore non ne fa parte.

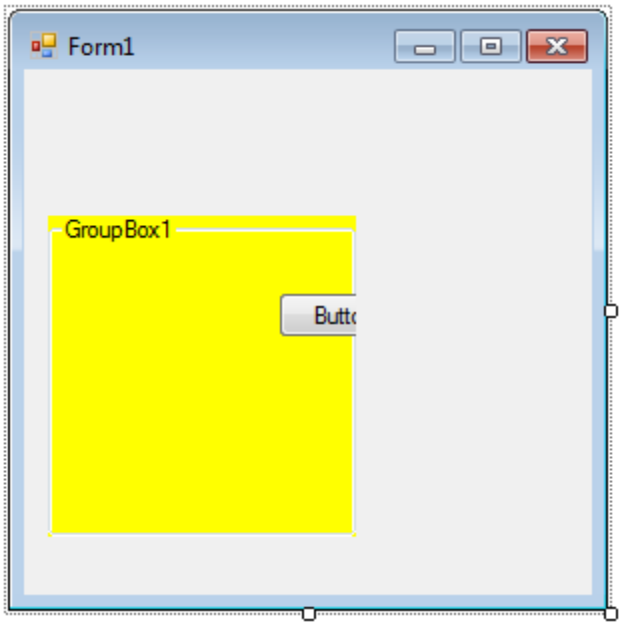


Figura 101: Un pulsante Button1 interno al GroupBox.

I controlli che si trovano in uno di questi contenitori assumono come riferimenti della loro proprietà **Location** (posizione) non più i bordi sinistro e superiore del form, ma i bordi sinistro e superiore del controllo contenitore.
Ad esempio, un pulsante Button che si trovi all'interno di un contenitore con la proprietà **Location = 8; 8** si trova a 8 pixel di distanza dal bordo sinistro e dal bordo superiore del contenitore stesso, a prescindere dalla posizione che questo ha nel form.

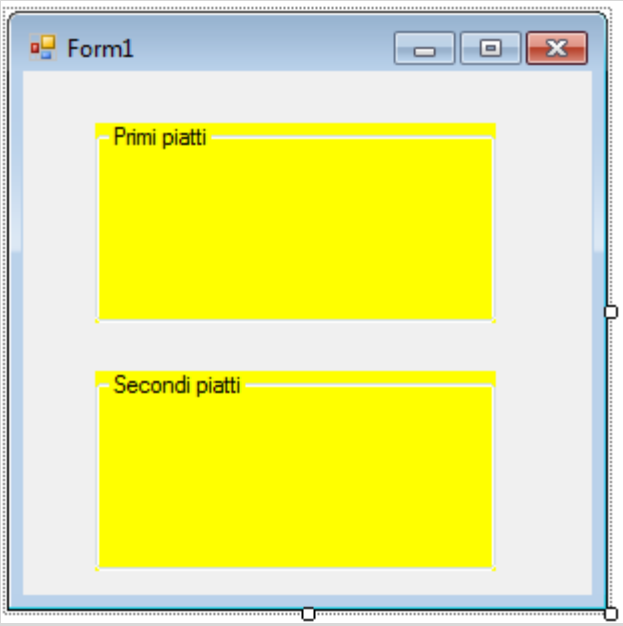
Altri effetti della dipendenza dei controlli contenuti nel FlowLayoutPanel nei riguardi del loro contenitore sono questi:

- Se il controllo contenitore è spostato nel form, si spostano assieme ad esso i controlli in esso contenuti.
- Se il controllo contenitore è copiato e replicato, si replicano assieme ad esso i controlli in esso contenuti.
- Se il controllo contenitore è cancellato, si cancellano assieme ad esso i controlli in esso contenuti.
- Se il controllo contenitore è impostato come invisibile (proprietà **Visible = False**) i controlli in esso contenuti sono ugualmente invisibili.
- Se il controllo contenitore è disabilitato (proprietà **Enabled = False**) i controlli in esso contenuti sono ugualmente disabilitati; il contenitore e i controlli in esso contenuti diventano un'area grigia che non risponde più ai movimenti del mouse.
- Se è necessario selezionare un gruppo di controlli dentro un contenitore non è possibile utilizzare il puntatore: i controlli da selezionare vanno cliccati uno a uno, tenendo premuto il tasto MAIUSC.

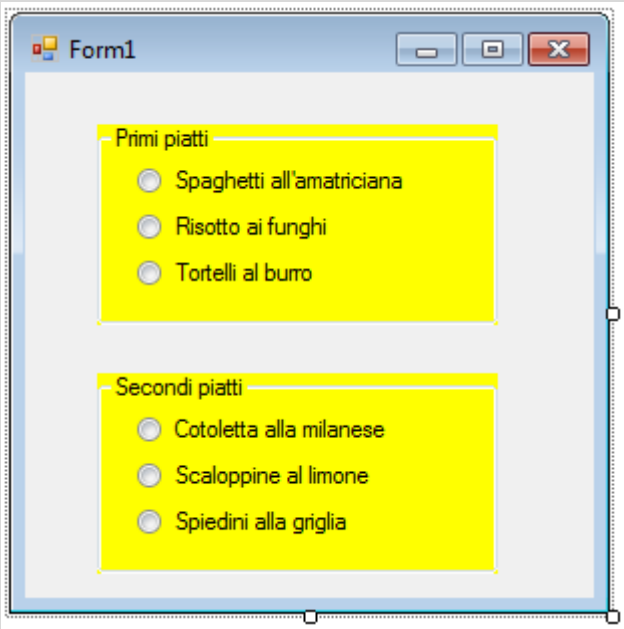
I tre contenitori FlowLayoutPanel, Panel e GroupBox separano i controlli in essi contenuti dagli altri controlli presenti nel form non solo dal punto di vista ***grafico***, ma anche dal punto di vista ***funzionale***.
Questa caratteristica assume una certa importanza se nel form sono presenti dei controlli **RadioButton** che, come abbiamo visto, consentono all'utente del programma di effettuare una scelta, tra più opzioni, con un *clic* del mouse.
Sono controlli a scelta esclusiva perché consentono una sola scelta tra tutte quelle presentate all'utente. Se l'utente cambia idea e seleziona un'altra opzione, l'opzione precedente viene automaticamente deselezionata.
Ora, supponiamo di avere in un form sei controlli RadioButton per scegliere una opzione in un elenco di cibi. Come abbiamo visto, selezionando uno dei sei cibi si deselezionano gli altri cinque. La scelta può essere cambiata, ma in ogni caso il cibo selezionato è sempre solo uno.
Supponiamo di avere gli stessi sei controlli RadioButton contenuti in due diversi contenitori GroupBox. In questo caso, i tre RadioButton contenuti in un box ignorano la presenza dei tre RadioButton contenuti nell'altro box e l'utente può effettuare una scelta sia nel primo box che nel secondo box.
Vedremo questa possibilità nel prossimo esercizio.

Esercizio 19: GroupBox e pulsanti RadioButton.

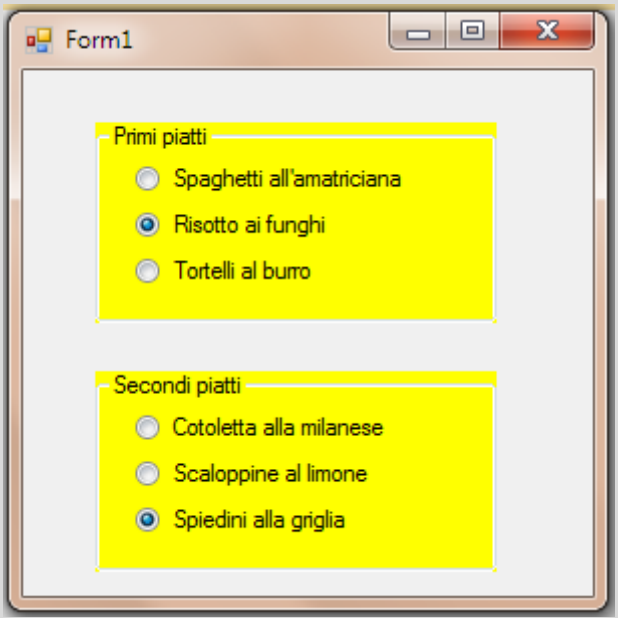
Apriamo un nuovo progetto e collochiamo nel form due controlli GroupBox.
Diamo al primo GroupBox il titolo "*Primi piatti*" (proprietà **Text = Primi piatti**), e diamo al secondo box il titolo "*Secondi piatti*" (proprietà **Text = Secondi piatti**).



Inseriamo nel box "*Primi piatti*" tre controlli RadioButton e scriviamo nelle loro proprietà Text, rispettivamente: "Spaghetti all'amatriciana", "*Risotto ai funghi", "Tortelli al burro".*
Inseriamo nel box "*Secondi piatti*" tre controlli RadioButton e scriviamo nelle loro proprietà Text, rispettivamente: "*Cotoletta alla milanese*", "*Scaloppine al limone*", "*Spiedini alla griglia*".



Mandiamo in esecuzione il programma e notiamo che è possibile scegliere un piatto nel box "*Primi piatti*" e un altro piatto nel box "*Secondi piatti*", cosa che non sarebbe possibile se i sei RadioButton fossero tutti contenuti in un unico GroupBox o fossero collocati liberamente nel form senza alcun contenitore.



47: Il controllo SplitContainer

Il controllo **SplitContainer** (contenitore di una divisione) visualizza l'area del form divisa in due regioni, denominate **Panel1** e **Panel2**, disposte in senso verticale o in senso orizzontale.
Ognuna delle due regioni contiene un controllo Panel, nel quale il programmatore può collocare altri controlli (immagini, pulsanti).
Il controllo visualizza una barra di divisione che l'utente del programma può spostare rispettivamente verso il basso o verso l'alto, verso destra o verso sinistra, per regolare la grandezza dell'area riservata alle due regioni.

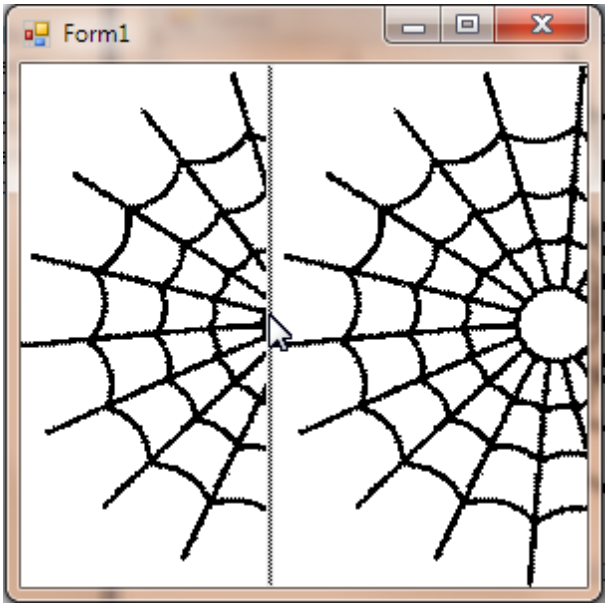


Figura 102: Il controllo SplitContainer in fase di esecuzione di un programma.

48: Il controllo TabControl

Il controllo **TabControl** (controllo a schede sovrapposte) visualizza su un unico form una serie di schede, tutte della stessa dimensione, contraddistinte ognuna da una etichetta.
Cliccando una di queste etichette si visualizza la scheda corrispondente, la quale può contenere testo, immagini e altri controlli quali Button, CheckBox, ListBox e simili.
Se il contenuto di una scheda eccede le dimensioni della scheda, il controllo TabControl visualizza le barre di scorrimento necessarie.

Figura 103: Il controllo TabControl e una scheda con le barre di scorrimento.

Per aggiungere nuove schede in un TabContrl bisogna cliccare il pulsante con i tre puntini nella proprietà **TabPages**. Si accede così all'**Editor dell'insieme TabPages**, nel quale è possibile aggiungere nuove schede cliccando il pulsante **Aggiungi**.
Sempre nell'**Editor dell'insieme TabPages** è possibile dare un titolo a ogni scheda impostandone la proprietà **Text**.
Una volta concluse le operazioni di inserimento di schede, premere il pulsante **OK**.

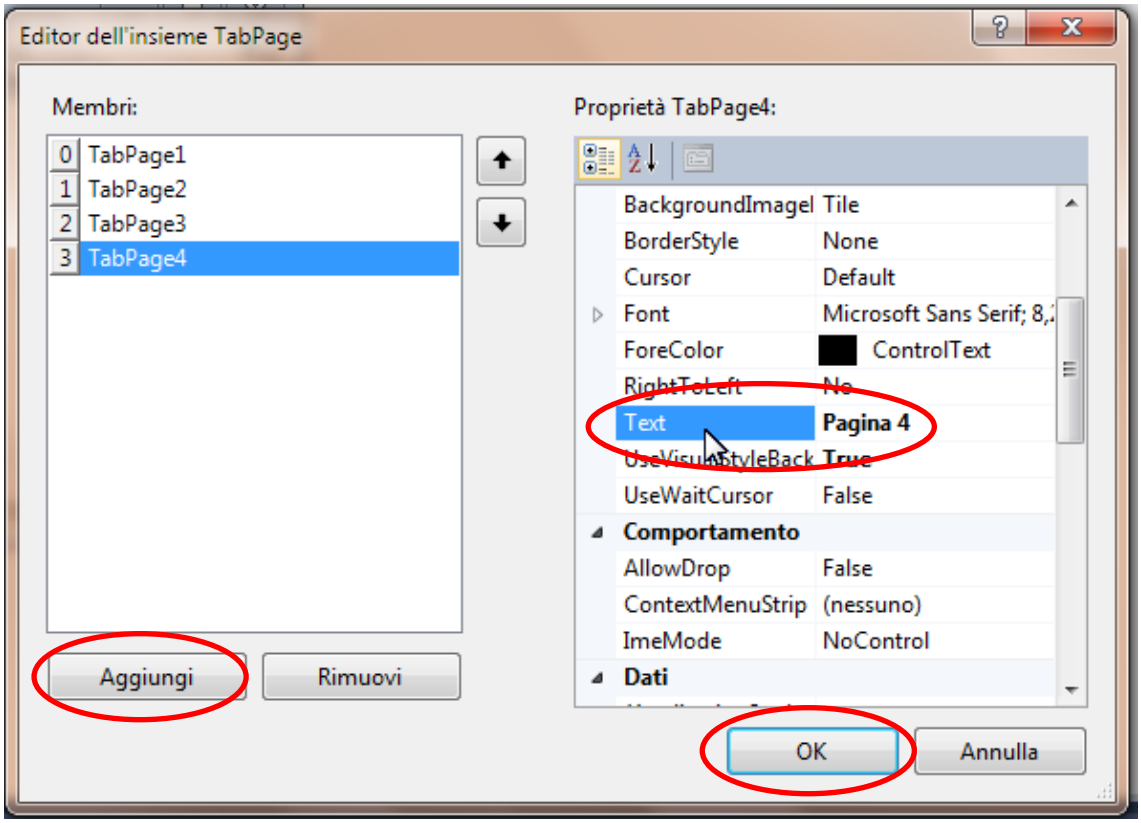
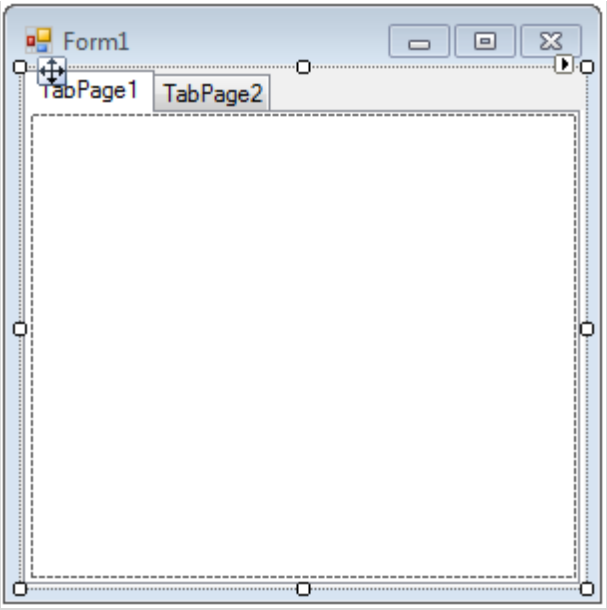


Figura 104: Aggiunta di nuove schede in un controllo TabControl.

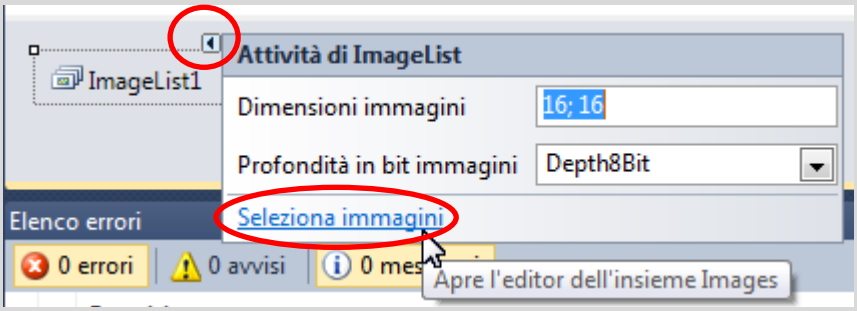
E' possibile associare un TabControl a un componente **ImageList** (Lista di immagini). Quando un TabControl è associato a una ImageList, è possibile visualizzare un'immagine a sinistra del titolo di ogni scheda.
E' quanto faremo nel prossimo esercizio.

Esercizio 20: Il controllo TabControl associato a un componente ImageList.

In questo esercizio creeremo un semplice programma con cinque schede, una per ogni numero, da 1 a 5.
I titolo delle cinque schede saranno UNO, DUE, TRE, QUATTRO, CINQUE.
Accanto a ogni titolo si vedrà un'immagine del numero corrispondente.
Cliccando la scheda relativa a un numero, l'utente potrà visualizzare, in un'immagine, lo stesso numero raffigurato dalle dita di una mano.
Apriamo un nuovo progetto e collochiamo nel Form1 un controllo TabControl come in questa immagine:



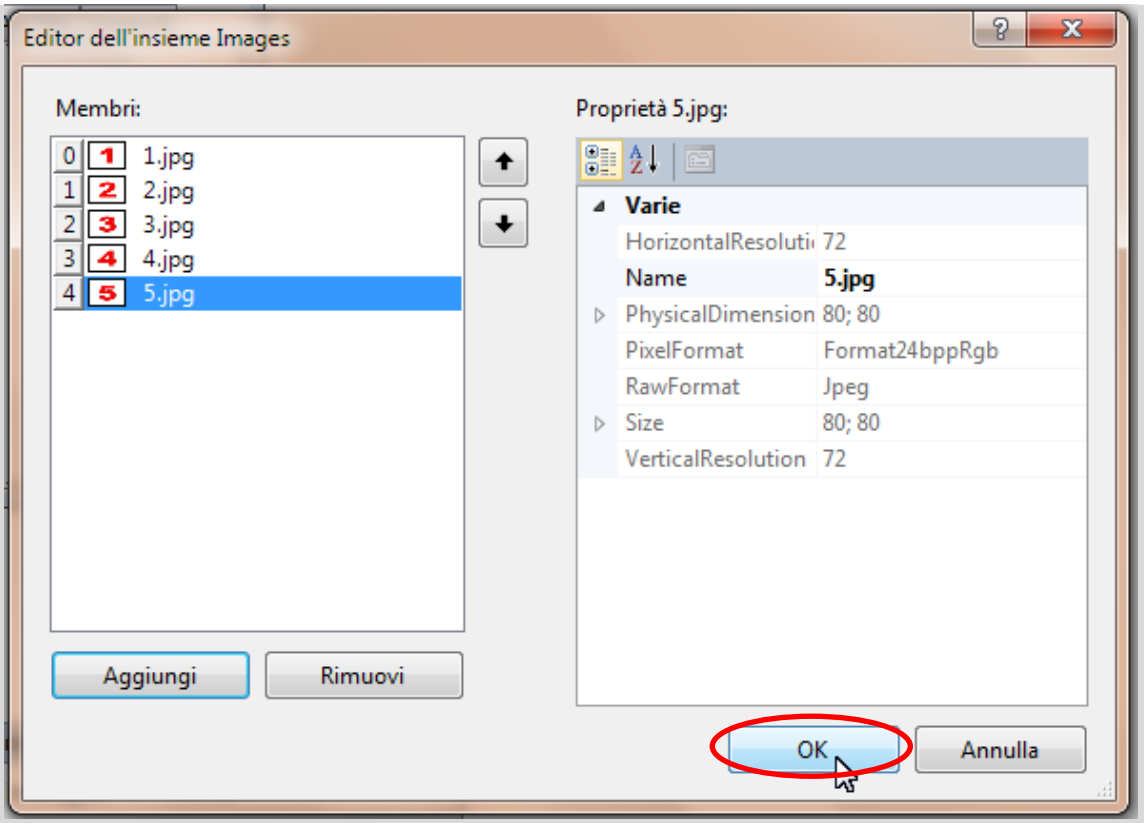
Ora collochiamo nel form un componente **ImageList**.
Trattandosi di un componente (oggetto non visibile all'utente) e non di un controllo, l'ImageList va a collocarsi nell'area al di sotto del Form1.
Facciamo un *clic* con il mouse sul pulsante con la freccina nera che compare sul bordo del componente ImageList in alto a destra: nella finestra **Attività di ImageList** facciamo un *clic* su **Seleziona immagini**:



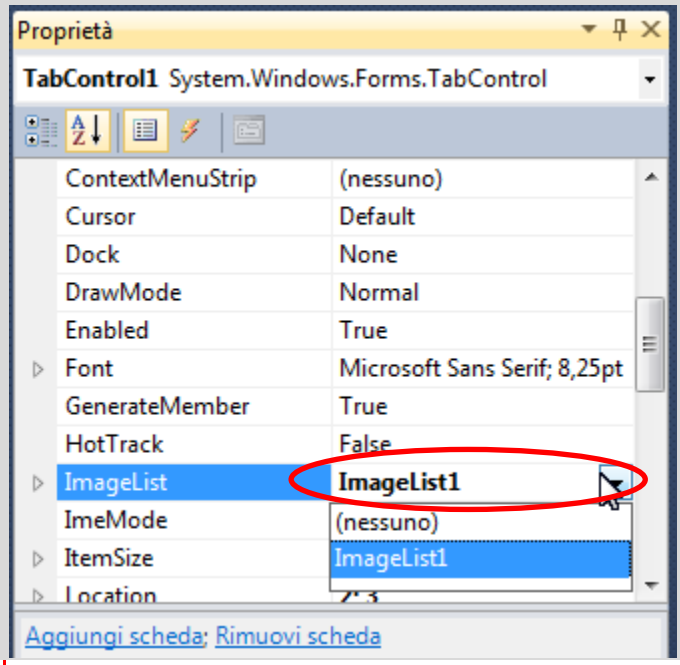
Accediamo così alla finestra dell'**Editor dell'insieme Images.** Cliccando il pulsante Aggiungi, inseriamo nella ImageList le immagini

- 1.jpg
- 2.jpg
- 3.jpg
- 4.jpg
- 5.jpg

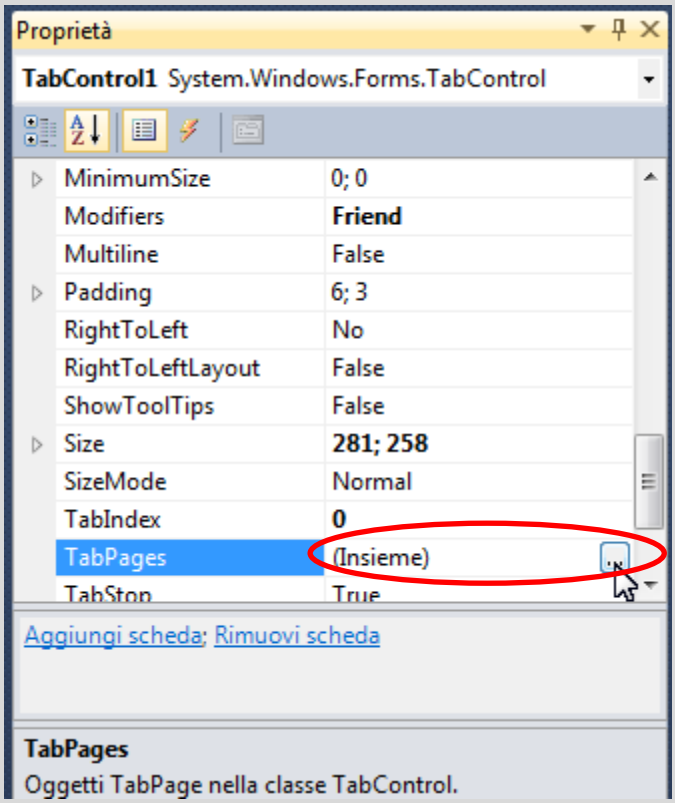
che si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Numeri**:



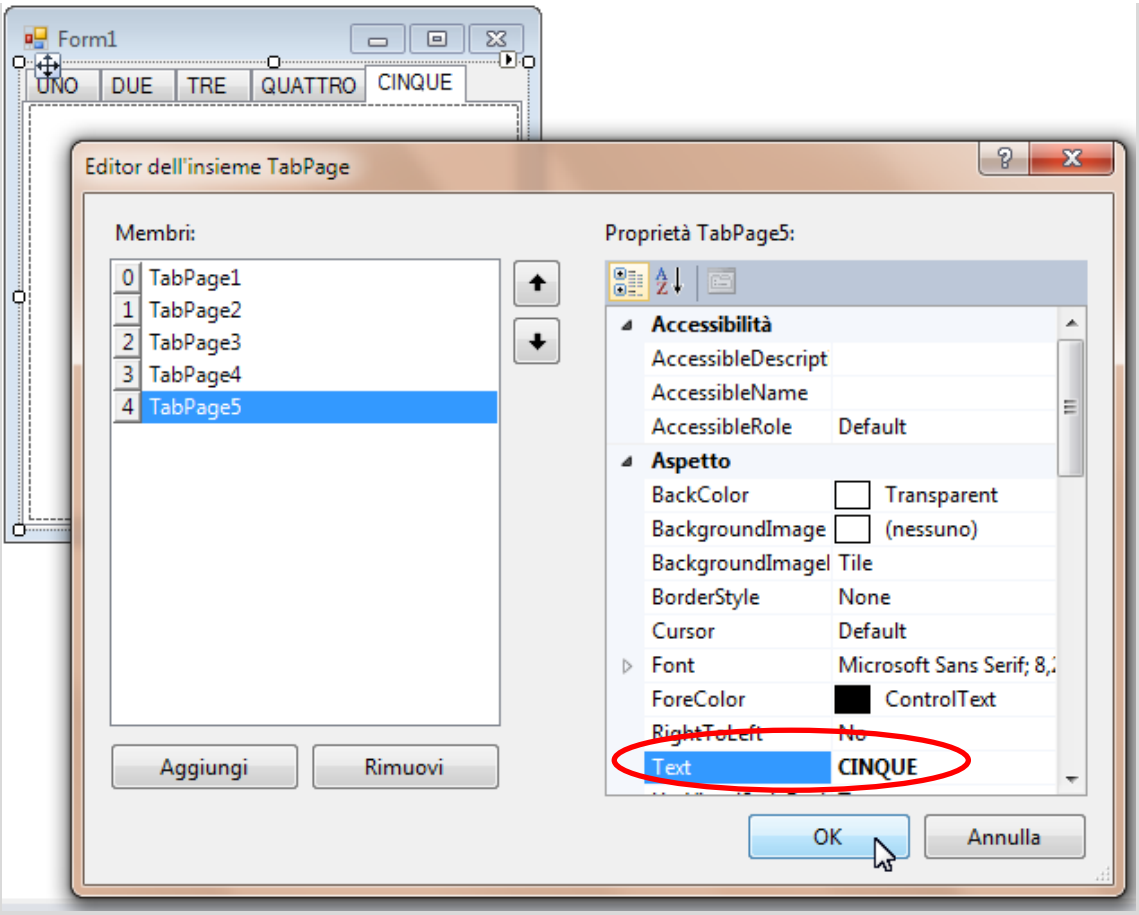
Dopo avere premuto il pulsante OK, torniamo a interessarci del controllo TabControl. Facciamo un *clic* sul controllo e accediamo alla finestra delle sue proprietà. Impostiamo la sua proprietà **ImageList =** ImageList1, in modo da collegare il TabControl alla ImageList che abbiamo appena creato:



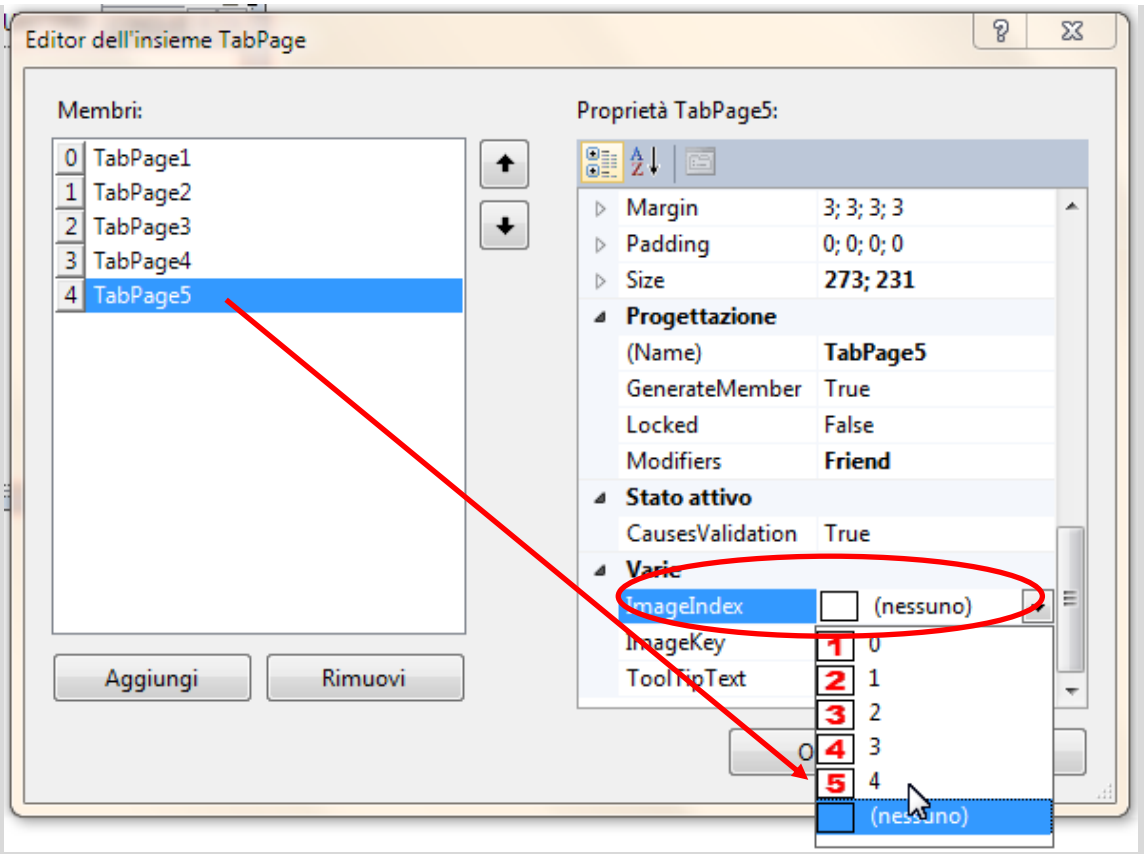
Ora facciamo un *clic* sul pulsante che compare nella proprietà TabPages del TabControl:



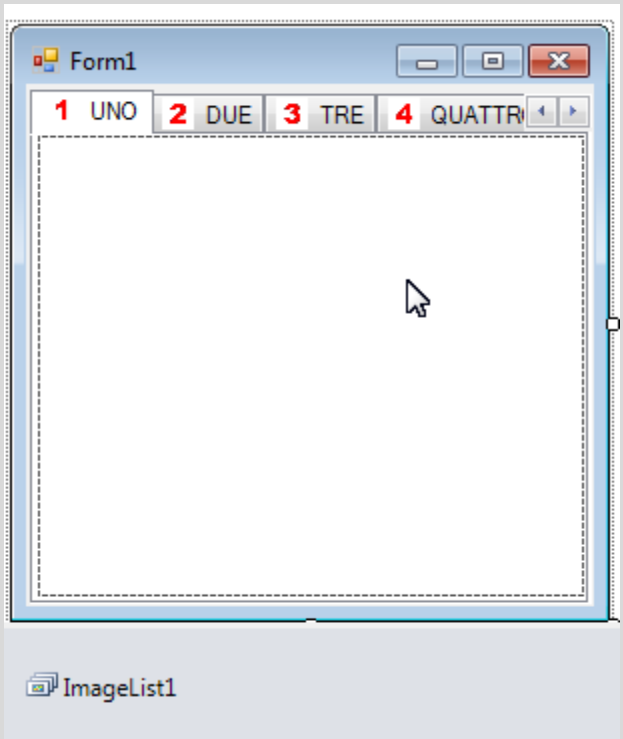
Qui, usando il pulsante Aggiungi, inseriamo una alla volta cinque schede, impostando la proprietà **Text** di ogni scheda con le parole UNO, DUE, TRE, QUATTRO, CINQUE.



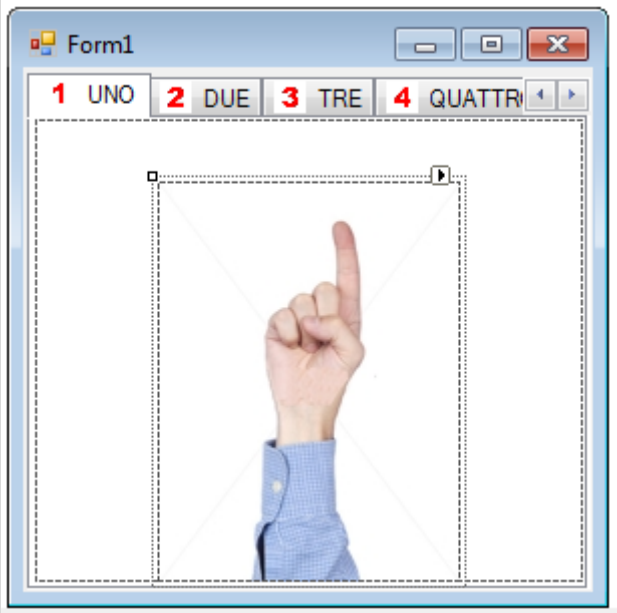
Una volta scritti i titoli delle cinque schede, cerchiamo la proprietà **ImageIndex** di ogni scheda e le assegniamo l'immagine, già caricata nella ImgeList, che corrisponde al suo titolo:



Il nostro TabControl ora si mostra così:



Inseriamo nella scheda UNO un controllo PictureBox e assegniamo come proprietà Image a questo controllo l'immagine della mano che mostra un dito, che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini / Mano**:



Procediamo inserendo un controllo PictureBox nelle altre schede DUE, TRE, QUATTRO, CINQUE, con l'immagine corrispondente.
Ecco un'immagine del programma in esecuzione:

49: Il controllo TableLayoutPanel

Il controllo **TableLayoutPanel** (riquadro di visualizzazione a tabella) visualizza i controlli in esso contenuti in righe e in colonne.
Il programmatore ha la facoltà di aggiungere o eliminare colonne e righe cliccando il pulsante con la freccina nera che compare in alto a destra del controllo.
Il controllo distanzia automaticamente tutte le colonne e tutte le righe presenti al suo interno in modo uniforme; il programmatore ha tuttavia la facoltà di modificare le dimensioni di colonne e righe.

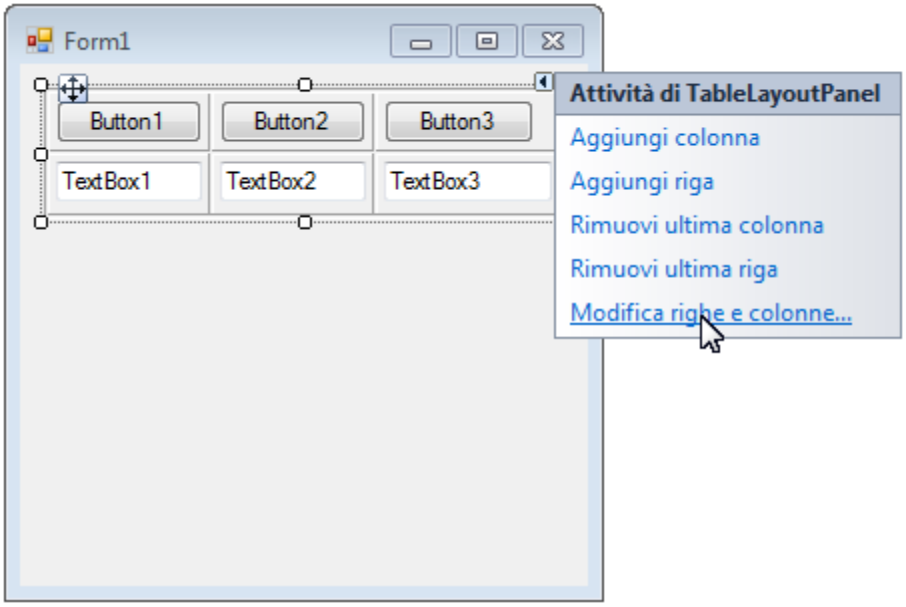


Figura 105: L' impostazione di colonne e righe in un controllo TabelLayoutPanel.

Una proprietà interessante di questo controllo, dal punto di vista grafico, è la proprietà **CellBorderStyle**, che consente di visualizzare in diversi modi le linee di divisione delle celle della tabella.
Nell'immagine seguente, la proprietà CellBorderStyle è impostata come **InsetDouble** (incavo doppio):

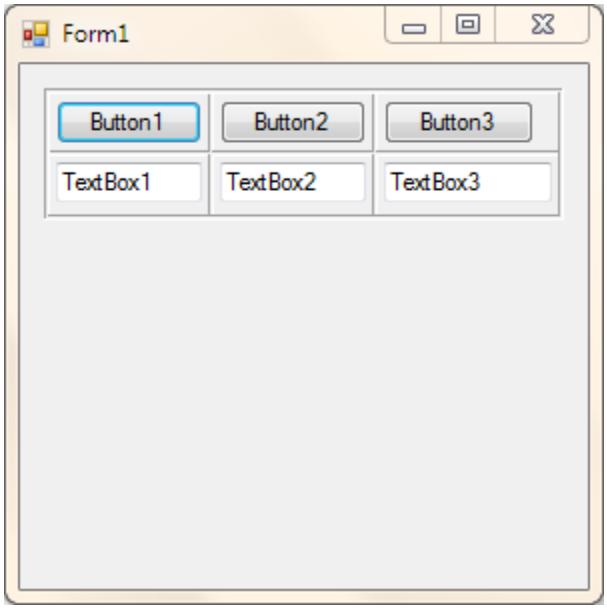


Figura 106: Un controllo TableLayoutPanel con tre colonne e due righe di controlli.

Capitolo 8: I COMPONENTI PER LA CREAZIONE DI STRISCE DI MENU O DI PULSANTI.

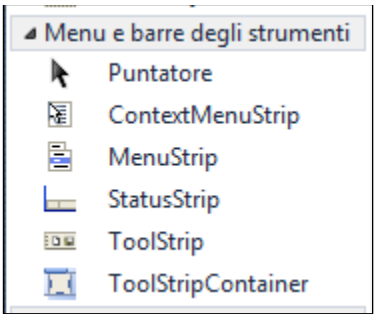


Figura 107: I controlli per la creazione di strisce di menu, nella Casella degli Strumenti.

50: Il componente ContextMenuStrip.

Il componente **ContextMenuStrip** (striscia di menu contestuale) crea un menu collegato in modo esclusivo a un singolo oggetto presente nel form; questo menu sarà visibile solo quando l'utente farà un *clic* con **il tasto destro** del mouse sull'oggetto in questione.
Il collegamento del ContextMenuStrip a un controllo presente nel form viene impostato dalla Finestra Proprietà del controllo **destinatario** e non del ContextMenuStrip.
Nel prossimo esercizio vedremo le modalità di collegamento di un ContextMenuStrip a una Label, per la creazione di un menu con tre opzioni che l'utente del programma potrà visualizzare facendo un *clic* sulla Label con il tasto destro del mouse.

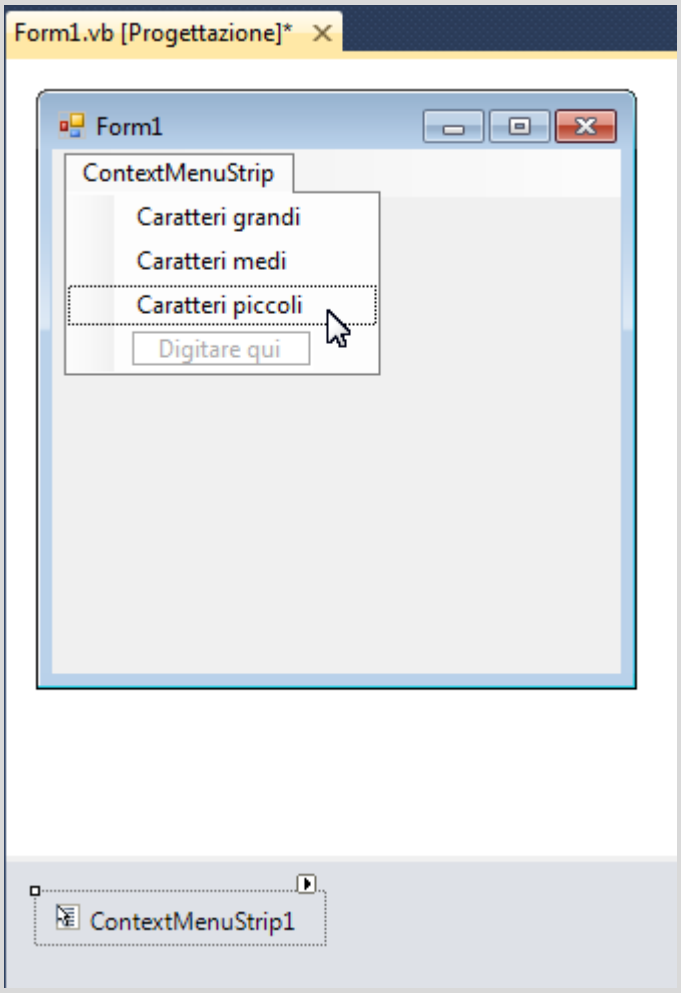
Esercizio 21: Creazione di un menu contestuale per un controllo Label.

Apriamo un nuovo progetto e inseriamo nel form una Label1 e un ContextMenuStrip1.

Quando il programma sarà in esecuzione, nella Label1 verrà visualizzato un testo. Facendo un *clic* con il tasto destro del mouse sulla Label, l'utente visualizzarà un menu con tre opzioni relative alla grandezza dei caratteri (grandi, medi, piccoli) del testo.

Iniziamo a impostare il menu con le tre opzioni.
Facciamo un *clic* sul ContextMenuStrip1, che compare nell'area sottostante al form, e nel menu che si apre in alto nel form scriviamo queste tre opzioni:

- **Caratteri grandi**
- **Caratteri medi**
- **Caratteri piccoli**



Facciamo un doppio *clic* sulla prima di queste opzioni (Caratteri grandi) e accediamo così alla Finestra del Codice dove troviamo già impostata la procedura che gestisce l'evento del *clic* del mouse su questa opzione del menu:

```
    Private Sub ToolStripMenuItem1_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripMenuItem1.Click


    End Sub
```

Completiamo la procedura scrivendo una riga di istruzioni perché il testo nella Label1 venga visualizzato con i caratteri "grandi":

```
    Private Sub ToolStripMenuItem1_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripMenuItem1.Click

        Label1.Font = New Font("Verdana", 14)

    End Sub
```

Ripetiamo questa operazione anche per le altre due opzioni del menu, sino a completare il codice del programma:

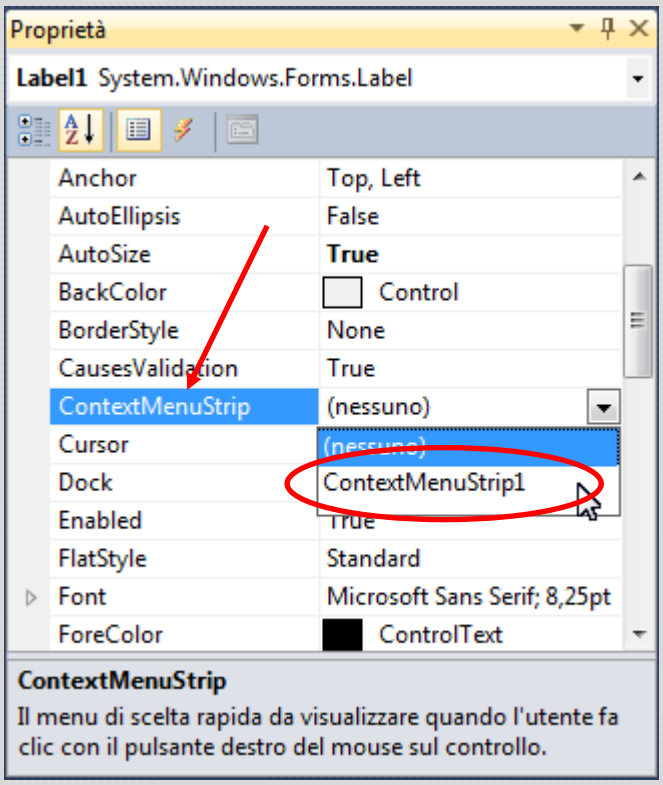
```
Public Class Form1

    Private Sub ToolStripMenuItem1_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripMenuItem1.Click

        Label1.Font = New Font("Verdana", 14)

    End Sub

    Private Sub CaratterimediToolStripMenuItem_Click(sender As System.Object, e
As System.EventArgs) Handles CaratterimediToolStripMenuItem.Click

        Label1.Font = New Font("Verdana", 12)

    End Sub

    Private Sub CaratteriPiccoliToolStripMenuItem_Click(sender As System.Object,
e As System.EventArgs) Handles CaratteriPiccoliToolStripMenuItem.Click

        Label1.Font = New Font("Verdana", 10)

    End Sub

End Class
```

Torniamo alla Finestra di Progettazione.
Facciamo un *clic* con il mouse sulla Label1 e nella Finestra Proprietà di questo controllo impostiamo queste proprietà:

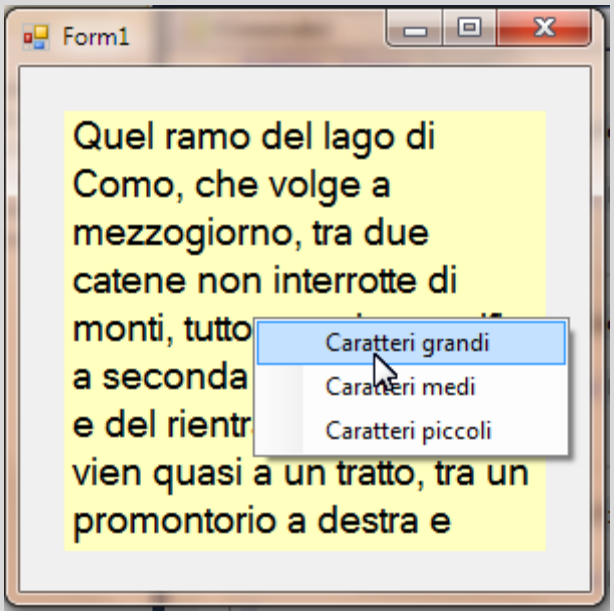
- **BackColor = 255; 255; 192**
- **Location = 22; 22**
- **Size = 240; 220**
- Nella proprietà **Text**, scriviamo un testo a piacere.
- Veniamo infine alla proprietà **ContextMenuStrip**, che è la più importante ai fini di questo esercizio, in quanto collega il controllo Label al componente ContextMenuStrip1 e al menu che abbiamo già creato.

A destra della proprietà **ContextMenuStrip** troviamo un pulsante con una freccina nera.

Facendo un *clic* su questo pulsante si visualizza l'elenco dei componenti ContextMenuStrip presenti nel form (nel nostro caso abbiamo solo il ContrextMenuStrip1) .
Facendo un *clic* su ContextMenuStrip1 si stabilisce il collegamento tra questo componente e la Label.



L'immagine che segue mostra il programma in esecuzione:



51: Il controllo MenuStrip.

Il controllo **MenuStrip** (striscia di menu) inserisce nel form, in alto, sotto la barra con il titolo, strisce di menu e sottomenu che servono a visualizzare in modo immediato tutta una serie di opzioni o di comandi a disposizione dell'utente.
Per quanto il controllo MenuStrip sia uno strumento molto articolato ed efficace, l'inserimento di una striscia di menu in un progetto è un'operazione piuttosto semplice.
Apriamo un nuovo progetto e, nella Casella degli Strumenti, facciamo un doppio *clic* su questo controllo.
Vediamo che esso va a collocarsi nella parte superiore del form, con due aree sensibili dalle quali si può avviare la creazione dei menu in due modi diversi. Le vediamo evidenziate in questa immagine:

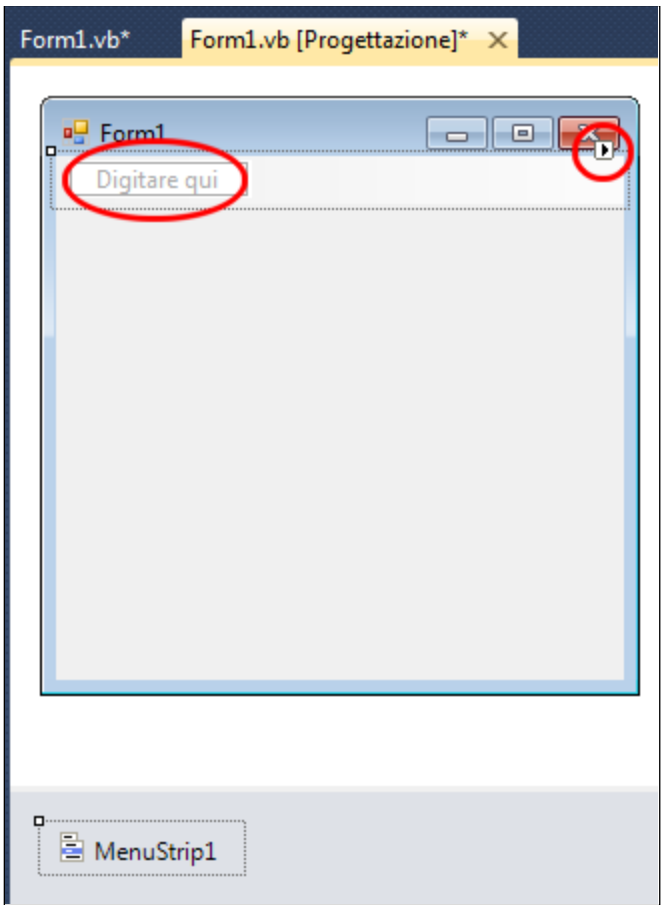


Figura 108: Il controllo MenuStrip.

Facciamo un *clic* sulla freccina in alto a destra; accediamo così a una finestra nella quale si possono scegliere alcune impostazioni per la striscia di menu che stiamo creando:

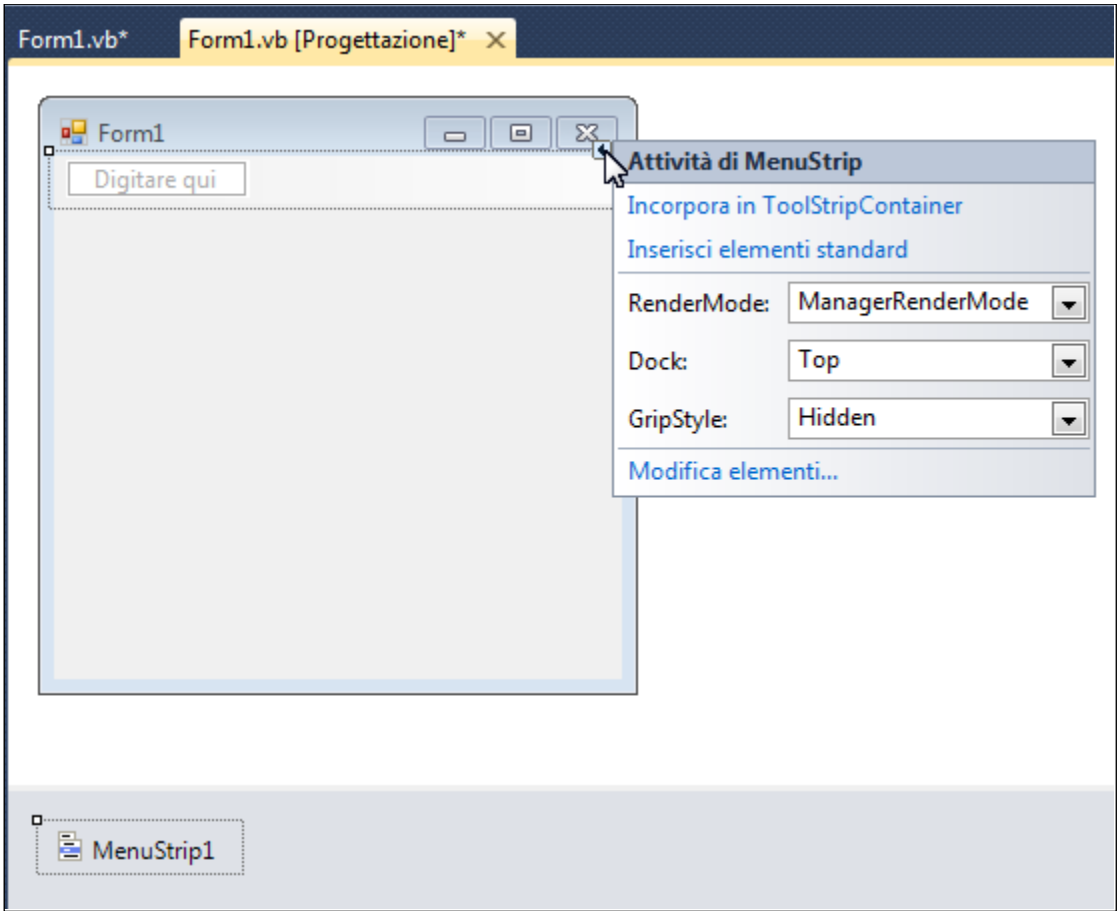


Figura 109: Le impostazioni di base del controllo MenuStrip.

Vediamo queste opzioni, una a una:

- **Incorpora in ToolStripContainer**: cliccando questa opzione, il **MenuStrip** può essere incorporato in un contenitore di menu assieme a una striscia di pulsanti (si veda più avanti il paragrafo dedicato al componente **ToolStripContainer**).
- **Inserisci elementi standard**: cliccando questa opzione si crea un menu standard, completo di icone, nel quale sono inseriti in modo automatico tutti gli item tipici delle applicazioni Windows.
- **RenderMode**: questa proprietà imposta la modalità grafica di visualizzazione dei sottomenu, quando il programma sarà in esecuzione. La modalità **System** è la più disadorna, la modalità **Professional** è la più elegante.
- **Dock** (ormeggio): l'impostazione di questa proprietà consente di collocare la striscia di menu su uno dei quattro lati del form (e dunque anche in posizione verticale) o in un'altra posizione, non legata al form, voluta dal programmatore.
- **GripStyle** (impostazione della presa): questa proprietà imposta la visualizzazione, a sinistra della striscia di menu, di una maniglia che consentirà all'utente di spostare la striscia per portare in primo piano, ad esempio, quella usata più frequentemente. Vediamo queste maniglie, nell'ambiente di progettazione di VB, in questa immagine:

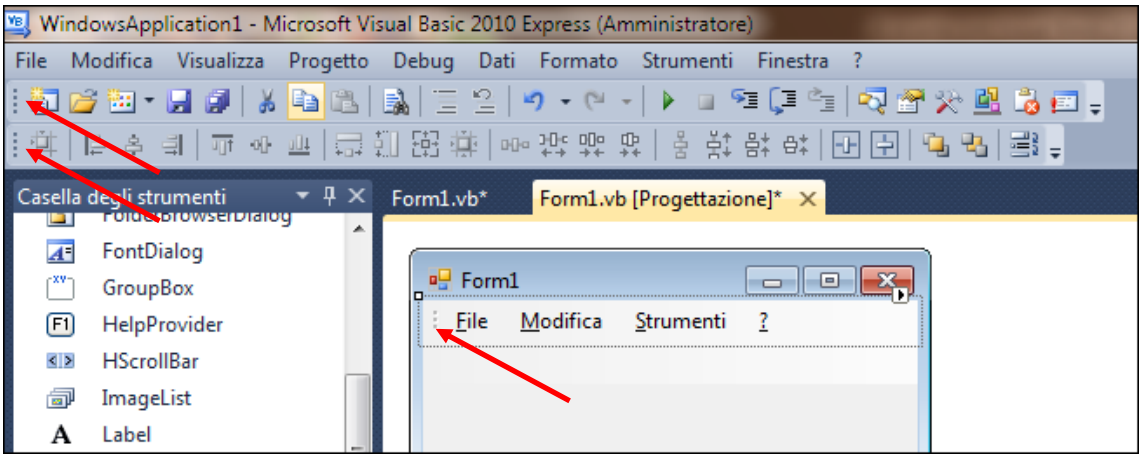


Figura 110: Le maniglie delle strisce di pulsanti dell'ambiente di progettazione di VB.

- **Modifica elementi**: questa ultima opzione consente di modificare il testo e l'organizzazione della striscia di menu e di sottomenu alla quale si sta lavorando.

Ora clicchiamo l'opzione **Inserisci elementi standard** e vediamo che la striscia di menu si riempie con tutti gli item tipici di una applicazione Windows, con le relative icone e le combinazioni di tasti di scelta rapida:

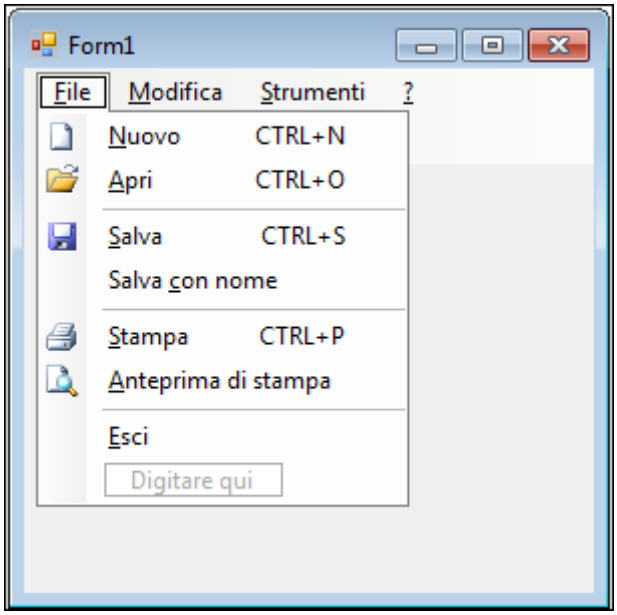


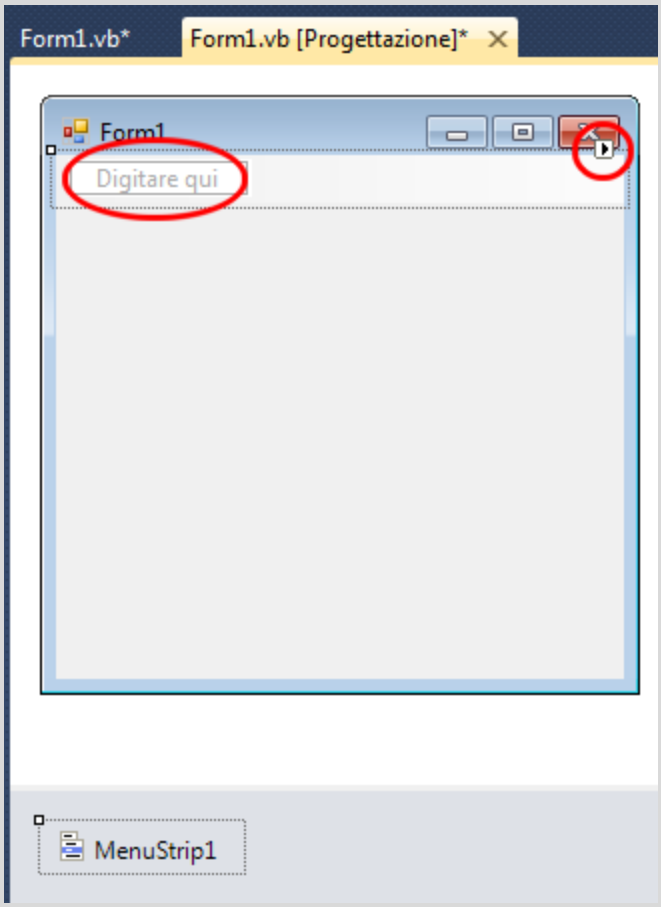
Figura 111: Il componente MenuStrip con gli elementi standard.

Naturalmente l'elenco di questi **elementi standard** può essere modificato dal programmatore cliccando l'opzione **Modifica elementi** ma si verifica spesso il caso che un programmatore debba creare una striscia di menu e sottomenu tagliati su misura per una determinata applicazione: in questi casi è preferibile impostare il MenuStrip

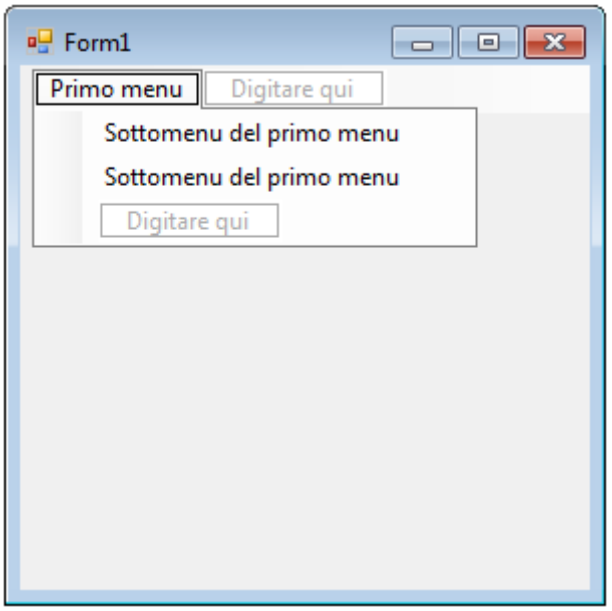
partendo dalla prima delle due aree sensibili, quella che si vede a sinistra con la scritta **Digitare qui**
E' quanto faremo nel prossimo esercizio.

Esercizio 22: Impostazioni del controllo MenuStrip.

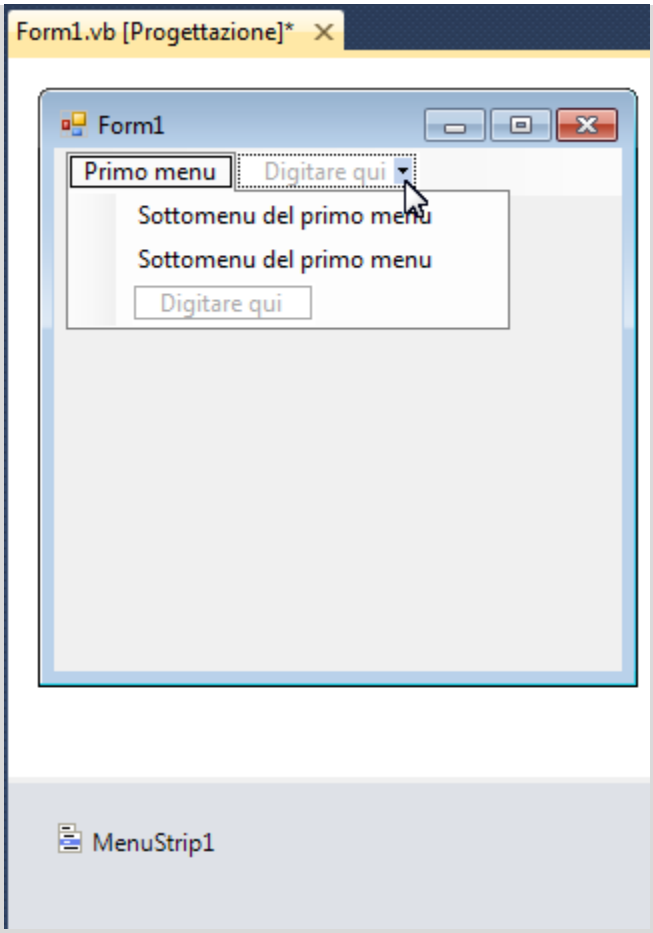
In questo esercizio vedremo come inserire in un progetto una striscia di menu, con i sottomenu che si aprono a tendina.
In questo esercizio ci limiteremo a un lavoro **grafico**: inseriremo una striscia di menu in un form, senza tuttavia scrivere le procedure per gestire i *clic* dell'utente del programma su questi menu.
Apriamo un nuovo progetto e inseriamo nel Form1 il controllo MenuStrip1.



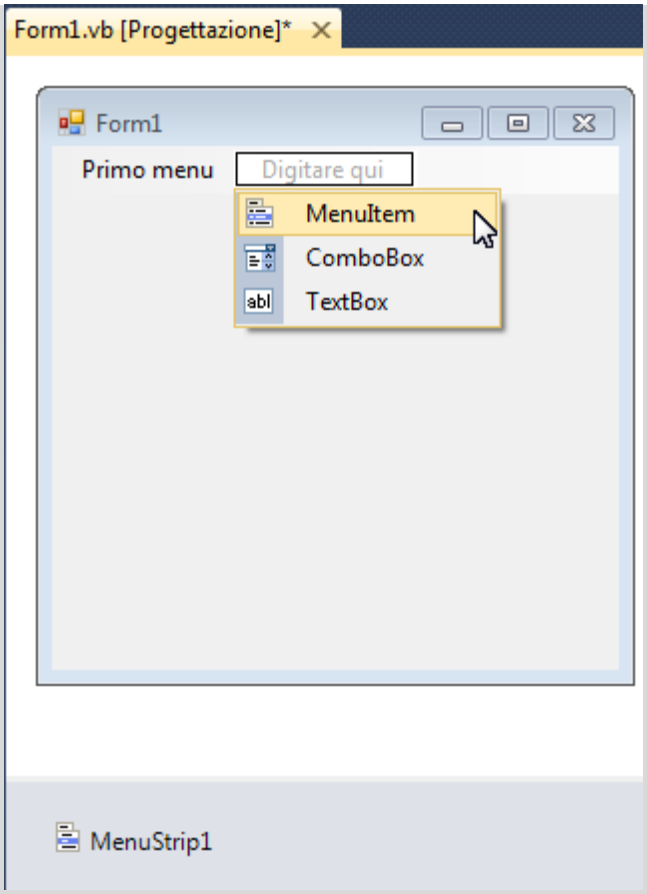
Nella prima area sensibile del controllo, a sinistra, vediamo un riquadro con il testo **Digitare qui**.
Facciamo un *clic* in questo riquadro e iniziamo a scrivere gli ipotetici item di un primo menu come in questa immagine:



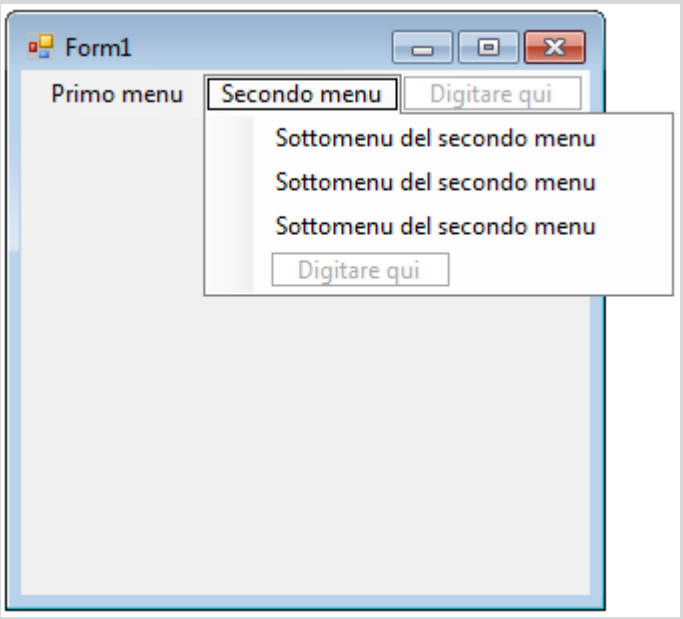
Conclusa la scrittura del **Primo menu**, notiamo che alla sua destra si apre un nuovo riquadro, anche questo con il testo **Digitare qui**, pronto a ricevere il **Secondo menu**. Ma prima di scrivere in questo riquadro fermiamoci un attimo: portiamo il mouse sul riquadro ma non facciamo *clic* sul testo **Digitare qui**.
Vediamo comparire, sul lato destro del riquadro, un pulsante con una freccina:



Facciamo un *clic* su questo pulsante e visualizziamo così un menu a tendina con la possibilità di scegliere alcuni oggetti da inserire nel menu: un **item** scritto, un controllo **ComboBox** oppure un controllo **TextBox**.



Con l'inserimento di questi oggetti è possibile creare una striscia di menu molto articolata e complessa; nel nostro caso continuiamo con la scrittura di item, creando un secondo menu con i suoi sottomenu:



Ora mandiamo in esecuzione il progetto e verifichiamo la visualizzazione dei due menu e dei loro sottomenu.
Come abbiamo detto all'inizio, menu e sottomenu sono inefficaci, in quanto non abbiamo scritto nessuna procedura per gestire i *clic* del mouse su di essi.
Vedremo nel prossimo esercizio un esempio di scrittura di queste procedure.

Esercizio 23: Creazione di una striscia di menu.

In questo esercizio creeremo un programma con una striscia di menu formata da due menu principali con i loro sottomenu.
I due menu avranno come intestazione, rispettivamente

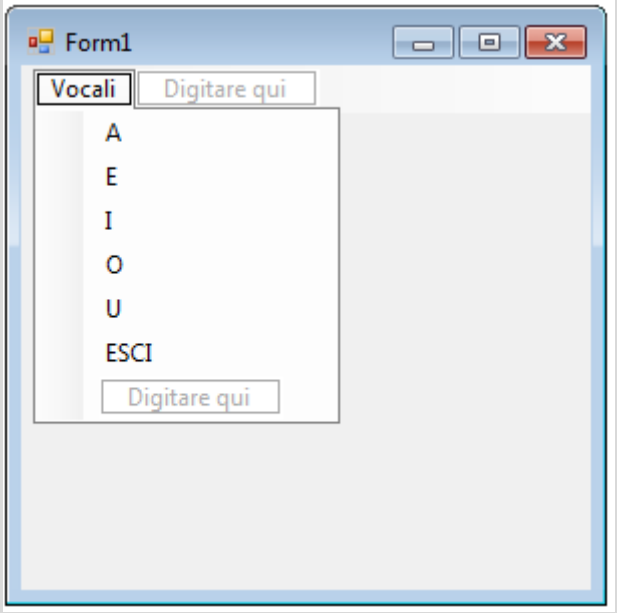
- **Vocali**
- **Numeri**

Ognuno di essi si aprirà a tendina su una serie di sottomenu. Lo schema completo dei menu e dei comandi che vogliamo far apparire nel programma è questo:

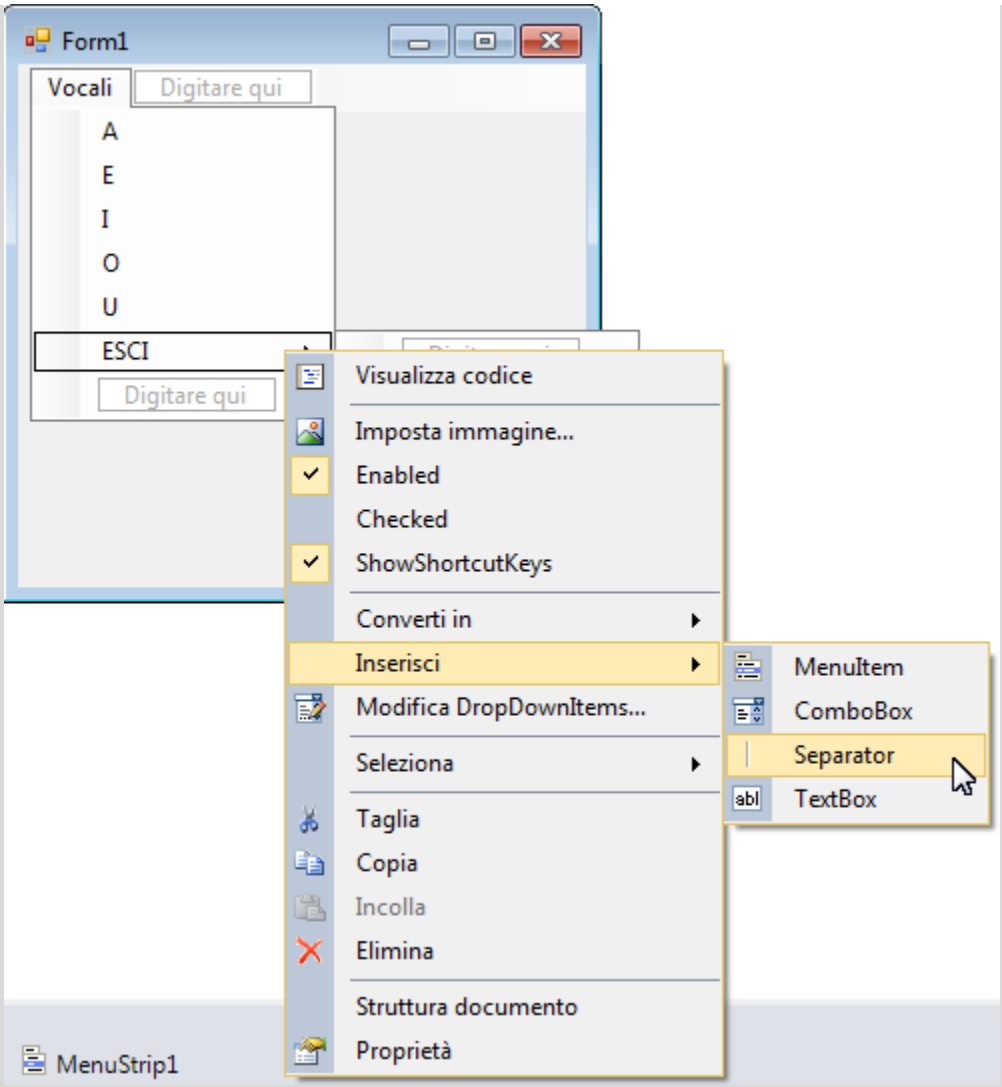
Vocali	**Numeri**
A	1
E	2
I	3
O	
U	
(linea di separazione)	
ESCI	

Gli item 1, 2 e 3 del menu **Numeri** avranno un'immagine.

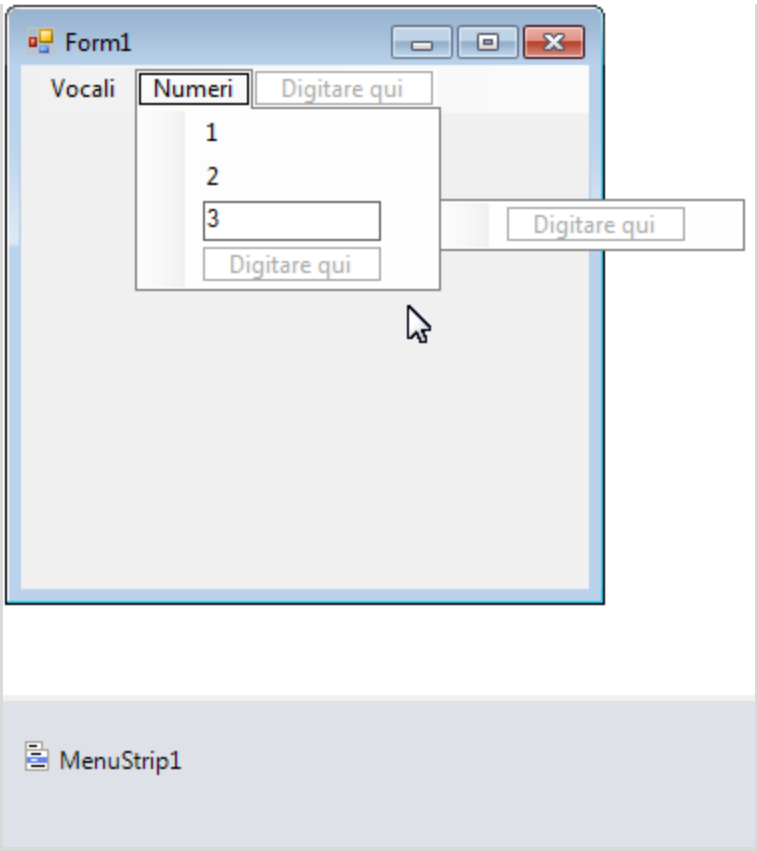
Apriamo un nuovo progetto e diamogli il nome **Strisce di menu e pulsanti**. Collochiamo nel form il componente **MenuStrip1**.
Iniziamo a scrivere il primo menu cliccando il riquadro **Digitare qui** e scriviamo gli item come in questa immagine:



Nel caso di errori, un item può essere cancellato in questo modo: *clic* con il tasto **destro** del mouse, poi *clic* su **Cancella**.
Un item può essere rinominato o corretto facendovi un doppio *clic* **lentamente** con il mouse.
Per inserire una linea di separazione tra i sottomenu con le vocali e il sottomenu con il comando **ESCI**, facciamo un *clic* con il tasto destro del mouse sull'item **ESCI**, poi *clic* su **Inserisci** e *clic* su **Separator**:



Scriviamo ora il secondo menu con alcuni sottomenu nel cui testo compaiono i numeri da 1 a 3:



Terminata questa operazione, facciamo un *clic* su uno di questi item, ad esempio sul sottomenu ESCI e accediamo alla Finestra Proprietà, dove notiamo che questo item ha la proprietà **Text = ESCI** e la proprietà **Name = ESCIToolStripMenuItem**. Questo è un nome dato automaticamente da VB; allo stesso modo VB ha dato automaticamente un nome a tutti i sottomenu**:**

- **AToolStripMenuItem**
- **EToolStripMenuItem**
- **IToolStripMenuItem**

Si tratta di nomi piuttosto complessi da gestire nella scrittura delle procedure che faremo tra poco, per cui è preferibile correggere tutte le proprietà **Name** di questi sottomenu come segue:

- **mnuA**
- **mnuE**
- **mnuI**
- **mnuO**
- **mnuU**
- **mnuESCI**
- **mnu1**
- **mnu2**
- **mnu3**

Terminata la scrittura dei nomi dei sottomenu, facciamo un doppio *clic* sul primo sottomenu, l'item con la vocale A.
Accediamo così alla Finestra del Codice, in cui troviamo una procedura già impostata per accogliere le istruzioni relative al *clic* del mouse sull'item di nome **mnuA**:

```
    Private Sub mnuA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuA.Click

    End Sub
```

Nella riga centrale di questa procedura, per ora vuota, scriviamo questa riga di istruzioni:

```
    Private Sub mnuA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuA.Click

        Me.Text = "A come APE"

    End Sub
```

La riga inserita istruisce il programma affinché, quando avverte un *clic* del mouse sul sottomenu A, scriva nel titolo del form le parole "A come APE".
Procediamo allo stesso modo scrivendo le procedure relative a tutti gli item:

```
Public Class Form1

    Private Sub mnuA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuA.Click

        Me.Text = "A come APE"

    End Sub

    Private Sub mnuE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuE.Click

        Me.Text = "E come EDERA"

    End Sub

    Private Sub mnuI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuI.Click

        Me.Text = "I come IMBUTO"

    End Sub

    Private Sub mnuO_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuO.Click

        Me.Text = "O come OCA"

    End Sub

    Private Sub mnuU_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuU.Click
```

```
        Me.Text = "U come UVA"

    End Sub
```

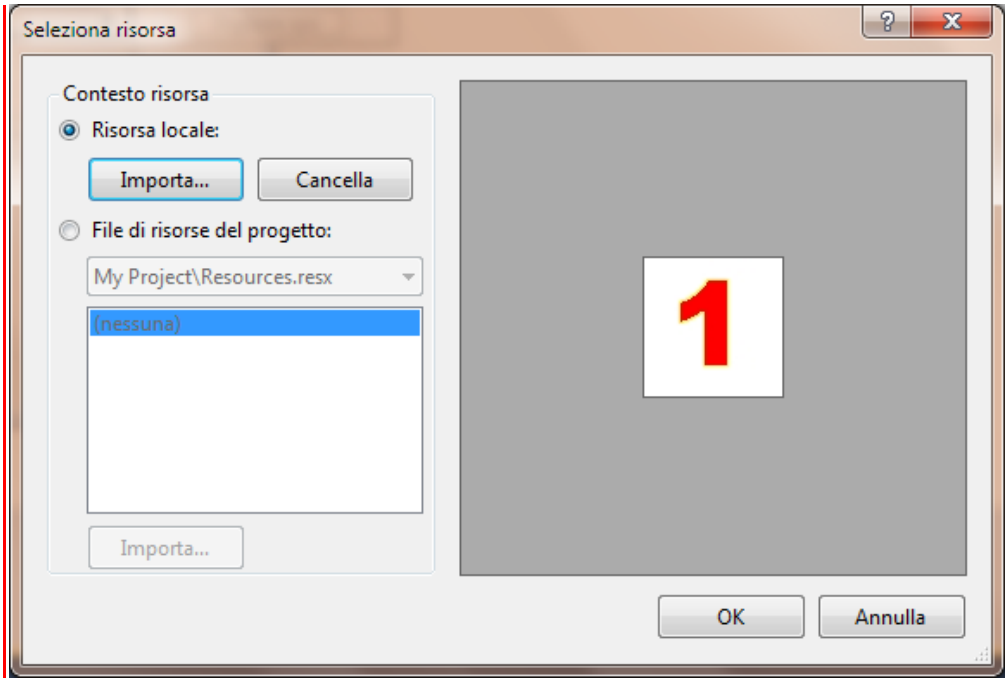
```
    Private Sub mnuESCI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuESCI.Click

        End

    End Sub
```

```
    Private Sub mnu1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnu1.Click

        Me.Text = "UNO"

    End Sub
```

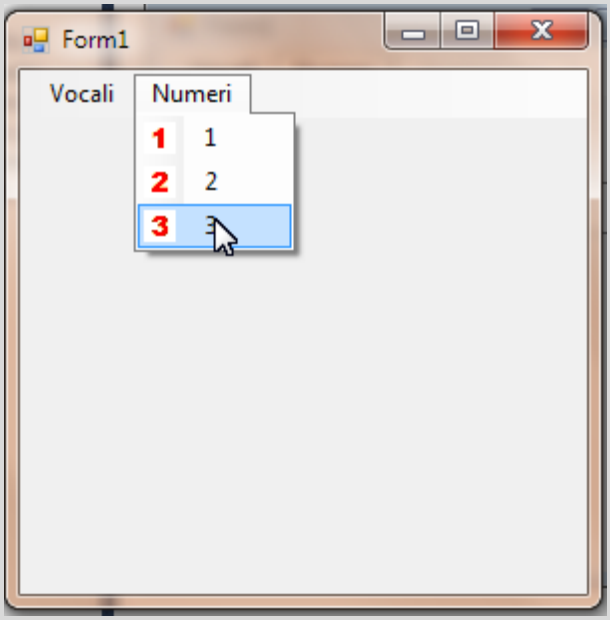
```
    Private Sub mnu2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnu2.Click

        Me.Text = "DUE"

    End Sub
```

```
    Private Sub mnu3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnu3.Click

        Me.Text = "TRE"

    End Sub

End Class
```

Notiamo che l'istruzione relativa all'item ESCI è semplicemente `End` (termina): un *clic* su questo sottomenu farà terminare il programma.
Ora mandiamo in esecuzione il tutto, per verificare gli effetti dei *clic* del mouse su questa striscia di menu.
Completiamo il programma inserendo tre immagini nei sottomenu con i numeri 1, 2 e 3.
Facciamo *clic* con il tasto destro del mouse sul sottomenu 1, poi *clic* su **Imposta immagine**. Accediamo così alla finestra **Selezione risorsa**, dove possiamo selezionare un'immagine adatta ad affiancare questo item:



Facciamo un *clic* sulla opzione **Risorsa locale** e sul pulsante **Importa…**
Andiamo alla ricerca della immagine 1.jpg, nella cartella **Documenti / A scuola con VB 2010 / Immagini / Numeri** e, dopo averla selezionata, facciamo un *clic* sul pulsante OK.
Procediamo allo stesso modo con la scelta di immagini per gli item 2 e 3, poi mandiamo in esecuzione il programma finito.
Ecco un'immagine del programma in esecuzione:



Prima di chiudere questo esercizio, preoccupiamoci di salvare il nostro lavoro con il nome **Strisce di menu e pulsanti** (clic sul pulsante **Salva tutto**), perché lo

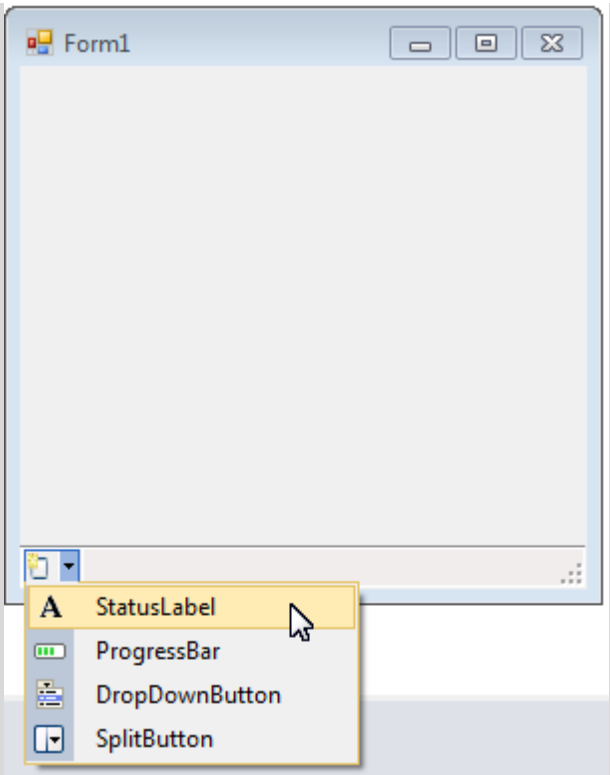
completeremo o con una striscia di pulsanti nel paragrafo dedicato al controllo **ToolStrip**.

52: Il controllo StatusStrip.

Il controllo **StatusStrip** (striscia di stato) crea una striscia, di solito nella parte inferiore del form, nella quale il programmatore visualizza informazioni di varia natura, che possono interessare l'utente ma che di solito non sono di vitale importanza per lo svolgimento del programma.
A questo scopo, la striscia creata con **StatusStrip** può contenere oggetti multiformi: pulsanti, riquadri di testo, barre di avanzamento, elenchi di item…; si tratta di oggetti simili ai controlli omonimi che si trovano nel form.
Le modalità di inserimento di oggetti in un controllo **StatusStrip** sono simili a quelle per la creazione di una striscia di menu con il controllo MenuStrip.
Vedremo nel prossimo esercizio l'uso di un controllo **Label** come riquadro di testo, all'interno di una **StatusStrip**, per visualizzare la data e l'ora correnti.

Esercizio 24: Il controllo Status Strip.

Apriamo un nuovo progetto e diamogli il nome **Striscia di stato**.
Inseriamo nel Form1 il controllo **StatusStrip1**.
Inseriamo nella striscia di stato, nella parte inferiore del form, un controllo **StatusLabel** (etichetta di testo):



Vediamo che questa StatusLabel assume automaticamente il nome **ToolStripStatusLabel1**.
Modifichiamo le sue proprietà:

- **Name = lblData**
- **Text = Data**

Nella Casella degli Strumenti, gruppo dei Componenti, facciamo un doppio *clic* sul componente **Timer** (temporizzatore).
Il Timer inserito nel progetto va a collocarsi nell'area sotto al Form1, di fianco al controllo StatusStrip1.
Facciamo un *clic* con il mouse sul componente Timer e nella Finestra Proprietà impostiamo queste sue proprietà:

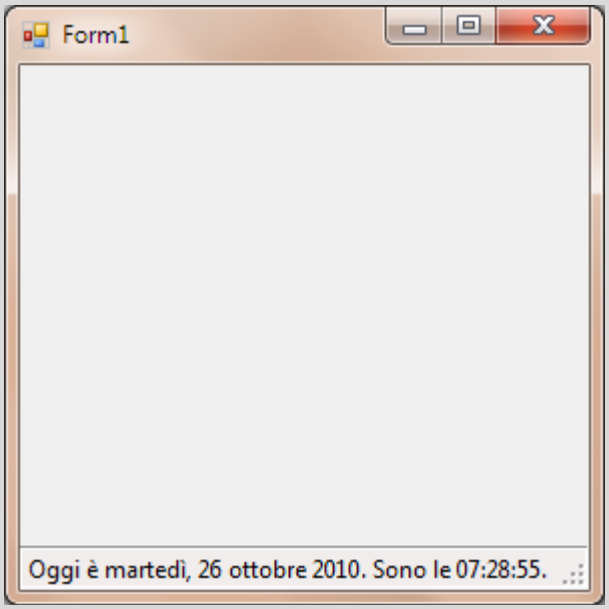
- **Enabled (attivato) = True** (Il timer sarà attivo da subito, appena il programma verrà avviato).
- **Interval = 1000** (Il timer manderà un segnale al programma, cioè creerà un evento, ogni 1000 millisecondi, vale a dire a ogni secondo).

Facciamo un doppio *clic* con il mouse sul componente Timer1, sotto il Form1 e accediamo così alla Finestra del Codice.
Qui troviamo già impostata una procedura finalizzata a gestire l'evento del tic del Timer1, allo scadere di ogni secondo.
La riga centrale di questa procedura è vuota e in attesa di istruzioni.
Completiamo la procedura in questo modo:

```
Public Class Form1
```

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        lblData.Text = "Oggi è " & Format(Now, "dddd, dd MMMM yyyy. ") & "Sono le
" & Format(Now, "hh:mm:ss.")

    End Sub

End Class
```

La riga centrale contiene le istruzioni affinché a ogni tic del Timer1 (ogni secondo) il programma scriva nel riquadro della lblData la data e l'ora correnti.
Si tratta di istruzioni piuttosto complesse perché per visualizzare la data e l'ora è necessario istruire il programma su come formattare questi dati.
La funzione **Format(Now, " ")** assolve questo compito: tra le virgolette si trova un modello di visualizzazione della data al quale il programma si attiene ogni volta che scrive la data e l'ora all'interno della lblData.
Mandiamo in esecuzione il programma e vediamo il risultato nella striscia di stato in basso nel form, allo scoccare di ogni secondo:



E' possibile cambiare le modalità di visualizzazione della data e dell'ora cambiando le impostazioni della funzione Format.
Ecco alcuni esempi:

```
        lblData.Text = "Oggi è il " & Format(Now, "dd/MM/yyyy, dddd. ") & "Sono
le " & Format(Now, "hh:mm.")

        lblData.Text = "Data: " & Format(Now, "dd/MM/yyyy. ") & "Ora " &
Format(Now, "hh:mm.")
```

53: Il controllo ToolStrip.

Il controllo **ToolStrip** (striscia di pulsanti e di altri controlli) consente di inserire nel form, in modo molto semplice, una striscia formata da pulsanti e da altri controlli come etichette di testo, caselle di testo, liste di item…
In molti programmi ideati per il sistema Windows i comandi a disposizione dell'utente sono visualizzati in due modi: con una striscia di menu e con una striscia di pulsanti.
Anche l'ambiente di programmazione di VB si presenta con una striscia di menu in alto e con una o più strisce di pulsanti sotto i menu:

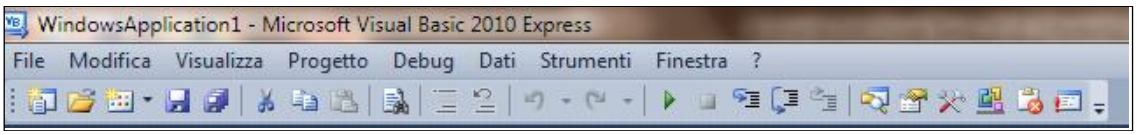
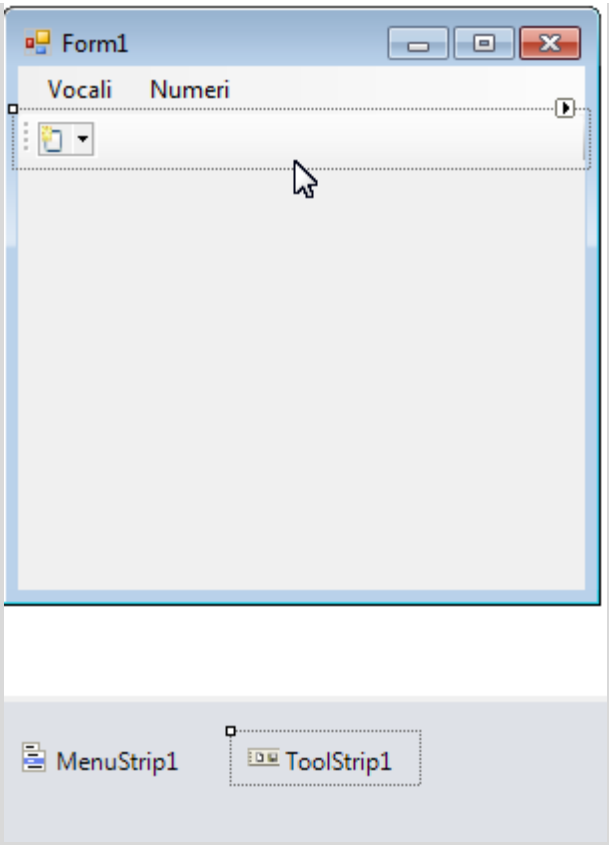


Figura 112: La striscia di menu e la striscia di pulsanti dell'ambiente di progettazione di VB.

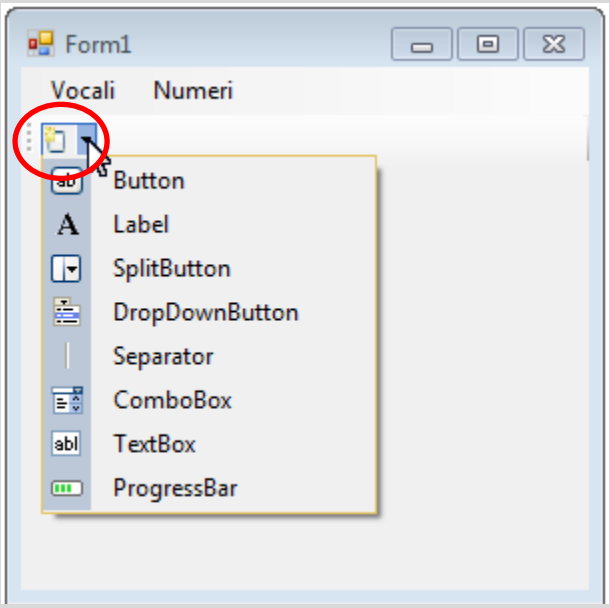
In una striscia di pulsanti, diversamente da quanto avviene in una striscia di menu, l'utente del programma può vedere contemporaneamente tutti i comandi di uso più frequente, senza dovere scorrere menu e sottomenu scritti.
Nel prossimo esercizio vedremo come inserire una striscia di pulsanti in un programma nel quale è già presente una striscia di menu.

Esercizio 25: Il controllo ToolStrip.

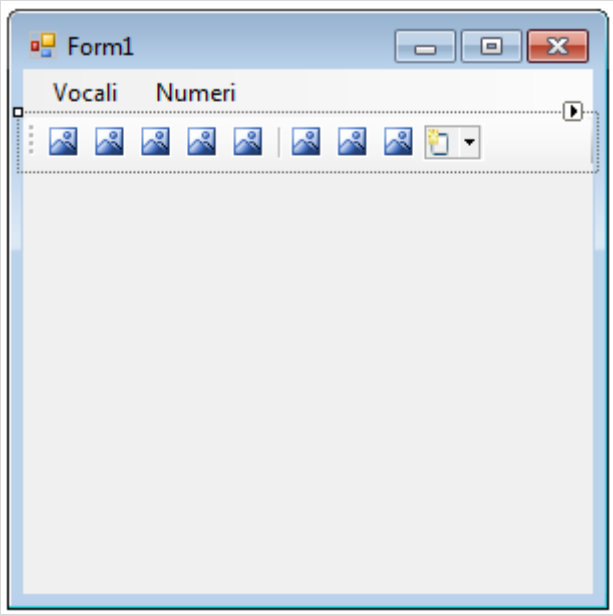
Riapriamo il progetto **Strisce di menu e di pulsanti**.
Lo abbiamo lasciato con la striscia di menu creati con il controllo **MenuStrip1**. Ora vi inseriamo anche il controllo **ToolStrip1**:



Il controllo **ToolStrip1** si colloca automaticamente sotto la striscia di menu già esistente.
Facendo un *clic* sul pulsante con la freccina a sinistra nel ToolStrip1, si apre un menu a tendina con l'elenco degli oggetti che è possibile inserire in questa striscia di pulsanti.



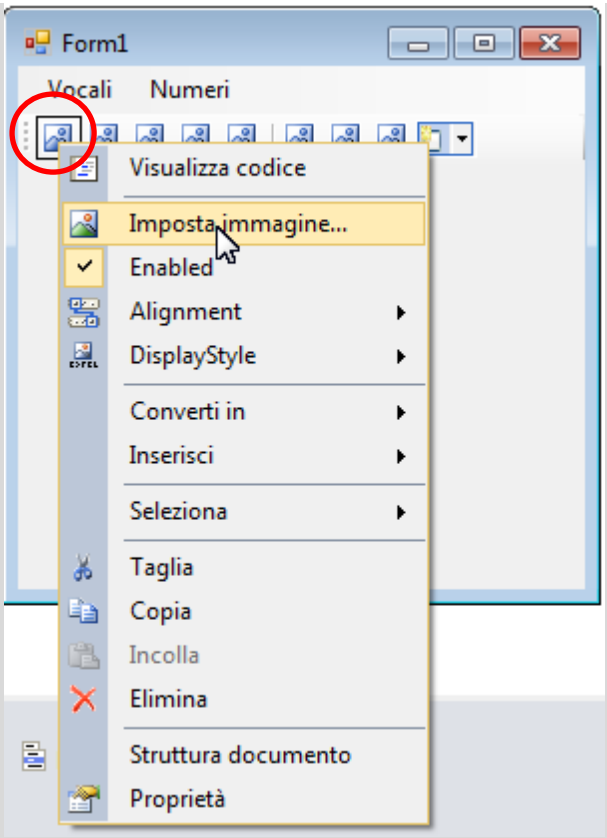
Procediamo inserendo cinque pulsanti Button (uno per ogni vocale), più un Separator, più altri tre pulsanti Button (uno per ogni numero):



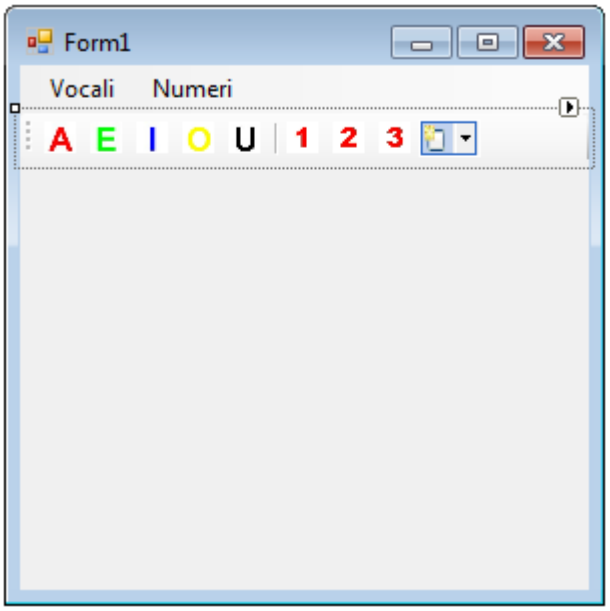
Ora assegniamo agli otto pulsanti, rispettivamente, queste proprietà:

- **Name = btnA Text = A**
- **Name = btnE Text = E**
- **Name = btnI Text = I**
- **Name = btnO Text = O**
- **Name = btnU Text = U**
- **Name = btn1 Text = 1**
- **Name = btn2 Text = 2**
- **Name = btn3 Text = 3**

Notiamo che questi pulsanti per certi aspetti non hanno il comportamento usuale dei pulsanti che si collocano nel form. Ad esempio, modificando la loro proprietà **Text** si modifica automaticamente la loro proprietà **ToolTipText**. Se andiamo a verificare, vediamo che le proprietà **ToolTipText** degli otto pulsanti sono ora identiche alle loro proprietà Text, cosa che non avviene con i pulsanti *normali* inseriti nei form.
Ora procediamo assegnando un'immagine a ognuno di questi otto pulsanti.
Facciamo *clic* con il tasto destro del mouse sul primo pulsante, poi *clic* su **Imposta immagine**:



Accediamo così alla finestra **Seleziona risorse**. Per questo primo pulsante selezioniamo l'immagine **vocale A.jpg**, che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini / Vocali**.
Procediamo selezionando per ogni pulsante, rispettivamente, le immagini **vocale E.jpg, vocale I.jpg, vocale O.jpg, vocale U.jpg**.
Le immagini per gli ultimi tre pulsanti, rispettivamente 1.jpg, 2.jpg e 3.jpg, si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Numeri**.
Abbiamo così completato la striscia di pulsanti in questo modo:



Ora abbiamo in questo programma una striscia di menu i cui item sono già collegati a procedure scritte nella Finestra del Codice, e una striscia di pulsanti per i quali ancora non abbiamo scritto alcuna istruzione.
Siccome in questo programma un *clic* del mouse su un menu ha lo stesso effetto di un *clic* sul pulsante corrispondente, possiamo utilizzare la stessa procedura, già scritta, per gestire i due eventi.
Facciamo un esempio: un *clic* sul menu Vocali / A (il cui nome è **mnuA**) ha lo stesso effetto di un *clic* sul primo pulsante (il cui nome è **btnA**), per cui la medesima procedura può gestire sia il *clic* del mouse sul **mnuA** che il *clic* sul **btnA**.
Apriamo la Finestra del Codice e scriviamo l' aggiunta che qui vediamo evidenziata, alla prima procedura:

```
    Private Sub mnuA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuA.Click, btnA.Click

        Me.Text = "A come APE"

    End Sub
```

Abbiamo scritto dunque una procedura il cui nome proprio è **mnuA_Click**, che ora gestisce (`Handles`) due eventi: **`mnuA.Click, btnA.Click`**.
Questo significa che, al verificarsi di uno di questi due eventi, la procedura eseguirà la riga di istruzioni centrale:

```
Me.Text = "A come APE"
```

Completiamo la scrittura di tutte le procedure aggiungendo a ognuna di esse l'evento del *clic* sul pulsante corrispondente.
Mandiamo in esecuzione il programma e notiamo che con un *clic* del mouse sui pulsanti si ottiene lo stesso effetto dei *clic* sui menu. Notiamo anche che al passaggio del mouse su un pulsante si visualizza la sua proprietà **ToolTipText**.

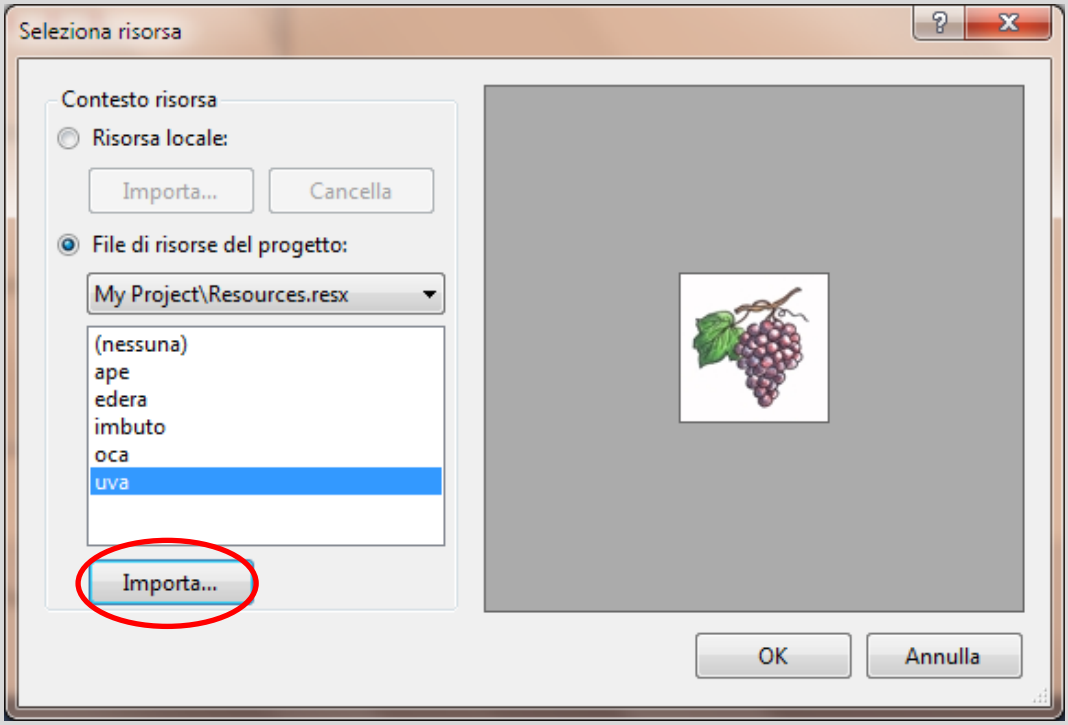
Per completare l'esercizio, inseriamo nel form un controllo **PictureBox** (riquadro di immagine) destinato a visualizzare un'immagine per ogni vocale cliccata:

- **A = ape**
- **E = edera**
- **I = imbuto**
- **O = oca**
- **U = uva**

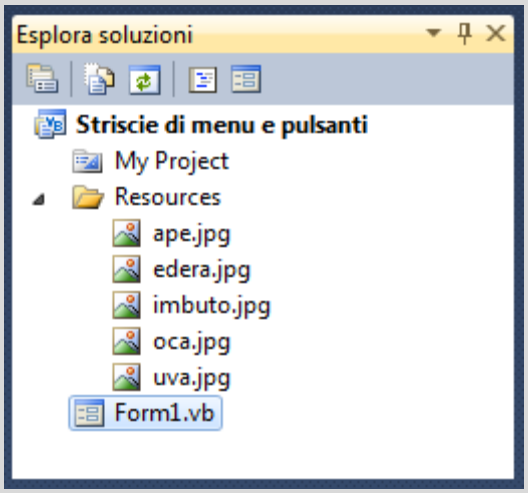
Fermiamo l'esecuzione del programma, se questo è in esecuzione, e inseriamo nel form un controllo **PictureBox** con queste proprietà:

- **Location = 100; 100**
- **Size = 80; 80**

Ora facciamo un *clic* sulla sua proprietà **Image** e accediamo alla finestra **Selezione risorsa**.
In questo programma non useremo una Risorsa locale, come abbiamo fatto negli esercizi precedenti, ma inseriremo le immagini necessarie nelle risorse del programma. Questo significa che le cinque immagini che useremo non verranno cercate di volta in volta nella cartella delle immagini, ma saranno allegate al programma; esse potranno essere facilmente prelevate dalle risorse, ogni volta che il programma lo richiederà.
Facciamo un *clic* dunque su **File di risorse del progetto**, poi un *clic* sul pulsante **Importa…** e inseriamo nelle risorse del programma le cinque immagini di cui abbiamo parlato pocanzi (ape, edera, imbuto, oca, uva), selezionandole nella cartella **Documenti / A scuola con VB 2010 / Immagini / Vocali**.



Terminata questa operazione, notiamo che nella finestra **Esplora soluzioni** è ora disponibile una cartella di Risorse che contiene le cinque immagini:



La progettazione grafica del programma è così terminata; restano da scrivere le istruzioni affinché il programma quando avverte un *clic* su un menu o su un pulsante, visualizzi all'interno del controllo PictureBox1 l'immagine corrispondente.
Apriamo la Finestra del Codice e torniamo a considerare la prima procedura, aggiungendovi una riga di istruzioni:

```
Private Sub mnuA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuA.Click, btnA.Click

        Me.Text = "A come APE"
        PictureBox1.Image = My.Resources.ape

    End Sub
```

Le nuova riga di istruzioni dice questo:

- quando avverti un *clic* con il mouse sul menu mnuA o sul pulsante bntA;
- assegna al controllo PictureBox1 l'immagine che si trova nelle risorse del programma, con il nome **ape**.

Inseriamo questa nuova riga in tutte le procedure e mandiamo in esecuzione il programma:

```
Public Class Form1

    Private Sub mnuA_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuA.Click, btnA.Click

        Me.Text = "A come APE"
        PictureBox1.Image = My.Resources.ape

    End Sub
```

```
    Private Sub mnuE_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuE.Click, btnE.Click

        Me.Text = "E come EDERA"
        PictureBox1.Image = My.Resources.edera

    End Sub
```

```
    Private Sub mnuI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuI.Click, btnI.Click

        Me.Text = "I come IMBUTO"
        PictureBox1.Image = My.Resources.imbuto

    End Sub
```

```
    Private Sub mnuO_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuO.Click, btnO.Click

        Me.Text = "O come OCA"
        PictureBox1.Image = My.Resources.oca

    End Sub
```

```
    Private Sub mnuU_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuU.Click, btnU.Click

        Me.Text = "U come UVA"
        PictureBox1.Image = My.Resources.uva

    End Sub
```

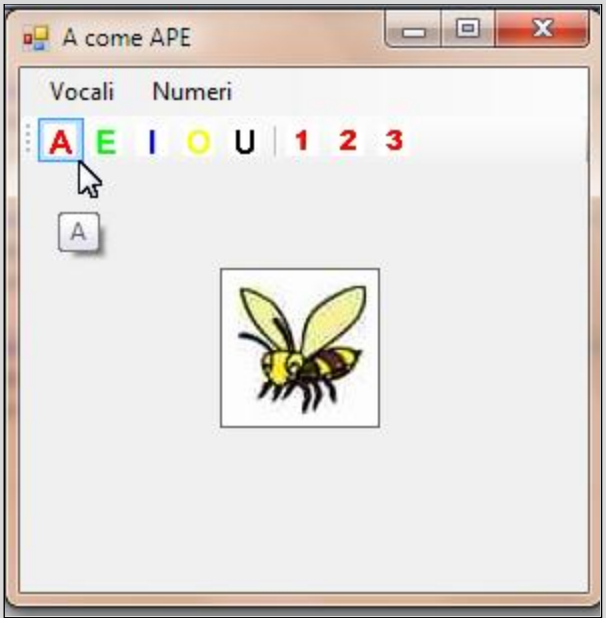
```
    Private Sub mnuESCI_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnuESCI.Click

        End

    End Sub
```

```
    Private Sub mnu1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnu1.Click, btn1.Click

        Me.Text = "UNO"
        PictureBox1.Image = btn1.Image

    End Sub
```

```
    Private Sub mnu2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnu2.Click, btn2.Click

        Me.Text = "DUE"
        PictureBox1.Image = btn2.Image

    End Sub
```

```
    Private Sub mnu3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles mnu3.Click, btn3.Click

        Me.Text = "TRE"
        PictureBox1.Image = btn3.Image

    End Sub

End Class
```

Notiamo che nelle ultime tre procedure la riga aggiunta relativa al PictureBox è diversa dalle altre, perché per i pulsanti con i numeri non abbiamo un'immagine corrispondente nelle risorse del programma.
Questi pulsanti tuttavia hanno già una loro immagine, che ne costituisce lo sfondo, per cui, per questi pulsanti, la riga di istruzione imposta come immagine del controllo PictureBox la stessa immagine del pulsante cliccato.
Ecco un'immagine del programma in esecuzione:

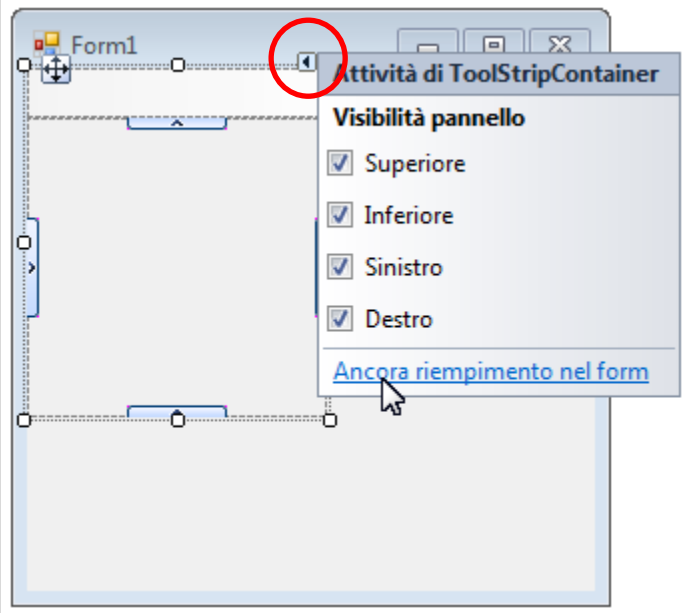


54: Il controllo ToolStripContainer.

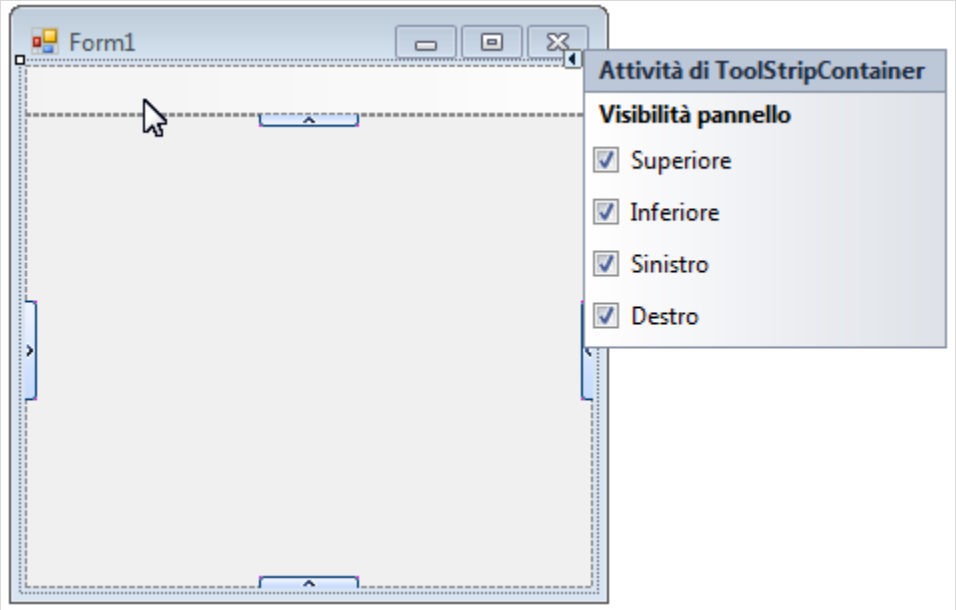
Il controllo **ToolStripContainer** (contenitore di strisce di pulsanti) crea sui quattro lati del form quattro aree riservate nelle quali l'utente del programma può muovere e collocare come meglio crede le strisce di pulsanti create dal programmatore.
Vedremo alcuni esempi del suo impiego nel prossimo esercizio.

Esercizio 26: Il controllo ToolStripContainer.

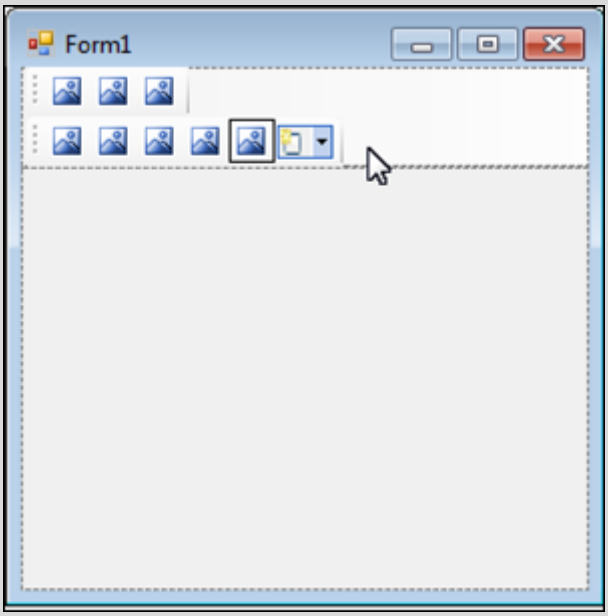
Apriamo un nuovo progetto e inseriamo nel form un controllo **ToolStripContainer**. Con un *clic* del mouse sulla freccia nera che si trova in alto a destra nel controllo, accediamo alla finestra delle **Attività di ToolStripContainer** e facciamo un *clic* sulla opzione **Ancora riempimento nel form**.



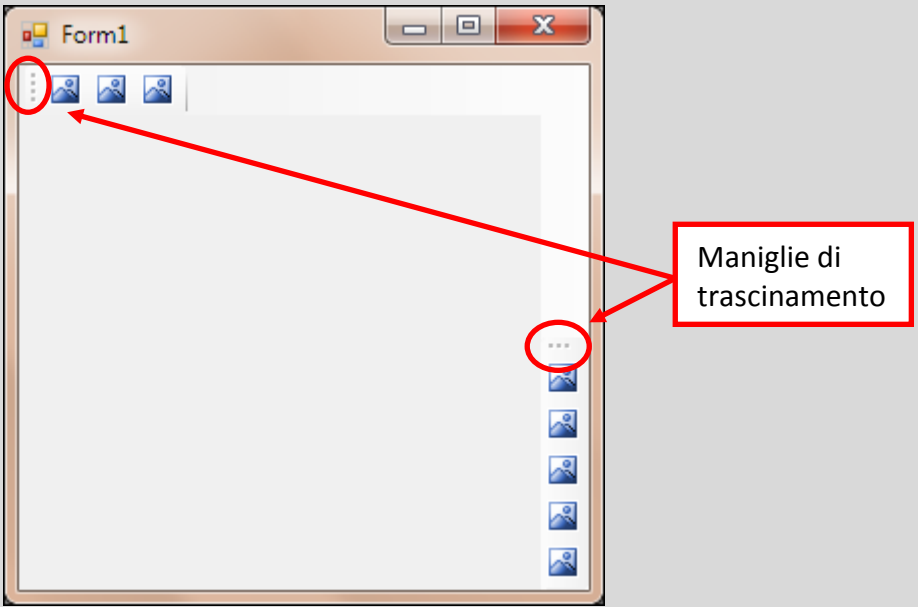
Con questa scelta, il controllo occupa tutto lo spazio disponibile nel form ed è pronto a visualizzare una striscia di menu sui quattro lati del form:



Ora inseriamo nel progetto due controlli ToolStrip: **ToolStrip1** e **ToolStrip2**, che vanno a collocarsi nella parte superiore del ToolStripContainer, uno sotto l'altro.
Inseriamo nel ToolStrip1 tre pulsanti Button e inseriamo nel ToolStrip2 cinque pulsanti Button, come in questa immagine:



Quando il programma è in esecuzione, notiamo che le due strisce di pulsanti, indipendenti una dall'altra, possono essere trascinate con il mouse, cliccando le loro maniglie di trascinamento, sui quattro lati del form e possono essere visualizzate in orizzontale o in verticale:



Capitolo 9: I COMPONENTI (CONTROLLI NON VISIBILI ALL'UTENTE).

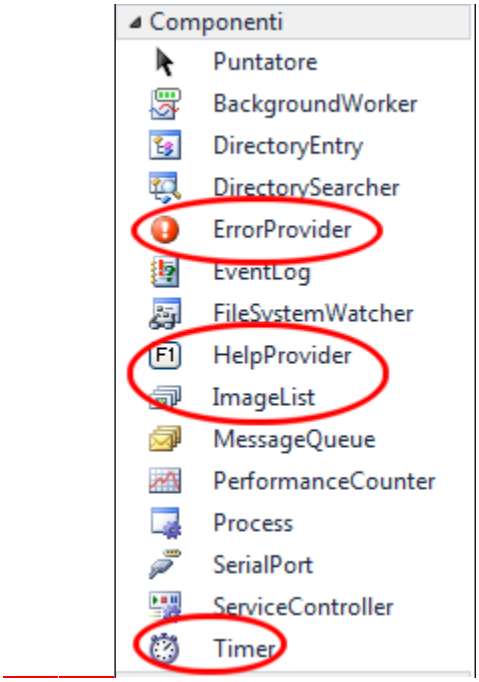


Figura 113: Il gruppo dei componenti nella Casella degli Strumenti.

Nel gruppo di componenti, nella Casella degli Strumenti, si trovano oggetti di uso piuttosto comune e oggetti il cui uso è invece del tutto particolare, generalmente finalizzato alla gestione e alla manutenzione del sistema operativo, di reti e/o di periferiche.
Ci limiteremo a vedere le modalità di funzionamento dei quattro componenti di uso più comune, evidenziati in rosso nella figura precedente:

- **ErrorProvider**
- **HelpProvider**
- **ImageList**
- **Timer**

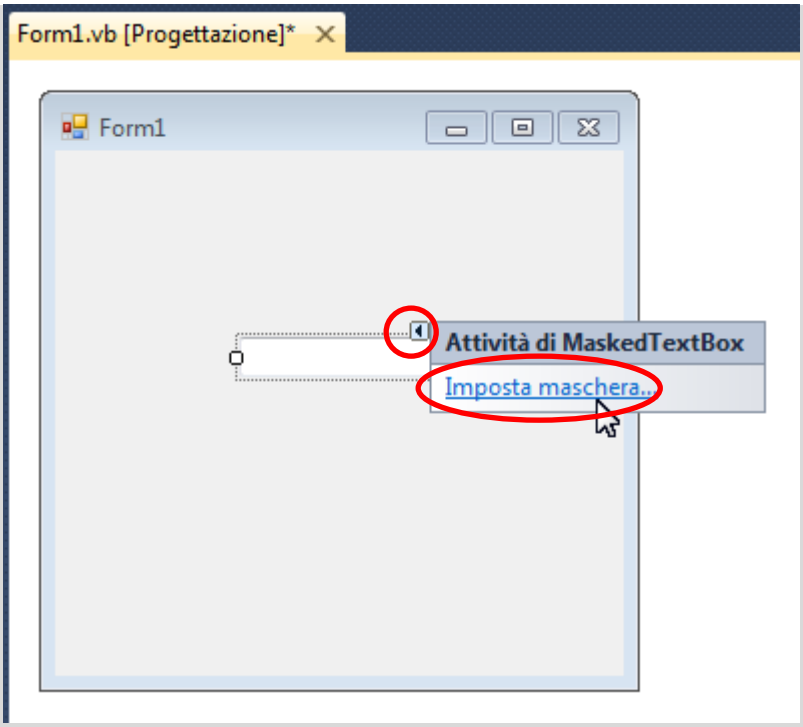
55: Il componente ErrorProvider.

Il componente **ErrorProvider** (segnalatore di errori) viene utilizzato per segnalare all'utente di un programma che ha commesso un errore in qualche operazione o qualche passaggio del programma.
Supponiamo che un programma chieda all'utente di scrivere una data, in una casella di testo, secondo il formato obbligatorio "**gg/mm/aaaa**"[39]: il componente **ErrorProvider** può essere utilizzato a guardia di tale formato, per segnalare all'utente eventuali errori commessi nella scrittura della data e per invitarlo a immettere la data nel formato richiesto.
La segnalazione dell'errore consiste nella visualizzazione di un bollino rosso con un punto esclamativo, di fianco al controllo in cui è stato commesso l'errore. Il bollino rosso può essere reso intermittente, impostandone la proprietà **Blink** nel modo desiderato.

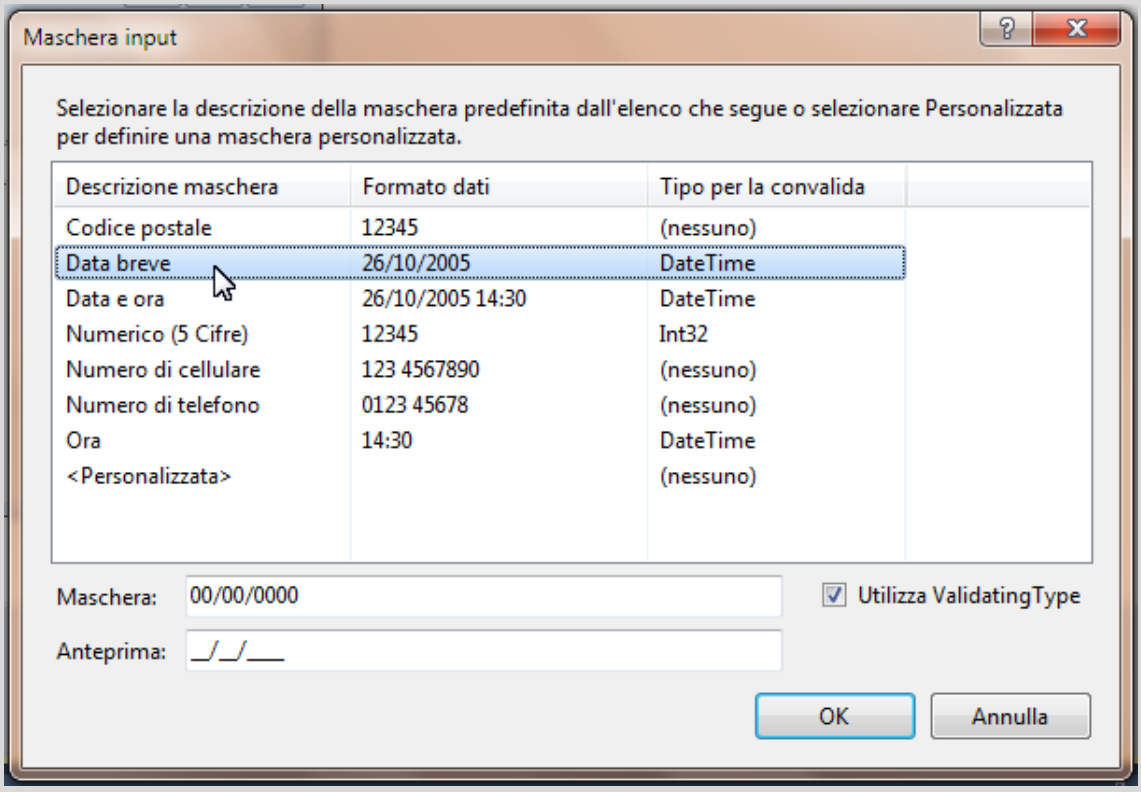
Esercizio 27: Il componente ErrorProvider e il controllo MaskedTextBox.

In questo esercizio vedremo all'opera un componente **ErrorProvider** collegato a un controllo **MaskedTextBox** (contenitore di testo a schema obbligato).
Il **MaskedTextBox** richiede l'immissione di una data di nascita secondo lo schema "**gg/mm/aaaa**" (due cifre per indicare il giorno, due cifre per il mese, quattro cifre per l'anno).
Questo **MaskedTextBox** è collegato a un componente **ErrorProvider**, che segnala un errore all'utente del programma, quando questi scrive una data con un formato diverso dallo schema obbligatorio.
Apriamo un nuovo progetto e inseriamo nel Form1 un controllo **MaskedTextEditor** e un componente **ErrorProvider**. Facciamo un *clic* con il mouse sul pulsante con la freccina nera in alto a destra nel **MaskedTextEditor** e poi un *clic* sul link **Imposta maschera**:

[39] Il formato "gg/mm/aaaa" indica che la data deve essere scritta con due cifre per il giorno, due cifre per il mese e quattro cifre per l'anno.

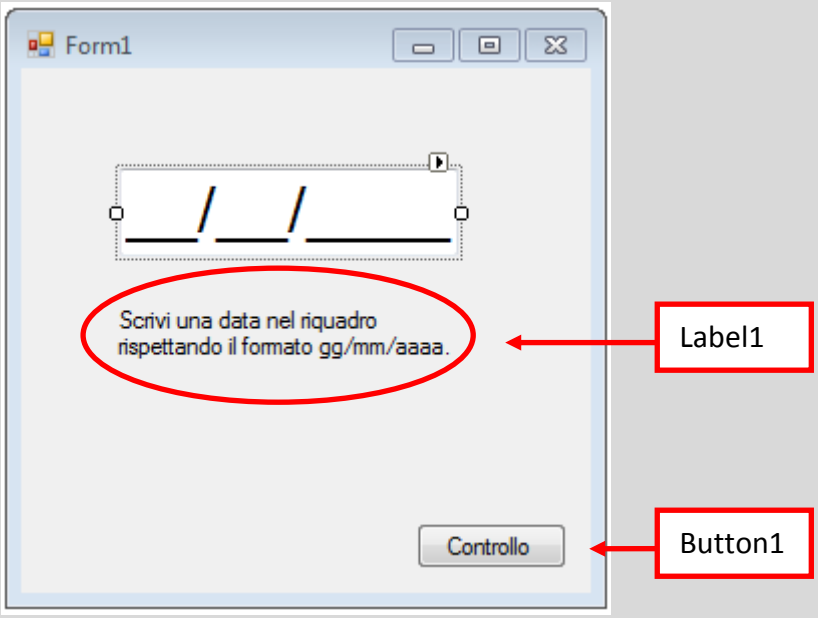


Accediamo così alla finestra **Maschera input**, dove facciamo un *clic* sull'impostazione **Data breve**:



La maschera che abbiamo scelto indica che nella casella di testo del MaskedTextBox l'utente potrà scrivere una data solo nel formato "gg/mm/aaaa".

Ora facciamo un *clic* sul controllo **MaskedTextBox** e accediamo alla finestra delle sue proprietà. Impostiamo la proprietà **Font = Microfoft Sans Serif a 24 punti**.
Inseriamo nel Form1 un controllo Label1 con la proprietà **Text** impostata come in questa immagine:



In basso a destra, inseriamo un pulsante Button, con la proprietà **Text = Controllo**.
Inseriamo infine nel programma un componente **ErrorProvider**, che andrà a collocarsi al di sotto del Form1 e non sarà visibile all'utente del programma.
Quando il programma sarà in esecuzione, l'utente potrà effettuare queste due operazioni:

1. scrivere una data nel contenitore di testo;
2. cliccare il pulsante **Controllo** per verificare se la data è stata scritta nel formato richiesto.

Per l'operazione di controllo della data e la visualizzazione del segnale di errore è necessario scrivere le istruzioni apposite nella Finestra del Codice.
Facciamo un doppio *clic* sul pulsante **Controllo**; accediamo così alla Finestra del Codice, dove troviamo già impostata la procedura che gestisce l'evento del *clic* del mouse su questo pulsante.
Nelle righe centrali di questa procedura vanno scritte le istruzioni che ci interessano.
La prima istruzione da scrivere è questa:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ErrorProvider1.SetError(MaskedTextBox1, "")

    End Sub
```

```
End Class
```

Questa prima riga ha lo scopo di eliminare eventuali messaggi di errori registrati in precedenza nel corso del funzionamento del programma.
La sua traduzione in linguaggio corrente: imposta il messaggio di errore (**SetError**) del componente **ErrorProvider1**, connesso alla casella di testo **MaskedTextBox1** uguale a **""** (cioè a **nulla**).
Notiamo che l'impostazione del messaggio di errore è scritta tra parentesi, con l'indicazione di due parametri:

1. **prima il controllo** al quale è associato il messaggio d'errore,
2. **poi il contenuto** del messaggio d'errore.

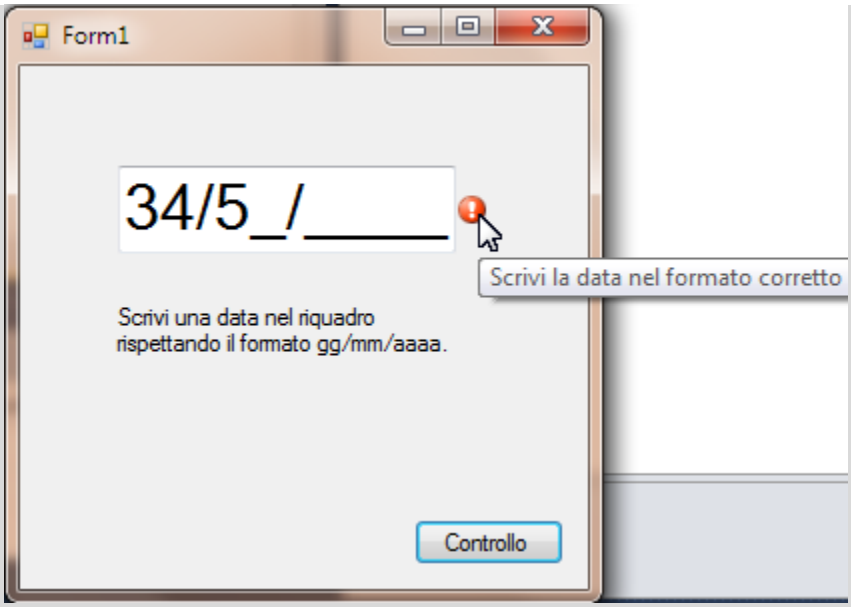
Dopo avere cancellato eventuali messaggi di errori precedenti, scriviamo le istruzioni affinché il programma controlli la data e segnali un messaggio di errore se questa non rispetta il formato obbligatorio:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ErrorProvider1.SetError(MaskedTextBox1, "")

        If MaskedTextBox1.MaskCompleted = False Then
            ErrorProvider1.SetError(MaskedTextBox1, "Scrivi la data nel formato
corretto")
        End If

    End Sub

End Class
```

Ecco la traduzione delle nuove istruzioni:

- **SE (If)** la maschera del **MaskedTextBox** non è stata completata (= **False**) **ALLORA (Then)**
- segnala l'errore relativo al **MaskedTextBox1** con questo testo: "Scrivi la data nel formato corretto").
- **FINE DELL'OPERAZIONE DI CONTROLLO (End If)**.

Mandiamo in esecuzione il programma e proviamo a scrivere alcune date incomplete, per visualizzare il messaggio di errore:

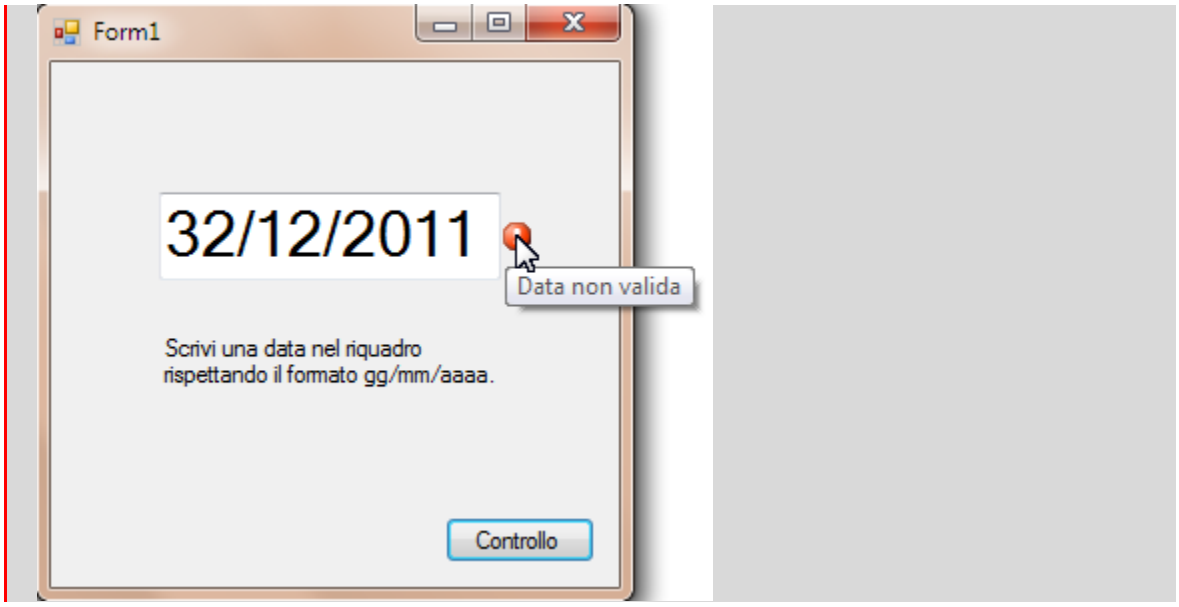


Il programma però esegue solo un controllo formale sulla data immessa, ma accetta testi che non sono date, come ad esempio: 35/60/2010.
Per eliminare anche questa possibilità di errore, aggiungiamo altre istruzioni per controllare la validità della data:

- **SE** (**If**) la data nel MaskedTextBox non è valida,
- **ALLORA** (**Then**) visualizza il messaggio di errore "Data non valida".

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ErrorProvider1.SetError(MaskedTextBox1, "")

        If MaskedTextBox1.MaskCompleted = False Then
            ErrorProvider1.SetError(MaskedTextBox1, "Scrivi la data nel formato
corretto")
        End If

        If IsDate(MaskedTextBox1.Text) = False Then
            ErrorProvider1.SetError(MaskedTextBox1, "Data non valida")
        End If

    End Sub

End Class
```

Ecco un'immagine del programma completo in esecuzione:



56: Il componente HelpProvider.

Il componente **HelpProvider** serve a visualizzare brevi messaggi di spiegazioni o di istruzioni (messaggi di *help*) connessi ai controlli presenti sul form; esso svolge dunque la funzione opposta a quella del componente ErrorProvider che abbiamo visto nel paragrafo precedente.
In fase di progettazione, il componente **HelpProvider** viene collegato a un controllo presente nel form, aggiungendo il messaggio di aiuto che si vuole mostrare all'utente. Quando l'utente del programma passa con il mouse sul controllo collegato a **HelpProvider** e preme il pusante **F1**, può visualizzare il messaggio di *help*.
Quando si inserisce un componente **HelpProvider** in un progetto, questi prende automaticamente il nome **HelpProvider1** e tutti i controlli già presenti nel form aggiungono alle loro proprietà la nuova proprietà **HelpString on HelpProvider1**.
Nella riga di questa proprietà, per ogni controllo che vogliamo connettere al componente **HelpProvider1**, deve essere scritta la frase che l'utente visualizzerà premendo il tasto **F1**.
Le due immagini che seguono mostrano, come esempio, un form con un componente **HelpProvider** e una serie di controlli **TextBox** in cui l'utente deve immettere dei dati. Il **TextBox** associato al **cognome** ha la proprietà **HelpString on HelpProvider1** impostata con questo testo: *Scrivi qui il tuo cognome.*
L'utente del programma, passando con il mouse sopra questo **TextBox** e premendo il tasto **F1**, può leggere il messaggio *Scrivi qui il tuo cognome*.

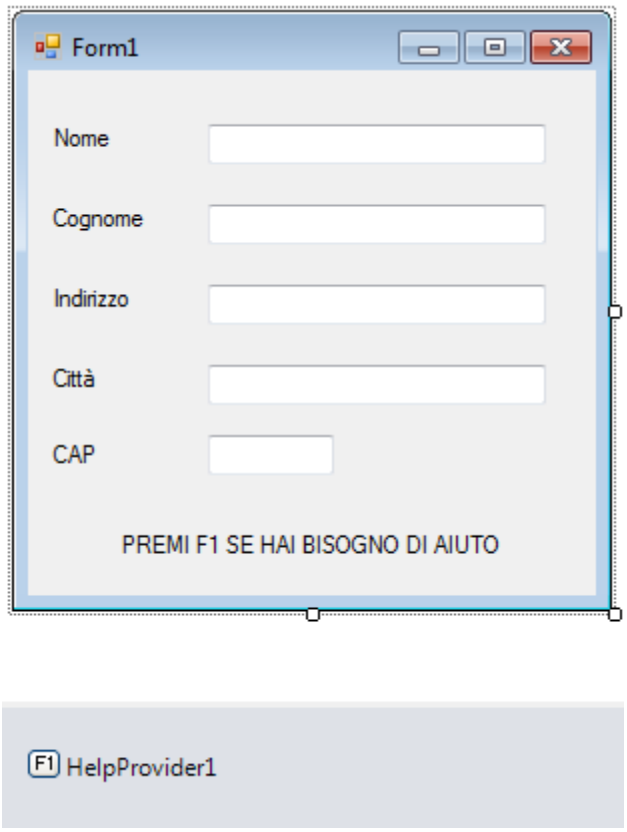


Figura 114: Il componente HelpProvider in fase di progettazione.

Figura 115: Il componente HelpProvider in fase di esecuzione.

Il componente **HelpProvider** può essere utilizzato, oltre che per visualizzare la semplice frase di aiuto che abbamo visto, anche per aprire, in una finestra separata, un

vero e proprio file di *help*, con un testo di dimensioni considerevoli, ad esempio un manuale o una guida con un indice di argomenti.
In questo caso, le proprietà **HelpString on HelpProvider1** dei vari controlli presenti sul form vanno lasciate vuote; bisogna invece impostare, nell'elenco delle proprietà del componente **HelpProvider**, la proprietà **HelpNamespace**.
Cliccando il pulsante con i tre puntini che si trova di fianco a questa proprietà, bisogna indicare il percorso completo del file che verrà visualizzato quando l'utente premerà il tasto **F1** durante l'esecuzione del programma.

57: Il componente ImageList.

Il componente **ImageList** consente di raccogliere in una unica lista tutte le immagini che verranno utilizzate in un programma. Questa lista di immagini può svolgere solamente la funzione di **raccoglitore** di immagini e non può essere utilizzata in alcun modo per mostrare immagini; anzi, il controllo stesso rimane **invisibile** quando si va in fase di esecuzione del programma.
Una lista di immagini mette semplicemente il suo contenuto a disposizione degli altri controlli che sono abilitati a mostrare delle immagini.
Questo componente svolge dunque una funzione simile a quella del file di risorse che abbiamo utilizzato nell'esercizio a pagina 256.
La differenza tra i due metodi sono queste:

- Il componente **ImageList** non accetta immagini di larghezza o altezza superiori a 256 pixel.
- Le immagini che si trovano nelle risorse del programma sono visualizzate nei vari controlli usando, per identificarle, il loro nome, mentre le immagini contenute in un **ImageList** sono visualizzate usando il numero d'ordine che occupano all'interno della lista. Questa caratteristica rende l'uso di un **ImageList** preferibile quando si debbono visualizzare immagini seguendo una numerazione progressiva.

In entrambi i casi (**ImageList** o **File di Risorse**) le immagini sono incorporate nel programma e salvate assieme ad esso, per cui:

- il caricamento delle immagini nella fase di esecuzione del programma è rapido e sicuro;
- la distribuzione delle immagini agli utenti del programma avviene in modo automatico, assieme al programma stesso, senza bisogno di operazioni aggiuntive.

L'**ImageList** è un componente, cioè un oggetto che non sarà visibile all'utente del programma; per questo, quando lo si inserisce in un form, esso va a collocarsi al di fuori dell'area del form, in basso:

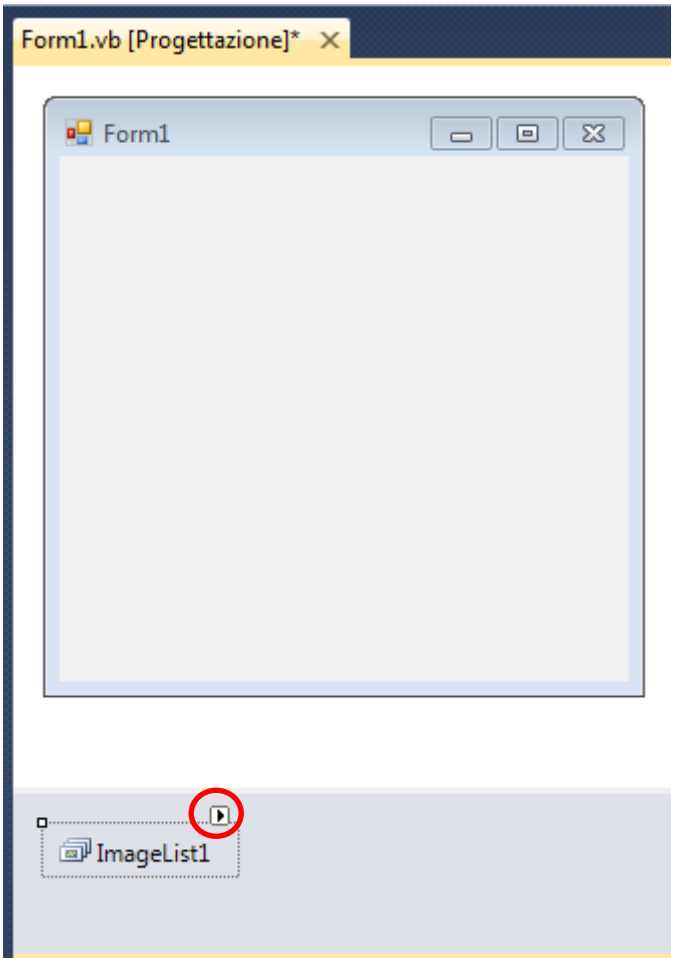


Figura 116: L'inserimento del componente ImageList in un form.

Facendo un *clic* sulla freccina in alto a destra nel componente si accede alla sua **Finestra delle Attività:**

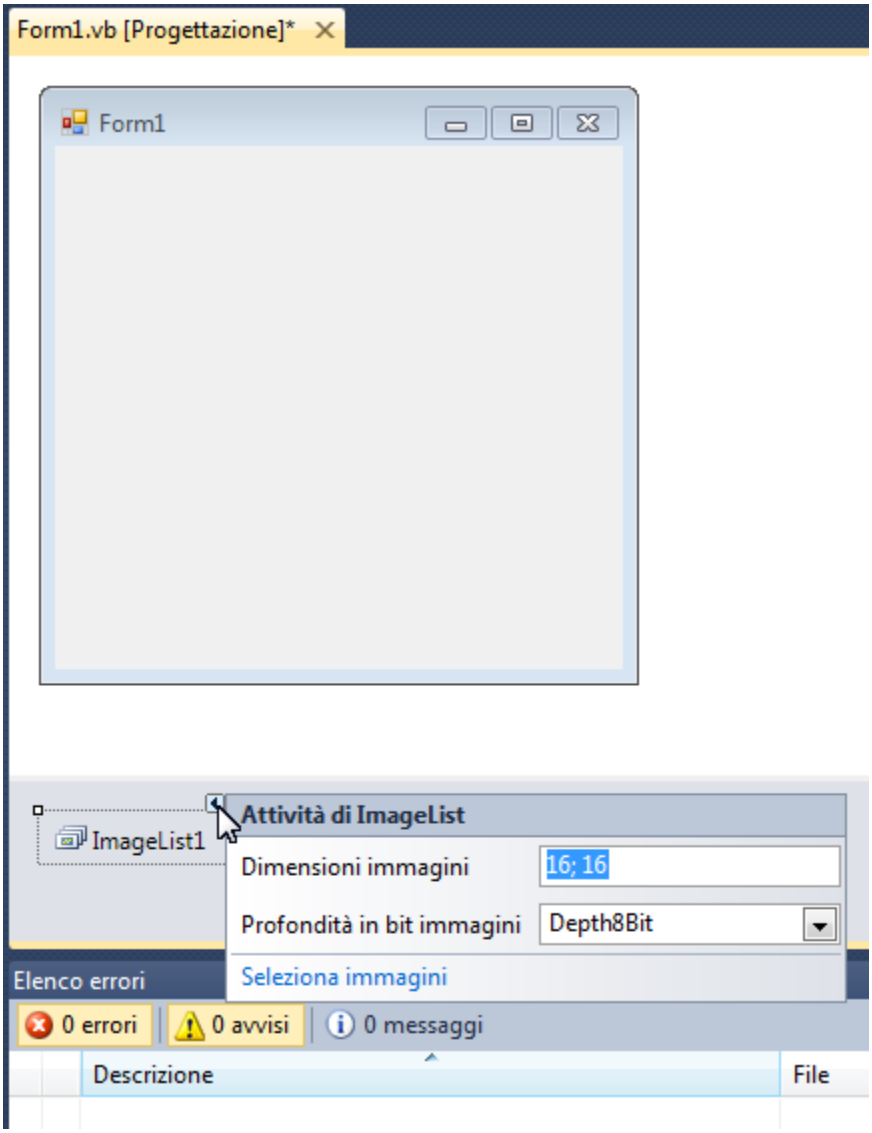


Figura 117: L'impostazione delle attività di un componente ImageList.

Qui possiamo impostare rapidamente le proprietà fondamentali di questo componente: il formato delle immagini, la loro qualità grafica e il loro elenco.

Nella casella **Dimensioni immagini** vanno indicate le dimensioni con le quali le immagini verranno immagazzinate nell'ImageList. Se si impostano le dimensioni a 120; 80 pixel, ad esempio, tutte le immagini verranno immagazzinate con queste dimensioni, a prescindere dalle loro dimensioni reali.
Se si vogliono creare liste di immagini con dimensioni diverse, è necessario creare un componente ImageList per ogni formato di immagini.
Le dimensioni massime delle immagini in una ImageList sono di 256 pixel di larghezza e 256 pixel di altezza.
Se si vogliono inserire in un progetto immagini di formato maggiore, o di formati diversi, e si desidera visualizzarle nelle loro dimensioni reali, è necessario inserirle nelle risorse del programma.

Il valore scritto nella casella **Profondità in bit immagini**[40] determina la qualità della loro visualizzzazione. E' opportuno farlo corrispondere alla profondità in bit delle immagini inserite nell'ImageList[41].

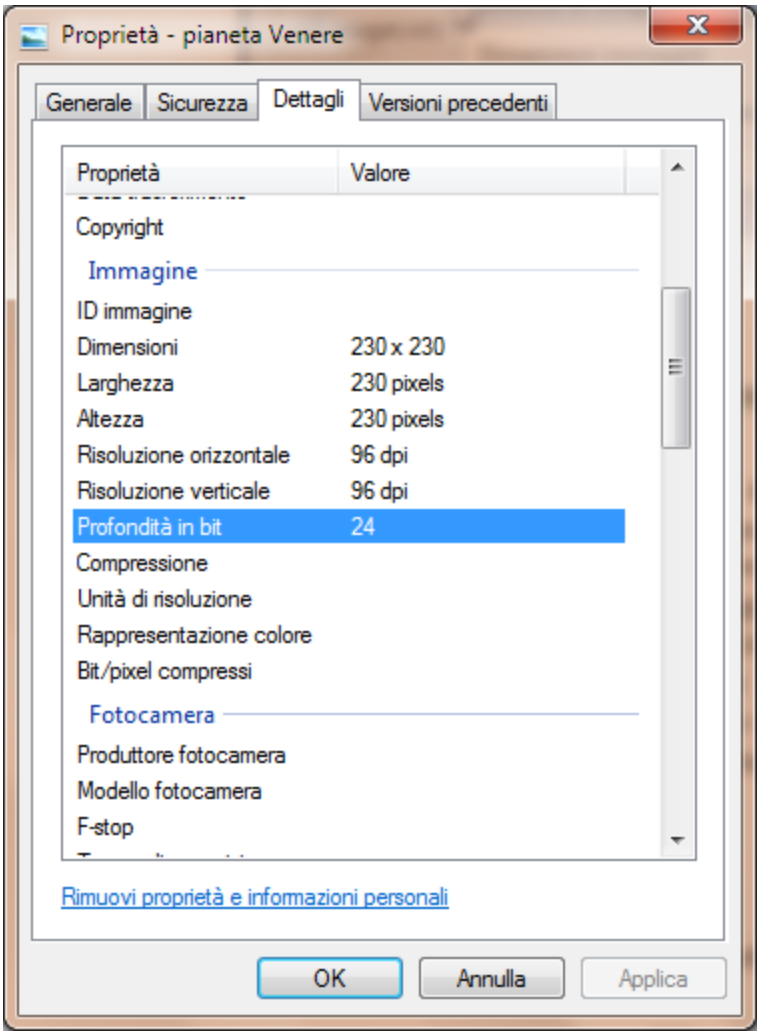


Figura 118: La Finestra Proprietà di un file grafico.

[40] La profondità in bit di un'immagine determina la quantità di informazioni sul colore disponibili per ogni pixel dell'immagine stessa. Maggiore è il numero bit di informazioni per pixel, maggiore è il numero di colori disponibili e più precisa è la rappresentazione dei colori. Ad esempio, un'immagine con una profondità di 1 bit può avere solo due colori per ogni pixel: il bianco o il nero. Un'immagine con una profondità di 8 bit avrà 2^8 possibilità, cioè 256, colori possibili per ogni pixel. Le immagini RGB (Red, Green, Blue) sono costituite da 3 canali di colore: il rosso, il verde e il blu. Un'immagine RGB a 8 bit ha quindi 256 valori possibili per ognuno dei tre canali di colori, vale a dire 256 x 256 x 256 combinazioni di colori per ogni pixel. Queste immagini RGB a 8 bit per canale (bpc) sono chiamate immagini a 24 bit (8 bit x 3 canali = 24 bit per ogni pixel).

[41] Questo valore può essere letto facendo un *clic* con il tasto destro del mouse sul file della immagine; menu Proprietà, scheda Dettagli.

Con un *clic* su **Seleziona immagini** si accede alla finestra dell'**Editor dell'insieme Images**, dove con semplici operazioni è possibile aggiungere, togliere o spostare immagini all'interno della lista.

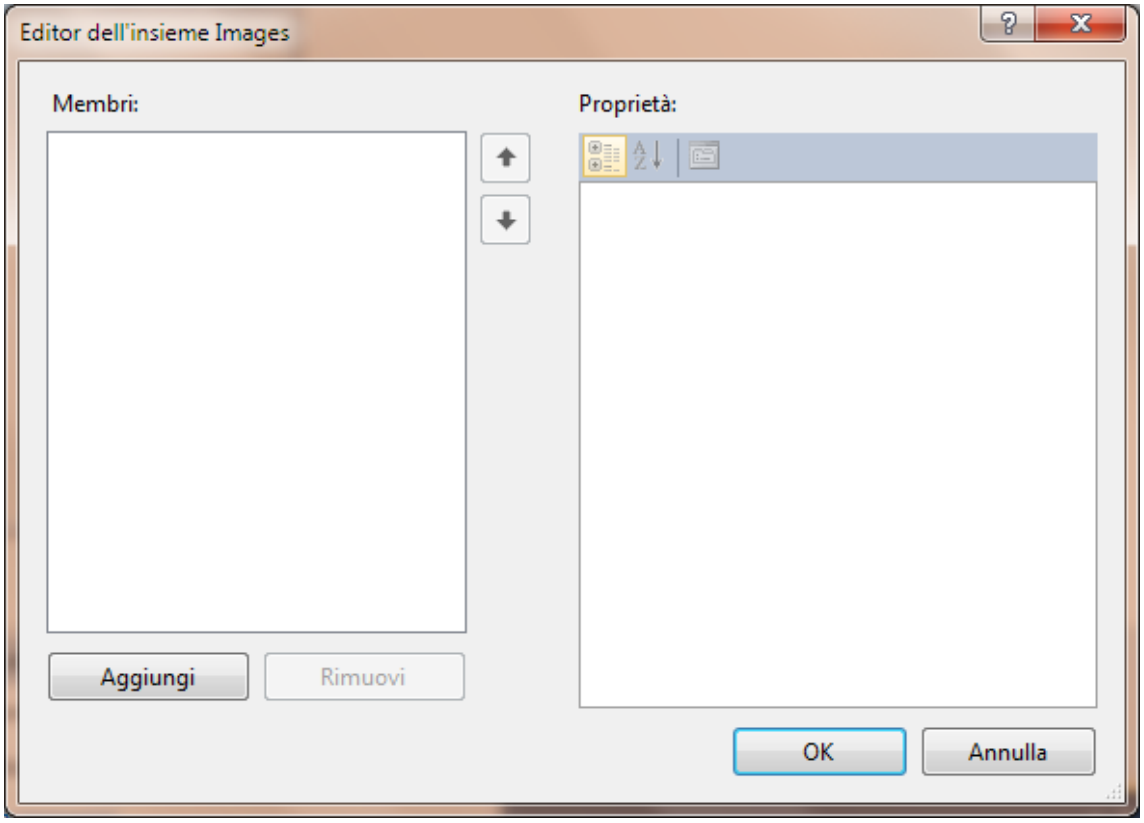


Figura 119: L'Editor dell'insieme Images di un componente ImageList.

Le tre operazioni che abbiamo visto in sequenza, e cioè

- Dimensioni immagini
- Profondità in bit immagini
- Seleziona immagini

possono essere effettuate anche dalla Finestra Proprietà del componente ImageList:

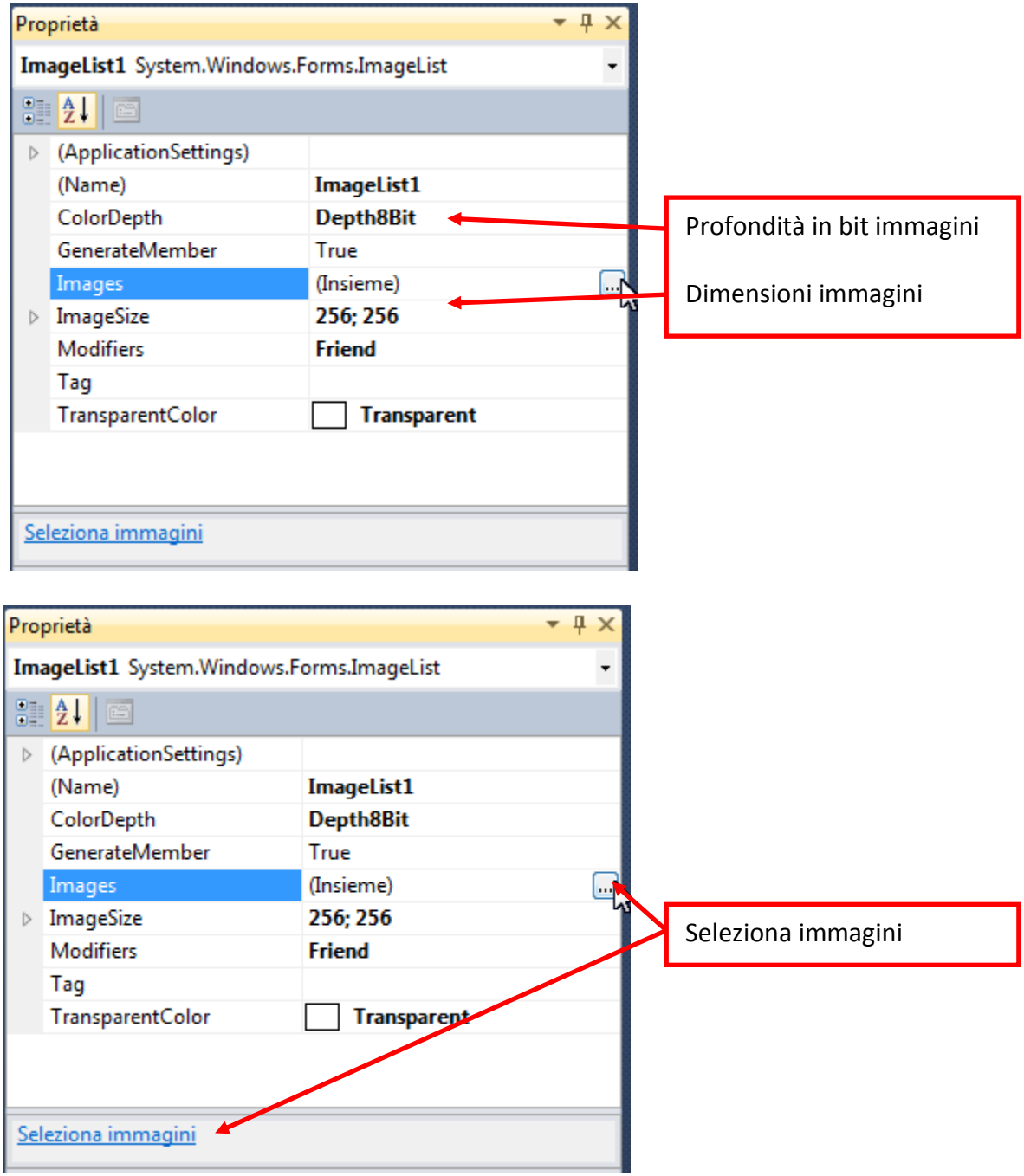


Figura 120: La Finestra Proprietà del componente ImageList.

Nella fase di esecuzione di un programma, per richiamare un'immagine dall'ImageList e visualizzarla, ad esempio, in un controllo PictureBox, è sufficiente fare riferimento al numero d'ordine che ha l'immagine all'interno dell'ImageList. In questo esempio, in un controllo PictureBox1 viene visualizzata l'immagine che si trova nel componente ImageList1 con il numero 12:

```
PictureBox1.Image = ImageList1.Images(12)
```

Ne vedremo un esempio nel prossimo esercizio.

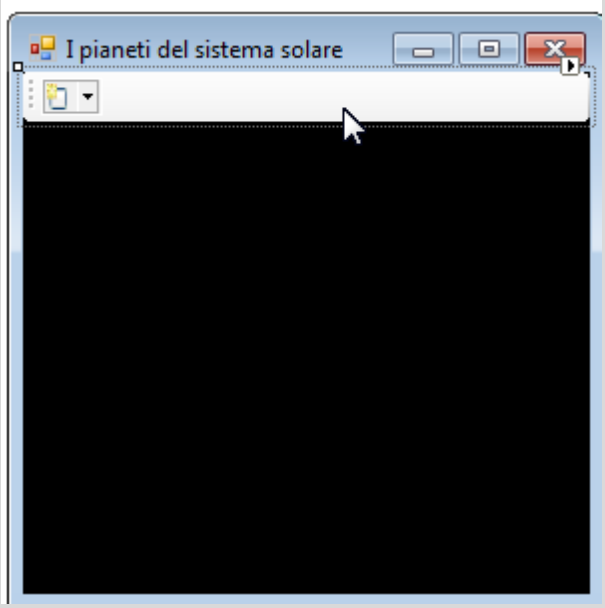
Esercizio 28: Il componente ImageList.

Vogliamo realizzare un programma dedicato ai pianeti del sistema solare.
Nel programma avremo una serie di otto pulsanti (uno per ogni pianeta) e un controllo PictureBox nel quale verranno via via visualizzate le immagini dei diversi pianeti.
Le immagini degli otto pianeti saranno immagazzinate in una lista di immagini; queste immagini verranno visualizzate in formato ridotto negli otto pulsanti e, a grandezza normale, in un controllo PictureBox.
Apriamo un nuovo progetto e diamogli il nome **Sistema solare**.

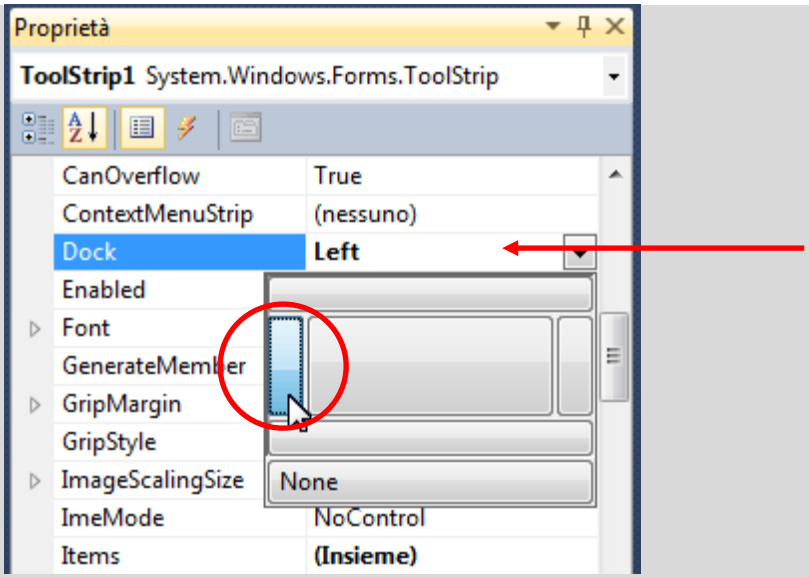
Facciamo un *clic* sul form e nella Finestra Proprietà impostiamo le sue proprietà

- **BackColor = Black**
- **Text = I pianeti del sistema solare**

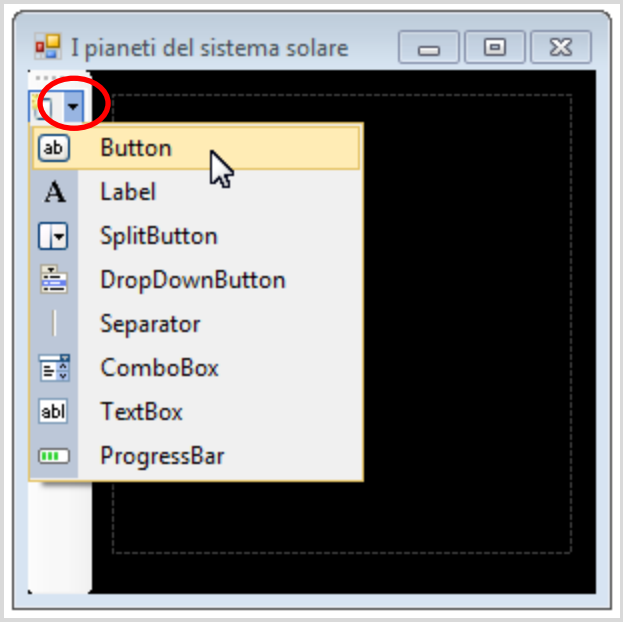
Ora inseriamo nel form un controllo **ToolStrip** (striscia di pulsanti):



Facciamo un *clic* sul controllo ToolStrip e, nella Finestra Proprietà, impostiamo la sua proprietà **Dock** (ormeggio) a sinistra, come in questa immagine:



Con questa impostazione, il ToolStrip va a collocarsi sul lato sinistro del form. Facciamo un *clic* sul pulsante con la freccina nera che si trova sul lato superiore del ToolStrip e inseriamo nella striscia di pulsanti un pulsante Button:



Questo è il primo pulsante di una serie di nove pulsanti (un pulsante per il sistema solare e un pulsante per ogni pianeta).
Impostiamo le proprietà di questo primo pulsante:

- **Name = Pulsante0**
- **Text = I pianeti del sistema solare**
- **Tag = 0**

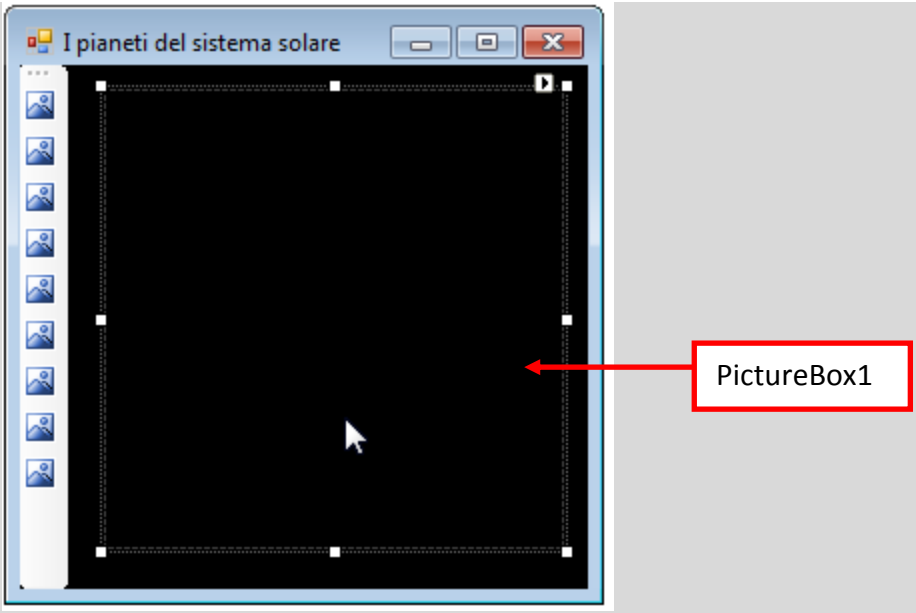
Ricordiamo che la proprietà **Tag** (cartellino) è simile a un segno di riconoscimento che viene assegnato a un controllo. In questo esercizio assegneremo alla porprietà Tag di ogni pulsante un numero, ma questo avrebbe potuto essere una parola, o una sigla.

La numerazione dei Tag di questi pulsanti servirà, durante l'esecuzione del programma, per identificare il pulsante premuto dall'utente e per visualizzare l'immagine con il numero corrispondente.
Procediamo completando una serie di nove pulsanti nella striscia di pulsanti, con queste proprietà:

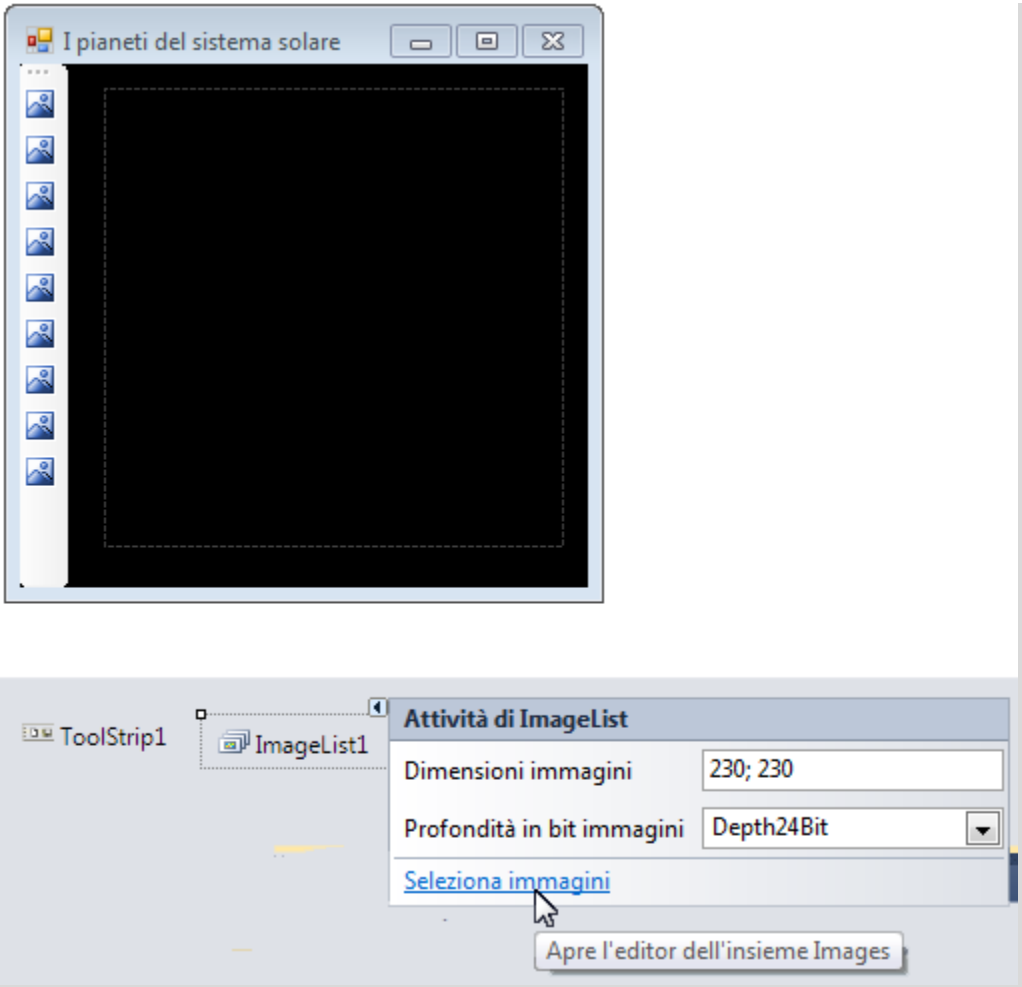
II pulsante	**Name = Pulsante1** **Text = Mercurio** **Tag = 1**
III pulsante	**Name = Pulsante2** **Text = Venere** **Tag = 2**
IV pulsante	**Name = Pulsante3** **Text = Terra** **Tag = 3**
V pulsante	**Name = Pulsante4** **Text = Marte** **Tag = 4**
VI pulsante	**Name = Pulsante5** **Text = Giove** **Tag = 5**
VII pulsante	**Name = Pulsante6** **Text = Saturno** **Tag = 6**
VIII pulsante	**Name = Pulsante7** **Text = Urano** **Tag = 7**
IX pulsante	**Name = Pulsante8** **Text = Nettuno** **Tag = 8**

Se si commette qualche errore ed è necessario eliminare un pulsante, è sufficiente cliccare il pulsante non desiderato e poi il pulsante **Taglia** (icona con le forbici).
Ora inseriamo nel form un controllo **PictureBox** con queste proprietà:

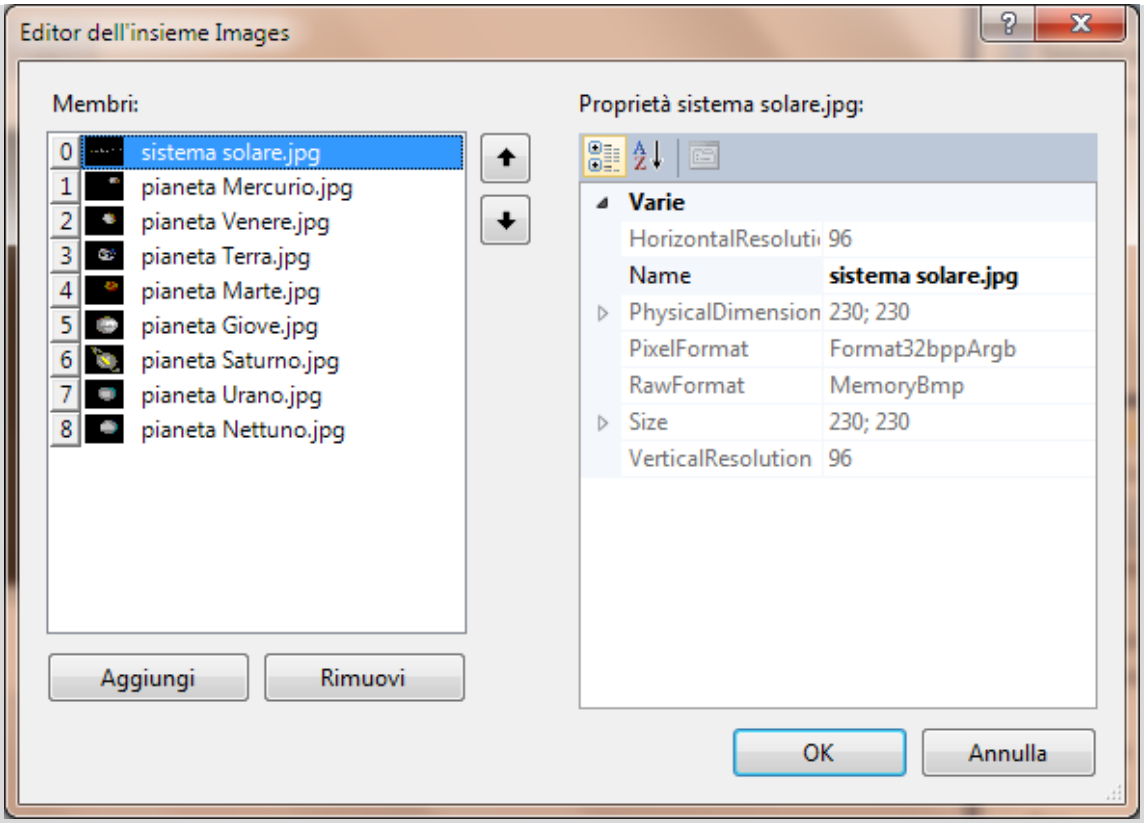
- **Location = 42; 12**
- **Size = 230; 230**



Inseriamo infine nel progetto un componente ImageList, destinato a contenere le immagini del sistema solare e degli 8 pianeti.
Trattandosi di un componente che non sarà visibile all'utente, l'ImageList si colloca al di fuori dell'area del Form1, in basso.
Facciamo un *clic* sulla freccina che si trova in alto a destra dell'ImageList e impostiamo le **dimensioni delle immagini** a 230 pixel di larghezza e 230 pixel di altezza, la **profondità in bit a 24 bit**, poi facciamo *clic* su **Seleziona immagini**:



Accediamo così **all'Editor dell'insieme Images**.
Cliccando il pulsante **Aggiungi**, inseriamo nell'ImageList le immagini del sistema solare e dei pianeti che si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Pianeti**:



Notiamo che le nove immagini sono numerate da 0 a 8 e non da 1 a 9, è per questo motivo che abbiamo numerato le proprietà Tag dei pulsanti da 0 a 8.
Terminato l'inserimento delle nove immagini facciamo un *clic* sul pulsante OK.
Ora non ci resta che scrivere le istruzioni, nella Finestra del Codice, affinchè il programma mostri l'immagine e il nome del pianeta quando l'utente farà un *clic* con il mouse su uno dei nove pulsanti.

Le prime istruzioni riguardano i pulsanti: vogliamo viusalizzare in formato ridotto, all'interno dei nove pulsanti, le immagini dei pianeti, prelevandole dal componente ImageList.

Facciamo un doppio *clic* sul Form1 e accediamo alla Finestra del Codice, con una procedura già impostata per gestire l'evento del caricamento del form (**MyBase.Load**) all'avvio del programma.
All'interno di questa procedura scriviamo i comandi necessari per inserire in ogni pulsante l'immagine del pianeta corrispondente:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Pulsante0.Image = ImageList1.Images(0)
        Pulsante1.Image = ImageList1.Images(1)
        Pulsante2.Image = ImageList1.Images(2)
        Pulsante3.Image = ImageList1.Images(3)
        Pulsante4.Image = ImageList1.Images(4)
        Pulsante5.Image = ImageList1.Images(5)
        Pulsante6.Image = ImageList1.Images(6)
```

```
        Pulsante7.Image = ImageList1.Images(7)
        Pulsante8.Image = ImageList1.Images(8)
    End Sub
```

Per visualizzare i nomi dei pianeti non occorre fare nulla: i nomi dei pianeti sono già associati alla proprietà Text di ogni pulsante e compariranno automaticamente al passaggio del mouse sopra ogni pulsante.
Scriviamo ora la procedura per la visualizzazione delle immagini dei pianeti all'interno del controllo PictureBox1.
Facciamo un *clic* sul primo pulsante in alto nella striscia di pulsanti.
Accediamo alla Finestra del Codice dove troviamo già impostata la procedura per gestire l'evento del *clic* su questo Pulsante0, che è il nome proprio del primo pulsante:

```
    Private Sub Pulsante0_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Pulsante0.Click

    End Sub
```

Facciamo attenzione, nella prima riga di questa procedura, alle ultime parole: questa procedura gestisce l'evento Pulsante0.Click. Noi sappiamo però che i pulsanti sono nove e che a ogni evento *clic* su ognuno di questi pulsanti l'azione che il programma dovrà effettuare sarà sempre la stessa: mostrare l'immagine di un pianeta nel controllo PictureBox1.
Possiamo dunque fare in modo che questa procedura si attivi a ogni *clic* su uno qualsiasi dei nove pulsanti, completando la prima riga in questo modo:

```
Private Sub Pulsante0_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Pulsante0.Click, Pulsante1.Click, Pulsante2.Click,
Pulsante3.Click, Pulsante4.Click, Pulsante5.Click, Pulsante6.Click,
Pulsante7.Click, Pulsante8.Click

    End Sub
```

Ora, esaminando sempre la prima riga della procedura, facciamo attenzione al parametro sender (mittente): quando si registra l'evento del *clic* su uno dei nove pulsanti, il programma non registra solo l'evento Click, ma registra anche chi ne è il mittente, cioè **da quale controllo** è partito l'evento.
Siccome a ogni pulsante è associata una proprietà Tag con un numero da 0 a 8, sapendo chi è il pulsante mittente dell'evento Click, possiamo sapere quale numero è associato a questo pulsante e dunque possiamo prelevare dall'ImageList e visualizzare nel PictureBox l'immagine con il numero d'ordine corrispondente:

```
Public Class Form1

    Private Sub Pulsante0_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Pulsante0.Click, Pulsante1.Click, Pulsante2.Click,
Pulsante3.Click, Pulsante4.Click, Pulsante5.Click, Pulsante6.Click,
Pulsante7.Click, Pulsante8.Click

        ' riserviamo una casella di memoria di tipo numerico (numeri interi)
        ' per registrare il numero del pulsante cliccato:
        Dim NumeroPulsante As Integer = sender.tag
```

```
        ' visualizziamo nel PictureBox1 l'immagine corrispondente al numero del
pulsante cliccato:
        PictureBox1.Image = ImageList1.Images(NumeroPulsante)

    End Sub

End Class
```

Nell'immagine seguente vediamo il programma in esecuzione:

Nel prossimo esercizio vedremo un esempio dell'uso di un componente **ImageList** (elenco di immagini) connesso a un controllo **ListBox**. In concreto, vedremo come alla scelta di un elemento all'interno del ListBox si può fare corrispondere la scelta di un'immagine all'interno dell'ImageList.

Esercizio 29: Bandiere (I).

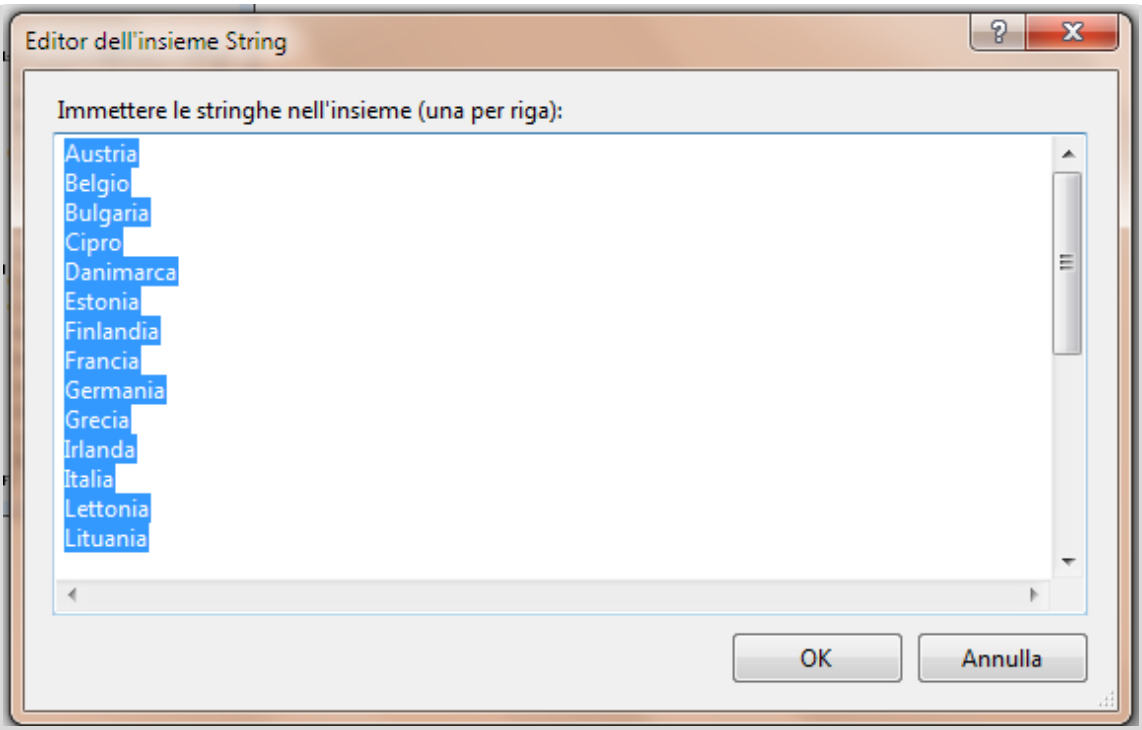
Obiettivo di questo esercizio è creare un programma che visualizzerà le bandiere delle 27 nazioni dell'Unione Europea, con l'aggiunta della bandiera dell'Unione.
Le immagini delle 28 bandiere saranno contenute in un componente **ImageList**.
L'elenco delle nazioni sarà scritto in un controllo **ListBox**.
Cliccando il nome della nazione nel controllo **ListBox**, verrà visualizzata la bandiera corrispondente, all'interno di in un controllo **PictureBox**.
Apriamo un nuovo progetto. Lo denominiamo **Bandiere Unione Europea**.
Nel Form1 inseriamo un controllo **ListBox** con queste proprietà:

- **Location = 163; 38**
- **Size = 109; 212**

Facciamo un *clic* sul pulsante che si trova nella riga della proprietà **Items (Insieme)** del controllo **ListBox**. Nella finestra che si apre, **Editor dell'insieme String**, dobbiamo scrivere l'elenco delle 27 nazioni dell'Unione Europea (più il nome dell'Unione Europea). Possiamo farlo rapidamente copiando questo elenco e incollandolo nella finestra[42]:

Austria
Belgio
Bulgaria
Cipro
Danimarca
Estonia
Finlandia
Francia
Germania
Grecia
Irlanda
Italia
Lettonia
Lituania
Lussemburgo
Malta
Paesi Bassi
Polonia
Portogallo
Regno Unito
Repubblica Ceca
Romania
Slovacchia
Slovenia
Spagna
Svezia
Ungheria
Unione Europea

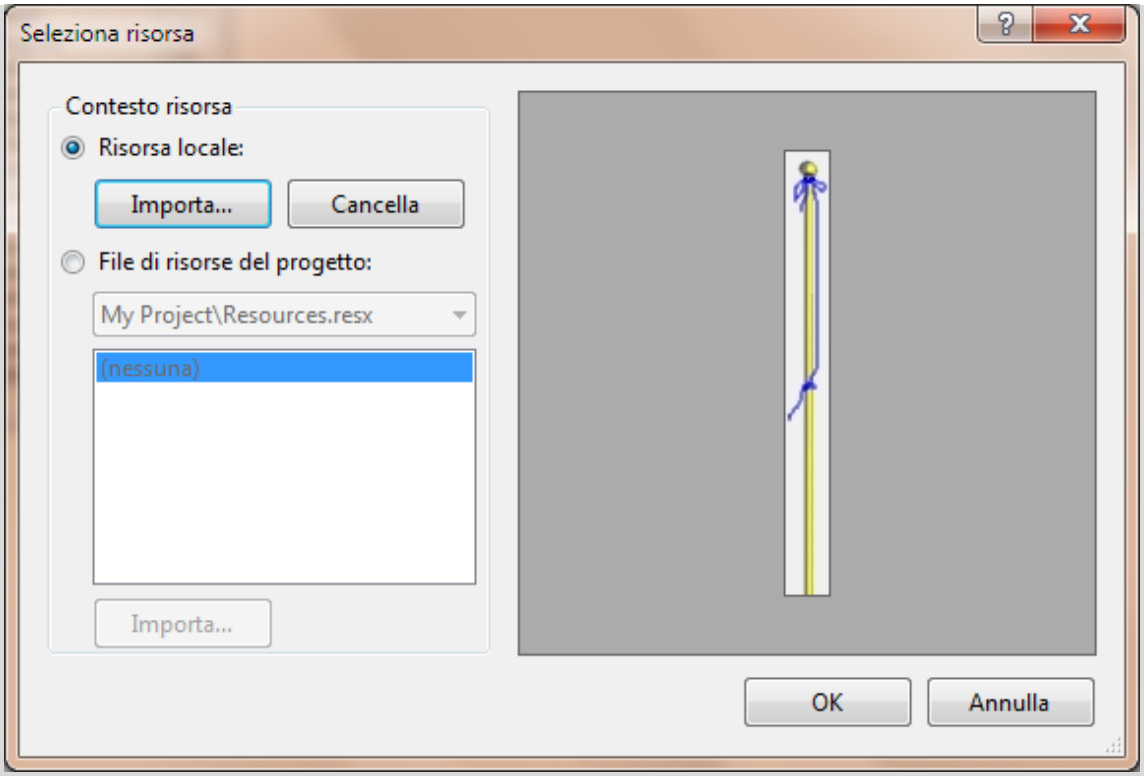
[42] Lo stesso elenco si trova nel file **Nazioni europee.txt**, nella cartella **Documenti \ A scuola con VB \ Testi**.



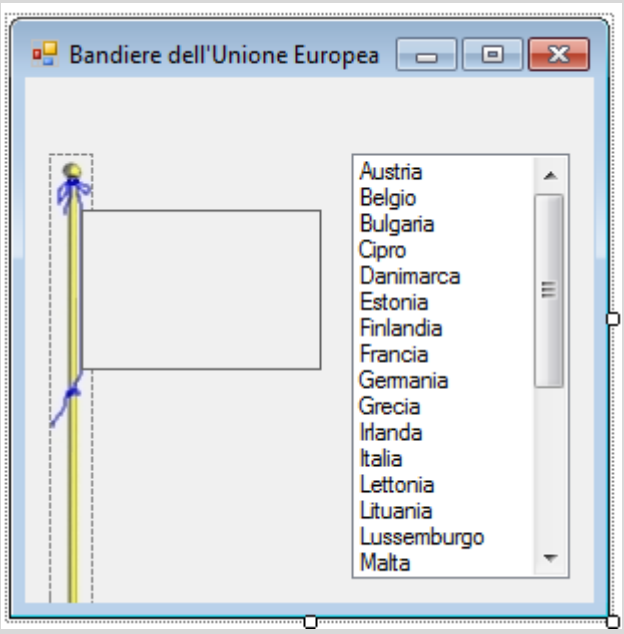
Premiamo il tasto OK; con questo abbiamo terminato il lavoro di preparazione del controllo **ListBox**.
Ora inseriamo nel Form1 due controlli **PictureBox**, con queste proprietà:

PictureBox1:
- **BorderStyle = FixedSingle**
- **Location = 28; 66**
- **Size = 120; 80**

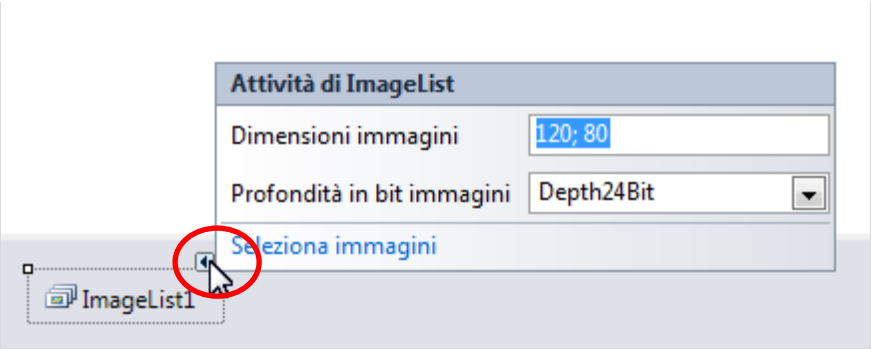
PictureBox2
- **Location = 12; 38**
- **Size = 22; 225**
- **Image**: facciamo un *clic* sul pulsante con i tre puntini. Accediamo alla finestra **Selezione risorsa**, facciamo un *clic* su **Risorsa** locale e andiamo a scegliere l'immagine **asta_bandiera.jpg** che si trova nella cartella **Documenti \ A scuola con VB \ Immagini \ Bandiere**.



Ecco come appare ora l'interfaccia del nostro programma:



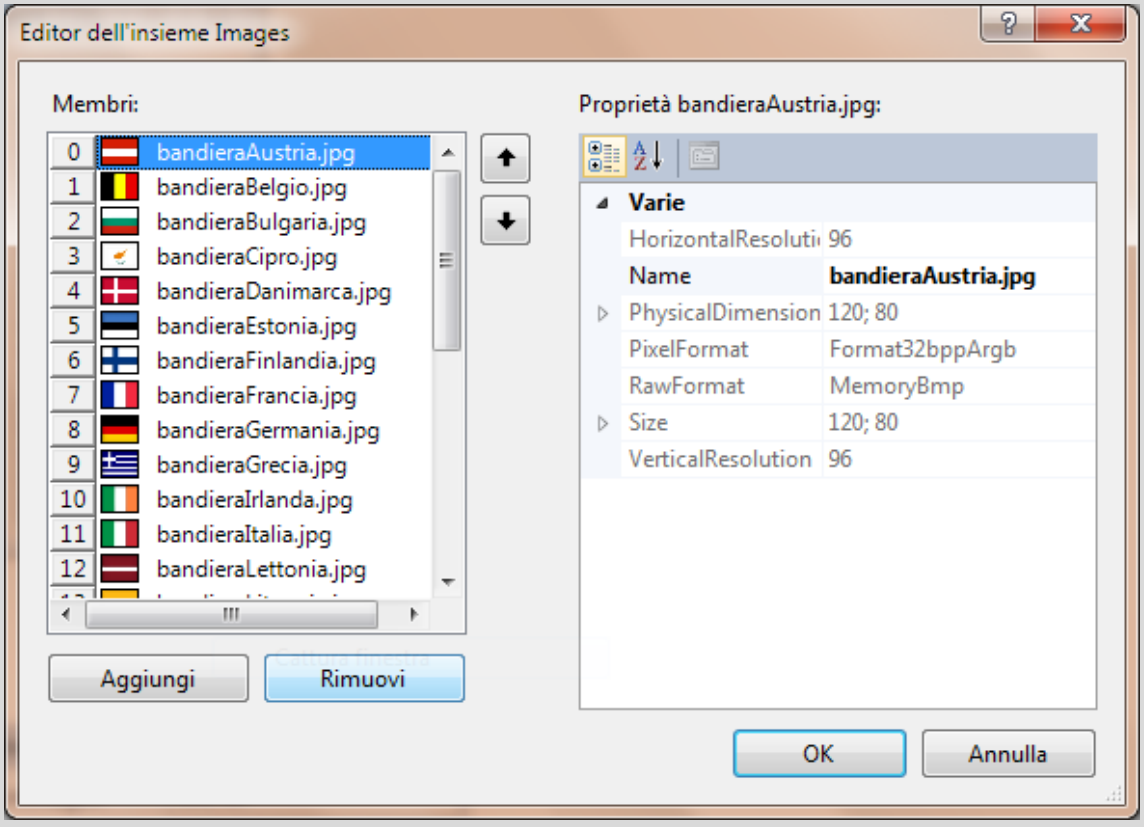
Non abbiamo ancora inserito nel programma le immagini delle 28 bandiere: lo faremo utilizzando un componente **ImageList** (elenco di immagini).
Inseriamo nel progetto un componente **ImageList** e facciamo un *clic* sul pulsante con la freccina nera che si trova nel suo angolo superiore destro:



Impostiamo alcune proprietà di questo **ImageList1**:

- **Dimensioni immagini = 120; 80**
- **Profondità in bit immagini = Depth24Bit**

Facciamo poi un *clic* su **Seleziona immagini** e accediamo alla finestra dell'**Editor dell'insieme Images**. Qui, cliccando il pulsante **Aggiungi**, inseriamo nell'ImageList le 28 immagini il cui nome inizia con "**bandiera…**", che troviamo nella cartella Documenti \ A scuola con VB \ Immagini \ Bandiere:



Premiamo il pulsante OK.
Passiamo ora alla scrittura del codice del programma.
Facciamo un doppio *clic* sul controllo **ListBox1** e accediamo alla Finestra del Codice, dove troviamo già impostata la procedura che gestisce l'evento del *clic* del mouse su uno degli items presenti nella ListBox.

Nello spazio vuoto di questa procedura scriviamo l'istruzione qui sotto evidenziata:

```
Public Class Form1

    Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged

        PictureBox1.Image = ImageList1.Images(ListBox1.SelectedIndex)

    End Sub


End Class
```

L'istruzione inserita può essere tradotta in questa termini: assegna al controllo **PictureBox1** l'immagine contenuta nella **ImageList1** il cui numero d'ordine corrisponde al numero d'ordine dell'item cliccato nel **ListBox1**.
Non ci sono altre istruzioni da scrivere.
L'immagine seguente mostra il programma in esecuzione:

Non dimentichiamo di salvare il progetto con il nome **Bandiere Unione Europea**, perché lo riprenderemo, più avanti, per aggiungervi nuove funzioni.

58: Il componente Timer.

Il computer dispone di un orologio interno che è il cuore del sistema, non solo perché memorizza l'ora e la data correnti, ma soprattutto perché regola le attività del computer e le ordina in senso cronologico.
L'orologio di sistema funziona anche quando il computer è spento.

Il componente **Timer** (temporizzatore) funziona basandosi su questo orologio interno; esso consente al programmatore di definire precisi intervalli di tempo, allo scadere di ognuno dei quali il **Timer** si incarica di mandare un segnale al programma.
Ad esempio, se l'intervallo definito dal programmatore è di un secondo, allo scadere di ogni secondo il Timer manda al programma un segnale, denominato **Tick**, che dal programma è considerato un vero e proprio evento, al pari degli eventi creati dall'utente con il mouse o con la tastiera.

Il **Timer** è un componente che rimane invisibile all'utente. Lo si colloca in un progetto prelevandolo con un doppio *clic* del mouse dalla Casella degli Strumenti; esso va a collocarsi fuori dal form, in basso.
E' possibile inserire in un progetto più di un Timer, che prenderanno in modo automatico i nomi Timer1, Timer2, … ; ognuno di essi funzionerà inviando i propri **Tick** secondo l'intervallo di tempo assegnatogli dal programmatore.

I Timer sono interlocutori attivi del programma, alla stregua dell'utente del programma. Mentre l'utente interagisce con il programma premendo pulsanti con il mouse o tasti sulla tastiera, i Timer interagiscono mandando un segnale allo scadere di ogni intervallo di tempo; allo scadere di ogni intervallo il programma registra l'accadere dell'evento **Tick** ed esegue le istruzioni connesse a questo evento.

La durata dell'intervallo tra un **Tick** e l'altro è espressa in millisecondi ed è definita dalla proprietà **Interval**.

- Impostando la proprietà **Interval = 1000** si ha dunque un **Tick** del Timer ogni secondo.
- Impostando la proprietà **Interval = 60000** si ha un **Tick** del Timer ogni minuto.
- Impostando la proprietà **Interval = 200** si ha un intervallo di tempo adatto ad animazioni grafiche con cambiamento di immagini in sequenza rapida.

Un timer può essere abilitato o disabilitato nella fase di progettazione o durante l'esecuzione di un programma (proprietà **Enabled = True** oppure **Enabled = False**).
Un Timer disabilitato ovviamente non produce eventi.
Per avviare o fermare un Timer dal codice del programma si possono utilizzare questi comandi:

```
Timer1.Start
' equivale a
Timer1.Enabled = True

Timer1.Stop
' equivale a
Timer1.Enabled = False
```

Un Timer serve, in un programma, ogni volta che si ha a che fare con il fattore tempo:

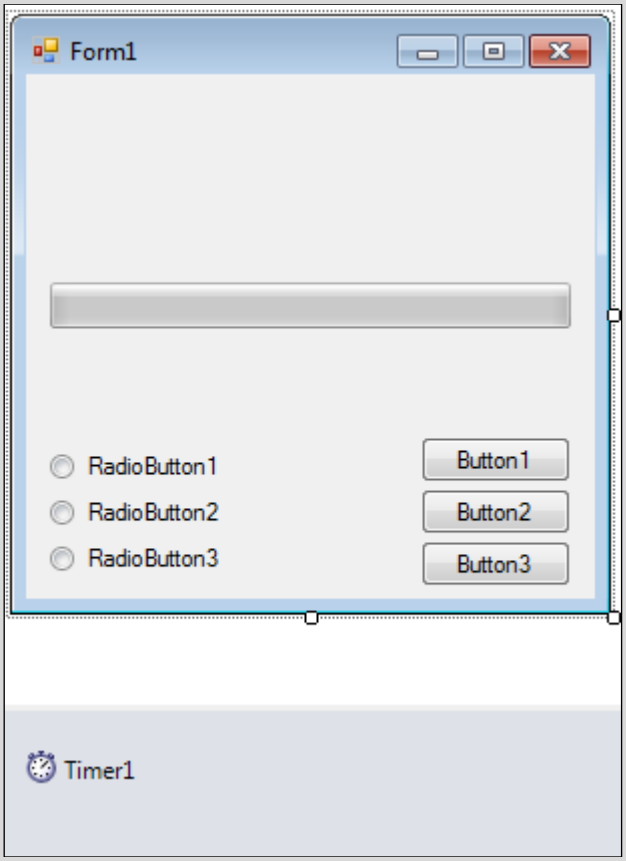
- quando è necessario misurare il tempo impiegato dall'utente nell'esecuzione di una operazione;
- all'interno di giochi in cui il tempo a disposizione dell'utente sia limitato;

- quando occorre ritardare l'esecuzione di un programma in attesa che il computer elabori dei dati;
- per creare effetti di animazione cambiando immagini in sequenza a brevi intervalli di tempo.

Il componente **Timer** è usato spesso in connessione a un controllo **ProgressBar** (barra di progressione) per visualizzare il trascorrere del tempo a disposizione dell'utente. Vedremo un esempio di questa connessione tra un Timer e una ProgressBar nel prossimo esercizio.

Esercizio 30: Il componente Timer.

In questo esercizio vedremo all'opera un componente **Timer** connesso a un controllo **ProgressBar**, con due pulsanti per avviare ed arrestare il Timer e con tre opzioni per impostare l'intervallo del Timer e, di conseguenza, la velocità di scorrimento della ProgressBar.
Apriamo un nuovo progetto e collochiamo nel form una ProgressBar, tre pulsanti Button e tre pulsanti RadioButton come in questa immagine:



Facciamo un *clic* sul Timer1, accediamo alla Finestra Proprietà e impostiamo le proprietà di questo componente come segue:

- **Enabled = False**
- **Interval = 500**

Impostiamo le proprietà **Name** e **Text** dei tre pulsanti Button come in questa tabella:

I pulsante	**Name = btnAvvia**	**Text = Avvia**
II pulsante	**Name = btnFerma**	**Text = Ferma**
III pulsante	**Name = btnRiprendi**	**Text = Riprendi**

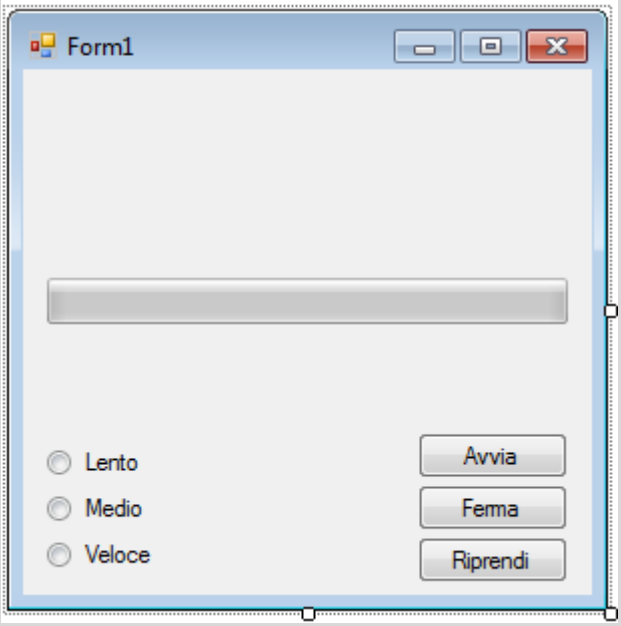
Ora impostiamo le proprietà **Name** e **Text** dei tre RadioButton come in questa tabella:

I RadioButton	**Name = btnLento**	**Text = Lento**
II RadioButton	**Name = btnMedio**	**Text = Medio**
III RadioButton	**Name = btnVeloce**	**Text = Veloce**

Non abbiamo bisogno di modificare le proprietà del controllo **ProgressBar**. Andiamo comunque a leggere le tre proprietà fondamentali di questo controllo:

- **Maximum = 100**
- **Minimum = 0**
- **Step = 10**

Con queste impostazioni, la parte colorata della ProgressBar andrà da un valore minimo uguale a 0 a un valore massimo uguale a 100, avanzando di 10 in 10 a ogni scatto.



L'elemento fondamentale per il funzionamento di questo programma è il Timer, che a ogni evento Tick farà avanzare la ProgressBar di uno scatto, grazie alla riga di istruzioni che adesso scriveremo.
Facciamo un doppio *clic* sul componente Timer, accediamo così alla Finestra del Codice, dove troviamo la procedura Timer1_Tick già impostata.
Nella riga centrale, vuota, scriviamo l'istruzione qui evidenziata in giallo:

```
Public Class Form1

    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ProgressBar1.PerformStep()

    End Sub

End Class
```

La riga inserita dice che a ogni Tick del Timer la ProgressBar deve avanzare di un scatto. Sappiamo che la ProgressBar andrà da 0 a 100, con scatti di 10 in 10, per cui dopo 10 scatti la barra sarà giunta al suo punto limite.
Ora passiamo alle procedure riguardanti i *clic* del mouse sui tre pulsanti Button e sui tre RadioButton. Le istruzioni da scrivere sono queste:

Evento	**Riga di istruzioni da scrivere nel codice**
clic sul **btnAvvia**	Timer1.Start
clic sul **btnFerma**	Timer1.Stop

clic sul **btnRiprendi**	Timer1.Start
clic sul **btnLento**	Timer1.Interval = 1000
clic sul **btnMedio**	Timer1.Interval = 500
clic sul **btnVeloce**	Timer1.Interval = 250

Ecco il codice completo del programma:

```
Public Class Form1

    Private Sub btnAvvia_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAvvia.Click

        ProgressBar1.Value = ProgressBar1.Minimum
        Timer1.Start()

    End Sub
```

```
    Private Sub btnFerma_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnFerma.Click

        Timer1.Stop()

    End Sub
```

```
    Private Sub btnRiprendi_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRiprendi.Click

        Timer1.Start()

    End Sub
```

```
    Private Sub btnLento_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs)Handles btnLento.CheckedChanged

        Timer1.Interval = 1000

    End Sub
```

```
    Private Sub btnMedio_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnMedio.CheckedChanged

        Timer1.Interval = 500

    End Sub
```

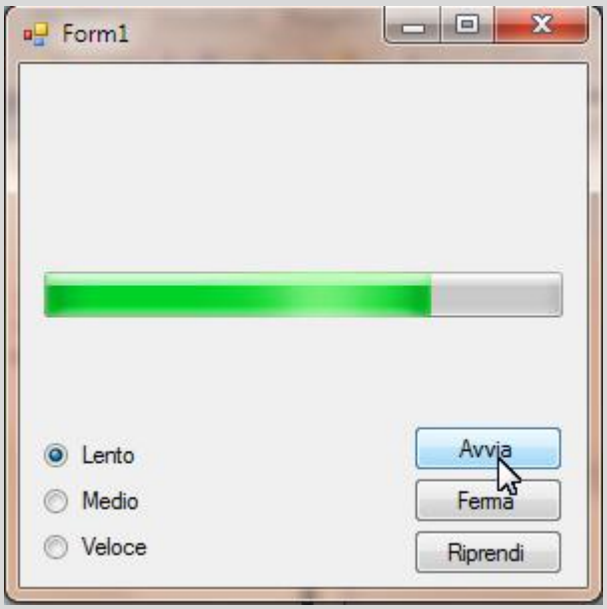
```
    Private Sub btnVeloce_CheckedChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnVeloce.CheckedChanged

        Timer1.Interval = 250

    End Sub
```

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        ProgressBar1.PerformStep()

    End Sub

End Class
```

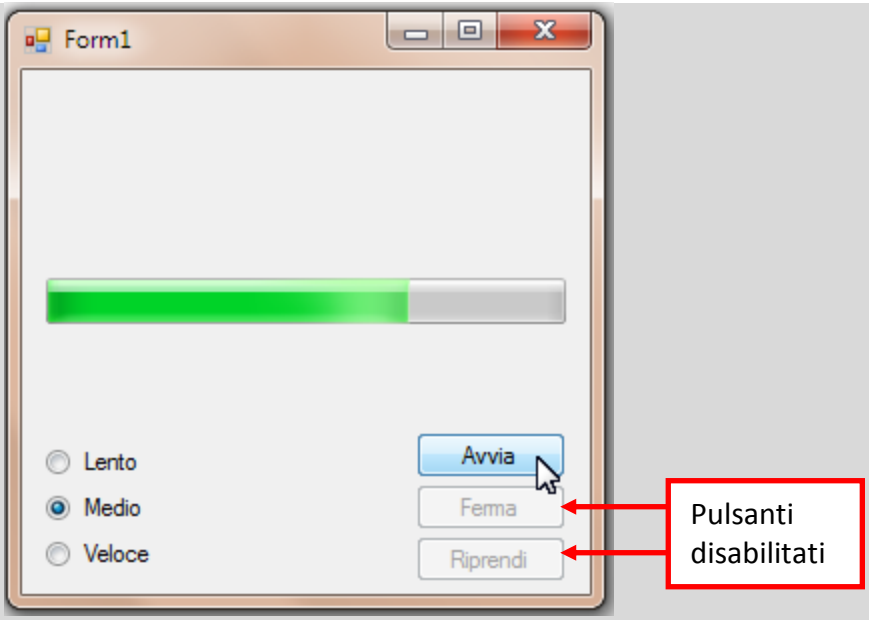
Notiamo nella procedura **btnAvvia_Click** questa riga di istruzioni:

```
        ProgressBar1.Value = ProgressBar1.Minimum
```

che può essere tradotta in questo modo: quando l'utente fa un *clic* sul pulsante **Avvia**, fai tornare la ProgressBar al suo valore minimo, cioè da capo.
Mandiamo in esecuzione il programma per vedere l'effetto dei *clic* sui vari pulsanti:



Si può perfezionare il programma con alcuni accorgimenti.
Fermiamo l'esecuzione. Andiamo nella Finestra Proprietà del **btnMedio** e impostiamo la proprietà **Checked = True**. Con questa impostazione, all'avvio del programma il Timer assumerà la proprietà **Interval = 500**, che è il valore connesso al RadioButton **Medio**.
All'avvio del programma la **ProgressBar** può essere solo **avviata**, per cui i pulsanti Ferma e Riprendi non dovrebbero essere abilitati.
Andiamo nella Finestra Proprietà del **btnFerma** e impostiamo la sua proprietà **Enabled = False**. Ripetiamo la stessa operazione per il pulsante **btnRiprendi**; in questo modo rimane abilitato solo il pulsante **btnAvvia**.
Ecco dunque come si presenta ora il programma al suo avvio:



Nella Finestra del Codice, nella procedura relativa al *clic* del mouse sul pulsante **btnAvvio** devono essere inserite le due righe di attivazione dei pulsanti **bntFerma** e **btnRiprendi**:

```
    Private Sub btnAvvia_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAvvia.Click

        ProgressBar1.Value = ProgressBar1.Minimum
        btnFerma.Enabled = True
        btnRiprendi.Enabled = True
        Timer1.Start()

    End Sub
```

Nel prossimo esercizio vedremo l'inserimento in un progetto di un form aggiuntivo che rimane visualizzato per pochi secondi e scompare automaticamente allo scadere dell'intervallo di tempo impostato nel componente Timer.

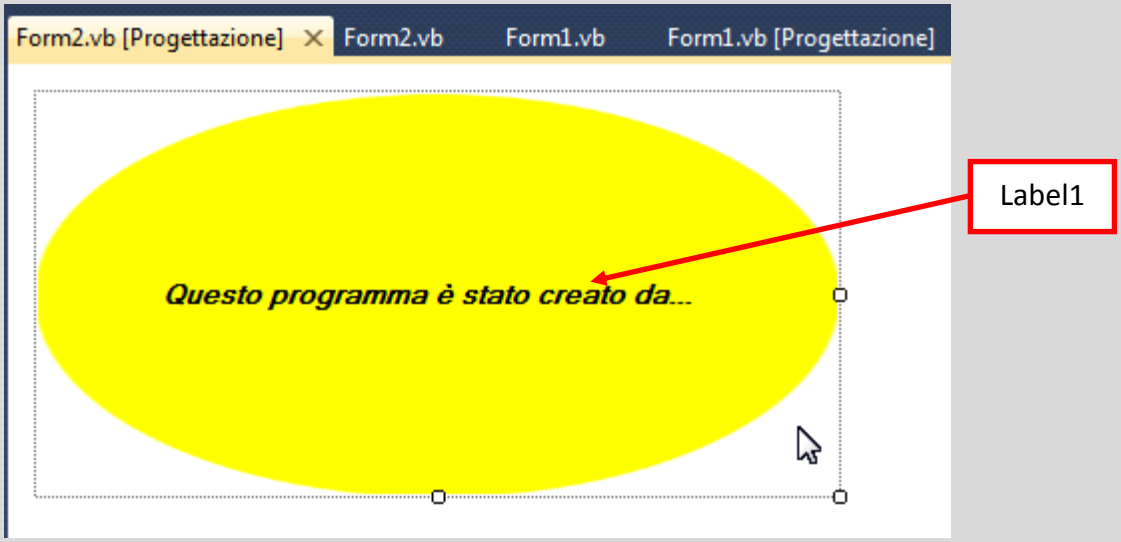
Esercizio 31: L'aggiunta di un form a scomparsa.

Apriamo un nuovo progetto. Facciamo un *clic* sul Form1 e nella Finestra Proprietà impostiamo la sua proprietà **StartPosition = CenterScreen** (quando il programma sarà in esecuzione, il form verrà visualizzato al centro del monitor).
Aggiungiamo un altro form al progetto: facciamo un *clic* sul menu **Progetto / Aggiungi Windows Form**.
Questo nuovo form, il **Form2**, è il form che verrà visualizzato in primo piano rispetto al **Form1**, all'avvio del programma, e si chiuderà, dopo alcuni secondi, per lasciare visibile solo il **Form1**.

Facciamo un *clic* sul **Form2**, accediamo alla Finestra Proprietà e impostiamo queste sue proprietà:

BackGroundImage	Facciamo un *clic* sul pulsante con i tre puntini e scegliamo una risorsa locale: l'immagine **tondo.jpg** che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini**
BackGroundImageLayout = Stretch	L'immagine tondo.jpg si adatterà alle dimensioni del Form.
FormBorderStyle = None	Nessun bordo: questo form si presenterà senza la barra in alto, senza titolo e senza i tre pulsanti in alto a destra per ridimensionarlo o per chiuderlo.
Size = 400; 200	
StartPosition = CenterScreen	Anche il Form2 verrà visualizzato al centro del monitor.
TopMost = True	Il form sarà visualizzato in primo piano, dunque coprirà il Form1 sottostante.
TransparencyKey = White	Il colore bianco che si vede nello sfondo del form sarà trasparente; rimarrà visibile solo il colore giallo del tondo.

Al centro di questo Form2 collochiamo un controllo **Label1** nella cui proprietà **Text** scriviamo *Questo programma è stato creato da…:*



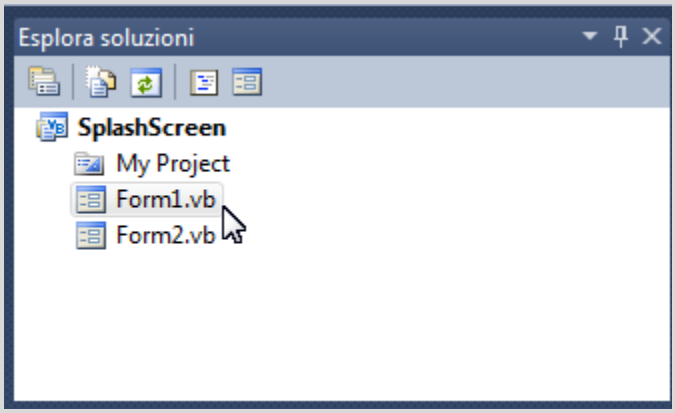
Avendo impostato la proprietà **TrasnparencyKey = White**, durante l'esecuzione del programma, di questo form sarà visibile solo l'ovale giallo.
Per fare chiudere in modo automatico il **Form2** useremo un componente Timer con la proprietà **Interval = 3000** (tre secondi): allo scadere dei tre secondi, il form si chiuderà e lascerà visibile solo il Form1.
Inseriamo dunque nel progetto un componente **Timer1**.
Notiamo che, trattandosi di un componente (oggetto non visibile all'utente del programma), il Timer1 va a collocarsi in un'area esterna al form, in basso.
Facciamo un *clic* sul Timer1 e nella Finestra Proprietà impostiamo queste sue proprietà:

- **Enabled = True** (il Timer sarà attivo da subito, all'avvio del programma);
- **Interval = 3000** (3 secondi: il Timer manderà un segnale al programma ogni 3 secondi).

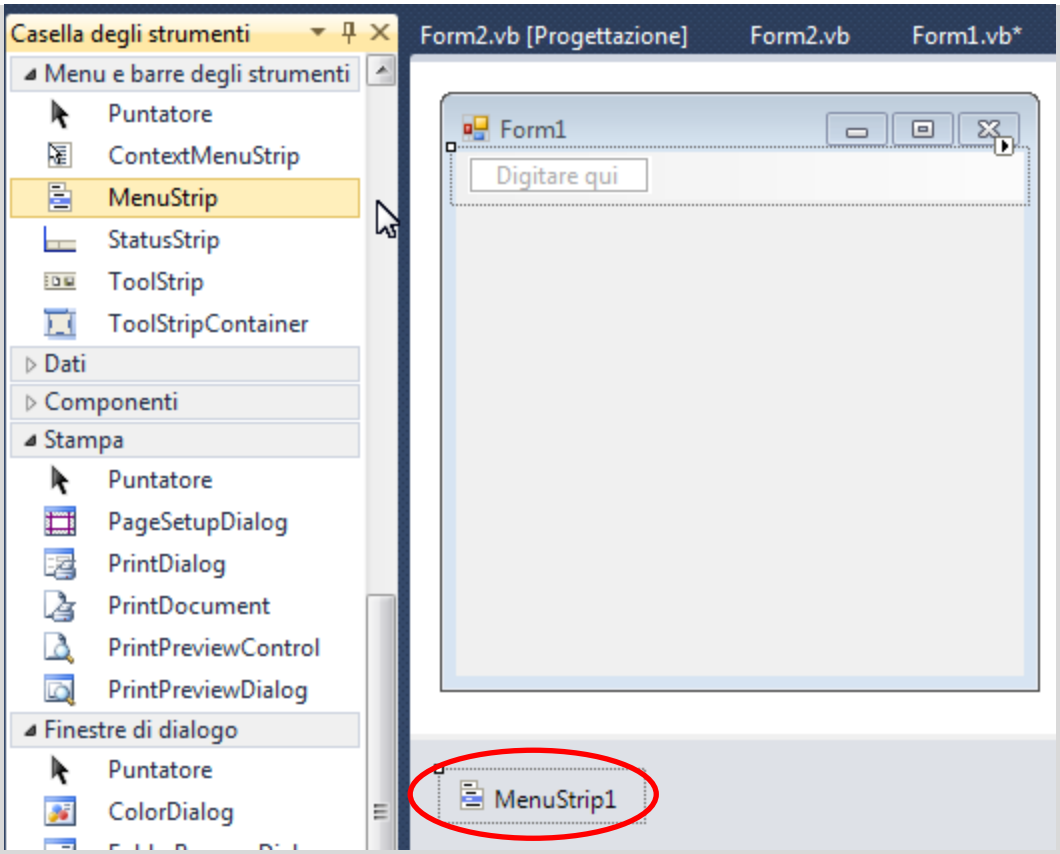
Facciamo un doppio *clic* sullo stesso Timer1. Accediamo così alla **Finestra del Codice** dove troviamo già impostata la procedura che gestisce l'evento del segnale del Timer, allo scadere dei tre secondi:

```
Public Class Form2

    Private Sub Timer1_Tick() Handles Timer1.Tick
        Me.Close()
    End Sub

End Class
```

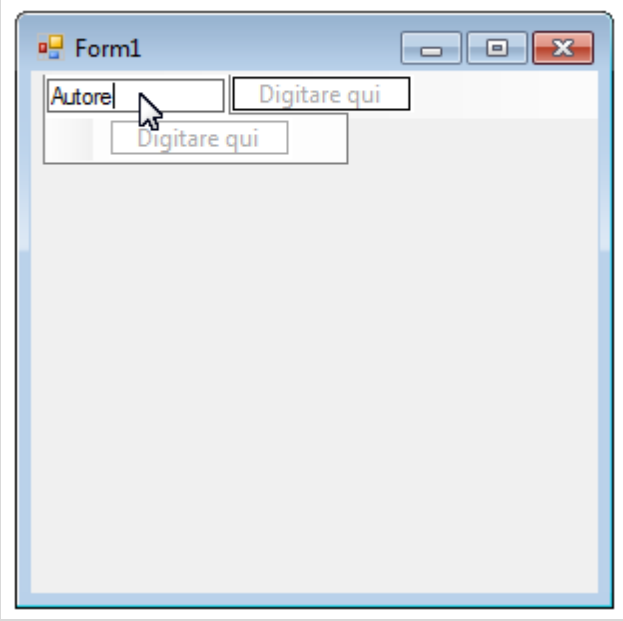
Nella riga centrale, evidenziata, è scritta l'istruzione per chiudere il Form2.
Ora torniamo a interessarci del Form1.
Nella Finestra **Esplora Soluzioni**, facciamo un *clic* sul **Form1**, per tornare alla Finestra della sua Progettazione:



Inseriamo nel Form1 un controllo **MenuStrip** (striscia di menu) prelevandolo, nella Casella degli Strumenti, dal gruppo **Menu e barre degli** strumenti:



In questa striscia di menu, scriviamo un solo menu: nella casella **Digitare qui** scriviamo la parola **Autore:**

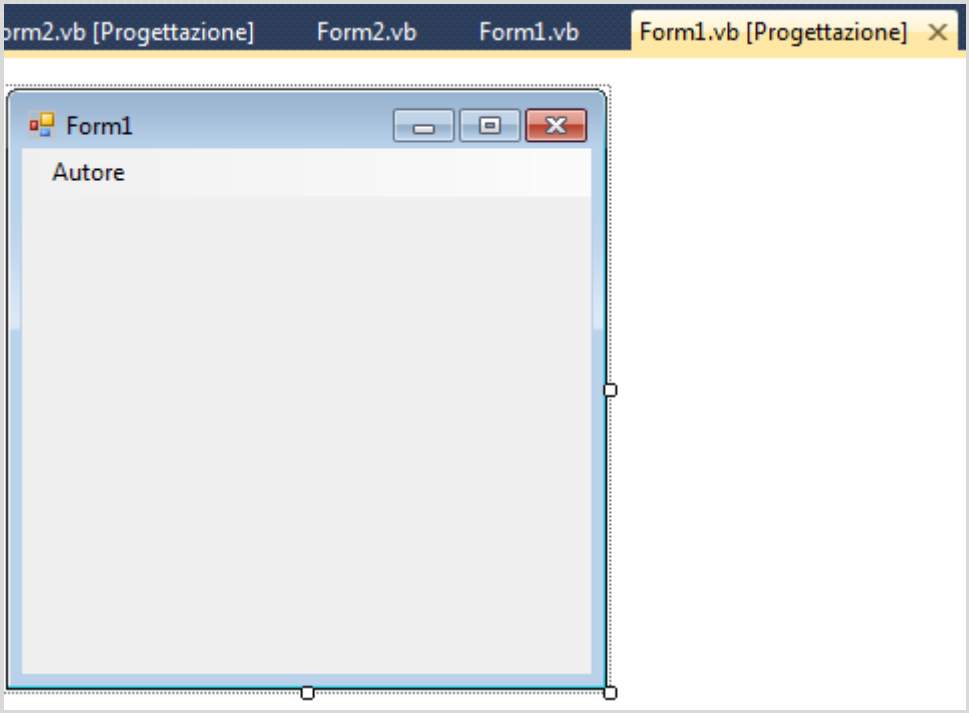


Con questo abbiamo terminato l'interfaccia del programma.

Facciamo un doppio *clic* sul Form1 e accediamo alla Finestra del Codice, dove troviamo già impostata la procedura che gestisce l'evento del caricamento del form in memoria, all'avvio del programma.
In questa procedura, scriviamo la riga centrale, che, al caricamento del Form1, comanda la visualizzazione del Form2:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Form2.Show()

    End Sub
```

Con il comando Form2.Show istruiamo il programma affinché al suo avvio, al momento del caricamento in memoria del Form1, visualizzi il Form2; ricordiamo che il **Form2** ha al suo interno un **Timer** che ne comanderà la chiusura dopo tre secondi, dopo di che rimarrà visibile all'utente del programma solo il Form1.
Ora torniamo alla Finestra di Progettazione del **Form1**:

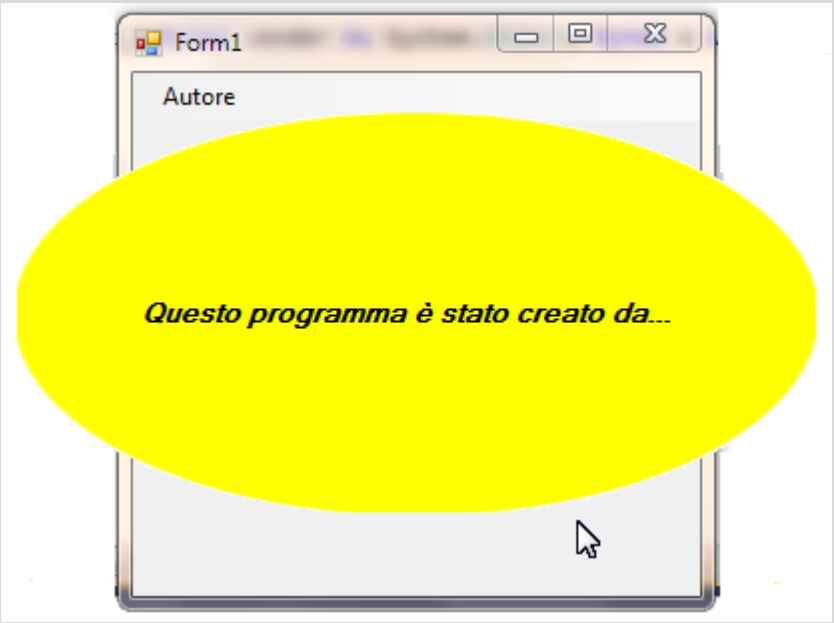


Facciamo un doppio *clic* sul menu **Autore** e accediamo ancora alla Finestra del Codice, dove troviamo impostata la procedura che gestisce l'evento del *clic* su questo menu.
Anche in questa procedura, nella sua riga centrale, scriviamo il comando per visualizzare il **Form2**:

```
    Private Sub AutoreToolStripMenuItem_Click_1(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AutoreToolStripMenuItem.Click

        Form2.ShowDialog()
```

```
    End Sub
```

L'immagine seguente mostra il programma in esecuzione; il testo della Label1, che si legge al centro del Form2, può essere completato a piacere, nella Finestra Proprietà, impostando la proprietà Text del controllo Label1 come si preferisce.



Capitolo 10: I COMPONENTI PER L'APERTURA DELLE FINESTRE DI DIALOGO CON L'UTENTE.

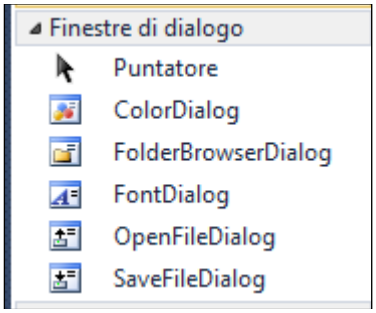


Figura 121: I componenti per l'apertura delle finestre di dialogo con l'utente, nella Casella degli Strumenti.

I componenti che fanno parte di questo gruppo visualizzano le finestre che normalmente appaiono nelle applicazioni Windows, soprattutto nei programmi di elaborazione di testi e di elaborazione di immagini, per consentire all'utente di scegliere un set di caratteri, un file da aprire, oppure le modalità di salvataggio di un file.

59: Il componente ColorDialog.

Il componente **ColorDialog** visualizza la finestra di Windows nella quale l'utente può scegliere un colore per uno sfondo, una cornice, un contorno. Per la scelta dei colori dei caratteri si usa invece il componente **FontDialog**.
Il componente ColorDialog, dopo essere stato inserito in un progetto, viene attivato con questo comando:

```
ColorDialog1.ShowDialog()
```

Il colore scelto dall'utente viene memorizzato da VB come `ColorDialog1.Color`
Nell'esempio seguente il colore scelto dall'utente è utilizzato per cambiare il colore di fondo del form:

```
Me.BackColor = ColorDialog1.Color
```

60: Il componente FolderBrowserDialog.

Il componente **FolderBrowserDialog** (finestra per la navigazione attraverso le cartelle) visualizza la finestra di Windows nella quale l'utente può scorrere le cartelle presenti nel disco fisso del computer o in altre unità collegate al computer, sino ad arrivare a scegliere la cartella preferita per una determinata operazione.
Il componente FolderBrowserDialog, dopo essere stato inserito in un progetto, viene attivato con questo comando:

```
FolderBrowserDialog1.ShowDialog()
```

La cartella scelta dall'utente viene memorizzata da VB come `FolderBrowserDialog1.SelectedPath` (Selected Path = percorso scelto).
Impostando la proprietà **RootFolder** del componente FolderBrowserDialog è possibile limitare la possibilità di scelta dell'utente.
Ad esempio, impostando la proprietà **RootFolder = MyPictures** l'utente avrà solo la possibilità di esplorare e di scegliere una cartella all'interno della cartella **Immagini** di Windows.

61: Il componente FontDialog.

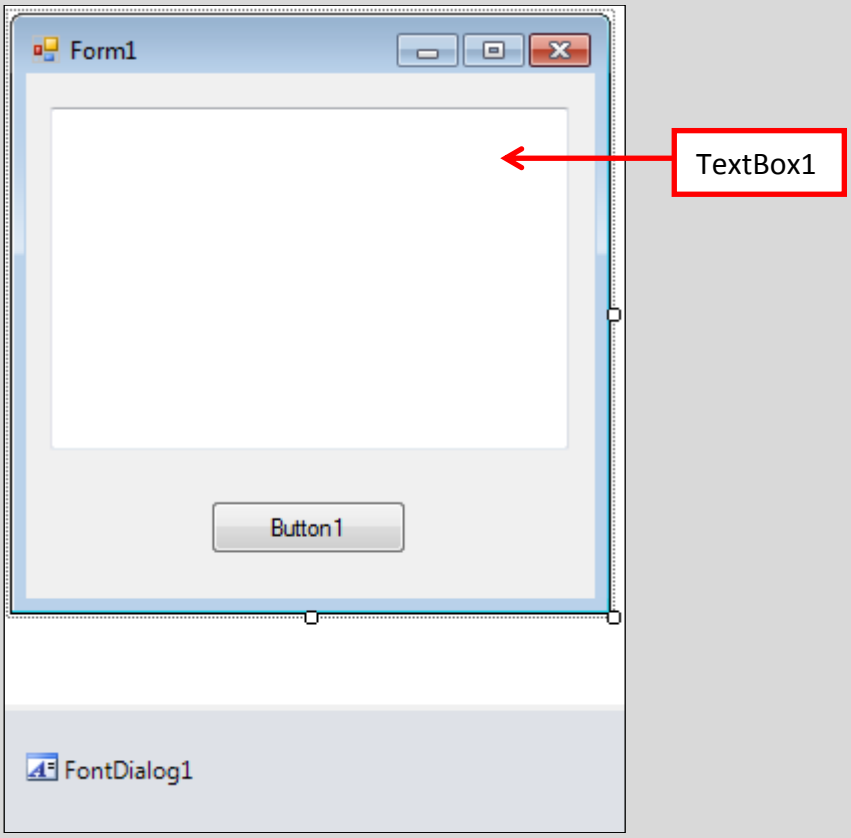
Il componente **FontDialog** (scelta del tipo e della grandezza dei caratteri) visualizza la finestra di Windows nella quale l'utente può scegliere il tipo di caratteri che preferisce e la loro grandezza.
Il componente FontDialog, dopo essere stato inserito in un progetto, viene attivato con questo comando:

```
FontDialog1.ShowDialog()
```

Il font scelto dall'utente viene memorizzato da VB come `FontDialog1.Font`.
Attenzione: se si modifica la proprietà **Font** di un form, la grandezza di questo form si adatta automaticamente alla grandezza dei caratteri; per evitare che il form assuma dimensioni non desiderate è necessario impostare la sua proprietà **AutoScaleMode = None**.
Tutti i controlli presenti in un form assumono automaticamente la proprietà Font del form, a meno che la proprietà Font di ognuno di loro sia impostata diversamente.
Nel prossimo esercizio vedremo un esempio di impostazione della proprietà **Font** per un controllo TextBox.

Esercizio 32: Il componente FontDialog.

In questo esercizio useremo il controllo **FontDialog** per consentire all'utente di un programma di cambiare i caratteri (tipo, dimensione e colore) di un testo visualizzato all'interno di una casella di testo.
Apriamo un nuovo progetto e collochiamo nel Form1 un controllo TextBox, un controllo Button e il componente FontDialog, come in questa immagine:



Impostiamo alcune proprietà di questi oggetti come indicato in questa tabella:

Per il **TextBox1**:

Multiline = True	Questa impostazione abilita il TextBox a disporre il testo su più righe, se necessario, mostrando la striscia di scorrimento verticale.
Text	Facciamo un *clic* sul pulsante con la freccina e scriviamo un testo di alcune righe.

Per il **Button1**:

- **Autosize = True**
- **Text = Scegli il carattere**

Per il **FontDialog1**:

ShowColor = True	Questa impostazione abilita la finestra di dialogo di mostrare l'opzione della scelta del colore.

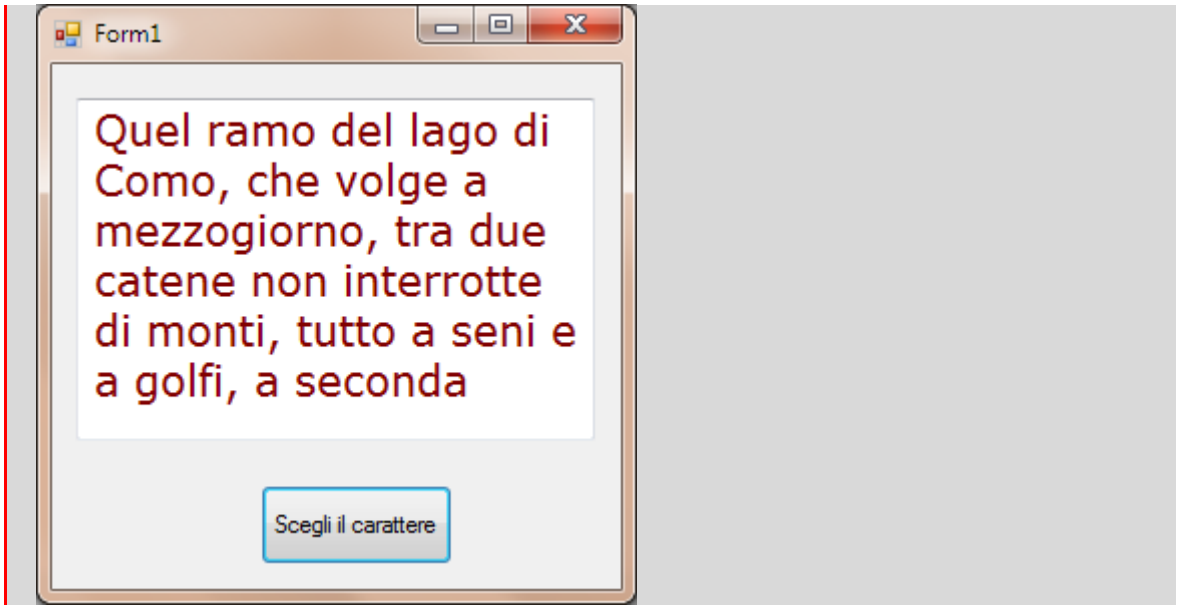
Ora facciamo un doppio *clic* sul pulsante Button1 e accediamo alla Finestra del Codice, dove troviamo già impostata la procedura per gestire l'evento di un *clic* sul Button1:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    End Sub

End Class
```

La procedura, come si vede dalla riga centrale vuota, è in attesa di istruzioni sul da farsi quando si verifica un *clic* sul Button1.
Inseriamo tre righe di istruzioni, scrivendole lentamente per approfittare dell'aiuto di **IntelliSense**:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        FontDialog1.ShowDialog()
        TextBox1.Font = FontDialog1.Font
        TextBox1.ForeColor = FontDialog1.Color

    End Sub

End Class
```

La prima delle tre righe inserite comanda la visualizzazione della finestra di dialogo per la scelta di un font di caratteri.
Quando l'utente ha fatto la sua scelta in questa finestra, le due righe di istruzioni successive assegnano al TextBox1 il set di caratteri e il colore scelti dall'utente.
L'immagine seguente mostra il programma in esecuzione:



62: I componenti OpenFileDialog e SaveFileDialog.

I componenti **OpenFileDialog** e **SaveFileDialog** visualizzano la finestra di Windows che consente all'utente di scegliere un file da aprire o il nome e la collocazione di un file da salvare.
I due componenti si attivano con questi comandi:

```
OpenFileDialog1.ShowDialog()

SaveFileDialog1.ShowDialog()
```

I nomi dei file scelti dall'utente sono memorizzati da VB come OpenFileDialog1.FileName e SaveFileDialog1.FileName.
In entrambi i controlli è possibile inserire un filtro di selezione dei file, per cui la scelta dell'utente può essere limitata all'apertura o al salvataggio di un determinato tipo di file (ad esempio: solo i file immagine, solo i file di testo, ecc.).
Il filtro di selezione è impostato dal programmatore nella proprietà **Filter** dei due componenti, come in questi esempi:

- Immagini GIF| * .gif
- Immagini JPG| * .jpg
- Immagini BMP|*.bmp
- Immagini GIF| * .gif| Immagini JPG| * .jpg| Immagini BMP|*.bmp
- Testi|*.txt

Il primo esempio limita la scelta dell'utente alle immagini di tipo .gif; il secondo esempio la limita alle immagini di tipo .jpg; il terzo la limita alle immagini di tipo .bmp; il quarto esempio consente la scelta di tre tipi di immagini .gif, .jpg e .bmp.
L'ultimo esempio limita la scelta ai file di testo di format .txt.
La sintassi con la quale va scritto il filtro di selezione è rigida:

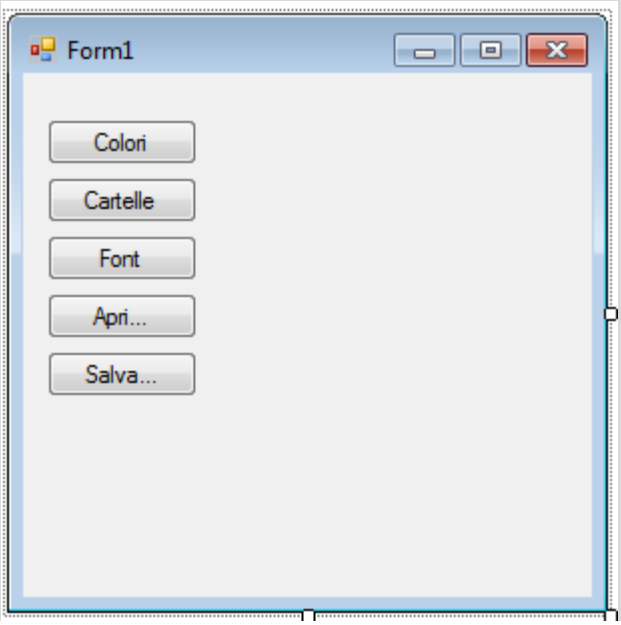
- prima viene il nome proprio della selezione (scelto dal programmatore),
- poi la barra verticale che si trova sotto il tasto ESC, in alto a sinistra nella tastiera del computer,
- poi un asterisco seguito dall'estensione del file. L'asterisco indica che la selezione accetta tutti i file con l'estensione indicata, qualsiasi sia il loro nome.

Nel prossimo esercizio vedremo un esempio di inserimento, in un unico programma, dei cinque componenti esaminati in questo capitolo:

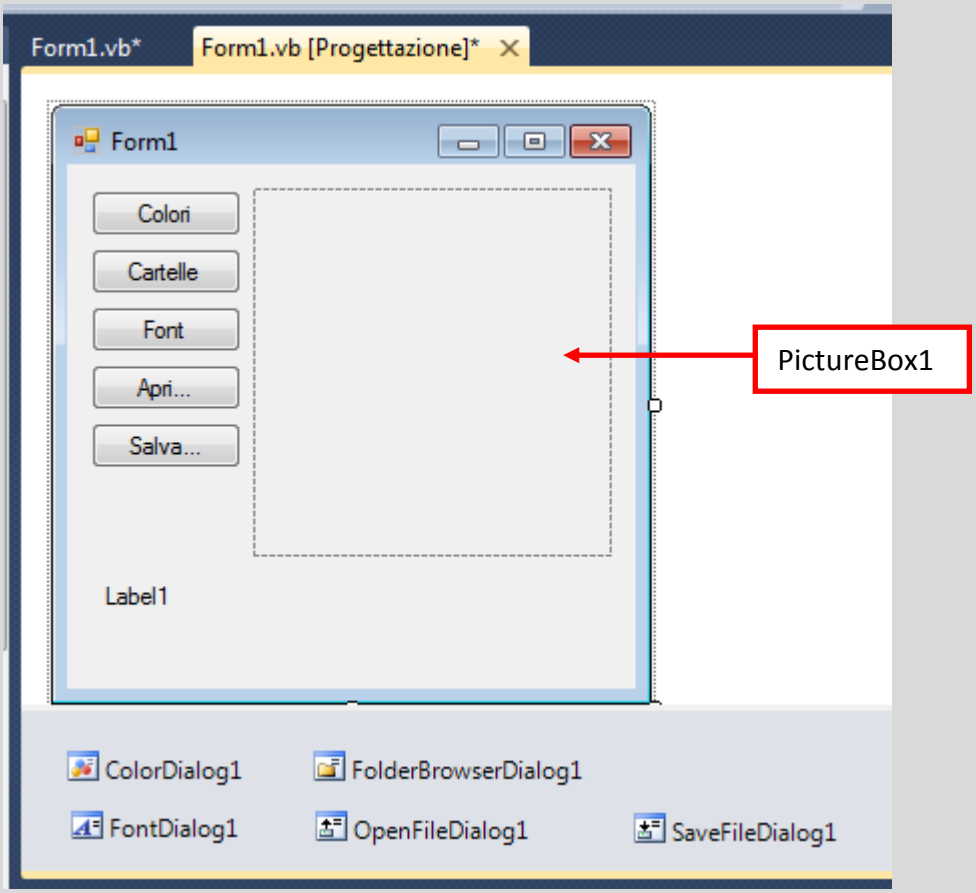
- ColorDialog,
- FolderBrowserDialog,
- FontDialog,
- OpenFileDialog,
- SaveFileDialog.

Esercizio 33: Apertura e uso delle finestre Windows di dialogo con l'utente.

Apriamo un nuovo progetto e inseriamo nel Form1 cinque pulsanti Button.
Impostiamo le loro proprietà **Text**, rispettivamente, con le parole **Colori**, **Cartelle**, **Font**, **Apri…**, **Salva…**, come in questa immagine:



A ognuno di questi cinque pulsanti sarà collegata l'apertura di una finestra di dialogo con l'utente.
Ora inseriamo nel Form1 un controllo Label1, un controllo PictureBox1 e i cinque componenti ColorDialog, FolderBrowserDialog, FontDialog, OpenFileDialog, SaveFileDialog, come in questa immagine:



Nella Finestra Proprietà, impostiamo le proprietà di questi oggetti:

Per il **Form1**:

AutoScroll = True	Questa impostazione farà apparire nel Form1 le barre di scorrimento verticale e orizzontale se le dimensioni del controllo PictureBox1 diventeranno maggiori di quelle del form.

Per il **PictureBox1**:

SizeMode = AutoSize	Questa impostazione fa sì che il controllo PictureBox adatti le sue dimensioni a quelle della immagine da visualizzare.

Per il **FolderBrowserDialog1**:

RootFolder = MyPictures	Questa impostazione limita la navigazione, da parte dell'utente, alla cartella **Immagini** di Windows.

Per il **FontDialog1**:

ShowColor = True	Questa impostazione consente all'utente del programma di scegliere anche il colore dei caratteri del testo.

Per l'**OpenFileDialog1**:

Filter = Immagini JPG| * .jpg	Questa impostazione limita la scelta delle immagini alle immagini di tipo .jpg.

Ora iniziamo a scrivere le procedure necessarie per attivare il collegamento tra i cinque pulsanti Button e i rispettivi componenti.
Facciamo un doppio *clic* sul primo pulsante **Button1** (**Colori**), e accediamo alla Finestra Proprietà, dove troviamo una procedura già impostata per gestire l'evento di un *clic* del mouse su questo pulsante.
Completiamo la procedura inserendo due righe di comandi:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ColorDialog1.ShowDialog()
        Button1.BackColor = ColorDialog1.Color

    End Sub

End Class
```

La prima delle due righe inserite comanda l'apertura della finestra di Windows dedicata alla scelta dei colori (`ColorDialog1.ShowDialog`).
La seconda riga assegna il colore scelto dall'utente allo sfondo del pulsante Button1.

Procediamo con la scrittura delle procedure relative agli altri pulsanti.
Per il **Button2** (**Cartelle**):

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

        FolderBrowserDialog1.ShowDialog()
```

```
        Label1.Text = FolderBrowserDialog1.SelectedPath

    End Sub
```

La prima delle due righe inserite comanda l'apertura della finestra di Windows dedicata alla navigazione tra le cartelle (`FolderBrowserDialog1.ShowDialog`). Avendo impostato la proprietà **RootFolder** di questo componente **= MyPictures**, l'utente potrà navigare solo all'interno della cartella **Immagini** di Windows.
La seconda riga trasferisce all'interno della Label1, come riga di testo, il percorso scelto dall'utente.

Per il **Button3** (**Font**):

```
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click

        FontDialog1.ShowDialog()
        Button3.Font = FontDialog1.Font
        Button3.ForeColor = FontDialog1.Color

    End Sub
```

La prima delle tre righe inserite comanda l'apertura della finestra di Windows dedicata alla scelta del tipo e delle dimensioni dei caratteri (`FontDialog1.ShowDialog`).
La seconda riga assegna al Button3 il font scelto dall'utente.
La terza riga assegna al font del Button3 il colore scelto dall'utente.

Per il **Button4** (**Apri…**):

```
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click

        OpenFileDialog1.ShowDialog()
        PictureBox1.Load(OpenFileDialog1.FileName)

    End Sub
```

La prima delle due righe inserite comanda l'apertura della finestra di Windows dedicata alla navigazione tra i file (`OpenFileDialog1.ShowDialog`). Sappiamo che, per impostazione della proprietà **Filter**, la finestra visualizzerà solo i file di immagini di formato .jpg.
La seconda riga visualizza il file scelto dall'utente all'interno del PictureBox1.
Avendo impostato la proprietà SizeMode del controllo PictureBox1 come **AutoSize**, il controllo stesso diventerà più o meno grande, a seconda delle dimensioni della immagine da visualizzare. Se le dimensioni del PictureBox1 eccederanno quelle del Form1, questo visualizzerà le barre di scorrimento, in quanto la sua proprietà **AutoScroll** è impostata **= True**.

Per il **Button5** (**Salva…**):

```
    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
```

```
        SaveFileDialog1.ShowDialog()

    End Sub
```

L'unica riga inserita per questo pulsante comanda l'apertura della finestra di Windows dedicata alla navigazione tra i file (`SaveFileDialog1.ShowDialog`) per effettuare il salvataggio di un file. Nel nostro caso non abbiamo alcun file da salvare, per cui nella procedura non sono scritte altre operazioni.

Capitolo 11: I CONTROLLI VISUAL BASIC POWERPACKS.

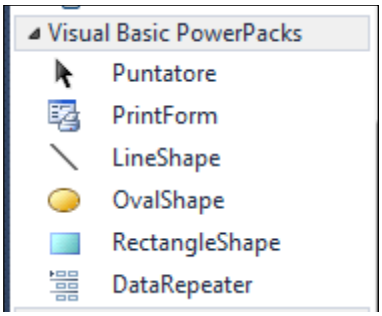


Figura 122: I controlli del gruppo Visual Basic PowerPacks, nella Casella degli Strumenti.

In questo capitolo analizziamo un gruppo di controlli che sono stati recuperati da Visual Basic 6, versione precedente del linguaggio VB, e trasferiti in VB possiamo dire *a grande richiesta*, grazie al successo che avevano incontrato tra i programmatori.

63: Il componente PrintForm.

Il componente **PrintForm** inserisce in un pogetto VB la possibilità di stampare un'immagine del form in modo diretto, rapido e semplice; con questo componente l'utente può stampare direttamente ciò che compare all'interno del form.
Il prodotto della stampa può risultare di qualità mediocre in quanto **PrintForm** non consente di sfruttare tutte le possibilità della stampante, ma la sua immediatezza rende l'uso di **PrintForm** ancora utile e sufficiente per le necessità di stampa di molti programmi.
Modalità di stampa più articolate e flessibili sono possibili utilizzando i componenti dedicati alle funzioni di stampa, che vedremo più avanti.

64: I controlli LineShape, OvalShape, RectangleShape.

I controlli **LineShape**, **OvalShape**, **RectangleShape** sono tre controlli di uso molto semplice, che servono a creare nel form figure geometriche ed effetti grafici di varia natura.

LineShape (forma lineare) disegna nel form linee separatrici con stili e grandezze diverse.
Aumentandone la larghezza, queste linee possono assumere l'aspetto di barre rettangolari.
La collocazione di un oggetto LineShape sul form ha quattro punti di riferimento:

- **X1** è la distanza dell'inizio della linea dal bordo sinistro del form;
- **X2** è la distanza della fine della linea dal bordo sinistro del form;
- **Y1** è la distanza dell'inizio della linea dal bordo superiore del form;
- **Y2** è la distanza della fine della linea dal bordo superiore del form.

Per collocare il controllo **LineShape** su un form bisogna selezionarlo nella Casella degli Strumenti, poi portarsi con il mouse nel form, fare *clic* con il mouse nel punto in cui si desidera far iniziare la linea e, tenendo premuto il pulsante del mouse, trascinare il puntatore fino al punto in cui si desidera far terminare la linea.
Facendo un *clic* con il mouse su una linea già disegnata nel form compaiono due maniglie di dimensionamento. Utilizzando queste maniglie è possibile spostare la linea nel form, oppure modificarne l'orientamento e la lunghezza.
Se si desidera modificare l'aspetto della linea è possibile cambiarne le proprietà **BorderColor**, **BorderStyle** e **BorderWidth** nella Finestra Proprietà.

Il controllo **OvalShape** (forma ovale) disegna nel form cerchi o forme ovali.
Il controllo **RectangleShape** (forma rettangolare) disegna nel form rettangoli o quadrati.
Il collocamento di un controllo **OvalShape** o **RectangleShape** nel form avviene in questo modo:

- selezionare il controllo che interessa nella Casella degli Strumenti, poi
- portarsi con il mouse nel form,
- fare *clic* con il mouse nel punto del form in cui si desidera che la forma geometrica inizi e, tenendo premuto il pulsante del mouse,
- trascinare il puntatore fino al punto in cui si desidera che la forma geometrica termini.

Dando una forma quadrata al riquadro di un controllo **OvalShape** si ottiene la visualizzazione di un cerchio.
Dando una forma quadrata al riquadro di un controllo **RectangleShape** si ottiene la visualizzazione di un quadrato.
È possibile creare figure **RectangleShape** con angoli normali o con angoli arrotondati, impostando la proprietà **CornerRadius** (raggio dell'angolo) con un valore a scelta da 0 (nessun arrotondamento) a 51 (arrotondamento massimo).

Facendo un *clic* con il mouse su uno di questi controlli compaiono le abituali maniglie di dimensionamento con le quali è possibile spostarlo o ridimensionarlo.
I controlli **OvalShape** e **RectangleShape** sono anche in grado di visualizzare un'immagine al loro interno.
Per quanto si tratti di oggetti eminentemente grafici, utili come sfondo di un testo o per evidenziare una parte dell'area del form, i tre controlli sono in grado di recepire molti eventi, tra i quali il *clic* e il doppio *clic* del mouse o il passaggio del mouse sopra di essi.
Hanno in comune queste proprietà:

BorderColor	Impostazione del colore di sfondo.
BorderStyle	Impostazione dello stile del contorno: ▪ **Solid** (linea continua); ▪ **Dash** (linea tratteggiata); ▪ **Dot** (punti); ▪ **DashDot** (un trattino e un punto); ▪ **DashDotDot** (un trattino e due punti).
BorderWidth	Impostazione della larghezza del contorno del controllo.
FillColor	Impostazione del colore di riempimento del controllo. Nel caso che il programmatore voglia visualizzare nello sfondo una sfumatura da un colore all'altro, questo è il primo dei due colori della sfumatura.
FillGradientColor	Impostazione del secondo colore della sfumatura del colore di fondo.
FillGradientStyle	Impostazione della modalità di sfumatura dei colori nello sfondo del controllo. Questa, nel passaggio dal primo al secondo colore, può essere impostata da sinistra a destra, dall'alto in basso, ecc.
FillStyle	Stile del riempimento. Questa proprietà deve essere impostata come Solid perché il colore di riempimento del controllo sia visibile, ma vi sono alcune decine di opzioni disponibili, per ottenere diversi effetti grafici.

Tabella 7: Le proprietà dei controlli OvalShape e RectangleShape.

Esercizio 34: I controlli LineShape, OvalShape, RectangleShape.

Apriamo un nuovo progetto e collochiamo nel Form1 i controlli LineShape, OvalShape e RectangleShape, con queste proprietà:

LineShape1	**BorderColor**	Navy
	BorderStyle	Dot
	BorderWidth	10
	X1	0
	X2	280
	Y1	0
	Y2	30

OvalShape1	**BorderColor**	Maroon
	BorderStyle	Solid
	BorderWidth	6
	FillColor	White
	FillGradientColor	Gray
	FillGradientStyle	Horizontal
	FillStyle	Solid
	Location	20; 40
	Size	240; 90

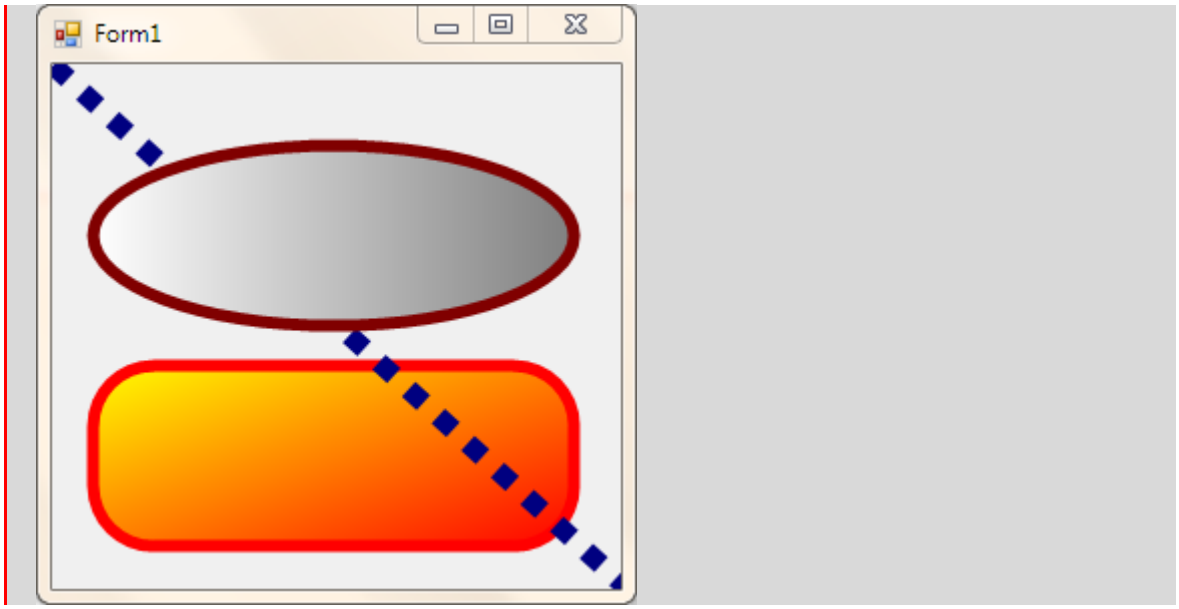
RectangleShape1	**BorderColor**	Red
	BorderStyle	Solid
	BorderWidth	6
	CornerRadius	30

FillColor	Yellow
FillGradientColor	Red
FillGradientStyle	ForwardDiagonal
FillStyle	Solid
Location	20; 150
Size	240; 90

Dopo avere impostato i tre controlli, andiamo a scrivere una procedura, nella Finestra del Codice, affinché il controllo LineShape1, nella fase di caricamento del programma, vada a collocarsi in diagonale nel form.
Facciamo un doppio *clic* sul form (all'esterno dei tre controlli che vi abbiamo collocato) e accediamo alla Finestra del Codice, dove troviamo già impostata la procedura relativa all'evento MyBase.Load (caricamento del form).
Nello spazio all'interno della procedura scriviamo quattro linee di codice:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        LineShape1.X1 = 0
        LineShape1.X2 = 315
        LineShape1.Y1 = 0
        LineShape1.Y2 = 287

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



65: Il controllo DataRepeater.

Il controllo **DataRepeater** crea un contenitore scorrevole di controlli che visualizzano dei dati.
Ad esempio, nel caso si debba visualizzare il catalogo di una biblioteca, i dati relativi a un singolo libro vengono visualizzati da alcune label, tutte contenute in un controllo DataRepeater che assume il ruolo di contenitore *prototipo*.
Il DataRepeater poi replica questo *prototipo*, con le label in esso contenute, per ogni riga del database che contiene il catalogo della biblioteca.
Il risultato è visualizzato in una tabella che l'utente può scorrere usando le barre di scorrimento.

(pagina vuota)

PARTE III: IL LINGUAGGIO DI PROGRAMMAZIONE.

Nella parte precedente abbiamo visto quali sono e come funzionano gli elementi a disposizione del programmatore per progettare l'interfaccia di un programma.
Negli esercizi contenuti in quella parte abbiamo già avuto modo di scrivere righe di istruzioni (cioè di codice).
In questa terza parte ci occuperemo a fondo della scrittura del codice, analizzandone le caratteristiche e le potenzialità: qui passeremo dall'analisi degli strumenti grafico/visivi a quella del linguaggio logico di VB.
Com'è facile immaginare, lo studio di questa parte sarà più arido e più ostico, ma non mancheranno piacevoli sorprese; in particolare, avremo la soddisfazione di fare svolgere al computer operazioni complesse quali la gestione di dati e di variabili secondo il verificarsi o meno di determinate condizioni, utilizzando operatori logici e matematici, comandi condizionati e iterativi, variabili e matrici di variabili, raggruppamenti di controlli, funzioni e procedure…

Capitolo 12: L'EDITOR DEL CODICE.

Iniziamo dall'analisi della **Finestra del Codice**, che è riprodotta nella figura seguente.
Ricordiamo che per visualizzare la Finestra del Codice bisogna prima aprire un nuovo progetto e poi compiere una di queste azioni:

- cliccare il Form1 nella Finestra Esplora soluzioni e fare *clic* sul comando Visualizza codice, oppure
- fare un doppio *clic* nel riquadro del Form1, oppure
- cliccare il menu Visualizza / Codice nella striscia dei menu, oppure
- premere il tasto F7.

66: La Finestra del Codice.

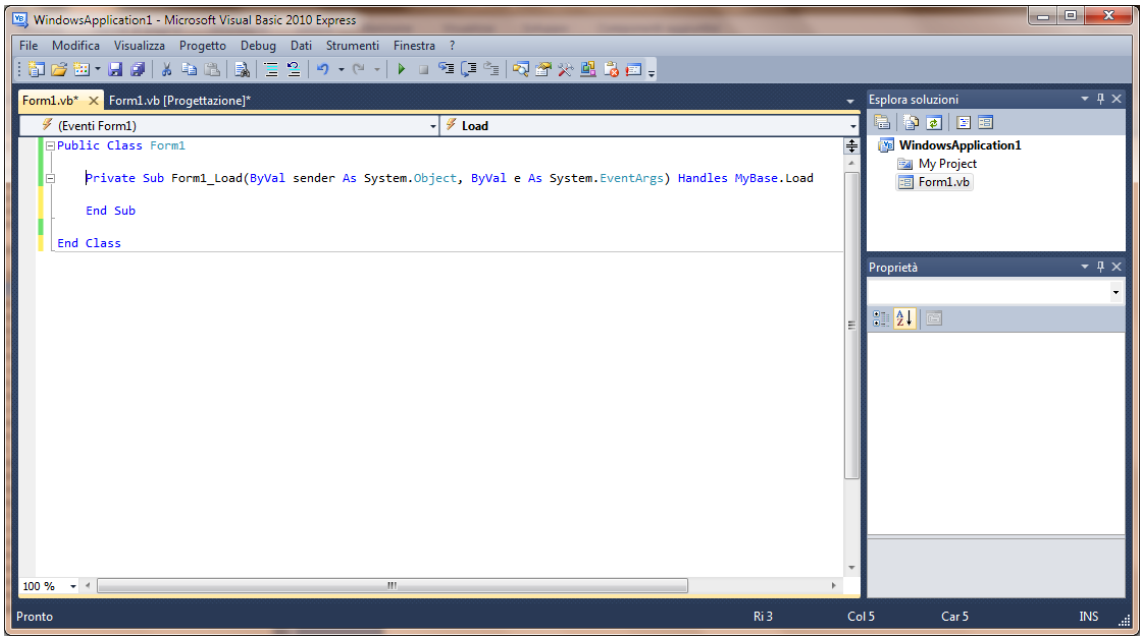


Figura 123: La Finestra del Codice.

Questa è la Finestra in cui il programmatore scrive il codice di un programma, cioè l'insieme delle istruzioni che determineranno le azioni al computer in risposta agli eventi causati dall'utente che userà il programma.
L'*editor* del codice, cioè lo strumento di scrittura del codice, funziona come un normale elaboratore di testi in cui parole, righe, frasi, paragrafi possono essere evidenziati e poi cancellati, copiati, spostati come se si trattasse di brani di un normale testo.
E' incorporato nell'*editor* del codice lo strumento **IntelliSense**, che controlla e supporta il lavoro di scrittura del codice con tre potenti funzioni di **correzione**, di **suggerimento** e di **informazione** per il programmatore, grazie alle quali:

- l'*editor* corregge automaticamente alcuni errori di battitura (maiuscole, spazi, virgolette) e formatta il testo in modo ordinato;
- quando il programmatore scrive il nome di un oggetto o di un controllo all'inizio di una riga di codice, l'*editor* gli fornisce automaticamente l'elenco di tutte le proprietà e di tutti gli eventi che possono essere riferiti a quell'oggetto o a quel controllo; la proprietà o l'evento appropriati possono essere scelti premendo la barra spazio sulla tastiera;
- l'*editor* informa il programmatore sulla sintassi delle istruzioni (cioè su come deve essere scritta una istruzione perché possa essere letta da VB): quando si inizia a scrivere il nome di un'istruzione valida, la sintassi corrispondente viene immediatamente visualizzata sotto la riga in cui si sta scrivendo.

Nell'esercizio seguente vedremo un esempio delle funzioni di supporto che **IntelliSense** svolge per il programmatore.

Esercizio 35: Scrittura del codice e supporto di IntelliSense.

In VB ogni oggetto è la concretizzazione di una Classe[43], cioè di un software creato da altri programmatori, e il form non sfuggge a questa regola.
Apriamo un nuovo progetto di VB e facciamo un doppio *clic* con il mouse sul Form1: accediamo così alla Finestra del Codice, dove troviamo già inizializzata la Classe del Form1:

```
Public Class Form1

        Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        End Sub

End Class
```

Tutto il codice che riguarda il Form1 e gli oggetti in esso contenuti va scritto all'interno di questa classe, nello spazio compreso tra le righe:

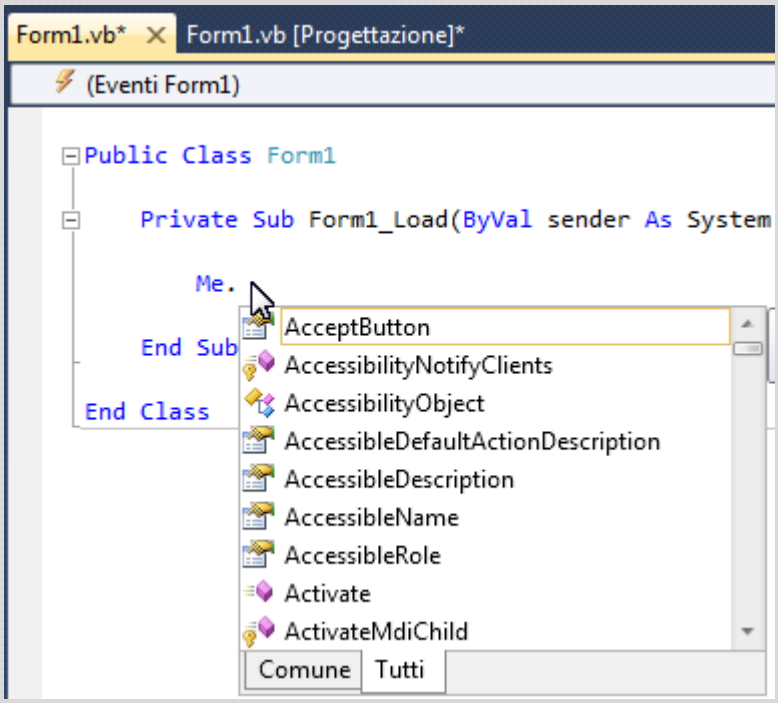
[43] Si veda il paragrafo 17: La programmazione orientata agli oggetti., a pag. 63.

```
Public Class Form1

End Class
```

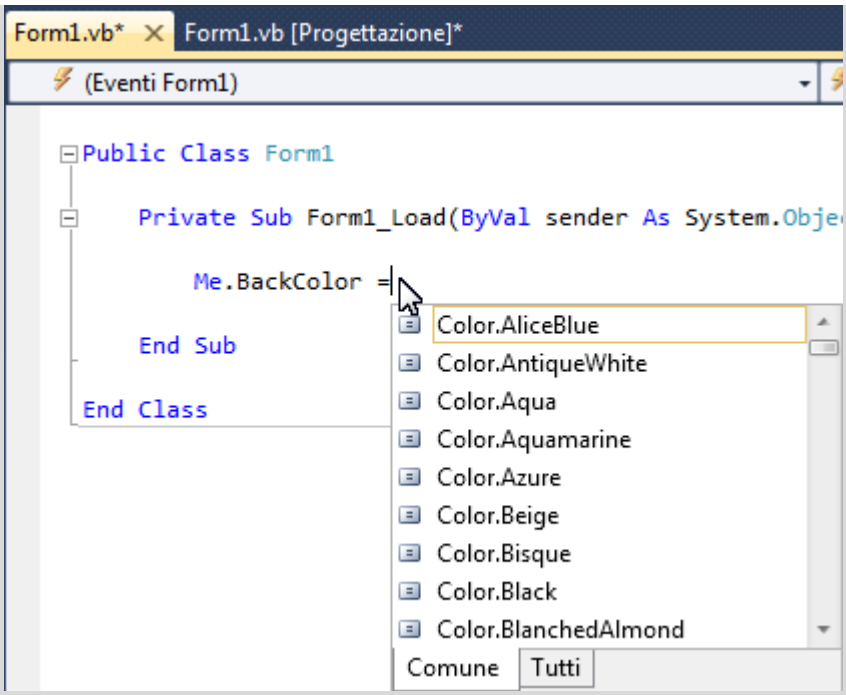
All'interno di queste righe troviamo, predisposta dall'*editor* del codice, la procedura che gestisce l'evento del caricamento del Form1 in memoria, cosa che avviene in genere all'avvio di un programma:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub
```

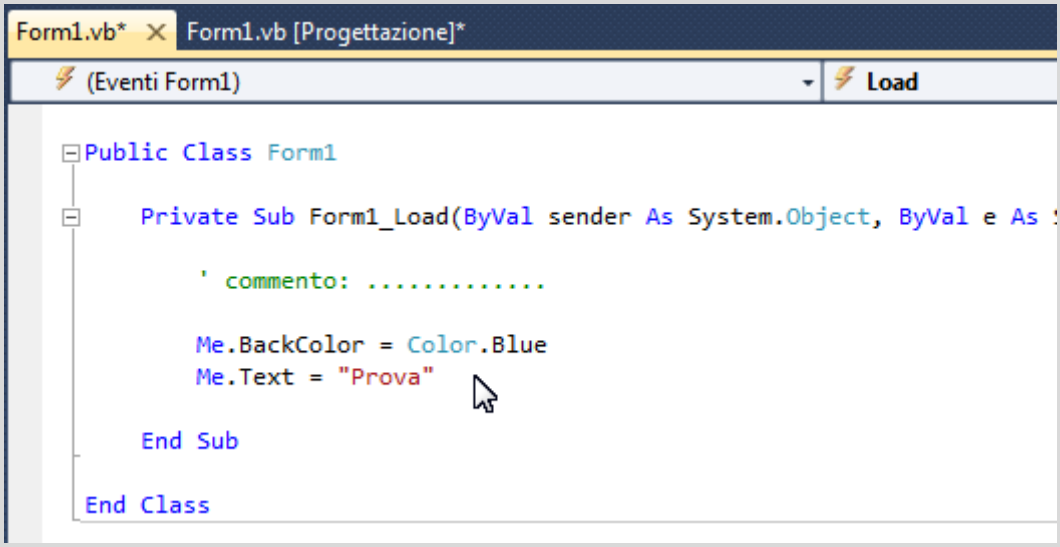
Nello spazio intermedio, vuoto, proviamo a scrivere una riga di istruzioni.
Iniziamo a scrivere la parola **Me** seguita da un **punto** (**Me** è il pronome che si riferisce al Form1).
Notiamo che, appena aggiungiamo il punto dopo Me, IntelliSense fornisce l'elenco delle proprietà e delle azioni disponibili per Me, cioè per il Form1:



In questo elenco di proprietà e di azioni, facciamo un *clic* sulla proprietà **BackColor**, poi scriviamo il simbolo = e vediamo che IntelliSense interviene ancora fornendo l'elenco dei colori disponibili:



Facciamo un *clic* sul colore blu.
Scriviamo ora un'altra istruzione per modificare la proprietà **Text** del Form1 (cioè il suo titolo), e aggiungiamo una riga di commento, completando la procedura in questo in questo modo:




```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        ' commento: .............

        Me.BackColor = Color.Blue
```

```
        Me.Text = "Prova"

    End Sub

End Class
```

Notiamo i diversi colori con i quali VB scrive il codice[44]:

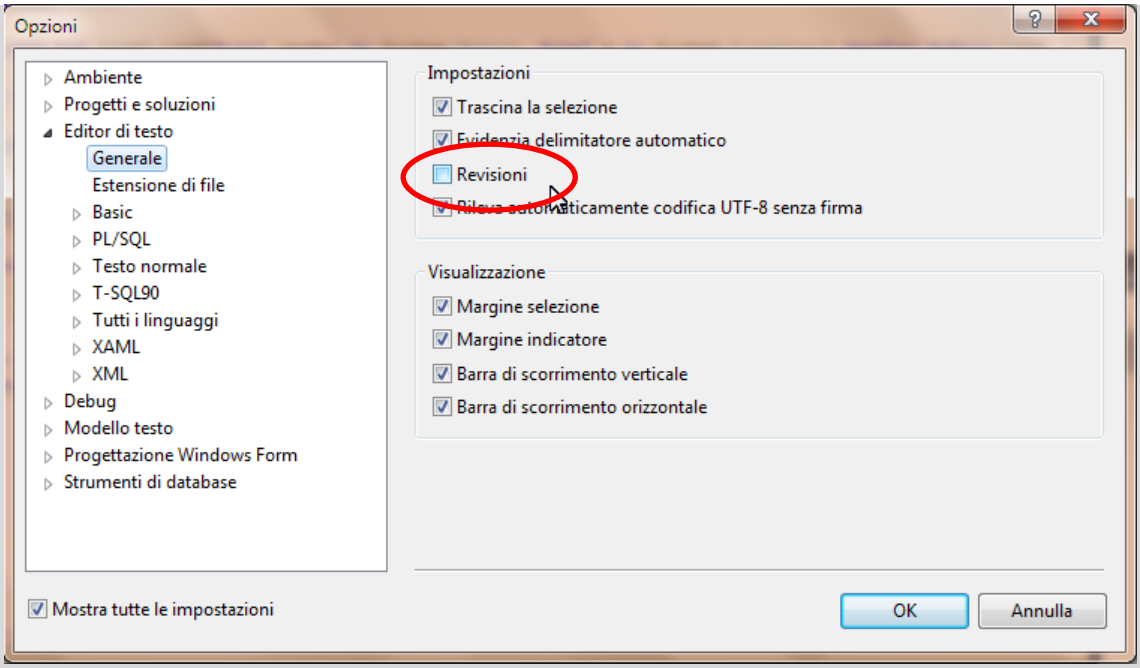
- le parole chiave della procedura, di cui il programmatore non può disporre, sono scritte in blu;
- le singole proprietà degli oggetti sono scritte in nero;
- i testi, parole o frasi scritti dal programmatore tra virgolette sono scritti in rosso;
- i commenti inseriti dal programmatore sono scritti in verde.

Notiamo ancora che cliccando una delle parole chiave della procedura:

- Private Sub
- Handles
- End Sub

tutte e tre queste parole chiave vengono evidenziate in grigio, a contrassegnare i limiti della procedura.
A sinistra della prima riga delle procedura, vediamo un quadratino con il segno -. Cliccandolo, otteniamo il risultato di chiudere la procedura. In questo caso il simbolo del quadratino diventa il segno +; cliccandolo, si ottiene l'effetto contrario di aprire la procedura. Questo meccanismo risulta utile quando il programmatore ha completato una procedura, magari lunga e complessa: chiudendola cliccando il simbolo -, ottiene il risultato di nasconderla alla vista, per spostarsi più agevolmente nella Finestra del Codice.
A sinistra di tutto il codice vediamo una sottile striscia verticale colorata, con tratti gialli o verdi. Le righe di codice affiancate da tratti gialli sono righe in lavorazione, le ultime sulle quali il programmatore è intervenuto, che non sono ancora state salvate. Le righe affiancate da tratti verdi sono invece righe di programmazione scritte in precedenza, ormai collaudate dal programmatore e salvate.
Questa striscia colorata verticale con tratti verdi e gialli può essere nascosta o visualizzata facendo un *clic* sul menu **Strumenti / Opzioni / Editor di testo / Revisioni**:

[44] Si descrivono qui i codici dei colori standard, impostati in modo automatico da VB, che possono però facilmente essere cambiati dal programmatore dal menu **Strumenti / Opzioni / Ambiente / Tipi di carattere e colori**.



Ancora, notiamo che nella barra blu in basso a destra nella Finestra del Codice compaiono alcune indicazioni relative alla posizione del mouse: qui è possibile leggere queste informazioni:

- il numero della riga sulla quale si trova il mouse (funzione utile quando VB segnala un errore di programmazione in una determinata riga);
- il numero della colonna sulla quale si trova il mouse;
- la posizione, all'interno della riga, del carattere sul quale si trova il mouse (funzione utile quando si vuole conoscere la lunghezza di una o più parole presenti nella riga);
- la sigla INS segnala che l'*editor* è in modalità di scrittura normale: ciò che il programmatore scrive si inserisce nel testo senza sostituire il testo precedente. Premendo il tasto INS sulla tastiera, compare invece la sigla SSC, a indicare che l'*editor* è in modalità di sovrascrittura: ciò che il programmatore scrive si sovrappone al testo precedente.

Le dimensioni dei caratteri del testo del Codice possono essere ingrandite, per leggere e verificare il testo più facilmente, tenendo premuto il tasto CTRL sulla tastiera e muovendo la rotella del mouse.

67: La sezione Generale del codice, o spazio dei nomi.

Apriamo un nuovo progetto di VB e facciamo un doppio *clic* con il mouse sul Form1: accediamo così alla Finestra del Codice, dove troviamo già impostate queste istruzioni:

```
Public Class Form1
```

```
        Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        End Sub

End Class
```

Facciamo un *clic* con il mouse a sinistra della prima riga (Public Class Form1), per abbassarla di una linea e per creare uno spazio vuoto **sopra di essa**.
Facendo un *clic* con il mouse nello spazio vuoto che si trova sopra Public Class Form1, ci accorgiamo di trovarci nella prima sezione del codice che è la sezione **Generale**, o **spazio dei nomi.**
Questa sezione è destinata a contenere istruzioni valide per tutta la Classe Form1 e, in particolare, è destinata a contenere:

- eventuali riferimenti ad archivi di software che il programmatore intende utilizzare, ma che non sono caricati automaticamente all'avvio di VB;
- eventuali modifiche alle impostazioni di base di un progetto VB.

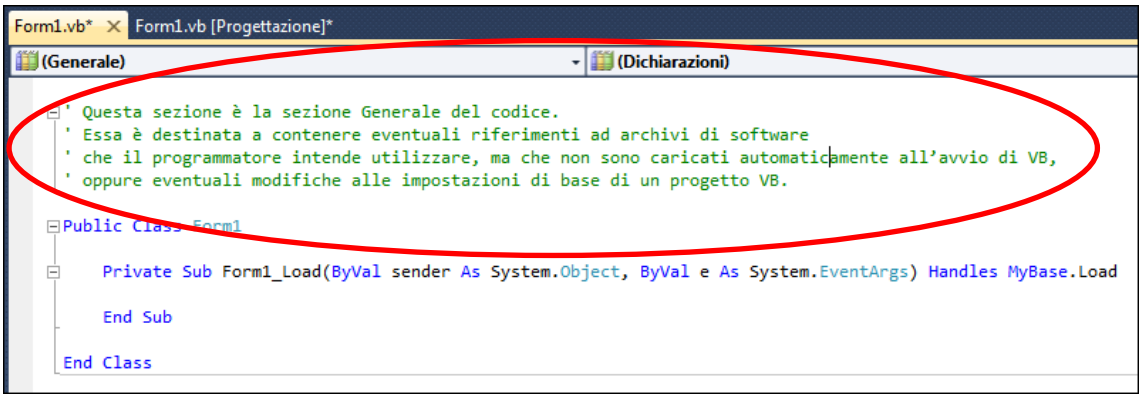


Figura 124: La sezione Generale del codice.

Negli esercizi e nelle applicazioni presentate in questo manuale accetteremo sempre le impostazioni di base di VB e non richiameremo classi di software aggiuntivi, per cui potremo ignorare l'esistenza di questa sezione **Generale** del codice, senza per questo precluderci la possibilità di creare programmi complessi e perfettamente funzionanti.
E' utile ricordare che se una riga di codice finisce per errore in questa sezione Generale, si riceve questo messaggio di errore: **Istruzione non valida nella spazio dei nomi**. Il problema si risolve cancellando ciò che si è scritto e riportandolo nello spazio della Classe Form1 o in una delle sue procedure.

68: Le procedure.

Nel codice che si trova sotto la sezione Generale, tra le righe

```
Public Class Form1


End Class
```

sono scritti tutti i comandi e le istruzioni che riguardano il Form1, gli oggetti in esso contenuti e gli eventi, le proprietà, le azioni riguardanti questi oggetti.
Diciamo subito che, essendo il Form1 *il padrone di casa*, tutte le istruzioni che lo riguardano direttamente all'interno di queste righe devono essere riferite al pronome **Me** e non al nome Form1.
VB segnala dunque come errori queste istruzioni:

```
Form1.BackColor = Color.Red
Form1.Size = New Size(800, 600)
```

che devono essere corrette utilizzando il pronome Me:

```
Me.BackColor = Color.Red
Me.Size = New Size(800, 600)
```

I diversi comandi e istruzioni sono raggruppati in parti o sezioni di codice ordinate, separate una dall'altra, ognuna delle quali è dedicata a uno degli eventi che possono accadere al Form o agli oggetti in esso contenuti durante lo svolgimento del programma.
Naturalmente non saranno scritte sezioni di codice per **tutti** gli eventi possibili per **tutti** gli oggetti presenti nel form, ma solo per quegli eventi che il programmatore ritiene necessario rilevare e gestire, ai fini del suo programma, per i quali scrive le istruzioni necessarie.
Ogni sezione di questo tipo si chiama **procedura**.
Una procedura è dunque l'elenco delle istruzioni che il programma esegue **quando accade un determinato evento** a **un determinato oggetto.**
In altri termini, quando il programma registra il verificarsi di un evento esegue la procedura collegata a questo evento dalla prima all'ultima riga e poi torna alla posizione di partenza (cioè torna ad attendere l'accadere di un altro evento).

Nella parte superiore della Finestra del Codice si aprono due menu a tendina:

- il menu a sinistra elenca gli oggetti presenti nel Form1 e consente di selezionare l'oggetto per il quale il programmatore vuole scrivere una procedura;
- il menu a destra elenca tutti gli eventi disponibili relativi all'oggetto selezionato, e consente di avviare la scrittura della procedura relativa a questo evento;

Ad esempio, per scrivere una procedura relativa all'evento **Paint** del **Form1** è sufficiente selezionare nel menu a **sinistra** gli eventi del **Form1** e nel menu a **destra** l'evento **Paint**:

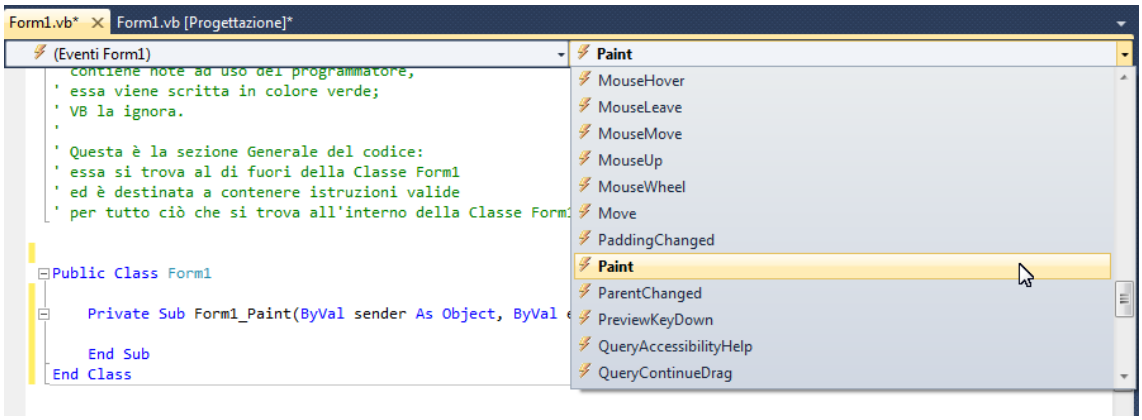


Figura 125: Selezione dell'evento Paint relativo al Form1.

Dopo avere selezionato l'evento Paint, notiamo che VB imposta in modo automatico la procedura destinata a gestire questo evento:

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

    End Sub
```

Analizziamo i termini di quanto scritto da VB:

- `Private` significa che la procedura appartiene alla Classe entro la quale è stata scritta (nell'esempio precedente al Form1) e non sarà visibile e tanto meno utilizzabile da altri form eventualmente presenti nel progetto.
- I termini `Sub …` e `End Sub` segnano rispettivamente l'inizio e il termine della procedura. Il programma eseguirà le righe di istruzioni contenute tra i due Sub ripetendo le stesse azioni in modo uguale ogni volta che registrerà l'evento che la procedura è incaricata di gestire (in questo caso l'evento Paint sul Form1).
- `Form1_Paint()` è il **nome proprio** della procedura. VB lo compone in modo automatico utilizzando il nome dell'oggetto interessato (il Form1) e il nome dell'evento che il programmatore vuol tenere sotto controllo (Paint), uniti da un trattino. Si tratta tuttavia di un nome che il programmatore può modificare a suo piacere, soprattutto per capire immediatamente, in seguito, quando il codice sarà più complesso, di cosa tratta questa procedura.
  La procedura funziona allo stesso modo e senza problemi, ad esempio, assegnandole il nome proprio **Disegna**:

```
    Private Sub Disegna(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

    End Sub
```

- `(ByVal sender As Object, ByVal e As System.Windows.Forms.PaintEventArgs)`: i due elementi inseriti tra parentesi sono dei parametri supplementari, che registrano alcune informazioni supplementari relative all'evento **Paint**, informazioni che vedremo in dettaglio in seguito.

- La frase che conclude la prima riga, `Handles Me.Paint`, indica che questa procedura gestisce (`Handles`) l'evento Paint riferito al Form1 (`Me`).

Per ogni evento che il programmatore vuole gestire, di ogni controllo, deve essere scritta una specifica routine di istruzioni.
È possibile scrivere più procedure che riguardano eventi diversi di uno stesso controllo. Ad esempio, se di un pulsante **Button1** vogliamo tenere sotto controllo gli eventi **Click**, **MouseEnter** e **DragDrop** è necessario scrivere una procedura diversa per ognuno dei tre eventi.
E' possibile che un controllo appaia nel form senza avere alcuna procedura associata: in questo caso il controllo in questione si limiterà a comparire nel form, ma nel corso dell'esecuzione del programma non produrrà alcun effetto.
Lo stesso discorso vale per gli eventi: per ogni evento che il programmatore vuole tenere in considerazione nello svolgimento del programma (un *clic* con il mouse, la pressione di un tasto o di un pulsante, la scrittura di una parola…) deve essere scritta una procedura nel codice.
Un evento produce effetti se ad esso corrisponde una procedura con istruzioni scritte nel codice; un evento rimane senza effetti se non sono previste righe di codice dedicate a questo evento.
Ad esempio, un pulsante di comando disegnato nel form avverte *sempre e comunque* il *clic* del mouse e *si abbassa* sotto questo *clic* ma, se non è scritta nel codice una procedura con istruzioni associate all'evento **Click**, il pulsante non produce alcun effetto nel programma.

69: Il parametro sender.

Abbiamo visto nel paragrafo precedente che una procedura registra in due parametri, scritti tra parentesi, alcune informazioni particolari sull'evento che è incaricata di gestire.
Questi due parametri sono il parametro **sender** e il parametro **e**.
Li vediamo, ad esempio, in questa procedura che gestisce il *clic* del mouse su un pulsante Button1:

```
    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

    End Sub
```

- Il parametro **e** (= informazioni sull'evento) varia a seconda del tipo di evento gestito dalla procedura. Lo utilizzeremo spesso negli esercizi dedicati alla grafica, per cui rimandiamo la lettrice o il lettore a quella parte del manuale.
- Il parametro **sender** (= informazioni sul mittente) invece è di uso costante in tutti gli eventi e di facile comprensione: questo parametro **corrisponde al controllo** che è stato oggetto dell'evento gestito dalla procedura.

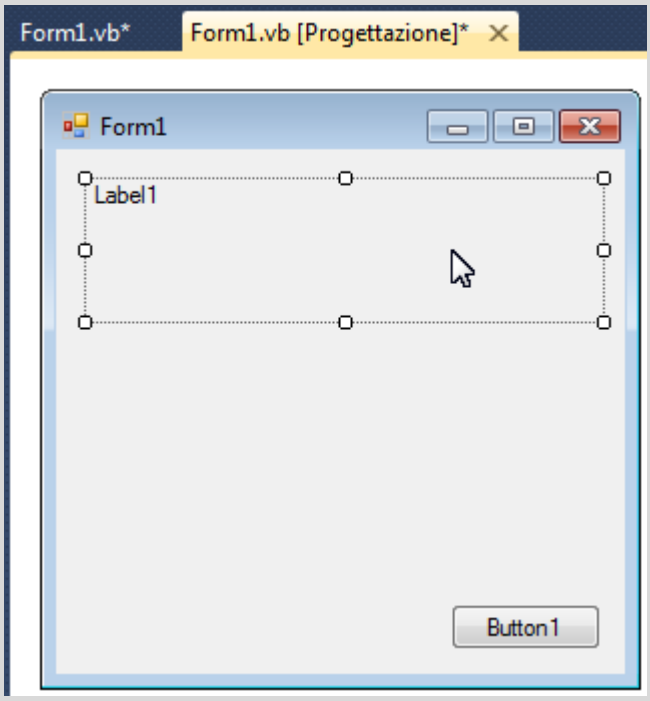
La procedura seguente gestisce l'evento **Click** su un pulsante **Button1**; il parametro **sender** corrisponderà dunque al pulsante Button1, assumendo tutte le proprietà e le capacità di azione del Button1:

```
    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

    End Sub
```

Ne avremo una prova nel prossimo esercizio.

Esercizio 36: Utilizzo del parametro sender.

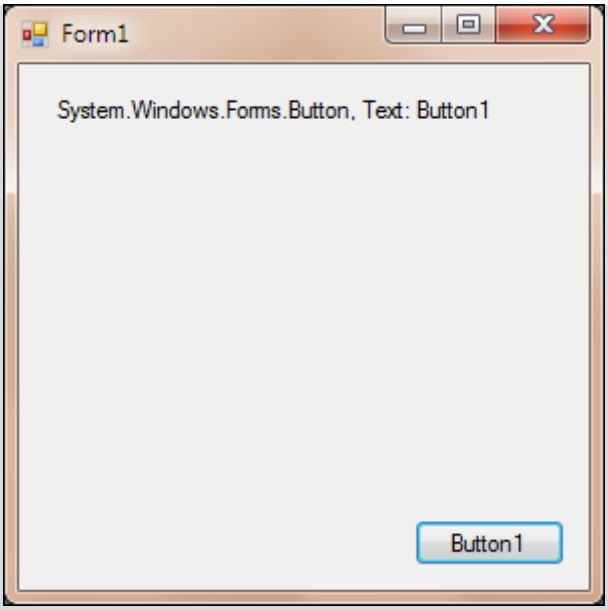
Apriamo un nuovo progetto. Inseriamo nel Form1 un pulsante **Button1** e un controllo **Label1** con la proprietà **AutoSize = False**, come in questa immagine:



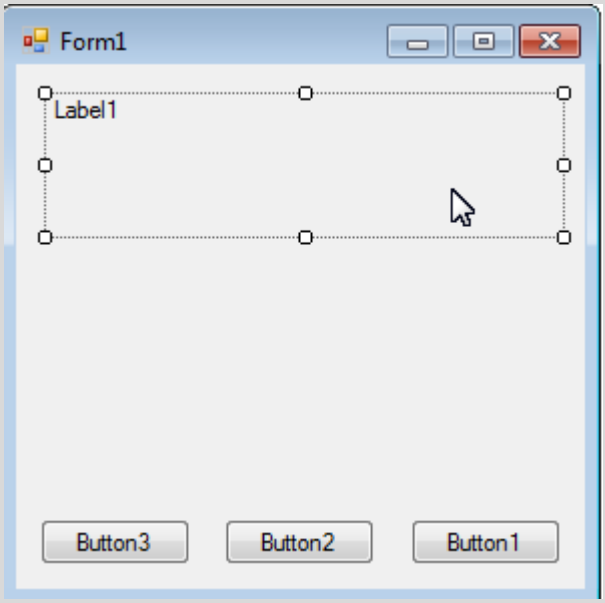
Scopo del programma è visualizzare nella Label1 il parametro **sender** registrato dalla procedura che gestisce il *clic* sul Button1.
Facciamo un doppio *clic* sul pulsante Button1 e completiamo il listato con una riga di istruzioni, come segue:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Label1.Text = sender.ToString

    End Sub

End Class
```

Al verificarsi dell'evento *clic* del mouse sul pulsante, dunque, la procedura utilizzerà il parametro **sender** per visualizzare all'interno della Label1 il nome del controllo sul quale si è verificato il clic.
Ecco un'immagine del programma in esecuzione:



Questo è un uso elementare, inutile dal punto di vista pratico, del parametro **sender**, ma è solo il primo passo di un cammino che ci porterà lontano.
Fermiamo l'esecuzione del programma e torniamo a visualizzare la Finestra Progettazione. Inseriamo nel progetto altri due pulsanti (**Button2** e **Button3**) come in questa immagine:



Ora vogliamo fare in modo che, al verificarsi dell'evento *clic* del mouse su uno dei tre pulsanti, la procedura utilizzi il parametro sender per sapere il nome del controllo sul quale si è verificato il clic, e lo visualizzi all'interno della Label1.
E' ovviamente possibile **replicare** la procedura che abbiamo già scritto per il *clic* sul Button1:

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Label1.Text = sender.ToString

    End Sub
```

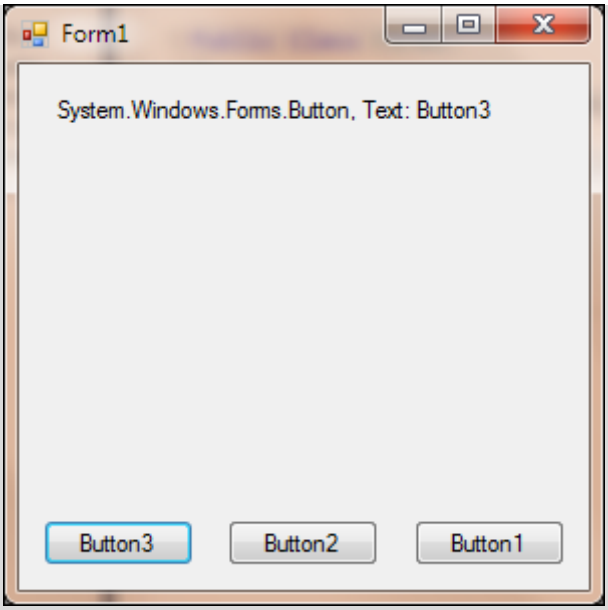
```
    Private Sub Button2_Click(sender As Object, e As System.EventArgs) Handles
Button2.Click

        Label1.Text = sender.ToString

    End Sub
```

```
    Private Sub Button3_Click(sender As Object, e As System.EventArgs) Handles
Button3.Click

        Label1.Text = sender.ToString

    End Sub

End Class
```

Oppure è possibile fare gestire alla prima procedura **Button1_Click** anche i *clic* sugli altri due pulsanti, accodando all'evento **Button1.Click** gli eventi **Button2.Click** e **Button3**.**Click**:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click, Button2.Click, Button3.Click

        Label1.Text = sender.ToString

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione, dopo un *clic* sul **Button3**:



Il parametro **sender** dunque corrisponde al pulsante che è stato premuto e può essere utilizzato per fare eseguire al programma azioni diverse a seconda del pulsante premuto.
Ne vediamo un esempio nel codice seguente, dove:

- un *clic* sul primo pulsante scrive un testo nella Label1;
- un *clic* sul secondo pulsante cambia il colore dello sfondo del Form1;
- un *clic* sul terzo pulsante termina il programma.

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click, Button2.Click, Button3.Click

        If sender.Text = "Button1" Then Label1.Text = "E' stato premuto il primo
pulsante"
        If sender.Text = "Button2" Then Me.BackColor = Color.Yellow
        If sender.Text = "Button3" Then End

    End Sub

End Class
```

Con solo tre pulsanti da tenere sotto controllo, la procedura `Button1_Click` è di semplice creazione. Se i pulsanti nel Form1 fossero una dozzina, però, la prima riga della procedura diventerebbe complessa da scrivere e poco maneggevole:

```
    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click, Button2.Click, Button3.Click, Button4.Click, Button5.Click,
Button6.Click, Button7.Click, Button8.Click, Button9.Click, Button10.Click,
Button11.Click, Button12.Click
```

Si può ovviare a questo problema ricorrendo a due modalità diverse, che chiameremo

- metodo **manuale** e
- metodo **automatico**.

Torniamo alla scrittura originale della procedura che gestisce il *clic* del mouse sul primo pulsante:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Label1.Text = sender.ToString

    End Sub

End Class
```

Con il metodo **manuale**, possiamo fare in modo che al momento del caricamento in memoria del form (evento **Me.Load**), ogni pulsante presente nel Form **agganci** a questa procedura il suo evento **Click**:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Label1.Text = sender.ToString

    End Sub

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' in apertura del Form1,
        ' aggiunge l'evento del clic del mouse sugli altri pulsanti
        ' alla procedura Button1_Click:

        AddHandler Button2.Click, AddressOf Button1_Click
        AddHandler Button3.Click, AddressOf Button1_Click

    End Sub

End Class
```

Il metodo manuale ha evidentemente dei limiti quantitativi: se nel Form1 ci fossero decine di pulsanti, la procedura Form1_Load e il lavoro del programmatore si dilungherebbero oltre misura.
Con il metodo **automatico** il problema si risolve in questo modo:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Label1.Text = sender.ToString

    End Sub

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load
```

```
        ' in apertura del Form1, il programma
        ' passa in rassegna tutti i pulsanti presenti nel form e
        ' aggiunge l'evento del clic del mouse su ogni pulsante
        ' alla procedura Button1_Click:

        For Each Pulsante As Button In Me.Controls
            AddHandler Pulsante.Click, AddressOf Button1_Click
        Next

    End Sub

End Class
```

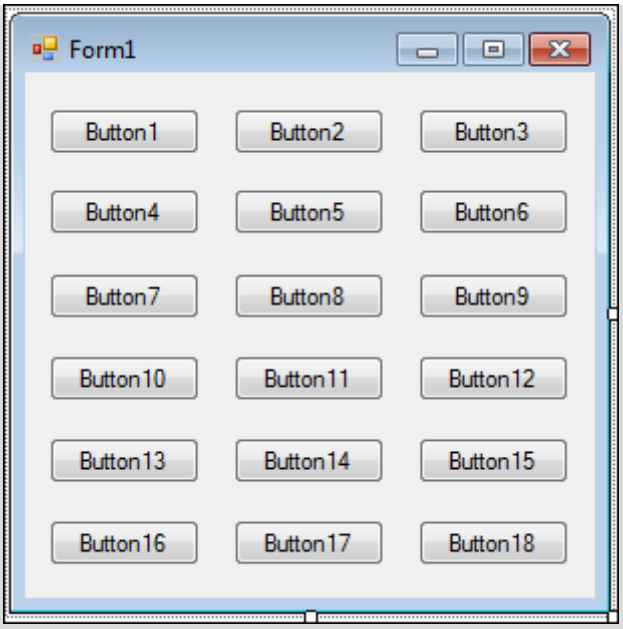
Le tre righe di istruzioni evidenziate in giallo sono in grado ovviamente di gestire un numero indefinito di pulsanti.
Grazie ad esse, il programma effettua queste operazioni:

- passa in rassegna tutti i pulsanti che sono presenti nel Form e
- aggancia l'evento **Click** di ognuno di essi alla procedura **Button1_Click**.

Oltre al metodo manuale e al metodo automatico che abbiamo visto in questo esercizio, è possibile accodare gli eventi Click di tutti i pulsanti presenti in un Form a una unica procedura con un terzo metodo che chiameremo **metodo grafico**. Ne vedremo un esempio nel prossimo esercizio.

Esercizio 37: Metodo grafico per accodare gli eventi Click di un gruppo di pulsanti a una unica procedura.

Apriamo un nuovo progetto e inseriamo nel Form 18 pulsanti Button, come nell'immagine seguente:



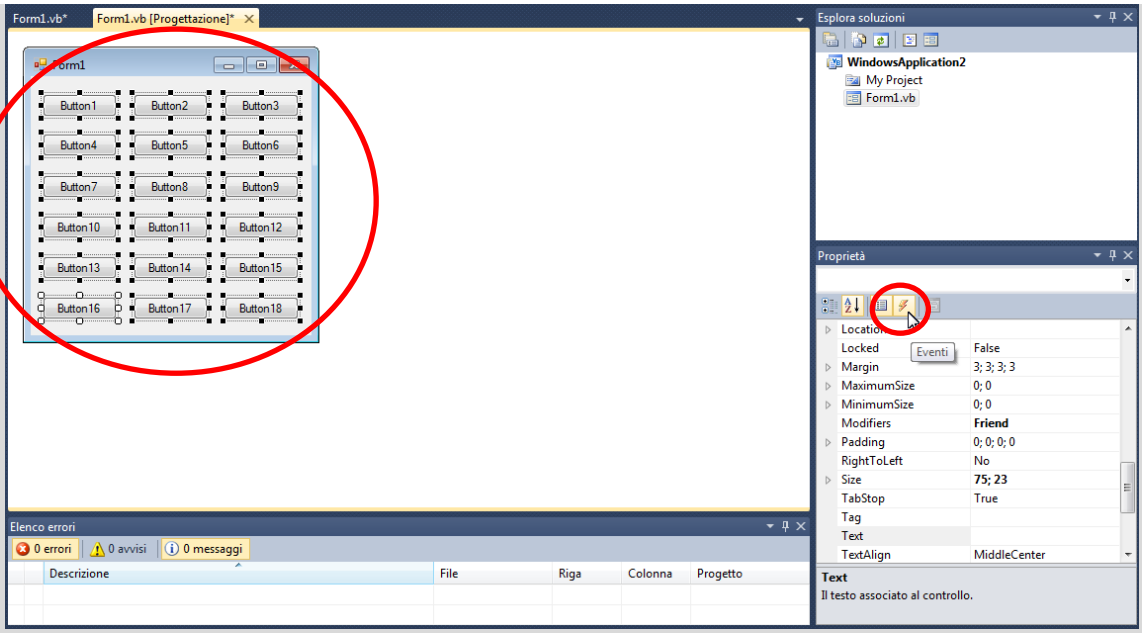
Lo schema di funzionamento del programma è questo: un *clic* con il mouse su uno dei 18 pulsanti farà comparire nella barra con il titolo del Form1 il testo che compare sul pulsante premuto (Button1, Button2, …).
Dobbiamo dunque creare una procedura unica in grado di gestire il *clic* del mouse su tutti i 18 pulsanti e che contenga questa istruzione:

```
Me.Text = sender.text
```

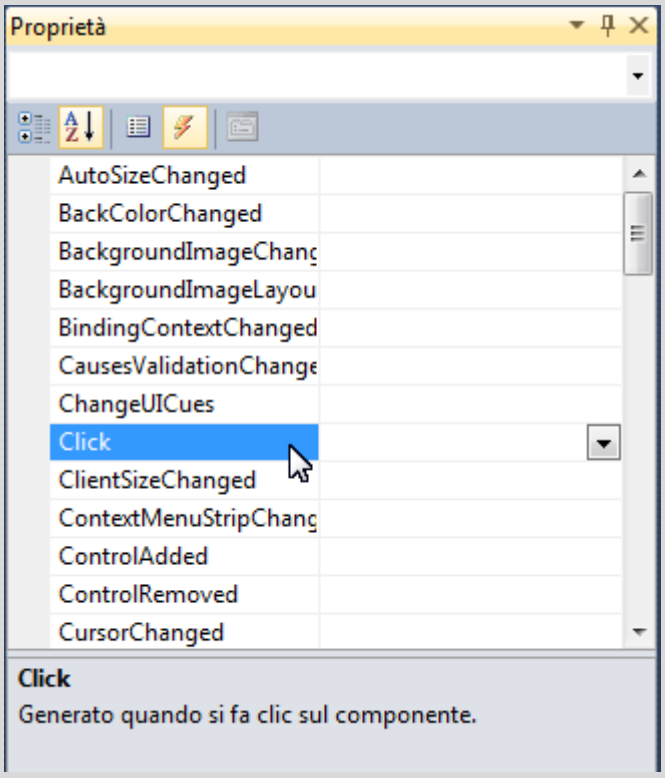
Per creare questa procedura con il **metodo automatico grafico**,

1. selezioniamo tutti i pulsanti con uno dei due metodi che conosciamo[45];
2. controlliamo che tutti i pulsanti mostrino le maniglie di dimensionamento, cioè siano selezionati;
3. nella **Finestra Proprietà**, facciamo un *clic* con il mouse sul pulsante con l'icona del **lampo** (**Eventi**):

[45] Si veda il paragrafo **Per selezionare** un gruppo di controlli, a pag. 143.



4. Nell'elenco degli eventi, facciamo un doppio *clic* con il mouse sull'evento che ci interessa: **Click**.

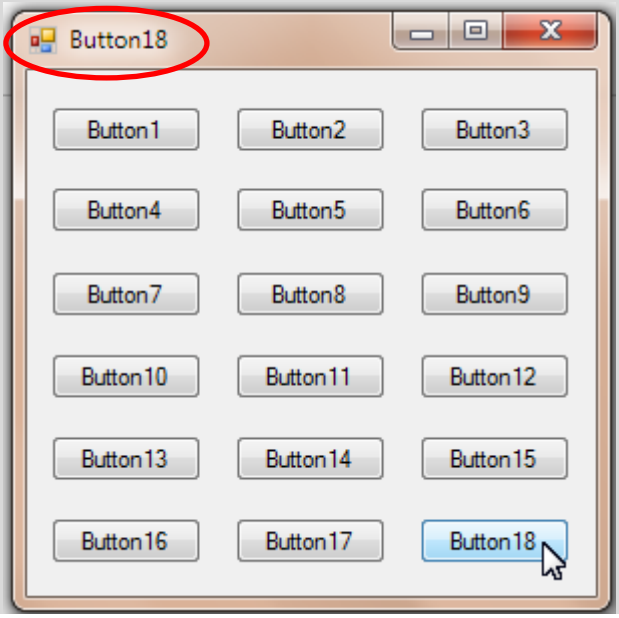


Si crea così in modo automatico, nella Finestra del Codice, la procedura destinata a gestire il *clic* del mouse su tutti i pulsanti selezionati, nella quale non resta da fare altro che scrivere l'istruzione evidenziata in giallo:

```
Public Class Form1
```

```
    Private Sub Button16_Click(sender As System.Object, e As System.EventArgs)
Handles Button9.Click, Button8.Click, Button7.Click, Button6.Click,
Button5.Click, Button4.Click, Button3.Click, Button2.Click, Button18.Click,
Button17.Click, Button16.Click, Button15.Click, Button14.Click, Button13.Click,
Button12.Click, Button11.Click, Button10.Click, Button1.Click

        Me.Text = sender.text

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



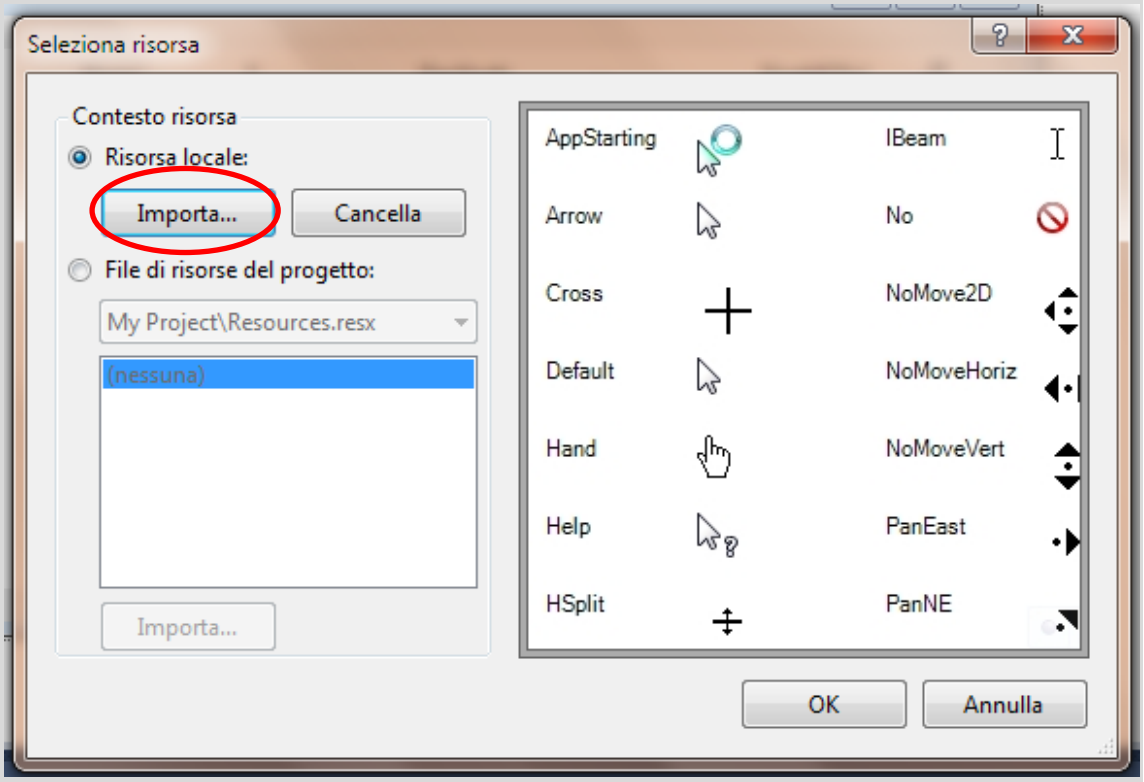
70: Gestire una famiglia di controlli.

Nel prossimo esercizio vedremo come gestire, utilizzando il parametro **sender**, una **famiglia di controlli**, cioè un gruppo di controlli aventi caratteristiche simili.
La **famiglia di controlli** che vedremo all'opera in questo esercizio è composta da 28 controlli Label; ogni singolo controllo della famiglia, per potere essere individuato singolarmente, è contraddistinto da un numero progressivo scritto nella sua proprietà **TabIndex**.

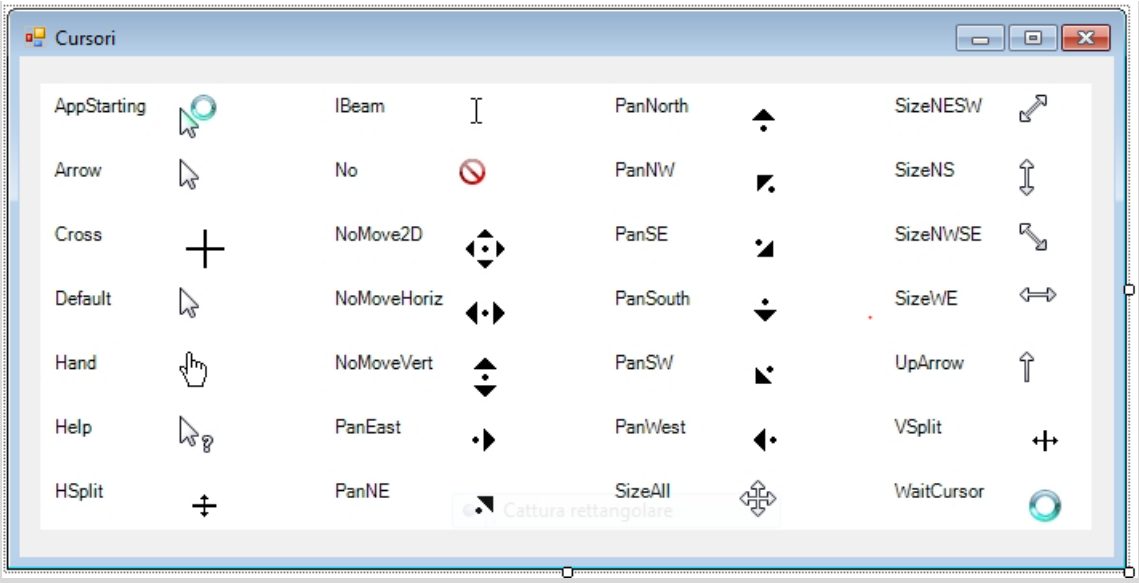
Esercizio 38: Cursori.

Questo programma visualizza i cursori del mouse presenti in VB: in un'immagine di fondo si vedranno tutti i cursori disponibili. L'utente, cliccando uno di questi cursori, lo sceglierà come cursore in uso, al posto del cursore precedente.
Apriamo un nuovo progetto, lo denominiamo **Cursori**.
Impostiamo queste proprietà del **Form1**:

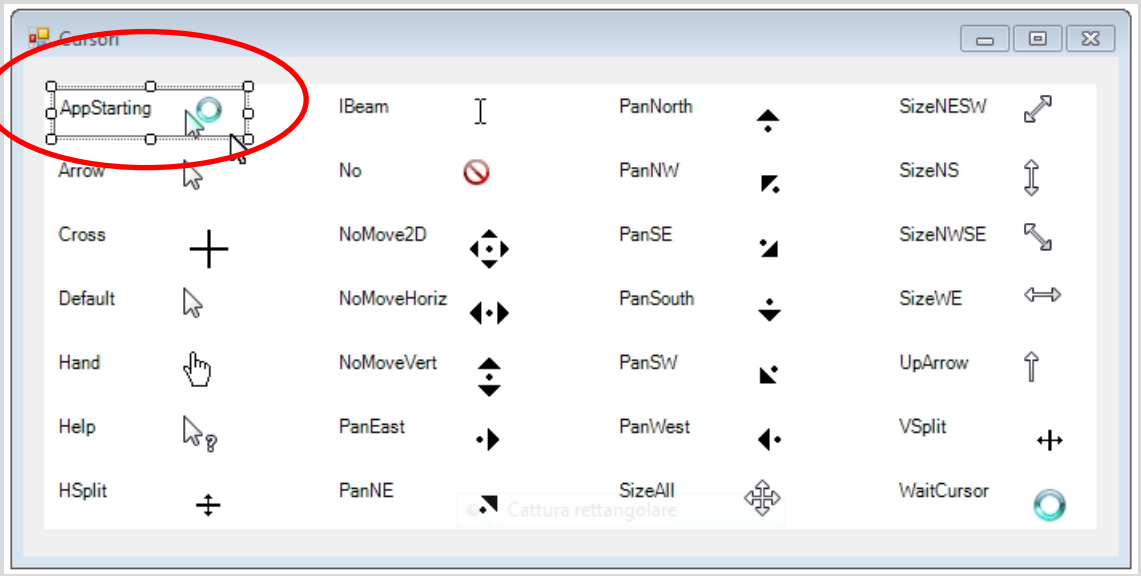
- **Size = 700; 350**
- **Text = Cursori**
- **StartPosition = CenterScreen**
- **BackgroundImage**: facciamo un *clic* sul pulsante con i tre puntini. Apriamo la finestra Selezione risorsa, facciamo un *clic* su Risorsa locale e andiamo a scegliere l'immagine **cursori.jpg** che si trova nella cartella **Documenti \ A scuola con VB \ Immagini.**



Ecco l'immagine del **Form1** con l'immagine di fondo:



In questa immagine si vedono tutti i cursori che VB mette a disposizione del programmatore, per essere utilizzati in determinate occasioni al posto del cursore di default (cioè al posto del cursore con la freccia bianca che compare normalmente)[46].
Ora dobbiamo fare in modo che ognuno dei cursori visibili nell'immagine, una volta cliccato con il mouse, diventi il cursore in uso al posto del cursore precedente.
Per fare questo iniziamo collocando su ogni cursore un controllo **Label** trasparente che, per quanto non visibile, può essere cliccato dall'utente del programma.
Sapendo quale Label è stata cliccata dall'utente, sapremo quale cursore assegnare al mouse.
Collochiamo dunque la prima **Label** sulla immagine del primo cursore:



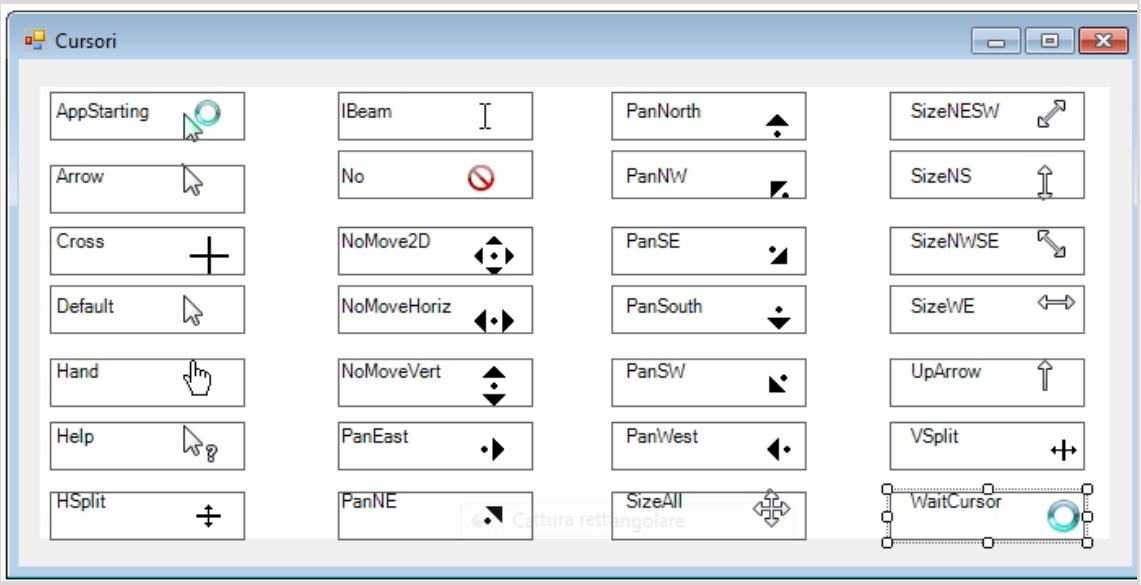
[46] L'elenco dei cursori può essere visto anche facendo un *clic* sul pulsante con la freccina nera, di fianco alla proprietà **Cursor** di qualsiasi controllo.

Facciamo un *clic* sulla Label, accediamo alla Finestra Proprietà e impostiamo alcune proprietà di questa **Label1**:

- **AutoSize = False**
- **BackColor = Transparent**
- **BorderStyle = FixedSingle**
- **Size = 120; 30**
- **Text =** *nulla* (cancellare la scritta Label1 e lasciare la proprietà vuota)

Una volta impostate le sue proprietà, facciamo un *clic* all'interno di questa **Label1** in modo da visualizzare le maniglie di dimensionamento. Ora copiamo questa Label1 con un *clic* sul pulsante **Copia**, nella barra delle icone, o premendo i pulsanti **CTRL + V** sulla tastiera.

Ora possiamo riprodurre questa Label, con un *clic* sul pulsante **Incolla**, nella barra delle icone, o premendo i pulsanti **CTRL + C** sulla tastiera. La riproduciamo più volte, sino ad avere una Label sopra l'immagine di ogni cursore. Arriveremo ad avere, sull'ultimo cursore, la **Label28**:

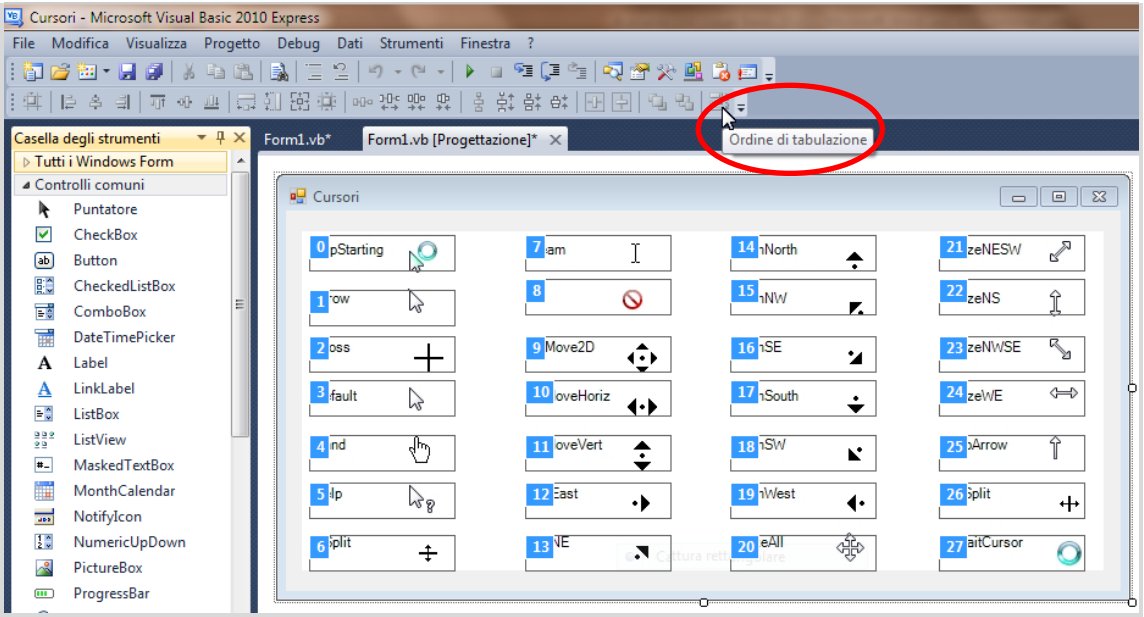


Le 28 Label che abbiamo inserito nel Form1 hanno la proprietà **TabIndex** impostata automaticamente da VB con un numero progressivo che va da 0 a 27: è questo numero che useremo, nel corso del programma, per sapere quale label è stata cliccata dall'utente e dunque quale cursore deve essere assegnato al mouse.

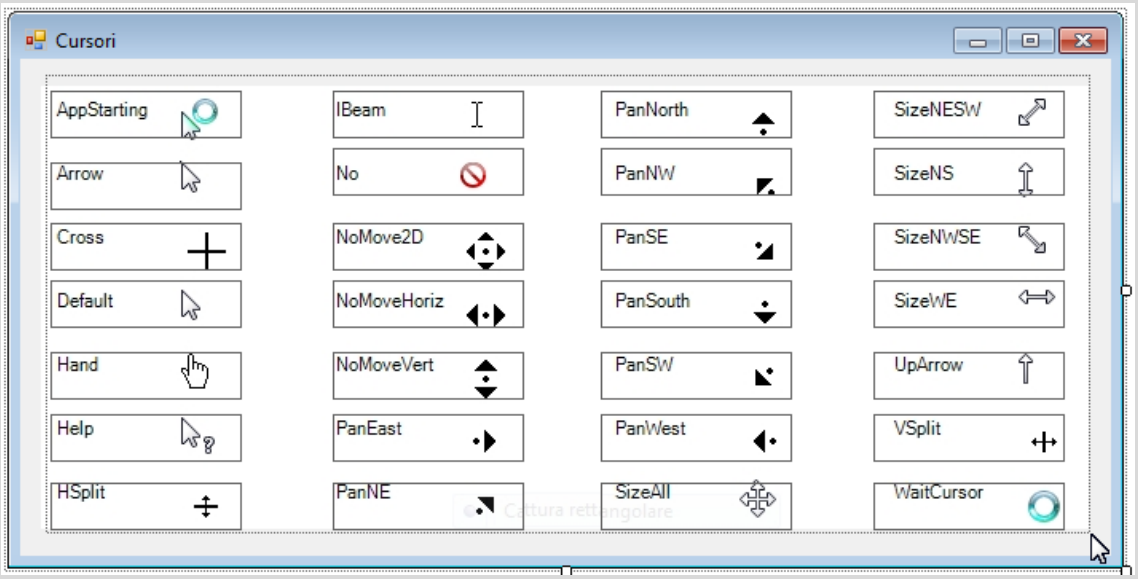
Possiamo verificare la proprietà **TabIndex** delle 28 Label in questo modo:

- Facciamo un *clic* sul menu Visualizza / Barre degli strumenti /Layout.
- Nella barra a icone che viene visualizzata, facciamo un *clic* sull'ultimo pulsante a destra : **Ordine di tabulazione**.

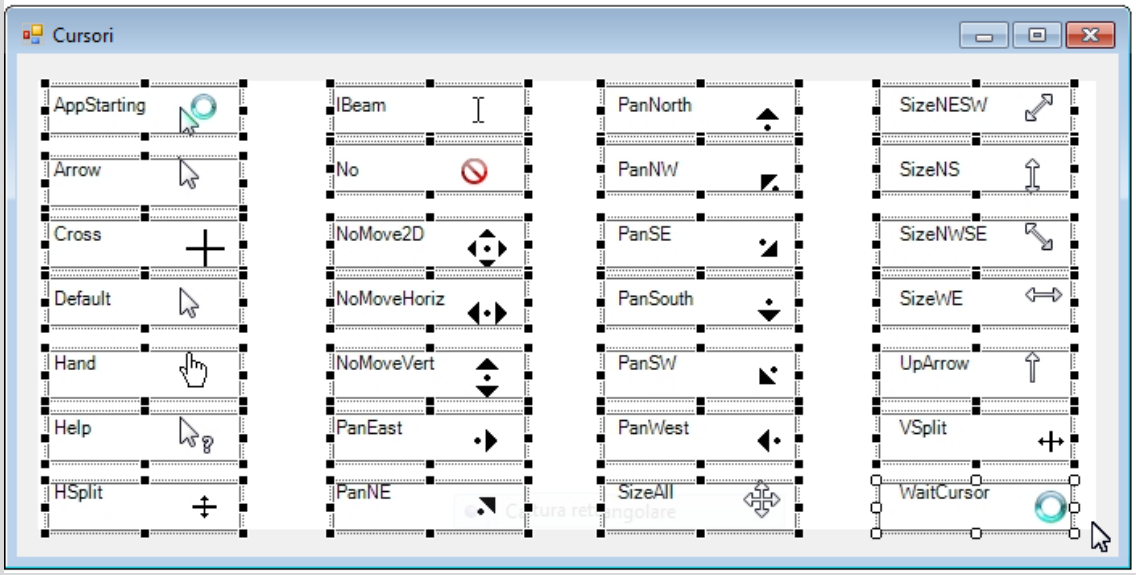
Vediamo così, nei quadratini a fondo azzurro, la proprietà **TabIndex** di tutte le Label:



Facciamo di nuovo un *clic* sul pulsante **Ordine di tabulazione**, per tornare a visualizzare la Finestra di Progettazione in modalità normale.
Le 28 Label compaiono con la proprietà **BorderStyle = Fixed Single** per comodità del programmatore, ma il riquadro rettangolare che le contorna è poco gradevole alla vista. Possiamo togliere tutti i contorni in questo modo: trasciniamo il puntatore attorno alle Label e tracciamo un rettangolo che le comprenda tutte:



Rilasciando il puntatore notiamo che tutte le 28 label risultano selezionate:



Nella Finestra Proprietà, possiamo impostare la proprietà **BorderStyle = None**; in questo modo tutte le Label saranno visualizzate senza contorno.
Passiamo ora alla scrittura del codice del programma.
Facciamo un doppio *clic* sulla **Label1**.
Accediamo così alla Finestra del Codice, dove troviamo già impostata la procedura che gestisce l'evento del *clic* del mouse su questa Label:

```
    Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Label1.Click

    End Sub
```

Questa procedura per ora gestisce l'evento del *clic* del mouse SOLO sulla Label1, ma ne faremo una procedura generica che gestirà l'evento del *clic* del mouse su TUTTE le 28 Label.
A questo scopo, cancelliamo l'ultima parte della procedura (Handles Label1.Click) e diamo alla procedura un nuovo nome: **SceltaCursore**.

```
    Private Sub SceltaCursore(ByVal sender As System.Object, ByVal e As
System.EventArgs)

    End Sub
```

Ora abbiamo una procedura che si chiama **SceltaCursore** ma, per ora, non le è assegnato alcun evento da gestire, in quanto abbiamo cancellato l'evento Handles Label1.Click e non lo abbiamo ancora sostituito. Inseriremo qui, tra poco, la gestione dell'evento del *clic* del mouse su tutte le 28 label.
Notiamo, tra le parentesi, il parametro **sender (= mittente)**: quando questa procedura gestirà l'evento del *clic* del mouse su tutte le Label, il parametro **sender** memorizzerà l'oggetto cliccato e la proprietà **sender.Tabindex** ci darà il numero corrispondente alla Label cliccata.

Ad esempio, se verrà cliccata la **Label10** il parametro **sender** memorizzerà l'oggetto **Label10**, la cui proprietà **sender.Tabindex** è a 9, e sarà dunque possibile assegnare al mouse il cursore corrispondente al numero 9.
Completiamo la procedura **SceltaCursore** in questo modo:

```
Private Sub SceltaCursore(ByVal sender As System.Object, ByVal e As
System.EventArgs)

    Select Case sender.TabIndex
        Case 0
            Me.Cursor = Cursors.AppStarting
        Case 1
            Me.Cursor = Cursors.Arrow
        Case 2
            Me.Cursor = Cursors.Cross
        Case 3
            Me.Cursor = Cursors.Default
        Case 4
            Me.Cursor = Cursors.Hand
        Case 5
            Me.Cursor = Cursors.Help
        Case 6
            Me.Cursor = Cursors.HSplit
        Case 7
            Me.Cursor = Cursors.IBeam
        Case 8
            Me.Cursor = Cursors.No
        Case 9
            Me.Cursor = Cursors.NoMove2D
        Case 10
            Me.Cursor = Cursors.NoMoveHoriz
        Case 11
            Me.Cursor = Cursors.NoMoveVert
        Case 12
            Me.Cursor = Cursors.PanEast
        Case 13
            Me.Cursor = Cursors.PanNE
        Case 14
            Me.Cursor = Cursors.PanNorth
        Case 15
            Me.Cursor = Cursors.PanNW
        Case 16
            Me.Cursor = Cursors.PanSE
        Case 17
            Me.Cursor = Cursors.PanSouth
        Case 18
            Me.Cursor = Cursors.PanSW
        Case 19
            Me.Cursor = Cursors.PanWest
        Case 20
            Me.Cursor = Cursors.SizeAll
        Case 21
            Me.Cursor = Cursors.SizeNESW
        Case 22
            Me.Cursor = Cursors.SizeNS
        Case 23
            Me.Cursor = Cursors.SizeNWSE
        Case 24
            Me.Cursor = Cursors.SizeWE
```

```
        Case 25
            Me.Cursor = Cursors.UpArrow
        Case 26
            Me.Cursor = Cursors.VSplit
        Case 27
            Me.Cursor = Cursors.WaitCursor
    End Select

End Sub
```

Possiamo tradurre le procedura in questo modo:

- quando avverti il *clic* del mouse su una **Label**
- controlla la sua proprietà **TabIndex** e scegli uno di questi casi:
- se la proprietà TabIndex è uguale a 0 assegna al Form1 il cursore AppStarting;
- se la proprietà TabIndex è uguale a 1 assegna al Form1 il cursore Arrow;
- se la proprietà TabIndex è uguale a 2 assegna al Form1 il cursore Cross;
- e così di seguito, sino al completamento delle 28 label.

Come abbiamo detto prima, però, la riga iniziale di questa procedura

```
Private Sub SceltaCursore(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

è ancora incompleta, perché non le è stato ancora assegnato alcun evento da gestire. Lo facciamo scrivendo una nuova procedura che il programma eseguirà al suo avvìo, al momento del caricamento in memoria del Form1.
Facciamo un doppio *clic* sul **Form1** e torniamo nella Finestra del Codice, dove troviamo la procedura del caricamento del Form già impostata:

```
    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs)
Handles MyBase.Load

    End Sub
```

Nello spazio centrale, vuoto, scriviamo queste istruzioni:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Each Etichetta As Label In Me.Controls
            AddHandler Etichetta.Click, AddressOf SceltaCursore
        Next

    End Sub
```

La prima e la terza riga aggiunte creano un ciclo di istruzioni che passerà in rassegna tutte le 28 label presenti nel form per effettuare queste operazioni:

- per ogni **Label** che si trova nel form
- assegna a questa **Label** il nome **Etichetta**
- aggiungi la gestione dell'evento **Etichetta.Click** alla procedura **SceltaCursore**, poi
- passa alla prossima label.

Abbiamo così assegnato alla procedura **SceltaCursore** la gestione dei *clic* sulle 28 Label.

Ora il programma è completo e possiamo mandarlo in esecuzione.

71: Creare controlli dalla Finestra del Codice.

L'inserimento di un controllo in un form, prelevandolo dalla Casella degli Strumenti, è una operazione semplice, pratica e ormai abituale per la lettrice o il lettore.
Si tratta di un'operazione che, in determinate situazioni, può essere convenientemente effettuata anche scrivendo apposite istruzioni nella Finestra del Codice del programma, senza effettuare alcuna azione grafico/visiva di trascinamento e di collocazione del controllo nel form.
E' possibile creare, dalla Finestra del Codice, qualsiasi controllo, dandogli un nome proprio e dichiarando a quale classe esso appartiene; il nuovo controllo assume tutte le caratteristiche del controllo della stessa classe che si trova nella Casella degli Strumenti.
Ad esempio, il listato seguente effettua queste operazioni:

- crea un nuovo controllo di nome **PulsanteMio**, appartenente alla classe dei pulsanti Button[47],
- lo dimensiona,
- lo colloca nel form all'avvio del programma e
- lo aggiunge all'elenco dei controlli presenti nel form.

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim PulsanteMio As New Button
        PulsanteMio.Size = New Size(75, 23)
        PulsanteMio.Location = New Point(12, 12)
        PulsanteMio.Text = "PulsantMio"
        Me.Controls.Add(PulsanteMio)

    End Sub

End Class
```

[47] Il termine ***crea*** è qui usato in modo improprio. In effetti, il listato ***non crea*** il nuovo pulsante, ma **crea una nuova istanza della Classe Button**, cioè dà una nuova realizzazione concreta alla programmazione che gestisce la Classe Button. Anche quando si preleva un controllo dalla Casella degli Strumenti e lo si colloca in un form non si crea un nuovo controllo, ma si crea una nuova istanza della Classe di programmazione di quel particolare tipo di controllo.

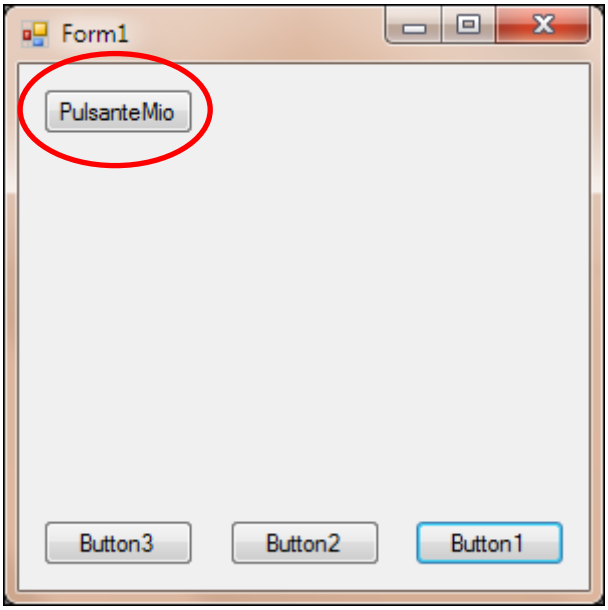
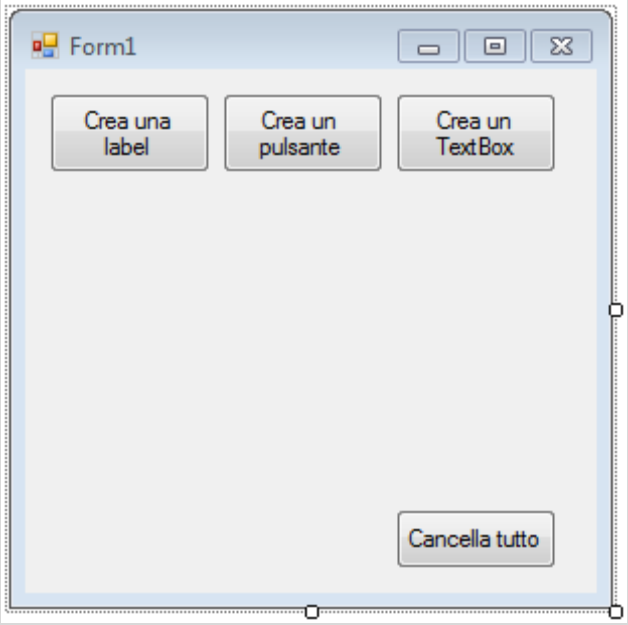


Figura 126: Creazione di un pulsante dalla Finestra del Codice.

Il nuovo pulsante **PulsanteMio** è in tutto e per tutto simile agli altri pulsanti collocati nel form; ne ha le stesse porprietà ed è in grado di riconoscere gli stessi eventi.
Nell'esercizio seguente vedremo come creare, dalla Finestra Codice, una Label, un pulsante Button e un TextBox che non erano presenti nel Form all'avvio del programma.

Esercizio 39: Creazione di tre controlli durante l' esecuzione di un programma.

Apriamo un nuovo progetto e inseriamo nel Form1 quattro pulsanti Button, come in questa immagine:



Impostiamo queste loro proprietà:

	Name	**Text**
Button1	btnCreaLabel	Crea una label
Button2	btnCreaPulsante	Crea un pulsante
Button3	btnCreaTextBox	Crea un TextBox
Button4	btnCancella	Cancella tutto

Apriamo la Finestra del Codice.
Nel menu **Generale** facciamo un *clic* sul controllo **btnCreaLabel** e nel menu **Dichiarazioni** facciamo un *clic* sull'evento **Click**.
Si imposta così la procedura di gestione dell'evento del *clic* del mouse sul pulsante **btnCreaLabel**, che avrà il compito di creare un nuova Label nel form:

```
    Private Sub btnCreaLabel_Click(sender As System.Object, e As
System.EventArgs) Handles btnCreaLabel.Click

    End Sub
```

Al suo interno dobbiamo scrivere queste istruzioni, che creano una nuova Label e ne impostano alcune proprietà:

```
        Dim lblNuova As New Label

        With lblNuova
             .Name = "lblNuova"
             .BackColor = Color.Blue
```

```
            .ForeColor = Color.Yellow
            .Location = New Point(12, 58)
            .Size = New Size(80, 40)
            .Text = "Questa è una label nuova"
        End With

        Me.Controls.Add(lblNuova)
```

- La prima riga avvia la creazione dell'oggetto chiamato **lblNuova**, che sarà un nuovo oggetto della Classe di oggetti **Label**.
- Nelle righe successive, da **With lblNuova** a **End With**, si impostano alcune proprietà della nuova Label (il nome, il colore di fondo, il colore di primo piano, la posizione, le dimensioni, il testo).
- Il ciclo di comandi che va da **With lblNuova** a **End With** è una comodità offerta al programmatore. VB dà per inteso che tutto quanto è scritto all'interno di questo ciclo riguarda il controllo **lblNuova**, senza ripeterne ogni volta il nome. Avremmo potuto ottenere lo stesso risultato scrivendo per esteso tutte le proprietà (senza il ciclo With… End) in questo modo:

```
    Private Sub btnCreaLabel_Click(sender As System.Object, e As
System.EventArgs) Handles btnCreaLabel.Click

Dim lblNuova As New Label

        lblNuova.Name = "lblNuova"
        lblNuova.BackColor = Color.Blue
        lblNuova.ForeColor = Color.Yellow
        lblNuova.Location = New Point(12, 60)
        lblNuova.Size = New Size(75, 39)
        lblNuova.Text = "Questa è una label nuova"

        Me.Controls.Add(lblNuova)

    End Sub
```

- Terminata l'impostazione delle proprietà della Label che verrà creata con un *clic* sul pulsante btnCreaLabel, l'ultima riga della procedura ne comanda l'inserimento tra i controlli presenti nel Form1:

```
        Me.Controls.Add(lblNuova)
```

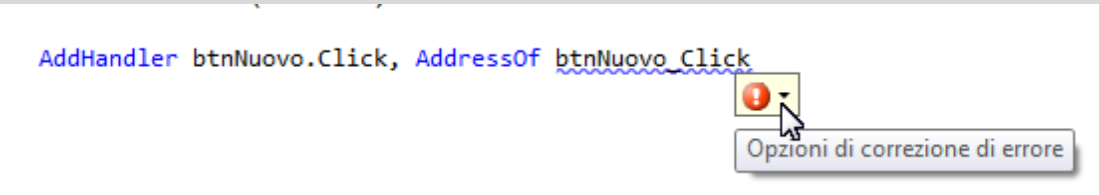
Continuiamo scrivendo una procedura simile per il **btnCreaPulsante** (un *clic* sul btnCreaPulsante creerà un altro pulsante di nome btnNuovo il quale, a sua volta, farà chiudere il programma):

```
Private Sub btnCreaPulsante_Click() Handles btnCreaPulsante.Click

        Dim btnNuovo As New Button
        With btnNuovo
            .Name = "btnNuovo"
            .Location = New Point(98, 58)
            .Size = New Size(80, 40)
            .Text = "ESCI"
        End With
```

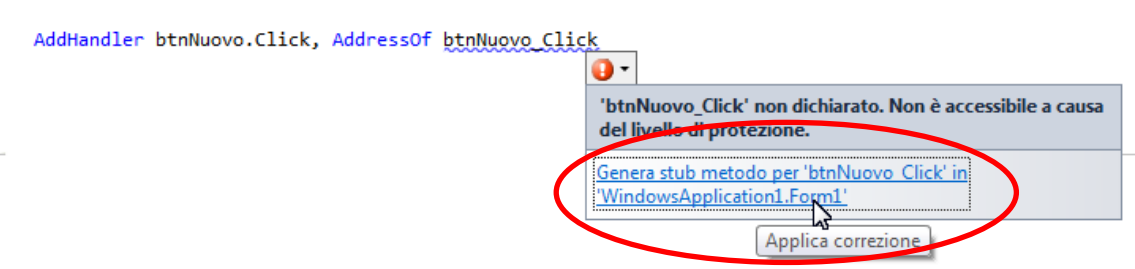
```
        Me.Controls.Add(btnNuovo)

        AddHandler btnNuovo.Click, AddressOf btnNuovo_Click

    End Sub
```

Notiamo che, al termine della procedura, dopo l'aggiunta del **btnNuovo** ai controlli presenti nel Form1, l'ultima riga dispone che l'evento del *clic* del mouse su questo nuovo pulsante, cioè l'evento **btnNuovo.Click**, sia gestito da un procedura che è chiamata **btnNuovo_Click.**
Questa procedura, però, ancora non esiste. Per crearla possiamo ricorrere a due modalità diverse.

La prima di queste modalità è offerta da **IntelliSense**.
Notiamo che al termine della riga di istruzioni evidenziata in giallo **IntelliSense** segnala un errore, in quanto la procedura **btnNuovoClick** non esiste:



Facendo un *clic* sulla freccina nera, per visualizzare l'errore, IntelliSense propone una soluzione: creare la procedura **btnNuovo_Click** che non esiste:



Accettiamo la correzione proposta da IntelliSense facendo un *clic* con il mouse su **Genera stub metodo per btnNuovo_Click**[48] . Nella Finestra del Codice, troviamo ora la procedura **btnNuovo_Click** preimpostata in questo modo:

```
    Private Sub btnNuovo_Click(sender As Object, e As EventArgs)
        Throw New NotImplementedException
    End Sub
```

[48] **Genera stub metodo** (= genera abbozzo di procedura) è una funzionalità di generazione automatica del codice di **IntelliSense**, che semplifica la creazione di una nuova procedura e la imposta automaticamente, deducendola dal contesto scritto dal programmatore.

Cancelliamo la riga centrale della procedura, che non ci serve, e scriviamo al suo posto il comando **End**, in quanto vogliamo che un *clic* del mouse sul pulsante btnNuovo faccia terminare il programma:

```
Private Sub btnNuovo_Click(sender As Object, e As EventArgs)
    End
End Sub
```

Facciamo un passo indietro e vediamo la seconda modalità per creare la procedura **btnNuovo_Click**.
Facciamo un *clic* con il tasto destro del mouse nella prima riga libera dopo la fine di una procedura, cioè dopo un **End Sub**.
Nel Menu che si apre, facciamo un *clic* su **Inserisci frammento di codice…,** quindi facciamo un doppio *clic* su **Modelli di Codice / Proprietà, procedure, eventi** e infine su **Definisci un sub privato**.
Viene così inserita nel codice una nuova procedura generica, vuota, impostata in questo modo:

```
Private Sub MyMethod()

End Sub
```

Cambiamo il nome MyMethod con il nome **btnNuovo_Click** e nello spazio intermedio della procedura, vuoto, scriviamo il comando **End** (il *clic* su questo pulsante farà terminare il programma):

```
Private Sub btnNuovo_click()
    End
End Sub
```

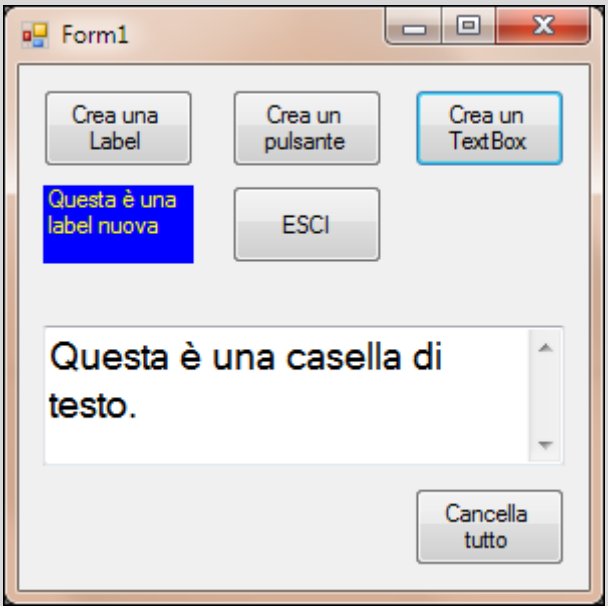
Con questo abbiamo concluso il lavoro dedicato al pulsante **btnCreaPulsante**: un *clic* su questo pulsante creerà il nuovo pulsante **btnNuovo** e un *clic* sul pulsante **btnNuovo** farà terminare il programma.
Continuiamo la scrittura del codice con la procedura relativa al pulsante **btnCreaTextBox**, che creerà un nuova casella di testo nel Fom1:

```
Private Sub btnCreaTextBox_Click() Handles btnCreaTextBox.Click

    Dim txtNuovo As New TextBox

    With txtNuovo
        .Name = "txtNuovo"
        .Multiline = True
        .ScrollBars = ScrollBars.Vertical
        .Font = New Font("Microsoft Sans Serif", 14)
        .Location = New Point(12, 130)
        .Size = New Size(260, 70)
        .Text = "Questa è una casella di testo."
    End With

    Me.Controls.Add(txtNuovo)

End Sub
```

Concludiamo la scrittura del codice con la procedura relativa al pulsante **btnCancella**, che ha il compito di eliminare dal Form1 i nuovi oggetti creati con gli altri pulsanti:

```
Private Sub btnCancella_Click() Handles btnCancella.Click

    Me.Controls.Remove(Me.Controls("lblNuova"))
    Me.Controls.Remove(Me.Controls("btnNuovo"))
    Me.Controls.Remove(Me.Controls("txtNuovo"))

End Sub
```

Ecco un'immagine del programma in esecuzione:



Capitolo 13: LE VARIABILI.

Computer è una parola che ha viaggiato molto, nello spazio e nel tempo.
Essa è giunta all'italiano dall'inglese, ma la sua radice risale al latino *computare,* che significa *contare*.
Una parola con la medesima radice latina, in italiano, è *computisteria*, che si riferisce all'attività del contare, del tenere in ordine i conti.
La lingua italiana, pur trovandosi così vicina al latino, non ha avuto gli stimoli forniti dal recente sviluppo dell'informatica e per indicare la macchina elaboratrice di dati non ha saputo coniare un termine proprio di diretta derivazione latina (*computatore*?), ma si è adattata a usare il termine *computer*, di derivazione latina con intermediazione dell'inglese.
Il nome *computer* dunque ci ricorda che questa è una macchina che elabora e tiene in ordine dei dati.
A differenza di una calcolatrice normale (che tratta solo dati numerici secondo i pochi tasti delle operazioni che appaiono sulla tastiera), un computer tratta elementi molto diversi tra di loro, compiendo operazioni molto più articolate e più complesse delle semplici operazioni aritmetiche.
Un computer, ad esempio, è in grado di tradurre in numeri e di manipolare immagini e suoni, oppure, per fare un esempio che ci riguarderà da vicino nelle prossime pagine, è in grado di manipolare numeri e testi per compiere operazioni aritmetiche e logiche, per fare delle analisi e delle comparazioni, per verificare il sussistere di determinate condizioni…
Il computer assegna a ogni elemento (numeri, parole, testi, immagini, suoni) uno spazio nella sua memoria.
La parte di memoria del computer preposta al trattamento di questi dati si chiama RAM (memoria ad accesso casuale); si tratta di un'area di memoria utilizzata quando il computer è acceso, durante lo svolgimento di programmi; essa viene svuotata quando il computer è spento.
Possiamo immaginare questa RAM come una grande scrivania fornita di migliaia di cassetti: il computer li riempie secondo le esigenze, collocandovi gli elementi (numeri, parole, testi, immagini, suoni) usati nel programma o nei programmi in esecuzione.
Per ritrovare il contenuto dei cassetti, ogni volta che se ne presenta la necessità, il computer ha bisogno che ogni cassetto sia contraddistinto da due **elementi**:

- il nome e
- il tipo di materiale contenuto nel cassetto.

L'evento straordinario in queste operazioni tuttavia non sta nella capacità del computer di usare il contenuto dei cassetti per fare calcoli, confronti o analisi, ma nella capacità di fare calcoli, confronti o analisi usando il nome che compare sui cassetti e non il loro contenuto.
Vediamo un esempio.

Un programma prevede che il computer effettui una serie di 10 sottrazioni, scelte a caso, con numeri inferiori a 1000:
120 - 34
450 - 67
780 - 110
ecc.
Il computer può eseguire queste sottrazioni come lo farebbe una normale calcolatrice, ma può fare una cosa enormemente più *intelligente*: può immagazzinare tutti i primi termini delle sottrazioni in un cassetto, al quale è assegnato ad esempio il nome MINUENDO, e può immagazzinare tutti i secondi termini delle sottrazioni in un altro cassetto, denominato ad esempio SOTTRAENDO.
Una volta definiti i due cassetti, per effettuare le dieci sottrazioni, al computer basta una sol'istruzione:
MINUENDO – SOTTRAENDO
Con questa istruzione si dice al computer di prendere un numero, dal secondo cassetto, e di sottrarlo a un altro numero preso dal primo cassetto.
Con l'operazione (MINUENDO – SOTTRAENDO), è possibile avere migliaia di sottrazioni diverse: è sufficiente cambiare il contenuto dei cassetti, la forma dell'operazione non cambia.
Questi cassetti, il cui contenuto può essere cambiato a piacere dal programmatore o dall'utente del programma, sono oggetti chiamati **variabili**.
Come vedremo, essi possono contenere numeri, parole, testi, altri oggetti, liste di elementi…
Inizieremo dai tipi di dati più semplici: le variabili di numeri e stringhe di testo.

72: Creazione di una variabile.

Per utilizzare una variabile è necessario in via preliminare dichiararla.
In VB si dichiara una variabile con il comando **Dim** che sta per *dimensiona (*= riserva uno spazio nella memoria).
La dichiarazione di una variabile richiede, oltre al comando **Dim**:

- l'assegnazione di un nome proprio e
- la dichiarazione del contenuto della variabile.

Ad esempio, la dichiarazione:

```
Dim MINUENDO As Integer
```

significa: dimensiona una variabile, il cui nome proprio è MINUENDO, destinata a contenere numeri interi.

Il nome proprio di una variabile è scelto dal programmatore, a sua discrezione.
Tale nome deve essere univoco: all'interno della stessa procedura non vi possono essere due variabili con lo stesso nome.
La scelta del nome deve rispettare queste regole:

- il nome di una variabile può consistere in una lettera (a, b, c, d, ... A, B, C, D, ...), una serie di lettere, una serie di lettere e numeri;
- il nome di una variabile può consistere in una parola o in un nome composto da più parole unite tra loro;
- il nome non può essere composto da due parti separate da uno spazio;
- il nome non può avere al suo interno i caratteri speciali che si trovano sulla tastiera (i caratteri con simboli diversi da numeri o lettere) : ! @ # $ % ^ & *, ecc.

73: Dichiarazione del tipo di contenuto di una variabile.

La dichiarazione del tipo di contenuto di una variabile avviene mediante l'istruzione **As** (= come, uguale a…).
Ecco alcuni esempi:

```
Dim MINUENDO As Integer
Dim QUOZIENTE As Single
Dim Nome As String
Dim Cognome As String
Dim RipostaEsatta As Boolean
```

Vediamo il significato di queste cinque dichiarazioni:

1. Dimensiona una variabile di nome `MINUENDO`, di tipo `Integer`, destinata a contenere numeri interi.
2. Dimensiona una variabile di nome `QUOZIENTE`, di tipo `Single`, destinata a contenere numeri con la virgola.
3. Dimensiona una variabile di nome `Nome`, di tipo `String`, destinata a contenere una o più parole.
4. Dimensiona una variabile di nome `Cognome`, di tipo `String`, destinata a contenere una o più parole.
5. Dimensiona una variabile di nome `RipostaEsatta`, di tipo `Boolean`, destinata a contenere solo due valori: True (vero) o False (falso).

Il contenuto di una variabile può consistere in numeri interi (**Integer**) o decimali (**Single**), in parole o testi che VB tratta come sempre come strisce di testo (**String**) a prescindere dalla loro lunghezza[49].
Le strisce di testo si differenziano rispetto ai dati numerici **perché sono delimitate da virgolette**; il termine **strisce** vuole rendere l'idea di una sequenza di caratteri tipografici, di qualsiasi tipo, contenuti tra due virgolette.
Sono variabili di tipo String, ad esempio, **tutte** le seguenti stringhe di caratteri contenute tra virgolette:

```
Dim Testo1 As String = "Mario"
Dim Testo2 As String = "Luigi Rossi"
Dim Testo3 As String = "12 agosto 1999"
Dim Testo4 As String = "**** XXXX ______"
Dim Testo5 As String = "[… omissis …]"
Dim Testo6 As String = "12000000"
Dim Testo7 As String = "3,14"
```

I tipi di variabili di uso più comune e maggiormente usati in questo manuale sono questi:

Nome	**Contenuto**
Boolean	Due sole possibilità: Vero o Falso.
Date	Date, dal 1 gennaio 0001 al 31 dicembre 9999.
Byte	Numeri interi positivi, da 0 a 255.
Integer	Numeri interi, da -2.147.483.648 a 2.147.483.647.
Single	Numeri con la virgola.
String	Testo.
Char	Carattere (lettere dell'alfabeto, cifre, simboli).

Tabella 8: I principali tipi di variabili.

[49] Una variabile in VB può contenere, oltre a numeri e stringhe di testo, anche oggetti diversi molto più complessi, e addirittura anche oggetti/controlli/componenti di VB. Vedremo queste possibilità più avanti, in questo stesso capitolo.

74: Assegnazione di un dato a una variabile.

Quando in un programma è dichiarata una variabile, Visual Basic *inizializza* questa variabile compiendo queste operazioni:

- riserva uno spazio adeguato a questa variabile nella memoria del computer;
- delimita l'area di validità della variabile (può essere una variabile valida solo all'interno di una procedura, oppure valida per tutto il codice);
- riserva il nome della variabile per evitare doppioni di nomi e per controllare in seguito eventuali errori di battitura da parte del programmatore;
- riserva il contenuto della variabile, secondo il tipo scelto dal programmatore;
- assegna nella variabile il valore "False" se si tratta di variabile di tipo Boolean, il numero 0 (se si tratta di variabile numerica come Byte, Integer o Single) o una stringa di testo vuota "" (se si tratta di una variabile String). Il valore "False", il numero 0 o la stringa vuota "" rimangono assegnati alla variabile sino a quando il programmatore o l'utente li sostituiscono con altri dati.

L'assegnazione di un dato, di un numero o di un testo in una variabile avviene utilizzando il simbolo = uguale. Questa operazione può essere fatta contestualmente, nel momento in cui si crea la variabile:

```
Dim MINUENDO As Integer = 12
Dim QUOZIENTE As Single = 23.45
Dim Nome As String = "Paolo"
Dim Cognome As String = "Rossi"
Dim EsercizioFinito As Boolean = False
Dim Lettera As Char = "A"
```

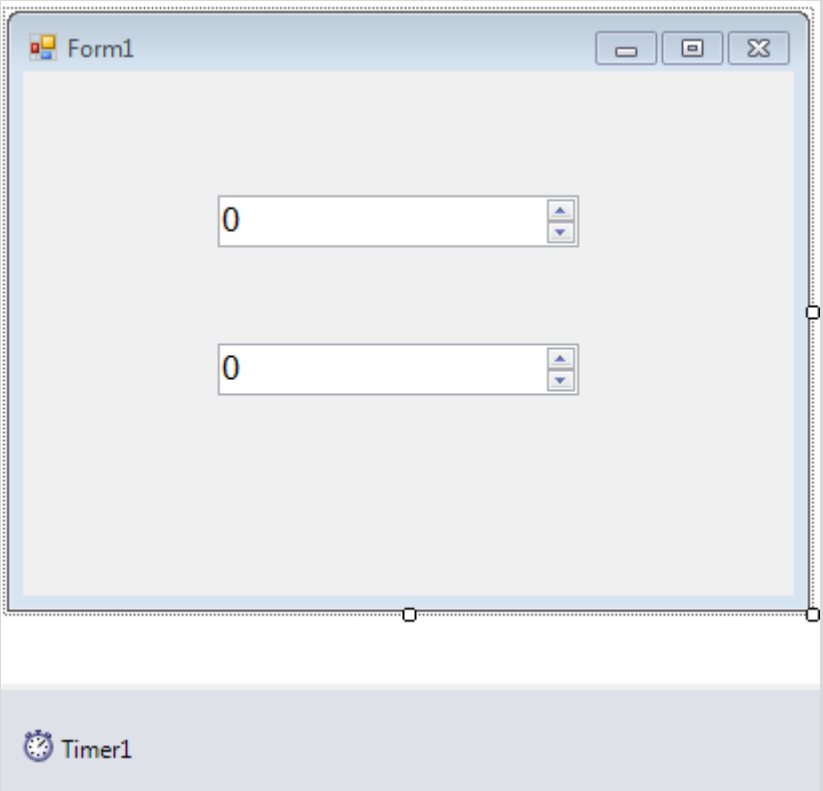
Oppure in momenti successivi, in parti separate del codice;

```
Dim MINUENDO As Integer
Dim QUOZIENTE As Single
Dim Nome As String
Dim Cognome As String
Dim EsercizioFinito As Boolean
Dim Lettera As Char

MINUENDO = 12
QUOZIENTE = 23.45
Nome = "Paolo"
Cognome = "Rossi"
EsercizioFinito = False
Lettera = "A"
```

Una volta create le variabili, il programmatore o l'utente del programma le possono riempire a piacere, rispettandone tuttavia il tipo di contenuto: non sarà possibile assegnare la parola "*Pierino*" alla variabile `MINUENDO` e non sarà possibile assegnare il numero 12 nella variabile `Nome` (o meglio: questa seconda operazione sarà possibile, ma VB tratterà il 12, scritto tra virgolette, come se fosse un nome e non un numero).
Nell'esercizio seguente vedremo all'opera due variabili di nome A e B, destinate a contenere numeri interi.

Esercizio 40: Confronto tra due variabili numeriche.

Apriamo un nuovo progetto e inseriamo nel Form1 due controlli **NumericUpDown** e un componente **Timer**, come in questa immagine:



Impostiamo queste proprietà del componente **Timer1**:

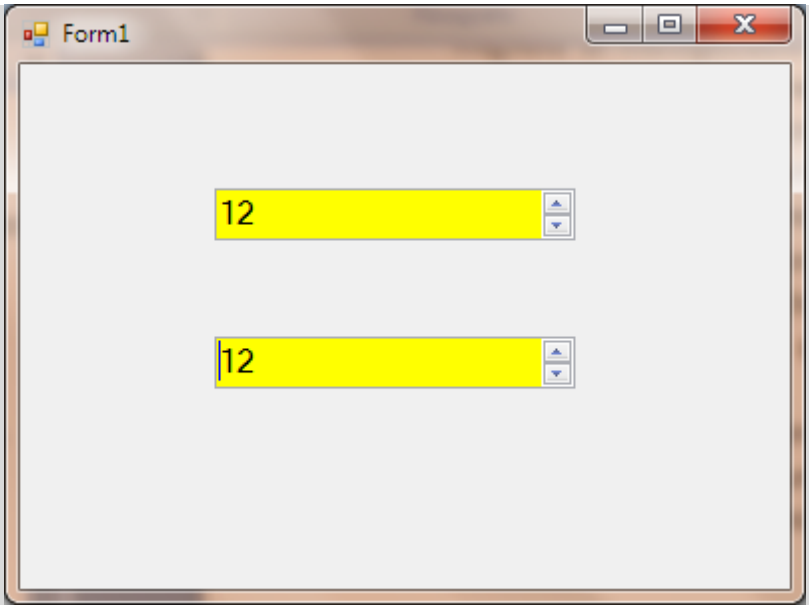
- **Enabled = True** (il Timer1 sarà attivo da subito, appena avviato il programma);
- **Interval = 1000** (il Timer1 causerà un evento **Tick** al trascorrere di ogni secondo).

Nel programma sono create due variabili, di nome **PrimoNumero** e **SecondoNumero**, destinate a contenere numeri interi.
A ogni tic del **Timer1**, il programma assegna alla variabile **PrimoNumero** il numero contenuto nel controllo **NumericUpDown1** e alla variabile **SecondoNumero** il numero contenuto nel controllo **NumericUpDown2**, poi effettua un confronto tra i due valori e, a seconda dell'esito di questo confronto, modifica il colore di sfondo dei due controlli **NumericUpDown**.
Una volta terminata la progettazione dell'interfaccia, è possibile copiare il codice seguente e incollarlo nella Finestra del Codice; le informazioni utili a comprendere il codice sono inserite nel listato.

```
Public Class Form1

    ' Crea due variabili di nome PrimoNumero e SecondoNumero, destinate a
contenere numeri interi.
    ' Queste due variabili collocate in quest'area del codice, al di fuori delle
procedure,
```

```
    ' sono valide per tutto il codice che riguarda il Form1 e sono riconosciute
da tutte le procedure in esso contenute.

    Dim PrimoNumero As Integer
    Dim SecondoNumero As Integer

    Private Sub Timer1_Tick() Handles Timer1.Tick

        ' a ogni tic del timer:
        ' alla variabile PrimoNumero viene assegnato il valore numerico contenuto
nel controllo NumericUpDown1
        ' alla variabile SecondoNumero viene assegnato il valore numerico
contenuto nel controllo NumericUpDown2

        PrimoNumero = NumericUpDown1.Value
        SecondoNumero = NumericUpDown2.Value

        ' qui avviene il confronto tra le due variabili, con modifica dei colori
di sfondo dei due controlli NumericUpDown:
        If PrimoNumero > SecondoNumero Then
            ' se PrimoNumero è maggiore di SecondoNumero:
            NumericUpDown1.BackColor = Color.Green
            NumericUpDown2.BackColor = Color.Red

        ElseIf PrimoNumero < SecondoNumero Then
            ' se PrimoNumero è minore di SecondoNumero:
            NumericUpDown1.BackColor = Color.Red
            NumericUpDown2.BackColor = Color.Green

        Else
            ' altrimenti (se PrimoNumero e SecondoNumero sono uguali):
            NumericUpDown1.BackColor = Color.Yellow
            NumericUpDown2.BackColor = Color.Yellow
        End If

    End Sub

End Class
```

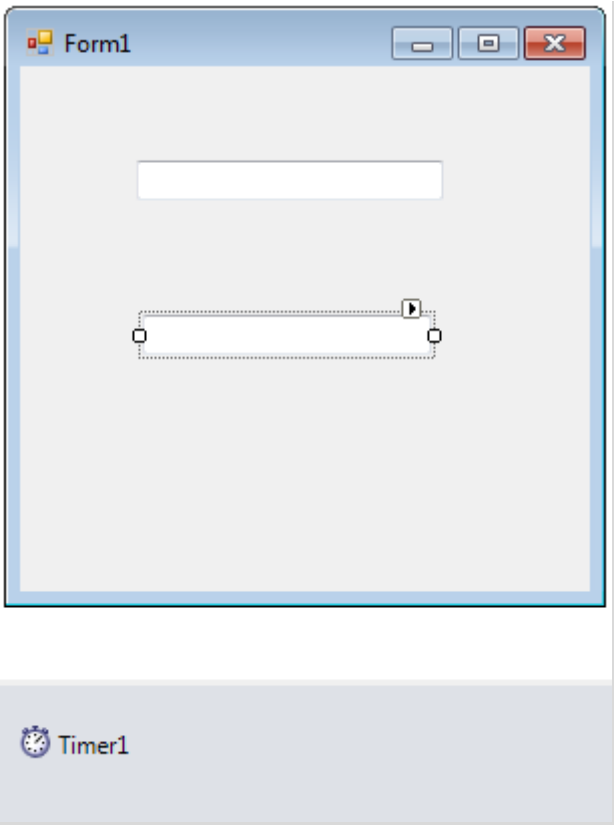
Ecco un'immagine del programma in esecuzione:



Ripeteremo di seguito lo stesso esercizio, utilizzando due variabili di tipo **String** (strisce di testo) al posto delle variabili **Integer**. In questa seconda versione, il programma confronta la lunghezza dei testi contenuti in due controlli TextBox.

Esercizio 41: Confronto tra due stringhe di testo.

Apriamo un nuovo progetto di VB come il progetto dell'esercizio precedente sostituendo i due controlli NumericUpDown con due controlli **TextBox**, come in questa immagine:



In questo esercizio, a ogni *tic* del Timer il programma effettua un controllo tra la lunghezza dei testi scritti nei due TextBox:

```
Public Class Form1

    ' Crea due variabili di nome PrimoNumero e SecondoNumero, destinate a
contenere numeri interi.

    Dim PrimoNumero As Integer
    Dim SecondoNumero As Integer
```

```
    Private Sub Timer1_Tick() Handles Timer1.Tick
        ' a ogni tic del timer:
        ' alla variabile PrimoNumero viene assegnato il numero corrispondente
alla lunghezza del testo contenuto nel TextBox1
        ' alla variabile SecondoNumero viene assegnato il numero corrispondente
alla lunghezza del testo contenuto nel controllo TextBox2
        ' (Length = lunghezza)

        PrimoNumero = TextBox1.Text.Length
        SecondoNumero = TextBox2.Text.Length

        ' qui avviene il confronto tra le due variabili, con modifica dei colori
di sfondo dei due controlli TextBox:
        If PrimoNumero > SecondoNumero Then
            ' se PrimoNumero è maggiore di SecondoNumero:
            TextBox1.BackColor = Color.Green
            TextBox2.BackColor = Color.Red

        ElseIf PrimoNumero < SecondoNumero Then
```

```
            ' se PrimoNumero è minore di SecondoNumero:
            TextBox1.BackColor = Color.Red
            TextBox2.BackColor = Color.Green

        Else
            ' altrimenti (se PrimoNumero e SecondoNumero sono uguali):
            TextBox1.BackColor = Color.Yellow
            TextBox2.BackColor = Color.Yellow
        End If

    End Sub

End Class
```

75: Assegnazioni successive di dati.

L'assegnazione di un secondo contenuto alla stessa variabile cancella e sostituisce il contenuto precedente per cui, ad esempio, al termine di queste istruzioni:

```
Dim MINUENDO As Integer = 12
MINUENDO = 25
```

il valore della variabile MINUENDO è uguale a 25.
Altro esempio: alla fine di questa sequenza di comandi:

```
Dim x As Integer = 10
x = 100
x = 1000
```

il contenuto dalla variabile x è uguale a 1000; i valori precedenti, ai fini del proseguimento del programma, non esistono più.
E' possibile tenere conto del valore precedente di una variabile in sequenze di operazioni di questo tipo:

```
Dim x As Integer = 10
x += 100
x += 1000
```

In questo caso:

- nella prima riga, al momento della sua creazione la variabile x contiene il valore 10;
- nella seconda riga si somma al valore 10 il valore 100 e la variabile x diventa dunque uguale a 110;
- nella terza riga si somma al valore precedente (110) il valore 1000 e la variabile x diventa dunque uguale a 1110.

Questa sequenza di comandi con immissione di nuovi dati nella variabile x può essere scritta anche così:

```
Dim x As Integer
x = 10
x = x + 100
x = x + 1000
```

Le espressioni

```
x = x + 1000
```

e

```
x += 1000
```

sono espressioni equivalenti, il risultato finale è identico. In questo manuale è adottata costantemente **la seconda modalità: +=.**

Le stesse operazioni in sequenza possono essere effettuate con variabili di tipo **String** (strisce di testo).
Al termine di questa serie di istruzioni, la variabile INCIPIT conterrà solo il testo "*che volge a mezzogiorno, …*", perché ogni nuova assegnazione di testo cancella e sostituisce l'assegnazione precedente:

```
Dim INCIPIT As String = "Quel ramo"
INCIPIT = " del lago "
INCIPIT = " di Como, "
INCIPIT = " che volge a mezzogiorno, …"
```

Per tenere conto del testo precedente e aggiungerlo al testo successivo si possono scrivere le istruzioni in questo modo:

```
Dim INCIPIT As String = "Quel ramo"
INCIPIT = INCIPIT & " del lago "
INCIPIT = INCIPIT & " di Como, "
INCIPIT = INCIPIT & " che volge a mezzogiorno, …"
```

oppure in questo modo:

```
Dim INCIPIT As String = "Quel ramo"
INCIPIT &= " del lago "
INCIPIT &= " di Como, "
INCIPIT &= " che volge a mezzogiorno, …"
```

Con l'una o l'altra di queste due modalità, al termine delle istruzioni il contenuto della variabile INCIPIT sarà uguale a "*Quel ramo del lago di Como, che volge a mezzogiorno, …*". In questo manuale è adottata costantemente **la seconda modalità: &=.**

La tabella seguente mostra un quadro complessivo delle operazioni di concatenazione che possono essere effettuate con il contenuto di variabili di tipo numerico o di tipo String.

Simbolo	Esempio	Descrizione dell'operazione e del risultato
^=	`Dim x as Integer = 5` `x ^= 2`	Eleva il valore della variabile x a potenza. In questo esempio x è elevata al quadrato; alla fine x contiene il valore 25.
***=**	`Dim x as Integer = 5` `x *= 2`	Moltiplica il valore della variabile x per il numero espresso dopo *=. In questo esempio il valore iniziale di x è moltiplicato per 2; alla fine x contiene il valore 10.
/=	`Dim x as Single = 5` `x /= 2`	Divide il valore della variabile x per il numero espresso dopo /=. In questo esempio il valore iniziale di x è diviso per 2; alla fine x contiene il valore 2,5.
\=	`Dim x as Single = 5` `x \= 2`	Divide il valore della variabile x per il numero espresso dopo \= e restituisce solo il valore intero del risultato. In questo esempio il valore iniziale di x è diviso per 2; alla fine x contiene il valore 2.
+=	`Dim x as Integer = 5` `x += 2`	Incrementa il valore della variabile x del numero espresso dopo +=. In questo esempio al valore iniziale di x è sommato il valore 2; alla fine x contiene il valore 7.
-=	`Dim x as Integer x = 5` `x -= 2`	Diminuisce il valore della variabile x del numero espresso dopo -=. In questo esempio al valore iniziale di x è sottratto il valore 2; alla fine x contiene il valore 3.
&=	`Dim x as String = "Paolo"` `x &= " Rossi"`	Si usa con variabili di testo. Concatena il valore iniziale della variabile x con il valore espresso dopo &=. In questo esempio al valore iniziale di x è aggiunta la parola "Rossi"; alla fine x contiene il testo "Paolo Rossi".

Tabella 9: Operazioni con variabili numeriche e variabili di tipo String.

76: Abbreviazioni.

E' possibile creare più variabili dello stesso tipo su una sola linea di codice, separandole con una virgola:

```
Dim A, B, C As Integer
Dim Nome, Cognome As String
```

Nella prima riga vengono create tre variabili destinate a contenere numeri interi.
Nella seconda riga vengono create due variabili destinate a contenere parole.

È possibile assegnare un valore a una variabile al momento della creazione della variabile, oppure in momenti successivi:

```
Dim X As Integer
Dim Nome As String
X = 100
Nome = "Paolo"
```

oppure

```
Dim X As Integer = 100
Dim Nome As String = "Paolo"
```

77: Area di validità delle variabili.

La creazione di una variabile può essere collocata in posizioni diverse all'interno del codice; a seconda di questa collocazione, varia il raggio di azione della variabile stessa.
Una variabile infatti può essere valida:

- per una singola procedura (in questo caso abbiamo una variabile locale limitata a quella procedura);
- per un form (in questo caso abbiamo una variabile di più ampio raggio, valida per tutte le procedure che fanno parte di quel form);
- per un progetto (in questo caso avremo una variabile pubblica, valida per tutte le procedure che fanno parte di tutti i form che costituiscono il progetto).

Le variabili di tipo **Public**, valide per tutti i form di un progetto, si trovano in form particolari, che non hanno interfaccia grafica, di nome **Moduli**.
Ecco, ad esempio, la finestra **Esplora soluzioni** di un progetto composto di tre form e un modulo:

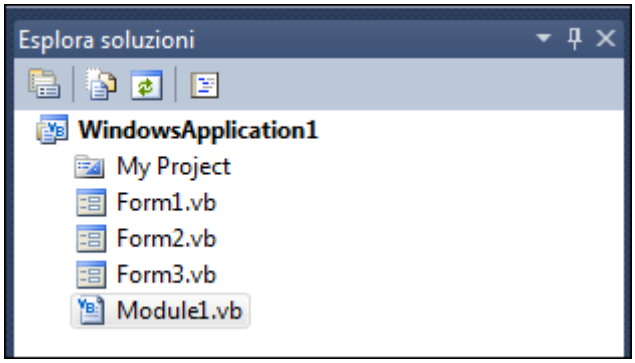


Figura 127: Un progetto con tre form e un modulo.

Nela Finestra del Codice del modulo sono create queste variabili che, essendo di tipo Public, sono valide per tutti i tre form che formano il progetto: la variabile Nome_Cognome, ad esempio, impostata nel Form1 come "Paolo Rosi", potrà essere corretta nel Form2 come "Paolo Rossi" e potrà essere visualizzata nel Form3:

```
Module Module1

    Public Nome_Cognome As String
    Public Data_di_Nascita As Date
    Public Indirizzo As String

End Module
```

Ecco invece alcuni esempi di creazione di variabili di tipo **Dim** (= dimensiona, crea), con diverso raggio di azione a livello di un singolo form:

```
    Public Class Form1

    Dim Nome_Cognome As String
    Dim Minuendo, Sottraendo As Integer

    Private Sub Form1_Click() Handles Me.Click

        Minuendo = 12
        Sottraendo = 8

        Dim Risposta As Integer
        Risposta = Minuendo - Sottraendo
        Me.Text = Risposta.ToString

    End Sub

    Private Sub Form1_DoubleClick() Handles Me.DoubleClick

        Minuendo = 24
        Sottraendo = 16

        Dim Risposta As Integer
        Risposta = Minuendo - Sottraendo
        Me.Text = Risposta.ToString
```

```
    End Sub

End Class
```

Le variabili **Nome_Cognome**, **Minuendo** e **Sottraendo**, create nello spazio di intestazione della Classe Form1, sono valide per tutto il codice che riguarda il Form1, e sono riconosciute da tutte le procedure scritte tra le righe

```
    Public Class Form1

    End Class
```

La variabile **Risposta**, creata all'interno della procedura **Form1_Click**, ha validità limitata a questa procedura. Una volta esaurito l'evento del *clic* del mouse sul Form1, e una volta conclusa questa procedura, la variabile `Risposta` termina la sua funzione; lo spazio in memoria che le era stato assegnato torna ad essere disponibile.
Lo stesso discorso vale per la variabile omonima, **Risposta**, creata all'interno della procedura **Form1_DoubleClick**: anche la validità di questa variabile è limitata a questa procedura. Una volta esaurito l'evento del doppio *clic* del mouse sul Form1, e una volta conclusa questa procedura, la variabile `Risposta` termina la sua funzione e lo spazio in memoria che le era stato assegnato torna a essere disponibile.
Nel prossimo esercizio vedremo un metodo pratico per visualizzare l'area di validità di una variabile.

Esercizio 42: Visualizzare l'area di validità delle variabili.

Apriamo un nuovo progetto; copiamo e incolliamo nella Finestra del Codice il listato che abbiamo analizzato prima:

```
    Public Class Form1

    Public Nome_Cognome As String
    Dim Minuendo, Sottraendo As Integer
```

```
    Private Sub Form1_Click() Handles Me.Click

        Minuendo = 12
        Sottraendo = 8

        Dim Risposta As Integer
        Risposta = Minuendo - Sottraendo
        Me.Text = Risposta.ToString

    End Sub
```

```
    Private Sub Form1_DoubleClick() Handles Me.DoubleClick

        Minuendo = 24
        Sottraendo = 16
```

```
        Dim Risposta As Integer
        Risposta = Minuendo - Sottraendo
        Me.Text = Risposta.ToString

    End Sub

End Class
```

Ora, facendo un *clic* su una qualsiasi delle variabili presenti in questo codice, tutte le istanze **valide** della stessa variabile vengono evidenziate in grigio.
Facendo un *clic* sulla variabile Sottraendo, ad esempio, si può notare che la sua validità spazia per tutte le procedure del Form1. Al contrario, con un *clic* sulla variabile Risposta si nota che la validità di questa variabile si limita alla procedura in cui essa è stata creata.

78: Trasferimento di dati tra due procedure.

E' di uso piuttosto frequente nei programmi la tecnica del passaggio di dati da una procedura all'altra.
Con questa tecnica, abbiamo una prima procedura che chiama in esecuzione un'altra procedura e nel far questo le trasmette uno o più dati, scritti tra parentesi.
La procedura ricevente, attivata dalla prima procedura, riceve il dato o i dati scritti tra parentesi e li utilizza come variabili per eseguire le funzioni che le sono affidate dal programmatore.

Vediamo in esempio.
In un programma abbiamo un form con quattro pulsanti Button e un controllo Label.
I quattro pulsanti Button sono incolonnati sulla destra del form, la Label è in alto a sinistra.
Funzionamento del programma: premendo uno dei quattro pulsanti Button, cambia la grandezza dei caratteri del testo visualizzato nella Label1.
Nel codice seguente vediamo quattro procedure che gestiscono il *clic* del mouse sui quattro pulsanti Button.
Ogni procedura imposta la grandezza dei caratteri nella Label1 rispettivamente a 12, 24, 36 e 48 punti:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Dim NuovoCarattere As New Font("Verdana", 12)
        Label1.Font = NuovoCarattere

    End Sub
```

```
    Private Sub Butto2_Click(sender As System.Object, e As System.EventArgs)
Handles Button2.Click

        Dim NuovoCarattere As New Font("Verdana", 24)
        Label1.Font = NuovoCarattere

    End Sub
```

```
    Private Sub Button3_Click(sender As System.Object, e As System.EventArgs)
Handles Button3.Click

        Dim NuovoCarattere As New Font("Verdana", 36)
        Label1.Font = NuovoCarattere

    End Sub
```

```
    Private Sub Button4_Click(sender As System.Object, e As System.EventArgs)
Handles Button4.Click

        Dim NuovoCarattere As New Font("Verdana", 48)
        Label1.Font = NuovoCarattere

    End Sub

End Class
```

E' evidente come nelle quattro procedure si ripeta la stessa operazione, con la sola modifica del dato relativo alla grandezza dei caratteri.
Nel codice seguente l'operazione comune, ripetuta quattro volte nelle quattro procedure, è affidata a un'unica procedura, denominata CambiaCarattere, che è **attivata** dalle altre procedure, **riceve** da esse il dato relativo alla grandezza del carattere e procede alla visualizzazione del testo nella Label1:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Call CambiaCarattere(12)

    End Sub
```

```
    Private Sub Butto2_Click(sender As System.Object, e As System.EventArgs)
Handles Button2.Click

        Call CambiaCarattere(24)

    End Sub
```

```
    Private Sub Button3_Click(sender As System.Object, e As System.EventArgs)
Handles Button3.Click

        Call CambiaCarattere(36)

    End Sub
```

```
    Private Sub Button4_Click(sender As System.Object, e As System.EventArgs)
Handles Button4.Click

        Call CambiaCarattere(48)

    End Sub

    Private Sub CambiaCarattere(Grandezza As Integer)

        Dim NuovoCarattere As New Font("Verdana", Grandezza)
        Label1.Font = NuovoCarattere

    End Sub

End Class
```

Notiamo, nel listato, che ogni procedura **Click** attiva (**Call** = chiama) la procedura comune **CambiaCarattere** e le **trasmette** un dato scritto tra parentesi.
La procedura **riceve** questo dato come una variabile numerica di tipo Integer e gli assegna il nome **Grandezza**, poi lo usa per visualizzare il testo nella Label1:

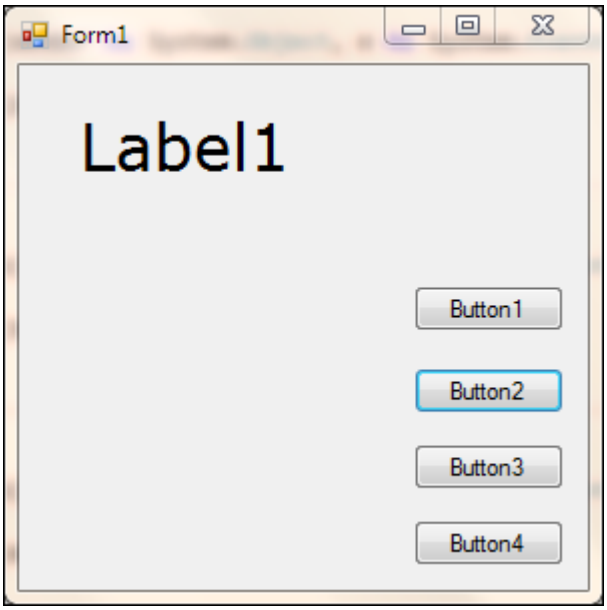


Figura 128: Trasferimento di dati tra procedure.

79: Trasferimento di dati tra procedure e funzioni.

Trasferimenti di dati, come quelli che abbiamo visto nel paragrafo precedente, possono avvenire anche con procedure particolari, chiamate **funzioni**, che svolgono un'attività prettamente di **servizio** per le altre procedure:

- sono attivate da esse;

- ricevono uno o più dati;
- elaborano questi dati secondo le istruzioni contenute nella funzione e
- restituiscono i dati elaborati alla procedura dalla quale sono state attivate.

Ecco, per esempio, una funzione che riceve due dati (**Base** e **Altezza**) e restituisce, dopo averli elaborati, l'area di un rettangolo:

```
    Function CalcolaArea(ByVal Base As Single, Altezza As Single) As Single

        Return Base * Altezza

    End Function
```

Ed ecco una funzione simile che riceve gli stessi dati, ma restituisce il calcolo del perimetro di un rettangolo:

```
    Function CalcolaPerimetro(ByVal Base As Single, Altezza As Single) As Single

        Return Base * 2 + Altezza * 2

    End Function
```

Queste due funzioni possono essere attivate, ad esempio, da una procedura che gestisce il *clic* su un pulsante Button1 in questo modo:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Dim Base As Single = 104.67
        Dim Altezza As Single = 44.56
        Dim Area As Single = CalcolaArea(Base, Altezza)
        Dim Perimetro As Single = CalcolaPerimetro(Base, Altezza)

        Dim TestoMessaggio As String
        TestoMessaggio = "Base: " & Base.ToString & vbCrLf
        TestoMessaggio &= "Altezza: " & Altezza.ToString & vbCrLf
        TestoMessaggio &= "Area: " & Area.ToString & vbCrLf
        TestoMessaggio &= "Perimetro: " & Perimetro.ToString

        MsgBox(TestoMessaggio)

    End Sub
```

```
    Function CalcolaArea(ByVal Base As Single, Altezza As Single) As Single

        Return Base * Altezza

    End Function
```

```
    Function CalcolaPerimetro(ByVal Base As Single, Altezza As Single) As Single

        Return Base * 2 + Altezza * 2
```

```
    End Function

End Class
```

In questo listato, la procedura Button1.Click attiva le due funzioni **CalcolaArea** e **CalcolaPerimetro**, trasmette loro le misure della base e dell'altezza del rettangolo e ne riceve il calcolo dell'area e del perimetro:

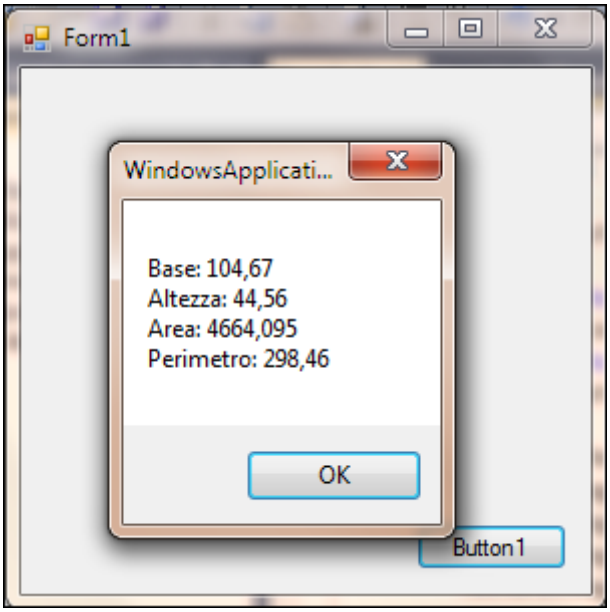


Figura 129: Trasferimento di dati tra procedure e funzioni.

Naturalmente i dati trasmessi o ricevuti dalla procedura e dalle funzioni devono essere dello stesso tipo (in questi esempi sono tutti di tipo Single, cioè numeri con la virgola), altrimenti si incorre in un errore nel programma.
Notiamo che le due funzioni ricevono i dati delle variabili **Base** e **Altezza** secondo la formula **ByVal**; essa indica che le funzioni debbono trattare i numeri contenuti nelle variabili così come sono. Nel caso contrario (**ByRef**) le funzioni sarebbero autorizzate a modificare eventualmente il contenuto di queste due variabili.
Notiamo ancora che le funzioni restituiscono il risultato delle loro elaborazioni con il comando **Return**. Esso può comunque essere sostituito, senza che il risultato cambi, dal nome della funzione:

```
    Function CalcolaArea(ByVal Base As Single, Altezza As Single) As Single

        CalcolaArea = Base * Altezza

    End Function
```

```
    Function CalcolaPerimetro(ByVal Base As Single, Altezza As Single) As Single

        CalcolaPerimetro = Base * 2 + Altezza * 2

    End Function
```

Notiamo, infine, che le funzioni terminano con il comando **End Function** e non con il comando **End Sub**, come le procedure normali.
Non sarà sfuggito alla lettrice o al lettore che nel listato di esempio le due funzioni CalcolaArea e CalcolaPerimetro sono un orpello che appesantisce il codice, in quanto i comandi che vi sono contenuti avrebbero potuto essere scritti nella procedura principale in questo modo:

```
        Dim Area As Single = Base * Altezza
        Dim Perimetro As Single = Base * 2 + Altezza * 2
```

In realtà, le funzioni svolgono un ruolo utile in un programma quando possono essere richiamate da molte procedure diverse: in questo caso costituiscono strumenti utili a snellire il codice e il lavoro del programmatore. Non bisogna dimenticare, inoltre, che le funzioni sono frammenti di codice che un programmatore può archiviare, catalogare ed utilizzare in altri programmi, quando gli si presentino esigenze simili.
IntelliSense fornisce uno strumento utile alla catalogazione delle funzioni: scrivendo tre volte il simbolo dell'apostrofo in testa ad una funzione, compaiono automaticamente alcune righe di commento che il programmatore può completare e che possono servire per archiviare una funzione con le informazioni sul suo contenuto e sul suo scopo:

```
    ''' <summary>
    ''' Calcola l'area di un rettangolo
    ''' </summary>
    ''' <param name="Base"></param>
    ''' <param name="Altezza"></param>
    ''' <returns></returns>
    ''' <remarks></remarks>

    Function CalcolaArea(ByVal Base As Single, Altezza As Single) As Single

        Return Base * Altezza

    End Function
```

L'inserimento dei tre apostrofi e delle righe di commento mettono in grado **IntelliSense** di fornire informazioni sui parametri della funzione, così come per tutti gli altri oggetti, man mano che il programmatore procede nella scrittura del codice:

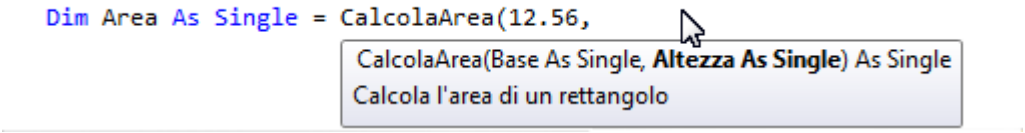


Figura 130: Riconoscimento di una funzione da parte di IntelliSense.

80: Variabili che contengono oggetti.

Nel capitolo precedente[50] abbiamo visto che è possibile creare variabili destinate a contenere qualsiasi oggetto di VB.
Ad esempio, questa riga di codice:

```
Dim PulsanteMio As New Button
```

crea un nuovo oggetto, di nome PulsanteMio, che sarà in tutto e per tutto identico ai pulsanti Button.
Nell'esercizio vedremo l'uso di variabili destinate a creare un nuovo Form e una nuova Label.

Esercizio 43: Creazione di oggetti durante l'esecuzione di un programma.

Apriamo un nuovo progetto e inseriamo nel Form1 un pulsante Button1, nella posizione che preferiamo.
Facciamo un doppio *clic* sul Button1; accediamo così alla Finestra del Codice, dove troviamo già impostata la procedura destinata a gestire l'evento del *clic* su questo pulsante Button1.
All'interno di questa procedura scriviamo la istruzioni affinché, quando l'utente farà un *clic* su questo pulsante, il programma crei un nuovo form, denominato NuovoForm, e all'interno del NuovoForm crei una nuova label denominata lblData, destinata a visualizzare la data e l'ora correnti.
Possiamo copiare e incollare il codice seguente: all'interno del quale sono spiegate le operazioni di dichiarazione e di creazione dei nuovi oggetti.

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' crea due variabili destinate a contenere due nuovi oggetti: un Form e
una Label
        ' (notare che la sintassi di questa operazione richiede la parola New =
nuovo):

        Dim NuovoForm As New Form
        Dim lblData As New Label

        With lblData
            ' Definisce le proprietà della nuova Label, destinata a visualizzare
la data e l'ora:
            .AutoSize = True
            .Location = New Point(50, 50)
            .Text = "Oggi è il " & Format(Now, "dd/MM/yyyy")
            .Text &= vbCrLf
```

[50] Paragrafo 71: Creare controlli dalla Finestra del Codice., a pag. 345.

```
            .Text &= vbCrLf
            .Text &= "Sono le ore " & Format(TimeOfDay, "HH:mm")
        End With

        With NuovoForm
            ' Definisce le proprietà del nuovo Form,
            ' aggiunge la Label lblData all'insieme degli oggetti del NuovoForm,
            ' poi visualizza il form:
            .Text = "Calendario"
            .StartPosition = FormStartPosition.CenterScreen
            .Controls.Add(lblData)
            .Show()
        End With

    End Sub

End Class
```

81: Matrici di variabili.

All'inizio di questo capitolo abbiamo paragonato le variabili a simboli apposti ai cassetti di una scrivania, in cui ogni cassetto è destinato a contenere un dato, un elemento, un oggetto.
VB prevede anche la possibilità di collocare in ogni cassetto non un solo oggetto, ma una serie di oggetti uguali, che proprio per il fatto di essere identici uno all'altro possono stare insieme, nello stesso cassetto, **con lo stesso nome ma distinti da un numero progressivo**.
Supponiamo di avere 10 quaderni identici: possiamo sistemare questi 10 quaderni in 10 cassetti diversi, un quaderno per ogni cassetto, contrassegnando i 10 cassetti con nomi diversi, oppure possiamo mettere tutti i quaderni assieme in un unico cassetto.
Questo cassetto unico potrà essere contrassegnato dal nome **Quaderni**; i quaderni in esso contenuti avranno come segno distintivo un numero progressivo.
Il contenuto del cassetto dei quaderni sarà dunque organizzato in questo modo:

- Quaderno(1)
- Quaderno(2)
- Quaderno(3)

Il contenuto di un cassetto simile, destinato a contenere **Matite**, potrebbe essere organizzato in questo modo:

- Matita(1)
- Matita(2)
- Matita(3)

e così via.

Una variabile destinata a contenere un gruppo di variabili dello stesso tipo, si chiama **matrice di variabili**.

Le matrici di variabili si creano come le variabili normali, con questa aggiunta: è necessario indicare al momento della loro creazione il numero degli elementi che faranno parte della matrice.
Ad esempio, per impostare uno schedario o un registro per una classe di 26 allievi è possibile creare queste matrici di variabili:

```
Dim Cognome(25) As String
Dim Nome(25) As String
Dim NumeroAssenze(25) As Integer
```

Siccome VB inizia a contare da 0, le tre dichiarazioni creano tre matrici di variabili con 26 elementi ognuna, numerati da 0 a 25.
L'assegnazione del contenuto alle matrici di variabili avviene nello stesso modo delle variabili normali:

```
Cognome(0) = "Malaguti"
Nome(0) = "Paola"
NumeroAssenze(0) = 12

Cognome(1) = "Rossi"
Nome(1) = "Paolo"
NumeroAssenze(1) = 12

Cognome(2) = "Colombo"
Nome(2) = "Giovanni"
NumeroAssenze(2) = 10

Cognome(3) = "Ferrari"
Nome(3) = "Giovanni"
NumeroAssenze(3) = 10
```

Una matrice di variabili può essere creata anche con questa sintassi:

```
Dim NumeriPrimi() As Integer = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}
```
[51]

Questa riga di codice crea una matrice di variabili di nome **NumeriPrimi**, destinata a contenere numeri interi. Notiamo che a differenza degli esempi precedenti il numero degli elementi di questa matrice non è indicato in anticipo tra le parentesi; in mancanza di questa indicazione, il programma ricava il numero degli elementi dall'elenco racchiuso tra le parentesi graffe.
In questo caso avremo a disposizione queste variabili:

```
NumeriPrimi(0) = 2
NumeriPrimi(1) = 3
NumeriPrimi(2) = 5
NumeriPrimi(3) = 7
' ...
NumeriPrimi(24) = 97
```

[51] La parentesi graffe si ottengono premendo sulla tastiera i tasti MAIUSC + ALT GR + parentesi quadre, oppure, nelle tastiere con il tastierino numerico, premendo i tasti ALT + 123 e ALT + 125.
I tasti ALT GR e ALT non hanno le stesse funzioni: il tasto ALT GR (grafica alternativa) si trova a destra della barra spazio, il tasto ALT (uso alternativo dei tasti) a sinistra della barra spazio.

L'uso delle matrici di variabili consente al programmatore di manipolare, estrapolare, modificare serie di dati scrivendo poche righe di istruzioni, utilizzando i cicli di comandi ripetuti, come il ciclo **For… Next** che vedremo nel prossimo capitolo.

82: Matrici di oggetti.

La tecnica delle matrici si applica non solo alle variabili, ma **a qualsiasi oggetto** di VB. Ecco alcuni esempi.

Questo comando crea una matrice di quattro oggetti **Point** destinati, ad esempio, a tracciare una linea segmentata:

```
Dim Punti() As Point = {New Point(30, 30), New Point(230, 30), New Point(200, 230), New Point(100, 230)}
```

Questo comando crea una matrice di quattro oggetti **Rectangle** destinati, ad esempio, a essere disegnati in serie uno dopo l'altro:

```
        Dim Rettangoli As Rectangle() = {New Rectangle(10, 10, 100, 150), New Rectangle(120, 10, 100, 50), New Rectangle(120, 70, 100, 50)}
```

Ecco una matrice di oggetti **Color**, che potrebbe essere utilizzata, ad esempio, per assegnare il colore di fondo a un controllo utilizzando il numero d'ordine che ogni colore occupa nella matrice:

```
Public Class Form1

    Dim Colore() As Color = {Color.Yellow, Color.Red, Color.Blue, Color.Black, Color.White, Color.Green}

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles Me.Load

        PictureBox1.BackColor = Colore(0)
        PictureBox2.BackColor = Colore(1)
        PictureBox3.BackColor = Colore(2)
        PictureBox4.BackColor = Colore(3)
        PictureBox5.BackColor = Colore(4)
        PictureBox6.BackColor = Colore(5)

    End Sub

End Class
```

Oppure, meglio:

```
Public Class Form1

    Dim Colore() As Color = {Color.Yellow, Color.Red, Color.Blue, Color.Black,
Color.White, Color.Green}

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim Contatore As Integer

        For Each Riquadro As PictureBox In Me.Controls
            Riquadro.BackColor = Colore(Contatore)
            Contatore += 1
        Next

    End Sub

End Class
```

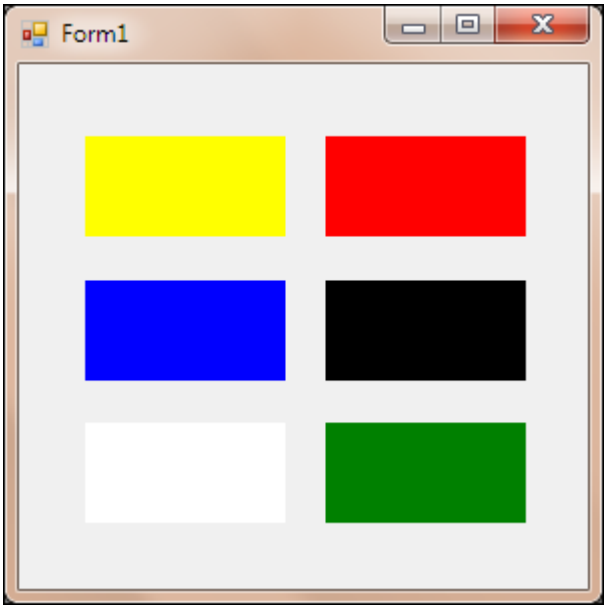


Figura 131: Utilizzo di una matrice di Colori.

83: Liste di variabili.

Le matrici di variabili sono state per decenni lo strumento primario per memorizzare e trattare dati di contenuto simile.
Si è andata affermando tuttavia, nel tempo, una tecnica diversa, che si è rivelata più flessibile: la tecnica delle **liste di variabili**.
Vediamo le principali differenze tra le due tecniche.

- Quando si crea una lista di variabili non è necessario dichiarare in via prioritaria il numero delle variabili che vi saranno memorizzate.

- Il numero della posizione di un elemento in una lista non è importante come la posizione di un elemento in una matrice.
- Gli elementi nuovi inseriti in una lista di variabili sono aggiunti con il comando **Add**, mentre gli elementi esistenti possono essere cancellati con i comandi
  - **RemoveAt(indice),** per rimuovere un elemento dalla lista indicandone il numero d'ordine;
  - **Remove(elemento),** per rimuovere un elemento dalla lista indicandone il nome;
  - **RemoveRange(inizio, fine),** per rimuovere gli elementi che si trovano all'interno dell'intervallo indicato.
- Il comando **Sort** consente di mettere in ordine progressivo gli elementi della lista.

Il comando per la creazione di una lista di variabili ha questa sintassi:

```
Dim Nomi As New List(Of String)
Nomi.Add("Paolo")
Nomi.Add("Luigi")
```

Notiamo che il programmatore deve definire due elementi:
- il nome da dare alla lista
- e, tra parentesi, richiede il tipo di dati che saranno inseriti nella lista stessa.

Tutti gli elementi raccolti in una lista List(Of String) devono essere di tipo String (testo).

Ecco la creazione di una lista di colori:

```
Dim Colori As New List(Of Color)
Colori.Add(Color.Red)
Colori.Add(TextBox1.BackColor)
```

Nel prossimo esempio vediamo la creazione e l'utilizzo di una lista di parole (stringhe di testo). I nomi della lista sono visualizzati in un controllo Label1 secondo l'ordine di inserimento nella lista, mentre in un controllo Label2 sono visualizzati in ordine alfabetico.

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Label1.Text = ""
        Label2.Text = ""

        ' Crea una lista di nomi:
        Dim Dinosauri As New List(Of String)
        ' Aggiungi alla lista alcuni nomi di dinosauri:
        Dinosauri.Add("Coelophysis")
        Dinosauri.Add("Amargasaurus")
        Dinosauri.Add("Mamenchisaurus")
        Dinosauri.Add("Deinonychus")
        Dinosauri.Add("Tyrannosaurus")
        Dinosauri.Add("Majungasaurus")
```

```
        Dinosauri.Add("Albertosaurus")

        ' Legge i nomi della lista uno ad uno e li visualizza nella Label1:
        For Each Nome In Dinosauri
            Label1.Text &= Nome & vbCrLf
        Next

        ' Mette in ordine alfabetico i nomi della lista:
        Dinosauri.Sort()

        ' Legge i nomi della lista uno a uno e li visualizza nella Label2:
        For Each Nome As String In Dinosauri
            Label2.Text &= Nome & vbCrLf
        Next
    End Sub

End Class
```

Nel prossimo esercizio vedremo le operazioni di creazione di una lista di variabili di tipo String e diverse operazioni possibili con gli elementi contenuti nella lista.
Vedremo come l'uso delle liste di variabili consente al programmatore di manipolare, estrapolare, modificare serie di dati scrivendo poche righe di istruzioni, utilizzando i cicli di comandi ripetuti, e in particolare il ciclo retto dai comandi **For Each… Next** che sarà illustrato nel prossimo capitolo.

Esercizio 44: Operazioni con una lista di variabili di tipo String.

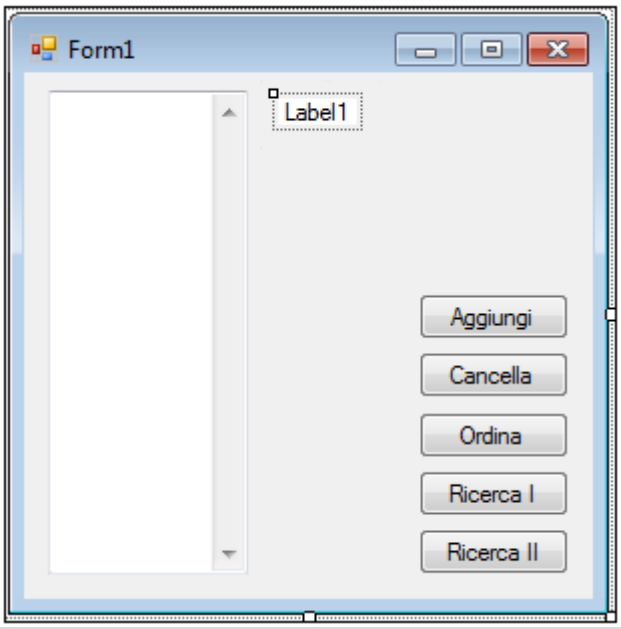
In questo esercizio vediamo la creazione di una lista di nomi di animali e la sua visualizzazione all'interno di un TextBox.
Nel form sono presenti cinque pulsanti, ad ognuno dei quali è affidata una di queste operazioni:

- aggiunta di nuovi nomi di animali;
- cancellazione di nomi;
- ordinamento alfabetico dei nomi;
- ricerca di nomi o di parti di nomi nell'elenco;
- ricerca del nome corrispondente ad un determinato numero d'ordine nella lista.

Inseriamo nel Form1

- un controllo **TextBox**
- un controllo **Label** e
- cinque pulsanti **Button**

come in questa immagine:



Proprietà del TextBox1:
- MultiLine = True
- ScrollBars = Vertical

Proprietà della Label1:
- BackColor = White

Proprietà dei cinque pulsanti Button:
I
- Name = cmdAggiungi
- Text = Aggiungi

II
- Name = cmdCancella
- Text = Cancella

III
- Name = cmdOrdina
- Text = Ordina

IV
- Name = cmdRicercaI
- Text = Ricerca I

V
- Name = cmdRicercaII
- Text = Ricerca II

Il codice seguente, da copiare e incollare nella Finestra del Codice, è ampiamente commentato e la lettrice o il lettore vi troveranno le informazioni per capirne il funzionamento.

Esso presenta tuttavia alcuni elementi di programmazione che sono illustrati più avanti nel manuale:

- il ciclo di comandi ripetuti di tipo **For Each… Next**;
- il procedimento di decisione basato su **If… Then**;
- la funzione **String.Contains**.

```
Public Class Form1

    ' Crea la lista di variabili Animali e le assegna un elenco di nomi:
    ' Notare che per l'assegnazione in questo caso si usa il comando
    ' From + l'elenco dei nomi tra parentesi graffe, separati da virgole.

    Dim Animali As New List(Of String) From {"zebra", "pulce", "cane", "gatto",
"leone", "coniglio", "merlo", "asino", "zanzara", "toro", "castoro", "picchio",
"zibellino", "albatros"}

    Private Sub Form1_Load() Handles MyBase.Load

        Label1.Text = ""

        ' attiva la procedura che visualizza nel TextBox tutti gli elementi della
lista Animali:
        Call VisualizzaElenco()

    End Sub
```

```
    Private Sub VisualizzaElenco()

        ' Questa è la procedura che visualizza nel TextBox tutti gli elementi
della lista Animali.

        TextBox1.Text = "Elenco generale:" & vbCrLf & vbCrLf

        ' Per ogni parola presenta nella lista Animali...
        For Each Parola As String In Animali
            ' ... aggiungi questa parola alla Label2
            TextBox1.Text &= Parola & vbCrLf
        Next

    End Sub
```

```
    Private Sub cmdAggiungi_Click(sender As System.Object, e As System.EventArgs)
Handles cmdAggiungi.Click

        ' Crea un InputBox nel quale l'utente del programma deve scrivere un
nome.
        ' Quando l'utente chiude l'InputBox, il nome scritto viene aggiunto alla
lista Animali:

        Animali.Add(InputBox("Scrivi il nome dell'animale da aggiungere alla
lista."))

        ' attiva la procedura che visualizza nel TextBox tutti gli elementi della
lista Animali:
        Call VisualizzaElenco()

    End Sub
```

```
    Private Sub cmdOrdina_Click(sender As System.Object, e As System.EventArgs)
Handles cmdOrdina.Click

        ' Metti i nomi della lista in ordine alfabetico:
        Animali.Sort()

        ' attiva la procedura che visualizza nel TextBox tutti gli elementi della
lista Animali:
        Call VisualizzaElenco()

    End Sub
```

```
    Private Sub cmdRicercaI_Click(sender As System.Object, e As System.EventArgs)
Handles cmdRicercaI.Click

        ' Crea un InputBox nel quale l'utente del programma deve scrivere una
parola.
        ' Quando l'utente chiude l'InputBox, la parola scritta è assegnata alla
variabile Trova:

        Dim Trova As String = InputBox("Scrivi il nome o le lettere del nome da
cercare.")

        Label1.Text = "Risultato della ricerca:" & vbCrLf & vbCrLf

        ' Per ogni parola presenta nella lista Animali...
        For Each Parola As String In Animali

            ' ... se questa parola contiene la stringa da cercare, aggiungila
alla Label1
            If Parola.Contains(Trova) Then
                Label1.Text &= Parola & vbCrLf
            End If
        Next

    End Sub
```

```
    Private Sub cmdRicercaII_Click(sender As System.Object, e As
System.EventArgs) Handles cmdRicercaII.Click

        ' Crea un InputBox nel quale l'utente del programma deve scrivere un
numero.
        ' Quando l'utente chiude l'InputBox, il numero scritto viene assegnato
alla variabile Trova:

        Dim Trova As Integer = InputBox("Scrivi numero d'ordine dell'animale che
ti interessa.")

        ' Scrive nella Label1 il nome il cui numero d'ordine nella lista
corrisponde al numero scritto dall'utente
        Label1.Text = "Risultato della ricerca:" & vbCrLf & vbCrLf &
Animali(Trova - 1)

    End Sub
```

```
    Private Sub cmdCancella_Click(sender As System.Object, e As System.EventArgs)
Handles cmdCancella.Click

        ' Visualizza un InputBox nel quale l'utente deve scrivere un nome.
```

```
        ' Quando l'utente chiude l'InputBox, il programma cancella il nome
scritto dalla lista

        Animali.Remove(InputBox("Scrivi il nome dell'animale che vuoi cancellare
dalla lista."))

        ' attiva la procedura che visualizza nel TextBox tutti gli elementi della
lista Animali:
        Call VisualizzaElenco()

    End Sub

End Class
```

Notiamo in particolare, nel codice, i comandi che gestiscono le operazioni di

- inserimento di nuovi nomi nella lista:

```
        Animali.Add()
```

- cancellazione di nomi dalla lista:

```
        Animali.Remove()
```

- ordinamento alfabetico dei nomi nella lista:

```
        Animali.Sort()
```

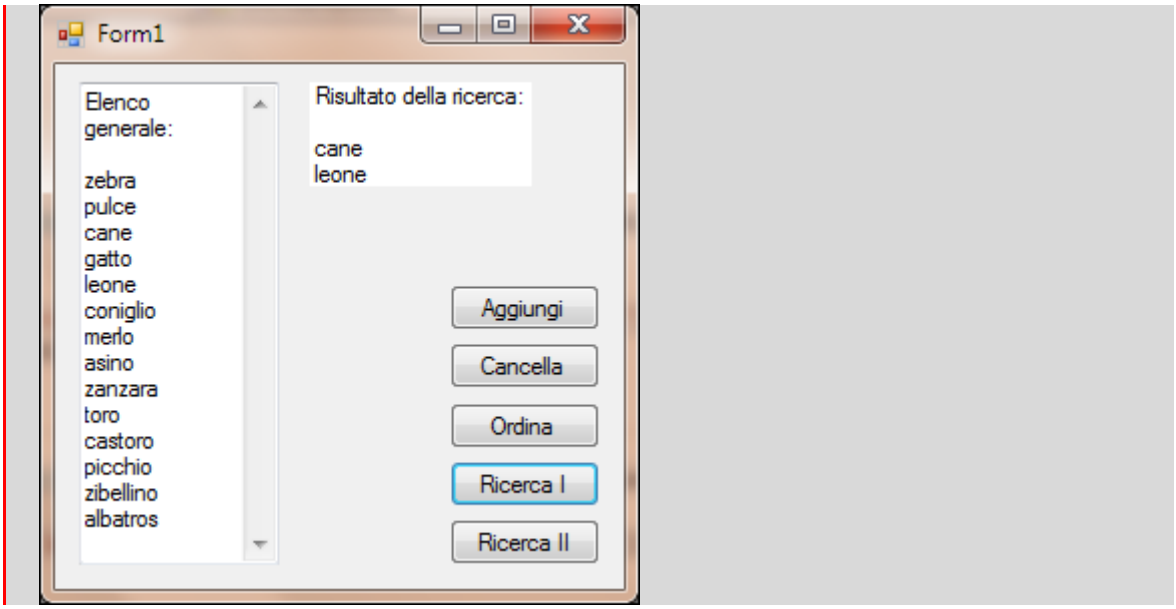
- ricerca di nomi o di parti di nomi nella lista:

```
            If Parola.Contains(Trova) Then
```

- ricerca del nome corrispondente ad un determinato numero d'ordine nella lista:

```
        Label1.Text = Animali(Trova - 1)
```

Ecco un'immagine del programma in esecuzione:



84: Liste di variabili e controlli ListBox.

Ci sono molte analogie tra i controlli **ListBox** e le liste di variabili di tipo **List(Of…)** che abbiamo visto al paragarafo precedente.
Il loro scopo è identico: memorizzare un elenco di item che l'utente del programma potrà vedere, ordinare e modificare in vari modi.
Molti comandi e istruzioni relativi a queste operazioni sono simili per i due controlli.
La differenza, sulla quale il programmatore deve orientare la sua scelta, consiste in questo:

- la lista di dati memorizzati in una lista **List(Of…)** è uno strumento **interno** al programma, invisibile all'utente, destinato a produrre effetti solo interagendo con altri contolli mediante il codice del programma;
- un **ListBox** invece è un oggetto concreto che si mostra direttamente all'utente, visualizzando i dati in esso contenuti.

Il programma seguente mette in risalto questa differenza.
In questo programma abbiamo un controllo ListBox1, visibile all'utente, con una lista di nomi comuni al singolare, e abbiamo una lista di tipo List(Of String), denominata NomiPlurali, invisibile all'utente, con i plurali dei nomi riportati nel ListBox1.
Quando l'utente seleziona un nome nel ListBox1, è visualizzato in un MsgBox il nome corrispondente, plurale, prelevato dalla lista NomiPlurali.

```
Public Class Form1

    ' Crea una lista, che rimane invisibile all'utente, con i nomi al plurale:
    Dim NomiPlurali As New List(Of String) From {"ali", "armi", "braccia",
"buoi", "centinaia", "ciglia", "dei", "dita", "labbra", "ginocchia", "lenzuola",
"orecchie", "paia", "uomini", "uova"}
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Me.Text = "PLURALI IRREGOLARI"
        ' All'avvio del programma immette nel ListBox1 15 nomi al singolare:
        With ListBox1
            .Items.Add("ala")
            .Items.Add("arma")
            .Items.Add("braccio")
            .Items.Add("bue")
            .Items.Add("centinaio")
            .Items.Add("ciglio")
            .Items.Add("dio")
            .Items.Add("dito")
            .Items.Add("labbro")
            .Items.Add("ginocchio")
            .Items.Add("lenzuolo")
            .Items.Add("orecchio")
            .Items.Add("paio")
            .Items.Add("uomo")
            .Items.Add("uovo")
        End With

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged(sender As System.Object, e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

        ' Quando l'utente clicca un nome nel ListBox1, è visualizzato un
messaggio con il nome
        ' corrispondente, al plurale, che ha lo stesso numero d'ordine nella
lista NomiPlurali:

        Dim NomeCliccato As Integer = ListBox1.SelectedIndex
        MsgBox("Il plurale di " & ListBox1.SelectedItem & " è " &
NomiPlurali(NomeCliccato))

    End Sub

End Class
```

Ecco un’immagine del programma in esecuzione:

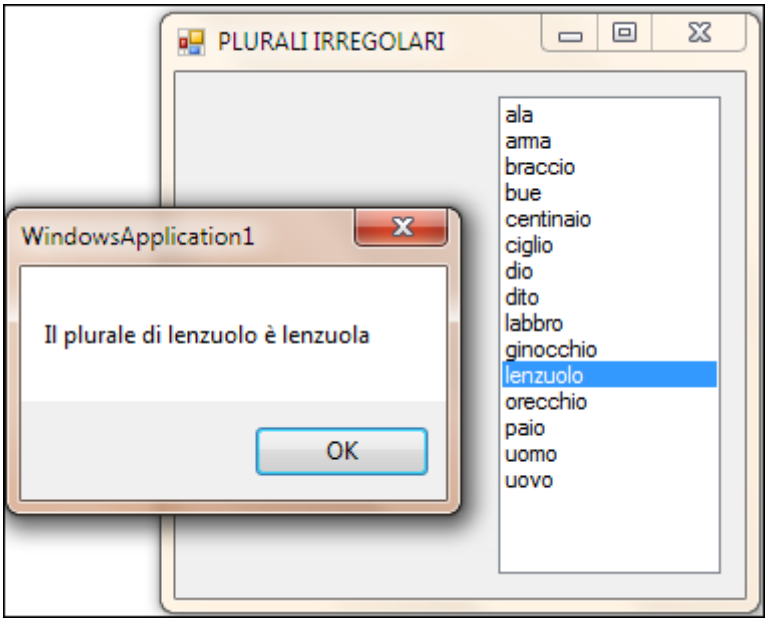


Figura 132: Un ListBox e una lista di variabili di tipo List (Of String).

ListBox e List (Of …) hanno comandi simili per gestire gli elementi della lista:

- per aggiungere un elemento alla lista:

```
Lista.Add("Paolo Rossi")
ListBox1.Items.Add("Paolo Rossi")
```

- per cancellare un elemento in base al suo numero d'ordine nella lista[52]:

```
Lista.RemoveAt(numeroX)
ListBox1.Items.RemoveAt(numeroX)
```

- per cancellare un elemento in base al suo contenuto:

```
Lista.Remove("Paolo Rossi")
ListBox1.Items.Remove("Paolo Rossi")
```

- per effettuare una ricerca tra gli elementi:

```
If Lista.Contains("xy") Then Beep()
If ListBox1.Items.Contains("xy") Then Beep()
```

- per sapere quanti elementi ci sono in una lista:

```
xy = Lista.Count
xy = ListBox1.Items.Count
```

- per ordinare gli elementi di una lista:

[52] E' necessario tenere presente che la numerazione degli elementi parte sempre da 0, per cui se in una lista ci sono 10 elementi, il primo elemento ha il numero d'ordine = 0, l'ultimo = 9.

```
Lista.Sort()
ListBox1.Sorted = True
```

- per conoscere il contenuto di un elemento in base al suo numero d'ordine nella lista:

```
Label1.Text = Lista(numeroX)
Label1.Text = ListBox1.Items(numeroX)
```

- per cancellare tutti gli elementi di una lista:

```
Lista.Clear()
ListBox1.Items.Clear()
```

- per riversare in un ListBox il contenuto di una lista:

```
ListBox1.Items.AddRange(Lista.ToArray)
```

- e, viceversa, per riversare in una lista il contenuto di un ListBox:

```
Lista.AddRange(ListBox1.Items.Cast(Of String))
```

Oltre a questi comandi, il controllo ListBox riconosce i comandi che riguardano la **selezione** di un elemento da parte dell'utente, cosa che non può essere fatta direttamente con gli elementi di una lista di tipo List (Of…):

- per selezionare un elemento in base al suo numero d'ordine nel ListBox:

```
ListBox1.SelectedIndex = xy
```

- viceversa, per ottenere il numero d'ordine di un elemento selezionato nel ListBox:

```
xy = ListBox1.SelectedIndex
```

- per ottenere il contenuto di un elemento selezionato nel ListBx:

```
xy = ListBox1.SelectedItem
```

Un controllo ListBox può visualizzare, oltre a testi e numeri, anche immagini. Un esempio di questa possibilità si trova al paragrafo 151, **Inserire immagini in un ListBox**., a pag. 662.
Nel prossimo esercizio vedremo alcuni esempi di gestione dei dati in un ListBox.

Esercizio 45: Gestione di dati con un ListBox.

In questo esercizio vediamo la creazione e la gestione di una lista con nomi di persone visualizzato in un controllo ListBox.

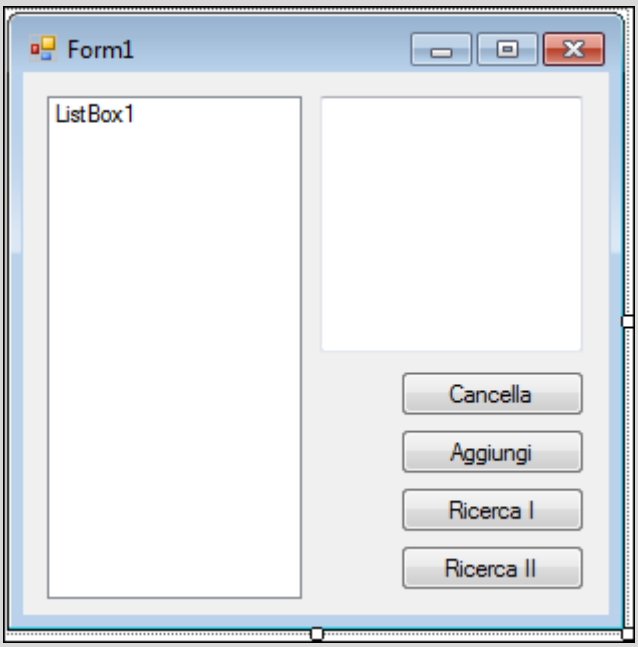
Nel form sono presenti quattro pulsanti, a ognuno dei quali è affidata una di queste operazioni:

- aggiunta di nuovi nomi di animali;
- cancellazione di nomi;
- ricerca di nomi o di parti di nomi nell'elenco;
- ricerca del nome corrispondente ad un determinato numero d'ordine nella lista.

Inseriamo nel Form1

- un controllo **TextBox** e
- quattro pulsanti **Button**

come in questa immagine:



Proprietà dei quattro pulsanti Button:

I

- Name = cmdCancella
- Text = Cancella

II

- Name = cmdAggiungi
- Text = Aggiungi

III

- Name = cmdRicercaI
- Text = Ricerca I

IV

- Name = cmdRicercaII
- Text = Ricerca II

Il codice seguente, da copiare e incollare nella Finestra del Codice, è ampiamente commentato; anche qui sono presenti alcuni elementi di programmazione che la lettrice o il lettore troveranno illustrati dettagliatamente più avanti nel manuale:

- il ciclo di comandi ripetuti di tipo **For… Next**;
- il procedimento di decisione basato su **If… Then**;
- la funzione **String.Contains**.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea una lista di Cognomi e una lista di Nomi che serviranno a riempire il
ListBox1 di 300 nomi casuali:
    Dim Cognomi As New List(Of String) From {"Rossi", "Castagna", "Bollati",
"Parisi", "Loiero", "Trevisan", "Boscolo", "Carraro", "Sartori", "Pavan",
"Ferrari", "Vianello", "Zuretti", "Basso", "Visentin", "Farina", "Moro", "Costa",
"Rossini", "Bandiera", "Boldrini", "Avanzi", "Corradi", "Salteri", "Fabbri",
"Crovetto"}
    Dim Nomi As New List(Of String) From {"Giacomo", "Stefano", "Enrico",
"Antonio", "Paolo", "Giulia", "Maria", "Marco", "Giulio", "Marta", "Luisa",
"Genoveffa", "Claudio", "Terenzio", "Paola", "Luigi", "Gino", "Michele",
"Francesca", "Daria", "Angela", "Patrizia", "Sabrina", "Pasquale", "Roberta",
"Roberto", "Angelo"}
```

```
    Private Sub cmdCancella_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdCancella.Click

        ' Cancella la persona corrispondente al numero d'indice dell'item
selezionato nel ListBox1:
        ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)
        TextBox1.Text = "La lista contiene " & ListBox1.Items.Count & " persone."
        cmdCancella.Enabled = False

    End Sub
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ListBox1.Sorted = True
        cmdCancella.Enabled = False
        TextBox1.ScrollBars = ScrollBars.Vertical

        Dim Sorteggio As New Random
        Randomize()
        Dim NuovaPersona As String

        ' Crea i nomi di 300 persone unendo Cognomi e Nomi scelti a caso dalle
due liste:

        For Contatore As Integer = 0 To 299
            NuovaPersona = Cognomi(Sorteggio.Next(1, Cognomi.Count)) & " "
            NuovaPersona &= Nomi(Sorteggio.Next(1, Nomi.Count))

            ' Se la NuovaPersona esiste già nella lista, il Contatore viene
riportato indietro di un
            ' numero e il nome non viene inserito nel ListBox1, altrimenti...
            If ListBox1.Items.Contains(NuovaPersona) Then
                Contatore -= 1
            Else
                ' il nuovo nome è inserito nel ListBox1:
                ListBox1.Items.Add(NuovaPersona)
            End If
```

```
        Next

        TextBox1.Text = "La lista contiene " & ListBox1.Items.Count & " persone."

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged(sender As System.Object, e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

        ' quando l'utente seleziona un item nel ListBox1, lo stesso item è
visualizzato nel TextBox, con il suo numero d'indice.
        ' Il primo nome nel ListBox ha l'indice = 0, ma per l'utente il primo
nome ha il numero 1, e così
        ' per tuttu gli altri nomi, per cui è necessaria una correzione:

        TextBox1.Text = ListBox1.SelectedItem & vbCrLf & "è al numero " &
ListBox1.SelectedIndex + 1 & "."
        cmdCancella.Enabled = True

    End Sub
```

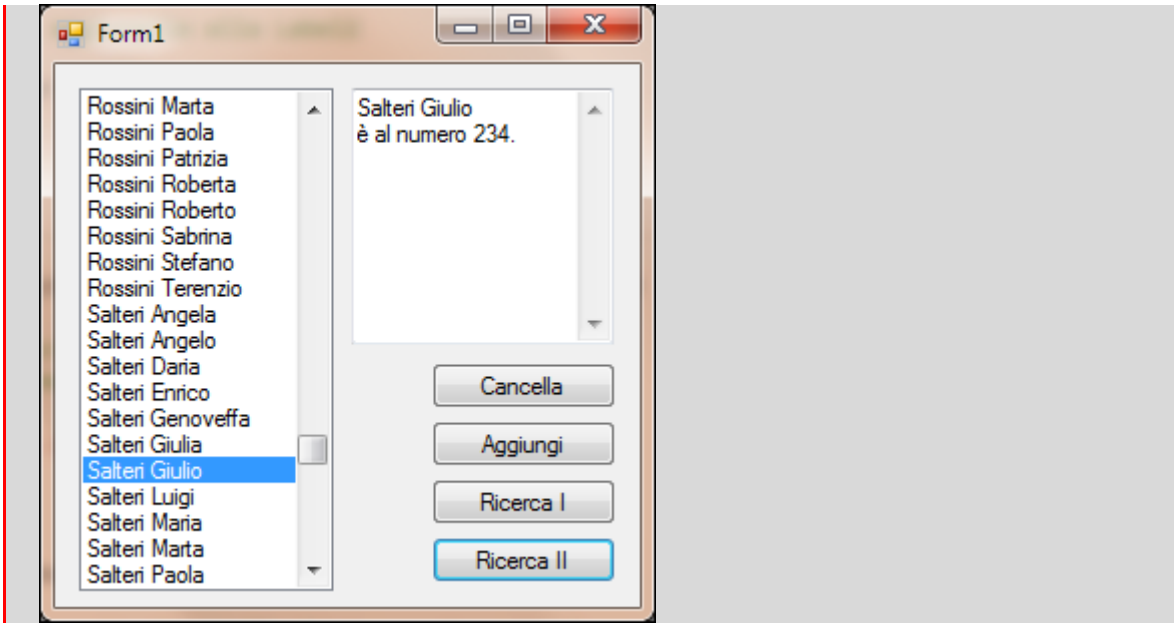
```
    Private Sub cmdAggiungi_Click(sender As System.Object, e As System.EventArgs)
Handles cmdAggiungi.Click

        Dim Sorteggio As New Random
        Randomize()
        Dim NuovaPersona As String

        NuovaPersona = Cognomi(Sorteggio.Next(1, Cognomi.Count)) & " "
        NuovaPersona &= Nomi(Sorteggio.Next(1, Nomi.Count))

        ' Visualizza un InputBox in cui l'utente può scrivere il nome di una
nuova persona (il programma
        ' propone un nuovo nome elaborato casualmente):

        NuovaPersona = InputBox("Scrivi cognome e nome della persona da
aggiungere", "Aggiungi", NuovaPersona)

        If ListBox1.Items.Contains(NuovaPersona) Then
            MsgBox("Questa persona è già nell'elenco.")
        Else
            ListBox1.Items.Add(NuovaPersona)
            TextBox1.Text = "La lista contiene " & ListBox1.Items.Count & "
persone."
        End If

    End Sub
```

```
    Private Sub cmdRicercaI_Click(sender As System.Object, e As System.EventArgs)
Handles cmdRicercaI.Click

        ' Crea un InputBox nel quale l'utente del programma può immettere un
cognome, un nome o un gruppo di lettere.
        ' Quando l'utente chiude l'InputBox, la parola scritta viene assegnata
alla variabile Trova:

        Dim Trova As String = InputBox("Scrivi un cognome, un nome o un gruppo di
lettere da cercare.", "Ricerca")
```

```
        TextBox1.Text = "Risultato della ricerca:" & vbCrLf & vbCrLf

        ' Scorre i nomi delle persone presenti nel ListBox...
        For Contatore = 0 To ListBox1.Items.Count - 1
            ' ... se questa parola contiene la stringa da cercare, aggiungila
alla Label2
            If ListBox1.Items(Contatore).Contains(Trova) Then
                TextBox1.Text &= ListBox1.Items(Contatore) & vbCrLf
            End If
        Next

    End Sub
```

```
    Private Sub cmdRicercaII_Click(sender As System.Object, e As
System.EventArgs) Handles cmdRicercaII.Click

        ' Crea un InputBox nel quale l'utente del programma deve scrivere un
numero.
        ' Quando l'utente chiude l'InputBox, il numero scritto viene assegnato
alla variabile Trova:

        Dim Trova As Integer = InputBox("Scrivi numero nella lista che ti
interessa.", "Ricerca")
        Trova -= 1

        If Trova > ListBox1.Items.Count - 1 Then
            MsgBox("Il numero non può essere maggiore di " &
ListBox1.Items.Count)
        Else
            ' Scrive nella Labe11 il nome della persona il cui numero d'ordine
nella lista corrisponde al numero immesso dall'utente:
            ListBox1.SelectedIndex = Trova
        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



85: Correzione degli errori nelle dichiarazioni di variabili.

IntelliSense opera attivamente nell'*editor* di VB per assistere il programmatore nell'uso delle variabili, quando questi incorre in errori di battitura o in dimenticanze.
L'errore più frequente che il programmatore può commettere consiste probabilmente nel richiamare una variabile, creata in precedenza, scrivendone il nome in modo errato.
Supponiamo che il programmatore abbia creato una variabile di tipo Integer con il nome **RisposteEsatte** e che, più avanti nel codice, si riferisca alla stessa variabile chiamandola invece **RispostEsatte**:

```
Dim RisposteEsatte As Integer
RispostEsatte = 12
```

In questo caso il programma non sarà in grado di riconoscere la variabile **RispostEsatte**, per cui **IntelliSense** segnala prontamente l'errore al programmatore, in coda alla riga in cui si è verificato l'errore e nella finestra **Elenco errori** che si trova sotto la Finestra del Codice[53]:

[53] Se la finestra non è visibile, è possibile ripristinarne la visualizzazione cliccando il menu **Visualizza / Altre finestre / Elenco errori**.

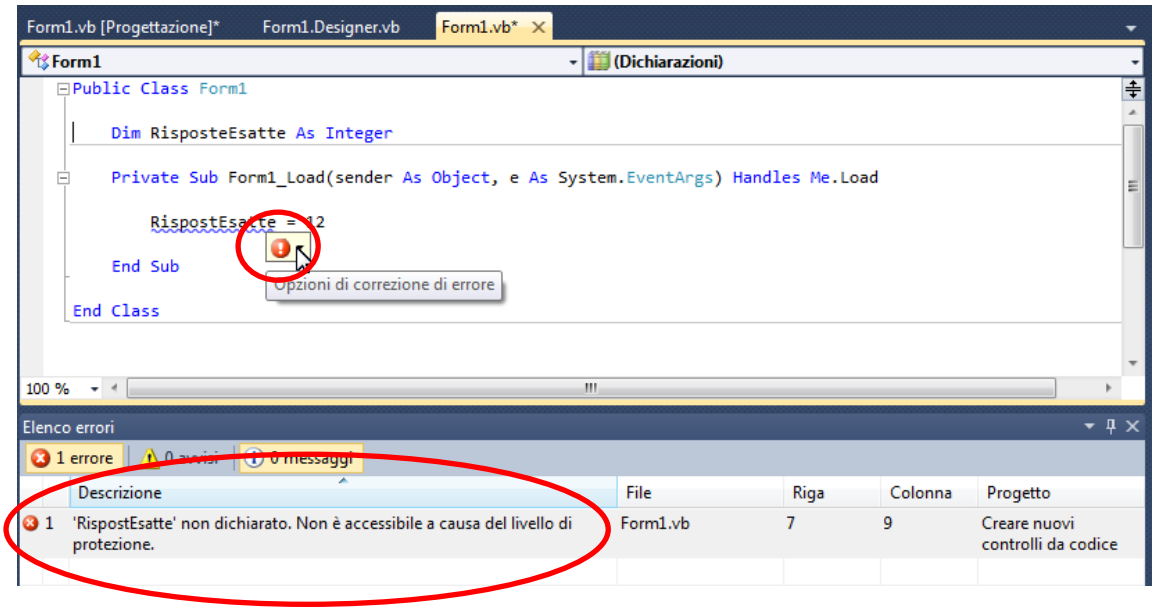


Figura 133: Segnalazione di errori da parte di IntelliSense.

Quando il programmatore crea una nuova variabile, l'*editor* del codice segnala che la variabile non è ancora stata utilizzata, cioè non è stata inizializzata:

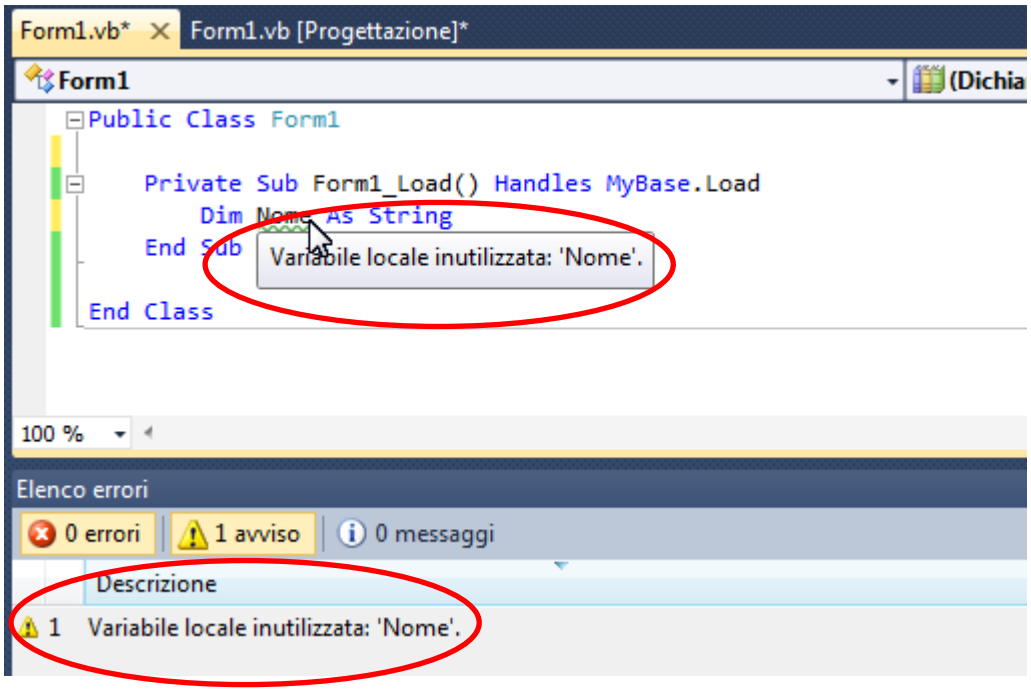
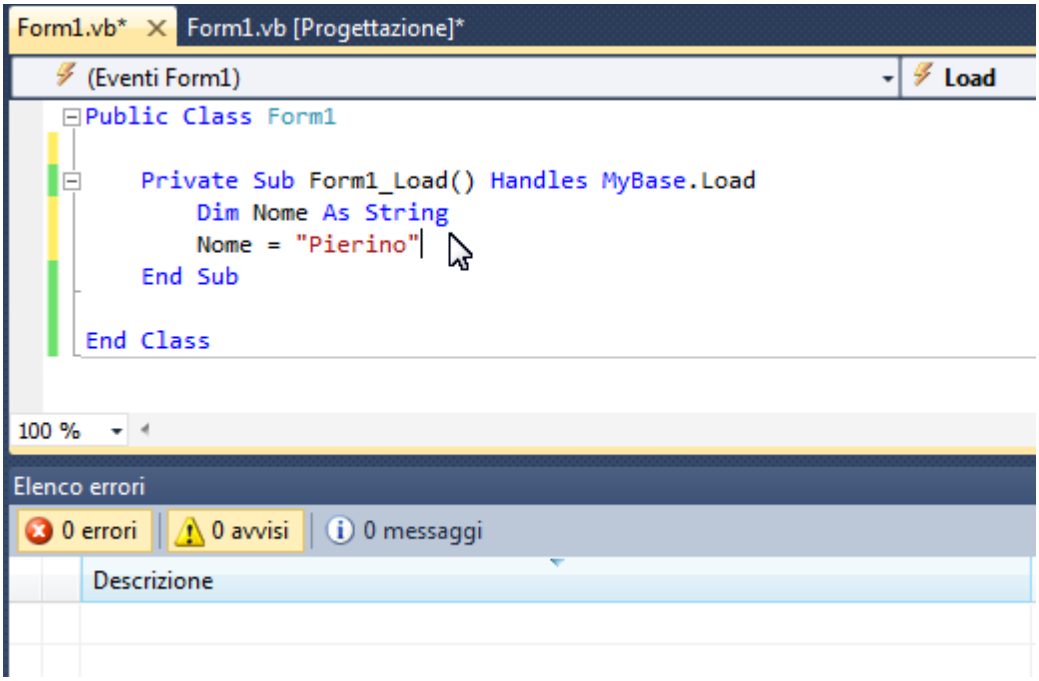


Figura 134: Segnalazione di una variabile non utilizzata.

Questi avvisi scompaiono nel momento in cui alla variabile in questione è assegnato un dato:


Form1.vb*
Form1.vb [Progettazione]*
(Eventi Form1)
Load
Public Class Form1
Private Sub Form1_Load() Handles MyBase.Load
Dim Nome As String
Nome = "Pierino"
End Sub
End Class
100 %
Elenco errori
0 errori
0 avvisi
0 messaggi
Descrizione

Capitolo 14: I CICLI DI COMANDI RIPETUTI.

In un programma può accadere al programmatore di dovere ripetere una serie di comandi uguali, più e più volte.
Supponiamo di creare un programma che visualizzi la serie dei numeri pari, da 2 a 100. Per ottenere questo risultato, il programmatore può dichiarare una matrice di variabili di 50 elementi, e all'avvio del programma può assegnare a ogni elemento uno dei 50 numeri pari:

```
Public Class Form1

    Dim NumeroPari(50) As Integer

    Private Sub Form1_Load() Handles MyBase.Load

        NumeroPari(1) = 2
        NumeroPari(2) = 4
        NumeroPari(3) = 6
        NumeroPari(4) = 8
        NumeroPari(5) = 10
        NumeroPari(6) = 12
        NumeroPari(7) = 14
        ' ecc., sino ad arrivare a
        NumeroPari(50) = 100

    End Sub

End Class
```

Per abbreviare e semplificare questo codice si può ricorrere a un ciclo di comandi ripetuti: un ciclo che dovrà ripetersi 50 volte, aumentando di due unità, a ogni passaggio, il valore assegnato alla variabile NumeroPari.
Le istruzioni da impartire al programma per effettuare questo ciclo possono essere descritte in questo modo:

- ripeti il comando seguente per 50 volte:
- assegna alla variabile NumeroPari() un numero maggiore di due unità rispetto al numero assegnato alla variabile NumeroPari() precedente;
- passa alla variabile NumeroPari( ) successiva.

Nel prossimo esercizio vedremo come scrivere queste istruzioni nella sintassi di VB.

Esercizio 46: Un ciclo di comandi ripetuti.

Questo programma crea una matrice di 50 variabili.
A ogni variabile è assegnato un numero pari, da 2 a 100.
Apriamo un nuovo progetto e, senza inserire alcun oggetto nel Form, copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea una matrice di variabili di nome NumeroPari, con 51 variabili:
    Dim NumeroPari(50) As Integer

    Private Sub Form1_Load() Handles MyBase.Load

        ' assegna il valore 0 alla prima variabile:
        NumeroPari(0) = 0

        ' crea una variabile di nome Contatore, che servirà a contare le
        ' ripetizioni del ciclo, sino a raggiungere il numero di 50 ripetizioni:
        Dim Contatore As Integer

        ' Avvia il ciclo di 50 ripetizioni: la variabile Contatore parte dal
numero 1, aumenterà di una unità a ogni ripetizione e arriverà a 50 con l'ultima
ripetizione:
        For Contatore = 1 To 50

            ' Assegna alla variabile NumeroPari(), il cui numero d'ordine
corrisponde al numero della variabile Contatore, il numero pari della variabile
precedente, amentato di due unità:
            NumeroPari(Contatore) = NumeroPari(Contatore - 1) + 2

            ' Effettuata questa operazione il ciclo torna da capo e passa alla
variabile NumeroPari() successiva, sino a quando la variabile Contatore sarà
uguale a 50:

        Next

    End Sub
```

Il ciclo di comandi ripetuti contiene dunque solo un comando, che viene ripetuto 50 volte. Lo vediamo in questo codice, ripulito dei commenti, evidenziato in giallo::

```
        For Contatore = 1 To 50
            NumeroPari(Contatore) = NumeroPari(Contatore - 1) + 2
        Next
```

Vediamo di capirne il funzionamento.
Supponiamo di essere all'inizio del ciclo: la variabile Contatore è uguale a 1.
Il comando

```
            NumeroPari(Contatore) = NumeroPari(Contatore - 1) + 2
```

può dunque essere letto in questo modo:

```
            NumeroPari(1) = NumeroPari(1 - 1) + 2
```

Cioè: assegna alla variabile NumeroPari(1) il valore della variabile precedente, cioè della variabile NumeroPari(0), aumentandolo di 2 unità; in questo modo, alla variabile NumeroPari(1) è assegnato il numero 2.

Supponiamo ora di essere verso la metà del ciclo: la variabile Contatore è uguale a 25. Il comando

```
NumeroPari(Contatore) = NumeroPari(Contatore - 1) + 2
```

può dunque essere letto in questo modo:

```
NumeroPari(25) = NumeroPari(25 - 1) + 2
```

Cioè: assegna alla variabile NumeroPari(25) il valore della variabile precedente, cioè della variabile NumeroPari(24), aumentandolo di 2 unità; in questo modo, alla variabile NumeroPari(25) è assegnato il numero 50.

Alla fine del ciclo, avremo dunque a disposizione nel programma 50 variabili, ognuna delle quali avrà come contenuto un numero pari:

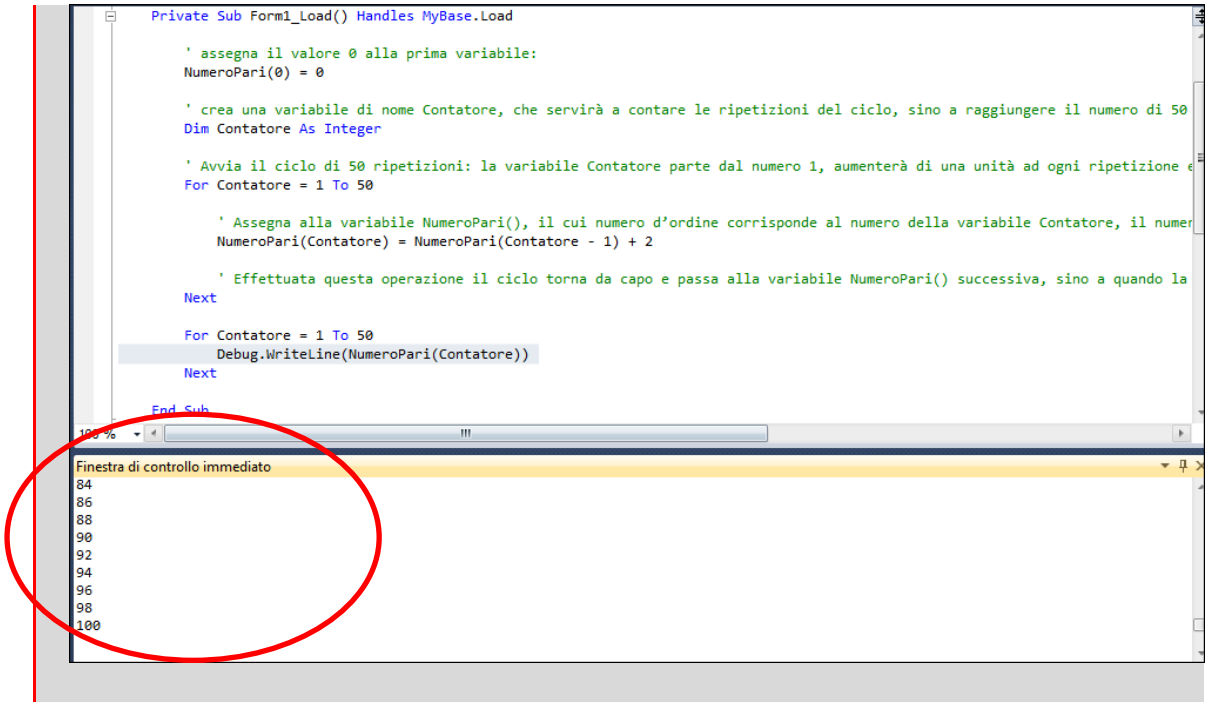
```
NumeroPari(1) = 2
NumeroPari(2) = 4
NumeroPari(3) = 6
NumeroPari(4) = 8
NumeroPari(5) = 10
NumeroPari(6) = 12
NumeroPari(7) = 14
' ecc., sino a
NumeroPari(50) = 100
```

Per controllare se le cose stanno effettivamente così, inseriamo nel codice un altro ciclo di comandi ripetuti, anche questo con un solo comando, da ripetere 50 volte:

```
For Contatore = 1 To 50
    Debug.WriteLine(NumeroPari(Contatore))
Next
```

Il comando evidenziato in giallo visualizza il contenuto delle 50 variabili della matrice NumeroPari() nella **Finestra di Controllo immediato**[54]:

[54] Se la **Finestra di controllo immediato** non è visibile, fare un *clic* con il mouse sul menu **Debug / Finestre / Controllo immediato**.



86: I cicli retti da For… Next.

I cicli retti da **For… Next** (= per… vai al prossimo), di cui abbiamo appena visto due esempi, si basano su un contatore, che è una variabile creata dal programmatore, che aumenta (o diminuisce) a ogni ripetizione del ciclo, sino a raggiungere il limite stabilito dal programmatore.
I comandi scritti tra la riga iniziale del ciclo For… e la riga finale Next vengono ripetuti a ogni passaggio, sino alla conclusione del ciclo.
Quando il contatore ha raggiunto il limite stabilito dal programmatore, il ciclo termina e il programma prosegue per la sua strada.
Nel prossimo esercizio vedremo altri esempi.

Esercizio 47: I cicli For… Next.

Apriamo un nuovo progetto e collochiamo al centro del Form1 un controllo ListBox.
Facciamo un doppio *clic* sul Form1 per accedere alla Finestra del Codice.
Qui troviamo già impostata la procedura che gestisce l'evento del caricamento del Form in memoria, all'avvio del programma:

```
Public Class Form1
```

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

End Class
```

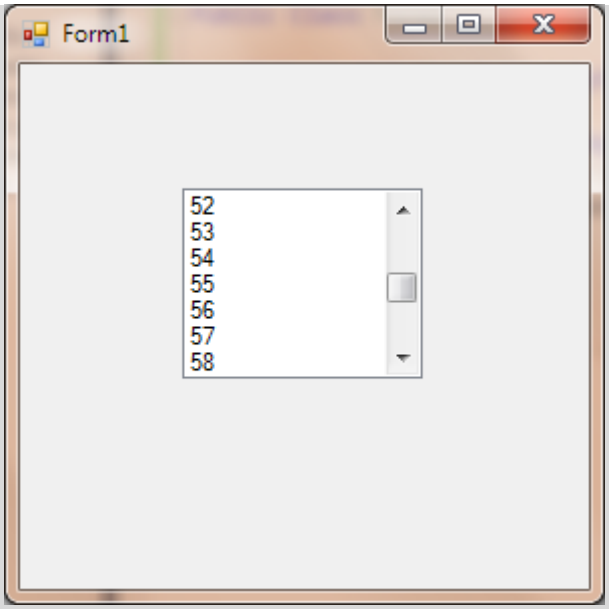
Nello spazio intermedio inseriamo un ciclo di comandi ripetuti, con l'unico comando che qui è evidenziato in giallo:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Contatore As Integer = 1 To 100
            ListBox1.Items.Add(Contatore)
        Next

    End Sub

End Class
```

Notiamo nelle righe del ciclo For… Next:

- La dichiarazione della variabile `Contatore`, direttamente all'interno del ciclo.
- Il ciclo di comandi ripetuti inizia con la parola For…: si tratta di un ciclo che verrà ripetuto 100 volte, aumentando ogni volta la variabile `Contatore` di una unità, da 1 a 100.
- A ogni passaggio, viene aggiunto agli item del controllo ListBox1 un nuovo item, con il valore corrispondente alla variabile `Contatore`: dunque vengono aggiunti al ListBox1 i numeri da 1 a 100.
- La riga Next (= prossima) conclude il ciclo e lo rimanda letteralmente *"alla prossima volta"*, cioè all' inizio, alla riga For…, sino a quando la variabile `Contatore` sarà uguale a 100.

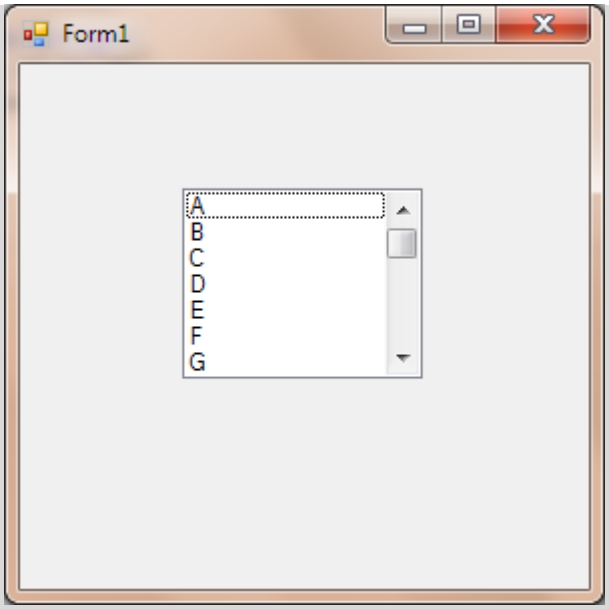
In questa immagine vediamo il programma in esecuzione:



Vediamo ora una variante di questo esercizio.
Al posto dei numeri da 1 a 100, vogliamo visualizzare nel controllo ListBox1 le lettere dell'alfabeto maiuscole. Sapendo che nell'elenco dei simboli presenti sulla tastiera le lettere maiuscole hanno un codice che va 65 a 90, è possibile scrivere un ciclo di comandi ripetuti in questo modo:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Contatore As Integer = 65 To 90
            ListBox1.Items.Add(Chr(Contatore))
        Next

    End Sub

End Class
```

Il comando evidenziato in giallo, ripetuto 26 volte, dal numero 65 al numero 90, aggiunge (**Add**) ogni volta nel ListBox1 il carattere (**Chr**) il cui codice corrisponde alla variabile Contatore.
In questa immagine vediamo il programma in esecuzione:



Altra variante: il codice seguente visualizza nel ListBox1 i numeri da 1 a 20 elevati al quadrato.

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Contatore As Integer = 1 To 20
            ListBox1.Items.Add(Contatore * Contatore)
        Next

    End Sub

End Class
```

Il parametro Step (= passo).

Negli esempi che abbiamo visto in precedenza, la variabile utilizzata come contatore in un ciclo di For… Next aumenta di una unità a ogni passaggio.
Un ciclo For… Next normale ha dunque un passo uguale a 1 (**Step 1**), che nella sintassi di VB non è necessario esplicitare.
Queste due istruzioni sono dunque equivalenti e la precisazione Step 1 nella seconda istruzione è superflua:

```
        For Contatore As Integer = 1 To 100
        For Contatore As Integer = 1 To 100 Step 1
```

E' possibile cambiare questa impostazione di base (Step 1) assegnando al ciclo For… Next un parametro Step maggiore di 1.

Nel codice seguente, ad esempio, la variabile Contatore aumenta di 2 unità a ogni passaggio e viene utilizzata per aggiungere a un controllo ListBox i numeri pari da 2 a 100:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Contatore As Integer = 2 To 100 Step 2
            ListBox1.Items.Add(Contatore)
        Next

    End Sub

End Class
```

Nel codice seguente la variabile Contatore aumenta di 10 unità a ogni passaggio e viene utilizzata per aggiungere a un controllo ListBox i numeri multipli di 10, da 10 a 1000:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Contatore As Integer = 10 To 1000 Step 10
            ListBox1.Items.Add(Contatore)
        Next

    End Sub

End Class
```

Il parametro **Step** può essere utilizzato anche in modo decrescente assegnandogli un numero **negativo**: in questo caso, a ogni passaggio del ciclo For… Next, la variabile usata come contatore diminuisce della quantità indicata dal parametro Step.
Utilizzando il parametro **Step -1**, il codice seguente visualizza in un controllo ListBox le lettere dell'alfabeto maiuscole, in ordine inverso, dalla Z alla A:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        For Contatore As Integer = 90 To 65 Step - 1

            ListBox1.Items.Add(Chr(Contatore))
        Next

    End Sub

End Class
```

Il comando Continue For.

Può accadere, in un ciclo di comandi ripetuti retto da For… Next, che il programmatore per motivi diversi desideri che il programma *salti un giro*, cioè passi al prossimo For… senza eseguire i comandi che si trovano nel ciclo For… Next.
Un esempio concreto: questo ciclo For… Next, che abbiamo visto in un esercizio precedente, inserisce in un controllo ListBox le lettere dell'alfabeto maiuscole:

```
For Contatore As Integer = 65 To 90
    ListBox1.Items.Add(Chr(Contatore))
Next
```

Le lettere inserite con questo codice, però, sono le 26 lettere dell'alfabeto inglese, i cui codici vanno da 65 a 90. All'interno di questo gruppo vi sono cinque lettere che non fanno parte dell'alfabeto italiano, e precisamente:

- la lettera J (codice 74);
- la lettera K (codice 75);
- la lettera W (codice 87);
- la lettera X (codice 88);
- la lettera Y (codice 89).

Se il programmatore desidera che nel controllo ListBox vengano visualizzate solo le lettere maiuscole dell'alfabeto italiano, può inserire nel ciclo **For… Next** cinque comandi **Continue For**, in modo che il programma, quando giunge ai *giri* 74, 75, 87, 88 e 89, passi al *giro* successivo senza eseguire il comando seguente, cioè senza inserire la lettera nel ListBox:

```
For Contatore As Integer = 65 To 90

    If Contatore = 74 Then Continue For
    If Contatore = 75 Then Continue For
    If Contatore = 87 Then Continue For
    If Contatore = 88 Then Continue For
    If Contatore = 89 Then Continue For

    ListBox1.Items.Add(Chr(Contatore))
Next
```

Ecco un'immagine di questo programma in esecuzione, senza le lettere dell'alfabeto inglese:

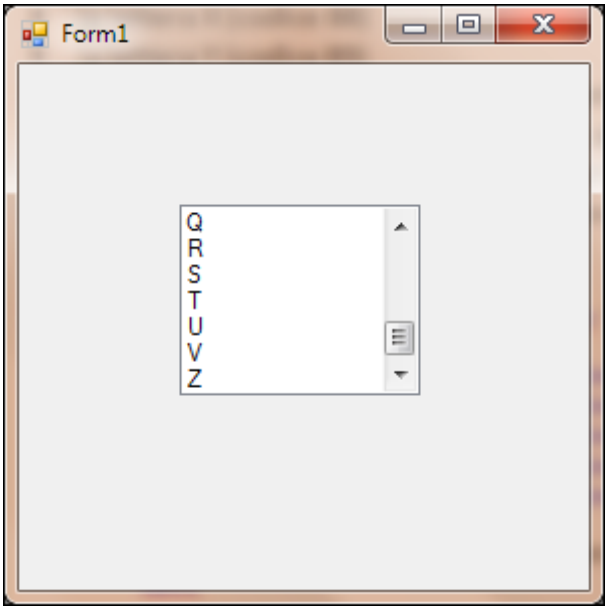


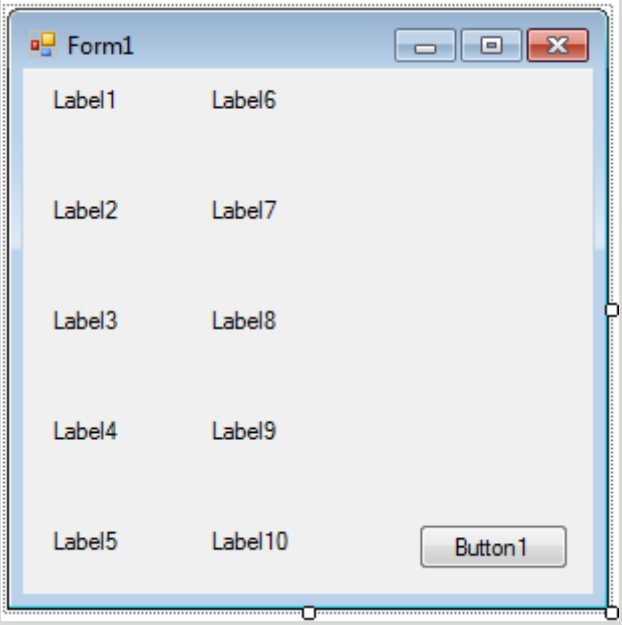
Figura 135: Un controllo ListBox con le lettere maiuscole dell'alfabeto italiano.

87: I cicli retti da For Each… Next.

I cicli For Each… Next (= per ogni… passa al prossimo) sono una versione particolare dei cicli For… Next; si distinguono da questi perché operano **senza utilizzare una variabile-contatore**.
In un ciclo For Each… Next, dunque, manca l'indicazione del numero delle ripetizioni del ciclo di comandi; è invece presente l'indicazione di un gruppo elementi o oggetti che sono coinvolti nel ciclo di comandi: il programma conta questi elementi e ripete il ciclo di comandi tante volte quanti sono gli elementi.
L'uso di un ciclo di comandi retto da For Each… Next è necessario, dunque, rispetto a un ciclo For… Next, quando non si conosce il numero degli elementi o degli oggetti coinvolti nel ciclo, oppure semplicemente quando non si vuole stare a contarli.
Vediamo un esempio nell'esercizio seguente.

Esercizio 48: Un ciclo For Each… Next con un gruppo di controlli.

Apriamo un nuovo progetto di VB e sistemiamo nel Form1 dieci controlli Label e un controllo Button1.



Funzionamento del programma: quando l'utente farà un *clic* sul pulsante Button1, lo sfondo delle dieci Label diventerà di colore giallo.
Facciamo un doppio *clic* sul pulsante Button1 e accediamo alla Finestra del Codice, dove troviamo già impostata la procedura che gestisce l'evento del *clic* del mouse sul Button1:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    End Sub
```

All'interno di questa procedura possiamo scrivere questa serie di istruzioni:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Label1.BackColor = Color.Yellow
        Label2.BackColor = Color.Yellow
        Label3.BackColor = Color.Yellow
        Label4.BackColor = Color.Yellow
        Label5.BackColor = Color.Yellow
        Label6.BackColor = Color.Yellow
        Label7.BackColor = Color.Yellow
        Label8.BackColor = Color.Yellow
        Label9.BackColor = Color.Yellow
        Label10.BackColor = Color.Yellow

    End Sub
```

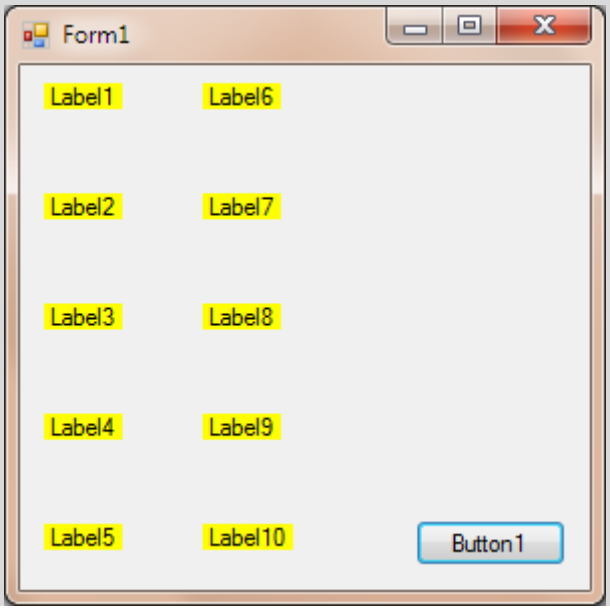
Oppure possiamo considerare le dieci Label come un gruppo di controlli e modificarne la proprietà BackColor con un ciclo **For Each… Next**.
In questo caso, i comandi da trasmettere al programma possono essere descritti in questo modo:

- **PER OGNI** controllo presente nel Form1
- **SE** il tipo di questo controllo è una Label, **ALLORA**
- modifica il colore dello sfondo di questo controllo;
- poi passa al prossimo controllo.

Di seguito vediamo la traduzione di questi comandi nella sintassi di VB.
Notiamo che all'interno del ciclo For Each… Next è presente un procedimento condizionato (If… Then, Se… Allora) che vedremo dettagliatamente più avanti.

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        For Each Control In Me.Controls
            If TypeOf Control Is Label Then
                Control.BackColor = Color.Yellow
            End If
        Next

    End Sub

End Class
```

Nella figura seguente vediamo il programma in esecuzione:

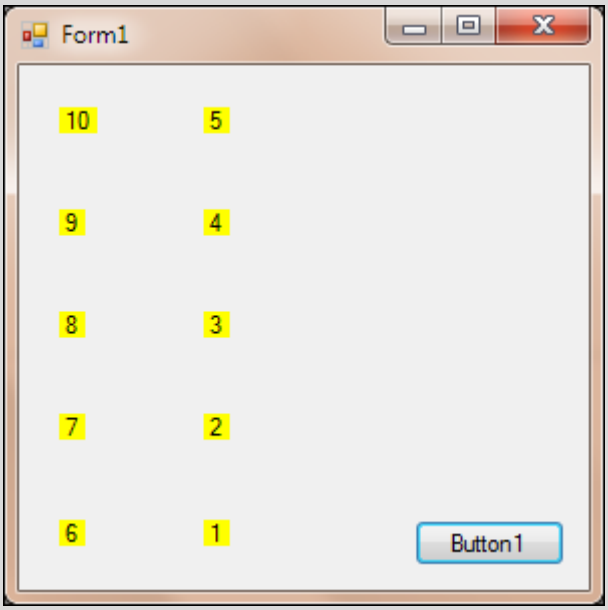


Per avere una numerazione progressiva ordinata di queste Label, possiamo inserire nel ciclo For Each… Next una variabile **Contatore**, che aumenta di una unità a ogni passaggio:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

```
        Dim Contatore As Integer = 0

        For Each Control In Me.Controls
            If TypeOf Control Is Label Then
                Contatore = Contatore + 1
                Control.BackColor = Color.Yellow
                Control.Text = Contatore
            End If
        Next

    End Sub

End Class
```

Ecco come si presenta ora il programma in esecuzione, dopo il *clic* sul Button1:



La numerazione inversa delle dieci Label (la Label1 ha il numero 10, la Label10 ha il numero 1) è dovuta al fatto che il programma, quando esegue un ciclo For Each… Next, parte dal controllo inserito per ultimo nel Form.

Per avere una numerazione progressiva ordinata da 1 a 10 nelle dieci label, dunque, possiamo impostare il valore iniziale della variabile **Contatore = 10** e diminuirlo di una unità a ogni passaggio del ciclo For Each… Next:

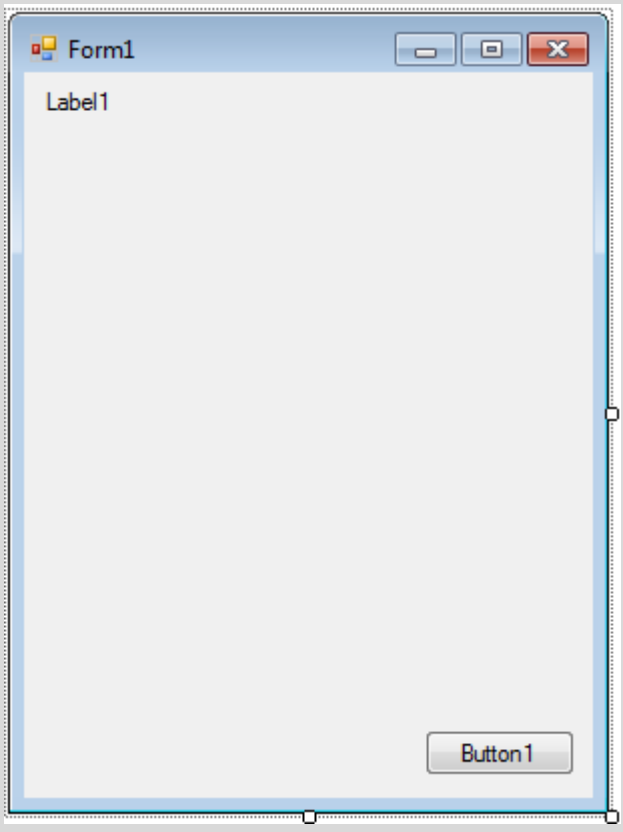
```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim Contatore As Integer = 10

        For Each Control In Me.Controls
            If TypeOf Control Is Label Then
                Control.BackColor = Color.Yellow
```

```
                Control.Text = Contatore
                Contatore = Contatore - 1
            End If
        Next

    End Sub

End Class
```

Nel prossimo esercizio vedremo un altro esempio di uso del ciclo di comandi ripetuti retto da For Each… Next. In questo caso il ciclo opera con le lettere che compongono una stringa di testo.
La stringa di testo in questione è questa: "ABCDEFGHILMNOPQRSTUVZ". Il ciclo For Each… Next separa nella stringa i singoli caratteri (Char) e li trascrive uno alla volta in una Label.

Esercizio 49: Un ciclo For Each… Next con una matrice di variabili.

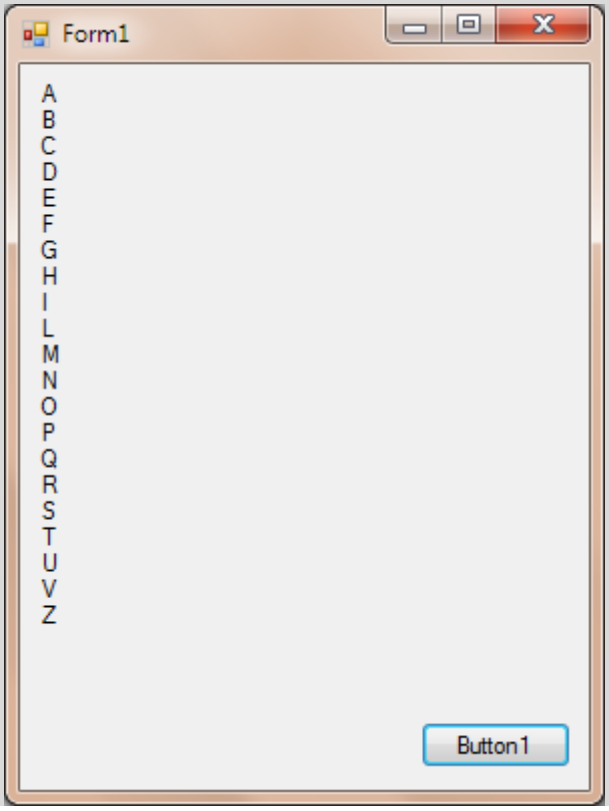
Apriamo un nuovo progetto.
Impostiamo la proprietà **Size** del Form1 **= 300; 400**, poi collochiamo all'interno del Form una Label e un pulsante Button1 come in questa immagine:



Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea la variabila AlfabetoItaliano con le lettere dell'alfabeto italiano:
    Dim AlfabetoItaliano As String = "ABCDEFGHILMNOPQRSTUVZ"

    Private Sub Button1_Click() Handles Button1.Click

        ' Ripulisce il contenuto del controllo Label1
        Label1.Text = ""

        ' Inizio il ciclo di comandi ripetuti usando la variabile Lettera, di
tipo Char (carattere):
        ' per ogni Lettera presente nells stringa AlfabetoItaliano...

        For Each Lettera As Char In AlfabetoItaliano
            ' ... scrivi questa lettera nella Label1 e vai a capo:

            Label1.Text &= Lettera & vbCrLf
            ' poi passa alla prossima lettera:
        Next

    End Sub

End Class
```

Nell'immagine seguente vediamo il programma in esecuzione.



88: I cicli retti a Do While… Loop.

I cicli retti da Do While… Loop (= esegui mentre… ripeti) contengono comandi che il programma continua a ripetere sino a quando sussiste una determinata condizione.
In questi cicli, dunque, il numero delle ripetizioni dei comandi non è dato da una variabile che funziona come contenitore (come nei cicli For… Next) e non è dato dal numero di elementi (come nei cicli For Each… Next); essi continuano a ripetere il ciclo di comandi per un numero indeterminato di volte, sino a quando permane la condizione descritta all'avvio del ciclo.
Ecco due esempi discorsivi di cicli di comandi retti da **Do While… Loop,** utile per comprenderne il funzionamento:

Su un tavolo ci sono 10 cornici vuote.

- Esegui i comandi che seguono (**Do**) mentre (**While**) sul tavolo rimangono cornici vuote;
- prendi una fotografia e mettila in una cornice;
- diminuisci il numero delle cornici vuote di una unità;
- se sul tavolo continuano ad esserci una o più cornici vuote, ripeti il ciclo (**Loop**) da capo.

In un Form abbiamo 10 controlli PictureBox.

- Esegui i comandi che seguono (**Do**) mentre (**While**) nel Form rimangono controlli PictureBox senza immagini;
- prendi un'immagine e inseriscila in un controllo PictureBox;
- diminuisci il numero dei controlli PictureBox disponibili di una unità;
- se nel Form continuano ad esserci dei controlli PictureBox vuoti, ripeti il ciclo (**Loop**) da capo.

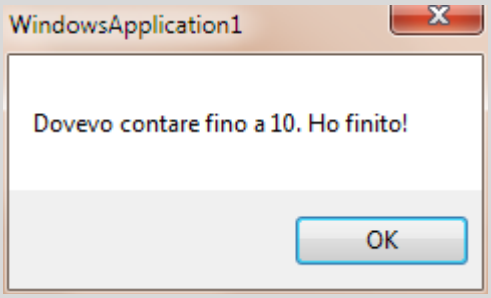
Esercizio 50: Un ciclo di comandi retto da Do While… Loop.

Questo programma visualizza in un messaggio un numero progressivo, e continua a ripetere questa azione per tutto il tempo in cui sussiste questa condizione: il numero visualizzato deve essere inferiore a 11.
Apriamo un nuovo progetto di VB e inseriamo nel Form1 un pulsante Button1.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea la variabile NumeroProgressivo destinata a contenere numeri
interi, assegnandole il valore 1:
        Dim NumeroProgressivo As Integer = 1
```

```
        ' Inizia il ciclo di comandi ripetuti retto da Do While:
        ' esegui questi comandi mentre la variabile NumeroProgressivo è
inferiore a 11:

        Do While (NumeroProgressivo < 11)

            ' visualizza un messaggio con il valore della variabile
NumeroProgressivo:
            MsgBox("Devo contare fino a 10. Sono arrivato a " &
NumeroProgressivo)

            ' aumenta la variabile NumeroProgressivo di una unità:
            NumeroProgressivo += 1

            ' Se la variabile NumeroProgressivo è inferiore a 11, ripeti il
ciclo da capo:
        Loop

        ' quando il ciclo è finito, visualizza questo messaggio:
        MsgBox("Dovevo contare fino a 10. Ho finito!")

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione, alla fine del ciclo Do… While:



Il ciclo Do While… Loop presente in questo listato può essere letto in questi termini:

- Esegui le righe che seguono sino a quando la variabile **NumeroProgressivo** rimane inferiore a 11;
- visualizza un box con il messaggio: "Devo contare fino a 10. Sono arrivato a " e aggiungi la cifra corrispondente alla variabile **NumeroProgressivo**;
- aumenta la variabile **NumeroProgressivo** di una unità;
- se la variabile **NumeroProgressivo** è inferiore a 11, ripeti il ciclo da capo.
- Quando il ciclo è finito visualizza un box con il messaggio: "Dovevo contare fino a 10. Ho finito!".

89: I cicli retti da Do Until… Loop.

I cicli di comandi retti da Do Until… Loop (= esegui fino a quando… ripeti) sono una variante dei cicli Do While… Loop, dai quali si differenziano per queste caratteristiche:

- un ciclo Do While… Loop si ripete **MENTRE** esiste una determinata condizione;
- un ciclo Do Until… Loop si ripete **FINCHE'** è raggiunta una determinata condizione.

Per vederne un esempio concreto, ripetiamo l'esercizio precedente utilizzando il ciclo Do Until… Loop.

Esercizio 51: Un ciclo di comandi retto da Do Until… Loop.

Creiamo un programma semplicissimo, per vedere il funzionamento dei cicli Do Until… Loop. Il programma visualizza in un messaggio un numero progressivo, e continua a ripetere il comando finché il numero arriva a 10.
Apriamo un nuovo progetto di VB e inseriamo nel Form1 un pulsante Button1.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' crea la variabile NumeroProgressivo destinata a contenere numeri interi
e le assegna il valore 1:
        Dim NumeroProgressivo As Integer = 1

        ' inizia il ciclo di comandi retto da Do Until:
        Do Until (NumeroProgressivo = 10)

            ' esegui questi comandi sino a quando la variabile NumeroProgressivo
è uguale a 10:
            ' visualizza un messaggio con il valore della variabile
NumeroProgressivo:
            MsgBox("Devo contare fino a 10. Sono arrivato a " &
NumeroProgressivo)

            ' aumenta la variabile NumeroProgressivo di una unità:
            NumeroProgressivo += 1

            ' ripeti il ciclo:
        Loop

        ' quando il ciclo è finito, visualizza questo messaggio:
        MsgBox("Dovevo contare fino a 10. Ho finito!")

    End Sub

End Class
```

Il programma si basa su un ciclo Do Until… Loop che può essere letto in questi termini:

- Esegui le righe che seguono sino a quando la variabile **NumeroProgressivo** sarà uguale a 10;

- visualizza un box con il messaggio: “Devo contare fino a 10. Sono arrivato a “; aggiungi la cifra corrispondente alla variabile **NumeroProgressivo**;
- aumenta la variabile **NumeroProgressivo** di una unità;
- torna a ripetere il ciclo.
- Quando il ciclo è finito visualizza un box con il messaggio: “Dovevo contare fino a 10. Ho finito!”.

Nel prossimo esercizio vedremo un altro esempio di utilizzo di un ciclo di comandi retto da Do Until… Loop, utilizzato per fermare temporaneamente l’esecuzione di un programma.
Le procedure di stop temporaneo sono a volte necessarie per fermare lo svolgimento di un programma per alcuni secondi: il tempo necessario per effettuare determinate operazioni, per completare un effetto grafico, per ascoltare un suono…
Nell’esercizio seguente vedremo la creazione di una procedura di questo tipo.

Esercizio 52: Creazione di una pausa in un programma.

Apriamo un nuovo progetto di VB e inseriamo nel Form1 un pulsante Button1.
Copiamo e incolliamo nella Finestra del Codice il listato seguente, poi mandiamo in esecuzione il programma.
Con un *clic* del mouse sul Button1 si sente un bip, poi il cursore cambia forma per tre secondi.
Terminata la pausa, si sente un altro bip e il cursore torna alla forma normale.
La gestione della pausa funziona in questo modo:
Il programma registra in una variabile OrarioFinePausa il momento in cui la pausa dovrà finire, poi confronta in continuazione la data corrente con l’OrarioFinePausa.
Quando la data corrente supera il valore di OrarioFinePausa, la pausa termina.

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        'Emette un bip e cambia il cursore del Form1:
        Beep()
        Me.Cursor = Cursors.WaitCursor

        ' Il comando seguente imposta una pausa del programma della durata di 3
secondi (= 3000 millisecondi).
        ' Nella variabile OrarioFinePausa viene memorizzato l'orario fittizio
dato dall'ora corrente (Date.Now) + i millisecondi della pausa:

        Dim OrarioFinePausa As Date = Date.Now.AddMilliseconds(3000)

        ' Il ciclo seguente si ripete per tutto il tempo in cui il valore della
data corrente (Date.Now) è inferiore al valore della variabile OrarioFinePausa.
        ' Quando il valore di Date.Now è superiore al valore di OrarioFinePausa,
il ciclo termina e il programma prosegue per la sua strada.
```

```
        Do Until Date.Now > OrarioFinePausa

        ' Il ciclo non contiene alcun comando:
        ' esso ruota semplicemente su se stesso sino a quando la data
corrente è maggiore della variabile OrarioFinePausa.

        Application.DoEvents()
        Loop

        ' Terminata l'esecuzione della pausa, il programma emette un bip e
ripristina il cursore del Form1:
        Beep()
        Me.Cursor = Cursors.Default

    End Sub

End Class
```

L'esigenza di creare una pausa nel corso del funzionamento di un programma si presenta spesso, per esigenze diverse e con modalità diverse.
Ora fermiamo l'esecuzione del programma che abbiamo creato in questo esercizio, torniamo alla Finestra Progettazione e inseriamo nel Form1 altri due pulsanti Button: Button2 e Button3.
Supponiamo che un *clic* su questi pulsanti attivi una pausa del programma di durata diversa:

- 1000 millisecondi per il Button1;
- 2000 millisecondi per il Button2;
- 3000 millisecondi per il Button3.

Per ottenere questo risultato possiamo ovviamente scrivere queste tre procedure:

```
    Private Sub Button1_Click() Handles Button1.Click
        Dim OrarioFinePausa As Date = Date.Now.AddMilliseconds(1000)
        Do Until Date.Now > OrarioFinePausa
        Loop
    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click
        Dim OrarioFinePausa As Date = Date.Now.AddMilliseconds(2000)
        Do Until Date.Now > OrarioFinePausa
        Loop
    End Sub
```

```
    Private Sub Button3_Click() Handles Button3.Click
        Dim OrarioFinePausa As Date = Date.Now.AddMilliseconds(3000)
        Do Until Date.Now > OrarioFinePausa
        Loop
    End Sub
```

In questi casi però è più utile creare una procedura a parte, finalizzata a gestire solo la gestione della pausa del programma. Ognuno dei tre pulsanti chiamerà in causa questa procedura, passandole la propria durata della pausa:

```
Public Class Form1
```

```
    Private Sub Button1_Click() Handles Button1.Click

        Beep()
        Me.Cursor = Cursors.WaitCursor

        ' Chiama in causa la procedura che gestisce la pausa del porgramma,
passandole il valore di 1000 millisecondi:
        Call Pausa(1000)

        ' Terminata la procedura di pausa, il programma riprende da questo punto:
        Beep()
        Me.Cursor = Cursors.Default

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        Beep()
        Me.Cursor = Cursors.WaitCursor

        ' Chiama in causa la procedura che gestisce la pausa del porgramma,
passandole il valore di 1000 millisecondi:
        Call Pausa(2000)

        ' Terminata la procedura di pausa, il programma riprende da questo punto:
        Beep()
        Me.Cursor = Cursors.Default

    End Sub
```

```
    Private Sub Button3_Click() Handles Button3.Click

        Beep()
        Me.Cursor = Cursors.WaitCursor

        ' Chiama in causa la procedura che gestisce la pausa del porgramma,
passandole il valore di 1000 millisecondi:
        Call Pausa(3000)

        ' Terminata la procedura di pausa, il programma riprende da questo punto:
        Beep()
        Me.Cursor = Cursors.Default

    End Sub
```

```
    Private Sub Pausa(DurataPausa As Integer)
        ' Questa procedura è chiamata in causa dai tre pulsanti:
        ' ognuno dei tre pulsanti passa a questa procedura la sua durata della
pausa
        ' espressa in millisecondi:

        Dim OrarioFinePausa As Date = Date.Now.AddMilliseconds(DurataPausa)
        Do Until Date.Now > OrarioFinePausa

         ' Il ciclo non contiene alcun comando:
         ' esso ruota semplicemente su se stesso sino a quando la data
corrente è maggiore della variabile OrarioFinePausa.

         Application.DoEvents()
```

```
        Loop

    End Sub

End Class
```

90: L'uscita anticipata da cicli o procedure.

Durante l'esecuzione di un programma si può presentare la necessità di uscire da un ciclo di comandi ripetuti prima che questo si concluda normalmente, perché le condizioni richieste dal programmatore si sono verificate in anticipo rispetto al previsto.
Il comando di uscita da un ciclo è **Exit**:

- per uscire da un ciclo retto da For… Next: **Exit For**
- per uscire da un ciclo retto da Do While… Loop: **Exit Do**
- per uscire da un ciclo retto da Do Until… Loop: **Exit Do**

In VB è disponibile anche un comando di uscita anticipata da una procedura: si tratta del comando **Exit Sub**, che utilizzeremo concretamente in seguito in alcuni esempi ed esercizi. Per ora ci basti sapere che, nell'esecuzione delle istruzioni contenute in una procedura, quando il programma giunge alla riga Exit Sub considera la procedura conclusa e ignora le righe di codice successive eventualmente presenti.

91: Sintassi.

Nei cicli di comandi retti da For… Next, da Do While… Loop o da Do Until… Loop il comando conclusivo del ciclo (Next o Loop) è ovviamente obbligatorio e la sua mancanza provoca un errore di sintassi.
E' difficile incorrere in questo errore perché quando si avvia la scrittura di un ciclo di comandi, l'*editor* di VB si preoccupa di chiudere il ciclo inserendo automaticamente la riga conclusiva con il termine Next o Loop.
Vediamo un esempio.
Nella finestra del codice di un programma, avviamo questo ciclo di comandi:

```
        For Contatore As Integer = 1 To 10
```

Notiamo che alla pressione del pulsante INVIO l'*editor* di VB aggiunge automaticamente la riga di chiusura del ciclo:

```
        For Contatore As Integer = 1 To 10
```

```
        Next
```

Lo stesso avviene per i cicli Do While… Loop e Do Until… Loop, che l'*editor* di VB chiude automaticamente la parola Loop:

```
        Dim Contatore As Integer = 1
        Do While Contatore < 11

        Loop
```

92: I cicli creati con il comando GoTo.

Il comando GoTo (= vai a…) si usa all'interno di una procedura per interrompere l'esecuzione della sequenza normale di comandi e rinviare il programma ad eseguire i comandi presenti in un'altra parte della stessa procedura.
Quando il rinvio punta a una parte precedente della procedura, rimanda cioè ad eseguire di nuovo alcuni comandi già eseguiti, si ha un ciclo di comandi ripetuti; questo ciclo continua sino a quando sussiste la condizione che ha fatto scattare il comando GoTo.
Vediamo un esempio descrittivo: un programma che riproduce il gioco della Tombola ha una riga di codice che comanda l'estrazione di un nuovo numero.
Subito sotto questa riga c'è l'operazione di controllo: se il numero estratto è già uscito, il programma deve tornare all'estrazione di un nuovo numero.
In termini più tecnici, l'esempio può essere tradotto così:

EstraiNuovoNumero:
- sorteggia un NuovoNumero
- se (**IF**) il NuovoNumero è già uscito, allora (**THEN**)
- vai (**GOTO**) alla riga **EstraiNuovoNumero**, altrimenti (**ELSE**)
- procedi con l'esecuzione del programma.

E' una tecnica che vedremo applicata concretamente in seguito, in alcuni esempi ed esercizi.

Capitolo 15: OPERAZIONI ARITMETICHE.

I dati contenuti nelle variabili possono essere utilizzati da VB per eseguire operazioni diverse, anche molto complesse:

- *operazioni matematiche;*
- *operazioni di comparazione;*
- *operazioni logiche.*

Questo capitolo e i due capitoli successivi sono dedicati a queste operazioni che stanno alla base della capacità del computer di prendere decisioni.

93: Gli operatori aritmetici.

VB esegue le operazioni aritmetiche con i dati delle variabili usando gli operatori aritmetici standard, riportati nella tabella seguente.

Simbolo	Operazione	Esempio	Risultato
+	addizione	12 + 7	19
-	sottrazione	12 – 5	7
*****	moltiplicazione	12 * 5	60
/	divisione decimale	100 / 8	12,5
****	divisione intera	100 \ 8	12
Mod	calcolo del resto della divisione	100 Mod 8	4
^	elevamento a potenza	4 ^ 2	16

Tabella 10: Gli operatori aritmetici.

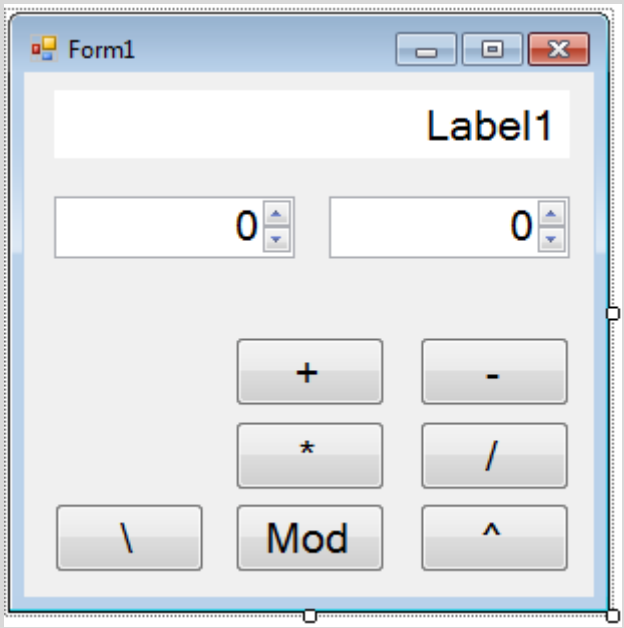
Nel prossimo esercizio gli operatori aritmetici di base sono utilizzati per creare una semplicissima calcolatrice.

Esercizio 53: Gli operatori aritmetici.

Apriamo un nuovo progetto e collochiamo sul **Form1**:

- un controllo **Label**
- due controlli NumericUpDown e
- sette controlli **Button**

come nell'immagine seguente:



Tutti questi controlli hanno la proprietà **Font = Microsoft Sans Serif a 16 punti**.
Il controllo **Label1** ha la proprietà **Autosize = False**, la proprietà **BackColor = White** e la proprietà **TextAlign = MiddleRight**.
I due controlli **NumericUpDown** hanno la proprietà **TextAlign = Right**.
I sette controlli **Button** hanno la proprietà **Text** impostata con i simboli di sette operatori aritmetici:

- **Button1: +**
- **Button2: -**
- **Button3: ***
- **Button4: /**
- **Button5: **
- **Button6: Mod**
- **Button7: ^**

Il programma funziona in questo modo: quando l'utente fa un *clic* con il mouse su uno dei pulsanti, il programma esegue l'operazione corrispondente, utilizzando i numeri presenti nei due controlli **NumericUpDown**; il risultato dell'operazione è visualizzato nella Label1.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
    Dim A As Integer
    Dim B As Integer
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' questo pulsante effettua l'addizione
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        Label1.Text = (A + B).ToString
    End Sub
```

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        ' questo pulsante effettua la sottrazione
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        Label1.Text = (A - B).ToString
    End Sub
```

```
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        ' questo pulsante effettua la moltiplicazione
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        Label1.Text = (A * B).ToString
    End Sub
```

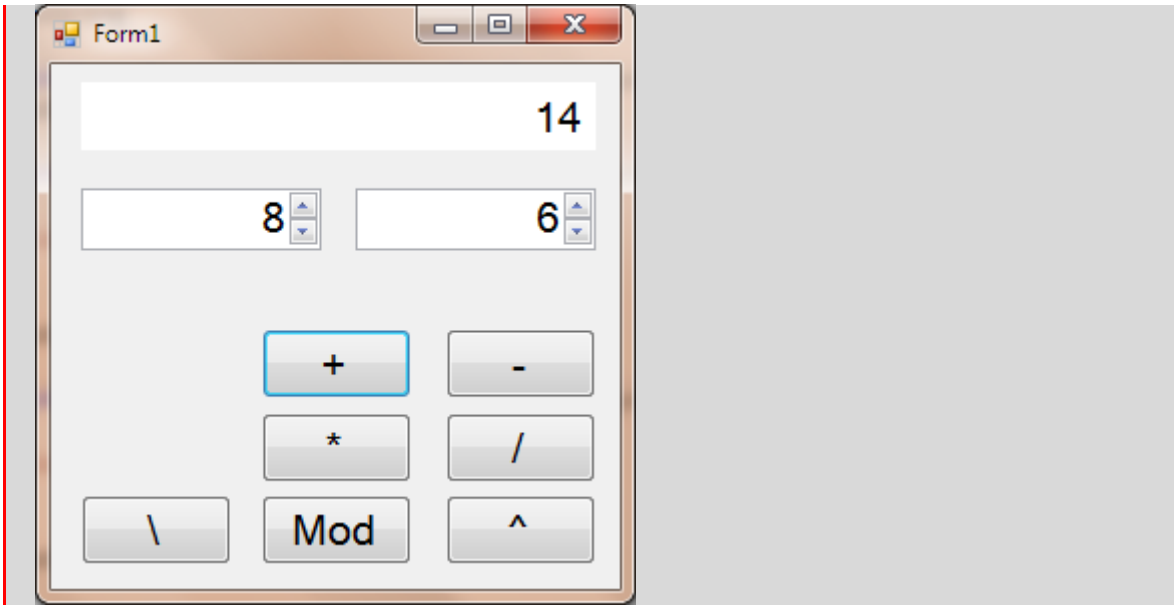
```
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        ' questo pulsante effettua la divisione
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        ' in caso di divisione per zero, il programma visualizza un messaggio di
errore ed esce da questa procedura:
        If B = 0 Then
            MsgBox("Errore: divisione per zero")
            Exit Sub
        End If
        Label1.Text = (A / B).ToString
    End Sub
```

```
    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        ' questo pulsante effettua la divisione, riportando solo la parte intera
del quoto
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        ' in caso di divisione per zero, il programma visualizza un messaggio di
errore ed esce da questa procedura:
        If B = 0 Then
            MsgBox("Errore: divisione per zero")
            Exit Sub
        End If
        Label1.Text = (A \ B).ToString
    End Sub
```

```
    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
        ' questo pulsante effettua la divisione, riportando solo il resto
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        ' in caso di divisione per zero, il programma visualizza un messaggio di
errore ed esce da questa procedura:
        If B = 0 Then
            MsgBox("Errore: divisione per zero")
            Exit Sub
        End If
        Label1.Text = (A Mod B).ToString
    End Sub
```

```
    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
        ' questo pulsante effettua l'elevamento a potenza
        A = NumericUpDown1.Value
        B = NumericUpDown2.Value
        Label1.Text = (A ^ B).ToString
    End Sub

End Class
```

Notiamo nel codice la creazione delle due variabili A e B, destinate a contenere i valori presenti nei due controlli NumericUpDown; queste due variabili sono utilizzate in tutte le operazioni successive.
Le procedure che gestiscono i *clic* sui sette pulsanti comandano le diverse operazioni; il risultato di ogni operazione è visualizzato nella Label1.
Per evitare rischi di disfunzioni o errori, prima di essere visualizzato nella Label1 ogni risultato è convertito in testo (`ToString`).
Notiamo che nelle tre procedure che gestiscono le divisioni, nel caso si prospetti una divisione per zero il programma visualizza un avviso all'utente, poi esce in modo anticipato dalla procedura (`Exit Sub`), senza effettuare l'operazione che causerebbe un errore e il blocco del programma stesso.
Ecco un'immagine del programma in esecuzione:



94: Sequenze di operatori aritmetici.

Gli operatori aritmetici possono essere combinati in espressioni numeriche di varia complessità. Nell'esecuzione di queste espressioni VB esegue prima le operazioni di moltiplicazione e di divisione, poi le operazioni di addizione e sottrazione.
Perché abbiano la precedenza rispetto alle altre operazioni, le addizioni e le sottrazioni debbono essere chiuse tra parentesi.
Vediamo alcuni esempi basati sull'uso di quattro variabili A, B, C e D che contengono questi valori:

- A = 5
- B = 10
- C = 4
- D = 2

Espressione con le variabili	**Espressione trascritta con il contenuto delle variabili**	**Ordine di esecuzione delle operazioni**	**Risultato**
A + B * C / D	5 + 10 * 4 / 2	Vengono eseguite prima la moltiplicazione e la divisione. L'addizione viene eseguita al passaggio successivo.	25

(A + B) * C / D	(5 + 10) * 4 / 2	Vengono eseguite prima l'addizione tra parentesi e la divisione. La moltiplicazione viene eseguita al passaggio successivo.	30
A + B * C – D	5 + 10 * 4 - 2	Viene eseguita prima la moltiplicazione. L'addizione e la sottrazione vengono eseguite al passaggio successivo.	43
(A + B) * (C – D)	(5 + 10) * (4 - 2)	Vengono eseguite prima l'addizione e la sottrazione tra le parentesi. La moltiplicazione viene eseguita al passaggio successivo.	30
A + B * C – D / D	5 + 10 * 4 - 2 / 2	Vengono eseguite prima la moltiplicazione e la divisione. L'addizione e la sottrazione vengono eseguite al passaggio successivo.	44
(A + B) * C – D / D	(5 + 10) * 4 - 2 / 2	Vengono eseguite prima l'addizione tra parentesi e la divisione. Al passaggio successivo, viene eseguita la moltiplicazione. La sottrazione viene eseguita alla fine.	59
((A + B * C) – D) / D	((5 + 10 * 4) - 2) / 2	Viene eseguita prima la moltiplicazione. Al passaggio successivo vengono eseguite l'addizione e la sottrazione tra parentesi. La divisione viene eseguita alla fine.	21,5
((A + B) * C – D) / D	((5 + 10) * 4 - 2) / 2	Viene eseguita prima l'addizione. Al passaggio successivo viene eseguita la moltiplicazione. Al passaggio successivo viene eseguita la sottrazione. La divisione viene eseguita alla fine.	29

Tabella 11: Ordine di esecuzione delle operazioni all'interno di espressioni aritmetiche.

Operazioni matematiche più complesse (ad esempio il calcolo della radice quadrata) possono essere effettuate utilizzando le funzioni matematiche di VB, che vedremo più avanti nel manuale[55].

95: Concatenazione di stringhe di testo.

L'operazione di addizione può essere effettuata anche con le variabili di tipo **String**, cioè le variabili che contengono testo, utilizzando il simbolo &.
Supponiamo di avere tre variabili di tipo String denominate A, B e C con questi valori:

```
Dim A As String = "Paolo"
Dim B As String = "Rossi"
Dim C As String
C = A & B
```

Alla variabile C viene assegnato il testo "PaoloRossi"; per ottenere uno spazio tra il nome e il cognome (cioè tra A e B) è necessario aggiungere una stringa con uno spazio vuoto nella concatenazione dei testi:

```
C = A & " " & B
```

L'operatore & consente anche di aggiungere parole o frasi a un testo iniziale, sino a fargli assumere la forma definitiva voluta dal programmatore.
L'esempio seguente mostra la proprietà Text di una Label1 che, partendo dalla prima riga di codice, si accresce di nuove parti di testo sino ad assumere la forma finale che verrà visualizzata nel programma:

```
Label1.Text = "Questo programma è stato realizzato "
Label1.Text &= "da Paolo Rossi,"
Label1.Text &= " Stefania Rossi e"
Label1.Text &= " Giovanni Bianchi."
```

Il testo finale della Label1 sarà dunque questo:

[55] Capitolo 20: FUNZIONI SU NUMERI., a pag. 448.

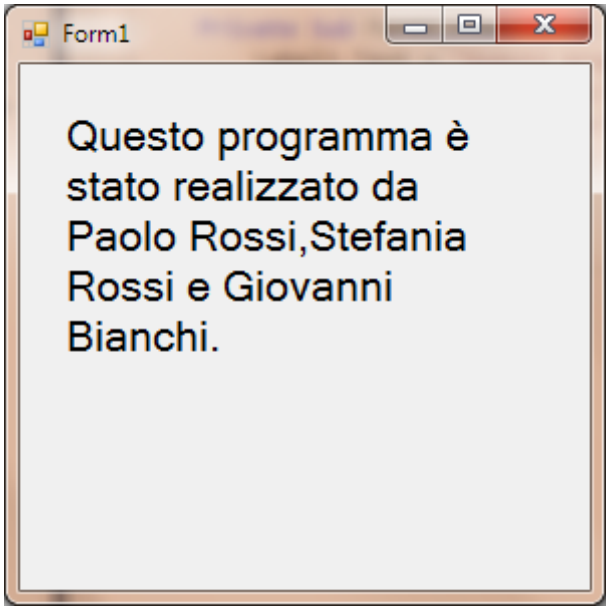


Figura 136: Concatenazione di stringhe di testo.

Notiamo che il testo così concatenato va a capo secondo le dimensioni del controllo Label1. Se il programmatore desidera che il testo vada a capo, ad esempio, dopo ogni nome, è necessario inserire nei punti desiderati la sigla **vbCrLf**[56]:

```
Label1.Text = "Questo programma è stato realizzato " & vbCrLf
Label1.Text &= "da Paolo Rossi," & vbCrLf
Label1.Text &= " Stefania Rossi e" & vbCrLf
Label1.Text &= " Giovanni Bianchi."
```

Ora il testo finale della Label1 si presenterà in questo modo:

[56] Ricordiamo che la sigla **vbCrLf** è l'acronimo di **visual basic Carriage return Line feed** (= ritorno del carrello e inizio di una nuova linea); inserita opportunamente in un testo, essa manda a capo, nella riga successiva, il testo scritto dopo la sigla.

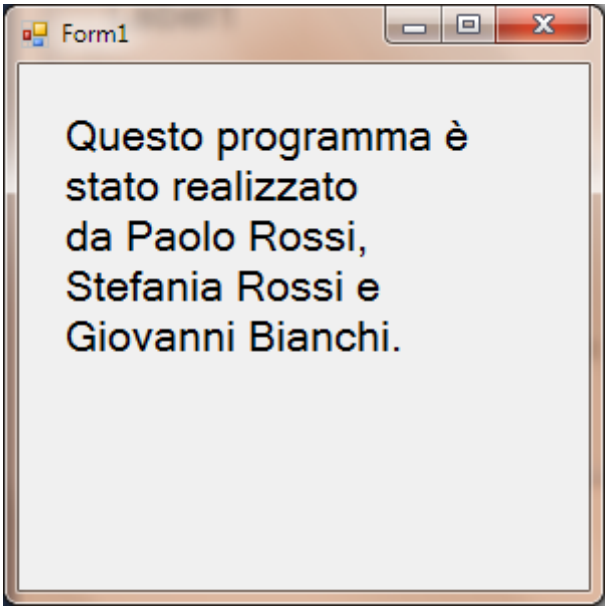


Figura 137: Concatenazione di stringhe di testo con la sigla vbCrlLf.

Lo stesso meccanismo di concatenazione del testo può essere utilizzato per scrivere il contenuto in una variabile di testo che, alla fine delle elaborazioni, può essere visualizzato in un controllo Label1:

```
Dim Info As String
Info = "Per fare partire il programma premi il tasto F1."
Info &= vbCrLf
Info &= "Per fermarlo premi il tasto F2."
Info &= vbCrLf
Info &= "Per uscire dal gioco premi il tasto F12."

Label1.Text = Info
```

96: Come si va a capo nella scrittura del codice.

Nonostante VB consenta facilmente di concatenare testi sino a formare anche testi molto lunghi, si può verificare l'esigenza di scrivere con una istruzione unica un testo anche molto lungo.
Ad esempio:

```
Label1.Text = "Questo programma è stato realizzato " & vbCrLf & "da Paolo
Rossi," & vbCrLf & " Stefania Rossi e" & vbCrLf & " Giovanni Bianchi."
```

In questo caso, per meglio leggere e controllare il codice, il programmatore ha la facoltà di spezzare e mandare a capo l'istruzione utilizzando il trattino di sottolineatura _.
Questo trattino, posto al termine di una riga di codice, indica a VB che l'istruzione, non finita, continua alla riga successiva. Ogni trattino di continuazione deve essere preceduto da uno spazio; la riga che conclude l'istruzione non ha il trattino di continuazione:

```
Label1.Text = "Questo programma è stato realizzato " & vbCrLf & _
    "da Paolo Rossi," & vbCrLf & _
    " Stefania Rossi e" & vbCrLf & _
    " Giovanni Bianchi."
```

Capitolo 16: OPERAZIONI DI COMPARAZIONE.

97: Gli operatori di comparazione.

Un procedimento razionale di decisione si basa su operazioni di comparazione dei dati, che portano a fare una determinata scelta anziché un'altra e a compiere una determinata azione anziché un'altra.
Si trovano spesso, nelle espressioni del linguaggio comune, operazioni comparative di dati o di situazioni:

- "Questa bicicletta è UGUALE a quella che mi hanno rubato".
- "Questo prodotto è MENO caro di quello, ma la qualità è PIÚ scadente".
- "Questo film è DIVERSO dai soliti film girati da questo regista".

A volte una singol'operazione comparativa è sufficiente per orientare una decisione in un senso o nell'altro, come in questi esempi:

- "La distanza tra la mia auto e quella che mi precede È INFERIORE alla distanza di sicurezza", quindi: **rallento**.
- "Il caffè di oggi È MENO dolce di quello che bevo di solito", quindi: **aggiungo un po' di zucchero**.
- "Il signore seduto in terza fila ha un'aria PIU' tranquillizzante di quello seduto in ultima fila", quindi: **mi siedo di fianco al signore in terza fila**.

Le operazioni di comparazione dei dati si effettuano con questi operatori comparativi:

Simbolo	Esempio	Traduzione
=	If X = Y …	Se X è uguale a Y …
>	If X > Y …	Se X è maggiore di Y …
<	If X < Y …	Se X è minore di Y …
>=	If X >= Y …	Se X è maggiore o uguale a Y …
<=	If X <= Y …	Se X è minore o uguale a Y …
<>	If X <> Y …	Se X è diverso da Y …

Tabella 12: Gli operatori di comparazione.

La comparazione di uguaglianza, oltre che con il simbolo =, può essere effettuata anche con il metodo **Equals:**

Equals()	If X.Equals(Y)	Se X è uguale a Y …

Mentre il risultato di un'operazione aritmetica è un numero, il risultato di un'operazione comparativa può avere solo due esiti: può essere **True** (Vero) oppure **False** (Falso). Le operazioni comparative sono dunque collocate dai programmatori, generalmente, dove l'esecuzione del programma si biforca in due direzioni: se una comparazione è vera il programma prende una direzione, se la stessa comparazione è falsa il programma prende l'altra direzione.

In VB fa eccezione a questa regola un metodo di comparazione particolare: si tratta del metodo **CompareTo()** (= confronta con), che mette in relazione due variabili e fornisce **tre** esiti diversi:

- -1, se la prima variabile è minore della seconda,
- 0, se le due variabili sono uguali o equivalenti,
- 1, se la prima variabile è maggiore della seconda.

A seconda dell'esito di questo metodo di comparazione, dunque, il programma avrà la possibilità di scegliere non fra due ma fra tre azioni diverse:

CompareTo()	X.CompareTo(Y) …	Se X è maggiore di Y il risultato è 1. Se X è uguale a Y il risultato è 0. Se X è minore di Y il risultato è -1.

Esempio:

```
Dim A As Integer = 12
Dim B As Integer = 25
Dim X As Integer
X = A.CompareTo(B)
```

Alla fine di queste istruzioni, il valore della variabile X sarà uguale a -1, perché il valore di A è inferiore al valore di B.

Le operazioni logiche effettuate con numeri o con stringhe, di cui vedremo di seguito alcuni esempi, non si trovano mai isolate nei codici dei programmi, ma sono sempre associate a operazioni condizionate di questo tipo:

- Se A > B allora esegui questi comandi …
- Se E <> B allora esegui questi comandi …

E spesso sono combinate all'interno di espressioni logiche di questo tipo:

- Se A > B e B <> C allora esegui questi comandi…
- Se A <> B oppure A = C allora esegui questi comandi …

98: La comparazione di variabili numeriche.

Supponiamo di avere all'interno di un programma queste variabili numeriche con questi valori:

```
Dim A As Integer = 10
Dim B As Integer = 10
Dim C As Integer = 15
Dim D As Integer = 20
```

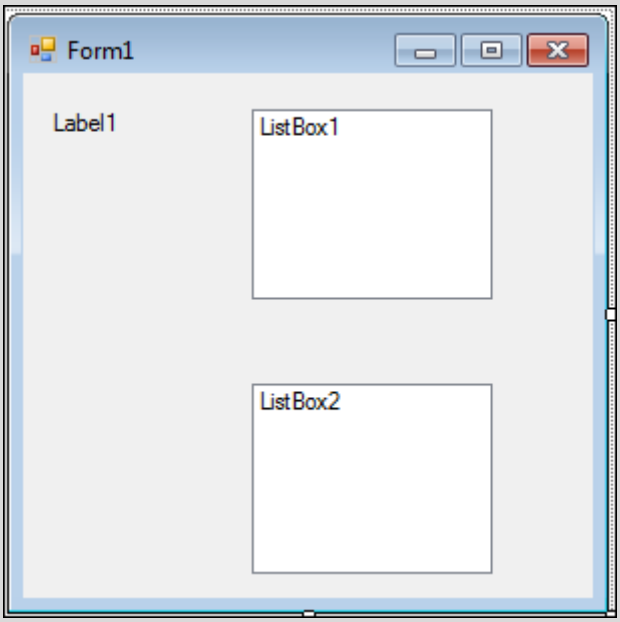
Sulla base di questi valori, vediamo nella colonna di sinistra di questa tabella una serie di operazioni comparative che danno il risultato **True** (cioè sono vere); nella colonna di destra vediamo invece una serie di operazioni comparative che danno il risultato **False** (cioè sono false):

Espressioni vere	**Espressioni false**
A = B	A <> B
A + B = D	A + B > D
A = D – B	A < D - B
C + C = A + D	C + C <> A + D
D > A	D <= A
A < C	A >= C
A <= B	A = B + C
A <= C	A < B
A <> D	A = C
	A = D

Nel prossimo esercizio proveremo l'esito di tutte queste operazioni di comparazione all'interno di un programma.

Esercizio 54: Comparazione di variabili numeriche.

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo **Label1** e due controlli **ListBox** come in questa immagine:



Impostiamo la proprietà **StartPosition** del Form1 **= CenterScreen**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim A As Integer = 10
    Dim B As Integer = 10
    Dim C As Integer = 15
    Dim D As Integer = 20
```

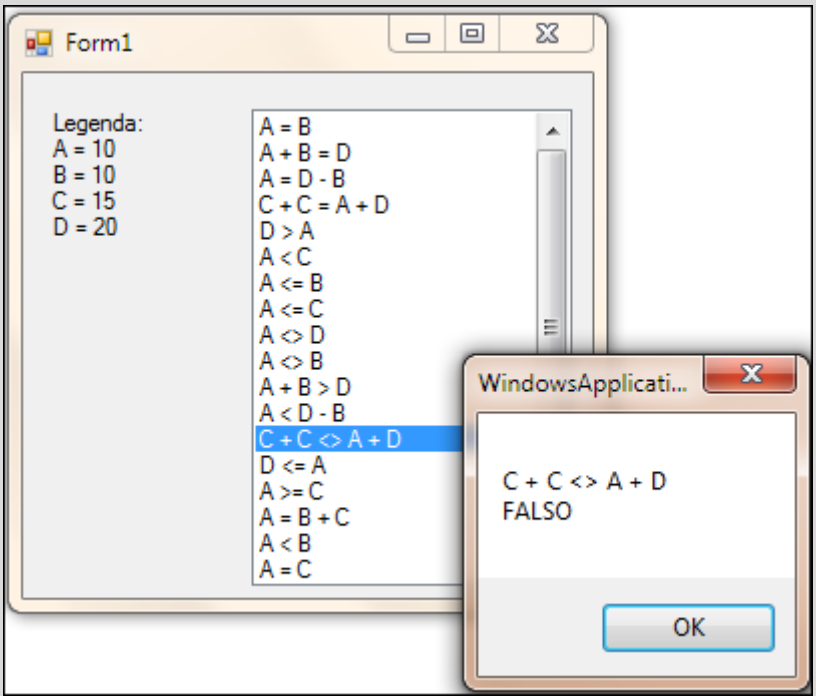
```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' all'avvio del programma, imposta il testo da visualizzare nella Label1:
        With Label1
            .Text = "Legenda:" & vbCrLf
            .Text &= "A = 10" & vbCrLf
            .Text &= "B = 10" & vbCrLf
            .Text &= "C = 15" & vbCrLf
            .Text &= "D = 20"
        End With

        ' imposta le dimensioni del ListBox1:
        ListBox1.Size = New Size(160, 240)

        ' nasconde il ListBox2
        ListBox2.Visible = False

        ' aggiunge gli items nei due ListBox.
        ' Il ListBox1 elencherà i testi delle operazioni di comparazione.
```

```
        ' Il ListBox2 (che rimarrà invisibile) elencherà invece i risultati delle
operazioni di comparazione.

        ListBox1.Items.Add("A = B")
        ListBox2.Items.Add(A = B)

        ListBox1.Items.Add("A + B = D")
        ListBox2.Items.Add(A + B = D)

        ListBox1.Items.Add("A = D - B")
        ListBox2.Items.Add(A = D - B)

        ListBox1.Items.Add("C + C = A + D")
        ListBox2.Items.Add(C + C = A + D)

        ListBox1.Items.Add("D > A")
        ListBox2.Items.Add(D > A)

        ListBox1.Items.Add("A < C")
        ListBox2.Items.Add(A < C)

        ListBox1.Items.Add("A <= B")
        ListBox2.Items.Add(A <= B)

        ListBox1.Items.Add("A <= C")
        ListBox2.Items.Add(A <= C)

        ListBox1.Items.Add("A <> D")
        ListBox2.Items.Add(A <> D)

        ListBox1.Items.Add("A <> B")
        ListBox2.Items.Add(A <> B)

        ListBox1.Items.Add("A + B > D")
        ListBox2.Items.Add(A + B > D)

        ListBox1.Items.Add("A < D - B")
        ListBox2.Items.Add(A < D - B)

        ListBox1.Items.Add("C + C <> A + D")
        ListBox2.Items.Add(C + C <> A + D)

        ListBox1.Items.Add("D <= A")
        ListBox2.Items.Add(D <= A)

        ListBox1.Items.Add("A >= C")
        ListBox2.Items.Add(A >= C)

        ListBox1.Items.Add("A = B + C")
        ListBox2.Items.Add(A = B + C)

        ListBox1.Items.Add("A < B")
        ListBox2.Items.Add(A < B)

        ListBox1.Items.Add("A = C")
        ListBox2.Items.Add(A = C)

        ListBox1.Items.Add("A = D")
        ListBox2.Items.Add(A = D)
```

```
        ListBox1.Items.Add("A = B * 2")
        ListBox2.Items.Add(A = B * 2)

    End Sub
```

```
    Private Sub ListBox1_Click(sender As Object, e As System.EventArgs) Handles
ListBox1.Click

        ' quando l'utente clicca un item nel ListBox1
        ' crea una variabile numerica di nome OperazioneCliccata.

        Dim OperazioneCliccata As Integer

        ' memorizza nella variabile OperazioneCliccata il numero d'ordine
dell'item cliccato dall'utente:
        OperazioneCliccata = ListBox1.SelectedIndex

        ' controlla nel ListBox2 il risultato corrispondente al numero
dell’operazione cliccata
        ' e a seconda di questo risultato visualizza il messaggio VERO o FALSO

        If ListBox2.Items(OperazioneCliccata) = True Then
            MsgBox(ListBox1.SelectedItem & vbCrLf & "VERO")
        Else
            MsgBox(ListBox1.SelectedItem & vbCrLf & "FALSO")
        End If

    End Sub

End Class
```

Mandiamo il programma in esecuzione; facciamo un *clic* con il mouse sugli item presenti nel ListBox e leggiamo nei messaggi che si susseguono gli esiti delle varie operazioni di comparazione tra le quattro variabili A, B, C e D:



99: La comparazione di stringhe di testo.

La comparazione di variabili che contengono stringhe (ad esempio parole come “Antonio”, “Mario”, “Luigi”) è utile quando occorre mettere in ordine alfabetico una serie di parole, oppure quando occorre controllare una *password* o una risposta scritta dall’utente.
Nella comparazione di parole VB si attiene a questi criteri:

- una lettera dell’alfabeto è *maggiore* della lettera che la segue (“A” > “B”)
- la parola più lunga è *maggiore* della parola più breve (“Mario” > “Pino”)

Supponiamo di avere all’interno di un programma queste variabili stringhe:

```
Dim A As String = "A"
Dim B As String = "B"
Dim C As String = "Alfa"
Dim D As String = "Beta"
Dim F As String = "Alfabeto"
```

Con queste cinque variabili, le seguenti operazioni comparative su stringhe di testo danno come risultato **True**:

- A > B
- C > D
- F > C
- C = C
- C <> F
- F > (A & B)

Teniamo sempre presente che la comparazione di stringhe di testo è ***case sensitive***, cioè fa distinzione tra lettere maiuscole e lettere minuscole, per cui, ad esempio :

- “Alfa” = “alfa” è **False**;
- “Alfa” < “alfa” è **True**, in quanto la serie delle lettere maiuscole viene prima della serie delle lettere minuscole;
- “Alfa” > “alfa” è **False.**

Capitolo 17: OPERAZIONI LOGICHE.

Per orientare una decisione occorre spesso prendere in considerazione e comparare tra di loro molti elementi: queste operazioni di comparazione vengono collegate in una espressione logica in base alla quale si adotta la decisione finale.
Negli esempi che seguono, vediamo i signori Tizio e Caio che devono prendere decisioni valutando di volta in volta due elementi: il sig. Tizio esige la presenza di entrambi gli elementi, il sig. Caio si accontenta che sia presente uno solo dei due elementi.

La cravatta
Oggi il sig. Tizio è uscito di casa con una idea fissa in testa:
"Se vedo una cravatta gialla di seta, la compero!".
Il sig. Caio, contemporaneamente, è uscito di casa con questa idea:
"Se vedo una cravatta gialla oppure una cravatta di seta, la compero!".
I due giungono contemporaneamente davanti a una vetrina in cui è esposta una sola cravatta blu, di seta. Chi tra i due signori la compera?[57]

Il sorpasso
Il sig. Tizio e il sig. Caio stanno viaggiando in auto, in colonna, molto lentamente, su una strada a due sole corsie, con una linea continua nel mezzo che vieta i sorpassi.
Il sig. Tizio pensa:
"Se non vedo auto arrivare dalla direzione opposta e non vedo una postazione di controllo della polizia stradale, tento il sorpasso".
Il sig. Caio pensa:
"Se non vedo auto arrivare dalla direzione opposta oppure se non vedo una postazione di controllo della polizia stradale, tento il sorpasso".
La situazione sulla strada è questa: si vedono arrivare auto dalla direzione opposta, ma non si vedono postazioni di controllo della polizia.
Poco dopo succede un incidente: quale dei due signori vi rimane coinvolto?[58]

Il picnic

[57] La compera il sig. Caio, perché per lui è sufficiente che la cravatta risponda a una sola delle due condizioni: essere gialla OPPURE essere di seta. Il sig. Tizio non la compera perché cerca una cravatta che risponda ad entrambe queste condizioni: essere gialla E essere di seta.
[58] Il sig. Caio, perché per effettuare il sorpasso si accontenta che sia vera una sola di queste due condizioni: la presenza di auto provenienti dal senso opposto OPPURE la presenza della polizia. Il sig. Tizio non tenta il sorpasso perché per lui è necessario che entrambe le condizioni siano vere (niente auto E niente polizia).

Il sig. Tizio e il sig. Caio stanno cercando un posto per fermarsi a fare un picnic. Il sig. Tizio vuole un posto alberato in riva al torrente, il sig. Caio vuole un posto alberato oppure un posto in riva al torrente.
Dopo un poco scorgono un bel bosco ombroso, con un fresco spiazzo erboso, ma il torrente è piuttosto lontano e dallo spiazzo lo si intravede appena.
Chi si ferma a fare il picnic nello spiazzo?[59]

100: Gli operatori logici.

Come i signori Tizio e Caio delle storielle precedenti, un programma sa mettere in relazione due operazioni comparative (vere o false) in una unica operazione logica che darà complessivamente un risultato vero o falso, utilizzando gli operatori logici elencati nella tabella seguente.

Operatore logico	Esempio	Traduzione
And	If x > y AND x > z	Se la prima comparazione è vera E se la seconda comparazione è vera…
Or	If x > y OR x > z	Se la prima comparazione è vera O se la seconda comparazione è vera…
XOR		Se la prima comparazione è diversa dalla seconda comparazione…

Tabella 13: Gli operatori logici.

Come per le operazioni di comparazione, il risultato di una operazione logica può avere solo due esiti: True oppure False.

[59] Si ferma il sig. Caio, perché per lui è sufficiente che sia vera una di queste due condizioni: il posto alberato OPPURE la vicinanza del torrente. Non si ferma il sig. Tizio, perché per lui è necessario che sussistano contemporaneamente le due condizioni: il posto alberato E il torrente.

101: Tavole di verità.

And

Una operazione logica con l'operatore **And** è **vera** solo nel caso in cui siano vere entrambe le comparazioni che fanno parte dell'operazione, secondo questa tavola di verità:

Prima comparazione		Seconda comparazione		Risultato dell'operazione
Vera	And	**Vera**	=	**Vera**
Vera	And	**Falsa**	=	**Falsa**
Falsa	And	**Vera**	=	**Falsa**
Falsa	And	**Falsa**	=	**Falsa**

Tabella 14: Tavola di verità per l'operatore And.

Or

Una operazione logica con l'operatore **Or** è **vera** nel caso in cui sia vera una delle due comparazioni che fanno parte dell'operazione, secondo questa tavola di verità:

Prima comparazione		Seconda comparazione		Risultato dell'operazione
Vera	Or	**Vera**	=	**Vera**
Vera	Or	**Falsa**	=	**Vera**
Falsa	Or	**Vera**	=	**Vera**
Falsa	Or	**Falsa**	=	**Falsa**

Tabella 15: Tavola di verità per l'operatore Or.

Xor

Una operazione logica con l'operatore **Xor** è **vera** nel caso in cui le comparazioni che fanno parte dell'operazione siano diverse tra loro, secondo questa tavola di verità:

Prima comparazione		Seconda comparazione		Risultato dell'operazione
Vera	Xor	**Vera**	=	**Falsa**
Vera	Xor	**Falsa**	=	**Vera**
Falsa	Xor	**Vera**	=	**Vera**
Falsa	Xor	**Falsa**	=	**Falsa**

Tabella 16: Tavola di verità per l'operatore Xor.

AndAlso

Una operazione logica con l'operatore **AndAlso** è **vera** solo nel caso in cui siano vere entrambe le comparazioni che fanno parte dell'operazione.
La tavola di verità dell'operatore AndAlso è identica a quella dell'operatore And, ma l'operatore **AndAlso** ha un funzionamento detto di *corto circuito*, più rapido dell'operatore **And**, perché se la prima comparazione è falsa l'operatore assegna a tutta l'operazione il valore "falso", senza prendere in considerazione la seconda comparazione.

OrElse

Una operazione logica con l'operatore **OrElse** è **vera** solo nel caso in cui sia vera una delle due comparazioni che fanno parte dell'operazione.
La tavola di verità dell'operatore OrElse è identica a quella dell'operatore Or, ma l'operatore **OrElse** ha un funzionamento detto di *corto circuito*, più rapido dell'operatore **Or**, perché se la prima comparazione è vera l'operatore assegna a tutta l'operazione il valore "vero", senza prendere in considerazione la seconda comparazione.

Vediamo alcuni esempi di operazioni logiche con gli operatori And, Or e XOr.

```
Dim A As Integer = 10
Dim B As Integer = 8
Dim C As Integer = 6
Dim Operazione1, Operazione2, Operazione3 As Boolean

Operazione1 = A > B And B > C
Operazione2 = B > A And B > C
```

```
Operazione3 = A > B And C > B
```

I risultati delle tre operazioni con l'operatore And sono rispettivamente True, False e False.

```
Operazione1 = A < B Or B < C
Operazione2 = B > A Or B > C
Operazione3 = A > B Or C > B
```

I risultati delle tre operazioni con l'operatore Or sono rispettivamente False, True e True.

```
Operazione1 = A > B Xor B > C
Operazione2 = B > A Xor B > C
Operazione3 = A > B Xor C > B
```

I risultati delle tre operazioni con l'operatore XOr sono rispettivamente False, True e True.

Le operazioni logiche non si trovano mai isolate nei programmi, ma si trovano inserite in procedimenti di decisione, per cui gli esempi delle righe precedenti vanno letti come **premesse** alla assunzione di decisioni da parte del computer:

- se il risultato dell'operazione logica è **vero**, allora il programma esegue determinati comandi;
- se il risultato dell'operazione logica è **falso**, allora il programma esegue altri comandi.

Le vedremo quindi concretamente all'opera all'interno dei procedimenti di decisione descritti nel prossimo capitolo.

Capitolo 18: PROCEDIMENTI DI DECISIONE.

La capacità di un programma di prendere decisioni dà all'utente l'illusione di avere a che fare con un'intelligenza interna al computer.
In realtà, se un'intelligenza esiste all'interno del computer, si tratta di un riflesso dell'intelligenza del programmatore, che ha predisposto tutte le condizioni perché un programma nel corso della sua esecuzione possa *prendere delle decisioni*.
Un programma è *istruito* dal programmatore per giungere a *scegliere* tra più possibilità, a *optare* per l'una o per l'altra, a *decidere* quale strada intraprendere quando è di fronte a una scelta.
Istruire il programma sul procedimento di decisione non significa dirgli in anticipo la scelta da compiere quando si troverà in una determinata situazione, ma istruirlo a valutare gli elementi della situazione e ad agire in un modo o nell'altro a seconda della valutazione effettuata.
Il procedimento di decisione del programma è simile a quello che noi adottiamo in situazioni come queste:

I
- Istruzione della decisione: "Se c'è la nebbia mi metto il soprabito".
- Verifica della situazione: la nebbia non c'è.
- Decisione: non metto il soprabito.

II
- Istruzione della decisione: "Se perdo il tram vado in bicicletta"
- Verifica della situazione: il tram è già passato.
- Decisione: vado in bicicletta.

III
- Istruzione della decisione: "Se ci sono posti liberi e se il tempo migliora andiamo allo stadio, altrimenti guardiamo la partita alla TV".
- Verifica della situazione: allo stadio ci sono posti disponibili; continua a piovere.
- Decisione: guardiamo la partita alla TV.

IV
- Istruzione della decisione: "Se c'è abbastanza benzina nel serbatoio e se i bambini resistono non ci fermiamo a questo distributore, ma al prossimo".

- Verifica della situazione: la riserva di carburante è ancora abbondante; i bambini devono fare pipì.
- Decisione: ci fermiamo a questo distributore.

V

- Istruzione della decisione: "Figlio mio, prima di sposarti o ti laurei o ti trovi un lavoro stabile".
- Verifica della situazione: il figlio non si è laureato, ma ha trovato un lavoro stabile.
- Decisione: il figlio si può sposare.

VI

- Istruzione della decisione: "Se riesco a riparare il nostro tosaerba o se il vicino ci presta il suo tosaerba, falcio l'erba del prato"
- Verifica della situazione: non riesco a riparare il tosaerba; il vicino mi presta volentieri il suo.
- Decisione: falcio l'erba del prato.

Osserviamo che nella I e nella II situazione è richiesta la valutazione di un singolo dato:

- I - C'è la nebbia?

Se il dato è vero metto il soprabito, se il dato è falso non lo metto.

- II - È passato il tram?

Se il dato è vero prendo la bicicletta, se il dato è falso prendo il tram.

Nella III e nella IV situazione è richiesta la valutazione di due elementi ed è richiesto che entrambi siano veri:

- III.a - Ci sono posti disponibili?
- III.b - Il tempo migliora?

Se entrambi i dati sono veri andiamo allo stadio, se uno solo dei due dati è falso guardiamo la partita alla TV.

- IV.a - C'è abbastanza benzina?
- IV.b - I bambini resistono?

Se entrambi i dati sono veri proseguiamo per il prossimo distributore, se uno solo dei due dati è falso ci fermiamo a questo distributore.

Nella V e nella VI situazione è ancora richiesta la valutazione di due elementi, ma questa volta è ritenuto sufficiente che uno solo tra i due sia vero:

- V.a - Si è laureato?
- V.b - Ha trovato un lavoro stabile?

Se uno solo dei due dati è vero il figlio si può sposare anche se l'altro dato è falso (si suppone che sia stato verificato in precedenza un altro dato: la partner è disponibile?).

- VI.a - Riesco a riparare il tosaerba?
- VI.b – Il vicino mi presta il suo tosaerba?

Se uno solo dei due dati è vero falcio l'erba, anche se l'altro dato è falso.

Un programma, durante la sua esecuzione, si trova spesso a dover prendere delle decisioni registrando situazioni di questo tipo:

- se l'utente del programma scrive X devo fare…, se l'utente scrive Y devo fare…
- se il numero X è più grande del numero Y devo fare…, altrimenti devo fare…
- se la stringa di testo "A" è uguale alla stringa di testo "B" devo fare…, altrimenti devo fare…

I *ragionamenti* che portano il programma ad assumere decisioni sono costruiti sull'analisi degli elementi che abbiamo visto nei due capitoli precedenti:

- operazioni comparative singole, oppure
- operazioni comparative collegate tra di loro all'interno di operazioni logiche.

102: If… Then…

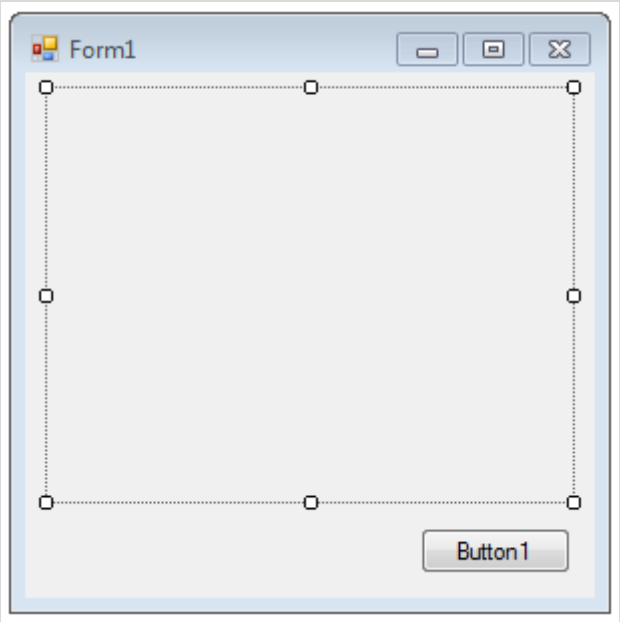
Un procedimento di analisi di operazioni comparative o di operazioni logiche nel codice di un programma è indicato dai termini **If...** (= se...) e **Then...** (= allora...).
Nelle situazioni viste in precedenza, in termini di VB avremmo potuto scrivere:

- **If** (piove) **Then** (metto il soprabito).
- **If** (il tram è passato) **Then** (vado in bicicletta).
- **If** (ci sono posti disponibili) **And** (il cielo è sereno) **Then** (andiamo allo stadio).
- **If** (c'è abbastanza benzina) **And** (i bambini resistono) **Then** (arriviamo al prossimo distributore).
- **If** (ti laurei) **Or** (ti trovi un lavoro stabile) **Then** (ti puoi sposare).
- **If** (riparo il tosaerba) **Or** (il vicino mi presta il suo tosaerba) **Then** (falcio il prato).

Dopo **If** è collocato l'elemento da valutare, dopo **Then** è scritta l'azione da compiere se l'elemento valutato è vero.
Notiamo ancora che nelle situazioni I e II dopo If c'è un solo dato da valutare: se il dato è vero viene eseguita l'azione scritta dopo Then.
Nelle situazioni III e IV dopo If ci sono due dati da valutare; questi due dati sono collegati tra di loro in una operazione logica con l'operatore And, per cui l'operazione logica è vera se entrambi i dati sono veri. Se l'operazione logica è vera, viene eseguita l'azione scritta dopo Then.
Nelle situazioni V e VI dopo If ci sono ancora due dati da valutare, ma questi due dati sono collegati tra di loro in una operazione logica con l'operatore Or, per cui l'operazione logica è vera se almeno uno dei due dati è vero. Se l'operazione logica è vera, viene eseguita l'azione scritta dopo Then.

Esercizio 55: Decisioni basate su singole operazioni di comparazione.

Creiamo un programma nel quale vedremo alcuni esempi di **operazioni comparative** singole inserite in procedimenti di decisioni di tipo **If… Then…**
Apriamo un nuovo progetto e collochiamo nel Form1 un controllo Label e un pulsante Button come nell'immagine seguente.



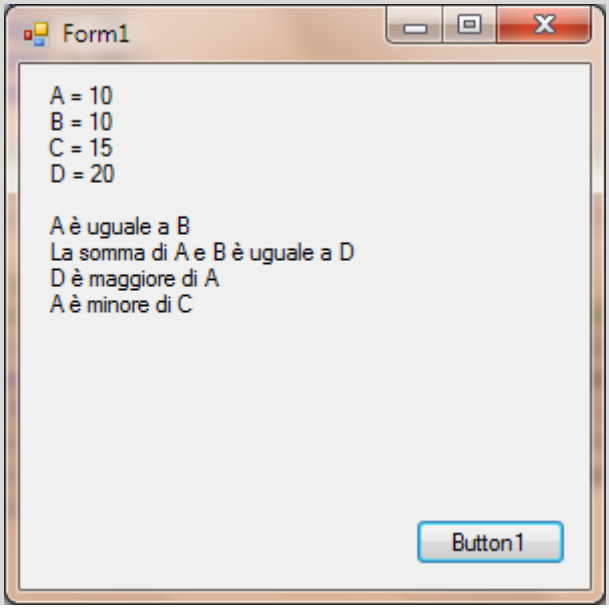
Funzionamento del programma: cliccando il Button1, all'interno della Label1 viene visualizzato un testo con 4 variabili numeriche e alcune comparazioni di queste variabili il cui risultato è "vero".
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim A As Integer = 10
    Dim B As Integer = 10
    Dim C As Integer = 15
    Dim D As Integer = 20
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ' scrive nella Label1 i nomi delle 4 variabili con il loro valore
        Label1.Text = "A = 10" & vbCrLf & "B = 10" & vbCrLf & "C = 15" & vbCrLf &
"D = 20"

        ' inserisce due linee di separazione nel testo della Label1:
        Label1.Text &= vbCrLf & vbCrLf

        ' Ecco una serie di sei procedimenti decisionali:
        ' se la comparazione è vera, allora aggiungi al testo della Label1...:
        If A = B Then Label1.Text &= ("A è uguale a B") & vbCrLf
        If A <> B Then Label1.Text &= ("A è diverso da B") & vbCrLf
```

```
        If A + B = D Then Label1.Text &= ("La somma di A e B è uguale a D") &
vbCrLf
        If A < B Then Label1.Text &= ("A è minore di B") & vbCrLf
        If D > A Then Label1.Text &= ("D è maggiore di A") & vbCrLf
        If A < C Then Label1.Text &= ("A è minore di C")

    End Sub

End Class
```

Mandiamo in esecuzione il programma e clicchiamo il Button1; vediamo il risultato nell'immagine seguente.
Notiamo che nel testo della Label1 appaiono solo le affermazioni corrispondenti a operazioni comparative vere; le frasi corrispondenti a operazioni comparative false vengono ignorate.

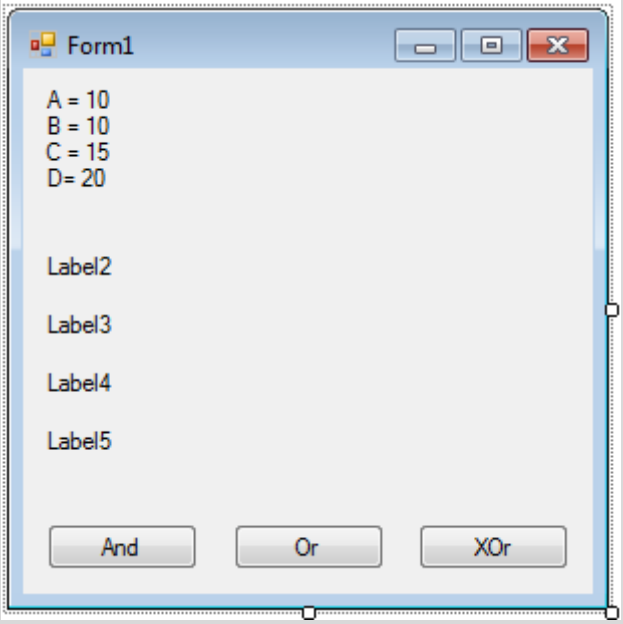


Esercizio 56: Decisioni basate su operazioni logiche.

In questo esercizio vedremo alcuni esempi di decisioni basate su **operazioni logiche**, effettuate con procedimenti di tipo **If… Then…**
Apriamo un nuovo progetto e collochiamo nel Form1 alcuni controlli, seguendo esattamente questo ordine, per il motivo che verrà spiegato alla fine dell'esercizio:

- tre pulsanti Button, **Button1**, **Button2** e **Button3**, la cui proprietà **Text** sarà uguale rispettivamente a **And**, **Or** e **Xor**;
- un controllo **Label1** con la proprietà **Text** impostata come nell'immagine seguente;
- quattro controlli Label: **Label2**, **Label3**, **Label4**, **Label5**.

Nell'immagine seguente vediamo il Form1 con questi controlli:



Funzionamento del programma: cliccando uno dei tre pulsanti And, Or o Xor, nelle Label saranno visualizzate alcune operazioni logiche di tipo And, Or e Xor.
Le Label con operazioni "vere" avranno lo sfondo di colore verde.
Le Label con operazioni "false" avranno lo sfondo di colore rosso.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim A As Integer = 10
    Dim B As Integer = 10
    Dim C As Integer = 15
    Dim D As Integer = 20

    ' Crea una matrice di quattro variabili di testo che conterranno il testo
da visualizzare nelle quattro label
    Dim TestoLabel(3) As String

    ' Crea una matrice di quattro variabili di tipo Boolean (vero/falso)
    Dim OperazioneLogica(3) As Boolean
```

```
    Private Sub Button1_Click() Handles Button1.Click

        TestoLabel(0) = "A > B And B > C"
        TestoLabel(1) = "B > A And B > C"
        TestoLabel(2) = "A > B And C > B"
        TestoLabel(3) = "A < D And C < D"

        OperazioneLogica(0) = A > B And B > C
        OperazioneLogica(1) = B > A And B > C
        OperazioneLogica(2) = A > B And C > B
        OperazioneLogica(3) = A < D And C < D
```

```
        ' chiama in esecuzione la procedura che controlla la validità di
queste operazioni logiche,
        ' passandole la proprietà Left del pulsante premuto:
        Call ControllaOperazioniLogiche(Button1.Left)

    End Sub
```

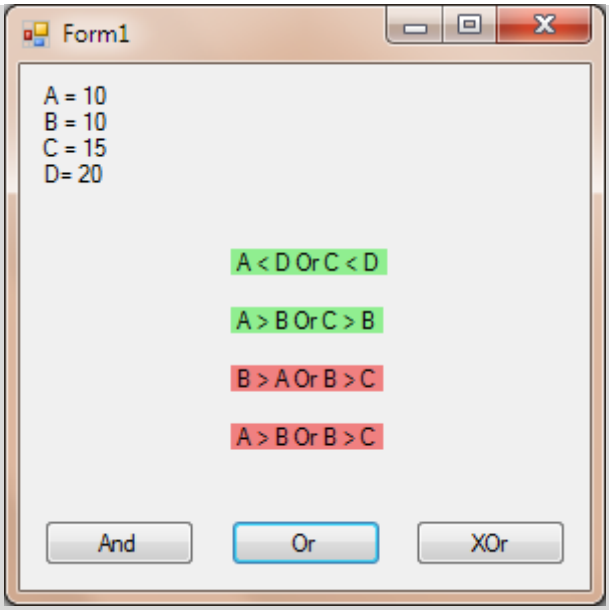
```
    Private Sub Button2_Click() Handles Button2.Click

        TestoLabel(0) = "A > B Or B > C"
        TestoLabel(1) = "B > A Or B > C"
        TestoLabel(2) = "A > B Or C > B"
        TestoLabel(3) = "A < D Or C < D"

        OperazioneLogica(0) = A > B Or B > C
        OperazioneLogica(1) = B > A Or B > C
        OperazioneLogica(2) = A > B Or C > B
        OperazioneLogica(3) = A < D Or C < D

        ' chiama in esecuzione la procedura che controlla la validità di
queste operazioni logiche,
        ' passandole la proprietà Left del pulsante premuto:
        Call ControllaOperazioniLogiche(Button2.Left)

    End Sub
```

```
    Private Sub Button3_Click() Handles Button3.Click

        TestoLabel(0) = "A > B Xor B > C"
        TestoLabel(1) = "B > A Xor B > C"
        TestoLabel(2) = "A > B Xor C > B"
        TestoLabel(3) = "A < D Xor C < D"

        OperazioneLogica(0) = A > B Xor B > C
        OperazioneLogica(1) = B > A Xor B > C
        OperazioneLogica(2) = A > B Xor C > B
        OperazioneLogica(3) = A < D Xor C < D

        ' chiama in esecuzione la procedura che controlla la validità di
queste operazioni logiche,
        ' passandole la proprietà Left del pulsante premuto:
        Call ControllaOperazioniLogiche(Button3.Left)

    End Sub
```

```
    Private Sub ControllaOperazioniLogiche(ByVal MargineSinistro As Integer)

        ' La procedura riceve, indicato tra parentesi,
        ' il valore della proprietà Left del pulsante "chiamante" e gli
assegna il nome MargineSinistro.
        ' Attività della procedura:
        ' crea una variabile numerica di nome Contatore, uguale a 0:
        Dim Contatore As Integer = 0

        ' inizia un ciclo For... Next che passa in rassegna le Label presenti
nel Form1,
```

```
        ' partendo dalla Label creata per ultima:

        ' per ogni controllo che si trova nel Form1..."
        For Each Control In Me.Controls

            ' se questo controllo è una label..."
            If TypeOf Control Is Label Then
                ' sposta la Label in linea con il pulsante che è stato
premuto e
                ' visualizza nella Label il testo dell'operazione logica che
corrisponde
                ' al valore della variabile Contatore:
                Control.left = MargineSinistro
                Control.text = TestoLabel(Contatore)

                ' se l'operazione logica è vera imposta il colore verde come
colore di sfondo della Label
                If OperazioneLogica(Contatore) = True Then Control.backcolor
= Color.LightGreen
                ' se l'operazione logica è vera imposta il colore rosso come
colore di sfondo della Label
                If OperazioneLogica(Contatore) = False Then Control.backcolor
= Color.LightCoral
            End If

            ' aumenta di una unità il valore della variabile Contatore:
            Contatore += 1
            ' se il valore di Contatore è arrivato a 4 significa che il
risultato delle 4 operazioni logiche è già stato scritto nelle 4 label, per
cui il ciclo For... Next deve essere interrotto:
            If Contatore = 4 Then Exit For
            ' se il ciclo delle 4 label non è finito, passa alla prossima
label:
        Next

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Ricordiamo che quando il ciclo di comandi retto da For Each… Next si riferisce a un gruppo di controlli, VB parte dall'ultimo controllo collocato sul Form.
Nel nostro esercizio, dovendo modificare nel corso del programma il contenuto dei controlli **Label2**, **Label3**, **Label4** e **Label5**, era necessario collocare queste quattro label per ultime nel form, per evitare di andare a modificare il contenuto della Label1, che in questo modo rimane invariato.

103: Sintassi dei procedimenti If… Then…

Un procedimento di decisione di tipo **If… Then**… che si conclude su una sola riga è **completo** e non richiede istruzioni aggiuntive.
Prendiamo un esempio dall'esercizio precedente:

```
If OperazioneLogica(Contatore) = True Then Control.backcolor =
Color.LightGreen
```

Questo procedimento, che nell'*editor* di VB si sviluppa su una unica riga, è completo.
Un procedimento di tipo **If… Then** che si sviluppa su più righe, invece, richiede una sintassi particolare:

- dopo il comando **Then** il codice deve andare a capo e passare all'istruzione successiva;
- il procedimento deve concludersi con il comando **End If**.

Prendiamo un altro esempio dall'esercizio precedente:

```
If TypeOf Control Is Label Then
    Control.left = MargineSinistro
    Control.text = TestoLabel(Contatore)
```

```
                If OperazioneLogica(Contatore) = True Then Control.backcolor
= Color.LightGreen
                If OperazioneLogica(Contatore) = False Then Control.backcolor
= Color.LightCoral
            End if
```

In questo esempio, le righe evidenziate in giallo mostrano un procedimento di decisione che si svolge per alcune righe di istruzioni e termina con il comando **End If**.
Nei procedimenti di questo tipo, articolati su più righe, la mancanza del comando **End If** conclusivo provoca un errore di sintassi.
E' tuttavia difficile incorrere in questo errore perché quando si avvia la scrittura di un procedimento **If… Then** su più righe, l'*editor* di VB si preoccupa di chiuderlo automaticamente aggiungendo la riga conclusiva **End If**.
Vediamo un esempio.
Nella finestra del codice di un programma, avviamo questo procedimento:

```
        If A > B Then
```

Al termine di questa riga, premiamo il pulsante INVIO: notiamo che l'*editor* di VB aggiunge automaticamente la riga di chiusura del procedimento:

```
        If A > B Then

        End if
```

104: If… Then… Else…

VB consente di prendere in considerazione anche un maggior numero di possibilità di azione e di fare *ragionamenti* più articolati, utilizzando assieme a **If… Then** una istruzione ulteriore: **Else**.
Si può utilizzare **Else** come nel linguaggio corrente si usa l'avverbio *altrimenti*, ogni volta che in un procedimento di decisione occorre prendere in considerazione più eventualità, una alternativa all'altra.
Un procedimento logico di questo tipo: "*se abbiamo abbastanza soldi andiamo al cinema, altrimenti stiamo in casa a guardare la TV*" si traduce in VB in questo schema e nelle righe di codice scritte di seguito:

- **Se** è vero che abbiamo abbastanza soldi
- **Allora** andiamo al cinema
- **Altrimenti** stiamo in casa a guardare la TV
- **Fine** del procedimento

Ecco un esempio di questo ragionamento in termini di VB (è possibile provarlo in un nuovo progetto con un Form vuoto):

```
Public Class Form1
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim A As Integer = 50
        '(soldi che abbiamo in tasca)
        Dim B As Integer = 22
        '(costo di due biglietti per il cinema)

        If A > B Then
            MsgBox("Andiamo al cinema")
        Else
            MsgBox("Restiamo a casa")
        End If

    End Sub

End Class
```

Notiamo che il procedimento di decisione si sviluppa su più righe:

- parte dalla riga iniziale con **If…**
- si sviluppa nella parte centrale con **Else…**
- si conclude nell'ultima riga con **End If**.

Il codice dell'esercizio precedente, nella parte che riguarda l'impostazione del colore di fondo delle Label, avrebbe potuto essere scritto in questo modo:

```
        ' se l'operazione logica è vera imposta il colore verde come colore di
fondo:
        If OperazioneLogica(Contatore) = True Then
            Control.BackColor = Color.LightGreen

        Else
            ' altrimenti imposta il colore rosso:
            Control.BackColor = Color.LightCoral
        End If
```

105: If… Then… Else… con elementi True/False.

Può accadere che in un procedimento di tipo If… Then… Else… siano messi in relazione tra loro elementi che possono assumere solo i valori **True** o **False**, quali:

- variabili di tipo Boolean, che possono essere solo True o False;
- proprietà degli oggetti che hanno solo le condizioni True o False (Visible, Enabled, AutoSize, …);
- operazioni di comparazione, il cui risultato può essere solo True o False.

In questi casi il procedimento If… Then… Else… può essere sostituito da una sola riga di codice che dà lo stesso risultato in modo sintetico e diretto.
Vediamo un esempio.

Collochiamo in un form un controllo NumericUpDown e un controllo Label, poi copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim A As Integer
    Dim B As Integer = 10

    Private Sub NumericUpDown1_ValueChanged(sender As Object, e As
System.EventArgs) Handles NumericUpDown1.ValueChanged

        A = NumericUpDown1.Value

        If A < B Then
            Label1.Visible = True
        Else
            Label1.Visible = False
        End If

    End Sub

End Class
```

Mandiamo in esecuzione il programma e modifichiamo il valore del controllo NumericUpDown.
Notiamo il procedimento If… Then… Else…:

- **se** la comparazione A < B è vera, il controllo Label1 è reso visibile;
- **altrimenti**, la Label1 è resa non visibile.

Abbiamo in questo procedimento una relazione tra elementi omogenei che possono assumere solo i valori True o False:

- la **proprietà** Visible della Label1 può essere solo True o False;
- la **comparazione** A < B può essere solo True o False.

Grazie a questa omogeneità, il procedimento If… Then… Else… nel listato può essere sintetizzato in questo modo, con lo stesso risultato:

```
Public Class Form1

    Dim A As Integer
    Dim B As Integer = 10

    Private Sub NumericUpDown1_ValueChanged(sender As Object, e As
System.EventArgs) Handles NumericUpDown1.ValueChanged

        A = NumericUpDown1.Value
        Label1.Visible = A < B

    End Sub

End Class
```

L'operatore Not.

Anche l'operatore **Not** può essere utilizzato per sintetizzare procedimenti di decisione riguardanti variabili o proprietà di tipo Boolean, che supportino solo i valori **True** o **False**.
Utilizzato invece di un procedimento di tipo If… Then… Else…, l'operatore **Not** analizza una variabile o una proprietà di un controllo e, qualsiasi il suo stato (**True** o **False**), la imposta nel suo contrario.
Vediamo un esempio.
Supponiamo di avere in un Form alcuni controlli e componenti attivi e visibili; per disattivarli o renderli non visibili durante l'esecuzione del programma, è possibile scrivere procedimenti di decisione di tipo **If… Then… Else…** che controllino una proprietà dell'oggetto e la trasformino nel suo contrario:

```
        ' Controlla la proprietà .Enable in un Timer: se il Timer è attivo lo
disattiva,
        ' altrimenti (se il Timer è disattivo) lo attiva:
        If Timer1.Enabled = True Then
            Timer1.Enabled = False
        Else
            Timer1.Enabled = True
        End If

        ' Controlla la proprietà .Visibile di una Label: se la Label è visibile
la rende non visibile,
        ' altrimenti (se la Label è non visibile) la rende visibile:
        If Label1.Visible = False Then
            Label1.Visible = True
        Else
            Label1.Visible = False
        End If
```

L'operatore **Not** consente di effettuare le stesse operazioni, con gli stessi risultati, in modo sintetico e diretto:

```
        Timer1.Enabled = Not Timer1.Enabled
        Label1.Visible = Not Label1.Visible
```

106: ElseIf…

Mentre con il comando **Else…** si possono prendere in considerazione solo due alternative, con il comando **ElseIf…** (= oppure se…) i procedimenti di decisione possono essere ampliati sino ad abbracciare un numero indefinito di alternative.
Ecco un esempio (è possibile provarlo in un nuovo progetto con un Form vuoto):

```
Public Class Form1
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim A As Integer = 15
        Dim B As Integer = 25

        If A > B Then
            MsgBox("A è maggiore di B" & vbCrLf & "La differenza è " & A - B)

        ElseIf A < B Then
            MsgBox("A è minore di B" & vbCrLf & "La differenza è " & B - A)

        Else
            MsgBox("A e B sono uguali")

        End If

    End Sub

End Class
```

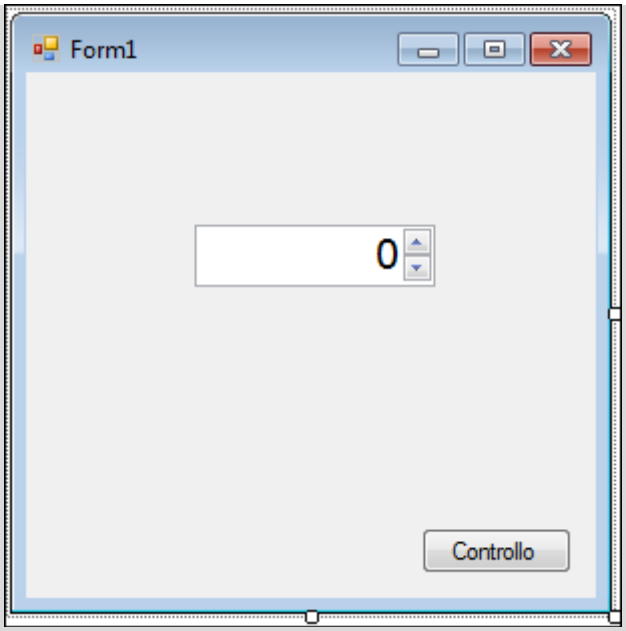
Nel prossimo esercizio vedremo un esempio più complesso.

Esercizio 57: ElseIf.

Scopo di questo esercizio è creare un programma per valutare un test con venti domande.
Il programma, dopo avere ricevuto il numero delle risposte esatte, deve esprimere un giudizio che va da “Non sufficiente” (meno di 11 risposte esatte) a “Ottimo” (più di 17 risposte esatte).
Apriamo un nuovo progetto e collochiamo nel Form1 un controllo **NumericUpDown**, in cui l’utente del programma immetterà il numero delle risposte esatte.
Proprietà del controllo **NumericUpDown**:

- **Font = Microsoft Sans Serif a 16 punti;**
- **TextAlign = Right.**

Collochiamo nel Form1 anche un pulsante **Button1**, con la proprietà **Text = Controllo**.



Copiamo e incolliamo nella Finestra del Codice questo listato:

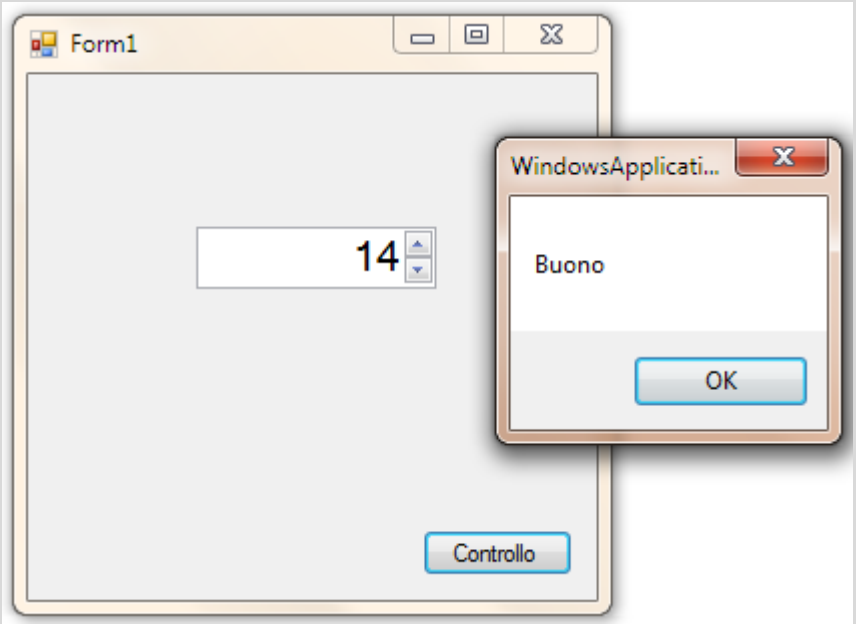
```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim NumeroRisposteEsatte As Integer

        ' prendi il numero delle risposte esatte dal controllo NumericUpDown:
        NumeroRisposteEsatte = NumericUpDown1.Value

        ' Analisi del numero delle risposte esatte e procedimento di
decisione del giudizio:

        If NumeroRisposteEsatte < 11 Then
            MsgBox("Non sufficiente")

        ElseIf NumeroRisposteEsatte > 10 And NumeroRisposteEsatte < 14 Then
            MsgBox("Sufficiente")

        ElseIf NumeroRisposteEsatte > 13 And NumeroRisposteEsatte < 16 Then
            MsgBox("Buono")

        ElseIf NumeroRisposteEsatte > 15 And NumeroRisposteEsatte < 18 Then
            MsgBox("Distinto")

        ElseIf NumeroRisposteEsatte > 17 And NumeroRisposteEsatte < 21 Then
            MsgBox("Ottimo")

        Else
            MsgBox("Non ci possono essere più di 20 risposte esatte!")
        End If

    End Sub
```

```
End Class
```

Vediamo che all'interno del processo che va da **If… a End If**, il computer analizza le varie possibilità:

- **se** è vero che la variabile `NumeroRisposteEsatte` è inferiore a 11 allora...
- **oppure se** è vero che la variabile `NumeroRisposteEsatte` è maggiore di 10 ma inferiore a 14 allora...
- **oppure se** è vero che la variabile `NumeroRisposteEsatte` è maggiore di 13 ma inferiore a 16 allora...
- **oppure se** è vero che la variabile `NumeroRisposteEsatte` è maggiore di 15 ma inferiore a 18 allora...
- **oppure se** è vero che la variabile `NumeroRisposteEsatte` è maggiore di 17 ma inferiore a 21 allora...
- **altrimenti,** in tutti gli altri casi …
- **fine** del procedimento.

Ecco un'immagine del programma in esecuzione:



107: Select Case… End Select.

Vediamo ora un altro procedimento di analisi di dati e di decisione, basato sul comando **Select Case… End Select**.
Si tratta di un procedimento che consente di effettuare scelte tra molte alternative, come **Else If**; l'uso di **Select Case… End Select** è preferibile, per la sua semplicità d'uso, quando si tratta di analizzare un numero molto ampio di condizioni.

In termini descrittivi, **Select Case… End Select** funziona in questo modo:

- **Seleziona il caso** in cui si trova la variabile **xy;**
- **se il caso è questo…** esegui queste operazioni;
- **se il caso è questo…** esegui queste operazioni;
- **se il caso è questo…** esegui queste operazioni;
  e così via, sino alla
- **fine** del procedimento di selezione.

Vediamo un esempio concreto.

Riprendiamo il caso dell'esercizio precedente; il programma deve esprimere un giudizio su un test con 20 domande, in base al numero delle risposte esatte.

Utilizzando **Select Case** invece di **If… Then**, il listato potrebbe essere scritto in questo modo:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim NumeroRisposteEsatte As Integer

        ' prendi il numero delle risposte esatte dal controllo NumericUpDown:
        NumeroRisposteEsatte = NumericUpDown1.Value

        Select Case NumeroRisposteEsatte
            Case 0 To 10
                MsgBox("Non sufficiente")
            Case 11 To 13
                MsgBox("Sufficiente")
            Case 14, 15
                MsgBox("Buono")
            Case 16, 17
                MsgBox("Distinto")
            Case 18, 19, 20
                MsgBox("Ottimo")
            Case Else
                MsgBox("Non ci possono essere più di 20 risposte esatte!")
        End Select

    End Sub

End Class
```

Notiamo come in **Select Case** possono essere indicate le varie possibilità:

- è possibile specificare i numeri delle risposte esatte (14, 15);
- è possibile indicare un intervallo di numeri, dal minore al maggiore (0 To 10, 11 To 13).

108: Analisi di variabili di testo con Select Case e If… Then.

I comandi **Select Case** e **If… Then…** non funzionano solo con analisi di variabili numeriche o di operazioni logiche basate su numeri, come abbiamo visto sino a ora, ma anche con variabili di testo.
Ne vedremo un esempio nel prossimo esercizio.

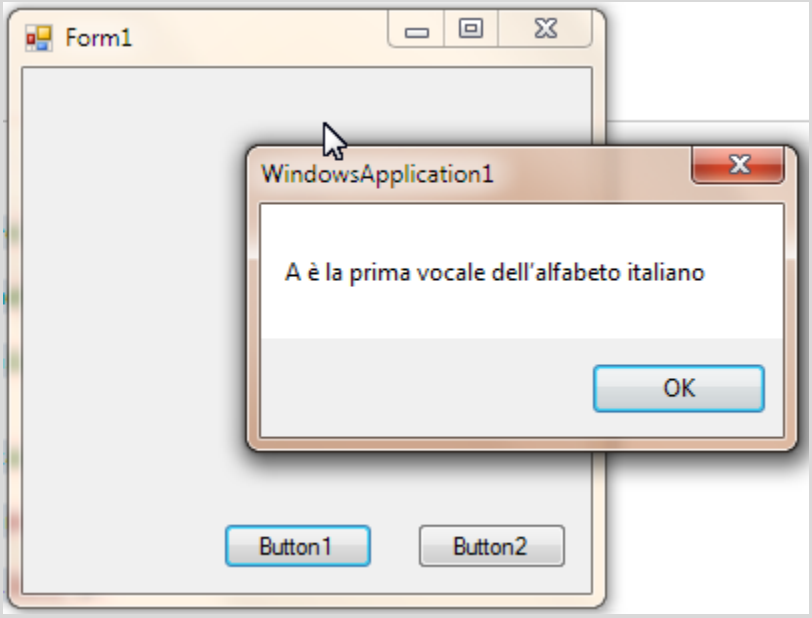
Esercizio 58: Analisi di variabili di testo.

In questo programma l'utente scrive una vocale in un box di immissione dati.
Il programma esamina la vocale e visualizza un messaggio relativo a questa vocale.
Apriamo un nuovo progetto e collochiamo nel Form1 due pulsanti Button: **Button1** e **Button2**.
I due pulsanti svolgono la stessa funzione (l'analisi della vocale immessa dall'utente), utilizzando però due procedimenti diversi: il **Button1** procede con **Select Case**, il **Button2** procede con **If… Then**.
Copiamo e incolliamolo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea la variabile Vocale, destinata a contenere un testo:
        Dim Vocale As String

        ' assegna alla variabile Vocale ciò che l'utente immette nel box di
dialogo:
        Vocale = InputBox("Scrivi una vocale")

        ' trasforma in maiuscolo la lettera che l'utente ha scritto:
        Vocale = Vocale.ToUpper

        'analizza la variabile Vocale e agisci di conseguenza:
        Select Case Vocale
            Case "A"
                MsgBox("A è la prima vocale dell'alfabeto italiano")
            Case "E"
                MsgBox("E è la seconda vocale dell'alfabeto italiano")
            Case "I"
                MsgBox("I è la terza vocale dell'alfabeto italiano")
            Case "O"
                MsgBox("O è la quarta vocale dell'alfabeto italiano")
            Case "U"
                MsgBox("U è la quinta vocale dell'alfabeto italiano")
            Case Else
                MsgBox("Non hai scritto una vocale")
        End Select

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        ' Crea la variabile Vocale, destinata a contenere un testo:
        Dim Vocale As String

        ' assegna alla variabile Vocale ciò che l'utente immette nel box di
dialogo:
        Vocale = InputBox("Scrivi una vocale")

        ' trasforma la lettera che l'utente ha scritto in maiuscolo:
        Vocale = Vocale.ToUpper

        'analizza la variabile Vocale e agisci di conseguenza:
        If Vocale = "A" Then
            MsgBox("A è la prima vocale dell'alfabeto italiano")
        ElseIf Vocale = "E" Then
            MsgBox("E è la seconda vocale dell'alfabeto italiano")
        ElseIf Vocale = "I" Then
            MsgBox("I è la terza vocale dell'alfabeto italiano")
        ElseIf Vocale = "O" Then
            MsgBox("O è la quarta vocale dell'alfabeto italiano")
        ElseIf Vocale = "U" Then
            MsgBox("U è la quinta vocale dell'alfabeto italiano")
        Else
            MsgBox("Non hai scritto una vocale")
        End If

    End Sub

End Class
```

Mandiamo in esecuzione il programma e notiamo che il risultato dell'uso dei due procedimenti (**Select Case** e **End If**) è identico:



Compaiono in questo esercizio due strumenti che VB offre al programmatore per dialogare con l'utente del suo programma:

- la funzione **InputBox**(), che crea una casella con la quale l'utente può comunicare dati al programma (in questo programma l'utente scrive una vocale);
- la funzione **MsgBox**(), che visualizza un box con un messaggio per l'utente.

Vedremo il funzionamento di InputBox e MsgBox nel prossimo capitolo.

Capitolo 19: LE FUNZIONI MSGBOX E INPUTBOX.

Le funzioni[60] **MsgBox** e **InputBox** sono due strumenti che visualizzano una finestra di dialogo con l'utente del programma.
In particolare:

- **MsgBox** serve a trasmettere all'utente informazioni o richieste alle quali l'utente può rispondere con un semplice Sì / No / Non so;
- **InputBox** serve invece a ricevere dall'utente dati più complessi (ad esempio: un nome, una data, una frase).

Entrambe le funzioni bloccano l'esecuzione del programma: quando compare una di queste finestre di dialogo, l'utente **deve sempre attivarsi** per rispondere, altrimenti il programma non prosegue.

109: La funzione MsgBox.

La funzione **MsgBox** è uno dei modi più semplici e più efficaci per visualizzare un messaggio per l'utente durante l'esecuzione di un programma.
Possiamo vedere un esempio di questi box contenitori di messaggi in tutti i programmi che funzionano in ambiente Windows: ad esempio, quando si chiude un lavoro senza averlo prima salvato compare abitualmente un box con un messaggio che chiede se si vuole salvare il lavoro o no. Davanti a questo messaggio l'utente ha tre possibilità di azione:

- salvare il lavoro e uscire dal programma;
- uscire dal programma senza salvare il lavoro fatto;
- tornare al programma e continuare il lavoro, annullando il comando di uscita dal programma.

A seconda del pulsante che viene premuto in questo box, il programma in esecuzione riceve risposta, esamina questa risposta e agisce di conseguenza, eseguendo una di queste istruzioni:

- se è stato premuto il pulsante **Salva** allora salva il lavoro e chiudi il programma;

[60] Il termine funzione indica una procedura che riceve dati dal programmatore o dall'utente, li elabora e restituisce il dato dell'elaborazione come una nuova variabile, utilizzabile nello svolgimento del programma. Si veda il paragrafo 79 a pag. 372.

- se è stato premuto il pulsante **Esci** allora chiudi il programma senza salvare il lavoro;
- se è stato premuto il pulsante **Annulla** allora annulla il comando di uscita dal programma.

La funzione **MsgBox** visualizza dunque un messaggio in una finestra di dialogo, generalmente corredato da più pulsanti (ad esempio: **Sì, No, Annulla**), e attende che l'utente faccia un *clic* con il mouse su uno dei pulsanti visualizzati.
A seconda del pulsante premuto dall'utente, la funzione restituisce un risultato **MsgBoxResult**, diverso per ogni pulsante. Ad esempio:

- il risultato è **MsgBoxResult.Yes**, se è stato premuto il pulsante SI;
- il risultato è **MsgBoxResult.No**, se è stato premuto il pulsante NO;
- il risultato è **MsgBoxResult.Cancel**, se è stato premuto il pulsante Annulla.

Analizzando la proprietà di MsgBoxResult il programma può *comprendere* quale scelta è stata effettuata dall'utente e può proseguire effettuando determinate azioni anziché altre.

Il programmatore non deve fare alcuna operazione grafica per visualizzare il box: è sufficiente una sola riga di codice per visualizzare sullo schermo una finestra con il testo del messaggio.
Il programmatore ha inoltre la facoltà di decidere:

- quali pulsanti visualizzare nel box assieme al messaggio;
- quale *stile* dare al messaggio, inserendovi un'icona e un suono appropriati;
- quale titolo dare al box.

Il comando più semplice per visualizzare un MsgBox è questo:

```
MsgBox("Vogliamo iniziare il lavoro?")
```

Questa istruzione visualizza nel box la domanda "Vogliamo iniziare il lavoro?"; il programmatore non ha impostato altre opzioni, per cui il box avrà il titolo del programma e conterrà un solo pulsante di risposta: **OK**.

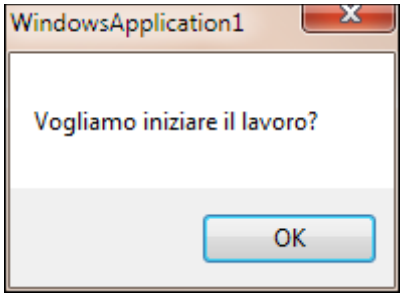


Figura 138: Un MsgBox con un solo pulsante.

In questo caso, con un unico pulsante, non sarà necessario analizzare il risultato MsgBoxResult, per identificare quale pulsante è stato premuto dall'utente.
Per visualizzare nel box più di un pulsante è possibile integrare la funzione come segue:

```
MsgBox("Vuoi uscire dal programma?", MsgBoxStyle.YesNo)
```

L'aggiunta della opzione **MsgBoxStyle.YesNo** visualizza nel box due pulsanti (Sì/No).
Ora, a differenza dell'esempio precedente, l'utente ha la possibilità di premere l'uno o l'altro pulsante, ed è quindi necessario che il programma analizzi la risposta dell'utente, per capire quale dei due pulsanti è stato premuto.

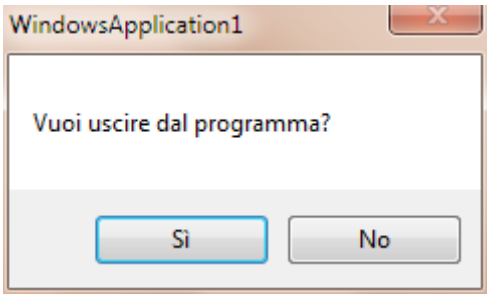


Figura 139: Un MsgBox con due pulsanti.

La risposta dell'utente è **MsgBoxResult.Yes** se l'utente preme il pulsante Sì, oppure **MsgBoxResult.No** se l'utente preme il pulsante No.
Una funzione MsgBox() con due pulsanti potrebbe dunque essere gestita in questo modo:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        MsgBox("Vuoi uscire dal programma?", MsgBoxStyle.YesNo)

        ' se l'utente ha cliccato il pulsante Sì, il programma termina:
        If MsgBoxResult.Yes Then
            Application.Exit()

            ' altrimenti prosegue per la sua strada:
        Else
            Application.DoEvents()
        End If

    End Sub

End Class
```

E' possibile visualizzare in un box altri pulsanti, utilizzando gli stili di visualizzazione elencati in questa tabella:

Stile	Pulsanti visualizzati nel box:
MsgBoxStyle.OKOnly	Un solo pulsante, con il testo OK

MsgBoxStyle.OKCancel	Due pulsanti, con il testo OK, Annulla
MsgBoxStyle.AbortRetryIgnore	Tre pulsanti, con il testo Interrompi, Riprova, Ignora
MsgBoxStyle.YesNoCancel	Tre pulsanti, con il testo Sì, No, Annulla
MsgBoxStyle.YesNo	Due pulsanti, con il testo Sì, No
MsgBoxStyle.RetryCancel	Due pulsanti, con il testo Riprova, Annulla

Tabella 17: Stili di visualizzazione dei pulsanti in un MsgBox.

La tabella seguente riporta i risultati restituiti da un MsgBox, a seconda del pulsante premuto dall'utente:

Pulsante	Risultato	Valore numerico
OK	MsgBoxResult.Ok	1
Annulla	MsgBoxResult.Cancel	2
Interrompi	MsgBoxResult.Abort	3
Riprova	MsgBoxResult.Retry	4
Ignora	MsgBoxResult.Ignore	5
Sì	MsgBoxResult.Yes	6
No	MsgBoxResult.No	7

Tabella 18: Risultati restituiti dalla funzione MsgBox.

È possibile dare al box uno stile particolare, per *rafforzarne* il messaggio, integrando la funzione come segue:

```
        MsgBox("Vuoi uscire dal programma?", MsgBoxStyle.YesNo +
MsgBoxStyle.Question)
```

Questo esempio visualizza nel messaggio un'icona con il punto di domanda:

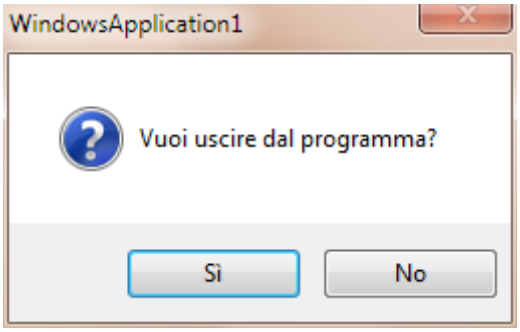
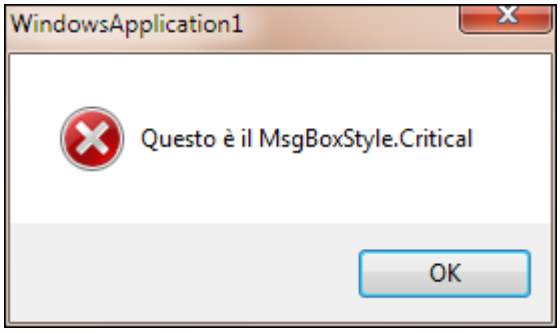
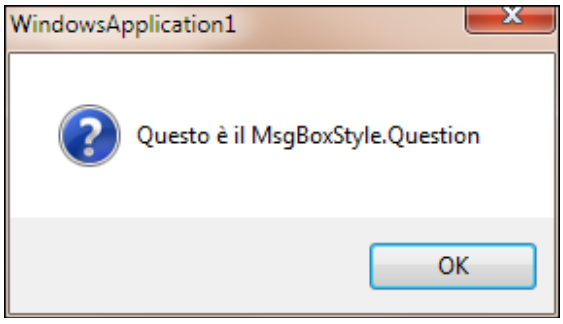
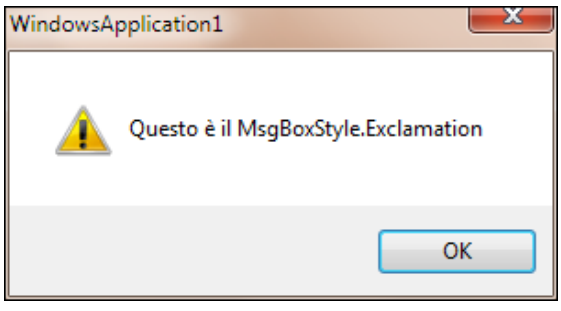


Figura 140: Un MsgBox con due pulsanti e un'icona.

Gli stili con icone a disposizione del programmatore sono quattro; li vediamo in questa tabella:

Parametro	Descrizione	Messaggio visualizzato
MsgBoxStyle.Critical	Richiama l'attenzione dell'utente su un elemento critico	
MsgBoxStyle.Question	Presenta all'utente una domanda	
MsgBoxStyle.Exclamation	Presenta all'utente un messaggio in tono sostenuto	

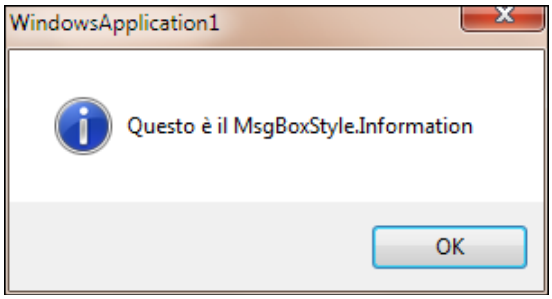
VbInformation	Informazione	

Tabella 19: Stili disponibili per i MsgBox.

Il primo di questi stili, MsgBoxStyle.Critical, è accompagnato da un suono particolare.
È possibile aggiungere un titolo al box, nella sua barra di intestazione, completando la scrittura della funzione come segue:

```
        MsgBox("Vuoi uscire dal programma?", MsgBoxStyle.YesNo +
MsgBoxStyle.Critical, "ATTENZIONE")
```

Notiamo che la funzione si struttura in tre parti, separate da virgole:

- nella prima parte si trovano il testo del messaggio o della domanda indirizzati all'utente del programma, scritti tra virgolette;
- nella parte centrale sono indicati gli elementi grafici (numero e tipo dei pulsanti, tipo di icona);
- nella terza parte si trova il titolo del box, scritto tra virgolette.

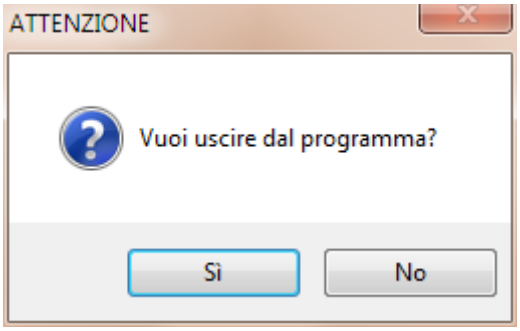


Figura 141: Un MsgBox con due pulsanti, un'icona e un titolo.

Se nella scrittura della funzione MsgBox() non viene specificato un titolo da parte del programmatore, il programma visualizza come titolo nella barra del box il nome della applicazione.

110: La funzione InputBox.

La funzione **InputBox** svolge in un certo senso un ruolo opposto a quello della funzione **MsgBox**: mentre lo scopo principale di MsgBox è **presentare** all'utente un messaggio e alcune opzioni a scelta obbligata, lo scopo principale di InputBox è **ricevere** informazioni dall'utente.
Anche la funzione MsgBox riceve delle informazioni dall'utente, quando questi preme uno dei pulsanti che appaiono nel box, ma la funzione InputBox è in grado di ricevere dall'utente informazioni aperte e più complesse (testi, parole, numeri).
InputBox visualizza sullo schermo una finestra di dialogo di questo tipo, con questa riga di istruzioni:

```
InputBox("Come ti chiami?")
```

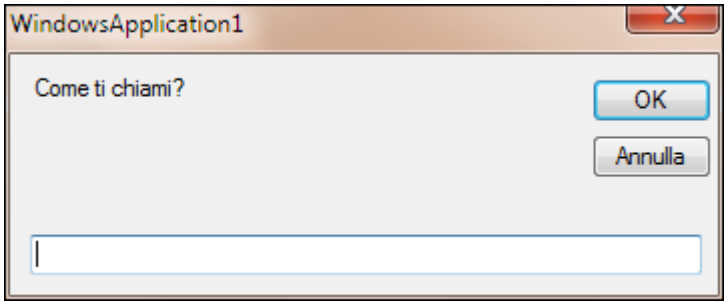


Figura 142: Una finestra InputBox.

Come si nota nell'immagine, il programma crea la finestra di dialogo aggiungendo al messaggio scritto dal programmatore ("Come ti chiami") questi elementi:

- un titolo nella barra in alto;
- una riga-riquadro in cui l'utente scriverà un dato;
- i due pulsanti OK e Annulla.

In questo InputBox di esempio, l'utente scrive il suo nome nella riga-riquadro, poi clicca il pulsante OK. A quel punto l'InputBox si chiude; perché il nome scritto dall'utente non vada perso è necessario che esso sia memorizzato in una variabile: ecco dunque che la riga precedente dovrebbe essere completata in questo modo:

```
Dim NomeUtente As String = InputBox("Come ti chiami?")
```

In questo modo, quando l'utente chiude la finestra dell'InputBox, il testo scritto nella riga-riquadro viene assegnato alla variabile NomeUtente, e può essere utilizzato nel proseguimento del programma.
E' possibile arricchire la finestra dell'InputBox aggiungendo un titolo nella barra in alto e un testo predefinito nella riga-riquadro riservata all'utente, in questo modo:

```
Dim NomeUtente As String = InputBox("Come ti chiami?", "CIAO", "Scrivi il
tuo nome in questa riga")
```

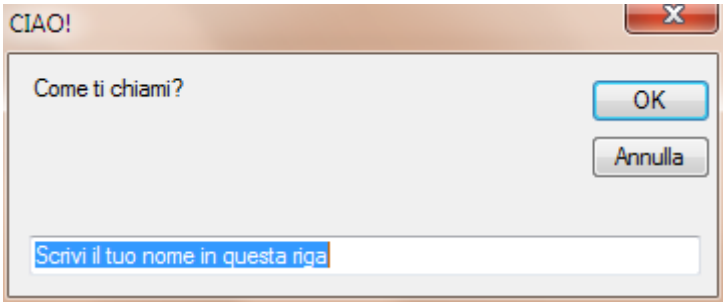


Figura 143: Una finestra InputBox con un titolo e un testo predefinito.

Notiamo che i parametri scritti tra parentesi sono tre, separati da virgole e scritti tra virgolette:

- il primo parametro contiene il testo del messaggio o della richiesta cui l'utente deve rispondere;
- il secondo parametro contiene il titolo della finestra di dialogo;
- il terzo parametro contiene il testo predefinito che compare evidenziato in blu nella riga-riquadro riservata all'utente, all'apertura del box.

E' possibile aggiungere due parametri aggiuntivi, per definire la posizione in cui verrà visualizzato l'InputBox sul monitor.
Questo comando, ad esempio, colloca l'InputBox in alto a sinistra nel Form1:

```
        Dim Nomeutente As String = InputBox("Come ti chiami?", "CIAO", "Scrivi
qui il tuo nome", Me.Left, Me.Top)
```

Questo comando, invece, colloca l'angolo superiore sinistro dell'InputBox al centro del Form1:

```
        Dim Nomeutente As String = InputBox("Come ti chiami?", "CIAO", "Scrivi
qui il tuo nome", Me.Left + Me.Width / 2, Me.Top + Me.Height / 2)
```

Nel prossimo esercizio vedremo un programma basato sull'uso delle due funzioni InuptBox e Msgbox.

Esercizio 59: Le funzioni InputBox e MsgBox.

Apriamo un nuovo progetto.
Copiamo e incolliamo nella Finestra del Codice questo listato:

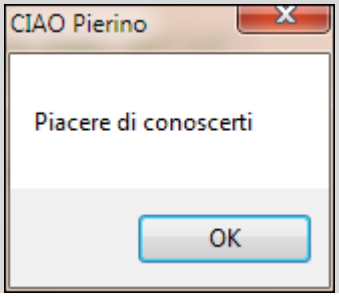
```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load
        Dim NomeUtente As String

ImmissioneNome:
```

```
        ' creazione dell'InputBox
        NomeUtente = InputBox("Come ti chiami?", "CIAO", "Scrivi qui il tuo
nome")

        ' controllo del dato immesso dall'utente:
        ' se il dato è uguale al testo predefinito, l'utente non ha scritto alcun
nome, per cui
        ' il programma torna a visualizzare l'InputBox:

        If NomeUtente = "Scrivi qui il tuo nome" Then
            MsgBox("Non hai scritto alcun nome", MsgBoxStyle.OkCancel
 + ", MsgBoxStyle.Critical, "CIAO")
            GoTo ImmissioneNome

            ' se l'utente ha premuto il tasto Annulla o se nella riga-riquadro
non è scritto alcun nome il programma termina:
        ElseIf NomeUtente = "" Then
            Application.Exit()

            ' altrimenti, se l'utente ha scritto un nome compare un MsgBox di
saluto:
        Else
            MsgBox("Piacere di conoscerti", vbOKOnly, "CIAO " & NomeUtente)
            Application.Exit()

        End If

    End Sub

End Class
```

Notiamo nel codice la presenza di un comando GoTo (= vai a…). Se l'utente fa un *clic* sul pulsante OK nell'InputBox senza modificare la scritta predefinita, il programma torna alla riga ImmisioneNome e ripete la visualizzazione dell'InputBox:

```
            GoTo ImmissioneNome
```

Notiamo anche che se l'utente preme il pulsante Annulla nell'InputBox, il programma riceve un testo vuoto, cioè una stringa uguale a "“”; in questo caso il programma viene fatto terminare.
Nell'immagine seguente vediamo il programma in esecuzione:



Se il testo che il programmatore vuole includere nelle funzioni InputBox o MsgBox è troppo lungo, è possibile scriverlo su più righe utilizzando la sigla **vbCrLf**, che forza il testo del messaggio ad andare a capo.

Ecco la ripetizione dell'esercizio precedente, con il testo dell'InputBox disposto su più righe:

```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load
        Dim NomeUtente As String

ImmissioneNome:
        ' creazione dell'InputBox
        NomeUtente = InputBox("Benvenuto o benvenuta" & vbCrLf & _
                              "tra i programmatori di Visual Basic 2010." &
vbCrLf & _
                              "Per cominciare: come ti chiami?", _
                              "CIAO!", "Scrivi il tuo nome in questa riga")

        ' controllo del dato immesso dall'utente:
        ' se il dato è uguale al testo predefinito, l'utente non ha scritto
alcun nome, per cui
        ' il programma torna a visualizzare l'InputBox:

        If NomeUtente = "Scrivi qui il tuo nome" Then
            MsgBox("Non hai scritto alcun nome", MsgBoxStyle.OkCancel
 + ", MsgBoxStyle.Critical, "CIAO")
            GoTo ImmissioneNome

            ' se l'utente ha premuto il tasto Annulla il programma termina:
        ElseIf NomeUtente = "" Then
            Application.Exit()

            ' se l'utente ha scritto un nome compare un MsgBox di saluto:
        Else
            MsgBox("Piacere di conoscerti", MsgBoxStyle.OKOnly, "CIAO " &
NomeUtente)
            Application.Exit()

        End If

    End Sub

End Class
```

Il risultato compare in questa figura:

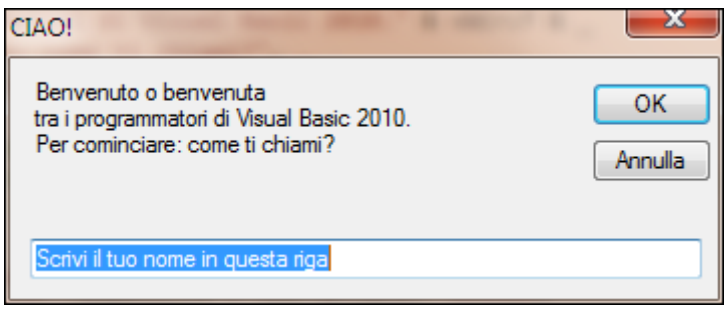


Figura 144: Una finestra InputBox con il testo scritto su più righe.

111: La classe MessageBox.

VB offre al programmatore la possibilità di creare box di messaggi, oltre che con la funzione MsgBox, utilizzando la classe MessageBox.
I due metodi sono identici ed hanno la stessa efficacia, per cui la scelta dipende dalle abitudini e dalle preferenze del programmatore[61].
Vi è una differenza nella disposizione dei parametri:
I parametri della funzione **MsgBox** sono tre, separati da virgole:

- testo del messaggio o della domanda, scritto tra virgolette,
- tipo e numero dei pulsanti + eventuale icona,
- titolo, scritto tra virgolette.

I parametri della classe **MessageBox** sono quattro, disposti in modo diverso:

- testo del messaggio o della domanda, scritto tra virgolette,
- titolo, scritto tra virgolette,
- tipo e numero dei pulsanti,
- eventuale icona.

Il comando più semplice per la creazione di un messaggio con la classe MessageBox è questo:

```
MessageBox.Show ("Sei pronto per iniziare il lavoro?",”Attenzione”)
```

Notiamo che il testo e il titolo sono scritti tra parentesi, separati da una virgola. In questo caso, quando l'utente preme l'unico pulsante visibile, con il testo OK, il programma prosegue senza ricevere alcun risultato dal box.
Per visualizzare nel box più di un pulsante occorre aggiungere il parametro MessageBoxButtons, come nella riga seguente:

```
MessageBox.Show("Vuoi uscire dal programma?", "Attenzione",
MessageBoxButtons.YesNo)
```

Per visualizzare una icona nel messaggio bisogna aggiungere un quarto elemento, il parametro MessageBoxIcon:

```
MessageBox.Show("Vuoi uscire dal programma?", "Attenzione",
MessageBoxButtons.YesNo, MessageBoxIcon.Question)
```

Notiamo che i quattro elementi che contribuiscono a creare il box sono separati da virgole. La scritta del codice, a prima vista laboriosa, è resa molto semplice da funzione **IntelliSense**, che suggerisce passo per passo le diverse opzioni disponibili.
A seconda del pulsante premuto dall'utente, la classe **MessageBox** restituisce un risultato **DialogResult**, diverso per ogni pulsante. Nel caso dell'esempio precedente, il risultato restituito dal MessageBox è **DialogResult.Yes** se l'utente preme il pulsante Sì, **DialogResult.No** se l'utente preme il pulsante No.

[61] Negli esercizi ed esempi di questo manuale abbiamo dato la preferenza a MsgBox.

La creazione di un box con la classe MessageBox con più pulsanti richiede l'uso di una delle sigle riportate in questa tabella:

	Pulsanti visualizzati
MessageBoxButtons.OKOnly	Un solo pulsante, con il testo OK
MessageBoxButtons.OKCancel	Due pulsanti, con il testo OK, Cancella
MessageBoxButtons.AbortRetryIgnore	Tre pulsanti, con il testo Interrompi, Riprova, Ignora
MessageBoxButtons.YesNoCancel	Tre pulsanti, con il testo Sì, No, Cancella
MessageBoxButtons.YesNo	Due pulsanti, con il testo Sì, No
MessageBoxButtons.RetryCancel	Due pulsanti, con il testo Riprova, Cancella

Tabella 20: Sigle per la visualizzazione dei pulsanti in un MessageBox.

La tabella seguente riporta i risultati restituiti dalla classe MessageBox quando l'utente preme un pulsante:

Pulsante premuto dall'utente	Risultato
OK	DialogResult.Ok
Cancella	DialogResult.Cancel
Termina	DialogResult.Abort
Riprova	DialogResult.Retry
Ignora	DialogResult.Ignore
Sì	DialogResult.Yes
No	DialogResult.No

Tabella 21: Risultati restituiti dalla classe MessageBox.

Le icone a disposizione per la classe MessageBox sono quattro; possono essere inserite nel box aggiungendo all'istruzione una di queste sigle:

Stile	Descrizione della icona
MessageBoxIcon.Critical	Attenzione!
MessageBoxIcon.Question	Punto di domanda
MessageBoxIcon.Exclamation	Punto esclamativo
MessageBoxIcon.Information	Informazione

Tabella 22: Elenco delle icone disponibili per la classe MessageBox.

Capitolo 20: FUNZIONI SU NUMERI.

Questo capitolo è dedicato all'analisi di alcune delle funzioni che VB mette a disposizione del programmatore per l'elaborazione di numeri.
Ricordiamo che il termine funzione indica una procedura che riceve dati dal programmatore o dall'utente, li elabora e restituisce il dato dell'elaborazione come una nuova variabile, utilizzabile nello svolgimento del programma.

112: La Classe Math.

Le funzioni che svolgono elaborazioni matematiche sono raccolte nella **classe Math.**
E' dunque necessario richiamare questa classe, nel codice del programma, ogni volta che si ricorre a una di queste funzioni.
In questa tabella riportiamo le funzioni matematiche di uso più comune:

Funzione	Azione
Truncate	(= taglia)
Floor	(= base)
Ceiling	(= tetto)
Abs	Restituisce il valore assoluto del numero indicato tra parentesi, trasformandolo, se occorre, da negativo a positivo.
Max	Restituisce il valore maggiore tra due numeri indicati tra parentesi.
Min	Restituisce il valore minore tra due numeri indicati tra parentesi.

Round	Arrotonda il numero indicato tra parentesi al numero intero più vicino o al numero specificato di posizioni decimali.
Sqrt	Restituisce la radice quadrata del numero indicato tra parentesi.

Tabella 23: Le funzioni della classe Math.

Math.Truncate

La funzione **Math.Truncate** (= taglia) restituisce il valore intero di un numero, eliminando i numeri dopo la virgola.
Vediamo un esempio:

```
Dim A, B, C As Single
A = 10.235
B = Math.Truncate(A)
C = Math.Truncate(A)
```

Risultato:

- A è uguale a 10.235
- B è uguale a 10
- C è uguale a 10.

Math.Floor

La funzione **Math.Floor** (= *base*) restituisce il valore intero di un numero, eliminando i numeri dopo la virgola.
La differenza tra le funzioni **Math.Truncate** e **Math.Floor** si manifesta nell'elaborazione di numeri negativi:

- la funzione **Mah.Truncate** elimina semplicemente i numeri dopo la virgola, mentre
- la funzione **Math.Floor** restituisce il primo numero intero negativo che sta *sotto* il numero negativo con la virgola da elaborare.

Esempi:

```
Dim A, B, C As Single
A = -10.235
B = Math.Truncate(A)
C = Math.Floor(A)
```

Risultato:
- A è uguale a -10,235
- B è uguale a -10
- C è uguale a -11.

Per capire come opera la funzione **Math.Floor** può essere utile osservare l'immagine seguente. Qui vediamo che il valore -10,236, oggetto dell'esempio, si colloca sotto il valore -10 per cui, togliendo i numeri dopo la virgola, il primo numero intero negativo che sta ***sotto*** -10,235 è il numero -11:

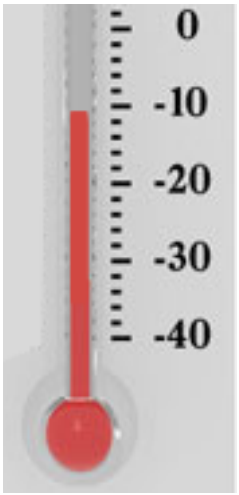


Figura 145: La funzione Math.Floor.

Math.Ceiling

La funzione **Math.Ceiling** (= *tetto*) restituisce il valore intero di un numero, eliminando i numeri dopo la virgola.
La differenza tra le funzioni **Math.Truncate** e **Math.Floor** si manifesta nell'elaborazione di numeri positivi:
- la funzione **Mah.Truncate** elimina semplicemente i numeri dopo la virgola, mentre
- la funzione **Math.Ceiling** restituisce il primo numero intero che sta *sopra* il numero con la virgola da elaborare.

Esempi:

```
Dim A, B, C As Single
A = 10.235
B = Math.Truncate(A)
C = Math.Ceiling(A)
```

Risultato:
- A è uguale a 10,235
- B è uguale a 10

- C è uguale a 11.

Anche per capire come opera la funzione **Math.Ceiling** può essere utile ricorrere ad un'immagine: qui il valore 10,235, oggetto dell'esempio, si colloca sopra il valore 10 per cui, togliendo i numeri dopo la virgola, il primo numero intero che sta *sopra* 10,235 è il numero 11.

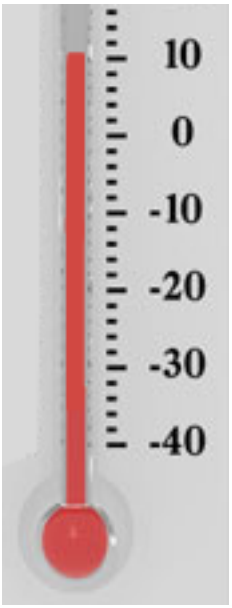


Figura 146: La funzione Math.Ceiling.

Math.Abs

La funzione **Math.Abs** restituisce il valore assoluto di un numero, cioè la sua distanza dallo zero, senza considerare se il numero è positivo o negativo.
Così, ad esempio, il valore assoluto del numero 3 è 3, e il valore assluto del numero -3 è 3.
Ecco un esempio di utilizzo della funzione Math.Abs in una procedura che calcola la differenza tra due numeri:

```
Dim EtàDiLuigi, EtàDiGianni, Differenza As Integer
EtàDiGianni = 12
EtàDiLuigi = 20
Differenza = Math.Abs(EtàDiGianni - EtàDiLuigi)
```

Il risultato aritmetico dell'operazione EtàDiGianni – EtàDiLuigi è uguale a –8 ma, per effetto della funzione Math.Abs, la funzione assegna alla variabile Differenza il numero 8.

Math.Max e Math.Min

Le due funzioni **Math.Max** e **Math.Min** restituiscono, rispettivamente, il numero maggiore o il numero minore tra i due numeri inseriti tra parentesi.

```
Dim PrimoNumero As Integer = 36
Dim SecondoNumero As Integer = 40
Dim NumeroMaggiore As Integer
Dim NumeroMinore As Integer

NumeroMaggiore = Math.Max(PrimoNumero, SecondoNumero)
NumeroMinore = Math.Min(PrimoNumero, SecondoNumero)
```

Risultato: alla variabile NumeroMaggiore è assegnato il valore 40, alla variabile NumeroMinore è assegnato il valore 36.

Math.Round

La funzione **Math.Round** arrotonda il numero indicato tra parentesi al numero intero maggiore o al numero intero minore.
E' possibile effettuare l'arrotondamento a un numero predefinito di decimali, indicato tra parentesi dopo il numero da arrotondare.

```
Dim PrimoNumero As Single = 36.556
PrimoNumero = Math.Round(PrimoNumero)

Dim SecondoNumero As Single = 36.556
SecondoNumero = Math.Round(SecondoNumero, 2)
```

Risultato:

- la variabile PrimoNumero è uguale a 37,
- la variabile SecondoNumero è uguale a 36,56.

Math.Sqrt

La funzione **Math.Sqrt** (= *square root*, radice quadrata) restituisce la radice quadrata del valore immesso tra parentesi.

```
Dim PrimoNumero As Integer = 25
Dim SecondoNumero As Integer
SecondoNumero = Math.Sqrt(PrimoNumero)
```

Risultato: la variabile SecondoNumero è uguale a 5.

113: La classe Random.

La classe Random fornisce le funzioni necessarie per la generazione di numeri casuali.
Se in un programma è previsto il sorteggio di uno o più numeri a caso è necessario scrivere le istruzioni per:

- attivare la classe Random;
- attivare il generatore di numeri casuali;
- scrivere il comando **Next** (= prossimo), indicando i valori minimo e massimo entro i quali si deve collocare il prossimo numero estratto.

Ecco alcuni esempi:

I

```
   ' attiva la classe Random
Dim Sorteggio As New Random
' avvia il generatore di numeri casuali basandolo sul timer del computer
Randomize()

Dim NumeroEstratto As Integer
'sorteggia un numero a caso, entro il 10 (da 0 a 9)
NumeroEstratto = Sorteggio.Next(10)
```

II

```
' attiva la classe Random
Dim Sorteggio As New Random
' avvia il generatore di numeri casuali basandolo sul timer del computer
Randomize()

Dim NumeroEstratto As Integer
'sorteggia un numero a caso da 10 a 14)
NumeroEstratto = Sorteggio.Next(10, 15)
```

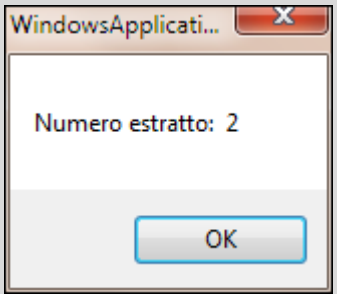
Esercizio 60: La classe Random.

Creiamo un programma per sorteggiare numeri a caso, da 1 a 90.
Apriamo un nuovo progetto e inseriamo nel Form1 un pulsante Button1.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' attiva la classe Random
        Dim Sorteggio As New Random
        ' avvia il generatore di numeri casuali basandolo sul timer del computer
        Randomize()
```

```
        Dim NumeroEstratto As Integer
        'sorteggia un numero a caso da 1 a 90)
        NumeroEstratto = Sorteggio.Next(1, 91)

        MsgBox("Numero estratto:  " & NumeroEstratto.ToString)

    End Sub

End Class
```

Notiamo che il sorteggio di un nuovo numero è avviato dal comando Next(); tra parentesi si trovano i limiti inferiori e superiore (aumentato di una unità) entro i quali deve essere effettuato il sorteggio.
Ecco un'immagine del programma in esecuzione



Nel prossimo esercizio, che continua e completa il programma Bandiere (I)[62], vedremo l'uso dell'estrazione di un numero casuale per la creazione di un quiz.

Esercizio 61: Bandiere (II).

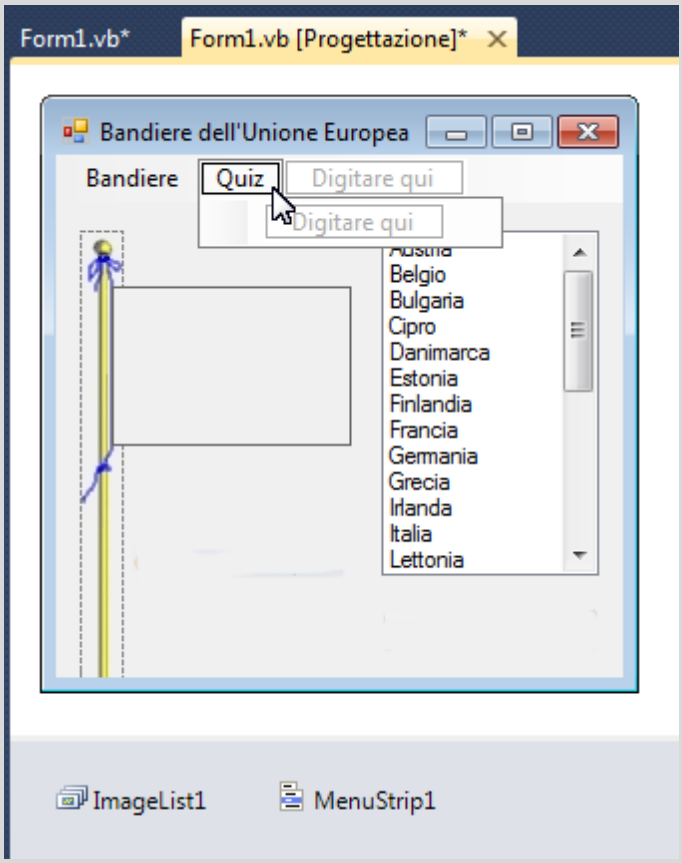
Nel programma **Bandiere (I)**, a un *clic* del mouse sul nome di una nazione corrisponde la visualizzazione di una bandiera all'interno del controllo PictureBox.
In questo esercizio completiamo il programma inserendovi la funzione contraria: verrà prima visualizzata una bandiera e l'utente del programma dovrà cliccarne il nome corrispondente nel controllo **ListBox**.
Avremo dunque un programma che utilizzerà gli stessi elementi in due modi diversi:

- la visualizzazione delle bandiere (la chiameremo opzione **Bandiere**);
- un quiz sul riconoscimento delle bandiere (la chiameremo opzione **Quiz**).

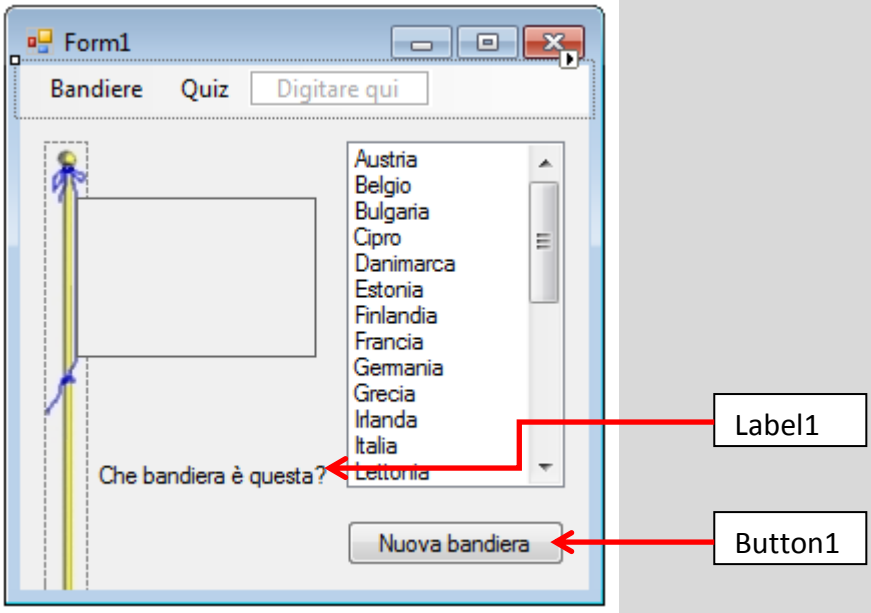
La scelta dell'uno o dell'altro modo verrà effettuata dall'utente cliccando un menu, mediante il quale l'utente potrà scegliere se vedere le bandiere o giocare con il quiz.
Le proprietà **Text** di questi due menu saranno appunto, rispettivamente, **Bandiere** e **Quiz**.

[62] Esercizio 29: Bandiere (I). 287.

Apriamo VB e, invece di aprire un nuovo progetto, apriamo il progetto **Bandiere Unione Europea** che abbiamo salvato in precedenza.
Inseriamo nel **Form1** un controllo **MenuStrip** (striscia di menu) e scriviamo il testo di due menu: **Bandiere** e **Quiz**:



Inseriamo nella parte inferiore del form un **pulsante** che comanderà la scelta di una nuova bandiera e una **label** in cui l'utente del programma potrà controllare l'esattezza delle sue risposte.
La proprietà **Text** del pulsante **Button1** è "Nuova bandiera**"**.
La proprietà **Text** dell'etichetta **Label1** è "Che bandiera è questa?".
I due controlli all'inizio del programma non sono visibili, dunque la loro proprietà **Visible** è impostata **= False**.



Completato l'interfaccia del programma, passiamo alla scrittura del codice.
Riportiamo di seguito il codice completo del programma finito: la lettrice o il lettore possono copiarlo e incollarlo nella Finestra del Codice.
Le spiegazioni necessarie alla sua comprensione sono inserite nel codice stesso[63].

```
Public Class Form1

    ' crea una variabile di nome NumeroBandiera, destinata a memorizzare un
    ' numero intero sorteggiato da 0 a 27, corrispondente alla bandiera
    ' sorteggiata di volta in volta::
    Dim NumeroBandiera As Integer

    ' Crea una variabile di nome Quiz, di tipo Boolean, cioè di tipo VERO/FALSO:
    ' se l'utente clicca il menu Quiz la variabile Quiz è uguale a VERO,
    ' se l'utente clicca il menu Bandiere la variabile Quiz è uguale a FALSO
    Dim Quiz As Boolean
```

```
    Private Sub ListBox1_SelectedIndexChanged() Handles
ListBox1.SelectedIndexChanged

        ' il clic su un item all'interno del ListBox ha effetti diversi, a
seconda del fatto che
        ' il programma sia in modo Bandiere o in modo Quiz.
        ' Se il programma NON E' IN MODO QUIZ
        ' il programma visualizza l'immagine nel controllo PictureBox:

        If Quiz = False Then
            PictureBox1.Image = ImageList1.Images(ListBox1.SelectedIndex)

            'altrimenti, cioè se il programma E' IN MODO QUIZ
        Else
            ' viene attivata la procedura ControllaRisposta()
```

[63] Ricordiamo che le righe di codice che iniziano con il segno dell'apostrofo contengono note che il programmatore inserisce nel codice a suo uso esclusivo. Esse compaiono nel codice scritte in verde. VB, nell'esecuzione del programma, le ignora.

```
            Call ControllaRisposta()
        End If

    End Sub
```

```
    Private Sub ControllaRisposta()

        ' questa procedura controlla la risposta dell'utente:
        ' se il numero dell'item cliccato nella ListBox corrisponde al numero
della bandiera
        ' sorteggiata, allora nella Label1 viene scritto che la risposta è
esatta:

        If ListBox1.SelectedIndex = NumeroBandiera Then
            Label1.Text = "Esatto!"

            ' altrimenti nella Label1 viene scritto che la risposta è sbagliata:
        Else
            Label1.Text = "No, prova ancora."
        End If

    End Sub
```

```
    Private Sub Button1_Click() Handles Button1.Click

        ' questo pulsante viene visualizzato quando il programma è in modo Quiz.
        ' Esso comanda il sorteggio di una nuova bandiera, che viene effettuato
in due fasi:

        ' prima fase:
        ' crea un oggetto di nome Sorteggio e di tipo Random (casuale):
        ' (questo oggetto è destinato a contenere un numero casuale tra 0 e 27:
        Dim Sorteggio As New Random
        ' avvia il generatore di numeri casuali basandolo sul timer del computer
        Randomize()

        ' seconda fase:
        ' assegna alla variabile NumeroBandiera il numero casuale contenuto
nell'oggetto Sorteggio.
        ' Notare che il sorteggio di un numero viene avviato con il comando Next.
        ' Il numero tra parentesi indica il valore massimo che il numero
sorteggiato potrà avere.
        ' Per evitare errori in questo caso si è usato il numero di immagini
presenti nel componente ImageList1
        NumeroBandiera = Sorteggio.Next(ImageList1.Images.Count)

        ' assegna al controllo PictureBox1 l'immagine presente nella ImageList
con il numero corrispondente al numero sorteggiato:
        PictureBox1.Image = ImageList1.Images(NumeroBandiera)

        ' riscrive il contenuto della Label1:
        Label1.Text = "Che bandiera è questa?"

    End Sub
```

```
    Private Sub Bandiere1ToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Bandiere1ToolStripMenuItem.Click

        ' il clic sul menu Bandiere disattiva il modo Quiz, per cui:
```

```
        ' la variabile Quiz viene impostata come False
        ' vengono nascosti all'utente il Button1 e la Label1
        ' e viene eliminata dal controllo PictureBox1 l'immagine eventualmente
presente:
        Quiz = False
        Button1.Visible = False
        Label1.Visible = False
        PictureBox1.Image = Nothing

    End Sub
```

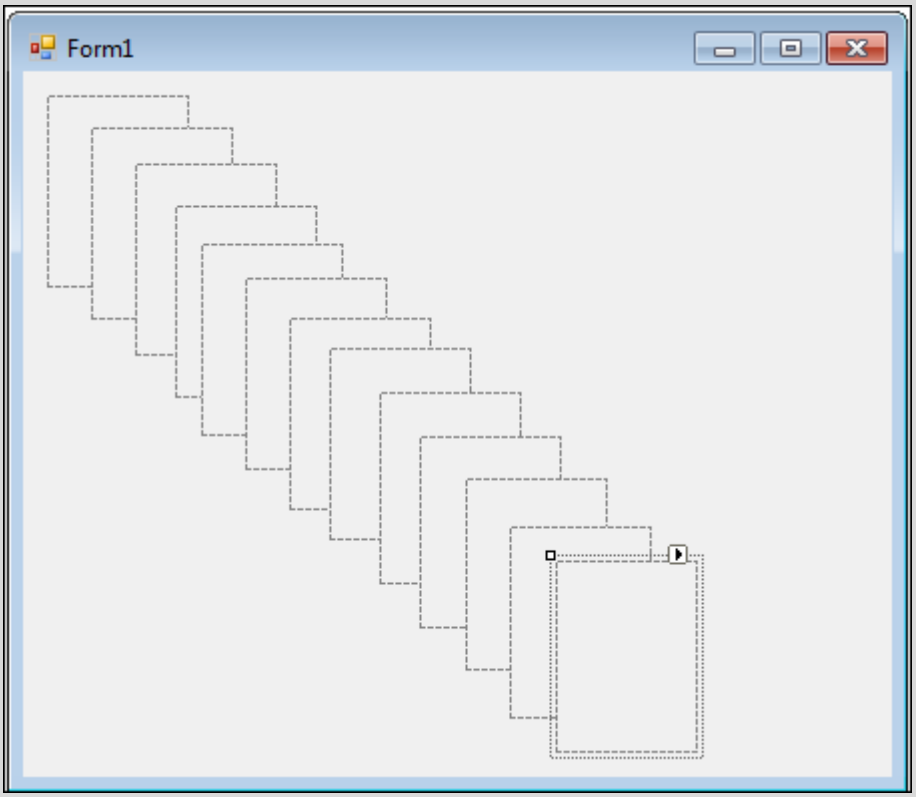
```
    Private Sub Bandiere2ToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Bandiere2ToolStripMenuItem.Click

        ' il clic sul menu Quiz attiva il modo Quiz, per cui:
        ' la variabile Quiz viene impostata come True
        ' vengono visualizzati il Button1 e la Label1
        ' e viene attivata la procedura Button1_Click per il sorteggio
        ' di una bandiera:
        Quiz = True
        Button1.Visible = True
        Label1.Visible = True
        Button1.PerformClick()

    End Sub

End Class
```

Notiamo nel listato il comando `Button1.PerformClick()`. Questo comando genera un *clic* sul pulsante Button1, come se questo fosse stato premuto con un *clic* del mouse. Con `Button1.PerformClick()`, dunque, senza che vi sia stato un reale *clic* del mouse, si genera un evento Button1.Click e si attiva la relativa procedura Button1_Click.
Ecco un'immagine del programma in esecuzione in modo **Quiz**:

Nel prossimo esercizio vedremo l'abbozzo di un gioco di carte: nel Form1 sono visualizzate in ordine 13 carte. Con un *clic* del mouse su un pulsante, si attiva una procedura che mescola queste carte, cambiandone in modo casuale la posizione.

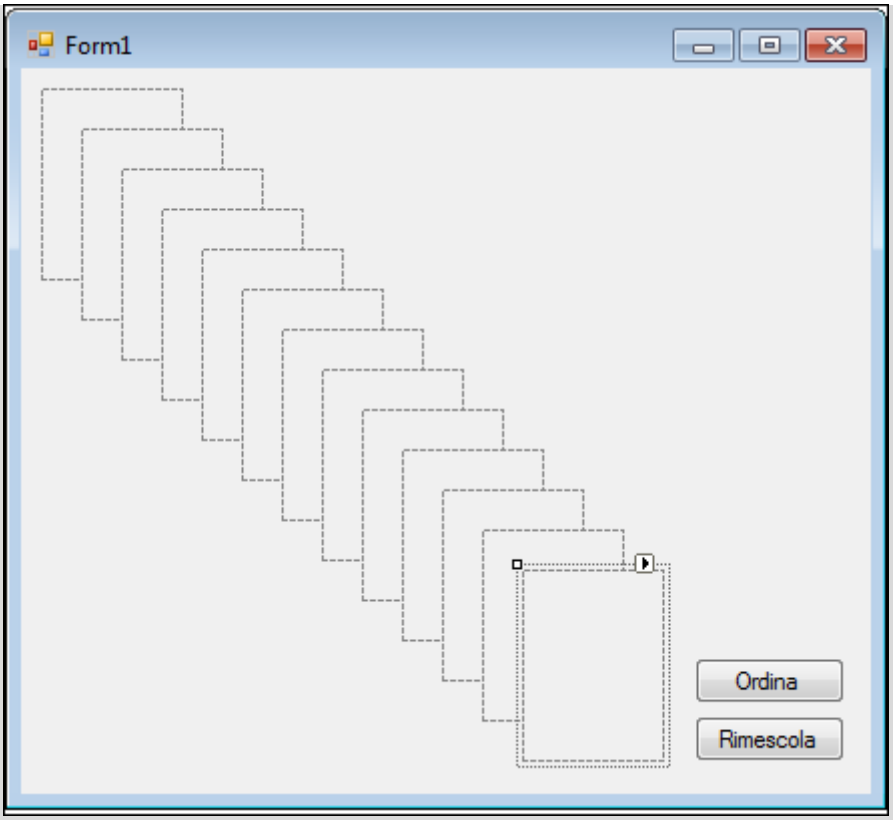
Esercizio 62: Mescolare le carte.

Apriamo un nuovo progetto. Gli diamo il nome "Mescolare le carte".
Diamo al Form1 queste dimensioni:
Size = 450; 400
Inseriamo nel Form1 un controllo PictureBox **in alto a sinistra**, di queste dimensioni:
Size = 71; 96
Ora copiamo e incolliamo questo controllo PictureBox, a scalare, sino ad avere nel Form1 **tredici** controlli PictureBox, come in questa immagine (il PictureBox creato per ultimo è quello più in basso):

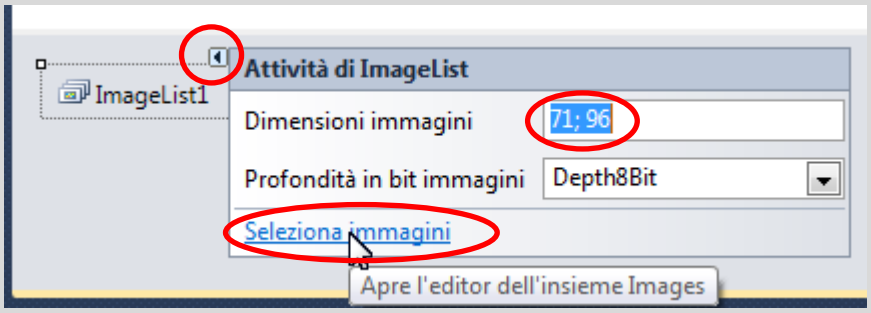


Inseriamo nel Form1, in basso a destra, due pulsanti Button con la proprietà Text impostata in questo modo:

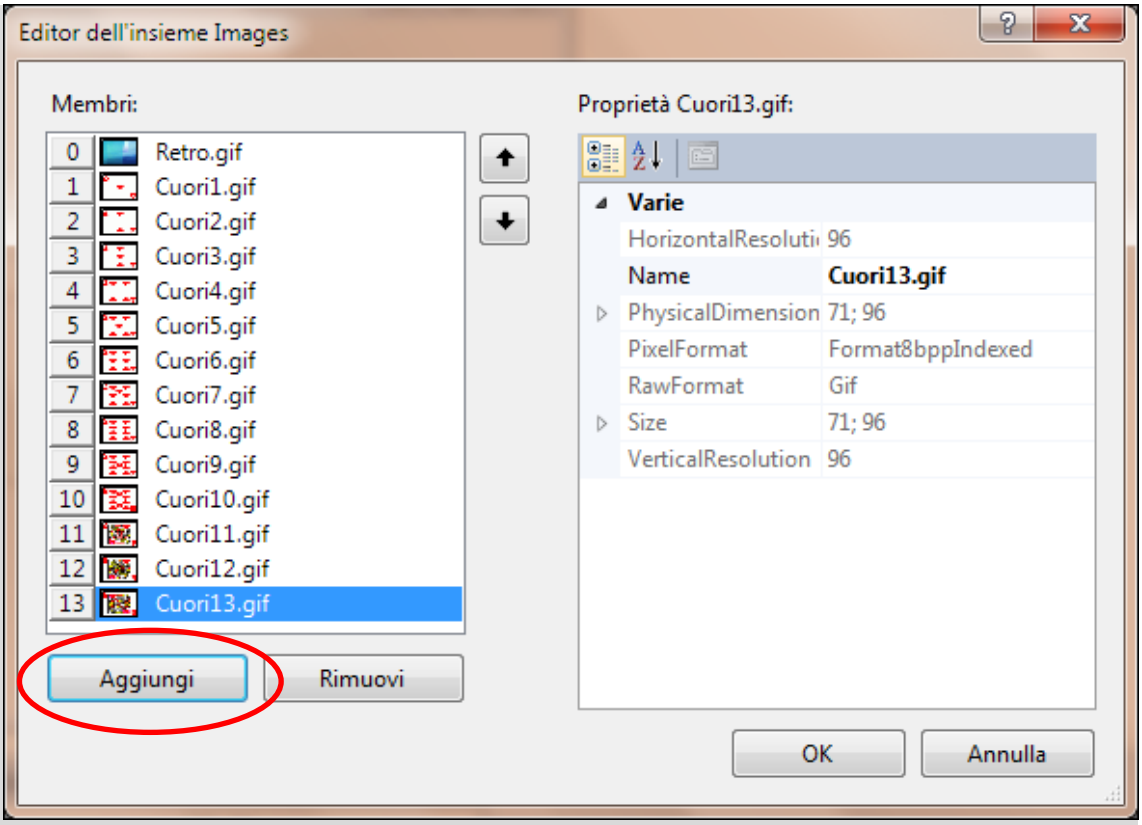
- Button1: **Text = Ordina**
- Button2: **Text = Rimescola**



Ora inseriamo nel progetto un componente ImageList, destinato a contenere le immagini delle carte che verranno visualizzate dal programma.
Facciamo un *clic* con il mouse sul pulsante con la freccina nera in alto a destra nel componente e apriamo la finestra della **Attività di ImageList**:



Nel riquadro **Dimensioni immagini** scriviamo **71; 96**.
Facciamo poi un *clic* su **Seleziona immagini**: apriamo così la finestra dell'**Editor dell'insieme Images,** dove inseriamo in ordine l'immagine **Retro.gif** e le tredici immagini con le carte di Cuori, che si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Carte**:



Notiamo che l'**Editor dell'insieme Images** assegna un numero progressivo alle immagini che vi abbiamo inserito: la prima immagine, Retro.gif, ha il numero 0, le tredici immagini delle carte di Cuori sono numerate da 1 a 13.

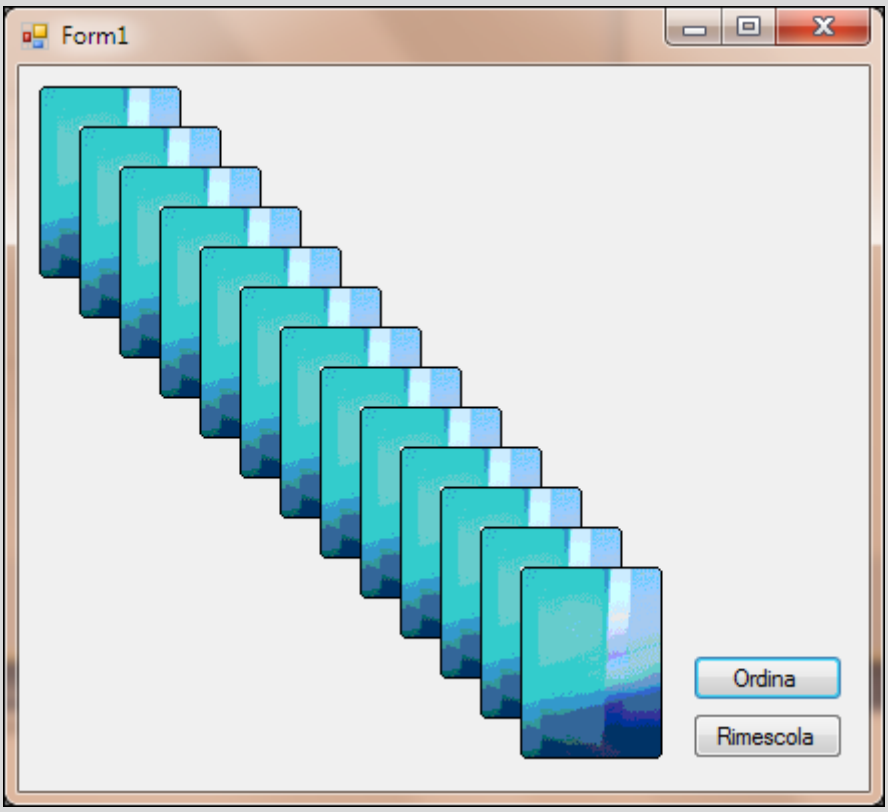
Possiamo ora scrivere nella Finestra del Codice la procedura che gestisce il caricamento in memoria del Form1 all'avvio del programma:

```
Public Class Form1

    ' la variabile NumeroCarta è una matrice destinata a memorizzare la sequenza
delle 13 carte
    Dim NumeroCarta(13) As Integer
    Dim Contatore As Integer
```

```
    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs)
Handles MyBase.Load

        ' questo ciclo For Each... Next passa in rassegna i controlli
PictureBox presenti nel Form1
        ' e visualizza in ognuno di essi l'immagine che nella ImageList ha il
n. 0 (il retro di una carta):

        For Each Control In Me.Controls
            If TypeOf Control Is PictureBox Then Control.Image =
ImageList1.Images(0)
        Next

    End Sub
```

```
End Class
```

Mandiamo in esecuzione il programma e vediamo l'effetto prodotto da questa procedura:



Fermiamo l'esecuzione del programma e torniamo alla Finestra del Codice.
Ora ci dedichiamo al primo dei due pulsanti: la procedura che gestisce l'evento del *clic* del mouse su questo pulsante è incaricata di visualizzare le 13 carte, in ordine, dall'Asso al Re:

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' al clic sul pulsante Ordina crea la sequenza ordinata delle carte, dal
n. 1 al n. 13

        For Contatore = 1 To 13
            NumeroCarta(Contatore) = Contatore
        Next

        ' e passa alla procedura che visualizza le carte:
        Call MostraLeCarte()

    End Sub
```

```
    Private Sub MostraLeCarte()
```

```
        ' questo ciclo For Each... Next passa in rassegna i controlli PictureBox
presenti nel Form1
        ' e visualizza in ognuno di essi l'immagine che nella ImageList ha il n.
0 (il retro di una carta):

        Contatore = 1
        For Each Control In Me.Controls
            If TypeOf Control Is PictureBox Then
                Control.Image = ImageList1.Images(0)
                Control.refresh()
                Call Pausa(50)
                Contatore += 1
            End If
        Next

        ' questo ciclo For Each... Next passa di nuovo in rassegna i controlli
PictureBox
        ' e visualizza in ognuno di essi un'immagine presa dalla ImageList
        ' secondo la sequenza contenuta nella matrice NumeroCarta:

        Contatore = 1
        For Each Control In Me.Controls
            If TypeOf Control Is PictureBox Then
                Control.Image = ImageList1.Images(NumeroCarta(Contatore))
                Control.refresh()
                Call Pausa(50)
                Contatore += 1
            End If
        Next

    End Sub
```

```
 Private Sub Pausa(DurataPausa As Integer)
        ' Questa procedura è chiamata in causa dai tre pulsanti:
        ' ognuno dei tre pulsanti passa a questa procedura la sua durata della
pausa
        ' espressa in millisecondi:

        Dim OrarioFinePausa As Date = Date.Now.AddMilliseconds(DurataPausa)
        Do Until Date.Now > OrarioFinePausa

         ' Il ciclo non contiene alcun comando:
         ' esso ruota semplicemente su se stesso sino a quando la data
corrente è maggiore della variabile OrarioFinePausa.

         Application.DoEvents()

        Loop

    End Sub
```

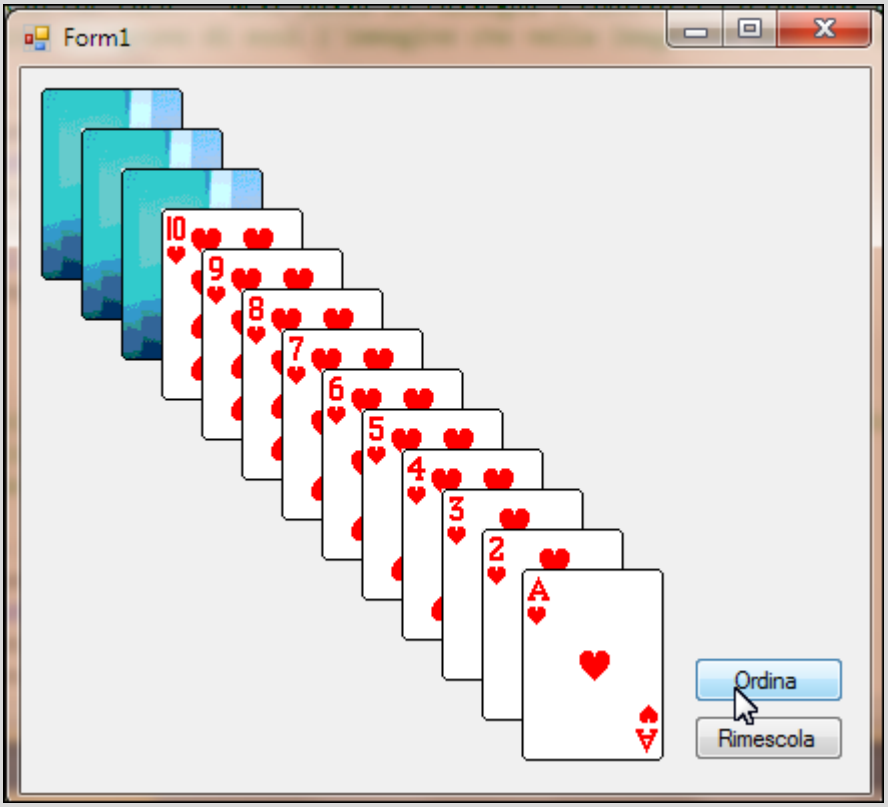
Notiamo che la procedura **Button1_Click** crea una serie ordinata di carte, impostando la matrice di variabili NumeroCarta in questo modo:

- NumeroCarta(1) = 1
- NumeroCarta(2) = 2
- …
- Numero Carta(13) = 13

Fatto questo, la procedura **Button1_**Click chiama in causa la procedura **MostraLeCart**e, che passa in rassegna i controlli PictureBox presenti nel Form1 e visualizza in ognuno di essi:

- dapprima l'immagine che nella ImageList ha il n. 0 (il retro di una carta);
- poi le 13 immagini con le carte di Cuori, richiamate dalla ImageList secondo la sequenza delle variabili della matrice NumeroCarta.

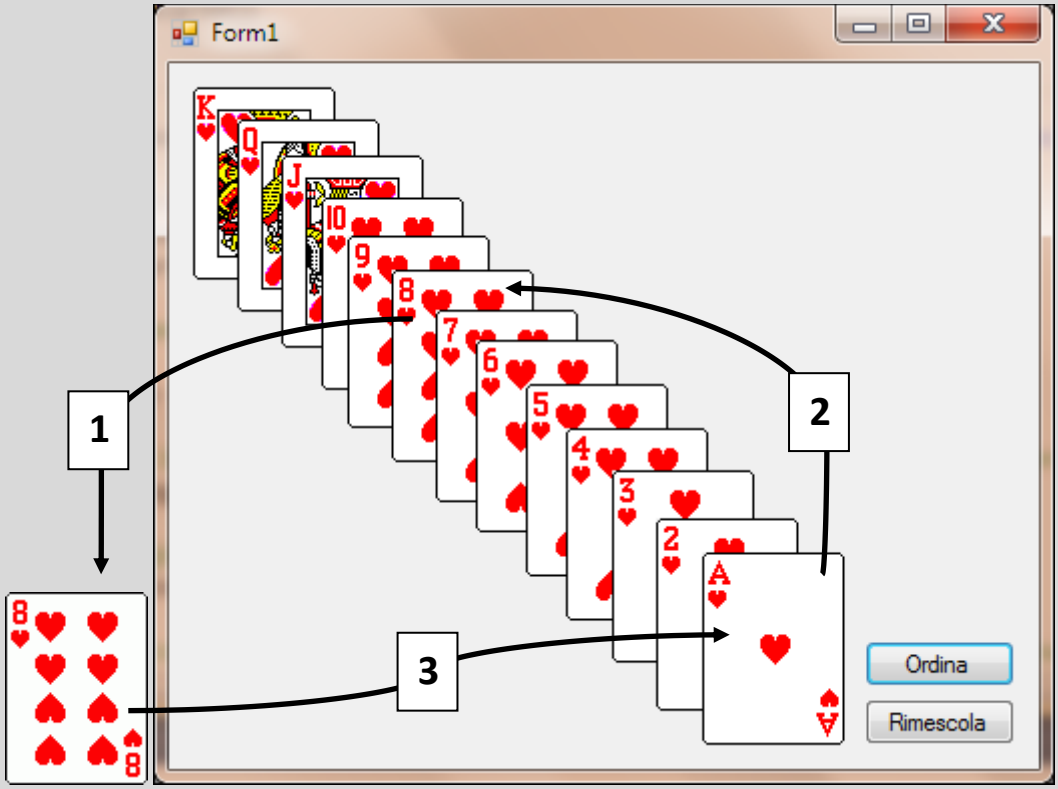
Notiamo che la procedura **MostraLeCarte**, a sua volta, chiama in causa la procedura **Pausa**, per rallentare la visualizzaizone delle carte nei 13 controlli PictureBox.
Questa procedura Pausa è presa pari pari dall'Esercizio 52: Creazione di una pausa in un programma. a pag. 419.
Ecco un'immagine del programma in esecuzione, dopo un *clic* sul pulsante **Ordina**:



Rest da scrivere la procedura più complessa: quella che gestisce l'evento del *clic* del mouse sul pulsante **Rimescola**.
Questa procedura sostanzialmente crea una nuova serie per la matrice di variabili NumeroCarta, in cui i valori di ogni variabile sono sorteggiati a caso, con un numero da 1 a 13:

- NumeroCarta(1) = numero casuale da 1 a 13;
- NumeroCarta(2) = numero casuale da 1 a 13;
- …
- Numero Carta(13) = numero casuale da 1 a 13.

Per evitare che a variabili NumeroCarta diverse possa capitare lo stesso numero casuale, la procedura effettua tre operazioni in successione, che vediamo visualizzate in questa immagine:

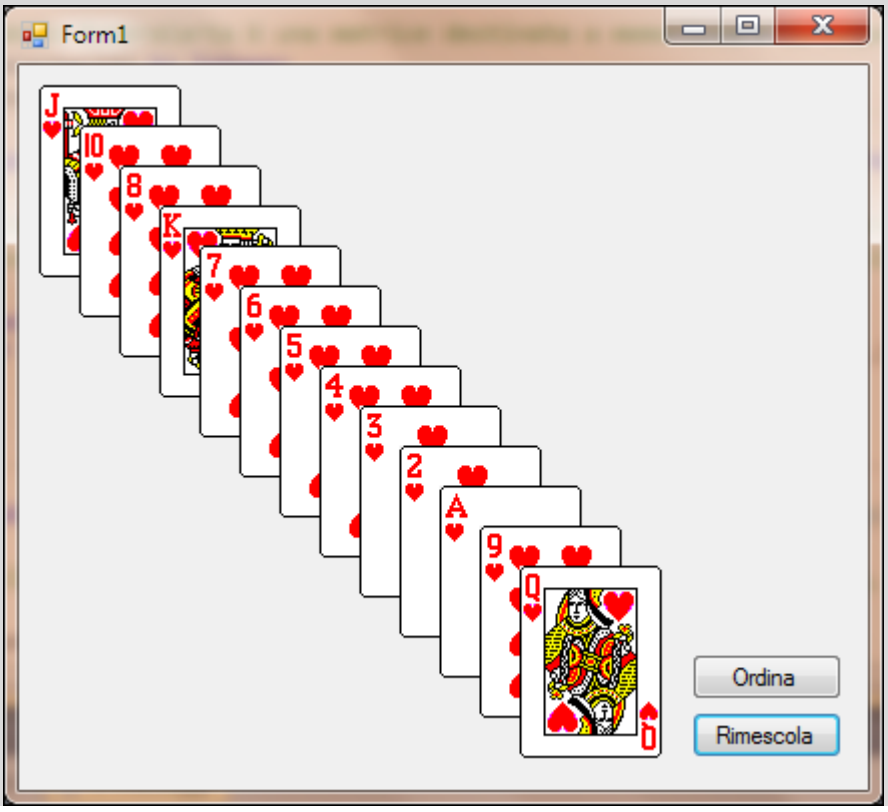


La procedura parte dalla prima carta, cioè dalla variabile NumeroCarta(1).
Per questa variabile, che in una sequenza ordinata contiene la carta n. 1, viene sorteggiata una nuova carta, supponiamo la carta n. 8.
Prima operazione: il programma sposta la carta n. 8 in una CartaInAttesa.
Seconda operazione: il programma sposta la carta n. 1 nella carta n. 8.
Terza operazione: il programma sposta la CartaInAttesa nella carta n. 1.
Queste operazioni sono ripetute per tutte le 13 carte, creando una nuova serie di carte, serie che è poi utilizzata nella procedura MostraLeCarte per visualizzare le carte in ordine casuale.

```
    Private Sub Button2_Click(sender As System.Object, e As System.EventArgs)
Handles Button2.Click

        Dim Contatore As Integer
        Dim NuovaCarta As Integer
        Dim CartaInAttesa As Integer

        Dim Sorteggio As New Random
        ' avvia il generatore di numeri casuali basandolo sul timer del computer
        Randomize()

        For Contatore = 1 To 13
            ' Prende in esame una carta alla volta, partendo dalla carta con il
n. 1, sino alla carta con il n. 13:
            ' sorteggia il numero di una nuova carta, da 1 a 13:
            NuovaCarta = Sorteggio.Next(1, 13)
```

```
            ' memorizza nella CartaInAttesa il numero della nuova carta
sorteggiata:
            CartaInAttesa = NumeroCarta(NuovaCarta)

            ' mette al posto della nuova carta sorteggiata la carta in esame
            NumeroCarta(NuovaCarta) = NumeroCarta(Contatore)

            ' infine, mette nella carta in esame la CartaInAttesa:
            NumeroCarta(Contatore) = CartaInAttesa
        Next

        ' dopo avere creato una nuova sequenza di carte, in modo casuale,
        ' passa alla procedura che visualizza le carte:
        Call MostraLeCarte()

    End Sub
```

Ecco un'immagine del programma in esecuzione, in versione completa, con la funzione di rimescolamento delle carte:



Capitolo 21: FUNZIONI SU STRINGHE DI TESTO.

In questo capitolo vedremo gli strumenti che VB offre per analizzare e modificare le stringhe di testo durante l'esecuzione di un programma.
Le stringhe di testo sono catene di caratteri alfanumerici, cioè di caratteri alfabetici (A/Z, a/z) e di caratteri numerici (0/9).
I caratteri numerici eventualmente presenti in una stringa di testo non sono considerati da VB come numeri ma come elementi di testo, alla stregua delle lettere dell'alfabeto.
Nella Finestra del Codice di un nuovo programma, proviamo a scrivere queste due righe di comandi:

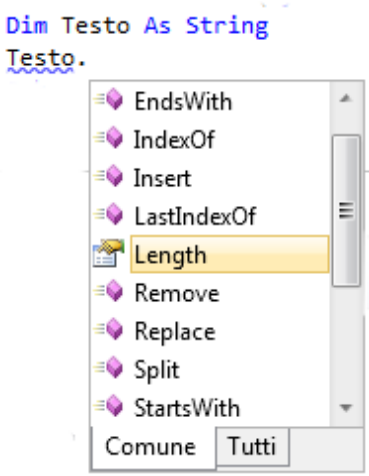


Figura 147: Proprietà e funzioni di una stringa di testo.

Notiamo che appena scriviamo il punto dopo la variabile Testo, IntelliSense mostra l'elenco delle proprietà e delle funzioni disponibili per le variabili stringa.
La proprietà Length (= lunghezza) appare preselezionata; sotto e sopra questa proprietà compare un elenco di funzioni che esamineremo in dettaglio in questo capitolo, con l'aggiunta di alcune funzioni che non compaiono in questo elenco me che sono tuttavia utili alla manipolazione di stringhe di testo.
Partiamo dalla proprietà che compare già evidenziata nell'elenco fornito da IntelliSense: la proprietà Length.

String.Length

La proprietà **Length** (= lunghezza) restituisce il numero di caratteri (più propriamente: il numero di oggetti **Char**) che si trovano in una stringa di testo.
In un testo con spazi tra le parole, la proprietà Length restituisce il numero di battute (comprensivo dei caratteri e degli spazi tra le parole). In questo esempio, dunque:

```
Dim Testo1 As String = "Paolo"
Dim Testo2 As String = "Paolo Rossi"
```

la proprietà Testo1.Lenght è uguale a 5, la proprietà Testo2.Length è uguale a 11.
L'uso di questa proprietà è utile, in particolare, quando occorre tenere sotto controllo la lunghezza di una parola o di un testo scritto dall'utente.
Supponiamo che all'utente del programma sia richiesto di scrivere in un TextBox una parola o una frase della lunghezza massima di un certo numero di caratteri: l'uso della proprietà Length consente di controllare la lunghezza del testo, per avviare eventuali operazioni di correzione:

- tagliare il testo riducendolo alle dimensioni desiderate, oppure
- visualizzare un messaggio perché l'utente riduca il testo.

Esercizio 63: La proprietà String.Length

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo TextBox, un controllo Label e un pulsante Button come in questa immagine:

Proprietà del TextBox1:

- **MultiLine = True**

- **Text = Scrivi qui un testo**

Proprietà del pulsante Button1:

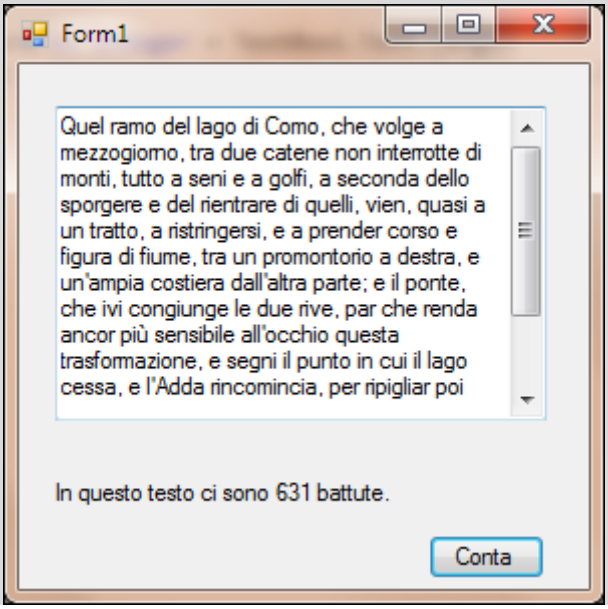
- **Text = Conta**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim Contatore As Integer = TextBox1.Text.Length
        Label1.Text = "In questo testo ci sono " & Contatore.ToString & "
battute."
    End Sub

End Class
```

Notiamo nel codice che la variabile Contatore è di tipo numerico (Integer). Per visualizzarla correttamente nel controllo Label, o nella proprietà Text di qualunque altro controllo, è consigliabile trasformarla in testo con la funzione ToString: Contatore.ToString

Ecco un'immagine del programma in esecuzione[64]:



[64] Il testo utilizzato in questa occasione, e in molti altri esercizi ed esempi, è l'incipit de "I Promessi Sposi", che si trova nella cartella **Documenti \ A scuola con VB \ Testi**.

String.Contains

La funzione **String.Contains** verifica se un testo da analizzare contiene una determinata parola, o un testo, o un gruppo di caratteri.
String.Contains restituisce una variabile di tipo Boolean (vero/falso): se la stringa da cercare esiste, la variabile è True, in caso contrario la variabile è False.
Esempio:

```
        Dim TestoDaAnalizzare As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        Dim EsitoAnalisi As Boolean = TestoDaAnalizzare.Contains("Brontolo")
        MsgBox("In questo caso EsitoAnalisi è " & EsitoAnalisi)
```

In questo caso la variabile EsitoAnalisi è uguale a True:

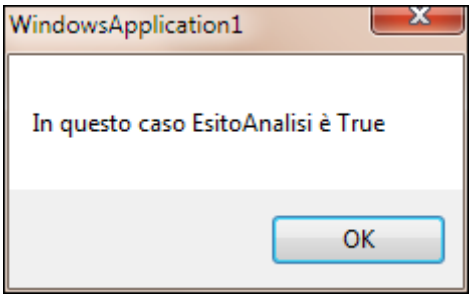


Figura 148: La funzione String.Contains.

Ricordiamo che la funzione **String.Contains** e le altre funzioni di ricerca o di comparazione di testi che vedremo più avanti sono *case sensitive*, cioè fanno distinzione tra le lettere maiuscole e le lettere minuscole, per cui, ad esempio, il nome “Brontolo” non è uguale a “brontolo”.

String.IndexOf

String.LastIndexOf

La funzione **String.IndexOf** (= indice di…) indica la posizione iniziale in cui si trovano una parola o un testo o un gruppo di caratteri contenuti nel testo da analizzare.
Esempio:

```
        Dim TestoDaAnalizzare As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        Dim EsitoAnalisi As Integer = TestoDaAnalizzare.IndexOf("Cucciolo")
        MsgBox("Cucciolo si trova nella posizione " & EsitoAnalisi.ToString)
```

In questo caso la funzione String.IndexOf restituisce il valore 9:

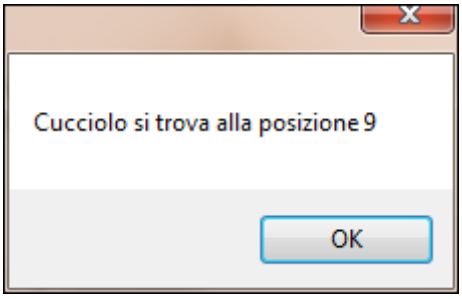


Figura 149: La funzione String.IndexOf.

La funzione **String.LastIndexOf** (= ultimo indice di…) indica invece la posizione iniziale dell'ultima occorrenza di una parola o un di testo o un gruppo di lettere all'interno del testo da analizzare.
Esempio:

```
        Dim TestoDaAnalizzare As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        Dim EsitoAnalisi As Integer = TestoDaAnalizzare.LastIndexOf("lo")
        MsgBox("L'ultima sillaba 'lo' si trova nella posizione " &
EsitoAnalisi.ToString)
```

In questo caso la funzione String.LastIndexOf restituisce il valore 55:

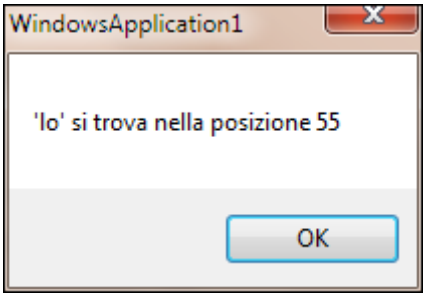


Figura 150: La funzione String.LastIndexOf.

String.StartsWith

String.EndsWith

Le due funzioni **String.StartsWith** (= inizia con…) e **String.EndsWith** (= termina con…) verificano se una parola o testo o un gruppo di caratteri si trovano all'inizio o alla fine del testo da analizzare.
Esempi:

```
        Dim TestoDaAnalizzare As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
```

```
        Dim EsitoAnalisi As Boolean = TestoDaAnalizzare.StartsWith("Brontolo")
        MsgBox("Brontolo è all'inizio dell'elenco = " & EsitoAnalisi)
```

In questo caso la variabile EsitoAnalisi sarà uguale a False perché la parola "Brontolo" non si trova all'inizio del testo da analizzare:

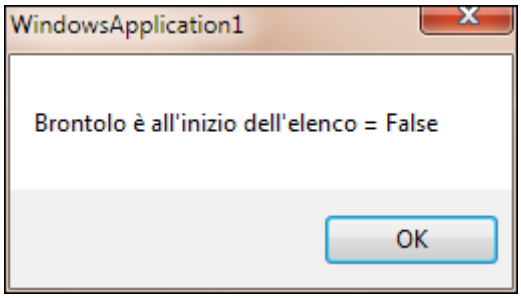


Figura 151: La funzione String.StartsWith.

```
        Dim TestoDaAnalizzare As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        Dim EsitoAnalisi As Boolean = TestoDaAnalizzare.EndsWith("olo.")
        MsgBox("L'elenco termina con le lettere olo. = " & EsitoAnalisi)
```

In questo caso la variabile EsitoAnalisi sarà uguale a True perché il gruppo di caratteri "olo." si trova alla fine del testo da analizzare:

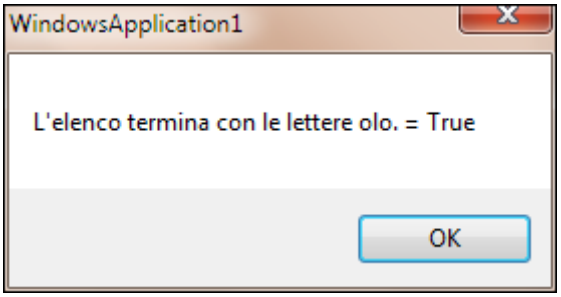
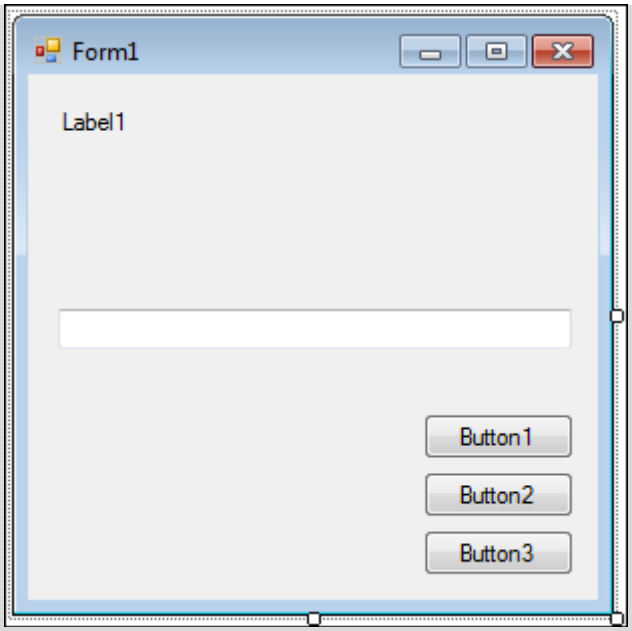


Figura 152: La funzione String.EndsWith.

Esercizio 64: Le funzioni Contains, IndexOf, StartsWith, EndsWith.

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo Label1, un controllo TextBox e tre pulsanti Button come in questa immagine:



Copiamo e incolliamolo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load() Handles Me.Load

        Button1.Text = "Contains"
        Button2.Text = "StartsWith"
        Button3.Text = "EndsWith"
        Label1.Text = "Mammolo, Cucciolo, Brontolo, Eolo, " & vbCrLf &
"Dotto, Gongolo, Pisolo."

        TextBox1.Text = "Scrivi una parola da cercare."
        TextBox1.SelectionStart = 0
        TextBox1.SelectionLength = TextBox1.Text.Length
        TextBox1.HideSelection = False
        TextBox1.TabIndex = 0

    End Sub
```

```
    Private Sub Button1_Click() Handles Button1.Click

        Dim TestoDaAnalizzare As String = Label1.Text
        Dim TestoDaCercare As String = TextBox1.Text

        Dim EsitoAnalisi As Boolean =
TestoDaAnalizzare.Contains(TestoDaCercare)
        MessageBox.Show(EsitoAnalisi.ToString, Button1.Text & " " &
TestoDaCercare)

        If EsitoAnalisi = True Then
            Dim Posizione As Integer =
TestoDaAnalizzare.IndexOf(TestoDaCercare)
            MessageBox.Show(TestoDaCercare & " si trova alla posizione " &
Posizione)
```

```
        End If

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        Dim TestoDaAnalizzare As String = Label1.Text
        Dim TestoDaCercare As String = TextBox1.Text

        Dim EsitoAnalisi As Boolean =
TestoDaAnalizzare.StartsWith(TestoDaCercare)
        MessageBox.Show(EsitoAnalisi.ToString, Button2.Text & " " &
TestoDaCercare)

    End Sub
```

```
    Private Sub Button3_Click() Handles Button3.Click

        Dim TestoDaAnalizzare As String = Label1.Text
        Dim TestoDaCercare As String = TextBox1.Text

        Dim EsitoAnalisi As Boolean =
TestoDaAnalizzare.EndsWith(TestoDaCercare)
        MessageBox.Show(EsitoAnalisi.ToString, Button3.Text & " " &
TestoDaCercare)

    End Sub

End Class
```

Mandiamo in esecuzione il programma e cliccando i tre pulsanti proviamo i diversi esiti delle funzioni Contains, IndexOf, StartsWith e EndsWith.

String.Insert

La funzione **String.Insert** (= inserisci…) inserisce una parola o testo o un gruppo di caratteri in un'altra stringa di testo, partendo da una determinata posizione.
Esempio:

```
        Dim TestoOriginale As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        TestoOriginale = TestoOriginale.Insert(0, "Biancaneve, ")
```

La funzione inserisce la parola "Biancaneve, " nella stringa TestoOriginale, partendo dalla posizione 0. Risultato: dopo l'inserimento, la stringa TestoOriginale è "Biancaneve, Mammolo, Cucciolo, Brontolo, Eolo, Dotto, Gongolo, Pisolo."

String.Remove

La funzione **String.Remove** (= elimina…) toglie un determinato numero di caratteri da un testo, partendo da una determinata posizione.
Esempio:

```
        Dim TestoOriginale As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        TestoOriginale = TestoOriginale.Remove(0, 9)
```

La funzione toglie dalla stringa TestoOriginale 9 caratteri, partendo dalla posizione 0. Risultato: dopo la rimozione dei primi 9 caratteri, la stringa TestoOriginale è: "Cucciolo, Brontolo, Eolo, Dotto, Gongolo, Pisolo."

String.Substring

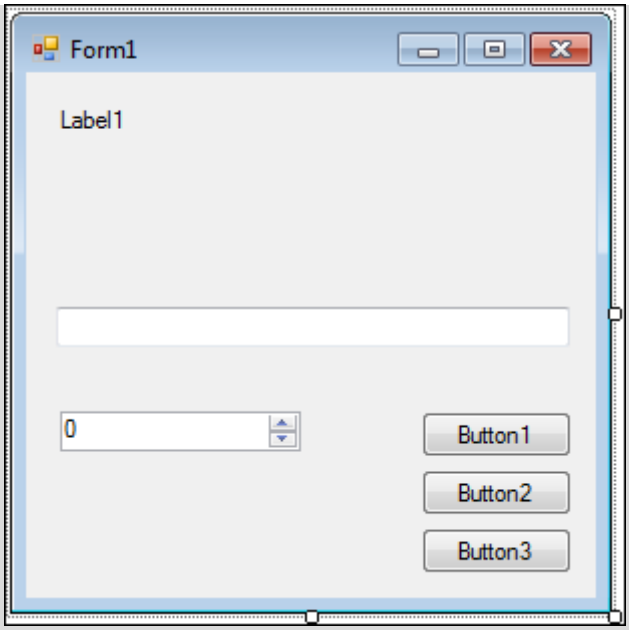
La funzione **String.Substring** (= sottostringa…) estrae un determinato numero di caratteri da un testo, partendo da una determinata posizione.
Esempio:

```
        Dim TestoOriginale As String = "Mammolo, Cucciolo, Brontolo, Eolo,
Dotto, Gongolo, Pisolo."
        Dim NuovaStringa As String = TestoOriginale.Substring(9, 8)
```

La funzione estrae dalla stringa TestoOriginale una stringa di 8 caratteri, partendo dal carattere che si trova nella posizione 9. La stringa NuovaStringa è dunque “Cucciolo”.

Esercizio 65: Le funzioni Insert, Remove, Substring.

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo Label1, un controllo TextBox, un controllo NumeriUpDown e tre pulsanti Button come nell’immagine seguente:



Copiamo e incolliamolo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Button1.Text = "Insert"
        Button2.Text = "Remove"
        Button3.Text = "Substring"
        Label1.Text = "Mammolo, Cucciolo, Brontolo, Eolo, " & vbCrLf &
"Dotto, Gongolo, Pisolo."

        TextBox1.Text = "Scrivi una parola."
        TextBox1.SelectionStart = 0
        TextBox1.SelectionLength = TextBox1.Text.Length
        TextBox1.HideSelection = False
        TextBox1.TabIndex = 0

    End Sub
```

```
    Private Sub Button1_Click() Handles Button1.Click

        Dim TestoIniziale As String = Label1.Text
        Dim TestoDaInserire As String = TextBox1.Text

        Dim NuovoTesto As String
        NuovoTesto = TestoIniziale.Insert(NumericUpDown1.Value, TextBox1.Text)
        MessageBox.Show(NuovoTesto, Button1.Text)

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        Dim TestoIniziale As String = Label1.Text
        Dim NuovoTesto As String
```

```
        NuovoTesto = TestoIniziale.Remove(NumericUpDown1.Value,
NumericUpDown2.Value)
        MessageBox.Show(NuovoTesto, Button2.Text)

    End Sub

    Private Sub Button3_Click() Handles Button3.Click

        Dim TestoIniziale As String = Label1.Text

        Dim TestoEstratto As String
        TestoEstratto = TestoIniziale.Substring(NumericUpDown1.Value,
NumericUpDown2.Value)
        MessageBox.Show(TestoEstratto, Button3.Text)

    End Sub

End Class
```

Mandiamo in esecuzione il programma e proviamo i diversi esiti delle tre funzioni Insert, Remove e Substring cliccando i tre pulsanti e modificando i valori del controllo NumericUpDown.

String.Trim

String.TrimStart

String.TrimEnd

Le funzioni **String.Trim**, **String.TrimStart** e **String.TrimEnd** eliminano dalle stringhe di testo eventuali spazi vuoti all'inizio o alla fine delle stringhe:

- **String**.**Trim** toglie gli spazi vuoti che si trovano all'inizio e alla fine della stringa.
- **String**.**TrimStart** toglie gli spazi vuoti che si trovano all'inizio della stringa.
- **String**.**TrimEnd** toglie gli spazi vuoti che si trovano alla fine della stringa.

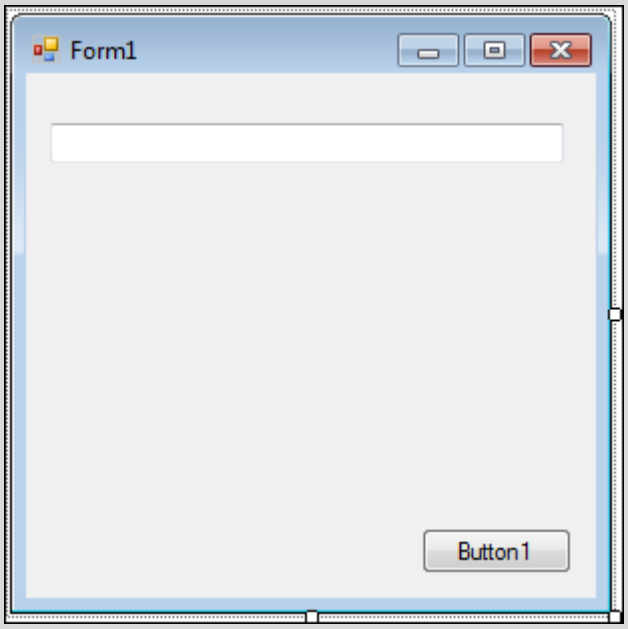
La funzione String.Trim è utile in modo particolare quando si tratta di controllare stringhe di testo immesse dall'utente del programma, per evitare che all'inizio o alla fine di queste stringhe vi siano degli spazi vuoti.
Ne vediamo un esempio nel prossimo esercizio.

Esercizio 66: Le funzioni Trim e Substring.

In un programma è richiesto che l'utente scriva il suo nome in un controllo TextBox1 e poi faccia un *clic* su un Button1 per iniziare il gioco.
Il programma non accetta nomi più lunghi di 12 caratteri, per cui il programmatore scrive una procedura di controllo che effettua queste operazioni:

- ripulisce il nome immesso dall'utente da eventuali spazi vuoti mediante la funzione String.Trim;
- controlla la lunghezza del nome immesso dall'utente mediante la proprietà String.Length;
- se questa lunghezza supera i 12 caratteri, estrae dal nome immesso dall'utente i primi 12 caratteri e li utilizza al posto del nome immesso dall'utente;
- visualizza un messaggio per avvisare l'utente del cambiamento effettuato.

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo TextBox1 e un pulsante Button1 come in questa immagine:

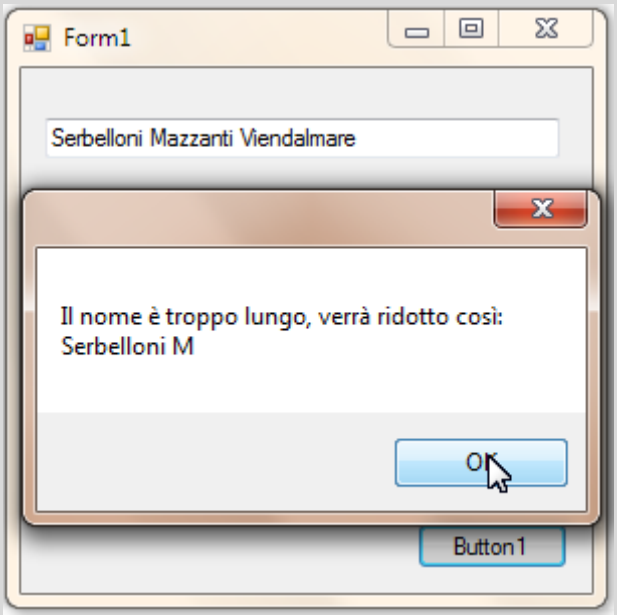


Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click
        ' dimensiona una stringa di testo nella quale immettere il nome del
giocatore:
        Dim NomeGiocatore As String

        ' assegna alla stringa NomeGiocatore ciò che l'utente ha scritto nel
TextBox1
        NomeGiocatore = TextBox1.Text

        ' ripulisci la stringa NomeGiocatore da eventuali spazi vuoti all'inizio
o alla fine:
        NomeGiocatore = NomeGiocatore.Trim

        ' se necessario, riduci la stringa NomeGiocatore a 12 caratteri:
        If NomeGiocatore.Length > 12 Then
            ' assegna alla stringa NomeGiocatore i primi 12 caratteri dalla
stringa stessa:
            NomeGiocatore = NomeGiocatore.Substring(0, 12)
            ' visualizza un messaggio disposto su due righe:
```

```
            MsgBox("Il nome è troppo lungo, verrà ridotto così: " & vbCrLf &
NomeGiocatore)

        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



String.ToLower

String.ToUpper

Le funzioni **String.ToLower** (= al minuscolo) e **String.ToUpper** (= al maiuscolo) trasformano una stringa di testo, rispettivamente, tutta in caratteri minuscoli o tutta in caratteri maiuscoli.
Esempio:

```
        Dim Testo As String = "Nel mezzo del cammin di nostra vita"
        Dim TestoMinuscolo As String = Testo.ToLower
        Dim TestoMaiuscolo As String = Testo.ToUpper
```

Dopo queste trasformazioni, la stringa TestoMinuscolo è "nel mezzo del cammin di nostra vita" e la stringa TestoMaiuscolo è "NEL MEZZO DEL CAMMIN DI NOSTRA VITA".

String.IsNullOrEmpty

String.IsNullOrWhiteSpace

Le due funzioni **String.IsNullOrEmpty** (= la stringa non contiene dati) e **String.IsNullOrWhiteSpace** (= la stringa non contiene dati o è vuota) si utilizzano all'interno di un programma quando occorre verificare che l'utente abbia scritto un testo, ove richiesto.
Le due funzioni restituiscono un valore vero o falso che può essere utilizzato, ad esempio, per visualizzare un messaggio che inviti l'utente a immettere di nuovo i dati richiesti.
Esempio:

```
Dim DatoImmesso As String = TextBox1.Text

If String.IsNullOrEmpty(DatoImmesso) Then
    MessageBox.Show("Non hai scritto nulla, riprova!")
ElseIf String.IsNullOrWhiteSpace(DatoImmesso) Then
    MessageBox.Show("Non hai scritto nulla, riprova!")
End if
```

String.Replace

La funzione **String.Replace** (= sostituisci) cerca in un testo tutti gli elementi che corrispondono a un criterio di ricerca li sostituisce con altri elementi indicati nella funzione stessa.
Nell'esempio seguente vediamo la sostituzione di tutte le vocali presenti in un testo con la vocale "A":

```
Dim TestoOriginario As String = "GARIBALDI FU FERITO"

TestoOriginario = TestoOriginario.Replace("E", "A")
TestoOriginario = TestoOriginario.Replace("I", "A")
TestoOriginario = TestoOriginario.Replace("O", "A")
TestoOriginario = TestoOriginario.Replace("U", "A")
```

Come risultato finale avremo questo testo: "GARABALDA FA FARATA".

String.Split

La funzione **String.Split** (= spezza) suddivide un testo in stringhe, usando come criterio di separazione uno o più caratteri determinati dal programmatore.
Le stringhe così create sono immesse in una matrice di variabili di tipo String.
Nell'esercizio seguente vediamo un esempio di estrazione di tutte le parole singole da un brano, dopo averne eliminato i segni di punteggiatura.
Le parole così ricavate sono numerate e immesse in un controllo ListBox.

Esercizio 67: Le funzioni Replace e Split.

Ariamo un nuovo progetto e inseriamo nel Form1 un controllo ListBox, un controllo TextBox e un pulsante Button, come in questa immagine:

Il controllo TextBox1 ha la proprietà **MultiLine = True**.
Copiamo e incolliamo questo testo nella proprietà **Text** del TextBox1:

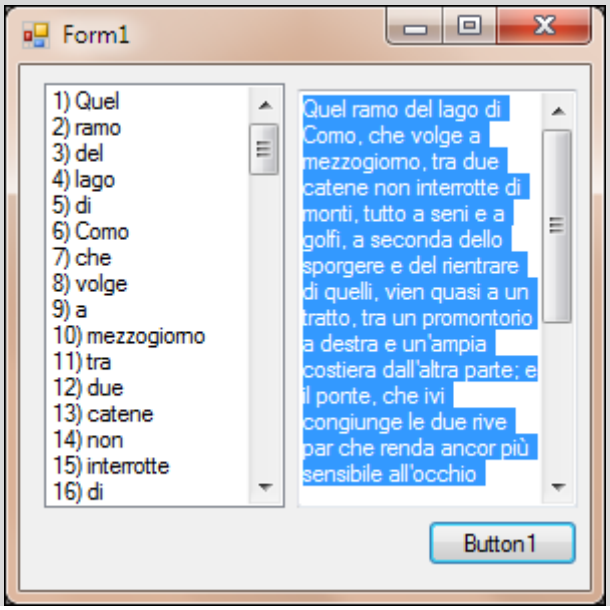
Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien quasi a un tratto, tra un promontorio a destra e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive par che renda ancor più sensibile all'occhio questa trasformazione e segni il punto in cui il lago cessa, e l'Adda ricomincia per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni...

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
```

```
    Private Sub Button1_Click() Handles Button1.Click

        ' crea la variabile TestoDaDividere e le assegna il testo visualizzato
nel TextBox1:
        Dim TestoDaDividere As String = TextBox1.Text

        ' elimina dalla stringa TestoDaDividere i segni di punteggiatura:
        TestoDaDividere = TestoDaDividere.Replace(",", "")
        TestoDaDividere = TestoDaDividere.Replace(";", "")
        TestoDaDividere = TestoDaDividere.Replace(".", "")

        ' Crea e imposta la variabile Contatore, che servirà a contare le singole
parole nella stringa TestoDaDividere:
        Dim Contatore As Integer = 1

        ' crea una matrice di variabili di testo di nome Parole e assegna a
questa matrice tutte le singole parole che si trovano nel TestoDaDividere,
utilizzando come elemento separatore lo spazio tra le parole:
        Dim Parole As String() = TestoDaDividere.Split(" ")

        ' il ciclo For Each... Next passa tutte le parole singole dalla matrice
Parole al controllo ListBox;
        ' ogni parola è preceduta da un numero d'ordine dato dalla variabile
Contatore:
        For Each Parola As String In Parole
            ListBox1.Items.Add(Contatore.ToString & ") " & Parola)
            ' a ogni passaggio il contatore aumenta di una unità:
            Contatore = Contatore + 1
        Next

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



String.ToCharArray

La funzione **String.ToCharArray** (= ad una matrice di variabili di tipo **Char**) scompone un testo nei caratteri che lo compongono e crea una matrice di variabili, di tipo Char (= caratteri), con tanti elementi quanti sono i caratteri del testo originale; ogni carattere del testo originale diventa un elemento della matrice di variabili creata da String.CharToArray.
Esempio:

```
Dim TestoDaDividere As String = "ABCDEFGHILMNOPQRSTUVZ"
Dim Caratteri() As Char = TestoDaDividere.ToCharArray

ListBox1.Items.Add(Caratteri(0))
ListBox1.Items.Add(Caratteri(4))
ListBox1.Items.Add(Caratteri(8))
ListBox1.Items.Add(Caratteri(12))
ListBox1.Items.Add(Caratteri(18))
```

In questo esempio, il programma prende il testo "ABCDEFGHILMNOPQRSTUVZ" e lo scompone nei 21 caratteri che lo compongono.
Con questi caratteri viene creata una matrice variabili di nome Caratteri, destinata a contenere dati di tipo Char (= caratteri).
Da questa matrice vengono poi prelevate, e aggiunte ad un ListBox1, le variabili con i numeri d'ordine 0, 4, 8, 12 e 18, corrispondenti alle cinque vocali:

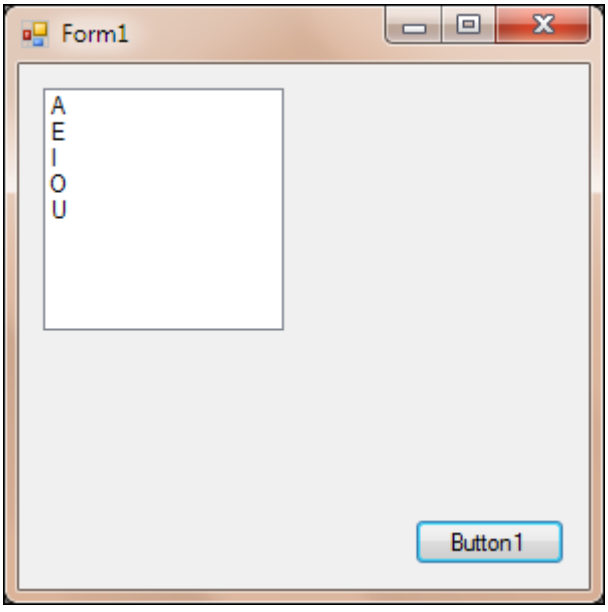


Figura 153: La funzione String,ToCharArray.

Choose

La funzione **Choose** preleva, da un elenco di oggetti, l'oggetto corrispondente al numero d'ordine indicato dal programmatore.
Può essere utile, in modo particolare, quando si tratta di estrapolare una stringa di testo da un elenco, come in questo esempio:

```
        Dim AggettivoOrdinale As String = Choose(3, "primo", "secondo", "terzo",
"quarto", "quinto", "sesto", "settimo")
        Dim NomeGiorno As String = Choose(3, "lunedì", "martedì", "mercoledì",
"giovedì", "venerdì", "sabato", "domenica")
        MsgBox("Il " & AggettivoOrdinale & " giorno è " & NomeGiorno)
```

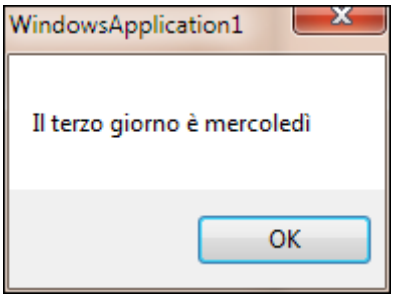


Figura 154: La funzione Choose.

In questo esempio, la funzione **Choose** preleva le stringhe corrispondenti al n. 3 nella lista di aggettivi e nella lista di nomi, poi le visualizza nel MsgBox.
Notare la sintassi della funzione: tra le parentesi va scritto innanzitutto il numero dell'elemento da estrapolare, poi, a seguire, tutti gli elementi dell'elenco.
La numerazione degli elementi parte da 1, per cui impostando Choose(0,…) si causa un errore nel programma.

Asc e AscW

Chr e ChrW

A ogni carattere prodotto dalla tastiera del computer corrisponde un numero di identificazione, tratto da un codice internazionale denominato ASCII[65].
Nella tabella seguente vediamo i caratteri corrispondenti ai codici ASCII da 13 a 125, che comprendono le lettere dell'alfabeto e i simboli di uso più frequente.

[65] **American Standard Code for Information Interchange** (Codice Americano Standard per lo Scambio di Informazioni).

Codice	Carattere	Codice	Carattere	Codice	Carattere	Codice	Carattere
13	Tasto INVIO	**51**	3	**76**	L	**101**	e
32	Barra SPAZIO	**52**	4	**77**	M	**102**	f
33	!	**53**	5	**78**	N	**103**	g
34	"	**54**	6	**79**	O	**104**	h
35	#	**55**	7	**80**	P	**105**	i
36	$	**56**	8	**81**	Q	**106**	j
37	%	**57**	9	**82**	R	**107**	k
38	&	**58**	:	**83**	S	**108**	l
39	'	**59**	;	**84**	T	**109**	m
40	(	**60**	<	**85**	U	**110**	n
41	)	**61**	=	**86**	V	**111**	o
42	*	**62**	>	**87**	W	**112**	p
43	+	**63**	?	**88**	X	**113**	q
44	,	**64**	@	**89**	Y	**114**	r
45	-	**65**	A	**90**	Z	**115**	s
46	.	**66**	B	**91**	[	**116**	t
47	/	**67**	C	**92**	\	**117**	u
48	0	**68**	D	**93**	]	**118**	v
49	1	**69**	E	**94**	^	**119**	w
50	2	**70**	F	**95**	_	**120**	x
		71	G	**96**	`	**121**	y
		72	H	**97**	a	**122**	z
		73	I	**98**	b	**123**	{
		74	J	**99**	c	**124**	|
		75	K	**100**	d	**125**	}

Tabella 24: I codici ASCII.

Notiamo che al numero 13 non corrisponde un carattere ma la pressione del tasto INVIO o ENTER.

Le funzioni **Asc** e **Chr** consentono al programmatore di operare con i numeri del codice ASCII e con i caratteri corrispondenti a questi numeri:

- la funzione **Asc** restituisce il numero di codice che corrisponde alla lettera o al simbolo scritto tra parentesi;
- la funzione **Chr**, viceversa, restituisce (o scrive) la lettera o il simbolo che corrisponde al numero di codice scritto tra parentesi.

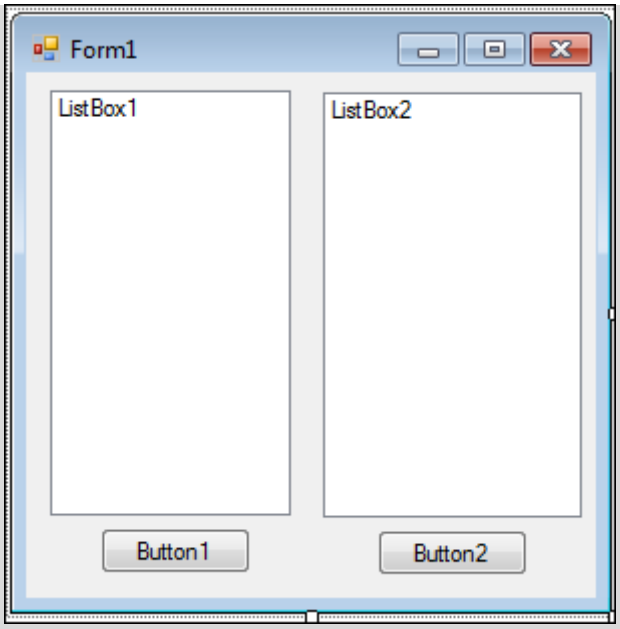
Esempio:

```
Dim NumeroDiCodice As Integer
NumeroDiCodice = Asc("A")  ' (NumeroDiCodice = 65)
NumeroDiCodice = Asc("B")  ' (NumeroDiCodice = 66)
NumeroDiCodice = Asc("C")  ' (NumeroDiCodice = 67)

Dim Lettera As String
Lettera = Chr(65) ' (Lettera = “A”)
Lettera = Chr(66) ' (Lettera = “B”)
Lettera = Chr(67) ' (Lettera = “C”)
```

La funzione **Asc** può essere utilizzata per sapere quale tasto sulla tastiera è stato premuto dall’utente. Ad esempio: se il codice **Asc** del tasto premuto è uguale a 13, questo significa che l’utente ha premuto il pulsante **INVIO**.
La funzione **Chr** può essere utilizzata anche per visualizzare simboli e lettere che non sono direttamente disponibili sulla tastiera, quali ad esempio, le parentesi graffe oppure lettere con simboli non presenti nella tastiera italiana.

Esercizio 68: Le funzioni Asc e Chr.

Apriamo un nuovo progetto e inseriamo nel Form1 due controlli ListBox e due pulsanti Button, come nell’immagine seguente.



Funzionamento del programma:

- Cliccando il Button1, nel ListBox1 verranno visualizzate le lettere maiuscole con i loro codici ASCII e nel ListBox2 verranno visualizzate le lettere minuscole con i loro codici ASCII.
- Cliccando il Button2, nel ListBox2 verranno visualizzati tutti i simboli e le lettere del codice ASCII, corrispondenti ai numeri da 0 a 255.

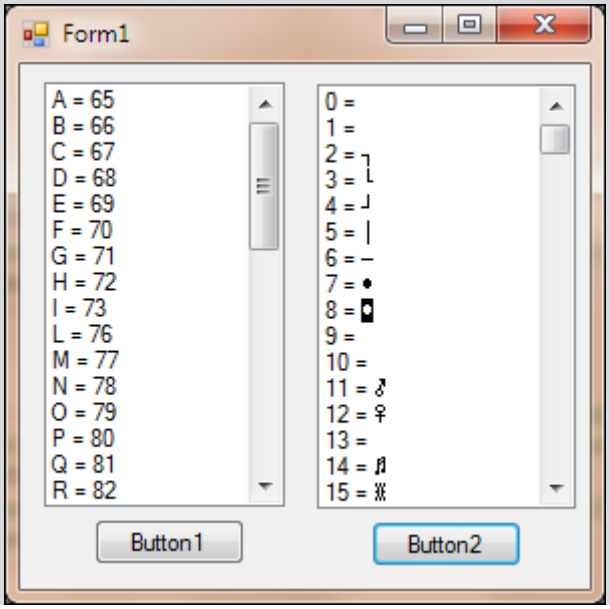
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click
        ' crea due stringhe di testo, una per l'alfabeto maiuscolo e l'altra per
l'alfabeto minuscolo:
        Dim AlfabetoMaiuscolo As String = "ABCDEFGHILMNOPQRSTUVZ"
        Dim AlfabetoMinuscolo As String = "ABCDEFGHILMNOPQRSTUVZ".ToLower

        ' ripulisci il ListBox1 e il ListBox2:
        ListBox1.Items.Clear()
        ListBox2.Items.Clear()

        ' crea una matrice di variabili di tipo Char con i caratteri della
AlfabetoMaiuscolo:
        Dim LettereMaiuscole() As Char = AlfabetoMaiuscolo.ToCharArray
        ' aggiungi tutti gli elementi della matrice LettereMaiuscole al ListBox1,
con il loro codice ASCII:
        For Each Lettera As Char In LettereMaiuscole
            ListBox1.Items.Add(Lettera & " = " & Asc(Lettera).ToString)
        Next

        ' crea una matrice di variabili di tipo Char con i caratteri della
AlfabetoMinuscolo:
        Dim LettereMinuscole() As Char = AlfabetoMinuscolo.ToCharArray
        ' aggiungi tutti gli elementi della matrice LettereMinuscole al ListBox1,
con il loro codice ASCII:
        For Each Lettera As Char In LettereMinuscole
            ListBox2.Items.Add(Lettera & " = " & Asc(Lettera).ToString)
```

```
        Next

    End Sub
```

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

        ' ripulisce il ListBox2:
        ListBox2.Items.Clear()

        For Contatore As Integer = 0 To 255
            ' a ogni passaggio del ciclo For... Next aggiungi al ListBox2 il
numero Contatore e il simbolo con il codice ASCII corrispondente:
            ListBox2.Items.Add(Contatore.ToString & " = " & Chr(Contatore))
        Next

    End Sub

End Class
```

Nell'immagine seguente vediamo il programma in esecuzione.



Il codice ASCII è nato attorno al 1960, in àmbito linguistico inglese, con lo scopo di raccogliere tutti i caratteri in uso in quell'àmbito.
Nel corso degli ultimi 20 anni è stato sviluppato un altro codice, denominato Unicode, che ha l'ambizione di raccogliere tutti i caratteri (lettere, numeri, simboli, ideogrammi, logogrammi) presenti nei linguaggi e nelle forme di scrittura di tutto il mondo.
Il codice Unicode raccoglie dunque decine di migliaia di simboli ai quali si può fare riferimento, in VB, utilizzando le funzioni **AscW** e **ChrW**. Queste due funzioni estese funzionano come le due funzioni ridotte che abbiamo visto in precedenza ma, mentre il valore massimo di Asc e Chr è 255, il valore massimo di AscW e ChrW arriva a 65535:

- i codici per i caratteri greci vanno da 890 a 1023;
- i codici per i caratteri cirillici vanno da 1024 a 1279;
- i codici per i caratteri ebraici vanno da 1424 a 1535;

- i codici per i caratteri arabi vanno da 1536 a 1791;
- i codici per i caratteri cinesi vanno
  - da 19968 a 20479;
  - da 20480 a 24575;
  - da 24576 a 28671;
  - da 28672 a 32767;
  - da 32768 a 36863;
  - da 36864 a 40959.

Possiamo visualizzare questi set di caratteri con il codice dell'esercizio precedente, modificando la procedura che gestisce l'evento del *clic* sul Button2.
Questa modifica, ad esempio, visualizza nel ListBox2 il primo set di caratteri cinesi:

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

        ' ripulisce il ListBox2:
        ListBox2.Items.Clear()

        For Contatore As Integer = 19968 To 20479
            ' a ogni passaggio del ciclo For... Next aggiungi al ListBox2 il
numero Contatore e il simbolo con il codice ASCII corrispondente:
            ListBox2.Items.Add(Contatore.ToString & " = " & ChrW(Contatore))
        Next

    End Sub
```

Ecco un'immagine del programma in esecuzione con la funzione ChrW:

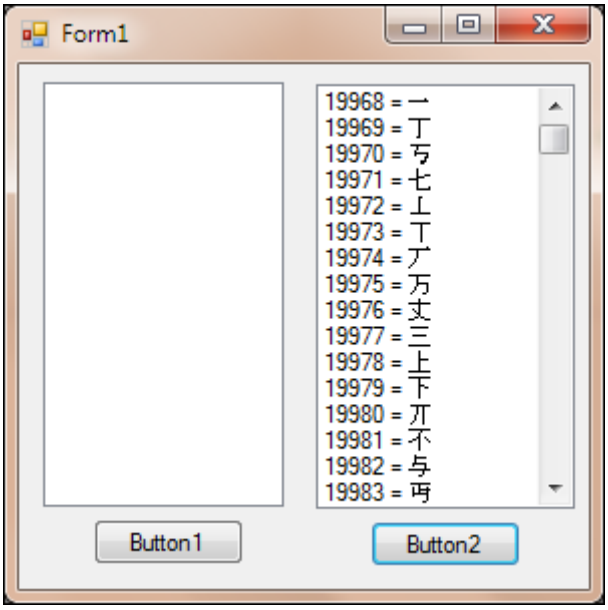


Figura 155: La funzione ChrW.

Capitolo 22: FORMATTAZIONE DI NUMERI, TESTI E DATE.

La funzione Format dà un formato uniforme a cifre, parole, date, perché questi possano essere visualizzati in modo ordinato in prospetti, tabelle, incolonnamenti, riquadri, sullo schermo o su un foglio stampato.

114: Formattazione di numeri.

Questa tabella mostra i principali simboli utilizzati per formattare variabili numeriche:

Simbolo	Azione	Esempio	Risultato
0	Forza l'apparizione di una cifra nella posizione in cui si trova lo zero. Se non ci sono cifre sufficienti nel numero da formattare, al loro posto è visualizzato uno zero.	Format(99, "000")	099
#	Produce gli stessi effetti di zero, con la differenza che se non ci sono cifre sufficienti nel numero da formattare, al posto del simbolo # non compare nulla.	Format(99, "###")	99
.	I simboli dopo il punto indicano quanti decimali debbono essere visualizzati.	Format(99, "000.00") Format(99, "###.##")	099,00 99
,	La virgola forza la cifra formattata a visualizzare il separatore delle migliaia.	Format(99, "0,000.00") Format(99, "#,###.##")	0.099,00 99
N	Formatta un numero in modo standard, con il separatore delle migliaia, la virgola e due numeri	Format(99, "N")	99,00

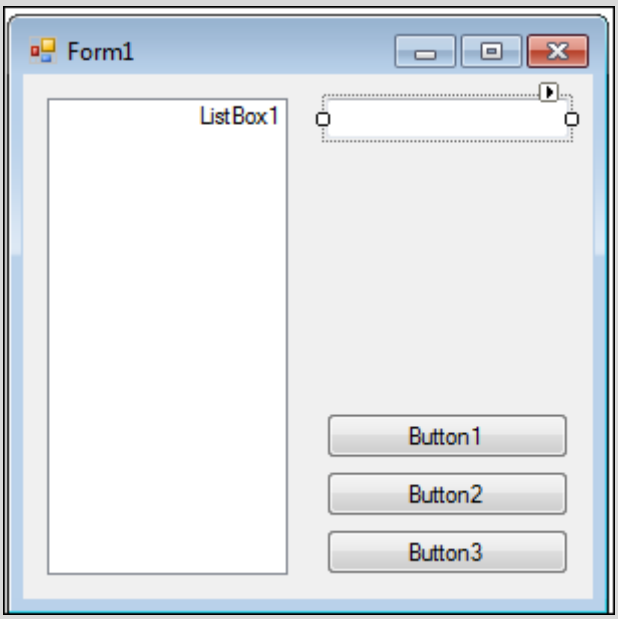
	decimali dopo la virgola.		
C	Formatta un importo nella moneta corrente, con il punto per le migliaia, la virgola, due cifre decimali dopo la virgola e il simbolo della moneta.	Format(99, "C")	99,00 €

Tabella 25: I simboli per la formattazione di cifre.

Nel prossimo esercizio vedremo alcuni esempi di visualizzazione di numeri con modalità di formattazione diverse.

Esercizio 69: Formattazione di numeri.

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo ListBox, un TextBox e tre pulsanti Button come in questa immagine:



Impostiamo questa proprietà del controllo **ListBox1**:

- **RightToLeft = Yes**

Lo schema di funzionamento del programma è questo: l'utente scrive un numero nel TextBox e clicca uno dei tre pulsanti. A seconda del pulsante cliccato, il numero scritto nel TextBox viene visualizzato nel ListBox con tre modalità di formattazione diverse;

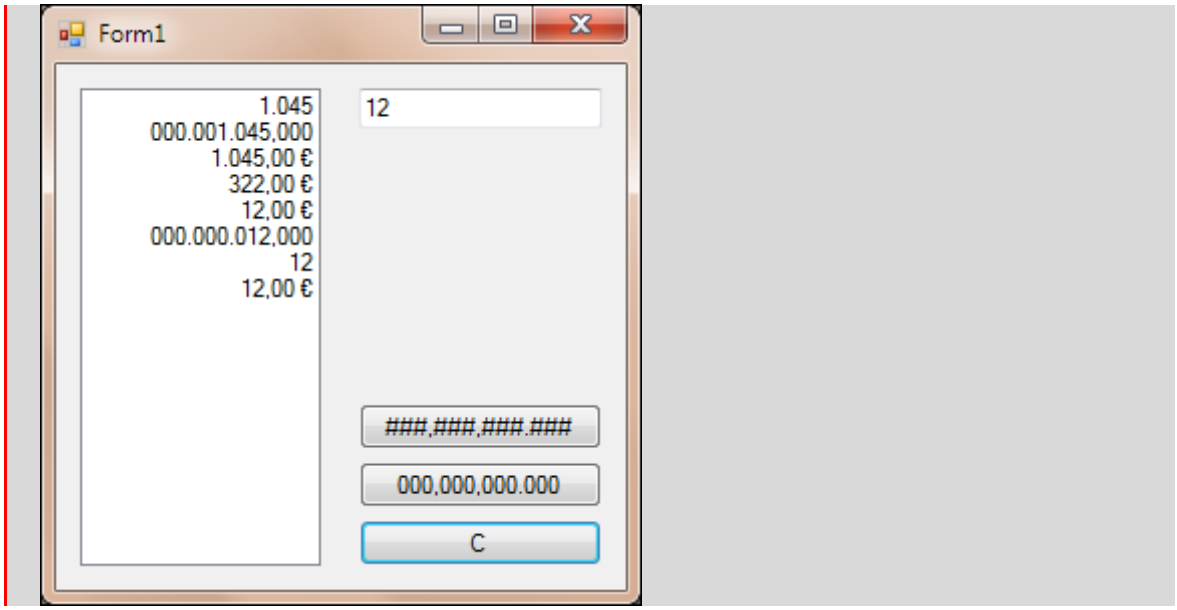
- **Format(Cifra, "###,###,###.###")**
- **Format(Cifra, "000,000,000.000")**
- **Format(Cifra, "C")**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' all'avvio del programma imposta la proprietà Text dei tre pulsanti e
porta il cursore sul TextBox1:
        Button1.Text = "###,###,###.###"
        Button2.Text = "000,000,000.000"
        Button3.Text = "C"
        TextBox1.TabIndex = 0

    End Sub
```

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click, Button2.Click, Button3.Click

        ' questa procedura gestisce il clic del mouse sui tre pulsanti Button: il
parametro sender corrisponde di volta in volta al pulsante premuto.

        ' dimensiona una variabile di tipo Single (numeri con la virgola):
        Dim Cifra As Single

        ' è possibile che l'utente commetta degli errori nella scrittura dei
numeri nel TextBox1.
        'Ad esempio, è possibile che scriva una lettera invece che un numero.
        '  Inseriamo un rimando GoTo a una linea successiva, per evitare il
blocco del programma:

        On Error GoTo GestioneErrori

        ' converti il contenuto del TextBox1 in una variabile di tipo Single:
        Cifra = CSng(TextBox1.Text)

        ' a seconda del pulsante Button premuto, formatta la cifra in un modo
diverso:
        If sender Is Button1 Then
            ListBox1.Items.Add(Format(Cifra, "###,###,###.###"))
        ElseIf sender Is Button2 Then
            ListBox1.Items.Add(Format(Cifra, "000,000,000.000"))
        Else
            ListBox1.Items.Add(Format(Cifra, "C"))

        End If

        ' esci dalla procedura, perché ciò che segue riguarda la correzione
degli errori:
        Exit Sub

GestioneErrori:
        MsgBox("Scrivi un numero valido.")

    End Sub

End Class
```

Ecco un’immagine del programma in esecuzione:



115: Formattazione di stringhe di testo.

VB offre la possibilità di formattare le stringhe di testo per visualizzarle, se necessario, allineate una sotto l'altra, come se fossero incolonnate in una tabella.
La funzione per formattare un testo è **String.Format()**, che è impostata dal programmatore scrivendo tra le parentesi due parametri:

- il numero delle colonne da visualizzare e
- la loro larghezza.

La larghezza delle colonne è indicata in numero di caratteri, per cui per incolonnare le parole in modo preciso è necessario usare un set di caratteri a dimensioni fisse, come il font **Courier New**.
Con questo font, a ogni lettera è assegnato il medesimo spazio; con un font proporzionale, invece, parole con lo stesso numero di caratteri possono avere lunghezze diverse, per cui il loro incolonnamento con String.Format() è impossibile.
Ecco un esempio di uso di un font fisso e di un font proporzionale:

- `font Courier New: mmmmm mmmmm mmmmm mmmmm mmmmm`
- `font Courier New: iiiii iiiii iiiii iiiii iiiii`

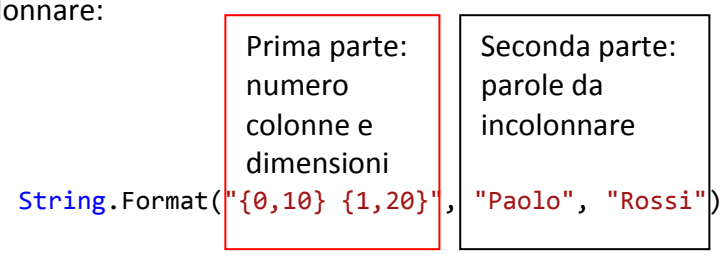
- font proporzionale: mmmmm mmmmm mmmmm mmmmm mmmmm
- font proporzionale: iiiii iiiii iiiii iiiii iiiii

Notiamo come il font Courier New mantiene incolonnate le stringhe di testo con gruppi di lettere "m" e gruppi di lettere "i", che invece il font proporzionale non fa.

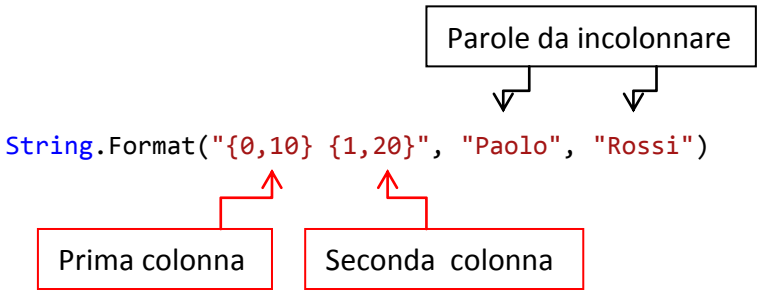
La sintassi della funzione String.Format per l' incolonnamento di parole su due colonne è questa:

```
String.Format("{0,10} {1,20}", "Paolo", "Rossi")
```

I parametri di questa funzione, scritti tra le parentesi tonde, si dividono in due parti:
Nella prima parte (in rosso nell'esempio) troviamo le indicazioni sul numero e la larghezza delle colonne; nella seconda parte (in nero nell'esempio) troviamo le parole da incolonnare:

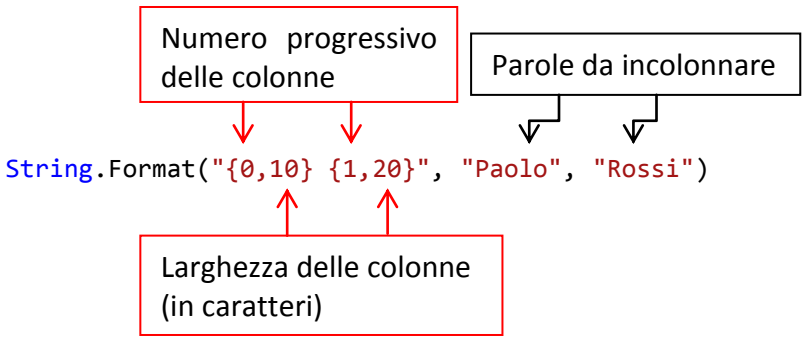


In questa riga di esempio, nella seconda parte dei parametri abbiamo due parole da incolonnare, per cui nella prima parte dei parametri troviamo la progettazione di due colonne:



Notiamo che le indicazioni relative alle colonne si trovano tra parentesi graffe (le parentesi graffe si scrivono premendo contemporaneamente sulla tastiera i tasti MAIUSC + ALT GR + parentesi quadre).
All'interno delle parentesi graffe, per ogni colonna, sono indicati due elementi:

- il numero d'ordine della colonna;
- la larghezza della colonna, espressa in caratteri.



Se l'indicazione della larghezza delle colonne è preceduta da un trattino si avrà l'allineamento delle parole a sinistra, in caso contrario si avrà l'allineamento delle parole a destra.

Ecco un esempio di allineamento a sinistra:

```
String.Format("{0,-10} {1,-20}", "Paolo", "Rossi")
```

e un esempio di allineamento a destra:

```
String.Format("{0,10} {1,20}", "Paolo", "Rossi")
```

L'immagine seguente mostra l'effetto d'incolonnamento ottenuto con questi comandi:

```
Label1.Font = New Font("Courier New", 12)
Label1.Text = String.Format("{0,-10} {1,-20}", "Paolo", "Rossi")
Label1.Text &= vbCrLf
Label1.Text &= String.Format("{0,-10} {1,-20}", "Giovanni", "Bianchi")
Label1.Text &= vbCrLf
Label1.Text &= String.Format("{0,-10} {1,-20}", "Mario", "Neri")
```

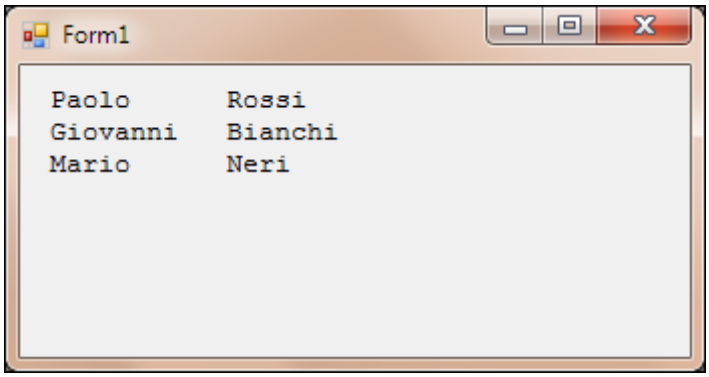


Figura 156: La funzione String.Format.

Esercizio 70: Incolonnamento di stringhe di testo.

In questo esercizio utilizzeremo la funzione **String.Format**() per visualizzare una tabella con l'elenco delle regioni italiane e il numero dei loro abitanti.
I nomi delle regioni e il numero degli abitanti saranno ordinati in due colonne:

- la prima con l'elenco dei nomi delle regioni allineati a sinistra,
- la seconda con l'elenco dei numeri degli abitanti allineati a destra.

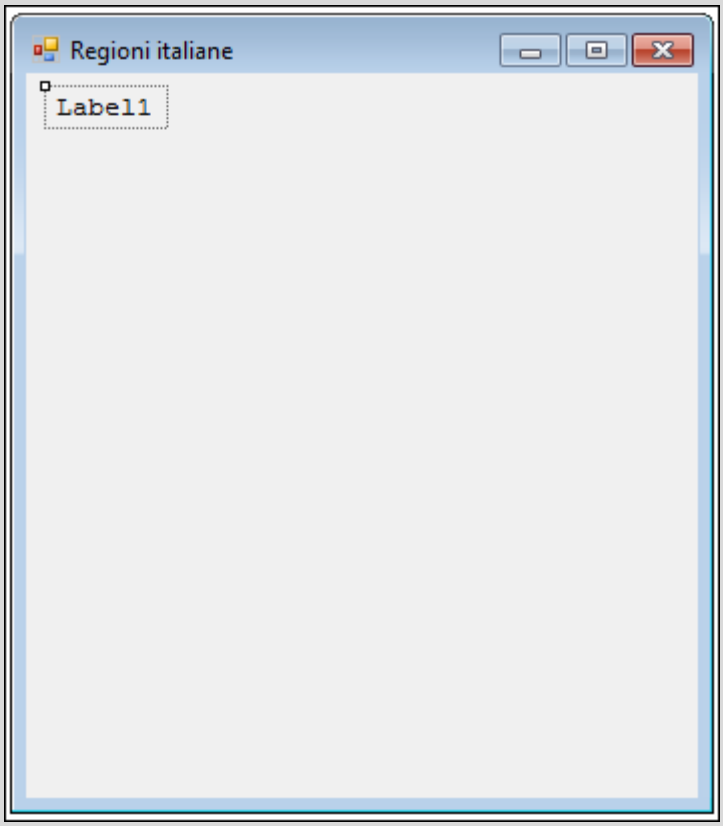
I dati sono estratti da due file di testo denominati rispettivamente **Regioni.txt** e **Abitanti.txt**, che si trovano nella cartella **Documenti / A scuola con VB 2010 / Testi.**
La funzione con la quale vengono letti e utilizzati i dati contenuti in questi file verrà spiegata più avanti nel manuale[66].
Apriamo un nuovo progetto.
Impostazione delle proprietà del **Form1**:

[66] Si veda il Capitolo 34: GESTIONE DI FILE ESTERNI AL PROGRAMMA., a pag. 780.

- **Size = 350; 400**
- **Text = Regioni italiane**

Inseriamo nel Form1 un controllo **Label1** con la proprietà

- **Font = Courier New a 10 punti:**

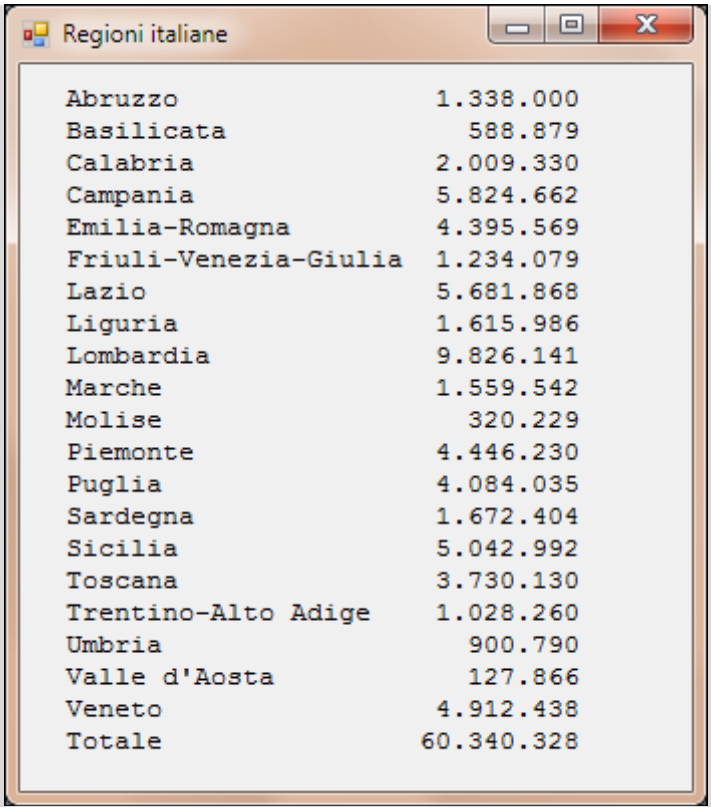


Ora copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' crea due matrici di variabili per la memorizzazione dei dati:
    Dim NomeRegione(20) As String
    Dim NumeroAbitanti(20) As String

    ' memorizza in una stringa di testo la collocazione dei due file Regioni.txt
e Abitanti.txt
    ' usati nel programma:
    Dim PercorsoFile As String =
My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\A scuola con VB
2010\Testi\"

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        ' (questa procedura gestisce l'evento Me.Load ed è eseguita all'avvio del
programma)

        Label1.Text = ""
        Dim Contatore As Integer
```

```
        ' apre il file Regioni.txt, ne estrae i dati uno alla volta, riga per
riga, e li immette nella matrice NomeRegione:
        Dim LeggiNomiRegioni As New System.IO.StreamReader(PercorsoFile &
"Regioni.txt")
        For Contatore = 0 To 20
            NomeRegione(Contatore) = LeggiNomiRegioni.ReadLine
        Next

        ' apre il file Abitanti.txt, ne estrae i dati uno alla volta, riga per
riga e li immette nella matrice NumeroAbitanti:
        Dim LeggiNumeriAbitanti As New System.IO.StreamReader(PercorsoFile &
"Abitanti.txt")
        For Contatore = 0 To 20
            NumeroAbitanti(Contatore) = LeggiNumeriAbitanti.ReadLine
        Next

        ' formatta i dati su due colonne, per visualizzarli all'interno della
Label1:
        ' la prima colonna è incolonnata a sinistra ed è lunga 21 caratteri;
        ' la seconda colonna è incolonnata a destra ed è lunga 10 caratteri;
        ' al termine di ogni riga la sigla vbcrlf manda il testo a capo:

        For Contatore = 0 To 20
            Label1.Text &= String.Format(" {0,-21} {1,10}",
NomeRegione(Contatore), NumeroAbitanti(Contatore)) & vbCrLf
        Next

    End Sub

End Class
```

Il programma al suo avvio legge i nomi delle regioni e i numeri degli abitanti dai due file di testo che devono trovarsi nella cartella **Documenti / A scuola con VB 2010 / Testi**, e li visualizza nella Label1 su due colonne, secondo le indicazioni della funzione String.Format.
L'immagine seguente mostra il programma in esecuzione:



116: Conversione di numeri in testo e viceversa.

Si presenta spesso, in un programma, l'esigenza di visualizzare un numero (una variabile di tipo numerico) in un controllo che invece è predisposto a ricevere e visualizzare stringhe di testo.
VB è in grado di capire quando una variabile numerica è trattata come testo, e di fare gli adattamenti necessari, per cui, ad esempio, questi comandi visualizzano correttamente la scritta "numero 10,25" all'interno di un Label, di un TextBox, di un ListBox:

```
Dim A As Single = 10.25
Label1.Text = "numero " & A
TextBox1.Text = "numero " & A
ListBox1.Items.Add("numero " & A)
```

Per evitare rischi, tuttavia, è buona norma scrivere il comando della trasformazione di un numero in testo, ove occorre, aggiungendo al numero la funzione **.ToString**:

```
Dim A As Single = 10.25
Label1.Text = "numero " & A.ToString
TextBox1.Text = "numero " & A.ToString
ListBox1.Items.Add("numero " & A.ToString)
```

Viceversa, può accadere di dovere utilizzare come variabile di tipo numerico un dato che è stato immesso dall'utente in un controllo come testo.
Supponiamo, ad esempio, che l'utente debba scrivere due numeri in due TextBox diversi e che il programma utilizzi questi due numeri per farne un'addizione.
In questo caso, i due numeri scritti nei due TextBox sono in realtà due testi e per essere prelevati e utilizzati come variabili numeriche richiedono un'operazione di conversione.
VB è in grado di capire quando una variabile di testo è utilizzata come variabile numerica, per cui, ad esempio, queste istruzioni funzionano senza problemi:

```
Dim PrimoNumero As Integer = TextBox1.Text
Dim SecondoNumero As Integer = TextBox2.Text
Dim Somma As Integer = PrimoNumero + SecondoNumero
Label1.Text = Somma
```

Anche in questo caso, tuttavia, per essere sicuri del funzionamento del programma in ogni eventualità, è preferibile inserire un'operazione esplicita di conversione in variabili numeriche dei numeri presenti come testo nei due TextBox.
Questa conversione si opera con CInt() o CSng():

- la funzione CInt() converte il testo in una variabile numerica di tipo Integer,
- la funzione CSng() converte il testo in una variabile numerica di tipo Single (numeri decimali).

Ecco l'esempio precedente riscritto con tutte le conversioni necessarie:

```
Dim PrimoNumero As Single = CSng(TextBox1.Text)
Dim SecondoNumero As Single = CSng(TextBox2.Text)
Dim Somma As Single = PrimoNumero + SecondoNumero
Label1.Text = Somma.ToString
```

Naturalmente le conversioni testo/numeri sono possibili se il testo da convertire contiene solo caratteri numerici, altrimenti si incorre in un errore nel funzionamento del programma.

117: Date e orari.

La funzione che consente di visualizzare la data e l'ora correnti in un programma è `Date`.
Essa ha due proprietà:

```
Date.Today '(= oggi)
Date.Now '(= adesso)
```

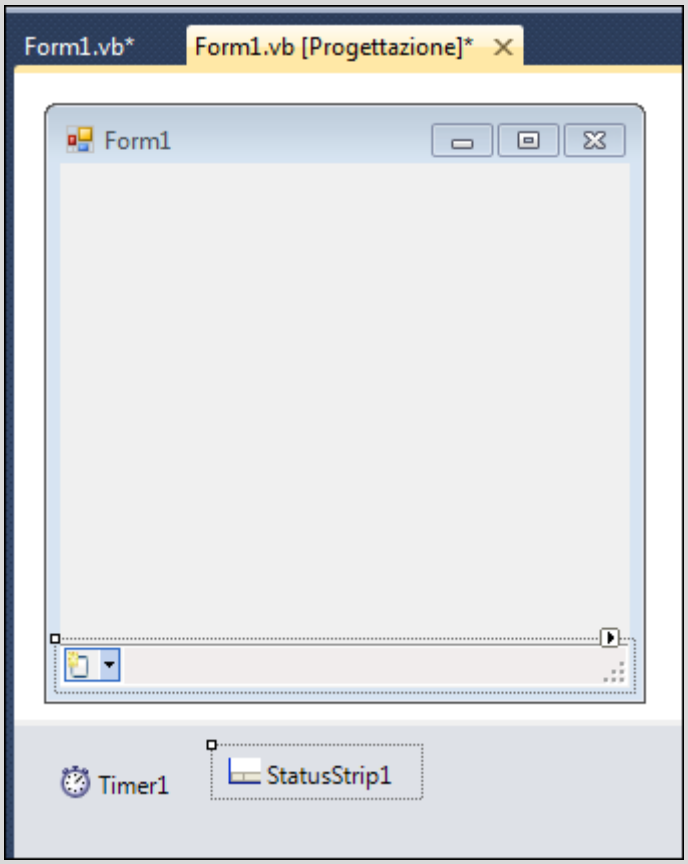
La proprietà `Date.Today` legge la **data** nel sistema operativo del computer e la visualizza in questo formato:
25/10/2010

La proprietà `Date.Now` legge la **data** e l'**ora** nel sistema operativo del computer e li visualizza in questo formato:
25/10/2010 12:21:21

Vedremo l'uso di queste funzioni nel prossimo esercizio.

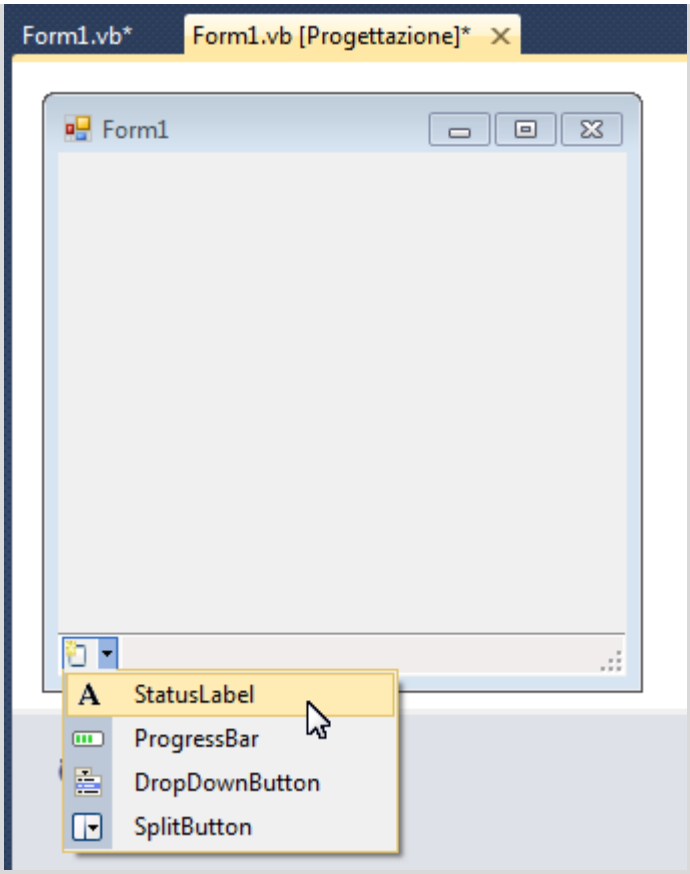
Esercizio 71: La funzione Date.Now.

Realizziamo un programma che visualizza, in una striscia in basso nel form, la data e l'ora correnti.
Apriamo un nuovo progetto e inseriamo nel Form1 un controllo **Timer** e un controllo **StatusStrip**, come in questa immagine:



Proprietà del controllo **Timer**:
- **Enabled = True**
- **Interval = 1000**

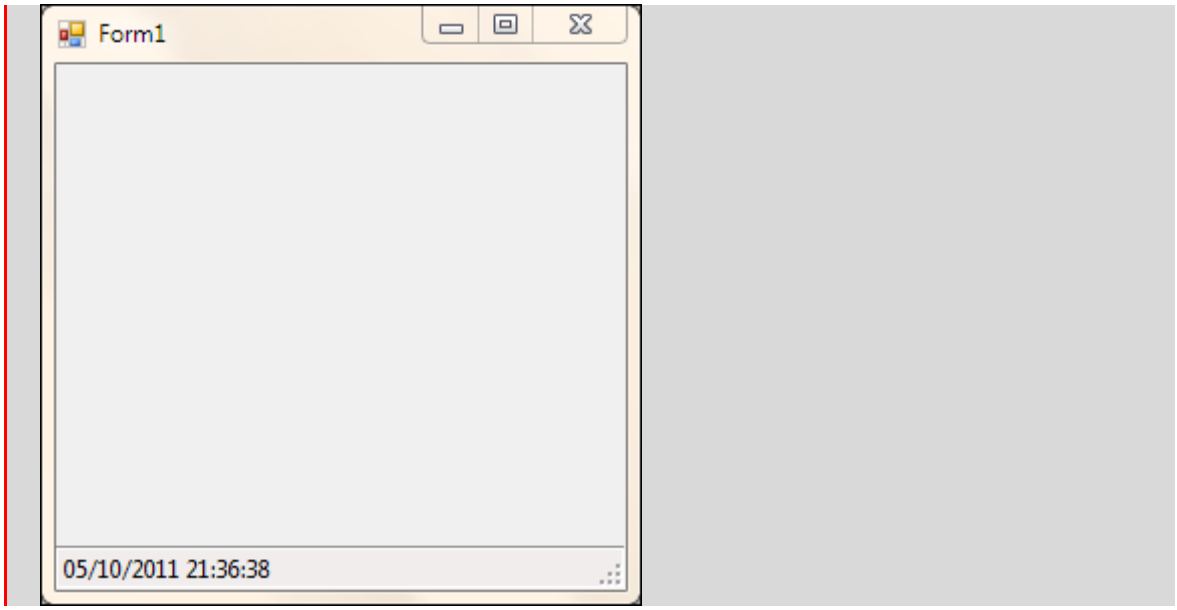
Inseriamo nel controllo **StatusStrip** un controllo **StatusLabel**, come in questa immagine:



Ora copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Timer1_Tick() Handles Timer1.Tick
        ToolStripStatusLabel1.Text = ""
        ToolStripStatusLabel1.Text = Date.Now

    End Sub

End Class
```

Quando il programma è in esecuzione, a ogni *tic* del timer nella striscia in basso nel form viene aggiornata l'ora, con la funzione Date.Now.
Ecco un'immagine del programma in esecuzione:



La funzione **Date.Today** ha alcuni parametri aggiuntivi che sono utili al programmatore per estrapolare determinati elementi dalla data corrente:

Comando	Risultato supponendo che la data corrente sia domenica 23/01/2011:
Date.Today.Day	23 (è il 23.mo giorno del mese)
Date.Today.DayOfWeek	0 (lo 0 corrisponde al primo giorno della settimana)
Date.Today.DayOfYear	23 (è il 23.mo giorno dell'anno)
Date.Today.Month	1 (è il primo mese dell'anno)
Date.Today.Year	2011 (è l'anno 2011)

Tabella 26: Parametri della funzione Date.Today.

I parametri della funzione **Date.Now** sono questi:

Comando	Risultato supponendo che siano le ore 21.30 di domenica 23/01/2011:
Date.Now.Hour	21
Date.Now.Minute	30
Date.Now.Second	23
Date.Now.Millisecond	561

Tabella 27: Parametri della funzione Date.Now.

118: Formattazione di date e orari.

Date e orari possono essere visualizzate in modi molto diversi, utilizzando la funzione Format().
Come avviene per la formattazione di cifre, anche per la formattazione di date e orari la funzione Format() deve contenere tra parentesi due parametri, che in questo caso sono:

- la data o l'orario da formattare;
- la modalità di formattazione.

Le due tabelle che seguono mostrano alcuni modi di formattazione rispettivamente della data (Date.Today) e dell'orario (Date.Now).

Date.Today

In questa tabella sono visualizzate le principali modalità di formattazione di una data, con i relativi risultati.

Modalità di formattazione	Risultato
Format(Date.Today, "d")	Data odierna in formato ridotto.
Format(Date.Today, "D")	Data odierna in formato esteso.

Modalità di formattazione	Risultato
Format(Date.Today, "dd"	Numero del giorno a 2 cifre.
Format(Date.Today, "ddd")	Nome del giorno abbreviato (le prime tre lettere).
Format(Date.Today, "dddd"	Nome completo del giorno corrente.
Format(Date.Today, "M")	Nome del mese corrente.
Format(Date.Today, "MM")	Numero del mese corrente, su due cifre.
Format(Date.Today, "MMM")	Nome del mese abbreviato (le prime tre lettere).
Format(Date.Today, "Y")	Anno.

Tabella 28: Principali modalità di formattazione di una data.

Date.Now

In questa tabella sono visualizzate le principali modalità di formattazione di un orario, con i relativi risultati.

Format(Date.Now, "t")	Ora corrente, in formato ridotto.
Format(Date.Now, "T")	Ora corrente, in formato esteso.
Format(Date.Now, "hh")	Ora corrente, nel formato a 12 ore.
Format(Date.Now, "HH")	Ora corrente, nel formato a 24 ore.
Format(Date.Now, "mm")	Minuti.
Format(Date.Now, "ss")	Secondi.
Format(Date.Now, "f")	Data odierna in formato esteso, orario corrente in formato ridotto.
Format(Date.Now, "ff")	Centesimi di secondo.
Format(Date.Now, "fff")	Millesimi di secondo.
Format(Date.Now, "F")	Data odierna in formato esteso, orario corrente in formato esteso.

Format(Date.Now, "g")	Data odierna in formato ridotto, orario corrente in formato ridotto.
Format(Date.Now, "G")	Data odierna in formato ridotto, orario corrente in formato esteso.

Tabella 29: Principali modalità di formattazione di un orario.

Nel prossimo esercizio vedremo alcuni esempi di visualizzazione di date e orari.

Esercizio 72: Formattazione di data e orario correnti.

Apriamo un nuovo progetto.
Copiamo e incolliamo nella Finestra del Codice il listato seguente. Non è necessario inserire alcun controllo: il codice provvede a dimensionare il form, a posizionarlo nello schermo e a creare al suo interno i controlli necessari.

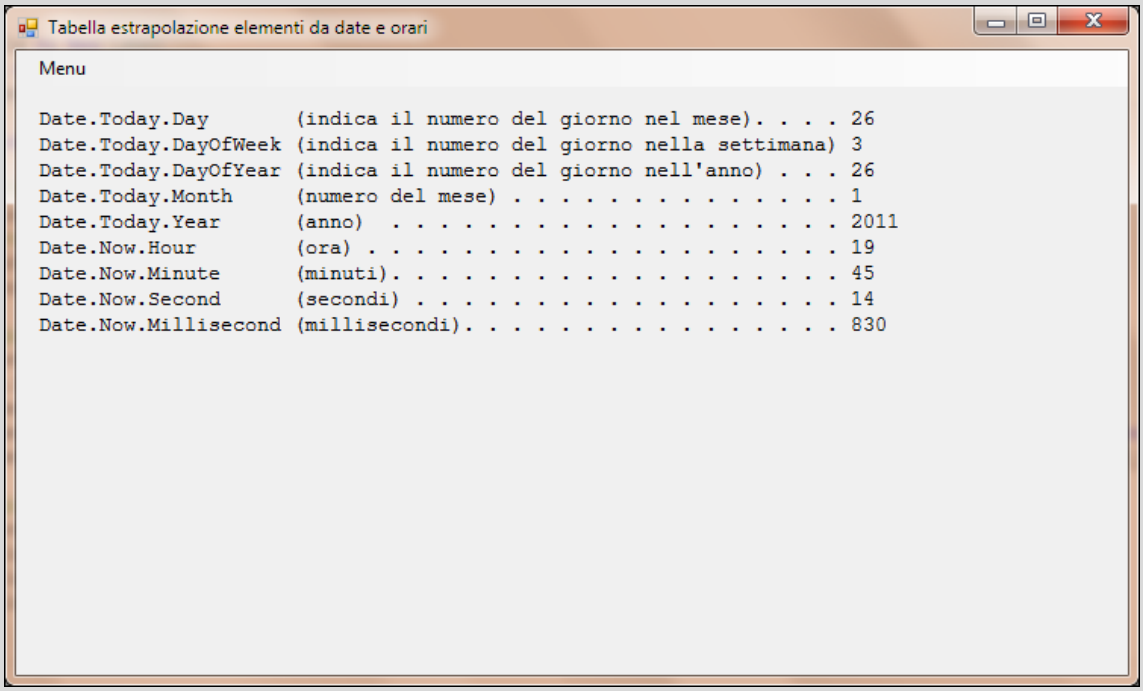
```
Public Class Form1

    Dim Tabella As New Label
    Dim Menu1 As New MenuStrip
    Dim MenuGenerale As New ToolStripMenuItem

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        'modifica le dimensioni del form
        Me.Size = New Size(740, 500)
        'centra il form nello schermo
        Dim Schermo As Rectangle = My.Computer.Screen.Bounds
        Me.Location = New Point((Schermo.Width - Me.Width) / 2, (Schermo.Height -
Me.Height) / 2)

        ' aggiungi al form un controllo MenuStrip con due item:
        Me.Controls.Add(Menu1)
        Menu1.Items.Add(MenuGenerale)
        MenuGenerale.Text = ("Menu")
        MenuGenerale.DropDownItems.Add("Estrapolazione di elementi da date e da
orari", Nothing, New EventHandler(AddressOf Tabella1_Click))
        MenuGenerale.DropDownItems.Add("Formattazione di date e di orari",
Nothing, New EventHandler(AddressOf Tabella2_Click))

        ' aggiungi al form un controllo label:
        Me.Controls.Add(Tabella)
        Tabella.Font = New Font("Courier New", 10)
        Tabella.Location = New Point(12, 54)
        Tabella.AutoSize = True
    End Sub

    Private Sub Tabella1_Click()
```

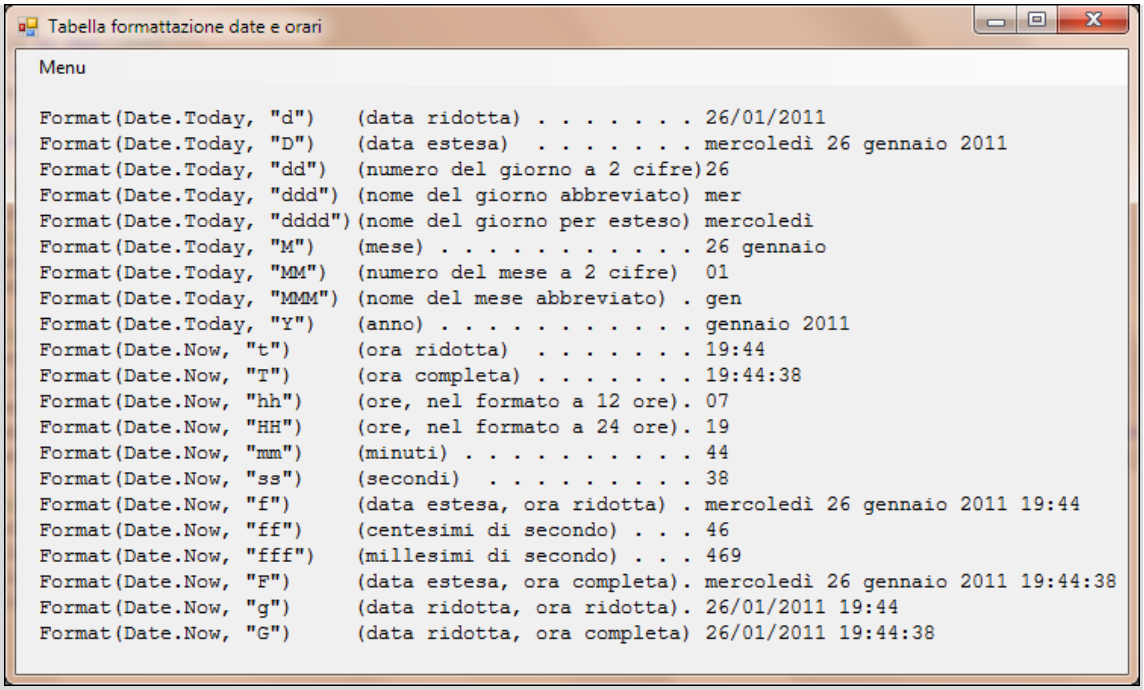
```
        Me.Text = "Tabella estrapolazione elementi da date e orari"
        Tabella.Text = ""
        Tabella.Text &= "Date.Today.Day       (indica il numero del giorno nel
mese). . . . " & Date.Today.Day & vbCrLf
        Tabella.Text &= "Date.Today.DayOfWeek (indica il numero del giorno nella
settimana) " & Date.Today.DayOfWeek & vbCrLf
        Tabella.Text &= "Date.Today.DayOfYear (indica il numero del giorno
nell'anno) . . . " & Date.Today.DayOfYear & vbCrLf
        Tabella.Text &= "Date.Today.Month     (numero del mese) . . . . . .
. . . . . " & Date.Today.Month & vbCrLf
        Tabella.Text &= "Date.Today.Year      (anno)  . . . . . .
. . . . . " & Date.Today.Year & vbCrLf
        Tabella.Text &= "Date.Now.Hour        (ora) . . . . . .
. . . . . " & Date.Now.Hour & vbCrLf
        Tabella.Text &= "Date.Now.Minute      (minuti). . . . . .
. . . . . " & Date.Now.Minute & vbCrLf
        Tabella.Text &= "Date.Now.Second      (secondi) . . . . . .
. . . . . " & Date.Now.Second & vbCrLf
        Tabella.Text &= "Date.Now.Millisecond (millisecondi). . . . . .
. . . . . " & Date.Now.Millisecond

    End Sub
```

```
    Private Sub Tabella2_Click()
        Me.Text = "Tabella formattazione date e orari"
        Tabella.Text = ""
        Tabella.Text &= "Format(Date.Today, ""d"")   (data ridotta) . . . . . .
" & Format(Date.Today, "d") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""D"")   (data estesa)  . . . . . .
" & Format(Date.Today, "D") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""dd"")  (numero del giorno a 2
cifre)" & Format(Date.Now, "dd") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""ddd"") (nome del giorno abbreviato)
" & Format(Date.Now, "ddd") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""dddd"")(nome del giorno per esteso)
" & Format(Date.Now, "dddd") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""M"")   (mese) . . . . . .
" & Format(Date.Now, "M") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""MM"")  (numero del mese a 2 cifre)
" & Format(Date.Now, "MM") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""MMM"") (nome del mese abbreviato) .
" & Format(Date.Now, "MMM") & vbCrLf
        Tabella.Text &= "Format(Date.Today, ""Y"")   (anno) . . . . . .
" & Format(Date.Now, "Y") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""t"")     (ora ridotta)  . . . . . .
" & Format(Date.Now, "t") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""T"")     (ora completa) . . . . . .
" & Format(Date.Now, "T") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""hh"")    (ore, nel formato a 12 ore).
" & Format(Date.Now, "hh") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""HH"")    (ore, nel formato a 24 ore).
" & Format(Date.Now, "HH") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""mm"")    (minuti) . . . . . .
" & Format(Date.Now, "mm") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""ss"")    (secondi)  . . . . . .
" & Format(Date.Now, "ss") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""f"")     (data estesa, ora ridotta) .
" & Format(Date.Now, "f") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""ff"")    (centesimi di secondo) . . .
" & Format(Date.Now, "ff") & vbCrLf
```

```
        Tabella.Text &= "Format(Date.Now, ""fff"")   (millesimi di secondo) . . .
" & Format(Date.Now, "fff") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""F"")     (data estesa, ora completa).
" & Format(Date.Now, "F") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""g"")     (data ridotta, ora ridotta).
" & Format(Date.Now, "g") & vbCrLf
        Tabella.Text &= "Format(Date.Now, ""G"")     (data ridotta, ora completa)
" & Format(Date.Now, "G")

    End Sub

End Class
```

Una volta in esecuzione, cliccando una delle due opzioni contenute nel Menu in alto è possibile visualizzare i risultati di diverse modalità di formattazione della data e dell'orario correnti:





Notiamo in questo esercizio che i controlli necessari al funzionamento del programma (la striscia di menu e il controllo label) non sono inseriti nel form dal programmatore nella fase di progettazione del programma, ma sono creati dal codice, all'avvio del programma.
Analizziamo, e teniamo come esempio, le modalità di creazione e di definizione delle proprietà del controllo Label:

```
Dim Tabella As New Label

' aggiungi al form il controllo Label:
Me.Controls.Add(Tabella)
Tabella.Font = New Font("Courier New", 10)
Tabella.Location = New Point(12, 54)
Tabella.AutoSize = True
```

119: Funzioni su date.

DateDiff

La funzione **DateDiff**() calcola la differenza tra due date in anni, mesi, settimane o giorni.
La sintassi del comando si compone di tre parametri scritti tra le parentesi.
Il primo parametro indica l'unità di misura della differenza (anni, mesi, settimane, giorni):

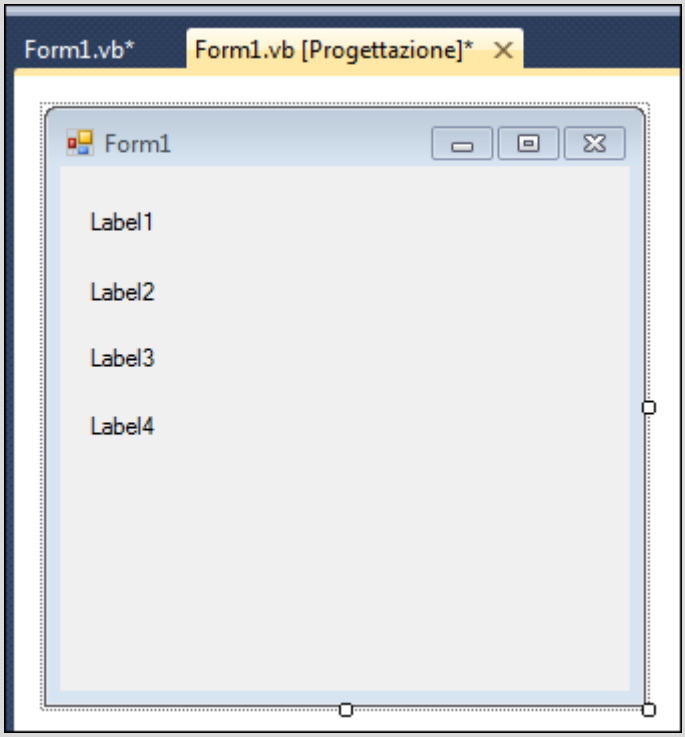
- per avere la differenza in anni bisogna specificare "yyyy" all'interno della funzione;

- per avere la differenza in mesi bisogna specificare “m”;
- per avere la differenza in settimane bisogna specificare “ww”;
- per avere la differenza in giorni bisogna specificare “d”.

Il secondo parametro indica la data iniziale per il calcolo della differenza.
Il terzo parametro indica la data finale.
Vedremo un esempio nel prossimo esercizio.

Esercizio 73: La funzione DateDiff

Apriamo un nuovo progetto e inseriamo nel form 4 controlli label:



Copiamo e incolliamo nella Finestra del Progetto questo codice:

```
Public Class Form1

    Private Sub Form1_Load() Handles Me.Load
        Me.Text = Date.Today

        Dim A, B, C, D As Integer

        A = DateDiff("yyyy", "01/01/2000", Date.Today)
        B = DateDiff("m", "01/01/2000", Date.Today)
        C = DateDiff("ww", "01/01/2000", Date.Today)
        D = DateDiff("d", "01/01/2000", Date.Today)

        Label1.Text = A.ToString
        Label2.Text = B.ToString
```

```
        Label3.Text = C.ToString
        Label4.Text = D.ToString

    End Sub

End Class
```

Mandiamo in esecuzione il progetto.
Al suo avvio (caricamento in memoria del Form e evento `Me.Load`) vengono visualizzati nelle quattro label:

- il numero degli anni trascorsi dal 1° Gennaio 2000 alla data corrente del computer;
- il numero dei mesi trascorsi dal 1° Gennaio 2000 alla data corrente del computer;
- il numero delle settimane trascorse dal 1° Gennaio 2000 alla data corrente del computer;
- il numero dei giorni trascorsi dal 1° Gennaio 2000 alla data corrente del computer.

DatePart

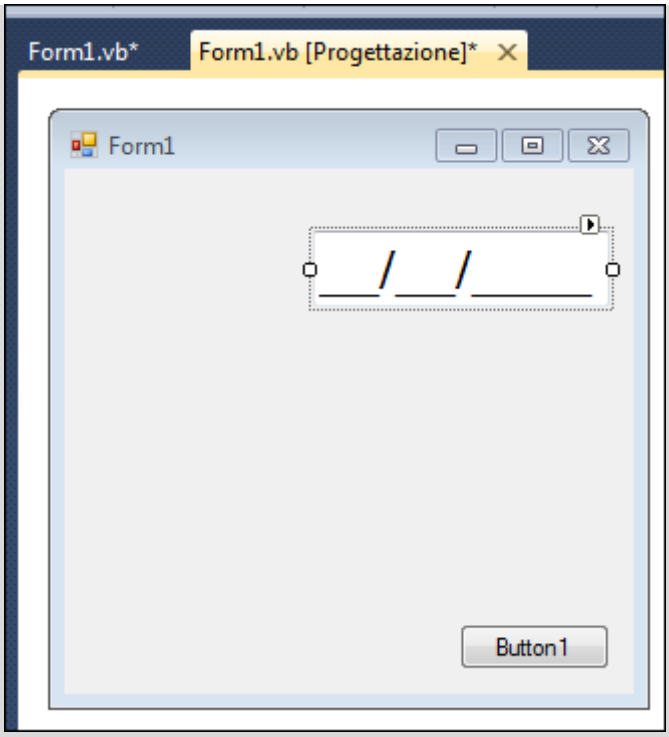
La funzione **DatePart**() estrae un singolo dato (giorno, mese o anno) all'interno di una data.
La sintassi del comando si compone di due parametri scritti tra le parentesi.
Il primo parametro deve indicare l'elemento da estrarre:

- per estrarre l'anno bisogna specificare "yyyy";
- per estrarre il mese bisogna specificare "m";
- per estrarre il numero del giorno (domenica = 1, sabato = 7) bisogna specificare "w".

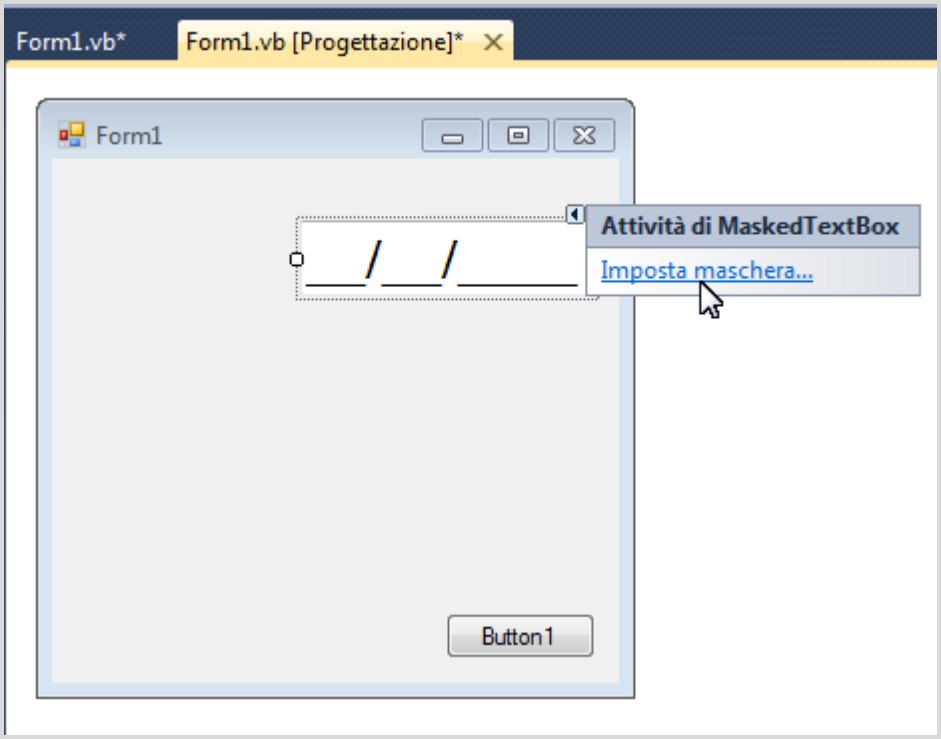
Il secondo parametro deve indicare la data da analizzare.
Il prossimo esercizio utilizza la funzione DatePart() per ricavare il nome del giorno da una data scritta dall'utente.

Esercizio 74: Che giorno era? Che giorno sarà?

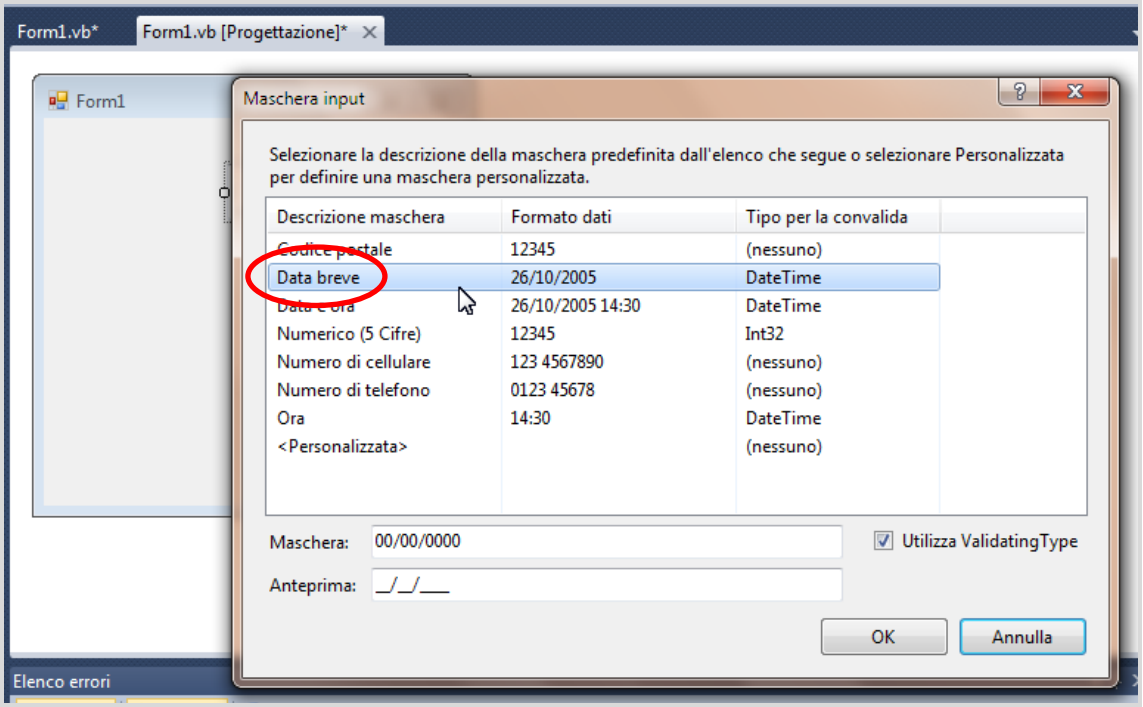
Questo programma indica a quale giorno della settimana corrisponde, o corrisponderà, una data immessa dall'utente del programma.
Il nome del giorno è rilevato dalla funzione DatePart("w", data immessa dall'utente).
L'immissione della data da parte dell'utente è guidata dall'uso di un controllo **MaskedTextBox** (contenitore di testo con un formato a maschera obbligata).
Apriamo un nuovo progetto e inseriamo nel form un pulsante **Button** e un controllo **MaskedTextBox**, come in questa immagine:



Impostiamo la proprietà **Font** del controllo MaskedTextBox1 **= Microsoft Sans Serif; 20,25pt.**
Facciamo un *clic* sul pulsante con la freccina nera in alto a destra del controllo **MaskedTextBox** e facciamo un *clic* su "**Imposta maschera…**":



Nella finestra che si apre, facciamo un *clic* su "**Data breve**", poi facciamo un *clic* su **OK**:



Ora copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click
        Dim DataDaEsaminare As Date
        Dim NumeroGiorno As Integer
        Dim Giorno As String

        ' controlla se la data immessa dall'utente è corretta:
        If IsDate(MaskedTextBox1.Text) Then

            ' se la data è corretta:
            DataDaEsaminare = MaskedTextBox1.Text

            ' estrapola il numero del gioro dalla data:
            NumeroGiorno = DatePart("w", DataDaEsaminare)

            ' trova il nome del giorno corrispondente al numero:
            Giorno = Choose(NumeroGiorno, "domenica", "lunedì", "martedì",
"mercoledì", "giovedì", "venerdì", "sabato")

            ' controlla se la data è precedente o successiva alla data corrente
            ' e scrive un messaggio con il tempo verbale corretto:
            If DateDiff("d", DataDaEsaminare, Date.Today) > 0 Then
                MsgBox("Il giorno " & DataDaEsaminare & " era " & Giorno & ".")
            ElseIf DateDiff("d", DataDaEsaminare, Date.Today) < 0 Then
                MsgBox("Il giorno " & DataDaEsaminare & " sarà " & Giorno & ".")
            Else
                MsgBox("Oggi " & DataDaEsaminare & " è " & Giorno & ".")
            End If
        Else
```

```
            ' se la data non è corretta:
            MsgBox("La data non è corretta!")

        End If

    End Sub

End Class
```

I commenti all'interno del codice consentono di capirne il funzionamento.
Il comando chiave è l'uso della funzione **DatePart**() per estrarre, dalla variabile **DataDaEsaminare** il numero del giorno:

```
            NumeroGiorno = DatePart("w", DataDaEsaminare)
```

La funzione **Choose**() viene poi utilizzata per fare corrispondere il nome del giorno della settimana al numero contenuto nella variabile NumeroGiorno.

120: Date e culture.

Le modalità di visualizzazione di date e di orari che abbiamo visto nelle pagine precedenti si riferiscono al nostro calendario ed alla nostra cultura.
Se in un programma si presenta l'esigenza di visualizzare calendari e modi di scrittura delle date diversi dai nostri, è necessario ricorrere alla funzione **.ToString**(), inserendo tra le parentesi la sigla della cultura alla quale si vuole fare riferimento.
Questo esempio visualizza la data corrente, in formato esteso, secondo la cultura araba:

```
        Label1.Text = Date.Today.ToString("D", New CultureInfo("ar"))
```

L'esercizio seguente visualizza la data corrente del computer secondo tutte le culture presenti nel sistema operativo.

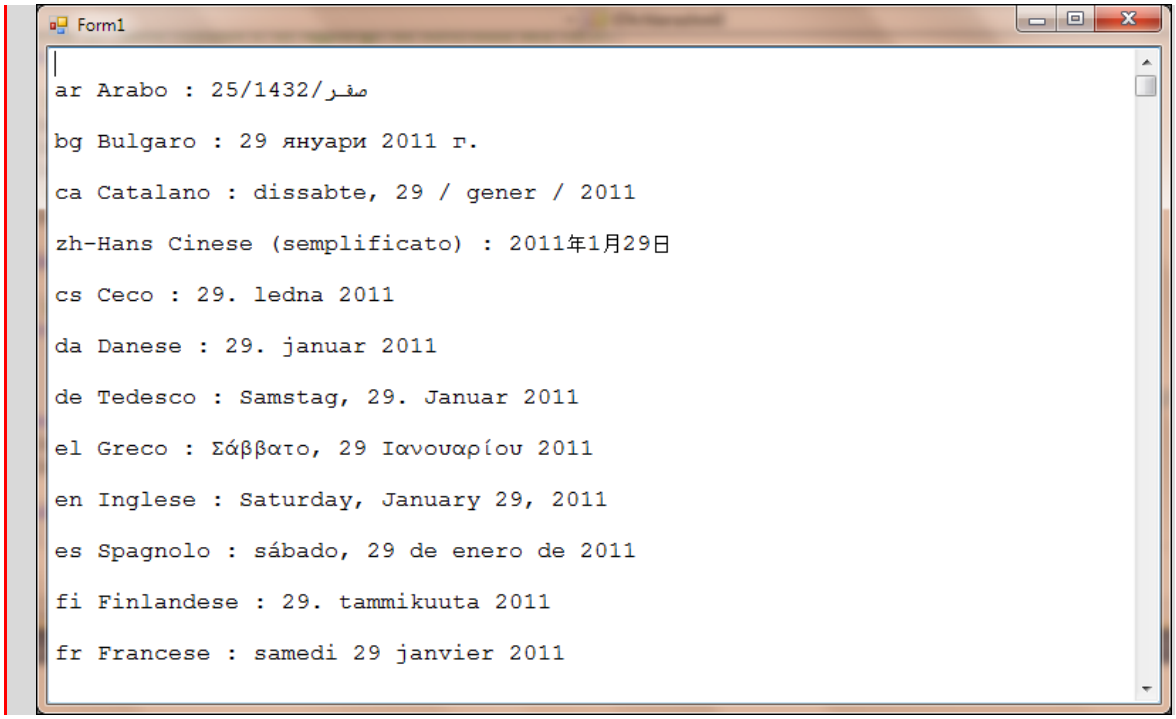
Esercizio 75: Date e culture presenti nel sistema operativo.

Questo programma visualizza all'interno di un TextBox la data corrente del computer secondo tutte le culture presenti nel sistema operativo.
Apriamo un nuovo progetto e incolliamo nella Finestra del Codice il listato seguente.
Non è necessario inserire alcun controllo: il codice dimensiona il form, lo posiziona nello schermo e crea al suo interno l'unico controllo necessario (un TextBox).

```
' Questa dichiarazione si colloca nella sezione Generale del codice (spazio
dei nomi), per importare il sistema di software globale:
Imports System.Globalization
```

```
Public Class Form1
    Dim ElencoDate As New TextBox

    Private Sub Form1_Load() Handles MyBase.Load
        'rileva le dimensioni dello schermo
        Dim SpazioLibero As Rectangle = Screen.GetWorkingArea(New Point(0,
0))
        'dimensiona il Form1
        Me.Height = 500
        Me.Width = 800
        'centra il Form1 nello schermo:
        Me.Location = New Point((SpazioLibero.Width - Me.Width) / 2, 100)

        'crea un nuovo TextBox e lo aggiunge ai controlli del form:
        Me.Controls.Add(ElencoDate)

        ' impostazione delle proprietà del TextBox:
        ElencoDate.Font = New Font("Courier New", 12)
        ElencoDate.Multiline = True
        ElencoDate.ScrollBars = ScrollBars.Vertical
        ' àncora il TextBox alle dimensioni del form:
        ElencoDate.Dock = DockStyle.Fill

        Dim SiglaCultura As String

        ' scorre tutte le culture installate nel sistema
        For Each cultura As CultureInfo In
CultureInfo.GetCultures(CultureTypes.AllCultures)
            SiglaCultura = cultura.IetfLanguageTag
            ' scrive la sigla, il nome della cultura e la data nel TextBox:
            ElencoDate.Text &= vbCrLf & SiglaCultura & " " &
cultura.DisplayName & " : " & Date.Today.ToString("D", New
CultureInfo(SiglaCultura)) & vbCrLf
        Next
        ElencoDate.SelectionStart = 0
        ElencoDate.SelectionLength = 0

    End Sub

End Class
```

L'immagine seguente mostra il programma in esecuzione:


Form1
ar Arabo : 25/1432/صفر
bg Bulgaro : 29 януари 2011 г.
ca Catalano : dissabte, 29 / gener / 2011
zh-Hans Cinese (semplificato) : 2011年1月29日
cs Ceco : 29. ledna 2011
da Danese : 29. januar 2011
de Tedesco : Samstag, 29. Januar 2011
el Greco : Σάββατο, 29 Ιανουαρίου 2011
en Inglese : Saturday, January 29, 2011
es Spagnolo : sábado, 29 de enero de 2011
fi Finlandese : 29. tammikuuta 2011
fr Francese : samedi 29 janvier 2011

PARTE IV: GRAFICA.

Prima di avventurarci nell'affascinante mondo della grafica, vediamo alcuni elementi essenziali perché l'interfaccia[67] di un programma si presenti in modo *amichevole* agli occhi dell'utente.

Progettare l'interfaccia di un programma.

La presentazione delle informazioni generali va collocata nella parte sinistra in alto dello schermo, che cade per prima sotto l'occhio dell'utente: qui l'utente troverà, se e quando necessario, l'insieme delle opzioni per muoversi all'interno del programma.
Queste informazioni generali possono essere esposte in forma di menu oppure in una colonna, sulla sinistra del form, separata e colorata in modo diverso dal resto del form.
Nell'angolo in alto a destra va invece collocato il pulsante per la chiusura del programma, presente in tutte le finestre del sistema operativo Windows.

I menu lineari o a cascata (o a tendina) sono elementi fondamentali di un'interfaccia *amichevole*:

- devono contenere il numero minore di comandi possibile;
- non devono contenere gruppi di parole o frasi;
- devono presentare vicini tra di loro, o all'interno di uno stesso raggruppamento, i comandi che rimandano a operazioni simili;
- devono offrire la possibilità di tasti di accesso rapido (F1, F2, CTRL+X, ecc.);
- devono eventualmente articolarsi in sottomenu, per evitare lunghi elenchi di opzioni all'interno di un menu unico.

Nella scelta delle parole da inserire in un menu è opportuno sfruttare le parole comunemente in uso in ambiente Windows, come ad esempio:

[67] L'interfaccia di un programma è ciò che collega l'utente del programma al programma stesso, è l'insieme delle informazioni in genere sempre visibili sullo schermo, che mettono in comunicazione un programma con l' utente.

- File
- Nuovo
- Apri
- Chiudi
- Salva
- Salva con nome...

Utilizzando le stesse parole del sistema operativo, in tutti i comandi in cui ciò è possibile, si ottiene il risultato di renderli immediatamente familiari all'utente del programma.
Lo stesso vale per i tasti di accesso rapido ai comandi (*shortcut*): anche in questo caso in ambiente Windows si sono imposte delle convenzioni che è bene conservare e utilizzare. Le combinazione di tasti CTRL+X, CTRL+C e CTRL+V, ad esempio, sono ormai saldamente abbinate nell'uso comune ai comandi di *editing* **Taglia**, **Copia** e **Incolla**; inserendoli nei nostri programmi avremo la certezza di presentare all'utente qualcosa che gli è familiare.
È opportuno non eccedere con il numero di *shortcut*, poiché i tasti di scelta rapida, per essere efficaci, debbono essere interiorizzati dall'utente quasi come automatismi: è ovvio che più il loro numero è ridotto più facile ne è la memorizzazione.

Le immagini sono altri elementi fondamentali per creare un'interfaccia gradevole e facilmente comprensibile.
La loro sovrabbondanza tende tuttavia ad appesantire l'interfaccia e a renderne più confusa la percezione.
Effetti grafici gradevoli ed efficaci si possono ottenere anche sfruttando le potenzialità grafiche del testo: l'uso di caratteri appropriati può rendere accattivante ed esteticamente gradevole una *finestra*, anche se questa non contiene immagini.

Il criterio fondamentale da tenere presente per la scelta dei caratteri è ovviamente la loro **leggibilità**: bisogna evitare la scelta di caratteri troppo elaborati, che prima di essere letti devono essere decifrati e dunque distolgono l'attenzione dal messaggio veicolato dal testo.
Nel dubbio, è sempre meglio scegliere i caratteri a ***bastoncino***[68], quali Arial, Verdana, Calibri, che non hanno alcun orpello decorativo e sono sempre da preferire nei programmi rivolti a bambini.
Attenzione a non lasciarsi tentare da *cocktails* di font e di colori: è molto difficile equilibrare la presenza di font diversi, per cui è meglio limitarsi al massimo a due tipi di caratteri nella stessa *finestra*.
Un testo, per essere letto più facilmente, deve essere scritto con colore molto scuro su fondo molto chiaro: l'ideale è nero su bianco, ma anche blu su giallo risulta in determinate situazioni facilmente leggibile.
Le maiuscole devono essere riservate ai titoli o alle lettere iniziali delle frasi; le frasi scritte tutte in maiuscolo sono poco leggibili perchè le maiuscole interrompono il

[68] I caratteri a bastoncino, sono detti caratteri *sans serif*, cioè caratteri senza le appendici (in gergo tecnico le "*grazie*") che compaiono in molti font alla base e nella parte superiore delle lettere.

percorso normale dell'occhio, ma anche troppe parole con iniziali maiuscole creano un effetto retorico che toglie efficacia al messaggio.
Anche la sottolineatura disturba la lettura; per evidenziare una parola o una frase è preferibile utilizzare il *corsivo* o il **grassetto**.
L'allineamento del testo deve essere mantenuto a sinistra; solo in rari casi (i titoli) si può ricorrere al testo centrato, più difficile da leggere. Sono da evitare la giustificazione del testo, che lascia spazi non controllabili tra le parole, e l'allineamento a destra che è del tutto inusuale nella nostra cultura.

Le proporzioni ideali di un form devono essere in rapporto 4 : 3 (esempio: 800 pixel di larghezza, 600 di altezza).
Per convenzione in uso nei sistemi operativi Windows, la *luce* che *illumina* il form e gli oggetti in esso contenuti proviene dall'angolo in alto a sinistra; le eventuali ombre di oggetti inseriti dal programmatore nel form, dunque, devono essere orientate in direzione basso/destra.

System.Drawing

L'archivio di software System.Drawing contiene le classi fondamentali che gestiscono la grafica, le immagini, gli elementi tipografici.
Questo archivio è importato automaticamente da VB in ogni nuova applicazione, per cui il programmatore non deve fare nulla per attivarlo e usarlo.
Il discorso cambia se il programmatore vuole operare con strumenti personalizzati, quali pennelli a tratteggio, riempimenti con colori gradienti, elaborazioni di immagini e di scritte.
Le classi per gestire questi strumenti più elaborati non si trovano nell'archivio di base System.Drawing ma si trovano in tre archivi di software diversi:

- System.Drawing.**Drawing2D** (personalizzazione degli strumenti grafici);
- System.Drawing.**Imaging** (gestione delle immagini);
- System.Drawing.**Text** (gestione del disegno di scritte).

System.Drawing.Drawing2D

System.Drawing.Imaging

System.Drawing.Text

- L'archivio System.Drawing.Drawing2D contiene le classi per personalizzare gli strumenti di base per il disegno: **penna e pennello**.
- L'archivio System.Drawing.Imaging contiene gli strumenti per salvare le immagini.

- L'archivio System.Drawing.Text contiene gli strumenti per disegnare scritte in modo personalizzato.

Questi tre archivi **non sono caricati automaticamente** all'avvio di un nuovo progetto VB, per cui il programmatore, se intende usare le classi che ne fanno parte, deve importare il relativo archivio dichiarandolo nell'apertura del codice della sua applicazione, nell'area chiamata Generale o NameSpace (= spazio dei nomi).
Ecco un esempio:

```
Imports System.Drawing.Drawing2D

Imports System.Drawing.Imaging

Imports System.Drawing.Text

Public Class Form1

    ' dichiarazione / creazione di uno strumento pennello personalizzato:
    Dim Pennello As New PathGradientBrush(AreaDaColorare)

End Class
```

In alternativa, il programmatore può indicare l'archivio in cui si trova ogni classe, nel momento in cui la usa. Questo secondo metodo è preferibile in applicazioni semplici, in cui i richiami a queste classi sono poco frequenti, ed è il sistema che useremo negli esempi e negli esercizi di questa parte del manuale:

```
Public Class Form1

    Dim Pennello As New Drawing2D.PathGradientBrush(AreaDaColorare)

End Class
```

Capitolo 23: DISEGNARE LINEE E AREE.

VB consente di disegnare su qualsiasi oggetto che faccia parte di un'applicazione, sui form e sui controlli in essi contenuti, stampante compresa.
Le attività descritte in questo capitolo si collocano all'interno degli archivi di software **System.Drawing** e **System.Drawing.Drawing2D**:

- l'archivio System.Drawing contiene le classi fondamentali che gestiscono la grafica;
- l'archivio System.Drawing.Drawing2D contiene gli strumenti di cui il programmatore può avere bisogno per personalizzare gli strumenti di disegno di base (penna e pennello).

Ricordiamo che il primo archivio è caricato automaticamente in un progetto VB, per cui il programmatore non deve fare nulla per utilizzarne le classi, mentre il secondo archivio deve essere attivato dal programmatore, specificando di volta in volta il richiamo all'archivio **Drawing2D**, oppure, più semplicemente, **importando** questo archivio di software nella sezione Generale, o spazio dei nomi, nella Finestra del Codice del progetto.

Perché VB possa disegnare linee, curve, figure, deve ricevere dal programmatore i punti di riferimento necessari, che possono essere:

- per disegnare una linea: il punto iniziale e quello finale;
- per disegnare una serie di linee: il punto iniziale, il punto finale e una serie di punti intermedi;
- per disegnare un rettangolo o un'ellisse: la posizione dell'angolo superiore sinistro, la larghezza e l'altezza;
- per disegnare un arco: la posizione dell'angolo superiore sinistro, la larghezza, l'altezza e l'apertura in gradi;
- per disegnare una curva Bézier: il punto d'inizio, il punto finale e i due punti di controllo. Li vediamo in questa immagine:

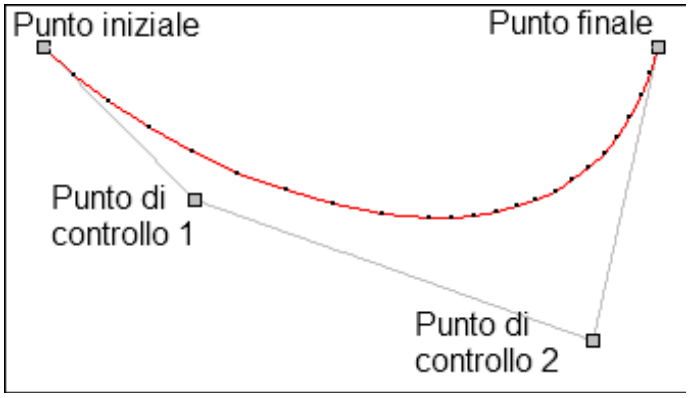


Figura 157: Una curva Bézier.

Gli strumenti essenziali per disegnare e colorare linee e aree sono:
- la classe **Graphics**, che individua o crea l'oggetto sul quale disegnare;
- la classe **Pen** (= penna), che determina il colore delle linee, la loro grandezza e il loro stile;
- la classe **Brush** (= pennello), che determina il modo con il quale debbono essere riempite o colorate le figure chiuse.

Le istruzioni per l'uso della penna e del pennello sono fornite dal programmatore con le **strutture Point, Size, Rectangle e Color** oltre alle classi **Brush** e **Pen.** Questi strumenti consentono di definire forme, dimensioni, stili e colori dei disegni tracciati.
Possiamo dunque dire, schematicamente, che:
- la classe **Graphics** crea un oggetto grafico;
- le classi **Pen** e **Brush** creano uno strumento di disegno (penna o pennello);
- le strutture **Point**, **Size , Rectangle e Color** definiscono caratteristiche, modalità e posizioni dello strumento di disegno.

Iniziamo la nostra analisi dall'ultimo punto, cioè dalla conoscenza delle quattro strutture Point, Size, Rectangle e Color, il che renderà più facile la comprensione del funzionamento delle classi Graphics, Pen e Brush.

121: La struttura Point.

La struttura **Point** memorizza la posizione di un punto sul piano.
Essa richiede l'indicazione di due parametri:
- la distanza del punto dal bordo sinistro (coordinata X) dell'oggetto che lo contiene;
- la distanza del punto dal bordo superiore (coordinata Y) dell'oggetto che lo contiene.

Ad esempio, questa riga indica un punto che si trova a 10 pixel di distanza dal bordo sinistro e a 20 pixel di distanza dal bordo superiore di un Form:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Dim NuovoPunto As New Point(10, 20)

    End Sub

End Class
```

Le due distanze possono anche essere scritte in questo modo:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Dim NuovoPunto As New Point
        NuovoPunto.X = 10 ' (distanza dal margine sinistro del Form)
        NuovoPunto.Y = 20 ' (distanza dal margine superiore del Form)

    End Sub

End Class
```

PointF.

Una struttura **PointF** è del tutto simile a una struttura **Point**, ma mentre in questa i valori delle coordinate X e Y sono valori Integer, cioè sono numeri interi, in una struttura PointF tali coordinate sono valori Single, ovvero sono numeri con la virgola, atti a definire la posizione di un punto in modo più preciso della struttura Point.
Questa struttura più precisa, con numeri decimali, è richiesta da VB per alcune funzioni che vedremo più avanti in alcuni esempi ed esercizi.

122: La struttura Size.

La struttura **Size** memorizza i dati relativi alle dimensioni di un'area rettangolare sullo schermo o all'interno di un oggetto.
Essa richiede l'indicazione di due parametri:

- la larghezza (**Width**) e
- l'altezza (**Height**) dell'area.

Ad esempio, questa riga definisce un'area rettangolare di 100 pixel di larghezza e 200 pixel di altezza:

```
        Dim Area1 As New Size(100, 200)
```

La larghezza e l'altezza dell'area possono anche essere scritte in questo modo:

```
Dim Area1 As New Size
Area1.Width = 100
Area1.Height = 200
```

ClientSize

La struttura **ClientSize** memorizza la larghezza e l'altezza di un controllo, escludendo gli elementi che non sono a disposizione del programmatore, quali i bordi, la barra del titolo e i menu.
Il Form1 che compare preimpostato all'apertura di un nuovo progetto, ad esempio, misura 300 pixel di larghezza e 300 pixel di altezza; la sua area utilizzabile, memorizzata nella struttura **ClientSize**, misura invece 284 pixel in larghezza e 262 pixel in altezza[69].

PreferredSize

La struttura **PreferredSize** adatta le dimensioni di un oggetto A (contenitore) alle dimensioni di un oggetto B, in esso contenuto, in modo che B sia visualizzato correttamente in A.
Supponiamo di avere un controllo Label1 contenuto in un contenitore GroupBox1: il comando

```
GroupBox1.Size = GroupBox1.PreferredSize
```

fa sì che le dimensioni del GroupBox si adattino alle dimensioni della Label1, in esso contenuta; più precisamente, la **larghezza** e l'**altezza** del contenitore GroupBox1 vengono modificate sino a visualizzare completamente la Label1 contenuta, ferma restando la posizione in cui si trova la Label1, che non viene modificata in alcun modo.

[69] Queste sono misure standard che possono differire secondo le impostazioni di barre e bordi delle finestre nel sistema operativo.

SizeF.

Quanto scritto sopra riguardo le differenze tra le strutture Point e PointF vale anche per quanto riguarda le differenze tra le strutture Size e SizeF.
La struttura SizeF, più precisa perché espressa in numeri con la virgola, è richiesta da VB, ad esempio, per il comando **MeasureString**, che vedremo più avanti, con il quale si memorizzano le dimensioni di un'area occupata da una scritta.

123: La struttura Rectangle.

La struttura **Rectangle** memorizza la posizione e le dimensioni di un'area rettangolare sullo schermo o all'interno di un oggetto.
Essa richiede l'indicazione di quattro parametri che indicano rispettivamente:

- la distanza del rettangolo dal bordo a sinistra del contenitore (coordinata **X**);
- la distanza del rettangolo dal bordo superiore del contenitore (coordinata **Y**);
- la larghezza (**Width**) del rettangolo;
- l'altezza (**Height**) del rettangolo.

Ad esempio, questa riga definisce un'area rettangolare che si trova a 10 pixel di distanza dal bordo sinistro del Form e a 20 pixel di distanza dal suo bordo superiore, di 100 pixel di larghezza e 200 pixel di altezza:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Dim Area1 As New Rectangle(10, 20, 100, 200)

    End Sub

End Class
```

I quattro parametri possono anche essere indicati in questo modo:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Dim Area1 As New Rectangle
        Area1.X = 10
        Area1.Y = 20
        Area1.Width = 100
        Area1.Height = 200

    End Sub

End Class
```

La struttura Rectangle è uno strumento adatto a memorizzare i dati relativi a posizione e dimensioni di oggetti che si trovano nel Form; questi dati possono essere ricavati in modi diversi:

- **utilizzando la proprietà Bounds** si memorizzano le coordinate della posizione di un controllo e le sue dimensioni, sino ai limiti esterni del controllo stesso;
- **utilizzando la proprietà ClientRectangle** si memorizzano le dimensioni di un controllo, escludendone le parti di cui il programmatore non può disporre: barre di scorrimento, bordi, barra del titolo;
- **utilizzando la proprietà DisplayRectangle** si memorizzano le dimensioni dell'area visibile di un controllo.

Per la maggior parte dei controlli, le proprietà ClientRectangle e DisplayRectangle memorizzano gli stessi dati; fanno eccezione alcuni controlli contenitori, come il controllo GroupBox.

Ecco un esempio relativo ad un Form1:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim Area1 As Rectangle = Me.Bounds
        Dim Area2 As Rectangle = Me.ClientRectangle
        Dim Area3 As Rectangle = Me.DisplayRectangle

        Label1.Text &= Area1.ToString & vbCrLf
        Label1.Text &= Area2.ToString & vbCrLf
        Label1.Text &= Area3.ToString & vbCrLf

    End Sub

End Class
```

Con questo listato, relativo al Form1 che VB crea automaticamente all'apertura di un nuovo progetto, le tre strutture Area1, Area2 e Area3 memorizzano questi dati:

- **Area1**: (x, y, 300, 300)
- **Area2**: (0, 0, 284, 262)
- **Area3**: (0, 0, 284, 262)

Ecco un esempio relativo ad un GroupBox:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim Area1 As Rectangle = GroupBox1.Bounds
        Dim Area2 As Rectangle = GroupBox1.ClientRectangle
        Dim Area3 As Rectangle = GroupBox1.DisplayRectangle

        Label1.Text &= Area1.ToString & vbCrLf
        Label1.Text &= Area2.ToString & vbCrLf
        Label1.Text &= Area3.ToString & vbCrLf
```

```
    End Sub

End Class
```

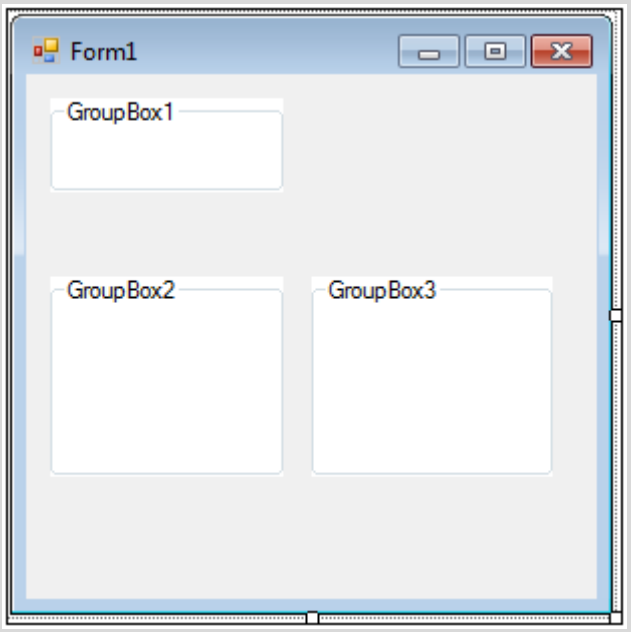
Con questo listato, relativo al Form1 che VB crea automaticamente all'apertura di un nuovo progetto, le strutture Area1, Area2 e Area3 memorizzano questi dati:

- **Area1**: (x, y, 211, 114)
- **Area2**: (0, 0, 211, 114)
- **Area3**: (0, 0, 205, 95)

Notiamo che, per il Form le proprietà ClientRectangle e DisplayRectangle memorizzano gli stessi dati, mentre i dati di ClientRectangle e DisplayRectangle sono diversi per il GroupBox.
Nel prossimo esercizio vedremo l'utilizzo delle strutture Rectangle per memorizzare le posizioni e le dimensioni di alcuni controlli Label contenuti in tre controlli GroupBox diversi.

Esercizio 76: Bounds, PreferredSize, ClientRectangle, DisplayRectangle.

In questo esercizio vedremo l'utilizzo delle proprietà Bounds, PreferredSize, ClientRectangle, DisplayRectangle per memorizzare e modificare le dimensioni di alcune Label contenute in tre **GroupBox**.
Apriamo un nuovo progetto e inseriamo nel form tre controlli GroupBox come in questa immagine:

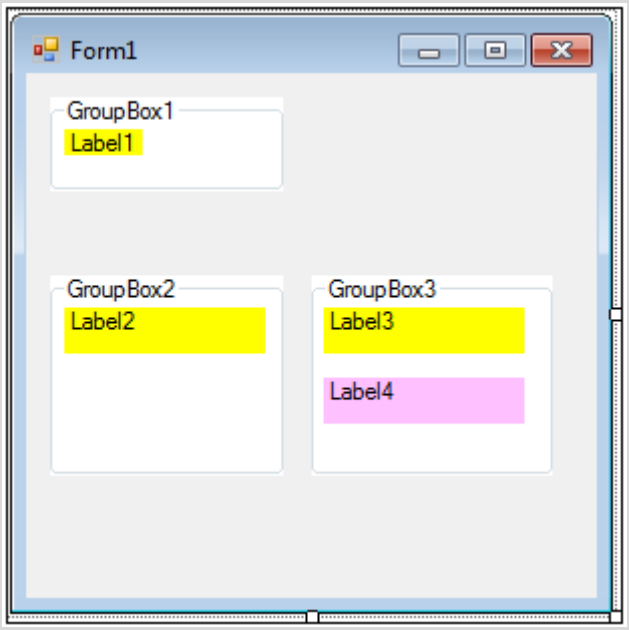


Ora inseriamo all'interno dei controlli **GroupBox** quattro controlli Label:

- **Label1** nel GroupBox1,
- **Label2** nel GroupBox2,
- **Label3** e **Label4** nel GroupBox3.

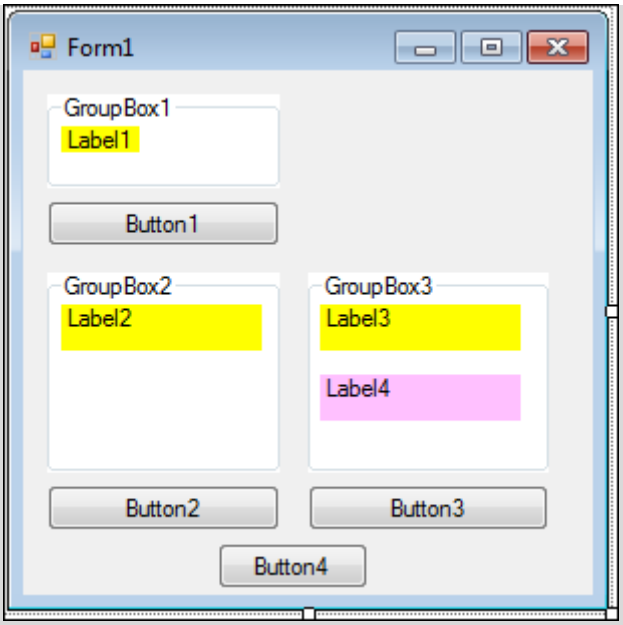
Impostiamo

- la proprietà **AutoSize** delle Label2, Label3 e Label4 = **False** e
- la proprietà **BackColor** delle quattro Label con i colori che si vedono in questa immagine:



Completiamo l'interfaccia del programma aggiungendo quattro pulsanti **Button**:

- Il primo, **Button1**, sarà utilizzato per modificare la Label1 nel GroupBox1.
- Il secondo, **Button2**, sarà utilizzato per la Label 2 nel GroupBox2.
- Il terzo, **Button3**, sarà utilizzato per la Label 3 e la Label4 nel Group3.
- Il quarto pulsante, **Button4**, servirà ad annullare le modifiche effettuate con gli altri pulsanti e a ripristinare la situazione di partenza del programma.



Facciamo un doppio *clic* sul Form1 e accediamo alla Finestra del Codice, dove troviamo già impostata la procedura che gestisce l'evento del caricamento del form all'avvio del programma:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

End Class
```

In questa procedura creiamo le variabili necessarie al funzionamento del programma.
I *clic* sui pulsanti **Button2** e **Button3** modificheranno le dimensioni della **Label2**, della **Label3** e della **Label4;** ora, perché il *clic* sul pulsante **Button4** faccia tornare tutto come prima è necessario che il programma memorizzi in partenza le posizioni e le dimensioni delle tre label.
La struttura **Rectangle** si presterà a questa operazione.
Ricordiamo che una struttura **Rectangle** definisce un'area di forma rettangolare all'interno del form, memorizzandone questi dati:

- la posizione **Left** (distanza dal bordo sinistro del form),
- la posizione **Top** (distanza dal bordo superiore del form),
- la larghezza e
- l'altezza.

All'avvio del nostro programma, creiamo dunque tre strutture **Rectangle**:

- **Rettangolo2**
- **Rettangolo3**
- **Rettangolo4**

In queste strutture memorizziamo, rispettivamente, la posizioni e le dimensioni della **Label2**, **Label3** e **Label4**.

La posizione e le dimensioni di un controllo sono memorizzati nella proprietà **Bounds** (confini); la proprietà **Label2.Bounds,** ad esempio, memorizza i quattro dati che definiscono la posizione e la dimensioni della Label2:

- la posizione **Left** (distanza dal bordo sinistro del form),
- la posizione **Top** (distanza dal bordo superiore del form),
- la sua larghezza e
- la sua altezza.

Le righe di comandi che seguono creano la struttura Rectangle2 e le assegnano i dati relativi alla posizione ed alle dimensioni della Label2:

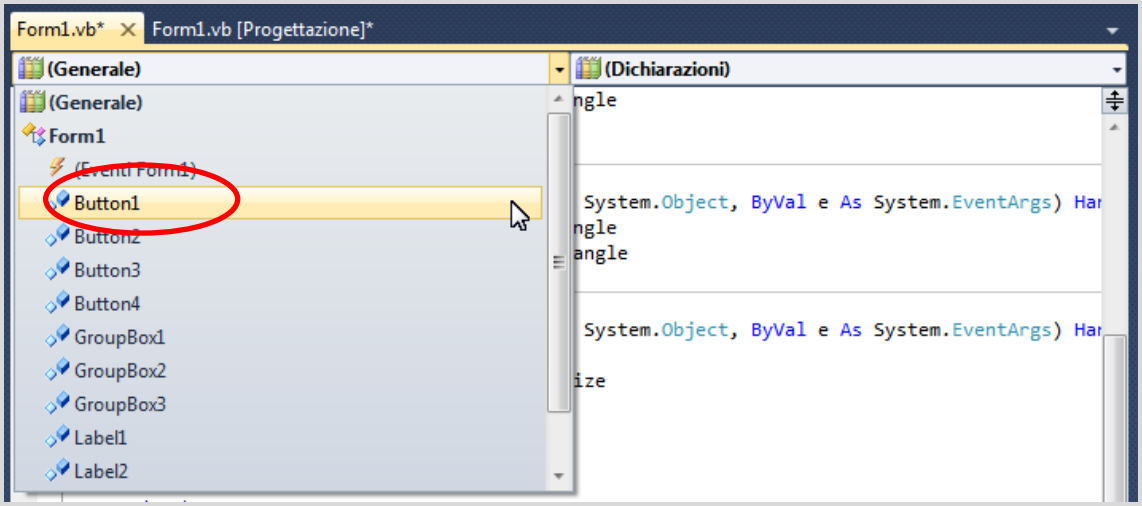
```
    Dim Rettangolo2 As New Rectangle
    Rettangolo2 = Label2.Bounds
```

Il codice seguente ripete la stessa operazione per tutte le Label, memorizzando le loro posizioni e dimensioni all'avvio del programma:

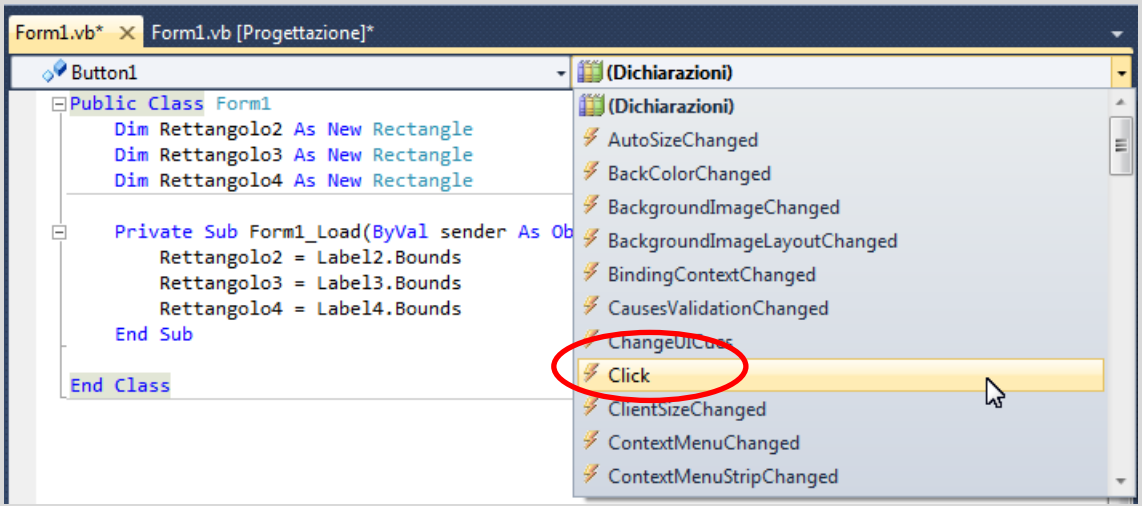
```
Public Class Form1

    ' Crea tre variabili di tipo Rectangle, serviranno a memorizzare le
dimensioni originali delle Label:
    Dim Rettangolo2 As New Rectangle
    Dim Rettangolo3 As New Rectangle
    Dim Rettangolo4 As New Rectangle

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        ' Memorizza le dimensioni originali delle Label:
        Rettangolo2 = Label2.Bounds
        Rettangolo3 = Label3.Bounds
        Rettangolo4 = Label4.Bounds

        ' Imposta il testo della Label1 e adatta le dimensioni del GroupBox1:
        Label1.Text = "PreferredSize"
        GroupBox1.Size = GroupBox1.PreferredSize

        ' Imposta il testo delle Label e dei pulsanti:
        Label2.Text = ""
        Label3.Text = ""
        Label4.Text = ""
        Button1.Text = "PreferredSize"
        Button2.Text = "ClientRectangle"
        Button3.Text = "DisplayRectangle"
        Button4.Text = "Ripristina"

    End Sub

End Class
```

Passiamo ora alla scrittura delle procedure che gestiscono gli eventi dei *clic* sui quattro pulsanti, iniziando dal Button1.

Nella Finestra del Codice, facciamo un *clic* con il mouse sul menu **Generale** che contiene l'elenco degli oggetti; nel menu a tendina che si apre facciamo un *clic* sul Button1.



Nel menu a destra, facciamo un *clic* sul menu delle **Dichiarazioni**; qui troviamo un elenco degli eventi che il **Button1** è un grado di riconoscere. Facciamo un *clic* sull'evento **Click** e completiamo la procedura nel modo seguente:




```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        If Label1.Text = "PreferredSize" Then
            Label1.Text = "Il GroupBox1 si adatta alle dimensioni della Label1"
        Else
            Label1.Text = "PreferredSize"
        End If
        GroupBox1.Size = GroupBox1.PreferredSize

    End Sub
```

All'interno della procedura abbiamo aggiunto queste istruzioni:

- Se il testo della Label2 è uguale a “PreferredSize”, allora cambialo con "Il GroupBox1 si adatta alle dimensioni della Label1", e viceversa.
- Siccome la proprietà **AutoSize** della Label1 è impostata **= True**, la lunghezza della Label1 si adatta al testo in essa contenuto.
- L’ultima riga della procedura fa sì che il componente **GroupBox1**, che contiene la **Label1**, si adatti alla nuova dimensione della Label1 (la proprietà **PreferredSize** adatta la posizione e le dimensioni di un controllo alla posizione ed alle dimensioni del controllo in esso contenuto).

Torniamo a cliccare il menu **Generale** e il menu **Dichiarazioni** per scrivere la procedura relativa all’evento del *clic* sul Button2:

```
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

        Label2.Text = "ClientRectangle"
        Label2.Bounds = GroupBox2.ClientRectangle

    End Sub
```

Le istruzioni scritte in questa procedura fanno assumere alla Label2 la posizione e le dimensioni del GroupBox2 che la contiene. Più precisamente, la Label2 assume le dimensioni dell’area ClientRectangle, che nel caso del controllo GroupBox vanno sino al suo limite **esterno**.

Torniamo a cliccare il menu Generale e il menu Dichiarazioni per scrivere la procedura relativa all’evento del *clic* sul Button3:

```
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click

        Label3.Text = "ClientRectangle"
        Label4.Text = "DisplayRectangle"
        Label3.Bounds = GroupBox3.ClientRectangle
        Label4.Bounds = GroupBox3.DisplayRectangle

    End Sub
```

Le istruzioni inserite in questa procedura fanno assumere alla Label3 e alla Label4 la posizione e le dimensioni del GroupBox3 che le contiene.

Con una differenza:

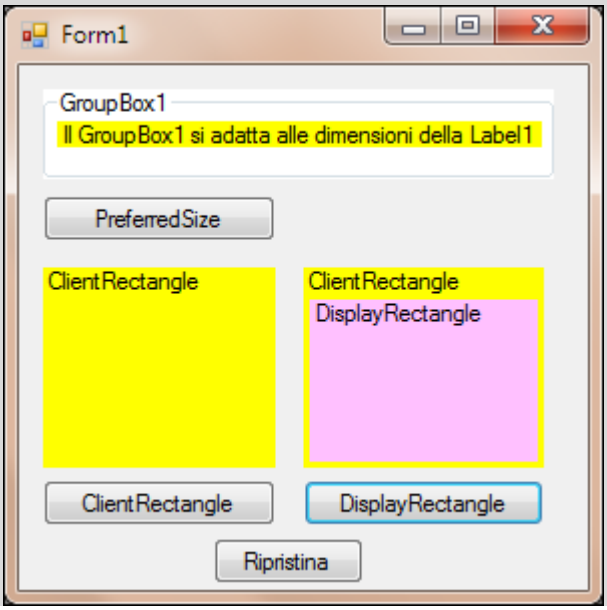
- la Label3 assume posizione e dimensioni dell’area ClientRectangle, che nel caso del controllo GroupBox vanno sino al suo limite **esterno**.
- la Label 4 assume posizione e dimensioni dell’area **DisplayRectangle**, che, nel caso del controllo GroupBox, vanno sino al suo limite **interno**.

Torniamo a cliccare il menu Generale e il menu Dichiarazioni per scrivere la procedura relativa all’evento del *clic* del mouse sul Button4, con le istruzioni per fare tornare tutti i controlli al loro stato iniziale:

```
    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click

        Label2.Bounds = Rettangolo2
```

```
        Label3.Bounds = Rettangolo3
        Label4.Bounds = Rettangolo4
        Label1.Text = "PreferredSize"
        GroupBox1.Size = GroupBox1.PreferredSize

        Label2.Text = ""
        Label3.Text = ""
        Label4.Text = ""

    End Sub
```

Ecco un'immagine del programma in esecuzione:



RectangleF.

Una struttura **RectangleF** è del tutto simile a una struttura **Rectangle**, ma mentre in questa i parametri sono espressi in numeri interi (Integer), in una struttura **RectangleF** sono espressi in numeri con la virgola (Single), atti a definire posizione e dimensioni di un'area in modo più preciso della struttura Rectangle.

124: Funzioni della struttura Rectangle.

Con la struttura Rectangle è possibile utilizzare queste funzioni che permettono di creare effetti e operazioni grafiche:

- Inflate
- Contains

- IntersectsWith
- Intersect.

Inflate

Il comando **Inflate(X,Y**) *gonfia* una struttura Rectangle, aumentandone la larghezza e l'altezza dei numeri di pixel indicati tra parentesi.
Eccone un esempio:

```
        Dim Area1 As New Rectangle(10, 20, 100, 200)
        Area1.Inflate(5, 5)
```

Dopo il comando Inflate(5, 5), l'area rettangolare sarà larga 105 pixel e alta 205 pixel.
Il comando Inflate può anche ridurre le dimensioni di una struttura Rectangle, impostando le quantità di pixel con numeri negativi:

```
        Dim Area1 As New Rectangle(10, 20, 100, 200)
        Area1.Inflate(-5, -5)
```

Contains

La funzione **Contains** verifica se una struttura Rectangle contiene un punto determinato o un'altra struttura Rectangle.
La sua sintassi è Rectangle1.Contains(Rectangle2).
Ecco due esempi che è possibile provare aprendo un nuovo progetto e copiando e incollando i listati nella Finestra del Codice.
Nel primo esempio, il comando Contains è utilizzato per verificare se una struttura Rectangle contiene un determinato punto:

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        Dim Area1 As New Rectangle(20, 20, 200, 100)
        Dim Punto As New Point(1, 1)
        If Area1.Contains(Punto) Then
            e.Graphics.FillRectangle(Brushes.Green, Area1)
        Else
            e.Graphics.FillRectangle(Brushes.Yellow, Area1)
        End If
    End Sub

End Class
```

In questo caso il punto non è contenuto nell'Area1, per cui l'Area1 è colorata di giallo.

Nel secondo esempio, il comando Contains è utilizzato per verificare se una struttura Rectangle A è contenuta in una struttura Rectangle B:

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Area1 As New Rectangle(20, 20, 200, 100)
        Dim Area2 As New Rectangle(50, 50, 50, 50)
        If Area1.Contains(Area2) Then
            e.Graphics.FillRectangle(Brushes.Green, Area1)
        Else
            e.Graphics.FillRectangle(Brushes.Yellow, Area1)
        End If

    End Sub

End Class
```

In questo caso l'Area2 è contenuta nell'Area1, per cui l'Area1 è colorata di verde.

IntersectsWith

La funzione **IntersectsWith** è utilizzata per determinare se due strutture Rectangle si intersecano una con l'altra, e dunque se hanno una parte in comune.
In questo esempio abbiamo due strutture Rectangle, Area1 e Area2, utilizzate per disegnare due rettangoli, il primo con il contorno rosso, il secondo con il contorno verde.
Il comando IntersectsWith è utilizzato per visualizzare il messaggio se i due rettangoli si intersecano o no:

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Area1 As New Rectangle(10, 10, 150, 150)
        Dim Area2 As New Rectangle(50, 50, 150, 150)

        e.Graphics.DrawRectangle(Pens.Red, Area1)
        e.Graphics.DrawRectangle(Pens.Green, Area2)

        If Area2.IntersectsWith(Area1) Then MsgBox("I due rettangoli si
intersecano.")

    End Sub

End Class
```

Intersect

Quando due strutture **Rectangle** si sovrappongono in parte una all'altra, il comando **Intersect** indica l'area rettangolare comune che le interseca.
L'area individuata da **Intersect** può essere utilizzata per modificare una delle due strutture che si intersecano e farle assumere le dimensioni dell'area in comune.
Ecco un esempio che è possibile provare aprendo un nuovo progetto e copiando e incollando il listato nella Finestra del Codice:

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Area1 As New Rectangle(10, 10, 150, 150)
        Dim Area2 As New Rectangle(50, 50, 150, 150)

        e.Graphics.DrawRectangle(Pens.Red, Area1)
        e.Graphics.DrawRectangle(Pens.Green, Area2)

        Area2.Intersect(Area1)
        e.Graphics.FillRectangle(Brushes.Green, Area2)

    End Sub
End Class
```

Nell'esempio abbiamo due strutture di tipo Rectangle, denominate **Area1** e **Area2**, utilizzate per disegnare due rettangoli, il primo con il bordo rosso, il secondo con il bordo verde. Il comando **Intersect,** che individua l'area comune ai due rettangoli, viene utilizzato per evidenziare la zona di intersezione delle due aree.
La zona di intersezione è colorata di verde:

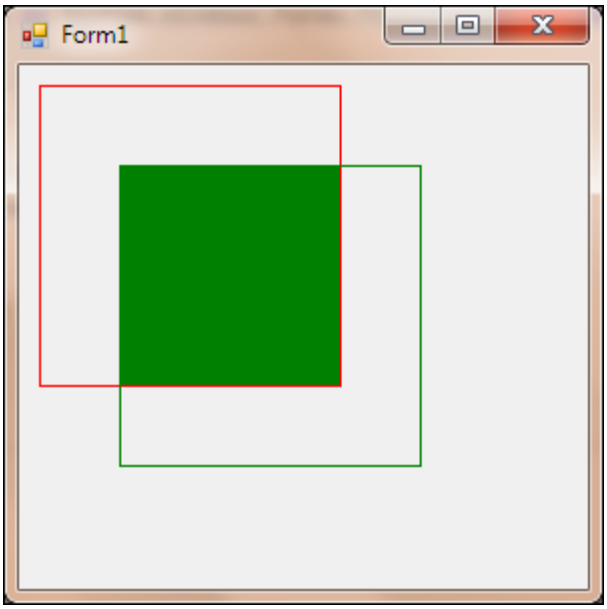


Figura 158: Il comando Intersect.

125: La struttura Color.

La struttura **Color** memorizza i dati relativi al colore che il programmatore vuole usare per una linea o un'area.
Per definire un colore è necessario indicare quattro parametri, le cui iniziali formano l'acronimo **ARGB** del colore:

- la proprietà **Alpha**;
- la proprietà **Red**;
- la proprietà **Green**;
- la proprietà **Blue**.

Il parametro **Alpha** indica il livello di trasparenza del colore, dalla tonalità completamente trasparente alla tonalità solida, su una scala che va da 0 (= massima trasparenza) a 255 (= nessuna trasparenza). La sua impostazione è facoltativa e può essere omessa. In caso di omissione, VB assegna ad Alpha il valore 255 (= nessuna trasparenza).
I parametri **Red**, **Green** e **Blue** indicano rispettivamente i livelli di colore Rosso, Verde e Blu che concorrono a formare il colore scelto. Ogni livello di colore va da 0 a 255. Ad esempio, questa riga definisce il colore nero, con il parametro Alpha (trasparenza) impostata sul livello 128 (= trasparenza al 50%) e i parametri Red, Green e Blue impostati sul livello 0:

```
Dim ColoreScelto = Color.FromArgb(128, 0, 0, 0)
```

I parametri Red, Green e Blue per comporre i colori principali sono elencati in questa tabella:

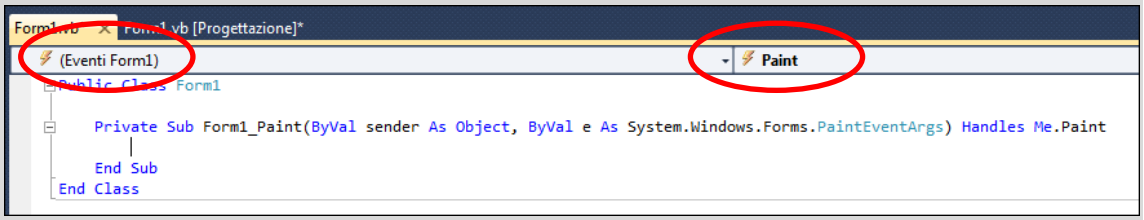
Colore	Esempio	Red	Green	Blue
nero	`Color.FromArgb(128, 0, 0, 0)`	**0**	**0**	**0**
blu	`Color.FromArgb(128, 0, 0, 255)`	**0**	**0**	**255**
verde	`Color.FromArgb(128, 0, 255, 0)`	**0**	**255**	**0**
cyan (azzurro)	`Color.FromArgb(128, 0, 255, 255)`	**0**	**255**	**255**
rosso	`Color.FromArgb(128, 255, 0, 0)`	**255**	**0**	**0**
magenta (fucsia)	`Color.FromArgb(128, 255, 0, 255)`	**255**	**0**	**255**
giallo	`Color.FromArgb(128, 255, 255, 0)`	**255**	**255**	**0**
bianco	`Color.FromArgb(128, 255, 255, 255)`	**255**	**255**	**255**

Tabella 30: I livelli Red, Green, Blue che compongono i colori principali.

VB offre tuttavia un'ampia rosa di colori *preconfezionati* che possono essere scelti indicando semplicemente il loro nome; l'elenco di questi colori è suggerito da **IntelliSense** mentre si scrive il codice di una proprietà che richieda l'indicazione di un colore.
Ne vediamo un esempio nel prossimo esercizio.

Esercizio 77: La struttura Color.

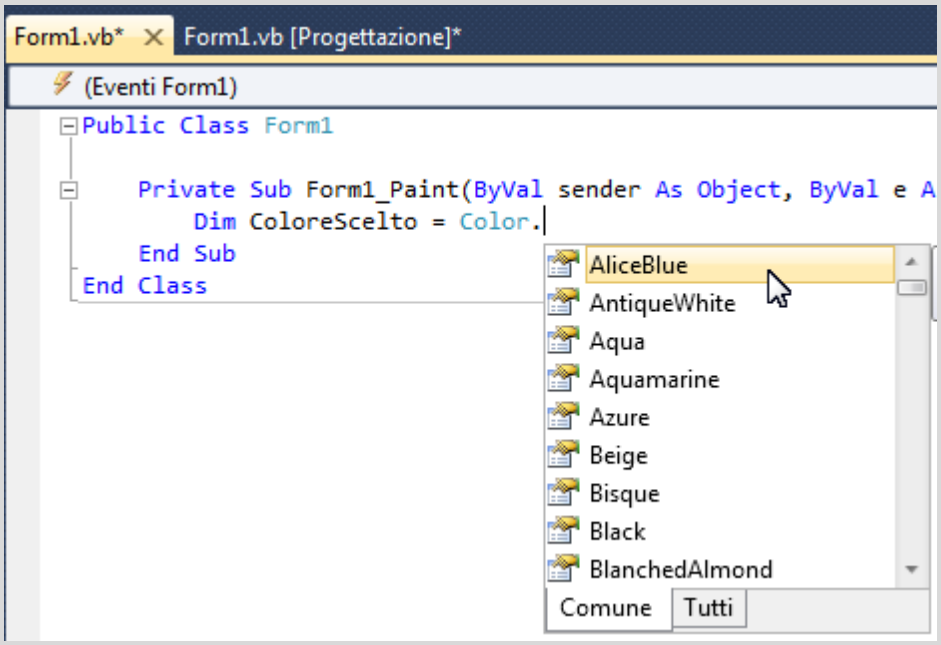
Apriamo un nuovo progetto e inseriamo nel Form1 un controllo **Button1**.
Apriamo la Finestra del Codice e tra gli eventi del Form1 facciamo un *clic* sull'evento Paint:



All'interno della procedura Form1_Paint iniziamo a scrivere questa riga di codice:

```
Dim ColoreScelto = Color.
```

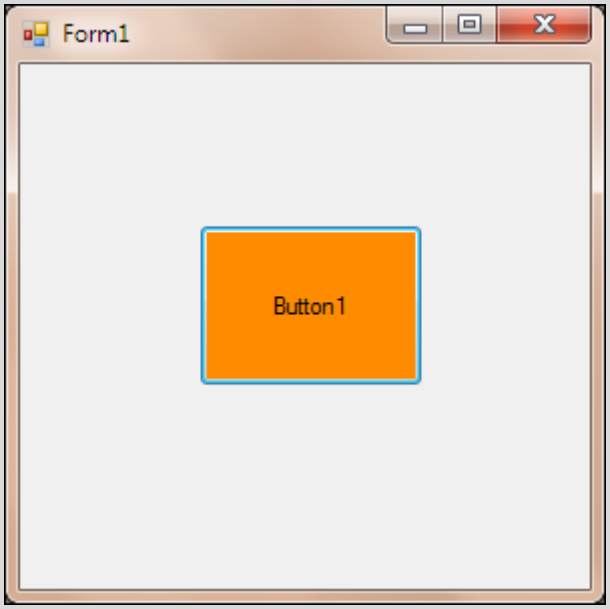
Notiamo che dopo avere scritto il punto **IntelliSense** visualizza l'elenco dei colori disponibili:



In questo elenco possiamo scegliere un colore e assegnarlo al Button1:

```
Public Class Form1

    Private Sub Form1_Paint() Handles Me.Paint

        Dim ColoreScelto = Color.DarkOrange
        Button1.BackColor = ColoreScelto

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



In questo modo abbiamo potuto scegliere un colore senza indicarne i valori R,G,B.
Se vogliamo modificare il livello di trasparenza del colore possiamo indicarne il valore Alpha aggiungendo una riga di istruzioni:

```
Public Class Form1

    Private Sub Form1_Paint() Handles Me.Paint

        Dim ColoreScelto = Color.DarkOrange
        ColoreScelto = Color.FromArgb(128, ColoreScelto)

        Button1.BackColor = ColoreScelto

    End Sub

End Class
```

126: La classe Graphics.

La classe Graphics fornisce gli strumenti per:
- individuare o creare una superficie grafica;
- disegnare su questa superficie.

Ogni controllo di VB ha una sua superficie grafica sulla quale il programmatore può disegnare linee, curve, forme, riempire aree, visualizzare immagini, disegnare scritte, il tutto con trasformazioni varie.
Una superficie grafica può essere creata dal programmatore oppure può essere individuata in modo automatico a VB:

- Una superficie grafica può essere creata dal programmatore con il comando **CreateGraphics**. Le tre linee di codice che seguono, ad esempio, creano tre superfici grafiche corrispondenti a tre oggetti (PictureBox1, Button1 e Label1) già collocati nel form.

```
Dim Superficie1 As Graphics = PictureBox1.CreateGraphics
Dim Superficie2 As Graphics = Button1.CreateGraphics
Dim Superficie3 As Graphics = Label1.CreateGraphics
```

- Una superficie grafica è individuata automaticamente da VB, per il form e per ogni controllo in esso contenuto, quando si attiva l'evento **Paint** di un oggetto (l'evento Paint di un oggetto si verifica ogni volta che questo oggetto viene visualizzato o modificato).

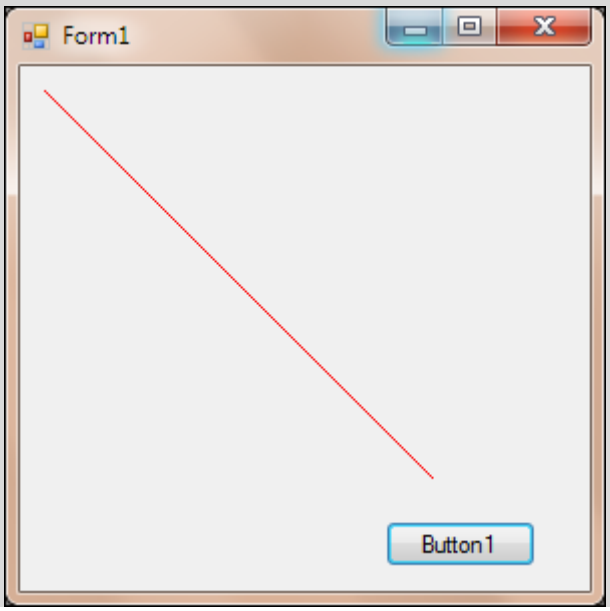
Le superfici grafiche create dal programmatore o da VB sono identiche ma possono produrre risultati diversi perché sono diversi i tempi del loro aggiornamento.
Supponiamo di avere un rettangolo disegnato in un controllo PictureBox.
Ogni volta che nel corso del programma questo PictureBox è visualizzato o modificato si verifica il suo evento Paint e si attiva la relativa procedura.
Ora, se le istruzioni per disegnare il rettangolo sono contenute all'interno di questa procedura, il rettangolo sarà disegnato ogni volta che si verifica l'evento Paint del PictureBox, e dunque sarà stabilmente visibile.
Se, al contrario, le istruzioni per disegnare il rettangolo nel PictureBox sono contenute in un'altra procedura, ad esempio una procedura Button1.Click, si corre il rischio che al verificarsi di un evento Paint del PictureBox il rettangolo non sia più visibile.
I tre esercizi che seguono mostrano alcuni esempi di creazione di superfici grafiche create dal programmatore o da VB in modo automatico.

Esercizio 78: Creazione di una superficie grafica con il comando CreateGraphics.

In questo esercizio vediamo la creazione di una superficie grafica con il comando CreateGraphics: la superficie creata corrisponde a un controllo PictureBox già collocato nel form dal programmatore.
Una volta creata la superficie grafica, al suo interno viene tracciata una linea trasversale di colore rosso.
Apriamo un nuovo progetto e collochiamo nel form un controllo PictureBox, più o meno grande come il form.
Poi collochiamo nel form un pulsante Button: con un *clic* su questo pulsante, l'utente del programma avvierà l'operazione grafica.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' crea una superficie grafica corrispondente al PictureBox1:
        Dim SuperficieGrafica As Graphics = PictureBox1.CreateGraphics

        ' traccia una linea su questa superficie grafica:
        SuperficieGrafica.DrawLine(Pens.Red, 0, 0, 200, 200)

    End Sub

End Class
```

L'immagine seguente mostra il programma in esecuzione:

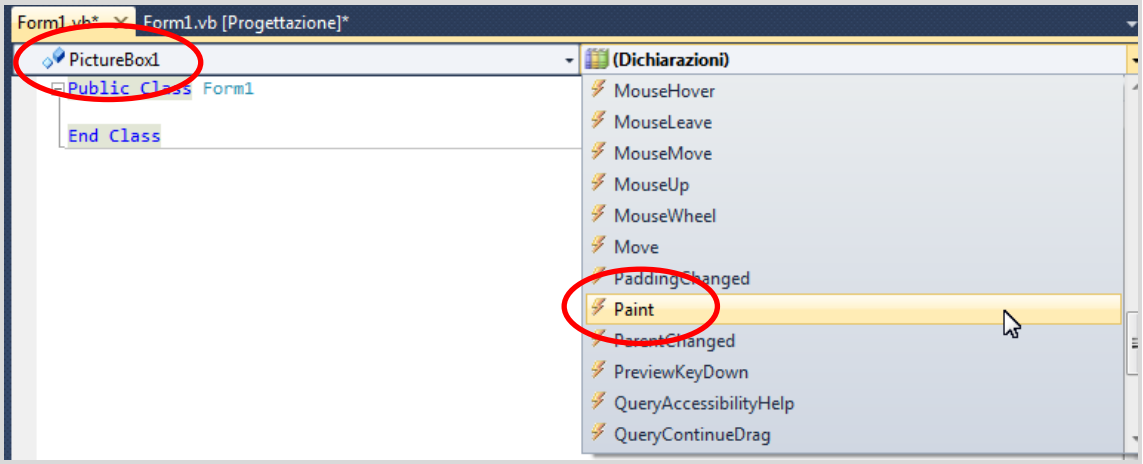


Nel prossimo esercizio vedremo lo stesso effetto grafico (disegno di una linea trasversale rossa all'interno di un controllo PictureBox), ottenuto in modo diverso:

- il disegno della linea non è collegato al *clic* sul Button1 ma è collegato all'evento Paint del controllo PictureBox;
- la superficie grafica non è creata dal programmatore, ma è individuata in modo automatico da VB, con la proprietà e.Graphic.

Esercizio 79: Creazione di una superficie grafica con l'evento Paint.

Apriamo un nuovo progetto e collochiamo nel form un controllo PictureBox, come nell'esercizio precedente.
In questo esercizio non serve il pulsante Button1 per avviare l'operazione grafica.
Apriamo la Finestra del Codice e facciamo un *clic* nella lista degli eventi del controllo PictureBox sull'evento Paint:



Vediamo che nella finestra del codice si imposta automaticamente la procedura relativa al verificarsi dell'evento Paint sul controllo PictureBox1:

```
Public Class Form1

    Private Sub PictureBox1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint

    End Sub

End Class
```

L'evento Paint si verifica automaticamente quando un controllo viene visualizzato o modificato; questa procedura, dunque, è destinata ad attivarsi quando il form è caricato in memoria e quando si visualizzano gli oggetti in esso contenuti, all'avvio del programma.
Notiamo tra le parentesi due elementi:

- la variabile **sender** indica l'oggetto che ha causato l'evento Paint;

- la variabile **e** indica l'oggetto destinato a ricevere le trasformazioni contenute nella procedura. Nel nostro esempio sarà il controllo PictureBox1.

L'oggetto **e** individuato automaticamente da VB ha due proprietà:
- la proprietà **e.Graphic** corrisponde alla superficie grafica del controllo PictureBox1;
- la struttura **e.ClientRectangle**, che vedremo in altri esempi più avanti, memorizza la posizione e le dimensioni della superficie e.Graphic.

Completiamo la procedura con le istruzioni per tracciare una linea trasversale all'interno del PictureBox:

```
Public Class Form1

    Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint

        e.Graphics.DrawLine(Pens.Red, 0, 0, 200, 200)

    End Sub

End Class
```

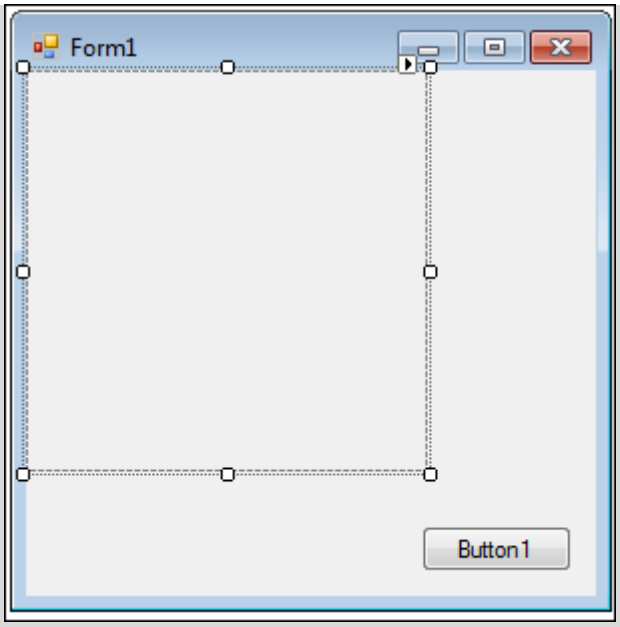
Mandando in esecuzione il programma, notiamo che la linea rossa all'interno del PictureBox viene tracciata all'avvio del programma.

Esercizio 80: Creazione di una superficie grafica con il comando CreateGraphics e con l'evento Paint.

Questo esercizio mostra la creazione di due superfici grafiche, corrispondenti entrambe a un unico controllo PictureBox:
- una superficie grafica è creata dal programmatore con il comando CreateGraphics,
- l'altra è individuata da VB con l'evento Paint.

Su queste due superfici grafiche sono poi disegnate due linee diagonali di colori diversi.

Apriamo un nuovo progetto e collochiamo nel form un pulsante **Button1** (in basso a destra) e un controllo **PictureBox1** (in alto a sinistra) con queste proprietà:
- Width = 200
- Height = 200



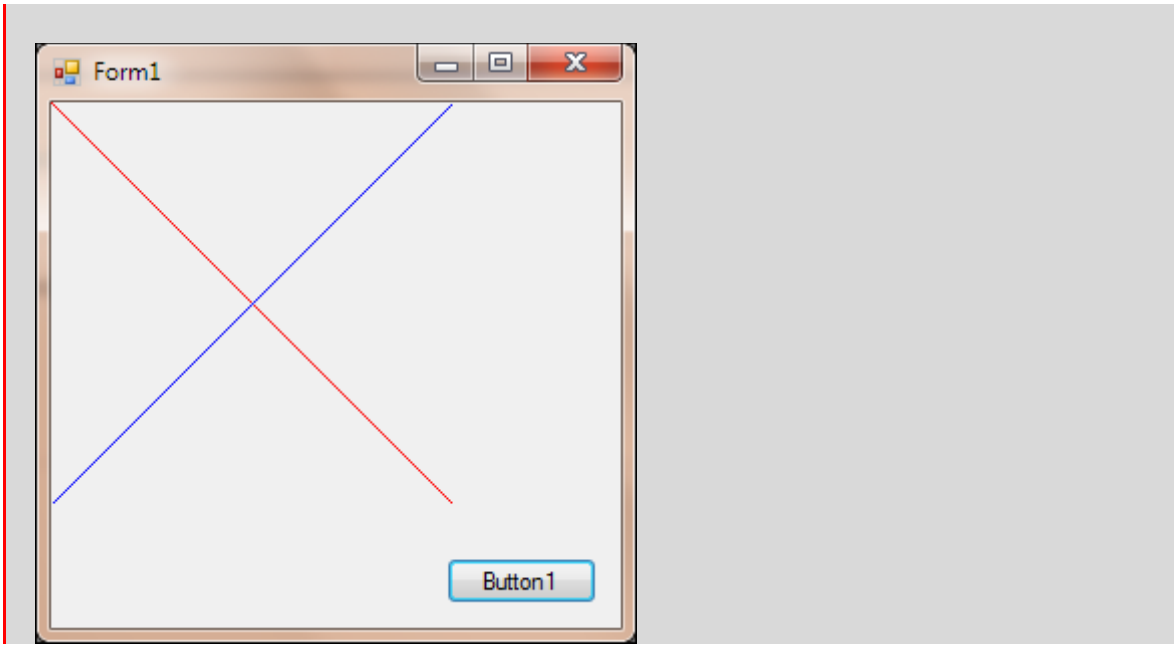
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint

        ' la superficie grafica è individuata da VB con la proprietà e.Graphics
        ' (questa superficie corrisponde al controllo PictureBox1)
        e.Graphics.DrawLine(Pens.Red, 0, 0, 200, 200)

    End Sub
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ' la superficie grafica è creata dal programmatore
        ' (la superficie corrisponde al controllo PictureBox1)
        Dim Superficie1 As Graphics = PictureBox1.CreateGraphics
        Superficie1.DrawLine(Pens.Blue, 0, 200, 200, 0)

    End Sub

End Class
```

- La prima procedura gestisce l'evento Paint del controllo PictureBox1: quando il controllo PictureBox1 viene visualizzato (all'avvio del programma), viene individuata da VB in modo automatico la superficie grafica corrispondente (`e.Graphics`); su questa superficie è tracciata una linea diagonale rossa.
- La seconda procedura gestisce l'evento Click sul pulsante Button1: quando si verifica questo evento, viene creata una superficie grafica di nome Superficie1, corrispondente al controllo PictureBox1: e su questa superficie grafica è tracciata una linea diagonale blu.

Ecco un'immagine del programma in esecuzione:



127: Comandi grafici.

Sulle superfici grafiche create con il comando **CreateGraphics** o creare da VB in seguito all'evento **Paint** è possibile disegnare linee e forme con gli strumenti Pen (penna) e Brush (pennello), oppure visualizzare e trasformare immagini e scritte.
I comandi grafici di uso più comune, che vedremo dettagliatamente nei prossimi capitoli, sono:

DrawArc	Traccia un arco o un ellisse.
DrawBezier	Traccia una curva Bézier.
DrawBeziers	Traccia una sequenza di curve Béziers.
DrawClosedCurve	Traccia una linea curva che unisce una serie di punti; VB chiude il tracciato unendo automaticamente il punto finale con il punto iniziale.
DrawCurve	Traccia una linea curva che unisce una serie di punti; la linea tracciata rimane aperta.
DrawEllipse	Traccia un'ellise o un cerchio. Per disegnare un cerchio è necessario disegnare un'ellisse con altezza pari alla larghezza.

DrawLine	Traccia una linea.
DrawLines	Traccia una sequenza di linee connesse l'una all'altra.
DrawPie	Disegna una parte di un diagramma circolare (grafico a torta).
DrawPolygon	Disegna un poligono; è simile al comando DrawLines, ma con DrawPolygon il punto finale è collegato al punto iniziale.
DrawRectangle	Disegna un rettangolo.
DrawRectangles	Disegna una serie di rettangoli.
FillClosedCurve	Colora l'area racchiusa da una linea curva.
FillEllipse	Colora l'area delimitata da un'ellise o da un cerchio.
FillPie	Colora una parte di un diagramma circolare (grafico a torta).
FillPolygon	Colora l'area interna di un poligono.
FillRectangle	Colora l'area interna di un rettangolo.
FillRectangles	Colora le aree interne di una serie di rettangoli.
DrawImage	Disegna un'immagine.
DrawString	Disegna una scritta.

Tabella 31: I comandi grafici di uso più comune.

In generale, per ogni comando è necessario indicare questi elementi:

- per disegnare una linea: il punto iniziale e quello finale;
- per disegnare una serie di linee: il punto iniziale, il punto finale e una serie di punti intermedi;
- per disegnare o per colorare un rettangolo o un'ellisse: la posizione dell'angolo superiore sinistro, la larghezza e l'altezza;
- per disegnare un arco: la posizione dell'angolo superiore sinistro, la larghezza, l'altezza e l'apertura in gradi;
- per disegnare una curva Bézier: il punto d'inizio, il punto finale e i due punti di controllo.

128: La struttura ClipRectangle.

Abbiamo visto che quando si verifica un evento Paint sul form o su un controllo, VB crea automaticamente una superficie grafica rettangolare corrispondente al form o al controllo.
Le informazioni relative alla posizione dei quattro angoli di questa superficie grafica vengono memorizzate da VB nella struttura **ClipRectangle**.
Nel prossimo esercizio vedremo l'uso di questa struttura per ricavare la posizione dei quattro angoli di una superficie grafica corrispondente a un controllo PictureBox.

Esercizio 81: La struttura ClipRectangle.

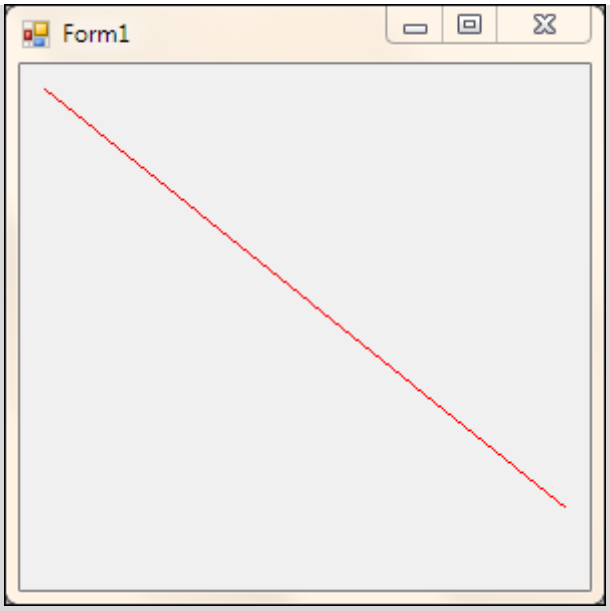
In questo esercizio i dati della struttura e.ClipRectangle sono utilizzati per tracciare una linea diagonale all'interno di una superficie grafica rettangolare.
Apriamo un nuovo progetto e inseriamo nel form un controllo PictureBox1.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint
        e.Graphics.DrawLine(Pens.Red,
                            e.ClipRectangle.Left,
                            e.ClipRectangle.Top,
                            e.ClipRectangle.Right,
                            e.ClipRectangle.Bottom)

    End Sub

End Class
```

Notiamo in questo codice i parametri del comando grafico DrawLine:

- dapprima viene indicato il colore dello strumento **Pens** (penna di disegno),
- poi sono indicati il punto dal quale partirà la linea (in alto a sinistra) e
- il punto in cui terminerà la linea (in basso a destra).

Le coordinate di questi due punti sono ottenute utilizzando i dati della struttura e.ClipRectangle.
Notiamo anche una particolarità nella scrittura del codice: il comando e.Graphics.DrawLine avrebbe potuto essere scritto su una sola linea. Qui, per comodità di lettura, il comando va a capo dopo ogni virgola; questa suddivisione non ne compromette la sintassi, in quanto VB è in grado di riconoscere un comando scritto su più righe**, se ogni riga va a capo con una virgola**.
Ecco un'immagine del programma in esecuzione:



All'avvio del programma è attivato automaticamente l'evento Paint del controllo PictureBox1; la procedura relativa a questo evento traccia una linea di colore rosso all'interno dell'oggetto grafico corrispondente al controllo PictureBox1, utilizzando i dati contenuti nella struttura e.ClipRectangle:

- la linea parte dal punto a sinistra in alto (e.ClipRectangle.Left, e.ClipRectangle.Top)
- e termina nel punto a destra, in basso (e.ClipRectangle.Right, e.ClipRectangle.Bottom).

La struttura **e.ClipRectangle** ricava la posizione dei quattro angoli della superficie grafica **visibile**, cioè quella parte di superfice grafica che non è coperta da altri oggetti.
Nel caso dell'esercizio precedente, se il controllo PictureBox1 è coperto a metà da un qualsiasi altro oggetto presente sullo schermo, l'area memorizzata dalla struttura e.ClientRectangle non corrisponde a tutta l'area del controllo PictureBox1, ma solo alla parte visibile di quest'area.
Se il programmatore ha l'esigenza di memorizzare i dati relativi a tutta l'area del PictureBox, compresa la parte eventualmente coperta e non visibile, deve ricorrere alla proprietà **ClientRectangle** del controllo interessato.
Ricordiamo che la proprietà **ClientRectangle** memorizza le dimensioni di un controllo, escludendone le parti di cui il programmatore non può disporre: barre di scorrimento, bordi, barra del titolo.

Il listato dell'esercizio precedente andrebbe dunque modificato in questo modo:

```
    Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint

        Dim Area1 As Rectangle = PictureBox1.ClientRectangle

        e.Graphics.DrawLine(Pens.Red,
                            Area1.Left,
```

```
                                Area1.Top,
                                Area1.Right,
                                Area1.Bottom)

    End Sub

End Class
```

129: Cambiare l'unità di misura delle lunghezze.

I comandi grafici assumono come unità di misura il pixel, che equivale a un punto sul monitor, cioè alla più piccola unità grafica disponibile sul monitor.
Questo comando, ad esempio, crea un'area rettangolare collocata a 10 pixel di distanza dal bordo sinistro, 10 pixel di distanza dal bordo superiore, larga 200 pixel e alta 150 pixel:

```
        Dim Ritaglio As New Rectangle(10, 10, 200, 150)
```

VB consente al programmatore di modificare tale unità di misura utilizzando il comando **PageUnit**. Eccone un esempio:

```
        e.Graphics.PageUnit = GraphicsUnit.Millimeter

        Dim Ritaglio As New Rectangle(10, 10, 200, 150)
```

Lo stesso comando che abbiamo visto sopra, cambiando unità di misura crea un'area rettangolare collocata a 10 millimetri di distanza dal bordo sinistro, 10 millimetri di distanza dal bordo superiore, larga 200 millimetri e alta 150 millimetri.

130: Ingrandire o ridurre una superficie grafica.

Il comando **ScaleTransform** ingrandisce o riduce una superficie grafica[70].
La sintassi del comando è questa:

```
e.Graphics.ScaleTransform(0.5, 0.5)
```

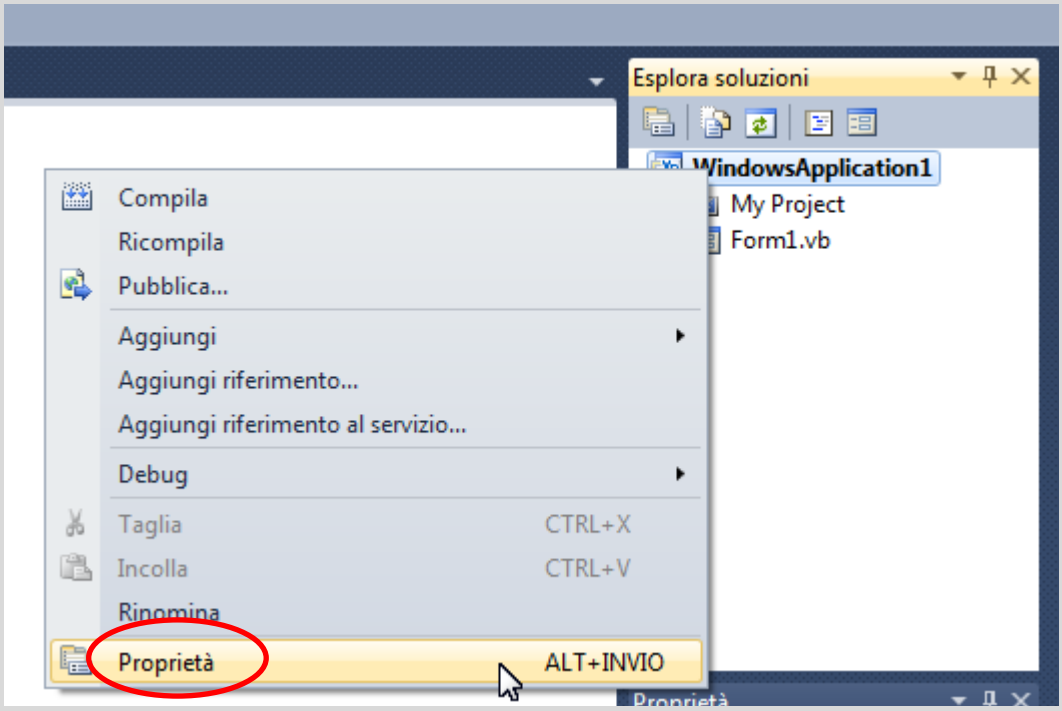
Tra parentesi sono indicati i fattori d'ingrandimento o di riduzione della larghezza e dell'altezza dell'oggetto grafico.

[70] In questo paragrafo si parla di ingrandimento o riduzione di una intera superficie grafica, che è cosa diversa dall'ingrandimento o riduzione di una sola immagine. Le operazioni di trasformazione di immagini sono illustrate più avanti, nel Capitolo 27: VISUALIZZARE E TRASFORMARE IMMAGINI. 612).

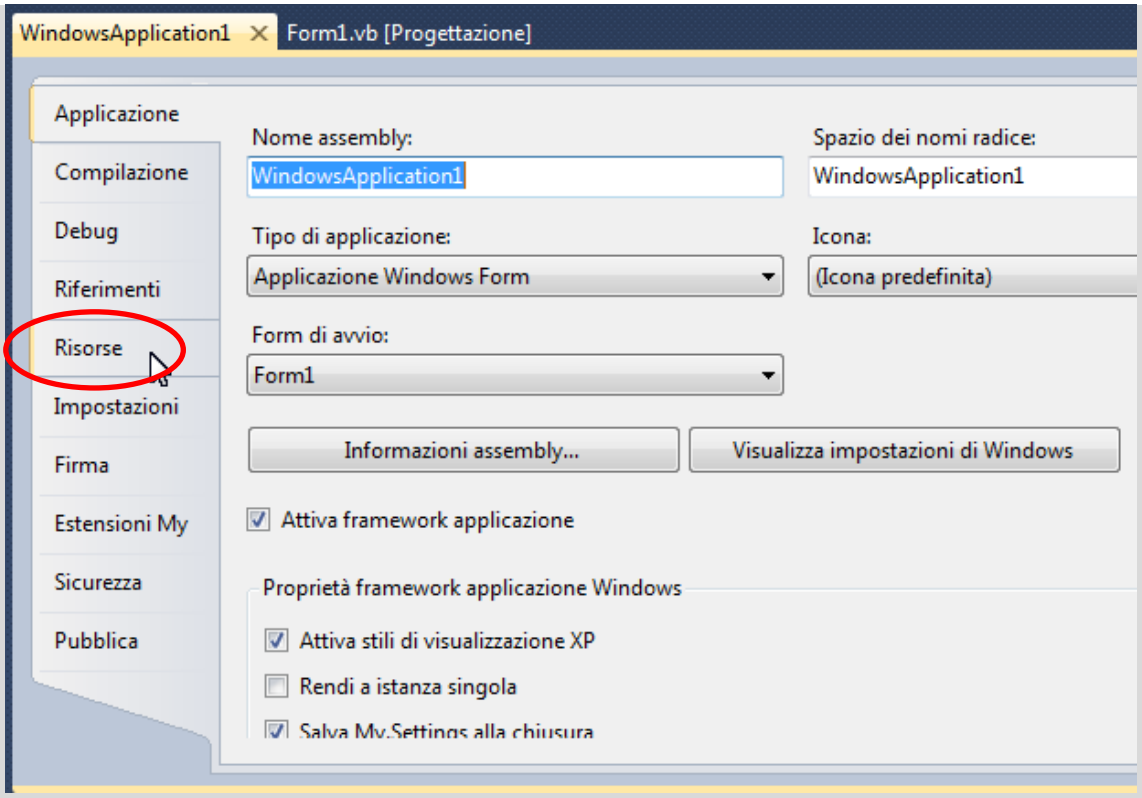
Esercizio 82: Il comando ScaleTransform.

In questo esercizio vediamo la creazione di due superfici grafiche, corrispondenti al form. In entrambe le superfici è visualizzata un'immagine della Terra, nella prima superficie l'immagine è visualizzata nelle sue dimensioni originali, nella seconda superficie è ridotta al 30% delle dimensioni originali.
Apriamo un nuovo progetto e inseriamo tre pulsanti Button (Button1, Button2 e Button3), nella parte inferiore del form.

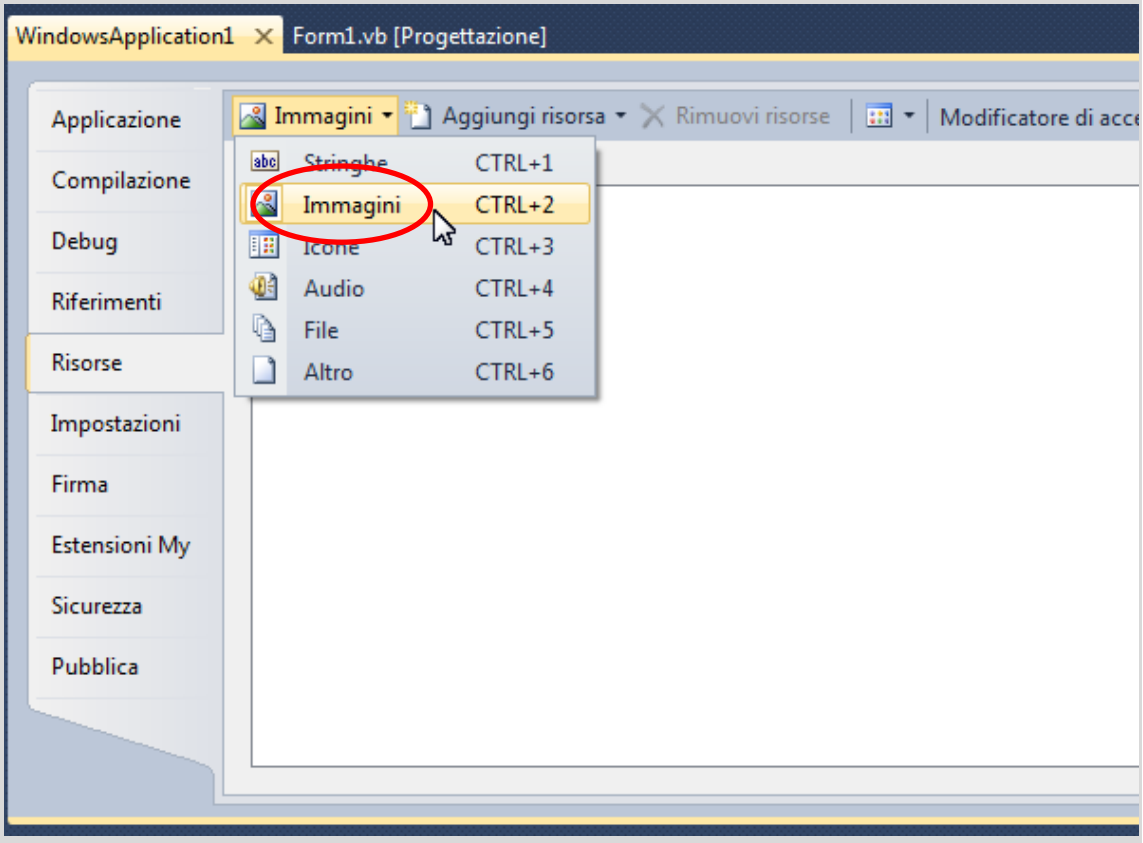
L'immagine utilizzata dal programma, Terra.bmp, si trova nella cartella **Documenti \ A scuola con VB 2010 \ Immagini**. La preleviamo da questa cartella per copiarla nelle risorse del programma.
Nella finestra **Esplora soluzioni**, facciamo un *clic* con il tasto destro del mouse sul nome dell'applicazione e poi sul menu Proprietà:



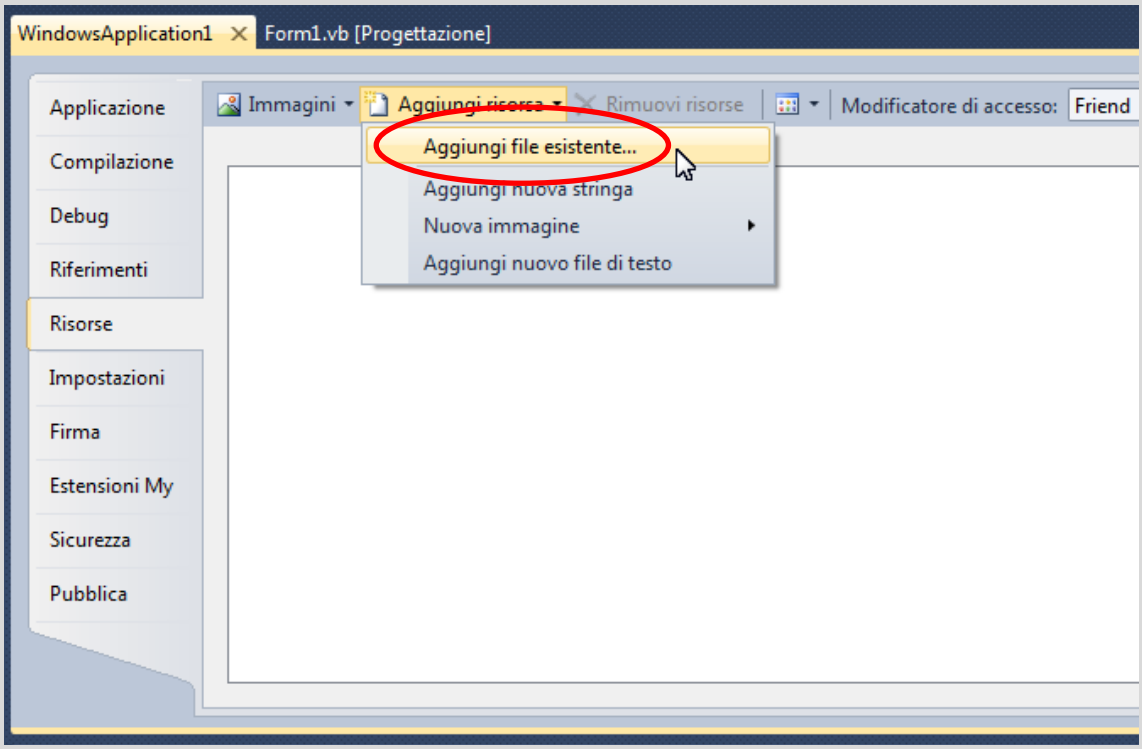
Nella scheda che si apre, facciamo *clic* su **Risorse**:



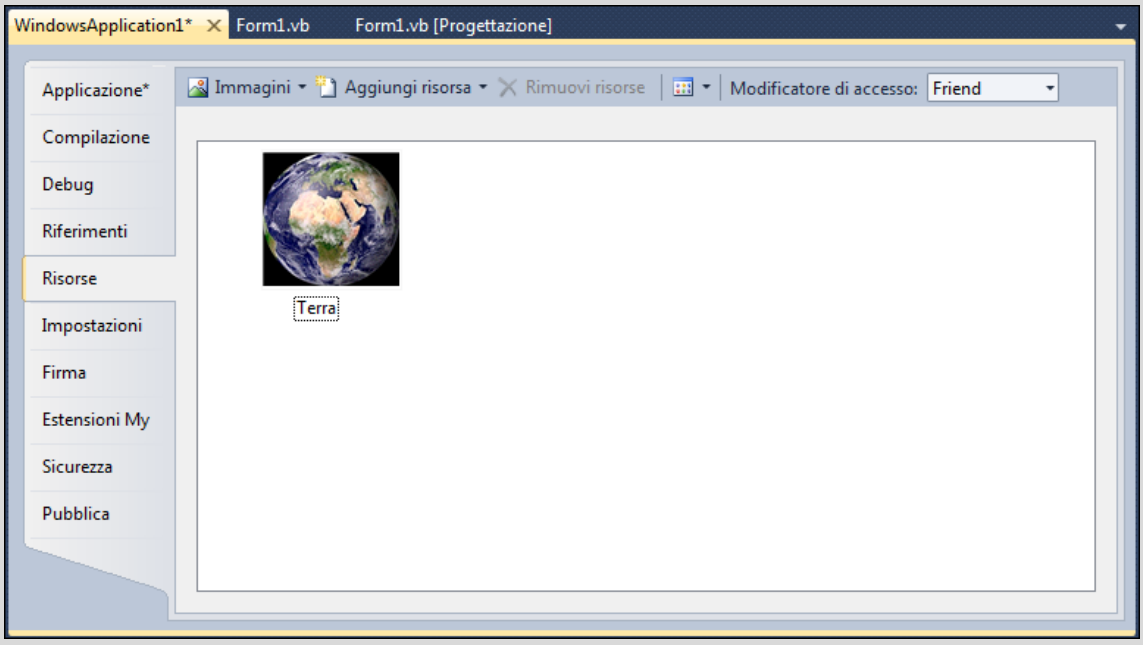
Quindi facciamo *clic* su **Immagini**:



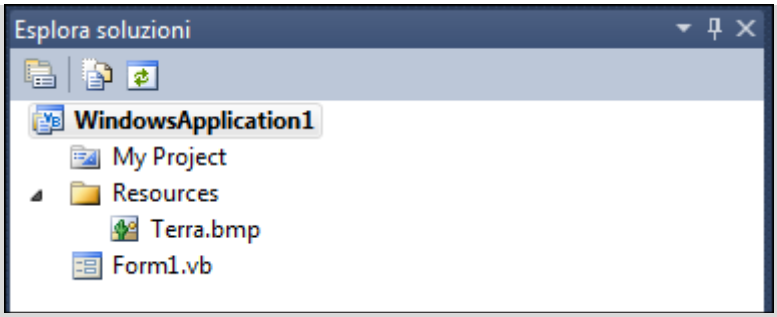
Aggiungi risorsa / Aggiungi file esistente:



Nella cartella **Documenti \ A scuola con VB \ Immagini** facciamo *clic* sull'immagine Terra.bmp:



Ecco come compare ora la finestra **Esplora soluzioni**:



Ora copiamo e incolliamo nella Finestra del Codice questo listasto:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' All'avvio del programma assegna allo sfondo del form il colore nero:
        Me.BackColor = Color.Black

    End Sub
```

```
    Private Sub Button1_Click() Handles Button1.Click

        ' Crea una superficie grafica corrispondente al form:
        Dim Superficie1 As Graphics = Me.CreateGraphics

        ' Crea il punto sinistra/alto di visualizzazione della immagine nella
superficie grafica
        Dim AngoloAltoSinistro As New Point(0, 0)

        ' Disegna l'immagine nella superficie grafica, nelle sue dimensioni
originali
        Superficie1.DrawImage(My.Resources.Terra, AngoloAltoSinistro)

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        ' Crea una superficie grafica corrispondente al form:
        Dim Superficie2 As Graphics = Me.CreateGraphics

        ' Crea il punto sinistra/alto di visualizzazione della immagine nella
superficie grafica
        Dim AngoloAltoSinistro As New Point(100, 100)

        ' Disegna l'immagine nella superficie grafica, ridotta al 30/100 delle
dimensioni originali
        Superficie2.ScaleTransform(0.3, 0.3)
        Superficie2.DrawImage(My.Resources.Terra, AngoloAltoSinistro)

    End Sub
```

```
    Private Sub Button3_Click(sender As Object, e As System.EventArgs) Handles
Button3.Click

        ' ripulisce il form:
```

```
        Me.Invalidate()

    End Sub

End Class
```

- La procedura Button1_Click crea una superficie grafica e vi disegna l'immagine nelle sue dimensioni originali.
- La procedura Button2_Click crea una superficie grafica e vi disegna l'immagine ridotta del 30%.
- La procedura Button3_Click ripulisce il form, cancellando le immagini disegnate dalle altre due procedure.

Ecco due immagini del programma in esecuzione:

131: Spostare una superficie grafica.

Il comando **TranslateTransform** sposta una superficie grafica in direzione orizzontale o verticale[71].
La sua sintassi è questa:

```
SuperficieGrafica.TranslateTransform(30, 30)
```

I due numeri tra parentesi indicano lo spostamento orizzontale e verticale della superficie grafica.
Ad esempio, nel programma dell'esercizio precedente possiamo modificare la procedura Button2_Click: invece della riduzione dell'immagine, possiamo inserirvi il comando per il suo spostamento:

```
    Private Sub Button2_Click() Handles Button2.Click

        ' Crea una superficie grafica corrispondente al form:
        Dim Superficie2 As Graphics = Me.CreateGraphics

        ' Crea il punto sinistra/alto di visualizzazione dell'immagine nella
superficie grafica
        Dim AngoloAltoSinistro As New Point(0, 0)

        ' Sposta la superficie grafica di 50 pixel a destra e in basso:
        Superficie2.TranslateTransform(50, 50)

        ' Copia l'immagine nella superficie grafica:
        Superficie2.DrawImage(My.Resources.Terra, AngoloAltoSinistro)

    End Sub
```

Ecco un'immagine del programma in esecuzione con questa variazione:

[71] In questo paragrafo si parla di spostamenti di un'intera superficie grafica, che è cosa diversa dallo spostamento di una sola immagine. Lo spostamento di un'immagine è un'operazione che può essere effettuata in modo più semplice, modificando la proprietà Location di un controllo PictureBox, come vedremo in un apposito esercizio.

Figura 159: Il comando TranslateTransform.

132: Ruotare una superficie grafica.

Il comando **RotateTransform** ruota una superficie grafica del numero di gradi indicato tra parentesi[72]:
La sua sintassi è questa:

```
SuperficieGrafica.RotateTransform(45)
```

Il numero scritto tra parentesi indica di quanti gradi deve essere ruotata la superficie.
Nell'esempio seguente continuiamo a modificare la procedura Button2_Click dell'ultimo esercizio: qui la superficie grafica viene sottoposta a tre operazioni in successione:
1. riduzione,
2. spostamento,
3. rotazione.

```
Private Sub Button2_Click() Handles Button2.Click

    ' Crea una superficie grafica corrispondente al form:
    Dim Superficie2 As Graphics = Me.CreateGraphics
```

[72] In questo paragrafo si parla di rotazione di una intera superficie grafica, che è cosa diversa dalla rotazione di una sola immagine. Le operazioni di rotazione di immagini sono illustrate più avanti nel manuale (Per ruotare un'immagine. a pag. 674).

```
        ' Crea il punto sinistra/alto di visualizzazione dell'immagine nella
superficie grafica
        Dim AngoloAltoSinistro As New Point(0, 0)

        ' Riduce la superficie grafica del 50%:
        Superficie2.ScaleTransform(0.5, 0.5)

        ' Sposta la superficie grafica di 500 pixel a destra e 400 in basso:
        Superficie2.TranslateTransform(500, 400)

        ' Ruota la superficie grafica di 180 gradi:
        Superficie2.RotateTransform(180)

        ' Copia l'immagine nella superficie grafica:
        Superficie2.DrawImage(My.Resources.Terra, AngoloAltoSinistro)

    End Sub
```

Ecco un'immagine del programma in esecuzione con questa variazione (l'immagine della Terra è ruotata di 180 gradi):

Figura 160: Il comando RotateTransform.

133: Migliorare la qualità grafica.

In certe condizioni, i contorni di forme geometriche o di immagini vengono visualizzati sul monitor con linee frastagliate: l'occhio umano arriva a vedere i pixel che le compongono e dunque, ad esempio, vede una linea obliqua come una scala di pixel quadrati.

E' un fenomeno che in termini tecnici si chiama **aliasing**: il termine sta a indicare che ciò che l'occhio vede è altro rispetto all'immagine originale.

Nei programmi di grafica per correggere il fenomeno di aliasing sono presenti strumenti di **antialiasing** che ammorbidiscono i contorni delle linee, facendo sfumare la visualizzazione dei singoli pixel.

Ne vediamo un esempio in questa immagine:

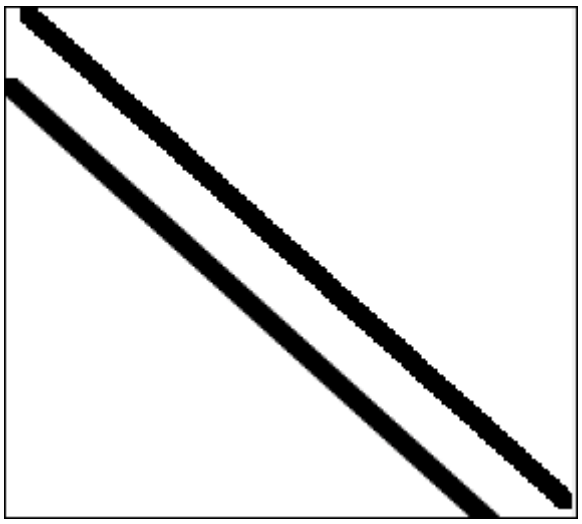


Figura 161: L'effetto alias (linea superiore) e la correzione antialias (linea inferiore).

VB mette a disposizione del programmatore due comandi diversi, per attivare la correzione antialiasing a disegni o a scritte:
Per i disegni:

```
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
```

per le scritte:

```
        e.Graphics.TextRenderingHint =
Drawing.Text.TextRenderingHint.AntiAlias
```

Nel prossimo esercizio vedremo un esempio di utilizzo di questi comandi.

Esercizio 83: I comandi di antialiasing.

Apriamo un nuovo progetto.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' modifica le dimensioni del Form1:
        Me.Width = 650
        Me.Height = 350

        ' crea uno strumento Penna che sarà utilizzato per disegnare due cerchi
        ' la Penna è di colore rosso,
        ' spessore 9 pixel:
        Dim Penna As New Pen(Brushes.Red, 9)

        ' disegna sulla superficie grafica del form un cerchio a sinistra
        ' senza la correzione AntiAlias:
        e.Graphics.DrawEllipse(Penna, 20, 10, 280, 280)

        ' disegna sulla superficie grafica del form un cerchio a destra
        ' con la correzione AntiAlias:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        e.Graphics.DrawEllipse(Penna, 320, 10, 280, 280)

        ' crea uno strumento Pennello che sarà utilizzato per scrivere
        ' la parola STOP nei due cerchi:
        Dim Pennello As Brush = Brushes.Black
        Dim Carattere As Font = New Font("Arial", 72)
        Dim Testo As String = "STOP"

        ' scrivi all'interno del primo cerchio la parola STOP
        ' senza la correzione AntiAlias:
        e.Graphics.DrawString(Testo, Carattere, Pennello, 10, 90)

        ' scrivi all'interno del secondo cerchio la parola STOP
        ' con la correzione AntiAlias:
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias
        e.Graphics.DrawString(Testo, Carattere, Pennello, 310, 90)

    End Sub

End Class
```

All'avvio del progetto, al verificarsi dell'evento Paint del Form1 vengono disegnati nel form due cerchi con la parola STOP al loro interno.
Il cerchio e la parola STOP di sinistra sono visualizzati senza la correzione AntiAliasing.
Ecco un'immagine del programma in esecuzione:


Form1
STOP
STOP

Capitolo 24: LA CLASSE PEN.

La classe **Pen** (= penna) crea strumenti per disegnare linee, curve, ellissi e perimetri di aree.
Le proprietà principali di questo strumento sono due:

- il colore e
- la larghezza del tratto.

Ecco un esempio di creazione di uno strumento Pen di colore blu, con uno spessore di 10 pixel:

```
Dim Penna As New Pen(Color.Blue, 10)
```
[73]

Una volta creato lo strumento Pen, è possibile modificarne le proprietà in questo modo:

```
Penna.Color = Color.Red
Penna.Width = 5
```

I segni che possono essere tracciati con la classe Pen sono elencati nella tabella seguente.
La lettrice o il lettore possono provare gli esempi, riportati nella colonna a destra, copiandoli e incollandoli nella finestra del codice di un progetto di VB in cui, nella Finestra del Codice, sia già impostata questa procedura Form1_Paint:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' la proprietà SmoothingMode.AntiAlias smussa i contorni del disegno:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim Penna As New Pen(Color.Red, 5)

        ' copiare e incollare al posto di questa riga il codice degli esempi

    End Sub

End Class
```

[73] E' possibile usare colori gradienti, cioè colori che scalano gradualmente da un colore all'altro, utilizzando la classe LinearGradientBush, illustrata a pag. 606.

Comando	Esempio da copiare
DrawArc Traccia un arco o un'ellisse.	e.Graphics.DrawArc(Penna, 10, 10, 200, 200, 180, 180)
DrawBezier Traccia una curva Bézier.	e.Graphics.DrawBezier(Penna, 10, 260, 240, 240, 200, 200, 10, 10)
DrawBeziers Traccia una sequenza di curve Béziers.	Dim Punti() As Point = {New Point(10, 10), New Point(250, 10), New Point(50, 250), New Point(250, 250), New Point(250, 10), New Point(250, 20), New Point(10, 250)} e.Graphics.DrawBeziers(Penna, Punti)
DrawClosedCurve Traccia una linea curva che unisce una serie di punti; la linea viene chiusa automaticamente, per cui il punto finale viene unito al punto iniziale.	Dim punti() As Point = {New Point(30, 30), New Point(230, 30), New Point(200, 230), New Point(100, 230)} e.Graphics.DrawClosedCurve(Penna, Punti)
DrawCurve Traccia una linea curva che unisce una serie di punti; il punto iniziale e il punto finale non coincidono.	Dim punti() As Point = {New Point(30, 30), New Point(230, 30), New Point(200, 230), New Point(100, 230)} e.Graphics.DrawCurve(Penna, Punti)
DrawEllipse Traccia un'ellisse o un cerchio. Vanno indicati tra parentesi i parametri di una superficie rettangolare nella quale si immagina che l'ellisse o il cerchio sono inscritti. Per disegnare un cerchio è necessario che questa superficie sia quadrata.	e.Graphics.DrawEllipse(Penna, 10, 10, 200, 200)
DrawLine Traccia una linea, dal primo punto al secondo punto.	e.Graphics.DrawLine(Penna, 10, 10, 200, 200)
DrawLines Traccia una sequenza di linee connesse l'una all'altra.	Dim Punti() As Point = {New Point(30, 30), New Point(230, 30), New Point(200, 230), New Point(100, 230)} e.Graphics.DrawLines(Penna, Punti)

DrawPie Disegna una parte di un diagramma circolare (grafico a torta).	e.Graphics.DrawPie(Penna, 10, 10, 250, 250, 45, 270)
DrawPolygon Disegna un poligono; è simile al comando DrawLines, ma con DrawPolygon il punto finale è collegato al punto iniziale.	Dim Punti() As Point = {New Point(30, 30), New Point(230, 30), New Point(200, 230), New Point(100, 230)} e.Graphics.DrawLines(Penna, Punti)
DrawRectangle Disegna un rettangolo.	e.Graphics.DrawRectangle(Penna, 10, 10, 150, 200)
DrawRectangles Disegna una serie di rettangoli.	Dim Rettangoli As Rectangle() = {New Rectangle(10, 10, 100, 150), New Rectangle(120, 10, 100, 50), New Rectangle(120, 70, 100, 50)} e.Graphics.DrawRectangles(Penna, Rettangoli)

Tabella 32: Azioni della Classe Pen.

I parametri richiesti da ogni comando, scritti tra parentesi, sono suggeriti da IntelliSense, man mano che si procede nella scrittura del codice.
Essi consistono in genere in questi elementi:

- il nome (arbitrario) che il programmatore vuole assegnare allo strumento Pen che sta creando;
- le coordinate x, y del punto iniziale della linea o dell'area;
- le coordinate di punti intermedi, se necessari;
- le coordinate x, y del punto finale o le misure Width, Height dell'area da disegnare.

Ad esempio, questo comando crea nella superficie grafica un cerchio inscritto in un quadrato (non visibile), il cui angolo superiore sinistro si trova a dieci pixel di distanza dal bordo sinistro e dal bordo superiore del form e ha le dimensioni di 100 pixel per lato:

```
e.Graphics.DrawEllipse(Penna, 10, 10, 100, 100)
```

Questo comando crea nella superficie grafica un rettangolo di 200 pixel di lunghezza e di 200 pixel di altezza, a 10 pixel di distanza dal bordo sinistro e dal bordo superiore del form:

```
e.Graphics.DrawRectangle(Penna, 10, 10, 200, 200)
```

Alcuni comandi richiedono il riferimento a un elenco di punti che devono essere dichiarati in una apposita matrice.

Ad esempio, questi comandi disegnano un poligono i cui vertici si collocano nei quattro punti elencati nella matrice di variabili denominata Punti:

```
        Dim Punti() As Point = {New Point(30, 30), New Point(230, 30), New
Point(200, 230), New Point(100, 230)}
        e.Graphics.DrawPolygon(Penna, Punti)
```

Il comando **DrawPie**, che disegna i grafici detti *a torta*, richiede due parametri aggiuntivi:

- la posizione (in gradi, da 0 a 360) da cui deve partire il disegno della fetta di torta (la posizione 0 corrisponde alla posizione delle ore 3, su un orologio analogico);
- la dimensione (in gradi, da 0 a 360) della fetta di torta da disegnare.

Ad esempio, il comando

```
        e.Graphics.FillPie(Brushes.Black, 10, 10, 250, 250, 0, 90)
```

disegna sulla superficie grafica una fetta di torta di 90 gradi (cioè ad angolo retto), partendo dalla posizione delle ore 3, arrivando alla posizione delle ore 6:

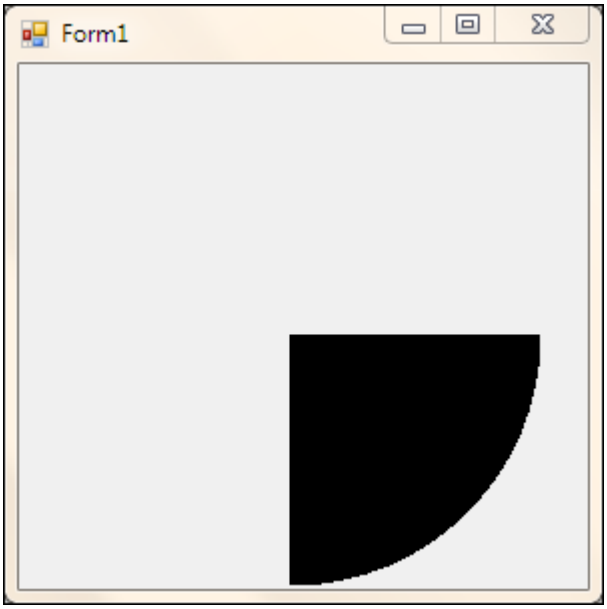


Figura 162: Un grafico a torta disegnato con il comando DrawPie.

Esercizio 84: Lo strumento Pen.

In questo esercizio proveremo alcuni comandi di tipo **Draw**, da eseguire con lo strumento **Pen**, per disegnare nel form una faccia sorridente.
Apriamo un nuovo progetto.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' crea un nuovo oggetto Penna, di colore nero, con lo spessore di 2
pixel:
        Dim Penna As New Pen(Color.Black, 2)

        ' tutti i comandi che seguono riguardano la superficie grafica
e.Graphics,
        ' sono scritti tra le righe  With e End With
        ' per evitare di scrivere e.graphics a ogni riga:
        With e.Graphics
            .SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
            .DrawEllipse(Penna, 10, 10, 200, 200)

            'disegna l'occhio e la pupilla a sinistra:
            .DrawEllipse(Penna, 50, 50, 30, 40)
            .DrawEllipse(Penna, 57, 70, 15, 20)

            'disegna l'occhio e la pupilla a destra:
            .DrawEllipse(Penna, 140, 50, 30, 40)
            .DrawEllipse(Penna, 147, 70, 15, 20)

            'disegna la bocca
            .DrawBezier(Penna, New Point(40, 130), New Point(65, 190), New
Point(155, 190), New Point(180, 130))
        End With

    End Sub

End Class
```

Nel codice, notiamo l'uso della tecnica di scrittura del codice

```
        With e.Graphics
            '.........
        End With
```

che consente di riferire un lungo elenco di proprietà o di azioni allo stesso oggetto senza ripetere ad ogni riga il nome dell'oggetto in questione.
Ecco un'immagine del programma in esecuzione:

La proprietà DashStyle

La proprietà **DashStyle** dello strumento **Pen** consente al programmatore di disegnare linee in modo continuo, tratteggiato o a puntini.
Se non si imposta la proprietà DashStyle, la linea disegnata ha il tratto continuo.
Il codice seguente crea in un form le cinque linee visualizzate nell'immagine seguente:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object,
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Penna As New Pen(Color.Black, 3)
        Penna.DashStyle = Drawing2D.DashStyle.Dash
        e.Graphics.DrawLine(Penna, 10, 10, 280, 10)

        Penna.Color = Color.Red
        Penna.DashStyle = Drawing2D.DashStyle.DashDot
        e.Graphics.DrawLine(Penna, 10, 30, 280, 30)

        Penna.Color = Color.Green
        Penna.DashStyle = Drawing2D.DashStyle.DashDotDot
        e.Graphics.DrawLine(Penna, 10, 50, 280, 50)

        Penna.Color = Color.Blue
        Penna.DashStyle = Drawing2D.DashStyle.Dot
        e.Graphics.DrawLine(Penna, 10, 70, 280, 70)

        Penna.Color = Color.Cyan
        Penna.DashStyle = Drawing2D.DashStyle.Solid
        e.Graphics.DrawLine(Penna, 10, 90, 280, 90)
```

```
    End Sub

End Class
```

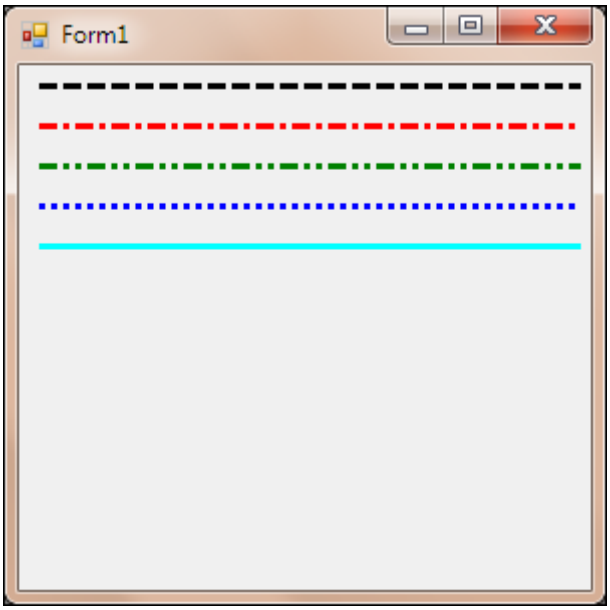


Figura 163: La proprietà DashStyle dell'oggetto Pen.

La proprietà StartCap

La proprietà EndCap

Le proprietà **StartCap** e **EndCap** dello strumento Pen consentono di disegnare forme particolari al punto iniziale e al punto finale di una linea.
Il codice seguente traccia in un form le cinque linee visualizzate nell'immagine seguente:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object,
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Penna As New Pen(Color.Black, 10)
        Penna.StartCap = Drawing2D.LineCap.DiamondAnchor
        Penna.EndCap = Drawing2D.LineCap.DiamondAnchor
        e.Graphics.DrawLine(Penna, 10, 10, 270, 10)

        Penna.Color = Color.Red
        Penna.StartCap = Drawing2D.LineCap.Round
        Penna.EndCap = Drawing2D.LineCap.Round
        e.Graphics.DrawLine(Penna, 10, 30, 270, 30)

        Penna.Color = Color.Green
```

```
        Penna.StartCap = Drawing2D.LineCap.Triangle
        Penna.EndCap = Drawing2D.LineCap.Triangle
        e.Graphics.DrawLine(Penna, 10, 50, 270, 50)

        Penna.Color = Color.Blue
        Penna.StartCap = Drawing2D.LineCap.ArrowAnchor
        Penna.EndCap = Drawing2D.LineCap.ArrowAnchor
        e.Graphics.DrawLine(Penna, 10, 70, 270, 70)

    End Sub

End Class
```

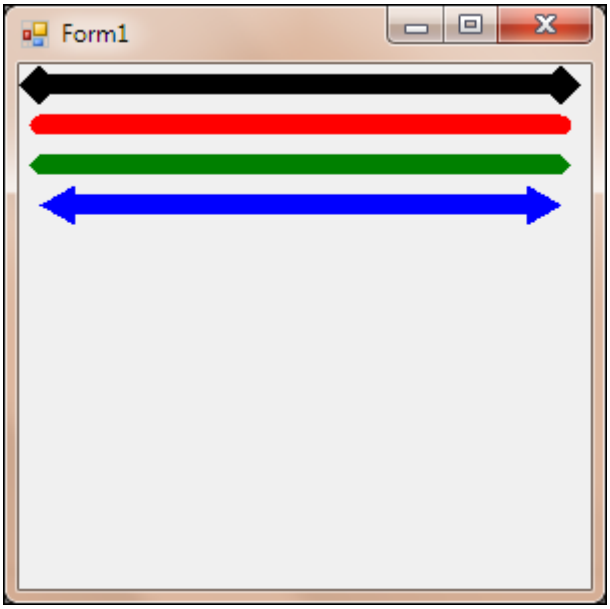


Figura 164: Le proprietà StartCap e EndCap dello strumento Pen.

La proprietà LineJoin

La proprietà **LineJoin** dello strumento **Pen** imposta le modalità di congiunzione tra i vari segmenti di in una linea spezzata.
Il codice seguente traccia in un form le tre linee visualizzate nell'immagine:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object,
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Penna As New Pen(Color.Red, 10)
        ' Crea una matrice di punti per disegnare una linea spezzata:
        Dim Punti As Point() = {New Point(10, 20), New Point(10, 100), New
Point(150, 50), New Point(230, 250), New Point(80, 120)}
        ' congiunzione "a piega"
```

```
        Penna.LineJoin = Drawing2D.LineJoin.Bevel
        e.Graphics.DrawLines(Penna, Punti)

        Dim Penna2 As New Pen(Color.Green, 10)
        Dim Punti2 As Point() = {New Point(30, 0), New Point(30, 80), New
Point(170, 30), New Point(250, 230), New Point(100, 100)}
        ' congiunzione "a punta"
        Penna2.LineJoin = Drawing2D.LineJoin.Miter
        e.Graphics.DrawLines(Penna2, Punti2)

        Dim Penna3 As New Pen(Color.Blue, 10)
        Dim Punti3 As Point() = {New Point(50, 0), New Point(50, 60), New
Point(190, 10), New Point(270, 210), New Point(120, 80)}
        ' congiunzione arrotondata
        Penna3.LineJoin = Drawing2D.LineJoin.Round
        e.Graphics.DrawLines(Penna3, Punti3)

    End Sub

End Class
```

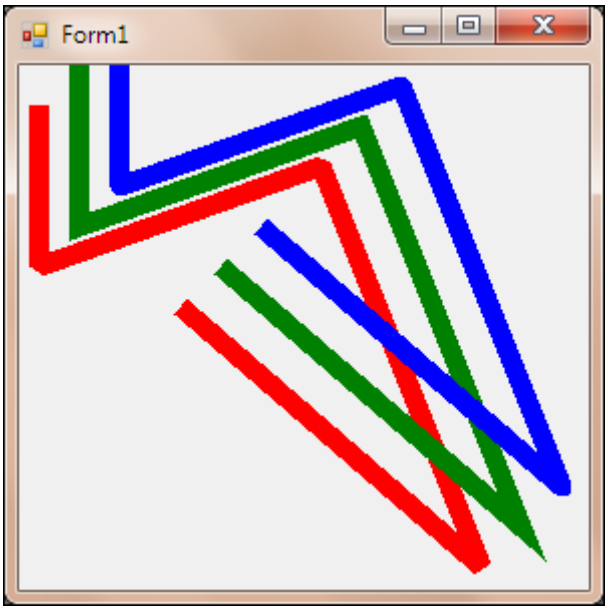


Figura 165: La proprietà LineJoin dello strumento Pen.

La proprietà CompoundArray

La proprietà **CompoundArray** (striscia composta) dello strumento **Pen** consente al programmatore di tracciare linee composte da linee interne parallele, di larghezza variabile, alternando parti colorate a parti non colorate.
L'immagine che segue mostra questo effetto nella creazione di tre linee di uguali dimensioni (50 pixel), rispettivamente di colore blu, verde e rosso:

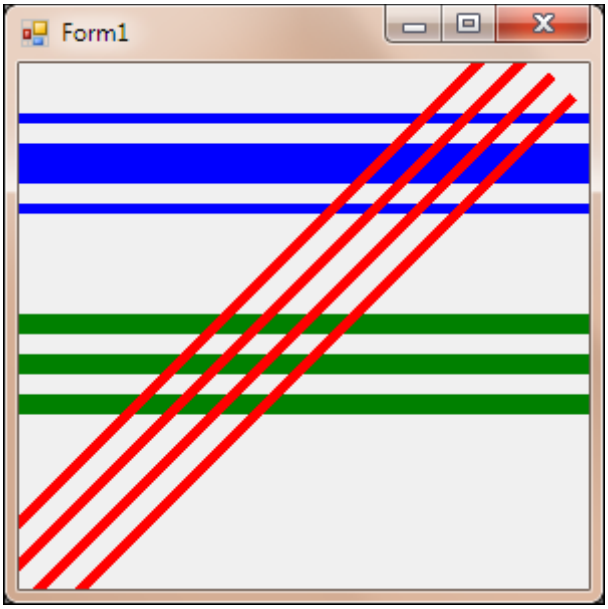


Figura 166: La proprietà CompoundArray dello strumento Pen.

A dispetto dell'apparenza, le linee create in questo programma sono tre: una blu e una verde orizzontali e una rossa obliqua; al loro interno hanno delle parti colorate e delle parti non colorate in misura variabile.
Per comprendere come sono impostate le parti colorate, si immagini la linea principale come una linea composta da dieci parti, su una scala che va da 0 a 1:
0 / 0.1 / 0.2 / 0.3 / 0.4 / 0.5 / 0.6 / 0.7 / 0.8 / 0.9 / 1
Questo comando crea uno strumento Penna1 e ne colora tre parti: da 0 a 0.1, poi da 0.3 a 0.7 e infine da 0.9 a 1 (è la linea blu che si vede in alto nella figura precedente):

```
        Penna1.CompoundArray = {0, 0.1, 0.3, 0.7, 0.9, 1}
```

Le tre linee della figura precedente, composte da parti colorate e parti non colorate, sono ottenute con il listato che segue.

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object,
ByVal e As System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Scomponi una penna di colore blu in dieci parti, su una scala che va da
0 a 1:
        ' colora la parte da 0 a 0.1
        ' colora le parti da 0.3 a 0.7
        ' colora la parte da 0.9 a 1
        Dim Penna1 As New Pen(Color.Blue, 50)
        Penna1.CompoundArray = {0, 0.1, 0.3, 0.7, 0.9, 1}
        e.Graphics.DrawLine(Penna1, 0, 50, 300, 50)

        ' Scomponi una penna di colore verde in dieci parti, su una scala che va
da 0 a 1:
        ' colora le parti da 0 a 0.2
```

```
        ' colora le parti da 0.4 a 0.6
        ' colora le parti da 0.8 a 1
        Dim Penna2 As New Pen(Color.Green, 50)
        Penna2.CompoundArray = {0, 0.2, 0.4, 0.6, 0.8, 1}
        e.Graphics.DrawLine(Penna2, 0, 150, 300, 150)

        ' Scomponi una penna di colore rosso in dieci parti, su una scala che va
da 0 a 1:
        ' colora la parte da 0 a 0.1
        ' colora le parti da 0.3 a 0.4
        ' colora la parte da 0.6 a 0.7
        ' colora la parte da 0.9 a 1
        Dim Penna3 As New Pen(Color.Red, 50)
        Penna3.CompoundArray = {0, 0.1, 0.3, 0.4, 0.6, 0.7, 0.9, 1}
        e.Graphics.DrawLine(Penna3, 0, 260, 260, 0)

    End Sub

End Class
```

134: La classe Pens.

La classe **Pens** (= penne) mette a disposizione del programmatore una linea di uso immediato, con diverse tonalità di colore, con lo spessore **fisso** di un pixel, non modificabile.
Uno strumento Pens si utilizza come uno strumento Pen normale, vale a dire indicando il punto iniziale e il punto finale della linea da disegnare.
Il codice seguente traccia in un form la linea visualizzata nell'immagine:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
        e.Graphics.DrawLine(Pens.Red, 10, 10, 200, 200)

    End Sub

End Class
```

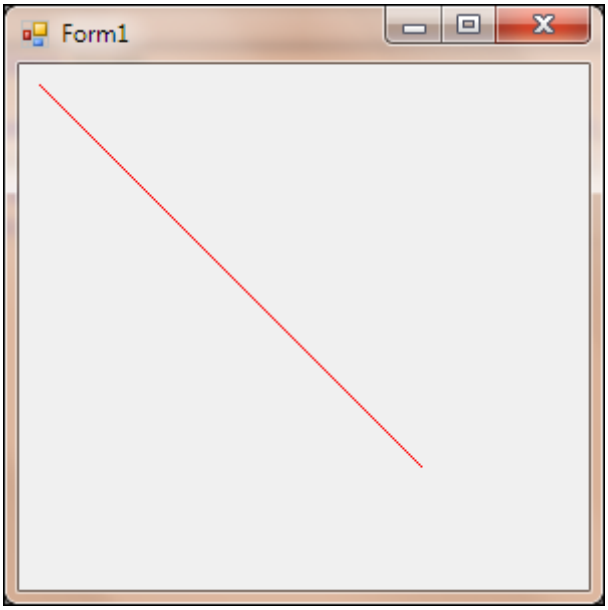


Figura 167: Una linea tracciata con la classe Pens.

Capitolo 25: LA CLASSE BRUSH.

La classe **Brush** (= pennello) crea strumenti per colorare aree chiuse.
La dichiarazione di uno strumento Brush richiede solo l'indicazione del colore del pennello. Ecco un esempio:

```
Dim Pennello As Brush = Brushes.Blue
```

Oppure, in alternativa, è possibile dichiarare un pennello solido (non trasparente) in questo modo:

```
Dim Pennello As New SolidBrush(Color.Black)
```

Uno strumento Pennello creato con questa seconda modalità può essere modificato in questo modo:

```
Pennello.Color = Color.Red
```

Le attività di coloritura che possono essere realizzate con la classe Brush sono elencate nella tabella seguente.
La lettrice o il lettore posono provare gli esempi, riportati nella colonna a destra, copiandoli e incollandoli nella finestra del codice di un progetto di VB in cui, nella Finestra del Codice, sia già stata impostata la procedura Form1_Paint:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        ' (la proprietà SmoothingMode.AntiAlias "smussa" i contorni del disegno)

        Dim Pennello As Brush = Brushes.Red

        ' copiare e incollare qui il codice degli esempi

    End Sub

End Class
```

Comando	Esempio da copiare
FillClosedCurve Colora l'area racchiusa da una linea curva.	Dim Punti() As Point = {New Point(20, 20), New Point(200, 20), New Point(150, 150), New Point(30, 150)} e.Graphics.FillClosedCurve(Pennello, Punti)
FillEllipse Colora l'area delimitata da un'ellisse o da un cerchio.	e.Graphics.FillEllipse(Pennello, 10, 10, 200, 200)
FillPie Colora una parte di un diagramma circolare (grafico a torta).	e.Graphics.FillPie(Pennello, 10, 10, 250, 250, 45, 270)
FillPolygon Colora l'area interna di un poligono.	Dim Punti() As Point = {New Point(30, 30), New Point(230, 30), New Point(200, 230), New Point(100, 230)} e.Graphics.FillPolygon(Pennello, Punti)
FillRectangle Colora l'area interna di un rettangolo.	e.Graphics.FillRectangle(Pennello, 10, 10, 150, 200)
FillRectangles Colora le aree interne di una serie di rettangoli.	Dim Rettangoli As Rectangle() = {New Rectangle(10, 10, 100, 150), New Rectangle(120, 10, 100, 50), New Rectangle(120, 70, 100, 50)} e.Graphics.FillRectangles(Pennello, Rettangoli)

Tabella 33: Azioni della Classe Brush.

I parametri richiesti da ogni comando, scritti tra parentesi, sono suggeriti da **IntelliSense**, man mano che si procede nella scrittura del codice.
Essi consistono in genere in questi elementi:

- il nome (arbitrario) che il programmatore vuole assegnare allo strumento Brush che sta creando;
- le coordinate x, y del punto iniziale dell'area da colorare;
- le coordinate di punti intermedi, se questi esistono;
- le dimensioni Width, Height (larghezza e altezza) dell'area da colorare.

Ad esempio, questo comando colora nella superficie grafica un cerchio di 200 pixel:

```
e.Graphics.FillEllipse(Pennello, 10, 10, 200, 200)
```

Alcuni comandi richiedono invece il riferimento a un elenco di punti che devono essere dichiarati in un'apposita matrice.
Ad esempio, questo comando colora nella superficie grafica un poligono i cui vertici si trovano nei punti elencati nella matrice di variabili denominata Punti:

```
        Dim Punti() As Point = {New Point(30, 30), New Point(230, 30), New
Point(200, 230), New Point(100, 230)}
        e.Graphics.FillPolygon(Pennello, Punti)
```

Il comando **FillPie**, che colora un grafico a torta, richiede due parametri aggiuntivi:

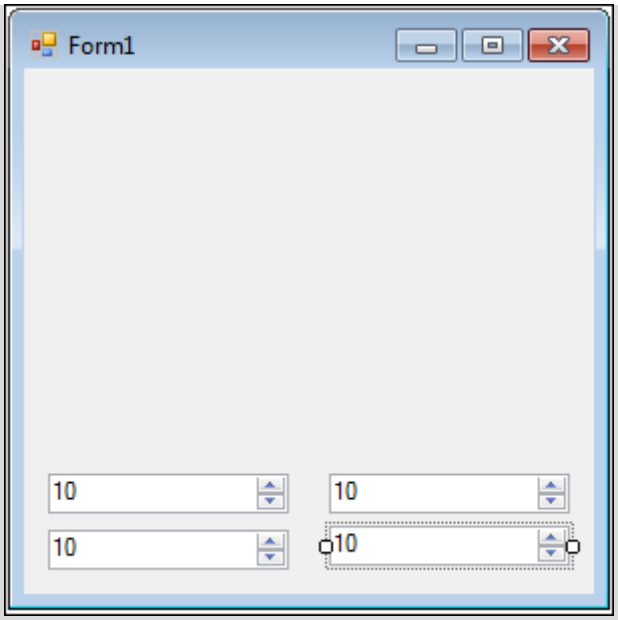
- la posizione (in gradi, da 0 a 360) da cui deve partire la coloritura della fetta di torta (la posizione 0 corrisponde su un orologio analogico alle ore 3);
- la dimensione (in gradi, da 0 a 360) della fetta di torta da colorare.

Ad esempio, questo comando colora una fetta di torta di 90 gradi (cioè ad angolo retto), partendo dalla posizione delle ore 3:

```
        e.Graphics.FillPie(Pennello, 10, 10, 250, 250, 0, 90)
```

Esercizio 85: Il comando FillPie.

In questo esercizio vedremo l'uso del comando **FillPie** per creare un grafico a torta.
Nel grafico sono visualizzate quattro quantità (dunque: quattro fette di torta), immesse dall'utente del programma con quattro controlli **NumericUpDown**.
Prima della rappresentazione grafica, il programma somma i valori presenti nei quattro controlli **NumeriUpDown** e li porta in scala a 360 (il cerchio ha 360 gradi, ogni spicchio rappresenta una parte del cerchio, il totale dei quattro spicchi è uguale a 360).
Apriamo un nuovo progetto e collochiamo nel form quattro controlli **NumericUpDown** come in questa immagine:



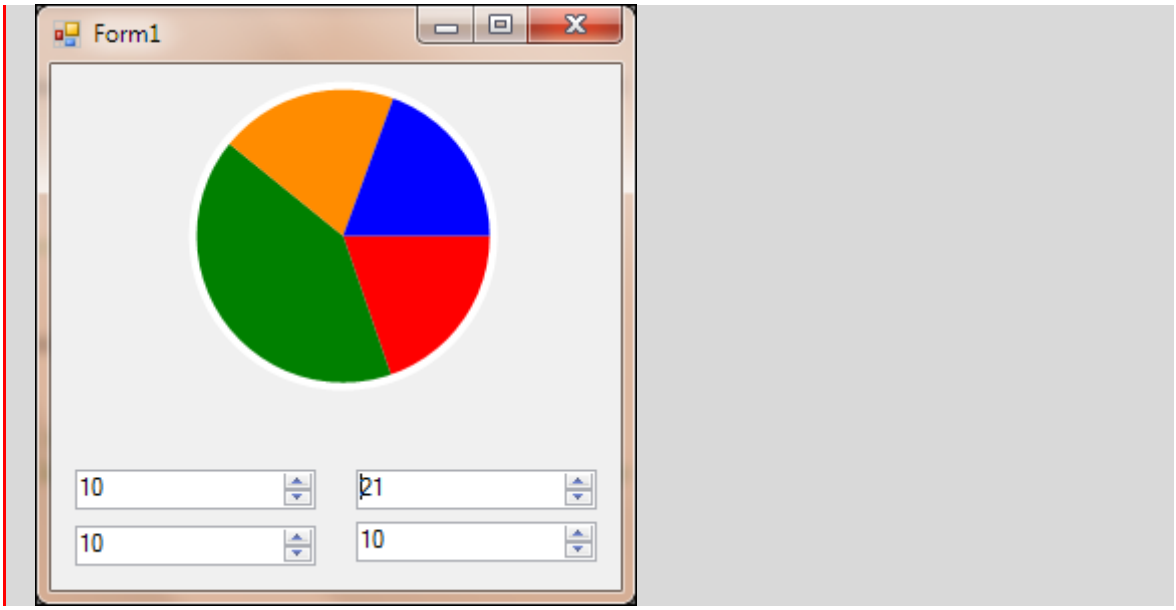
La proprietà **Value** dei quattro controlli **NumericUpDown** è impostata **= 10**.
La proprietà **DoubleBuffered** del **Form1** è impostata **= True**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Contatore As Integer
        ' migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' calcola la somma dei valori contenuti nei quattro controlli
NumericUpDown:
        Dim SommaValori As Integer
        For Each Controllo In Me.Controls
            If TypeOf Controllo Is NumericUpDown Then SommaValori = SommaValori +
Controllo.value
        Next

        ' Crea una matrice con i valori numerici contenuti nei 4 controlli
NumericUpDown.
        ' Ogni valore è riportato in scala a 360 (per riempire il cerchio la
somma dei 4 valori deve essere uguale a 360 gradi)
        Dim Valore(3) As Integer
        Valore(0) = NumericUpDown1.Value * 360 / SommaValori
        Valore(1) = NumericUpDown2.Value * 360 / SommaValori
        Valore(2) = NumericUpDown3.Value * 360 / SommaValori
        Valore(3) = 360 - Valore(0) - Valore(1) - Valore(2)

        ' Calcola l'angolo d’inizio di ogni fetta di torta.
        ' L'angolo d’inizio di una fetta è dato dall'angolo d’inizio della fetta
precedente + la grandezza della fetta precedente:
        Dim AngoloIniziale(3) As Integer
        For Contatore = 1 To 3
```

```
            AngoloIniziale(Contatore) = AngoloIniziale(Contatore - 1) +
Valore(Contatore - 1)
        Next

        ' Crea una matrice con quattro colori:
        Dim Colore() As Color = {Color.Red, Color.Green, Color.DarkOrange,
Color.Blue}

        ' Ciclo per disegnare le quattro fette di torta:
        ' a ogni passaggio si crea un nuovo pennello con uno dei 4 colori
inseriti nella matrice dei colori.
        For Contatore = 0 To 3
            Dim Pennello As New SolidBrush(Colore(Contatore))
            e.Graphics.FillPie(Pennello, 70, 10, 150, 150,
AngoloIniziale(Contatore), Valore(Contatore))
        Next

        ' Crea una penna per disegnare la circonferenza bianca attorno al grafico
a torta:
        Dim Penna As New Pen(Color.White, 4)
        e.Graphics.DrawEllipse(Penna, 70, 10, 150, 150)

    End Sub
```

```
    Private Sub NumericUpDown1_ValueChanged(sender As System.Object, e As
System.EventArgs) Handles NumericUpDown1.ValueChanged,
        NumericUpDown2.ValueChanged,
        NumericUpDown3.ValueChanged,
        NumericUpDown4.ValueChanged

        ' A ogni clic su uno dei quattro controlli NumericUpDown,
        ' ripulisci il form e attiva la procedura Form1_Paint:
        Me.Invalidate()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



135: La classe LinearGradientBrush.

La classe **LinearGradientBrush** consente di colorare un'area grafica (un rettangolo, un cerchio, un poligono) con sfumature di colore che passano gradualmente da un colore all'altro, tra due colori scelti dal programmatore (ad esempio: dal rosso al giallo).
Questo codice colora in un form il cerchio che si vede nell'immagine seguente, sfumando i colori dal rosso al giallo:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' la proprietà SmoothingMode.AntiAlias "smussa" i contorni del disegno:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim Pennello As New Drawing2D.LinearGradientBrush(New Rectangle(0, 0,
200, 200), Color.Red, Color.Yellow, 45)

        e.Graphics.FillEllipse(Pennello, 0, 0, 200, 200)

    End Sub

End Class
```

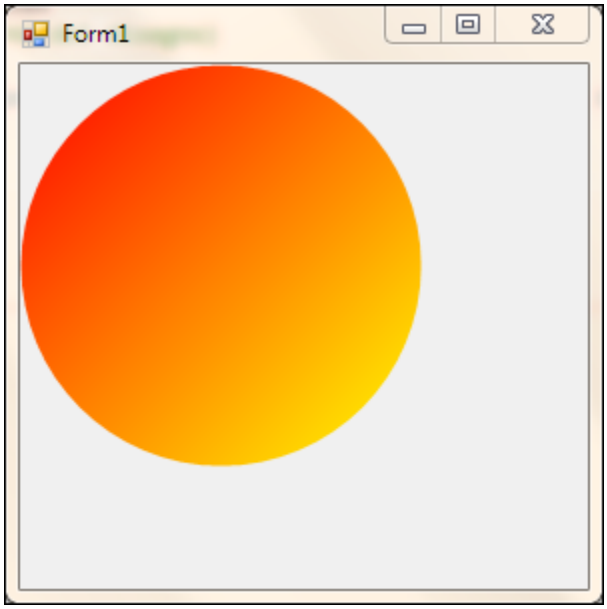


Figura 168: La classe LinearGradientBrush.

Notiamo nel codice che la classe **LinearGradientBrush** richiede l'indicazione di questi quattro parametri, tra parentesi:

- la creazione di un pennello di forma rettangolare, le cui dimensioni devono corrispondere a quelle dell'oggetto da colorare;
- il colore iniziale;
- il colore finale;
- la direzione della sfumatura dei colori, espressa in gradi.

Nell'esempio precedente, la direzione della sfumatura dei colori ha un'inclinazione di 45 gradi.
Ecco un esempio con inclinazione a 0 gradi:

```
        Dim Pennello As New Drawing2D.LinearGradientBrush(New Rectangle(0, 0,
200, 200), Color.Red, Color.Yellow, 0)
```

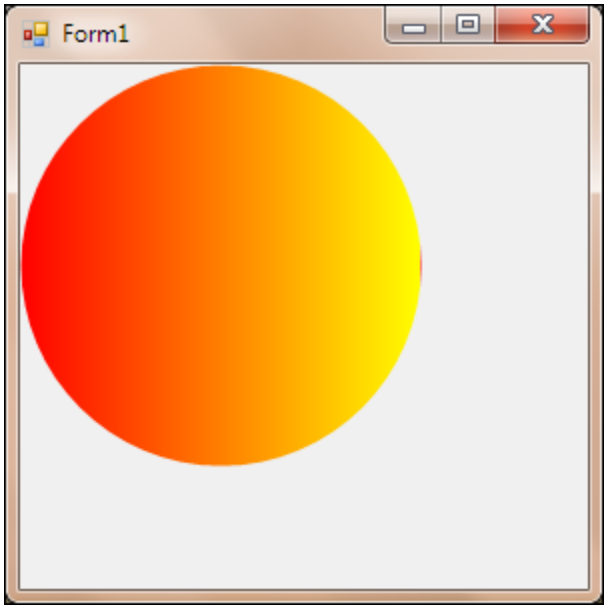


Figura 169: La classe LinearGradientBrush con direzione da sinistra a destra.

Altro esempio con inclinazione a 90 gradi:

```
        Dim Pennello As New Drawing2D.LinearGradientBrush(New Rectangle(0, 0,
200, 200), Color.Red, Color.Yellow, 90)
```

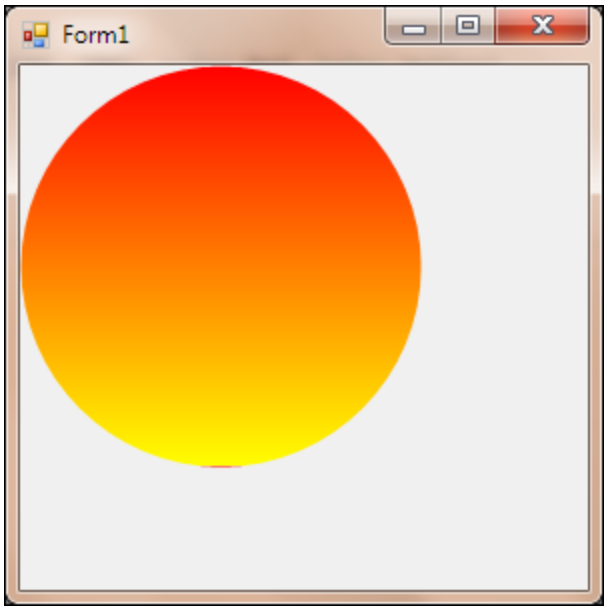


Figura 170: La classe LinearGradientBrush con direzione dall'alto al basso.

Se la grandezza del pennello corrisponde alle dimensioni dell'area da colorare (come negli esempi precedenti), si ha una figura riempita dal passaggio da un colore all'altro. Se, invece, la grandezza del pennello è inferiore a quella dell'area da colorare, il passaggio da un colore all'altro si ripete più volte all'interno dell'area colorata:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' la proprietà SmoothingMode.AntiAlias "smussa" i contorni del disegno:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim Pennello As New Drawing2D.LinearGradientBrush(New Rectangle(0, 0, 20,
20), Color.Red, Color.Yellow, 90)

        e.Graphics.FillEllipse(Pennello, 0, 0, 200, 200)

    End Sub

End Class
```

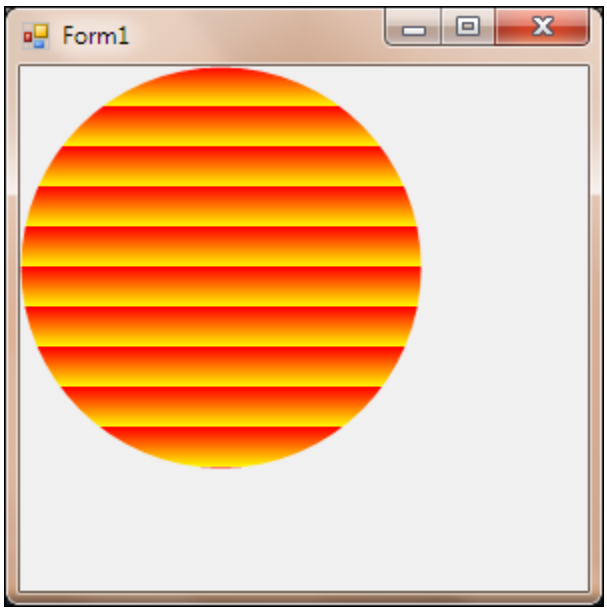


Figura 171: La classe LinearGradientBrush con ripetizione del passaggio da un colore all'altro.

La classe **LinearGradientBrush** può essere utilizzata anche per colorare **linee** con sfumature di colore che scalano da un colore all'altro.
Questo listato disegna una circonferenza, con colore gradiente dal blu al rosso, con inclinazione di 45 gradi:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Pennello As New Drawing2D.LinearGradientBrush(New Rectangle(10, 10,
200, 200), Color.Red, Color.Blue, 45)

        Dim Penna As New Pen(Pennello, 5)
        e.Graphics.DrawEllipse(Penna, 10, 10, 200, 200)
```

```
    End Sub

End Class
```

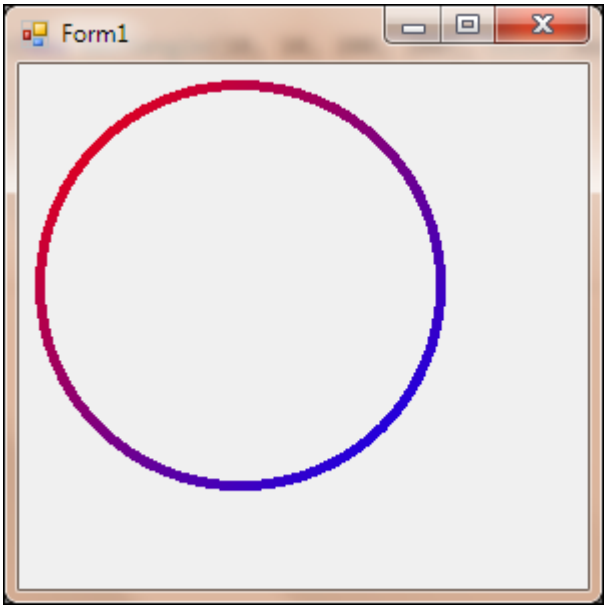


Figura 172: Una circonferenza colorata con la classe LinearGradientBrush.

Se si vuole colorare una linea con colori gradienti, invece di creare un pennello rettangolare è preferibile creare un pennello lineare, con l'indicazione del punto iniziale e del punto finale della linea da colorare.
Questo listato disegna una linea in diagonale, con colore gradiente dal blu al rosso:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Pennello As New Drawing2D.LinearGradientBrush(New Point(10, 10), New
Point(250, 250), Color.Red, Color.Blue)

        Dim Penna As New Pen(Pennello, 10)
        e.Graphics.DrawLine(Penna, 10, 10, 250, 250)

    End Sub

End Class
```

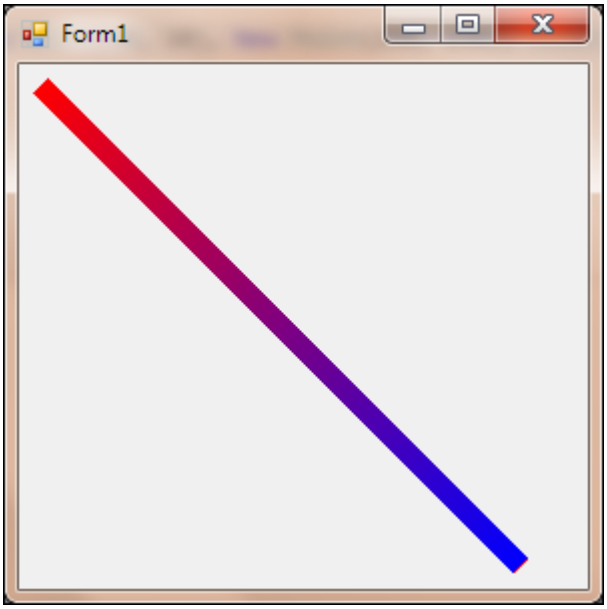


Figura 173: Una linea colorata con la classe LinearGradientBrush.

Da notare, nel codice di questi due ultimi esempi, che lo strumento **Penna**, con il quale vengono eseguiti i comandi DrawEllipse e DrawLine, al posto del parametro relativo al colore indica lo strumento **Pennello** e lo utilizza come se fosse un colore.

136: La classe SetBlendTriangularShape.

La classe **LinearGradientBrush** consente di scegliere, all'interno della figura da colorare, la posizione in cui si desidera che sia visualizzato il secondo colore: la proprietà **SetBlendTriangularShape**() determina questo punto, lungo una scala che va da 0 (inizio della figura) a 1 (fine della figura).
Esempi:

- **SetBlendTriangularShape(0):** il secondo colore compare all'inizio della direzione del colore;
- **SetBlendTriangularShape(0.5):** il secondo colore compare a metà della direzione del colore;
- **SetBlendTriangularShape(1):** il secondo colore compare alla fine della direzione del colore.

In questo listato, il secondo colore (giallo) è visualizzato nel primo quarto della superficie dell'oggetto da colorare:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object,
    ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
```

```
        ' (la proprietà SmoothingMode.AntiAlias "smussa" i contorni del disegno)

        Dim Pennello As New Drawing2D.LinearGradientBrush(New Point(10, 10), New
Point(250, 250), Color.Blue, Color.Yellow)
        Pennello.SetBlendTriangularShape(0.25)
        e.Graphics.FillEllipse(Pennello, 10, 10, 250, 250)
    End Sub

End Class
```

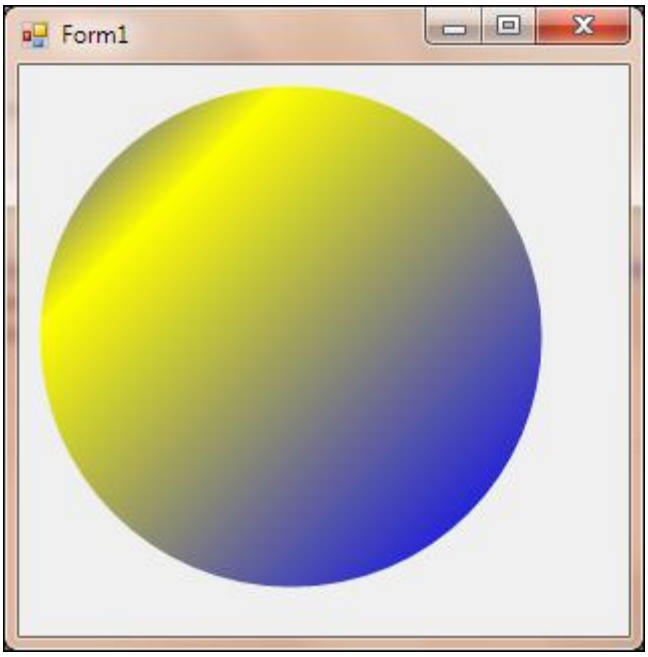


Figura 174: La proprietà SetBlendTriangularShape della classe LinearGradientBrush.

VB mette a disposizione del programmatore altre modalità per colorare un'area utilizzando il passaggio da un colore all'altro, o tra più colori, partendo anche dall'interno della figura, in direzione radiale.
Queste modalità sono possibili con oggetti/percorsi grafici creati con la classe **GraphicPath** che vedremo nel prossimo capitolo.

137: La classe HatchBrush.

La classe **HatchBrush** (= pennello con tratteggio) consente di riempire un'area con tratteggi di forme e colori diversi.
Gli stili di tratteggio disponibili sono circa 50; per ogni stile è possibile scegliere il colore del tratteggio e il colore dello sfondo.
Ecco due esempi:

```
Public Class Form1
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim Pennello As New
Drawing2D.HatchBrush(Drawing2D.HatchStyle.LargeConfetti, Color.Red, Color.Pink)

        e.Graphics.FillEllipse(Pennello, 10, 10, 200, 200)
    End Sub

End Class
```

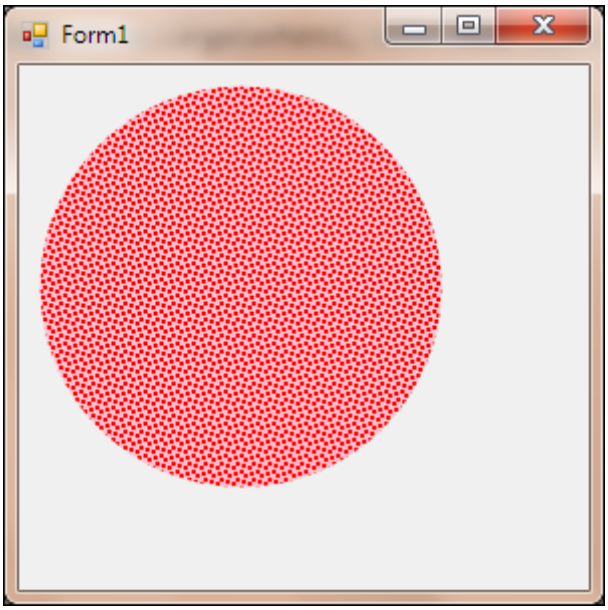


Figura 175: La classe HatchBrush con lo stile HatchStyle.LargeConfetti.

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        Dim Punti() As Point = {New Point(20, 20), New Point(200, 20), New
Point(150, 150), New Point(30, 150)}
        e.Graphics.FillClosedCurve(Brushes.Red, Punti)
        Dim Pennello As New Drawing2D.HatchBrush(Drawing2D.HatchStyle.Cross,
Color.Red, Color.Pink)
        e.Graphics.FillClosedCurve(Pennello, Punti)

    End Sub

End Class
```

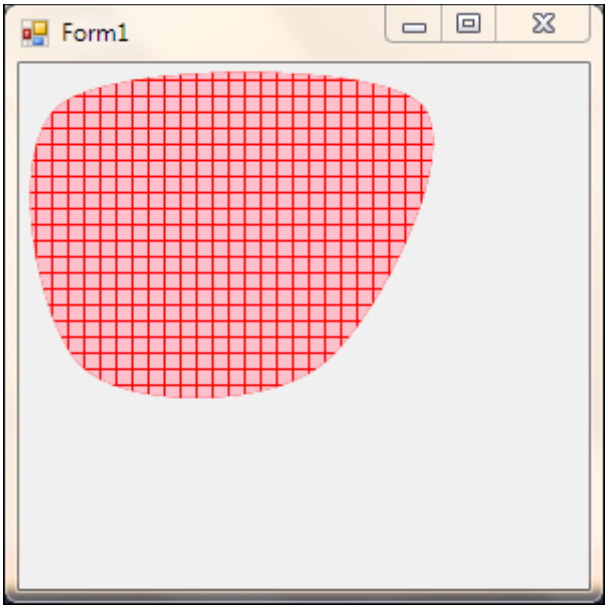


Figura 176: La classe HatchBrush con lo stile HatchStyle.Cross.

Notiamo che nella dichiarazione del pennello **HatchBrush** l'indicazione dello stile del tratteggio è seguita dall'indicazione di due colori: il colore del tratteggio in primo piano e il colore dello sfondo.

Esercizio 86: Stili disponibili per il pennello HatchBrush.

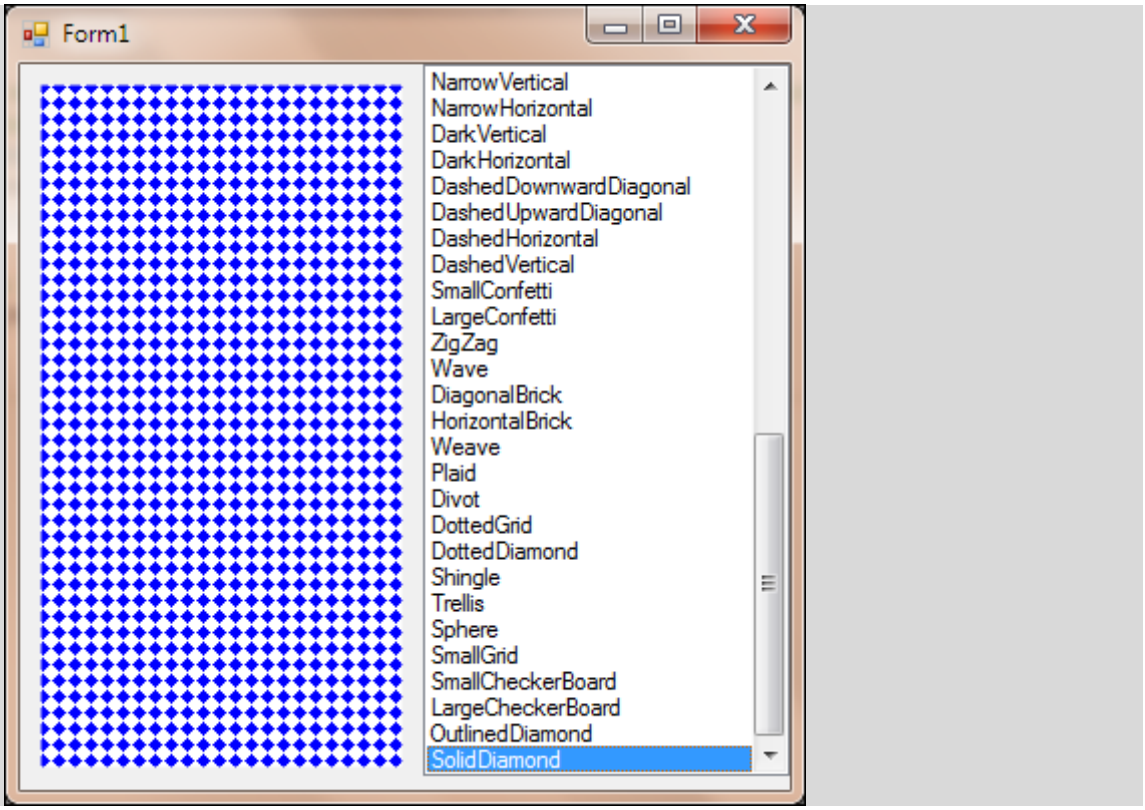
In questo esercizio creeremo un programma finalizzato a visualizzare tutti gli stili di tratteggio disponibili per la classe **HatchBrush**.
Apriamo un nuovo progetto e collochiamo nel Form1 un controllo **ListBox**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load

        'imposta le dimensioni del Form1 e del controllo ListBox1:
        Me.Width = 400
        Me.Height = 400
        ListBox1.Left = 200
        ListBox1.Width = 183
        ListBox1.Dock = DockStyle.Right

        'crea una matrice con tutti gli stili di tratteggio di HatchBrush
        Dim ElencoTratteggi As Array =
Drawing2D.HatchStyle.GetValues(GetType(Drawing2D.HatchStyle))
        'aggiunge i nomi degli stili di tratteggio al ComboBox1

        For Each Tratteggio As Drawing2D.HatchStyle In ElencoTratteggi
```

```
            ListBox1.Items.Add(Tratteggio)
        Next

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged1(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

        ' crea una superficie grafica corrispondente al Form1:
        Dim AreaDaTratteggiare As Graphics = Me.CreateGraphics
        'ripulisce lo sfondo del form ripristinando il colore di fondo originale
        AreaDaTratteggiare.Clear(Me.BackColor)

        AreaDaTratteggiare.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        ' (la proprietà SmoothingMode.AntiAlias "smussa" i contorni del disegno)

        'prende dal controllo ListBox il nome del tratteggio cliccato dall'utente
e lo converte in uno dei 50 stili di HatchStyle
        ' (il tratteggio sarà di colore blu, lo sfondo di colore bianco):
        Dim Tratteggio As Drawing2D.HatchStyle
        Tratteggio = CType(ListBox1.SelectedItem, Drawing2D.HatchStyle)
        Dim Pennello As New Drawing2D.HatchBrush(Tratteggio, Color.Blue,
Color.White)

        'riempie un'area rettangolare con il tratteggio selezionato:
        AreaDaTratteggiare.FillRectangle(Pennello, 10, 10, 180, 340)

    End Sub

End Class
```

I colori dei tratteggi sono fissati dal listato: colore blu per il tratteggio, colore bianco per lo sfondo.
Ecco un'immagine del programma in esecuzione:



138: La classe TextureBrush.

La classe **TextureBrush** (= pennello con trama) consente di riempire un'area con un'immagine ripetuta più volte.
L'esempio seguente utilizza l'immagine "ape", che si trova nella cartella **Documenti \ A scuola con VB \ Immagini \ Vocali**.

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' crea una superficie grafica corrispondente al Form1:
        Dim AreaDaRiempire As Graphics = Me.CreateGraphics()

        ' crea uno strumento Pennello
        ' con l'immagine "ape" dalla cartella Documenti / A scuola con VB 2010 /
Immagfini / Vocali:
        Dim PercorsoFile As String =
My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\A scuola con VB
2010\Immagini\Vocali\ape.jpg"
        Dim Pennello As New TextureBrush(Image.FromFile(PercorsoFile))

        Pennello.WrapMode = Drawing2D.WrapMode.Tile

        AreaDaRiempire.FillRectangle(Pennello, 10, 10, 250, 250)
```

```
    End Sub

End Class
```

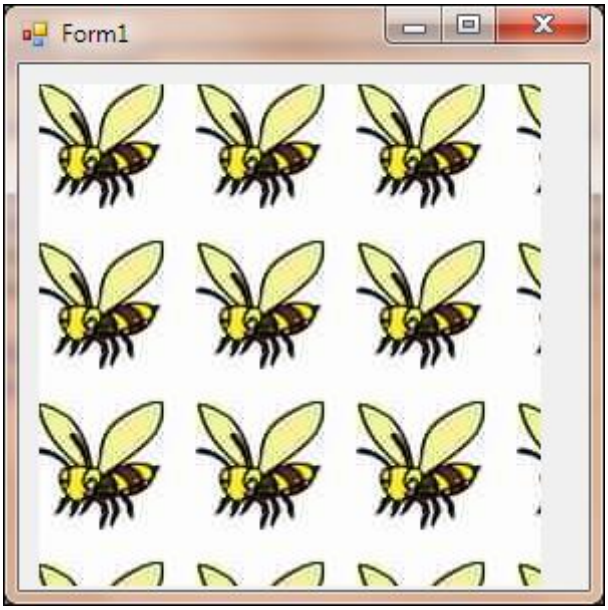


Figura 177: La classe TextureBrush con la proprietà WrapMode = Tile.

L'immagine può essere ripetuta con modalità diverse impostando la proprietà **WrapMode**:

- ripetizione dell'immagine *a mattonella*, senza modifiche:

```
Pennello.WrapMode = Drawing2D.WrapMode.Tile
```

- ripetizione dell'immagine con riflessione a specchio orizzontale:

```
Pennello.WrapMode = Drawing2D.WrapMode.TileFlipX
```

- ripetizione dell'immagine con riflessione a specchio verticale:

```
Pennello.WrapMode = Drawing2D.WrapMode.TileFlipY
```

- ripetizione dell'immagine con riflessione a specchio orizzontale e verticale:

```
Pennello.WrapMode = Drawing2D.WrapMode.TileFlipXY
```

Ecco un esempio di quest'ultima modalità:

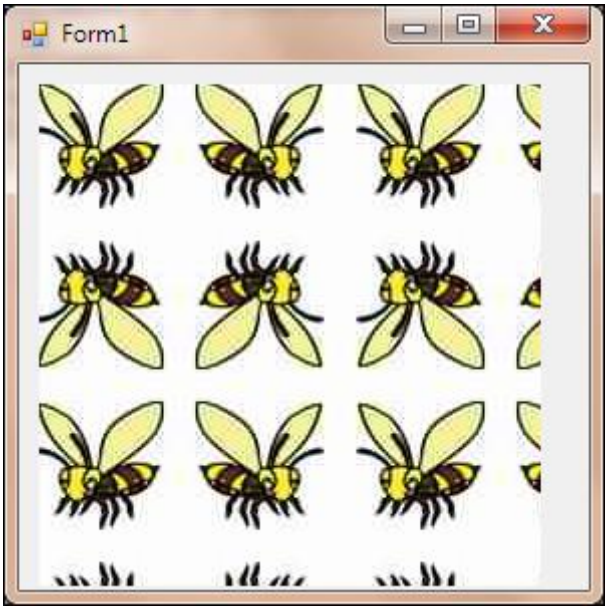


Figura 178: La classe TextureBrush con la proprietà WrapMode = TileFlipXY.

139: La classe Brushes.

La classe **Brushes** (= pennelli) mette a disposizione del programmatore un pennello di uso immediato, di cui è possibile scegliere il colore.
Ecco un esempio:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        e.Graphics.FillRectangle(Brushes.Red, 10, 10, 200, 200)

    End Sub

End Class
```

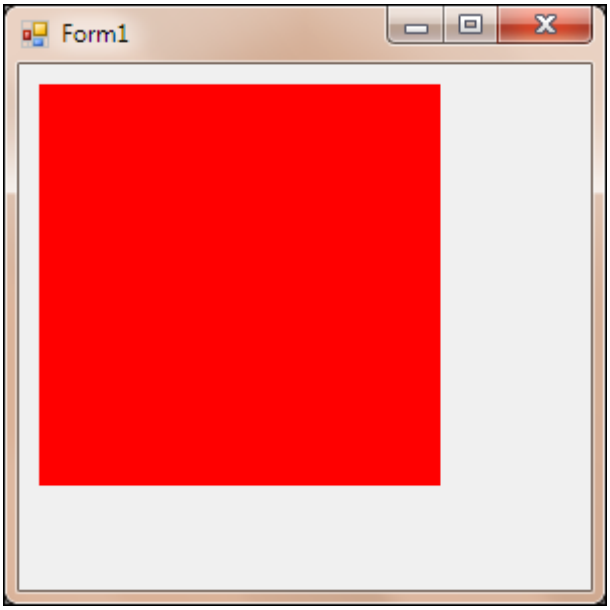


Figura 179: Un rettangolo colorato con la classe Brushes.

Esercizio 87: Lo strumento Brushes.

In questo esercizio proveremo alcuni comandi di tipo **Fill**, da eseguire con lo strumento Brushes, per disegnare nel form una faccia sorridente.
Apriamo un nuovo progetto e incolliamo nella Finestra del Codice questo listato:

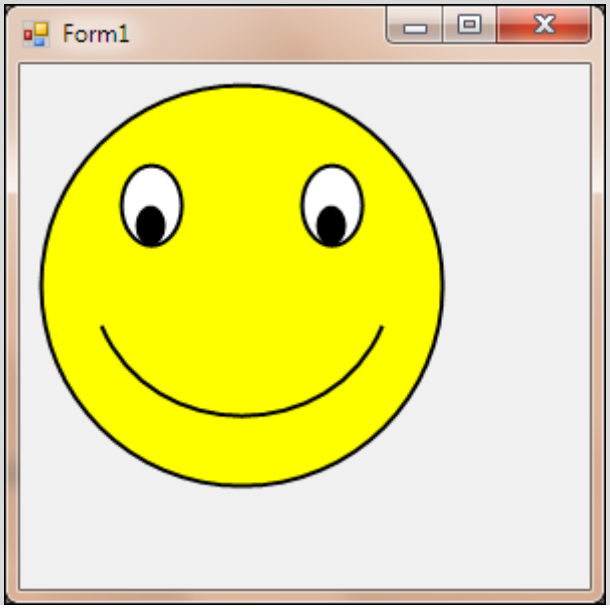
```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        Dim Penna As New Pen(Color.Black, 2)

        ' tutti i comandi che seguono riguardano l'oggetto e.Graphics, per cui
evitiamo di ripeterne il nome a ogni riga:

        With e.Graphics
            .SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
            .FillEllipse(Brushes.Yellow, 10, 10, 200, 200)
            .DrawEllipse(Penna, 10, 10, 200, 200)

            'disegna l'occhio e la pupilla a sinistra:
            .FillEllipse(Brushes.White, 50, 50, 30, 40)
            .DrawEllipse(Penna, 50, 50, 30, 40)
            .FillEllipse(Brushes.Black, 57, 70, 15, 20)

            'disegna l'occhio e la pupilla a destra:
            .FillEllipse(Brushes.White, 140, 50, 30, 40)
            .DrawEllipse(Penna, 140, 50, 30, 40)
            .FillEllipse(Brushes.Black, 147, 70, 15, 20)

            'disegna la bocca
```

```
            .DrawBezier(Penna, New Point(40, 130), New Point(65, 190), New
Point(155, 190), New Point(180, 130))

        End With

    End Sub

End Class
```

Notiamo l'uso della tecnica di scrittura del codice con i comandi With… End With:

```
        With e.Graphics

            ' ..................

        End With
```

che consentono di evitare la ripetizione del nome e.Graphics a ogni riga.
Ecco un'immagine del programma in esecuzione:



Capitolo 26: LE CLASSI GRAPHICSPATH E REGION.

Abbiamo visto nei due capitoli precedenti attività di disegno con gli strumenti **Pen** e **Brush** su superfici grafiche create con la classe **Graphic**.
In questo capitolo vedremo attività di disegno su oggetti creati con la classe **GraphicsPath** (= percorso di oggetti grafici).
Questa classe consente di unire in un percorso grafico diversi oggetti grafici (linee, rettangoli, ellissi, curve) e consente di trattarli come se fossero un unico oggetto, mettendo insieme anche elementi disparati, senza limiti di numero.
I percorsi grafici così creati sono generalmente chiusi e riempiti con un colore unito, o con colori gradienti, o con un'immagine.
Nel capitolo precedente abbiamo visto esempi con due colori gradienti, cioè due colori che scalano gradualmente dall'uno all'altro colore, creati utilizzando la classe LinearGradientBrush.
I percorsi creati con la classe GraphicsPath possono essere riempiti con colori gradienti **con più di due colori e in direzione radiale**, cioè con colori che sfumano dal centro verso il bordo esterno delle figure.

Per creare un nuovo oggetto **GraphicsPath** è sufficiente aggiungere a un percorso, uno alla volta, linee, rettangoli, curve, ellissi, stringhe di testo, con questi comandi:

- AddArc
- AddEllipse
- AddPolygon
- AddBezier
- AddLine
- AddRectangle
- AddCurve
- AddPie
- AddString

Questi comandi aggiungono a un oggetto **GraphicsPath** le stesse forme che è possibile disegnare direttamente su un oggetto **Graphic** con i comandi **Draw** e **Fill** che abbiamo visto nei capitoli precedenti.
La sintassi dei comandi **Add…** è identica alla sintassi dei comandi **Draw…** corrispondenti che abbiamo visto nel capitolo dedicato alla classe **Pen**, con l'omissione del primo parametro (il parametro relativo alla scelta dello strumento **Pen**), perché tutti gli oggetti che fanno parte dello stesso oggetto **GraphicsPath** vengono disegnati con un unico strumento **Pen**, uguale per tutti.

L'esempio seguente mostra la differenza tra le due classi.
Il programma disegna due ellissi, a sinistra, con il comando **DrawEllipse** e disegna, a destra, un percorso grafico che comprende due ellissi:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        Me.Width = 600
    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        Dim Penna1 As New Pen(Color.Red, 5)
        Dim Penna2 As New Pen(Color.Green, 5)
        e.Graphics.DrawEllipse(Penna1, 10, 10, 200, 100)
        e.Graphics.DrawEllipse(Penna2, 10, 120, 200, 100)

        Dim Oggetto As New Drawing2D.GraphicsPath
        Oggetto.AddEllipse(320, 10, 200, 100)
        Oggetto.AddEllipse(320, 120, 200, 100)
        e.Graphics.DrawPath(Penna1, Oggetto)

    End Sub

End Class
```

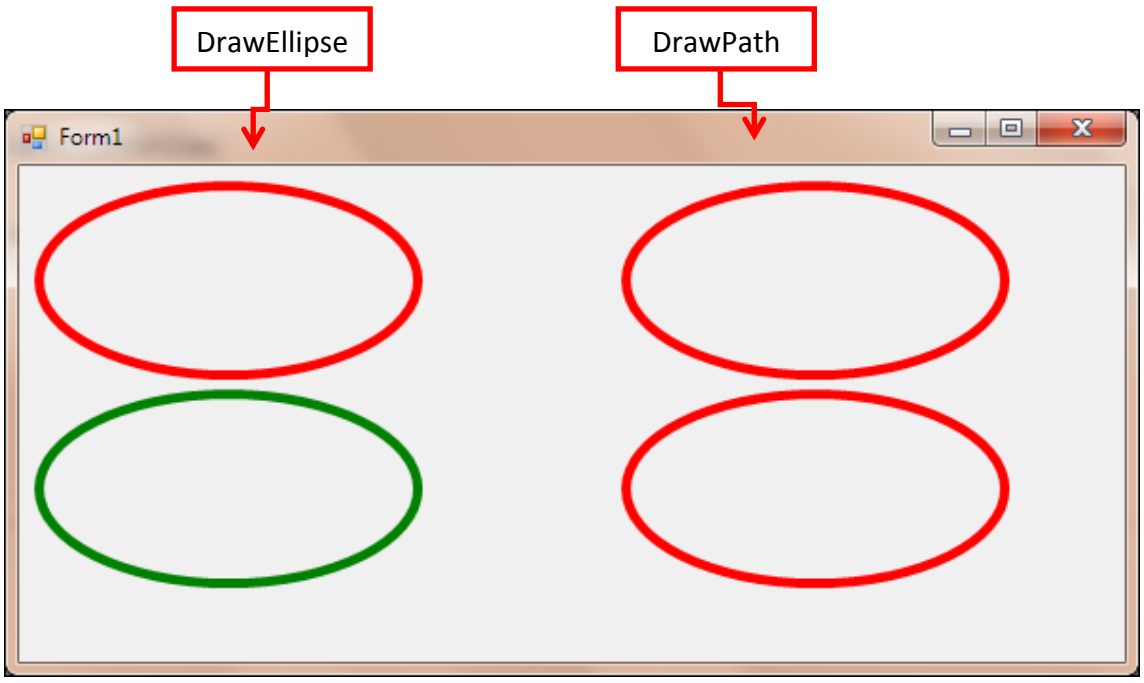


Figura 180: I comandi DrawEllipse e DrawPath.

Le differenze nella visualizzazione delle ellissi sono minime; in particolare, per disegnare le due ellissi a sinistra con il comando **DrawEllipse** possono essere usati

strumenti **Pen** di colori diversi, mentre con il comando **DrawPath** tutto l'oggetto creato con la classe **GraphicsPath** (le due ellissi a destra) è disegnato da un unico strumento Penna di un solo colore.
L'uso di un percorso grafico presenta, rispetto all'uso di un oggetto grafico, questi vantaggi:

- una volta che l'oggetto/percorso è stato definito, è facile replicarlo più volte;
- è possibile colorare un oggetto/percorso grafico con più di due colori gradienti, con scorrimento delle gradazioni dal centro dell'oggetto verso l'esterno.

L'esempio seguente disegna due ellissi: quella a sinistra è colorata con il comando **FillEllipse**, con colori gradienti lineri (da sinistra a destra), quella a destra è colorata con il comando **FillpPath**, con colori gradienti che vanno dal rosso al giallo, dal centro verso l'esterno.

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        'raddoppia la larghezza del form:
        Me.Width = 600

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità grafica del disegno:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' crea uno strumento pennello con colori gradienti dal rosso al giallo:
        Dim Pennello1 As New Drawing2D.LinearGradientBrush(New Point(10, 10), New
Point(200, 240), Color.Red, Color.Yellow)
        ' colora la prima ellisse:
        e.Graphics.FillEllipse(Pennello1, 10, 10, 200, 240)

        ' crea un oggetto/percorso grafico:
        Dim Oggetto As New Drawing2D.GraphicsPath
        ' aggiungi all'oggetto/percorso un'ellisse:
        Oggetto.AddEllipse(320, 10, 200, 240)

        ' crea uno strumento pennello per colorare l'oggetto/percorso con colori
gradienti (notare le proprietà Center Color e SurroundColors di questo pennello:
        Dim Pennello2 As New Drawing2D.PathGradientBrush(Oggetto)
        Pennello2.CenterColor = Color.Yellow
        Dim Gammacolori As Color() = {Color.Red}
        Pennello2.SurroundColors = Gammacolori
        e.Graphics.FillPath(Pennello2, Oggetto)

    End Sub

End Class
```

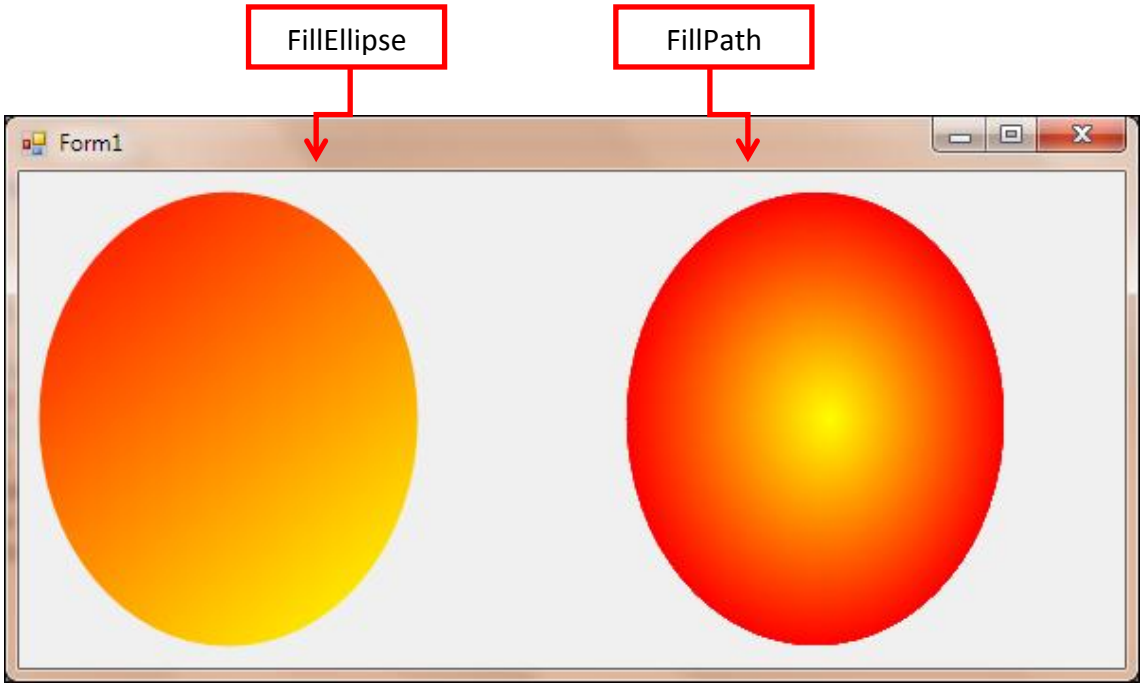


Figura 181: I comandi FillEllipse e FillPath.

Notiamo nel codice che la creazione di uno strumento **PathGradientBrush** per colorare un oggetto/percorso con colori gradienti richiede l'impostazione di due proprietà:

- la proprietà **CenterColor**, che assegna al pennello il colore che si troverà al centro della figura, e
- la proprietà **SurroundColors**, che assegna al pennello la gamma dei colori che dovranno trovarsi ai bordi cella figura.

Nel nostro esempio, al bordo della figura si trova solo il colore rosso, per cui la gamma dei colori della proprietà SurroundColors è composta solo dal colore rosso.

140: Colori gradienti in direzione radiale.

Vediamo come colorare un oggetto/percorso grafico, creato con la classe **GraphicPath**, con colori gradienti in direzione radiale, dal centro verso l'esterno della figura.
Il codice seguente crea un oggetto/percorso, gli assegna un'ellisse e lo colora con un pennello che va dal giallo (al centro) al rosso (all'esterno):

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        'Crea un nuovo oggetto di tipo 'percorso grafico' e vi aggiunge
un'ellisse:
        Dim AreaDaColorare As New Drawing2D.GraphicsPath()
```

```
        AreaDaColorare.AddEllipse(10, 10, 200, 240)

        ' Assegna a questo oggetto/percorso un pennello con colori gradienti dal
centro all'esterno:
        Dim Pennello As New Drawing2D.PathGradientBrush(AreaDaColorare)

        ' determina il colore al centro del pennello:
        Pennello.CenterColor = Color.Yellow

        ' determina i colori all'esterno del pennello (in questo esempio solo
uno: il colore rosso):
        Dim Gammacolori As Color() = {Color.Red}
        Pennello.SurroundColors = Gammacolori

        ' colora l'oggetto/percorso
        e.Graphics.FillPath(Pennello, AreaDaColorare)

    End Sub

End Class
```

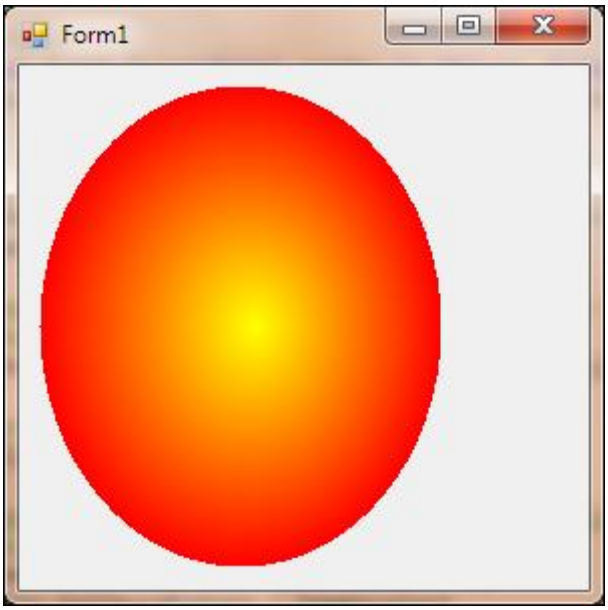


Figura 182: Il comando FillPath con un solo colore all'esterno.

Il codice seguente crea un oggetto/percorso, gli assegna un rettangolo e lo colora con un pennello con colori gradienti che vanno dal giallo (al centro) a quattro colori (all'esterno):

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Rettangolo As New Rectangle(10, 10, 200, 240)
        'Crea un nuovo oggetto di tipo 'percorso grafico' e vi aggiunge una
ellisse:
```

```
        Dim AreaDaColorare As New Drawing2D.GraphicsPath()
        AreaDaColorare.AddRectangle(Rettangolo)

        ' Assegna a questo oggetto/percorso un pennello con colori gradienti dal
centro all'esterno:
        Dim Pennello As New Drawing2D.PathGradientBrush(AreaDaColorare)

        ' determina il colore al centro del pennello:
        Pennello.CenterColor = Color.Yellow

        ' determina i quattro colori all'esterno del pennello:
        Dim Gammacolori As Color() = {Color.Red, Color.Green, Color.Blue,
Color.Black}
        Pennello.SurroundColors = Gammacolori

        ' colora l'oggetto/percorso
        e.Graphics.FillPath(Pennello, AreaDaColorare)

    End Sub

End Class
```

I quattro colori esterni indicati nel programma (rosso, verde, blu e nero), vengono visualizzati nell'ordine, a partire dall'angolo superiore sinistro:

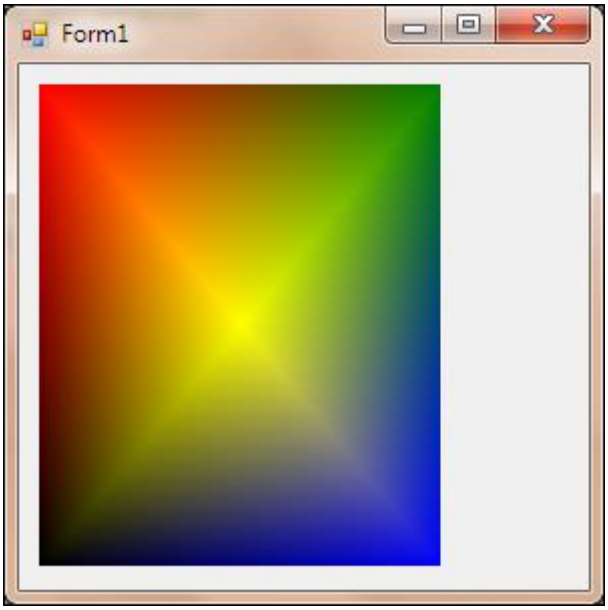


Figura 183: Il comando FillPath con quattro colori all'esterno.

La posizione del colore centrale (nel programma precedente il colore centrale è il giallo) può essere definita dal programmatore mediante la proprietà CenterPoint:

```
        Pennello.CenterPoint = New Point(60, 60)
```

Ecco un esempio:

```
Public Class Form1
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim Rettangolo As New Rectangle(10, 10, 200, 240)
        'Crea un nuovo oggetto di tipo 'percorso grafico' e vi aggiunge una
ellissi:
        Dim AreaDaColorare As New Drawing2D.GraphicsPath()
        AreaDaColorare.AddRectangle(Rettangolo)

        ' Assegna a questo oggetto/percorso un pennello con colori gradienti dal
centro all'esterno:
        Dim Pennello As New Drawing2D.PathGradientBrush(AreaDaColorare)

        ' determina il colore al centro del pennello:
        Pennello.CenterColor = Color.Yellow
        ' sposta il colore centrale al punto 60, 60 della figura:
        Pennello.CenterPoint = New Point(60, 60)

        ' determina i colori all'esterno del pennello:
        Dim Gammacolori As Color() = {Color.Red, Color.Green, Color.Blue,
Color.Black}
        Pennello.SurroundColors = Gammacolori

        ' colora l'oggetto/percorso
        e.Graphics.FillPath(Pennello, AreaDaColorare)

    End Sub

End Class
```

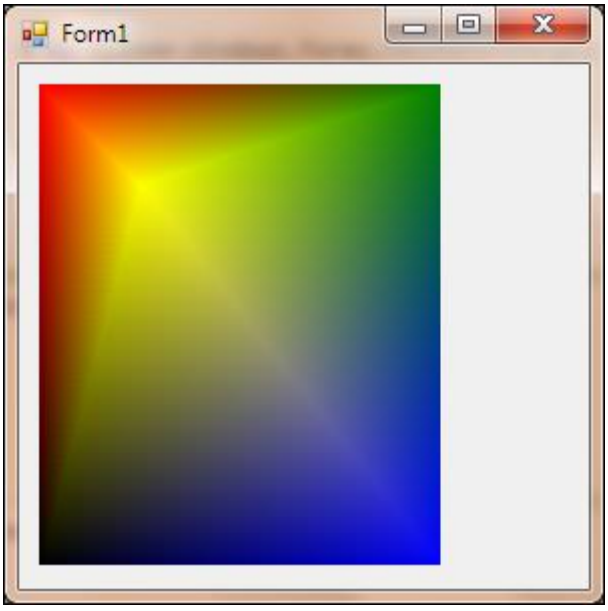


Figura 184: Il comando FillPath con il colore centrale in posizione decentrata.

Il codice seguente crea un oggetto/percorso a forma di cuore, utilizzando due gruppi di curve Bézier.

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        Dim SeriePunti1() As Point = {New Point(150, 40), New Point(200, 20), New
Point(300, 0), New Point(150, 180)}
        Dim SeriePunti2() As Point = {New Point(150, 40), New Point(100, 20), New
Point(0, 0), New Point(150, 180)}

        Dim Cuore As New Drawing2D.GraphicsPath
        Cuore.AddBeziers(SeriePunti1)
        Cuore.AddBeziers(SeriePunti2)

        ' Assegna a questo oggetto/percorso un pennello con colori gradienti dal
centro all'esterno:
        Dim Pennello As New Drawing2D.PathGradientBrush(Cuore)

        ' determina il colore al centro del pennello:
        Pennello.CenterColor = Color.White

        ' determina i colori all'esterno del pennello (in questo esempio solo
uno: il colore rosso):
        Dim Gammacolori As Color() = {Color.Red}
        Pennello.SurroundColors = Gammacolori

        ' colora l'oggetto/percorso
        e.Graphics.FillPath(Pennello, Cuore)
    End Sub

End Class
```

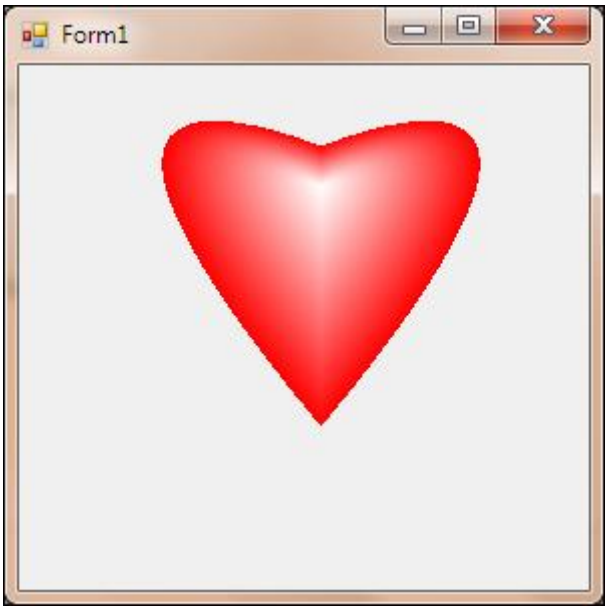


Figura 185: Un oggetto/percorso a forma di cuore, creato con due curve Bézier.

141: La Classe Region.

La classe **Region** consente di ritagliare un oggetto (un controllo o il form stesso) per dargli una forma originale, creata dal programmatore, diversa dalle forme standard; il controllo così ritagliato avrà le stesse funzionalità del controllo originale.
La nuova forma del controllo o del form è impostata con un oggetto/percorso grafico, preventivamente definito con la classe GraphicsPath.
L'esempio seguente crea un oggetto/percorso grafico con la classe GraphicsPath, gli assegna un cerchio e ritaglia un pulsante Button, dandogli la forma di questo cerchio.
La forma iniziale del pulsante è questa:

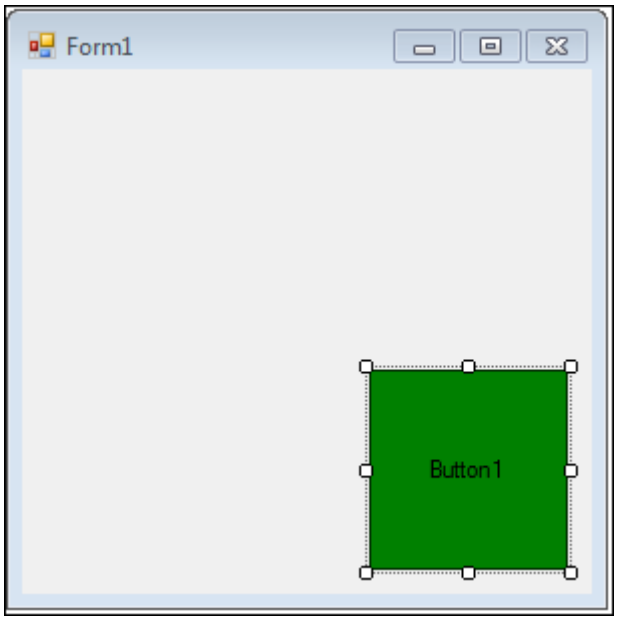


Figura 186: Un pulsante da ritagliare con la classe Region.

Questo è il codice che ritaglia il pulsante, dandogli la forma di un cerchio inscritto in un'area quadrata che parte dal pixel 10, 10, all'interno del pulsante Button1, e ha i lati di 80 pixel.

```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load

        ' crea un nuovo oggetto/percorso grafico:
        Dim NuovoPercorso As New Drawing2D.GraphicsPath

        ' assegna una ellisse al percorso grafico:
        NuovoPercorso.AddEllipse(10, 10, 80, 80)

        'ritaglia il Button1 dandogli la forme del percorso grafico:
        Button1.Region = New Region(NuovoPercorso)
```

```
    End Sub

End Class
```

Ecco lo stesso pulsante, dopo l'operazione di ritaglio, quando il programma è in esecuzione:

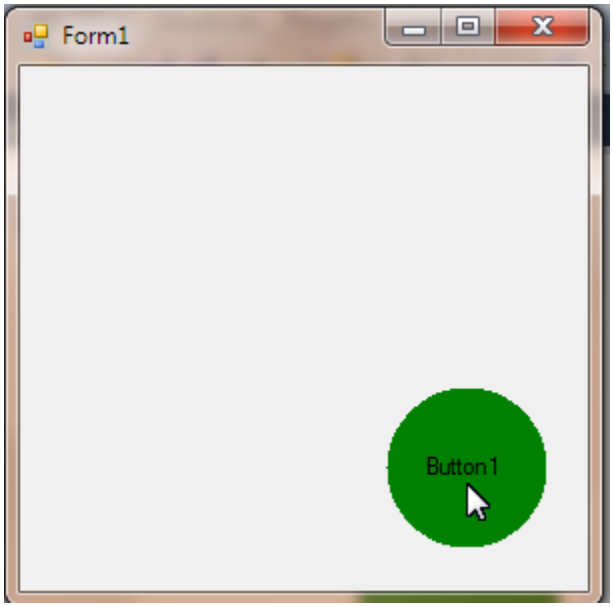


Figura 187: Un pulsante ritagliato con la classe Region.

Ecco un altro esempio di ritaglio dello stesso pulsante: il codice seguente crea un oggetto/percorso grafico a forma di poligono con soli tre punti (dunque: un triangolo) e ritaglia il pulsante dandogli questa forma:

```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load

        ' crea un nuovo oggetto/percorso grafico:
        Dim NuovoPercorso As New Drawing2D.GraphicsPath

        ' crea una matrice di punti con tre punti corrispondenti ai tre vertici
della forma triangolare che si vuole dare al pulsane Button1:
        Dim Punti() As Point = {New Point(10, 10), New Point(80, 50), New
Point(10, 80)}

        ' assegna un poligono con i tre punti al percorso grafico:
        NuovoPercorso.AddPolygon(Punti)

        'ritaglia il Button1 dandogli la forma del percorso grafico:
        Button1.Region = New Region(NuovoPercorso)

    End Sub

End Class
```

Ecco lo stesso pulsante con la forma triangolare, dopo l'operazione di ritaglio, quando il programma è in esecuzione:

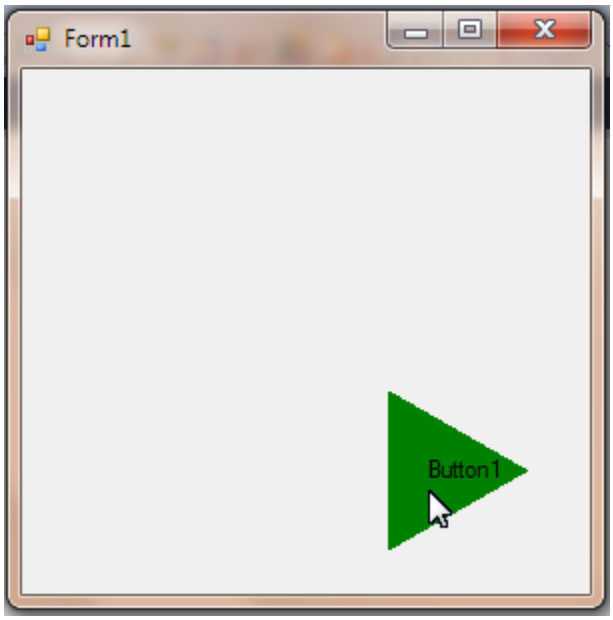


Figura 188: Un pulsante ritagliato a forma di triangolo.

Con le stesse modalità è possibile ritagliare un form.
Il codice seguente ritaglia un form dandogli la forma di un cerchio.
Prendiamo, come esempio, il form con l'immagine della Terra che abbiamo usato in numerosi esercizi, e applichiamo a questo form una classe Region con un oggetto/percorso a forma di cerchio:

Figura 189: Un form con l'immagine della Terra.

```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load

        ' crea un nuovo oggetto/percorso grafico:
        Dim NuovoPercorso As New Drawing2D.GraphicsPath

        ' assegna un'ellisse al percorso grafico:
        NuovoPercorso.AddEllipse(15, 30, 265, 260)

        'ritaglia il form dandogli la forma del percorso grafico:
        Me.Region = New Region(NuovoPercorso)

    End Sub

End Class
```

Ecco lo stesso form, dopo l'operazione di ritaglio, con il programma in esecuzione:

Figura 190: Un form ritagliato con la classe Region.

L'esempio seguente mostra un form ritagliato con un oggetto/percorso più complesso, formato da tre elementi: un semicerchio a sinistra, un rettangolo al centro e un semicerchio a destra.

```
Public Class Form1

    Private Sub Form1_Load() Handles MyBase.Load

        ' crea un nuovo oggetto/percorso grafico:
        Dim NuovoPercorso As New Drawing2D.GraphicsPath

        ' crea e assegna al percorso grafico l'ellisse di sinistra:
        NuovoPercorso.AddPie(0, 0, 200, 200, 90, 180)

        ' crea e assegna al percorso grafico il rettangolo centrale:
        Dim Rettangolo As New Rectangle(100, 0, 100, 200)
        NuovoPercorso.AddRectangle(Rettangolo)
```

```
        ' crea e assegna al percorso grafico l'ellisse di destra:
        NuovoPercorso.AddPie(100, 0, 200, 200, 270, 180)

        'ritaglia il form dandogli la forma del percorso grafico:
        Me.Region = New Region(NuovoPercorso)

    End Sub

End Class
```

Ecco il form ritagliato, quando il programma è in esecuzione:

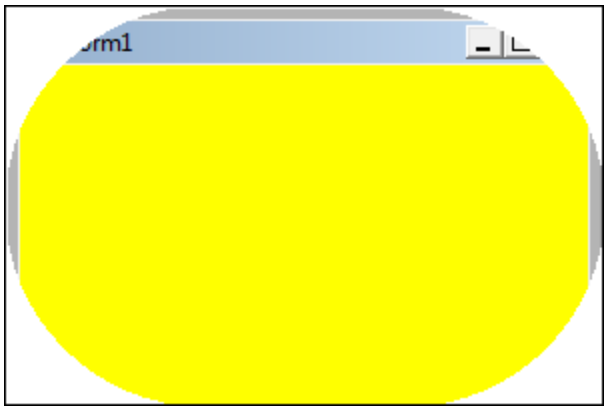


Figura 191: Un form ritagliato a forma ovale.

Esercizio 88: Ritaglio di un form con una scritta e lo sfondo trasparente.

In questo esercizio vediamo la creazione di un form ritagliato con la classe **Region**, sulla base di un oggetto/percorso creato con la classe **GraphicsPath**.
In questo caso l'oggetto/percorso creato con la classe GraphicsPath non è una forma geometrica, ma una scritta.
Lo sfondo del form è trasparente.
Siccome nell'operazione di ritaglio la barra del titolo del form è destinata a scomparire, il codice del programma comprende anche due procedure per gestire le operazioni di spostamento del form e di chiusura del programma, che normalmente si trovano nella barra in alto.
Il controllo dei movimenti del mouse, qui usato per spostare il form, è illustrato dettagliatamente in un altro capitolo del manuale[74].
Un doppio *clic* del mouse sulla parte visibile del form chiude il programma.
Apriamo un nuovo progetto, copiamo e incolliamo nella Finestra del Codice questo listato:

[74] Capitolo 33: GESTIONE DEL MOUSE., a pag. 755.

```
Public Class Form1

    ' Crea una variabile per registrare la posizione del mouse quando viene
premuto il tasto sinistro del mouse sul form:
    ' (una variabile di tipo Point memorizza due parametri: X e Y, cioè distanza
dal bordo sinistro e distanza dal bordo superiore)
    Dim PosizioneMouse As Point
```

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        'imposta il form: senza bordo, colore di sfondo rosso, cursore "mano",
dimensioni 300 x 300 pixel:
        Me.FormBorderStyle = Windows.Forms.FormBorderStyle.None
        Me.BackColor = Color.Red
        Me.Cursor = Cursors.Hand
        Me.Width = 300
        Me.Height = 300

        ' crea un nuovo oggetto/percorso, che comprenderà un quadrato, un cerchio
e una scritta:
        Dim NuovoPercorso As New Drawing2D.GraphicsPath()

        ' crea un quadrato nell'oggetto/percorso:
        Dim Rettangolo As New Rectangle(0, 0, 300, 300)
        NuovoPercorso.AddRectangle(Rettangolo)

        ' aggiunge un cerchio all'oggetto/percorso:
        NuovoPercorso.AddEllipse(30, 30, 240, 240)

        ' aggiunge una scritta all'oggetto/percorso:
        Dim Testo As String = "PROVA" & vbCrLf & "di" & vbCrLf & "RITAGLIO"
        Dim FormatoScritta As New StringFormat
        FormatoScritta.LineAlignment = StringAlignment.Center
        FormatoScritta.Alignment = StringAlignment.Center
        NuovoPercorso.AddString(Testo, New FontFamily("arial"), FontStyle.Bold,
36, Rettangolo, FormatoScritta)

        'ritaglia il form secondo l'oggetto/percorso creato con la classe
GraphicsPath
        Me.Region = New Region(NuovoPercorso)

    End Sub
```

```
    Private Sub Form1_DoubleClick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.DoubleClick

        ' un doppio clic del mouse chiude il programma
        Me.Close()

    End Sub

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown

        ' registra la posizione del mouse quando il mouse viene premuto sul form
        PosizioneMouse.X = e.X
        PosizioneMouse.Y = e.Y
```

```
    End Sub

    Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove

        ' sposta la finestra sullo schermo seguendo il movimento del mouse
(quando il pulsante sinistro è premuto)
        If e.Button = Windows.Forms.MouseButtons.Left Then
            Me.Location = New Point(Me.Left + (e.X - PosizioneMouse.X), Me.Top +
(e.Y - PosizioneMouse.Y))
        End If

    End Sub

End Class
```

Nel codice, notiamo una particolarità.
L'oggetto/percorso è costruito con tre oggetti sovrapposti l'uno all'altro: un quadrato, un cerchio e una scritta:

```
        NuovoPercorso.AddRectangle(Rettangolo)

        NuovoPercorso.AddEllipse(30, 30, 240, 240)

        NuovoPercorso.AddString(Testo, New FontFamily("arial"),
FontStyle.Bold, 36, Rettangolo, FormatoScritta)
```

Mandando in esecuzione il programma, notiamo che gli oggetti non si sovrappongono **sommandosi** uno all'altro, altrimenti vedremmo solo un quadrato rosso indistinto, in quanto il cerchio e la scritta si confonderebbero nel quadrato più grande sottostante.
Gli oggetti si sovrappongono invece **alternandosi** l'uno all'altro, per cui ogni oggetto assume il comportamento contrario a quello dell'oggetto sottostante.
Se il primo oggetto è *non trasparente*, il secondo oggetto è *trasparente*, il terzo oggetto è *non trasparente*, ecc.
Nel nostro caso abbiamo dunque:

- un quadrato non trasparente;
- un cerchio trasparente;
- una scritta non trasparente.

Ecco un'immagine del programma in esecuzione:


PROVA
di
RITAGLIO

Capitolo 27: VISUALIZZARE E TRASFORMARE IMMAGINI.

Un'immagine digitale è una griglia ordinata di dati in codice binario; il computer *vede* un'immagine come una sequenza di cifre 0 e 1[75].
Un file che contiene un'immagine è diviso in sezioni diverse che riportano rispettivamente informazioni su:

- il formato del file;
- il sistema di codifica con il quale il file è stato realizzato;
- il livello di compressione dei dati utilizzato per risparmiare memoria;
- la mappatura vera e propria della immagine, pixel per pixel.

142: Formati d'immagini.

Le immagini digitali si suddividono in due grandi famiglie: le immagini **vettoriali** e le immagini a mappa di punti (**bitmap**).

Immagini vettoriali

Un'immagine vettoriale è visualizzata sul monitor come un insieme di elementi geometrici: linee, curve, poligoni, cerchi o ellissi.
Ingrandire o ridurre un'immagine vettoriale significa ingrandire o ridurre gli elementi geometrici che la compongono; in queste operazioni si mantengono al meglio le qualità grafiche dell'immagine originale (in particolare: la sua nitidezza).
I formati di file più diffusi per salvare e utilizzare immagini vettoriali sono i formati .wmf e .emf, chiamati anche **metafile**.

[75] Il termine *digitali* deriva dall'inglese *digit* (cifra) ed equivale a *numeriche*.

Immagini a mappe di punti (bitmap).

Un'immagine a mappa di punti (**bitmap**) è visualizzata sul monitor come una griglia di *pixel*, cioè come una griglia di punti, che sono gli elementi più piccoli in cui si può suddividere lo schermo.
In pratica, un'immagine bitmap larga 30 pixel ed alta 30 pixel viene visualizzata sul monitor come una griglia di 900 punti (30 x 30), in cui ogni punto è portatore di una minuscola *macchia* di uno specifico colore.
Qui sotto vediamo un'immagine di queste dimensioni (30 x 30 pixel):

Figura 192: Un'immagine a mappa di punti.

Qui vediamo la stessa immagine ingrandita in modo da rendere visibili i 900 pixel che la compongono (il pixel colorato in rosso ha le coordinate 10, 3):

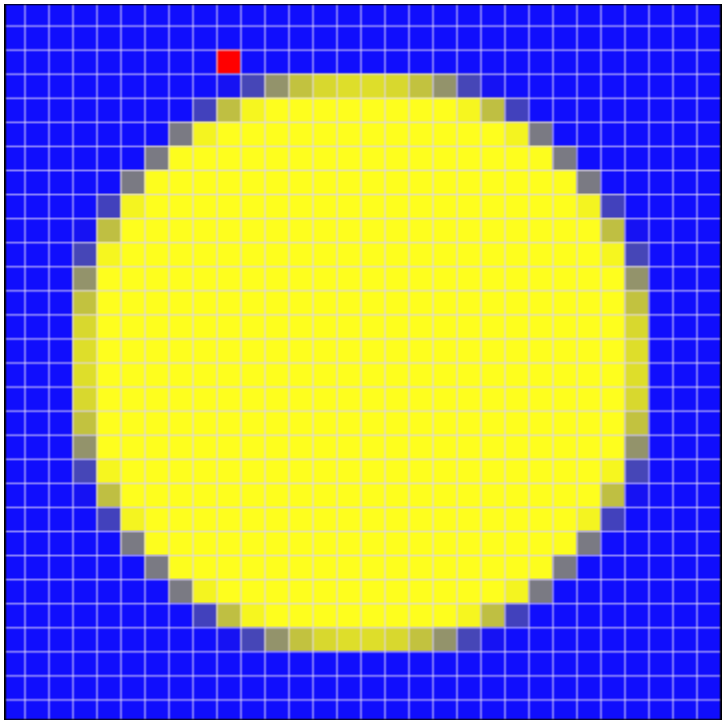


Figura 193: Inserimento di un'imagine a mappa di punti.

I pixel sono di formato talmente ridotto che l'occhio umano non riesce a percepirli separatamente, per cui queste immagini ci appaiono come un insieme **continuo** di linee, di motivi, di aree colorate.
Ogni singolo pixel ha una collocazione precisa nella griglia ed è individuabile indicandone le coordinate come si farebbe nel gioco della battaglia navale: questo

consente di ritoccare le immagini con estrema precisione, agendo anche punto per punto.
La densità dei pixel presenti nel monitor è chiamata **risoluzione**.
La misura di questa risoluzione è calcolata contando il numero di punti presenti in un pollice e si esprime in **dpi** (*dots per inch*, punti per ogni pollice).
I monitor hanno generalmente una risoluzione di 96 dpi; in un programma VB è dunque consigliabile utilizzare immagini con la stessa risoluzione, in modo che vi sia corrispondenza tra la risoluzione delle immagini e quella dello schermo.

Le immagini bitmap sono salvate nel formato **.bmp**. Questo è il formato che memorizza il maggior numero di dati relativi all'immagine, per cui occupa uno spazio maggiore nella memoria del computer.
Esistono diversi sistemi di compressione dei dati di una bitmap per salvare le immagini in file che occupano uno spazio minore di memoria, senza danneggiare troppo la qualità dell'immagine originale.
I più diffusi tra questi sistemi di compressione creano file nei formati .jpg, .gif, .png.
Come scelta predefinita VB salva le immagini nel formato .png, ma il programmatore può scegliere altri formati.

143: Visualizzare immagini.

La visualizzazione di un'immagine sul monitor in un programma VB può essere ottenuta in due modi diversi:

- il programmatore **assegna l'immagine** a un controllo (Form, Label, PictureBox, ecc.);
- il programmatore **crea una superficie grafica**, con i metodi che conosciamo[76], e **disegna l'immagine** su questa superficie.

Il primo metodo consiste in un'operazione *fisica* piuttosto che in un'operazione *grafica*: il programmatore mette un'immagine in un controllo, come se mettesse una fotografia in una cornice.
Il secondo metodo richiede al programmatore due operazioni:

- la creazione di una superficie grafica;
- il disegno dell'oggetto bitmap sulla superficie grafica.

Tra i due metodi vi è una differenza fondamentale, da tenere presente prima di optare per l'uno o per l'altro:

- l'immagine assegnata a un controllo permane in questo controllo sino a quando viene cancellata o sostituita da un'altra immagine, secondo le istruzioni scritte nel programma;

[76] Paragrafo 126: La classe Graphics., a pag. 553.

- l'immagine disegnata su una superficie grafica permane su questa superficie sino a quando si verifica un evento Paint. L'evento Paint del form si verifica, ad esempio, quando l'utente del programma modifica le dimensioni del form; in questo caso, tutte le superfici grafiche presenti nel form sono ridisegnate con la procedura Form1_Paint; le immagini sono dunque ridisegnate solo se si trovano all'interno della procedura Form1_Paint.

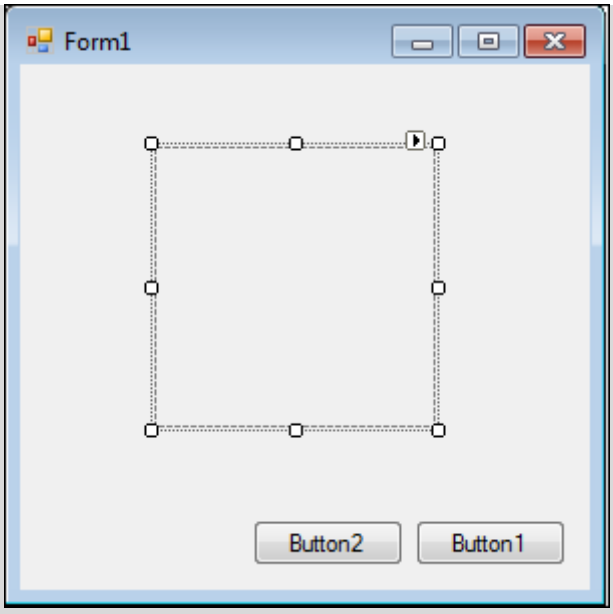
L'immagine da visualizzare, in entrambi i metodi, può trovarsi nel disco fisso del computer o nelle risorse del programma che saranno parte integrante del programma stesso, quando questo sarà finito.
Nel primo caso, quando si distribuirà il programma l'immagine presa dal disco fisso dovrà essere distribuita **assieme al programma**; nel secondo caso sarà sufficiente distribuire **solo il programma**, perché le risorse del programma ne sono parte integrante.

144: Assegnare un'immagine a un controllo.

E' possibile assegnare un'immagine a un form e a tutti i controlli che hanno la proprietà **Image**: **PictureBox**, **Button**, **Label**, ecc.; il controllo espressamente progettato per contenere immagini è il **PictureBox** (= contenitore d'immagine).
Il metodo più semplice per assegnare un'immagine a questi controlli nella fase di progettazione di un'applicazione consiste nell'impostare le proprietà Image o BackgroundImage che abbiamo visto nella parte iniziale di questo manuale.
Vedremo di seguito, invece come è possibile effettuare queste operazioni dal codice, con il programma in esecuzione.
E' possibile assegnare a un controllo un'immagine che si trova nel computer del programmatore, oppure nelle risorse che possono essere inserite nel programma come se ne fossero parte integrante.
I due metodi sono esemplificati negli esercizi che seguono.

Esercizio 89: Visualizzare un'immagine esistente nel computer.

Questo programma carica un'immagine in memoria, prelevandola dal disco fisso del computer, e la assegna a un controllo **PictureBox** al verificarsi dell'evento *clic* su un pulsante Button.
Apriamo un nuovo progetto, inseriamo nel form due controlli Button e un controllo PictureBox, come in questa immagine:



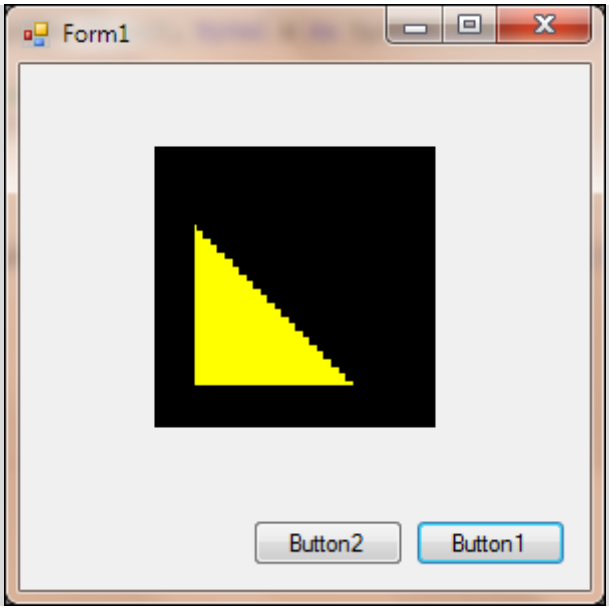
Il controllo PictureBox ha le dimensioni di 140 x 140 pixel.
Ora copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        Dim PercorsoFile As String
        PercorsoFile = My.Computer.FileSystem.SpecialDirectories.MyDocuments &
("\A scuola con VB 2010\Immagini\Triangolo.bmp")
        ' Preleva l'immagine "Triangolo" dal disco fisso e la assegna al
PictureBox:
        PictureBox1.Image = Image.FromFile(PercorsoFile)

    End Sub

    Private Sub Button2_Click() Handles Button2.Click

        ' Cancella l'immagine dal PictureBox:
        PictureBox1.Image = Nothing

    End Sub

End Class
```

Notiamo che il programma preleva l'immagine Triangolo.bmp dalla Cartella Documenti \ A scuola con VB 2010 \ Immagini:

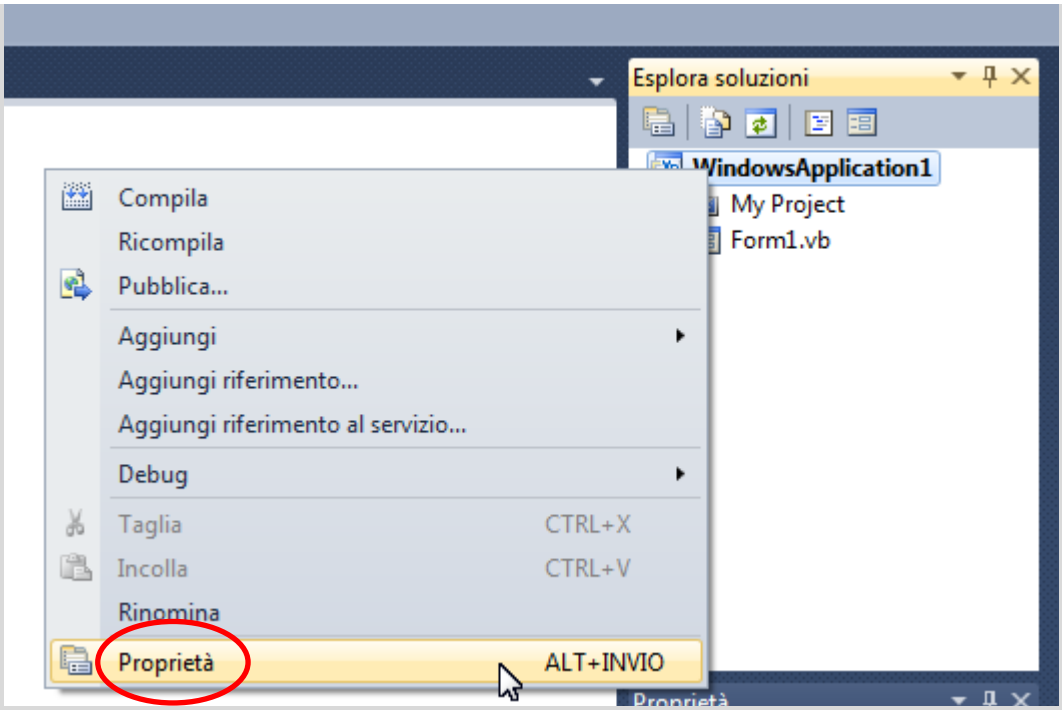
- Quando si verifica l'evento *clic* sul Button1, l'immagine Triangolo.bmp è assegnata al controllo PictureBox1.
- Quando si verifica l'evento *clic* sul Button2, l'immagine viene cancellata.

Ecco un'immagine del programma in esecuzione:

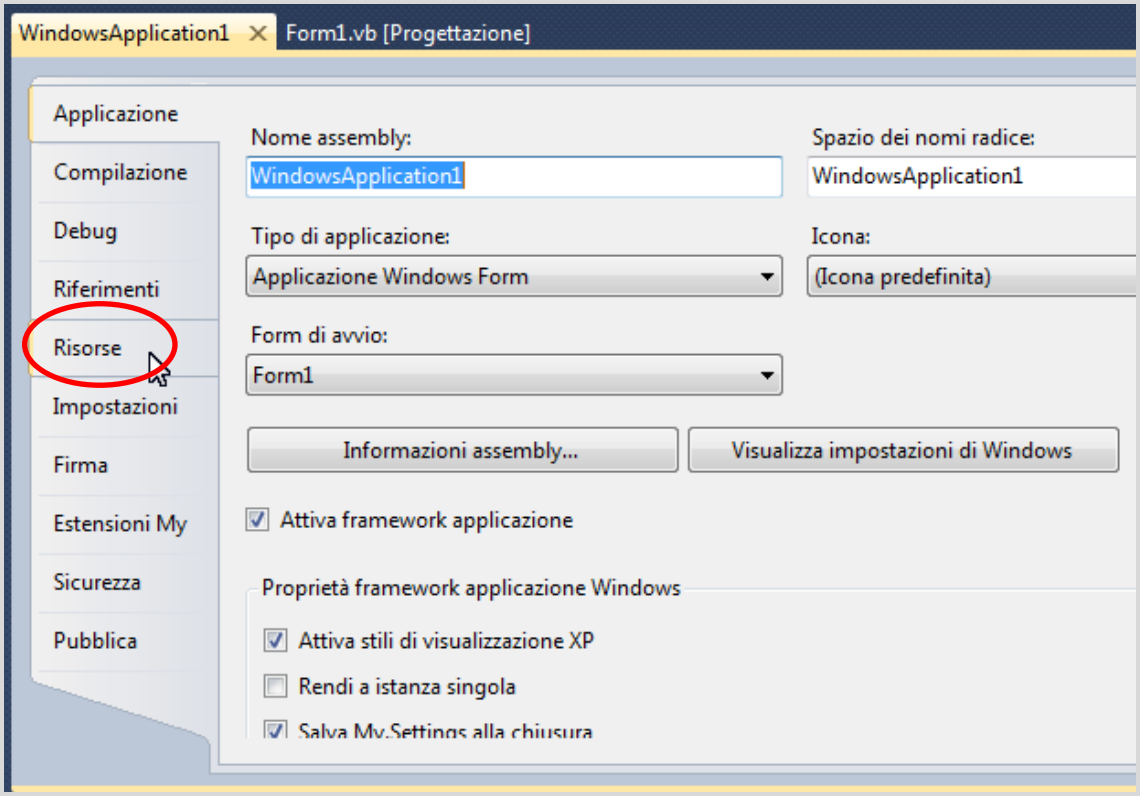


Esercizio 90: Visualizzare un'immagine dalle risorse del programma.

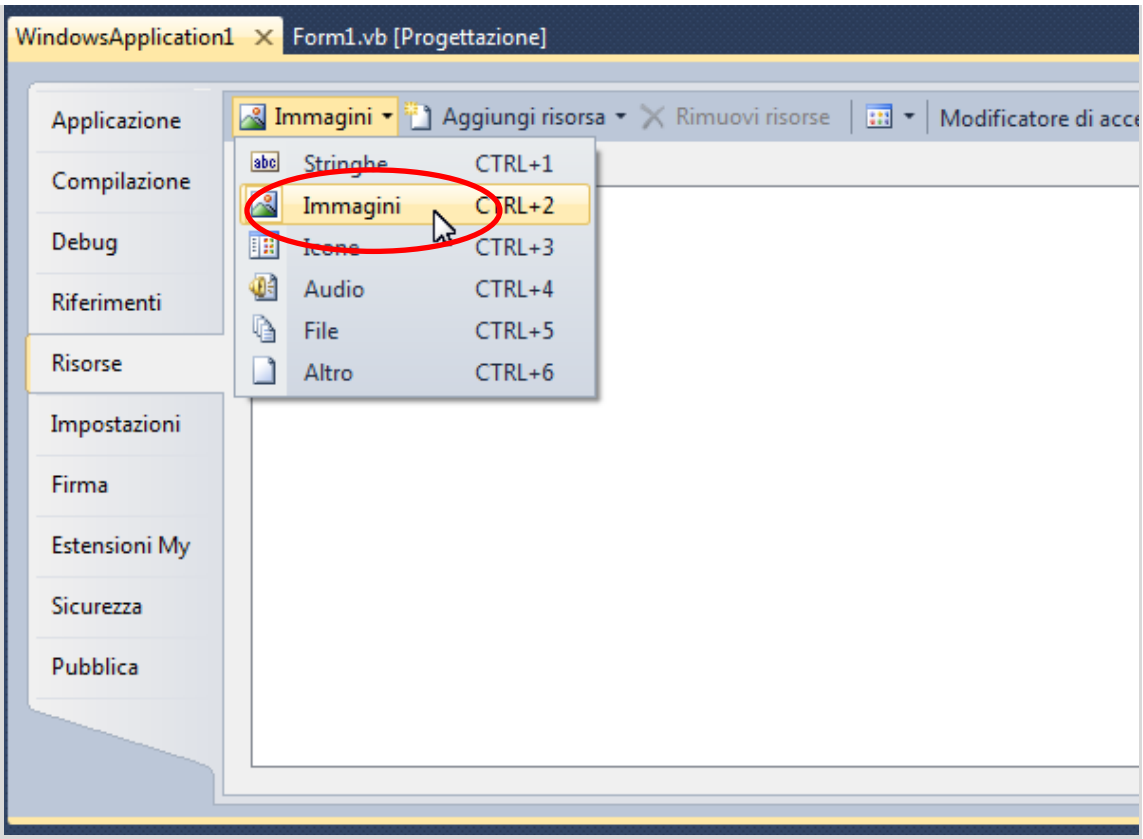
Questo programma carica un'immagine in memoria, prelevandola dalle risorse del programma, e la assegna a un controllo PictureBox.
Apriamo un nuovo progetto, inseriamo nel form due controlli Button e un controllo PictureBox come nell'esercizio precedente.
Per il funzionamento del programma è necessario copiare l'immagine Triangolo.bmp, presente nella cartella **Documenti \ A scuola con VB 2010 \ Immagini**, nelle risorse del programma.
Per fare questo, nella finestra **Esplora soluzioni**, facciamo un *clic* con il tasto destro del mouse sul nome dell'applicazione e poi sul menu **Proprietà**:



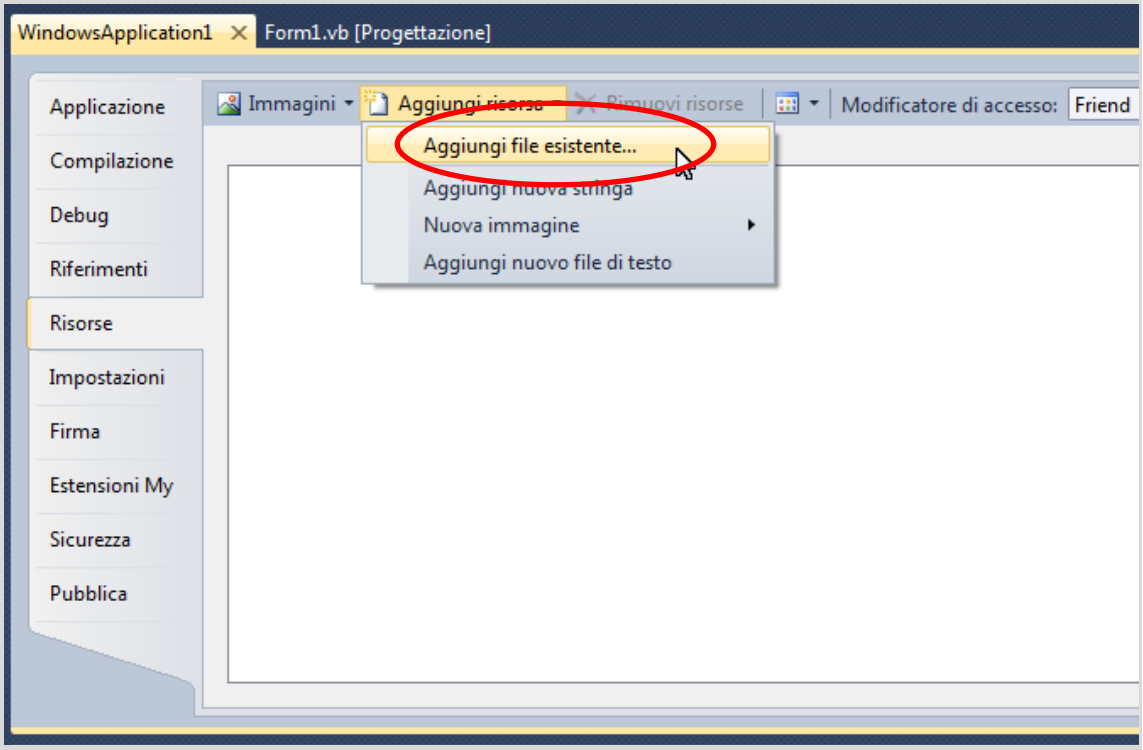
Nella scheda che si apre, facciamo *clic* su **Risorse**:



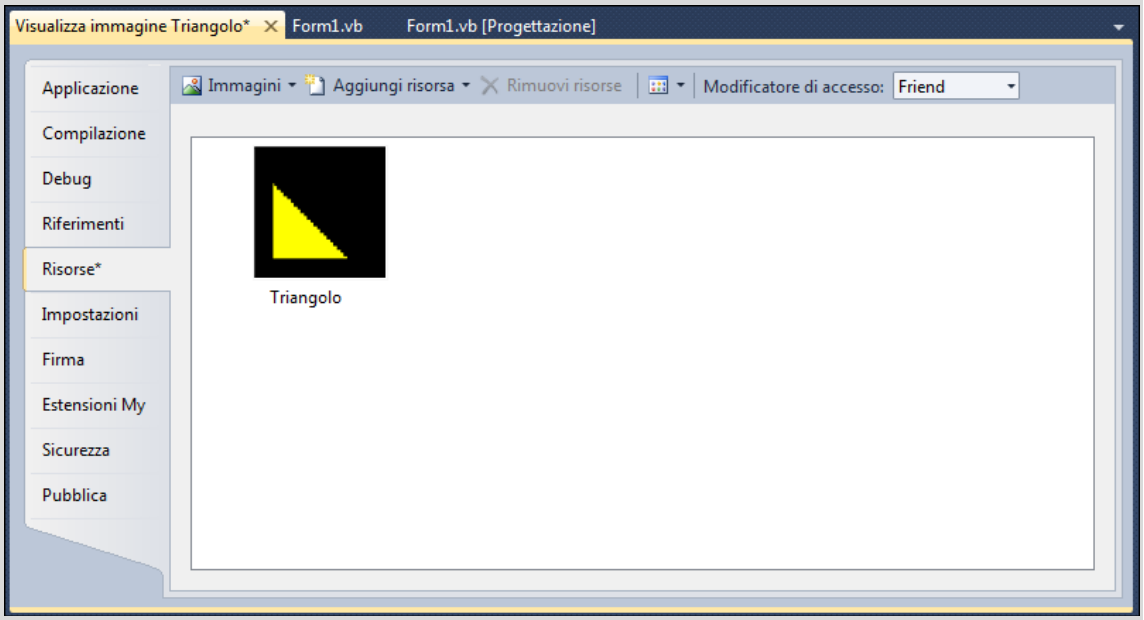
Quindi facciamo *clic* su **Immagini**:



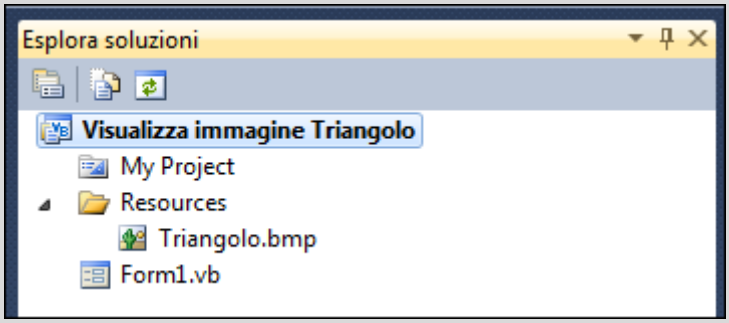
Clic su **Aggiungi file esistente**:



Nella cartella **Documenti \ A scuola con VB \ Immagini** facciamo *clic* sull'immagine Triangolo.bmp:



Ecco come compare ora la finestra **Esplora soluzioni**:

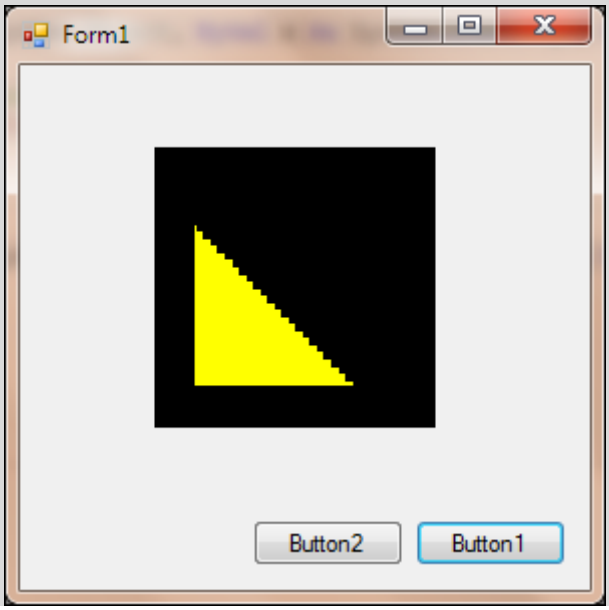


Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click(s) Handles Button1.Click

        ' Preleva l'immagine "Triangolo" dalle risorse del programma e la assegna
al PictureBox:
        PictureBox1.Image = My.Resources.Triangolo

    End Sub

    Private Sub Button2_Click() Handles Button2.Click

        ' Cancella l'immagine dal PictureBox:
        PictureBox1.Image = Nothing

    End Sub
```

```
End Class
```

La prima procedura assegna al controllo PictureBox l'immagine prelevata dalle risorse, la seconda procedura cancella l'immagine contenuta nel PictureBox.
Il risultato finale è identico a quello dell'esercizio precedente:



145: Trasformare un'immagine.

Nei due esercizi precedenti abbiamo visto come visualizzare in un controllo un'immagine presa dal disco fisso del computer o dalle risorse del programma.
VB fornisce al programmatore strumenti per elaborare questa immagine, con operazioni di trasformazione da effettuare prima che l'immagine sia visualizzata in un controllo.
Le operazioni da effettuare in questo caso sono tre:

- lettura dell'immagine dal disco fisso o dalle risorse del programma;
- elaborazione dell'immagine;
- assegnazione dell'immagine a un controllo.

Le attività della seconda fase, elaborazione dell'immagine, quando sono di una certa complessità non si svolgono direttamente sull'immagine originale, ma su un'immagine virtuale che rimane invisibile all'utente del programma durante le operazioni di trasformazione, e che viene poi visualizzata su un controllo.
Questa immagine virtuale è creata dal programmatore utilizzando le classi **Metafile** o **Bitmap**:

- con la classe **Metafile** si possono elaborare immagini vettoriali nei formati .wmf e .emf;

- con la classe **Bitmap** si possono elaborare immagini nei formati .bmp, .gif, .exif, .jpg o .jpeg, .png, .tiff, .ico.

Essendo le immagini a matrici di punti di gran lunga le più diffuse, negli esercizi di questo manuale ci limiteremo all'uso della classe **Bitmap**.

Per creare un'immagine virtuale con la classe Bitmap è sufficiente un comando di questo tipo:

```
        Dim ImmagineVirtuale As Bitmap = My.Resources.Triangolo
```

Il codice seguente mostra un esempio di utilizzo di un'immagine virtuale; si tratta di una modifica dell'ultimo esercizio: qui vediamo la fase di creazione dell'immagine virtuale e la fase di assegnazione di questa immagine virtuale a un controllo PictureBox:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Preleva l'immagine "Triangolo" dalle risorse del programma e la assegna
a un'immagine virtuale:
        Dim ImmagineVirtuale As Bitmap = My.Resources.Triangolo

        ' *** Qui si inseriscono le trasformazioni dell'immagine virtuale

        ' Assegna l'immagine virtuale al controllo PictureBox:
        PictureBox1.Image = ImmagineVirtuale

    End Sub

    Private Sub Button2_Click() Handles Button2.Click

        ' Cancella l'immagine dal PictureBox:
        PictureBox1.Image = Nothing

    End Sub

End Class
```

Nello spazio intermedio segnato da asterischi si collocheranno le trasformazioni che vedremo nei prossimi paragrafi. In questo esempio, non sono previste trasformazioni, per cui mandando il programma in esecuzione si avrà lo stesso risultato dell'ultimo esercizio:

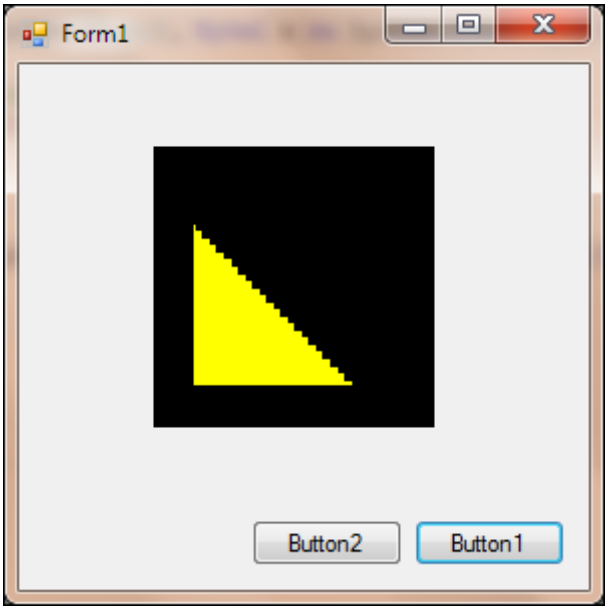


Figura 194: Creazione e uso di un'immagine virtuale con la classe Bitmap.

146: Ruotare un'immagine.

La rotazione di un'immagine può essere effettuata su un piano bidimensionale (rotazione di 90, 180, 270 gradi, procedendo in senso orario come si può ruotare una fotografia disposta su un tavolo) oppure tridimensionale, rotando l'immagine sul suo asse centrale orizzontale o verticale.
Il comando utilizzato per queste operazioni è **RotateFlip**, che consente al programmatore di scegliere tra diversi modi di rotazione dell'immagine, con diverse combinazioni di rotazioni bidimensionali e/o tridimensionali.
Ne vediamo un esempio nel codice seguente, da copiare e incollare nella Finestra del Codice dello stesso programma utilizzato negli ultimi esercizi:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea un'immagine virtuale nella memoria del programma, con la classe
Bitmap:
        Dim ImmagineVirtuale As Bitmap = My.Resources.Triangolo

        ' Ruota l'immagine virtuale di 90 gradi sul piano bidimensionale:
        ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate90FlipNone)

        'Assegna l'immagine virtuale al controllo PictureBox1:
        PictureBox1.Image = ImmagineVirtuale

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        PictureBox1.Image = Nothing

    End Sub

End Class
```

Il comando della rotazione è contenuto in questa riga:

```
        ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate90FlipNone)
```

Il tipo di rotazione **Rotate90FlipNone** indica una rotazione sul piano di 90 gradi, senza alcuna rotazione tridimensionale.
Ecco un'immagine del programma in esecuzione:

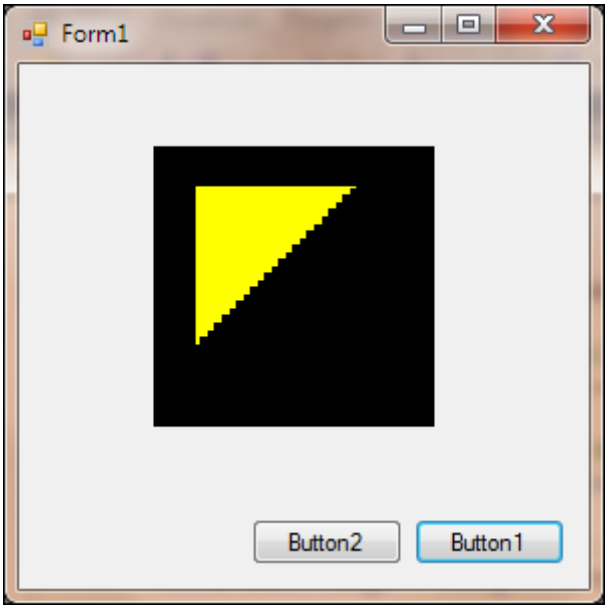


Figura 195: Rotazione di un'immagine di tipo Rotate90FlipNone.

Impostando il comando RotateFlip in questo modo:

```
        ImmagineVirtuale.RotateFlip(RotateFlipType.RotateNoneFlipX)
```

si ottiene una rotazione dell'immagine in senso orizzontale, attorno al suo asse verticale:

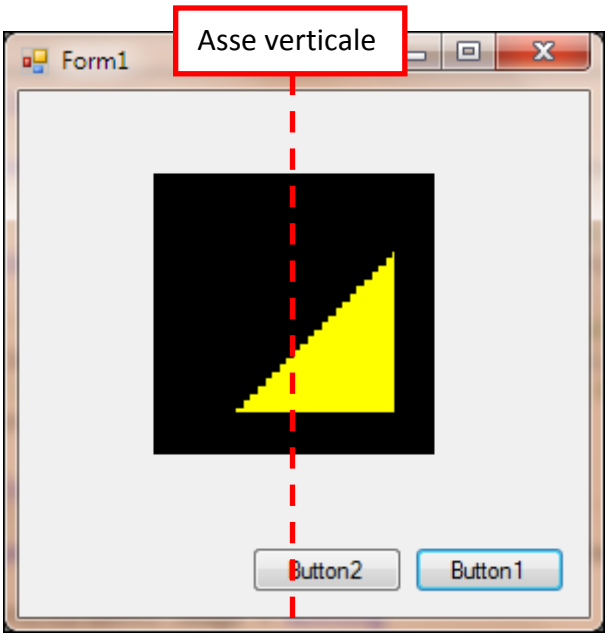


Figura 196: Rotazione di un'immagine sul suo asse verticale.

Impostando il comando RotateFlip in questo modo:

```
ImmagineVirtuale.RotateFlip(RotateFlipType.RotateNoneFlipY)
```

si ottiene una rotazione dell'immagine in senso verticale, attorno al suo asse orizzontale:

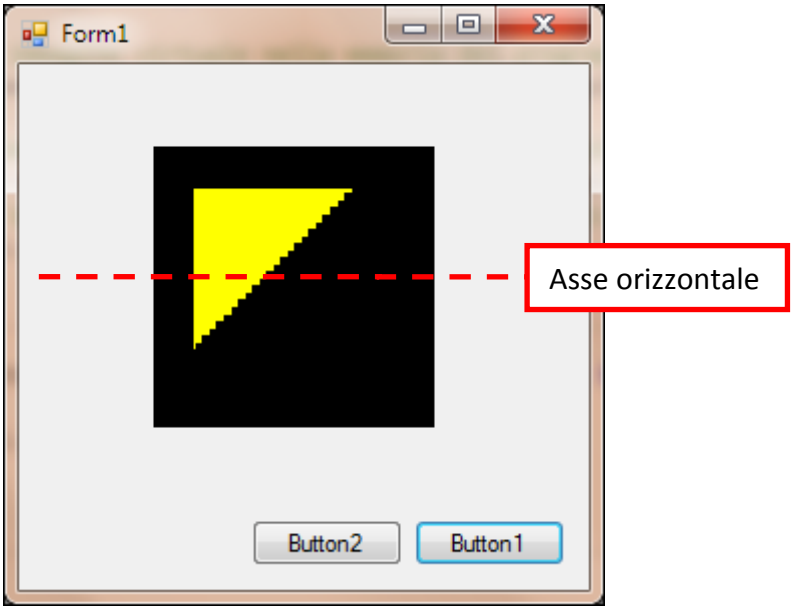


Figura 197: Rotazione di un'immagine sul suo asse orizzontale.

147: Rendere trasparente un colore in un'immagine.

Un altro effetto disponibile su un'immagine virtuale creata con la classe Bitmap è la gestione della trasparenza: il programmatore può decidere quale, tra i colori della immagine originale, risulterà invisibile durante il programma.
Il comando che gestisce la trasparenza è

```
ImmagineVirtuale.MakeTransparent()
```

Il colore scritto tra parentesi sarà il colore trasparente. Se tra le parentesi non è scritto alcun colore, il colore trasparente è il **nero**.
Vediamo un esempio nel codice seguente, da copiare e incollare nella Finestra del Codice dell'ultimo esercizio:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea un'immagine virtuale nella memoria del programma, con la classe
Bitmap:
        Dim ImmagineVirtuale As Bitmap = My.Resources.Triangolo

        ' Crea un'immagine nella memoria del programma
        ImmagineVirtuale.MakeTransparent()

        'Assegna l'immagine virtuale al controllo PictureBox1:
        PictureBox1.Image = ImmagineVirtuale

    End Sub

    Private Sub Button2_Click() Handles Button2.Click

        PictureBox1.Image = Nothing

    End Sub

End Class
```

Mandando in esecuzione questo programma, otteniamo questo risultato:

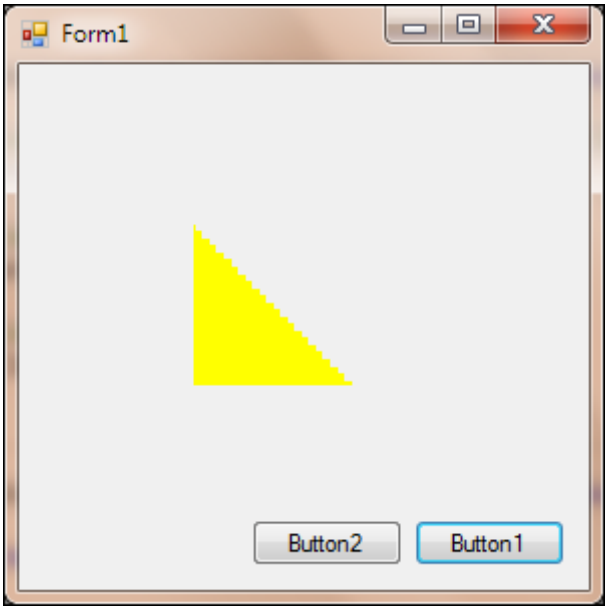


Figura 198: Trasparenza del colore di fondo in un'immagine.

Sostituendo la riga

```
ImmagineVirtuale.MakeTransparent()
```

con questa riga

```
ImmagineVirtuale.MakeTransparent(Color.Yellow)
```

otteniamo la trasparenza del colore giallo:

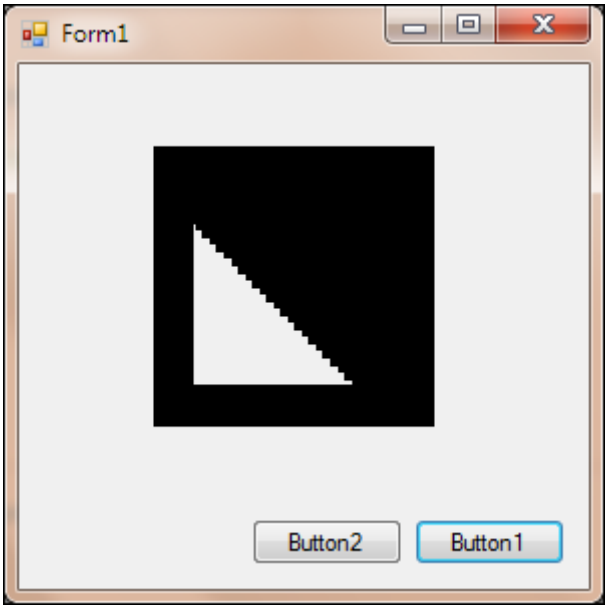


Figura 199: Trasparenza di un colore in un'immagine.

148: Disegnare immagini su superfici grafiche.

Nei paragrafi precedenti abbiamo visto come visualizzare immagini con il metodo che assegna l'immagine al controllo destinato a contenerla: si tratta di un metodo facile da usare che però non si adatta o si adatta male a trasformazioni complesse dell'immagine originale, o a trasformazioni effettuate in rapida sequenza.
Per eseguire operazioni di questo tipo è necessario ricorrere a un metodo alternativo di visualizzazione delle immagini, con il quale il programmatore **crea una superficie grafica** e **disegna l'immagine** direttamente su questa superficie.
La superficie grafica così creata può avere le stesse dimensioni del controllo che la contiene e l'immagine finale sembra visualizzata direttamente nel controllo, ma in realtà il controllo è **sottostante** la superficie grafica, come se l'immagine fosse disegnata su una pellicola posta **sopra** il controllo stesso.

Il disegno di un'immagine su una superficie grafica si ottiene con il comando **DrawImage**. Questo comando richiede che in precedenza il programmatore abbia provveduto alla creazione di una superficie grafica sulla quale disegnare/visualizzare l'immagine virtuale[77].

Esercizio 91: Disegno di un'immagine su una superficie grafica.

In questo esercizio vediamo come disegnare un'immagine su una superficie grafica, prelevando l'immagine originale dalle risorse del programma, con il passaggio intermedio della creazione di un'immagine virtuale.
Le operazioni preliminari sono identiche a quelle dell'Esercizio 90, a pag. 644.
Il codice da copiare e incollare nella Finestra del Codice è questo:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea una superficie grafica corrispondente al controllo PictureBox1:
        Dim Superficie1 As Graphics = PictureBox1.CreateGraphics

        ' Crea un'immagine virtuale con la classe Bitmap:
        Dim ImmagineVirtuale As Bitmap = My.Resources.Triangolo

        ' Crea il punto d'inizio del disegno dell'immagine nella Superficie1:
        Dim AngoloAltoSinistra As New Point(0, 0)

        ' Disegna l'immagine nel form a partire dal punto AngoloAltoSinistra:
        Superficie1.DrawImage(ImmagineVirtuale, AngoloAltoSinistra)
```

[77] Paragrafo 126: La classe Graphics., a pag. 553.

```
    End Sub

    Private Sub Button2_Click() Handles Button2.Click

        PictureBox1.Invalidate()

    End Sub

End Class
```

Le diverse operazioni sono spiegate all'interno del codice.
Notiamo in particolare, nella prima procedura:

- la creazione di una superficie grafica con la classe Graphics;
- la creazione di un'immagine virtuale con la classe Bitmap;
- la creazione di un punto (AngoloAltoSinistro) dal quale fare partire il disegno dell'immagine;
- il disegno dell'immagine con il comando DrawImage.

Nella seconda procedura notiamo la cancellazione dell'immagine disegnata nel controllo PictureBox1 con il comando **Invalidate**.

149: Ingrandire e ridurre immagini.

L'ingrandimento o la riduzione di un'immagine si ottengono mediante la creazione di un'immagine virtuale, indicando la posizione e le dimensioni che si vogliono dare alla immagine modificata.
Nell'esercizio precedente, modifichiamo il codice come segue:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea un oggetto immagine nella memoria del programma
        Dim Immagine1 As Bitmap = My.Resources.Triangolo

        ' Crea una superficie grafica corrispondente al controllo PictureBox1:
        Dim Superficie1 As Graphics = PictureBox1.CreateGraphics

        ' Crea il rettangolo di visualizzazione dell'immagine nella Superficie1
        Dim Rettangolo As New Rectangle(0, 0, 150, 150)

        ' Disegna l'immagine nel form
        Superficie1.DrawImage(Immagine1, Rettangolo)

    End Sub

    Private Sub Button2_Click() Handles Button2.Click

        PictureBox1.Invalidate()

    End Sub
```

```
End Class
```

Notiamo nel codice che la riduzione dell'immagine viene effettuata definendo un'area rettangolare nella quale si collocherà l'immagine modificata:

```
        Dim Rettangolo As New Rectangle(0, 0, 150, 150)
```

I parametri dell'area rettangolare, scritti tra parentesi, indicano il punto in alto a sinistra in cui collocare l'immagine trasformata e le nuove dimensioni (larghezza e altezza) che l'immagine dovrà avere.
Ecco un'immagine del programma in esecuzione:

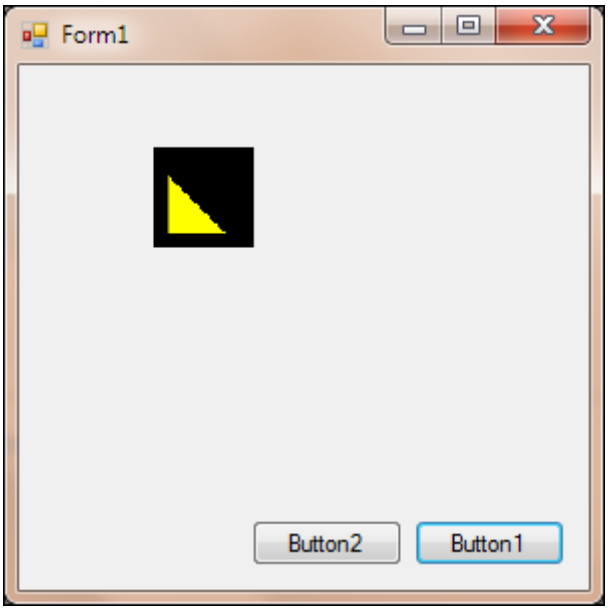


Figura 200: Riduzione di un'immagine.

150: Distorcere un'immagine.

Anche l'effetto di distorsione di un'immagine si ottiene mediante la creazione di un'immagine virtuale, i cui quattro angoli sono spostati secondo le indicazioni del programmatore.
I controlli contenitori di immagini in VB sono di forma rettangolare e un'immagine virtuale creata sulla base di questi controlli è di foma rettangolare.
La distorsione dell'immagine consiste dunque nello spostamento degli angoli del rettangolo per farne un romboide o parallelogramma.
Le nuove posizioni in cui si devono collocare i quattro angoli sono contenute in una matrice di punti.

Essendo la figura finale, dopo la distorsione, un parallelogramma, è necessario indicare solo le nuove posizioni di primi tre angoli, in quanto la posizione del quarto angolo è calcolata in modo automatico da VB.
Nell'esercizio precedente, modifichiamo il codice come segue:

```
Public Class Form1

    Private Sub Button1_Click() Handles Button1.Click

        ' Crea una superficie grafica corrispondente al controllo PictureBox1:
        Dim Superficie1 As Graphics = PictureBox1.CreateGraphics

        ' Crea un'immagine virtuale con la classe Bitmap:
        Dim ImmagineVirtuale As Bitmap = My.Resources.Triangolo

        Dim Punti(2) As Point
        'angolo superiore a sinistra:
        Punti(0) = New Point(0, 0)
        'angolo superiore a destra:
        Punti(1) = New Point(100, 30)
        ' angolo inferiore a sinistra:
        Punti(2) = New Point(30, 100)    '

        ' Disegna l'immagine deformata nella superficie grafica:
        Superficie1.DrawImage(ImmagineVirtuale, Punti)

    End Sub
```

```
    Private Sub Button2_Click() Handles Button2.Click

        PictureBox1.Invalidate()

    End Sub

End Class
```

Notiamo, nel codice, la creazione di una matrice di punti: ogni punto indica la nuova posizione di un angolo del rettangolo originale.
Ecco un'immagine del programma in esecuzione (i punti indicati dal programmatore sono cerchiati in rosso, il punto calcolato da VB è cerchiato in blu):

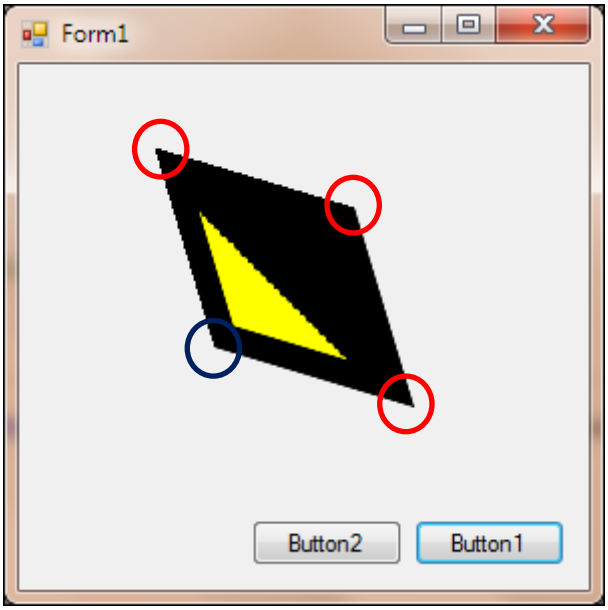
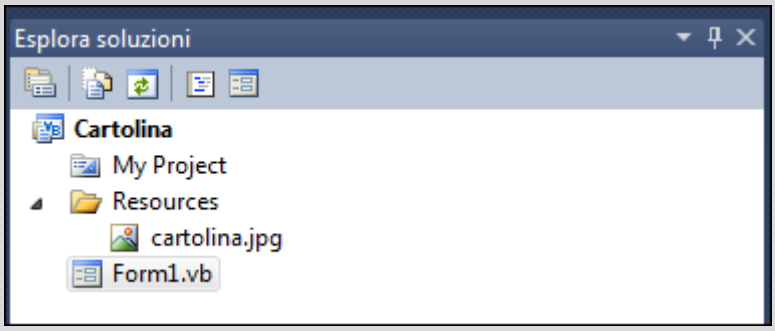


Figura 201: Distorsione di un'immagine.

Esercizio 92: Distorsione di un'immagine.

In questo esercizio creiamo un programma con il quale l'utente potrà distorcere un'immagine spostandone gli angoli con il mouse.
Apriamo un nuovo progetto.
Seguendo le indicazioni che si trovano nell'Esercizio 90: Visualizzare un'immagine dalle risorse del programma. 644, apriamo le risorse di questo progetto e vi inseriamo l'immagina cartolina.jpg, che si trova nella cartella Documenti \ A scuola con VB \ Immagini:



Impostiamo queste proprietà del Form1:
- BackColor = Black
- DoubleBuffered = True

(la proprietà DoubleBuffered riduce l'effetto di sfarfallìo quando il form viene ridisegnato velocemente).

Collochiamo nel Form1 un controllo PictureBox e gli assegniamo l'immagine cartolina.jpg, che si trova nelle risorse del programma.
Collochiamo ancora nel Form1 tre pulsanti Button, delle dimensioni di 16 x 16 pixel, come in questa immagine:

Con il programma in esecuzione, l'utente potrà trascinare questi pulsanti cliccandoli con il tasto sinistro del mouse; gli angoli dell'immagine si muoveranno con i pulsanti deformando l'immagine.
In pratica, a ogni movimento di uno dei pulsanti il programma registra la posizione dei tre pulsanti e ridisegna l'immagine collocandone i quattro angoli nelle nuove posizioni.
Nel codice vediamo anche alcune tecniche di controllo dei movimenti del mouse (registrazione del punto in cui il mouse è premuto e registrazione dei suoi movimenti), queste tecniche verranno esposte più dettagliatamente più avanti nel manuale, nel Capitolo 33: GESTIONE DEL MOUSE.791.
Ora copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' crea un'immagine virtuale, non visibile all'utente, destinata a ricevere le
trasformazioni grafiche prima che queste siano visualizzate nel form:
    ' questa immagine virtuale assume le dimensioni e il contenuto del Form1:

    Dim ImmagineVirtuale As Bitmap = New Bitmap(Me.Width, Me.Height)

    'crea una superficie grafica corrispondente alla immagine virtuale appena
creata:
    Dim Superficie1 As Graphics = Graphics.FromImage(ImmagineVirtuale)

    ' crea due variabili per memorizzare la posizione del pulsante spostato dal
mouse
    ' ricordiamo che una variabile di tipo Point ha una struttura con due
parametri:
    ' punto.X = distanza da sinistra
```

```
    ' punto.Y = distanza dall'alto

    Dim PosizioneVecchia As Point
    Dim PosizioneNuova As Point
```

```
    Private Sub Clic(ByVal sender As Object, ByVal e As MouseEventArgs) Handles
Button1.MouseDown, Button2.MouseDown, Button3.MouseDown

        ' quando il mouse si abbassa su uno dei pulsanti, la variabile
PosizioneVecchia ne memorizza la posizione:
        PosizioneVecchia = e.Location

    End Sub
```

```
    Private Sub MuoviMouse(ByVal sender As Object, ByVal e As MouseEventArgs)
Handles Button1.MouseMove, Button2.MouseMove, Button3.MouseMove

        ' questa procedura gestisce lo spostamento del Button cliccato dal mouse
e la distorsione della immagine del triangolo.

        ' La variabile sender (= mittente) memorizza quale pulsante è stato
cliccato dal mouse, tra i tre pulsanti presenti nel form.
        ' La variabile e (= evento) memorizza quale azione è stata compiuta sul
pulsante con il mouse.

        If e.Button = MouseButtons.Left Then
            ' i comandi tra If e End If vengono eseguiti solo se il mouse viene
spostato sul form tenendo premuto il tasto sinistro del mouse:

            ' memorizza la posizione del mouse
            PosizioneNuova = e.Location
            ' e sposta il pulsante cliccato, portandolo nella nuova posizione:
            sender.Location = New Point(sender.Left + (PosizioneNuova.X -
PosizioneVecchia.X), sender.Top + (PosizioneNuova.Y - PosizioneVecchia.Y))

            ' ripulisce la superficie grafica dal disegno precedente:
            Superficie1.Clear(Color.Black)

            ' memorizza i tre punti mediani in cui si trovano i tre pulsanti
            Dim Punti(2) As Point
            Punti(0) = New Point(Button1.Left + 8, Button1.Top + 8)
            Punti(1) = New Point(Button2.Left + 8, Button2.Top + 8)
            Punti(2) = New Point(Button3.Left + 8, Button3.Top + 8)

            ' disegna la superficie grafica con l'immagine cartolina (ogni angolo
corrisponde a uno dei punti sopra elencati)
            ' (ricordiamo che questo disegno viene effettuato nell’immagine
virtuale, per cui non è visibile)
            Superficie1.DrawImage(My.Resources.cartolina, Punti)

            ' copia l'immagine virtuale nel Form1, rendendola visibile:
            Me.BackgroundImage = ImmagineVirtuale

            ' aggiorna il Form1:
            Me.Refresh()

        End If

    End Sub
```

```
End Class
```

Ecco un'immagine del programma in esecuzione:

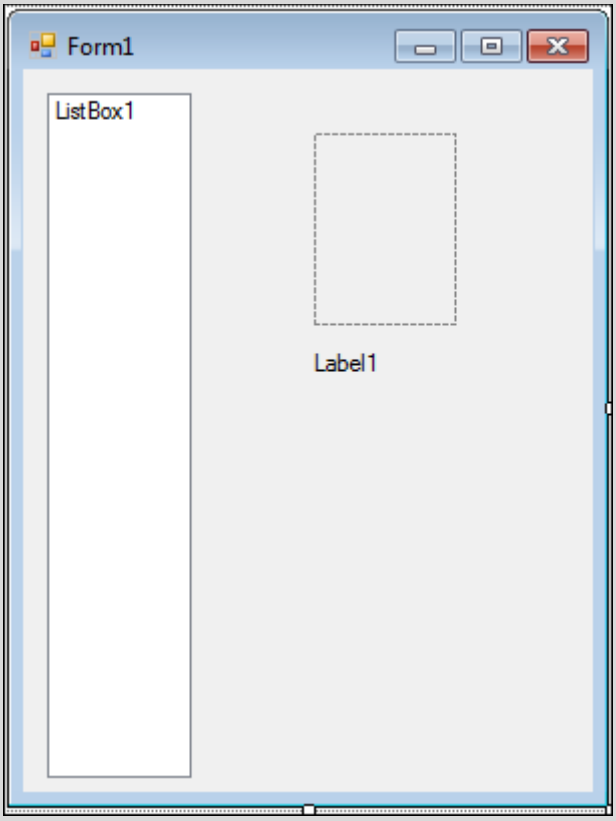
151: Inserire immagini in un ListBox.

Nell'esercizio seguente vedremo come è possibile inserire un gruppo di immagini in un controllo ListBox, normalmente progettato per visualizzare un elenco di opzioni scritte.
In questo programma notiamo in particolare:

- la creazione di una lista di immagini;
- l'assegnazione di queste immagini al ListBox, una ad una, adattandone le dimensioni allo spazio disponibile;
- la gestione della selezione di una carta nel ListBox da parte dell'utente.

Esercizio 93: Un ListBox con immagini.

Questo programma visualizza, in un ListBox, le carte di un mazzo di carte da ramino, in formato ridotto, una carta per ogni riga del ListBox.
Quando l'utente clicca una carta, questa è visualizzata nelle sue dimensioni originali in un PictureBox, e il suo nome è visualizzato in una Label.
Il programma utilizza le immagini che si trovano nella cartella **Documenti \ A scuola con VB \ Immagini \ Carte**.
Apriamo un nuovo progetto e inseriamo nel Form un ListBox, un PictureBox e una Label, come in questa immagine:



Proprietà del Form1:

- Size = 300; 400

Proprietà del PictureBox:

- Size = 71; 96

Copiamo e incolliamo nella Finestra del Codice questo listato, che contiene i commenti necessari alla sua comprensione:

```
Public Class Form1

    ' Crea una variabile per memorizzare il percorso della cartella con le
immagini delle carte:
    Dim PercorsoImmagini As String
```

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Me.Load

        Label1.Text = ""

        ' All'avvio del programma, crea una lista di immagini con le immagini
contenute nella cartella
        ' Documenti \ A scuola con VB 2010 \ Immagini \ Carte:

        PercorsoImmagini = My.Computer.FileSystem.SpecialDirectories.MyDocuments
& "\A scuola con VB 2010\Immagini\Carte"
        Dim Immagini = My.Computer.FileSystem.GetFiles(PercorsoImmagini).ToList
```

```
        ' la variabile PercorsoImmagini sarà simile a questo esempio:
        ' PercorsoImmagini = C:\Users\PaoloRossi\Documents\A scuola con VB
2010\Immagini\Carte

        ' la proprietà DrawMode.OwnerDrawFixed indica che la grafica del ListBox
sarà gestita dal programmatore,
        ' e che tutti gli oggetti visualizzati nel ListBox saranno ridotti ad una
medesima altezza, indicata dal programmatore:

        ListBox1.DrawMode = DrawMode.OwnerDrawFixed
        ListBox1.ItemHeight = 40

        ' Inserisce nel ListBox le immagini delle carte.
        ' Ogni immissione causa una variazione nel ListBox e dunque avvia la
porcedura ListBox1_DrawItem
        ' che adatta ogni immagine alle dimensioni della riga del ListBox:

        For Each Carta In Immagini
            ListBox1.Items.Add(Carta)
        Next

    End Sub
```

```
    Private Sub ListBox1_DrawItem(ByVal sender As Object, ByVal e As
System.Windows.Forms.DrawItemEventArgs) Handles ListBox1.DrawItem

        ' La proprietà DrawMode del ListBox è impostata su OwnerDrawFixed, per
cui
        ' ogni nuova immagine causa l'evento ListBox1.DrawItem e avvia questa
procedura.

        ' disegna lo sfondo della riga con il colore di sistema, in modo che,
quando sarà cliccata, questa funzionerà come una riga 'normale':
        e.DrawBackground()


        ' memorizza la posizione e le dimensioni dell'area (della riga) in cui va
collocata l'immagine e
        ' memorizza il numero progressivo di questa immagine:
        Dim DimensioniRiga As Rectangle = e.Bounds
        Dim AltezzaRiga As Integer = ListBox1.ItemHeight
        Dim NumeroProgressivo As Integer = e.Index

        ' recupera l'immagine che corrisponde all'indice progressivo della riga
nel ListBox:
        Dim Immagine =
Bitmap.FromFile(ListBox1.Items(NumeroProgressivo).ToString)

        ' Questi comandi controllano la dimensione dell'immagine e se queste
eccedono le dimensioni
        ' della riga nel ListBox le riducono in proporzioni uguali:

        Dim AreaImmagine As SizeF
        If Immagine.Height > AltezzaRiga Then
            Dim Riduzione As Single = AltezzaRiga / Immagine.Height
            AreaImmagine.Width = Immagine.Width * Riduzione
            AreaImmagine.Height = Immagine.Height * Riduzione
        Else
            ' se l'immagine non eccede l'altezza della riga le sue dimensioni
rimangono immutate:
```

```
            AreaImmagine.Width = Immagine.Width
            AreaImmagine.Height = Immagine.Height
        End If

        ' disegna l'immagine di una carta in una riga del ListBox, lasciando un
margine di 10 pixel a sinistra:
        e.Graphics.DrawImage(Immagine, DimensioniRiga.Left + 10,
DimensioniRiga.Top, AreaImmagine.Width, AreaImmagine.Height)

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged

        ' quando l'utente fa un clic su una carta nel ListBox,
        ' memorizza nella variabile NomeImmagine il percorso in cui si trova la
carta
        ' e visualizza la carta nel PictureBox1.

        Dim NomeImmagine = ListBox1.SelectedItem.ToString
        PictureBox1.Image = Image.FromFile(NomeImmagine)

        ' Inizia qui una serie di operazioni sulla variabile NomeImmagine per
estrapolarne
        ' il seme e il nome della carta da scrivere in maiuscolo sotto
l'immagine.

        ' I operazione
        NomeImmagine = My.Computer.FileSystem.GetName(NomeImmagine)
        ' risultato:
        ' NomeImmagine = Cuori1.gif

        ' II operazione
        ' per eliminare l'estensione .gif dal nome, estrapola da NomeImmagine la
parte della stringa
        ' che va dalla prima lettera al punto:
        NomeImmagine = NomeImmagine.Substring(0, InStr(NomeImmagine, ".") - 1)
        ' risultato:
        ' NomeImmagine = Cuori1

        ' III operazione
        ' determina il seme della carta:
        Dim Seme As String
        If NomeImmagine.Contains("Cuori") Then
            Seme = "Cuori"
        ElseIf NomeImmagine.Contains("Quadri") Then
            Seme = "Quadri"
        ElseIf NomeImmagine.Contains("Fiori") Then
            Seme = "Fiori"
        ElseIf NomeImmagine.Contains("Picche") Then
            Seme = "Picche"
        Else
            Label1.Text = ""
            Exit Sub
        End If
        ' risultato:
        ' Seme = Cuori

        ' IV operazione:
```

```
        ' cancella dal NomeImmagine il seme, in modo da avere solo il numero
della carta:
        NomeImmagine = NomeImmagine.Remove(0, Seme.Length)
        Dim NumeroCarta As Integer = CInt(NomeImmagine)
        ' risultato:
        ' NomeImmagine = 1

        ' V operazione:
        ' Crea una matrice con i nomi delle carte e assegna alla carta cliccata
il nome corrispondente al suo numero nella lista:
        Dim NomeCarta() As String = {"Asso", "Due", "Tre", "Quattro", "Cinque",
"Sei", "Sette", "Otto", "Nove", "Dieci", "Fante", "Donna", "Re"}
        NomeImmagine = NomeCarta(NumeroCarta - 1)
        ' risultato:
        ' NomeImmagine = Asso

        ' Visualizza il nome della carta, scritto in maiuscolo, nella Label1
        Label1.Text = (NomeImmagine & " di " & Seme).ToUpper

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

152: Riepilogo.

Il disegno di un'immagine, nel corso della esecuzione di un programma, avviene su una **superficie grafica**, che qui chiameremo **e.Graphics.**
Per le operazioni grafiche più elaborate l'immagine originale prima di essere disegnata nella superficie grafica **e.Graphics** deve essere **copiata** in una nuova **bitmap**, invisibile all'utente del programma, che qui chiameremo **ImmagineVirtuale**.

Negli esempi che seguono si presuppone che sia avviato un nuovo progetto con l'immagine casetta.jpg nelle risorse del programma.
*Questa immagine si trova nella cartella **Documenti \ A scuola con VB \ Immagini**.*
Gli esempi possono essere provati inserendo il listato di ogni esempio in queste righe di codice:

```
Public Class Form1

    ' IL LISTATO DELL'ESEMPIO VA INSERITO QUI

End Class
```

Per disegnare un'immagine

bisogna impostare il punto (angolo sinistra/alto) dal quale si desidera fare partire il disegno dell'immagine:

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)

        e.Graphics.DrawImage(ImmagineVirtuale, 0, 0)

    End Sub
```

Figura 202: Disegnare un'immagine con e.Graphics.DrawImage.

Per spostare un'immagine

bisogna impostare il punto (angolo sinistra/alto) dal quale si desidera fare partire il disegno dell'immagine:

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        e.Graphics.DrawImage(ImmagineVirtuale, 100, 50)

    End Sub
```

Figura 203: Spostare un'immagine.

Per centrare un'immagine nel form

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        Dim Sinistra As Integer = (Me.ClientRectangle.Width -
ImmagineVirtuale.Width) / 2
        Dim Altezza As Integer = (Me.ClientRectangle.Height -
ImmagineVirtuale.Height) / 2
        e.Graphics.DrawImage(ImmagineVirtuale, Sinistra, Altezza)

    End Sub
```

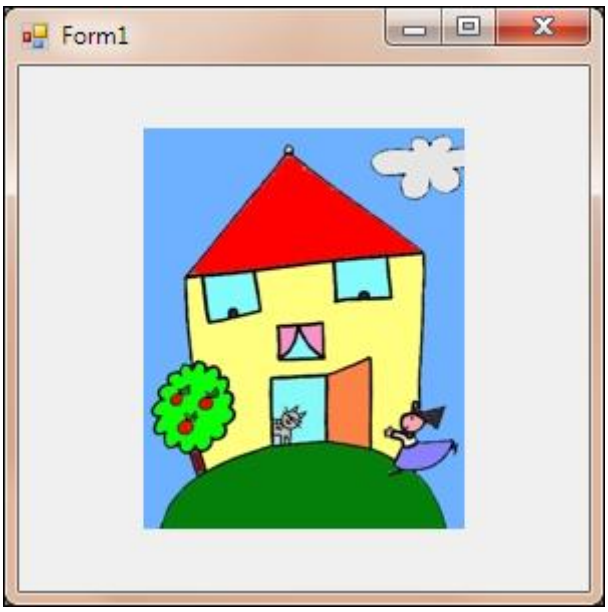


Figura 204: Centrare un'immagine nel form.

Per ingrandire o ridurre un'immagine

bisogna creare un rettangolo con le dimensioni desiderate per l'immagine:

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        Dim Rettangolo As New Rectangle(0, 0, 320, 400)
        e.Graphics.DrawImage(ImmagineVirtuale, Rettangolo)

    End Sub
```

Figura 205: Ingrandire un'immagine.

Per copiare una parte di un'immagine

bisogna creare un rettangolo con le dimensioni della parte ritagliata dall'originale:

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        Dim Rettangolo As New Rectangle(120, 120, 60, 60)
        e.Graphics.DrawImage(ImmagineVirtuale, Rettangolo, 120, 120, 60, 60,
GraphicsUnit.Pixel)

    End Sub
```

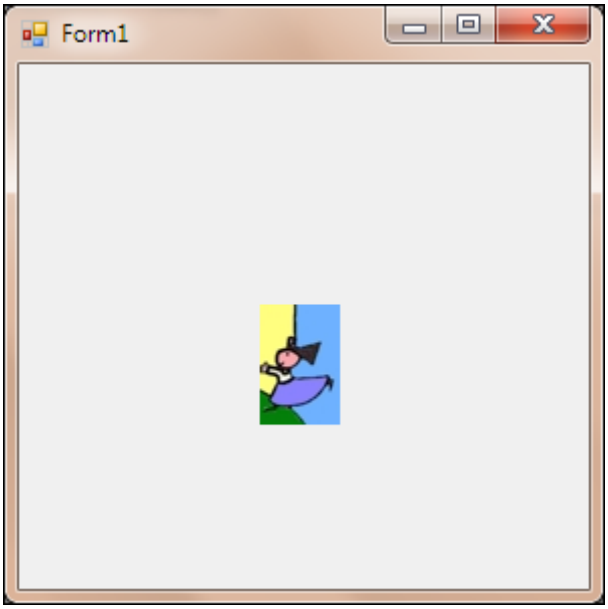


Figura 206: Copiare una parte di un'immagine.

Per ingrandire una parte di un'immagine

bisogna creare un rettangolo con le nuove dimensioni che si vogliono dare alla parte ritagliata dall'originale:

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        Dim Rettangolo As New Rectangle(120, 120, 120, 120)
        e.Graphics.DrawImage(ImmagineVirtuale, Rettangolo, 120, 120, 60, 60,
GraphicsUnit.Pixel)

    End Sub
```

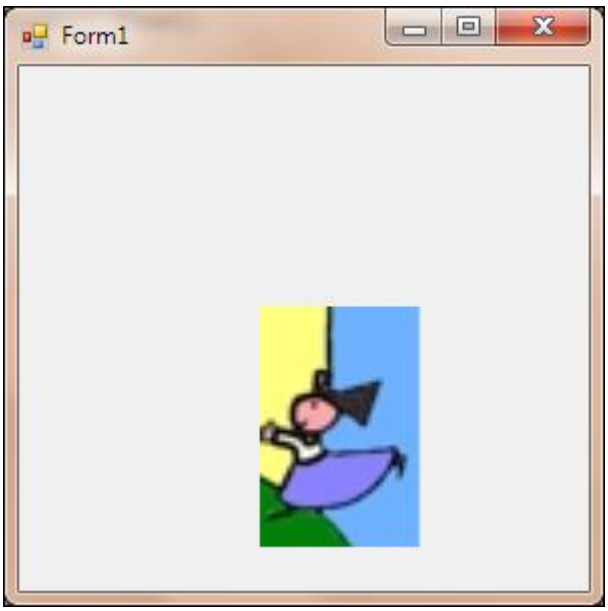


Figura 207: Ingrandire una parte di un'immagine.

Per ruotare un'immagine

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate180FlipNone)
        e.Graphics.DrawImage(ImmagineVirtuale, 0, 0)

    End Sub
```

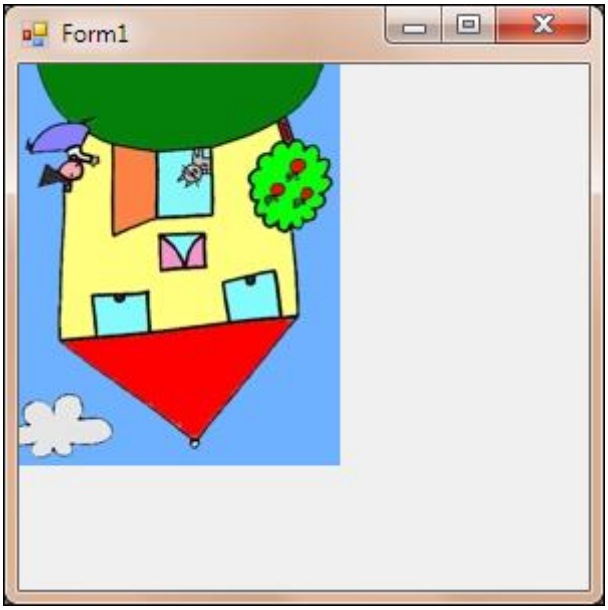


Figura 208: Ruotare un'immagine.

Per riflettere un'immagine a specchio

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        ImmagineVirtuale.RotateFlip(RotateFlipType.RotateNoneFlipX)
        e.Graphics.DrawImage(ImmagineVirtuale, 0, 0)

    End Sub
```

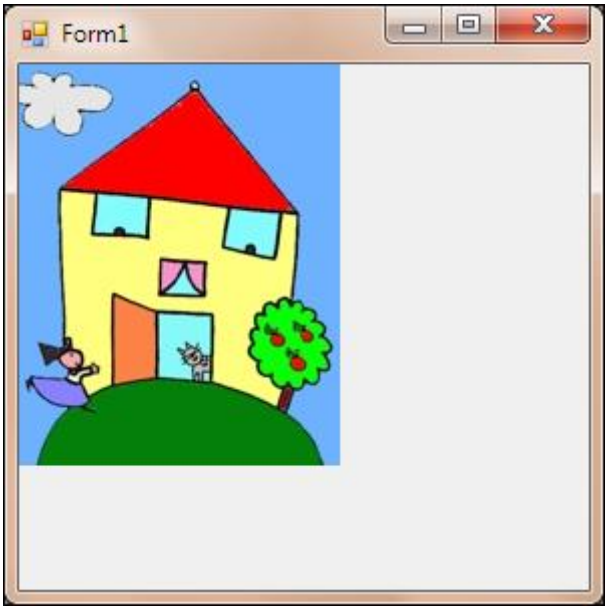


Figura 209: Riflettere un'immagine in verticale.

Per distorcere un'immagine

bisogna creare un parallelogramma, indicando le posizioni di tre angoli:

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim ImmagineVirtuale As New Bitmap(My.Resources.casetta)
        Dim Punti(2) As Point
        'angolo superiore a sinistra:
        Punti(0) = New Point(10, 10)
        'angolo superiore a destra:
        Punti(1) = New Point(200, 30)
        ' angolo inferiore a sinistra:
        Punti(2) = New Point(30, 200)    '
        e.Graphics.DrawImage(ImmagineVirtuale, Punti)

    End Sub
```

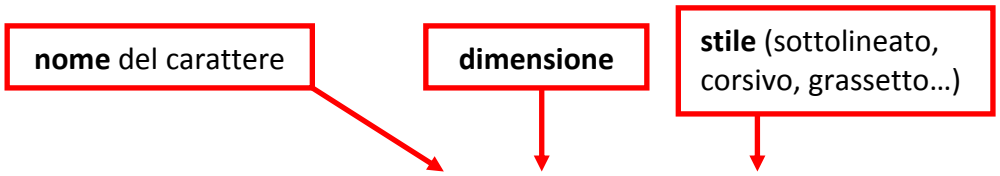
Figura 210: Distorcere un'immagine.

Capitolo 28: DISEGNARE SCRITTE.

Iniziamo con una premessa, riguardante la scelta dei caratteri.
In VB è possibile modificare la proprietà **Font** di un Form e di ogni controllo in esso contenuto, in fase di progettazione, dalla Finestra Proprietà.
Durante l'esecuzione del programma non è più possibile impostare questa proprietà direttamente, ma ogni volta che si desidera modificare la proprietà Font di un oggetto occorre creare un nuovo oggetto **Font.**
Ecco un esempio di come modificare il tipo di carattere in una casella di testo, in fase di esecuzione di un programma:

```
Dim NuovoCarattere As New Font("Arial", 36, FontStyle.Bold)
TextBox1.Font = NuovoCarattere
```

In questo capitolo avremo spesso a che fare con creazioni di nuovi caratteri, per cui ricordiamo che un nuovo carattere deve essere scelto indicando tre parametri, separati da virgole:




```
Dim NuovoCarattere As New Font("Arial", 36, FontStyle.Bold)
TextBox1.Font = NuovoCarattere
```

153: DrawString.

Il comando **DrawString** disegna una scritta in una superficie grafica.
Il codice seguente, al verificarsi dell'evento Paint sul form (cioè all'avvio del programma) esegue queste operazioni in successione:

- crea una scritta,
- crea un nuovo carattere,
- crea un oggetto pennello di colore blu e
- disegna la scritta su una superficie grafica corrispondente al Form1:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Testo As String = "Questo testo" & vbCrLf & "è disegnato" & vbCrLf &
"nel form."
        Dim Carattere As New Font("Arial", 25, FontStyle.Bold)
        Dim Pennello As Brush = Brushes.Blue
        Dim PuntoInizio As New Point(10, 10)

        e.Graphics.DrawString(Testo, Carattere, Pennello, PuntoInizio)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

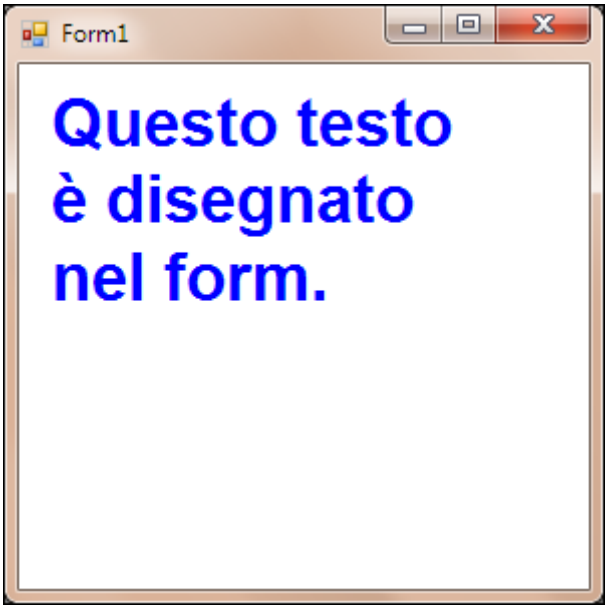


Figura 211: Il comando DrawString.

Notiamo che il comando DrawString richiede l'indicazione di quattro parametri, scritti tra parentesi:

```
        e.Graphics.DrawString(Testo, Carattere, Pennello, PuntoInizio)
```

1. la stringa di testo da disegnare;
2. il tipo di carattere;
3. il tipo di pennello;
4. il punto d'inizio della scritta.

154: Centrare una scritta.

Per disegnare una scritta al centro di una superficie grafica si ricorre alla classe **StringFormat**, che imposta la formattazione del testo:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Testo As String = "Questo testo" & vbCrLf & "è centrato" & vbCrLf
& "nel form."
        Dim Carattere As New Font("Arial", 25, FontStyle.Bold)
        Dim Pennello As Brush = Brushes.Blue

        ' ritaglia come area di disegno il rettangolo che corrisponde al
form:
        Dim AreaDisegno As Rectangle = Me.ClientRectangle

        ' crea un nuovo formato StringFormat:
        Dim Formato As New StringFormat
        ' centratura in senso verticale:
        Formato.LineAlignment = StringAlignment.Center
        ' centratura in senso orizzontale:
        Formato.Alignment = StringAlignment.Center

        e.Graphics.DrawString(Testo, Carattere, Pennello, AreaDisegno,
Formato)

    End Sub
```

```
    Private Sub Form1_Resize(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Resize

        ' ripulisce il form quando il form è ridimensionato (allargato o
ridotto)
        Me.Invalidate()

    End Sub

End Class
```

Ecco un'immagine di questo programma in esecuzione:

Figura 212: Una scritta centrata nel form.

Notiamo che, avendo definito un formato per centrare la scritta nel form, tra i parametri di **DrawString** non troviamo più il punto d'inizio della scritta, ma troviamo invece l'indicazione della superficie grafica in cui va collocata e centrata la scritta stessa:

```
        e.Graphics.DrawString(Testo, Carattere, Pennello, AreaDisegno,
Formato)
```

1. la stringa di testo da disegnare;
2. il tipo di carattere;
3. il tipo di pennello;
4. la superficie in cui va collocata la scritta;
5. il formato della scritta.

Nel codice precedente notiamo due riferimenti a proprietà dell'oggetto grafico **e**:

- **e.Graphics** indica l'oggetto all'interno del quale sarà eseguito il comando DrawString;
- **e.ClientRectangle** indica le coordinate di questo oggetto, cioè il punto iniziale, la larghezza e l'altezza.

In pratica, la proprietà **e.ClientRectangle** crea un rettangolo che corrisponde all'oggetto in cui andrà collocata la scritta. Nel codice precedente, essa memorizza le coordinate del Form1; usando la proprietà e.ClientRectangle, il programmatore può dunque indicare le dimensioni di questo rettangolo, senza preoccuparsi di misurarle.

Notiamo ancora la procedura **Form_Resize**, che contiene il comando **Me.Invalidate**. Quando l'utente ridimensiona il form (lo allarga o lo restringe), questa procedura ripulisce la superficie grafica e dunque cancella il testo visualizzato. Nello stesso

tempo, però, essendo cambiate le dimensioni del form, si attiva l'evento Paint che ridisegna la scritta ricollocandola al centro del form ridimensionato.

155: La struttura MeasureString.

La struttura **MeasureString** memorizza le dimensioni (larghezza e altezza) dell'area rettangolare occupata da una scritta.
Può essere utilizzata, come nell'esercizio seguente, per creare uno sfondo colorato e/o un contorno a una scritta, senza doversi sobbarcare laboriose misurazioni.
La struttura **MeasureString** è di tipo **SizeF**, vale a dire che i dati in essa contenuti sono espressi in numeri con la virgola e non in numeri interi.

Esercizio 94: La struttura MeasureString.

Questo programma traccia una scritta nel form e utilizza la struttura **MeasureString** per misurare lo spazio occupato da questa scritta. Le dimensioni ricavate con MeasureString sono assegnate a un rettangolo che viene colorato come sfondo della scritta.
Apriamo un nuovo progetto e inseriamo nel Form1 un controllo **NumericUpDown**, con la proprietà **Value = 24**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim Dimens_Carattere As Integer

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Comandi per il miglioramento della qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias

        Dim Testo As String = "Spazio" & vbCrLf & "occupato" & vbCrLf & "dal
testo"
        Dim TipoCarattere As New Font("Arial", Dimens_Carattere)
        e.Graphics.DrawString(Testo, TipoCarattere, Brushes.Black, 10, 10)

        ' Misura l'area occupata dal testo.
        Dim SpazioTesto As New SizeF(e.Graphics.MeasureString(Testo,
TipoCarattere))

        ' Colora il rettangolo corrispondente all'area del testo.
        e.Graphics.FillRectangle(Brushes.Yellow, 10, 10, SpazioTesto.Width,
SpazioTesto.Height)
```

```
        ' Traccia il contorno del rettangolo corrispondente all'area del testo.
        e.Graphics.DrawRectangle(Pens.Black, 10, 10, SpazioTesto.Width,
SpazioTesto.Height)

        ' Disegna il testo.
        e.Graphics.DrawString(Testo, TipoCarattere, Brushes.Red, 10, 10)

    End Sub
```

```
    Private Sub NumericUpDown1_ValueChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles NumericUpDown1.ValueChanged

        Dimens_Carattere = NumericUpDown1.Value
        Me.Invalidate()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

A ogni *clic* sul controllo NumericUpDown il programma esegue queste operazioni:

- cambia la dimensione dei caratteri della scritta (Dimens_Carattere = **NumericUpDown1.Value);**
- ripulisce il form (**Me.Invalidate**);
- attiva la procedura **Form1_Paint**;
- effettua una nuova misurazione della scritta, con la struttura **MeasureString**;
- ridisegna il rettangolo colorato e la scritta.

156: Ruotare una scritta.

L'effetto della rotazione di una scritta si ottiene creando una superficie grafica, destinata a contenere la scritta, e ruotandola con il sistema che abbiamo visto al paragrafo 132, a pag. 586.
Il codice seguente disegna la scritta "Testo ruotato di 45°" su una superficie grafica corrispondente al form, ruotata di 45 gradi.

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Comandi per il miglioramento della qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias

        ' Ruota la superficie grafica di 45 gradi a destra:
        e.Graphics.RotateTransform(45)

        ' Crea il carattere da utilizzare per la scritta:
        Dim Carattere As New Font("Arial", 16, FontStyle.Bold)
        ' Prendi le misure della scritta:
        Dim MisureScritta As SizeF = e.Graphics.MeasureString("Testo ruotato di
45°", Carattere)

        ' Colora un rettangolo di colore giallo, come sfondo della scritta:
        e.Graphics.FillRectangle(Brushes.Yellow, 30, -10, MisureScritta.Width,
MisureScritta.Height)
        ' Disegna un contorno di colore nero:
        e.Graphics.DrawRectangle(Pens.Black, 30, -10, MisureScritta.Width,
MisureScritta.Height)
        ' Disegna la scritta in alto a sinistra:
        e.Graphics.DrawString("Testo ruotato di 45°", Carattere, Brushes.Red, 30,
-10)

    End Sub

End Class
```

Da notare, nel codice, la struttura **MeasureString**

```
        Dim MisureScritta As SizeF = e.Graphics.MeasureString("Testo ruotato di
45°", Carattere)
```

che è utilizzata per prendere le misure del testo da scrivere. **MeasureString** è una struttura di tipo **SizeF**, che memorizza con precisione le dimensioni (larghezza e altezza) dello spazio occupato da una scritta.
Nel nostro esempio, i dati ricavati con MeasureString sono memorizzati nella struttura MisureScritta, utilizzata per creare il rettangolo di sfondo della scritta.
Ecco un'immagine del programma in esecuzione:

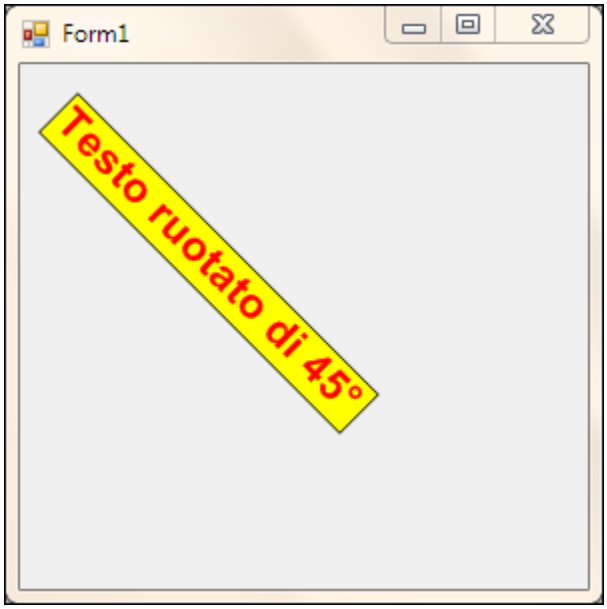


Figura 213: Rotazione di una scritta.

157: Scrivere in verticale.

La scrittura di un testo in verticale, su un lato del form, si ottiene creando un nuovo formato per la stringa (**StringFormat**), e impostandone la proprietà **StringFormatFlags**. E' possibile dare alla proprietà StringFormatFlags le impostazioni date nell'esempio seguente:

- StringFormatFlags.DirectionVertical: imposta il testo in verticale;
- StringFormatFlags.NoWrap: esclude il ritorno a capo se il testo è troppo lungo;
- StringFormatFlags.DirectionRightToLeft: colloca il testo sulla destra del form.

Ecco un esempio:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Comandi per il miglioramento della qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias

        ' Crea il carattere da utilizzare per la scritta:
        Dim Carattere As New Font("Arial", 25, FontStyle.Bold Or
FontStyle.Italic)

        'Crea il formato da utilizzare per la scritta:
        Dim Formato As New StringFormat
        Formato.FormatFlags = StringFormatFlags.DirectionVertical Or
StringFormatFlags.NoWrap Or StringFormatFlags.DirectionRightToLeft
```

```
        e.Graphics.DrawString("Testo verticale", Carattere, Brushes.Red,
Me.ClientRectangle, Formato)

    End Sub

End Class
```

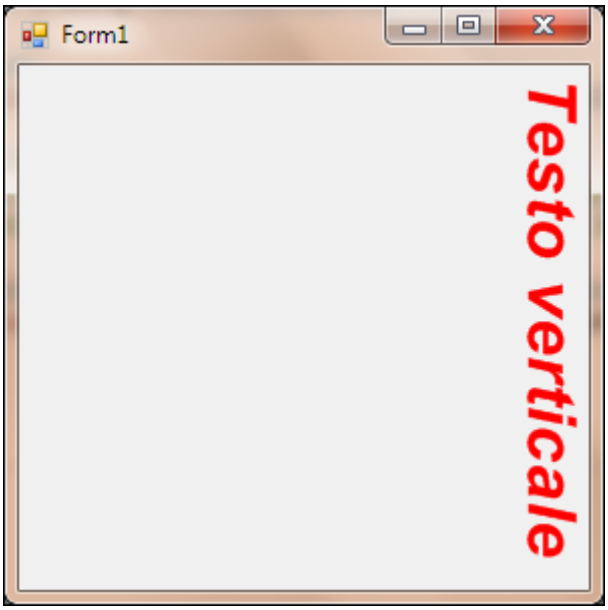


Figura 214: Testo orientato in verticale, sulla destra del form.

158: Colorare una scritta con colori gradienti lineari.

Il codice seguente disegna nel form una scritta colorata con colori gradienti dal blu al rosso, dall'alto al basso[78].
Anche questo programma si basa sulla struttura **MeasureString**, per misurare le dimensioni dello spazio occupato dalla scritta; tali dimensioni sono assegnate a un rettangolo utilizzato per colorare il testo.

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Migliora la qualità grafica:
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias

        ' Crea un nuovo carattere e il testo da scrivere:
        Dim Carattere As New Font("Arial", 16, FontStyle.Bold)
```

[78] Le istruzioni di base per creare colori gradienti lineari si trovano al paragrafo 135, a pag. 589.

```
        Dim Testo As String = "Testo colorato" & vbCrLf & "con colori gradienti,"
& vbCrLf & "dal blu al rosso," & vbCrLf & "da sinistra a destra."

        ' Misura l'area occupata dal testo:
        Dim SpazioTesto As New SizeF()
        SpazioTesto = e.Graphics.MeasureString(Testo, Carattere)

        ' Crea il rettangolo di sfondo della screitta:
        Dim Rettangolo As New RectangleF(0, 0, SpazioTesto.Width,
SpazioTesto.Height)

        ' Crea il pennello con i colori gradienti con inclinazione a 90°,
dall'alto al basso:
        Dim Pennello As New Drawing2D.LinearGradientBrush(Rettangolo, Color.Blue,
Color.Red, 90)

        ' Disegna la scritta
        e.Graphics.DrawString(Testo, Carattere, Pennello, 0, 0)

    End Sub

End Class
```

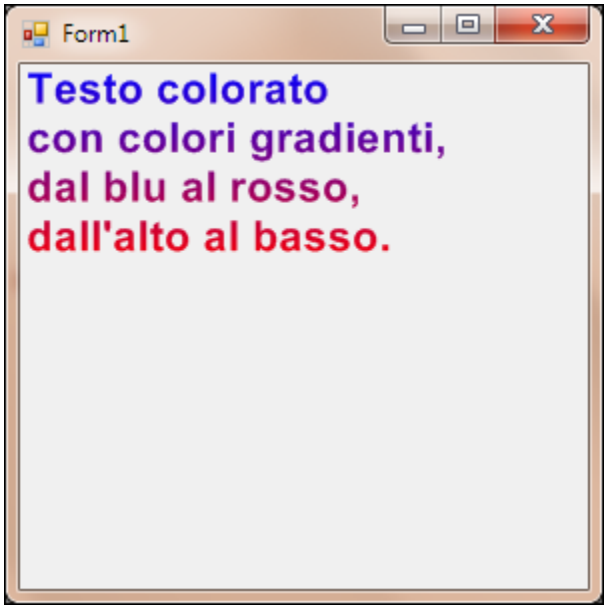


Figura 215: Una scritta colorata con colori gradienti, dall'alto al basso.

159: Colorare una scritta con colori gradienti radiali.

Il codice seguente disegna nel form una scritta colorata con colori gradienti dal giallo (al centro) al blu, rosso e verde (all'esterno).

Il programma fa uso della proprietà **MeasureString** per misurare le dimensioni dello spazio occupato dalla scritta; tali dimensioni sono assegnate a un percorso/oggetto utilizzato per colorare il testo[79].

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Migliora la qualità grafica:
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias

        ' Crea un nuovo carattere e il testo da scrivere:
        Dim Carattere As New Font("Arial", 16, FontStyle.Bold)
        Dim Testo As String = "Testo colorato" & vbCrLf & "con colori gradienti,"
& vbCrLf & "dal centro verso l'esterno," & vbCrLf & "con il giallo al centro," &
vbCrLf & "e il rosso, il verde e il blu" & vbCrLf & "all'esterno."

        ' Misura l'area occupata dal testo:
        Dim SpazioTesto As New SizeF()
        SpazioTesto = e.Graphics.MeasureString(Testo, Carattere)

        ' Crea il rettangolo di sfondo della scritta come percorso/oggetto per i
colori gradienti radiali:
        Dim AreaDaColorare As New Drawing2D.GraphicsPath
        AreaDaColorare.AddRectangle(New Rectangle(0, 0, SpazioTesto.Width,
SpazioTesto.Height))

        ' Crea il pennello con i colori gradienti:
        Dim Pennello As New Drawing2D.PathGradientBrush(AreaDaColorare)
        ' determina il colore al centro del pennello:
        Pennello.CenterColor = Color.Yellow

        ' determina la gamma di colori all'esterno del pennello:
        Dim Gammacolori As Color() = {Color.Red, Color.Blue, Color.Green}
        Pennello.SurroundColors = Gammacolori

        ' Disegna la scritta
        e.Graphics.DrawString(Testo, Carattere, Pennello, 0, 0)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

[79] Le istruzioni di base per creare colori gradienti in direzione radiale si trovano al paragrafo 140, a pag. 607.

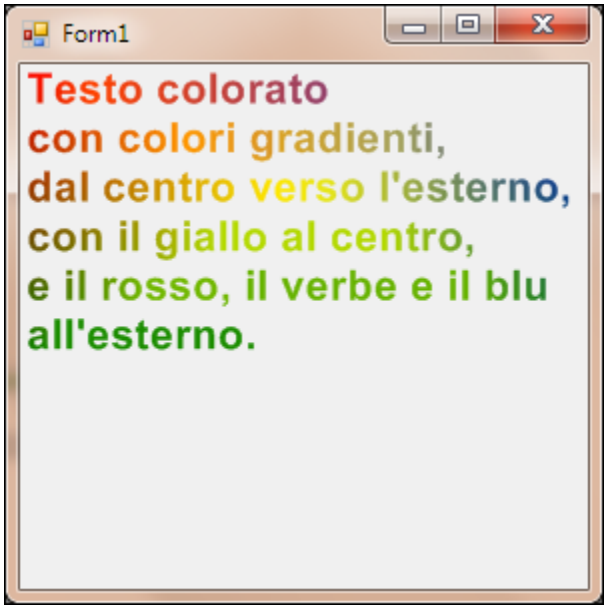


Figura 216: Una scritta colorata con colori gradienti radiali.

160: Tabulazioni e tabelle.

Il comando **StringFormat.SetTabStops** imposta una serie di tabulazioni per la scrittura di un testo in colonne e in tabelle.
La sintassi del comando è questa:

```
Formato.SetTabStops(0, Tabulazioni)
```

I parametri tra parentesi indicano, rispettivamente, la distanza che il testo deve mantenere dall'inizio della sua colonna e i punti d'inizio delle varie colonne (tabulazioni).
Il comando richiede dunque che, prima del suo uso, venga impostata una matrice con i punti d'inizio delle colonne (tabulazioni).

Ecco un esempio di uso del comando **StringFormat.SetTabStops** per la formattazione di un testo in sette colonne.
Il punto d'inizio della prima colonna è 0, i punti d'inizio delle sei colonne successive sono contenuti in una matrice (ogni colonna inizia a 30 pixel di distanza dalla colonna precedente):

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
```

```
        ' Testo da scrivere in sette colonne (il comando vbTab forza la parola
successiva a collocarsi nella colonna successiva):
        Dim Testo As String = "Lunedì" & vbTab & "Martedì" & vbTab & "Mercoledì"
& vbTab & "Giovedì" & vbTab & "Venerdì" & vbTab & "Sabato" & vbTab & "Domenica"

        ' Crea una matrice con i punti d'inizio delle colonne:
        Dim Tabulazioni As Single() = {30, 30, 30, 30, 30, 30}

        ' Formattazione del testo secondo le tabulazioni già impostate:
        Dim Formato As New StringFormat
        Formato.SetTabStops(0, Tabulazioni)

        ' Definisce l'area rettangolare in cui è visualizzato il testo
        ' e scrive il testo in blu con il formato su 7 colonne:
        Dim AreaTesto As New Rectangle(20, 20, 500, 100)
        e.Graphics.DrawString(Testo, Me.Font, Brushes.Blue, AreaTesto, Formato)

    End Sub

End Class
```

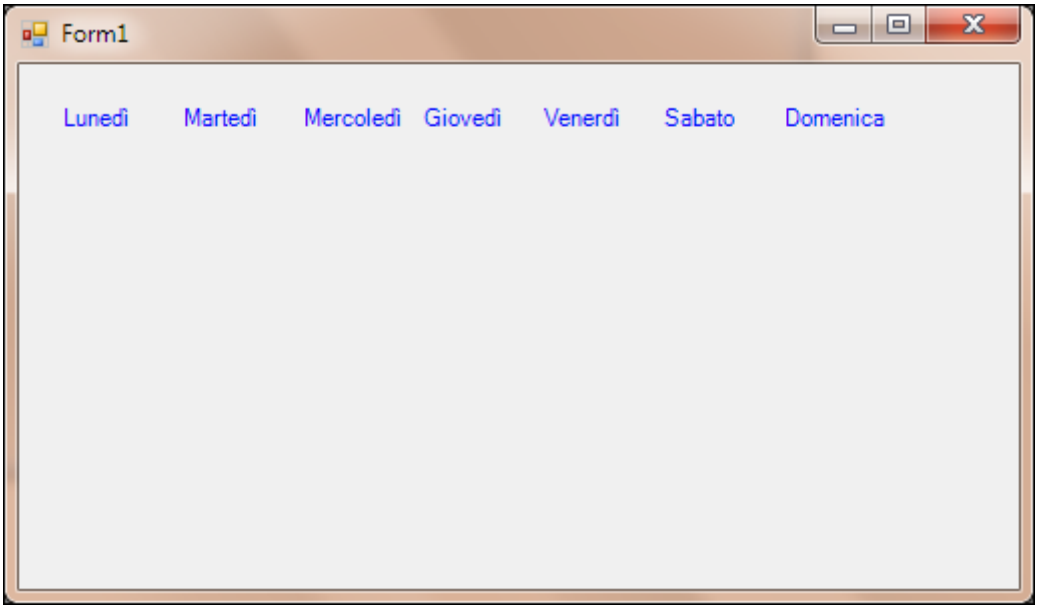


Figura 217: Il comando StringFormat.SetTabStops.

Nei prossimi due esercizi vedremo l'uso del comando StringFormat.SetTabStops per la creazione di due tabelle: la prima con quattro colonne, la seconda con due colonne.

Esercizio 95: Il comando StringFormat.SetTabStops.

Questo programma scrive un testo sul form con il comando DrawString.
Il testo è formattato in quattro colonne, con il comando StringFormat.SetTabStops.

Apriamo un nuovo progetto; copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load
        ' Impostazioni iniziali del form:
        Me.Text = "Orario partenze e arrivi"
        Me.Width = 400
        Me.Height = 210
    End Sub

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Scelta del carattere:
        Dim Carattere As New Font("Arial", 14)

        ' Il testo da scrivere è composto dalle 6 righe che seguono.
        ' Il comando vbTab sposta la parola che lo segue alla tabulazione
successiva,
        ' cioè colloca la parola che lo segue nella colonna successiva.
        ' Il comando vbCrLf termina una riga e manda il testo a capo.
        Dim Testo As String = "Partenze" & vbTab & " " & vbTab & "Arrivi" &
vbCrLf & _
 "10.30" & vbTab & "Catania" & vbTab & "10.25" & vbTab & "Palermo" & vbCrLf & _
 "10.55" & vbTab & "Roma" & vbTab & "10.40" & vbTab & "Bari" & vbCrLf & _
 "11.10" & vbTab & "Napoli" & vbTab & "11.20" & vbTab & "Pisa" & vbCrLf & _
 "11.50" & vbTab & "Cagliari" & vbTab & "12.00" & vbTab & "Cagliari" & vbCrLf & _
 "12.10" & vbTab & "Marsiglia" & vbTab & "12.30" & vbTab & "Roma"

        ' Il testo è scritto in 4 colonne.
        ' La prima colonna inizia alla posizione 0.
        ' I punti d'inizio della II, III e IV colonna sono fissati da queste
tabulazioni:
        Dim Tabulazioni As Single() = {80, 100, 80}

        ' Formattazione del testo su quattro colonne, secondo le tabulazioni già
impostate:
        Dim Formato As New StringFormat
        Formato.SetTabStops(0, Tabulazioni)

        ' Definisce l'area rettangolare in cui è visualizzato il testo
        ' e scrive il testo in blu con la formattazione già impostata a 4
colonne:
        Dim AreaTesto As New Rectangle(20, 20, 400, 150)
        e.Graphics.DrawString(Testo, Carattere, Brushes.Blue, AreaTesto, Formato)

        ' disegna due rettangoli per dividere il testo in due tabelle:
        Dim Tabella1 As New Rectangle(10, 10, 180, 150)
        Dim Tabella2 As New Rectangle(195, 10, 180, 150)
        e.Graphics.DrawRectangle(Pens.Red, Tabella1)
        e.Graphics.DrawRectangle(Pens.Green, Tabella2)

    End Sub

End Class
```

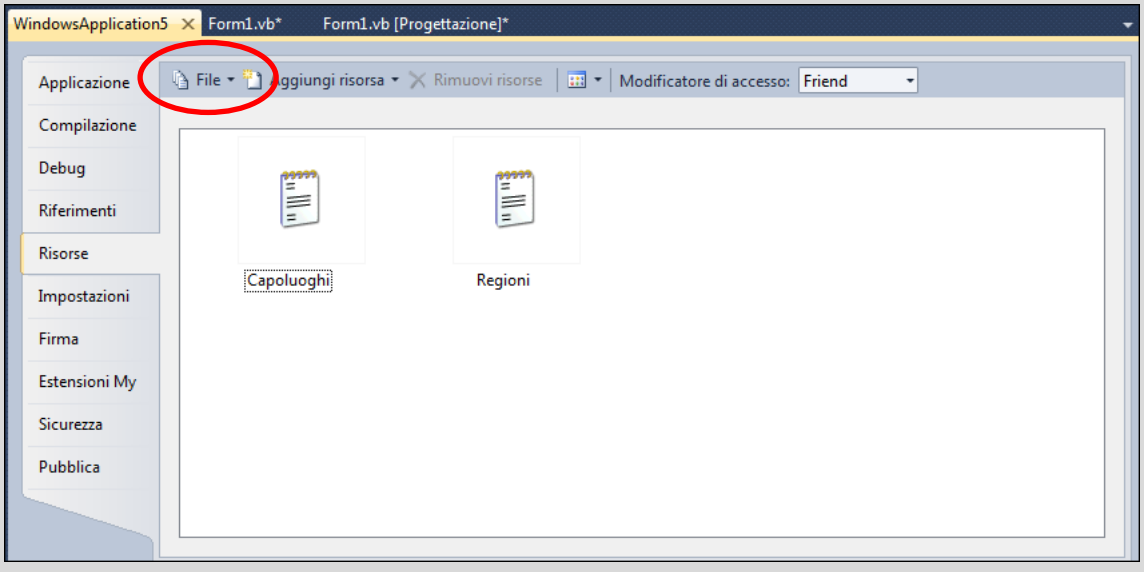
Ecco un'immagine del programma in esecuzione:

Esercizio 96: Regioni italiane e città capoluogo.

Questo programma crea una tabella con un testo su due colonne: nella prima colonna si leggono i nomi delle 20 regioni italiane, nella seconda colonna i nomi delle rispettive città capoluogo di regione.
I nomi delle regioni e delle città sono ricavati da due file di testo inseriti nelle risorse del programma.
Apriamo un nuovo progetto.
Inseriamo nelle risorse del programma i due file di testo

- Regioni.txt e
- Capoluoghi.txt

che si trovano nella cartella **Documenti / A scuola con VB 2010 / Testi**:



Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Dichiara due matrici di variabili per la memorizzazione dei nomi delle
regioni e delle città:
    Dim Regione(19) As String
    Dim Capoluogo(19) As String
```

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Me.Width = 440
        Me.Height = 460
        Me.Text = "Città capoluogo delle regioni italiane"

        Dim Contatore As Integer

        ' apre il file Regioni.txt, ne legge i dati riga per riga, e li immette
nella matrice Regione:
        Dim LeggiRegioni As New System.IO.StringReader(My.Resources.Regioni)
        For Contatore = 0 To 19
            Regione(Contatore) = LeggiRegioni.ReadLine
        Next

        ' apre il file Capoluoghi.txt, ne legge i dati riga per riga e li immette
nella matrice Capoluogo:
        Dim LeggiCapoluoghi As New
System.IO.StringReader(My.Resources.Capoluoghi)
        For Contatore = 0 To 19
            Capoluogo(Contatore) = LeggiCapoluoghi.ReadLine
        Next

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

        ' Crea un nuovo carattere e ne misura le dimensioni
        ' (il programma deve conoscere l'altezza dei caratteri, per tracciare le
linee orizzontali tra una riga e l'altra)
        Dim Carattere As New Font("Arial", 12)
        Dim DimensioniCaratteri As SizeF = Me.CreateGraphics.MeasureString("Abc",
Carattere)
        Dim AltezzaCaratteri As Single = DimensioniCaratteri.Height - 1.5

        ' Crea l'area rettangolare che conterrà il testo di 20 righe:
        Dim Tabella As New RectangleF(20, 30, 400, AltezzaCaratteri * 20)
        ' Traccia in grigio il contorno di quest'area rettangolare:
        e.Graphics.DrawRectangle(Pens.Gray, 10, 30, Tabella.Width,
Tabella.Height)

        ' Crea un formato con due colonne: il punto d'inizio della prima colonna
è 0, il punto d'inizio della seconda colonna è 200:
        Dim Formato As New StringFormat
        Dim Tabulazioni() As Single = {200}
        Formato.SetTabStops(0, Tabulazioni)

        ' Traccia una linea verticale per separare le due colonne:
        e.Graphics.DrawLine(Pens.Gray, 200, Tabella.Top, 200, Tabella.Bottom)
```

```
        ' Traccia 20 linee orizzontali per separare le 20 righe:
        For i As Integer = 0 To 19
            e.Graphics.DrawLine(Pens.Gray, Tabella.Left - 10, Tabella.Top + i *
AltezzaCaratteri, Tabella.Right - 10, Tabella.Top + i * AltezzaCaratteri)
        Next i

        ' Componi il testo con nomi delle 20 regioni e i nomi delle 20 città
capoluogo
        ' (il comando vbTab sposta la parola successiva nella colonna successiva,
        ' il comando vbCrLf forza il testo ad andare a capo):
        Dim Testo As String = ""
        For Riga As Integer = 0 To 19
            Testo &= Regione(Riga) & vbTab & Capoluogo(Riga) & vbCrLf
        Next Riga

        ' Scrivi il testo:
        e.Graphics.DrawString(Testo, Carattere, Brushes.Black, Tabella, Formato)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

Capitolo 29: ANIMAZIONI GRAFICHE.

In questo capitolo vedremo alcuni esempi di animazioni grafiche ottenute con spostamenti di linee, figure geometriche, colori, immagini e scritte.
Questi spostamenti si basano su un controllo Timer, inserito nel programma, il cui *tic*, a intervalli regolari, fornisce istruzioni affinché il colore, l'immagine o la scritta si spostino di pochi pixel dalla posizione precedente: questi spostamenti minimi danno all'utente l'illusione di vedere oggetti in movimento.

Lo schema di base per il funzionamento di queste animazioni si riduce a due procedure:

- a ogni *tic* del timer viene cancellato il contenuto del form;
- questa cancellazione genera un evento Paint;
- la procedura dell'evento Paint contiene le indicazioni per l'aggiornamento del disegno.

Ecco il listato di queste due procedure:

```
Public Class Form1

    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        'cancella il contenuto del form / causa l'evento Paint:
        Me.Invalidate()

    End Sub

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' istruzioni per disegnare o ridisegnare gli oggetti

    End Sub

End Class
```

Lo spostamento rapido di elementi all'interno di una superficie grafica può creare uno sfarfallìo piuttosto fastidioso; per ridurlo o eliminarlo occorre impostare la proprietà **DoubleBuffered** del **Form** = **True**. Con questa impostazione, le modifiche grafiche avvengono in un primo momento nella memoria del computer e sono visualizzate solo dopo questo passaggio intermedio. L'operazione è talmente rapida da non essere percepibile, e consente di dare stabilità alla visione delle immagini.

161: Linee e punti in movimento.

I due esercizi che seguono mostrano come creare due semplici animazioni.
Nel primo esercizio, una linea rossa attraversa il form da sinistra a destra, con un percorso a zig zag casuale.
A ogni *tic* del timer, viene aggiunto un segmento alla linea già tratteggiata. In questo caso dunque il contenuto del form non è cancellato e ridisegnato a ogni *tic* del timer (non vi è alcun evento Paint), ma il nuovo segmento semplicemente si aggiunge a quelli già esistenti.

Esercizio 97: Una linea in movimento.

Apriamo un nuovo progetto.
Proprietà del Form1: **DoubleBuffered = True**.
Inseriamo nel progetto un controllo Button1, in basso a destra nel Form, e un controllo **Timer** con queste proprietà:
Enabled = False
Interval = 100
Copiamo e incolliamo nella Finestra del Codice questo listato:

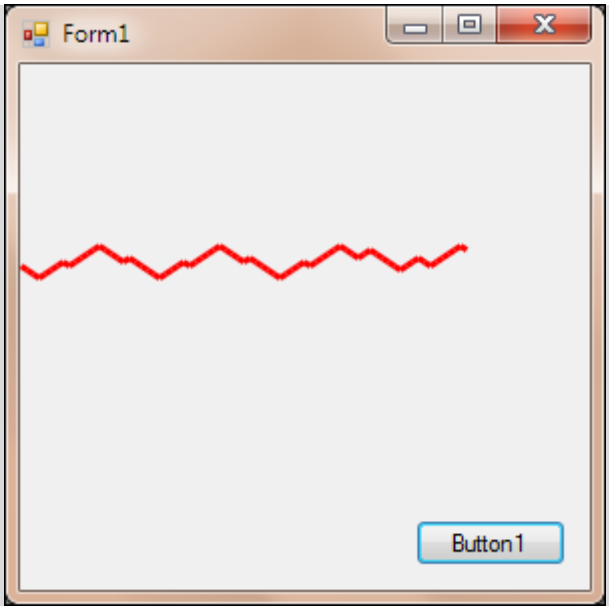
```
Public Class Form1

    ' crea una superficie grafica corrispondente al Form1:
    Dim Superficie As Graphics = Me.CreateGraphics

    ' variabili per memorizzare i punti di partenza e i punti di arrivo delle
linee:
    Dim Sinistra As Integer = 0
    Dim Altezza As Integer = 100
    Dim SinistraPrecedente As Integer = 0
    Dim AltezzaPrecedente As Integer = 100

    ' variabile per sorteggiare la direzione alto/basso
    Dim Direzione As Integer = 1
```

```
    Private Sub Timer1_Tick(sender As Object, e As System.EventArgs) Handles
Timer1.Tick

        ' migliora la qualità grafica:
        Superficie.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' A ogni tick del timer il programma traccia una linea che va dal punto
precedente al nuovo punto.
        ' Il punto precedente è dato dalle variabili SinistraPrecedente,
AltezzaPrecedente.
        ' Il nuovo punto è dato dalle variabili Sinistra, Altezza.


        ' sposta il punto di arrivo della linea di 3 pixel verso destra:
        Sinistra += 3
```

```
        ' avvia il generatore di numeri casuali e sorteggia un numero a caso da 1
a 2:
        Dim Sorteggio As New Random
        Randomize()
        Direzione = Sorteggio.Next(1, 3)

        Select Case Direzione
            Case 1 ' la linea va verso l'alto
                Altezza -= 2
            Case 2 ' la linea va verso il basso
                Altezza += 2
        End Select

        ' crea uno strumento penna per tracciare la linea e traccia la linea:
        Dim Penna As New Pen(Color.Red, 3)
        Superficie.DrawLine(Penna, SinistraPrecedente, AltezzaPrecedente,
Sinistra, Altezza)

        ' memorizza il punto di arrivo come punto di partenza per la prossima
linea:
        SinistraPrecedente = Sinistra
        AltezzaPrecedente = Altezza

        ' se la linea esce dal limite destro del form arresta il timer:
        If Sinistra > Me.ClientRectangle.Width Then Timer1.Enabled = False

    End Sub
```

```
    Private Sub Button1_Click_1(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' il clic sul pulsante attiva o disattiva il Timer:
        Timer1.Enabled = Not (Timer1.Enabled)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Nel prossimo esercizio, con qualche variazione all'esercizio precedente, un punto rosso attraversa il form da sinistra a destra, con un percorso a zig zag casuale.
In questo caso, a ogni *tic* del Timer1 si cancella il contenuto del Form1, e dunque si attiva un evento Paint che ridisegna il contenuto del Form con il punto rosso nella sua nuova posizione.

Esercizio 98: Un punto in movimento.

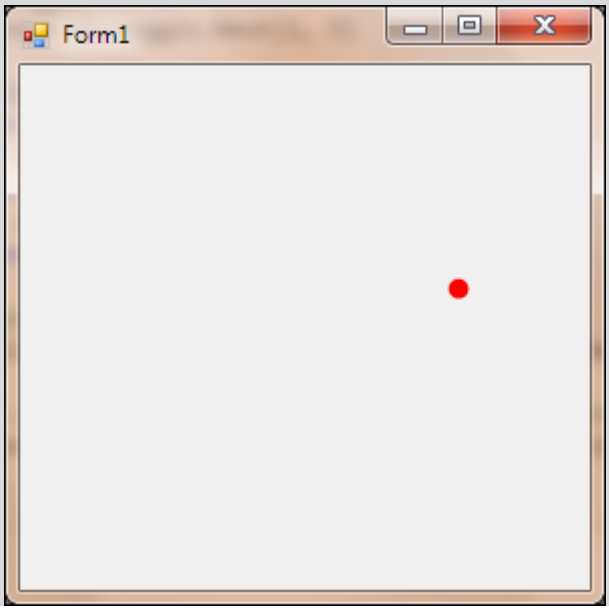
Apriamo un nuovo progetto.
Proprietà del Form: **DoubleBuffered = True**.
Inseriamo nel progetto un controllo **Timer1** con queste proprietà:
Enabled = True
Interval = 100
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' variabili per definire la posizione del punto rosso:
    Dim Sinistra As Integer = 0
    Dim Altezza As Integer = 100

    ' variabile per sorteggiare la direzione alto/basso
    Dim Direzione As Integer = 1
```

```
    Private Sub Timer1_Tick(sender As Object, e As System.EventArgs) Handles
Timer1.Tick

        ' A ogni tic del timer cancella il contenuto del form a attiva l'evento
Paint:
        Me.Invalidate()
```

```
    End Sub

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' sposta la posizione del punto rosso di 3 pixel verso destra:
        Sinistra += 3

        ' avvia il generatore di numeri casuali e sorteggia un numero a caso da 1
a 2:
        Dim Sorteggio As New Random
        Randomize()
        Direzione = Sorteggio.Next(1, 3)

        Select Case Direzione
            Case 1 ' il punto rosso va verso l'alto
                Altezza -= 2
            Case 2 ' il punto rosso va verso il basso
                Altezza += 2
        End Select

        ' disegna il punto rosso nella nuova posizione:
        e.Graphics.FillEllipse(Brushes.Red, Sinistra, Altezza, 10, 10)

        ' se il punto rosso esce dal limite destro del form torna a comparire sul
lato sinistro:
        If Sinistra > Me.ClientRectangle.Width Then Sinistra = 0

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



162: Figure geometriche in movimento.

Nel prossimo esercizio vedremo un *pallone* che a ogni tic del timer si *gonfia* di 3 pixel, sino a toccare il bordo del form.

Esercizio 99: Un pallone che si gonfia.

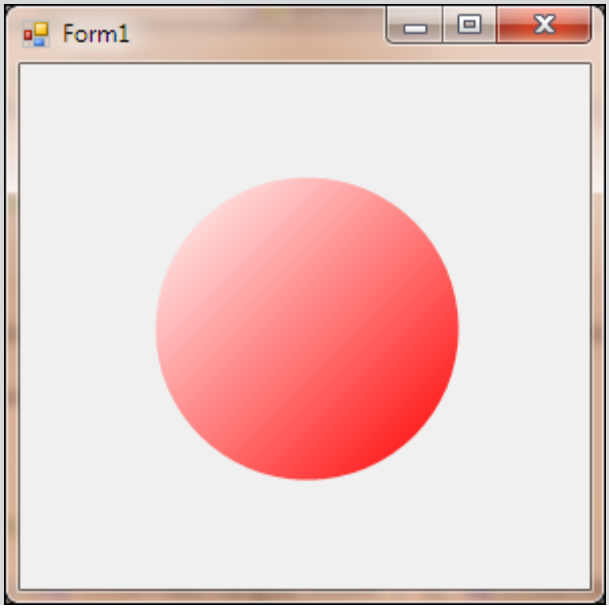
Apriamo un nuovo progetto.
Proprietà del Form: **DoubleBuffered** = True.
Inseriamo nel progetto un controllo Timer con queste proprietà:
Enabled = True
Interval = 100
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea un rettangolo di 1 pixel per lato, al centro del form:
    Dim Rettangolo As New Rectangle(Me.ClientRectangle.width / 2,
Me.ClientRectangle.height / 2, 1, 1)
```

```
    Private Sub Timer1_Tick(sender As Object, e As System.EventArgs) Handles
Timer1.Tick

        ' cancella il contenuto del form e attiva l'evento Paint
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' "gonfia" il rettangolo di 3 pixel per lato
        Rettangolo.Inflate(3, 3)

        ' se il rettangolo tocca il bordo superiore del form riporta il
rettangolo al centro del form e alle dimensioni originali di 1 pixel per lato:
        If Rettangolo.Top < 0 Then Rettangolo = New
Rectangle(Me.ClientRectangle.Width / 2, Me.ClientRectangle.Height / 2, 1, 1)

        ' crea un pennello con colori gradienti per colorare il pallone
        Dim Pennello As New Drawing2D.LinearGradientBrush(Rettangolo,
Color.White, Color.Red, 45)

        ' disegna il pallone:
        e.Graphics.FillEllipse(Pennello, Rettangolo)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Nel prossimo esercizio vedremo un esempio di creazione di un orologio analogico, con una tecnica che richiede qualche spiegazione preventiva.
Le lancette dell'orologio che indicano le ore, i minuti e i secondi sono disegnate con il comando **DrawPie**, che traccia un diagramma circolare o grafico a fetta di torta.
Ricordiamo che il comando DrawPie richiede questi parametri:

```
e.Graphics.DrawPie(Penna, 0, 0, 200, 200, 0, 360)
```

I parametri tra parentesi, nel loro ordine, contengono questi dati:

0, 0	Posizione sinistra/alto del rettangolo in cui è iscritto il diagramma circolare.
200, 200	Posizione destra/basso del rettangolo in cui è iscritto il diagramma circolare.
0, 360	Grado iniziale dell'arco (il grado 0 corrisponde alle ore 3 , in un orologio analogico) e ampiezza dell'arco espressa in gradi (una ampiezza pari a 360 gradi prende tutto il diagramma circolare).

Tabella 34: Parametri del comando DrawPie.

Un comando DrawPie con i parametri che abbiamo esaminato genera dunque questa figura:

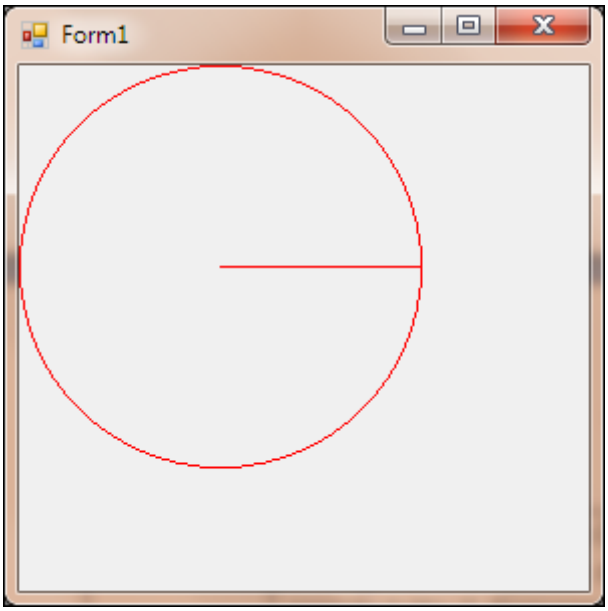


Figura 218: Disegno di un diagramma circolare.

Immaginiamo che il raggio del diagramma sia una lancetta dell'orologio: aggiungendo un timer, è sufficiente muovere l'angolo d'inizio del diagramma (da 0 a 360) per avere la lancetta in movimento.
Proviamo questo codice, dopo avere inserito in un progetto un controllo **Timer** con la proprietà **Enabled = True**.

```
Public Class Form1

    Dim Secondi As Integer = 1
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        ' disegna il diagramma circolare, muovendo il grado iniziale, cioè il
raggio
        e.Graphics.DrawPie(Pens.Red, 0, 0, 200, 200, Secondi, 360)

    End Sub
```

```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ' imposta il valore del timer a 1000 millisecondi, cioè un tic ogni
secondo:
        Timer1.Interval = 1000
        Secondi += 1
        If Secondi > 360 Then Secondi = 0
        Me.Invalidate()

    End Sub
```

```
End Class
```

Ora, mandando in esecuzione il programma, vediamo una *lancetta* che si muove all'interno di una circonferenza.
Per cancellare la circonferenza e lasciare visibile solo la lancetta, è sufficiente disegnare un'altra circonferenza nella stessa posizione e delle stesse dimensioni, ma con il colore di sfondo del form.
In questo modo la circonferenza di colore rosso, nella quale si muove la *lancetta*, è *nascosta* da un'altra circonferenza identica, non visibile perché ha lo stesso colore del form:

```
        Dim Penna As New Pen(Me.BackColor, 3)
        e.Graphics.DrawEllipse(Penna, 0, 0, 200, 200)
```

Ecco il codice completo per visualizzare la lancetta dei secondi di un orologio analogico:

```
Public Class Form1
    Dim Secondi As Integer = 1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
        ' disegna il diagramma circolare, muovendo il grado iniziale, cioè il
raggio
        e.Graphics.DrawPie(Pens.Red, 0, 0, 200, 200, Secondi, 360)

        Dim Penna As New Pen(Me.BackColor, 3)
        e.Graphics.DrawEllipse(Penna, 0, 0, 200, 200)

    End Sub
```

```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ' imposta il valore del timer a 1000 millisecondi, cioè un tic ogni
secondo:
        Timer1.Interval = 1000
        Secondi += 1
        If Secondi > 360 Then Secondi = 0
        Me.Invalidate()

    End Sub

End Class
```

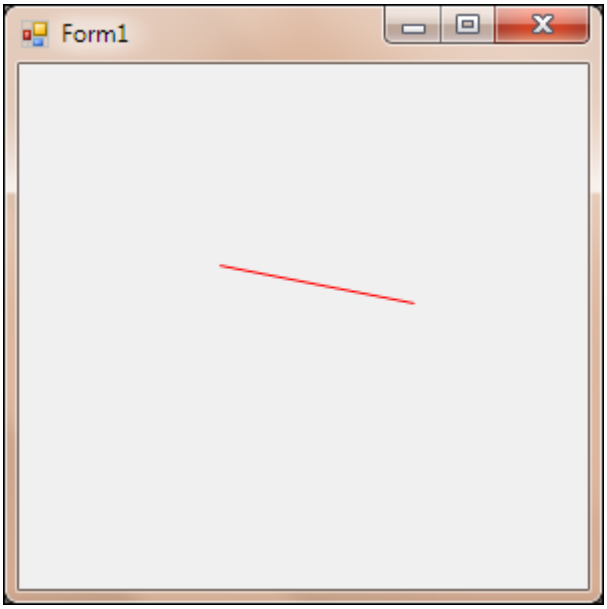
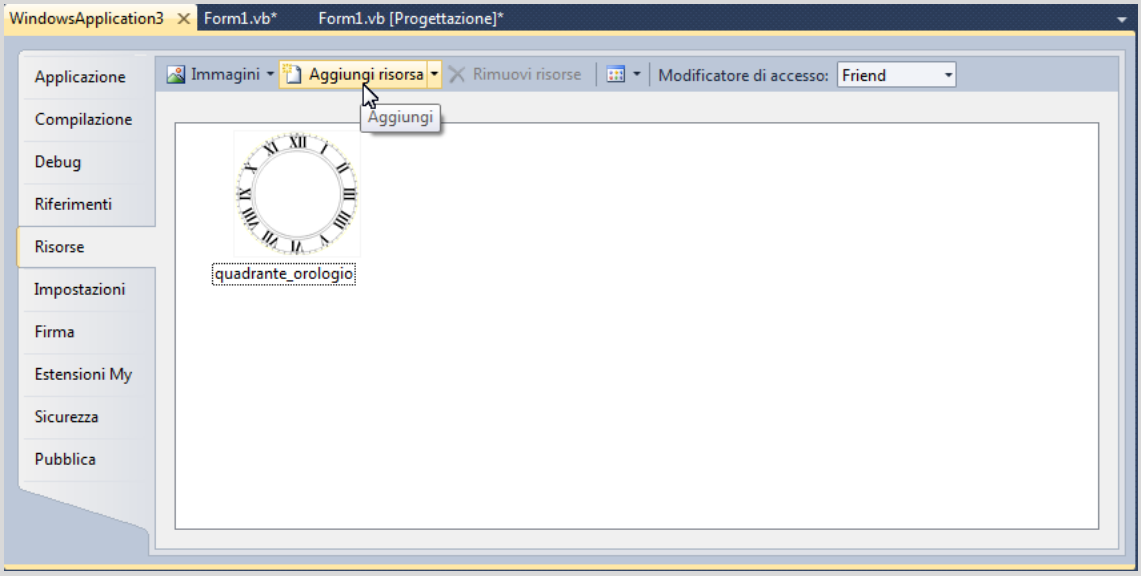


Figura 219: La lancetta dei secondi di un orologio analogico .

Esercizio 100: Costruiamo un orologio analogico.

Apriamo un nuovo progetto.
Inseriamo nel progetto un controllo Timer.
Inseriamo nelle risorse del programma l'immagine **Quadrante_orologio.gif** che si trova nella cartella **Documenti \ A scuola con VB \ Immagini**:



Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' crea un'immagine virtuale con l'immagine del quadrante dell'orologio: sarà
lo sfondo del form
    Dim ImmagineOrologio As Bitmap = My.Resources.Quadrante_orologio
```

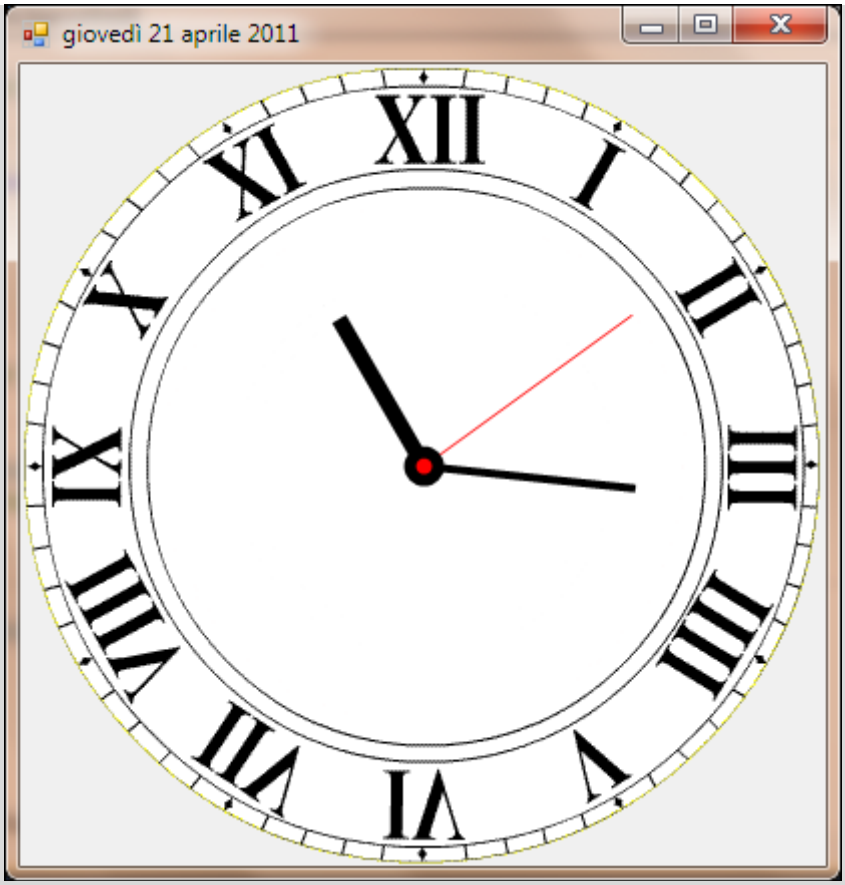
```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        ' imposta le proprietà del form e del timer:
        Me.Width = 418
        Me.Height = 438
        Me.DoubleBuffered = True
        Timer1.Enabled = True
        Timer1.Interval = 1000 ' questo intervallo corrisponde a un tic del timer
a ogni secondo

    End Sub
```

```
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick

        ' a ogni tic del timer cancella il contenuto del form e causa un evento
Paint del form:
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' individua il centro dello schermo e disegna l'immagine di sfondo con il
quadrante dell'orologio al centro del form:
        Dim PuntoCentrale As New Point(Me.ClientRectangle.Width / 2,
Me.ClientRectangle.Height / 2)
        e.Graphics.DrawImage(ImmagineOrologio, PuntoCentrale.X - 200,
PuntoCentrale.Y - 200)

        ' scrive la data nella barra del titolo del form:
        Me.Text = Format(Date.Today, "D")

        ' estrae il numero dei secondi dall'orario corrente,
        ' lo riporta su una scala di 360 gradi (60 secondi * 6 = 360) e
        ' imposta il grado di partenza del diagramma circolare dei secondi.
        ' Nota: la correzione + 270 è dovuta al fatto che l'ora 0 in un orologio
analogico
        ' corrisponde al grado 270 in un diagramma circolare di VB.
        Dim Secondi As Integer = (Date.Now.Second * 6) + 270

        ' estrae il numero dei minuti dall'orario corrente
        ' lo riporta su una scala di 360 gradi (60 minuti * 6 = 360) e
        ' imposta il grado di partenza del diagramma circolare dei minuti:
        Dim Minuti As Integer = (Date.Now.Minute * 6) + 270

        ' estrae il numero delle ore dall'orario corrente
        ' lo riporta su una scala di 360 gradi (12 ore * 30 = 360) e
```

```
        ' imposta il grado di partenza del diagramma circolare delle ore:
        Dim Ore As Integer = (Date.Now.Hour * 30) + 270

        Dim ColoreSfondo As Color = Color.White 'Me.BackColor

        ' disegna il diagramma circolare piccolo per le ore e cancella la
circonferenza, lasciando visibile solo la lancetta:
        Dim PennaOre As New Pen(Color.Black, 8)
        e.Graphics.DrawPie(PennaOre, PuntoCentrale.X - 90, PuntoCentrale.Y - 90,
180, 180, Ore, 360)
        Dim CancellaOre As New Pen(ColoreSfondo, 10)
        e.Graphics.DrawEllipse(CancellaOre, PuntoCentrale.X - 90, PuntoCentrale.Y
- 90, 180, 180)

        ' disegna il diagramma circolare medio per i minuti e cancella la
circonferenza, lasciando visibile solo la lancetta:
        Dim PennaMinuti As New Pen(Color.Black, 4)
        e.Graphics.DrawPie(PennaMinuti, PuntoCentrale.X - 110, PuntoCentrale.Y -
110, 220, 220, Minuti, 360)
        Dim CancellaMinuti As New Pen(ColoreSfondo, 8)
        e.Graphics.DrawEllipse(CancellaMinuti, PuntoCentrale.X - 110,
PuntoCentrale.Y - 110, 220, 220)

        ' disegna il diagramma circolare grande per i secondi e cancella la
circonferenza, lasciando visibile solo la lancetta:
        Dim PennaSecondi As New Pen(Color.Red, 1)
        e.Graphics.DrawPie(PennaSecondi, PuntoCentrale.X - 130, PuntoCentrale.Y -
130, 260, 260, Secondi, 360)
        Dim CancellaSecondi As New Pen(ColoreSfondo, 3)
        e.Graphics.DrawEllipse(CancellaSecondi, PuntoCentrale.X - 130,
PuntoCentrale.Y - 130, 260, 260)

        ' disegna un cerchietto centrale nero:
        e.Graphics.FillEllipse(Brushes.Black, PuntoCentrale.X - 10,
PuntoCentrale.Y - 10, 20, 20)
        ' disegna un cerchietto centrale rosso:
        e.Graphics.FillEllipse(Brushes.Red, PuntoCentrale.X - 4, PuntoCentrale.Y
- 4, 8, 8)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Notiamo che, allargando o restringendo il form, l'orologio resta in posizione centrale. Ciò è dovuto al fatto che il ridimensionamento causa un evento Paint del form, come il *tic* del timer, e dunque attiva la procedura Form1_Paint che ricolloca l'immagine del quadrante al centro del form.

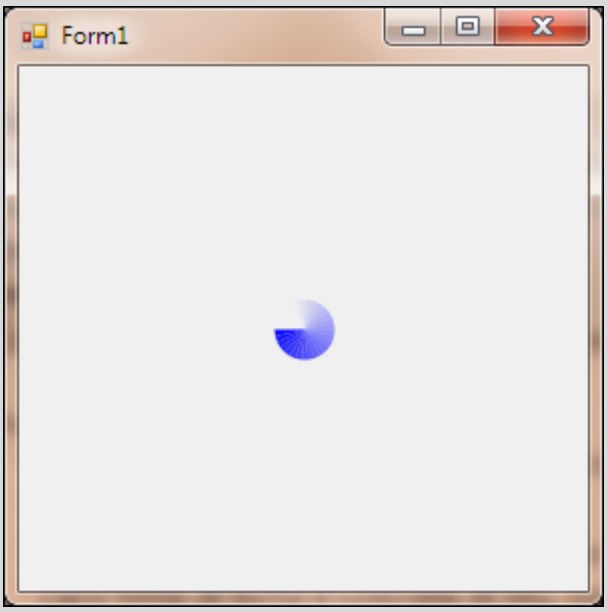
163: Colori in movimento.

Nei prossimi esercizi vedremo alcuni effetti grafici di animazione ottenuti con la gestione dei colori o della loro trasparenza.

Esercizio 101: Un cerchio rotante.

Creiamo un cerchio rotante che può essere utilizzato in alcuni programmi per gestire i momenti di attesa o per indicare all'utente che il programma sta elaborando dei dati ed è temporaneamente bloccato.

Il programma disegna al centro del form un piccolo cerchio colorato, con colore gradiente dal blu al nero, che ruota attorno al suo centro.
Per comprendere il listato è necessario conoscere il comando FillPie, con i suoi parametri[80].
Vediamo subito un'immagine del programma in esecuzione:



Per capire il funzionamento del programma bisogna immaginare il cerchio, al centro del form, diviso in 24 spicchi o fette di torta.
Ogni fetta di torta è di 15 gradi, per cui i 24 spicchi coprono tutto il cerchio (24 spicchi * 15 gradi = 360 gradi).
Ogni spicchio è colorato di blu, ma il livello di trasparenza del colore aumenta di spicchio in spicchio.
Il livello di trasparenza massimo è 255.
In questo programma il primo spicchio ha il livello di trasparenza = 10, poi questo livello aumenta di 10 in 10, sino ad arrivare a 250 con l'ultimo spicchio. Il colore blu dei primi spicchi è dunque ben visibile, mentre il colore blu degli ultimi spicchi si confonde sempre più con il colore nero dello sfondo del form.
L'effetto di animazione è dato da un componente Timer. A ogni *tic* del timer, il form è cancellato e il cerchio è ridisegnato spostando in avanti di una posizione il primo spicchio con livello massimo di colore blu:

- Al primo tic del timer, dunque, lo spicchio n. 1 ha il livello massimo di colore blu e lo spicchio n. 24 è trasparente.
- Al secondo tic del timer, lo spicchio n. 2 ha il livello massimo di colore blu e lo spicchio n. 1 è trasparente.
- Al terzo tic del timer, lo spicchio n. 3 ha il livello massimo di colore blu e lo spicchio n. 3 è trasparente, e così via.

[80] Capitolo 25: LA CLASSE BRUSH. 589.

Apriamo un nuovo progetto.
Proprietà del Form1: **DoubleBuffered = True**
Inseriamo nel progetto un controllo Timer con queste proprietà:

- **Enabled = True**
- **Interval = 10**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
    ' Il programma disegna in successione 24 fette di torta.
    ' Ogni fetta di torta è di 15 gradi.
    ' Le 24 fette di torta riempiono il cerchio (24 * 15 = 360 gradi)
    ' Ogni fetta di torta è colorata di blu, con il livello di trasparenza in
aumento dalla prima all'ultima fetta.

    ' Variabile per memorizzare l'angolo di partenza della prima fetta di torta
con il colore blu pieno:
    Dim AngoloDiPartenza As Integer
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' Inizia il ciclo per disegnare le 24 fette di torta:
        For Contatore As Integer = 1 To 24

            ' Crea il pennello di colore blu per colorare le 24 fette di torta
            Dim Pennello As New SolidBrush(Color.Blue)

            Dim Trasparenza As Integer
            ' Modifica il livello di trasparenza del colore blu a ogni passaggio:
            ' la trasparenza del colore blu va da 10 a 240, aumentando di 10
punti a ogni passaggio:
            Trasparenza = Contatore * 10
            Pennello.Color = Color.FromArgb(Trasparenza, Color.Blue)

            ' Posiziona il cerchio al centro del form, anche se l'utente
ingrandisce o riduce il form:
            Dim Sinistra As Integer = Me.ClientRectangle.Width / 2 - 15
            Dim Altezza As Integer = Me.ClientRectangle.Height / 2 - 15

            ' Disegna una fetta di torta di 15 gradi, partendo dalla posizione
finale della fetta precedente:
            e.Graphics.FillPie(Pennello, Sinistra, Altezza, 30, 30,
AngoloDiPartenza + Contatore * 15, 15)

        Next

    End Sub
```

```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ' Aumenta di 15 gradi l'angolo di partenza della prima fetta di torta:
        AngoloDiPartenza = AngoloDiPartenza + 15
        If AngoloDiPartenza > 360 Then AngoloDiPartenza = 0
```

```
        ' Ripulisci il form e attiva l'evento Paint del form che ridisegna le 24
fette di torta:
        Me.Invalidate()

    End Sub

End Class
```

Nel prossimo esercizio vedremo un effetto di animazione ottenuto muovendo i colori di sfondo del form.

Esercizio 102: Colori gradienti in movimento.

Rivediamo il programma illustrato al paragrafo140: Colori gradienti in direzione radiale. 626.
Questo programma colora lo sfondo del form con cinque colori gradienti in direzione radiale: al centro il colore del form è giallo, ai bordi nero, verde, rosso e blu.
L'effetto di movimento è dato dallo spostamento del punto centrale dei colori gradienti: a ogni tic del controllo Timer, questo punto si porta su una nuova posizione, ruotando in senso circolare, dando così l'illusione di movimento.
Apriamo un nuovo progetto e inseriamo nel form un controllo Timer.
Proprietà del **Form1**:

- **DoubleBuffered = True**

Proprietà del **Timer**:

- **Enabled = True**
- **Interval = 200**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
    ' Crea due variabili per memorizzare la posizione del colore centrale (il
giallo):
    Dim Sinistra As Integer = 100
    Dim Altezza As Integer = 200

    ' Crea una variabile per memorizzare i tic del Timer
    Dim Contatore As Integer = 0
```

```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        Contatore += 1
        If Contatore = 9 Then Contatore = 1

        ' A seconda del numero dei tic del timer, assegna al colore centrale una
nuova posizione:
        Select Case Contatore
            Case 1
                Sinistra = 50
                Altezza = 150
```

```
            Case 2
                Sinistra = 100
                Altezza = 100

            Case 3
                Sinistra = 150
                Altezza = 50

            Case 4
                Sinistra = 200
                Altezza = 100

            Case 5
                Sinistra = 250
                Altezza = 150

            Case 6
                Sinistra = 200
                Altezza = 200

            Case 7
                Sinistra = 150
                Altezza = 250

            Case 8
                Sinistra = 100
                Altezza = 200

        End Select

        ' fa scattare l'evento Form1_Paint che ripulisce e ridisegna il form:
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        'Crea un rettangolo corrispondente alle dimensioni del form:
        Dim Rettangolo As New Rectangle(0, 0, Me.ClientRectangle.Width,
Me.ClientRectangle.Height)

        'Crea un nuovo oggetto di tipo 'percorso grafico' e vi aggiunge il
rettangolo corrispondente al form:
        Dim AreaDaColorare As New Drawing2D.GraphicsPath()
        AreaDaColorare.AddRectangle(Rettangolo)

        ' Assegna a questo oggetto/percorso un pennello con colori gradienti dal
centro all'esterno:
        Dim Pennello As New Drawing2D.PathGradientBrush(AreaDaColorare)
        ' determina il colore al centro del pennello:
        Pennello.CenterColor = Color.Yellow
        ' determina la posizione del colore centrale:
        Pennello.CenterPoint = New Point(Sinistra, Altezza)
        ' determina i colori all'esterno del pennello:
        Dim Gammacolori As Color() = {Color.Red, Color.Green, Color.Blue,
Color.Black}
        Pennello.SurroundColors = Gammacolori

        ' colora l'oggetto/percorso
```

```
        e.Graphics.FillPath(Pennello, AreaDaColorare)

    End Sub

End Class
```

164: Immagini in movimento.

Il modo più semplice per spostare un'immagine sul form è inserire l'immagine in un controllo PictureBox e quindi modificare la posizione, cioè la proprietà **Location**, del PictureBox.
Nel programma di esempio seguente, la proprietà Sinistra imposta la distanza fra l'angolo superiore sinistro del controllo e il lato sinistro del form, mentre la proprietà Altezza imposta la distanza fra l'angolo superiore sinistro del controllo e la parte superiore del form, per cui:

- incrementando il valore della proprietà Sinistra il controllo si muove verso destra,
- incrementando il valore della proprietà Altezza il controllo si muove verso il basso.
- A ogni *clic* sul pulsante Button1, il controllo PictureBox1 si sposta di 12 pixel a sinistra e 12 pixel in basso.

Per provare il programma è necessario inserire nel form un controllo **PictureBox** e un pulsante **Button**.

```
Public Class Form1

    Dim Sinistra As Integer = 12
    Dim Altezza As Integer = 12

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click
        Sinistra += 12
        Altezza += 12
        PictureBox1.Location = New Point(Sinistra, Altezza)
    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

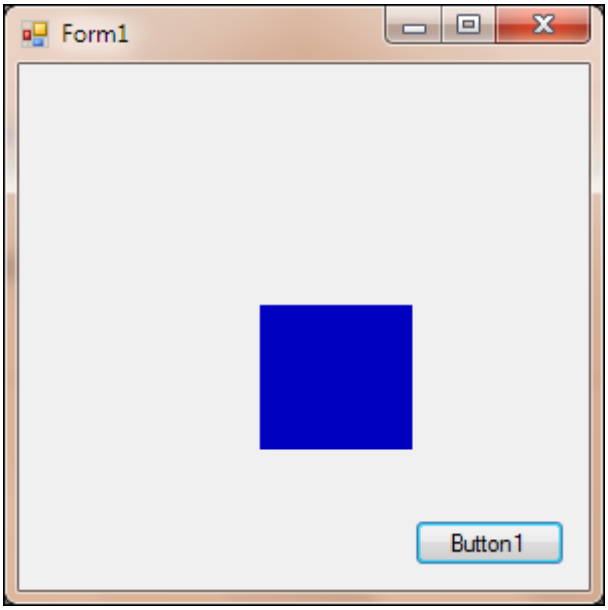


Figura 220: Spostamento di un controllo PictureBox sul form.

Questo spostamento si ottiene in modo molto semplice, ma questa tecnica non si presta a essere utilizzata in spostamenti di immagini più complessi, in rapida sequenza, con i quali si voglia rendere l'illusione di un oggetto in movimento graduale continuo.
Lo spostamento rapido e continuo di un PictureBox sul form, infatti, avviene sempre per scatti successivi che non danno l'idea di un movimento continuo.
Se si vuole ottenere questo effetto è preferibile operare con superfici grafiche virtuali, che visualizzano il loro contenuto solo al termine della trasformazione o dello spostamento del loro contenuto.
E' quanto faremo nel prossimo esercizio, dove si vedono due palloni muoversi sul form: il primo pallone si nuove in tutte le direzioni e rimbalza contro i bordi del form; il secondo pallone si muove avanti e indietro, ruotando la sua immagine durante il movimento.

Esercizio 103: Due palloni in movimento.

Apriamo un nuovo progetto.
Proprietà del **Form1**:

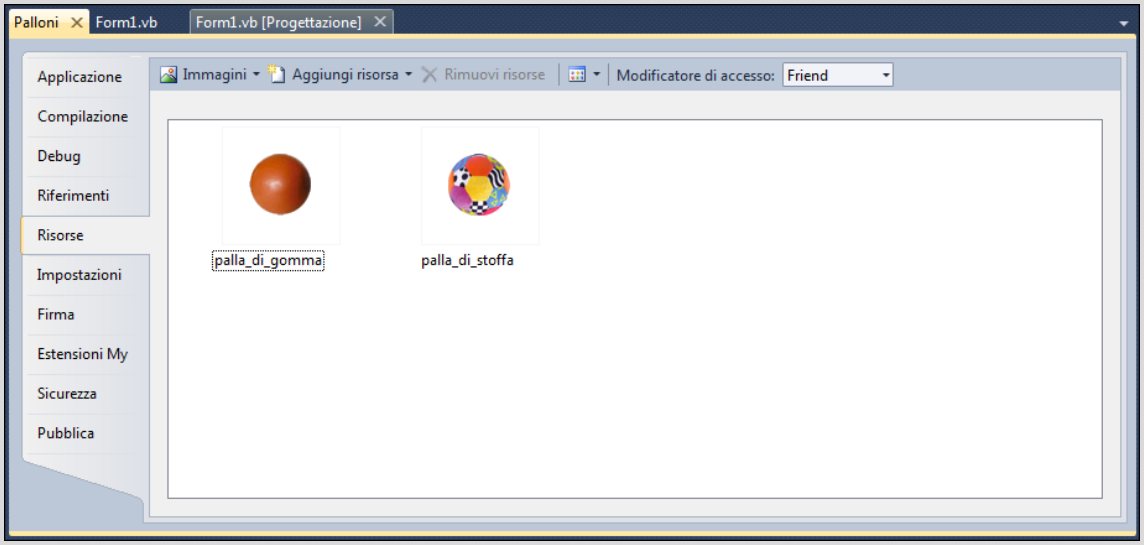
- DoubleBuffered = True
- BackgroundColor = White

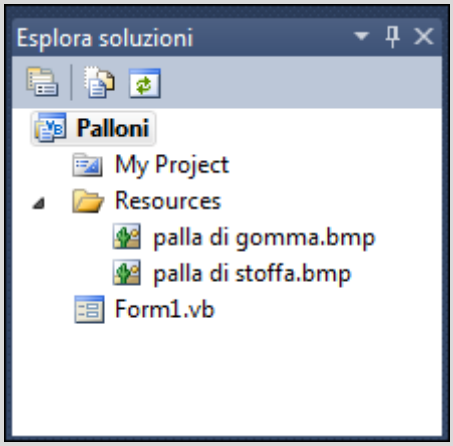
Ora inseriamo nelle risorse del programma le due immagini

- palla_di_gomma.bmp
- palla di stoffa.bmp

che si trovano nella cartella **Documenti \ A scuola con VB \ Immagini**.

La procedura per inserire le immagini nelle risorse del programma è descritta nell'Esercizio 90: Visualizzare un'immagine dalle risorse del programma. 644.
Ecco le immagini relative a questa operazione:





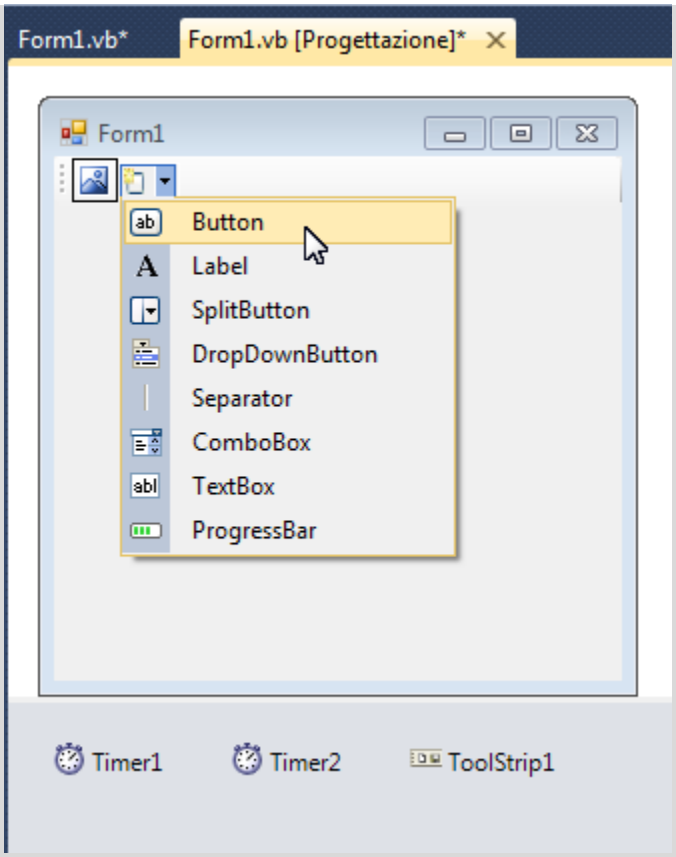
Torniamo a occuparci del form.
Inseriamo nel form due controlli Timer con queste proprietà:
Timer1:

- **Enabled = False**
- **Interval = 50**

Timer2:

- **Enabled = False**
- **Interval = 100**

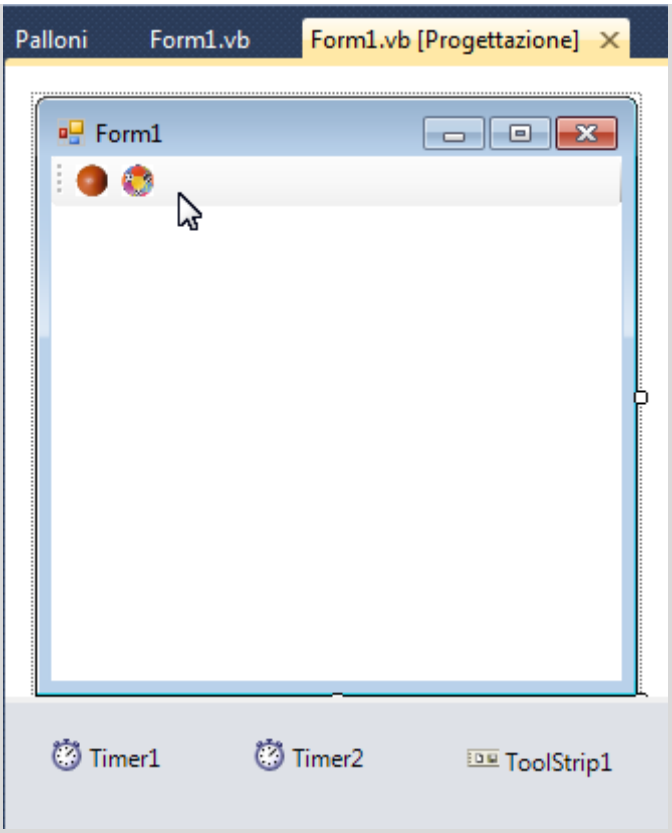
Ora inseriamo nel form un controllo ToolStrip e attiviamo al suo interno due pulsanti Button:



Inseriamo nei due pulsanti le due immagini con i palloni che abbiamo a disposizione nelle risorse del programma e impostiamo la loro proprietà **Text** come segue:

- per il ToolStripButton1: **Text = palla di gomma**
- per il ToolStripButton2: **Text = palla di stoffa**

Il form si presenta ora in questo modo:



Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
    ' Memorizza i limiti entro i quali si muoveranno i palloni:
    ' a sinistra, il limite è il bordo sinistro del form:
    Dim Sinistra As Integer = 0
    ' in basso, il limite è dato dall'altezza dell'area utilizzabile del form:
    Dim Altezza As Integer = Me.ClientRectangle.Height

    ' Crea due variabili per controllare le direzioni dei palloni:
    Dim Ritorno As Boolean = False
    Dim Caduta As Boolean = False

    ' Crea un'immagine virtuale con l'immagine della palla di gomma
    Dim ImmagineVirtuale As Bitmap = My.Resources.palla_di_gomma
```

```
    Private Sub ToolStripButton1_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripButton1.Click

        ' Con il primo pulsante prendiamo l'immagine della palla di gomma
        ' e attiviamo il Timer1:
        Sinistra = 0
        Altezza = Me.ClientRectangle.Height - 50
        ImmagineVirtuale = My.Resources.palla_di_gomma
        Timer1.Enabled = True
        Timer2.Enabled = False

    End Sub
```

```
    Private Sub ToolStripButton2_Click_1(sender As System.Object, e As
System.EventArgs) Handles ToolStripButton2.Click

        ' Con il secondo pulsante prendiamo l'immagine della palla di stoffa e
attiviamo il Timer2:
        Sinistra = 0
        Altezza = Me.ClientRectangle.Height - 50
        ImmagineVirtuale = My.Resources.palla_di_stoffa
        Timer1.Enabled = False
        Timer2.Enabled = True

    End Sub
```

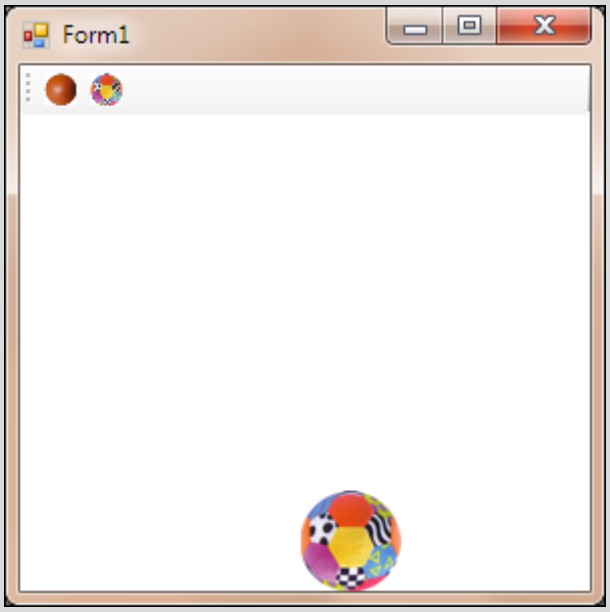
```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick
        ' il Timer1 controlla i movimenti della palla di gomma, che rimbalza
contro le "pareti" del form:

        Me.Invalidate()

        If Sinistra > (Me.ClientRectangle.Width - 50) Then
            Ritorno = True
        ElseIf Sinistra < 0 Then
            Ritorno = False
        End If

        If Ritorno = True Then
            Sinistra -= 3
        Else
            Sinistra += 3
        End If

        If Altezza < 25 Then
            Caduta = True
        ElseIf Altezza > Me.ClientRectangle.Height - 50 Then
            Caduta = False
        End If

        If Caduta = True Then
            Altezza += 5
        Else
            Altezza -= 5
        End If

    End Sub
```

```
    Private Sub Timer2_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer2.Tick

        ' il Timer2 controlla i movimenti della palla di stoffa, che rotola
avanti e indietro
        ' sul limite inferiore del form:

        Me.Invalidate()

        If Sinistra > (Me.ClientRectangle.Width - 50) Then
            Ritorno = True
        ElseIf Sinistra < 10 Then
            Ritorno = False
        End If
```

```
        If Ritorno = True Then
            Sinistra -= 10
        Else
            Sinistra += 10
        End If

        Altezza = Me.ClientRectangle.Height - 50

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' ruota l'immagine della palla di stoffa, se questa è in movimento:
        If Timer2.Enabled = True Then
ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate90FlipNone)

        e.Graphics.TranslateTransform(Sinistra, Altezza)
        e.Graphics.DrawImage(ImmagineVirtuale, 0, 0)

    End Sub

End Class
```

Mandiamo in esecuzione il programma.
Cliccando i due pulsanti, si visualizza l'animazione del pallone corrispondente:



Il codice del programma non dovrebbe presentare difficoltà alla lettrice o al lettore che abbiano seguito il percorso compiuto sino a qui sulla programmazione grafica.
Ricordiamo che la proprietà Me.ClientRectangle memorizza le dimensioni utili del form, vale a dire lo spazio a disposizione del programmatore, senza considerare i bordi e la barra blu in alto.

165: Immagini in sequenza.

Un altro effetto di animazione può essere ottenuto visualizzando sul form o all'interno di un controllo una serie di immagini in rapida sequenza.
Il programma seguente mostra il disegno animato di una rana che cammina. L'animazione è ottenuta alternando due immagini della rana, all'interno di un controllo PictureBox.

Esercizio 104: Animazione con due immagini in sequenza.

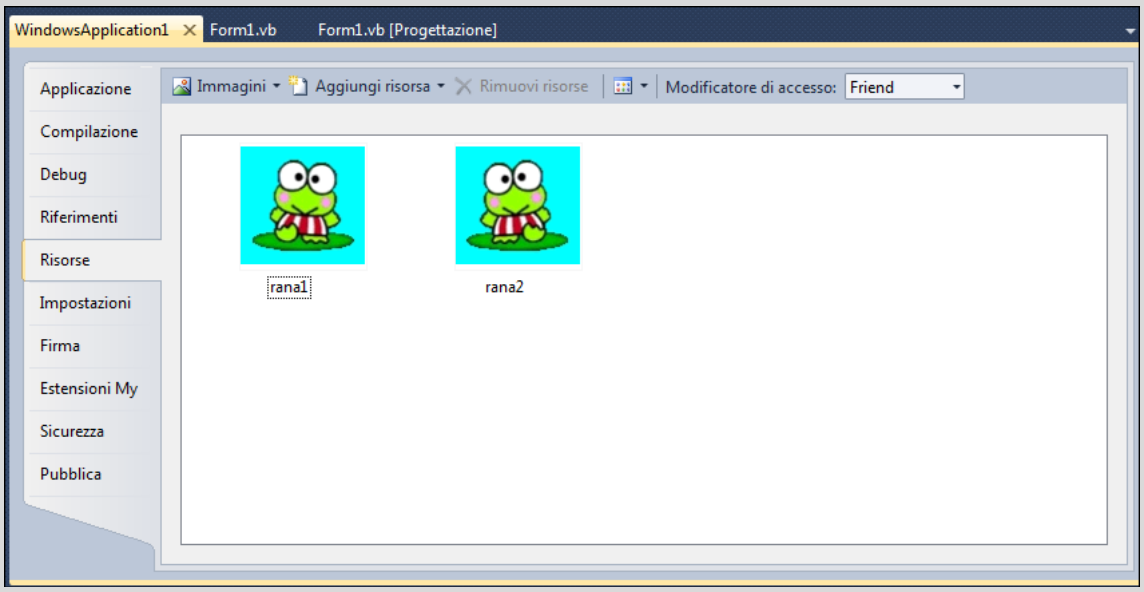
Apriamo un nuovo progetto.
Proprietà del **Form1**:

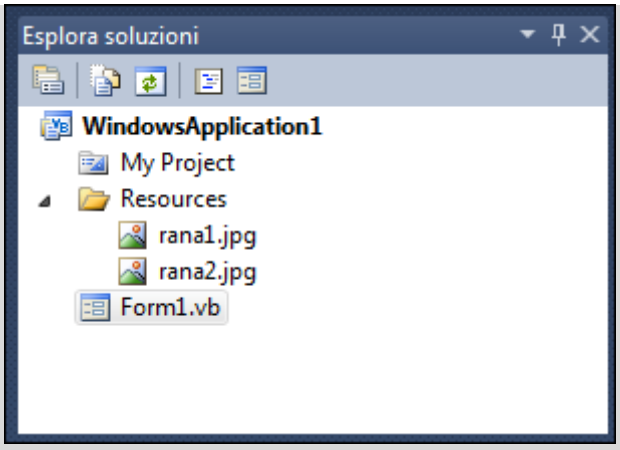
- **DoubleBuffered = True**
- **BackgroundColor = Cyan**

Inseriamo nelle risorse del programma le due immagini

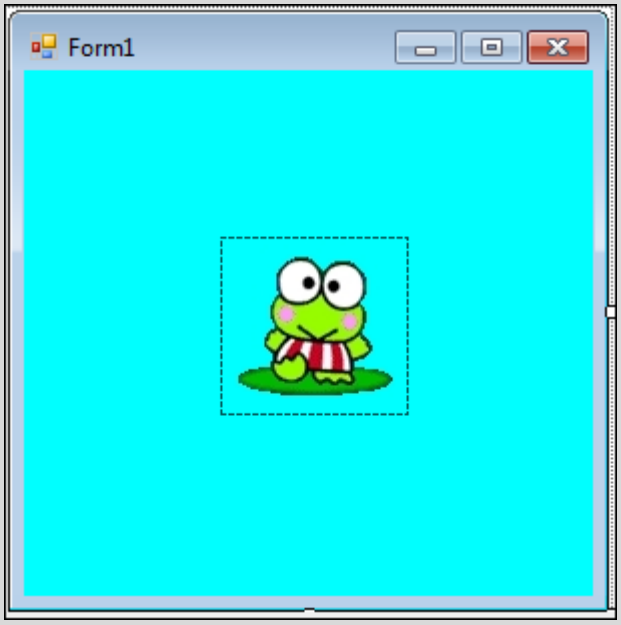
- **rana1.jpg**
- **rana2.jpg**

che si trovano nella cartella **Documenti \ A scuola con VB \ Immagini**.
La procedura per inserire le immagini nelle risorse del programma è descritta nell'Esercizio 90: Visualizzare un'immagine dalle risorse del programma. 644.
Ecco le immagini relative a questa operazione:





Inseriamo nel form un controllo PictureBox e visualizziamo nel PictureBox la prima delle due immagini con la rana:



Infine, inseriamo nel progetto un controllo **Timer** con queste proprietà:

- **Enabled = True**
- **Interval = 200**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
    ' crea una variabile per controllare l'alternarsi delle due immagini:
    Dim PassoSinistro As Boolean
```

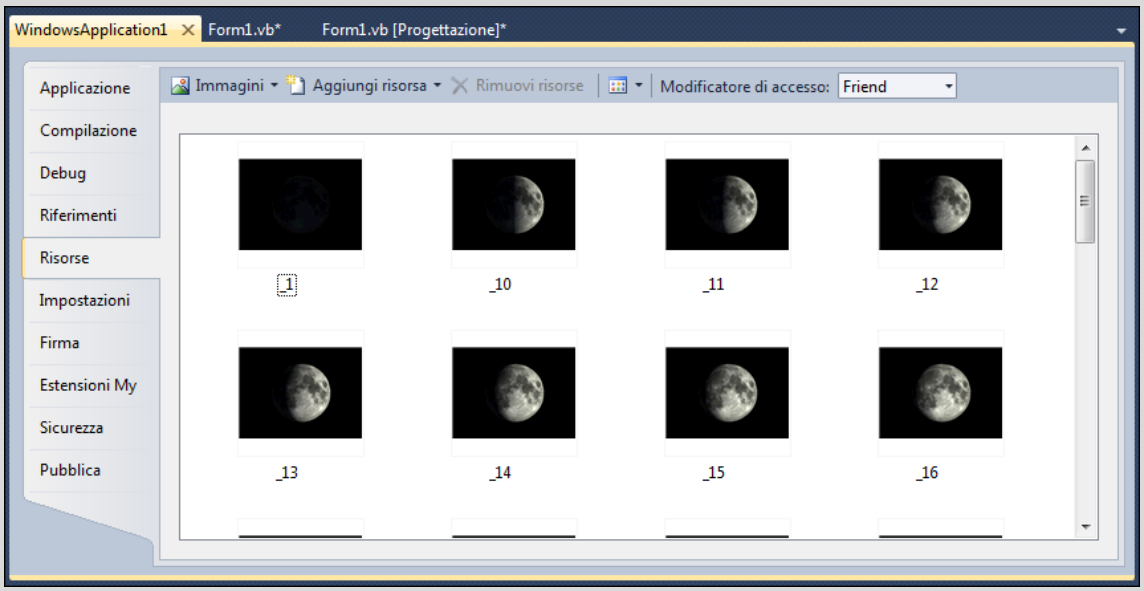
```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        PassoSinistro = Not (PassoSinistro)

        If PassoSinistro = True Then
            PictureBox1.Image = My.Resources.rana1
```

```
        Else
            ' altrimenti:
            PictureBox1.Image = My.Resources.rana2
        End If

    End Sub

End Class
```

Il timer si attiva all'avvio del programma e fa alternare le due immagini della rana all'interno del PictureBox1, dando l'illusione che la rana cammini.

Esercizio 105: Le fasi lunari.

In questo programma, che ricalca il programma dell'esercizio precedente, le immagini in sequenza sono 36 immagini delle fasi lunari; il tic del timer determina il succedersi di queste 36 immagini sullo sfondo del form.
Apriamo un nuovo progetto.
Inseriamo nelle risorse del programma le 36 immagini che si trovano nella cartella **Documenti \ A scuola con VB \ Immagini \ Fasi lunari**.
La procedura per inserire le immagini nelle risorse del programma è descritta nell'eEsercizio 90: Visualizzare un'immagine dalle risorse del programma.644.
Ecco l'immagine relativa a questa operazione:



Notiamo che VB ha inserito queste immagini nelle risorse del programma modificandone il nome: ogni numero è preceduto dal trattino "_", che non compare nel nome originale; dovremo tenere conto di questo trattino nella scrittura del codice.
Inseriamo nel form, in basso, due pulsanti Button.
Inseriamo infine un controllo **Timer** con queste impostazioni:

- Enabled = True
- Interval = 100.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' crea una variabile per controllare i tic del timer:
    Dim Contatore As Integer = 1
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' all'avvio del programma, scrivi il testo dei due pulsanti button:
        Button1.Text = "Pausa"
        Button2.Text = "Rallenta"

    End Sub
```

```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ' se il contatore supera il numero 36 è necessario riprendere da capo,
dalla prima immagine:
        If Contatore > 36 Then Contatore = 1

        ' componi il nome dell'immagine unendo il trattino _ e il numero del
contatore:
        Dim NomeImmagine As String = "_" & CStr(Contatore)
        ' visualizza l'immagine dalle risorse del programma:
        Me.BackgroundImage = My.Resources.ResourceManager.GetObject(NomeImmagine)

        ' aumenta il contatore dei tic di una unità:
        Contatore += 1

    End Sub
```

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click
        ' il pulsante Button1 gestisce la pausa e la ripresa del programma,
fermando o riavviando il timer:

        If Button1.Text = "Pausa" Then
            Timer1.Enabled = False
            Button1.Text = "Riprendi"
        Else
            Timer1.Enabled = True
            Button1.Text = "Pausa"
        End If

    End Sub
```

```
    Private Sub Button2_Click(sender As System.Object, e As System.EventArgs)
Handles Button2.Click
        ' il pulsante Button2 gestisce l'accelerazione o il rallentamento del
timer:

        If Button2.Text = "Rallenta" Then
            Timer1.Interval = 400
```

```
            Button2.Text = "Accelera"
        Else
            Timer1.Interval = 200
            Button2.Text = "Rallenta"
        End If

    End Sub

End Class
```

I *tic* del timer fanno alternare le 36 immagini delle fasi lunari che si trovano nelle risorse del programma.
Il nome dell'immagine da visualizzare, a ogni *tic*, è dato dall'unione di un trattino e del numero del contatore dei *tic*.
Notiamo che il recupero della immagine dalle risorse del programma avviene con il comando **GetObject(NomeImmagine)** e non direttamente, con il nome dell'immagine, come nell'esercizio precedente.

```
        ' componi il nome dell'immagine unendo il trattino _ e il numero del
contatore:
        Dim NomeImmagine As String = "_" & CStr(Contatore)
        ' visualizza l'immagine dalle risorse del programma:
        Me.BackgroundImage = My.Resources.ResourceManager.GetObject(NomeImmagine)
```

Ecco un'immagine del programma in esecuzione:

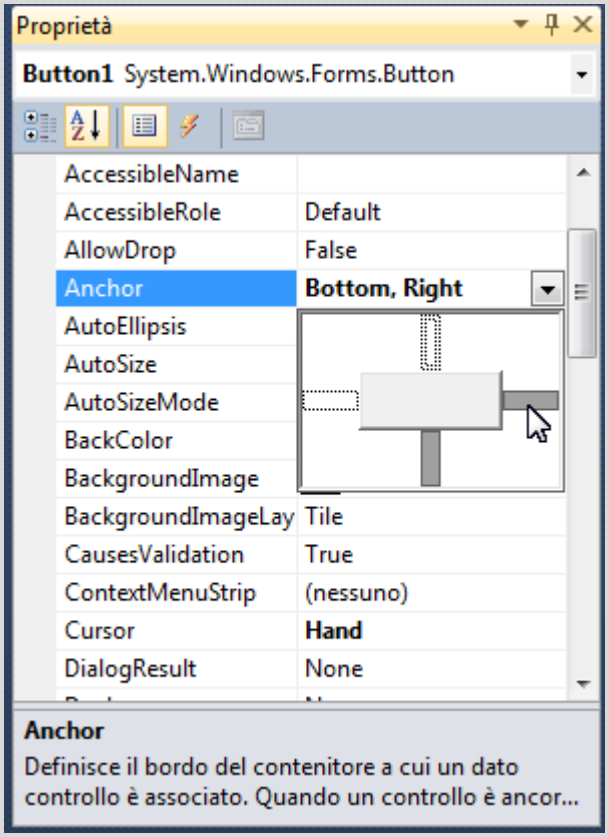
166: Scritte in movimento.

Anche il movimento di scritte in un form o in un controllo si basa sui *tic* di un timer: a ogni *tic*, la posizione della scritta varia di alcuni pixel, dando l'impressione visiva di un movimento continuo.
Gli esercizi che seguono mostrano movimenti di scritte in diverse direzioni, effettuati con diverse modalità.

Esercizio 106: Testo scorrevole in orizzontale.

Apriamo un nuovo progetto.
Impostiamo la proprietà del Form1 **DoubleBuffered = True**, per evitare lo sfarfallìo dell'immagine.
Inseriamo nel form un pulsante **Button1** con queste proprietà:

- Anchor = Bottom, Right
- Cursor = Hand



Inseriamo nel progetto un **Timer**, con la proprietà **Interval = 10**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1
```

```
    Dim Testo As String = "Testo scorrevole da destra a sinistra"
    Dim Carattere As New Font("Arial", 16, FontStyle.Bold)
    ' variabili per memorizzare la posizione del testo:
    Dim Sinistra As Integer = 0
    Dim Altezza As Integer = 0

    ' misura e memorizza la posizione e le dimensioni dlel'area rettangolare
occupata dal testo:
    Dim Area As SizeF = Me.CreateGraphics.MeasureString(Testo, Carattere)
```

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        ' spostamento laterale del testo di un pixel a sinistra
        Sinistra -= 1

        ' se la posizione di sinistra del testo è inferiore alla lunghezza del
testo,
        ' (cioè: tutto il testo è uscito dal form, a sinistra)
        ' allora reimposta la posizione di sinistra del testo uguale alla
grandezza del form
        ' (cioé: al bordo destro del form)
        If Sinistra < -Area.Width Then
            Sinistra = Me.ClientRectangle.Width
            ' imposta un nuovo valore per l'altezza del testo:
            Dim Sorteggio As New Random
            Randomize()
            Altezza = Sorteggio.Next(0, Me.ClientRectangle.Height)
        End If

        ' questo comando cancella il contenuto del form e nello stesso tempo
attiva l'evento Paint del form,
        ' la cui procedura riscrive il testo nella nuova posizione:
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' migliora la qualità della resa grafica del testo:
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias
        ' disegna il testo a partire dalla posizione di sinistra
        e.Graphics.DrawString(Testo, Carattere, Brushes.Blue, New
PointF(Sinistra, Altezza))

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        ' il clic sul pulsante attiva o disattiva il timer:
        Timer1.Enabled = Not Timer1.Enabled

    End Sub

End Class
```

Ecco un’immagine del programma in esecuzione:

Notiamo che, allargando il form, lo scorrimento del testo si adatta alle nuove dimensioni e il pulsante Button rimane nell'angolo in basso a destra.

Lo scorrimento del testo da sinistra a destra si ottiene con queste modifiche:

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        ' spostamento laterale del testo di un pixel a destra
        Sinistra += 1

        ' se la posizione di destra del testo supera la larghezza del form,
        ' (cioè: tutto il testo è uscito dal form, a destra)
        ' allora reimposta la posizione di sinistra del testo in modo che la
scritta si venga a trovare oltre il bordo sinistro del form:
        If Sinistra > Me.ClientRectangle.Width Then Sinistra = 0 - Area.Width

        ' questo comando cancella il contenuto del form e nello stesso tempo
attiva l'evento Paint del form,
        ' la cui procedura riscrive il testo nella nuova posizione:
        Me.Invalidate()

    End Sub
```

Lo scorrimento del testo dall'alto al basso si ottiene con queste modifiche:

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        ' spostamento del testo di un pixel verso il basso
        Altezza += 1

        ' se la posizione verticale del testo supera il limite inferiore del form
        ' (cioè: tutto il testo è uscito dal form, in basso)
        ' allora reimposta la posizione del testo riportandolo in alto
```

```
        ' (cioé: oltre il limite superiore del form)
        If Altezza > Me.ClientRectangle.Height Then
            Altezza = 0 - Area.Height
        End If

        ' questo comando cancella il contenuto del form e nello stesso tempo
attiva l'evento Paint del form,
        ' la cui procedura riscrive il testo nella nuova posizione:
        Me.Invalidate()

    End Sub
```

Lo scorrimento del testo dal basso verso l'alto si ottiene con queste modifiche:

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        ' spostamento del testo di un pixel verso l'alto
        Altezza -= 1

        ' se la posizione verticale del testo supera il limite superiore del form
        ' (cioè: tutto il testo è uscito dal form, in alto)
        ' allora reimposta la posizione del testo riportandolo in basso
        ' (cioé: oltre il limite inferiore del form)
        If Altezza < 0 - Area.Height Then
            Altezza = Area.Height
        End If

        ' questo comando cancella il contenuto del form e nello stesso tempo
attiva l'evento Paint del form,
        ' la cui procedura riscrive il testo nella nuova posizione:
        Me.Invalidate()

    End Sub
```

Nel prossimo esercizio vedremo un testo che scorre dal basso verso l'alto, sfumando man mano verso la trasparenza che sale verso il bordo superiore del form.

Esercizio 107: Testo scorrevole in verticale a scomparsa.

Apriamo un nuovo progetto.
Impostiamo la proprietà del Form1

- DoubleBuffered = True, per evitare lo sfarfallìo dell'immagine.

Inseriamo nel form un pulsante **Button1** con queste proprietà:

- Anchor = Bottom, Right
- Cursor = Hand

Inseriamo nel progetto un **Timer**, con le proprietà

- Enabled = True
- Interval = 10.

Copiamo e incolliamo nella Finestra del Codice questo listato:

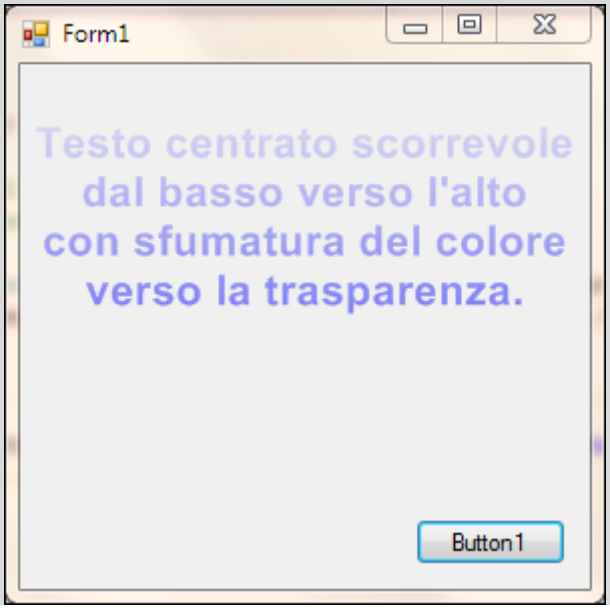
```
Public Class Form1

    Dim Testo As String = "Testo centrato scorrevole" & vbCrLf & "dal basso verso
l'alto" & vbCrLf & "con sfumatura del colore" & vbCrLf & "verso la trasparenza."
    Dim Carattere As New Font("Arial", 16, FontStyle.Bold)

    ' misura lo spazio rettangolare occupato dalla scritta
    Dim Area As SizeF = Me.CreateGraphics.MeasureString(Testo, Carattere)

    ' variabili per memorizzare la posizione iniziale del testo nel form:
    Dim Sinistra As Integer = 0
    Dim Altezza As Integer = 0
```

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick

        ' muove il testo di un pixel verso l'alto
        Altezza -= 1

        ' se la posizione verticale del testo supera il limite superiore del form
        ' (cioè: tutto il testo è uscito dal form, in alto)
        ' allora reimposta la posizione del testo riportandolo in basso
        ' oltre il limite inferiore del form
        If Altezza < 0 - Area.Height Then
            Altezza = Me.ClientRectangle.Height
        End If

        ' questo comando cancella il contenuto del form,
        ' ma nello stesso tempo attiva l'evento Paint del form,
        ' la cui procedura riscrive il testo nella nuova posizione:
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' ottimizza la qualità grafica del testo:
        e.Graphics.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias

        ' crea un nuovo formato StringFormat:
        Dim Formato As New StringFormat
        ' centratura in senso orizzontale:
        Formato.Alignment = StringAlignment.Center

        ' l'effetto a scomparsa del testo è dato dall'uso di un pennello con
colori gradienti, della stessa grandezza del form.
        ' Questo pennello scrive il testo con il colore blu nella parte bassa del
form e con il colore trasparente nella parte alta.

        Dim Pennello As New Drawing2D.LinearGradientBrush(Me.ClientRectangle,
Color.Transparent, Color.Blue, 90)
        e.Graphics.DrawString(Testo, Carattere, Pennello, New
PointF(Me.ClientRectangle.Width / 2, Altezza), Formato)

    End Sub
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        ' il clic sul pulsante attiva o disattiva il timer:
        Timer1.Enabled = Not Timer1.Enabled
    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Capitolo 30: STAMPA.

Croce e delizia di ogni programmatore, la stampa di testi, dati o immagini è una funzione che accompagna spesso i programmi per i computer; tale abbinamento è ancora più stretto in ambito scolastico, dove la stampa su carta è spesso utilizzata per la documentazione, la conservazione e la diffusione di materiali diversi.
Per gestire la funzione di stampa VB mette a disposizione del programmatore questi controlli:
Nel gruppo **Stampa:**

- **PageSetupDialog**: consente di gestire alcune opzioni relative al foglio da stampare, compresi i margini e l'orientamento del foglio;
- **PrintDialog**: consente di scegliere la stampante e le pagine da stampare all'inrerno di un documento;
- **PrintDocument**: crea un oggetto grafico sul quale è possibile disegnare linee, forme, scritte e immagini da inviare alla stampante;
- **PrintPreviewControl:** visualizza l'anteprima della stampa, senza alcuna altra funzione;
- **PrintPreviewDialog**: visualizza l'anteprima della stampa con alcune funzioni di visualizzazione e con la possibilità di passare direttamente alla stampa.

Nel gruppo **Visual Basic PowerPacks**:

- **PrintForm**: consente di stampa diretta e immediata di un form, con gli oggetti in esso contenuti.

In questo capitolo vedremo l'uso dei tre controlli fondamentali per la stampa: **PrintForm, PrintDocument e PrintPreviewDialog**.

La gestione della stampa con VB può essere effettuata in due modi diversi, con risultati di qualità diversa, a seconda delle esigenze del programmatore:

- con il controllo **PrintForm** è possibile, scrivendo poche righe di codice, **mandare** in stampa l'interfaccia del programma visibile sul monitor, con tutto ciò che in essa è contenuto (grafica e testi);
- con il componente **PrintDocument** è possibile elaborare un oggetto grafico con linee, forme, scritte e immagini da inviare alla stampante.

Il **primo** metodo è semplice e immediato, il **secondo** è più laborioso ma offre maggiori possibilità di adattamento della fase di stampa e garantisce risultati di qualità migliore.

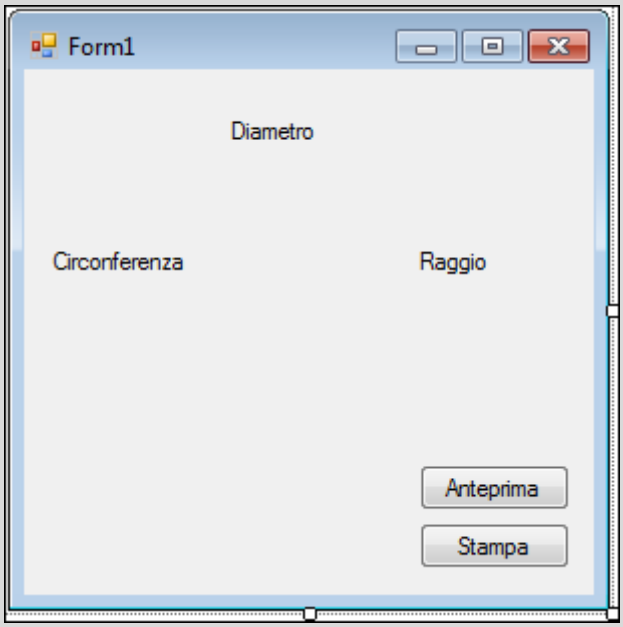
167: Il controllo PrintForm.

Il componente **PrintForm** è uno dei controlli comuni di VB.
Si trova nell'ultimo gruppo della lista dei controlli di VB: è il gruppo denominato **Visual Basic PowerPacks**, che comprende i controlli giunti a Visual Basic 2010 dalle versioni precedenti di Visual Basic.
Come suggerisce il suo nome, questo controllo stampa immediatamente tutto ciò che è visibile sul monitor nel momento in cui si avvia la funzione di stampa.
La facilità d'uso del controllo e la sua immediatezza hanno ovviamente un rovescio della medaglia: la scelta del materiale da stampare e la qualità della stampa sono rigide e lasciano al programmatore uno spazio di manovra molto ridotto.

Si può ricorrere a **PrintForm** quando non si hanno particolari esigenze di qualità e dunque non si ritiene opportuno costruire una normale funzione di stampa, e quando tutto ciò che si desidera stampare è visibile nel form che si manda in stampa.
Nel prossimo esercizio vedremo un esempio di utilizzo del comando **PrintForm** nella sua forma più semplice.

Esercizio 108: Il comando PrintForm.

Questo programma disegna nel form una circonferenza con il suo diametro e il raggio, e consente di visualizzare l'anteprima di stampa o di mandare in stampa il contenuto del form.
Apriamo un nuovo progetto e inseriamo nel form tre **Label** e due pulsanti **Button** come in questa immagine:



La proprietà **Text** dei cinque controlli, come si vede nell'immagine, è impostata come segue:

- Label1: **Text = Circonferenza**
- Label2: **Text = Diametro**
- Label 3: **Text = Raggio**
- Button1: **Text = Anteprima**
- Button2: **Text = Stampa**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' Colora di bianco lo sfondo del form per la stampa:
        Me.BackColor = Color.White

        ' Colora il testo delle tre etichette:
        Label1.ForeColor = Color.Black
        Label2.ForeColor = Color.Blue
        Label3.ForeColor = Color.Red

        ' Disegna il raggio, il diametro e la circonferenza:
        e.Graphics.DrawPie(Pens.Red, 10, 10, 200, 200, 0, 360)
        e.Graphics.DrawPie(Pens.Blue, 10, 10, 200, 200, 90, 180)
        e.Graphics.DrawEllipse(Pens.Black, 10, 10, 200, 200)

    End Sub
```

```
    Private Sub Button1_Click() Handles Button1.Click

        ' Attiva la procedura per nascondere temporaneamente i due pulsanti
Button1 e Button2:
        Call NascondiPulsanti()

        ' Comanda l'anteprima di stampa.
        PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
        PrintForm1.Print()

        ' Attiva la procedura per rendere di nuovo visibili i due pulsanti
Button1 e Button2:
        Call VisualizzaPulsanti()

    End Sub
```

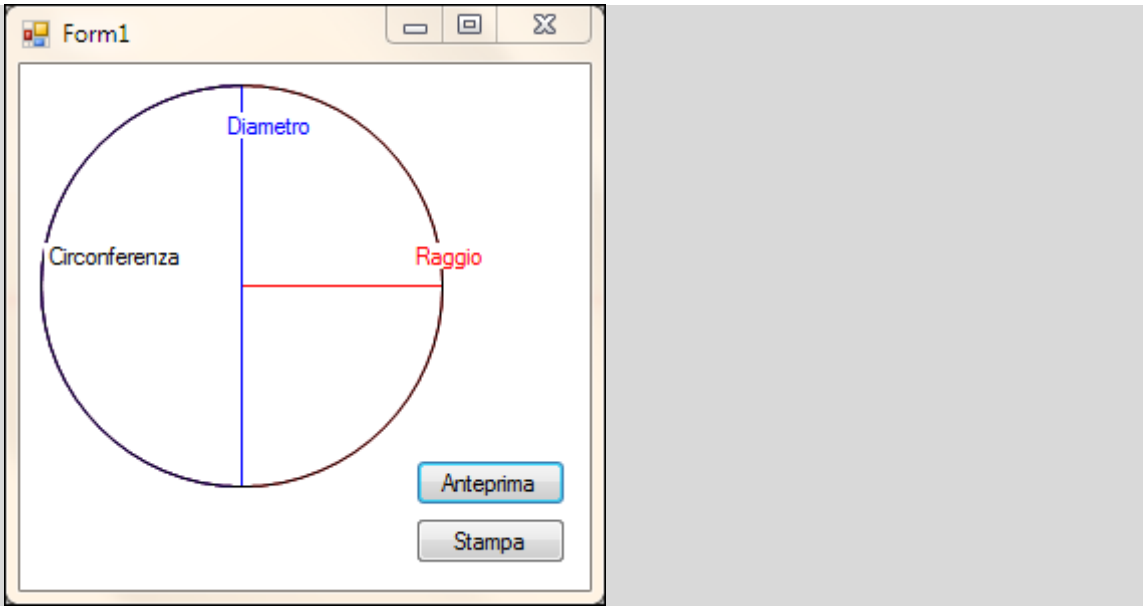
```
    Private Sub Button2_Click() Handles Button2.Click

        ' Attiva la procedura per nascondere temporaneamente i due pulsanti
Button1 e Button2:
        Call NascondiPulsanti()
```

```
        ' Comanda la stampa.
        PrintForm1.PrintAction = Printing.PrintAction.PrintToPrinter
        PrintForm1.Print()

        ' Attiva la procedura per rendere di nuovo visibili i due pulsanti
Button1 e Button2:
        Call VisualizzaPulsanti()

    End Sub
```

```
    Sub NascondiPulsanti()

        ' Procedura per nascondere temporaneamente i due pulsanti Button1 e
Button2
        For Each Controllo As Control In Me.Controls
            If TypeOf Controllo Is Button Then Controllo.Visible = False
        Next

        ' Esegui la procedura prima di procedere con il programma:
        Application.DoEvents()

    End Sub
```

```
    Sub VisualizzaPulsanti()

        ' Procedura per rendere visibili i due pulsanti Button1 e Button2
        For Each Controllo As Control In Me.Controls
            If TypeOf Controllo Is Button Then Controllo.Visible = True
        Next

        ' Esegui la procedura prima di procedere con il programma:
        Application.DoEvents()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Osserviamo, nel codice dell'esercizio precedente, i due comandi

```
PrintForm1.PrintAction = Printing.PrintAction. PrintToPreview
PrintForm1.PrintAction = Printing.PrintAction.PrintToPrinter
```

che dirigono la stampa, rispettivamente, verso l'anteprima di stampa sul monitor e verso la stampante.
Questo comando stampa il contenuto del form, eliminando i bordi e la barra in alto:

```
PrintForm1.Print()
```

Per stampare l'intero form, con bordi e barra, è necessario impostare il comando PrintForm1.Print come segue:

```
PrintForm1.Print(Me, PowerPacks.Printing.PrintForm.PrintOption.FullWindow)
```

Per stampare un form troppo grande, non interamente visibile sul monitor, è necessario impostare il comando PrintForm1.Print come segue:

```
PrintForm1.Print(Me, PowerPacks.Printing.PrintForm.PrintOption.Scrollable)
```

Per stampare solo la parte visibile di un form troppo grande, non interamente visibile sul monitor, è necessario impostare il comando PrintForm1.Print come segue:

```
PrintForm1.Print(Me, PowerPacks.Printing.PrintForm.PrintOption.ClientAreaOnly)
```

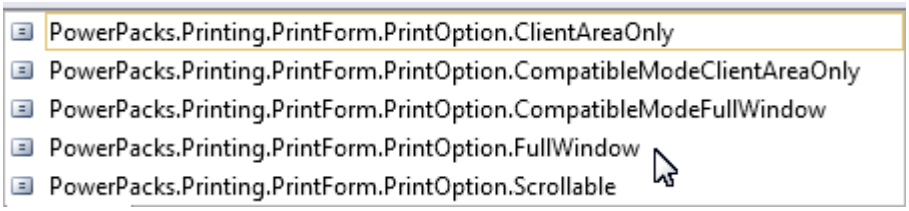


Figura 221: Modalità di cattura dell'area da stampare con il controllo PrintForm.

Il controllo PrintForm, come tutti i controlli, è in grado di registrare alcuni eventi che possono accadere durante lo svolgimento del programma. Il loro elenco è visibile nei menu a tendina nella Finestra di Progettazione:

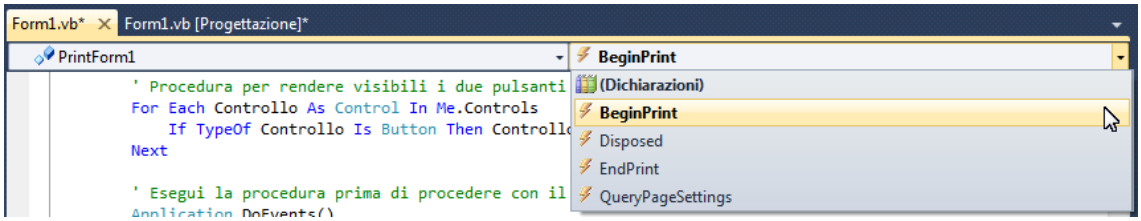


Figura 222: L'elenco degli eventi del controllo PrintForm.

Utilizzando l'evento **PrintForm1.BeginPrint** è possibile inserire nel codice dell'esercizio precedente una procedura con un messaggio da visualizzare all'avvio della stampa del documento:

```
    Private Sub PrintForm1_BeginPrint(sender As Object, e As
System.Drawing.Printing.PrintEventArgs) Handles PrintForm1.BeginPrint

        MsgBox("Inizia la stampa")

    End Sub
```

168: Il componente PrintDocument.

Il componente **PrintDocument** è l'oggetto che rappresenta la stampante dell'utente del programma; esso deve essere inserito in ogni programma con funzioni di stampa, a meno che il programmatore si accontenti dell'uso del controllo PrintForm che abbiamo visto nel paragrafo precedente.
Per inserire **PrintDocument** in un progetto è sufficiente cliccarlo nella Casella degli Strumenti e portarlo nell'applicazione, come ogni altro controllo.
Dopo avere inserito l'oggetto **PrintDocument** in un progetto, con il comando

```
        PrintDocument1.Print()
```

si crea una superficie virtuale sulla quale è possibile disegnare o scrivere; tutto ciò che è disegnato o scritto in questa superficie è inviato alla stampante o all'anteprima di stampa.
L'uso della superficie grafica creata con **PrintDocument** è simile all'uso della superficie grafica creata con la classe **Graphics**[81]: ad esempio, per disegnare linee o figure geometriche si usano i comandi **DrawLine**, **DrawRectangle**, **FillRectangle**, per scrivere un testo si usa il comando **DrawString**, ecc.
La differenza essenziale tra la classe Graphics e il componente PrintDocument sta nella destinazione del loro contenuto:

- i comandi della classe Graphics sono diretti al monitor;
- i comandi del componente PrintDocument sono diretti alla stampante e consentono di impostare tutte le opzioni di stampa (scelta della stampante, dimensioni del foglio, dimensioni dei margini, numero delle copie).

Facciamo un riepilogo e un confronto.

1.
Con la classe Graphics, il comando **Me.Invalidate** causa la ripulitura del form e attiva la procedura che gestisce l'evento **Me.Paint**. L'evento **Me.Paint** crea la superficie grafica **e.Graphics** sulla quale il programmatore può disegnare o scrivere; il prodotto finale è inviato al **monitor**:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' la superficie grafica creata da VB corrisponde al Form1,
        ' per cui questa linea è tracciata sul monitor:
        e.Graphics.DrawLine(Pens.Red, 0, 0, 200, 200)

    End Sub

End Class
```

2.
Dopo avere inserito in un progetto un componente PrintDocument, il comando **PrintDocument1.Print** attiva la procedura che gestisce l'evento **PrintDocument1.PrintPage.** L'evento **PrintDocument1.PrintPage** crea la superficie grafica **e.Graphics** su cui è possibile disegnare o scrivere; il prodotto finale è inviato alla stampante:

[81] Paragrafo 126: La classe Graphics. 570.

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        PrintDocument1.Print()

    End Sub
```

```
    Private Sub PrintDocument1_PrintPage(sender As Object, e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        ' la superficie grafica creata da VB corrisponde alla stampante,
        ' per cui questa linea è tracciata sulla stampante:
        e.Graphics.DrawLine(Pens.Red, 0, 0, 200, 200)

    End Sub

End Class
```

169: Il componente PrintPreviewDialog.

Assieme all'oggetto **PrintDocument**, VB mette a disposizione del programmatore il componente **PrintPreviewDialog**, che consente di vedere sul monitor l'anteprima delle operazioni di stampa.
Per non consumare carta inutilmente, nelle prossime prove di stampa useremo dunque l'oggetto PrintPreviewDialog.
Il componente PrintDocument, la cui presenza è comunque necessaria, sarà utilizzato come strumento di supporto a PrintPreviewDialog e non come strumento per la stampa.
Dopo avere inserito in un progetto il componente **PrintPreviewDialog** , l'anteprima di stampa si ottiene con questi comandi:

```
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()
```

La lettrice o il lettore troveranno dunque negli esercizi che seguono il codice con il comando di stampa **PrintDocument** presente ma disattivato:

```
        ' PrintDocument1.Print()
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()
```

Il componente PrintPreviewDialog richiede che nel progetto sia inserito il componente **PrintDocument**, in quanto quest'ultimo crea la superficie grafica che è visualizzata nell'anteprima di stampa.

La connessione **PrintPreviewDialog1.Document = PrintDocument1** è dunque indispensabile per il funzionamento di PrintPreviewDialog e può essere impostata oltre che dal codice del programma anche dalla Finestra delle Proprietà di PrintPreviewDialog:

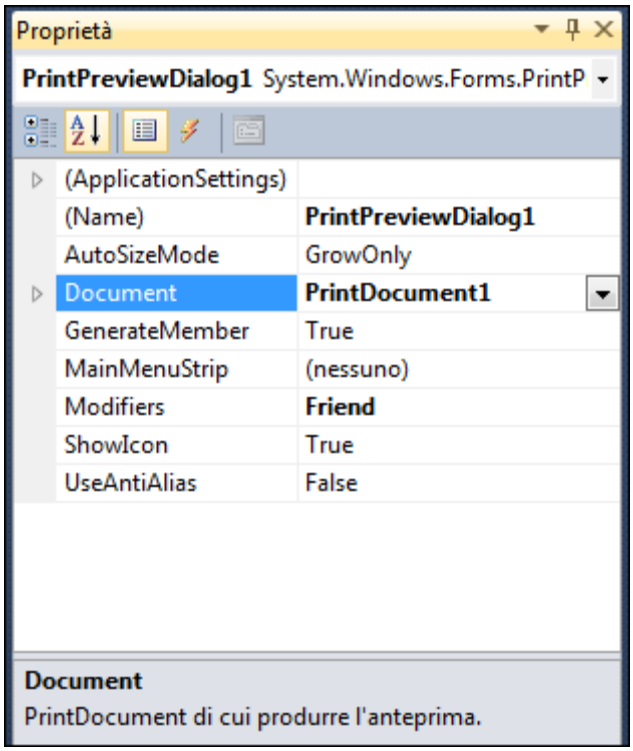


Figura 223: La connessione tra PrintPreviewDialog e PrintDocument.

Nel prossimo esercizio vedremo ancora in parallelo i diversi àmbiti di azione della classe Graphics e del componente PrintDocument.
Seguiranno due esercizi nei quali sono ripetuti alcuni esercizi grafici già realizzati in precedenza; in questa occasione, ciò che allora era inviato al monitor verrà ricreato per essere inviato alla stampante.

Esercizio 109: La classe Graphics e il componente PrintDocument.

In questo programma vedremo un esempio di funzionamento in parallelo della classe Graphics (disegno su monitor) e del componente PrintDocument (disegno su stampante).
Apriamo un nuovo progetto e inseriamo nel form:

- un pulsante **Button**;
- un controllo **PrintDocument**;
- un controllo **PrintPreviewDialog**.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click
        ' Causa l'evento PrintPage dell'oggetto PrintDocument e avvia la relativa
procedura.
        ' Il comando PrintDocument1.Print è disattivato per risparmiare carta,
sono invece attivi i due comandi per l'anteprima di stampa:

        'PrintDocument1.Print()
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()

    End Sub
```

```
    Private Sub PrintDocument1_PrintPage(sender As Object, e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        ' Disegna un cerchio nella stampante:
        e.Graphics.DrawEllipse(Pens.Blue, 0, 0, 200, 200)

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Disegna un cerchio nel form:
        e.Graphics.DrawEllipse(Pens.Blue, 0, 0, 200, 200)

    End Sub

End Class
```

Esercizio 110: Creazione e stampa di un disegno.

Questo programma traccia un disegno e lo invia alla stampante.
Il disegno da stampare è lo stesso dell'Esercizio 87: Lo strumento Brushes. 621.
Apriamo un nuovo progetto e inseriamo nel form:

- un pulsante **Button;**
- un controllo **PrintDocument;**
- un controllo **PrintPreviewDialog.**

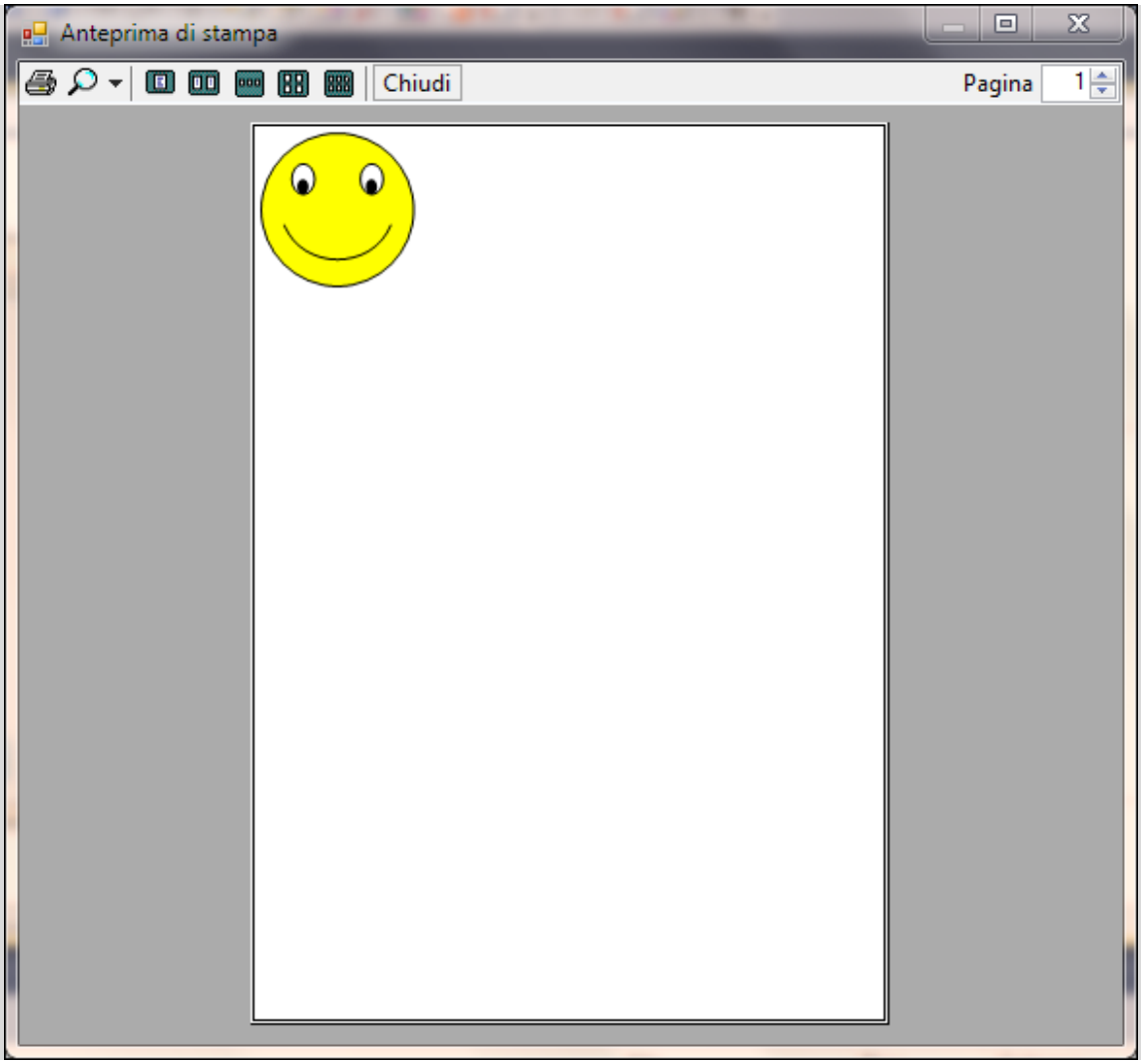
Per inviare il disegno alla stampante e non allo schermo del monitor, prendiamo dal listato di quell'esercizio la parte che riguarda l'evento Me.Paint e la inseriamo nella procedura che gestisce l'evento PrintDocument1.PrintPage:

```
Public Class Form1

    Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage
```

```
        Dim Penna As New Pen(Color.Black, 2)

        ' tutti i comandi che seguono riguardano l'oggetto e.Graphics, per cui
evitiamo di ripeterne il nome a ogni riga:
        With e.Graphics
            .SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
            .FillEllipse(Brushes.Yellow, 10, 10, 200, 200)
            .DrawEllipse(Penna, 10, 10, 200, 200)

            'disegna l'occhio e la pupilla a sinistra:
            .FillEllipse(Brushes.White, 50, 50, 30, 40)
            .DrawEllipse(Penna, 50, 50, 30, 40)
            .FillEllipse(Brushes.Black, 57, 70, 15, 20)

            'disegna l'occhio e la pupilla a destra:
            .FillEllipse(Brushes.White, 140, 50, 30, 40)
            .DrawEllipse(Penna, 140, 50, 30, 40)
            .FillEllipse(Brushes.Black, 147, 70, 15, 20)

            'disegna la bocca
            .DrawBezier(Penna, New Point(40, 130), New Point(65, 190), New
Point(155, 190), New Point(180, 130))

        End With

    End Sub
```

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' Il comando PrintDocument1.Print è disattivato per risparmiare carta,
sono invece attivi i due comandi per
        ' l'anteprima di stampa:

        ' PrintDocument1.Print()
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione, con l'anteprima di stampa:



Esercizio 111: Creazione e stampa di una tabella.

Questo programma crea una tabella e la invia alla stampante.
La tabella da stampare è la stessa che abbiamo visto nell'Esercizio 95: Il comando StringFormat.SetTabStops.689.
Apriamo un nuovo progetto e inseriamo nel form:

- un pulsante **Button**;
- un controllo **PrintDocument**;
- un controllo **PrintPreviewDialog**.

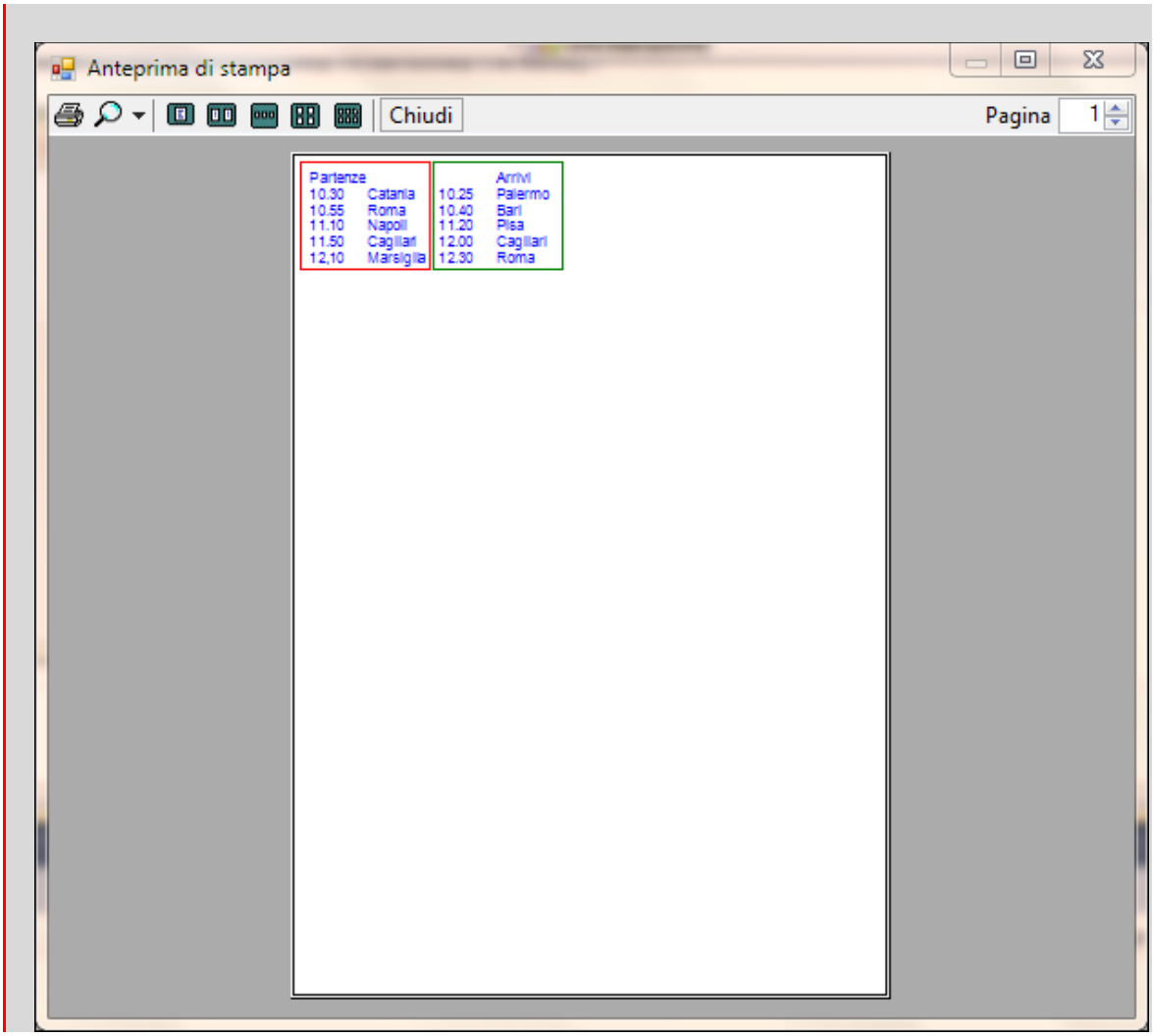
Per inviare la tabella alla stampante e non allo schermo del monitor, prendiamo dal listato di quell'esercizio la parte che riguarda l'evento **Me.Paint** e la inseriamo nella procedura che gestisce l'evento **PrintDocument1**.**PrintPage**:

```
Public Class Form1
```

```
    Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        ' Scelta del carattere:
        Dim Carattere As New Font("Arial", 14)

        ' Il testo da scrivere è composto dalle sei righe che seguono.
        ' Il comando vbTab sposta la parola che lo segue alla tabulazione
successiva,
        ' cioè colloca la parola che lo segue nella colonna successiva.
        ' Il comando vbCrLf termina una riga e manda il testo a capo.
        Dim Testo As String = "Partenze" & vbTab & " " & vbTab & " " & vbTab &
"Arrivi" & vbCrLf & _
 "10.30" & vbTab & "Catania" & vbTab & "10.25" & vbTab & "Palermo" & vbCrLf & _
 "10.55" & vbTab & "Roma" & vbTab & "10.40" & vbTab & "Bari" & vbCrLf & _
 "11.10" & vbTab & "Napoli" & vbTab & "11.20" & vbTab & "Pisa" & vbCrLf & _
 "11.50" & vbTab & "Cagliari" & vbTab & "12.00" & vbTab & "Cagliari" & vbCrLf & _
 "12.10" & vbTab & "Marsiglia" & vbTab & "12.30" & vbTab & "Roma"

        ' Il testo è scritto in quattro colonne.
        ' La prima colonna inizia alla posizione 0.
        ' I punti d'inizio della II, III e IV colonna sono fissati da queste
tabulazioni:
        Dim Tabulazioni As Single() = {80, 100, 80}

        ' Formattazione del testo su quattro colonne, secondo le tabulazioni già
impostate:
        Dim Formato As New StringFormat
        Formato.SetTabStops(0, Tabulazioni)

        ' Definisce l'area rettangolare in cui è visualizzato il testo
        ' e scrive il testo in blu con la formattazione già impostata a 4
colonne:
        Dim AreaTesto As New Rectangle(20, 20, 400, 150)
        e.Graphics.DrawString(Testo, Carattere, Brushes.Blue, AreaTesto, Formato)

        ' disegna due rettangoli per dividere il testo in due tabelle:
        Dim Tabella1 As New Rectangle(10, 10, 180, 150)
        Dim Tabella2 As New Rectangle(195, 10, 180, 150)
        e.Graphics.DrawRectangle(Pens.Red, Tabella1)
        e.Graphics.DrawRectangle(Pens.Green, Tabella2)

    End Sub
```

```
    Private Sub Button1_Click() Handles Button1.Click

        ' Il comando PrintDocument1.Print è disattivato per risparmiare carta,
sono invece attivi i due comandi per
        ' l'anteprima di stampa:

        ' PrintDocument1.Print()
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione, con l'anteprima di stampa:



170: La struttura e.PageBounds.

Un foglio di carta di formato A4 (è il formato dei fogli per le fotocopie) misura 841 punti in larghezza e 1190 punti in altezza.
L'area totale utilizzabile da una stampante misura 827 punti in larghezza e 1.169 punti in altezza.
I limiti di quest'area sono memorizzati nella struttura **e.PageBounds**, che ha questi parametri:

- e.PageBounds.X = 0 (limite a sinistra);
- e.PageBounds.Y = 0 (limite superiore);
- e.PageBounds.Width = 827 (larghezza);
- e.PageBounds.Height = 1169 (altezza).

I limiti della parte del foglio effettivamente utilizzabile per la stampa, cioè di quella parte che si trova all'interno dei margini di stampa, sono invece memorizzati nella struttura **e.MarginBounds.**
Considerato che i margini di stampa predefiniti di VB sono di 100 pixel per ogni lato, la struttura **e.MarginBounds** ha questi parametri:

- e.PageBounds.X = 100 (limite a sinistra);
- e.PageBounds.Y = 100 (limite superiore);
- e.PageBounds.Width = 627 (larghezza);
- e.PageBounds.Height = 969 (altezza).

Nei due prossimi esercizi vedremo due esempi di utilizzo di **e.PageBounds** e **e.MarginBounds**.

Esercizio 112: Stampare un disegno al centro di un foglio.

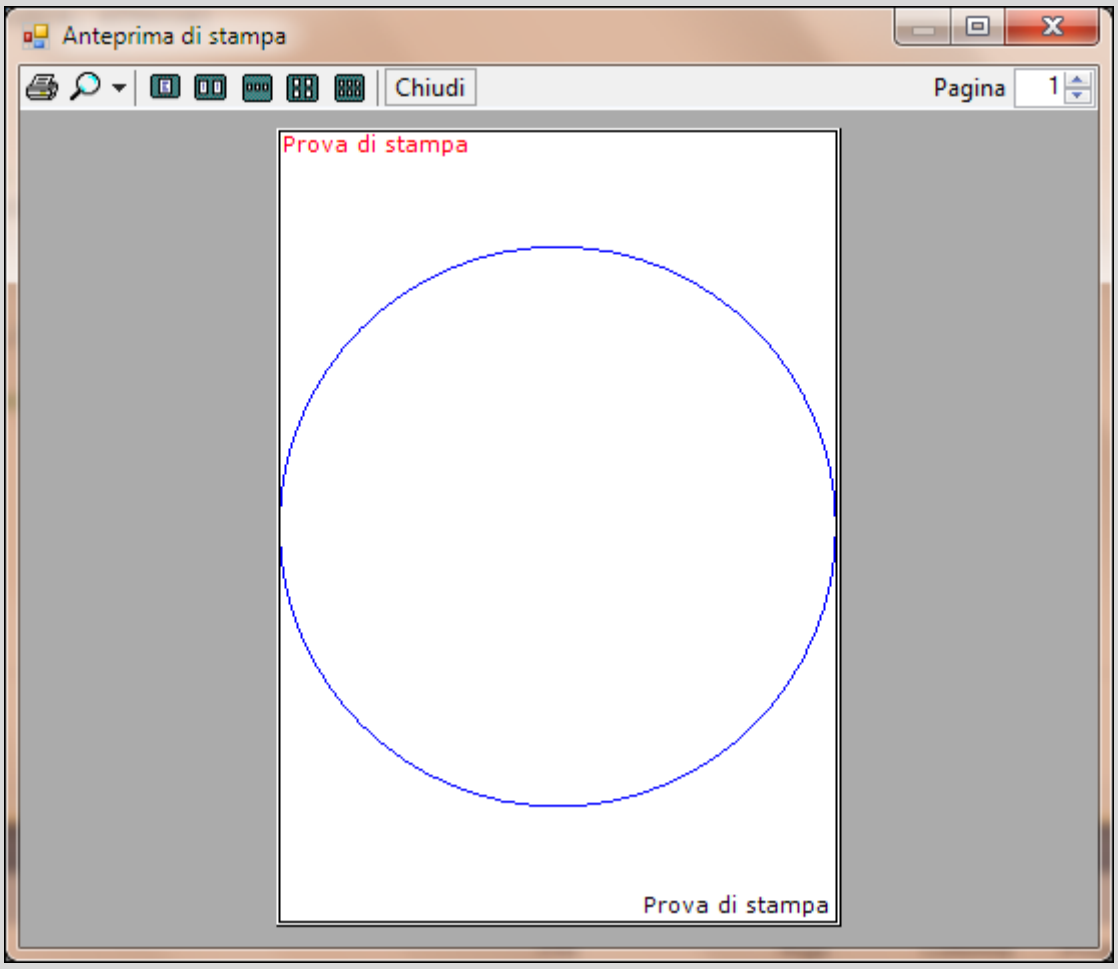
In questo programma vediamo l'uso della struttura **e.PageBounds** per stampare una circonferenza al centro di un foglio di carta.
Il programma dà per scontato che il foglio sia di formato A4, con un'area utilizzaible per la stampa di 827 punti in larghezza e 1.169 punti in altezza.

```
Public Class Form1

    Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        Dim Testo As String = "Prova di stampa"
        Dim Carattere As New Font("Verdana", 24, FontStyle.Regular,
GraphicsUnit.Point)

        ' Misura le dimensioni dell'area occupata dal testo:
        Dim AreaTesto As New SizeF(e.Graphics.MeasureString(Testo, Carattere))

        ' Crea due variabili per collocare nel foglio il testo "Prova di stampa"
e la circonferenza
        Dim Sinistra As Integer = 0
        Dim Altezza As Integer = 0

        ' Scrivi il testo in alto a sinistra:
        e.Graphics.DrawString(Testo, Carattere, Brushes.Red, Sinistra, Altezza)

        ' Scrivi il testo in basso a destra:
        Sinistra = e.PageBounds.Width - AreaTesto.Width
        Altezza = e.PageBounds.Height - AreaTesto.Height
        e.Graphics.DrawString(Testo, Carattere, Brushes.Black, Sinistra, Altezza)

        ' Centra nella pagina una circonferenza grande come il foglio,
        ' cioè di 827 punti di diametro:
        Sinistra = (e.PageBounds.Width - 827) / 2
        Altezza = (e.PageBounds.Height - 827) / 2
        ' Disegna nella pagina un cerchio di 827 punti di diametro:
```

```
        e.Graphics.DrawEllipse(Pens.Blue, Sinistra, Altezza, 827, 827)

    End Sub
```

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' Il comando PrintDocument1.Print è disattivato per risparmiare carta,
sono invece attivi i due comandi per l'anteprima di stampa:

        ' PrintDocument1.Print()
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione, con l'anteprima di stampa:



171: Stampare un testo.

Concludiamo questo capitolo dedicato alla stampa con la realizzazione di un elaboratore di testi in cui è possibile vedere all'opera:

- alcune proprietà e azioni del controllo **TextBox** (casella di testo);
- l'oggetto **ClipBoard**;
- alcune modalità di apertura e salvataggio di file;
- alcune modalità di formattazione e di stampa di un testo.

Le **Proprietà** del TextBox utilizzate nel programma sono queste:

TextBox1.Dock	*Imposta la posizione e le dimensioni del TextBox. Nel programma questa proprietà è impostata = Fill, affinché il TexBox occupi tutta l'area disponibile nel Form1.*
TextBox1.Multiline	*Dispone il testo su più linee.*
TextBox1.BackColor	*Memorizza il colore dello sfondo o imposta un nuovo colore per lo sfondo.*
TextBox1.ForeColor	*Memorizza il colore dei caratteri o imposta un nuovo colore per i caratteri.*
TextBox1.Font	*Memorizza il tipo di caratteri in uso o imposta un nuovo tipo di caratteri.*
TextBox1.ScrollBars	*Imposta se e quali barre di scorrimento saranno visualizzate. Nel programma questa proprietà è impostata = Vertical .*
TextBox1.Modified	*Determina se il testo è stato modificato o no. Il valore di questa proprietà può essere solo True o False.*
TextBox1.Text	*Memorizza il testo del TextBox o imposta un nuovo testo.*
TextBox1.SelectedText	*Memorizza il testo o la parte del testo selezionata dall'utente.*

Le **Azioni** del TextBox utilizzate nel programma sono queste:

TextBox1.Clear()	*Cancella il contenuto del TextBox.*

TextBox1.Paste()	*Incolla nel TextBox il contenuto della ClipBoard.*
TextBox1.Undo()	*Annulla l'ultima operazione effettuata dall'utente.*

L'oggetto **ClipBoard** utilizzato nel programma è quella parte della memoria temporanea del computer nella quale sono immagazzinati i dati utilizzati nelle operazioni di tipo copia/incolla.
E' una parte di memoria che tutti i programmi installati nel computer utilizzano in comune, quando l'utente desidera passare dei dati da un programma ad un altro.
In questo programma, il testo selezionato dall'utente è inviato alla ClipBoard se l'utente preme i pulsanti **Copia** o **Taglia**, ed è invece recuperato dalla ClipBoard se l'utente preme il pulsante **Incolla**.

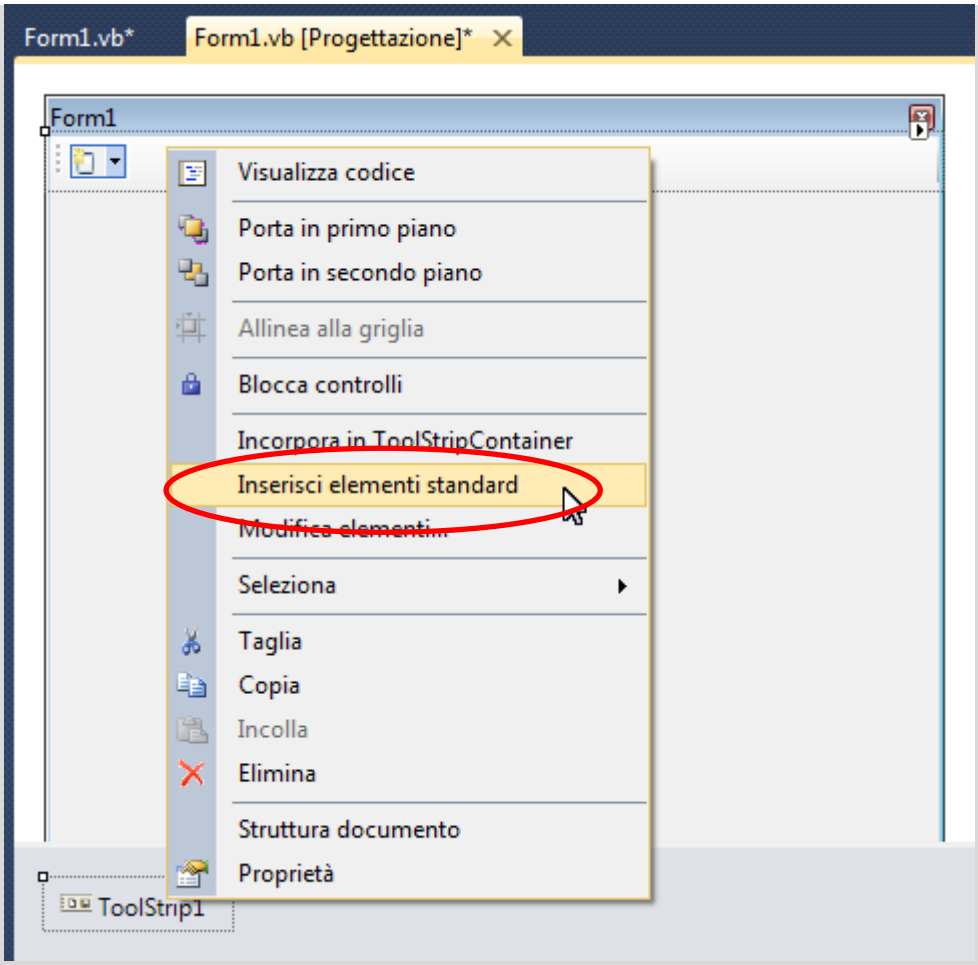
Compaiono nel programma anche i comandi per aprire o salvare un file di testo; questi comandi verranno illustrati più avanti, nel Capitolo 34: GESTIONE DI FILE ESTERNI AL PROGRAMMA. 828.

Esercizio 113: Primi testi.

Apriamo un nuovo progetto. Gli diamo il nome "Primi testi".
Proprietà del Form1:

- **FormBorderStyle** = FixedToolWindow
- **Icon**: scegliamo l'icona PrimiTesti, che si trova nella cartella Documenti / A scuola con VB 2010 / Immagini / Icone.
- **Size** = 450; 600
- **StartPosition** = CenterScreen
- **Text** = Primi testi

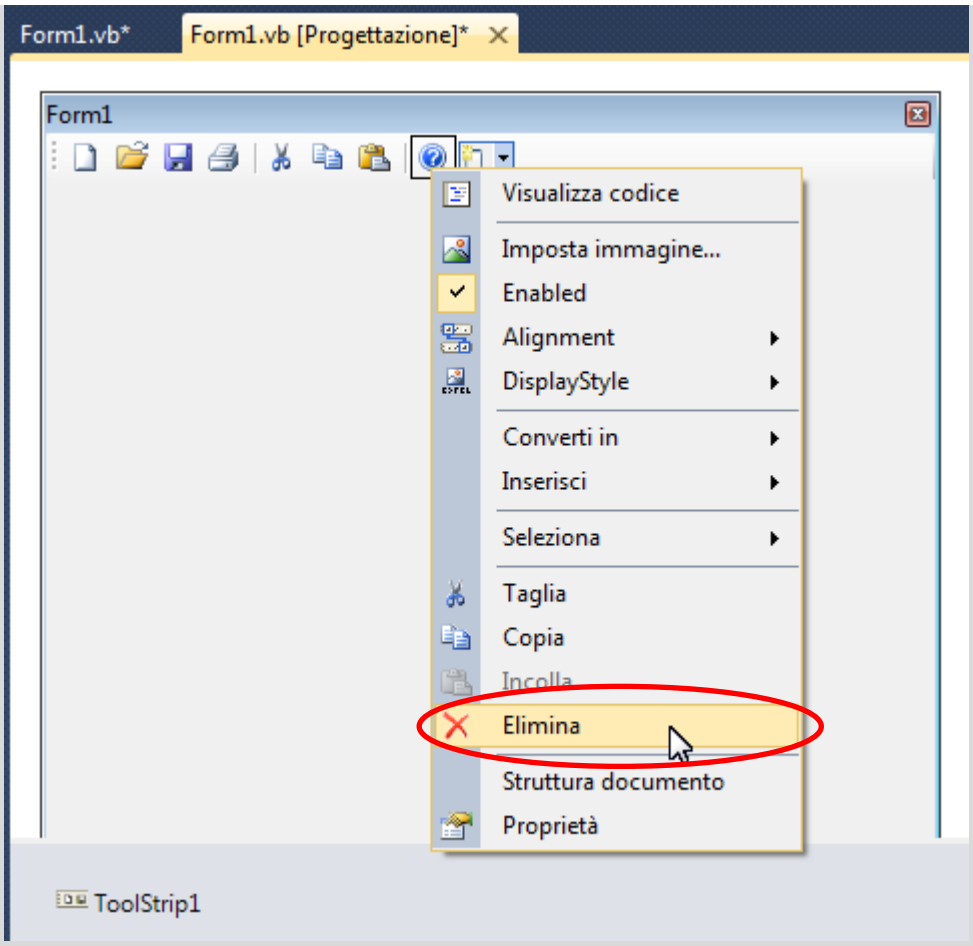
Ora inseriamo nel Form1 un controllo ToolStrip. Notiamo che questo va a collocarsi sotto la barra del titolo del Form1.
Facciamo un *clic* con il tasto destro del mouse sul controllo ToolStrip e nel menu che si apre facciamo un *clic* su **Inserisci elementi standard**:



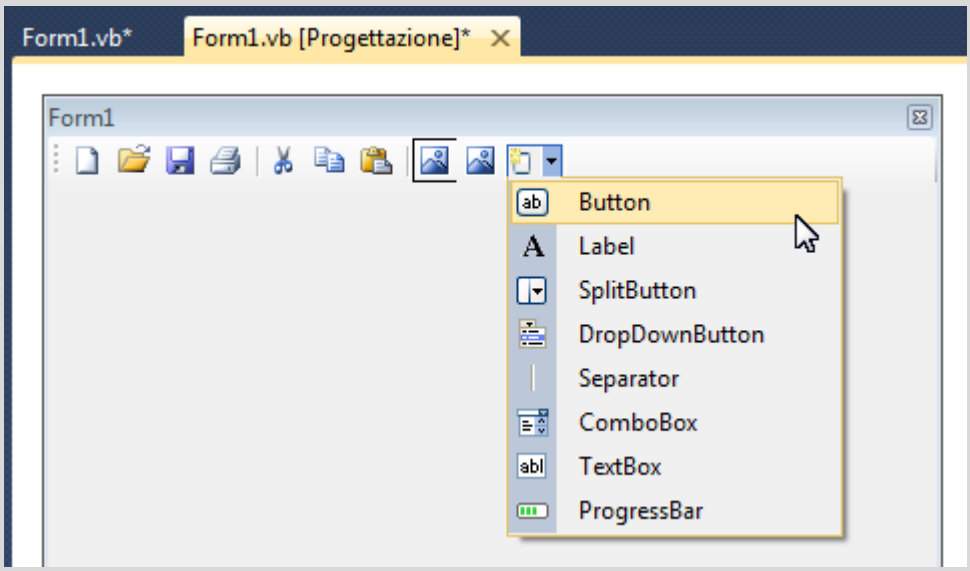
Questa opzione inserisce nel controllo ToolStrip un gruppo di pulsanti di uso comune, con le relative icone:

- Apri
- Nuovo
- Salva
- Stampa
- Taglia
- Copia
- Incolla

L'ultimo pulsante della serie è il pulsante Informazioni, che in questo programma è superfluo: lo eliminiamo facendo un *clic* sul pulsante con il tasto destro del mouse, poi un *clic* su **Elimina**, oppure sul tasto CANC o DEL.



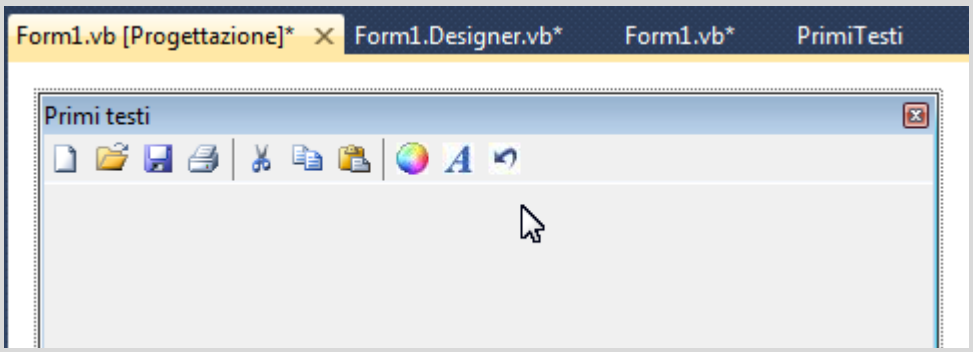
Al suo posto, aggiungiamo al controllo ToolStrip tre pulsanti Button, facendo ogni volta un *clic* sul pulsante con la freccia nera in basso:



Ora, facendo un *clic* con il tasto destro del mouse su ognuno di essi, e un *clic* su **Imposta immagine…**, assegniamo ai tre nuovi pulsanti, rispettivamente, le immagini

- Colori
- Font
- Annulla

che si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Icone**.
Ecco come si presenta ora il controllo ToolStrip, finito:



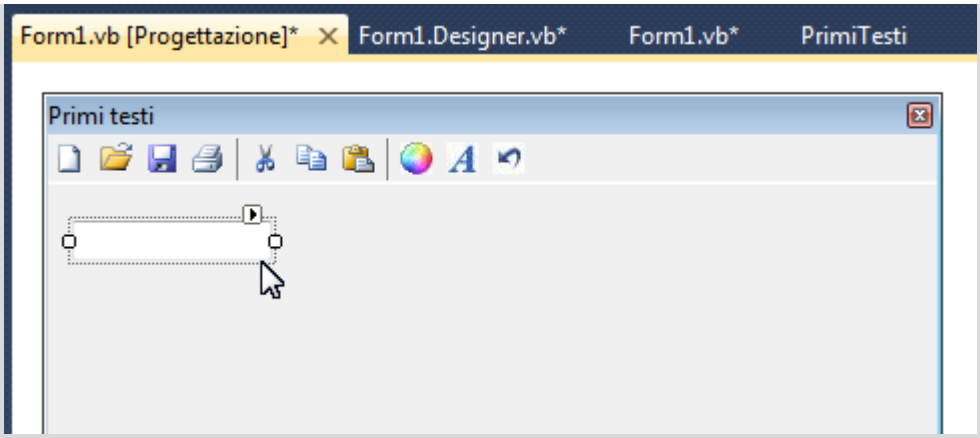
Facciamo un *clic* sul primo di questi pulsanti e, nella Finestra delle Proprietà, modifichiamo la proprietà **Name**: invece di **NuovoToolStripButton** scriviamo semplicemente **Nuovo**. Questo renderà il testo del listato più facilmente leggibile.
Procediamo in questo modo modificando la proprietà **Name** degli altri pulsanti:

- **Apri**
- **Salva**
- **Stampa**
- **Taglia**
- **Copia**
- **Incolla**
- **Colori**
- **Font**
- **Annulla**

Notiamo anche la proprietà ToolTipText di questi pulsanti: i pulsanti creati automaticamente hanno questa proprietà già impostata.
Per gli ultimi tre pulsanti modifichiamo la proprietà **ToolTip** in questo modo:

- **Scegli un colore**
- **Scegli un carattere**
- **Annulla**

Ora inseriamo nel Form1 l'oggetto principale di questo programma: il controllo **TextBox1**, nel quale l'utente scriverà i suoi testi.



Le proprietà di questo TextBox1 sono impostate dal codice, per cui non resta da fare che copiare il listato e incollarlo nella Finestra del Codice.
Il listato è ampiamente commentato, in particolare per quanto riguarda la funzione di stampa, per facilitarne la comprensione.

```
Public Class Form1
    Dim TestoDaStampare As String
```

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        ' Imposta la proprietà del TextBox1:
        With TextBox1
            .Dock = DockStyle.Fill
            .Multiline = True
            .BackColor = Color.White
            .ForeColor = Color.Black
            .Font = New Font("Arial", 18)
            .ScrollBars = ScrollBars.Vertical
        End With

        OpenFileDialog1.Filter = "File di testo|*.txt"
        SaveFileDialog1.FileName = "Scrivi il titolo"
        SaveFileDialog1.Filter = "Testo|*.txt"

    End Sub
```

```
    Private Sub Nuovo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Nuovo.Click

        ' Prima di iniziare un nuovo testo verifica se è stato salvato il testo
eventualmente in uso:
        If OpzioneSalvataggio() = True Then TextBox1.Clear()

    End Sub
```

```
    Private Sub Apri_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Apri.Click

        ' Prima di aprire un nuovo file verifica se è stato salvato il file
eventualmente in uso:
        If OpzioneSalvataggio() = True Then
```

```
            OpenFileDialog1.FileName = ""
            If OpenFileDialog1.ShowDialog = DialogResult.OK Then
                ' cancella il contenuto della casella di testo
                TextBox1.Clear()
                ' e visualizza il novo testo:
                TextBox1.Text =
My.Computer.FileSystem.ReadAllText(OpenFileDialog1.FileName)

            End If
        End If

    End Sub
```

```
    Private Sub Salva_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Salva.Click

        If SaveFileDialog1.ShowDialog = DialogResult.OK Then
            My.Computer.FileSystem.WriteAllText(SaveFileDialog1.FileName,
TextBox1.Text, False)
        TextBox1.Modified = False
        End If

    End Sub
```

```
    Private Sub Taglia_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Taglia.Click

        If TextBox1.SelectedText <> "" Then
            Clipboard.SetText(TextBox1.SelectedText)
        End If
        TextBox1.SelectedText = ""

    End Sub
```

```
    Private Sub Copia_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Copia.Click

        If TextBox1.SelectedText <> "" Then
            Clipboard.SetText(TextBox1.SelectedText)
        End If

    End Sub
```

```
    Private Sub Incolla_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Incolla.Click

        TextBox1.Paste()

    End Sub
```

```
    Private Sub Colori_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Colori.Click

        ' Apre la finestra ColorDialog sul colore attualmente in uso
        ColorDialog1.Color = TextBox1.ForeColor

        ' imposta il nuovo colore del testo:
```

```
        If ColorDialog1.ShowDialog = DialogResult.OK Then
            TextBox1.ForeColor = ColorDialog1.Color
        End If

    End Sub
```

```
    Private Sub Font_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Font.Click

        ' Apre la finestra di dialogo sul font attualmente in uso:
        FontDialog1.Font = TextBox1.Font

        ' imposta il nuovo font:
        If FontDialog1.ShowDialog = DialogResult.OK Then
            TextBox1.Font = FontDialog1.Font
        End If

    End Sub
```

```
    Private Sub Annulla_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Annulla.Click

        TextBox1.Undo()

    End Sub
```

```
    Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing

        ' verifica e chiede l'eventuale salvataggio del testo scritto nel
TextBox1
        ' se il ritorno è = False significa che è stato premuto il tasto
"Annulla"
        ' in questo caso il programma non è chiuso

        If OpzioneSalvataggio() = False Then
            e.Cancel = True
        End If

    End Sub
```

```
    Function OpzioneSalvataggio() As Boolean

        ' Questa funzione, attivata dai pulsanti Nuovo e Apri e alla chiusura del
programma,
        ' controlla se il testo è stato modificato e non ancora salvato:
        ' in questo caso verifica se l'utente vuole salvarlo o no:

        If TextBox1.Modified = True Then

            Select Case MsgBox("Il testo è stato modificato e non ancora
salvato." & vbCrLf & "Vuoi salvarlo?", MsgBoxStyle.YesNoCancel +
MsgBoxStyle.Question, "Primi testi")

                Case MsgBoxResult.Yes
                    Salva.PerformClick() ' salva il testo e procede
                    Return True
                Case MsgBoxResult.No
                    Return True ' procede senza salvare il testo
```

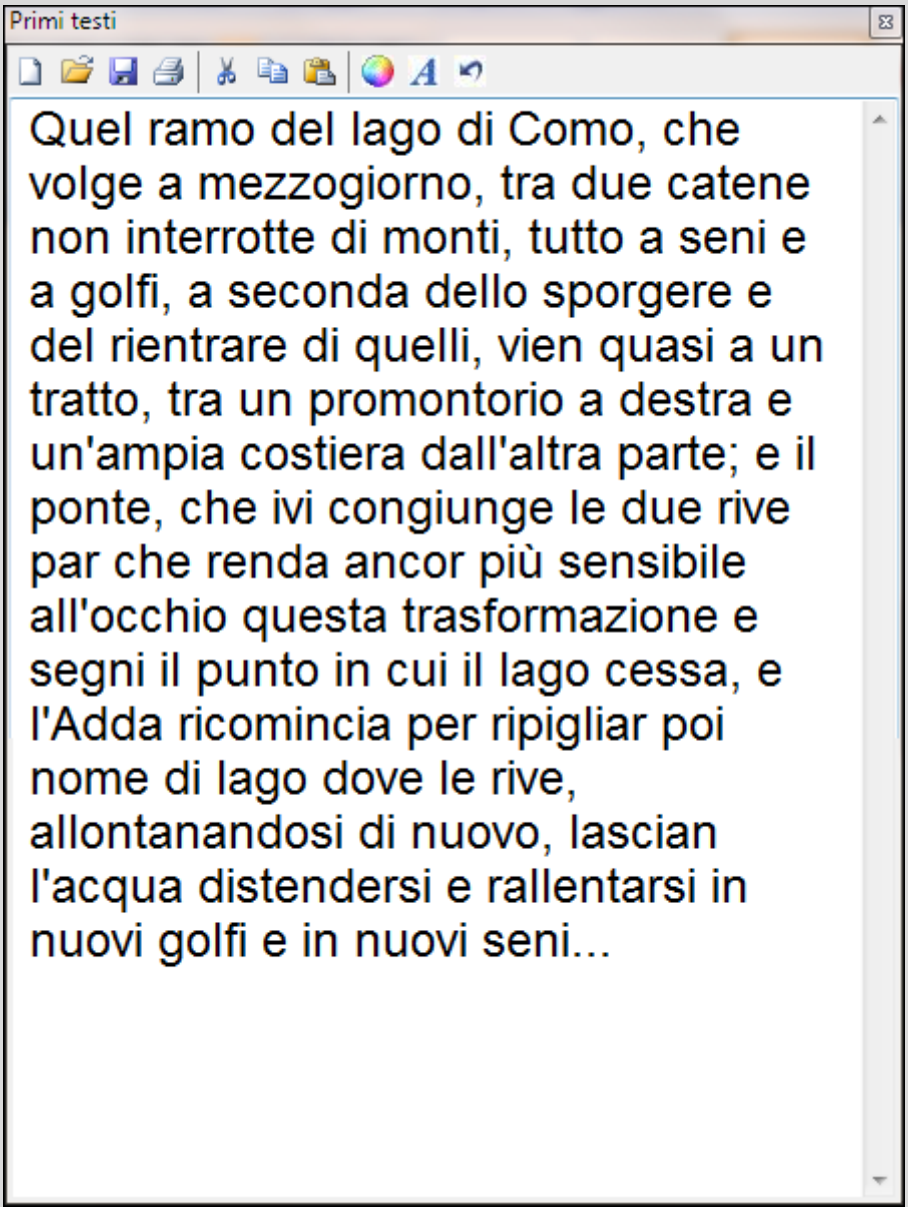
```
                Case Else ' annulla l'operazione
                    Return False
            End Select

        Else ' procede: il testo è già stato salvato e non è stato modificato:
            Return True
        End If

    End Function
```

```
    Private Sub Stampa_Click(sender As System.Object, e As System.EventArgs)
Handles Stampa.Click

        TestoDaStampare = TextBox1.Text
        ' associa la previsione di stampa al componente PrintDocument1
        ' e causa l'evento PrintDocument1.PrintPage

        PrintPreviewDialog1.Document = PrintDocument1
        ' mostra l'anteprima di stampa
        PrintPreviewDialog1.ShowDialog()

    End Sub
```

```
    Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage


        Dim CaratterixPagina As Integer
        Dim LineexPagina As Integer
        ' misura quanti caratteri, nel font scelto dall'utente, possono essere
contenuti entro i margini della pagina,
        ' quindi invia questo numero di caratteri alla prima pagina,
        ' poi li cancella dal TestoDaStampare e invia alla pagina successiva i
caratteri restanti,
        ' e così via, sino all'esaurimento dei caratteri del TestoDaStampare.
        e.Graphics.MeasureString(TestoDaStampare, TextBox1.Font,
e.MarginBounds.Size, StringFormat.GenericTypographic, CaratterixPagina,
LineexPagina)

        ' disegna il testo entro i margini predefiniti della pagina.
        e.Graphics.DrawString(TestoDaStampare, TextBox1.Font, New
SolidBrush(TextBox1.ForeColor), e.MarginBounds)

        ' elimina la parte di testo già inviata alla stampa:
        TestoDaStampare = TestoDaStampare.Remove(0, CaratterixPagina)

        ' se il testo non è completamente stampato perché è più lungo di una
pagina,
        ' imposta la proprietà e.HasMorePages = True
        ' Questa impostazione causa un altro evento PrintDocument1.Print e dunque
        ' la ripetizione di questa procedura sino all'esaurimento delle pagine da
stampare.

        If TestoDaStampare.Length > 0 Then
            e.HasMorePages = True
        Else
            e.HasMorePages = False
        End If

    End Sub
```

```
End Class
```

Ecco un'immagine del programma in esecuzione:



PARTE V: L'OGGETTO MY.

VB mette a disposizione del programmatore un oggetto particolare, denominato My (= mio), che semplifica la gestione di molte azioni nei programmi.
Questo oggetto si articola in sette settori, ognuno dei quali riguarda la gestione di elementi dell'applicazione o di funzioni del sistema operativo del computer:

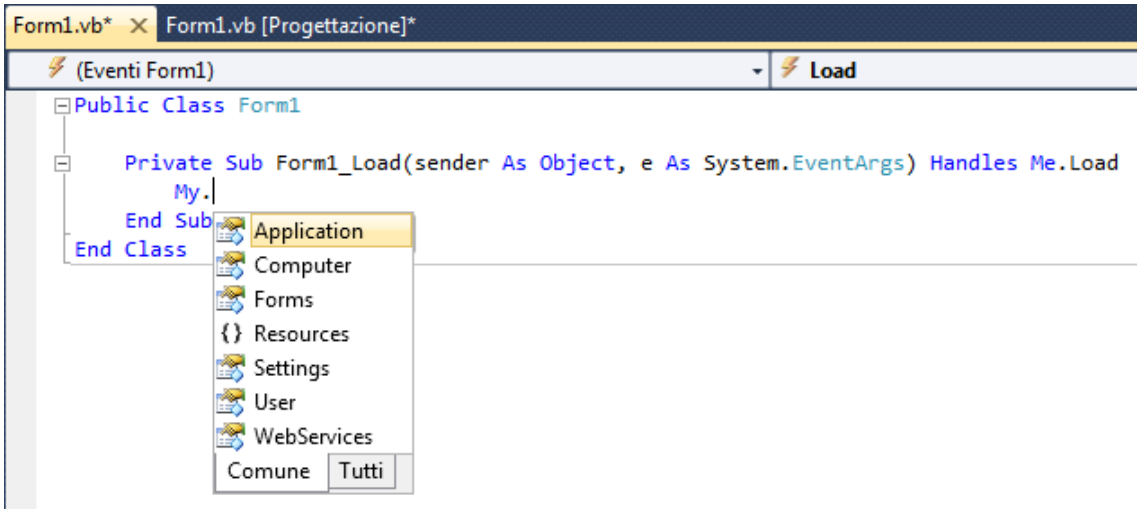


Figura 224: I settori dell'oggetto My.

Ci occuperemo innanzitutto del settore **My.Computer**, che fornisce strumenti per gestire alcune funzioni del computer:

- My.Computer.**Audio**, per la gestione dei suoni;
- My.Computer.**Keyboard**, per la gestione della tastiera;
- My.Computer.**Mouse**, per la gestione del mouse;
- My.Computer. **Filesystem**, per la gestione dei file (salvataggio e lettura).

Ci occuperemo infine di

- **My.Application**, che fornisce strumenti per gestire alcune proprietà della applicazione in progettazione;
- **My.Resources**, che fornisce strumenti per accedere a immagini, testi, file audio inseriti nella applicazione.

Capitolo 31: MY.COMPUTER.AUDIO.

My.Computer.Audio si occupa della gestione dei file sonori all'interno di un programma. Di uso molto semplice, consente di eseguire un file audio con una sola riga di codice.
Apriamo un nuovo progetto e impostiamo la procedura relativa all'evento Form1_Load:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

    End Sub

End Class
```

Nella riga intermedia, ora vuota, attiviamo il settore audio del computer e notiamo i suoi tre comandi

- **Play**,
- **PlaySystemSound** e
- **Stop**,

che analizzeremo di seguito:

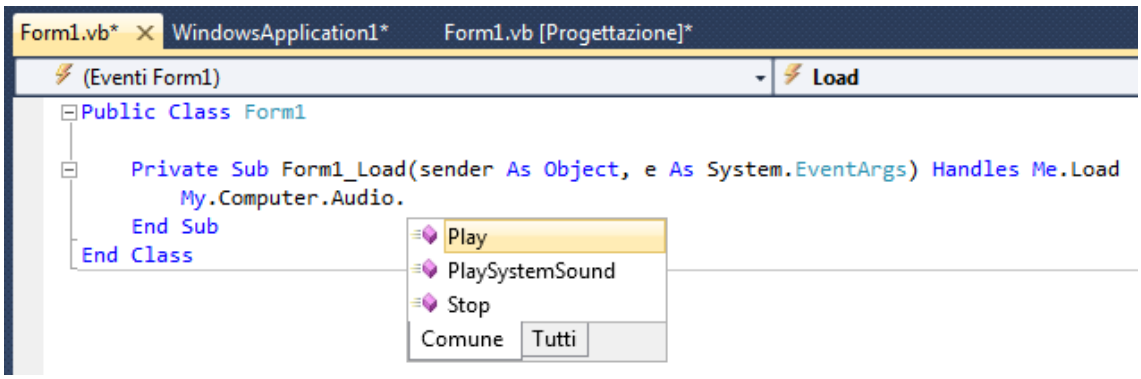


Figura 225: I comandi del settore My.Computer.Audio.

My.Computer.Audio.Stop

My.Computer.Audio.Stop arresta l'esecuzione di un file audio.
Ecco un esempio:

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
Button1.Click

    My.Computer.Audio.Stop()

End Sub
```

My.Computer. Audio.PlaySystemSound.

My.Computer.Audio.PlaySystemSound riproduce uno dei file audio del sistema operativo.
In Windows 7 sono disponibili cinque file audio di sistema, utilizzabili come semplici risposte sonori alle azioni dell'utente del programma:

- **Asterisk**
- **Beep**
- **Exclamation**
- **Hand**
- **Question**

Ecco un esempio del loro utilizzo in un programma VB:

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
Button1.Click

    My.Computer.Audio.PlaySystemSound(Media.SystemSounds.Exclamation)

End Sub
```

My.Computer.Audio.Play.

My.Computer.Audio.Play riproduce un file audio presente nel computer o nelle risorse del programma.
Ecco un esempio: il listato seguente riproduce, all'avvio del programma, il file "gallo.wav", che si trova nella cartella **Documenti \ A scuola con VB \ Audio**.

```
Public Class Form1
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Dim File As String =
My.Computer.FileSystem.SpecialDirectories.MyDocuments & ("\A scuola con VB
2010\Audio\gallo.wav")
        My.Computer.Audio.Play(File, AudioPlayMode.Background)

    End Sub

End Class
```

Le modalità di riproduzione del file sono tre:

- **asincrona:** con la modalità asincrona (**AudioPlayMode.Background**) il file è eseguito in sottofondo, mentre il programma procede normalmente nella sua esecuzione.

```
        My.Computer.Audio.Play(File, AudioPlayMode.Background)
```

- **continua:** con la modalità continua (**AudioPlayMode.BackgroundLoop**) il file è eseguito in sottofondo ed è ripetuto in continuazione.

```
        My.Computer.Audio.Play(File, AudioPlayMode.BackgroundLoop)
```

- **sincrona:** con la modalità asincrona (**AudioPlayMode.WaitToComplete**) mentre il file audio è in riproduzione il programma si arresta, per riprendere il suo corso normale alla fine della esecuzione del file audio.

```
        My.Computer.Audio.Play(File, AudioPlayMode.WaitToComplete)
```

I file audio possono essere collocati nella stessa cartella in cui si trova il programma. Avremo allora un comando di questo tipo:

```
        My.Computer.Audio.Play(My.Application.Info.DirectoryPath & "/gallo.wav",
AudioPlayMode.Background)
```

Oppure (**opzione preferibile**), i file audio possono essere collocati nelle risorse del programma, come nell'esempio che segue.

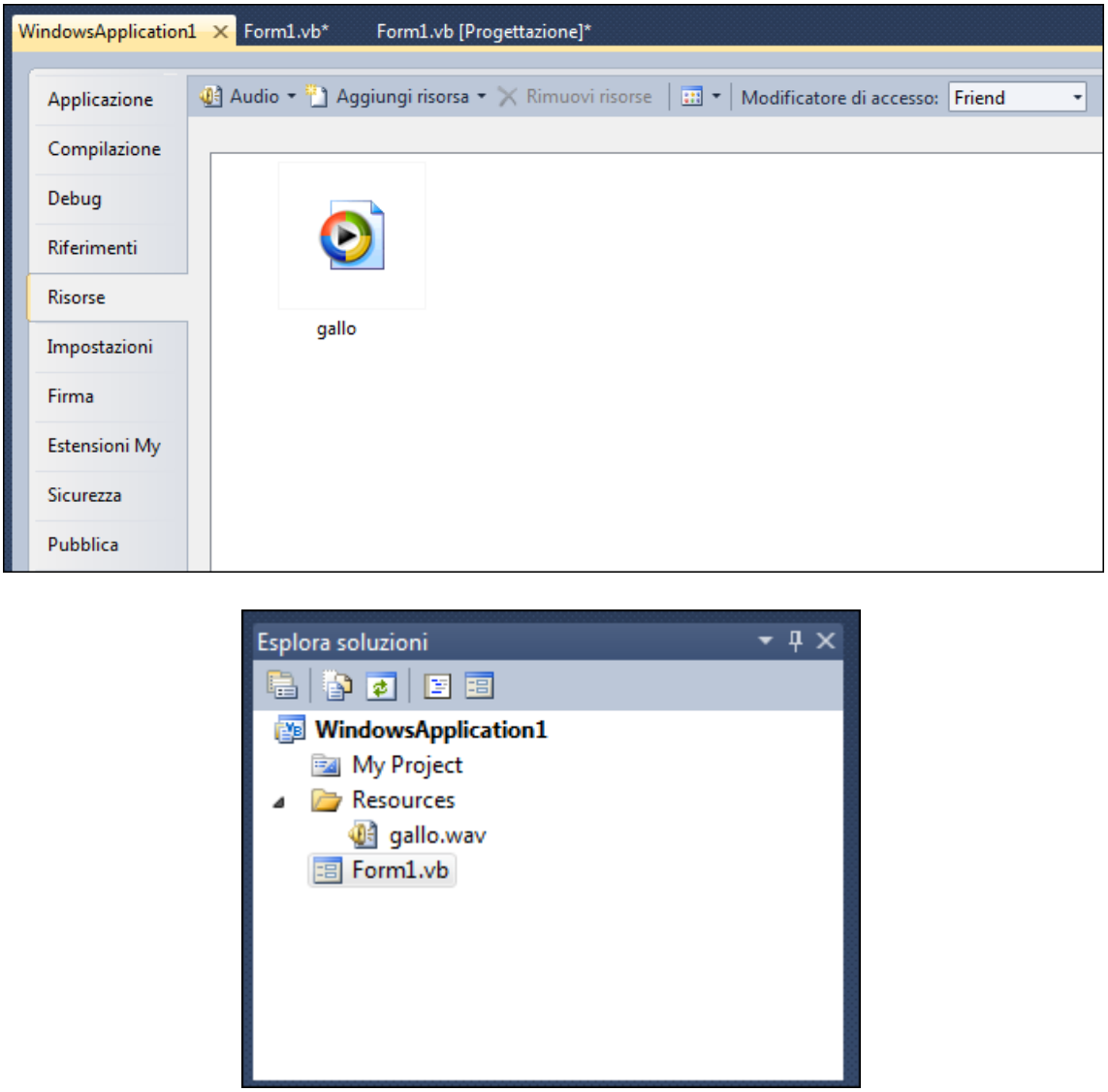


Figura 226: Inserimento di un file audio nelle risorse del programma.

In questo caso il comando per la loro esecuzione è:

```
My.Computer.Audio.Play(My.Resources.gallo, AudioPlayMode.Background)
```

Un programma VB può eseguire direttamente solo file in formato **.wav**.
Per utilizzare file audio di altro tipo (.mid., mp3, .ram, .wma) in un programma VB è dunque necessario convertirli preventivamente nel formato .wav, utilizzando un programma di conversione dei file audio.
I file .wav contengono suoni registrati e digitalizzati, salvati in modo da riprodurre il più fedelmente possibile il suono originale.
Possono contenere musiche, voci, effetti sonori; anche se occupano nella memoria del computer spazi relativamente consistenti, si tratta di file audio molto diffusi.

E' possibile crearli autonomamente: per fare questo è sufficiente collegare al computer un microfono per registrare suoni, voci o rumori, oppure un riproduttore di musiche come un lettore di CD audio.

Per creare file .wav nei sistemi operativi Windows 98 e Windows XP è disponibile un registratore di suoni duttile ed efficace: ne illustriamo il funzionamento in coda a questo capitolo.
Il registratore di suoni inserito nei sistemi operativi Windows Vista e Windows 7[82] è uno strumento molto meno duttile ed efficace: i file audio possono essere salvati solo nel formato .wma e per essere utilizzati in un programma VB devono essere convertiti nel formato .wav.

Figura 227: Il registratore di suoni in Windows Vista e Windows 7.

172: La classe Media.SoundPlayer.

Per la riproduzione di suoni il programmatore può disporre anche di un'apposita classe, esistente al di fuori dell'oggetto My: si tratta della classe **Media.SoundPlayer**, che offre al programmatore le stesse possibilità offerte da My.Computer.Audio.
Eccone un esempio:

```
Public Class Form1

    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs)
Handles MyBase.Load

        ' crea un nuovo oggetto della classe Media.SoundPlayer:
        Dim RiproduttoreSuoni As New Media.SoundPlayer

        ' carica nel nuovo oggetto il file audio esistente nelle risorse del
programma (stream = flusso di dati)
        RiproduttoreSuoni.Stream = My.Resources.gallo

        ' e lo riproduce in modalità asincrona:
        RiproduttoreSuoni.Play()
```

[82] Vi si accede dal menu **Tutti i programmi** / **Accessori**.

```
    End Sub

End Class
```

Oppure, in modo più diretto:

```
Public Class Form1

    Private Sub Form1_Load(sender As System.Object, e As System.EventArgs)
Handles MyBase.Load

        ' crea un nuovo oggetto della classe Media.SoundPlayer:
        Dim RiproduttoreSuoni As New Media.SoundPlayer(My.Resources.gallo)

        ' e lo riproduce in modalità asincrona:
        RiproduttoreSuoni.Play()

    End Sub

End Class
```

La classe **Media.SoundPlayer** offre al programmatore, una volta che è stato creato un oggetto di tipo Media.SoundPlayer, la possibilità di usarlo nel programma richiamando semplicemente il nome di questo oggetto, senza ripetere il nome del file audio.
Ne vedremo un esempio nel prossimo esercizio.

Esercizio 114: Riproduzione di suoni con la classe Media.Player.

Questo programma carica nella memoria un file audio e lo riproduce in modalità varie. I comandi di esecuzione sono collegati a quattro pulsanti Button e dunque si trovano in quattro procedure diverse, ma il file audio viene caricato in memoria solo una volta, all'avvio del programma.
Apriamo un nuovo progetto e inseriamo nel form quattro pulsanti Button1.
Inseriamo nelle risorse del programma il file **gallo.wav** che si trova nella cartella **Documenti \ A scuola con VB \ Audio**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea un nuovo oggetto della classe Media.SoundPlayer:
    Dim RiproduttoreSuoni As New Media.SoundPlayer(My.Resources.gallo)

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' riproduce il file in modalità asincrona:
        RiproduttoreSuoni.Play()
```

```
    End Sub

    Private Sub Button2_Click(sender As System.Object, e As System.EventArgs)
Handles Button2.Click

        ' riproduce il file in modalità continua:
        RiproduttoreSuoni.PlayLooping()

    End Sub

    Private Sub Button3_Click(sender As System.Object, e As System.EventArgs)
Handles Button3.Click
        ' riproduce il file in modalità sincrona:
        RiproduttoreSuoni.PlaySync()

    End Sub

    Private Sub Button4_Click(sender As System.Object, e As System.EventArgs)
Handles Button4.Click

        ' ferma l'esecuzione del file audio
        RiproduttoreSuoni.Stop()

    End Sub

End Class
```

173: Il registratore di suoni di Windows 98 e Windows XP.

Uno strumento utile per registrare i file nel formato .wav è il **Registratore di suoni** che si avvia dal menu Start / Programmi / Accessori / Svago / Registratore di suoni.

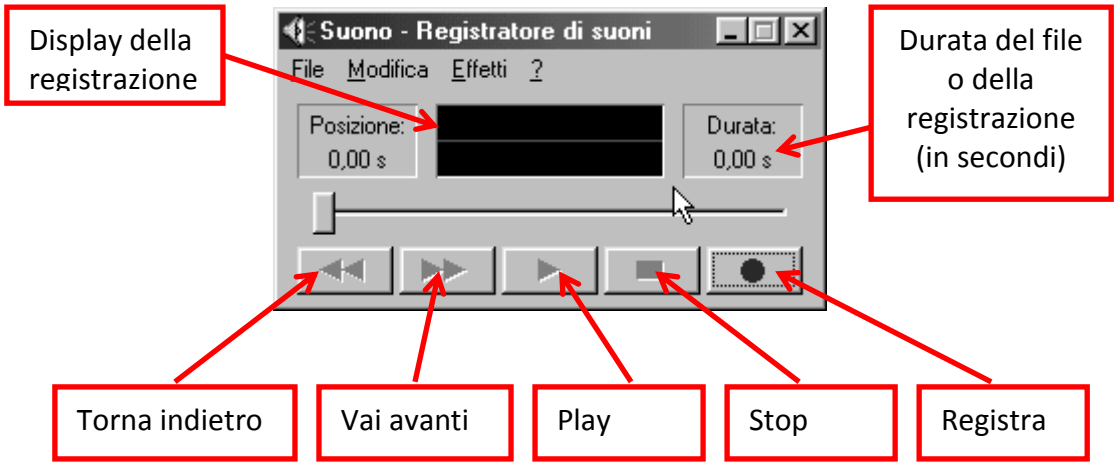


Figura 228: Avvio del registratore di suoni di Windows 98 e XP.

Il programma presenta un'interfaccia con cinque pulsanti, con le funzioni abituale di un lettore di CD musicali:

- il primo pulsante riporta l'esecuzione di un file wav all'inizio;
- il secondo pulsante porta l'esecuzione di un file alla fine;
- il terzo pulsante (**Play**) avvia l'esecuzione di un file;
- il quarto pulsante (**Stop**) blocca l'esecuzione o la registrazione di un file;
- l'ultimo pulsante avvia la registrazione di un nuovo file.

Prima di provare a registrare un nuovo file, apriamo il menu **File / Proprietà** (in alto a sinistra):

Figura 229: Il menu Proprietà del registratore di suoni.

Vediamo aprirsi la Finestra Proprietà del suono, che è possibile impostare in modi diversi, a seconda che si desiderino:

- una maggiore qualità di registrazione, oppure
- una registrazione di qualità ridotta che occupi minori risorse di memoria del computer.

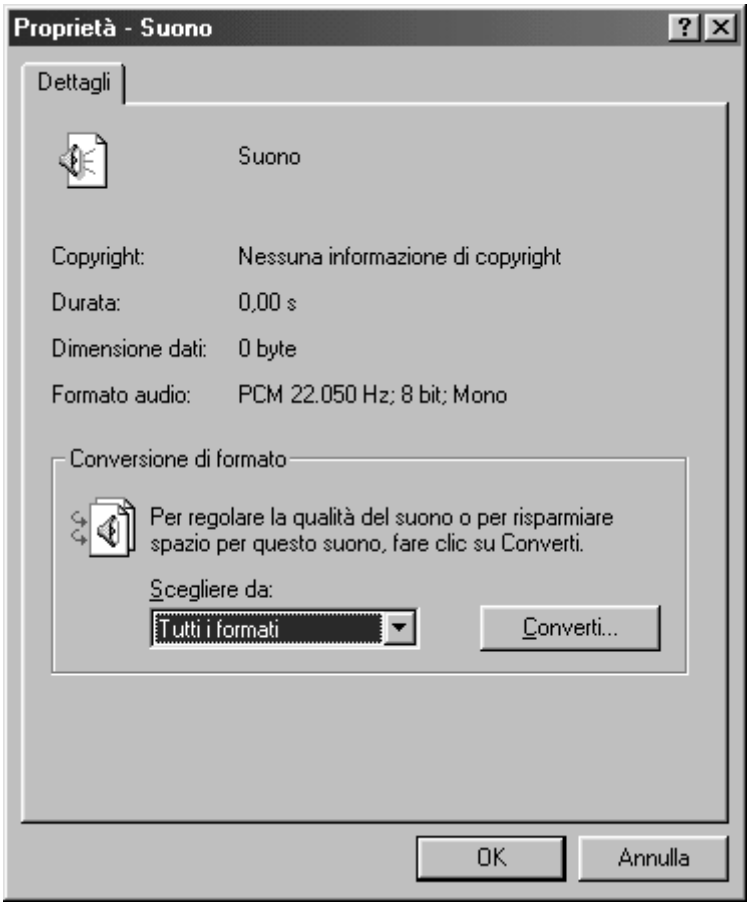


Figura 230: Il menu Proprietà del suono da registrare.

Facciamo un *clic* con il mouse sul pulsante **Converti** e nella finestra che si apre indichiamo qual è il formato di registrazione che desideriamo:

- un suono di qualità radio è monofonico e occupa 22 Kb di memoria per ogni secondo di registrazione;
- un suono di qualità CD è stereofonico e occupa 172 Kb di memoria per ogni secondo di registrazione;
- un suono di qualità telefono è monofonico e occupa solo 11 Kb di memoria per ogni secondo di registrazione.

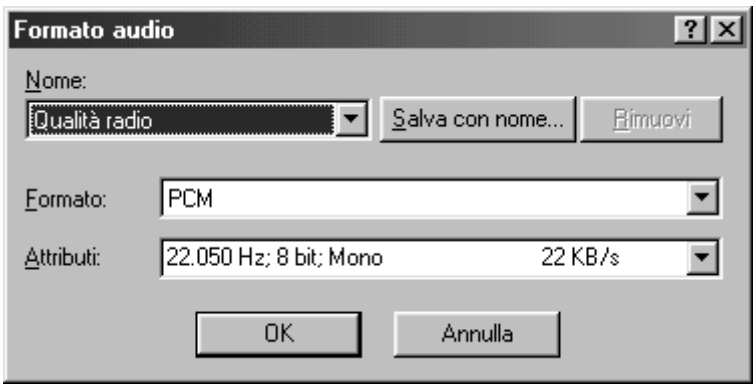


Figura 231: Impostazione delle qalità del file audio da registrare.

Definito il formato (supponiamo di avere mantenuto la qualità media - radio - che compare di *default* nella finestra del formato audio), possiamo ora dedicarci alla registrazione e alla creazione di un nuovo file di tipo .wav.
Dopo avere collegato un microfono al computer, per registrare un file è necessario premere l'ultimo pulsante che compare nell'interfaccia del Registratore di suoni, contraddistinto dal **pallino rosso**.
Mentre la registrazione è in corso è possibile controllarne l'andamento nel display rettangolare: se la forma d'onda che compare è troppo piatta il livello di registrazione è troppo basso, se la forma d'onda è troppo alta il volume è troppo alto e c'è il rischio di distorsioni nella registrazione. In entrambi ne risulterà un file audio di scarsa qualità.
Quando la registrazione è finita bisogna premere il pulsante **Stop**; a questo punto è possibile riascoltare il proprio lavoro premendo il pulsante **Play**.
Dopo avere registrato un file, è possibile apportarvi correzioni aprendo il menu Opzioni.

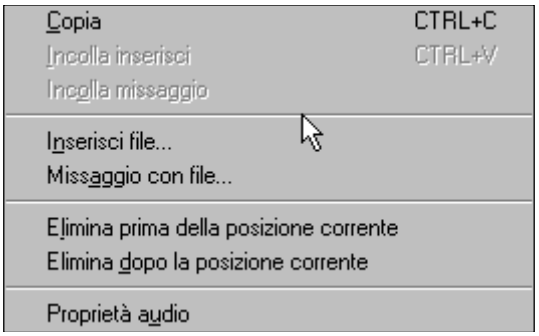


Figura 232: Il menu Opzioni del Registratore di suoni.

Tra le opzioni che compaiono, particolarmente utili sono queste due, che consentono di eliminare i tempi vuoti che si creano spesso all'inizio o alla fine di una registrazione:

- Elimina prima della posizione corrente
- Elimina dopo la posizione corrente.

Altre manipolazioni della registrazione sono possibili aprendo il menu **Effetti**:

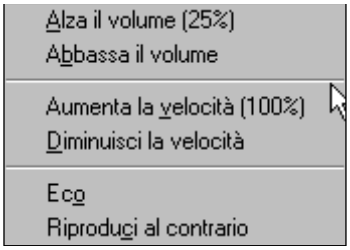


Figura 233: Il menu Effetti del Registratore di suoni.

Qui, in particolare, è possibile correggere il volume della registrazione, modificarne la velocità di riproduzione, aggiungere l'effetto dell'eco.
Terminata la fase di sistemazione della registrazione, è necessario salvare il file prodotto aprendo il menu **File\ Salva con nome...**
Nella finestra che si apre notiamo che compare di default, per il salvataggio del nuovo file, l'estensione *.wav:

Figura 234: Salvataggio di un file con il Registratore di suoni.

Capitolo 32: GESTIONE DELLA TASTIERA.

My.Computer.Keyboard consente di controllare la pressione di alcuni tasti sulla tastiera del computer.
Ecco un esempio: questo listato, all'avvio del programma, controlla se il tasto CAPS LOCK (blocca maiuscole) è premuto oppure no.

```
Public Class Form1

    Private Sub Form1_Load() Handles Me.Load

        If My.Computer.Keyboard.CapsLock = True Then
            MsgBox("Attenzione: il tasto BLOCCA MAIUSCOLE è premuto")
        Else
            MsgBox("Attenzione: il tasto BLOCCA MAIUSCOLE non è premuto")
        End If

    End Sub

End Class
```

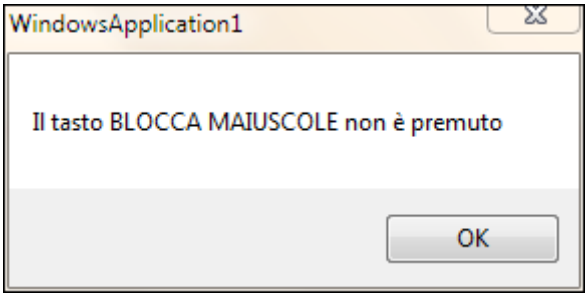


Figura 235: Controllo della pressione del tasto CAPS LOCK.

Con lo stesso sistema è possibile rilevare la pressione dei tasti ALT, CTRL, NUM LOCK e SHIFT:

```
If My.Computer.Keyboard.AltKeyDown = True Then…
If My.Computer.Keyboard.CtrlKeyDown = True Then…
If My.Computer.Keyboard.NumLock = True Then…
If My.Computer.Keyboard.ShiftKeyDown = True Then…
```

174: Gli eventi KeyDown e KeyPress.

Le indicazioni fornite da **My.Computer.Keyboard** consentono di controllare in modo diretto la pressione di alcuni tasti, ma sono limitate a pochi tasti.
In un programma si può presentare invece l'esigenza di tenere sotto controllo anche altri tasti:

- i tasti con le lettere;
- i tasti con i numeri;
- i quattro tasti con le frecce direzione;
- il tasto ENTER (INVIO).

Supponiamo, ad esempio, di avere un programma in cui l'utente deve scrivere un numero telefonico in una riga di testo (TextBox1) e poi deve validare questo numero premendo il tasto INVIO.
Questo programma deve essere in grado di riconoscere se l'utente ha premuto un numero oppure no, per evitare errori di battitura, e deve essere in grado di riconoscere la pressione del tasto INVIO.
A queste operazioni sono preposte le procedure che gestiscono gli eventi che si verificano alla pressione di un tasto sulla tastiera; in particolare, in questo caso:

- la procedura TextBox1.KeyDown riconosce la pressione del tasto INVIO;
- la procedura TextDown.KeyPress riconosce la pressione di caratteri (lettere, numeri, simboli).

Quando l'utente di un programma preme un tasto si verificano questi tre eventi:

- **KeyDown**, quando è premuto un tasto;
- **KeyPress**, quando è premuto un carattere (lettera, numero, simbolo, barra spazio);
- **KeyUp**, quando il tasto è rilasciato.

Negli esempi ed esercizi di questo capitolo ci interesseremo degli eventi **KeyDown** e **KeyPress**, che forniscono le indicazioni necessarie per sapere quale tasto è stato premuto dall'utente.
Tra i due eventi vi è una differenza sostanziale:

- L'evento **KeyDown** riconosce la pressione di qualsiasi tasto sulla tastiera: i tasti normalmente usati per scrivere e di tutti i tasti che si trovano ai bordi della tastiera: in senso orario sono i tasti ESC, i tasti funzione da F1 a F12, i tasti INS, DEL, BACK SPACE, ENTER, SHIFT, i quattro tasti direzione e i tasti ALT, CTRL, CAPS LOCK).
  KeyDown legge il nome e il numero corrispondenti a ogni tasto premuto, ma non distingue se questo è stato premuto in modalità normale o in modalità SHIFT (maiuscolo) e, dunque, non distingue se è stata premuta una lettera maiuscola o una lettera minuscola.
- L'evento **KeyPress** riconosce **solo** la pressione dei tasti connessi a **caratteri**, cioè ai simboli utilizzati nella scrittura (lettere, numeri, simboli e barra spazio).
  Utilizzando questo evento è possibile sapere se un tasto è stato premuto in modalità normale o in modalità SHIFT e, dunque, se è stata premuta una lettera maiuscola o una lettera minuscola.

Questa differenza è evidenziata graficamente nell'immagine seguente, nella quale i tasti interessati dall'evento KeyDown sono racchiusi dalla linea blu, mentre quelli interessati dall'evento KeyPress sono racchiusi dalla linea rossa.

Figura 236: I tasti interessati dagli eventi KeyDown e KeyPress.

L'esempio che segue visualizza i diversi effetti causati dai due eventi.
Apriamo un nuovo progetto, collochiamo nel Form1 un controllo Label e copiamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_KeyDown(sender As Object, e As
System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown

        Me.Text = ""
        Me.Font = New Font("Verdana", 10)

        Label1.Text = "dati forniti dall'evento Me.KeyDown:" & vbCrLf
        Label1.Text &= "proprietà e.Alt (tasto ALT) = " & e.Alt.ToString & vbCrLf
        Label1.Text &= "proprietà e.Shift (tasto maiuscole) = " &
e.Shift.ToString & vbCrLf
        Label1.Text &= "proprietà e.Control (tasto CTRL) = " & e.Control.ToString
& vbCrLf
        Label1.Text &= "proprietà e.KeyCode = " & e.KeyCode.ToString & vbCrLf
        Label1.Text &= "proprietà e.KeyValue = " & e.KeyValue & vbCrLf & vbCrLf

    End Sub
```

```
    Private Sub Form1_KeyPress(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs) Handles Me.KeyPress

        Me.Text = "E' stato premuto il tasto " & e.KeyChar
        Label1.Text &= "dati forniti dall'evento Me.KeyPress:" & vbCrLf
        Label1.Text &= "proprietà e.KeyChar = " & e.KeyChar

    End Sub

End Class
```

Notiamo che in entrambe le procedure Me.KeyDown e Me.KeyPress è presente il parametro "**e**", che rappresenta la tastiera del computer.
Le sue proprietà principali, visibili nel listato, sono:

Per l'evento **Me.KeyDown**:
- **e.Alt** (indica se il tasto ALT è premuto o no);
- **e.Shift** (indica se il tasto SHIFT, maiuscole, è premuto o no);
- **e.Control** (indica se il tasto CTRL è premuto o no);
- **e.KeyCode** (indica il nome con il quale VB individua il tasto premuto)
- **e.KeyValue** (indica il numero con il quale VB individua il tasto premuto)

Per l'evento **Me.KeyPress**:
- **e.KeyChar** (indica il carattere premuto, riconosce solo i tasti con le lettere maiuscole, le lettere minuscole, i numeri).

Notiamo che anche le proprietà dell'evento Me.KeyDown offrono la possibilità di controllare la pressione dei tasti ALT, SHIFT e CTRL, come My.Computer.Keyboard.
Mandiamo in esecuzione il programma e vediamo i valori che assumono tutte queste proprietà premendo tasti diversi:

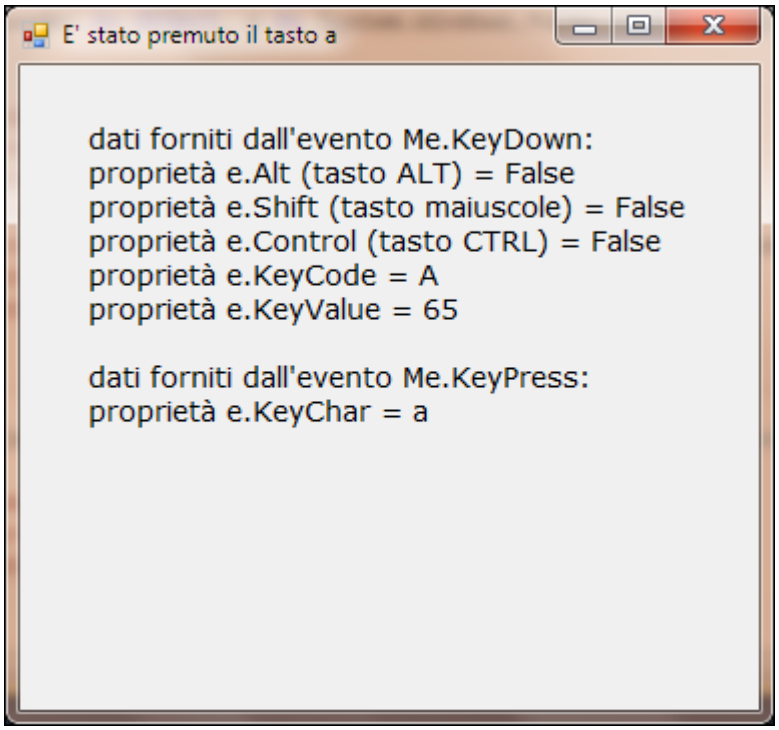


Figura 237: Proprietà degli eventi KeyDown e KeyPress del tasto "a".

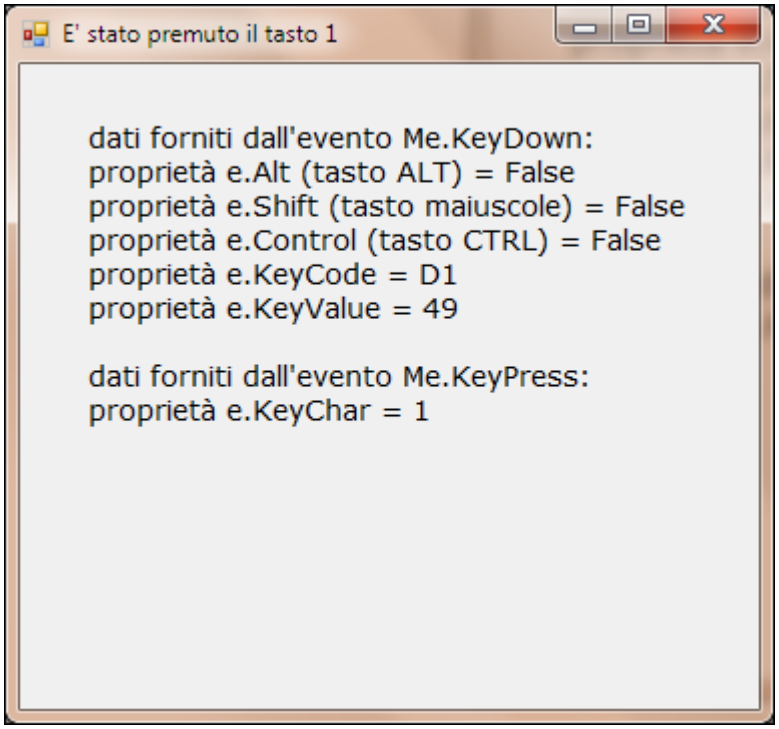


Figura 238: Proprietà degli eventi KeyDown e KeyPress del tasto "1".

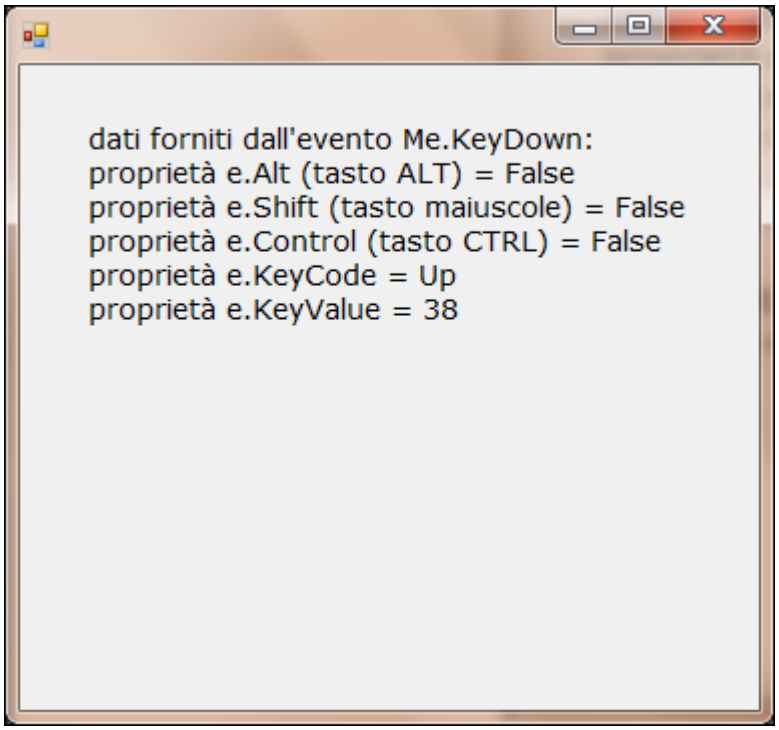


Figura 239: Proprietà dell'event KeyDown del tasto "freccia su".

Osserviamo in particolare, nelle tre immagini precedenti, i valori assunti dalle proprietà e.KeyCode, e.KeyValue, nell'evento KeyDown, e dalla proprietà e.KeyChar, nell'evento KeyPress:

	Evento KeyDown		**Evento KeyPress**
	Proprietà e.**KeyCode** (**nome** del tasto)	Proprietà e.**KeyValue** (**numero** del tasto)	Proprietà e.**KeyChar** (**carattere** scritto dal tasto)
pressione del tasto "a" minuscolo	**A**	**65**	**a**
pressione del tasto "1"	**D1**	**49**	**1**
pressione del tasto freccia su	**Up**	**38**	

Notiamo cha la pressione del tasto con la freccia su è ignorata dall'evento KeyPress, in quanto questo tasto si colloca al di fuori dell'area rossa che abbiamo visto nella figura precedente.

Ricapitolando, possiamo affermare che:

- se il programmatore ha bisogno di sapere quale **carattere** è stato premuto dall'utente, deve utilizzare l'evento **KeyPress** e la proprietà e.**KeyChar**.
  Il carattere premuto dall'utente è memorizzato in una variabile di tipo Char, il cui contenuto può essere analizzato con questi comandi:

```
    Private Sub Form1_KeyPress(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs) Handles Me.KeyPress

        Dim Carattere As Char = e.KeyChar
        Label1.Text = Carattere

    End Sub
```

- se il programmatore ha bisogno di sapere quale **tasto** è stato premuto dall'utente, deve utilizzare l'evento **KeyDown** e le sue proprietà e.**KeyCode** (nome del tasto) e e.**KeyValue** (numero del tasto).

Nella tabella che segue vediamo il quadro completo dei valori delle proprietà **KeyCode** e **KeyValue** per tutta la tastiera.

Tasto	**KeyCode** (nome)	**KeyValue** (numero)
BACKSPACE	Back	8
TAB	Tab	9
ENTER	Return	13
SHIFT	ShiftKey	16
CTRL	ShiftKey	17
ALT	ControlKey	18
CAPS LOCK	Capital	20
ESCAPE	Escape	27
PAGE UP	PageUp	33
PAGE DOWN	Next	34
END	End	35
HOME	Home	36
FRECCIA SINISTRA	Left	37
FRECCIA SU	Up	38
FECCIA DESTRA	Right	39
FRECCIA GIU’	Down	40
INSERT	Insert	45
DELETE	Delete	46
NUM LOCK	NumLock	144
0	D0	48
1	D1	49
2	D2	50
3	D3	51
4	D4	52
5	D5	53
6	D6	54
7	D7	55
8	D8	56
9	D9	57

Tasto	**KeyCode** (nome)	**KeyValue** (numero)
A	A	65
B	B	66
C	C	67
D	D	68
E	E	69
F	F	70
G	G	71
H	H	72
I	I	73
J	J	74
K	K	75
L	L	76
M	M	77
N	N	78
O	O	79
P	P	80
Q	Q	81
R	R	82
S	S	83
T	T	84
U	U	85
V	V	86
W	W	87
X	X	88
Y	Y	89
Z	Z	90
F1	F1	112
F2	F2	113
F3	F3	114
F4	F4	115
F5	F5	116
F6	F6	117
F7	F7	118
F8	F8	119
F9	F9	120
F10	F10	121
F11	F11	122
F12	F12	123

Tabella 35: Le proprietà KeyCode e KeyValue.

175: Filtraggio dei caratteri.

L'evento **KeyDown** offre al programmatore la possibilità di filtrare i caratteri scritti dall'utente, ed eventualmente di accettarne alcuni e rifiutarne altri.
Come abbiamo visto, alla pressione di un tasto con lettere, numeri o simboli, l'evento **KeyPress** memorizza il carattere premuto nella proprietà e.KeyChar.
Il carattere contenuto in e.KeyChar può essere analizzato e filtrato secondo alcuni criteri grazie ai quali, se il carattere premuto non è accettabile ai fini del programma, la sua scrittura è *saltata* o, meglio, è considerata come già effettuata.
Gli strumenti per effettuare queste analisi sono le funzioni

- **Char.IsLetter**
- **Char.IsNumber**
- **Char.IsSymbol**
- **Char.IsPunctuation**
- **Char.IsWhiteSpace**

che consentono, rispettivamente, di stabilire se il carattere premuto è una lettera, un numero, un simbolo, un segno di punteggiatura o uno spazio.
Il salto della scrittura di un carattere indesiderato avviene mediante il comando

- **e.Handled = True**

che può essere tradotto in questo modo: "*se il carattere premuto è un carattere indesiderato, considera la sua scrittura come già eseguita e passa oltre*".

Vediamo un esempio di tutto questo in un elenco di azioni in successione:

1. L'utente scrive in un TextBox la lettera "A".
2. L'evento TextBox1.KeyPress memorizza la lettera "A" nella proprietà e.KeyChar.
3. Il procedimento If Char.IsNumber(e.KeyChar) verifica se il contenuto di e.KeyChar è un numero oppure no.
4. Se il programmatore desidera che siano accettati solo numeri, la scrittura della lettera "A" è saltata, considerata come già effettuata, con il comando e.Handled = True.

Il listato seguente mostra un esempio delle azioni di filtraggio dei caratteri consentite dalle proprietà della struttura **Char**[83].
Per provare il listato apriamo un nuovo progetto, inseriamo nel form un controllo TextBox1, copiamo e incolliamo il codice nella Finestra del Codice.

```
Public Class Form1
```

[83] Non confondiamo questa struttura Char con la funzione Chr() che abbiamo visto nel Capitolo 21: FUNZIONI SU STRINGHE DI TESTO.
Le funzioni della struttura Char (Char.IsNumber, Char.IsLetter ecc.) che vediamo qui **filtrano** i caratteri premuti sulla tastiera in base a determinati criteri.
La funzione Chr() **scrive** il carattere il cui codice ASCII è indicato tra parentesi.
Avremo dunque, ad esempio:
Chr(65) = "A"
Char.IsLetter(Chr(65)) = True

```
    Private Sub TextBox1_KeyPress(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress

        If Char.IsNumber(e.KeyChar) Then MsgBox("E' stato premuto il numero " &
e.KeyChar)
        If Char.IsLetter(e.KeyChar) Then MsgBox("E' stata premuta la lettera " &
e.KeyChar)
        If Char.IsSymbol(e.KeyChar) Then MsgBox("E' stato premuto il simbolo " &
e.KeyChar)
        If Char.IsPunctuation(e.KeyChar) Then MsgBox("E' stata premuto un segno
di punteggiatura")
        If Char.IsWhiteSpace(e.KeyChar) Then MsgBox("E' stata premuta la barra
spazio " & e.KeyChar)

    End Sub

End Class
```

Nel prossimo esercizio vedremo un esempio di un'azione di filtraggio eseguita con
Il procedimento **If Char.IsNumber**, in un TextBox che accetta solo la scrittura di numeri.

Esercizio 115: Controllo dei tasti con i numeri da 0 a 9.

In questo programma abbiamo un TextBox (casella di testo) in cui l'utente deve scrivere un numero di telefono.
L'utente dunque può scrivere in questo TextBox solo numeri; il programma verifica se il carattere premuto dall'utente corrisponde a un numero oppure no e, se questo non corrisponde a un numero, il programma ne salta la scrittura.
Il programma si basa sull'evento KeyPress, sulla proprietà e.KeyChar e sulla proprietà Char.IsNumber.
Apriamo un nuovo progetto e inseriamo nel form un controllo TextBox.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub TextBox1_KeyPress(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress

        If Char.IsNumber(e.KeyChar) = False Then e.Handled = True

    End Sub

End Class
```

176: La proprietà e.KeyChar.

Nell'esercizio seguente vedremo un esempio di controllo dei tasti con le lettere.
Si tratta di un gioco didattico: il giocatore deve premere sulla tastiera la lettera corrispondente a quella che vede *cadere* nel monitor, prima che questa *tocchi terra*.
Nel programma vediamo l'utilizzo dell'evento KeyPress per determinare quale carattere è stato premuto (proprietà e.KeyChar) e per confrontarlo con la lettera scritta con la funzione Chr().

Esercizio 116: La proprietà e.KeyChar.

In questo programma abbiamo delle lettere maiuscole, sorteggiate a caso, che *cadono* dall'alto del form.
L'utente deve premere sulla tastiera i tasti corrispondenti alle lettere che vede cadere, prima che queste *tocchino* terra, cioè prima che escano dal bordo inferiore del form.
Il programma si basa su questi elementi:

- L'evento **Timer1.Tick**: a ogni *tic* del timer il form viene ridisegnato; la posizione in altezza delle lettere viene aumentata, per cui si ottiene l'effetto visivo delle lettere che *cadono*.
- L'evento **Me.Paint**: ridisegna il form, disegnando le lettere nella nuova posizione con il comando grafico **DrawString**.
- La procedura **PartenzaNuovaLettera** gestisce il sorteggio di una nuova lettera, con le sue proprietà (carattere, colore).
- L'evento **Me.KeyPress** si attiva quando l'utente preme un tasto sulla tastiera; la procedura **Form1_KeyPress** controlla se il tasto premuto corrisponde a una delle lettere che si vedono cadere nel form.

Per comprendere meglio il funzionamento del programma, ne vedremo prima una versione ridotta ai suoi elementi essenziali.
Apriamo un nuovo progetto e inseriamo nel form un controllo **Timer**.
Proprietà del **Form1**:

- DoubleBuffered = True

Proprietà del **Timer1**:

- Enabled = True
- Interval = 20

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea una variabile per memorizzare il codice ASCII della lettera che cade
dall'alto del form.
    ' Per iniziare, il codice = 65 corrisponde alla lettera A.
    Dim CodiceLettera As Integer = 65
    Dim Sinistra As Integer = 120
```

```
    Dim Altezza As Integer = 0
```

```
    Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ' Ripulisce il form e attiva l'evento Me.Paint.
        Me.Invalidate()

    End Sub
```

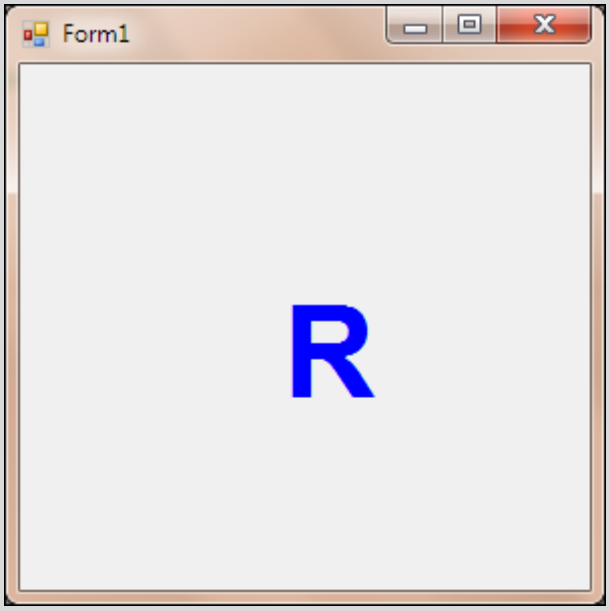
```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Ridisegna il form, con la lettera nella sua nuova posizione.
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim Carattere As New Font("Arial", 48, FontStyle.Bold)
        Altezza += 2

        e.Graphics.DrawString(Chr(CodiceLettera), Carattere, Brushes.Blue,
Sinistra, Altezza)

        ' Se la posizione Altezza della lettera esce dal bordo inferiore del
form,
        ' allora la lettera ha toccato terra; si avvia la procedura per
sorteggiare una nuova lettera.

        If Altezza > Me.ClientRectangle.Height Then
            Call PartenzaNuovaLettera()
        End If

    End Sub
```

```
    Private Sub PartenzaNuovaLettera()

        ' Questa procedura gestisce il sorteggio di una nuova lettera e delle sue
proprietà.
        ' Attiva la classe Random
        Dim Sorteggio As New Random

        ' Avvia il generatore di numeri casuali basandolo sul timer del computer
        Randomize()

        ' Sorteggia una lettera maiuscola (i codici ASCII delle lettere maiuscole
vanno dal 65 al 90):
        CodiceLettera = Sorteggio.Next(65, 91)

        ' La nuova lettera riparte dall'alto, fuori dal limite superiore del
form:
        Altezza = -30

    End Sub
```

```
    Private Sub Form1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles Me.KeyPress

        ' Controlla se il tasto premuto dall'utente corrisponde alla lettera che
sta cadendo:
```

```
        If e.KeyChar = Chr(CodiceLettera) Then
            ' Se il tasto corrisponde alla lettera che cade, è sorteggiata una
nuova lettera:
            Call PartenzaNuovaLettera()
        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Ricordiamo che le lettere che cadono sono lettere **maiuscole**, per cui per premere il tasto esatto è necessario premere anche il tasto delle maiuscole.

Dopo avere compreso il funzionamento della versione semplificata del programma, passiamo alla versione completa, in cui le lettere che cadono sono cinque, contemporaneamente, di dimensioni e di colori diversi.
Apriamo un nuovo progetto.
Gli diamo il nome "Pioggia di lettere".
Il **Form1** ha la proprietà **StartPosition = CenterScreen.**
Inseriamo nel form un controllo **Label1** e un controllo **Timer**. Le proprietà del Form, del controllo Label e del Timer sono definite dal codice del programma.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea cinque variabili per memorizzare i codici ASCII delle cinque lettere
che cadono dall'alto del form.
    ' Per iniziare, il codice ASCII = 32 corrisponde ad uno spazio vuoto, dunque
all'inizio del programma non si vede alcuna lettera.
    Dim CodiceLettera() As Integer = {32, 32, 32, 32, 32}
```

```
    ' Crea cinque variabili per memorizzare la grandezza dei caratteri di ogni
singola.
    ' La grandezza andrà da 24 a 48.
    Dim GrandezzaCarattere() As Integer = {48, 48, 48, 48, 48}

    ' Crea cinque variabili per memorizzare la velocità di caduta di ogni singola
lettera.
    ' La velocità iniziale per tutte le lettere è = 5, poi il programma sorteggia
velocità casuali, da 1 a 2 pixel per ogni tic del timer.
    Dim Velocità() As Integer = {5, 5, 5, 5, 5}

    ' Crea cinque variabili per memorizzare le posizioni di ogni singola lettera
che cade dall'alto del form.
    Dim Sinistra() As Integer = {20, 80, 140, 200, 260}
    Dim Altezza() As Integer = {0, 0, 0, 0, 0}

    ' Crea una lista di cinque colori.
    Dim Colore As New List(Of Color) From {Color.Black, Color.Red, Color.Yellow,
Color.Blue, Color.Green}

    ' Crea una variabile per memorizzare il numero dei tasti premuti
correttamente.
    Dim LettereEsatte As Integer
```

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        ' Le lettere che cadono sono tutte maiuscole, per cui è necessario
controllare
        ' che il tasto BLOCCA MAIUSCOLE sia premuto:
        If My.Computer.Keyboard.CapsLock = False Then
            MsgBox("Premi il tasto CAPS LOCK (blocca maiuscole)")
            End
        End If

        ' All'avvìo del programma, imposta il form:
        With Me
            .DoubleBuffered = True
            .Width = 400
            .Height = 300
            .FormBorderStyle = Windows.Forms.FormBorderStyle.FixedToolWindow
            .Text = "Pioggia di lettere"
            .BackColor = Color.White
        End With

        With Label1
            .AutoSize = False
            .BackColor = Color.Maroon
            .ForeColor = Color.White
            .Text = "PREMI LE LETTERE PRIMA CHE CADANO A TERRA"
            .TextAlign = ContentAlignment.MiddleCenter
            .Width = Me.ClientRectangle.Width
            .Height = 23
            .Left = 0
            .Top = Me.ClientRectangle.Height - 23
        End With

        Timer1.Interval = 20
        Timer1.Enabled = True
```

```
    End Sub
```

```
    Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Timer1.Tick
        ' Ad ogni tic del timer, cancella il contenuto del form, causando
l'evento Form.Paint:
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Questa procedura ridisegna il form, con le cinque lettere nelle nuove
posizioni.
        ' Migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' In questo ciclo For... Next sono definite le proprietà di ognuna delle
cinque lettere:
        ' la grandezza e il colore del carattere, la posizione della lettera.

        For Contatore As Integer = 0 To 4
            Dim Carattere As New Font("Arial", GrandezzaCarattere(Contatore),
FontStyle.Bold)
            Altezza(Contatore) = Altezza(Contatore) + Velocità(Contatore)

            Dim Pennello As New SolidBrush(Color.White)
            Pennello.Color = Colore(Contatore)

            ' Disegna una lettera:
            e.Graphics.DrawString(Chr(CodiceLettera(Contatore)), Carattere,
Pennello, Sinistra(Contatore), Altezza(Contatore))

            ' Se la posizione Altezza della lettera supera la posizione Top del
controllo Label1,
            ' allora la lettera ha toccato terra e si avvia la procedura per
sorteggiare una nuova lettera
            ' che tornerà a cadere dell'alto del form.

            If Altezza(Contatore) > Label1.Top Then
                Call PartenzaNuovaCodiceLettera(Contatore)
            End If
        Next

    End Sub
```

```
    Private Sub PartenzaNuovaCodiceLettera(Contatore As Integer)

        ' Questa procedura gestisce il sorteggio di una nuova lettera e delle sue
proprietà.
        ' Attiva la classe Random
        Dim Sorteggio As New Random
        ' Avvia il generatore di numeri casuali basandolo sul timer del computer
        Randomize()

NuovoSorteggioLettera:
```

```
        ' Sorteggia una lettera maiuscola (i codici ASCII delle lettere maiuscole
vanno dal 65 al 90):
        Dim NuovoCodice As Integer = Sorteggio.Next(65, 91)
        If CodiceLettera.Contains(NuovoCodice) Then GoTo NuovoSorteggioLettera

        CodiceLettera(Contatore) = NuovoCodice

        ' Sorteggio delle proprietà della nuova lettera:
        GrandezzaCarattere(Contatore) = Sorteggio.Next(24, 49)
        Velocità(Contatore) = Sorteggio.Next(1, 3)
        Sinistra(Contatore) = Sorteggio.Next(Contatore * 60 + 20, Contatore * 60
+ 60)

        ' La nuova lettera parte dall'alto, fuori dal limite superiore del form:
        Altezza(Contatore) = -GrandezzaCarattere(Contatore)

    End Sub
```

```
    Private Sub Form1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles Me.KeyPress

        ' Controlla se il tasto premuto dall'utente corrisponde a una delle
cinque lettere che stanno cadendo:

        For Contatore As Integer = 0 To 4
            If e.KeyChar = Chr(CodiceLettera(Contatore)) Then
                ' Se il tasto corrisponde a una lettera, la lettera viene
cancellata
                ' (il codice ASCII = 32 corrisponde a uno spazio vuoto)

                CodiceLettera(Contatore) = 32
                My.Computer.Audio.PlaySystemSound(Media.SystemSounds.Exclamation)

                ' Aumenta e visualizza il numero delle risposte esatte:
                LettereEsatte += 1
                Label1.Text = CStr(LettereEsatte)
            End If
        Next

    End Sub

End Class
```

Ecco un’immagine del programma in esecuzione:

177: La proprietà e.KeyCode.

Nell'esercizio seguente vedremo un esempio di controllo dei tasti direzione (i quattro tasti con le frecce in basso a destra sulla tastiera).
Premendo uno di questi tasti, l'immagine di un bambino si muove sullo schermo del monitor, visualizzando in rapida successione dei fotogrammi che danno l'illusione che il bambino cammini.
Trattandosi di tasti e non di caratteri, questo programma si basa sull'evento KeyDown e sulla proprietà e.KeyCode (nome del testo premuto).

Esercizio 117: La proprietà E.KeyCode.

Il programma si basa sull'immagine **Passegiata.bmp** che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini**:

L'immagine misura 96 pixel in larghezza x 128 pixel in altezza, e può essere divisia in 12 riquadri di 32 x 32 pixel ciascuno:

Ogni riquadro mostra un *fotogramma* della camminata di un bambino:
- nella prima riga si vedono i tre fotogrammi della camminata verso il basso;
- nella seconda riga si vedono i tre fotogrammi della camminata verso sinistra;
- nella terza riga si vedono i tre fotogrammi della camminata verso destra;
- nella quarta riga si vedono i tre fotogrammi della camminata verso l'alto.

Il programma prende i tre fotogrammi di una di queste righe e li mostra in successione, dando l'impressione che il bambino cammini. La riga scelta (cioè: la direzione della camminata) dipende dalla pressione di uno dei quattro tasti direzione che si trovano in basso a destra sulla tastiera.

Apriamo un nuovo progetto e inseriamo nelle risorse del programma l'immagine **Passeggiata** che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini**. Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea l'immagine originale.
    ' Questa immagine è divisa in 12 riquadri, disposti su 4 file, con tre
riquadri per ogni fila.
    Dim ImmagineOriginale As New Bitmap(My.Resources.Passeggiata)

    ' Crea una variabile per memorizzare la riga di tre fotogrammi utilizzati per
dare l'illusione che
    ' il bambino cammini:
    Dim RigaFotogrammi As Integer = 0

    ' Per ogni direzione, in ogni riga sono disponibili tre fotogrammi:
    ' la variabile Contatore, che andrà da 0 a 2, serve a memorizzare la
successione dei fotogrammi:
    Dim Contatore As Integer = 0

    ' Crea due variabili per localizzare nel form l'immagine del bambino che
cammina:
    Dim Sinistra As Integer = 0
    Dim Altezza As Integer = 0
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' All'avvio del programma, imposta il colore dello fondo del form come il
colore dello sfondo dell'immagine Passeggiata e migliora la resa grafica:
```

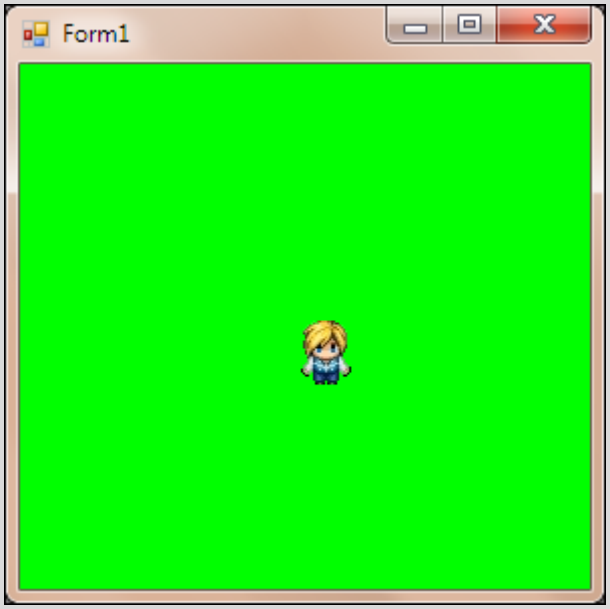
```
        Me.BackColor = Color.Lime
        Me.DoubleBuffered = True

    End Sub
```

```
    Private Sub Form1_KeyDown(sender As Object, e As
System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown
        ' Procedura per gestire la pressione dei tasti direzione:

        Select Case e.KeyCode
            Case Is = Keys.Right
                RigaFotogrammi = 2
                Sinistra += 8
            Case Is = Keys.Down
                RigaFotogrammi = 0
                Altezza += 8
            Case Is = Keys.Left
                RigaFotogrammi = 1
                Sinistra -= 8
            Case Is = Keys.Up
                RigaFotogrammi = 3
                Altezza -= 8
        End Select

        ' Questi comandi fermano lo spostamento dell'immagine del bambino,
        ' se questa supera i limiti del form:
        If Sinistra < 0 Then Sinistra = 0
        If Altezza < 0 Then Altezza = 0
        If Sinistra > Me.ClientRectangle.Width - 32 Then Sinistra =
Me.ClientRectangle.Width - 32
        If Altezza > Me.ClientRectangle.Height - 32 Then Altezza =
Me.ClientRectangle.Height - 32

        ' Ripulisce il form e causa l'evento Form1.Paint, con la relativa
procedura, per disegnare l'immagine del bambino nella nuova posizione:
        Me.Invalidate()

        ' Aumenta il contatore di una unità, da 0 a 2, per passare in rassegna
uno alla volta
        ' i tre fotogrammi di ogni direzione:
        Contatore += 1
        If Contatore = 2 Then Contatore = 0

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        ' Questa procedura disegna l'immagine del bambino nella sua nuova
posizione:

        ' Crea un rettangolo delle dimensioni di un fotogramma (32 x 32) e lo
colloca nella nuova posizione:
        Dim Rettangolo As New Rectangle(Sinistra, Altezza, 32, 32)

        ' Disegna la nuova immagine del bambino nella nuova posizione.
        ' Le variabili Contatore e RigaFotogrammi dicono quale fotogramma deve
essere visualizzato dall'immagine originale.
        '
        e.Graphics.DrawImage(ImmagineOriginale, Rettangolo, Contatore * 32,
RigaFotogrammi * 32, 32, 32, GraphicsUnit.Pixel)
```

```
    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Dopo avere compreso le modalità di funzionamento del programma, possiamo arricchirlo inserendo nel form alcuni ostacoli ai movimenti dell'immagine con il bambino.
Inseriamo nel form due controlli PictureBox che fungeranno da ostacoli; le loro proprietà sono impostate dal codice seguente.
Ora copiamo e incolliamo nella Finestra del Codice questo listato che riprende in toto il listato precedente, con le aggiunte relative ai due ostacoli inseriti nel form:

```
Public Class Form1
    ' Crea l'immagine originale.
    ' Questa immagine è divisa in 12 riquadri, disposti su 4 file, con tre
riquadri per ogni fila.
    Dim ImmagineOriginale As New Bitmap(My.Resources.Passeggiata)
    ' Crea una variabile per memorizzare la riga di tre fotogrammi utilizzati per
dare l'illusione che
    ' il bambino cammini:
    Dim RigaFotogrammi As Integer = 0

    ' Per ogni direzione, in ogni riga sono disponibili tre fotogrammi:
    ' la variabile Contatore, che andrà da 0 a 2, serve a memorizzare la
successione dei fotogrammi:
    Dim Contatore As Integer = 0

    ' Crea due variabili per localizzare nel form l'immagine del bambino che
cammina:
    Dim Sinistra As Integer = 0
    Dim Altezza As Integer = 0
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load
        ' All'avvio del programma, imposta il colore dello fondo del form come il
colore dello sfondo
        ' dell'immagine Passeggiata e migliora la resa grafica:
        Me.BackColor = Color.Lime
        Me.DoubleBuffered = True

        Me.BackColor = Color.Lime
        Me.DoubleBuffered = True

        ' Imposta le proprietà dei due controlli PictureBox:
        With PictureBox1
            .BackColor = Color.Red
            .Location = New Point(72, 72)
            .Size = New Size(72, 72)
        End With

        With PictureBox2
            .BackColor = Color.Red
            .Location = New Point(118, 118)
            .Size = New Size(72, 72)
        End With

    End Sub
```

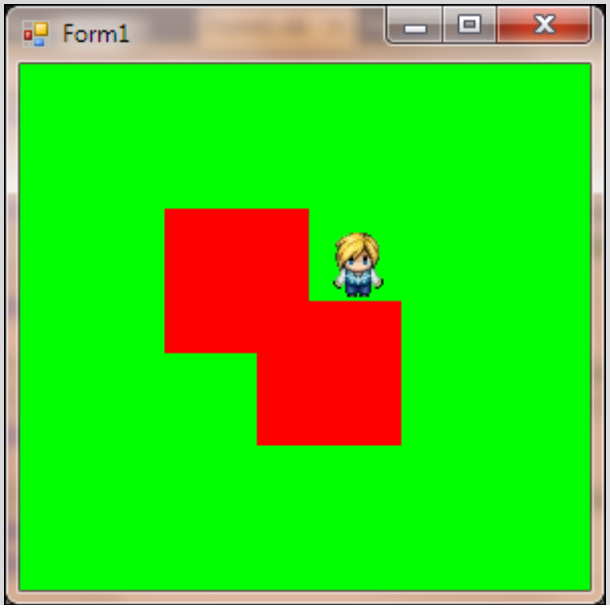
```
    Private Sub Form1_KeyDown(sender As Object, e As
System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown

        ' Crea la variabile ProssimaPosizione per memorizzare i dati della
posizione in cui l'immagine andrà
        ' a collocarsi al prossimo movimento:
        Dim ProssimaPosizione As Rectangle

        Select Case e.KeyCode

            Case Is = Keys.Right
                ' Il bambino va a destra, le immagini si trovano nella terza
RigaFotogrammi dell'immagine originale:
                RigaFotogrammi = 2
                ' Crea il rettangolo ProssimaPosizione per controllare se la via
è libera oppure no.
                ProssimaPosizione = New Rectangle(Sinistra + 4, Altezza, 32, 32)
                ' Invia alla funzione ViaLibera i dati relativi alla prossima
posizione, per sapere se la via sarà libera o no.
                If ViaLibera(ProssimaPosizione) Then Sinistra += 4

            Case Is = Keys.Down
                ' Il bambino va a verso il basso, le immagini si trovano nella
prima RigaFotogrammi dell'immagine originale:
                RigaFotogrammi = 0
                ' Crea il rettangolo ProssimaPosizione per controllare se la via
è libera oppure no.
                ProssimaPosizione = New Rectangle(Sinistra, Altezza + 4, 32, 32)
                ' Invia alla funzione ViaLibera i dati relativi alla prossima
posizione, per sapere se la via sarà libera o no.
                If ViaLibera(ProssimaPosizione) Then Altezza += 4

            Case Is = Keys.Left
```

```
                ' Il bambino va a sinistra, le immagini si trovano nella seconda
RigaFotogrammi dell'immagine originale:
                RigaFotogrammi = 1
                ' Crea il rettangolo ProssimaPosizione per controllare se la via
è libera oppure no.
                ProssimaPosizione = New Rectangle(Sinistra - 4, Altezza, 32, 32)
                ' Invia alla funzione ViaLibera i dati relativi alla prossima
posizione, per sapere se la via sarà libera o no.
                If ViaLibera(ProssimaPosizione) Then Sinistra -= 4
            Case Is = Keys.Up
                ' Il bambino va versi l'alto, le immagini si trovano nella quarta
RigaFotogrammi dell'immagine originale:
                RigaFotogrammi = 3
                ' Crea il rettangolo ProssimaPosizione per controllare se la via
è libera oppure no.
                ProssimaPosizione = New Rectangle(Sinistra, Altezza - 4, 32, 32)
                If ViaLibera(ProssimaPosizione) Then Altezza -= 4
        End Select

        ' Questi comandi fermano lo spostamento dell'immagine del bambino,
        ' se questa supera i limiti del form:
        If Sinistra < 0 Then Sinistra = 0
        If Altezza < 0 Then Altezza = 0
        If Sinistra > Me.ClientRectangle.Width - 32 Then Sinistra =
Me.ClientRectangle.Width - 32
        If Altezza > Me.ClientRectangle.Height - 32 Then Altezza =
Me.ClientRectangle.Height - 32

        ' Ripulisce il form e causa l'evento Form1.Paint, con la relativa
procedura, per disegnare
        ' l'immagine del bambino nella nuova posizione:
        Me.Invalidate()

        ' Aumenta il contatore di una unità, da 0 a 2, per passare in rassegna
uno alla volta
        ' i tre fotogrammi di ogni direzione:
        Contatore += 1
        If Contatore = 2 Then Contatore = 0

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        ' Questa procedura disegna l'immagine del bambino nella sua nuova
posizione:

        ' Crea un rettangolo delle dimensoni di un fotogramma (32 x 32) e lo
colloca nella nuova posizione:
        Dim Rettangolo As New Rectangle(Sinistra, Altezza, 32, 32)

        ' Disegna la nuova immagine del bambino nella nuova posizione.
        ' Le variabili Contatore e RigaFotogrammi dicono quale fotogramma deve
essere visualizzato,
        ' dall'immagine originale.
        '
        e.Graphics.DrawImage(ImmagineOriginale, Rettangolo, Contatore * 32,
RigaFotogrammi * 32, 32, 32, GraphicsUnit.Pixel)

    End Sub
```

```
    Private Function ViaLibera(ProssimaPosizione As Rectangle) As Boolean

        ' Questa funzione riceve i dati relativi alla ProssimaPosizione della
immagine con il bambino
        ' e li confronta con i dati delle posizioni dei due controlli PictureBox.
        ' Se c'è intersezione, la funzione restituisce il valore ViaLibera =
False e il bambino è bloccato,
        ' altrimenti restituisce il valore ViaLibera = True e l'immagine del
bambino può muoversi.
        If ProssimaPosizione.IntersectsWith(PictureBox1.Bounds) Or
ProssimaPosizione.IntersectsWith(PictureBox2.Bounds) Then
            Return (False)
        Else
            Return (True)

        End If

    End Function

End Class
```

Ecco un'immagine del programma in esecuzione:



178: La proprietà e.KeyValue.

Nel prossimo esercizio vedremo un programma basato sull'evento KeyDown e sulla proprietà e.KeyValue (numero del testo premuto).

Esercizio 118: La proprietà e.KeyValue.

Apriamo un nuovo progetto e inseriamo nel Form1 un controllo PictureBox con la proprietà **SizeMode = AutoSize**.
Funzionamento del programma: quando l'utente fa un *clic* su uno dei tasti con le lettere, il PictureBox visualizza un'immagine della lettera premuta.
Se l'utente fa un *clic* su uno degli altri tasti, il PictureBox non visualizza alcuna immagine.
Le immagini da visualizzare si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Alfabeto**: si tratta delle 26 lettere dell'alfabeto inglese, in 26 immagini di formato .gif, numerate da 1 a 26.
Il programma si basa sull'evento KeyDown: quando l'utente preme un tasto sulla tastiera, l'evento KeyDown registra la proprietà e.KeyValue del tasto premuto, cioè il suo numero di codice.
Sappiamo che i tasti con le lettere da A a Z hanno i codici da 65 a 90, per cui se il codice del tasto premuto è minore di 65 o maggiore di 90, il programma non visualizza alcuna immagine.
Se invece il codice del tasto premuto è tra 65 e 90, e dunque abbiamo la pressione di un tasto/lettera, il programma effettua queste operazioni:

- sottrae al codice il numero 64;
- visualizza nel PictureBox l'immagine il cui nome corrisponde al risultato dell'operazione.

Esempio:

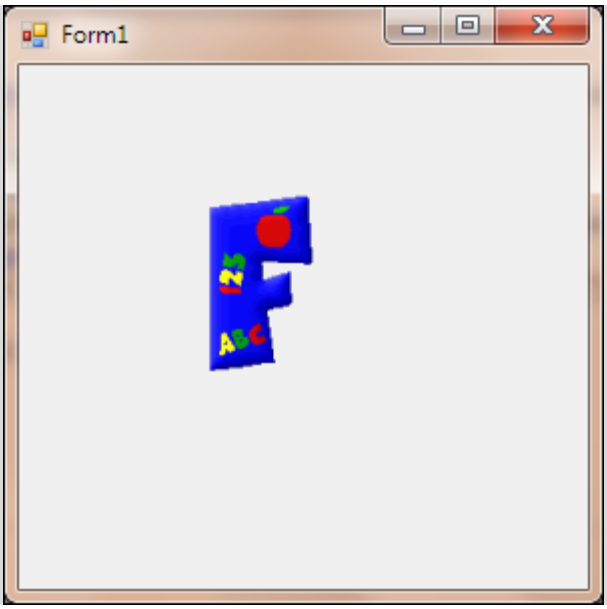
- è stato premuto il tasto con la lettera "A", il cui codice è 65;
- l'operazione 65 - 64 dà come risultato 1 e, dunque,
- il programma visualizza nel PictureBox l'immagine 1.gif.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim Percorso As String =
My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\A scuola con VB
2010\Immagini\Alfabeto\"

    Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown

        If e.KeyValue > 64 And e.KeyValue < 91 Then
            Dim Codice As String = (e.KeyValue - 64).ToString
            PictureBox1.Image = Image.FromFile(Percorso & Codice & ".gif")
            My.Computer.Audio.PlaySystemSound(Media.SystemSounds.Asterisk)
```

```
        Else
            PictureBox1.Image = Nothing
            My.Computer.Audio.PlaySystemSound(Media.SystemSounds.Beep)
        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Capitolo 33: GESTIONE DEL MOUSE.

My.Computer.Mouse consente di conoscere due proprietà del mouse: l'esistenza o meno della rotella e il numero di linee che scorrono avanti o indietro a ogni scatto della rotella. Le due proprietà sono:

- My.Computer.Mouse.WheelExists
- My.Computer.Mouse.WheelScrollLines

Sono proprietà **di sola lettura**, in quanto non possono essere modificate dall'interno di un programma (per modificare il numero di linee che il mouse fa scorrere a ogni scatto della rotella è necessario modificare le proprietà del mouse nel Pannello di Controllo del sistema operativo).

Questo listato mostra le due proprietà all'opera:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        If My.Computer.Mouse.WheelExists = True Then
            Dim NumeroLinee As Integer = My.Computer.Mouse.WheelScrollLines
            MsgBox("Il mouse fa scorrere " & NumeroLinee & " linee a ogni scatto
della rotella.")
        Else
            MsgBox("Il mouse non ha la rotella.")
        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

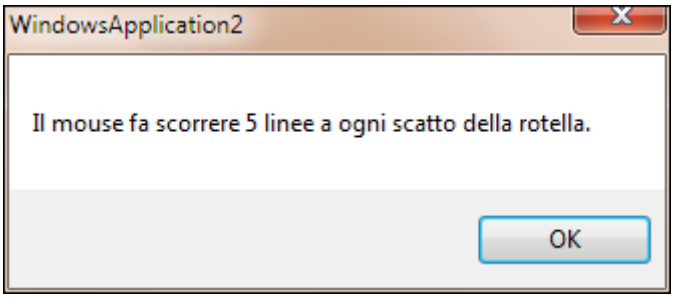


Figura 240: Le proprietà WheelExists e WheelScrollLines.

Come per la tastiera, anche per la gestione degli eventi del mouse è necessario ricorrere alle procedure che gestiscono i singoli eventi.
Gli eventi che l'utente di un programma può causare con il mouse sono:

- il *clic* con il pulsante sinistro, utilizzato in Windows, ad esempio, per chiudere le finestre e per uscire dai programmi;
- il *clic* con il pulsante destro, utilizzato in Windows, ad esempio, per visualizzare la finestra con il menu dei comandi disponibili per un determinato oggetto;
- il *doppio clic*, utilizzato per aprire le finestre e per lanciare i programmi;
- il passaggio del mouse sopra un oggetto;
- il trascinamento di un oggetto, utilizzato per spostare le finestre sul desktop, per dimensionarle, per trascinare gli oggetti nel cestino...
- lo scorrimento della rotella centrale, se il mouse ne è dotato.

Nei prossimi paragrafi vedremo esempi ed esercizi per la gestione di ognuno di essi.

179: Gli eventi MouseEnter e MouseLeave.

Gli eventi **MouseEnter** e **MouseLeave** accadono, rispettivamente, quando il puntatore del mouse entra nell'area occupata da un controllo, oppure quando ne esce.
Vediamo un esempio del loro utilizzo nel listato che segue.
Per provare il listato, è necessario inserire in un form due controlli PictureBox, con i colori di sfondo scelti a piacere in modo che i due controlli siano visibili nel form.

```
Public Class Form1

    Private Sub PictureBox1_MouseEnter(sender As Object, e As System.EventArgs)
Handles PictureBox1.MouseEnter

        PictureBox2.Visible = False

    End Sub

    Private Sub PictureBox1_MouseLeave(sender As Object, e As System.EventArgs)
Handles PictureBox1.MouseLeave

        PictureBox2.Visible = True

    End Sub

End Class
```

Il senso delle istruzioni scritte nel listato è questo:

- quando il mouse entra nell'area occupata dal PictureBox1, il controllo PictureBox2 è reso non visibile;
- quando il mouse esce dall'area occupata dal PictureBox1, il controllo PictureBox2 torna ad essere visibile.

Nell'esercizio seguente vedremo un esempio più complesso: quando il mouse entra nell'area di un controllo PictureBox, questo evento attiva un Timer che fa ruotare l'immagine nel PictureBox. Quando il mouse esce dall'area del controllo PictureBox, il Timer è disattivato e la rotazione dell'immagine cessa.

Esercizio 119: Gli eventi MouseEnter e MouseLeave.

Apriamo un nuovo progetto e inseriamo nel form un controllo **PictureBox** e un componente **Timer**.
Impostiamo queste proprietà:
Per il Form1

- **DoubleBuffered** = True

Per il Timer1

- **Enabled** = False
- **Interval** = 100

Per il PictureBox1

- **Image**: prendiamo l'immagine **Spirale**, che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini**.
- **Location** = 228; 206
- **Size** = 44; 44
- **SizeMode** = AutoSize

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Quando il mouse entra nel controllo PictureBox1 attiva il Timer:
    Private Sub PictureBox1_MouseEnter(sender As Object, e As System.EventArgs)
Handles PictureBox1.MouseEnter

        Timer1.Enabled = True

    End Sub
```

```
    ' Quando il mouse esce dal controllo PictureBox1 disattiva il Timer:
    Private Sub PictureBox1_MouseLeave(sender As Object, e As System.EventArgs)
Handles PictureBox1.MouseLeave

        Timer1.Enabled = False

    End Sub
```

```
    Private Sub Timer1_Tick_1(sender As System.Object, e As System.EventArgs)
Handles Timer1.Tick

        ' A ogni tic del timer crea un'immagine virtuale e copia nell'immagine
virtuale il contenuto del controllo PictureBox1:
        Dim ImmagineVirtuale As New Bitmap(PictureBox1.Image)

        ' Ruota l'immagine virtuale di 90 gradi:
```

```
        ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate90FlipNone)

        ' Aggiorna il contenuto del controllo PictureBox1:
        PictureBox1.Image = ImmagineVirtuale

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

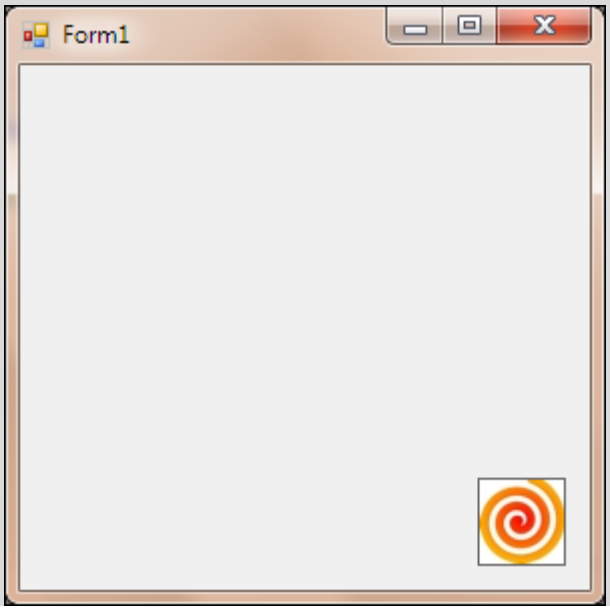


Figura 241: Gestione degli eventi MouseEnter e MouseLeave.

180: L'evento MouseHover.

L'evento **MouseHover** accade quando il puntatore del mouse si sofferma sull'area occupata da un controllo.
Si può utilizzare questo evento, ad esempio, per evidenziare un oggetto o un testo quando il mouse vi è sopra, come nel listato che segue.
Per provare il listato, è necessario inserire in un form un controllo Label, con un testo scritto a piacere.

```
Public Class Form1

    Private Sub Label1_MouseHover(sender As Object, e As System.EventArgs)
Handles Label1.MouseHover

        Label1.ForeColor = Color.Blue

    End Sub
```

```
    Private Sub Form1_MouseHover(sender As Object, e As System.EventArgs) Handles
Me.MouseHover

        Label1.ForeColor = Color.Black

    End Sub

End Class
```

Il senso delle istruzioni è questo:

- quando il puntatore del mouse si sofferma sull'area del controllo Label, visualizza il testo in blu;
- quando il puntatore del mouse si sofferma sull'area del form libera da controlli, torna a visualizzare il testo del controllo Label1 in nero.

181: Gli eventi Click, DoubleClick e MouseWheel.

Gli eventi **Click, DoubleClick** e **MouseWheel** accadono, rispettivamente, quando l'utente preme i pulsanti del mouse con un *clic* o un doppio *clic*, oppure quando fa scorrere la rotella centrale.

L'evento **Click** è l'evento principe nei sistemi operativi moderni: in pochi decenni gli utenti di computer in tutto il mondo hanno imparato a conoscerlo e a praticarlo con naturalezza.
Vediamo un esempio di utilizzo di questo evento nel listato che segue.
Per provare il listato, è necessario inserire in un form due controlli PictureBox, con i colori di sfondo scelti a piacere.

```
Public Class Form1

    Private Sub PictureBox1_Click(sender As Object, e As System.EventArgs)
Handles PictureBox1.Click

        ' Se il controllo PictureBox2 è visibile rendilo non visibile, o
viceversa:
        PictureBox2.Visible = Not PictureBox2.Visible

    End Sub

End Class
```

Il senso delle istruzioni è questo:

- quando avviene un evento Click del mouse sul riquadro PictureBox1 esegui una di queste azioni:
- se il controllo PictureBox2 è visibile, allora rendilo non visibile
- altrimenti, se il controllo PictureBox2 è non visibile, allora rendilo visibile.

Ecco ora un esempio di utilizzo dell'evento **DoubleClicK**.
Per provare il listato, è necessario inserire in un form due controlli PictureBox, con i colori di sfondo scelti a piacere.

```
Public Class Form1

    Private Sub PictureBox1_DoubleClick(sender As Object, e As System.EventArgs)
Handles PictureBox1.DoubleClick

        ' Cambia il colore di sfondo del controllo PictureBox2:
        If PictureBox2.BackColor = Color.Yellow Then
            PictureBox2.BackColor = Color.Green
        Else
            PictureBox2.BackColor = Color.Yellow
        End If

    End Sub

End Class
```

Il senso delle istruzioni è questo:
- quando avviene un evento DoubleClick del mouse sul riquadro PictureBox1 esegui una di queste azioni:
- se lo sfondo di PictureBox2 è giallo, allora fallo diventare verde
- altrimenti, se lo sfondo di PictureBox2 non è giallo, allora fallo diventare giallo.

Vediamo un esempio di utilizzo dell'evento **MouseWheel** nel listato che segue.
Per provare il listato, è necessario inserire in un form un controllo PictureBox, con il colore di sfondo scelto a piacere.

```
Public Class Form1

    Private Sub Me_MouseWheel(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseWheel

        ' Muovi il controllo PictureBox1 in alto o in basso secondo il movimento
della rotella del mouse:
        PictureBox1.Top += e.Delta / 40

    End Sub

End Class
```

Il senso delle istruzioni è questo:
- quando avviene un evento MouseWheel (uno scatto della rotella del mouse) sul form esegui queste azioni:
- muovi in alto o in basso il controllo PictureBox1, modificando la sua proprietà Top. La proprietà Top viene modificata sommando (o sottraendo) il valore e.Delta (il numero di linee che la rotella del mouse fa scorrere a ogni scatto), diviso per 40. Tale numero è positivo quando la rotella va verso l'alto, è negativo quando va verso il basso.

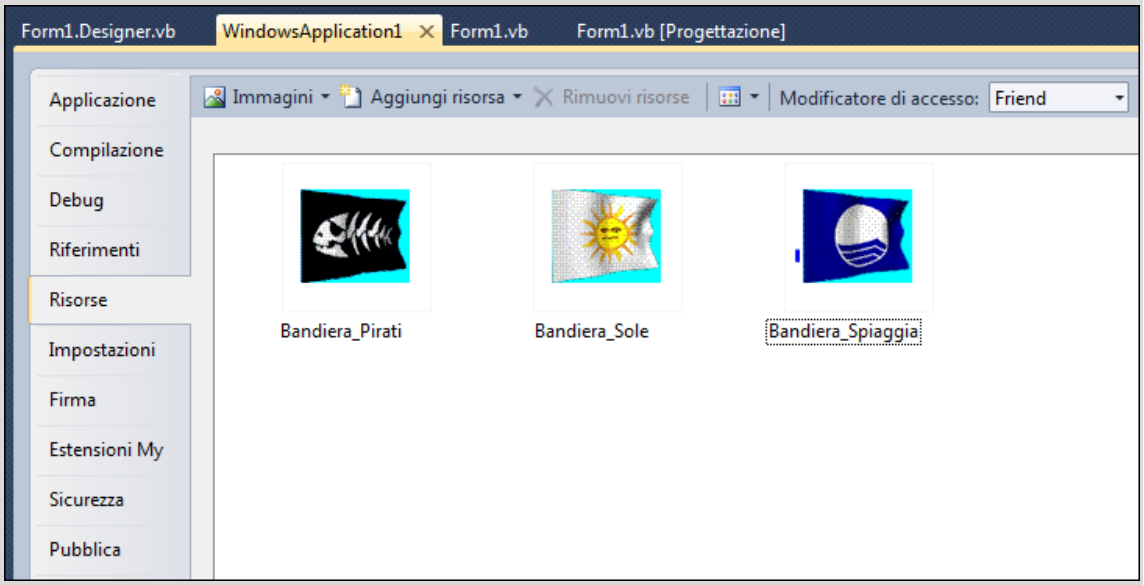
Nell'esercizio seguente vedremo un esempio più complesso di utilizzo degli eventi **Click** e **MouseWheel**.

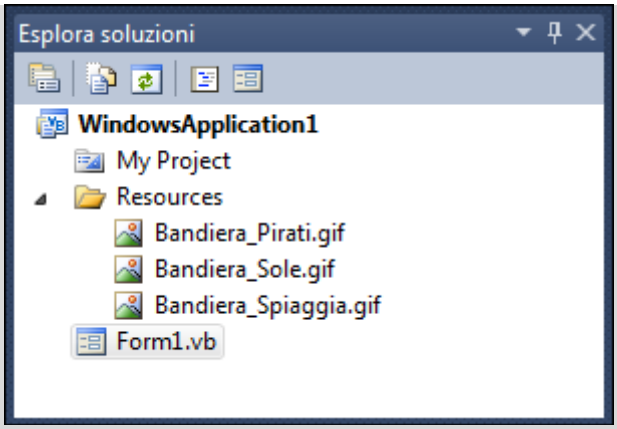
Esercizio 120: Gli eventi Click e MouseWheel.

Questo programma mostra come sfondo del form il disegno di un vascello il cui albero maestro è senza bandiera.
Cliccando con il mouse uno dei tre boccaporti del vascello, l'utente sceglie una bandiera; girando la rotella del mouse fa salire o scendere la bandiera sull'albero maestro.
Apriamo un nuovo progetto.
Impostiamo le proprietà del form:

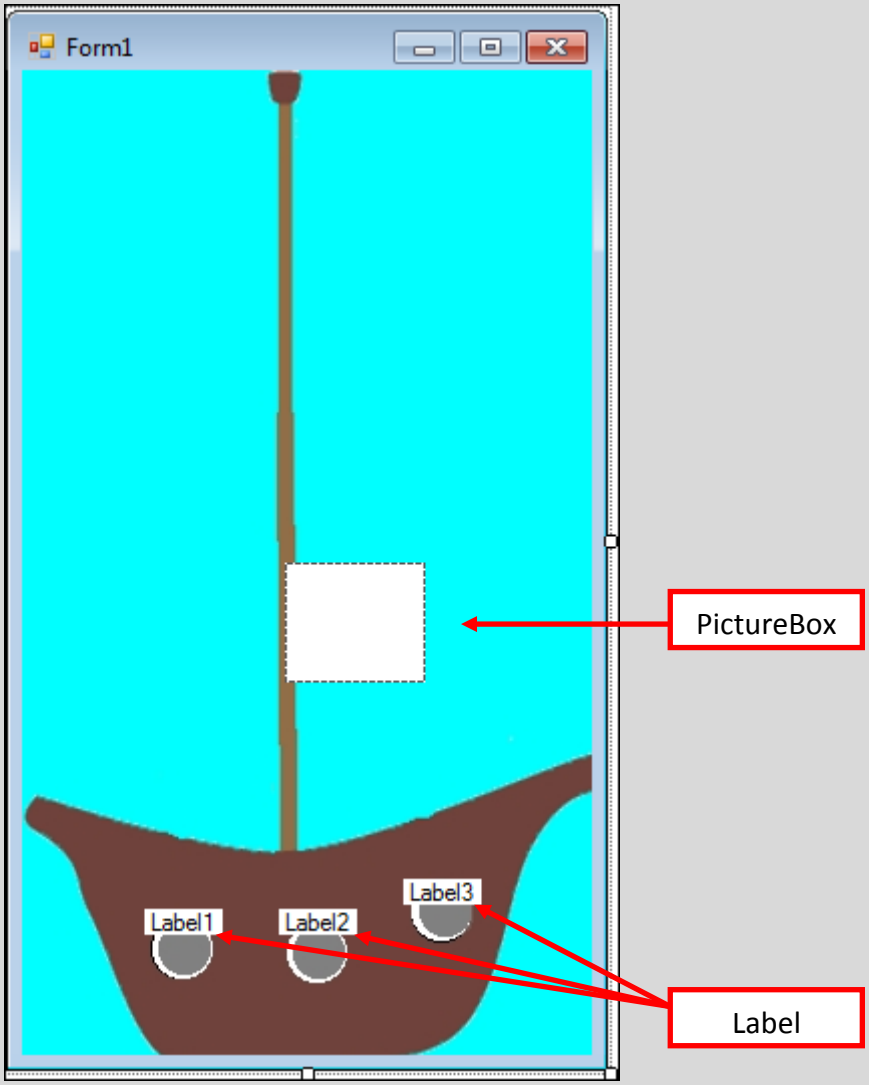
- **Size** = 300; 530
- **BackGroundImage**: prendiamo l'immagine **Vascello.jpg** che si trova nella cartella **Documenti / A scuola con VB 2010 / Immagini / Rotella mouse**.

Inseriamo nelle risorse del programma le immagini delle tre bandiere che si trovano nella stessa cartella:





Ora inseriamo nel form un controllo PictureBox, posizionato sull'albero maestro, e tre controlli Label, uno per ogni boccaporto, come in questa immagine:



Le proprietà di questi controlli sono impostate dal codice.
A ogni boccaporto corrisponde la scelta di una bandiera.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        With Me
            ' Blocca il form al centro dello schermo:
            .FormBorderStyle = Windows.Forms.FormBorderStyle.FixedToolWindow
            .StartPosition = FormStartPosition.CenterScreen
        End With

        ' Assegnazione delle proprietà al controllo PictureBox:
        With PictureBox1
            .Size = New Size(70, 60)
            .BackColor = Color.Transparent
            .Location = New Point(134, 250)
        End With

        ' Assegnazione delle proprietà ai tre controlli Label:
        For Each Control In Me.Controls
            If TypeOf Control Is Label Then
                Control.autosize = False
                Control.text = ""
                Control.backcolor = Color.Transparent
                Control.size = New Size(40, 40)
                Control.cursor = Cursors.Hand
            End If
        Next

    End Sub
```

```
    Private Sub Label1_Click(sender As Object, e As System.EventArgs) Handles
Label1.Click
        ' Scelta di una delle tre bandiere:
        PictureBox1.Image = My.Resources.Bandiera_Pirati

    End Sub
```

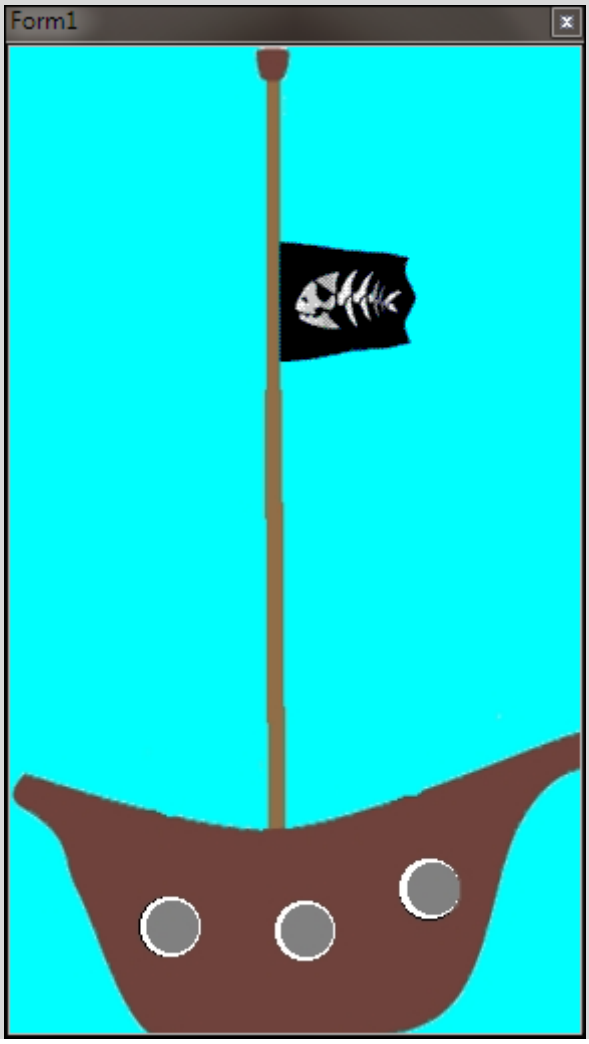
```
    Private Sub Label2_Click(sender As Object, e As System.EventArgs) Handles
Label2.Click
        ' Scelta di una delle tre bandiere:
        PictureBox1.Image = My.Resources.Bandiera_Sole

    End Sub
```

```
    Private Sub Label3_Click(sender As Object, e As System.EventArgs) Handles
Label3.Click
        ' Scelta di una delle tre bandiere:
        PictureBox1.Image = My.Resources.Bandiera_Spiaggia

    End Sub
```

```
    Private Sub Form1_MouseWheel(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseWheel
        ' Gestione del movimento della bandiera sull'albero maestro.
        ' Definizione dei limiti in alto e in basso:
        If PictureBox1.Top > 280 Then PictureBox1.Top = 280
```

```
        If PictureBox1.Top < 30 Then PictureBox1.Top = 30

        ' Spostamento della bandiera.
        ' Il numero indicato dalla proprietà e.delta corrisponde alle linee che
la rotella fa scorrere a ogni suo scatto;
        ' questo numero viene qui diviso per 40, per rallentare gli spostamenti
della bandiera.
        PictureBox1.Top += e.Delta / 40

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



182: Gli eventi MouseDown e MouseUp.

Gli eventi **MouseDown** e **MouseUp** accadono, rispettivamente, quando l'utente del programma abbassa uno dei due pulsanti del mouse oppure quando rilascia questo pulsante.
L'evento del *clic* del mouse, dunque, si colloca al centro, in ordine di tempo, tra l'evento **MouseDown** che lo precede e l'evento **MouseUp** che lo segue.

L'evento **MouseDown** consente al programmatore di controllare alcuni elementi che invece l'evento **Click** non prende in considerazione.
In particolare, con l'evento **MouseDown** è possibile sapere:

- quale pulsante del mouse è stato premuto;
- in quale punto del form o dell'oggetto è stato effettuato il *clic*;
- se sono stati effettuati entrambi i *clic* di un evento doppio *clic*.

Nell'esercizio che segue vedremo un esempio di utilizzo di questi dati.

Esercizio 121: Gli eventi MouseDown e MouseUp.

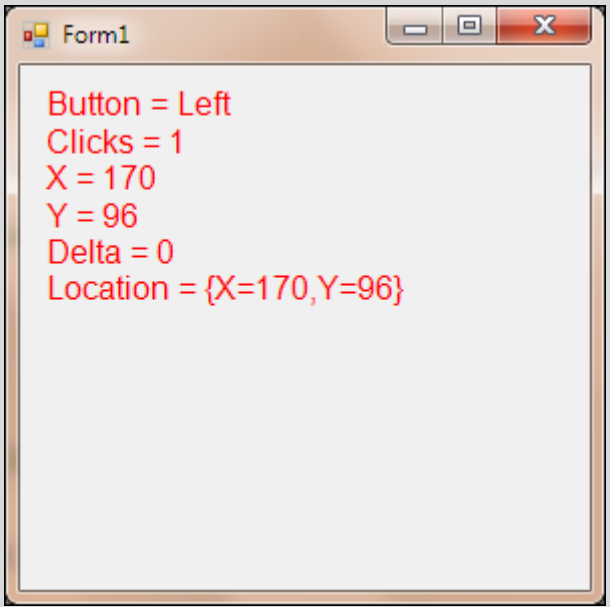
Questo programma memorizza alcune informazioni relative al mouse ed alla sua posizione. Quando l'utente clicca il form con il mouse, il programma visualizza un testo con tali dati.
Apriamo un nuovo progetto.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim Testo As String

    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown, Me.MouseWheel
        ' All'abbassarsi del mouse, questa procedura crea un testo con una serie
di dati relativi al mouse.
        ' Cancella il testo precedente:
        Testo = ""

        ' Inserisci il dato relativo al pulsante premuto e vai a capo:
        Testo &= "Pulsante premuto = " & e.Button.ToString & vbCrLf
        ' Inserisci il dato relativo al numero dei clic in un evento doppio clic:
tale numero può essere 1 (il doppio clic non è completo) o 2 (il doppio clic è
completo):
        Testo &= "Numero di clic effettuati = " & e.Clicks & vbCrLf
        ' Inserisci il dato relativo alla posizione a sinistra del punto in cui
il mouse si abbassa:
        Testo &= "Posizione X = " & e.X & vbCrLf
        ' Inserisci il dato relativo alla posizione in altezza del punto in cui
il mouse si abbassa:
        Testo &= "Posizione Y = " & e.Y & vbCrLf
```

```
        ' Inserisci il dato relativo al numero di linee che scorrono a ogni clic
della rotella:
        Testo &= "Scorrimento della rotella = " & e.Delta & vbCrLf
        ' Inserisci i dati relativi alla posizione del punto in cui il mouse si
abbassa:
        Testo &= "Posizione = " & e.Location.ToString

        ' Cancella il form, causa l'evento Form1.Paint
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Carattere As New Font("Arial", 12)
        Dim Pennello As Brush = Brushes.Red
        Dim PuntoInizio As New Point(10, 10)
        e.Graphics.DrawString(Testo, Carattere, Pennello, PuntoInizio)

    End Sub

    Private Sub Form1_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseUp

        ' Al rialzarsi del mouse, esegui un bip:
        Beep()

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Tra i dati forniti dall'evento **MouseDown** hanno una particolare rilevanza le proprietà

- e.X
- e.Y

- e.Location

Queste proprietà forniscono le coordinate del punto in cui si è verificato il *clic* del mouse.
Ricordiamo che le due variabili X e Y memorizzano la posizione del puntatore del mouse sul form:

- e.X registra la distanza dal bordo sinistro;
- e.Y registra la distanza dal bordo superiore;
- e.Location invece è una struttura che registra entrambe le proprietà e.X ed e.Y

L'uso di questi dati in un programma consente di tenere conto degli spostamenti del mouse.
Ecco, ad esempio, un programma che disegna un cerchio attorno ad un punto non predefinito dal programmatore, ma scelto di volta in volta dall'utente:

```
Public Class Form1
    ' Colloca il punto iniziale al di fuori del Form1:
    Dim PuntoCliccato As Point = New Point(-25, -25)
```

```
    Private Sub Form1_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown

        ' Registra il punto in cui è stato effettuato il clic del mouse:
        PuntoCliccato = e.Location

        ' Cancella il contenuto del form e causa l'evento Form1.Paint:
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Migliora la qualità grafica:
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        ' Crea un'area destinata a contenere il cerchio. L'area è di 50 pixel per
lato.
        ' Per fare sì che il punto cliccato con il mouse sia al centro del
cerchio,
        ' l'area viene collocata a 25 pixel a sinistra e 25 pixel sopra il punto
cliccato con il mouse.
        Dim AreaCerchio As Rectangle
        AreaCerchio = New Rectangle(New Point(PuntoCliccato.X - 25,
PuntoCliccato.Y - 25), New Size(50, 50))

        Dim Pennello As New SolidBrush(Color.Blue)
        e.Graphics.FillEllipse(Pennello, AreaCerchio)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

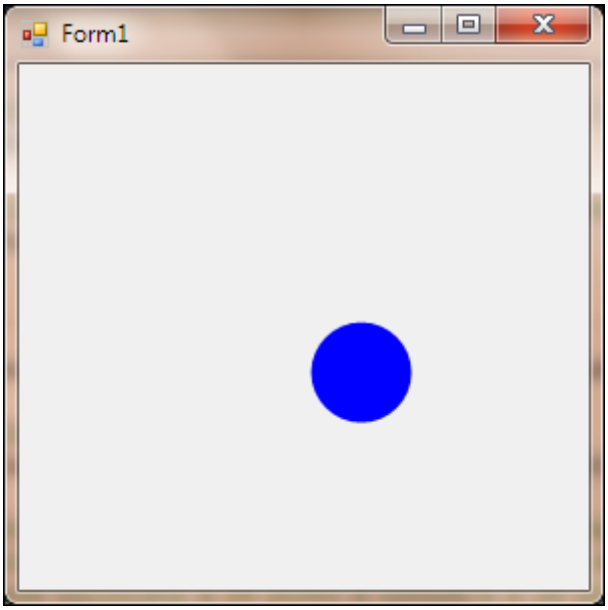


Figura 242: Uso della proprietà e.Location.

183: L'evento MouseMove.

L'evento **MouseMove** accade quando l'utente muove il puntatore del mouse sull'area occupata da un form o da un controllo.
E' un evento continuativo, che si ripete durante tutta la fase di movimento del mouse, registrando in continuazione le variazioni della posizione del mouse.
Come l'evento **MouseDown**, anche **MouseMove** registra quale pulsante del mouse è eventualmente abbassato.
Di seguito vedremo un programma che gestisce il trascinamento del mouse, cioè lo spostamento del mouse su un oggetto tenendo un pulsante del mouse premuto.
In questo programma il trascinamento del mouse è utilizzato per spostare un cerchio nel form: l'utente può disegnare un cerchio con il *clic* del mouse nel punto in cui preferisce e, tenendo il mouse abbassato, può trascinare questo cerchio nel form.
Per evitare lo sfarfallìo dell'immagine durante gli spostamenti del cerchio, ricordiamo di impostare questa proprietà del Form1:

- **DoubleBuffered** = True.

```
Public Class Form1

    ' Colloca il punto iniziale al di fuori del Form1:
    Dim PuntoCliccato As Point = New Point(-25, -25)
```

```
    Private Sub Form1_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown
```

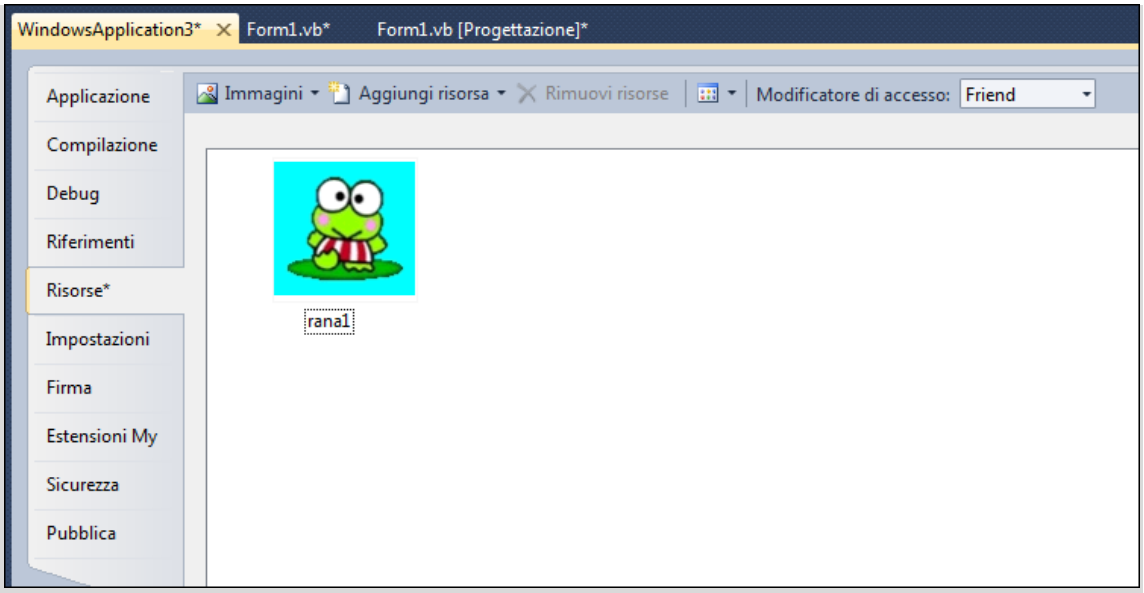
```
        ' Memorizza il punto in cui è stato effettuato il clic,
        ' cambia il cursorse del mouse e
        ' cancella il form e causa l'evento Form1.Paint:
        PuntoCliccato = e.Location
        Me.Cursor = Cursors.Hand
        Me.Invalidate()

    End Sub
```

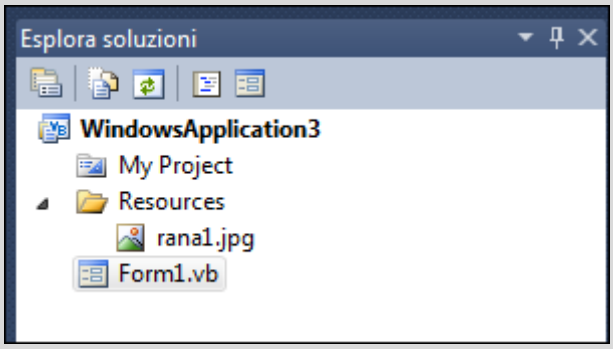
```
    Private Sub Form1_MouseMove(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove

        ' Se l'utente muove il mouse tenendo il pulsante sinistro abbassato...
        If e.Button = Windows.Forms.MouseButtons.Left Then
            ' ... memorizza il punto in cui trova il mouse:
            PuntoCliccato = e.Location
            ' cancella il form e causa l'evento Form.Paint:
            Me.Invalidate()
        End If

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim AreaCerchio As Rectangle
        AreaCerchio = New Rectangle(New Point(PuntoCliccato.X - 20,
PuntoCliccato.Y - 20), New Size(40, 40))

        Dim Pennello As New SolidBrush(Color.Blue)
        e.Graphics.FillEllipse(Pennello, AreaCerchio)

    End Sub

    Private Sub Form1_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseUp

        Me.Cursor = Cursors.Default

    End Sub

End Class
```

Esercizio 122: L'evento MouseMove.

In questo programma il mouse viene utilizzato per trascinare un'immagine nel form: quando l'utente muove il mouse tenendo il tasto sinistro abbassato, l'immagine segue lo spostamento del puntatore.
Apriamo un nuovo progetto e inseriamo nelle risorse del programma l'immagine **rana1.jpg** che si trova nella cartella **Documenti \ A scuola con VB \ Immagini**:





Proprietà del Form1:

- **BackgroundColor = Cyan**

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Colloca il punto iniziale al di fuori del Form1:
    Dim PuntoCliccato As Point = New Point(50, 50)
    Dim ImmagineVirtuale As Bitmap = My.Resources.rana1
```

```
    Private Sub Form1_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown

        PuntoCliccato = e.Location
        Me.Cursor = Cursors.Hand
        Me.Invalidate()

    End Sub
```

```
    Private Sub Form1_MouseMove(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove

        ' Se l'utente muove il mouse tenendo il pulsante sinistro abbassato...

        If e.Button = Windows.Forms.MouseButtons.Left Then
```

```
            ' ... memorizza il punto in cui trova il mouse:
            PuntoCliccato = e.Location

            ' cancella il form e causa l'evento Form.Paint:
            Me.Invalidate()

        End If

    End Sub
```

```
    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        e.Graphics.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias

        Dim AngoloSinistraAlto As New Point(PuntoCliccato.X - 58, PuntoCliccato.Y
- 50)
        e.Graphics.DrawImage(ImmagineVirtuale, AngoloSinistraAlto)

    End Sub
```

```
    Private Sub Form1_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseUp

        Me.Cursor = Cursors.Default

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

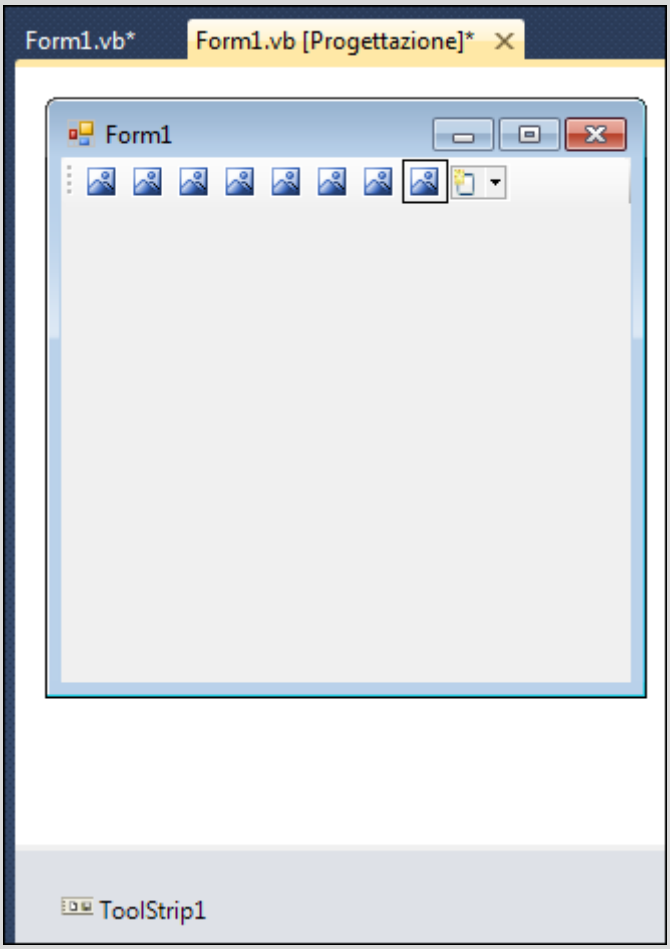
Nel prossimo esercizio vedremo un altro utilizzo dell'evento MouseMove, in questo caso per scrivere o disegnare a mano libera, con il mouse, sul form.

Esercizio 123: Disegni a mouse libero.

In questo programma l'utente può tracciare linee, scrivere o disegnare sul form muovendo il mouse con il pulsante sinistro premuto.
Il programma si basa sull'evento **MouseMove** e sulla memorizzazione di due punti (la posizione precedente e la posizione attuale del mouse) e unisce i due punti con una linea colorata.
La successione delle operazioni è questa:

- quando il mouse si muove sul form, l'evento MouseMove registra la nuova posizione del mouse;
- il comando DrawLine traccia una linea tra la posizione precedente e la posizione attuale del mouse;
- la posizione attuale è memorizzata come posizione precedente, in attesa del prossimo movimento del mouse e della ripetizione del ciclo.

Inseriamo nel Form1 un controllo ToolStrip con 8 pulsanti Button che serviranno all'utente per scegliere il colore del disegno:



Le proprietà di questi 8 pulsanti sono impostate dal codice del programma.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim PuntoCliccato As Point = New Point(0, 0)
    Dim PuntoPrecedente As Point = New Point(0, 0)
    Dim Colore As Color = Color.Black

    ' Crea una superficie grafica corrispondente all'area del form:
    Dim AreaDisegno As Graphics = Me.CreateGraphics
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' Migliora la qualità della grafica:
        AreaDisegno.SmoothingMode = Drawing2D.SmoothingMode.HighQuality

        ' Imposta le proprietà degli 8 pulsanti nel ToolStrip1:
        ToolStripButton1.BackColor = Color.Black
        ToolStripButton1.ToolTipText = "Nero"
        ToolStripButton2.BackColor = Color.Blue
        ToolStripButton2.ToolTipText = "Blu"
        ToolStripButton3.BackColor = Color.Cyan
        ToolStripButton3.ToolTipText = "Azzurro"
        ToolStripButton4.BackColor = Color.Green
        ToolStripButton4.ToolTipText = "Verde"
        ToolStripButton5.BackColor = Color.Yellow
        ToolStripButton5.ToolTipText = "Giallo"
        ToolStripButton6.BackColor = Color.Magenta
        ToolStripButton6.ToolTipText = "Viola"
        ToolStripButton7.BackColor = Color.Red
        ToolStripButton7.ToolTipText = "Rosso"
        ToolStripButton8.BackColor = Color.White
        ToolStripButton8.ToolTipText = "Bianco"

    End Sub
```

```
    Private Sub Form1_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown

        If e.Button = Windows.Forms.MouseButtons.Right Then
            ' Se l'utente ha premuto il tasto destro del mouse, il contenuto del
form viene cancellato:
            Me.Invalidate()
        Else
            ' Altrimenti il programma memorizza il punto in cui è stato
effettuato il clic,
            ' e cambia il cursorse del mouse:
            PuntoPrecedente = e.Location
            Me.Cursor = Cursors.Hand
        End If

    End Sub
```

```
    Private Sub Form1_MouseMove(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseMove

        ' Se l'utente muove il mouse tenendo il pulsante sinistro abbassato...
```

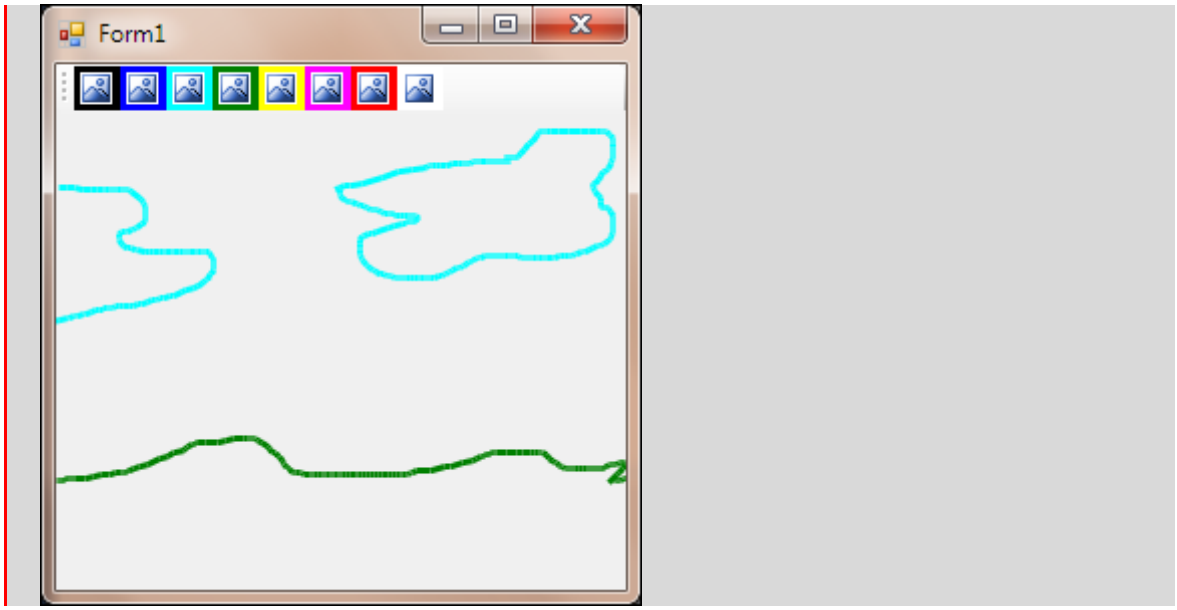
```
        If e.Button = Windows.Forms.MouseButtons.Left Then
            ' ... memorizza il nuovo punto in cui trova il mouse:
            PuntoCliccato = e.Location
        End If

        Dim Penna As New Pen(Colore, 3)

        ' Traccia una linea tra il punto precedente e il punto attuale in cui si
trova il mouse:
        AreaDisegno.DrawLine(Penna, PuntoPrecedente, PuntoCliccato)

        ' Memorizza il punto attuale come punto precedente, per la prossima
linea:
        PuntoPrecedente = PuntoCliccato

    End Sub
```

```
    Private Sub Form1_MouseUp(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles Me.MouseUp

        Me.Cursor = Cursors.Default

    End Sub
```

```
    Private Sub ToolStripButton1_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripButton1.Click,
        ToolStripButton2.Click,
        ToolStripButton3.Click,
        ToolStripButton4.Click,
        ToolStripButton5.Click,
        ToolStripButton6.Click,
        ToolStripButton7.Click,
        ToolStripButton8.Click

        Colore = sender.backcolor

    End Sub
```

```
    Private Sub Form1_Resize(sender As Object, e As System.EventArgs) Handles
Me.Resize

        ' ad ogni modifica delle dimensioni del Form1 da parte dell'utente, è
modificata l'area di disegno:
        AreaDisegno = Me.CreateGraphics

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



184: L'evento DragDrop.

L'evento **DragDrop** (letteralmente: *trascina* e *lascia cadere*) è l'azione che l'utente di un programma compie quando, dopo avere ***catturato*** e ***trascinato*** un oggetto premendo il tasto sinistro del mouse, lo lascia ***cadere***, rilasciando il tasto del mouse, su un altro oggetto.
E' un movimento che dà l'illusione dello spostamento fisico di un oggetto da una parte all'altra del monitor, ed è ormai entrato nella consuetudine degli utenti di computer.
Dal punto di vista tecnico, questo trascinamento è in realtà un'operazione simile al **copia e incolla** che si effettua con i testi: in entrambi i casi abbiamo una sorgente, dalla quale l'oggetto è copiato, e una destinazione, nella quale l'oggetto è incollato; nel mezzo delle due operazioni, la memoria del computer conserva una copia dell'oggetto in questione.

In un programma VB, l'operazione si svolge in queste fasi successive:

1. **Clic del mouse sull'oggetto sorgente**: il comando **DoDragDrop** indica qual è l'oggetto sorgente, **cosa** deve essere trasferito dell'oggetto sorgente e con quale **modalità** (nei nostri esercizi questa modalità sarà sempre **copy**).
   L'oggetto sorgente è copiato automaticamente da VB in un oggetto temporaneo di nome **Data**.
2. **Trascinamento del mouse con il pulsante sinistro**: quando il mouse entra nell'area di un controllo, il comando **Data.GetDataPresent** controlla se questo controllo è di tipo idoneo a ricevere il contenuto dell'oggetto temporaneo **Data.**
3. **Rilascio del mouse sul controllo destinatario**: il contenuto dell'oggetto **Data** è copiato nell'oggetto destinatario con il comando **Data.GetData**, che conclude lo spostamento.

Il programma di esempio, che vedremo di seguito, mostra in modo dettagliato il succedersi di queste fasi.
Richiede che nel Form1 siano presenti tre controlli:

- un controllo Label1 con la proprietà **Text = Primo testo**
- un controllo Label2 con la proprietà **Text = Secondo testo**
- un controllo TextBox1 con la proprietà **AllowDrop = True**

La proprietà **AllowDrop = True** (= *ricezione consentita*), indica che il controllo in questione accetta le azioni di **DragDrop**.

Nella prima procedura vediamo la fase 1.
In questa procedura l'operazione di trascinamento è preparata con il comando il comando **DoDragDrop**, che comprende due parametri:

- il contenuto della Label11;
- la modalità copia.

Con questo comando, il programma crea automaticamente una copia del testo della Label1 e la colloca in un oggetto temporaneo, non visibile, denominato **Data.**
Ora qualsiasi controllo presente nel form, che abbia la proprietà **AllowDrop = True** (*ricezione consentita*), è un potenziale destinatario dell'operazione DragDrop.

```
    Private Sub Label1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Label1.MouseMove, Label2.MouseMove

        ' Il tasto sinistro del mouse premuto e spostato su una delle due label
memorizza il suo testo
        ' e lo prepara per essere copiato in un oggetto destinatario:

        If e.Button = Windows.Forms.MouseButtons.Left Then
            sender.DoDragDrop(sender.Text, DragDropEffects.Copy)
        End If

    End Sub
```

Nella seconda procedura vediamo la fase 2.
Quando il mouse passa sopra un controllo potenziale destinatario, si attiva l'evento **DragEnter** di questo controllo.
Nella procedura che gestisce questo evento, sono inseriti questi due comandi:

- il comando **Data.GetDataPresent**, per controllare se i dati contenuti nell'oggetto temporaneo **Data** sono di tipo idoneo a essere ricevuti dal destinatario;
- il comando **e.Effect**, per impostare il cursore del mouse secondo l'azione di trascinamento che si sta preparando.

```
    Private Sub TextBox1_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox1.DragEnter

        ' Il puntatore del mouse entra nell'area del TextBox1, destinatario del
trascinamento.
        ' Il programma controlla se il contenuto dell'oggetto Data è idoneo a
essere copiato nel TextBox e in caso affermativo cambia il cursore del mouse:
```

```
        If e.Data.GetDataPresent(DataFormats.Text) Then
            ' Visualizza il cursore della copia:
            e.Effect = DragDropEffects.Copy
        End If

    End Sub
```

Nella terza procedura vediamo l'ultima fase del trascinamento.
Quando l'utente rilascia il pulsante del mouse sopra un controllo destinatario valido, si attiva l'evento **DragDrop** di questo controllo.
Nella procedura che gestisce questo evento, il comando **Data.GetData** copia il contenuto dell'oggetto temporaneo **Data**, e lo incolla nell'oggetto destinatario.

```
    Private Sub TextBox1_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox1.DragDrop

        ' Al rilascio del pulsante del mouse, incolla il testo nel TextBox1:
        TextBox1.Text = e.Data.GetData(DataFormats.Text)

    End Sub
```

Ecco il codice completo e un'immagine di questo programma in esecuzione:

```
Public Class Form1

    Private Sub Label1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Label1.MouseMove, Label2.MouseMove

        ' Il tasto sinistro del mouse premuto e spostato su una delle due label
memorizza il suo testo
        ' e lo prepara per essere copiato in un oggetto destinatario:

        If e.Button = Windows.Forms.MouseButtons.Left Then
            sender.DoDragDrop(sender.Text, DragDropEffects.Copy)
        End If

    End Sub

    Private Sub TextBox1_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox1.DragEnter

        ' Il puntatore del mouse entra nell'area del TextBox1, destinatario del
trascinamento.
        ' Il programma controlla se il contenuto dell'oggetto Data è idoneo a
essere copiato nel TextBox e in caso affermativo cambia il cursore del mouse:

        If e.Data.GetDataPresent(DataFormats.Text) Then
            ' Visualizza il cursore della copia:
            e.Effect = DragDropEffects.Copy
        End If

    End Sub

    Private Sub TextBox1_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles TextBox1.DragDrop
```

```
        ' Al rilascio del pulsante del mouse, incolla il testo nel TextBox1:
        TextBox1.Text = e.Data.GetData(DataFormats.Text)

    End Sub

End Class
```

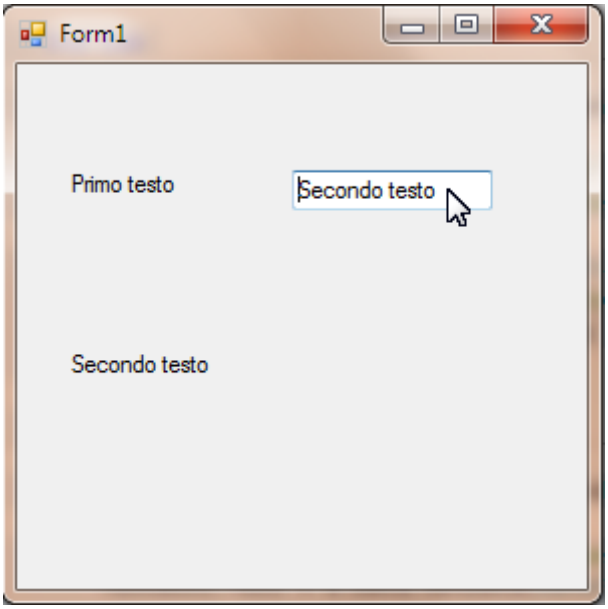


Figura 243: L'evento DragDrop.

Nei prossimi esercizi vedremo quattro diversi esempi di utilizzo dell'evento **DragDrop**:

- Esercizio 124: DragDrop di un testo tra due Label.
- Esercizio 125: DragDrop d'immagini tra due PictureBox.
- Esercizio 126: DragDrop d'immagini tra cinque PictureBox.
- Esercizio 127: DragDrop di item tra due Listbox.

Esercizio 124: DragDrop di un testo tra due Label.

In questo programma abbiamo due controlli Label, rispettivamente con la proprietà **Text = Alessandro Manzoni** e **I promessi sposi**.

La prima Label è l'oggetto sorgente, mentre la seconda Label è l'oggetto destinazione.
La proprietà **AllowDrop** della seconda label è dunque impostata = **True**.
Quando il programma è in esecuzione, è possibile fare un *clic* sulla prima label e trascinare la scritta "Alessandro Manzoni" sulla scritta "I promessi sposi". A quel punto, rilasciando il mouse il testo contenuto nella Label sorgente viene copiato nella Label destinazione.
Apriamo un nuovo progetto, con due controlli Label impostati come descritto sopra.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Label1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Label1.MouseMove

        ' Se l'utente muove il mouse con il tasto sinistro premuto, memorizza il
testo della Label1:

        If e.Button = Windows.Forms.MouseButtons.Left Then
            Label1.DoDragDrop(Label1.text, DragDropEffects.Copy)
        End If

    End Sub
```

```
    Private Sub Label2_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles Label2.DragEnter

        ' Il puntatore del mouse entra nell'area della Label2.
        ' Il programma controlla se in memoria è contenuto un testo:
        If e.Data.GetDataPresent(DataFormats.Text) Then
            ' Questo comando visualizza il cursore della copia:
            e.Effect = DragDropEffects.Copy
        End If

    End Sub
```

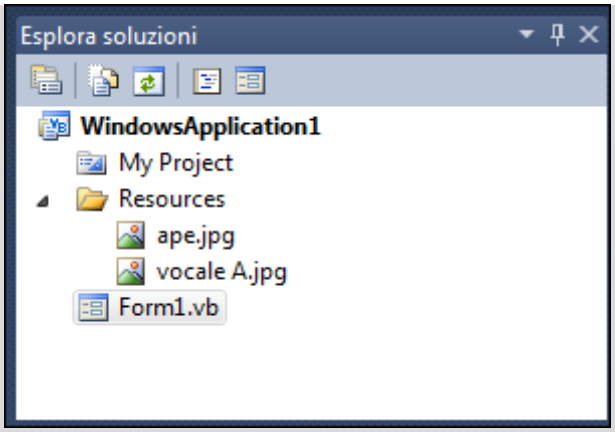
```
    Private Sub Label2_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles Label2.DragDrop

        ' Al rilascio del pulsante del mouse, incolla il testo nella Label2,
oggetto destinazione:
        Label2.Text = e.Data.GetData(DataFormats.Text)
        Label1.Visible = False
        MsgBox("Esatto!")

        ' Ripristina la situazione di partenza:
        Label2.Text = "I promessi sposi"
        Label1.Visible = True

    End Sub

End Class
```

Nel prossimo esercizio l'evento DragOver è utilizzato per spostare un'immagine tra due controlli PictureBox.

Esercizio 125: DragDrop d'immagini tra due PictureBox.

Apriamo un nuovo progetto.
Inseriamo nel form due controlli PictureBox; scopo del programma è consentire il trascinamento di un'immagine, con il mouse, dal primo al secondo PictureBox.
La proprietà **AllowDrop** del controllo PictureBox2 deve essere impostata = **True**, ma questo non può essere fatto dalla Finestra Proprietà, in quanto la proprietà **AllowDrop** non è disponibile in questa finestra per i controlli PictureBox.
L'impostazione di questa proprietà va dunque scritta nel codice del programma, all'avvio del programma o comunque prima di effettuare l'operazione di trascinamento.
Inseriamo nelle risorse del programma queste due immagini che si trovano nella cartella **Documenti \ A scuola con VB \ Immagini \ Vocali**:

- ape.jpg e
- Vocale A.jpg



Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Me.Load

        ' Imposta le proprietà dei due controlli PictureBox all'avvio del
programma:
        PictureBox1.SizeMode = PictureBoxSizeMode.AutoSize
        PictureBox2.SizeMode = PictureBoxSizeMode.AutoSize
        PictureBox1.Image = My.Resources.vocale_A
        PictureBox2.Image = My.Resources.ape

        ' Imposta il controllo PictureBox2 come destinazione del trascinamento di
dati:
        PictureBox2.AllowDrop = True

    End Sub
```

```
    Private Sub PictureBox1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseMove

        ' Se l'utente muove il mouse con il tasto sinistro premuto, memorizza
l'immagine contenuta
        ' nel controllo PictureBox1:
        If e.Button = Windows.Forms.MouseButtons.Left Then
            PictureBox1.DoDragDrop(PictureBox1.Image, DragDropEffects.Copy)
        End If

    End Sub
```

```
    Private Sub PictureBox2_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles PictureBox2.DragEnter

        ' Quando il puntatore del mouse entra nell'area del PictureBox2 per
l'operazione DragDrop,
        ' modifica il cursore del mouse:
        e.Effect = DragDropEffects.Copy

    End Sub
```

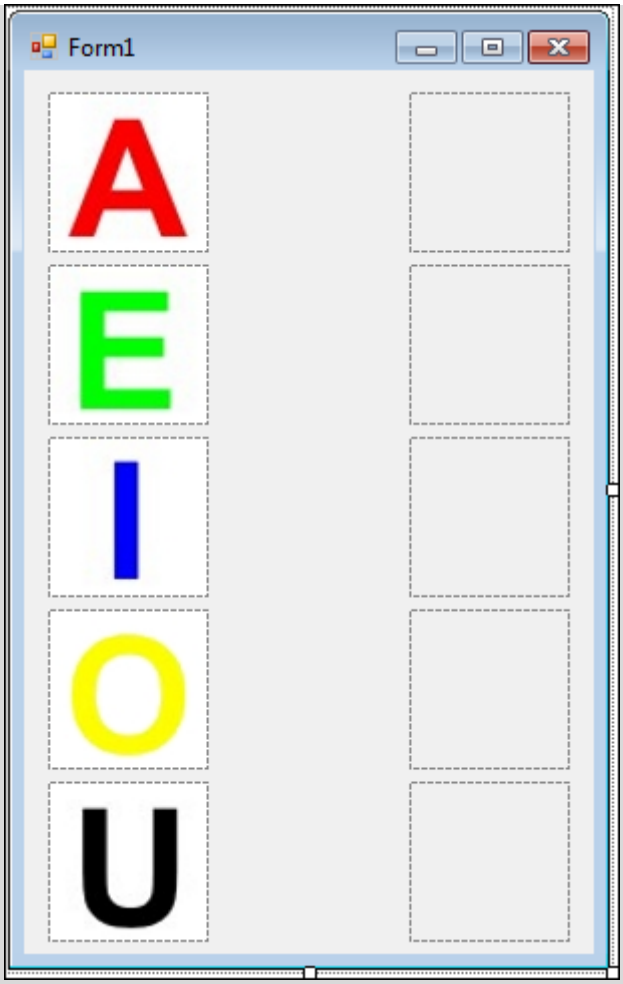
```
    Private Sub PictureBox2_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles PictureBox2.DragDrop
```

```
        ' Quando il pulsante del mouse è rilasciato, copia in PictureBox2
l'immagine memorizzata:
        PictureBox2.Image = e.Data.GetData(DataFormats.Bitmap)
        PictureBox1.Visible = False

        MsgBox("OK")
        ' Ripristina la situazione iniziale:
        PictureBox1.Visible = True
        PictureBox2.Image = My.Resources.ape

    End Sub

End Class
```

Nel prossimo esercizio vedremo un programma più complesso, che consente il trascinamento di immagini tra diversi PictureBox.

Esercizio 126: DragDrop d'immagini tra cinque PictureBox.

In questo programma abbiamo nel form dieci controlli PictureBox con dieci immagini: le cinque immagini nella colonna di sinistra sono le immagini sorgente, le cinque immagini nella colonna di destra sono le immagini destinazione.
L'utente deve trascinare ogni immagine sorgente sull'immagine destinazione corrispondente.
Apriamo un nuovo progetto e inseriamo nel form dieci controlli PictureBox, come in questa immagine:



Proprietà del Form1:

- **Size** = 300; 480

Proprietà di tutti i PictureBox:

- **Size** = 80; 80

Impostiamo la proprietà **Name** dei cinque PictureBox della colonna di sinistra con questi nomi:

- Sorgente0
- Sorgente1
- Sorgente2
- Sorgente3
- Sorgente4

Impostiamo la proprietà **Image** di questi PictureBox con le immagini delle vocali che si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Vocali**.

Impostiamo la proprietà **Name** dei cinque PictureBox della colonna di destra con questi nomi:

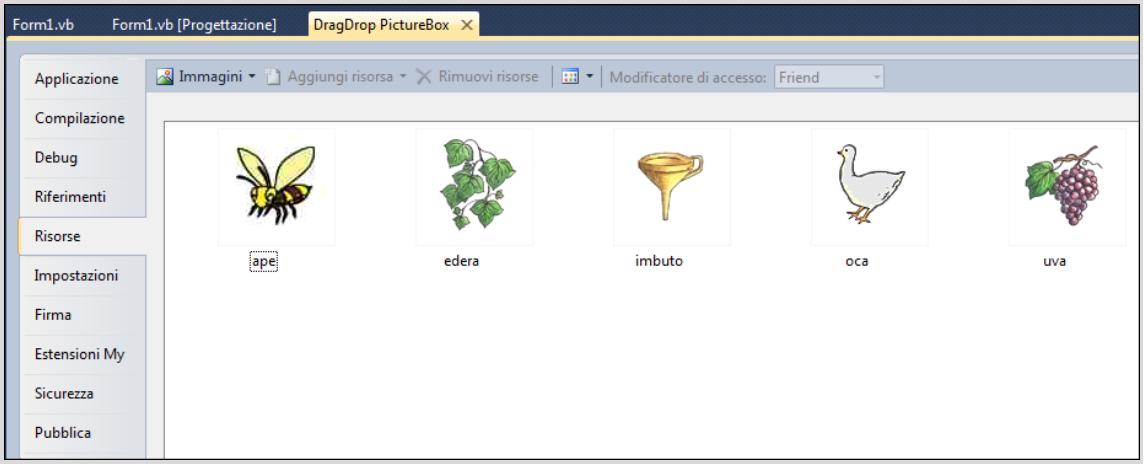
- Destinazione0
- Destinazione1
- Destinazione2
- Destinazione3

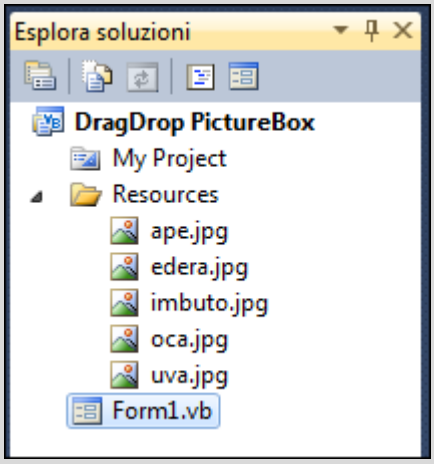
- Destinazione4

Inseriamo nelle risorse del programma le immagini

- ape.jpg
- edera.jpg
- imbuto.jpg
- oca.jpg
- uva.jpg

che si trovano nella cartella **Documenti / A scuola con VB 2010 / Immagini / Vocali**.





Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Dim ImmagineOrigine As PictureBox
    Dim PosizioneSorgente(4) As Point
    Dim PosizioneDestinazione(4) As Point
    Dim RisposteEsatte As Integer = 0
```

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        ' Memorizza le posizioni dei cinque controlli Sorgente e dei cinque
controlli Destinazione:
```

```
        PosizioneSorgente(0) = Sorgente0.Location
        PosizioneSorgente(1) = Sorgente1.Location
        PosizioneSorgente(2) = Sorgente2.Location
        PosizioneSorgente(3) = Sorgente3.Location
        PosizioneSorgente(4) = Sorgente4.Location

        PosizioneDestinazione(0) = Destinazione0.Location
        PosizioneDestinazione(1) = Destinazione1.Location
        PosizioneDestinazione(2) = Destinazione2.Location
        PosizioneDestinazione(3) = Destinazione3.Location
        PosizioneDestinazione(4) = Destinazione4.Location

        ' Assegna ai controlli PictureBox un tag (cartellino) che servirà per
controllare
        ' se l'immagine destinazione corrisponde all'oggetto sorgente:

        Sorgente0.Tag = "A"
        Sorgente1.Tag = "E"
        Sorgente2.Tag = "I"
        Sorgente3.Tag = "O"
        Sorgente4.Tag = "U"
        Destinazione0.Tag = "A"
        Destinazione1.Tag = "E"
        Destinazione2.Tag = "I"
        Destinazione3.Tag = "O"
        Destinazione4.Tag = "U"

        ' Abilita le immagini destinazione a ricevere dati:
        Destinazione0.AllowDrop = True
        Destinazione1.AllowDrop = True
        Destinazione2.AllowDrop = True
        Destinazione3.AllowDrop = True
        Destinazione4.AllowDrop = True

        Call AssegnaPosizioni()

    End Sub
```

```
    Private Sub Sorgente1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Sorgente0.MouseDown,
Sorgente1.MouseDown, Sorgente2.MouseDown, Sorgente3.MouseDown,
Sorgente4.MouseDown

        ' Con l'evento MouseDown su uno dei controlli Sorgente, inizia una
operazione di trascinamento:
        ImmagineOrigine = sender

    End Sub
```

```
    Private Sub Sorgente1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles Sorgente0.MouseMove,
Sorgente1.MouseMove, Sorgente2.MouseMove, Sorgente3.MouseMove,
Sorgente4.MouseMove

        ' Se è premuto il pulsante sinistro del mouse...
        If e.Button = Windows.Forms.MouseButtons.Left Then

            ' Inizia il trascinamento del controllo originale:
            sender.DoDragDrop(sender.Image, DragDropEffects.Copy)
        End If
```

```
    End Sub
```

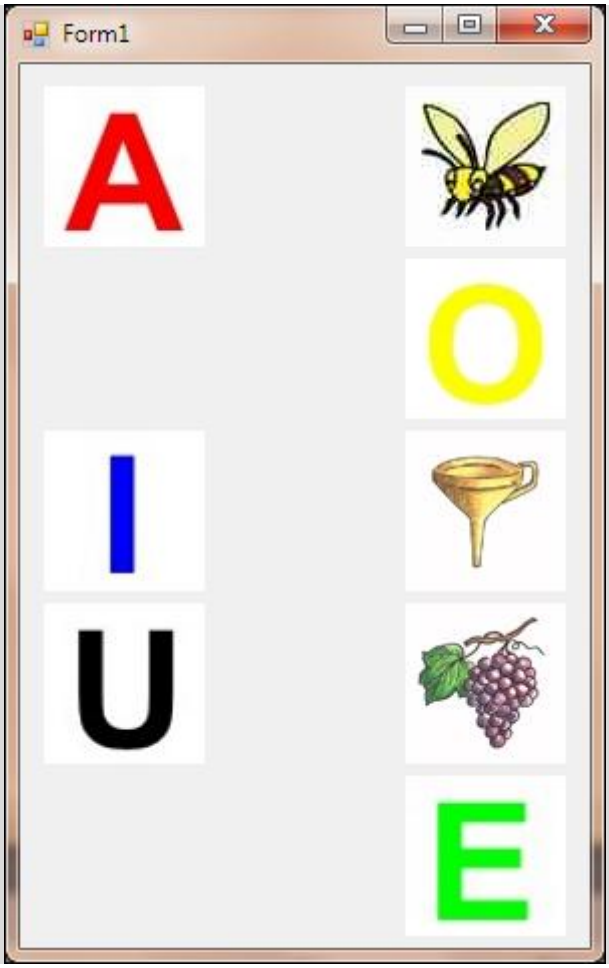
```
    Private Sub Destinazione1_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles Destinazione0.DragEnter,
Destinazione1.DragEnter, Destinazione2.DragEnter, Destinazione3.DragEnter,
Destinazione4.DragEnter

        e.Effect = DragDropEffects.Copy

    End Sub
```

```
    Private Sub Destinazione1_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles Destinazione0.DragDrop,
Destinazione1.DragDrop, Destinazione2.DragDrop, Destinazione3.DragDrop,
Destinazione4.DragDrop

        ' Copia l'immagine del controllo Sorgente nel controllo Destinazione
        sender.Image = e.Data.GetData(DataFormats.Bitmap)

        ' Nasconde l'immagine Sorgente:
        ImmagineOrigine.Visible = False
        Beep()
        RisposteEsatte += 1

        If RisposteEsatte = 5 Then

            If MsgBox("Benissimo. Vuoi provare ancora?", MsgBoxStyle.YesNo Or
MsgBoxStyle.Exclamation) = MsgBoxResult.Yes Then
                Call AssegnaPosizioni()
            Else
                End
            End If

        End If

    End Sub
```

```
    Private Sub AssegnaPosizioni()
        Dim Sorteggio As New Random
        Randomize()

        RisposteEsatte = 0

        ' Rimescola le posizioni delle immagini sorgente:
        For Contatore As Integer = 0 To 4
            Dim Cambio = Sorteggio.Next(5)
            Dim PosizioneProvvisoria = PosizioneSorgente(Cambio)
            PosizioneSorgente(Cambio) = PosizioneSorgente(Contatore)
            PosizioneSorgente(Contatore) = PosizioneProvvisoria
        Next

        ' e le assegna ai PictureBox sorgenti:
        Sorgente0.Location = PosizioneSorgente(0)
        Sorgente1.Location = PosizioneSorgente(1)
        Sorgente2.Location = PosizioneSorgente(2)
        Sorgente3.Location = PosizioneSorgente(3)
        Sorgente4.Location = PosizioneSorgente(4)
```

```
        ' Rimescola le posizioni delle immagini destinazione:
        For Contatore As Integer = 0 To 4
            Dim Cambio = Sorteggio.Next(5)
            Dim PosizioneProvvisoria = PosizioneDestinazione(Cambio)
            PosizioneDestinazione(Cambio) = PosizioneDestinazione(Contatore)
            PosizioneDestinazione(Contatore) = PosizioneProvvisoria
        Next

        'e le assegna ai controlli PictureBox destinazione:
        Destinazione0.Location = PosizioneDestinazione(0)
        Destinazione1.Location = PosizioneDestinazione(1)
        Destinazione2.Location = PosizioneDestinazione(2)
        Destinazione3.Location = PosizioneDestinazione(3)
        Destinazione4.Location = PosizioneDestinazione(4)

        Sorgente0.Visible = True
        Sorgente1.Visible = True
        Sorgente2.Visible = True
        Sorgente3.Visible = True
        Sorgente4.Visible = True

        'Ripristina le immagini nei controlli PictureBox destinazione:
        Destinazione0.Image = My.Resources.ape
        Destinazione1.Image = My.Resources.edera
        Destinazione2.Image = My.Resources.imbuto
        Destinazione3.Image = My.Resources.oca
        Destinazione4.Image = My.Resources.uva

    End Sub

End Class
```

Ecco un’immagine del programma in esecuzione:

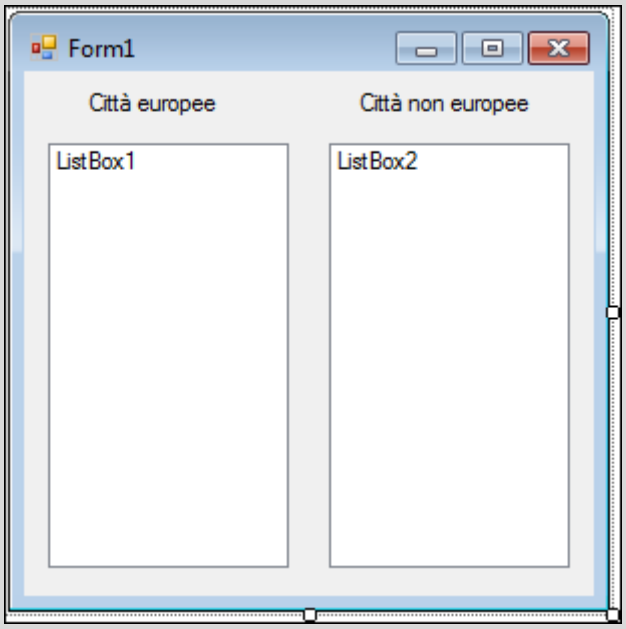


L'ultimo esercizio sull'evento DragDrop riguarda il trascinamento di item (di singole voci) da una lista, per portarli in un'altra lista.

Esercizio 127: DragDrop di item tra due Listbox.

Questo programma crea due elenchi di città (europee e non europee) in due controlli ListBox. L'utente del programma può cliccare i nomi delle città per spostarli da un elenco all'altro, sino a formare due elenchi corretti con tutte le città europee nel primo ListBox e tutte le città non europee nel secondo ListBox.
A ogni operazione di trasferimento, il nome della città cliccato è preso da uno dei due ListBox e trasferito nell'altro; a trasferimento completato, lo stesso nome viene cancellato dal ListBox originario.
Il controllo è effettuato in questo modo: all'avvio il programma crea un primo testo di controllo con l'elenco di tutte le città europee. A ogni spostamento, il programma crea un secondo testo con l'elenco dei nomi delle città che rimangono nel primo ListBox.
Se i due testi coincidono, il programma termina con un messaggio di congratulazioni.

Apriamo un nuovo progetto e inseriamo nel form due controlli Label e due controlli ListBox come in questa immagine:



Proprietà dei due ListBox:

- **AllowDrop** = True
- **Sorted** = True

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea un oggetto ListBox, per memorizzare quale dei due ListBox è stato
cliccato:
    Dim ListaSorgente As ListBox

    ' Crea due matrici con gli elenchi delle città:
    Dim CittàEuropee As New List(Of String) From {"Milano", "Bucarest",
"Helsinki", "Lisbona", "Berna", "Kiev", "Stoccolma", "Valencia", "Lione",
"Vienna"}
    Dim CittàNonEuropee As New List(Of String) From {"New York", "Pechino",
"Tripoli", "Tunisi", "Buenos Aires", "Lima", "San Paolo", "San Francisco",
"Miami", "Bagdad", "Tel Aviv", "Lahore"}

    Dim TestoControllo1 As String
    Dim TotaleCittàEuropee As Integer
    Dim TotaleCittàNonEuropee As Integer
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        TotaleCittàEuropee = CittàEuropee.Count
        TotaleCittàNonEuropee = CittàNonEuropee.Count

        ' All'avvio del programma, riversa il contenuto delle liste
        ' CittàEureopee e CittàNonEuropee nel ListBox1:
        ListBox1.Items.AddRange(CittàEuropee.ToArray)
```

```
        ListBox1.Items.AddRange(CittàNonEuropee.ToArray)

        ' Crea un testo di controllo con l'elenco delle città europee in ordine
alfabetico:
        CittàEuropee.Sort()
        For Each Città As String In CittàEuropee
            TestoControllo1 &= Città
        Next

    End Sub
```

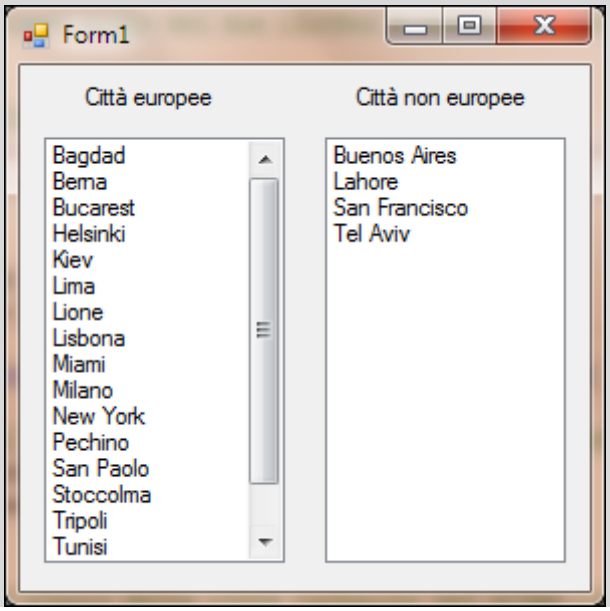
```
    Private Sub List_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ListBox1.MouseDown,
ListBox2.MouseDown

        ' Se il clic è avvenuto nello spazio vuoto del ListBox non è stata
cliccata alcuna città,
        ' per cui esci da questa procedura:
        If sender.SelectedIndex < 0 Then Exit Sub

        ' Verifica quale città è stata cliccata:
        sender.SelectedIndex = sender.IndexFromPoint(e.X, e.Y)

        ' Memorizza quale, tra le due liste, è la lista sorgente:
        ListaSorgente = sender

        ' Comanda l'avvio della procedura di spostamento del nome della città
selezionata:
        sender.DoDragDrop(sender.SelectedItem.ToString, DragDropEffects.Move)

    End Sub
```

```
    Private Sub List_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles ListBox1.DragEnter,
ListBox2.DragEnter

        e.Effect = DragDropEffects.Move

    End Sub

    Private Sub List_DragLeave(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ListBox2.DragLeave, ListBox1.DragLeave

        ' Togli lo sfondo blu alla città selezionata in un ListBox, quando il
mouse esce da questa ListBox:
        sender.SelectedIndex = -1

    End Sub
```

```
    Private Sub List_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles ListBox1.DragDrop, ListBox2.DragDrop

        ' Copia il nome della città selezionata nella nuova lista:
        sender.items.add(e.Data.GetData(DataFormats.Text))
        ' Cancella dalla lista sorgente il nome della città spostato nella lista
destinazione:
        ListaSorgente.Items.Remove(e.Data.GetData(DataFormats.Text))
```

```
        ' Controlla se gli spostamenti sono finiti e sono stati effettuati in
modo corretto.
        ' Crea un TestoControllo2 con l'elenco delle città del LitBox1, da
confrontare con
        ' il TestoControllo1 creato all'inizio del programma:

        Dim TestoControllo2 As String

        For Each Città As String In ListBox1.Items
            TestoControllo2 &= Città
        Next

        If TestoControllo1 = TestoControllo2 Then
            Beep()
            MsgBox("Esatto!")
            End
        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



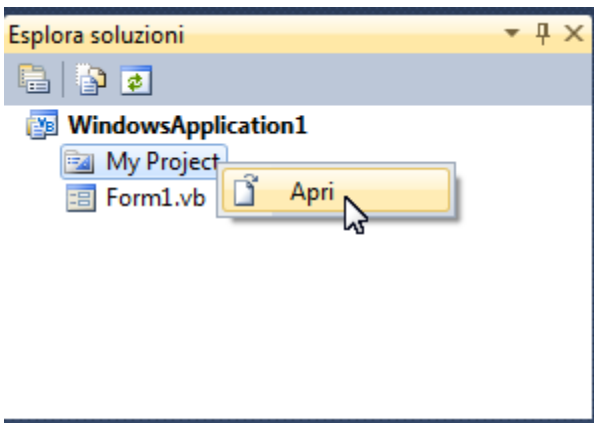
Capitolo 34: GESTIONE DI FILE ESTERNI AL PROGRAMMA.

Molti programmi hanno l'esigenza di interagire con file che si trovano al loro esterno o che sono prodotti dagli utenti quando usano questi programmi.
In questo capitolo ci occuperemo di alcune *ramificazioni* dell'oggetto My che possono essere utilizzate per gestire immagini, testi e file audio esterni al programma, nelle due direzioni: sia per utilizzare file già esistenti, sia per salvare file creati dall'utente.
Si tratta di:

- My.Resources.
- My.Computer.FileSystem.
- My.Application.

185: My.Resources.

L'oggetto **My.Resources** è lo strumento principale per inglobare in un programma file audio, file di immagini e file di testo da utilizzare al momento opportuno durante lo svolgimento del programma stesso.
Con **My.Resources** queste risorse sono unite al programma e sono distribuite automaticamente assieme ad esso, senza che il programmatore debba preoccuparsi di inserirle nel pacchetto di distribuzione del suo prodotto.
Vediamo di seguito le operazioni necessarie per inserire suoni, immagini e testi nelle risorse del programma.
Nella finestra Esplora soluzioni, facciamo un *clic* con il tasto destro del mouse su My Project. Questo consente l'accesso alla Finestra Proprietà del progetto:



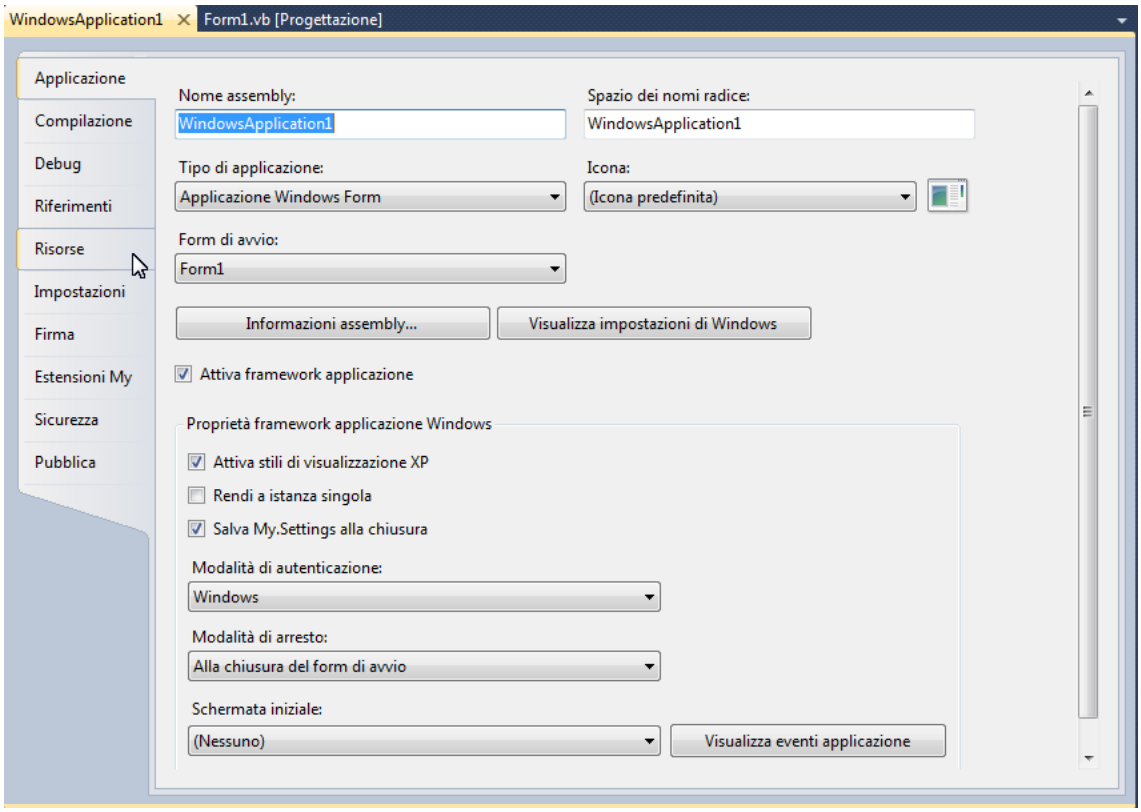


Figura 244: Apertura delle proprietà di un progetto.

Qui, cliccando la scheda **Risorse** si accede alla finestra delle risorse del programma, con l'elenco degli oggetti che vi possono essere inseriti:

- stringhe di testo,
- immagini,
- icone,
- file audio,
- file di testo.

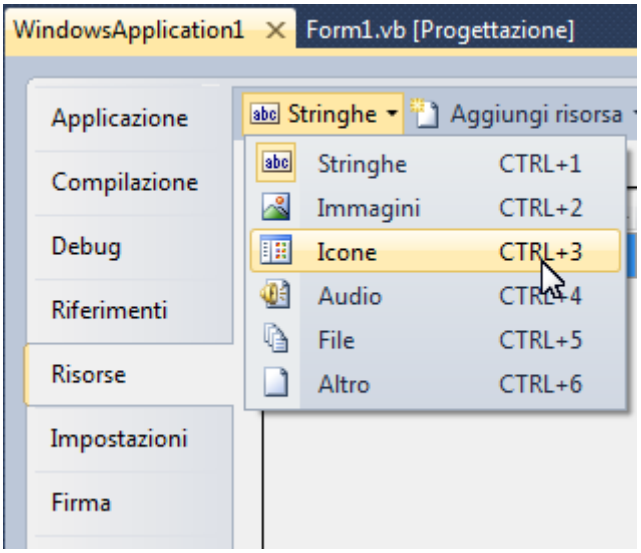


Figura 245: Gli oggetti inseribili nelle risorse del programma.

Cliccando una di queste voci è possibile dotare il programma degli elementi esterni eventualmente necessari al suo funzionamento.
Il programmatore ha due possibilità:

- può utilizzare un file già esistente nel suo computer (clic su **Aggiungi file esistente**…), oppure
- può creare un nuovo file. A questo scopo VB gli consente di utilizzare un *editor* di immagini e un *editor* di testi:

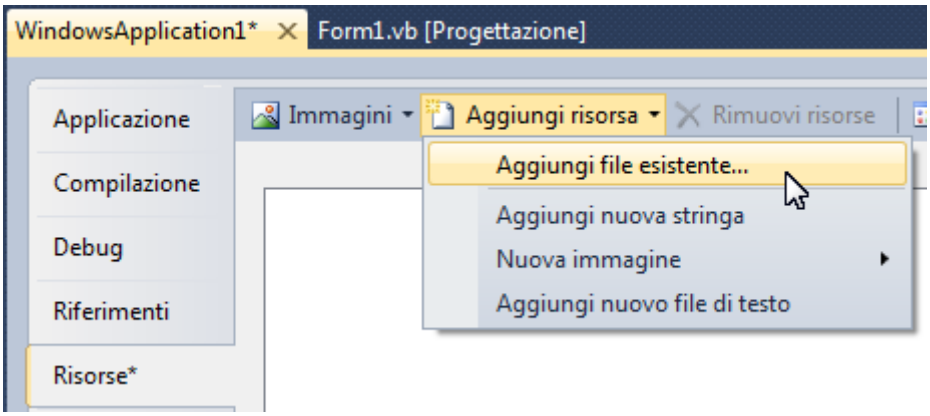


Figura 246: Scelta del tipo di file da inserire nelle risorse del programma.

- L'*editor* di immagini attivato da VB è quello predefinito nel computer del programmatore;
- l'*editor* di testi invece è un *editor* di testi specifico di VB che funziona come un semplice elaboratore di testi, simile al **Blocco Note** di **Windows**.

Questi due strumenti possono essere utilizzati per ritoccare e adattare immagini o file di testo già inseriti nelle risorse del programma.

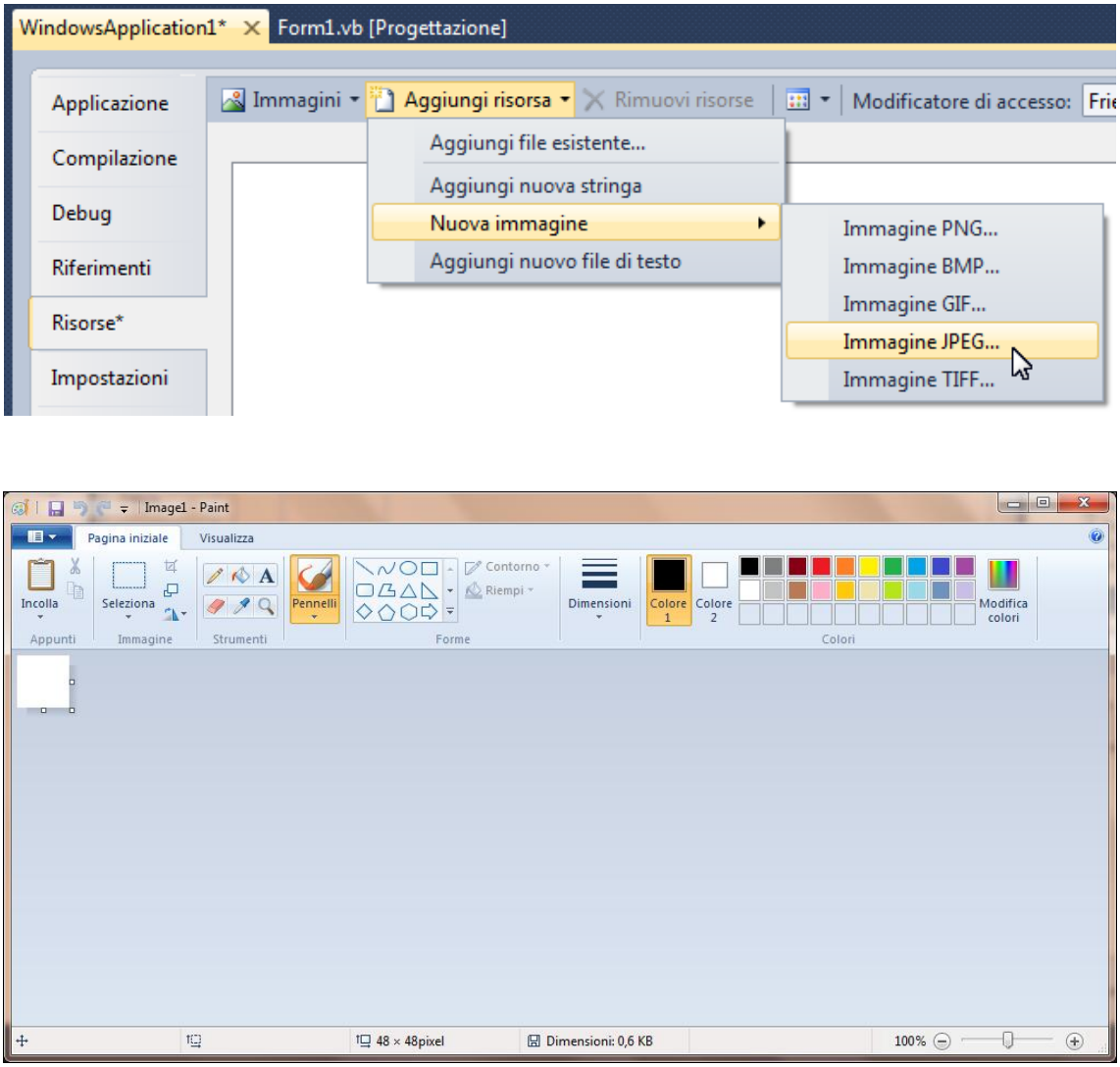
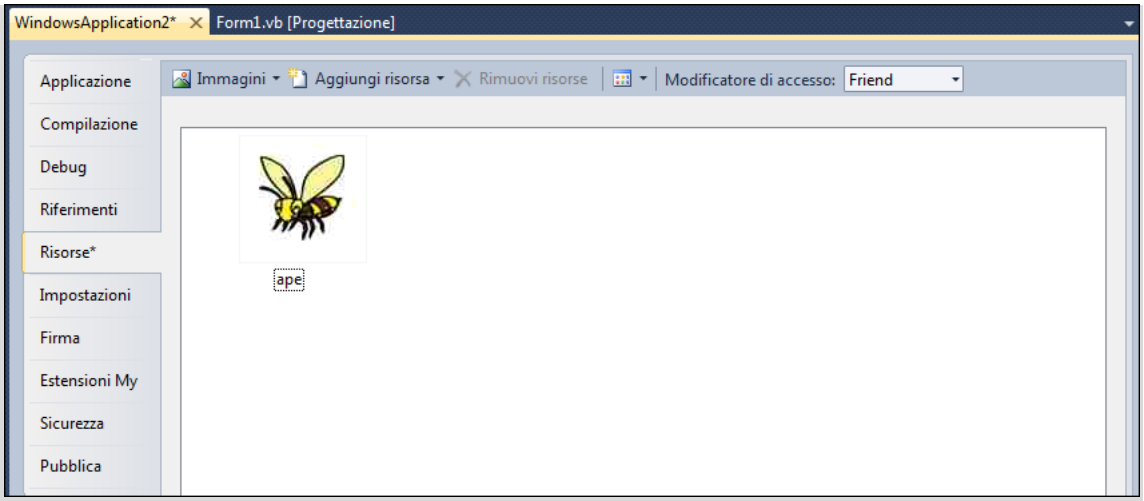


Figura 247: L'*editor* di immagini Paint di Windows.

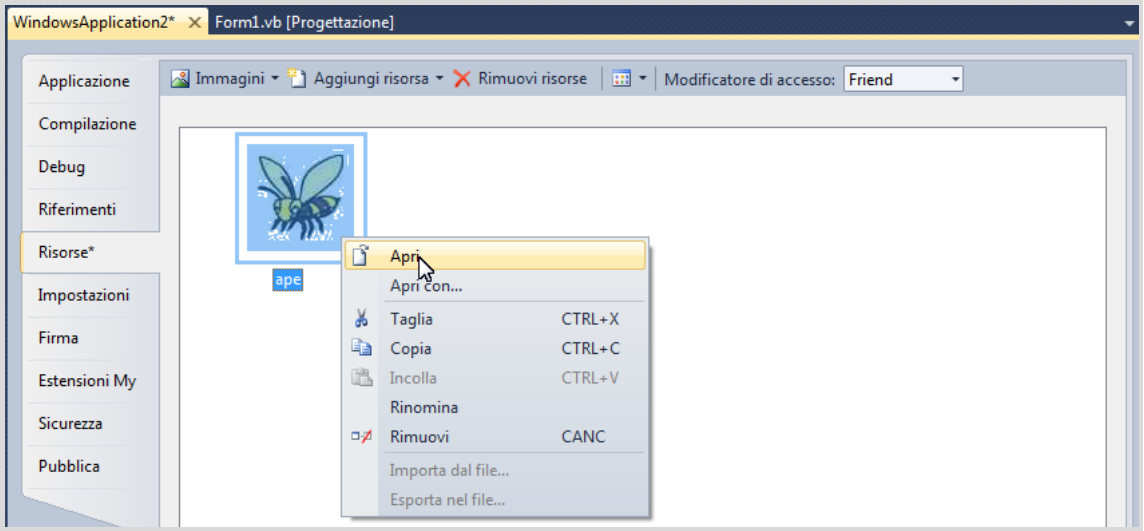
Nel prossimo esercizio vedremo un esempio di utilizzo dell'*editor* di immagini per ritoccare un'immagine già inserita nelle risorse del programma.

Esercizio 128: Ritocco di un' immagine con l'*editor* di immagini Paint di Windows.

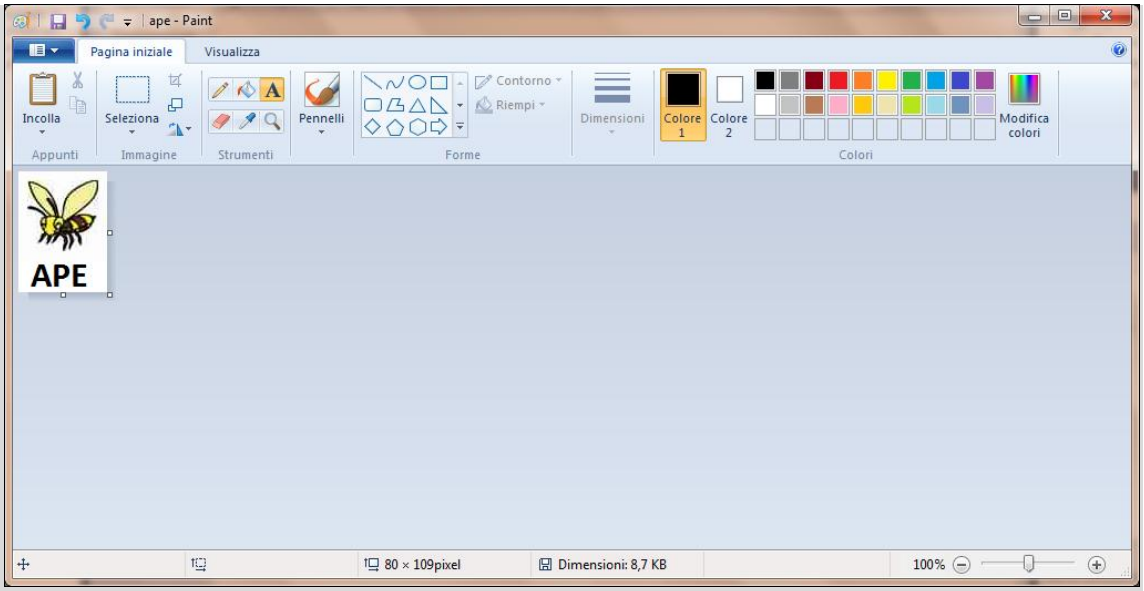
Apriamo un nuovo progetto e inseriamo nelle risorse del programma l'immagine **ape.jpg**, che troviamo nella cartella **Documenti \ A scuola con VB \ Immagini \ Vocali**:



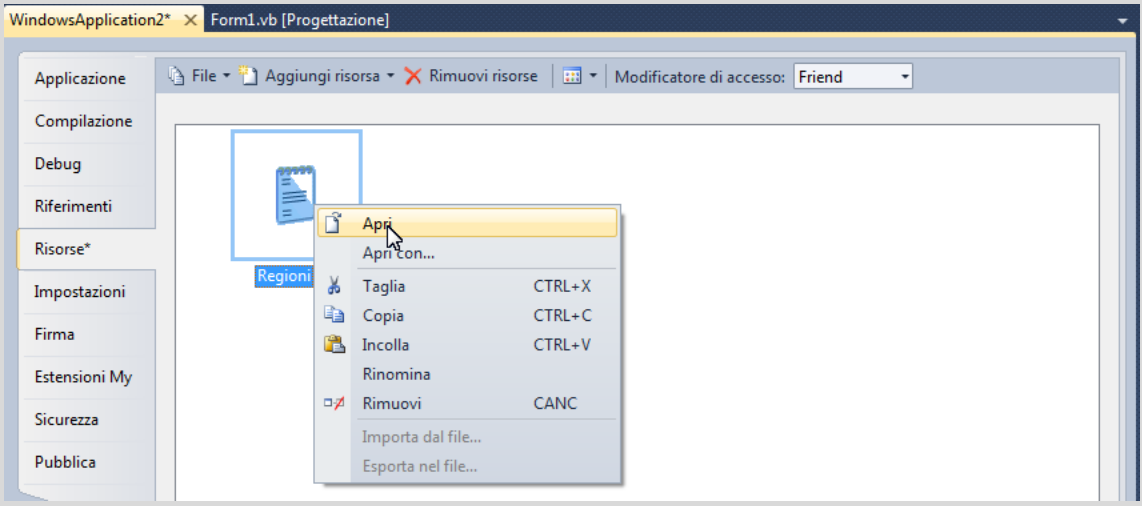
Vogliamo inserire in questa immagine la scritta **APE**.
Per fare questo, facciamo un *clic* con il tasto destro del mouse sull'immagine e facciamo un *clic* su **Apri**:



Accediamo in questo modo all'*editor* d'immagini predefinito nel sistema operativo in uso. Nell'immagine seguente, l'*editor* d'immagini predefinito è il programma **Paint** di Windows. Con **Paint** possiamo inserire nella nostra immagine la scritta **APE**:



Effettuata questa operazione, facciamo un *clic* su **Salva** nel menu a tendina in alto a sinistra. In questo modo abbiamo corretto e salvato l'immagine in uso nel programma; l'immagine originale è rimasta inalterata.
Allo stesso modo si può procedere per correggere un file di testo già inserito nelle risorse del programma.
Inseriamo nelle risorse del programma il file **Regioni.txt** che si trova nella cartella **Documenti \ A scuola con VB \ Testi**.



Facciamo un *clic* con il tasto destro del mouse sull'icona del file e facciamo *clic* su **Apri**. Accediamo in questo modo all'*editor* di testi con il quale possiamo correggere l'elenco delle regioni, ad esempio, cancellando l'ultima voce: *Totale*.
Fatta questa correzione possiamo chiudere l'*editor* di testi: il file Regioni.txt è ora presente nelle risorse del programma nella versione corretta; la versione originale non è stata modificata.

Una volta inseriti i materiali necessari al programma nelle risorse del programma, il loro utilizzo avviene in modo molto semplice. Ecco alcuni esempi, riguardanti rispettivamente l'utilizzo di un file audio, un file immagine e un file di testo:

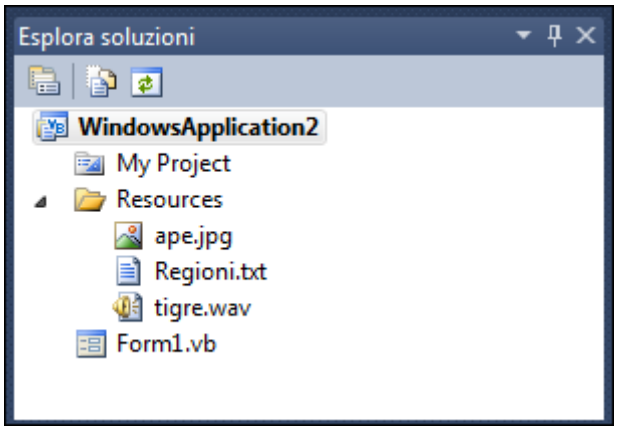


Figura 248: Tre file, di tipo jpg, txt e wav, nelle risorse di un programma.

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' Riproduce il file tigre.wav:
        My.Computer.Audio.Play(My.Resources.tigre, AudioPlayMode.Background)

        ' Visualizza l'immagine ape.jpg in un PictureBox:
        PictureBox1.Image = My.Resources.ape

        ' Visualizza il testo Regioni.txt in una Label:
        Label1.Text = My.Resources.Regioni
        ' Visualizza il testo Regioni.txt in un TextBox:

        TextBox1.Text = My.Resources.Regioni

    End Sub

End Class
```

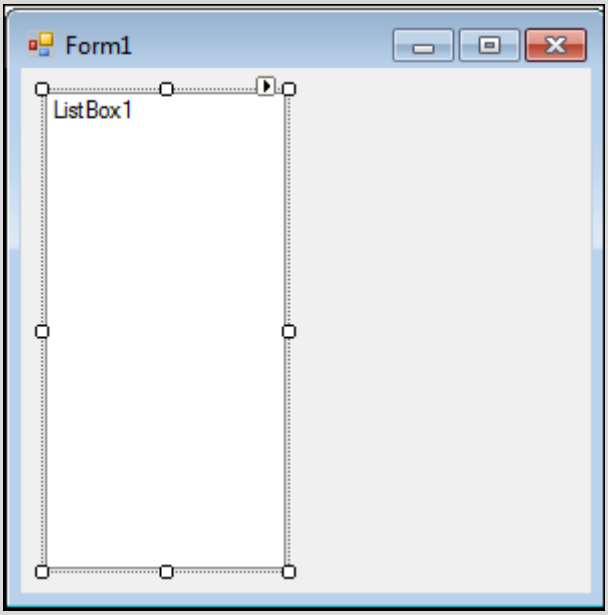
186: Scomporre un file di testo in una matrice di variabili.

Si verifica spesso nei programmi l'esigenza di utilizzare un file di testo non come un testo unico per un TextBox o una Label, ma come un elenco, un insieme di elementi da separare e da utilizzare singolarmente.

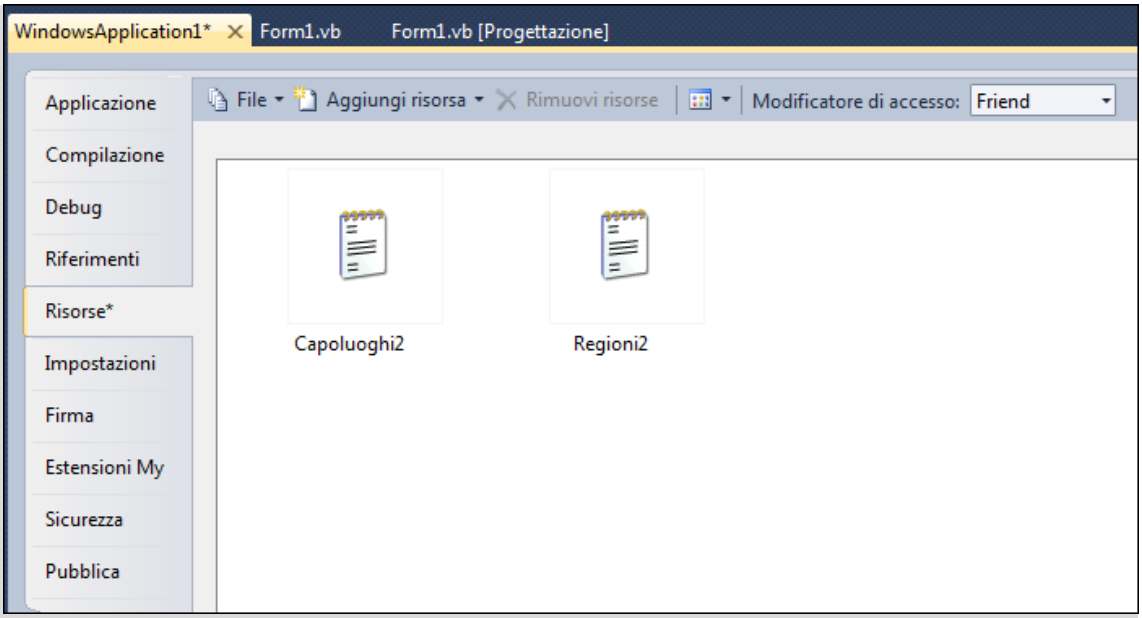
Ad esempio, il file **Regioni.txt**, che abbiamo visto nell'esercizio precedente, può essere visualizzato in una Label come un testo unico con i nomi delle 20 regioni italiane, oppure può essere letto dal programma e suddiviso **riga per riga**, in modo avere 20 elementi separati.
Questi elementi separati, inseriti in una **matrice di variabili**, o in una lista di tipo **List (Of String**) o in un controllo **ListBox**, si prestano poi a essere utilizzati in modi diversi, all'interno del programma.
Nel prossimo esercizio vediamo un esempio di scomposizione in variabili di un file di testo collocato dal programmatore nelle risorse del programma.
Più avanti, in questo stesso capitolo, vedremo un esempio simile, di scomposizione in variabili di un file di testo scelto dall'utente del programma.

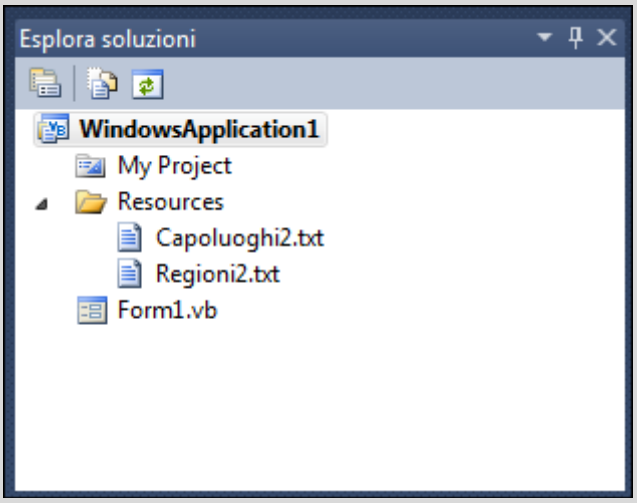
Esercizio 129: Scomporre un testo in una matrice di variabili - I.

Apriamo un nuovo progetto.
Inseriamo nel Form1 un controllo **ListBox1**,come in questa immagine:

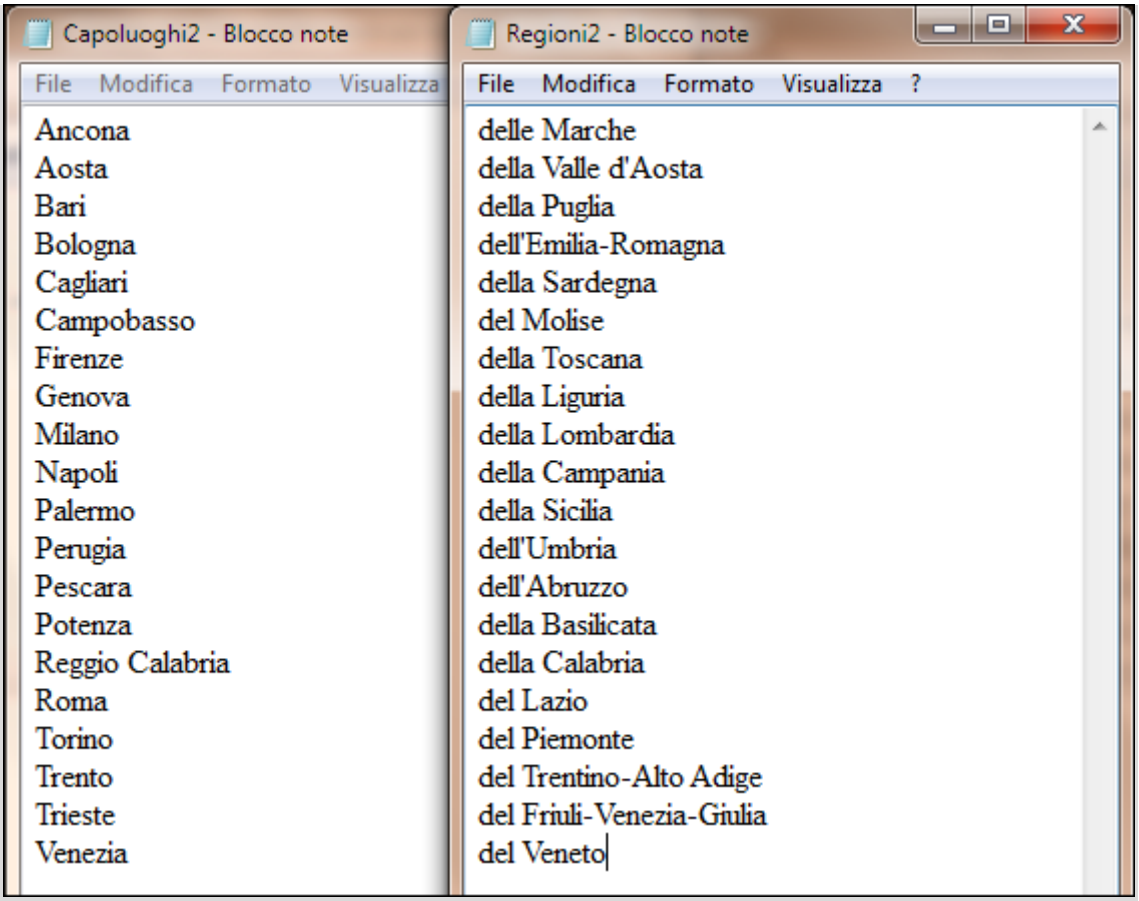


Inseriamo nelle risorse del programma i file **Capoluoghi2.txt** e **Regioni2.txt** che si trovano nella cartella **Documenti \ A scuola con VB \ Testi**:





I file **Capoluoghi2** e **Regioni2** contengono questi testi:



Il programma legge il file **Capoluoghi2** e immette i nomi delle 20 città in un controllo ListBox, dove questi nomi potranno essere cliccati e scelti singolarmente dall'utente. Le 20 scritte del file **Regioni2** sono inserite invece in una matrice di variabili, dalla quale verranno richiamate una alla volta, in corrispondenza della città cliccata nel ListBox1, per formare il testo di un messaggio all'utente.
Copiamo e incolliamo nella Finestra del Codice questo listato:

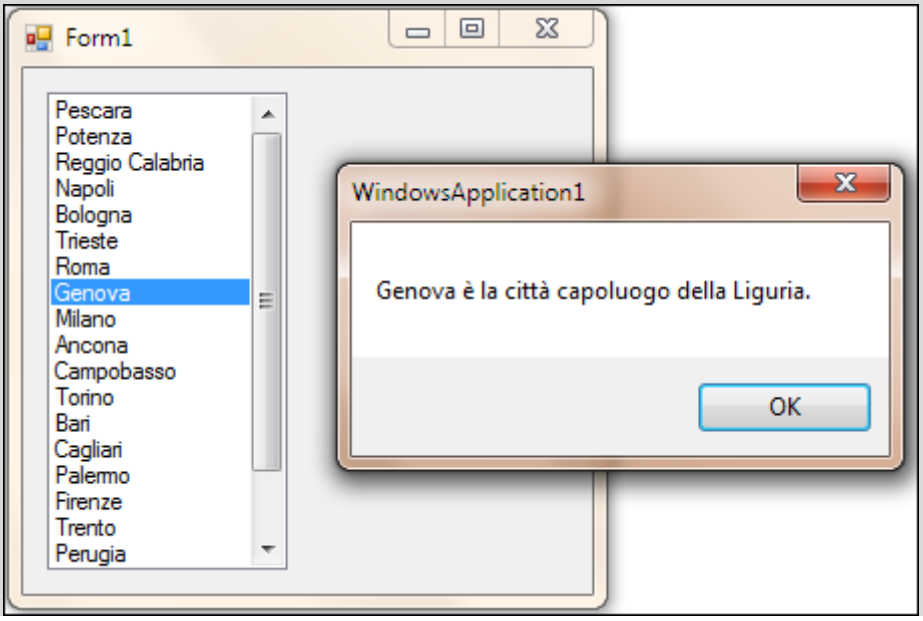
```
Public Class Form1

    'Crea due matrici di variabili
    Dim Capoluogo() As String
    Dim Regione() As String
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' Apre e legge il file Capoluoghi2.txt:
        Dim Elenco As String = My.Resources.Capoluoghi2

        ' Spezza il file Capoluoghi2.txt riga per riga (elemento separatore è il
segno vbcrlf = a capo)
        ' e immette gli elementi così ottenuti in una matrice di variabili
denominata "Capoluogo":
        Capoluogo = Split(Elenco, vbCrLf)
```

```
        ' Immette tutti gli elementi della matrice "Capoluogo" nel controllo
ListBox:
        ListBox1.Items.AddRange(Capoluogo)

        ' Apre e legge il file Regioni2.txt:
        Elenco = My.Resources.Regioni2

        ' Spezza il file Regioni2.txt riga per riga (elemento separatore è il
segno vbcrlf = a capo)
        ' e immette gli item così ottenuti in una matrice di variabili denominata
"Regione":
        Regione = Split(Elenco, vbCrLf)

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged(sender As System.Object, e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

        MsgBox(ListBox1.SelectedItem & " è la città capoluogo " &
Regione(ListBox1.SelectedIndex) & ".")

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



187: My.Resources.ResourceManager.GetObject

Negli esempi che abbiamo visto sino a ora, un file inserito tra le risorse del programma viene utilizzato in modo molto semplice, indicandolo con il suo nome:

```
PictureBox1.Image = My.Resources.ape
Label1.Text = My.Resources.Regioni
TextBox1.Text = My.Resources.Regioni
```

In un programma tuttavia, può verificarsi spesso l'esigenza di utilizzare un file che si trova nelle risorse, indicandolo non con un nome ma con una variabile.
Supponiamo, ad esempio, di avere nelle risorse di un programma l'immagine **ape** e di avere nello stesso programma questa variabile di tipo String:

```
Dim NomeFile As String = "ape"
```

In questa situazione, questo comando visualizza l'immagine in un controllo PictureBox:

```
PictureBox1.Image = My.Resources.ape
```

Mentre questo comando causa un errore nel programma:

```
PictureBox1.Image = My.Resources.(NomeFile)
```

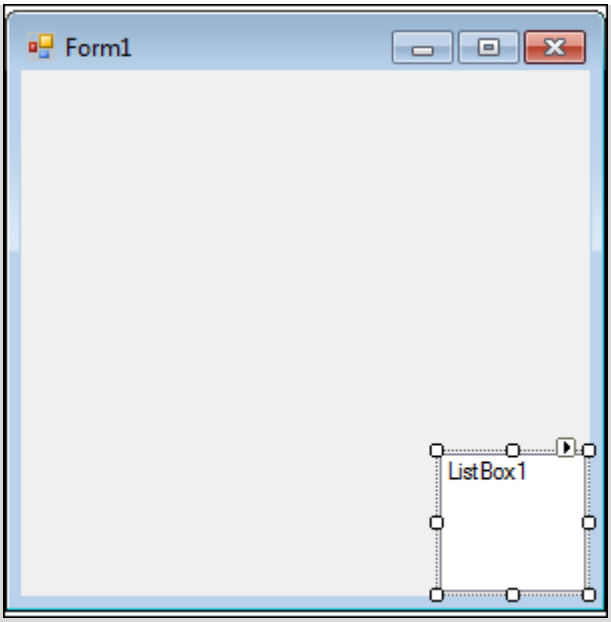
Il problema si risolve utilizzando il comando
My.Resources.ResourceManager.GetObject:

```
Dim NomeFile As String = "ape"
PictureBox1.Image = My.Resources.ResourceManager.GetObject(NomeFile)
```

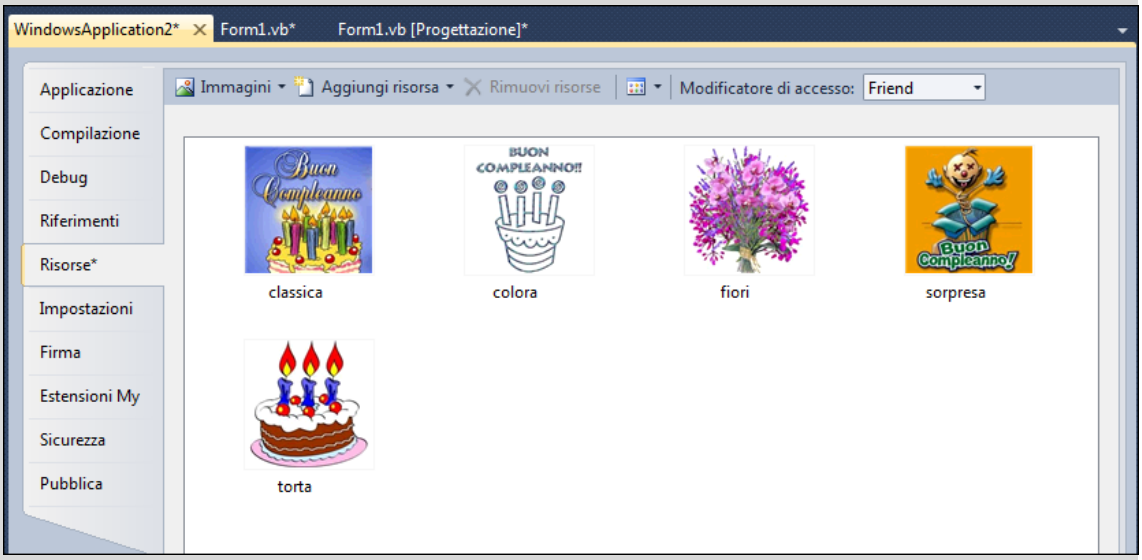
Abbiamo già visto un esempio dell'utilizzo di questo comando nell'Esercizio 105: Le fasi lunari. 720.
In quel caso le immagini presenti nelle risorse del programma hanno come nomi i numeri da 1 a 36 e il programma le utilizza una alla volta richiamandole secondo il loro nome.
Nell'esercizio seguente vedremo un esempio simile, con cinque immagini inserite nelle risorse del programma con dei nomi comuni.

Esercizio 130: My.Resources.ResourceManager.GetObject.

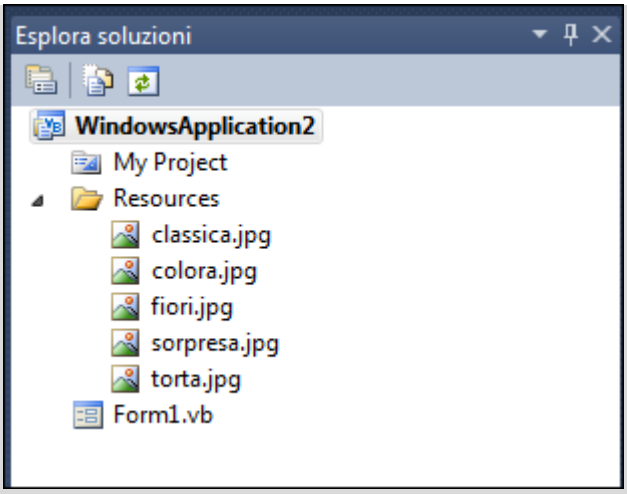
Apriamo un nuovo progetto e inseriamo nel form, nell'angolo in basso a destra, un controllo ListBox come in questa immagine:



Ora inseriamo nelle risorse del programma le cinque immagini che si trovano nella cartella **Documenti \ A scuola con VB 2010 \ Immagini \ Auguri**:



Ecco come appare la finestra Esplora soluzioni del programma:



Il programma, al suo avvio, inserisce nel ListBox1 i nomi di queste cinque immagini. Quando l'utente clicca una delle immagini nel ListBox1, l'immagine corrispondente è prelevata dalle risorse del programma e visualizzata come sfondo del Form1. Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Me.BackgroundImageLayout = ImageLayout.Center

        With ListBox1
            .Items.Add("classica")
            .Items.Add("colora")
            .Items.Add("fiori")
            .Items.Add("sorpresa")
            .Items.Add("torta")
        End With

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged(sender As System.Object, e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

        Me.BackgroundImage =
My.Resources.ResourceManager.GetObject(ListBox1.SelectedItem)

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

188: My.Computer.FileSystem.

L'oggetto **My.Computer.FileSystem** consente al programmatore di accedere al disco fisso del computer, alle sue cartelle e ai suoi file, per leggere dati necessari al funzionamento del programma, per salvare dati prodotti con il programma, per creare, rinominare, cancellare cartelle o file.
Noi ci limiteremo all'utilizzo di **My.Computer.FileSystem** per due semplici operazioni: il salvataggio e la lettura di file di testo in formato .txt.
Il salvataggio e la lettura di immagini avvengono con modalità diverse che vedremo nel prossimo paragrafo.
Anche Il salvataggio e la lettura di file di database (banche di dati organizzati in tabelle) avvengono con modi diversi, che vedremo in altra parte del manuale.

Il salvataggio e la lettura di file di testo con **My.Computer.FileSystem** si compiono con una sola riga di istruzioni, la cui sintassi è:

- per il salvataggio

```
My.Computer.FileSystem.WriteAllText()
```

- per la lettura

```
My.Computer.FileSystem.ReadAllText()
```

Vediamo alcuni esempi.
Questa procedura gestisce il salvataggio, in un file .txt, del testo contenuto in un TextBox1:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        My.Computer.FileSystem.WriteAllText("C:\FileDiTesto.txt", TextBox1.Text,
False)

    End Sub

End Class
```

I parametri tra parentesi, separati da virgole, indicano:

1. la collocazione e il nome del file di testo che deve essere creato;
2. il testo da salvare;
3. la modalità di salvataggio: se questa è impostata su False, il nuovo file sostituisce quello precedente; se è impostata su True, il nuovo testo va ad aggiungersi al testo già esistente nel file.

Ecco un esempio dell'operazione inversa.
Questa procedura visualizza in un controllo TextBox1 il testo contenuto in un file .txt:

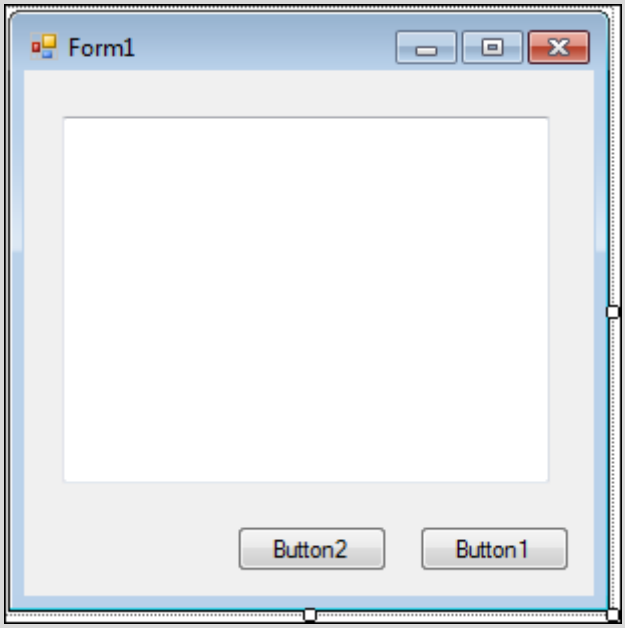
```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        TextBox1.Text = My.Computer.FileSystem.ReadAllText("C:\FileDiTesto.txt")

    End Sub

End Class
```

In questo caso è richiesto un unico parametro; l'indicazione della collocazione e del nome del file di testo da recuperare.
Nei due esempi che abbiamo visto, la collocazione e il nome del file da salvare o da recuperare sono indicati dal programmatore (**"C:\FileDiTesto.txt"**); per consentire all'utente di effettuare una scelta diversa bisogna ricorrere ai componenti **SaveFileDialog** e **OpenFileDialog**[84], di cui vedremo un esempio nel prossimo esercizio.

[84] 62, a pag. 310.

Esercizio 131: Salvataggio e recupero di un testo.

In questo programma abbiamo un controllo **TextBox** (casella di testo) nel quale l'utente può scrivere un testo da salvare o in cui può visualizzare un testo già salvato. Il nome e il percorso del file da salvare o del file da aprire sono indicati dall'utente del programma.
Apriamo un nuovo progetto e inseriamo nel form un controllo TextBox e due pulsanti Button, come in questa immagine:



Proprietà del TextBox:
Multiline = True
Size = 240; 180
ScroolBars = Vertical
Inseriamo nel progetto i componenti **OpenFileDialog** e **SaveFileDialog**.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Apri.Click

        ' APERTURA DI UN FILE DI TESTO
        ' Cancella il contenuto del TextBox1 e il nome del file eventualmente
scelto in precedenza:
        TextBox1.Clear()
        OpenFileDialog1.FileName = ""

        ' Imposta un filtro per la ricerca di file di testo nella finestra di
dialogo:
        OpenFileDialog1.Filter = "File di testo|*.txt"
```

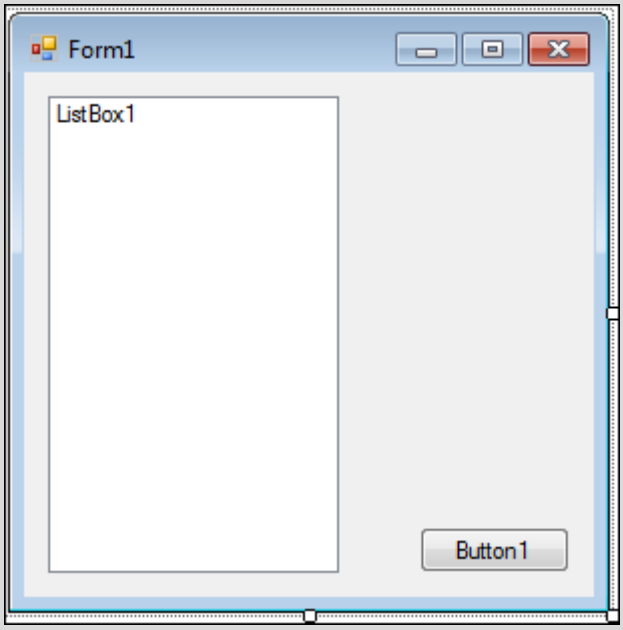
```
        ' Verifica se l'utente ha scelto un file e visualizza questo file nel
TextBox:
        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            TextBox1.Text =
My.Computer.FileSystem.ReadAllText(OpenFileDialog1.FileName)
        End If

    End Sub
```

```
    Private Sub Button2_Click(sender As Object, e As System.EventArgs) Handles
Button2.Click

        ' SALVATAGGIO DI UN FILE DI TESTO
        ' Cancella il nome del file eventualmente salvato in precedenza:
        SaveFileDialog1.FileName = ""

        ' Imposta un filtro per il salvataggio di file di testo nella finestra di
dialogo:
        SaveFileDialog1.Filter = "File di testo|*.txt"

        ' Verifica se l'utente ha scritto un nome e un percorso valido e salva il
file:
        If SaveFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            My.Computer.FileSystem.WriteAllText(SaveFileDialog1.FileName,
TextBox1.Text, False)
        End If

    End Sub

End Class
```

Nel prossimo esercizio vedremo un altro esempio di apertura di un file di testo e della sua scomposizione, riga per riga, in elementi separati da inserire in un controllo ListBox.
In questo caso il file di testo è scelto dall'utente del programma, mediante il componente **OpenFileDialog.**

Esercizio 132: Scomporre un testo in una matrice di variabili - II.

Apriamo un nuovo progetto.
Inseriamo nel form un componente pulsante **Button1** e un controllo **ListBox1**.
Il controllo **ListBox** ha la proprietà **Sorted = True**.

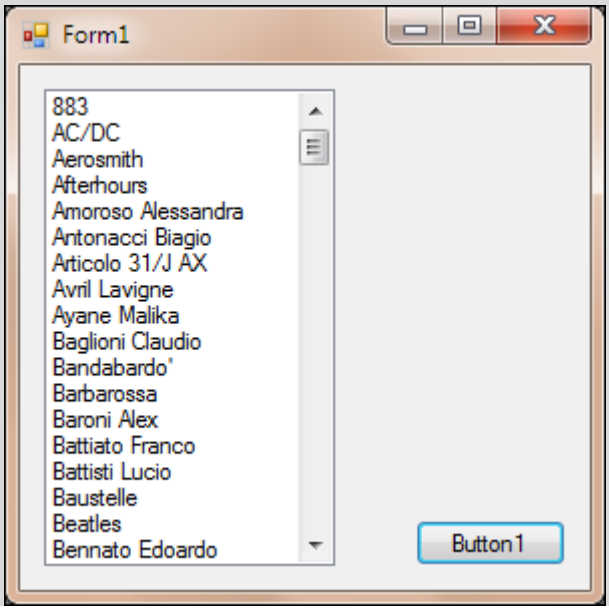


Inseriamo nel progetto un componente **OpenFileDialog.**
Questo programma legge un file di testo, scelto dall'utente, lo scompone e lo immette riga per riga in un controllo ListBox, considerando ogni riga come un singolo elemento che può essere cliccato dall'utente.
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        ' Cancella il contenuto del ListBox:
        ListBox1.Items.Clear()

        ' APERTURA DI UN FILE DI TESTO
        OpenFileDialog1.FileName = ""

        ' Imposta un filtro per la ricerca di file di testo nella finestra di
dialogo:
        OpenFileDialog1.Filter = "File di testo|*.txt"

        ' Verifica se l'utente ha scelto un file:
        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then

            ' Apre questo file e ne immette il contenuto nella variabile Testo:
            Dim Testo As String =
My.Computer.FileSystem.ReadAllText(OpenFileDialog1.FileName)
```

```
            ' Spezza il file Testo riga per riga (elemento separatore è il segno
vbcrlf = a capo)
            ' e immette gli elementi così ottenuti in una matrice di variabili
denominata "Riga":
            Dim Riga() = Split(Testo, vbCrLf)

            ' Immette tutti gli elementi della matrice "Riga" nel controllo
ListBox:
            ListBox1.Items.AddRange(Riga)

        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione (in questo esempio l'utente ha scelto il file **Cantanti e complessi.txt** che si trova nella cartella **Documenti \ A scuola con VB 2010 \ Testi**):



189: My.Computer.FileSystem.SpecialDirectories.

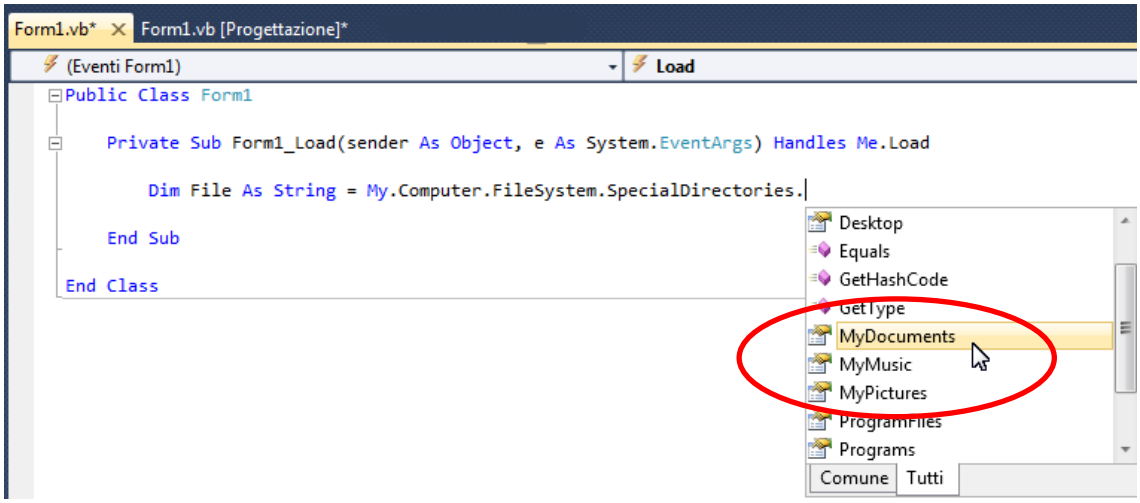


Figura 249: Utilizzo di SpecialDirectories in un programma.

L'oggetto **My.Computer.FileSystem.SpecialDirectories** consente al programmatore di guidare un programma ad accedere ad alcune cartelle del computer sul quale verrà utilizzato il programma.
Si tratta, in particolare, delle cartelle che nelle ultime versioni dei sistemi operativi Windows sono definite *Raccolte*:

- Documenti
- Immagini
- Musica

Il percorso di queste cartelle comprende il nome dell'utente che ha l'autorizzazione per aprirne e gestirne i file:

- C:\Users**NomeUtente**\Music
- C:\Users\ **NomeUtente** \Documents
- C:\Users\ **NomeUtente** \Pictures

Si tratta dunque di percorsi che non possono essere conosciuti a priori dal programmatore; l'oggetto **My.Computer.FileSystem.SpecialDirectories** legge questi percorsi sul computer ospite, e dunque consente al programmatore di aprire e gestire file che si trovano nelle cartelle Documenti, Immagini o Musica:

```
Dim Percorso As String
Percorso = My.Computer.FileSystem.SpecialDirectories.MyDocuments
Percorso = My.Computer.FileSystem.SpecialDirectories.MyPictures
Percorso = My.Computer.FileSystem.SpecialDirectories.MyMusic
```

In alcuni esercizi di questo manuale, ad esempio, abbiamo fatto ricorso a

My.Computer.FileSystem.SpecialDirectories.MyDocuments per utilizzare alcuni file di testo che si trovano nella cartella Documenti \ A scuola con VB \ Testi[85].

In un programma che consenta all'utente di scegliere un file mediante il componente **OpenFileDialog**, può essere opportuno indirizzare l'utente verso le cartelle Documenti, Immagini o Musiche del suo disco fisso.
Ecco un esempio: in questo programma è presente un componente OpenFileDialog, mediante il quale l'utente può scegliere un'immagine da utilizzare come sfondo per il Form1. La cartella iniziale, sulla quale si apre in prima istanza la finestra del componente OpenFileDialog, è la cartella Immagini dell'utente che sta utilizzando il programma:

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        OpenFileDialog1.InitialDirectory =
My.Computer.FileSystem.SpecialDirectories.MyPictures
        OpenFileDialog1.Title = "Cartella immagini"
        OpenFileDialog1.FileName = "Immagini"
        OpenFileDialog1.Filter = "BMP (*.bmp)|*.bmp|JPG (*.jpg)|*.jpg|GIF
(*.gif)|*.gif|PNG (*.png)|*.png"

        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            Me.BackgroundImage = Image.FromFile(OpenFileDialog1.FileName)
        Else
            Exit Sub
        End If

    End Sub

End Class
```

190: L'archivio System.IO.

Come l'oggetto **My.Computer.FileSystem**, l'archivio di software **System.IO** (Input / Output) consente al programmatore di accedere al disco fisso del computer, alle sue cartelle e ai suoi file, per leggere o salvare dati.
Rispetto a **My.Computer.FileSystem**, l'archivio **System.IO** fornisce funzioni aggiuntive che vanno ben oltre le esigenze di un programmatore di base, o fornisce le stesse funzioni in modo più diretto.
Nel prossimo esercizio vedremo un esempio di utilizzo di System.IO per aprire un file di testo, leggerne le proprietà e scomporlo riga per riga per trarne gli elementi da immettere in un ListBox.

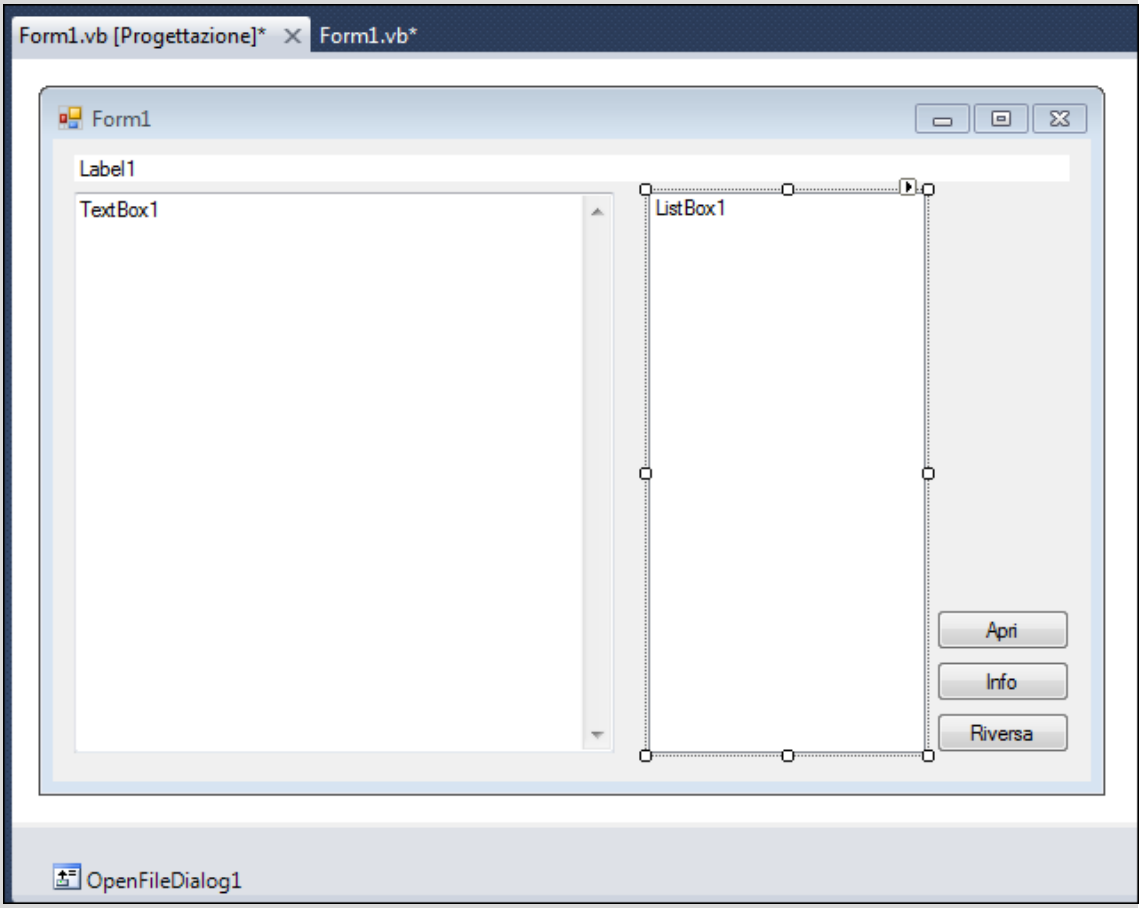
[85] Esercizio 70: Incolonnamento di stringhe di testo. 515; Esercizio 133: L'archivio System.IO. 823.

Esercizio 133: L'archivio System.IO.

Apriamo un nuovo progetto.
Diamo al Form1 le dimensioni 600 x 300 pixel, poi inseriamo nel form

- una Label1;
- un TextBox1;
- un ListBox1;
- un componente OpenFileDialog1 e
- tre pulsanti Button

come in questa immagine:



I tre pulsanti Button hanno queste proprietà:
I
Name = cmdApri
Text = Apri
II
Name = cmdInfo
Text = Info
III
Name = cmdRiversa
Text = Riversa

- Un *clic* sul primo pulsante apre e visualzza un nuovo file di testo.
- Un *clic* sul secondo pulsante visualizza alcune informazioni sul file aperto.
- Un *clic* sul terzo pulsante scompone il file riga per riga e fa di ogni riga un elemento da inserire nel ListBox. L'utente avrà la possibilità di cliccare singolarmente questi elementi.

Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea una variabile per memorizzare il percorso e il nome di un file di
testo sul quale operare.
    ' Al suo avvio il programma apre automaticamente il file Capoluoghi che si
trova nella cartella
    ' Documenti \A scuola con VB 2010\Testi
    Dim File As String = My.Computer.FileSystem.SpecialDirectories.MyDocuments &
"\A scuola con VB 2010\Testi\Capoluoghi.txt"
```

```
    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Activated

        Call VisualizzaTesto()

    End Sub
```

```
    Private Sub VisualizzaTesto()

        TextBox1.Text = ""
        Label1.Text = File
        TextBox1.Text = IO.File.ReadAllText(File)

    End Sub
```

```
    Private Sub cmdApri_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdApri.Click

        OpenFileDialog1.FileName = ""
        ' Imposta un filtro per la ricerca di file di testo nella finestra di
dialogo:
        OpenFileDialog1.Filter = "File di testo|*.txt"

        ' Controlla se l'utente ha scelto un file:
        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            Label1.Text = OpenFileDialog1.FileName
            File = OpenFileDialog1.FileName
        Else
            Exit Sub
        End If

        Call VisualizzaTesto()

    End Sub
```

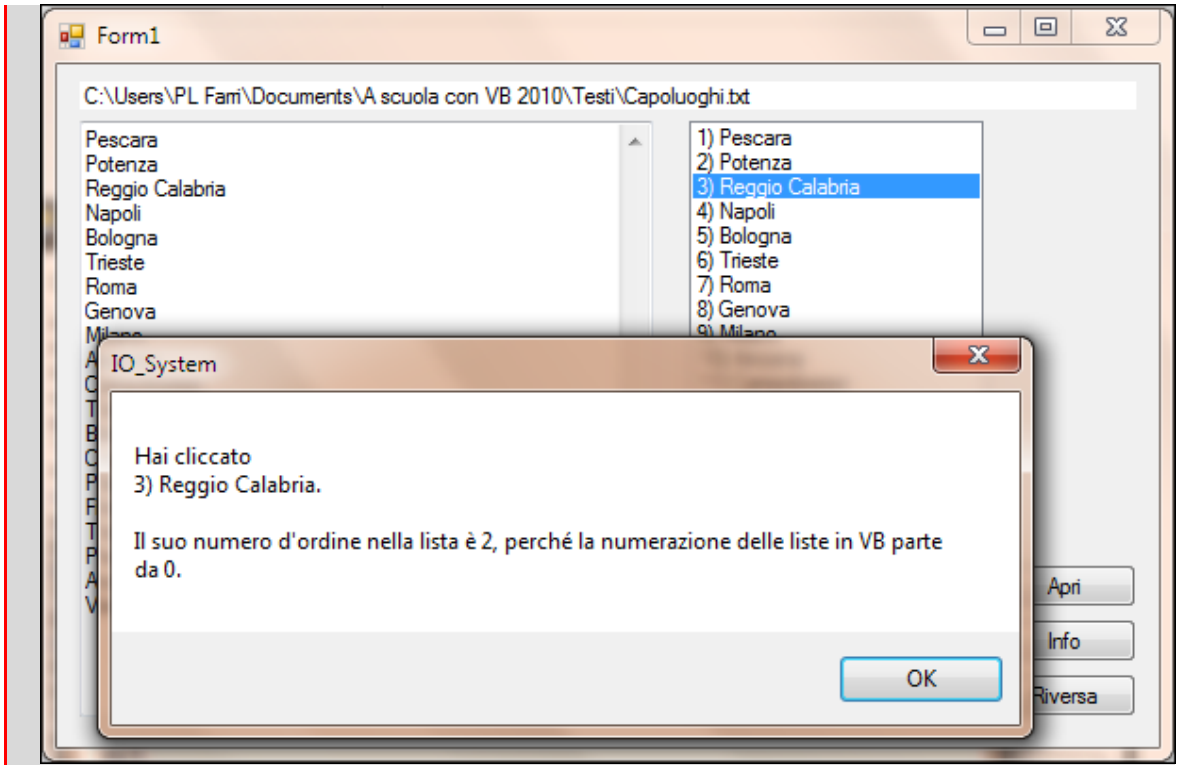
```
    Private Sub cmdRiversa_Click(sender As System.Object, e As System.EventArgs)
Handles cmdRiversa.Click

        ListBox1.Items.Clear()
```

```
        ListBox1.Items.AddRange(IO.File.ReadAllLines(File))

        For Contatore As Integer = 0 To ListBox1.Items.Count - 1
            ListBox1.Items(Contatore) = Contatore + 1.ToString & ") " &
ListBox1.Items(Contatore)
        Next

    End Sub
```

```
    Private Sub cmdInfo_Click(sender As System.Object, e As System.EventArgs)
Handles cmdInfo.Click

        TextBox1.Text = ""
        Dim Testo As String

        Testo &= ("Percorso completo: ") & IO.Path.GetFullPath(File) & vbCrLf &
vbCrLf
        Testo &= ("Cartella: ") & IO.Path.GetDirectoryName(File) & vbCrLf &
vbCrLf
        Testo &= ("Nome del file: ") & IO.Path.GetFileNameWithoutExtension(File)
& vbCrLf & vbCrLf
        Testo &= ("Tipo di file: ") & IO.Path.GetExtension(File) & vbCrLf &
vbCrLf
        Testo &= ("Lunghezza in byte: ") & New IO.FileInfo(File).Length & vbCrLf
& vbCrLf
        Testo &= ("Attributi file: ") & New IO.FileInfo(File).Attributes.ToString
& vbCrLf & vbCrLf
        Testo &= ("Sola lettura: ") & New IO.FileInfo(File).IsReadOnly & vbCrLf &
vbCrLf
        Testo &= ("Data creazione: ") & New
IO.FileInfo(File).CreationTime.ToString & vbCrLf & vbCrLf
        Testo &= ("Ultima lettura: ") & New IO.FileInfo(File).LastAccessTime &
vbCrLf & vbCrLf
        Testo &= ("Ultima scrittura: ") & New IO.FileInfo(File).LastWriteTime &
vbCrLf & vbCrLf

        TextBox1.Text = Testo

    End Sub
```

```
    Private Sub ListBox1_SelectedIndexChanged(sender As System.Object, e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

        MsgBox("Hai cliccato " & vbCrLf & ListBox1.SelectedItem & "." & vbCrLf &
vbCrLf & _
               "Il suo numero d'ordine nella lista è " & ListBox1.SelectedIndex &
", perché la numerazione delle liste in VB parte da 0.")

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



191: Aprire e salvare immagini.

La gestione di file con immagini è diversa dalla gestione di file di testo.
I due comandi per l'apertura o il salvataggio di un file immagine hanno questa sintassi:

```
PictureBox1.Image = Image.FromFile("C:/prova.bmp")
PictureBox1.Image.Save("C:/prova.bmp")
```

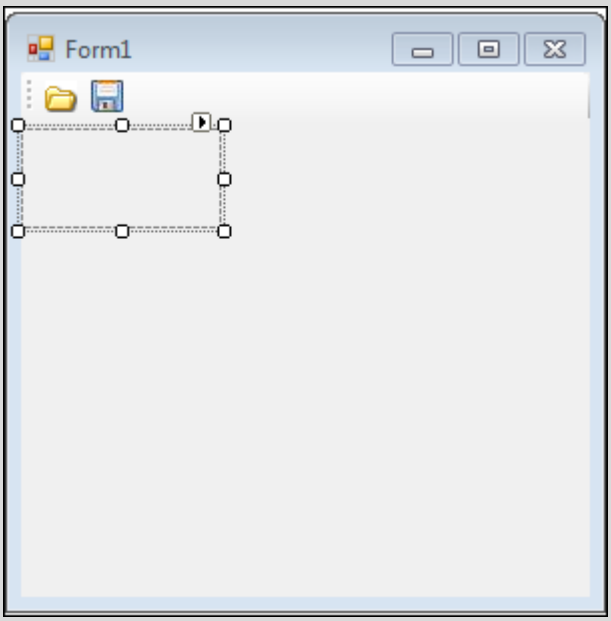
Oppure, nei casi in cui il nome e il percorso del file sono scelti dall'utente del programma:

```
PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)
PictureBox1.Image.Save(SaveFileDialog1.FileName)
```

Nel prossimo esercizio vedremo un esempio di quest'ultima modalità: nel programma sono inseriti i componenti SaveFileDialog e OpenFileDialog che consentono all'utente di visualizzare un'immagine e di salvarla, dopo averla modificata.

Esercizio 134: Aprire, modificare e salvare immagini.

Apriamo un nuovo progetto.
Inseriamo nel form un componente **ToolStrip**.
In questo componente attiviamo due pulsanti **ToolStripButton**, rispettivamente con le immagini **Apri.jpg** e **Salva.jpg** che si trovano nella cartella **Documenti \ A scuola con VB 2010 \ Immagini**:



Ora inseriamo nel programma un componente **OpenFileDialog**, un componente **SaveFileDialog** e un controllo **PictureBox**.
Il **Form1** ha la proprietà **AutoScroll = True**, in modo da visualizzare le barre di scorrimento se l'immagine nel PictureBox eccede le dimensioni del form.
Il controllo **PictureBox1** ha queste proprietà:
Location = 0; 28
Size = AutoSize
Copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    ' Crea una variabile per memorizzare gli spostamenti del mouse:
    Dim PuntoPrecedente As Point = New Point(0, 0)

    ' Crea una superficie grafica :
    Dim AreaDisegno As Graphics

    ' Crea una variabile per memorizzare il nome e il percorso dell'immagine
scelta dall'utente:
    Dim PercorsoImmagine As String
```

```
    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        ' Il programma deve creare una superficie grafica corrispondente al
controllo
        ' PictureBox, sulla quale l'utente potrà disegnare con il mouse.
```

```
        ' Problema: il controllo PictureBox in apertura di programma è vuoto;
        ' non è possibile creare una superficie grafica da un'immagine nulla.
        ' Rimedio: creiamo un oggetto bitmap virtuale, corrispondente al
PictureBox, anche se vuoto:

        Dim ImmagineVirtuale As Bitmap
        ImmagineVirtuale = New Bitmap(PictureBox1.Width, PictureBox1.Height)
        ' Ora il controllo PictureBox ha un'immagine, anche se solo virtuale e
vuota:
        PictureBox1.Image = ImmagineVirtuale

        ' AreaDisegno = PictureBox1.creategraphics
        AreaDisegno = Graphics.FromImage(PictureBox1.Image)

        ' Migliora la qualità della grafica:
        AreaDisegno.SmoothingMode = Drawing2D.SmoothingMode.HighQuality

        ' Imposta la proprietà SizeMode del PictureBox in modo che il controllo
si adatti al contenuto:
        PictureBox1.SizeMode = PictureBoxSizeMode.AutoSize
        ' Imposta la proprietà AutoScroll del form in modo che questo visualizzi
le barre di
        ' scorrimento, se necessario:
        Me.AutoScroll = True

    End Sub
```

```
    Private Sub PictureBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseDown

        If e.Button = Windows.Forms.MouseButtons.Right Then

            ' Se l'utente ha premuto il tasto destro del mouse...
            ' ripristina la situazione iniziale:
            PictureBox1.Image = Image.FromFile(PercorsoImmagine)
            AreaDisegno = Graphics.FromImage(PictureBox1.Image)
            AreaDisegno.SmoothingMode = Drawing2D.SmoothingMode.HighQuality

        Else

            ' Altrimenti memorizza il punto in cui è stato effettuato il clic, e
cambia il cursore del mouse:
            PuntoPrecedente = e.Location
            PictureBox1.Cursor = Cursors.Hand

        End If

    End Sub
```

```
    Private Sub PictureBox1_MouseMove(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseMove

        ' Se l'utente muove il mouse tenendo il pulsante sinistro abbassato...
        If e.Button = Windows.Forms.MouseButtons.Left Then

            Dim PuntoCliccato As Point
            ' Memorizza il nuovo punto in cui trova il mouse:
            PuntoCliccato = e.Location

            Dim Penna As New Pen(Brushes.Red, 3)
```

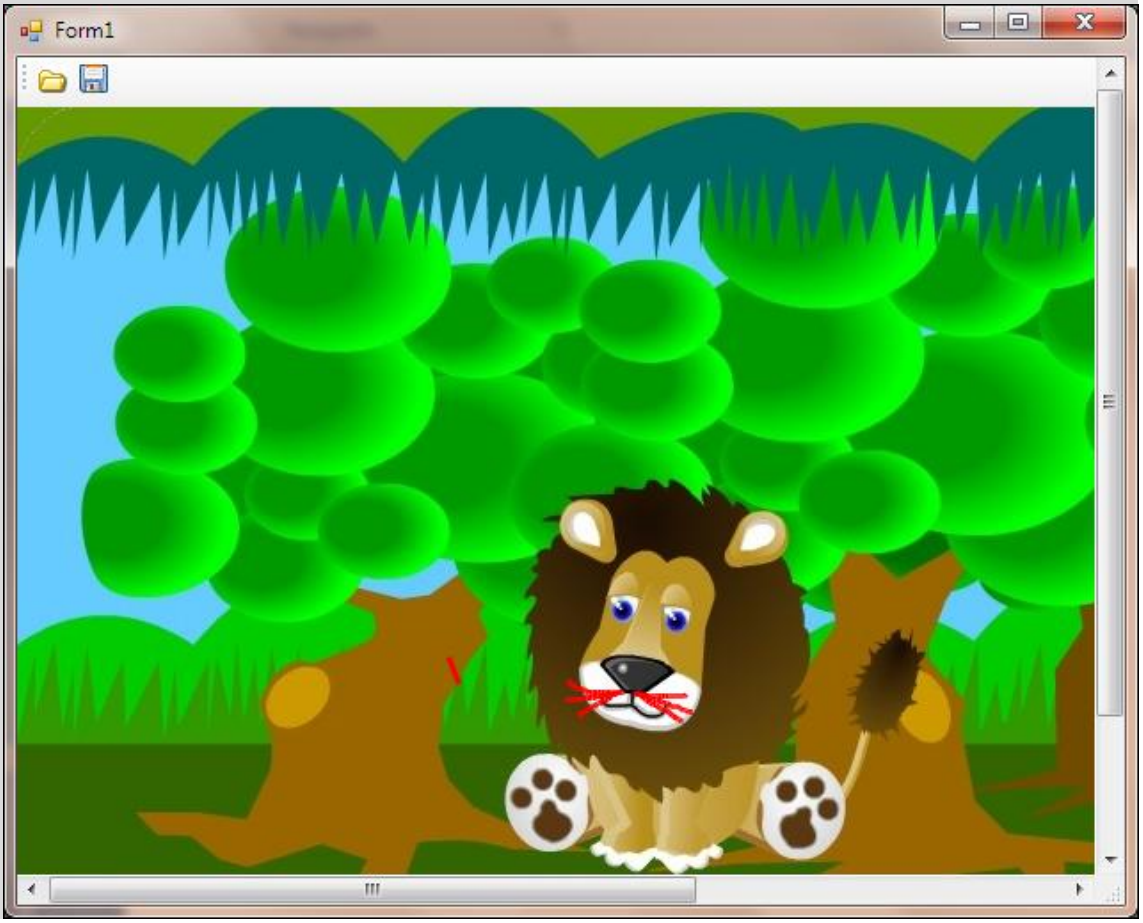
```
            ' Traccia una linea tra il punto precedente e il punto attuale in cui
si trova il mouse:
            AreaDisegno.DrawLine(Penna, PuntoPrecedente, PuntoCliccato)

            ' Memorizza il punto attuale come punto precedente, per la prossima
linea:
            PuntoPrecedente = PuntoCliccato

            PictureBox1.Invalidate()

            ' Elimina lo strumento Penna per liberare risorse di memoria nel
computer:
            Penna.Dispose()

        End If

    End Sub
```

```
    Private Sub PictureBox1_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseUp

        PictureBox1.Cursor = Cursors.Default

    End Sub

    Private Sub ToolStripButton1_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripButton1.Click
        OpenFileDialog1.FileName = ""

        ' Imposta un filtro per la ricerca di file con immagini:
        OpenFileDialog1.Filter = "File
immagini|*.bmp;*.jpeg;*.jpg;*.png;*.gif;*.wmf;*.tif"

        ' Verifica se l'utente ha scelto un file e visualizza questo file nel
TextBox:
        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then

            PercorsoImmagine = OpenFileDialog1.FileName
            PictureBox1.Image = Image.FromFile(PercorsoImmagine)

        End If
```

```
        ' Aggiorna la superficie grafica con la nuova immagine scelta
dall'utente:
        AreaDisegno = Graphics.FromImage(PictureBox1.Image)
        AreaDisegno.SmoothingMode = Drawing2D.SmoothingMode.HighQuality

    End Sub

    Private Sub ToolStripButton2_Click(sender As System.Object, e As
System.EventArgs) Handles ToolStripButton2.Click

        SaveFileDialog1.FileName = ""

        ' Imposta un filtro per il salvataggio di file di testo nella finestra di
dialogo:
        ' (come impostazione predefinita VB salva le immagini nel formto .png)
        SaveFileDialog1.Filter = "File immagini|*.png"
```

```
        ' Verifica se l'utente ha scritto un nome e un percorso valido e salva il
file:
        If SaveFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then

            PictureBox1.Image.Save(SaveFileDialog1.FileName)

        End If

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione (i baffi rossi del leone sono un'aggiunta dell'utente del programma):



192: My.Application.

My.Application fornisce strumenti per gestire alcune proprietà della applicazione in fase di progettazione.
My.Application.Info fornisce informazioni su diversi elementi dell'applicazione in progettazione: titolo del programma, autore, descrizione del contenuto, versione e altro ancora.
Ne vedremo un esempio nel prossimo esercizio.

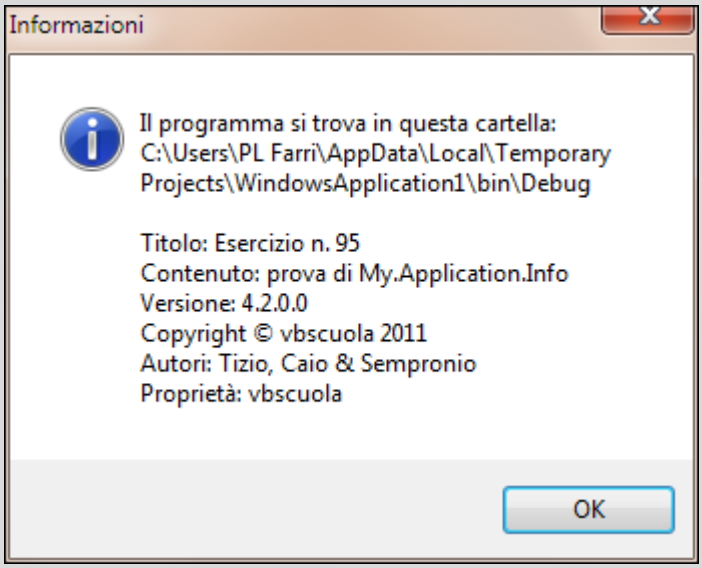
Esercizio 135: Uso di My.Application.Info.

In questo esercizio vediamo l'uso di **My.Application.Info** per ottenere alcune informazioni sull'applicazione in fase di progettazione.
Le informazioni raccolte man mano sono inserite in un testo che alla fine è visualizzato in un MsgBox.
Apriamo un nuovo progetto.
Copiamo e incolliamo nella Finestra del Codice questo listato:

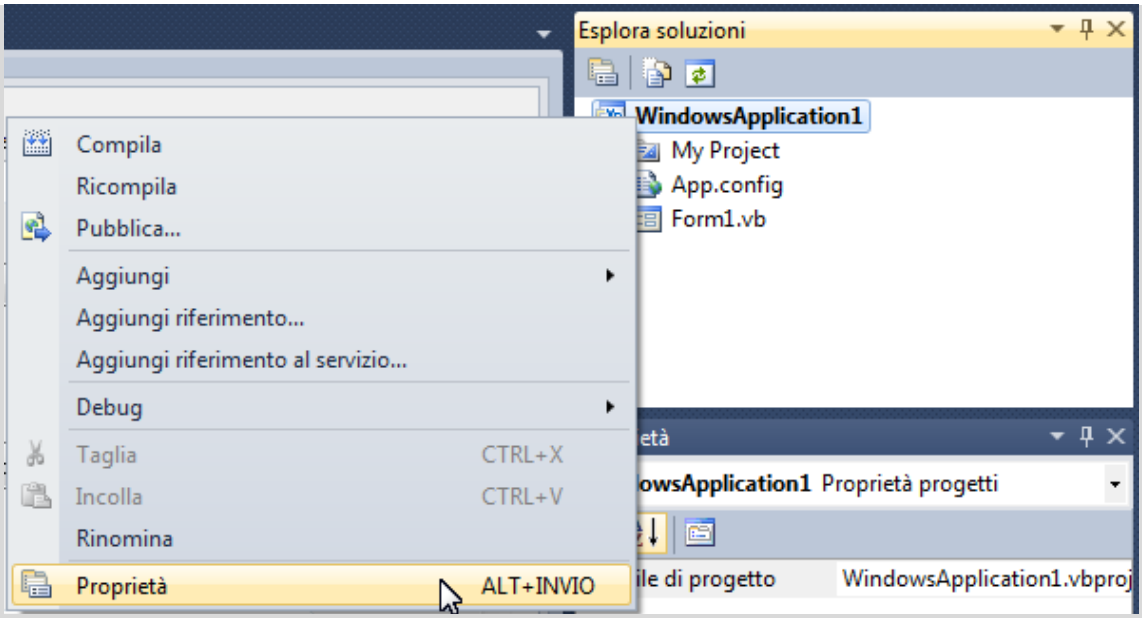
```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' all'avvio del programma, crea un oggetto destinato a contenere testo:
        Dim TestoMessaggio As String

        ' My.Application.Info.DirectoryPath legge la cartella in cui si trova il
programma:
        TestoMessaggio = "Il programma si trova in questa cartella: " & vbCrLf &
My.Application.Info.DirectoryPath & vbCrLf & vbCrLf

        ' My.Application.Info.Title legge il titolo del programma:
        TestoMessaggio &= "Titolo: " & My.Application.Info.Title & vbCrLf

        ' My.Application.Info.Description legge la descrizione del programma:
        TestoMessaggio &= "Contenuto: " & My.Application.Info.Description &
vbCrLf

        ' My.Application.Info.Version legge la versione corrente del programma:
        TestoMessaggio &= "Versione: " & My.Application.Info.Version.ToString &
vbCrLf

        ' My.Application.Info.Copyright legge i diritti di proprietà del
programma:
        TestoMessaggio &= My.Application.Info.Copyright & vbCrLf

        ' My.Application.Info.CompanyName legge i nomi degli autori del
programma:
        TestoMessaggio &= "Autori: " & My.Application.Info.CompanyName & vbCrLf
```

```
        ' My.Application.Info.Trademark legge i diritti di proprietà del
programma:
        TestoMessaggio &= "Proprietà: " & My.Application.Info.Trademark & vbCrLf

        ' visualizza il testo in un Message Box:
        MsgBox(TestoMessaggio, MsgBoxStyle.Information, "Informazioni")
        ' termina il programma
        End

    End Sub

End Class
```

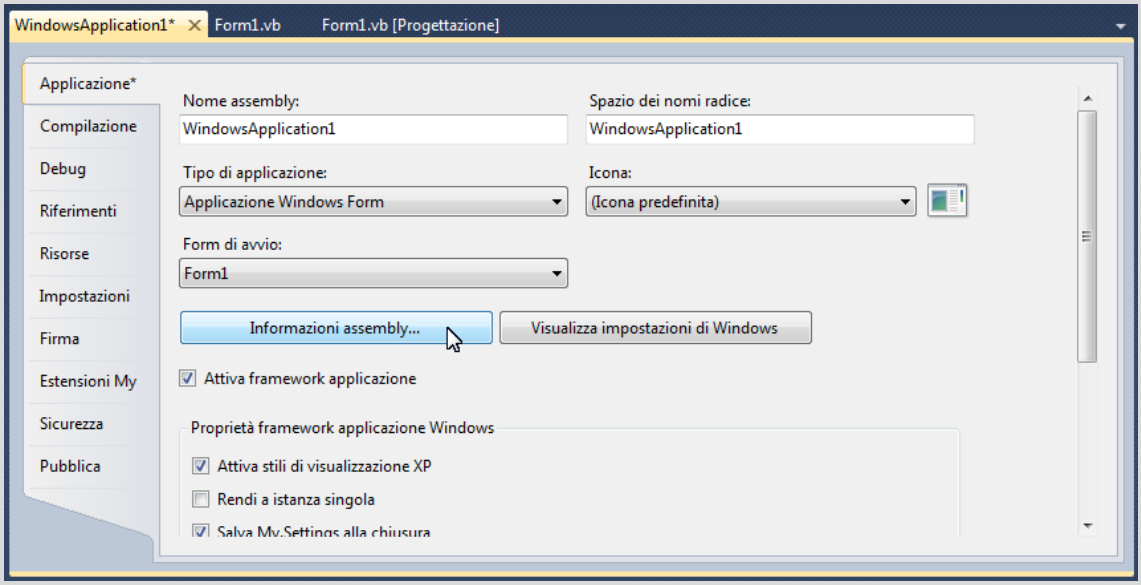
Ecco un'immagine del programma in esecuzione:

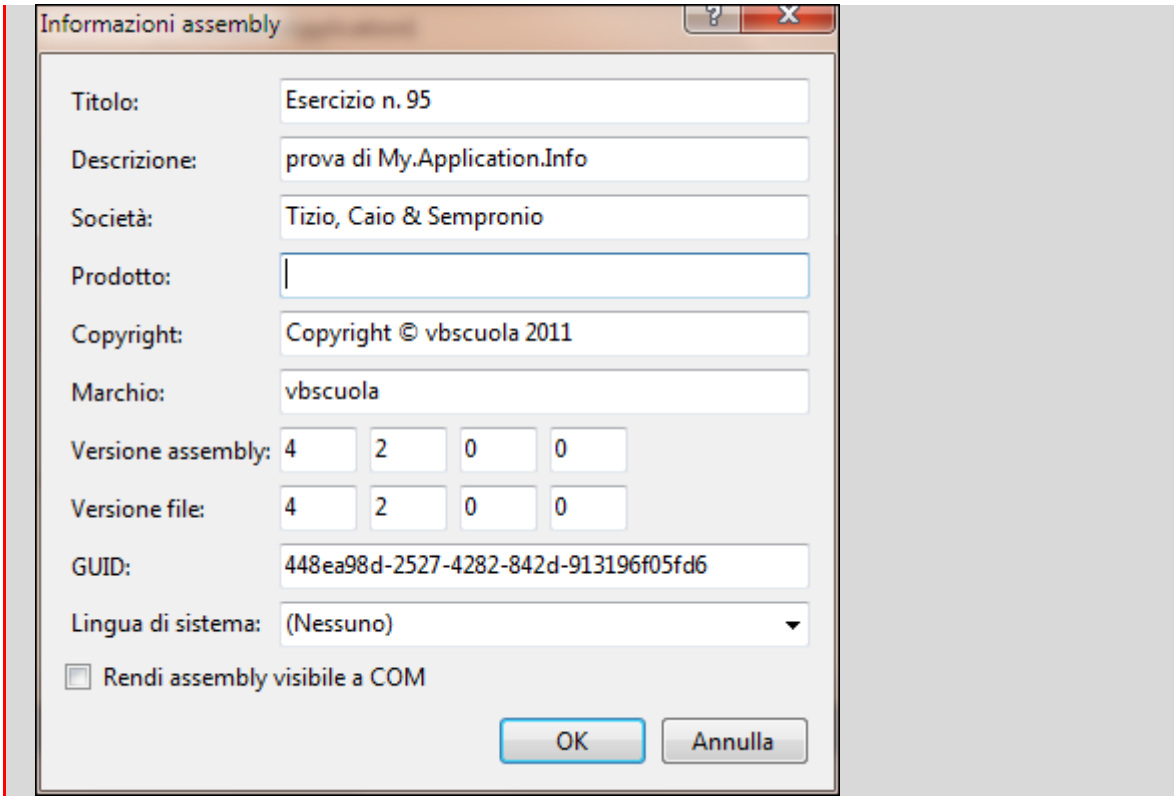


I dati che questo programma legge e riporta nel MsgBox sono impostati dal programmatore nelle proprietà del progetto.
Con un *clic* con il tasto destro del mouse sul nome dell'applicazione, apriamo la scheda delle sue proprietà:



Qui apriamo la scheda Informazioni **Assembly** e inseriamo i dati che ci interessano:

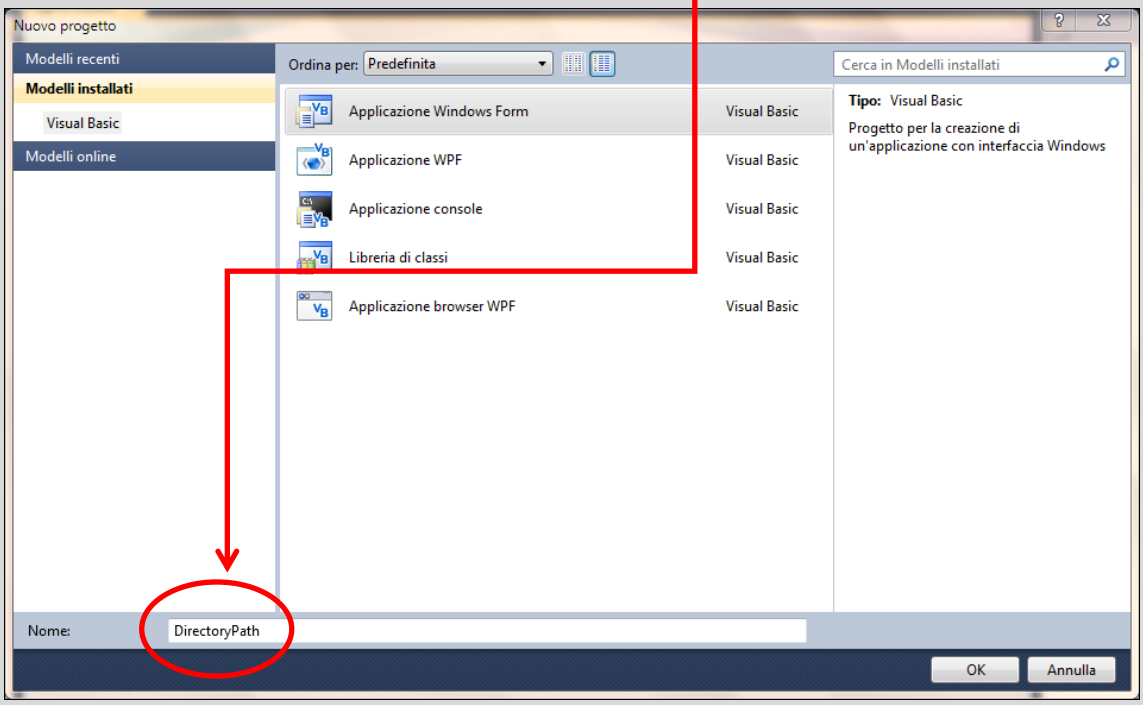




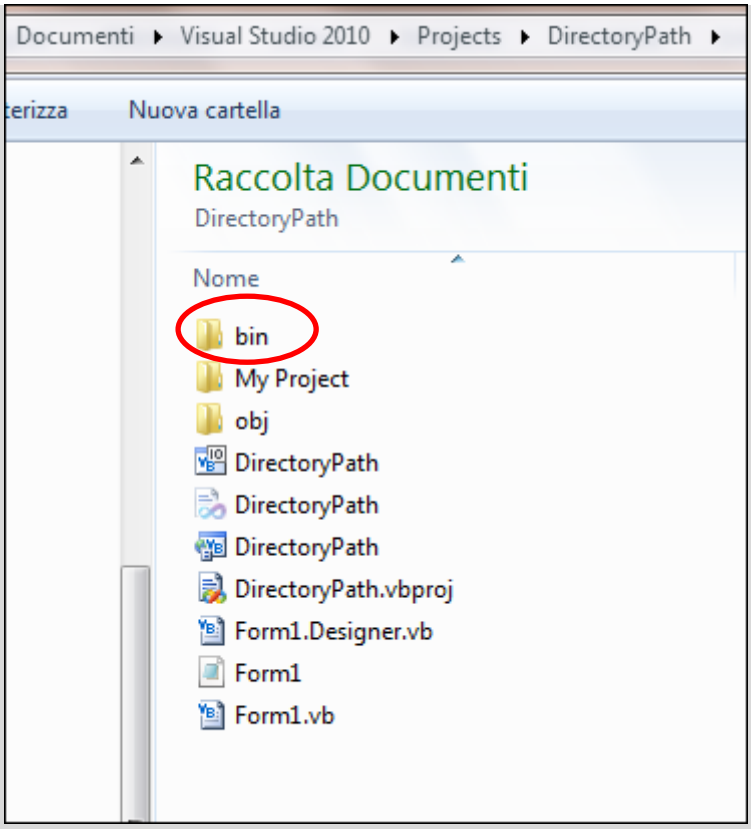
Un'informazione di particolare utilità è fornita da **My.Application.Info.DirectoryPath**, che indica in quale cartella si trova il file eseguibile della applicazione.
Questa indicazione è necessaria - ad esempio - quando il programmatore decide di distribuire assieme al suo programma altri file (audio, immagini o testi) che non sono compresi nelle risorse del programma.
In questo caso, i file che accompagnano il programma devono essere collocati nella cartella in cui si trova il programma base e **My.Application.Info.DirectoryPath** fa sì che il programma conosca questa cartella, per accedervi e per recuperarne i file necessari.
Nell'esercizio seguente vediamo un esempio di utilizzo di **My.Application.Info.DirectoryPath** per visualizzare un'immagine inserita nella cartella in cui si trova il programma base.

Esercizio 136: Apertura di un file con My.Application.Info.DirectoryPath.

Apriamo un nuovo progetto e diamogli il nome **DirectoryPath**:



Inseriamo nel form un pulsante **Button1** e un controllo **PictureBox**.
Prima di procedere, **salviamo** il progetto con un *clic* sull'icona **Salva tutto**.
Con **Esplora risorse** di Windows vediamo nella cartella **Documenti \ Visual Studio 2010 \ Projects \ DirectoryPath** le sottocartelle del progetto create da VB:



La sottocartella **DirectoryPath \ bin \ Debug** è quella che ci interessa, ai fini di questo esercizio, perché è qui che VB colloca il programma base durante la fase di progettazione.
Copiamo l'immagine **casetta.jpg** che si trova nella cartella **Documenti \ A scuola con VB \ Immagini** e incolliamola nella cartella **DirectoryPath \ bin \ Debug.**
Ora copiamo e incolliamo nella Finestra del Codice questo listato:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        PictureBox1.Image = Image.FromFile(My.Application.Info.DirectoryPath &
"/casetta.jpg")


    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

Quando questo programma sarà installato su altri computer, l'immagine **casetta.jpg** che nel computer del programmatore, si trova nella cartella **DirectoryPath \ bin \ Debug**, ovviamente non sarà disponibile nel computer dell'utente.
Il programmatore dovrà dunque preoccuparsi di distribuire questa immagine assieme al suo programma e dovrà fare in modo che il programma e l'immagine **casetta.jpg** finiscano nella stessa cartella. Vedremo queste operazioni più avanti, nel capitolo dedicato alla distribuzione del progetto finito.

PARTE VI: GLI ARCHIVI DI DATI.

[…] E giudicò che la biblioteca di San Vittore era davvero magnifica, specialmente per certi libri che vi trovò: dei quali ecco il catalogo:
[…]
Marmotretus, De babuinis et scimiis, cum commento Dorbellis;
Decretum Universitatis Parisiensis super poppelonitute donnettarum ad placitum;
[…]
Ingeniositas invocandi Diabolos et Diabolas; per Magistrum Guingolfum;
La Pietanza dei perpetuoni;
La Moresca degli eretici;
Le Fanfaluche del Caetano;
Immollagrugni Doctoris cherubici, De origine gattemortuarum et torticollorum ritibus, libri septem;
Sessantanove Breviari ben bisunti;
Il Pancione dei Cinque Ordini dei Mendicanti;
[…]
Cacatorium Medicorum
Lo Spazzacamino degli Astrologhi […][86]

E' difficile immaginare l'esistenza degli esseri umani senza liste da compilare, elenchi da scorrere, cataloghi da consultare… Siamo archivi di informazioni, viventi e in continua evoluzione. La memoria è il magazzino delle nostre conoscenze e nello stesso tempo è l'insieme delle regole con le quali registriamo le nuove informazioni, le colleghiamo a quelle già esistenti e le richiamiamo alla nostra coscienza quando ci servono.

La tecnologia informatica sin dall'inizio si è proposta come strumento di supporto a questa esigenza di memorizzazione di dati, creando forme di archivi flessibili nelle dimensioni, ma rigidi dal punto di vista delle regole di memorizzazione, quali lo schema

[86] Da: F. Rabelais, Gargantua e Pantagruele, Torino, Einaudi, 2005.

ad albero, che organizza tutti i file registrati nei dischi fissi, e le tabelle a doppia entrata dei fogli di calcolo e degli archivi.

Il modello di archivio di dati (**database**) disponibile in VB è quello della tabella a doppia entrata.
Ogni archivio è composto di una o più tabelle con i dati distribuiti su schede (**record**), che contengono tutte un uguale numero di campi (**field**).
Nell'esempio seguente, vediamo un archivio di dati formato da una sola tabella; questa tabella raccoglie alcune informazioni su alcuni libri.
Nella tabella sono memorizzati i dati di quattro libri, vi troviamo dunque quattro schede (una scheda per ogni libro, ogni scheda ha sei campi):

	Campo (field) **titolo**	Campo (field) **autore**	Campo (field) **editore**	Campo (field) n. pag.	Campo (field) **anno di ediz.**	Campo (field) **incipit**
Scheda (record) 1	Il suo libro	Bianchi	Pinco	236	1997	C'era una volta…
Scheda (record) 2	Il tuo libro	Rossi	Pallino	102	2010	Non ricordo…
Scheda (record) 3	Il mio libro	Sempronio	Tizio	156	2007	In quel tempo…
Scheda (record) 4	Questo libro	Verdi	Caio	386	1990	La pagina è bianca …

Tabella 36: Esempio di archivio di dati formato da una tabella con quattro schede.

Ogni riga (scheda) contiene le informazioni su un libro; le informazioni per ogni libro sono specificate nelle colonne (campi) che l'autore del database ha ritenuto necessario inserire nella tabella.
Ovviamente il numero di campi di una scheda, cioè la quantità e il tipo di informazioni relative a ogni record, variano secondo le esigenze di chi progetta l'archivio di dati; in questo caso, trattandosi di libri, esse potrebbero includere, per esempio, il codice identificatore del libro, il luogo e il numero di edizione, l'autore della prefazione, la collocazione nello scaffale, ecc.
E' dunque necessario che il creatore di un database rifletta accuratamente sulle sue esigenze e su ciò che intende memorizzare: aggiungere (o eliminare) campi in una fase successiva è possibile ma non consigliabile, in quanto nel migliore dei casi si dovrà procedere alla revisione di ogni singola scheda (riga) della tabella.

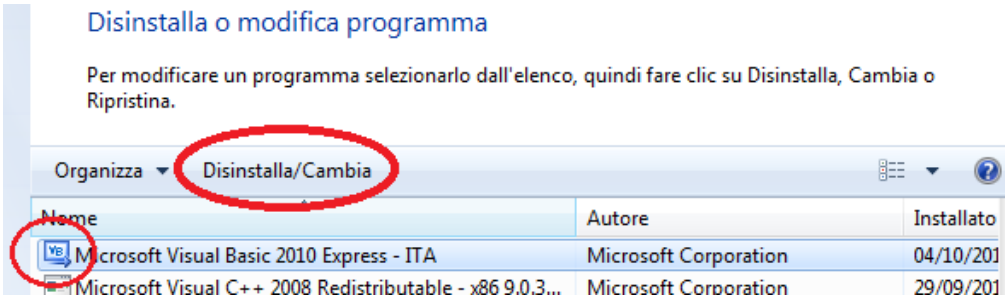
Il numero delle schede è aperto: esso riflette l'andamento delle operazioni di registrazione/cancellazione di dati e non è necessario definirle preventivamente nella fase di progettazione di un database.

Lo strumento per creare e gestire archivi di dati in VB è **Microsoft SQL Server 2008 Express SP1**.
L'acronimo **SQL** (generalmente pronunciato *siquel*) sta per Structured Query Language (Linguaggio Strutturato di Ricerca).
Il **SQL** è un linguaggio di programmazione espressamente creato, e poi sviluppato nel tempo[87], per la gestione dinamica di archivi di dati; non fa parte del linguaggio VB e la sua installazione è facoltativa, da selezionare durante la procedura di installazione di VB:

Figura 250: L'installazione di Microsoft SQL Server 2008 Express SP1.

Qualora SQL Server 2008 non fosse stato installato nel sistema assieme al linguaggio VB, occorre procedere in questo modo:

- aprire il **Pannello di Controllo**, selezionare **Disinstalla un programma**, cercare e selezionare la voce **Microsoft Visual Basic** (o Visual Studio) 2010 Express e fare *clic* **su Disinstalla/Cambia**;



[87] La prima versione, chiamata SEQUEL, fu sviluppata da IBM all'inizio degli anni settanta.

- nella finestra successiva selezionare **Aggiungi prodotti facoltativi** e proseguire con un *clic* su **Avanti**;

- Concludere l'aggiornamento scegliendo di installare nel sistema anche **SQL Server 2008 Express© Service Pack 1**.

In questi capitoli dedicati agli archivi di dati vedremo in particolare le funzionalità di **SQL Server Compact Edition 3.5© R2**, un componente specifico per la gestione di archivi non condivisi in rete e collocati su singoli computer.

Capitolo 35: CREAZIONE DELL'ARCHIVIO.

Il linguaggio SQL è un linguaggio di programmazione autonomo, dedicato alla creazione e della gestione di database; è dotato di propri comandi, operatori, modi di definizione delle variabili diversi da quelli di VB.
In queste pagine ci limiteremo ad esaminarne gli elementi essenziali, necessari per creare e interagire con un archivio di dati.
Proponiamo alle lettrici e ai lettori un percorso concreto, che potrà servire come esempio per lo sviluppo di progetti simili: la creazione di un programma dedicato alla gestione dell'anagrafe degli alunni di una classe.
I due capitoli dedicati a questo progetto non contengono esercizi, ma il loro contenuto è in pratica un unico esercizio, al termine del quale le lettrici e i lettori avranno sperimentato le operazioni necessarie per creare un archivio di dati e un'interfaccia per interagire con esso.

L'applicazione **Anagrafe alunni**, completa, con i sorgenti, è disponibile nella cartella **Documenti \ A scuola con VB \ Applicazioni.**

Obiettivo del progetto è la creazione di un programma per la gestione dell'anagrafe degli alunni di una classe: l'utente del programma potrà aggiungere, modificare e cancellare alcuni dati relativi a nascita, residenza e caratteristiche fisiche di ogni alunno; potrà inoltre filtrare le stesse informazioni per ottenere gruppi omogenei di alunni che rispondano a determinati criteri di ricerca.

In questo capitolo sono illustrate le fasi del progetto dedicate al database:

- Creazione della **struttura** dell'archivio. Per struttura dell'archivio s'intendono le tabelle che lo compongono (nel nostro caso ne avremo solo una) e le colonne (i campi) che ne fanno parte. Di ogni campo bisognerà indicare il nome, il contenuto e il tipo di dati, specificando se si tratterà di un testo, di un numero o di una data.
- Collegamento dell'archivio di dati al progetto VB mediante la creazione di una apposita **connessione** tra il database e il programma.
- Creazione di uno strumento (il **dataset**) contenente le istruzioni per l'interazione con l'archivio e la sua interrogazione, alla ricerca di dati, da parte del programma VB.

Nel prossimo capitolo vedremo la progettazione dell'interfaccia, cioè delle finestre necessarie (il form principale e i form da esso dipendenti) perché l'utente possa accedere alle varie funzioni del programma.

193: Connessione a un archivio dati.

Apriamo un nuovo progetto di VB; gli diamo il nome **Anagrafe alunni**:

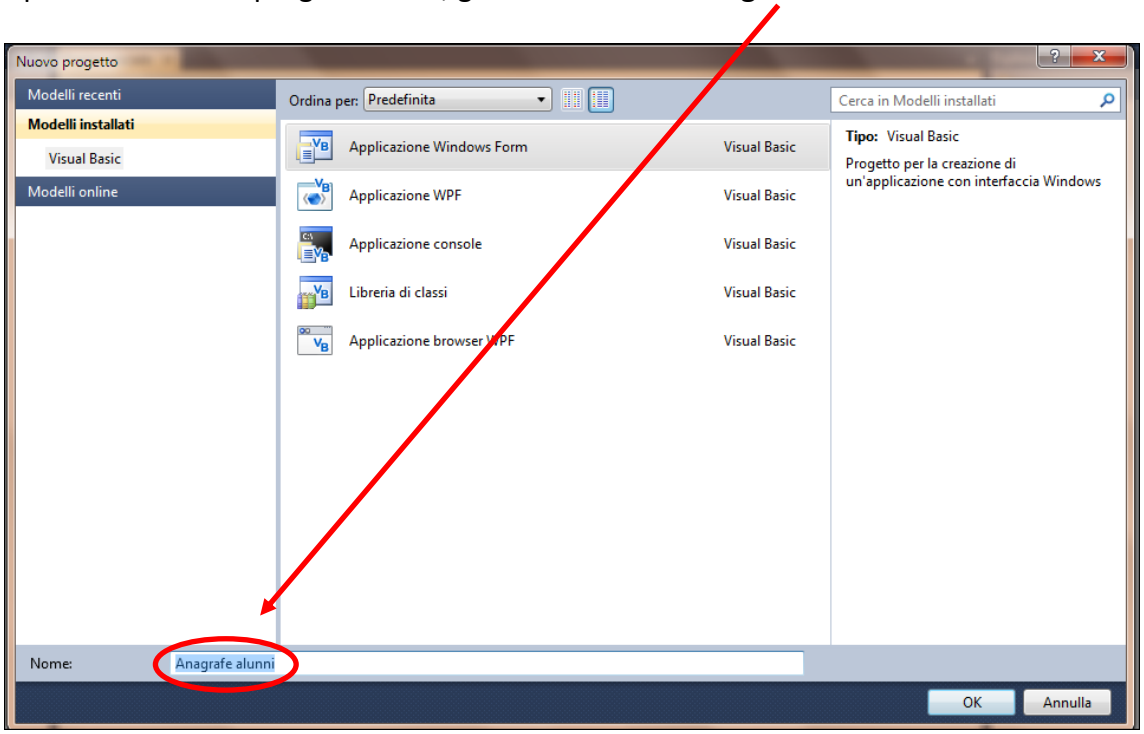


Figura 251: Avvio del progetto Anagrafe alunni.

Nella finestra **Esplora soluzioni**, con un *clic* del tasto destro del mouse sul file **Form1.vb**, scegliamo il comando **Rinomina** e modifichiamo il nome del file in **frmPrincipale.vb**
Nella **Finestra Proprietà**, modifichiamo la proprietà **Name** del **Form1**:

- (**Name**) = **frmPrincipale**

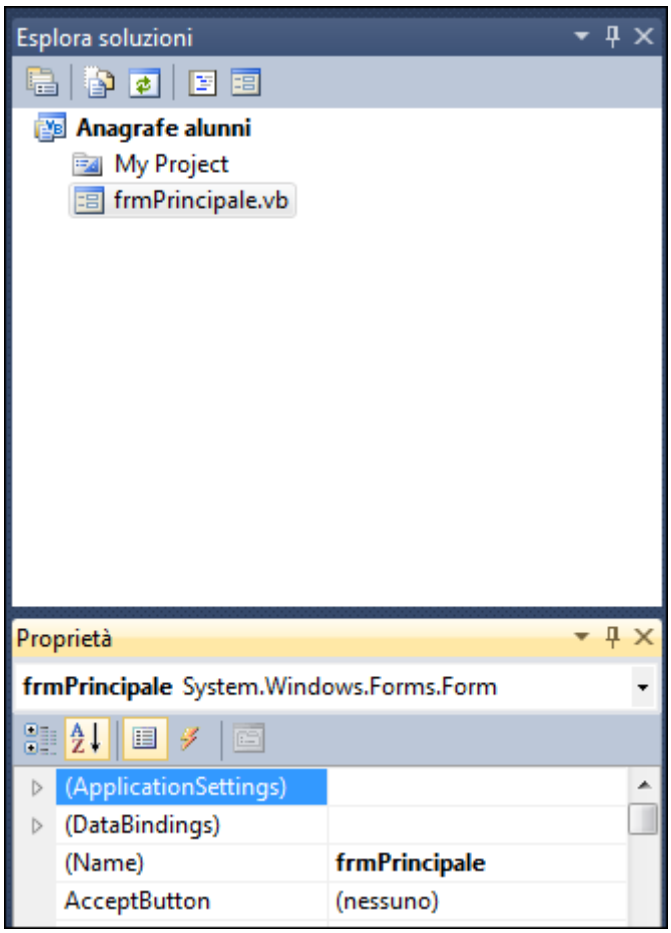


Figura 252: Denominazione del form principale.

Con un *clic* sul pulsante **Salva tutto,** nella barra delle icone, mettiamo al sicuro il progetto, confermando le opzioni che compaiono nel box **Salva progetto**.
Il progetto Anagrafe alunni è così salvato nella cartella **Documenti / Visual Studio 2010 / Projects / Anagrafe alunni**.

Iniziamo le operazioni per la creazione dell'archivio di dati: nella barra dei menu testuali di VB, in alto, facciamo un *clic* su **Dati**.
Questo menu mostra per ora solo due voci:

- **Mostra origine dati** (cioè: mostra gli archivi eventualmente già presenti nel progetto, e
- **Aggiungi nuova origine dati** (nel caso che non siano presenti archivi nel progetto o comunque si voglia aggiungere un nuovo archivio a quelli già esistenti).

Nel nostro caso non abbiamo ancora alcun archivio dati nel progetto, per cui facciamo un *clic* su **Aggiungi nuova origine dati**…

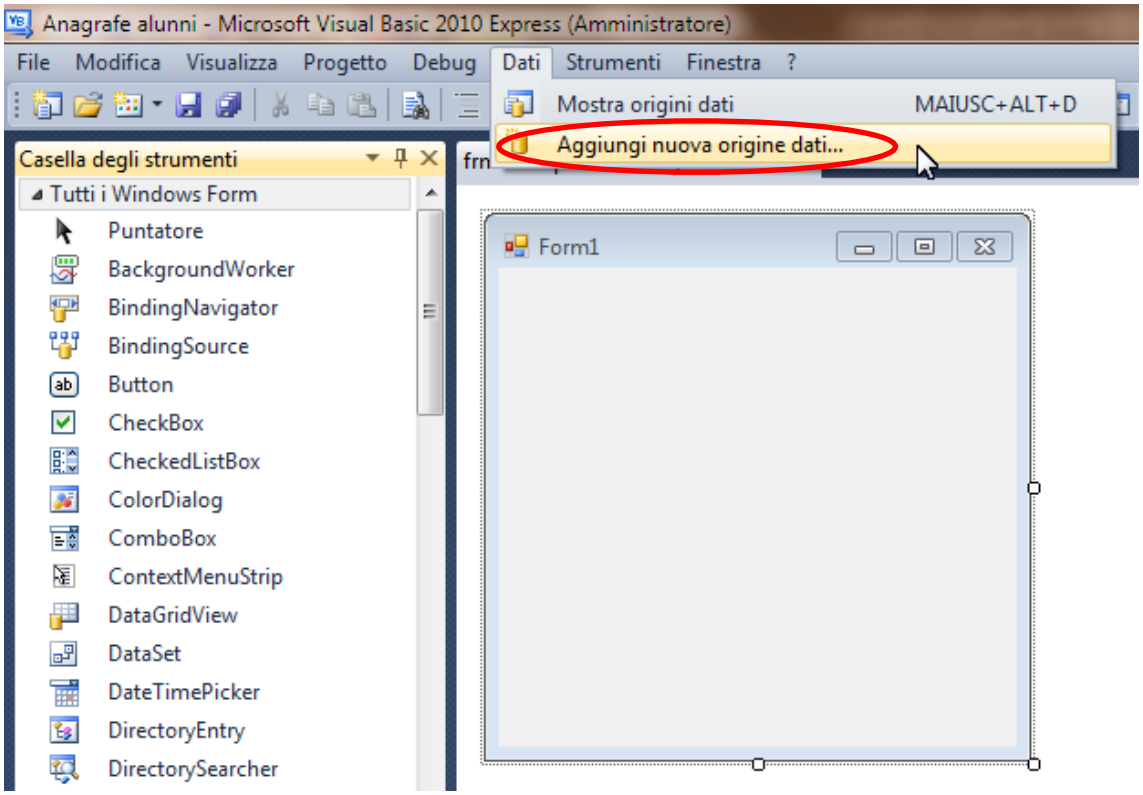


Figura 253: Avvio dell'inserimento di un nuovo archivio di dati nel progetto.

Si susseguono ora alcune finestre denominate **Configurazione guidata origine dati** che ci guidano alla creazione del nuovo database e al suo collegamento con il programma.

Iniziamo dalla finestra **Seleziona un tipo di origine dati**: qui ci viene chiesto di speficare quale tipo di **nuova origine dati** intendiamo creare.
Facciamo un *clic* su **Database** (archivio di dati) e su **Avanti**:

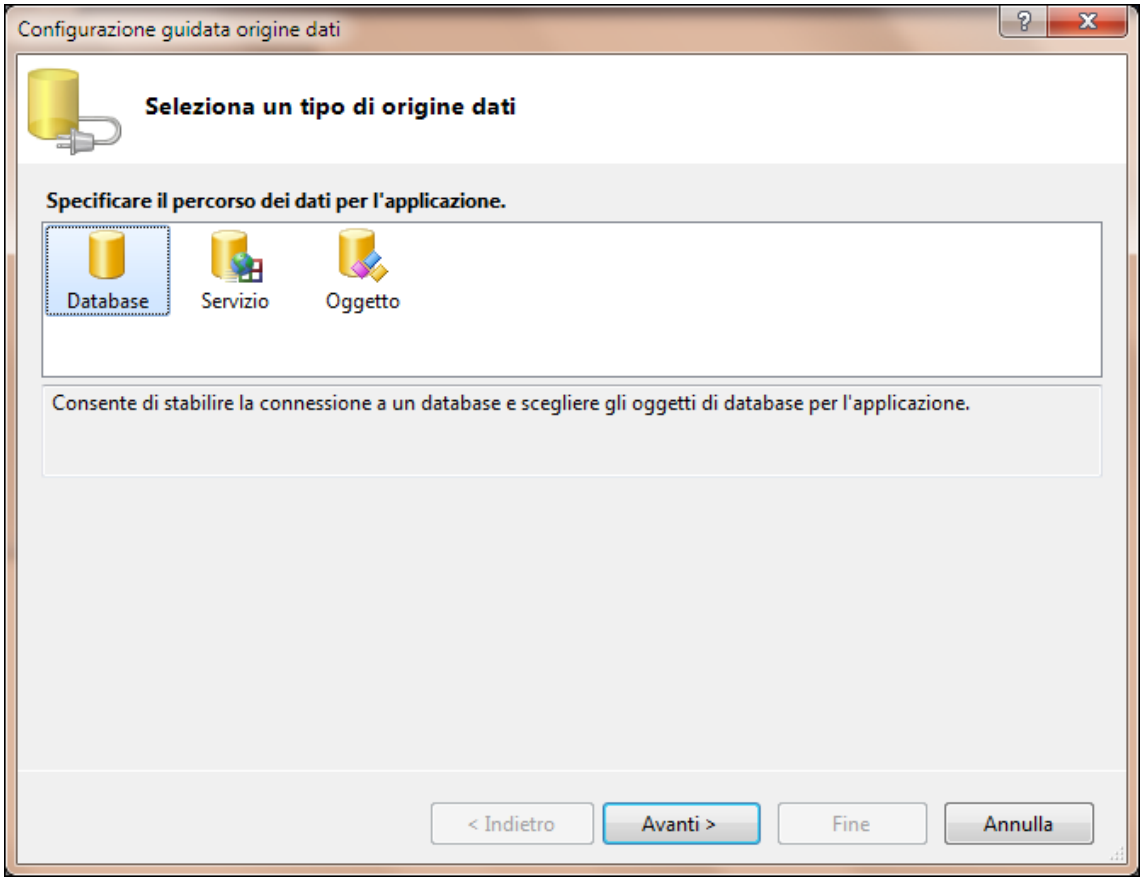


Figura 254: Scelta del tipo di origine dei dati.

Come modello di database, scegliamo il modello **Dataset** (= tabella con righe e colonne) e facciamo un *clic* sul pulsante **Avanti.**

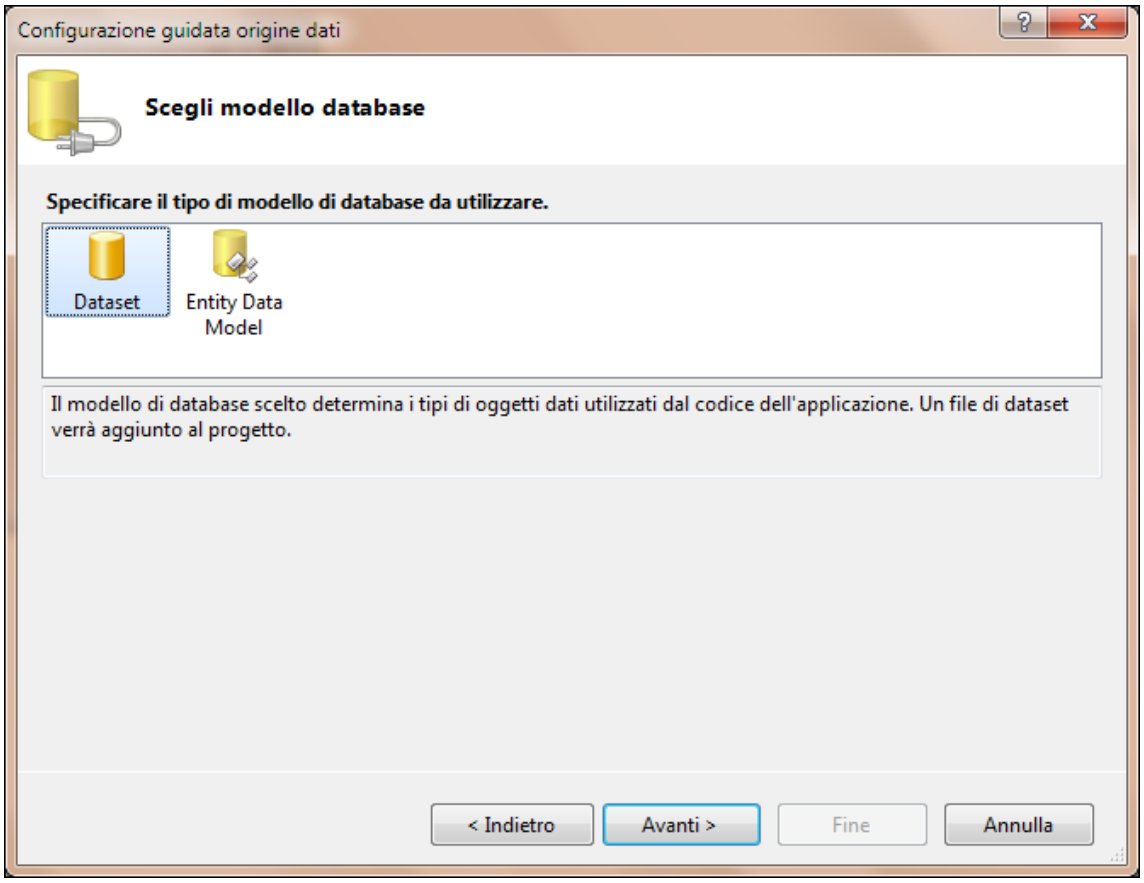


Figura 255: Scelta del modello di archivio di dati.

Ora dobbiamo impostare la connessione tra programma e il database.
Per impostare questa connessione è necessario indicare come si chiama l'archivio, e qual è la sua collocazione.
Per ora non esiste ancora alcuna connessione, per cui facciamo un *clic* su **Nuova connessione** per crearne una nuova:

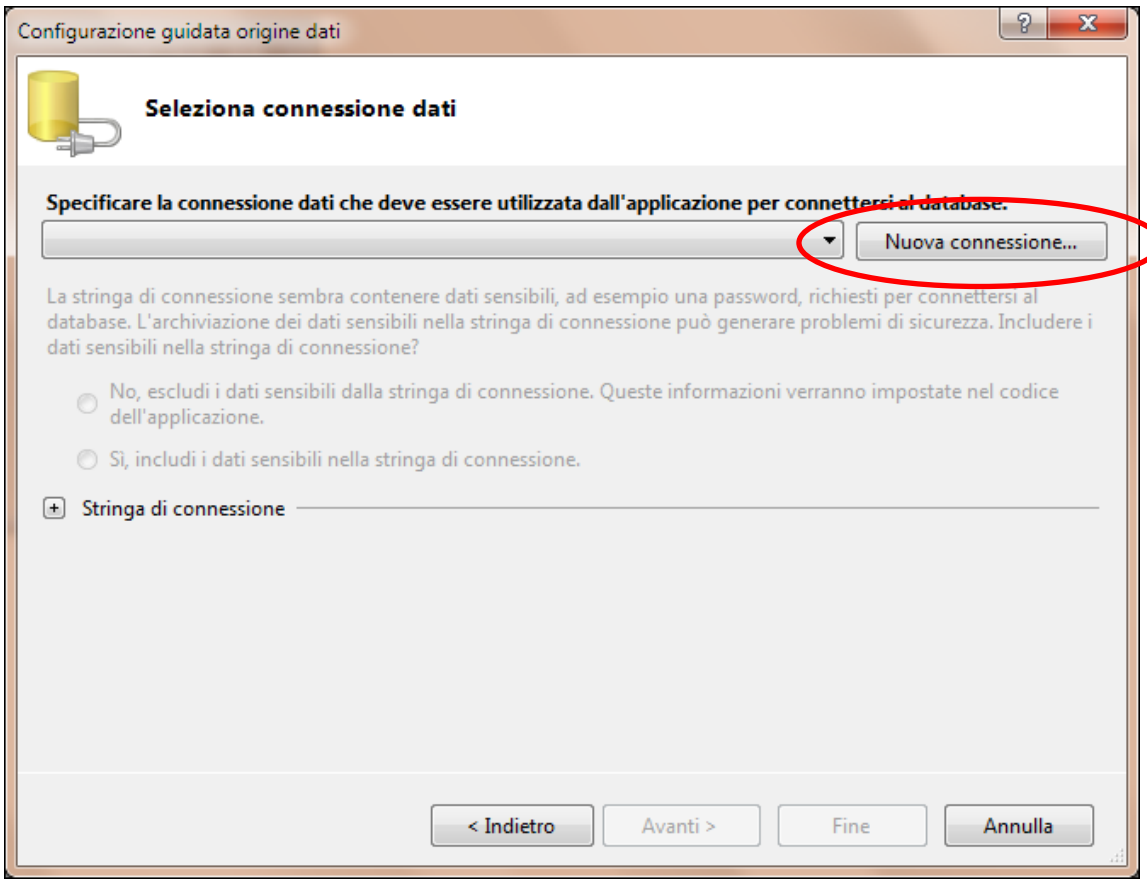


Figura 256: Scelta della connessione all'archivio di dati.

Nella finestra che si apre, **Aggiungi connessione**, effettuiamo queste operazioni:

- Controlliamo che **l'origine dati** corrisponda a **SQL Server Compact 3.5**. In caso contrario facciamo un *clic* su **Modifica**, per cercare questa origine nell'elenco che si apre. Se SQL Server CE 3.5 non compare nell'elenco, è necessario ripetere la procedura di installazione di VB.
- Specifichiamo che l'origine dei dati si troverà nelle **Risorse del computer** (clic sulla voce corrispondente)
- Facciamo *clic* su **Crea**, per creare un nuovo database.

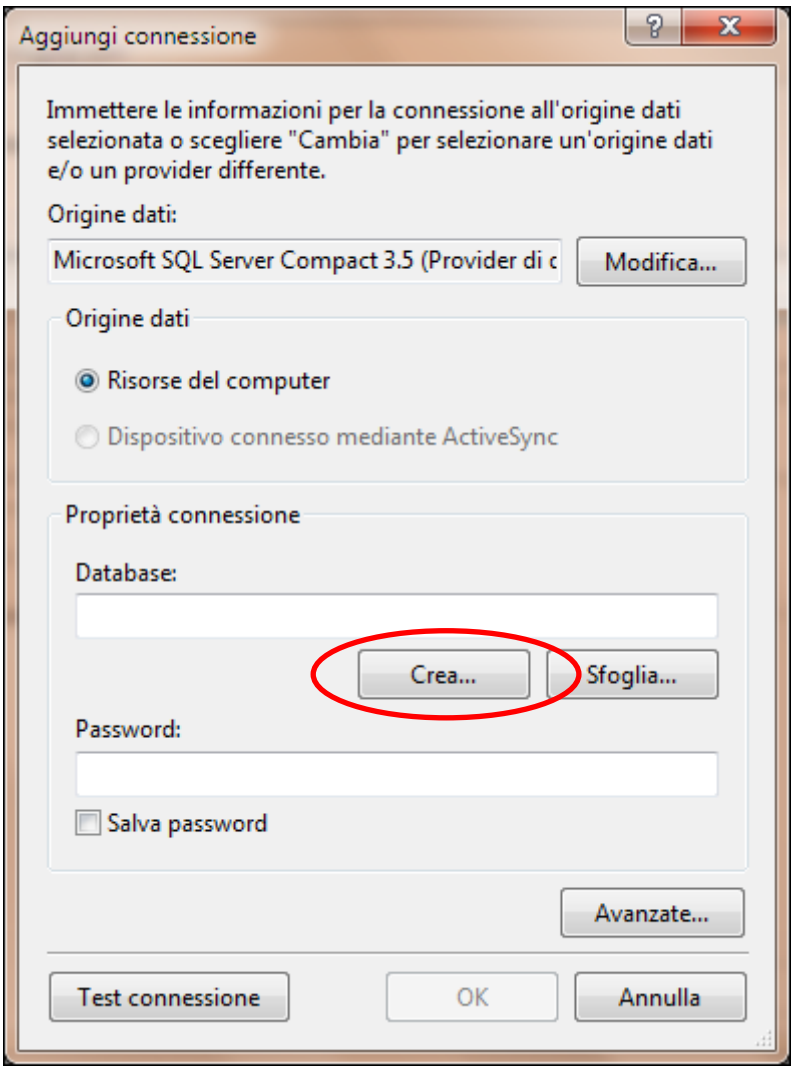


Figura 257: Creazione di un nuova connessione programma / database.

Siamo giunti così al momento della creazione vera e propria del nuovo database: in questa finestra ne indichiamo la collocazione, il nome e la lingua.
Nel nostro esempio l'archivio si chiamerà **Anagrafe** e si troverà nel disco fisso del computer in questa collocazione:
C:/Anagrafe[88]
Omettiamo l'impostazione di una password e facciamo *clic* su **OK**:

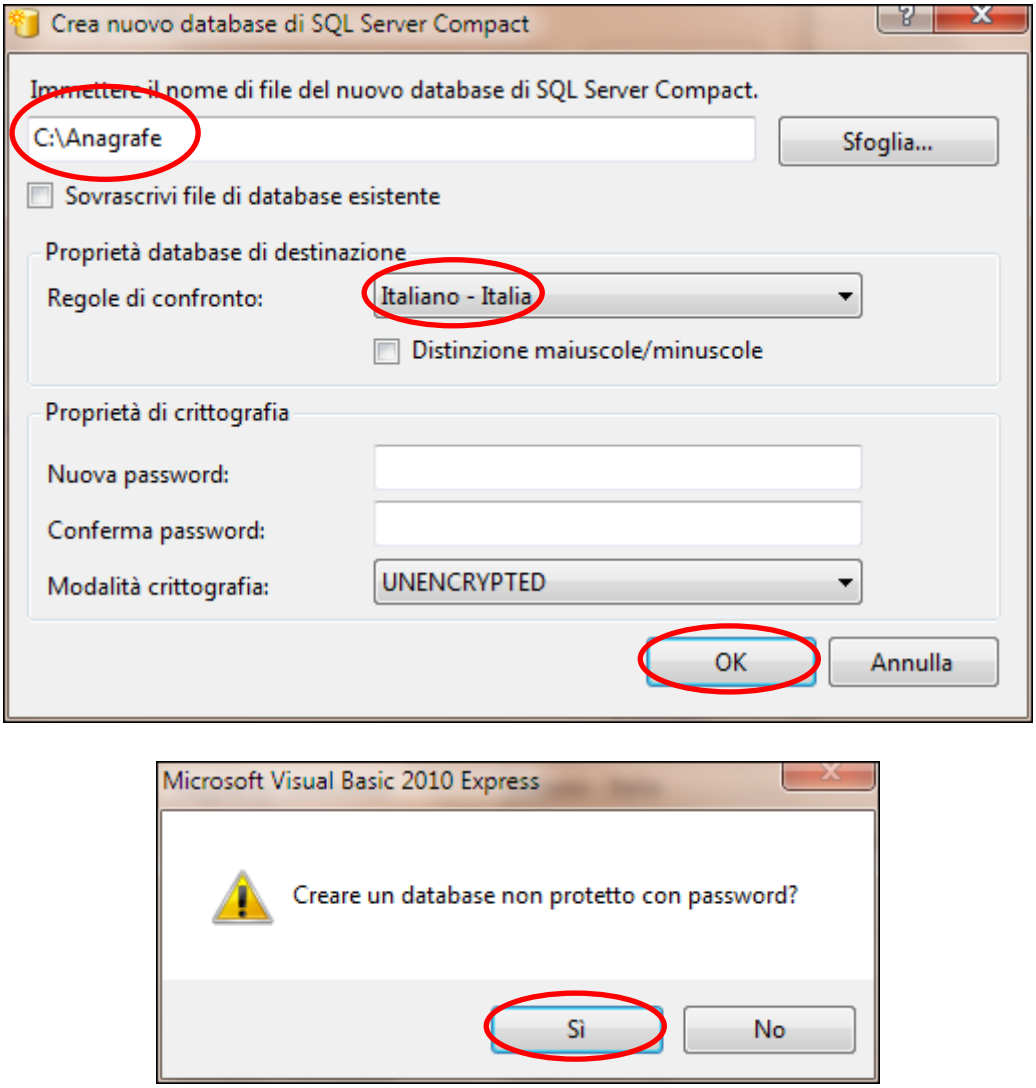


Figura 258: Creazione di un nuovo database.

Ora disponiamo di un database e possiamo ripercorrere a ritroso le finestre che ci hanno portato sino qui, per connettere il database al programma.

[88] Per creare l'archivio nella collocazione suggerita (C:/Anagrafe) è necessario che l'utente del computer abbia i privilegi di **amministratore**. In caso contrario, è possibile modificare il percorso e salvare il database nella cartella **documenti**, che non ha problemi di autorizzazioni.

Tornati alla finestra **Aggiungi connessione** troviamo ora impostata la connessione al database appena creato: **Anagrafe.sdf**. L'estensione **.sdf** è un acronimo che sta per **SQL Database File**, è utilizzata per i file di database creati per Microsoft SQL Server Compact.
Facciamo un *clic* sul pulsante **Test connessione**.

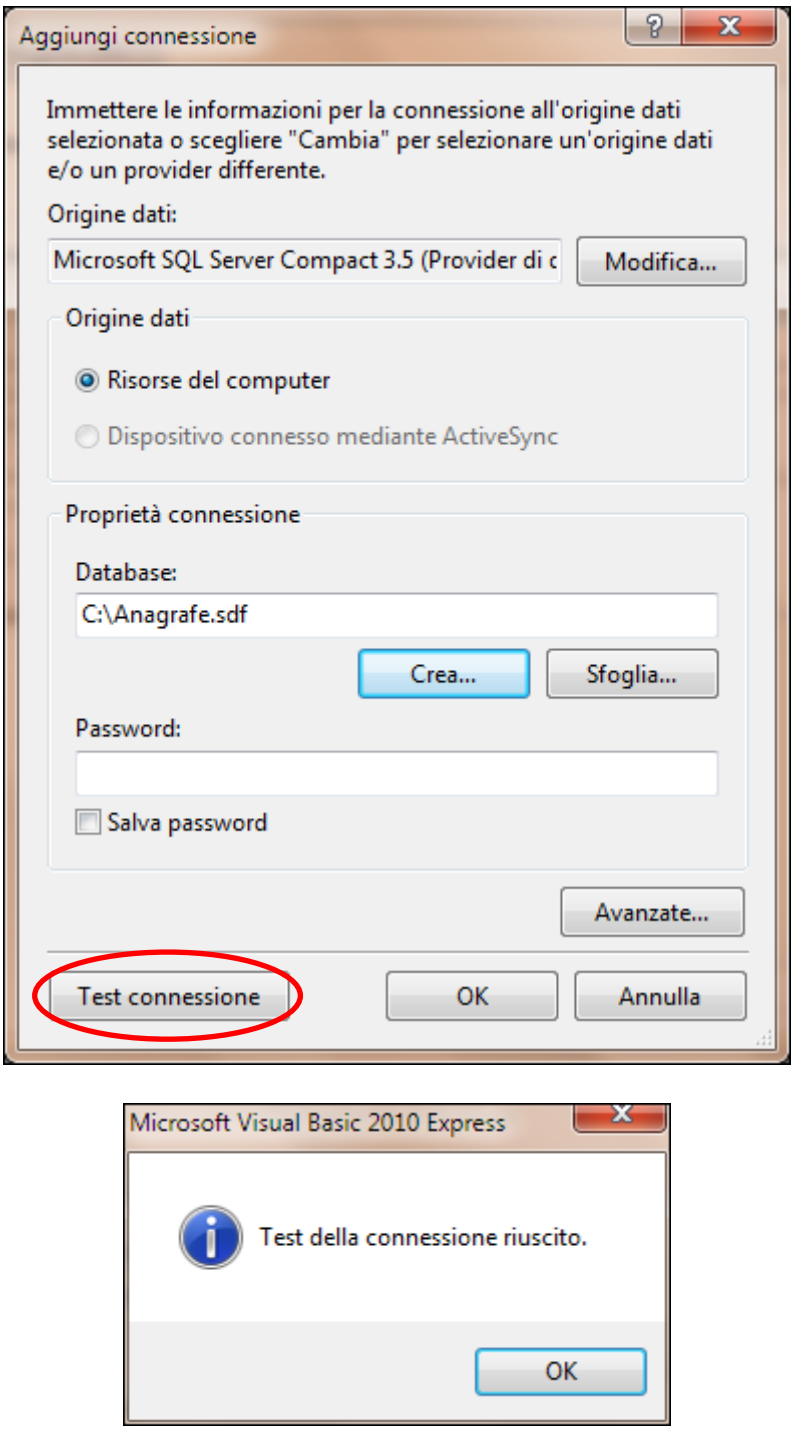
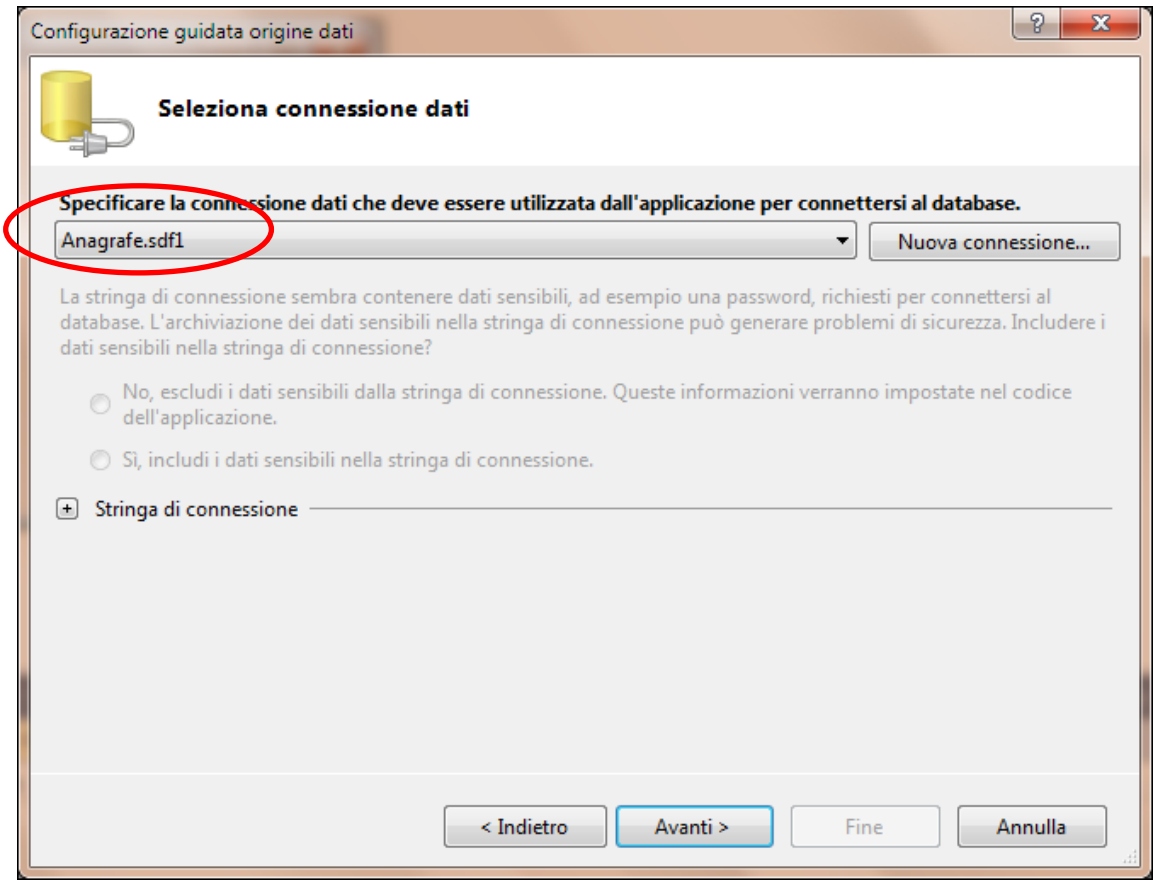
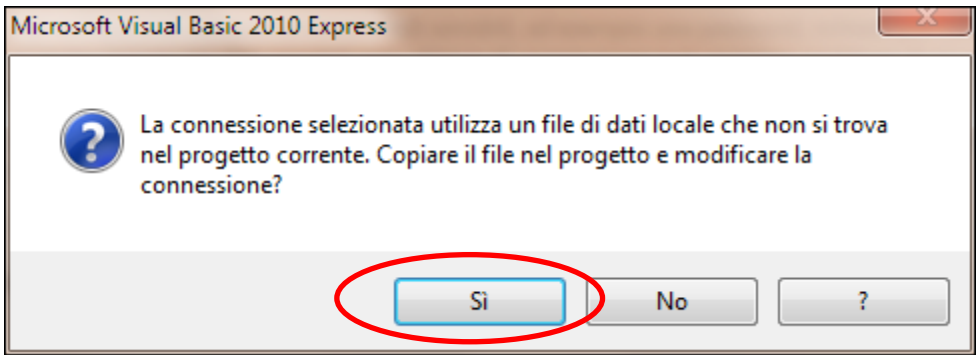


Figura 259: Il test di connessione al nuovo database.

Torniamo così alla finestra **Seleziona connessione dati**, che abbiamo lasciato vuota al passaggio precedente. Qui ora compare, già selezionata, la connessione **Anagrafe.sdf**. Facciamo un *clic* su **Avanti** per confermarla, e nelle finestre che seguono, confermiamo tutte le richieste che si succedono, sino a concludere queste operazioni con un *clic* sul pulsante **Fine**.



Siccome abbiamo creato il database nella posizione **C:/Anagrafe**, VB chiede di farne una copia nelle cartelle del progetto. Facciamo *clic* su **Sì**: da questo momento il database **Anagrafe.sdf** è copiato nella sottocartella **Anagafe alunni / bin / Debug**. Il programma che stiamo creando opererà con questa copia, dimenticando il file originale, che non verrà più modificato.



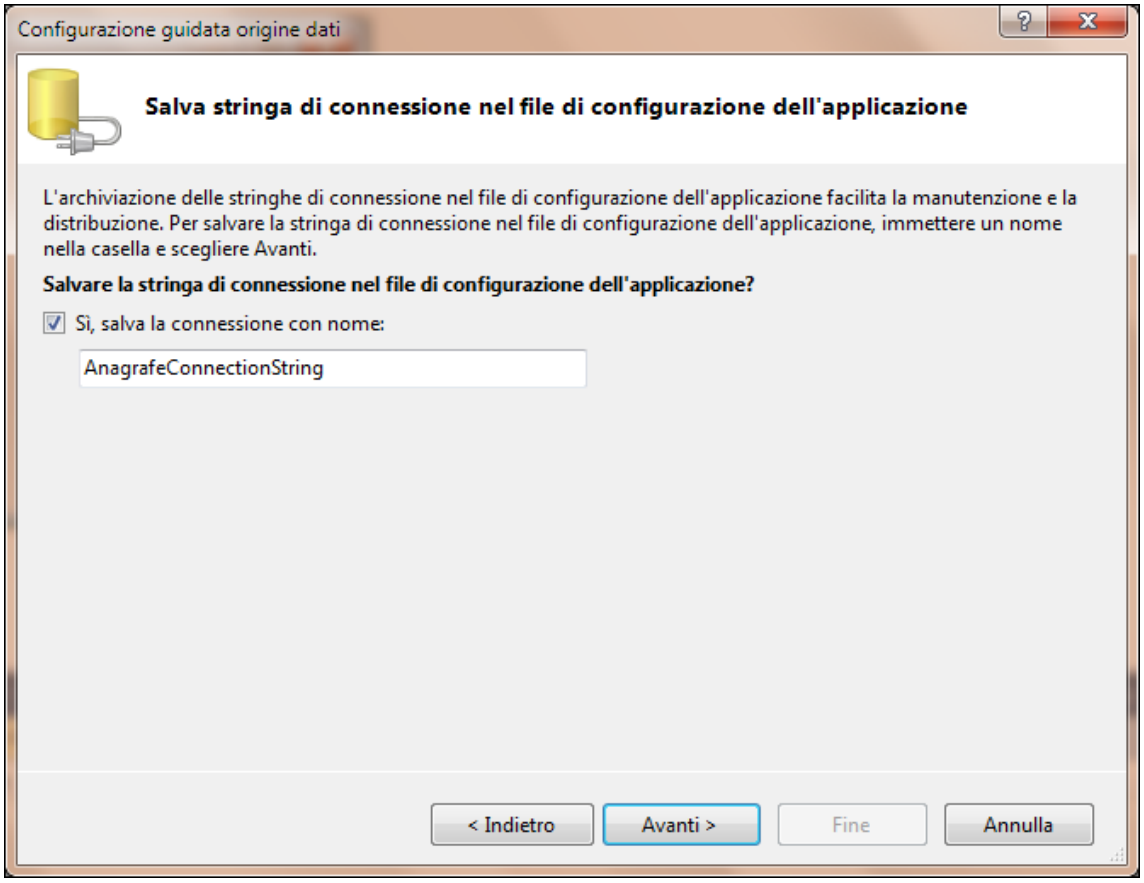


Figura 260: Selezione e salvataggio della connessione dati.

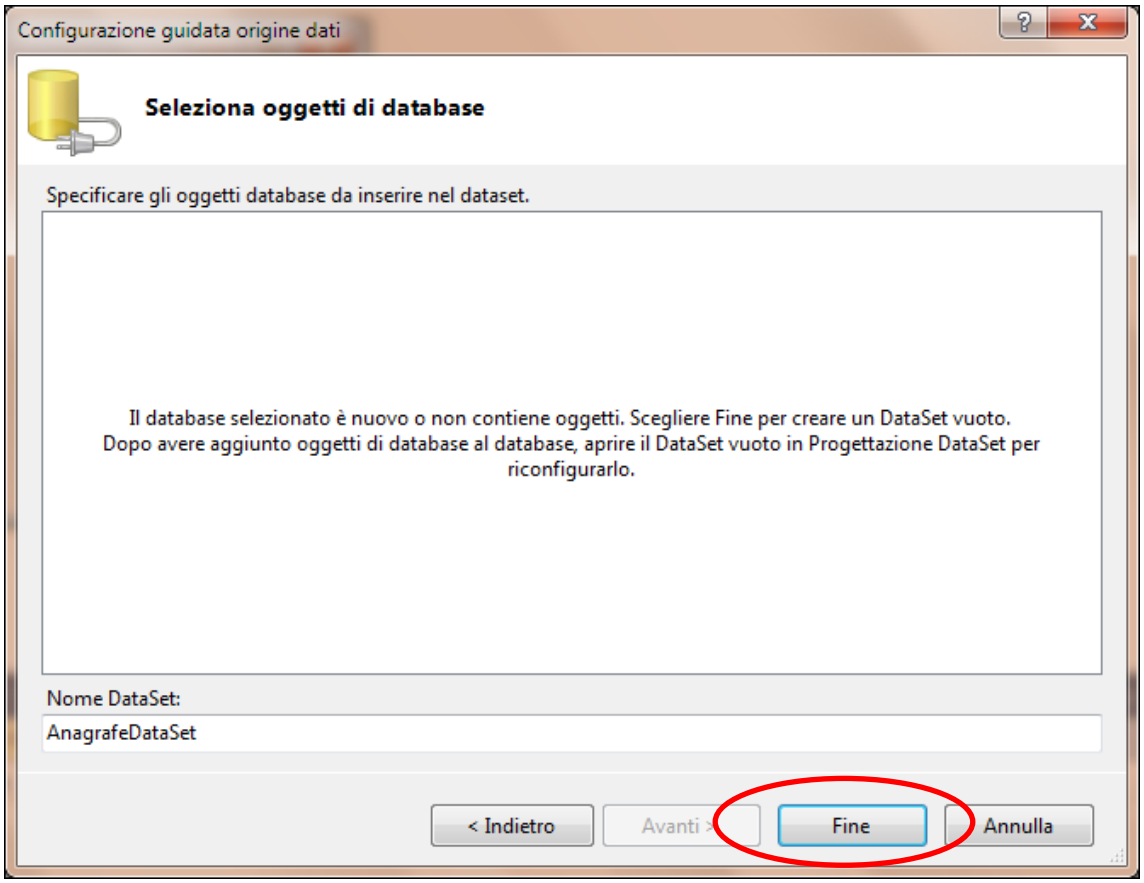


Figura 261: Termine delle operazioni per la creazione del database e del suo collegamento al programma.

194: Creazione della struttura dell'archivio.

Ora nella finestra **Esplora soluzioni** del nostro progetto vediamo un file **Anagrafe.sdf** che segnala l'inclusione nel progetto di un archivio di dati, e abbiamo uno strumento per la connessione tra questo archivio e il programma: si tratta del file **AnagrafeDataSet.xsd**):

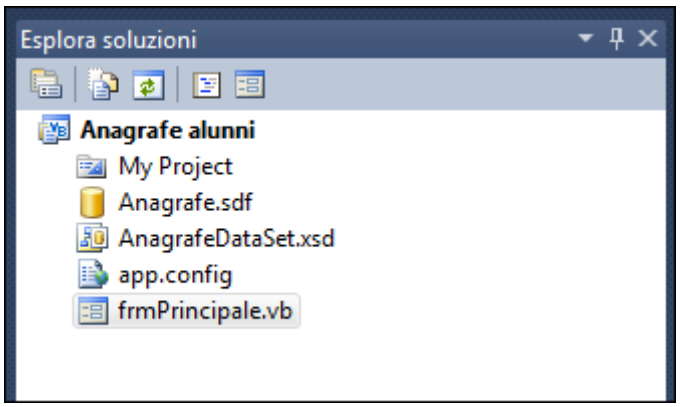


Figura 262: I nuovi file del database anagrafe nella finestra Esplora soluzioni.

Il database **Anagrafe.sdf** per ora è vuoto; non contiene nemmeno una tabella con un elenco di campi in cui i dati possano essere inseriti. E' dunque necessario creare almeno una tabella, con un certo numero di colonne; prima di creare materialmente questa tabella, che costituirà la struttura del nostro archivio, prepariamo uno schema con l'elenco dei campi e del tipo di dati che vi inseriremo.
Ipotizziamo dunque per il nostro database sarà dotato di una tabella che chiameremo **Alunni**, che a sua volta conterrà questi **campi**:

Campi	Contenuto previsto	Tipo di dati previsti in questo campo[89]
Cognome e nome	Una stringa di testo.	nvarchar
Data di nascita	Una data scritta nel formato gg/mm/aaaa.	datetime
Luogo di nascita	Una stringa di testo.	nvarchar
Residenza	Una stringa di testo.	nvarchar
Indirizzo	Una stringa di testo.	nvarchar

[89] Il linguaggio SQL indica i tipi di dati con nomi diversi da quelli utiizzati da VB: al termine di questo capitolo è riportata una tabella alla quale rimandiamo la lettrice o il lettore, per comprendere le caratteristiche dei tipi di dati previsti in questo schema.

Provincia (o stato) di nascita	Una stringa di testo.	nvarchar
Telefono	Una stringa di testo.	nvarchar
Peso (kg)	Un numero intero.	int
Altezza (cm)	Un numero intero.	int

Notiamo, in questo schema, che nel campo **Telefono** non sono previsti dati di tipo numerico: poiché non si prevedono calcoli con i numeri di telefono, questi numeri saranno trattati nel programma come stringhe di testo.
Lo schema con i campi e il tipo di dati che saranno contenuti nella tabella va progettato con attenzione, per evitare integrazioni e o modifiche successive che, per quanto possibili, sono laboriose e rischiose per il funzionamento del programma.
La massima cura deve essere posta nella previsione del tipo dei dati che verranno accettati nei campi della tabella: queste impostazioni devono essere a prova di utente ma, nello stesso tempo, debbono evitare l'uso spropositato di risorse di memoria del computer.
Facciamo un esempio: prevedere dei dati di tipo **int** nei campi **Peso** e **Altezza** è una scelta fatta per facilitarne la comprensione, ma costituisce uno spreco di risorse di memoria, in quanto i dati di tipo **int** possono contenere numeri interi che vanno da –2.147.483.648 a +2.147.483.647; nel nostro caso, sarebbe stato sufficiente dichiarare dati di tipo **tinyint** (numeri interi da 0 a 255).

Passiamo ora alla creazione materiale della prima (e nel nostro caso unica) tabella del nostro archivio di dati.
Nella finestra **Esplora soluzioni**, facciamo un doppio *clic* sul file **Anagrafe.sdf**. Vediamo che nello spazio abitualmente occupato dalla **Casella degli Strumenti**, a sinistra del form, compare ora la nuova finestra: **Esplora database**.

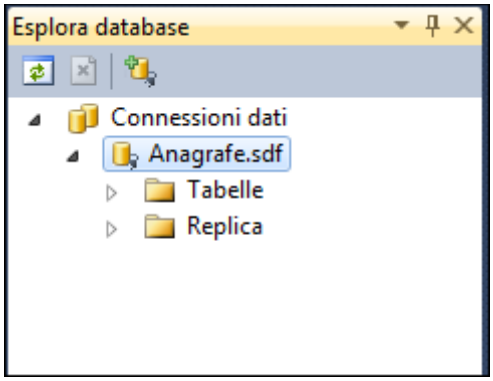


Figura 263: La finestra Esplora database.

Le due finestre **Esplora database** e **Casella degli Strumenti** sono in realtà sovrapposte l'una all'altra. Cliccando in basso a sinistra, infatti, è possibile visualizzare l'una o l'altra:

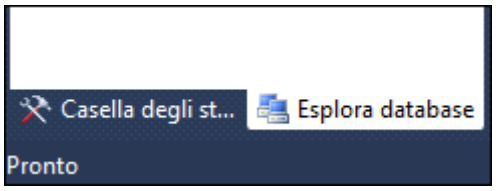


Figura 264: Passaggio dalla Casella degli Strumenti alla finestra Esplora database.

Facciamo un *clic* con **il tasto destro** del mouse sulla cartella **Tabelle** e scegliamo la voce **Crea tabella**:

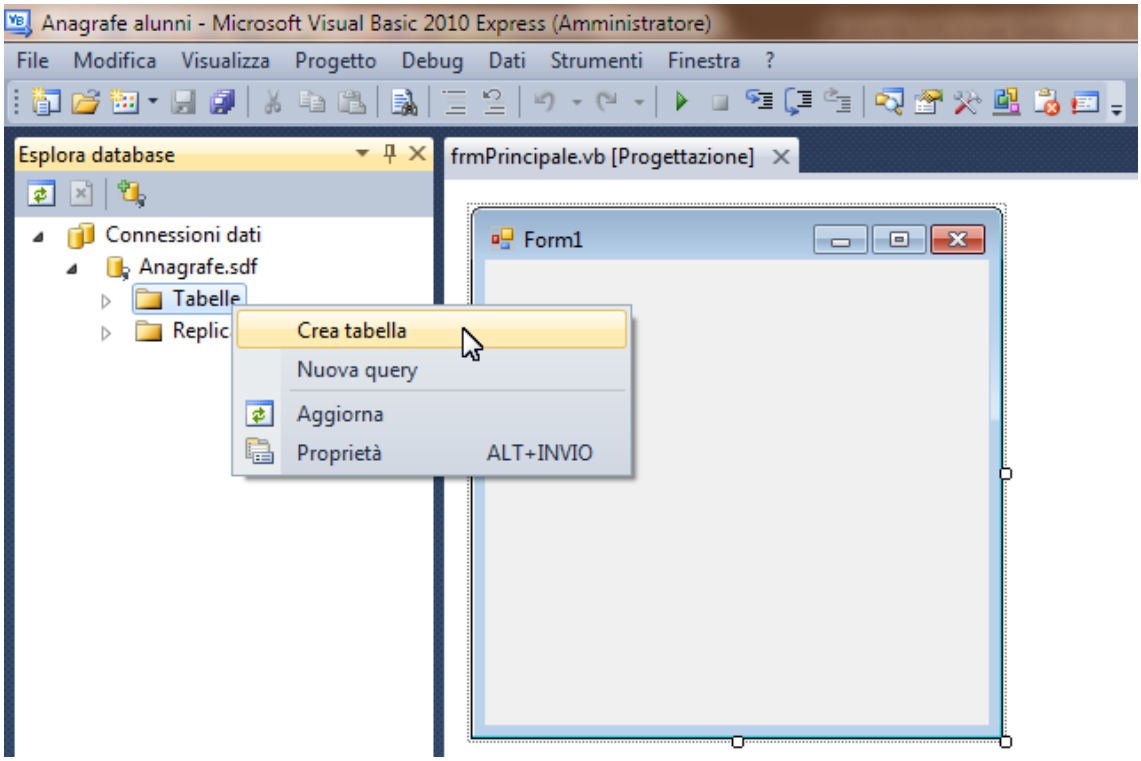


Figura 265: Creazione di una nuova tabella.

La finestra che si apre è destinata a definire l'impostazione e il contenuto della tabella. Le diamo il nome **Alunni** e iniziamo a immettere i nomi dei campi (o delle colonne) con i relativi dati:

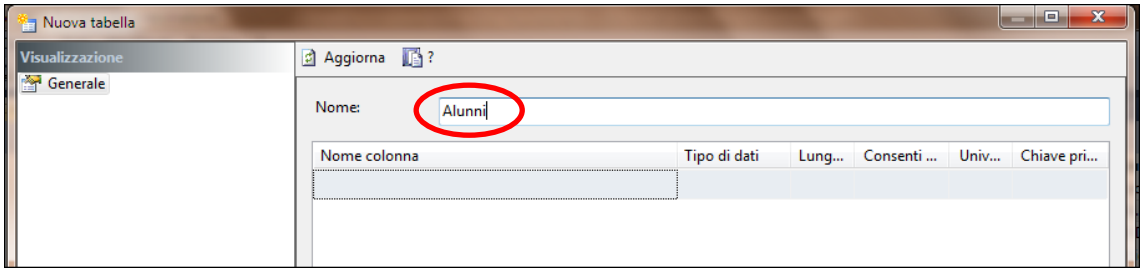


Figura 266: Creazione di una nuova tabella per il database.

Il **primo campo** da immettere… **non è previsto** nello schema che avevamo ipotizzato!
Si tratta di un campo che non dovrebbe mai mancare in ogni tabella di un archivio: esso definisce la **chiave primaria** della tabella con il compito di identificare ciascuna scheda (**record**) in modo inequivocabile, con un numero assegnato automaticamente.
Questo numero progressivo parte da 1 e aumenta di una unità per ogni nuova scheda (nel nostro caso per ogni nuovo alunno). Esso rimane assegnato alla scheda come un cartellino di identificazione e non viene più modificato, anche nella eventualità di cancellazione di alunni. Ad esempio, se in un archivio sono presenti le schede di 20 alunni, l'ultima scheda è identificata con il numero 20. Se viene cancellata la scheda n. 15 e viene poi inserita una nuova scheda, la nuova scheda ha il numero 21, mentre il numero 15 rimane non assegnato.
Impostiamo dunque questo primo campo speciale con questi i dati:

- **Nome colonna = NumeroProgressivo**
- **Tipo di dati = int**
- **Lunghezza = 4**
- **Consenti valori Null = No**
- **Univoca = Sì**
- **Chiave primaria = Sì**

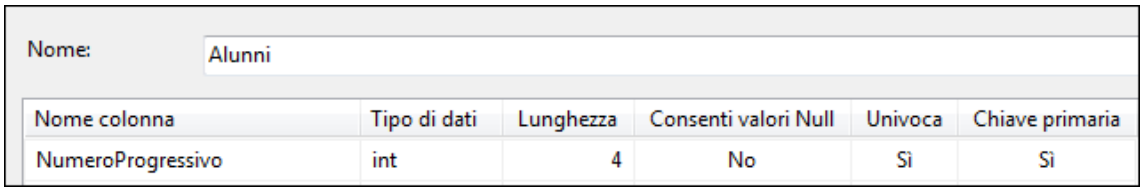


Figura 267: Impostazione del primo campo della tabella Alunni.

Con tali impostazioni, questo campo avrà dunque il nome **NumeroProgressivo**, conterrà dati di tipo **int** (numeri interi) e non ammetterà valori nulli; il dato in esso contenuto sarà univoco, cioè non ammetterà doppioni, e sarà utilizzato come chiave primaria nelle operazioni di organizzazione e ricerca di dati all'interno dell'archivio.

Solo per questo campo, impostiamo alcune altre proprietà, nella parte in basso nella finestra:

- **Identity = True**
- **Incremento Identity = 1**
- **Valore di inizializzazione Identity = 1**

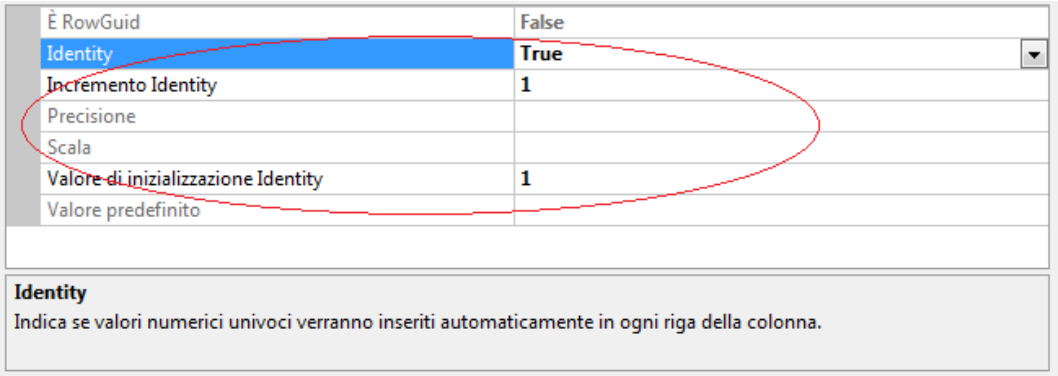
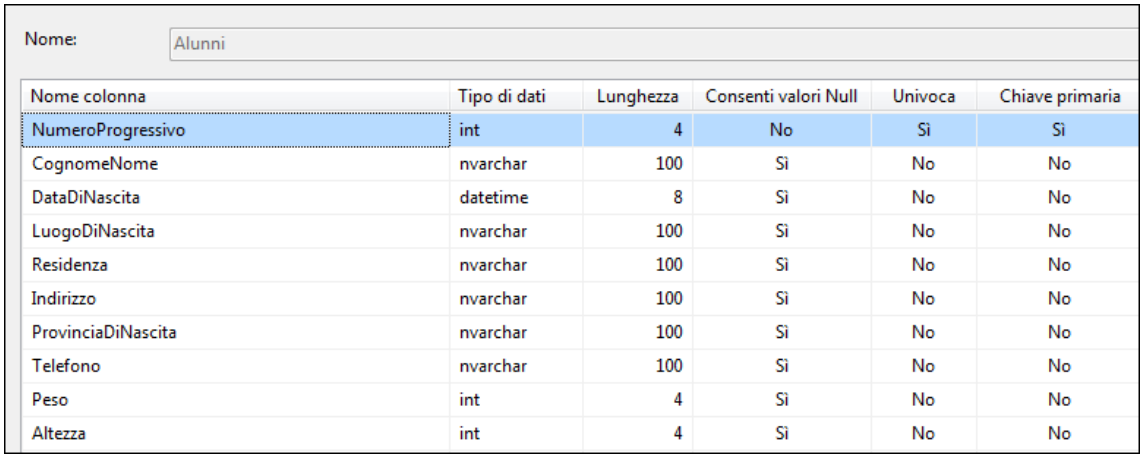


Figura 268: Impostazione del campo con la chiave primaria.

Per effetto di queste impostazioni, i valori numerici contenuti in questo campo saranno inseriti automaticamente dal programma, partendo dal n. 1, con incremento di una unità per ogni scheda immessa nel database (cioè per ogni alunno).
In pratica: quando l'utente immetterà i nomi e i dati degli alunni, il campo **NumeroProgressivo** provvederà a numerare automaticamente ogni alunno, partendo dal n. 1.
Ora completiamo la struttura della tabella **Alunni** riportando fedelmente i valori che compaiono in questa immagine:



Nome colonna	Tipo di dati	Lunghezza	Consenti valori Null	Univoca	Chiave primaria
NumeroProgressivo	int	4	No	Sì	Sì
CognomeNome	nvarchar	100	Sì	No	No
DataDiNascita	datetime	8	Sì	No	No
LuogoDiNascita	nvarchar	100	Sì	No	No
Residenza	nvarchar	100	Sì	No	No
Indirizzo	nvarchar	100	Sì	No	No
ProvinciaDiNascita	nvarchar	100	Sì	No	No
Telefono	nvarchar	100	Sì	No	No
Peso	int	4	Sì	No	No
Altezza	int	4	Sì	No	No

Figura 269: Completamento della struttura della tabella alunni.

Notiamo le differenza del primo campo rispetto a tutti gli altri:

- il primo campo **non consente valori nulli**, tutti gli altri sì;
- il suo contenuto **è univoco** (non consente doppioni), per tutti gli altri non lo è;
- **è la chiave primaria**, cioè il primo elemento che il programma prenderà in considerazione per l'organizzazione dei dati dell'archivio.

Al termine, confermiamo la struttura della tabella con un *clic* su OK.

Torniamo alla finestra **Esplora database**.
Notiamo che essa contiene ora la tabella **Alunni** con i suoi dieci campi. Il campo della chiave primaria è indicato dall'icona con la chiave:

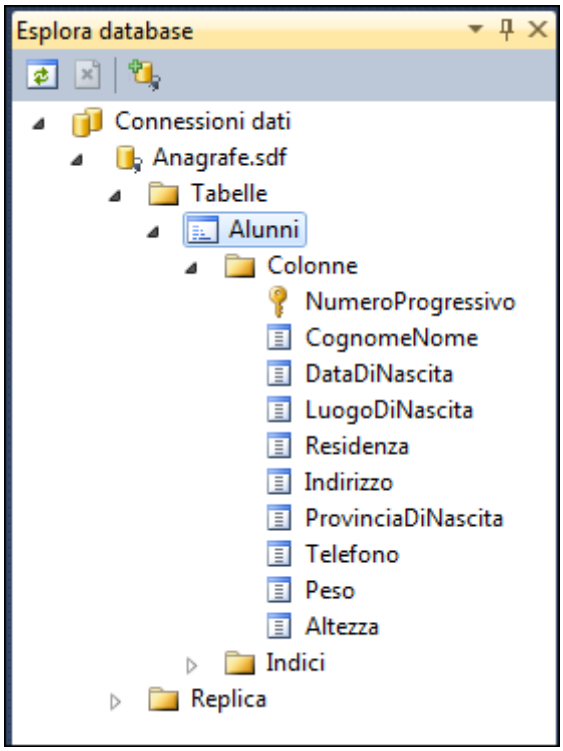


Figura 270: Visualizzazione della struttura della tabella alunni.

195: Creazione del DataSet.

Ora disponiamo di un programma (**Anagrafe alunni**) e di un database (**Anagrafe.sdf**), ma, per quanto inseriti nello stesso progetto, si tratta ancora di entità separate l'una dall'altra.
La prossima operazione consiste nella creazione di uno strumento di collegamento: questo strumento è il **DataSet** (= sistemazione di dati), che si colloca in una posizione intermedia tra il programma e l'archivio di dati.
Possiamo immaginare il **Set** di **Dati** come un set cinematografico: mentre là si mettono in scena le azioni che si girano per un film, qui si impostano le azioni da effettuare nel database.
Il **DataSet** dunque conterrà le istruzioni per le operazioni di interazione, previste e volute dal programmatore, tra il programma e il database: inserimento di dati, cancellazione, modifica di dati, interrogazione del database per estrapolarne le schede che rispondano a determinati criteri.
Nel nostro caso il **DataSet** esiste già, è stato creato in modo automatico ed è ancora vuoto: si tratta del file **AnagrafeDataset.xsd**[90]. Lo troviamo nella finestra **Esplora Soluzioni**, a destra del form:

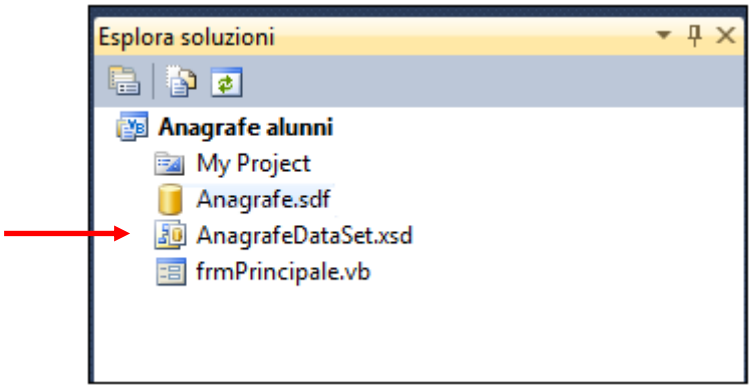


Figura 271: Il file del dataset nella finestra Esplora soluzioni.

[90] L'estensione .xsd sta per **XML Schema Definition**; essa indica che il file contiene istruzioni per selezionare gli elementi di un documento di tipo .xml (**Extensible Markup Language**). Le istruzioni nel file .xsd verificano se ogni voce contenuta nel documento .xml risponde o meno a determinate condizioni.

Con un doppio *clic* su questo file, accediamo alla finestra di impostazione del **DataSet**: La finestra che si apre è vuota, ma un avviso indica come si deve procedere per inserirvi nuovi elementi: facciamo un *clic* con il tasto destro del mouse.

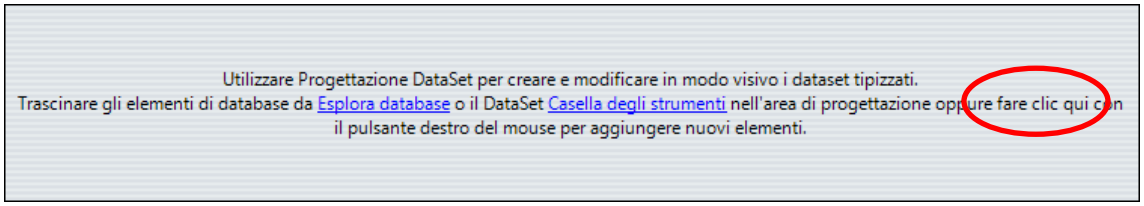


Figura 272: Inizio dell'inserimento di nuovi elementi nel dataset.

Nella finestra **Esplora database**, a sinistra del form, apriamo lo schema ad albero relativo al database **Anagrafe.sdf**, fino a visualizzare la tabella **Alunni**: questa tabella, trascinata con il tasto sinistro del mouse nella finestra del **DataSet**, ne costituirà il primo e unico elemento:

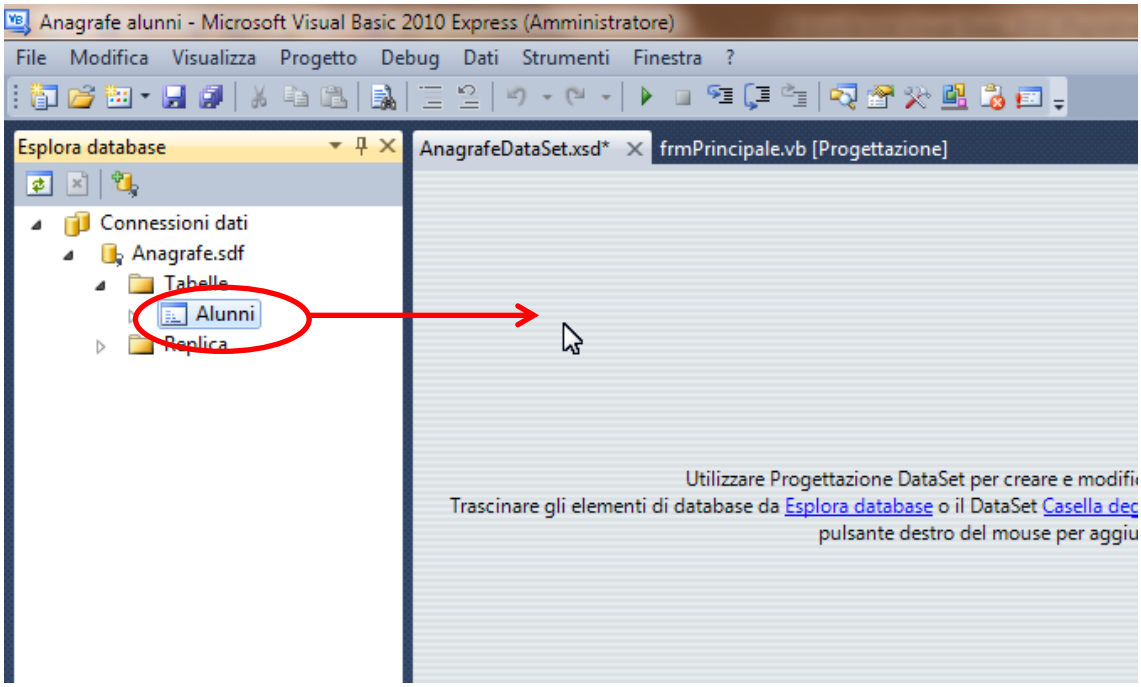


Figura 273: Trascinamento di una tabella nel dataset.

Chiudiamo la finestra del **DataSet** e facciamo un *clic* sul pulsante Sì per salvarne il contenuto:

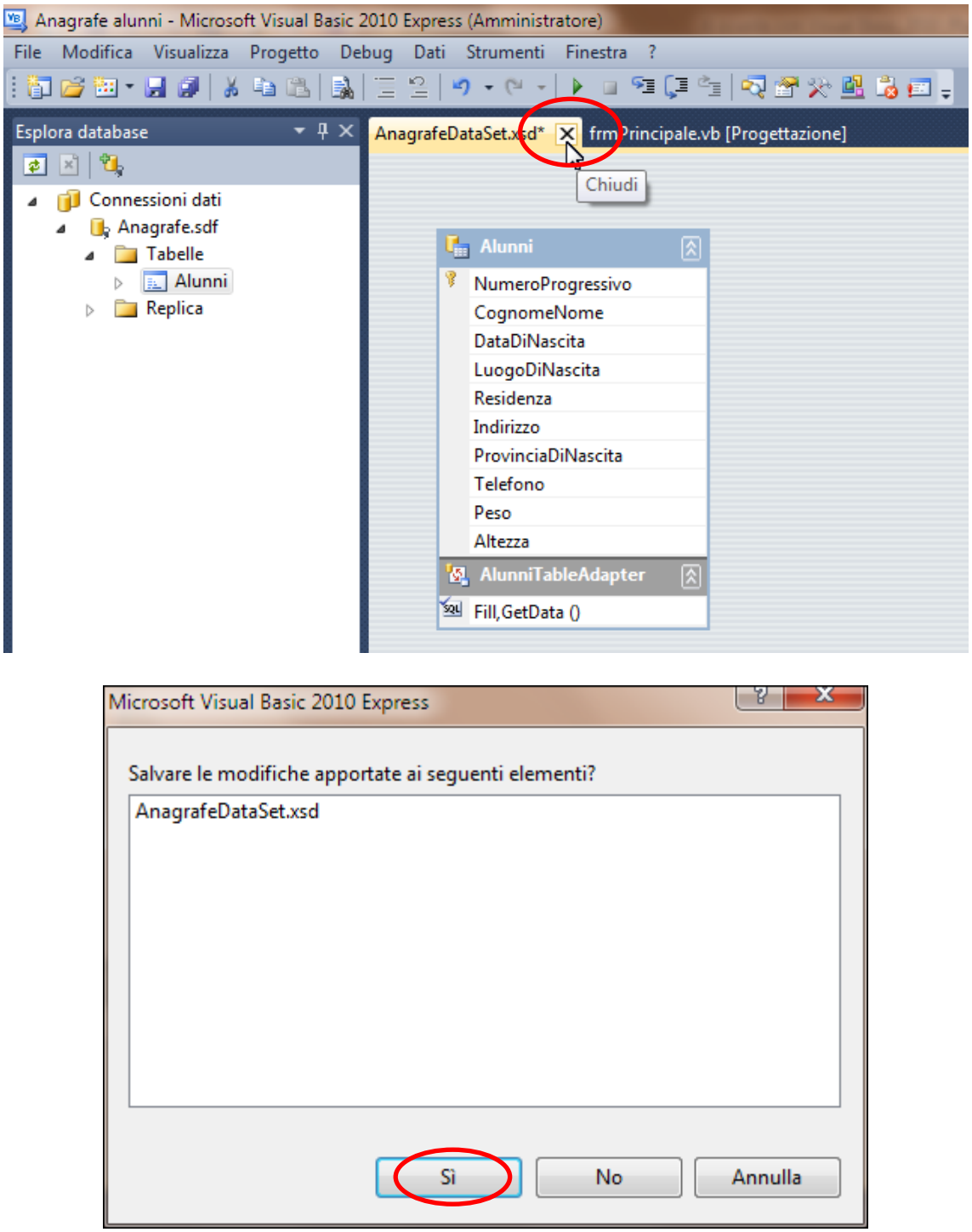


Figura 274: Salvataggio del contenuto del DataSet.

196: Le query per la gestione dei dati.

Dopo avere inserito la tabella **Alunni** nel **DataSet**, dobbiamo ora inserirvi le istruzioni di interazione secondo noi necessarie tra il database e il programma: inserimento, cancellazione e modifica di dati, selezione secondo determinati criteri.
Queste istruzioni saranno scritte in linguaggio SQL, in espressioni tutto sommato di facile comprensione, che si chiamano ***query*** (= ricerca).
Le *query* contengono comandi e/o operatori scritti secondo le regole grammaticali e sintattiche del linguaggio **SQL**; esse agiscono da intermediari tra un programma e un database e sono indispensabili per gestire le informazioni contenute nelle tabelle di un archivio.
Nel nostro caso creeremo *query* per aggiungere (**insert**) schede (**record**) o per cancellarle (**delete**); *query* per aggiornare e/o modificare (**update**) le informazioni contenute in determinati campi.
Altre *query*, infine, saranno create per selezionare (**select**) gli alunni in base a determinati criteri.
Le *query* si impostano nella **finestra del DataSet** e sono memorizzate nel **TableAdapter** di ciascuna tabella.

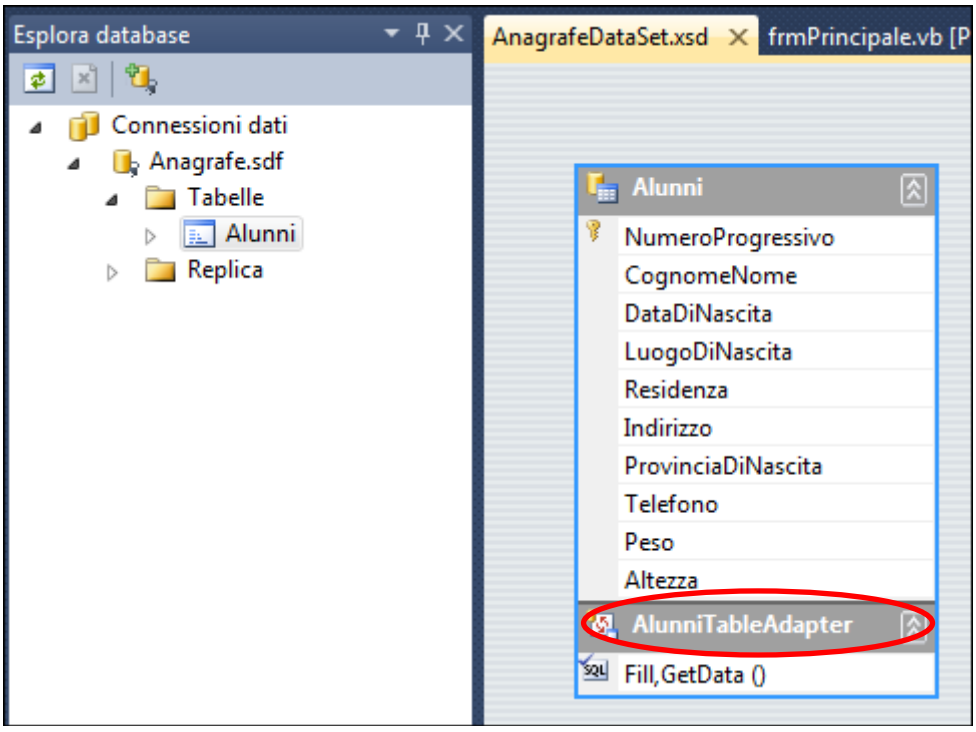


Figura 275: Il Table Adapter della tabella Alunni.

L'oggetto **AlunniTableAdapter** è dunque lo strumento preposto alla comunicazione con la tabella **Alunni**, del database **Anagrafe.sdf**.
Un oggetto **TableAdapter** effettua queste operazioni:

- invia i dati modificati o aggiornati dall'utente del programma al database;
- esegue le *query* in esso contenute e restituisce le informazioni ottenute, organizzate in nuove tabelle.

Per creare le *query* che ci serviranno per la gestione dei dati dell'anagrafe degli alunni, facciamo un doppio clic, nella finestra **Esplora soluzioni,** sul file **AnagrafeDataset.xsd** e torniamo ad aprire la finestra del **DataSet**: qui facciamo un *clic* con il tasto destro del mouse sull'oggetto **AlunniTableAdapter**, poi un *clic* con il tasto sinistro sul menu **Aggiungi / Query**:

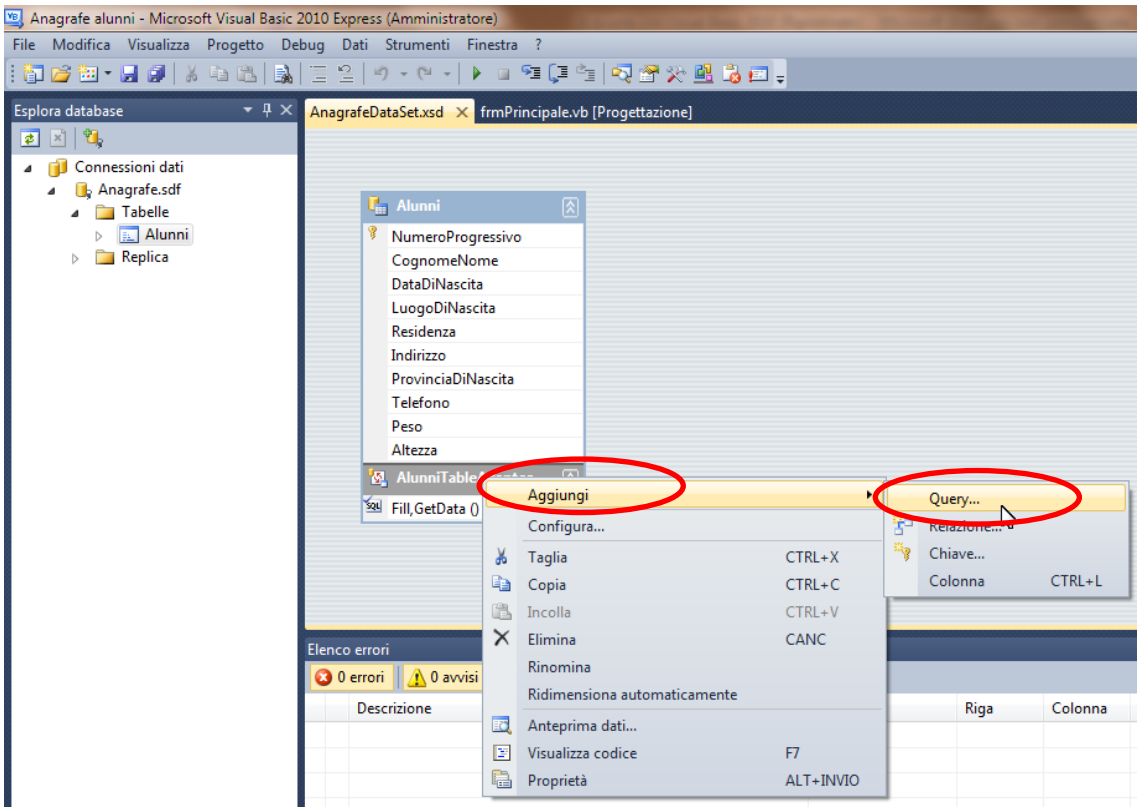


Figura 276: Aggiunta di una nuova *query* nel TableAdapter.

Nel nostro **TableAdapter** aggiungeremo:

- una *query* per l'inserimento di nuove schede (nuovi alunni);
- una *query* per l'eliminazione di schede;
- quattro diverse *query* per la selezione di gruppi di alunni, secondo nome, provincia di nascita, altezza, peso.

Attenzione: la creazione di nuove *query* avviene con procedimenti guidati e in buona parte automatici, ma finchè non si conclude il procedimento di creazione di tutte le *query* è opportuno non chiudere la finestra del **DataSet**.

197: Creazione di una query per l'inserimento di dati.

Cominciamo ad aggiungere al **TableAdapter** la *query* per inserire nuovi alunni nella tabella A**lunni**.
Nella prima finestra accettiamo di scrivere ex novo le istruzioni SQL e proseguiamo con un *clic* sul pulsante **Avanti**:

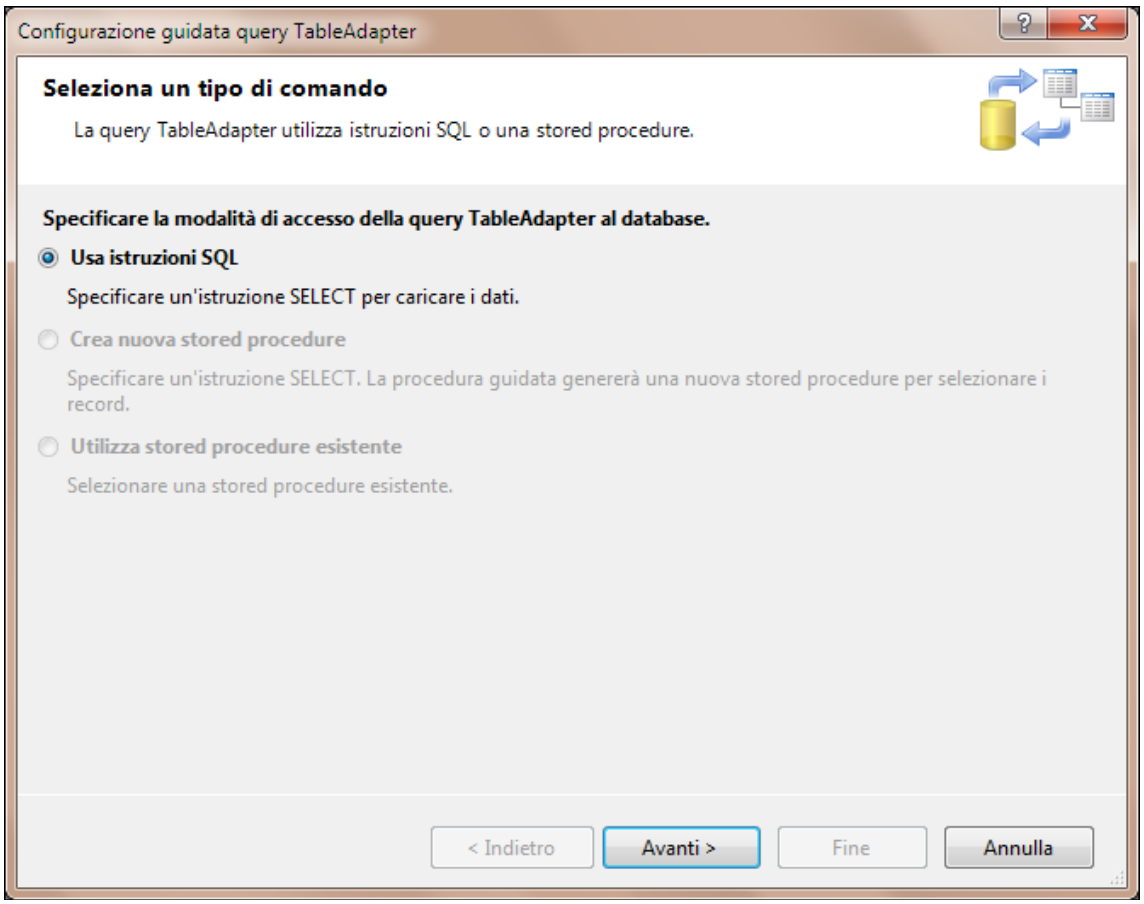


Figura 277: Avvio della creazione di una nuova *query*.

Per creare la **query** di inserimento di nuovi dati facciamo un *clic* sulla opzione **INSERT** e continuiamo con un *clic* su **Avanti**:

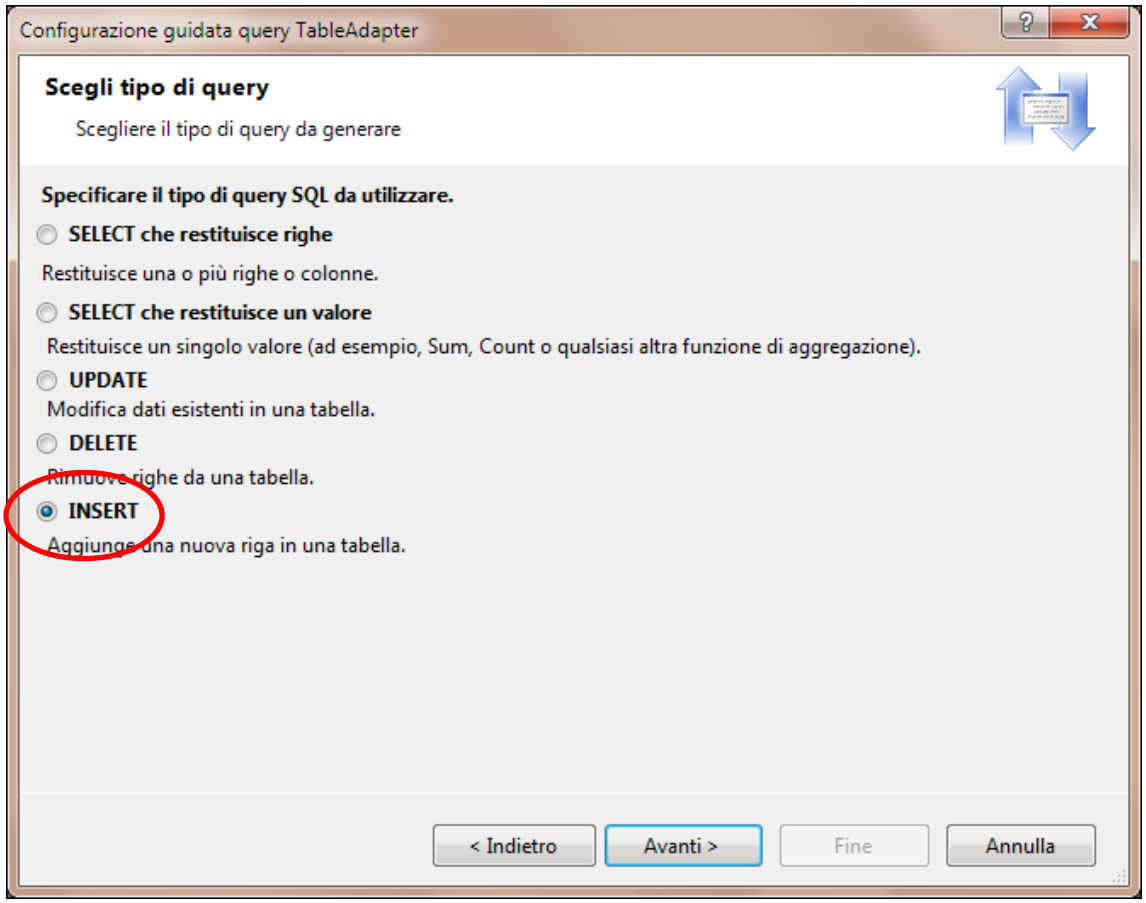


Figura 278: Avvio della creazione di una query di tipo INSERT.

Nella scheda successiva vediamo l'istruzione **SQL INSERT** già impostata:

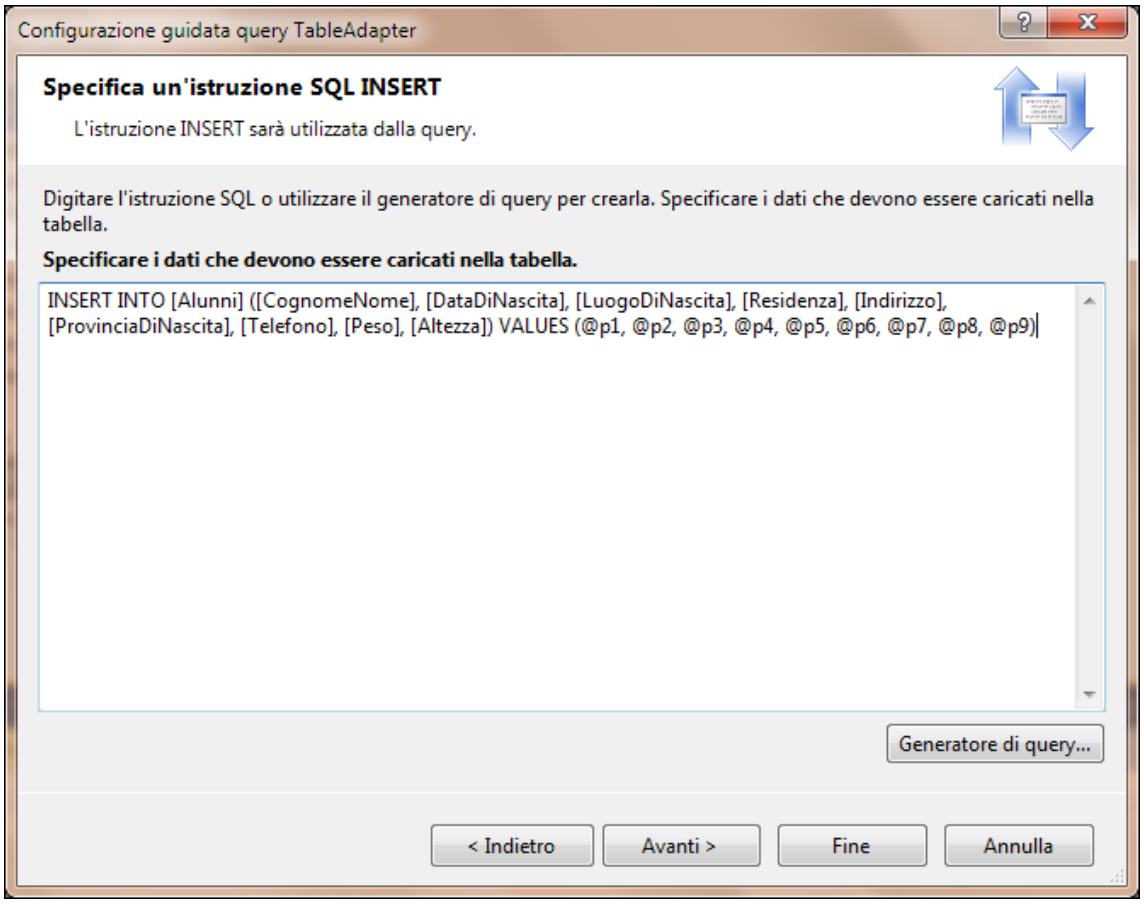


Figura 279: L'istruzione SQL INSERT.

Vediamo di comprenderne il contenuto.
L'istruzione, scritta in linguaggio SQL, è divisa in tre parti:

- la prima parte definisce l'azione da compiere: Inserisci nella tabella alunni;
- la seconda parte elenca i campi della tabella alunni che dovranno ricevere dei dati, quando la *query* sarà in esecuzione; notiamo che in questo elenco manca il primo campo, **NumeroProgressivo**, che come sappiamo è un campo speciale a numerazione progressiva automatica e non è a disposizione dell'utente del programma;
- la terza parte elenca i valori attesi per ciascuno dei campi della tabella.

Non è possibile sapere in anticipo, ovviamente, quale data specifica, quale indirizzo preciso o quale peso saranno immessi dall'utente nei singoli campi; l'istruzione SQL usa perciò dei segnaposto (@p1, @p2, ...) che al momento dell'esecuzione della *query*, dovranno essere sostituiti con i reali valori scritti dall'utente del programma.

Facciamo *clic* sul pulsante **Avanti** per arrivare alla conclusione della impostazione della *query*, ma prima diamo a questa *query* un nome facilmente comprensibile che non potrà essere confuso con altre *query* che svolgeranno compiti diversi: **InserimentoDati**.

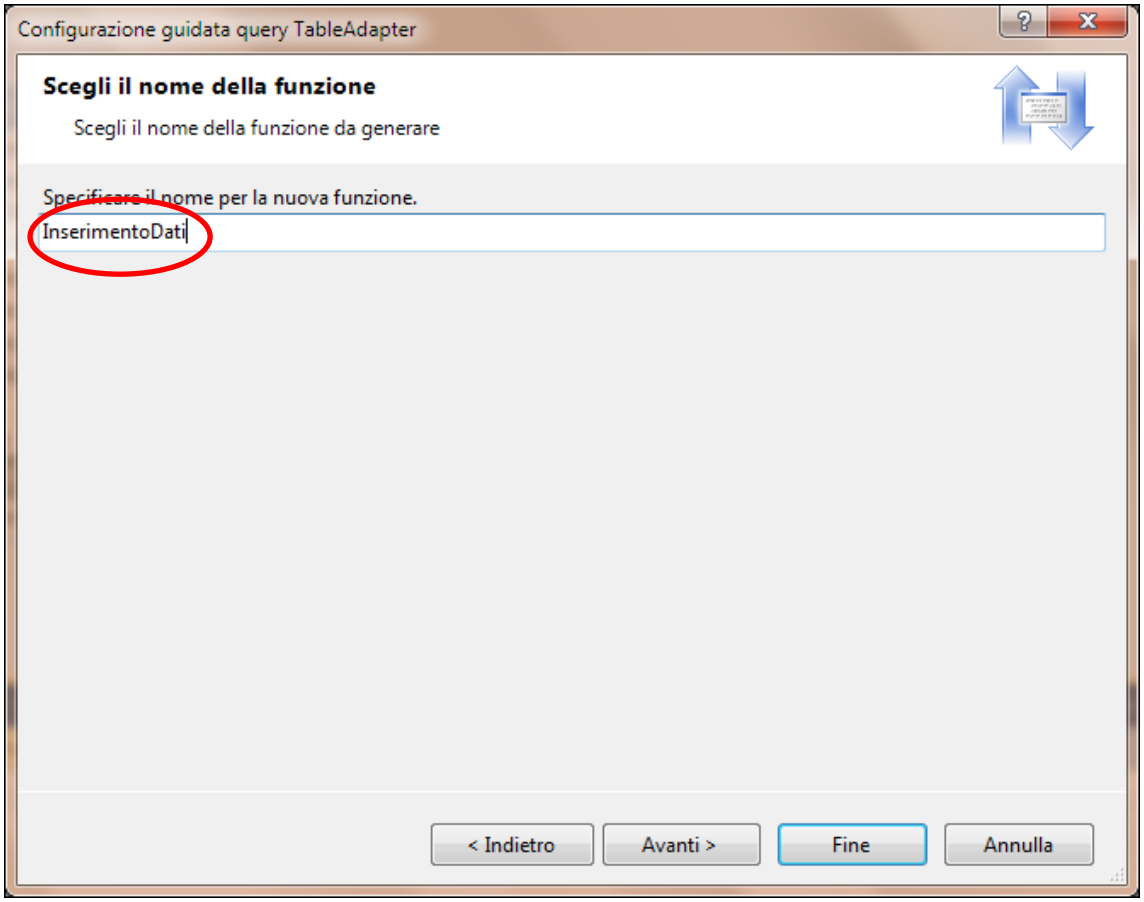


Figura 280: Scelta del nome della *query* INSERT.

Chiudiamo la finestra di generazione della *query* **InserimentoDati** con un *clic* sul pulsante **Fine**:

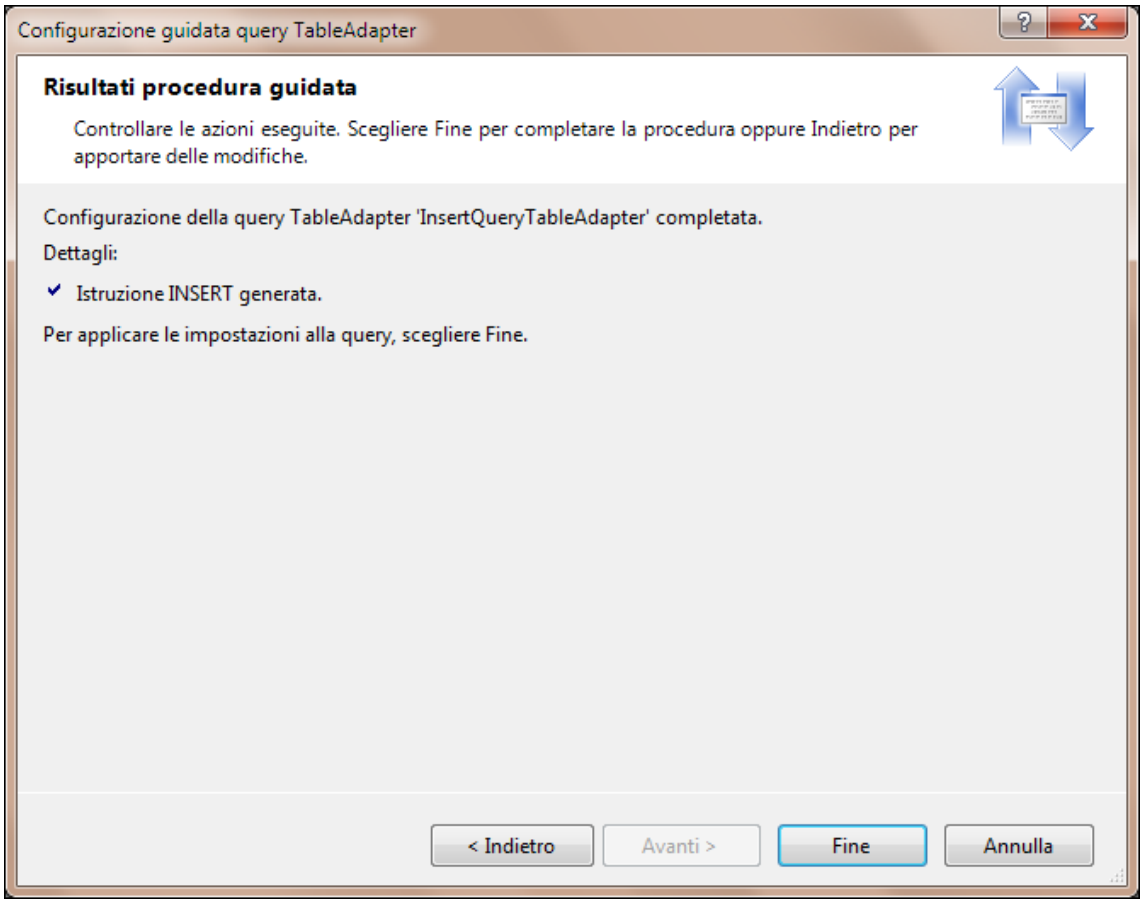


Figura 281: Conclusione della creazione della *query* INSERT.

Ora possiamo verificare, nella finestra del **DataSet**, che il **TableAdapter** contiene la prima *query*:

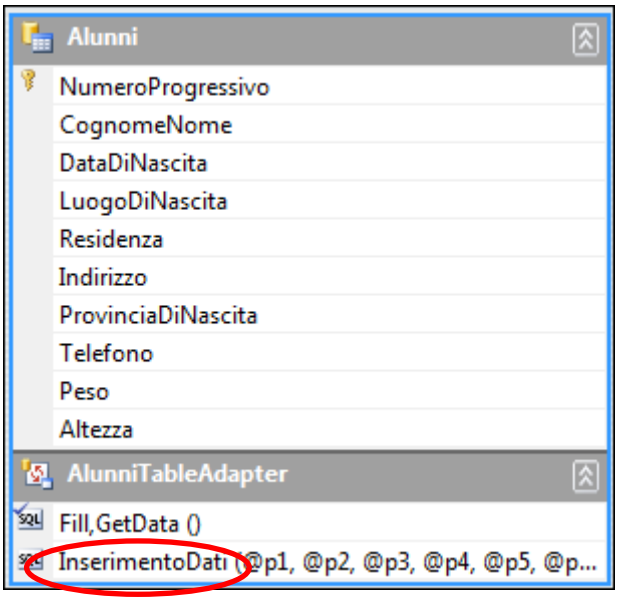


Figura 282: La query InserimentoDati nel TableAdapter.

198: Creazione di una query per l'eliminazione di dati

Passiamo ora a definire una *query* da utilizzare quando l'utente dovrà cancellare un alunno dall'elenco della classe.
Ripetiamo le operazioni di avvio della creazione di una nuova *query* con un *clic* del mouse (pulsante destro) su **AlunniTableAdapter**.

Nella scheda con la scelta del tipo di **query**, ora dobbiamo creare una *query* di cancellazione di dati, per cui facciamo un *clic* su **DELETE** (cancella) e continuiamo con un *clic* su Avanti:

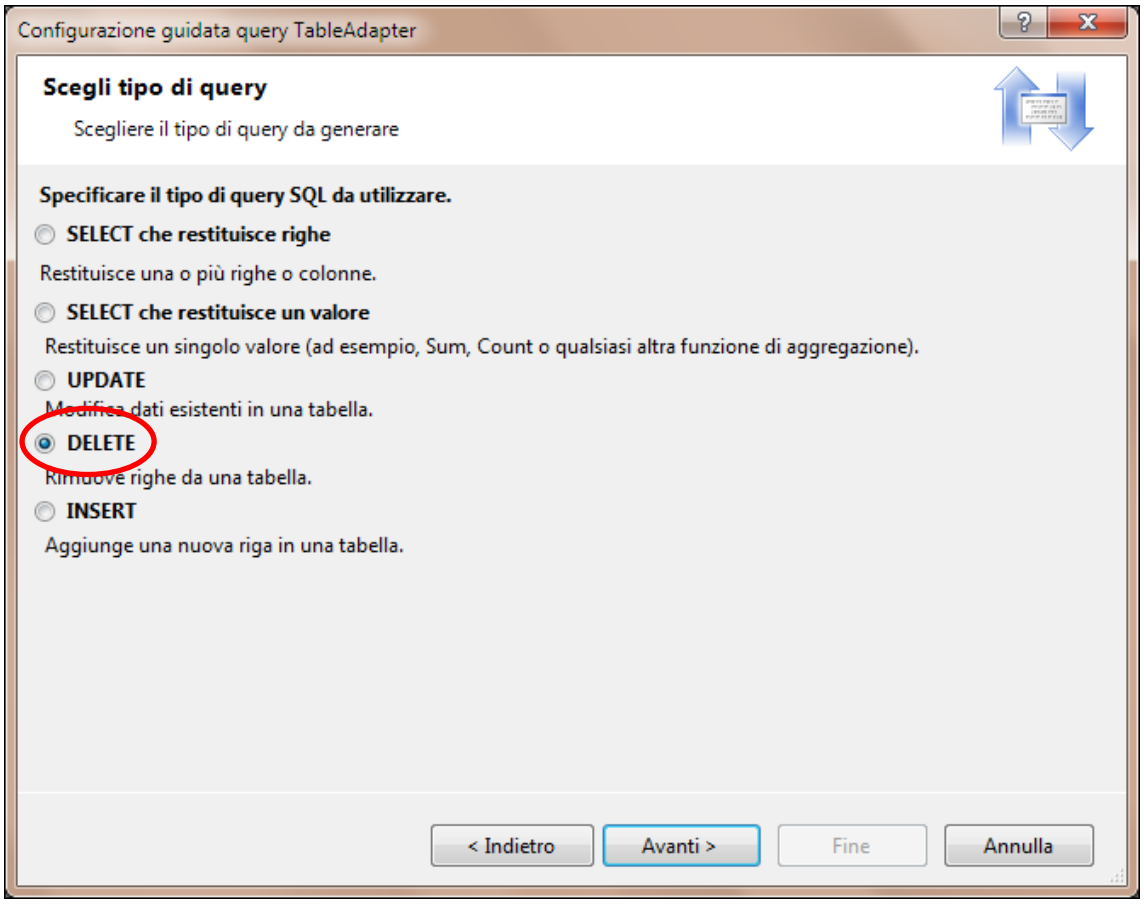


Figura 283: Avvio della creazione di una *query* di tipo DELETE.

Nella scheda successiva vediamo l'istruzione **SQL DELETE** già impostata:

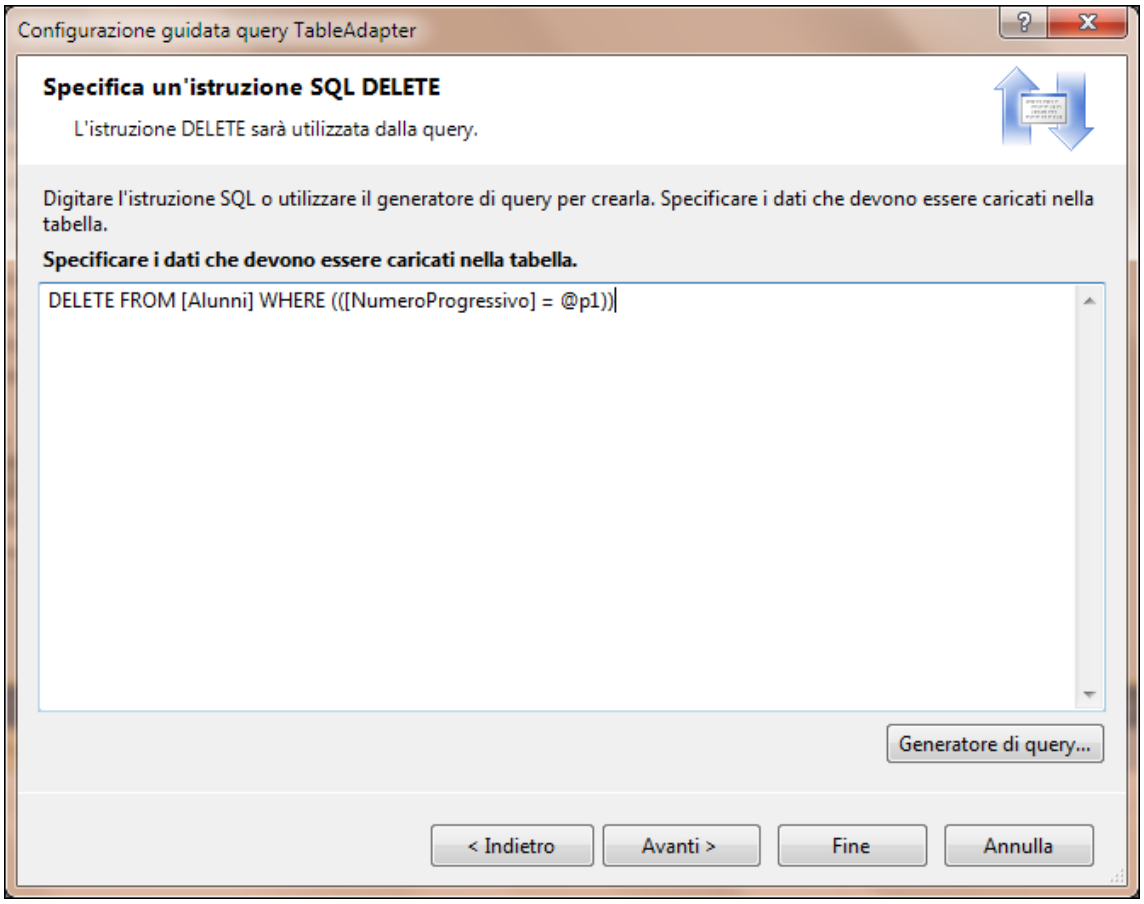


Figura 284: L'istruzione SQL DELETE.

Questa istruzione SQL è divisa in due parti:

- la prima parte definisce l'azione da compiere: **Cancella** dalla tabella **Alunni**;
- la seconda parte stabilisce la condizione necessaria per attuare l'eliminazione della scheda: cancella la riga in cui il valore del campo NumeroProgressivo è uguale a @p1. Anche in questa istruzione il parametro @p1 indica un segnaposto che verrà riempito quando l'utente indicherà in maniera univoca l'alunno in uscita dalla classe.

Confermiamo il testo proposto e facciamo *clic* su **Avanti**.

Arriviamo così alla conclusione della definizione di questa *query*, alla quale diamo il nome **CancellazioneDati**:

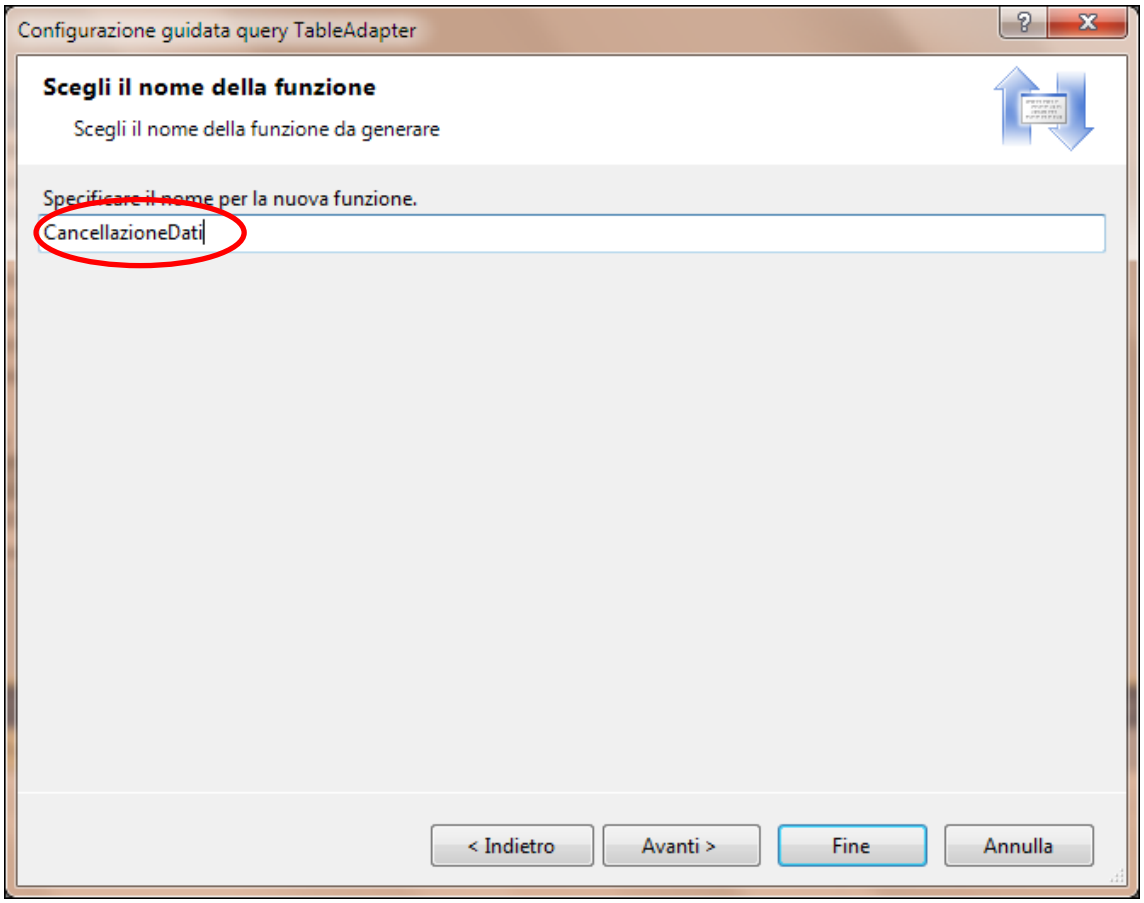


Figura 285: Conclusione della creazione della query CancellazioneDati.

In una tabella possono essere necessarie *query* diverse destinate all'eliminazione delle informazioni. Trattandosi di un elenco di alunni, ad esempio, può essere utile eliminare schede in base alla data di nascita, o alla città di residenza, ecc. Possono essere create, dunque, più *query* di eliminazione con criteri di selezione differenziati.
Nel nostro programma l'utente potrà eliminare schede solo in base al loro numero progressivo univoco, perciò ci limitiamo a questa unica *query* di tipo DELETE.

Verifichiamo, nella finestra del **DataSet**, che il **TableAdapter** contiene ora due *query*:

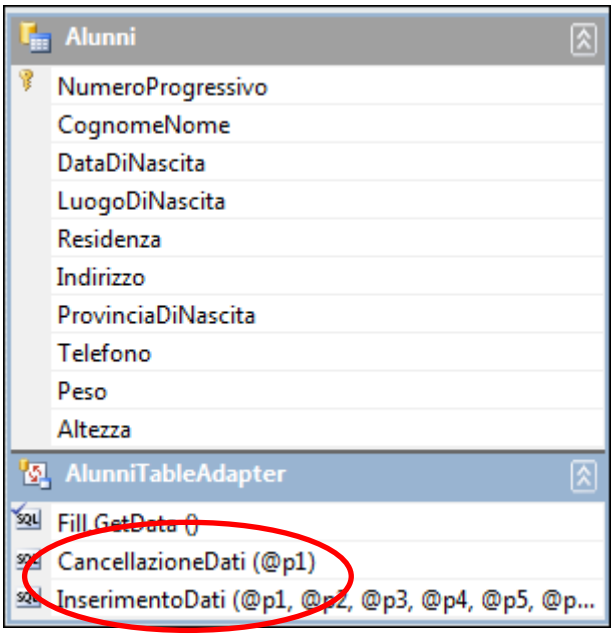


Figura 286: Le *query* CancellazioneDati e InserimentoDati nel TableAdapter.

199: Creazione di query per la selezione di dati.

Dopo avere visto le *query* per inserire o cancellare dati nel database, vediamo come creare delle *query* per l'interrogazione del database, cioè per selezionare, all'interno dell'archivio, quei dati che rispondono a determinati criteri di ricerca.
Ripetiamo le operazioni iniziali per la creazione di una nuova *query*; in questo caso dobbiamo creare una *query* di selezione, per cui facciamo un *clic* sull'opzione **SELECT** e sul pulsante **Avanti**:

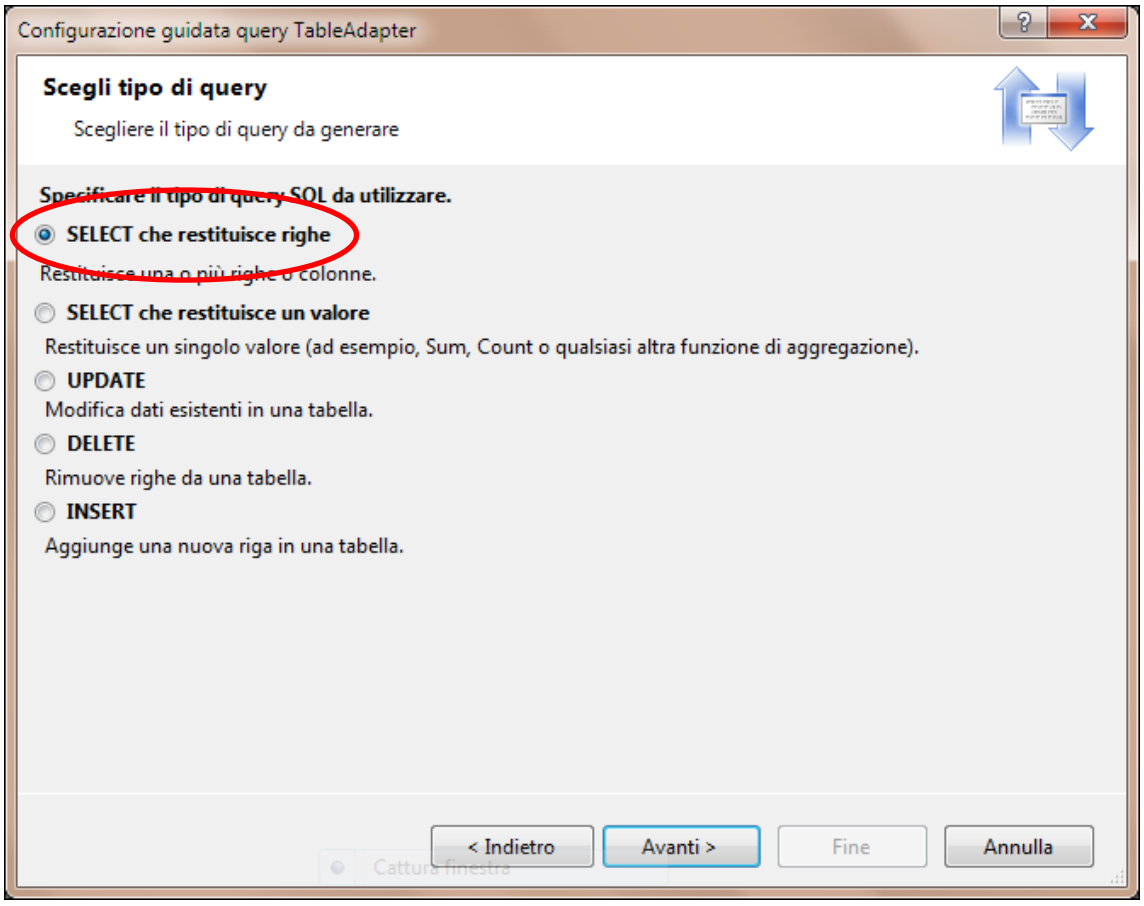


Figura 287: Avvio della creazione di una *query* di tipo SELECT.

Nella scheda successiva vediamo l'istruzione **SQL SELECT** reimpostata, ma in questo caso **incompleta**: questa procedura guidata non può conoscere i criteri di selezione dei dati voluti dal programmatore, per cui l'istruzione è scritta solo a metà.
Spetta al programmatore il compito di completarla, secondo le sue intenzioni, rispettando il lessico e la sintassi del linguaggio SQL.

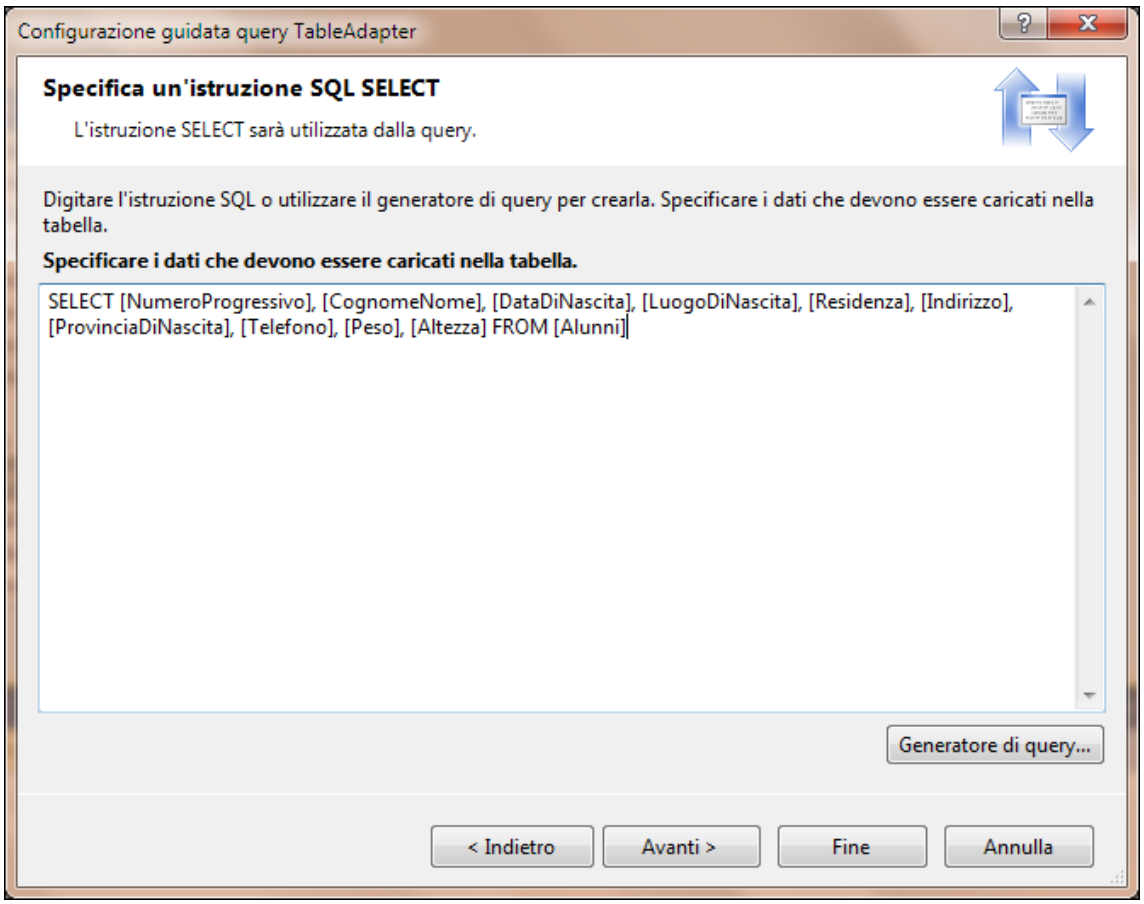


Figura 288: Preimpostazione dell'istruzione SQL SELECT.

Vogliamo completare l'istruzione in modo che questa query selezioni, nella tabella, tutti gli alunni nati in una determinata provincia.
In linguaggio corrente questa istruzione suona più o meno così:

- **SCEGLI**, nella tabella A**lunni**, tutte le schede
- **IN CUI** il campo **ProvinciaDiNascita** è **uguale** a quello immesso dall'utente del programma;
- e **ORDINA** le schede scelte secondo l'ordine alfabetico del campo **CognomeNome**.

Traduciamo questi comandi in linguaggio SQL: aggiungiamo all'istruzione queste due righe (è possibile copiarle e incollarle nella finestra dell'istruzione):
WHERE (ProvinciaDiNascita = @param1)
ORDER BY CognomeNome

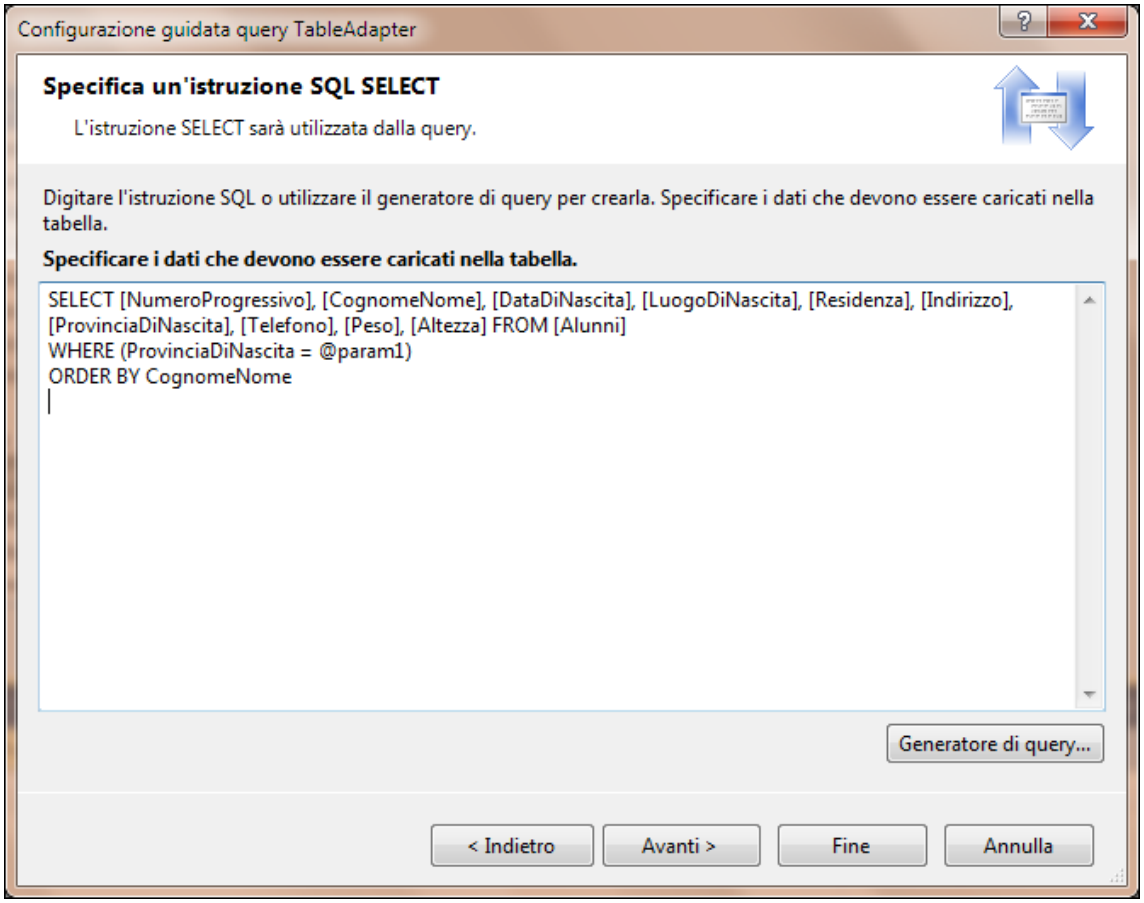


Figura 289: Completamente dell'istruzione SELECT.

Nella prima riga
WHERE (ProvinciaDiNascita = @param1)
si dichiara che interessano le schede in cui la provincia di nascita è uguale al dato indicato dall'utente del programma.
In questa riga, l'operatore **=** potrebbe essere sostituito dall'espressione **<>**, con risultati ovviamente opposti: con **<>** sono selezionate le schede in cui il campo ProvinciaDiNascita è diverso dal dato scritto dall'utente del programma.
Potremmo anche restringere il campo della ricerca aggiungendo, nella stessa riga che contiene WHERE, un secondo criterio:
WHERE (ProvinciaDiNascita = @param1) AND (Peso > @Param2)
e, volendo, un altro criterio ancora:
WHERE (ProvinciaDiNascita = @param1) AND (Peso > @Param2) AND (Altezza < @Param3)
con il risultato di avere selezioni sempre più ristrette.
Al posto dell'operatore logico **AND** si può scrivere **OR**: in questo caso saranno selezionate tutte le schede che rispondono al criterio **x oppure** al criterio **y**, con il risultato di ampliare il numero delle schede selezionate.
L'ultima riga nell'istruzione:
ORDER BY CognomeNome

indica che le schede selezionate dovranno essere elencate in ordine alfabetico secondo il campo **CognomeNome**.

Facciamo un *clic* sul pulsante **Avanti** e diamo un **nome** a questo metodo di selezione dei dati che abbiamo appena creato.
Vediamo la riga **Nome metodo** già impostata con il nome **FillBy** (= riempi la selezione secondo…);
Siccome creeremo altri metodi di selezione dei dati nella tabella **Alunni**, a questo primo metodo diamo un nome preciso e inequivocabile:
FillByProvinciaDiNascita (= riempi la selezione secondo la provincia di nascita)**.**
Facciamo *clic* sul pulsante Fine per completarlo e inserirlo nel **DataSet**.

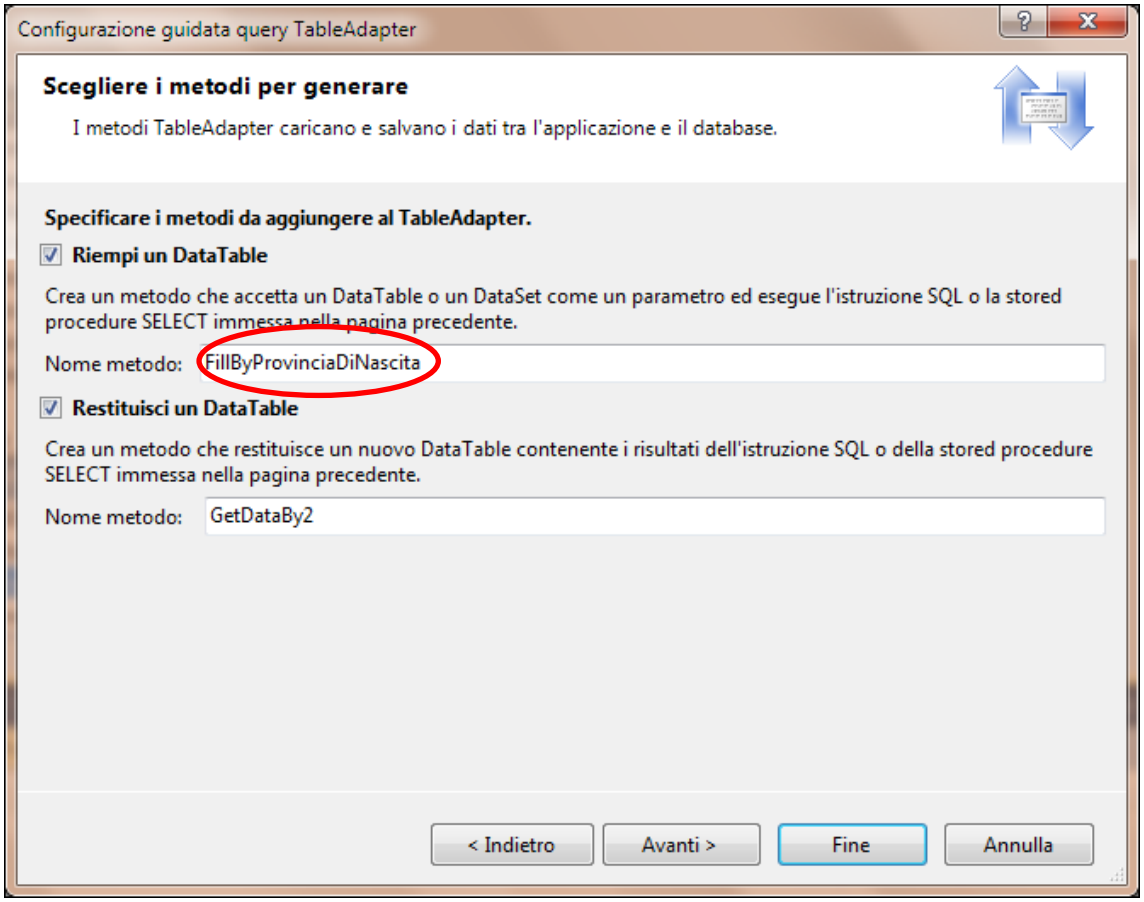


Figura 290: Completamento di una *query* di tipo SELECT.

Ora imposteremo un'altra *query* di tipo **SELECT**, con un altro metodo di selezione dei dati: in questo caso, vogliamo che nell'archivio dei dati siano selezionate tutte le schede che contengono in un determinato campo **una sequenza di lettere**.
Ad esempio, possiamo cercare tutti i riscontri delle parole *via* o *piazza* nel campo che contiene gli indirizzi, oppure possiamo cercare tutti i riscontri della sequenza *ros* nel campo che contiene cognomi e nomi.

Ripetiamo le fasi iniziali della creazione di una *query* di tipo **SELECT** e, giunti nella finestra delle istruzioni per il nuovo metodo di selezione dei dati, completiamo l'istruzione SELECT con queste due righe:
WHERE (CognomeNome LIKE @Param1)
ORDER BY Altezza

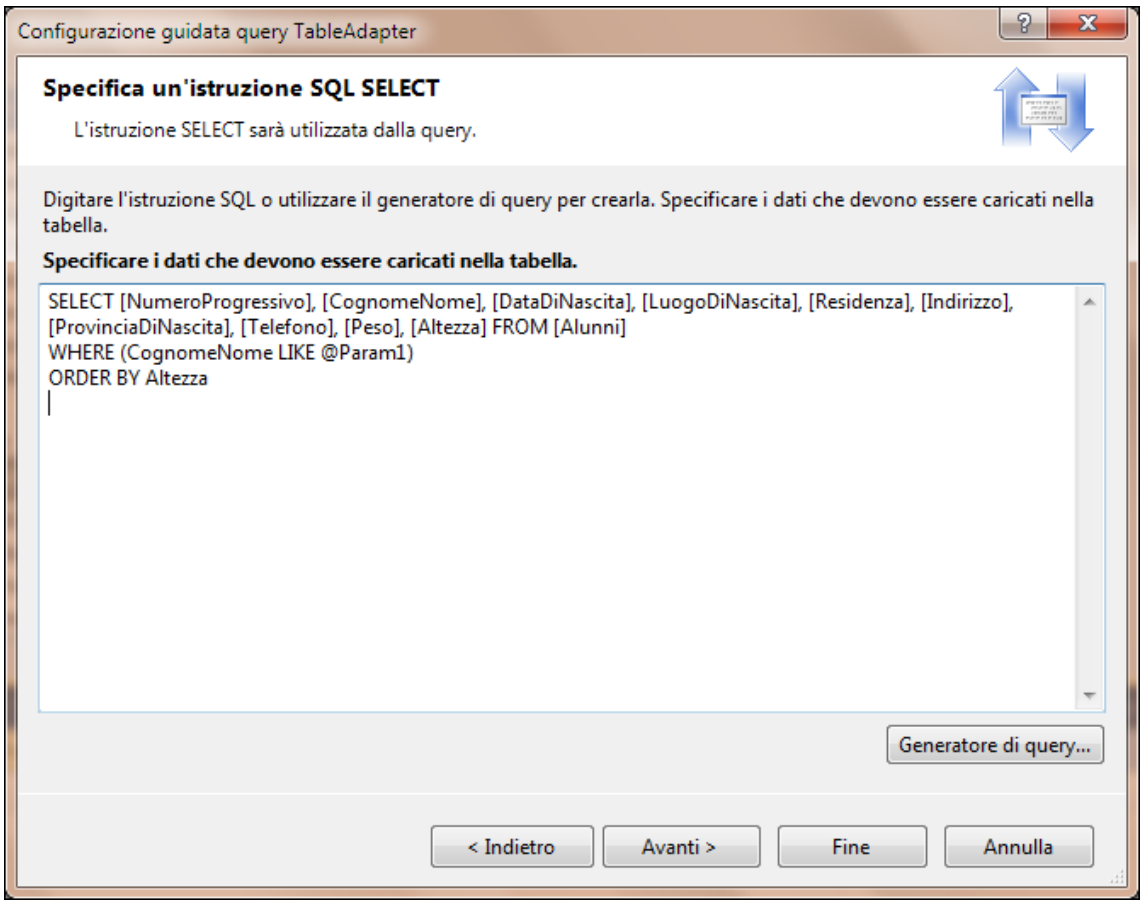


Figura 291: Creazione di una nuova *qyery* di tipo SELECT.

Notiamo che nella riga **WHERE** troviamo l'operatore **LIKE** e non il simbolo =.
Questo operatore consente di effettuare ricerche anche sulla base di informazioni parziali, per selezionare tutte le schede in cui il parametro di ricerca immesso dall'utente è **contenuto** in un determinato campo.

Confermiamo il testo di questo metodo di ricerca e facciamo *clic* su **Avanti**.

Prima di concludere la creazione di questo metodo di ricerca, gli diamo il nome **FillByCognomeNome**:

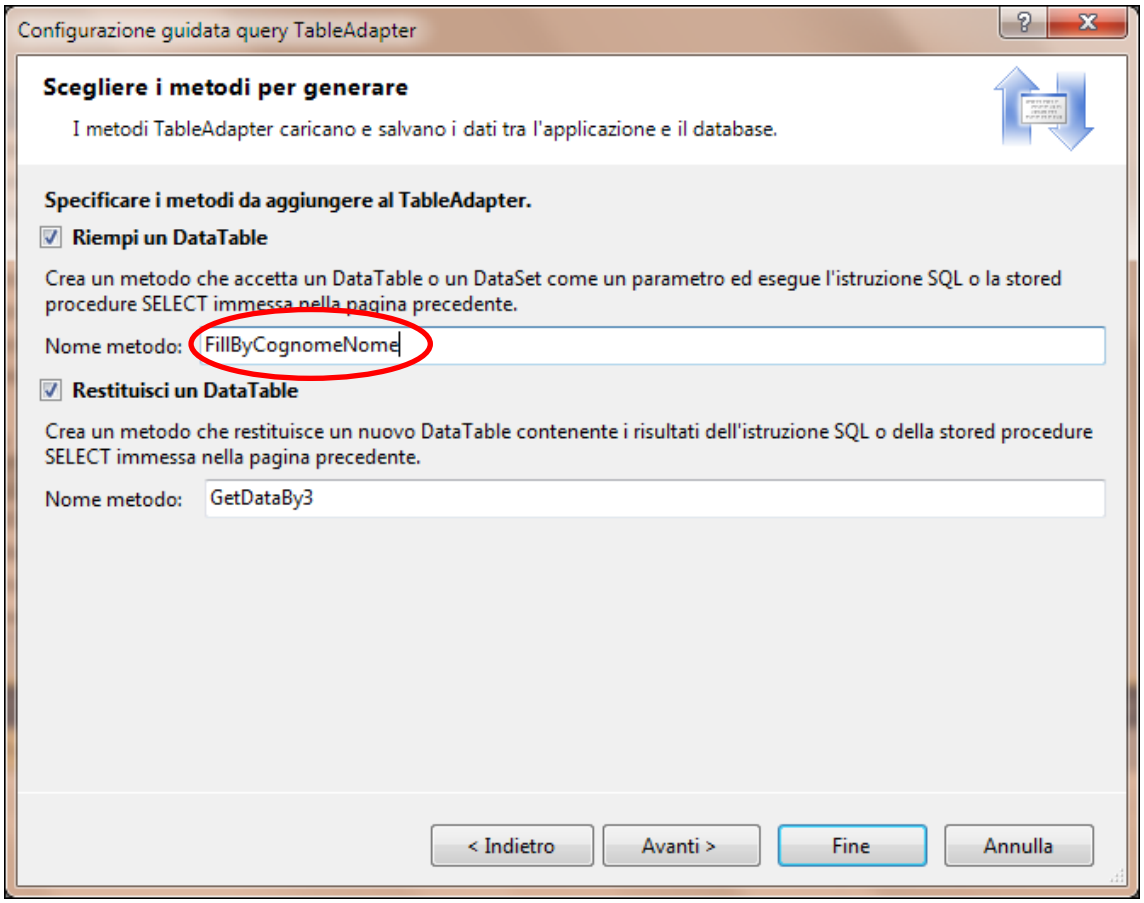


Figura 292: Completamento di una query di tipo SELECT.

Inseriamo ora nel **TableAdapter** altre due *query* di tipo **SELECT**, per selezionare nell'archivio, rispettivamente:

- gli alunni con altezza maggiore del dato immesso dall'utente del programma;
- gli alunni con peso maggiore del dato immesso dall'utente del programma.

Nella tabella seguente vediamo le indicazioni per creare queste due *query*, con le righe da aggiungere alle istruzioni e i loro nomi:

Righe che completano l'istruzione	Nome del metodo
WHERE (Peso > @Param1) ORDER BY Peso	FillByPeso
WHERE (Altezza > @Param1) ORDER BY Altezza	FillByAltezza

Con l'inserimento di queste due ultime *query* di selezione dei dati, il nostro **DataSet** è completo. Vi troviamo la tabella **Alunni** con i suoi campi e il **TableAdapter** con

- una *query* per l'inserimento dei dati,
- una *query* per la cancellazione dei dati, e
- quattro *query* di selezione dei dati secondo la provincia di nascita, il nome, l'altezza e il peso.

Resta da modificare la *query* **Fill**, creata automaticamente da VB quando abbiamo aggiunto la tabella **Alunni** al **DataSet**: si tratta della *query* usata per visualizzare il contenuto della tabella stessa, una *query* che in sostanza ha il compito di selezionare **tutti** i dati presenti nel database:

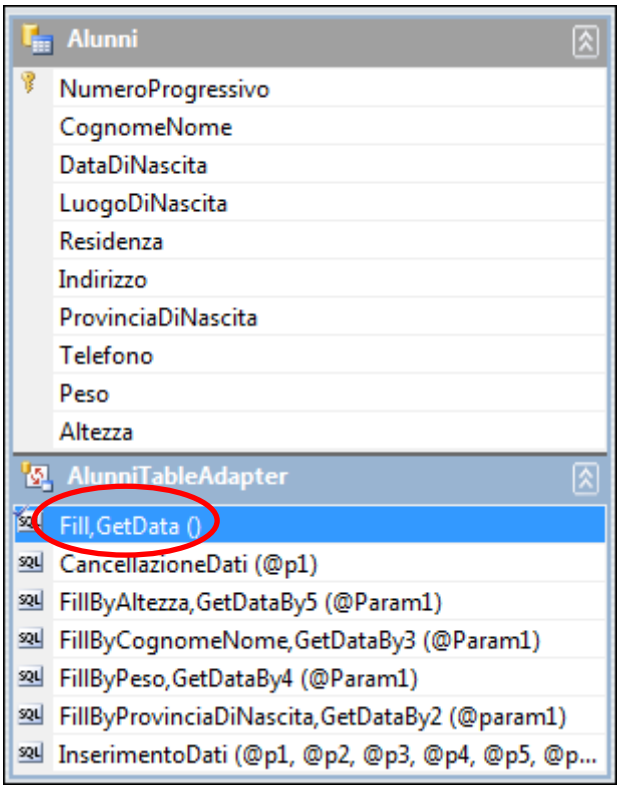


Figura 293: Il dataset completo di tabella e query, con la query Fill da modificare.

Impostiamo questa *query* di visualizzazione di tutti i dati in modo che gli alunni siano elencati in ordine alfabetico, secondo il campo **CognomeNome**.

Per modificare questa *query*, ma anche per apportare eventuali correzioni o modifiche alle altre *query*, operiamo in questo modo:

- facciamo un *clic* con il tasto destro del mouse sul nome della query (in questo caso la query Fill, prima nell'elenco);
- facciamo un *clic* sulla opzione Configura:

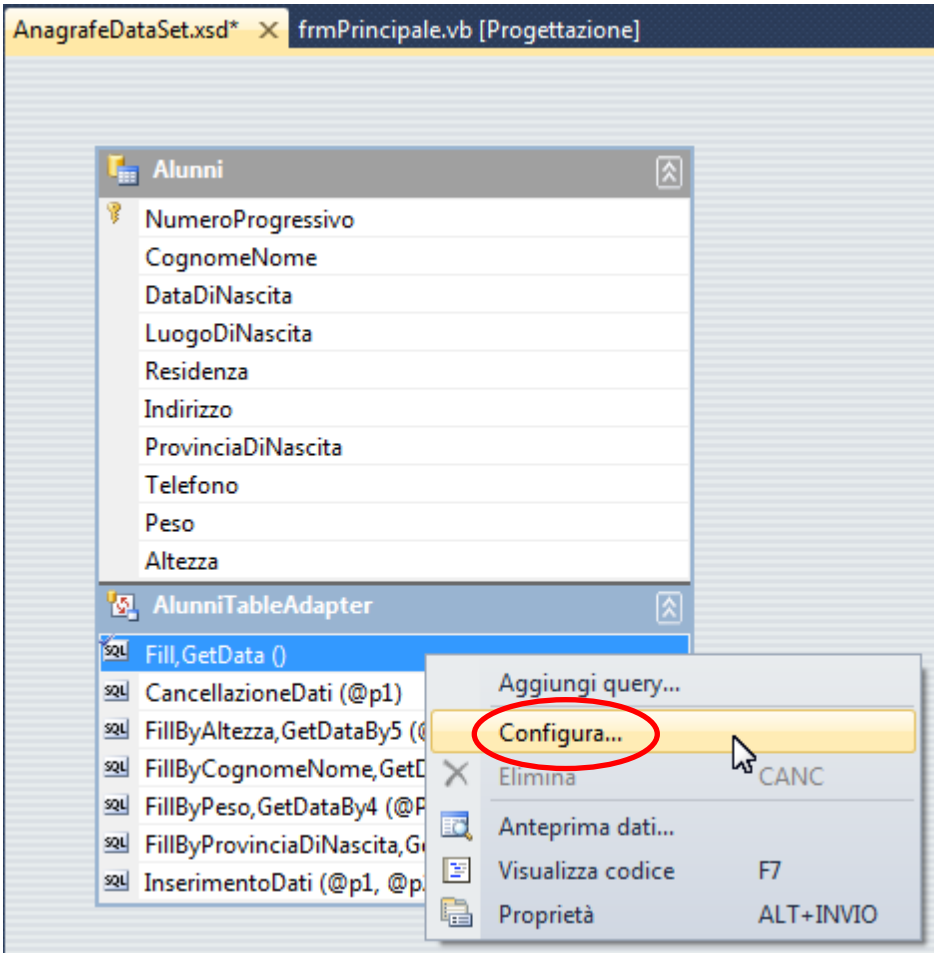


Figura 294: Modifica della *query* Fill.

Completiamo l'istruzione SQL con questa riga:
ORDER BY CognomeNome

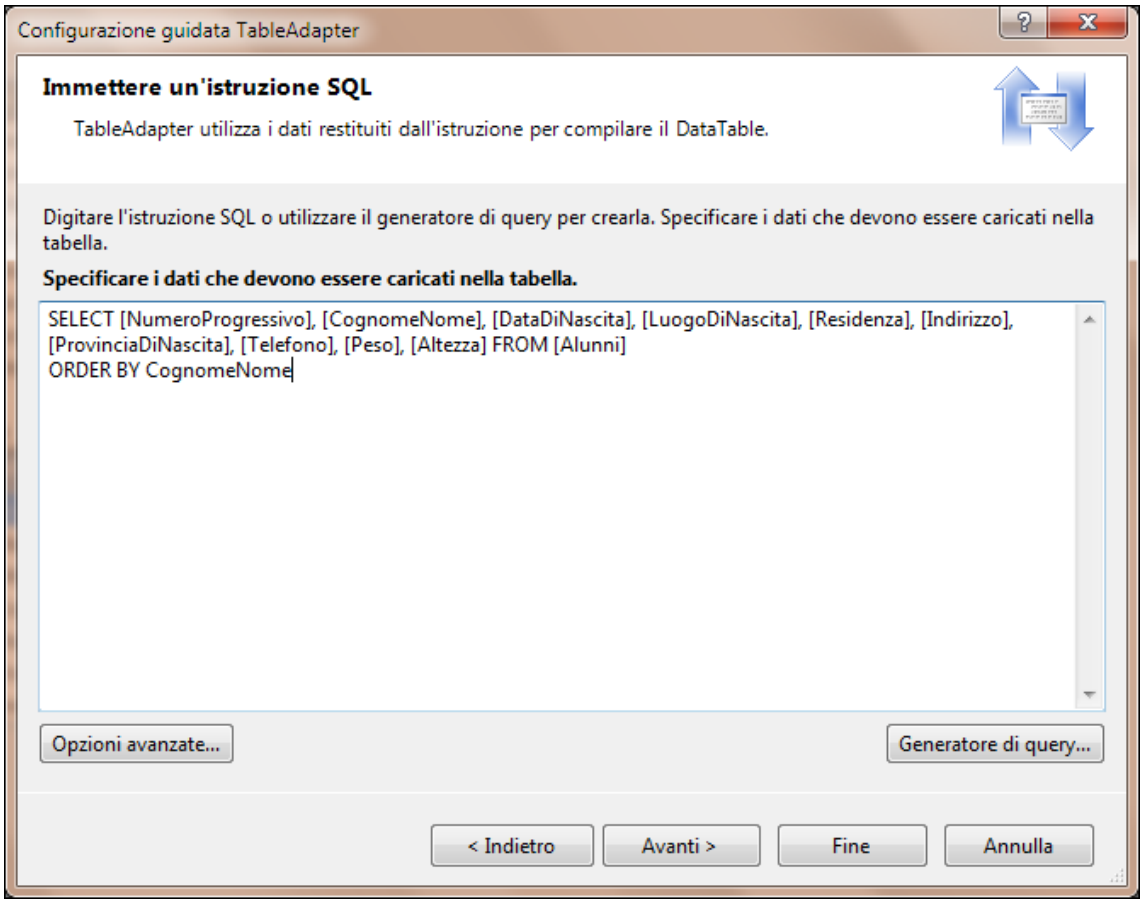


Figura 295: Completamento delle istruzioni per la *query* Fill.

Facciamo un *clic* sul pulsante **Fine** per concludere l'operazione.

200: Salvataggio e chiusura del dataset.

Ora il **DataSet** e il **TableAdapter** sono completi, non abbiamo altre *query* da inserirvi. Chiudiamo la finestra del **DataSet** ricordando di fare *clic* su **Sì** per salvare il lavoro svolto.

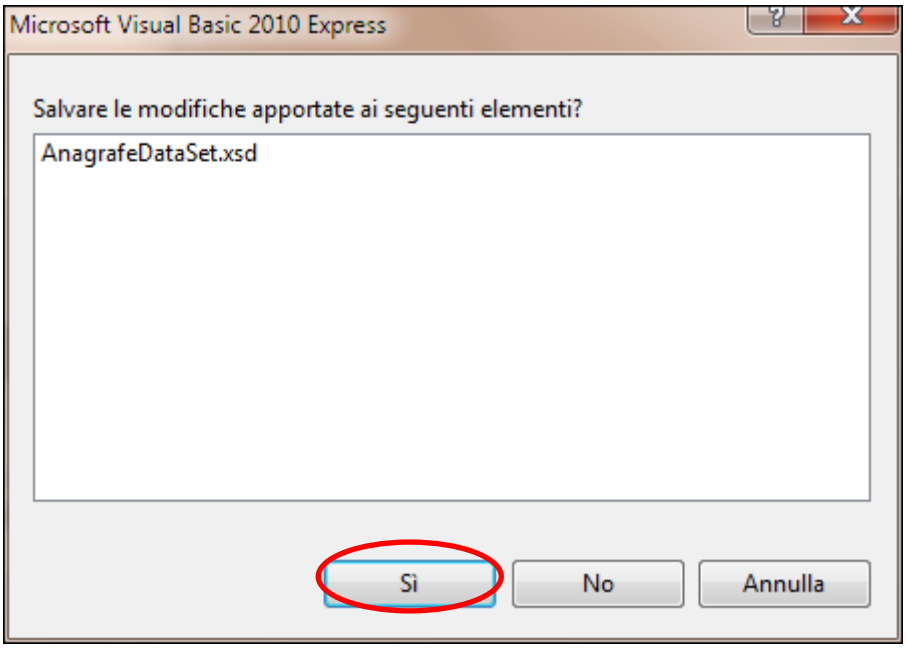


Figura 296: Chiusura e salvataggio del DataSet.

201: Denominazione dei principali tipi di dati in SQL.

Nome	Descrizione
nchar	Stringa di testo di lunghezza fissa, con al massimo 4.000 caratteri.
nvarchar	Stringa di testo di lunghezza variabile, con al massimo 4.000 caratteri.
tinyint	Numeri interi da 0 a 255.
smallint	Numeri interi da -32.768 a 32.767.
int	Numeri interi da -2.147.483.648 a 2.147.483.647.
bigint	Numeri interi da -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.
numeric	Numeri con la virgola.
money	Cifre relative a monete, da -922.337.203.685.477,5808 a 922.337.203.685.477,5807.
real	Numeri interi da -3.40E + 38 a 3.40E + 38.
datetime	Date dal 1 Gennaio 1753 al 31 dicembre 9999.

Tabella 37: Denominazione dei principali tipi di dati nel linguaggio SQL.

Capitolo 36: CREAZIONE DELL'INTERFACCIA.

Riprendiamo la progettazione del programma **Anagrafe alunni**, iniziata nel capitolo precedente.
Ricapitoliamo il cammino percorso:

- abbiamo aperto un nuovo progetto per la creazione di un archivio con i dati degli alunni di una classe;
- abbiamo visto come si crea un archivio di dati e come lo si inserisce un progetto VB;
- come si crea una tabella con i suoi campi, scegliendo per ciascun campo il tipo di dati idoneo;
- come si crea un **DataSet** per il collegamento tra un programma VB e un database, con un **TableAdapter** per la gestione e la selezione dei dati nella tabella.

Questo capitolo è dedicato alla creazione dell'interfaccia del programma, che sarà costituito da un form principale e da alcuni form secondari.
Vedremo la collocazione di oggetti e di pulsanti in queste finestre, i cui eventi avranno effetti sul database, per l'inserimento o la cancellazione di dati, per le eventuali correzioni e per le ricerche nell'archivio secondo criteri definiti dall'utente del programma.
Alla lettrice o al lettore non sarà sfuggito che sino a questo momento non abbiamo scritto una riga di codice; nelle prossime pagine colmeremo la lacuna.

202: Schema dell'interfaccia.

Il form principale di **Anagrafe alunni** svolge due funzioni:

- visualizza gli elenchi di alunni;
- presenta all'utente il menu del programma: da qui si può accedere agli altri form che hanno finalità specifiche per la gestione e la selezione delle schede dell'archivio.

La figura seguente mostra come il form principale sarà collegato a quattro form secondari, dedicati, rispettivamente, a queste funzioni:

- inserimento di schede con i dati di nuovi alunni;
- cancellazione o modifica di schede già esistenti;
- ricerca di informazioni in base a determinati criteri, e stampa dei dati ottenuti;
- stampa di un biglietto di auguri in occasione di compleanni.

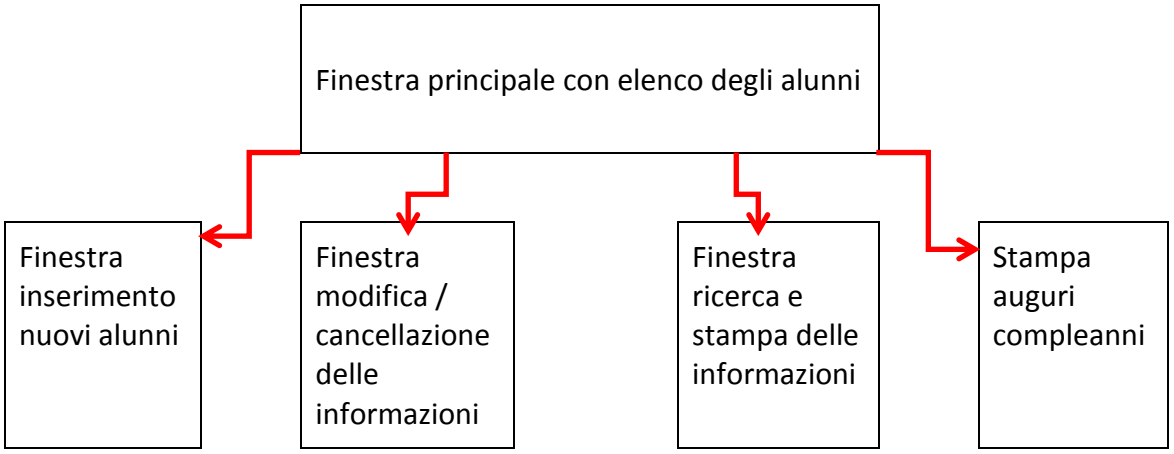


Figura 297: Schema del form principale del programma Anagrafe alunni.

Ma prima di iniziare a inserire gli oggetti nel form principale del programma, è necessario predisporre la schermata di VB con gli strumenti che saranno utili nel lavoro.

Facciamo un *clic* sul menu **Visualizza**, nella barra dei menu in alto, e poi su **Barre degli strumenti / Progettazione dati**:

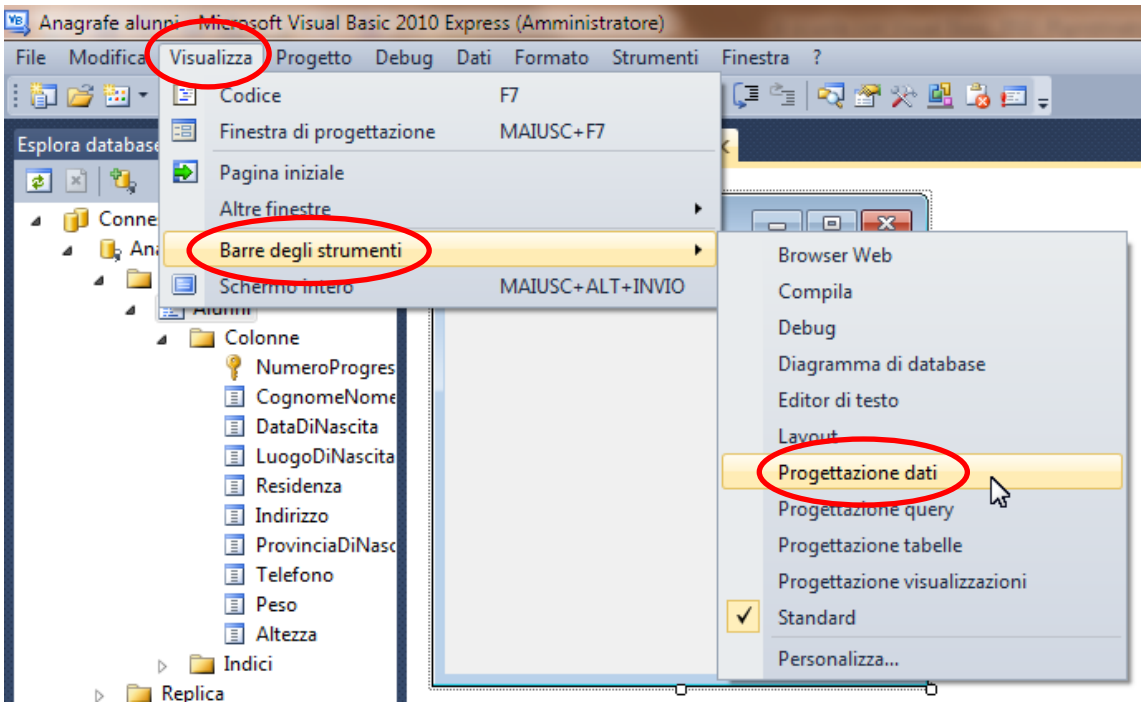


Figura 298: Visualizzazione della barra delle icone Progettazione dati.

La barra delle icone **Progettazione dati** si colloca al di sotto della barra Standard di VB:

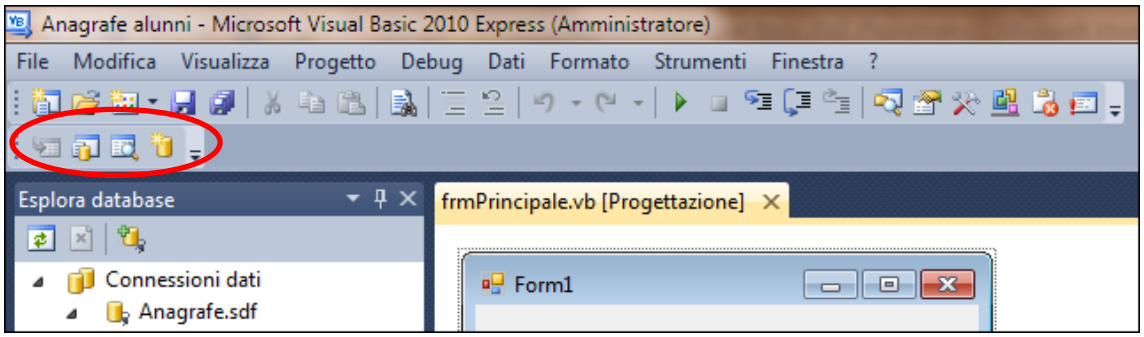


Figura 299: La barra delle icone Progettazione dati.

Ora nella barra **Progettazione dati** facciamo un *clic* sull'icona **Mostra origine** dati:

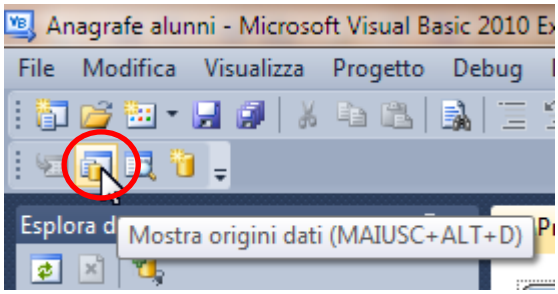


Figura 300: L'icona Mostra origine dati.

Accediamo così al pannello **Origini dati**, che contiene il **DataSet** già creato:

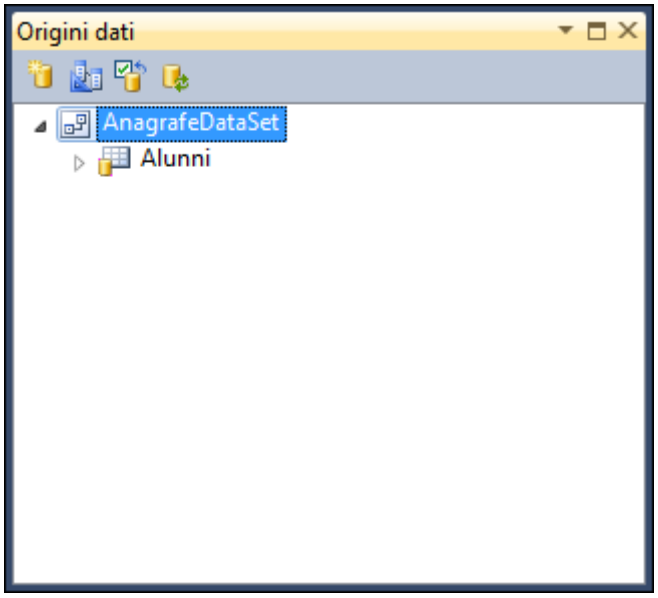


Figura 301: Il riquadro Origini dati con AnagrafeDataSet.

Trasciniamo il pannello **Origini dati** esattamente **sopra** la barra della finestra **Esplora soluzioni,** a destra del form, e rilasciamo il mouse:

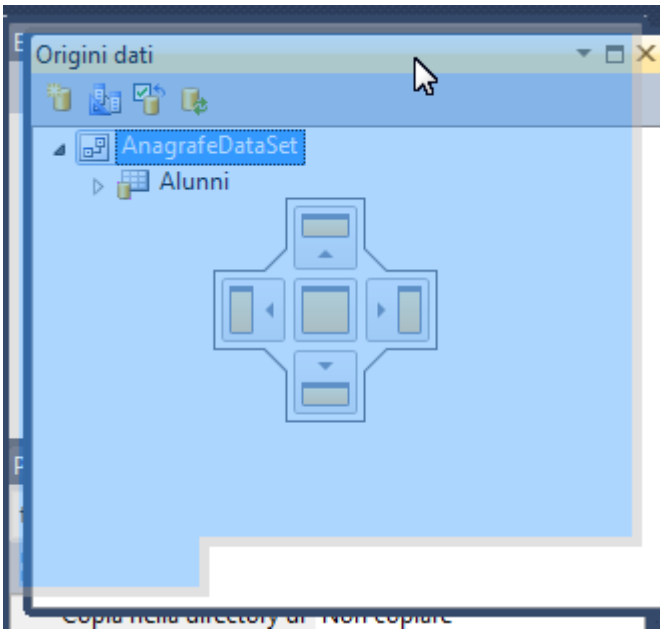


Figura 302: Trascinamento del pannello Origini dati sulla barra della finestra Esplora soluzioni.

Rilasciando il mouse, vediamo che la finestra **Origini dati** si è sovrapposta alla finestra **Esplora soluzioni**: si può passare da una finestra all'altra con un *clic* del mouse:

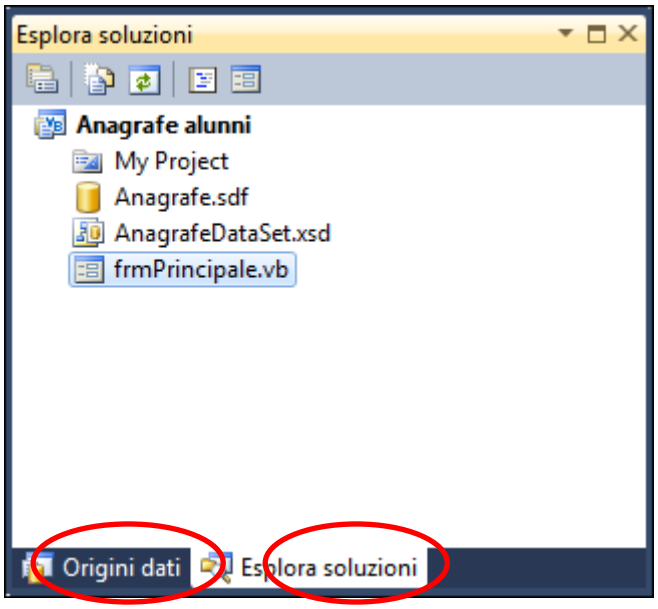


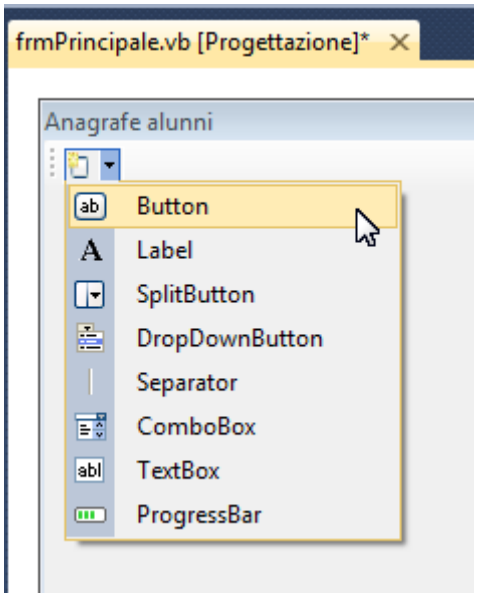
Figura 303: Passaggio dalla finestra Esplora soluzioni alla finestra Origini dati.

Nella finestra **Esplora soluzioni** facciamo un *clic* sul file **frmPrincipale.vb** e torniamo a visualizzare il form di base del nostro programma.

Al form di base abbiamo già dato il nome **frmPrincipale**. Ora impostiamo queste altre sue proprietà:

FormBorderStyle = FixedSingle	*Imposta la visualizzazione tradizionale del form e della sua barra in alto.*
MaximizeBox = False	*L'utente non potrà allargare le dimensioni del form.*
MinimizeBox = False	*L'utente non potrà ridurre le dimensioni del form.*
Showicon = False	*Nessuna icona nella barra del testo del form, in alto a sinistra.*
ShowInTaskbar = False	*Nessuna icona, per questo programma, nella Barra delle applicazioni.*
Size = 800; 570	*Dimensioni del form.*
StartPosition = CenterScreen	*Colloca il form al centro dello schermo.*
Text = Anagrafe alunni	*Questo è il testo che compare nella barra del form, in alto a sinistra.*

Ora preleviamo dalla **Casella degli Strumenti** e inseriamo nel form un componente **ToolStrip1**, destinato a contenere cinque pulsanti; inseriamo nel **ToolStrip1** il primo pulsante Button:



e continuiamo allo stesso modo sino a inserirvi cinque pulsanti:

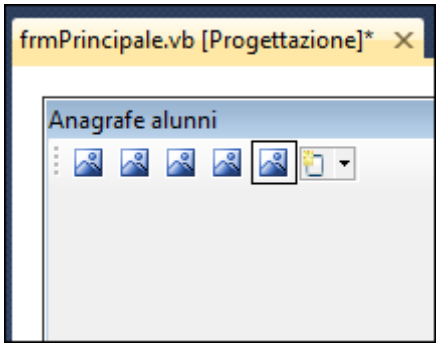


Figura 304: Inserimento di 5 pulsanti nel componente ToolStrip1.

Ora, con un *clic* su ognuno di questi cinque pulsanti, ne impostiamo queste proprietà:

Pulsante	**I**	**II**	**III**	**IV**	**V**
(Name)	cmdInserisci	cmdModifica	cmdRicerca	cmdCompleanni	cmdEsci
Display Style	Text	Text	Text	Text	Text
Text	Inserimento nuovi alunni	Modifiche e cancellazioni	Ricerche e stampa	Stampa auguri	Esci

Inseriamo nel form un componente **StatusStrip**. Questo componente va a collocarsi nella parte inferiore del form. Vi inseriamo una **label** (etichetta), destinata a contenere informazioni sull'uso del programma. Proprietà di questa label:
(Name) = lblIstruzioni
Text = Istruzioni

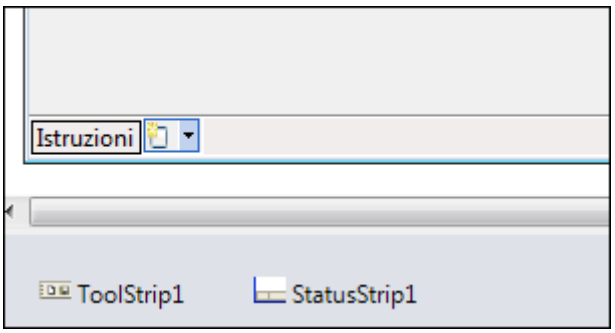


Figura 305: Inserimento di un componente StatusStrip con la label Istruzioni.

Inseriamo nel form anche un controllo **ListBox** che rimarrà non visibile all'utente: esso servirà come deposito d'informazioni temporanee, per registrare il nome e la data di nascita degli alunni che festeggiano il compleanno nel giorno corrente.
La proprietà **Visibile** del controllo **ListBox1** è dunque impostata = **False.**
Ecco come si presenta il form principale allo stato attuale:

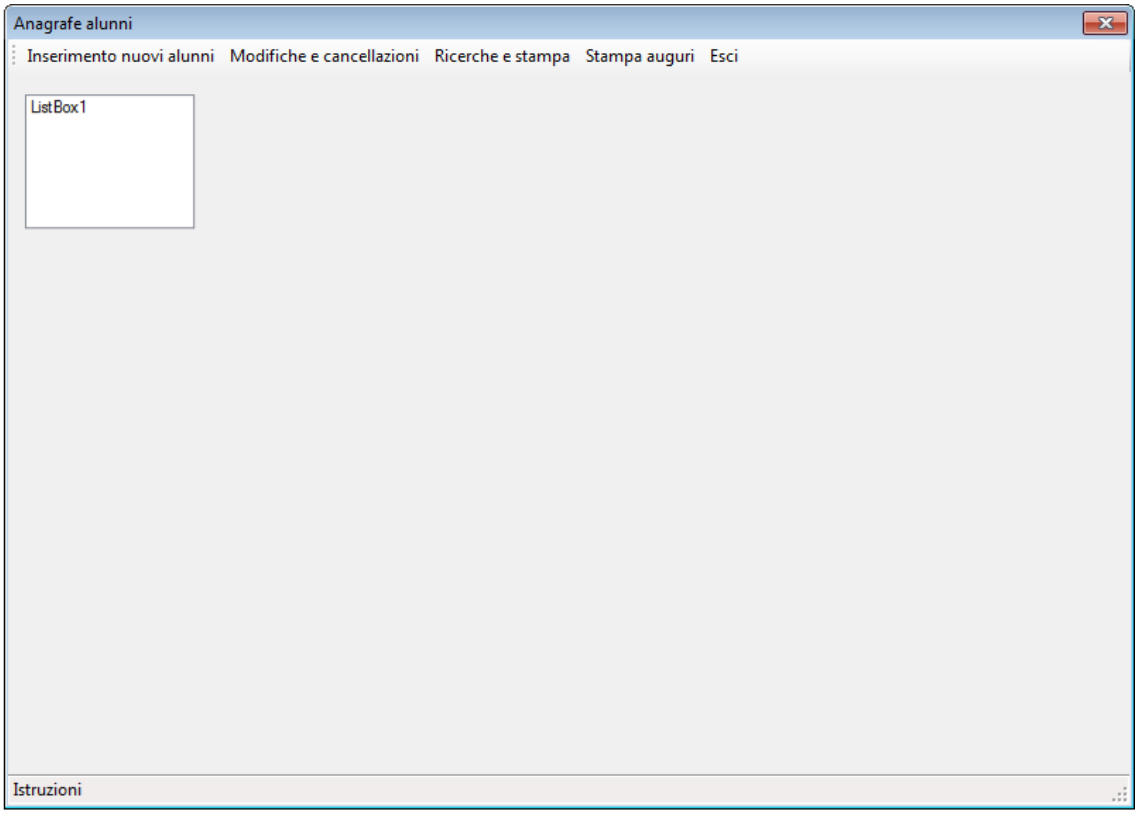


Figura 306: Il frmPrincipale.

Con un doppio *clic* sul pulsante **Esci**, apriamo la **Finestra del Codice** per scrivervi subito la procedura che gestisce la fine del programma.
Questa procedura, e le altre che seguiranno, possono essere copiate e incollate nella **Finestra del Codice**, collocandole tra le righe che conterranno tutto il listato relativo a questo form:

```
Public Class frmPrincipale

    ' collocare qui la procedura per la chiusura del programma, e
    ' tutte le procedure che seguiranno

End Class
```

La prima procedura da inserire è questa:

```
Private Sub cmdEsci_Click() Handles cmdEsci.Click

    'chiude il programma
    Application.Exit()

End Sub
```

All'avvio del programma, i pulsanti **Modifiche** e **Ricerche e stampe** risulteranno visibili e attivi se nell'archivio sono presenti dei dati, altrimenti l'utente potrà solo cliccare i pulsanti per **inserire** nuovi dati (pulsante **Nuovi alunni**) o per **uscire** dal programma pulsante **Esci**).
Per la gestione dell'evento *clic* su questi altri pulsanti è necessario attendere l'inserimento nel progetto dei form aggiuntivi che verranno attivati da questi pulsanti.

Completiamo questo **frmPrincipale** con una tavola in cui saranno visualizzate le informazioni sui singoli alunni.
La creazione e l'adattamento grafico di questa tavola richiedono una serie di operazioni.
Iniziamo trascinando la tabella **Alunni** dal pannello **Origini dati** nella finestra del **frmPrincipale:**

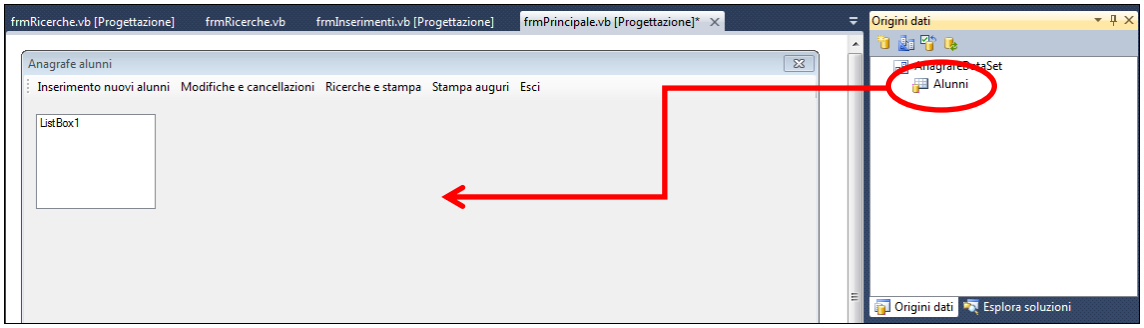


Figura 307: Trascinamento della tabella Alunni nel frmPrincipale.

Rilasciando il mouse, troviamo nel **frmPrincipale** la nuova tavola, ancora da dimensionare:

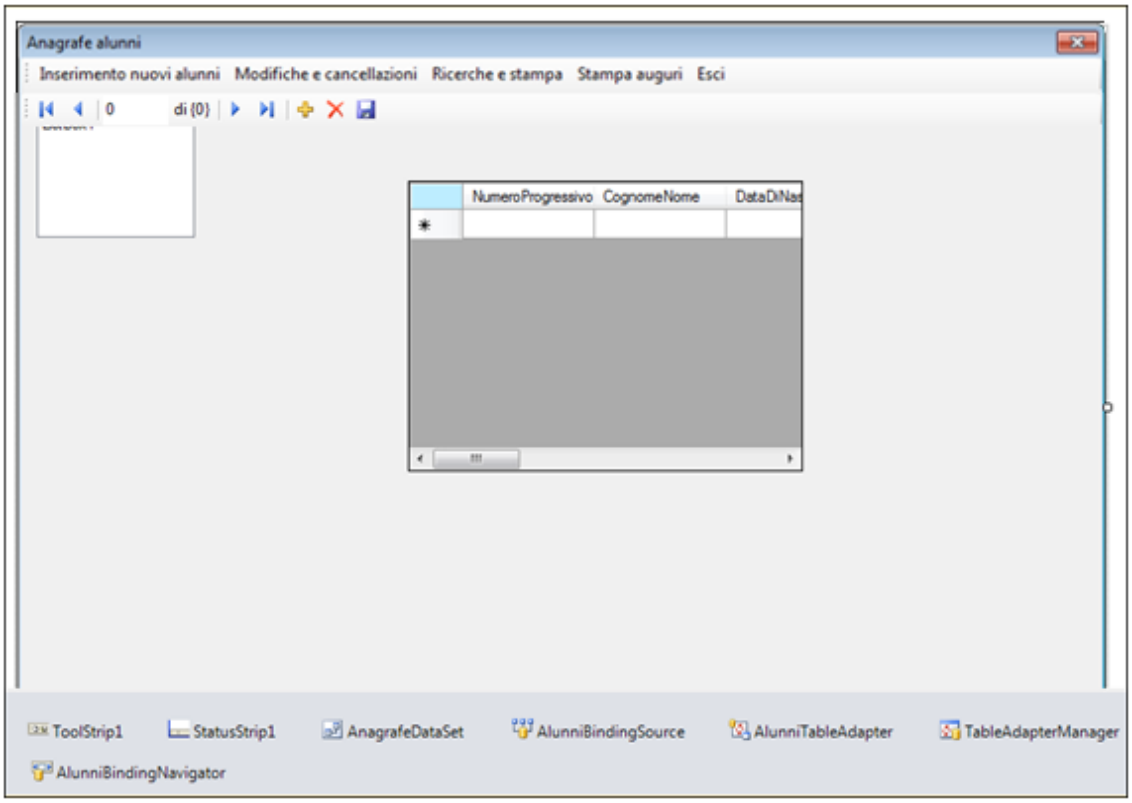


Figura 308: La tabella Alunni nel frmPrincipale.

Notiamo, nella **Finestra Proprietà**, che VB ha assegnato automaticamente a questa tavola il nome **AlunniDataGridView** (= visualizzazione della griglia dei dati degli alunni). Notiamo, nello spazio sottostante il **frmPrincipale**, che VB vi ha inserito automaticamente altri componenti, oltre al **ToolStrip1** e allo **StatusStrip** che erano già presenti:

- **AnagrafeDataSet** (= sistemazione di dati: è lo strumento che abbiamo creato nel capitolo precedente);
- **AlunniBindingSource** (= fonte per il collegamento dei dati);
- **AlunniTableAdapter** (= adattatore della tabella **Alunni**);
- **TableAdapterManager** (= gestione della tabella Alunni);
- **AlunniBindingNavigator** (= strumento di visualizzazione dei dati uno a uno).

L'ultimo componente, **AlunniBindingNavigator**, è uno strumento per scorrere le schede del database una a una; lo useremo nel form dedicato alle modifiche dei dati ma non in questo form, per cui lo eliminiamo da questa finestra con un *clic* del tasto destro del mouse sul componente e poi un *clic* sul menu **Elimina**:

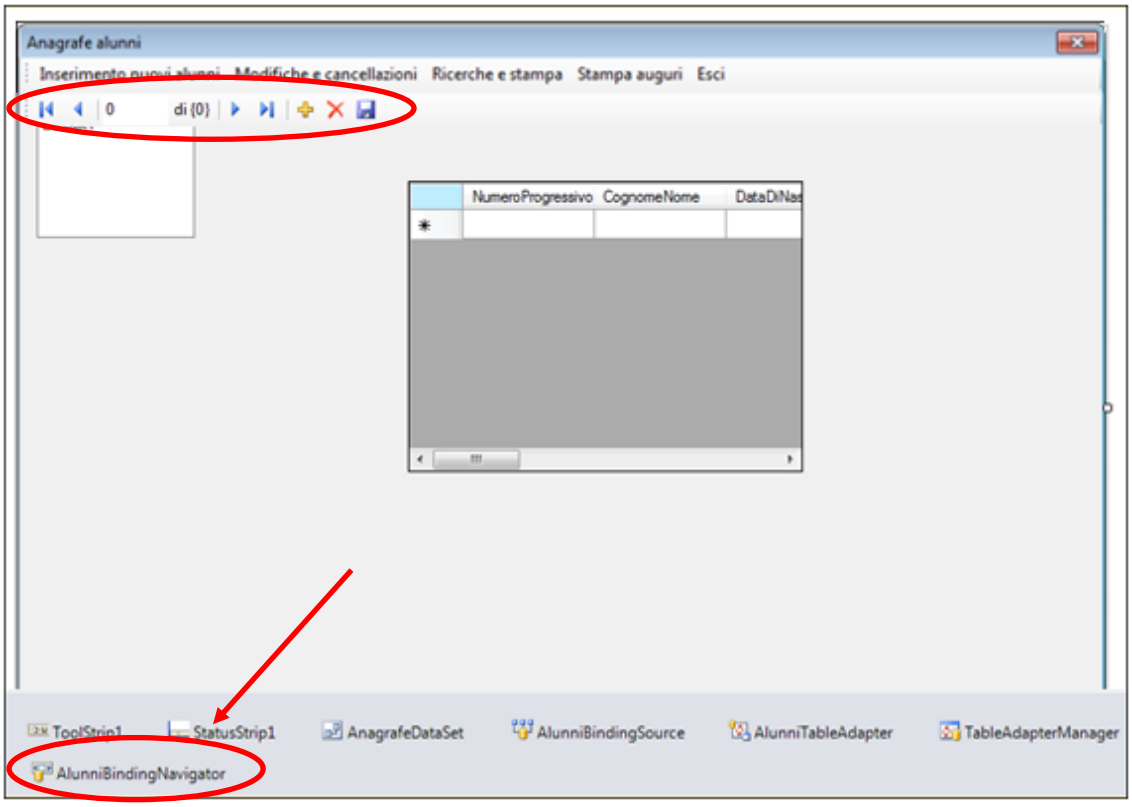


Figura 309: Eliminazione del componente AlunniBindingNavigator.

Impostiamo alcune proprietà della griglia **AlunniDataGridView** appena inserita nel form.
Facciamo un *clic* sulla griglia, poi un altro *clic* sul bordo, sulla freccina nera che compare in alto a destra.
Apriamo così il menu **Attività di DataGridView**, dove disattiviamo le voci

- Abilita aggiunta,
- Abilita modifica,
- Abilita eliminazione.

Attiviamo invece

- Abilita riordinamento colonne

e facciamo un *clic* sul comando **Ancora nel contenitore padre**, per dimensionare la griglia in modo che essa occupi tutto lo spazio disponibile nel form.

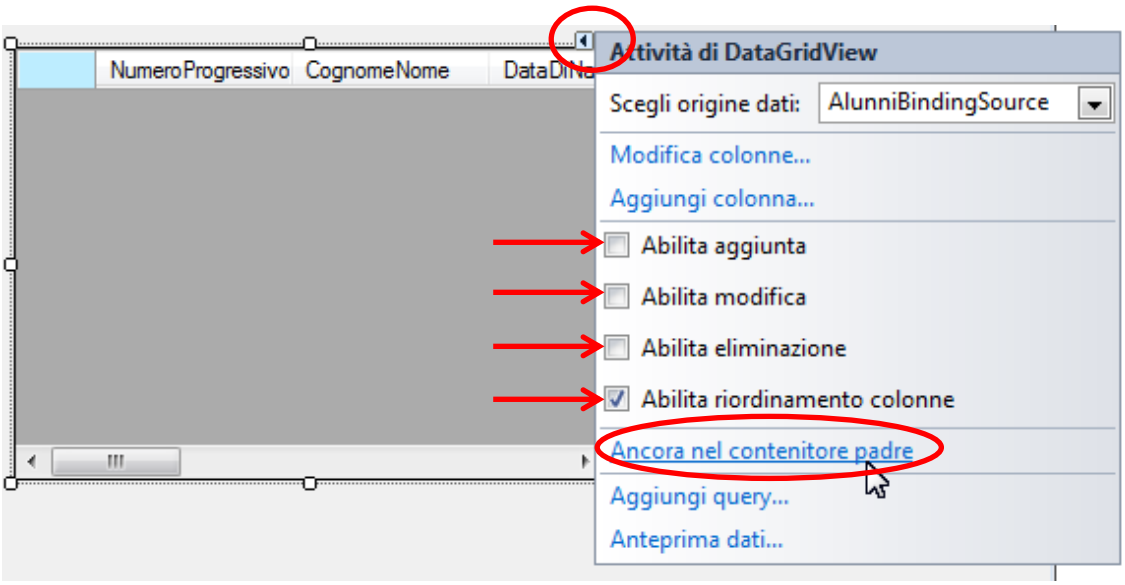


Figura 310: Impostazione delle attività della griglia AlunniDataGridView.

Con queste impostazioni, i dati visualizzati nella griglia saranno al riparo da interventi accidentali da parte dell'utente del programma: questi non potrà scrivere e modificare i dati direttamente nella griglia; avrà solo la facoltà di ordinare le schede in base al contenuto delle colonne.

Ora, nello stesso menu delle **Attività di DataGridView**, facciamo un *clic* sulla voce **Modifica colonne**… e accediamo alla finestra con le impostazioni di visualizzazione delle colonne:

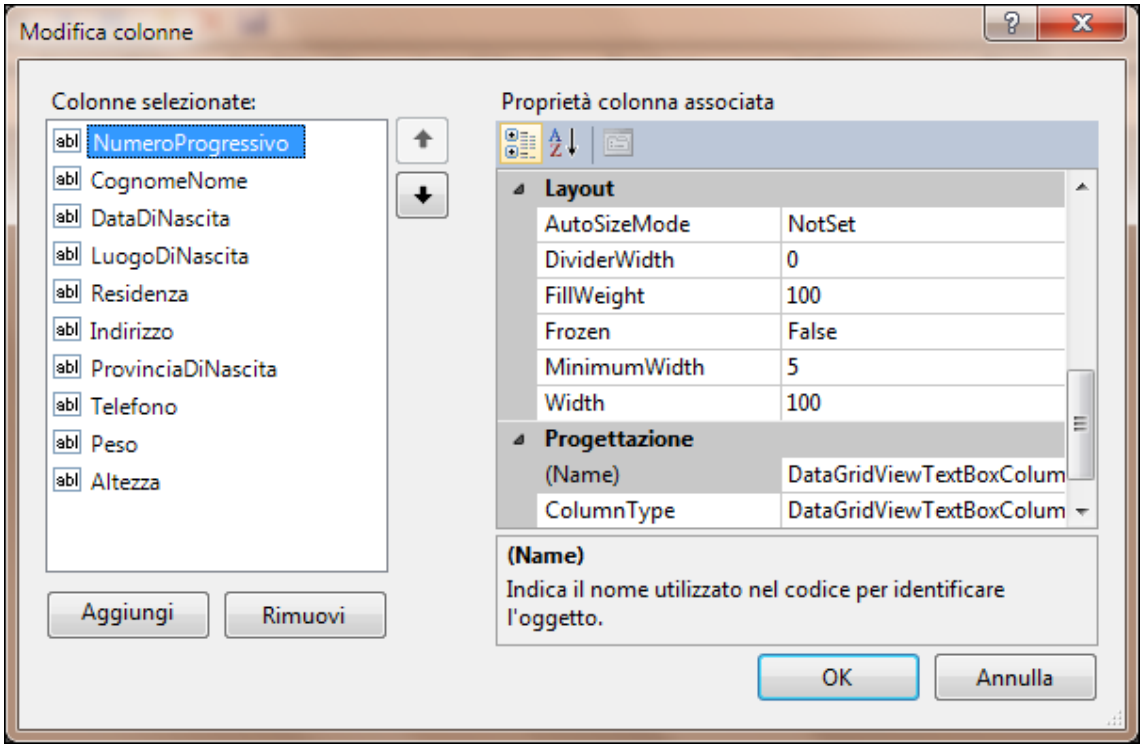


Figura 311: La finestra Modifica colonne di DataGridView.

In questa finestra **Modifica colonne** possiamo decidere innanzitutto quali colonne di dati vogliamo visualizzare nella griglia (i dati delle colonne nascoste nella griglia continueranno comunque a essere presenti e aggiornati nell'archivio).
Il luogo e la provincia di nascita sono informazioni che si usano raramente, per cui nascondiamo le colonne

- LuogoDiNascita e
- ProvinciaDiNascita

selezionandole con un *clic* del mouse e poi con un *clic* sul pulsante **Rimuovi**.

Ecco le otto colonne che saranno visibili:

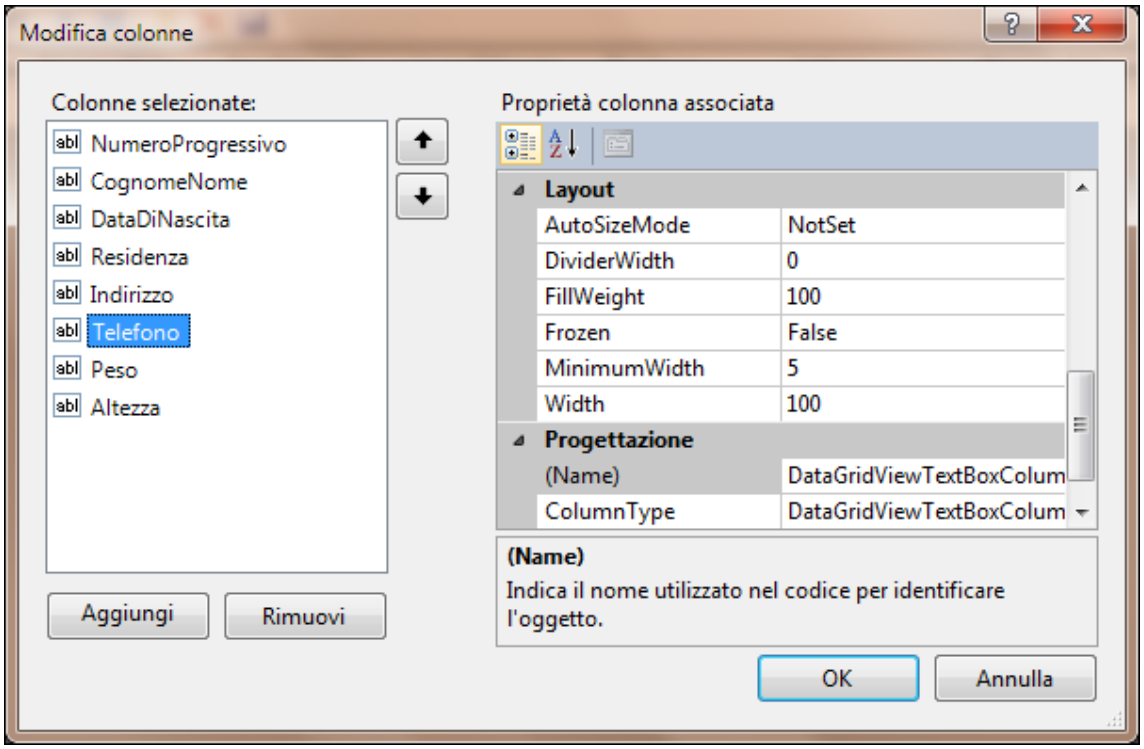


Figura 312: Selezione delle colonne della griglia AlunniDataGridView.

L'elenco che compare nella parte destra della finestra (**Proprietà colonna associata**), ci permette di impostare alcune proprietà delle colonne, allo scopo di ottimizzare la visibilità delle informazioni contenute nella tavola.
Per ciascuna colonna, modificheremo queste proprietà:

- **Frozen** (indica se negli scorrimenti orizzontali della tabella la colonna rimarrà bloccata al suo posto oppure si sposterà assieme alla tabella);
- **HeaderText** (è la didascalìa di intestazione della colonna);
- **ReadOnly** (indica se l'utente può modificare o meno il contenuto della colonna);
- **Width** (impostazione della larghezza della colonna).

Per la prima colonna, **NumeroProgressivo**, impostiamo queste proprietà:

- Frozen = True
- HeaderText = N.
- ReadOnly = True
- Width = 20

Procediamo con le altre colonne, impostandone le proprietà in questo modo:

Colonna **CognomeNome**:

Frozen = False
HeaderText = Alunna/o

ReadOnly = True
Width = 150

Colonna **DataDiNascita**:
Frozen = False
HeaderText = Nata/o il
ReadOnly = True
Width = 100

Colonna **Residenza**:
Frozen = False
HeaderText = Residente a
ReadOnly = True
Width = 100

Colonna **Indirizzo**:
Frozen = False
HeaderText = Indirizzo
ReadOnly = True
Width = 200

Colonna **Telefono**:
Frozen = False
HeaderText = Tel.
ReadOnly = True
Width = 80

Colonna **Peso**:
Frozen = False
HeaderText = Peso kg
ReadOnly = True
Width = 50

Colonna **Altezza**:
Frozen = False
HeaderText = Altezza cm
ReadOnly = True
Width = 50

Al termine, confermiamo tutte le impostazioni con un *clic* sul pulsante OK.
Le impostazioni della proprietà **Width** possono essere corrette, anche in seguito, per raggiungere il livello ottimale di visualizzazione della griglia e dei dati in essa contenuti.

Ecco come compare ora la griglia di visualizzazione dei dati nel **frmPrincipale**:

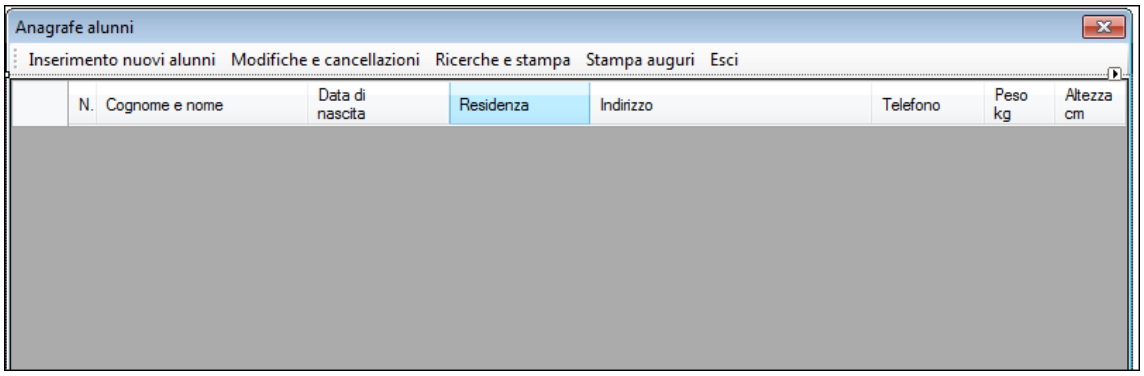


Figura 313: Completamento della griglia AlunniDataGridView.

Torniamo a aprire la **Finestra del Codice**: vi troviamo alcune procedure inserite automaticamente da VB.
Possiamo cancellare le procedure relative al componente **AlunniBindingNavigator**, che abbiamo eliminato dal progetto.
Aggiungiamo una procedura relativa all'evento della attivazione del **frmPrincipale**: in apertura del programma e ogni volta che l'utente torna a questo form, cioè ogni volta che lo riattiva, la procedura esegue alcuni controlli e visualizza oggetti e istruzioni a seconda della situazione:

```
Private Sub frmPrincipale_Activated(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Activated

    AlunniDataGridView.Visible = False
    cmdModifica.Enabled = False
    cmdRicerca.Enabled = False
    cmdCompleanni.Enabled = False
    lblIstruzioni.Text = "L'archivio non contiene dati."

    ' Visualizza i dati nella griglia:
    Me.AlunniTableAdapter.Fill(Me.AnagrafeDataSet.Alunni)

    ' Se la tabella contiene dei dati...
    If Me.AnagrafeDataSet.Alunni.Count > 0 Then
        ' In caso positivo visualizza la griglia e attiva i pulsanti:
        AlunniDataGridView.Visible = True
        cmdModifica.Enabled = True
        cmdRicerca.Enabled = True
        lblIstruzioni.Text = "L'archivio contiene i dati di un solo alunno."
        If Me.AnagrafeDataSet.Alunni.Count > 1 Then lblIstruzioni.Text =
"L'archivio contiene i dati di " & Me.AnagrafeDataSet.Alunni.Count & "
alunni."
        lblIstruzioni.Text &= " Premi i pulsanti in alto per inserire nuovi
alunni, per correggere i dati o per fare ricerche."
    Else
        ' in caso negativo nasconde o disattiva griglia, informazioni e
pulsanti:
    End If
```

```
    ' Ripulisce il ListBox1:
    ListBox1.Items.Clear()

    ' Crea la variabile Compleanno che conterrà la data di nascita di ciascun
alunno:
    Dim Compleanno As Date

    ' Controlla tutte le righe della tabella alla ricerca della data di
nascita di ciascun alunno:
    For Each Riga As DataGridViewRow In AlunniDataGridView.Rows

        Compleanno = Riga.Cells("DataGridViewTextBoxColumn3").Value

        ' Se giorno e mese della data di nascita di un alunno corrispondono a
giorno e mese odierni...
        If Compleanno.Day = Date.Today.Day And Compleanno.Month =
Date.Today.Month Then
            '... allora la riga del compleanno è evidenziata in giallo
            Riga.DefaultCellStyle.BackColor = Color.Yellow

            ' Aggiunge nome e data di nascita dell'alunno che compie gli anni
al controllo ListBox1:
            ' questi dati saranno usati in un altro form, per l'eventuale
stampa del biglietto di auguri:

ListBox1.Items.Add(Riga.Cells("DataGridViewTextBoxColumn2").Value.ToString &
"-" & Compleanno.ToShortDateString)
            ' rende attivo il pulsante di stampa degli auguri
            cmdCompleanni.Enabled = True
            lblIstruzioni.Text = "La riga gialla segnala un compleanno."
            If ListBox1.Items.Count - 1 > 1 Then lblIstruzioni.Text = "Le
righe gialle segnalano un compleanno."

        End If
    Next

End Sub
```

Lasciamo a questo punto la scrittura del codice per il **frmPrincipale**; vi torneremo dopo avere inserito nel progetto i form secondari.

203: Aggiunta delle finestre secondarie.

E' giunto il momento di aggiungere al progetto quattro form secondari, che saranno dedicati, rispettivamente:

- all'inserimento di nuovi alunni;
- alla modifica o alla cancellazione di dati esistenti;
- all'effettuazione di ricerche all'interno dei dati esistenti;
- alla stampa di un biglietto di auguri in occasione di compleanni.

Nella barra dei menu di VB, facciamo un *clic* sul menu **Progetto / Aggiungi Windows form**:

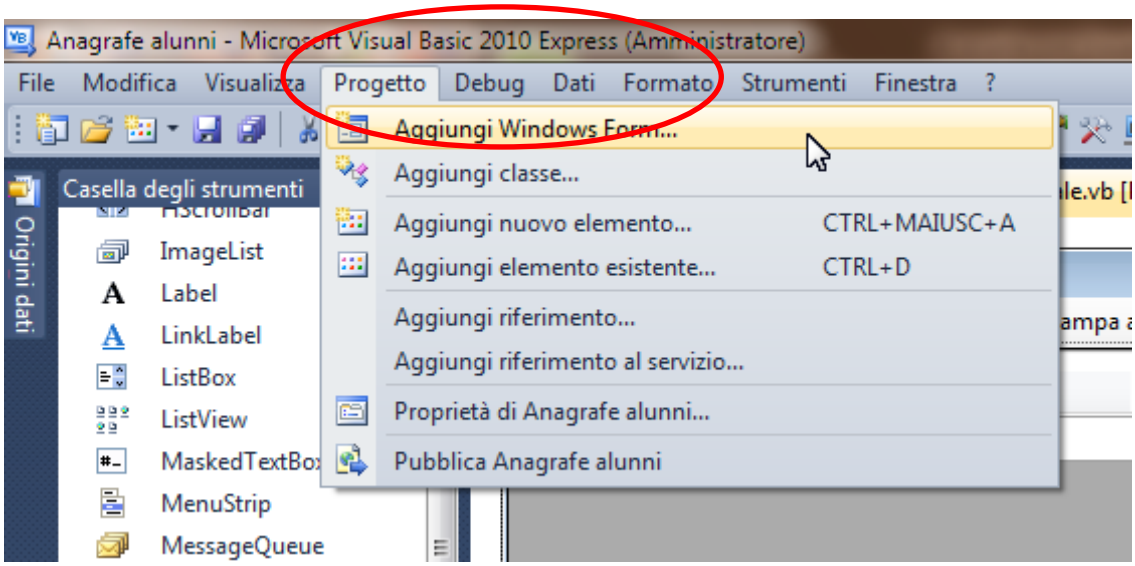


Figura 314: L'aggiunta di un nuovo form al progetto.

Nella finestra che si apre, scriviamo in basso il nome del nuovo form; al posto di Form1 scriviamo **frmInserimenti.vb** e facciamo *clic* sul pulsante **Aggiungi**.
Questa operazione va effettuata quattro volte, per inserire nel progetto altrettanti form i cui file saranno denominati, rispettivamente:

- **frmInserimenti.vb**
- **frmModifiche.vb**
- **frmRicerche.vb**
- **frmAuguri.vb**

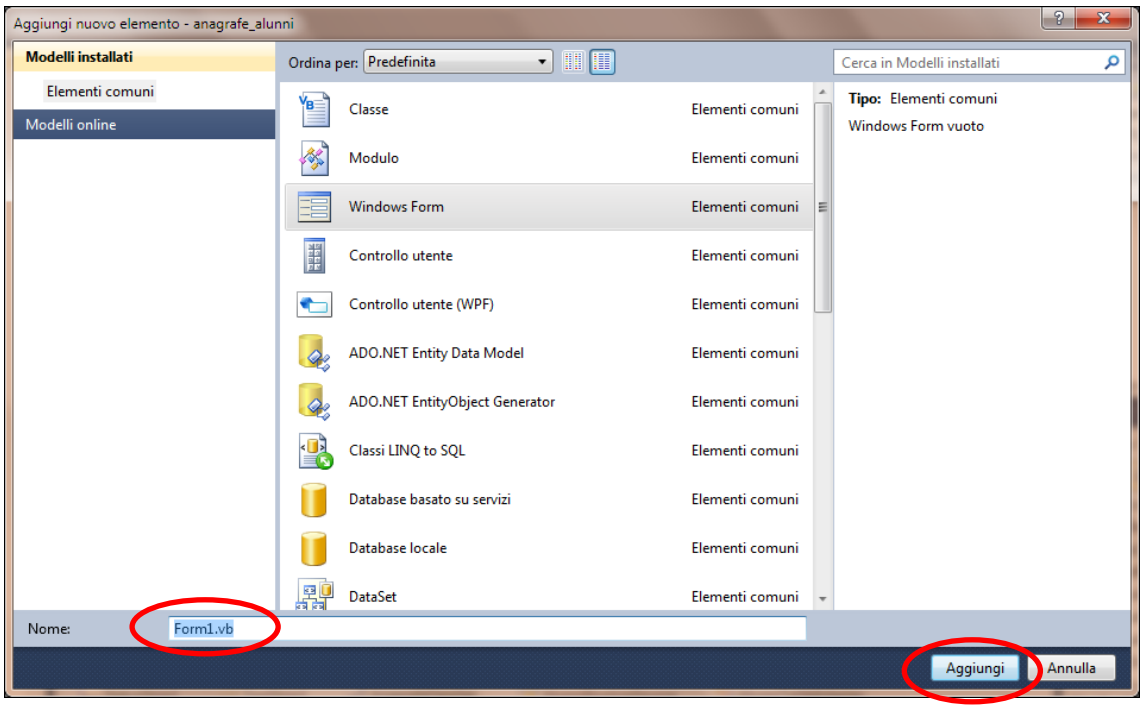


Figura 315: Inserimento e denominazione di un nuovo form.

Dopo l'inserimento di questi quattro form, la finestra **Esplora soluzioni** del nostro progetto si presenta in questo modo:

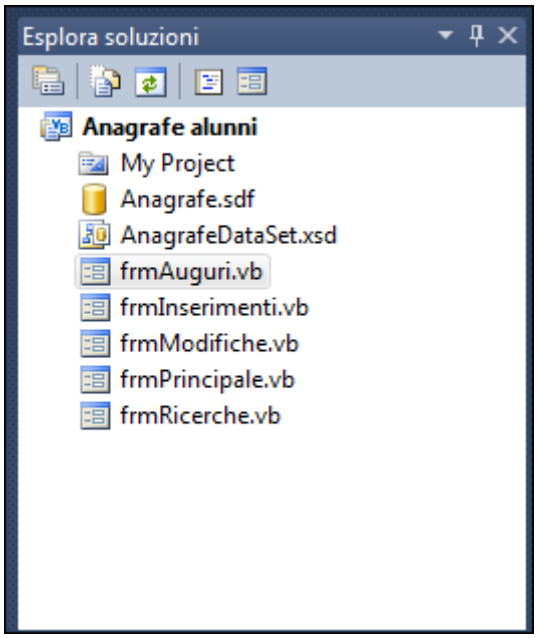


Figura 316: I nuovi form nella finestra Esplora soluzioni.

Possiamo ora scrivere, nella **Finestra del Codice** del frmPrincipale, le procedure per aprire i quattro form che abbiamo appena inserito nel progetto:

```
    Private Sub cmdInserisci_Click() Handles cmdInserisci.Click

    ' apre il form di inserimento delle nuove informazioni
    frmInserimenti.ShowDialog()

End Sub
```

```
Private Sub cmdModifica_Click() Handles cmdModifica.Click

    ' apre il form in cui si modificano/eliminano le informazioni
    frmModifiche.ShowDialog()

End Sub
```

```
Private Sub cmdRicerca_Click() Handles cmdRicerca.Click

    'apre il form con la ricerca e la stampa delle informazioni
    frmRicerche.ShowDialog()

End Sub
```

```
Private Sub cmdCompleanni_Click() Handles cmdCompleanni.Click

    'apre il form con la stampa degli auguri
    frmAuguri.ShowDialog()

End Sub
```

204: La finestra per l'inserimento dei dati.

Nella finestra **Esplora soluzioni**, facciamo un doppio *clic* sul form **frmInserimenti.vb**.

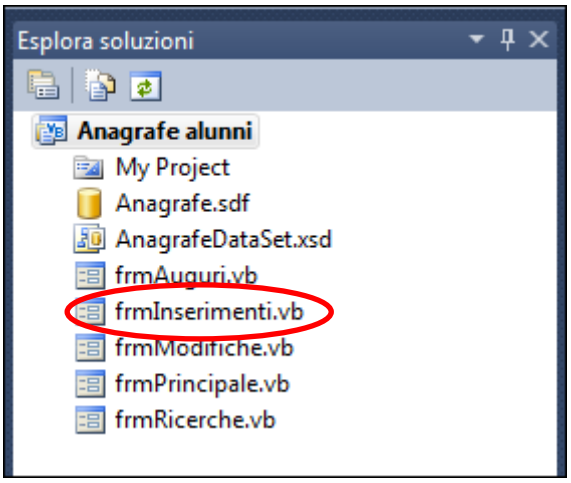


Figura 317: Apertura del frmInserimenti.

Questa è la finestra che l'utente userà per inserire nell'archivio i dati degli alunni. Impostiamo le proprietà del form in questo modo:

FormBorderStyle = FixedSingle	Imposta la visualizzazione tradizionale del form e della sua barra in alto.
MaximizeBox = False	L'utente non potrà allargare le dimensioni del form.
MinimizeBox = False	L'utente non potrà ridurre le dimensioni del form.
Showicon = False	Nessuna icona nella barra del testo del form, in alto a sinistra.
ShowInTaskbar = False	Nessuna icona, per questo programma, nella Barra delle applicazioni.
Size = 800; 570	Dimensioni del form.
StartPosition = CenterScreen	Colloca il form al centro dello schermo.
Text = Inserimento nuovi alunni	Questo è il testo che compare nella barra del form, in alto a sinistra.

Inseriamo nel form questi controlli, collocandoli seguendo l'esempio dell'immagine seguente:

- Un componente **ToolStrip** nel quale inseriamo tre pulsanti.

Con un *clic* su ognuno di questi tre pulsanti, ne impostiamo queste proprietà:

Pulsante	**I**	**II**	**III**
(Name)	cmdInserisci	cmdSalva	cmdEsci
DisplayStyle	Text	Text	Text
Text	Nuova/o alunna/o	Salva	Esci

- Inseriamo un componente StatusStrip, destinato a contenere solo una etichetta con informazioni sull'uso del programma. Proprietà di questa label:
  (Name) = lblIstruzioni
  Text = Istruzioni

- Inseriamo nove **TextBox**: questi TextBox sono le caselle di testo in cui l'utente scriverà i dati relativi a ogni nuovo alunno. Impostiamo la loro proprietà (**Name**), rispettivamente, con questi nomi:
  txtCognomeNome
  txtDataDiNascita
  txtLuogoDiNascita
  txtProvinciaDiNascita
  txtResidenza
  txtIndirizzo
  txtTelefono
  txtPeso
  txtAltezza
  In ciascuno di questi **TextBox**, partendo dal primo, impostiamo il valore della proprietà **TabIndex** con un numero crescente, **da 0 a 8**.

- Inseriamo nove controlli Label (etichette) che serviranno come didascalìe per i nove TextBox. Nessun evento sarà collegato a queste etichette, per cui non è necessario impostare alcuna delle loro proprietà.

Sistemiamo in modo ordinato le caselle di testo con le rispettive etichette in due colonne:

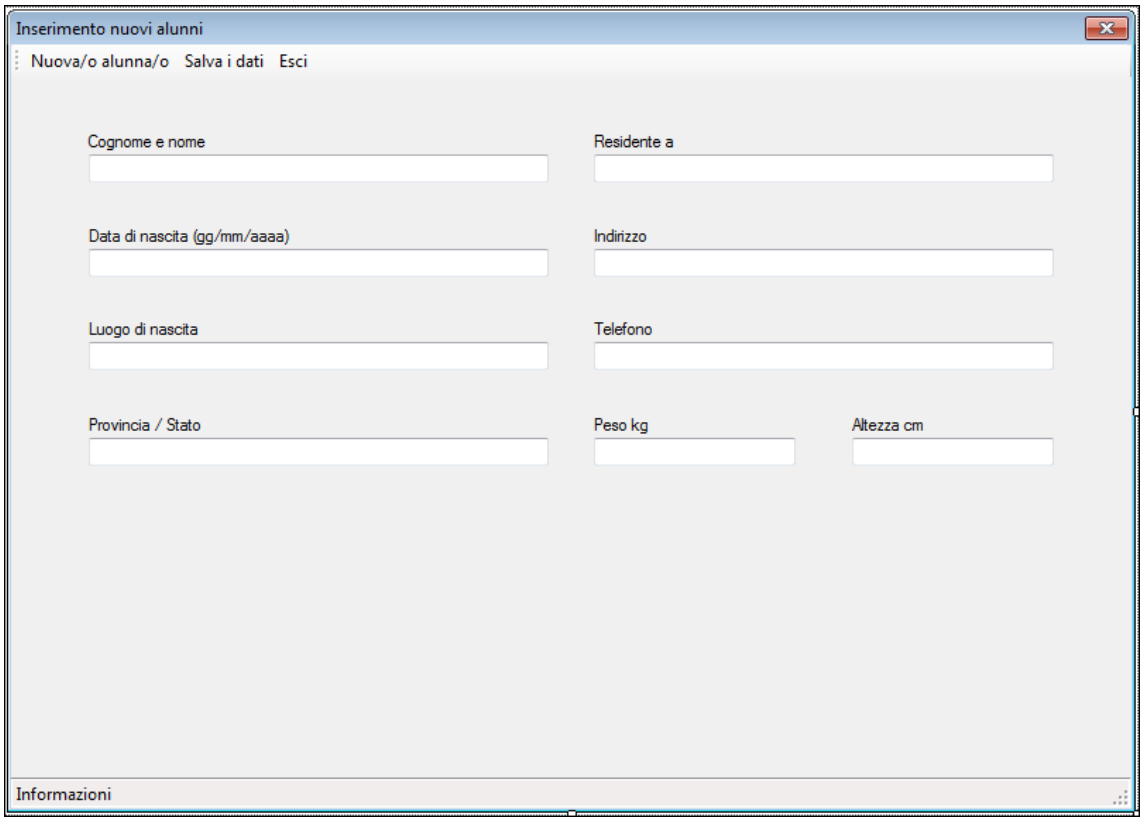


Figura 318: Il frmInserimenti con i suoi oggetti e componenti.

Con un doppio *clic* sul pulsante **Esci**, accediamo alla **Finestra del Codice**. Qui creiamo le variabili che saranno valide in tutto il codice e scriviamo la procedura per la chiusura del form.
La chiusura del form in realtà è gestita da due procedure, perché il form può essere chiuso in due modi diversi:

- con un *clic* sul pulsante **Esci** oppure
- con un *clic* **sull'icona in alto e destra** nella barra del form.

La procedura attivata con il *clic* sul pulsante **Esci** è **cmdEsci.Click**: questa procedura, prima di chiudere il form, controlla se ci sono dei dati da salvare e in caso affermativo, chiede all'utente se desidera salvarli.
La procedura attivata con un *clic* **sull'icona in alto e destra** nella barra del form è **Me.Closing**. Anche questa procedura, prima della effettiva chiusura del form, controlla se ci sono dei dati da salvare e in caso affermativo rimanda il programma alla procedura **cmdEsci.Click**.

```
Public Class frmInserimenti

    Dim TastoValido As Boolean = False
```

```
    Private Sub cmdEsci_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdEsci.Click

        If cmdSalva.Enabled = False Then
            ' Se il cmdSalva è disabilitato, non ci sono dati da salvare, per cui
il form può essere chiuso e la procedura termina qui:
            Me.Close()
            'libera la memoria occupata dal frmInserimenti:
            Me.Dispose()
            Exit Sub
        End If

        ' Altrimenti (il cmdSalva è abilitato) ci sono dati non ancora
salvati.da salvare.
        ' Il programma chiede all'utente se desidera salvarli o se vuole
annullare la chiusura del form:

        Select MsgBox("Vuoi salvare i dati?", MsgBoxStyle.YesNoCancel +
MsgBoxStyle.Question, "DATI NON ANCORA SALVATI")

            '... se l'utente vuole salvare i dati viene eseguito l'evento
clic sul pulsante che salva le informazioni
            Case MsgBoxResult.Yes
                cmdSalva.PerformClick()

                '... se l'utente annulla l'operazione la procedura termina
qui:
            Case MsgBoxResult.Cancel
                Exit Sub

                ' ... se l'utente non vuole salvare i dati, chiude il form
            Case MsgBoxResult.No
                cmdSalva.Enabled = False
                Me.Close()
                'libera la memoria occupata dal frmInserimenti:
                Me.Dispose()
        End Select

    End Sub
```

```
    Private Sub frmInserimenti_FormClosing(sender As Object, e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing

        ' se il cmdSalva è abilitato, significa che nel form ci sono dei dati da
salvare e prima di procedere si chiede conferma all'utente:

        If cmdSalva.Enabled = True Then
            cmdEsci.PerformClick()
            e.Cancel = True
        End If

    End Sub

End Class
```

Scriviamo ora la procedura con i comandi connessi all'evento **frmInserimenti_Load**, all'apertura di questo form.
Questa procedura prepara i **TextBox** per le operazioni di inserimento di nuovi dati, poi assegna un testo all'etichetta **lblInformazioni.**

```
    Private Sub frmInserimenti_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' cerca tutti i controlli TextBox contenuti in questo form:
    For Each CasellaDiTesto As Control In Me.Controls
        ' per ogni TextBox:
        If TypeOf CasellaDiTesto Is TextBox Then
            ' collega l'evento della pressione di un tasto su una casella di
testo alla procedura PressioneTasto:
            AddHandler CasellaDiTesto.KeyDown, AddressOf PressioneTasto
        End If
    Next

    ' porta il cursore sul primo TextBox:
    txtCognomeNome.Focus()

    ' Visualizza le informazioni nella label in basso a sinistra:
    lblInformazioni.TextAlign = ContentAlignment.TopLeft
    lblInformazioni.Text = "Premi INVIO per confermare i dati" & vbCrLf & "La
data di nascita deve essere scritta nel formato gg/mm/aaaa (10 caratteri)."

    ' disabilita il pulsante Salva i dati, perché non vi sono ancora dati da
salvare:
    cmdSalva.Enabled = False

End Sub
```

Quando si digita un carattere all'interno di uno dei **TextBox**, la procedura **PressioneTasto** verifica se è stato premuto il tasto **INVIO;** in questo caso passa il **focus** al **TextBox** successivo, seguendo l'ordine della proprietà **TabIndex**.
Per i **TextBox** destinati a contenere numeri, la routine esegue una verifica supplementare, per impedire che l'utente vi scriva caratteri alfabetici:

```
Private Sub PressioneTasto(sender As Object, e As
System.Windows.Forms.KeyEventArgs)

    ' Se è stato premuto il tasto INVIO...
    If e.KeyCode = 13 Then
        ' ...non si aggiunge alcun carattere al TextBox
        e.Handled = True
        ' e viene simulata la pressione del tasto TABulatore per passare al
TextBox successivo
        SendKeys.Send("{TAB}")
    End If

    ' Se l'utente ha iniziato a immettere dei dati, abilita il pulsante per il
salvataggio dei dati:
    If txtCognomeNome.Text <> "" Then cmdSalva.Enabled = True

        ' inizia il controllo del contenuto dei TextBox 'numerici':
```

```
        Dim NumeroTextBox As Integer = sender.TabIndex

        ' la proprietà TabIndex dei tre TextBox oggetto di questo controllo
porta i numeri 6, 7 e 8.

        Select Case NumeroTextBox
            ' se nei TextBox telefono peso e altezza sono scritti solo
numeri, la variabile NumeroValido assume il valore VERO:
            Case 6, 7, 8
                Select Case e.KeyCode
                    Case 47 To 58, 96 To 105
                        TastoValido = True
                    Case Keys.Back 'permette la cancellazione dell'operazione
                        TastoValido = True
                    Case Else
                        TastoValido = False
                End Select
        End Select

    End Sub
```

```
Private Sub txtTelefono_KeyPress(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs) Handles txtTelefono.KeyPress,
txtPeso.KeyPress, txtAltezza.KeyPress

        ' Questa procedura gestisce la scrittura del carattere corrispondente
al tasto premuto
        ' in uno dei tre TextBox numerici:
        ' se la variabile TastoValido = False, la procedura annulla la
scrittura del tasto premuto,
        ' proseguendo come se il carattere fosse già stato scritto:

        If TastoValido = False Then e.Handled = True

End Sub
```

Completiamo il codice di questo form con le procedure che gestiscono gli eventi clic sugli altri due pulsanti: Nuova/o alunna/o e Salva i dati.

```
Private Sub cmdInserisci_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdInserisci.Click

    For Each CasellaDiTesto As Control In Me.Controls
        ' ricerca i TextBox nel form e ne cancella il contenuto:
        If TypeOf CasellaDiTesto Is TextBox Then CasellaDiTesto.Text = ""
    Next

    ' Disbilita questi pulsanti:
    cmdInserisci.Enabled = False
    cmdSalva.Enabled = False

    ' Porta il cursore sul primo TextBox:
    txtCognomeNome.Focus()

End Sub
```

La procedura per il salvataggio dei dati contiene una serie di operazioni preliminari finalizzate a controllare il contenuto dei **TextBox**.
In particolare, la procedura effettua un controllo approfondito della data, perché questa non può essere inserita nel database se non è scritta dall'utente in modo corretto (**gg/mm/aaaa**).
Il contenuto del TextBox **txtDataDiNascita** viene sottoposto a questi controlli:

- la data deve essere formata da 10 caratteri complessivi;
- la barra "/" deve trovarsi nella data in terza e sesta posizione;
- la data deve essere una data valida (non può contenere lettere);
- il numero del giorno deve essere inferiore a 32;
- il numero del mese deve essere inferiore a 13;
- l'anno di nascita indicato deve essere distante di almeno 3 anni dall'anno corrente (l'alunno non può avere meno di 3 anni);
- l'anno di nascita non può essere distante più di 100 anni dall'anno corrente (l'alunno non può avere più di 100 anni).

```
Private Sub cmdSalva_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdSalva.Click

    ' Preparazione del salvataggio dei dati:
    ' vengono cancellati eventuali spazi vuoti all'inizio o alla fine del testo
nei TextBox e
    ' i dati vengono trascritti in lettere maiuscole:
    txtCognomeNome.Text = txtCognomeNome.Text.Trim.ToUpper
    txtLuogoDiNascita.Text = txtLuogoDiNascita.Text.Trim.ToUpper
    txtResidenza.Text = txtResidenza.Text.Trim.ToUpper
    txtIndirizzo.Text = txtIndirizzo.Text.Trim.ToUpper
    txtProvinciaDiNascita.Text = txtProvinciaDiNascita.Text.Trim.ToUpper
    txtTelefono.Text = txtTelefono.Text.Trim

    ' Si assegna il numero 0 ai campi Peso e Altezza, se questi sono vuoti:
    If txtPeso.Text.Trim = "" Then txtPeso.Text = "0"
    If txtAltezza.Text.Trim = "" Then txtAltezza.Text = "0"

        ' ***Inizia il controllo del formato e della validità della data.

        ' elimina eventuali spazi vuoti iniziali o finali nella data:
        txtDataDiNascita.Text = txtDataDiNascita.Text.Trim
        Dim DataProvvisoria As String = txtDataDiNascita.Text
        ' controlla la lunghezza della data:
        If DataProvvisoria.Length <> 10 Then GoTo Correzione
        ' controlla la presenza delle barre di separazione delle parti della
data:
        If DataProvvisoria.Substring(2, 1) <> "/" Or DataProvvisoria.Substring(5,
1) <> "/" Then GoTo Correzione
        ' controlla se il testo digitato corrisponda a una data, senza la
presenza di caratteri estranei:
        On Error GoTo Correzione
        Nascita = CDate(DataProvvisoria)
        ' controlla se il giorno immesso è inferiore a 32
        If Nascita.Day > 31 Then GoTo correzione
        ' controlla se il mese immesso è inferiore a 13
        If Nascita.Month > 12 Then GoTo correzione
```

```
        ' controlla che l'anno di nascita scritto dall'utente non sia troppo
vicino all'anno corrente...
        If Nascita.Year > (Date.Today.Year - 3) Then GoTo Correzione
        ' ... o troppo lontano:
        If Nascita.Year < (Date.Today.Year - 100) Then GoTo Correzione

        ' *** Termina il controllo della data

        ' Messaggio di conferma dei dati, prima del loro salvataggio:
        Dim TestoDiConferma As String = txtCognomeNome.Text & vbCrLf
        TestoDiConferma &= "Data di nascita: " & txtDataDiNascita.Text & vbCrLf
        TestoDiConferma &= "Luogo di nascita: " & txtLuogoDiNascita.Text & vbCrLf
        TestoDiConferma &= "Provincia di nascita: " & txtProvinciaDiNascita.Text
& vbCrLf
        TestoDiConferma &= "Residenza: " & txtResidenza.Text & vbCrLf
        TestoDiConferma &= "Indirizzo: " & txtIndirizzo.Text & vbCrLf
        TestoDiConferma &= "Telefono: " & txtTelefono.Text & vbCrLf
        TestoDiConferma &= "Peso: kg " & txtPeso.Text.ToString & vbCrLf
        TestoDiConferma &= "Altezza: cm " & txtAltezza.Text.ToString

        If MsgBox(TestoDiConferma, MessageBoxButtons.YesNo +
MessageBoxIcon.Question, "CONFERMI QUESTI DATI?") = MsgBoxResult.No Then
            ' Se l'utente non conferma i dati, esci della procedura di
salvataggio:
            Exit Sub
        End If

        ' Se l'utente conferma i dati, si avvia la query InserimentoDati,
indicando i valori da inserire in ogni parametro della query:
        frmPrincipale.AlunniTableAdapter.InserimentoDati(txtCognomeNome.Text,
txtDataDiNascita.Text, txtLuogoDiNascita.Text, txtResidenza.Text,
txtIndirizzo.Text, txtProvinciaDiNascita.Text, txtTelefono.Text, txtPeso.Text,
txtAltezza.Text)

    ' La query Fill aggiorna la tabella nel form principale con i dati
salvati nel DataSet,
    ' e più precisamente salvati nella tabella Alunni:
    frmPrincipale.AlunniTableAdapter.Fill(frmPrincipale.AnagrafeDataSet.Alunni)

    ' Disabilita il pulsante per il salvataggio dei dati:
    cmdSalva.Enabled = False

    ' Uscita dalla procedura, le righe che seguono riguardano solo la correzione
della data:
    Exit Sub

Correzione:

    MsgBox("La data non è valida.", MsgBoxStyle.OkOnly + MsgBoxStyle.Critical)
    txtDataDiNascita.Focus()

End Sub
```

Il codice del **frmInserimenti** è così finito.
Mandiamo in esecuzione il programma per controllarne il funzionamento e iniziamo a inserire alcuni dati nel database:

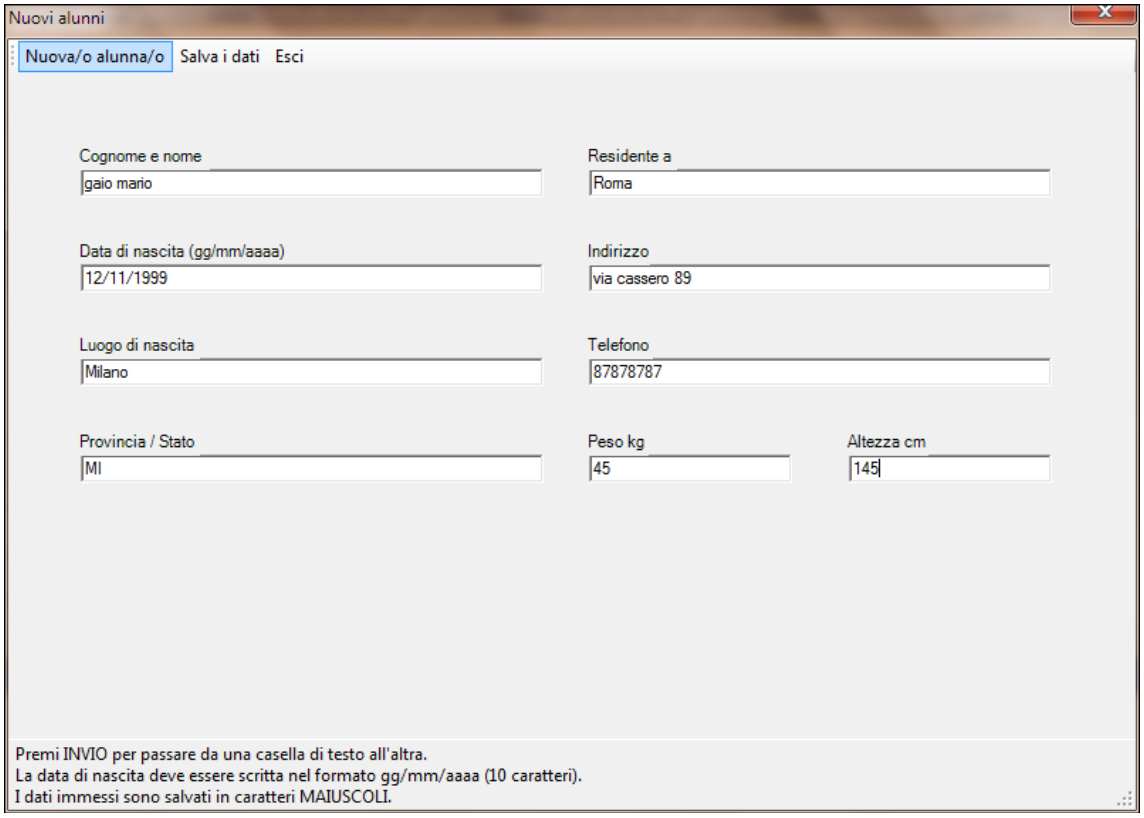


Figura 319: L'inserimento di nuovi dati.

Salviamo i dati inseriti e chiudiamo il form.

Torniamo così al form principale, dove troviamo in tabella i dati inseriti.
Eventuali compleanni nel giorno corrente sono segnalati in giallo:

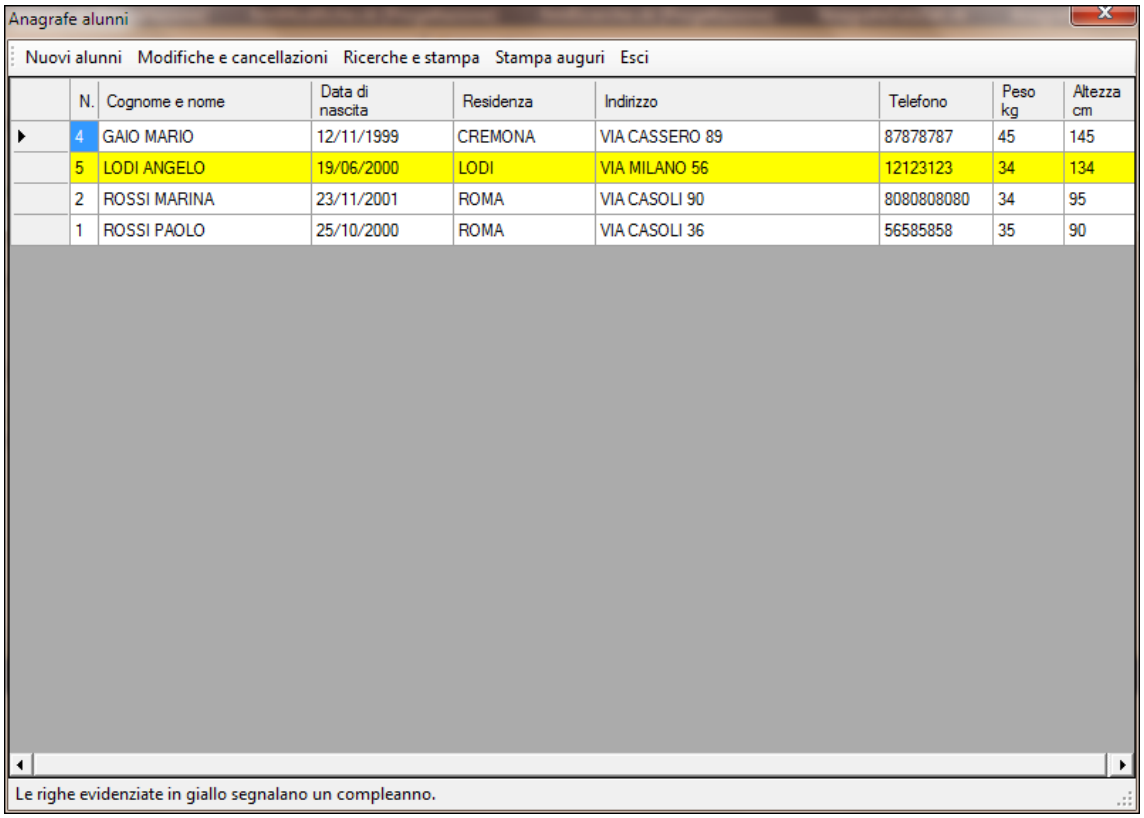


Figura 320: Un database avviato, con i dati relativi a quattro alunni.

205: La finestra per la modifica e la cancellazione di dati.

Il **frmModifiche** è dedicato alla modifica (o correzione) dei dati e alla cancellazione di schede diventate inutili o obsolete; esso consentirà dunque di tenere costantemente aggiornate le informazioni custodite nel nostro archivio.
Nella finestra **Esplora soluzioni** facciamo un doppio *clic* sul form **frmModifiche.vb**.

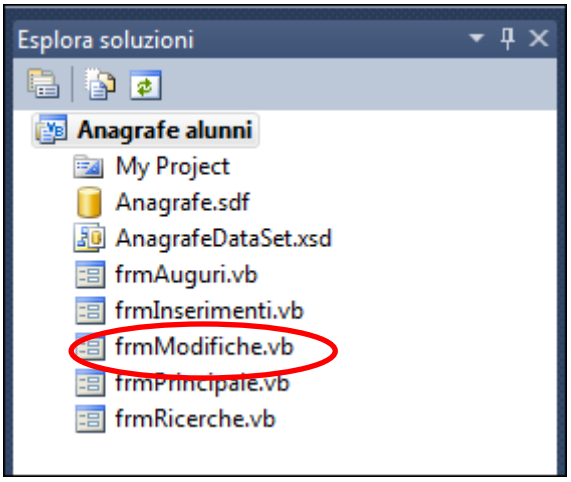


Figura 321: Apertura del frmModifiche.

Questa è la finestra che l'utente userà per modificare o cancellare i dati degli alunni presenti nell'archivio.
Impostiamo le proprietà del form in questo modo:

FormBorderStyle = FixedSingle	Imposta la visualizzazione tradizionale del form e della sua barra in alto.
MaximizeBox = False	L'utente non potrà allargare le dimensioni del form.
MinimizeBox = False	L'utente non potrà ridurre le dimensioni del form.
Showicon = False	Nessuna icona nella barra del testo del form, in alto a sinistra.
ShowInTaskbar = False	Nessuna icona, per questo programma, nella Barra delle applicazioni.
Size = 400; 500	Dimensioni del form.
StartPosition = CenterScreen	Colloca il form al centro dello schermo.

Text = Modifica o cancellazione dati	Questo è il testo che compare nella barra del form, in alto a sinistra.

Ora preleviamo dalla **Casella degli Strumenti** e inseriamo nel form un componente **ToolStrip1**, destinato a contenere un solo pulsante Button: il **cmdEsci**.
Proprietà del pulsante:

- **(Name) = cmdEsci**
- **DisplayStyle = Text**
- **Text = Esci**

Inseriamo nel form un componente **StatusStrip,** destinato a contenere una sola etichetta con informazioni sull'uso del programma. Proprietà della label:

- **(Name) = lblIstruzioni**
- **Text = Istruzioni**

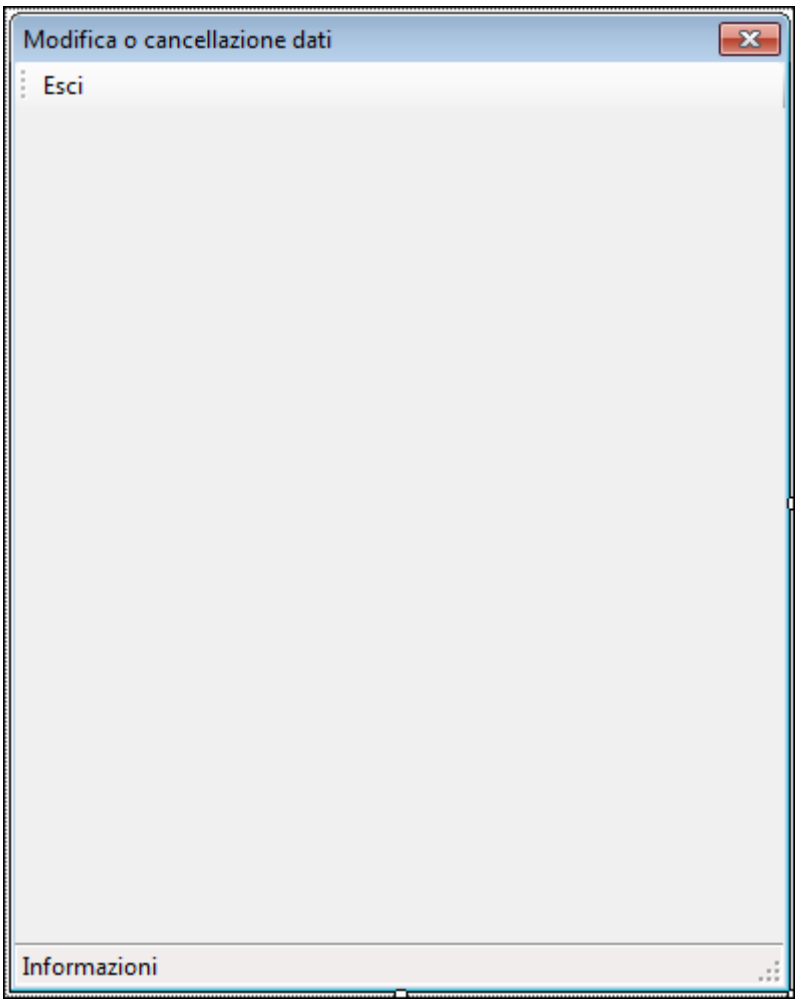


Figura 322: Il frmModifiche con i componenti ToolStrip e StatusStrip.

Torniamo a visualizzare il pannello **Origini dati** (se questo non è più disponibile tra le finestre del progetto facciamo un *clic* sul menu **Dati / Mostra origini dati**):

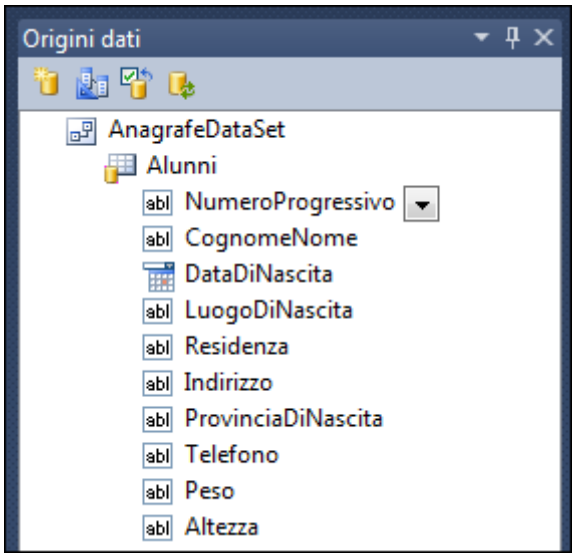


Figura 323: Il pannello Origini Dati.

Con un *clic* sul campo **NumeroProgressivo**, si apre un menu a tendina in cui vediamo l'elenco dei controlli ai quali questo campo può essere associato.
In questo caso lo associamo a un controllo Label, perché il testo di una Label non è modificabile dall'utente (ricordiamo che il numero progressivo degli alunni è assegnato in modo automatico dal **database** e non può essere modificato dall'utente):

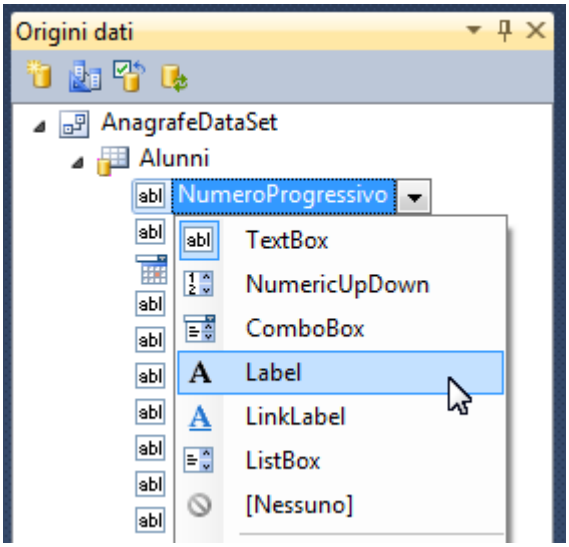


Figura 324: L'associazione del campo NumeroProgressivo a un controllo Label.

Procediamo con il secondo campo, **CognomeNome**, da associare a un **TextBox**:

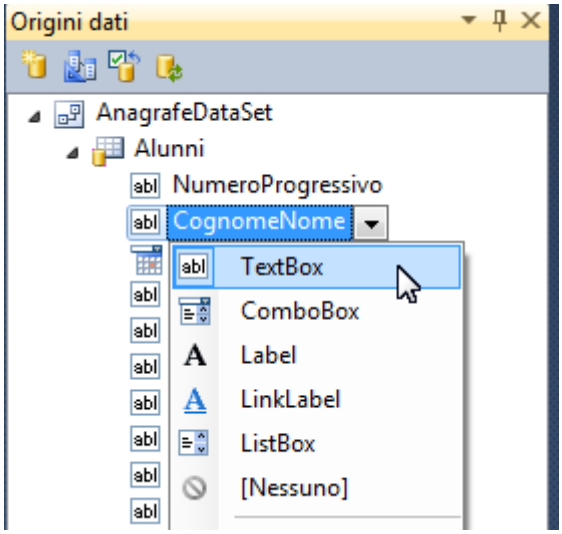


Figura 325: L'associazione del campo CognomeNome a un controllo TextBox.

Il Campo **DataDiNascita** va associato a un controllo **DateTimePicker**:

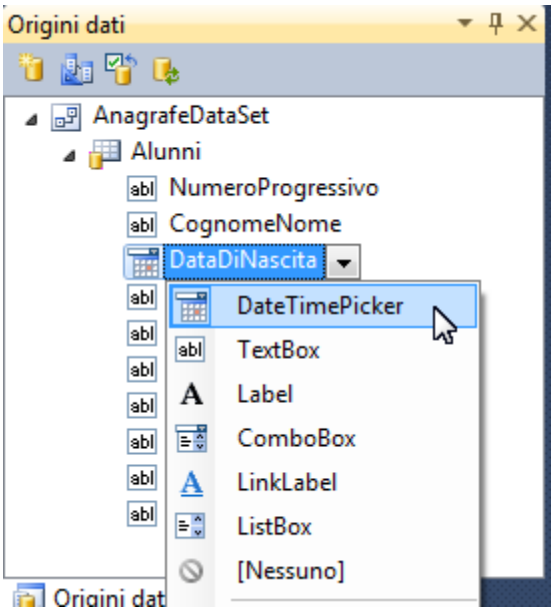


Figura 326: L'associazione del campo DataDiNascita a un controllo DateTimePicker.

Completiamo le associazioni dei campi, associando tutti i campi successivi a controlli **TextBox**.
Al termine delle associazioni dei campi ai controlli, avremo dunque otto campi associati a caselle di testo, un campo associato a una label e un campo associa a un controllo DateTimePicker:

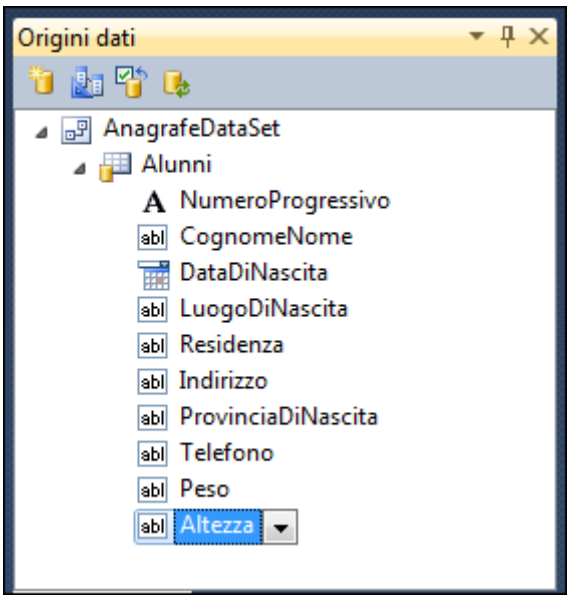


Figura 327: Completamento delle associazioni dei campi ai controlli.

Iniziamo ora le operazioni di trasferimento all'interno del **frmModifiche** dei campi associati ai controlli, trasferimento che va fatto campo per campo, un campo alla volta.

Facciamo un *clic* sul primo campo, **NumeroProgressivo**, nel pannello **Origini Dati**. Tenendo il pulsante sinistro del mouse premuto, trasciniamo questo campo all'interno del **frmModifiche** e quindi rilasciamo il pulsante del mouse:

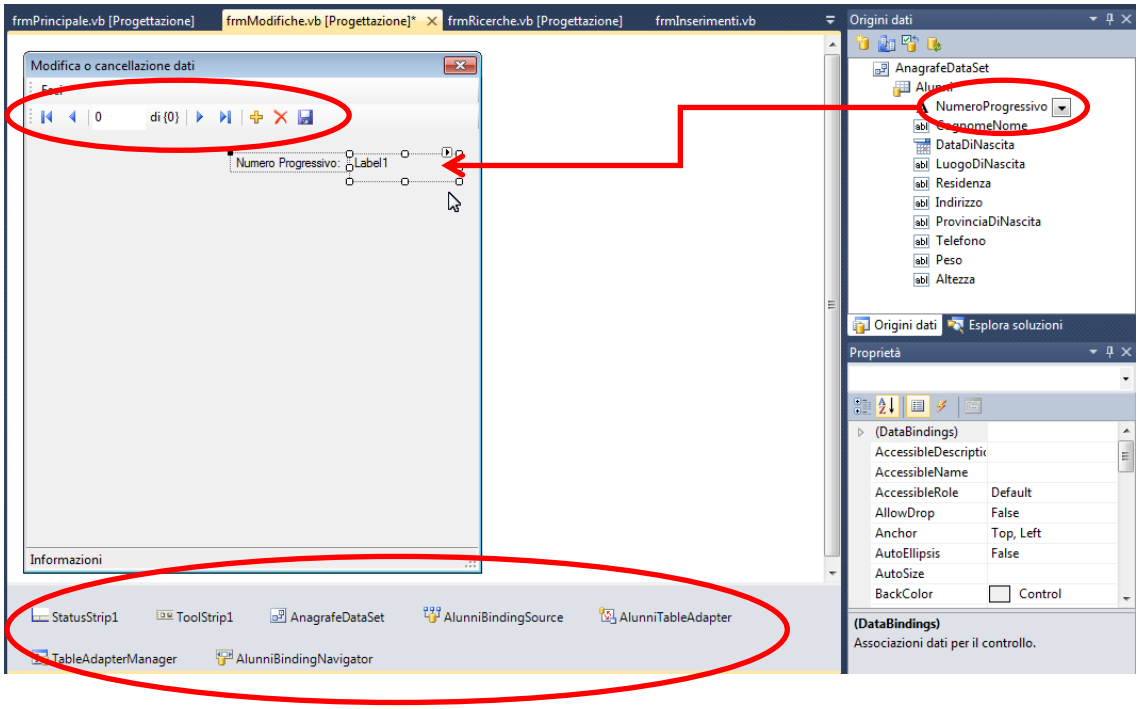


Figura 328: Completamento delle associazioni dei campi ai controlli.

Notiamo che nella parte esterna del **frmModifiche**, in basso, vediamo ora inseriti nel form alcuni nuovi componenti.
Tra di essi troviamo il componente **AlunniBindingNavigator**: si tratta della barra che troviamo in alto, subito sotto l'intestazione del form: questo è lo strumento che ci permetterà di navigare tra i dati del database, scorrendo le schede degli alunni una a una, per apportarvi eventuali modifiche o per cancellarle.
In esso vediamo:

- le frecce per scorrere i dati da una scheda all'altra;
- il pulsante per cancellare i dati di un alunno;
- il pulsante per salvare le modifiche.

Nel componente **AlunniBindingNavigator** è presente anche il pulsante **BindingNavigatorAddNewItem**, per aggiungere nuove schede al database.
Siccome per l'inserimento di nuove schede abbiamo già progettato il frmInserimenti, questo pulsante è superfluo e può anzi essere fonte di problemi nel funzionamento del programma, per cui lo nascondiamo cliccandolo con il mouse e impostando la sua proprietà **Visible = False**.

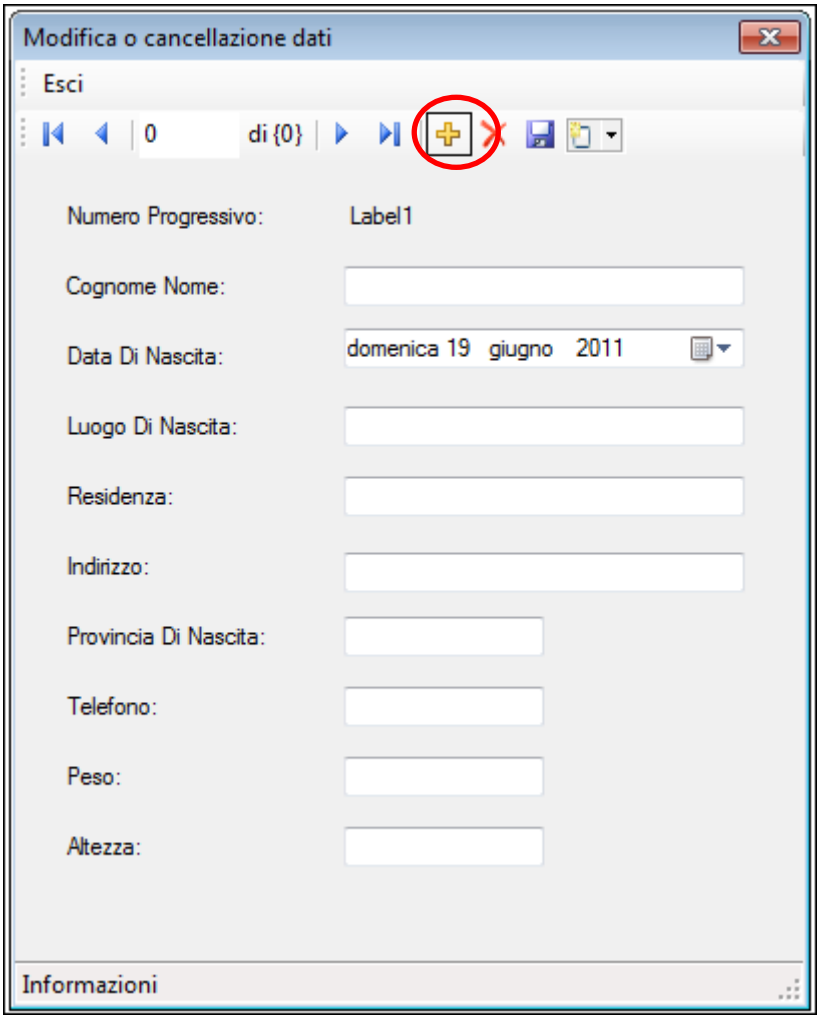


Figura 329: Il pulsante BindingNavigatorAddNewItem, da eliminare.

Dopo avere trascinato all'interno del **frmModifiche** il campo **NumeroProgressivo**, procediamo allo stesso modo sistemando nel form tutti gli altri campi.
Sistemiamo in ordine, su due colonne, i controlli che vediamo nel form, con le etichette incolonnate a sinistra e i controlli dedicati alla visualizzazione dei dati incolonnati a destra:

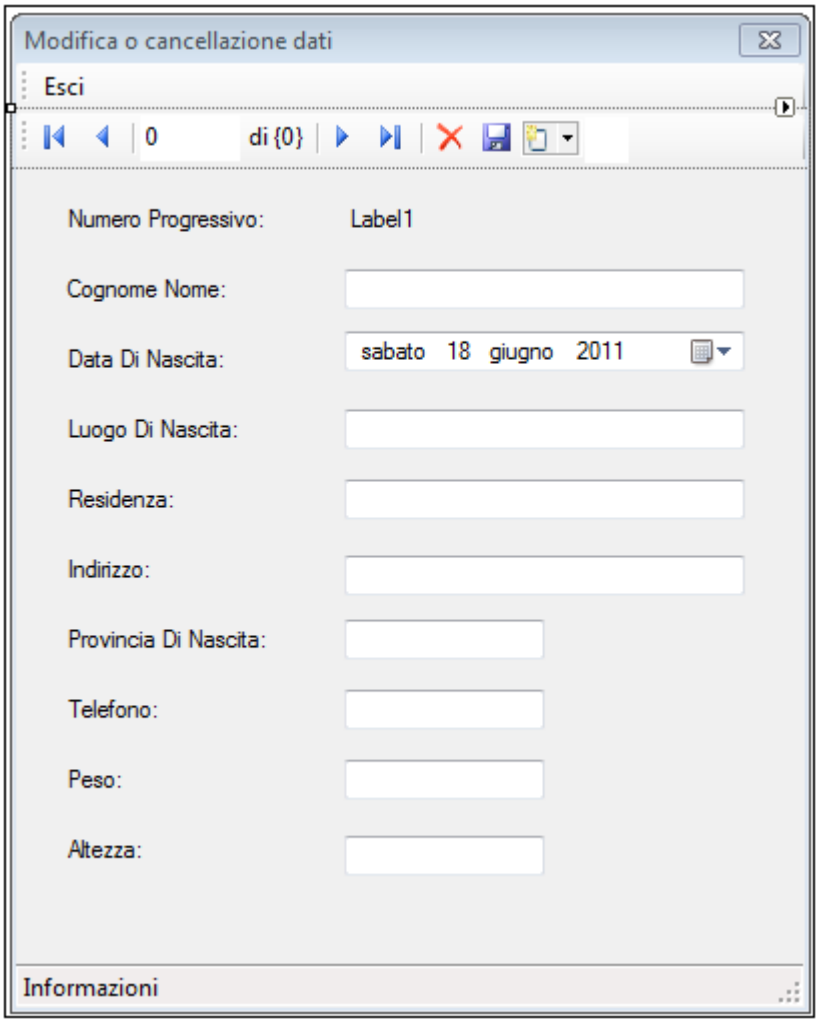


Figura 330: Completamento del frmModifche con i campi associati ai controlli.

Non modifichiamo in alcun modo la proprietà **Name** assegnata in modo automatico ai vari controlli, perché modificando i nomi dei controlli si perderebbe la loro associazione ai campi nella tabella dell'archivio.

Iniziamo a scrivere il codice di questo form partendo dai comandi di chiusura.
Anche questo form può essere chiuso in due modi diversi:

- con un *clic* sul pulsante **Esci** oppure
- con un *clic* **sull'icona in alto e destra** nella barra del form.

La procedura attivata con il *clic* sul pulsante **Esci** è **cmdEsci.Click**: questa procedura, prima di chiudere il form, controlla se ci sono dei dati da salvare e in caso affermativo, chiede all'utente se desidera salvarli.
La procedura attivata con un *clic* **sull'icona in alto e destra** nella barra del form è **Me.Closing**. Anche questa procedura, prima dell'effettiva chiusura del form, controlla se ci sono dei dati da salvare e in caso affermativo rimanda il programma alla procedura **cmdEsci.Click**.

Facciamo un doppio *clic* sul pulsante **cmdEsci** e accediamo alla **Finestra del Codice**.
Qui troviamo già impostate in modo automatico alcune procedure - da non modificare - dedicate alla gestione dei dati con i componenti presenti nel form.
Inseriamo nella parte iniziale del codice la dichiarazione di una variabile valida per tutto il form, e queste due procedure:

```
Public Class frmModifiche

    Dim TastoValido As Boolean = False
```

```
    Private Sub cmdEsci_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdEsci.Click

        ' Verifica lo stato del pulsante per il salvataggio dei dati:
        ' se il pulsante è abilitato, allora sono state operate delle modifiche
ai dati,
        ' per cui viene simulato un clic sullo stesso pulsante di salvataggio
        If AlunniBindingNavigatorSaveItem.Enabled = True Then
            AlunniBindingNavigatorSaveItem.PerformClick()
            ' A salvataggio effettuato, il pulsante viene disabilitato
            AlunniBindingNavigatorSaveItem.Enabled = False
        End If

        'la query Fill riempe la tabella della finestra principale con i nuovi
dati modificati

frmPrincipale.AlunniTableAdapter.Fill(frmPrincipale.AnagrafeDataSet.Alunni)
        Me.Close()
        Me.Dispose()

    End Sub
```

```
    Private Sub frmInserimenti_FormClosing(sender As Object, e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing

        ' Se il pulsante di salvataggio dati  è abilitato, nel form ci sono dei
dati non salvati, per cui
        ' prima di procedere alla chiusura del form si chiede conferma
all'utente:

        If AlunniBindingNavigatorSaveItem.Enabled = True Then
```

```
            cmdEsci.PerformClick()
            e.Cancel = True
        End If

    End Sub
```

Ora passiamo alla procedura che gestisce l'apertura del **frmModifiche**. La troviamo già impostata nella **Finestra del Codice**; la integriamo in questo modo:

```
    Private Sub frmModifiche_Load(sender As System.Object, e As System.EventArgs)
Handles MyBase.Load

    ' Visualizza le informazioni nella label in basso a sinistra:
    lblInformazioni.TextAlign = ContentAlignment.TopLeft
    lblInformazioni.Text = "Premi le frecce nella barra in alto per scorrere le
schede degli alunni."

    'cerca tutti i controlli TextBox presenti nel form:
    For Each CasellaDiTesto As Control In Me.Controls
        If TypeOf CasellaDiTesto Is TextBox Then
            ' collega l'evento della pressione di un tasto su una casella di
testo alla procedura PressioneTasto:
            AddHandler CasellaDiTesto.KeyDown, AddressOf PressioneTasto
            ' collega l'evento del cambiamento dei dati in una casella di testo
alla procedura CambiamentoTesto:
            AddHandler CasellaDiTesto.KeyPress, AddressOf CambiamentoTesto
        End If
    Next

    ' Scegliamo di visualizzare la data nel formato breve gg/mm/aaaa:
    DataDiNascitaDateTimePicker.Format = DateTimePickerFormat.Short
    'questa riga di codice, inserita automaticamente da VB, carica i dati nella
tabella AnagrafeDataSet.alunni
    Me.AlunniTableAdapter.Fill(Me.AnagrafeDataSet.Alunni)

    ' In apertura del form, non vi sono dati da salvare, per cui il pulsante per
il salvataggio dei dati è disabilitato.
    AlunniBindingNavigatorSaveItem.Enabled = False

End Sub
```

Rimangono da scrivere le istruzioni per le procedure **PressioneTasto** e **CambiamentoTesto**, che abbiamo collegato ai **TextBox** presenti nel form.
Quando l'utente preme un tasto all'interno di un **TextBox** e ne modifica il contenuto, sono necessarie alcune operazioni:

- bisogna controllare che sia presente il dato **CognomeNome**, imprescindibile per il funzionamento del programma;
- bisogna controllare che nei TextBox con numeri (**Telefono**, **Peso**, **Altezza**) siano scritti solo numeri;
- il contenuto del **TextBox** deve essere trascritto in caratteri MAIUSCOLI;
- il pulsante **cmdSalva** deve essere attivato, perché ci sono dei nuovi dati da salvare.

Tali operazioni sono affidate a queste due procedure:

```
Sub PressioneTasto(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs)

        ' Analizza il contenuto dei TextBox 'numerici':

        Select Case sender.name

            Case "TelefonoTextBox", "AltezzaTextBox", "PesoTextBox"
                ' controlla che questi TextBox contengano solo numeri:

                Select Case e.KeyCode
                    Case 47 To 58, 96 To 105 ' codici dei tasti con i numeri
sulla tastiera e sul tastierino numerico
                        TastoValido = True
                    Case Keys.Back ' permette la cancellazione a ritroso
                        TastoValido = True
                    Case Else
                        TastoValido = False
                End Select

        End Select

    End Sub
```

```
Private Sub CambiamentoTesto(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs)

        ' Questa procedura gestisce la scrittura del carattere corrispondente dei
tasti premuti nei TextBox:

        Select Case sender.name

            Case "TelefonoTextBox", "AltezzaTextBox", "PesoTextBox"
                ' nei TextBox 'numerici', se la variabile NumeroValido = False,
la procedura annulla la scrittura del tasto premuto,
                ' proseguendo come se il carattere fosse già stato scritto:

                If TastoValido = False Then
                    e.Handled = True
                    Exit Sub
                End If

            Case Else
                ' altrimenti scrivi il carattere in maiuscolo:
                e.KeyChar = UCase(e.KeyChar)
        End Select

        AlunniBindingNavigatorSaveItem.Enabled = True

End Sub
```

Aggiungiamo una procedura per attivare il pulsante di salvataggio dei dati se in una scheda viene modificata la data di nascita:

```
    Private Sub DataDiNascitaDateTimePicker_ValueChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
DataDiNascitaDateTimePicker.ValueChanged
```

```
    ' abilita il pulsante per il salvataggio dei dati:
    AlunniBindingNavigatorSaveItem.Enabled = True

End Sub
```

Concludiamo il codice di questo form completando le procedure, già avviate automaticamente da VB, per il salvataggio delle modifiche e la cancellazione di schede:

```
Private Sub AlunniBindingNavigatorSaveItem_Click(sender As System.Object, e As
System.EventArgs) Handles AlunniBindingNavigatorSaveItem.Click

    ' Il campo CognomeNome è imprescindibile per il proseguimento del programma,
    ' per cui se la casella è vuota il pulsante di salvataggio dei dati è
disabilitato
    ' e la procedura termina qui:
    If CognomeNomeTextBox.Text = "" Then
        MsgBox("La casella 'Cognome e nome' non può essere vuota.",
MsgBoxStyle.OkOnly + MsgBoxStyle.Critical)
        Exit Sub
    End If

    AlunniBindingNavigatorSaveItem.Enabled = False
    Me.Validate()
    Me.AlunniBindingSource.EndEdit()
    Me.TableAdapterManager.UpdateAll(Me.AnagrafeDataSet)

End Sub
```

```
Private Sub BindingNavigatorDeleteItem_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BindingNavigatorDeleteItem.Click

    ' Dopo avere cancellato una scheda, abilita il pulsante per il salvataggio
dei dati:
    AlunniBindingNavigatorSaveItem.Enabled = True

End Sub
```

Ora possiamo mandare in esecuzione il programma per controllare, in particolare, il funzionamento di questa finestra per le modifiche dei dati:

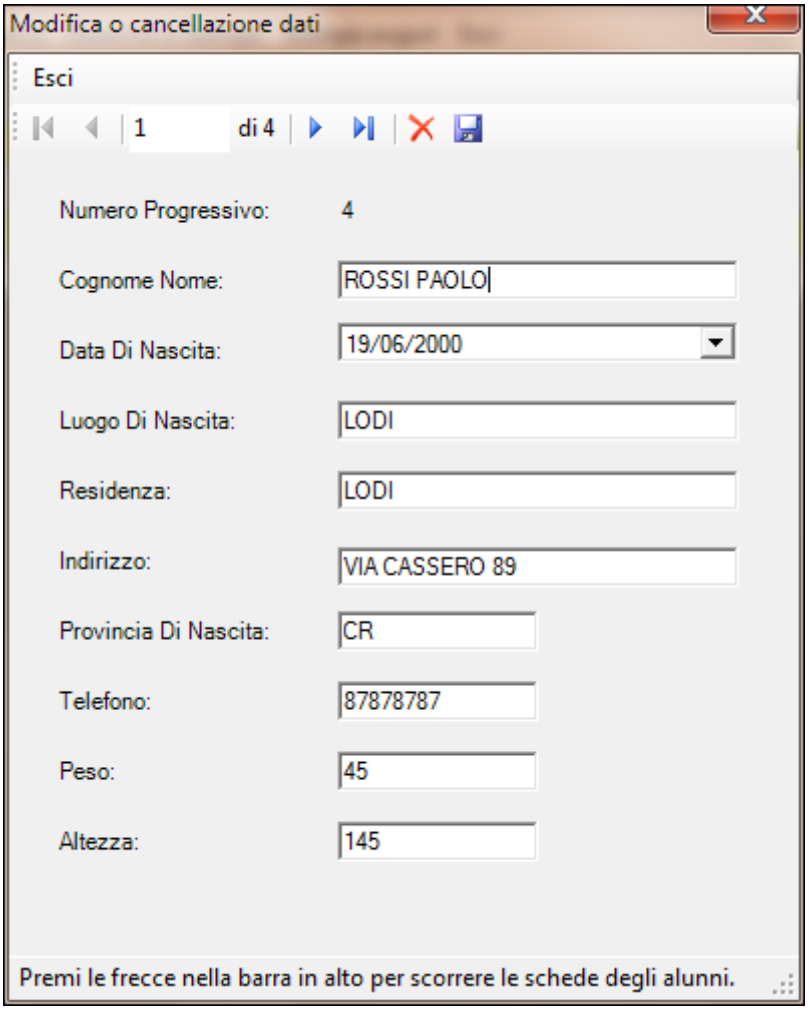


Figura 331: La modifica di dati già esistenti.

206: La finestra per la ricerca e la stampa di dati.

Nella finestra **Esplora soluzioni** facciamo un doppio *clic* su form **frmRicerche.vb**.

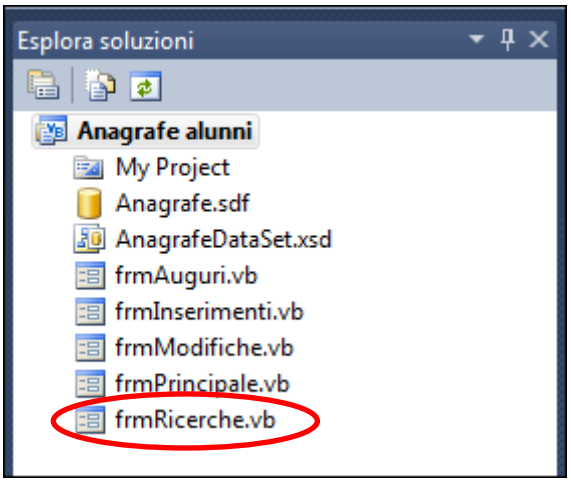


Figura 332: Apertura del frmRicerche.

Ogni archivio di dati svolge una doppia funzione: organizzare e conservare le informazioni in strutture definite, e rendere disponibili i dati raccolti per interrogazioni fondate su uno o più criteri specifici.
Questa è la finestra che l'utente userà per effettuare ricerche, tra i dati dell'archivio, ed eventualmente per stampare i risultati di queste ricerche.
Nel nostro caso l'utente potrà interrogare l'archivio scegliendo uno di questi criteri di selezione dei dati:

- un nome o alcune lettere da cercare nel campo Cognome e nome;
- la provincia o lo stato di nascita;
- il peso (tutti gli alunni con peso maggiore di…);
- l'altezza (tutti gli alunni con altezza maggiore di…).

Una tabella nel form visualizzerà i nomi selezionati in base al criterio scelto di volta in volta, mostrando, per ogni alunno, il nome, l'indirizzo e il numero di telefono.
Due pulsanti, infine, permetteranno la stampa su carta dell'elenco risultato dalla ricerca oppure di tutti i dati presenti nel database.

Iniziamo impostando le proprietà del **frmRicerche** nel modo indicato di seguito:

FormBorderStyle = FixedSingle	Imposta la visualizzazione tradizionale del form e della sua barra in alto.
MaximizeBox = False	L'utente non potrà allargare le dimensioni del form.
MinimizeBox = False	L'utente non potrà ridurre le dimensioni del form.

Showicon = False	Nessuna icona nella barra del testo del form, in alto a sinistra.
ShowInTaskbar = False	Nessuna icona, per questo programma, nella Barra delle applicazioni.
Size = 800; 570	Dimensioni del form.
StartPosition = CenterScreen	Colloca il form al centro dello schermo.
Text = Ricerca e stampa di dati	Questo è il testo che compare nella barra del form, in alto a sinistra.

Inseriamo nel form questi componenti:

- Un componente **ToolStrip** nel quale inseriamo un pulsante **DropDownButton** e tre pulsanti **Button**.
  Con un *clic* su ognuno di questi quattro controlli, ne impostiamo le proprietà:

Pulsante	**DropDownButton1**	**Button1**	**Button2**	**Button3**
(Name)	cmdAvviaRicerca	cmdStampaRicerca	cmdStampaTutto	cmdEsci
AutoSize	False	True	True	True
DisplayStyle	Text	Text	Text	Text
Text	Avvia una ricerca…	Stampa la ricerca	Stampa tutti i dati	Esci
Size	200; 25			

- Inseriamo un componente **StatusStrip,** destinato a contenere solo una etichetta con informazioni sull'uso del programma. Proprietà di questa label:
  (Name) = lblIstruzioni
  Text = Istruzioni

Ecco come si presenta il form dopo l'inserimento di questi oggetti:

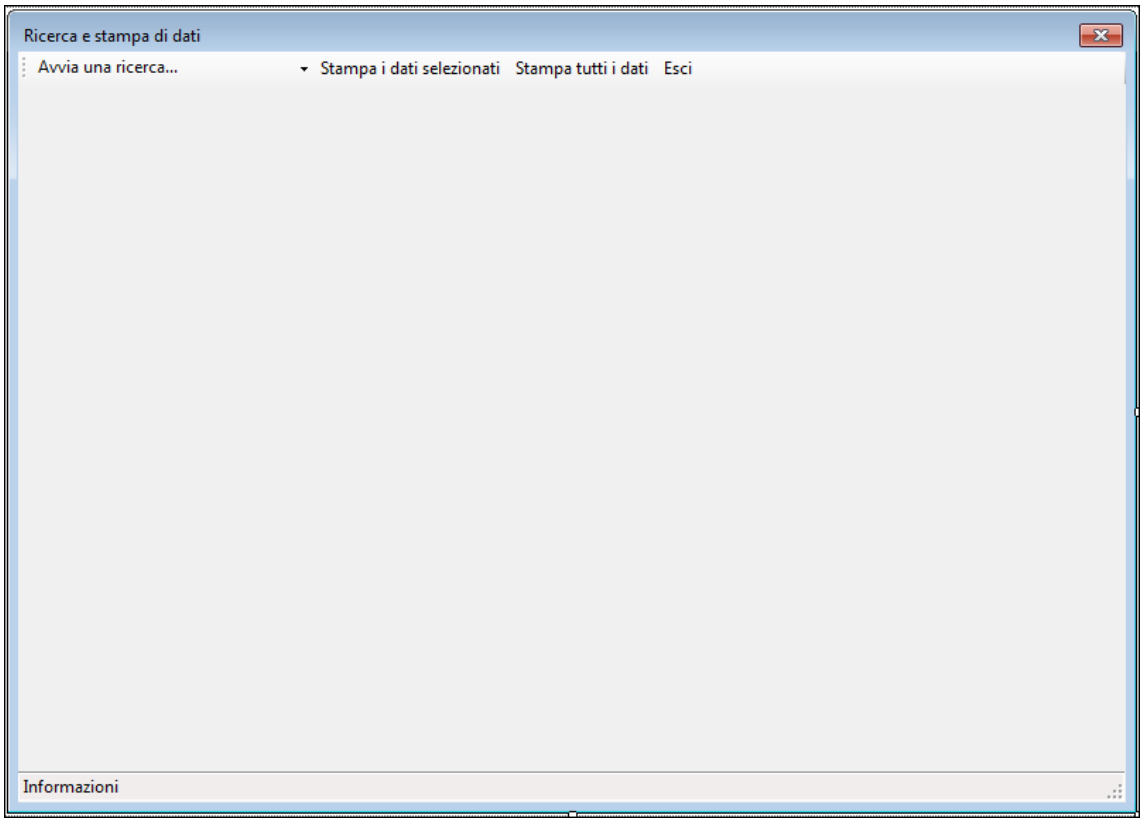


Figura 333: Il frmRicerche con i componenti ToolStrip e StatusStrip.

Il **DropDownButton** al quale abbiamo dato il nome **cmdAvviaRicerca** è un pulsante con un menu a tendina: in questo menu dobbiamo inserire quattro voci, per le quattro modalità di ricerca che abbiamo previsto: per nome, per provincia di nascita, per peso, per altezza.

Facciamo un *clic* sulla freccia nera a destra di questo pulsante e nella casella **Digitare qui** scriviamo la prima voce:
per nome o per lettere nel nome
Con un *clic* sul tasto INVIO passiamo alla successiva casella **Digitare qui**. Scriviamo in successione queste altre voci:
secondo la provincia di nascita
secondo il peso (maggiore di…)
secondo l'altezza (maggiore di…)

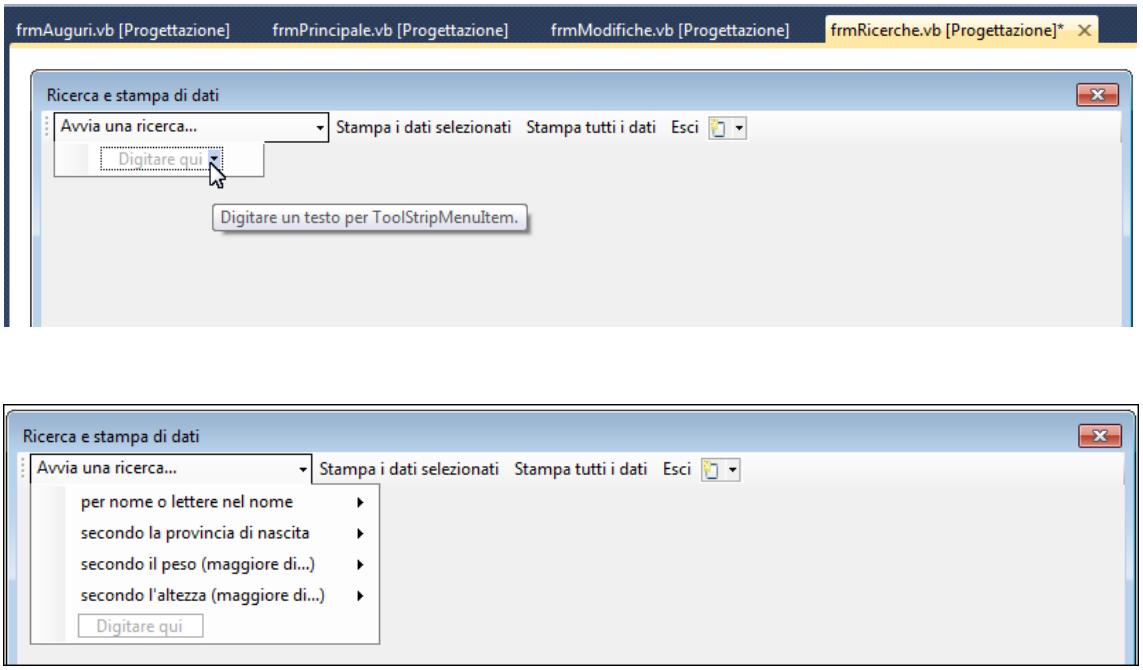


Figura 334: Impostazione dei menu del pulsante 'Avvia una ricerca…'.

L'utente del programma, quando vorrà eseguire una ricerca, dovrà dunque cliccare il pulsante **Avvia una ricerca…**, dovrà cliccare una delle quattro voci per scegliere un criterio di ricerca e dovrà naturalmente **scrivere** il parametro in base al quale il programma effettuerà la ricerca.
Questo parametro di ricerca verrà scritto in quattro **TextBox**, uno per ogni voce del menu, che ora inseriremo a fianco di queste voci.

Notiamo che a destra di ognuna delle voci del menu si trova una freccia nera, mediante la quale si accede alla possibilità di collegare ogni voce a un controllo.
Associamo alla prima voce '**per nome o per lettere nel nome'** un controllo **TextBox** al quale assegniamo queste proprietà:
(Name) = txtNome
BackColor = Yellow
Size = 50; 23
Tag = 0

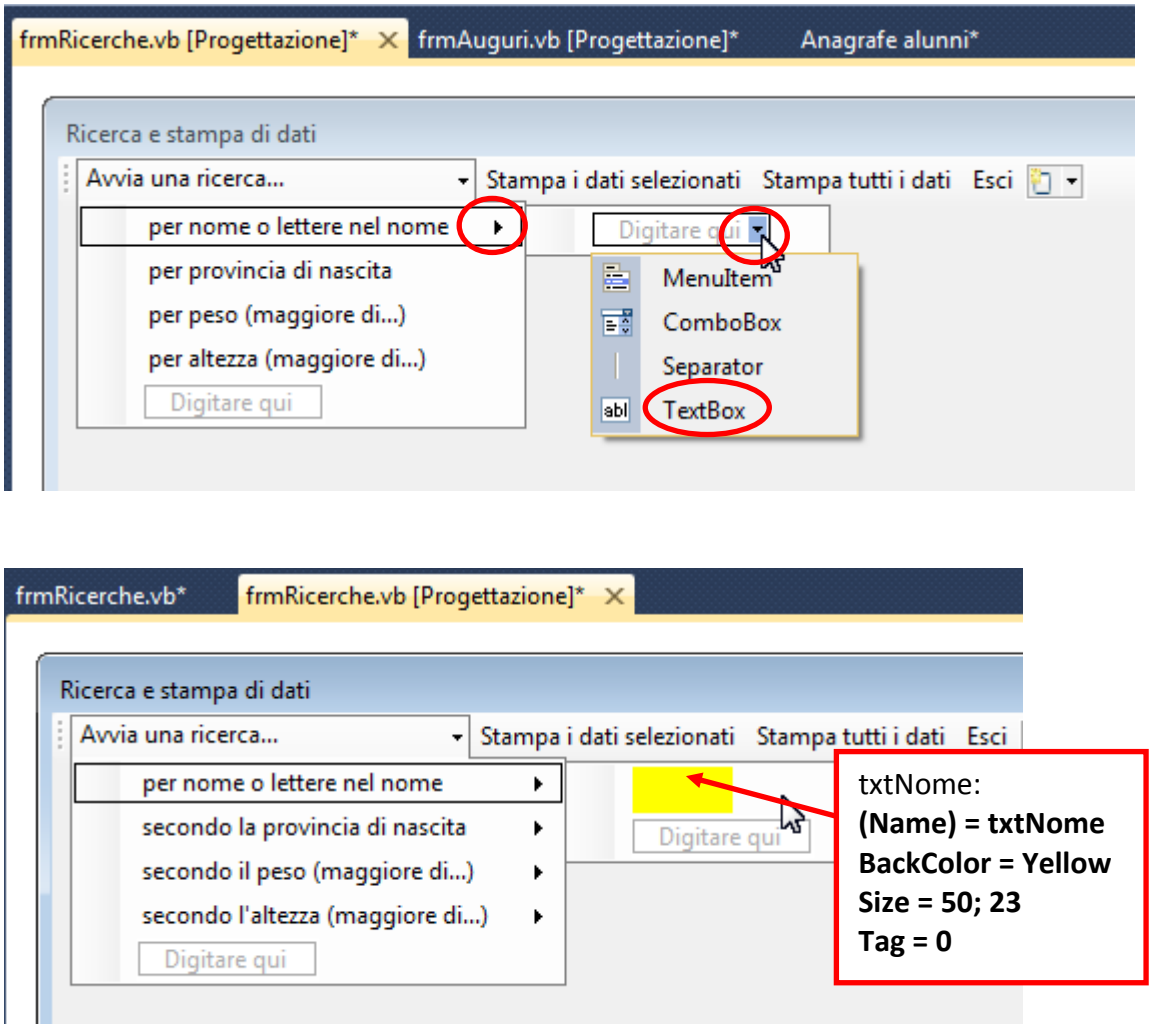


Figura 335: Associazione di un txtNome al primo criterio di ricerca.

Procediamo allo stesso modo con le altre tre voci, collegando ognuna di esse a un controllo **TextBox**.
Impostiamo le proprietà di questi TextBox come segue:

Menu	**Proprietà del TextBox corrispondente:**
secondo la provincia di nascita	**(Name) = txtNome** **BackColor = Yellow** **Size = 50; 23** **Tag = 1**
secondo il peso (maggiore di…)	**(Name) = txtPeso** **BackColor = Yellow** **Size = 50; 23** **Tag = 2**
secondo l'altezza (maggiore di…)	**(Name) = txtAltezza** **BackColor = Yellow** **Size = 50; 23** **Tag = 3**

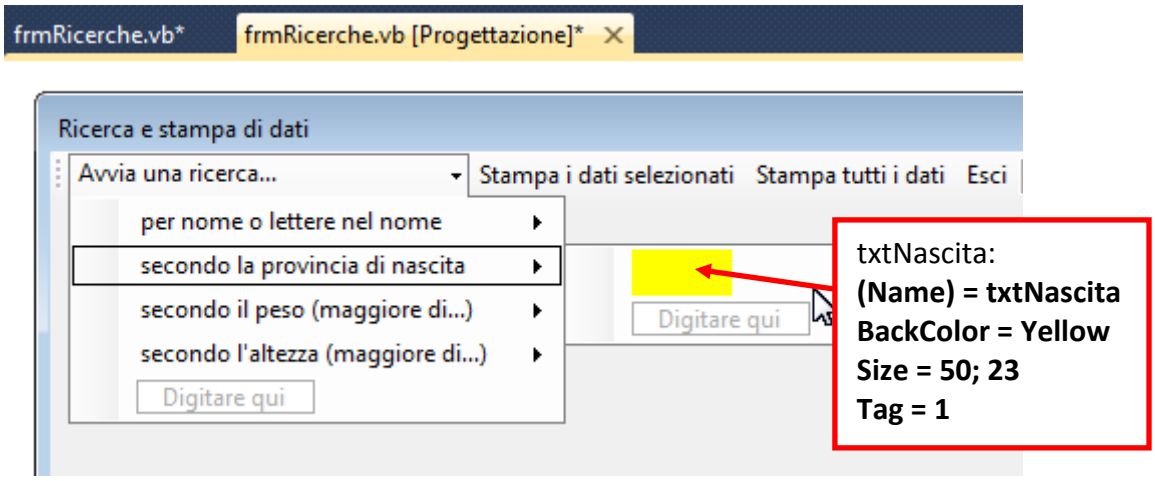


Figura 336: Associazione del TextBox txtNascita al secondo criterio di ricerca.

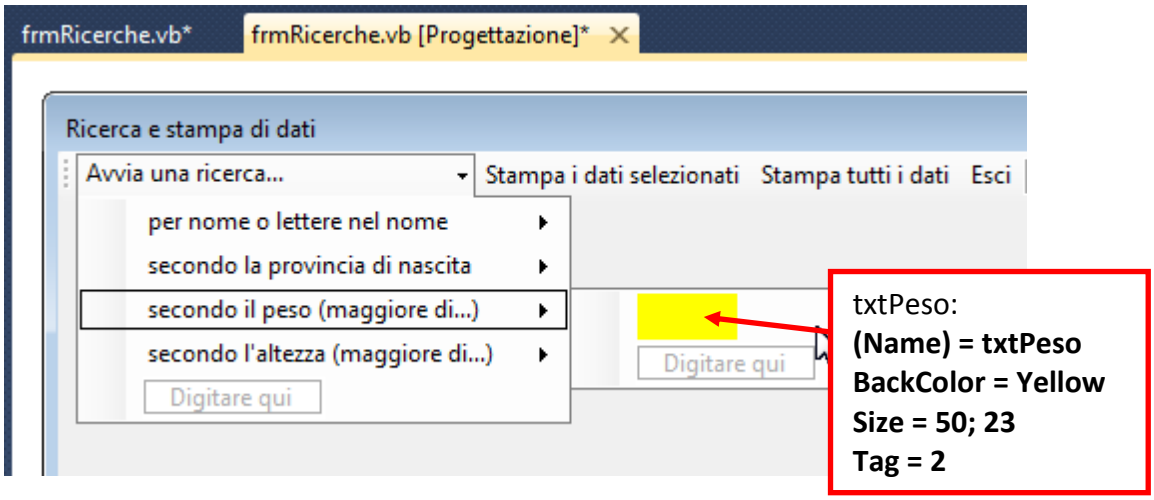


Figura 337: Associazione del TextBox txtPeso al terzo criterio di ricerca.

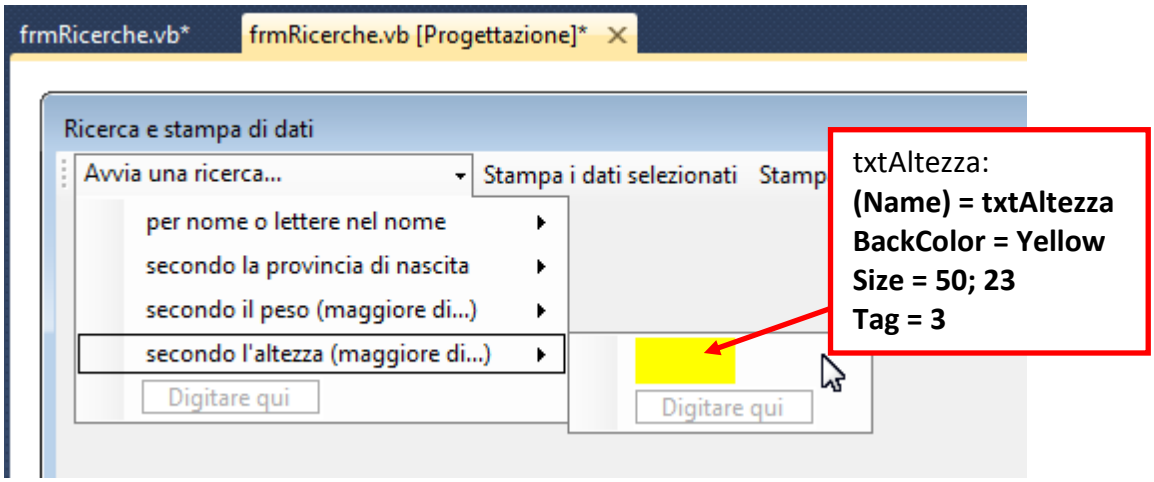


Figura 338: Associazione del TextBox txtAltezza all'ultimo criterio di ricerca.

Ora, per completare questo **frmRicerche**, dobbiamo inserirvi una tabella nella quale saranno visualizzate le informazioni sui gruppi di alunni selezionati secondo i criteri di ricerca.
Per la creazione e l'adattamento grafico di questa tavola procediamo come abbiamo già fatto per il form **Principale**.

Apriamo il pannello **Origini dati**. Se questo non è visibile, facciamo un *clic* nella barra dei menu di VB sul menu **Dati / Mostra origini** dati.
Trasciniamo la tabella **Alunni** dal pannello **Origini dati** nella finestra del **frmPrincipale:**

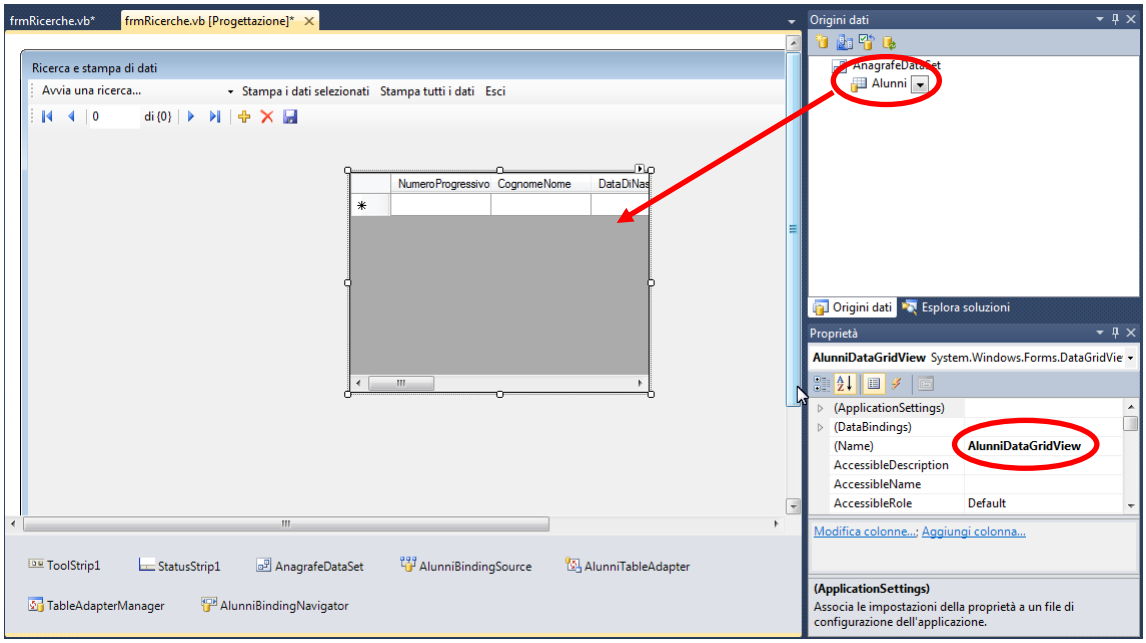


Figura 339: Trascinamento della tabella Alunni nel frmRicerche.

Notiamo, nella **Finestra Proprietà**, che VB ha assegnato automaticamente a questa tavola il nome **AlunniDataGridView** (= visualizzazione della griglia dei dati degli alunni).
Notiamo, nello spazio sottostante il **frmPrincipale**, che VB vi ha inserito automaticamente altri componenti, oltre al **ToolStrip1** e allo **StatusStrip** che erano già presenti:

- **AnagrafeDataSet** (= sistemazione di dati);
- **AlunniBindingSource** (= fonte per il collegamento dei dati);
- **AlunniTableAdapter** (= adattatore della tabella **Alunni**);
- **TableAdapterManager** (= gestione della tabella Alunni);
- **AlunniBindingNavigator** (= strumento di visualizzazione dei dati uno a uno).

L'ultimo componente, **AlunniBindingNavigator**, è lo strumento che serve per scorrere le schede del database una a una; in questo form non è necessario, per cui lo eliminiamo da questa finestra con un *clic* del tasto destro del mouse e poi un *clic* sul menu **Elimina**:

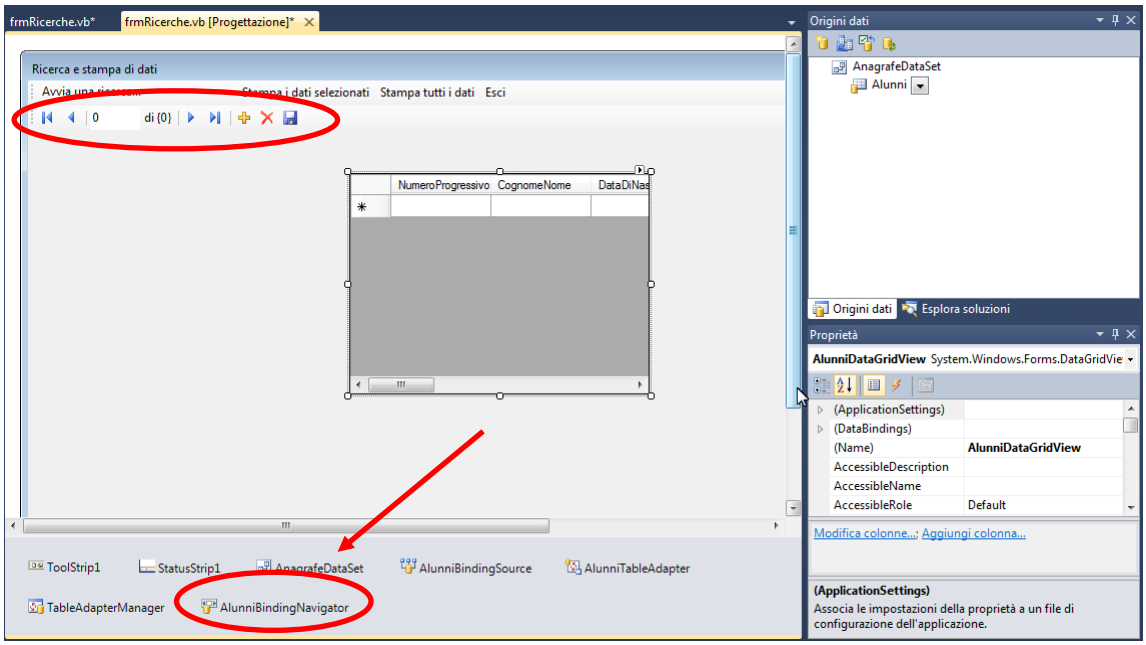


Figura 340: Eliminazione dal frmRicerche del componente AlunniBindingNavigator.

Ora impostiamo alcune proprietà della griglia **AlunniDataGridView** appena inserita nel form.
Facciamo un *clic* sulla griglia, poi un altro *clic* sul bordo, sulla freccina nera che compare in alto a destra.
Apriamo così il menu **Attività di DataGridView**, dove disattiviamo le voci

- Abilita aggiunta,
- Abilita modifica,
- Abilita eliminazione.

Attiviamo invece

- Abilita riordinamento colonne

e facciamo un *clic* sul comando **Ancora nel contenitore padre**, per dimensionare la griglia in modo che essa occupi tutto lo spazio disponibile nel form.

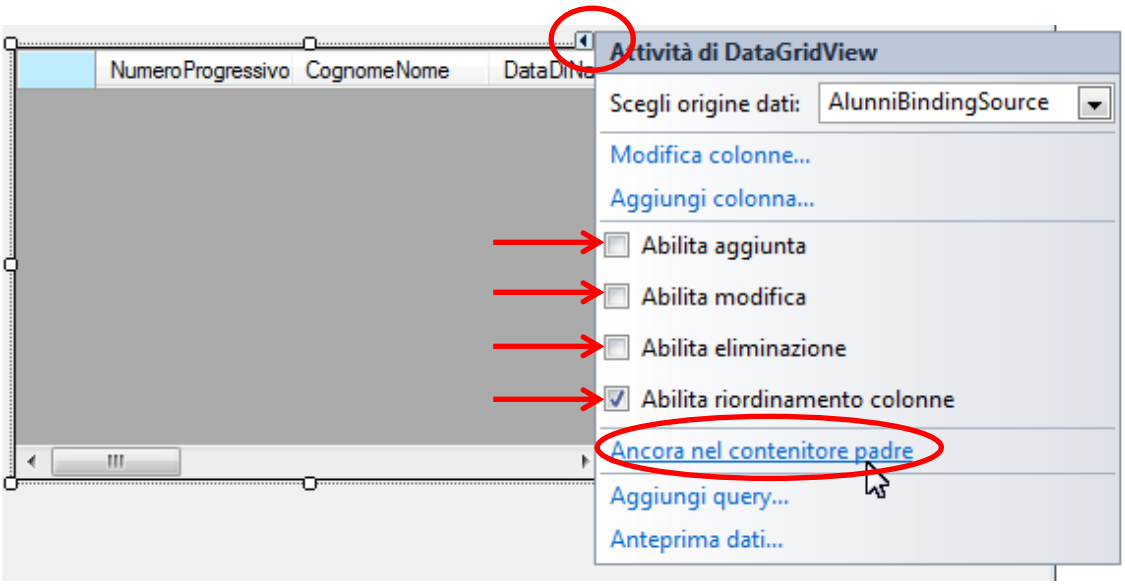


Figura 341: Impostazione delle attività della griglia AlunniDataGridView.

Con queste impostazioni, i dati visualizzati nella griglia saranno al riparo da interventi accidentali da parte dell'utente del programma: questi non potrà scrivere e modificare i dati direttamente nella griglia; avrà solo la facoltà di ordinare le schede in base al contenuto delle colonne.

Ora, nello stesso menu delle **Attività di DataGridView**, facciamo un *clic* sulla voce **Modifica colonne**… e accediamo alla finestra con le impostazioni di visualizzazione delle colonne:

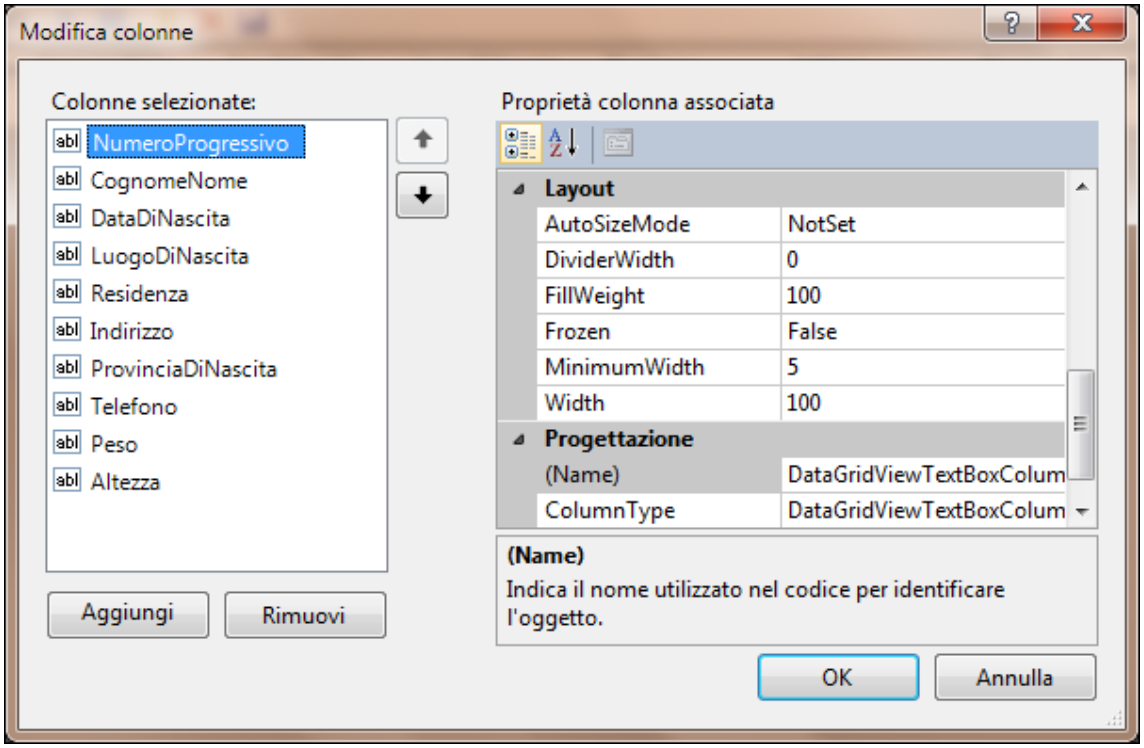


Figura 342: La finestra Modifica colonne di DataGridView.

Quando l'utente effettuerà una ricerca nell'archivio di dati, nella griglia **DataGridView** verranno visualizzati i nomi degli alunni rispondenti al criterio di selezione.
Non è necessario, perciò, che in questa griglia compaiano tutti i dati di una scheda/alunno, possiamo limitarci, ad esempio, a visualizzare il nome, l'indirizzo e il telefono, per facilitare un ipotetico contatto diretto degli alunni selezionati.
In questa finestra **Modifica colonne** rimuoviamo le colonne non necessarie facendo un *clic* sul pulsante **Rimuovi** (i dati delle colonne rimosse dalla griglia continueranno comunque a essere presenti e aggiornati nell'archivio).
Lasciamo visibili solo quattro colonne, come nell'immagine seguente:

- **CognomeNome**
- **Residenza**
- **Indirizzo**
- **Telefono**

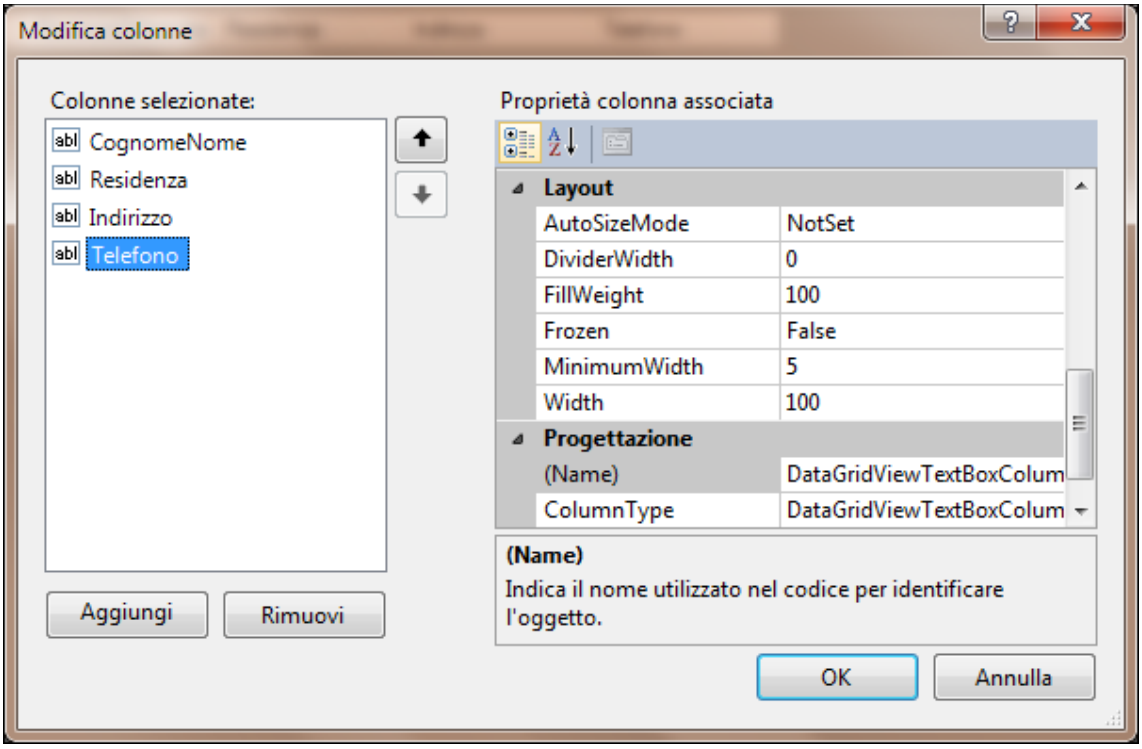


Figura 343: Selezione delle colonne della griglia AlunniDataGridView.

L'elenco che compare nella parte destra della finestra (**Proprietà colonna associata**), ci permette di impostare alcune proprietà delle colonne, allo scopo di ottimizzare la visibilità delle informazioni contenute nella tavola.
Per ciascuna colonna, modificheremo queste proprietà:

- **HeaderText** (è la didascalìa di intestazione della colonna);
- **ReadOnly** (indica se l'utente può modificare o meno il contenuto della colonna);
- **Width** (impostazione della larghezza della colonna.

Colonna	HeaderText	ReadOnly	Width
CognomeNome	Alunna/o	True	150
Residenza	Residente a	True	100
Indirizzo	Indirizzo	True	200
Telefono	Tel.	True	80

Al termine, confermiamo tutte le impostazioni con un *clic* sul pulsante OK.

Abbiamo così completato l'interfaccia del frmRicerche:

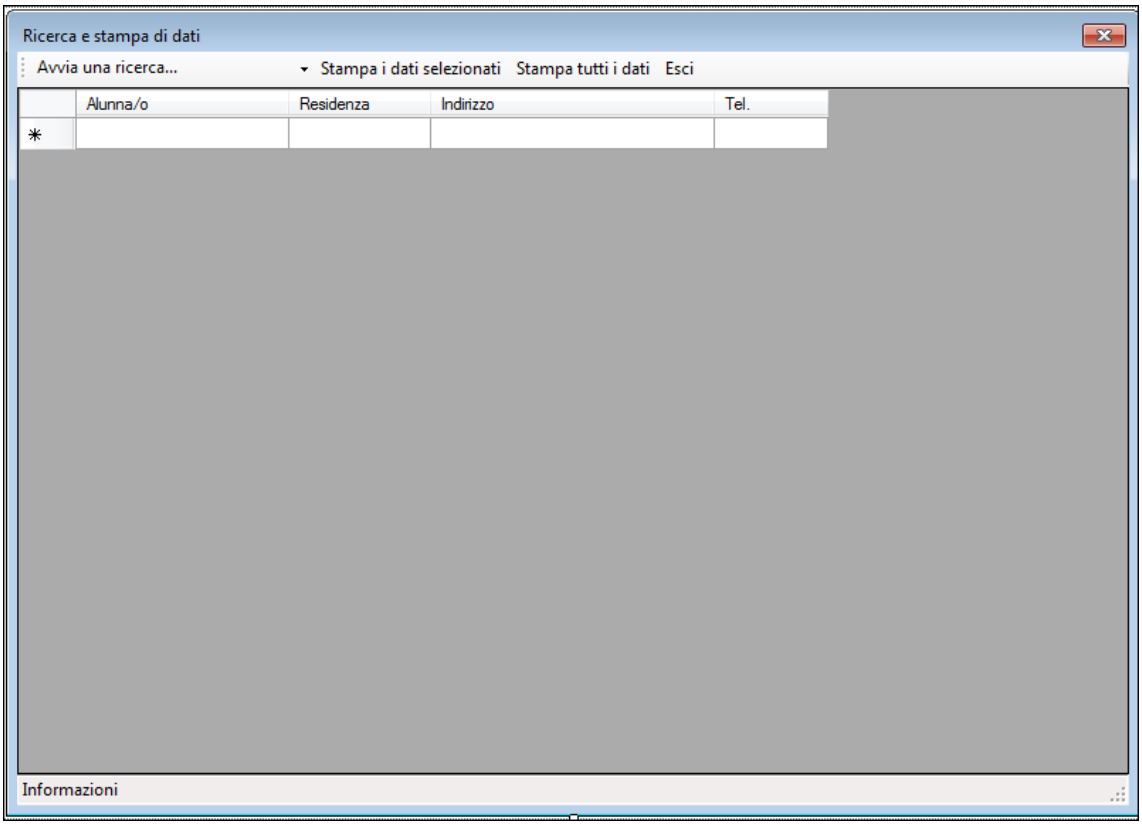


Figura 344: L'interfaccia del frmRicerche.

Iniziamo la scrittura del codice impostando due variabili che verranno utilizzate in seguito, e scriviamo la procedura per la chiusura del form.
Facciamo un doppio *clic* sul pulsante **cmdEsci** e accediamo alla **Finestra del Codice**.
In questo **frmRicerche** non abbiamo l'esigenza controllare se vi sono o meno dati da salvare, per cui è sufficiente scrivere questa procedura:

```
Public Class frmRicerche

    Dim Titolo As String = ""
    Dim NumeroValido As Boolean = False
    Dim ElencoDaStampare As List(Of String)

    Private Sub cmdEsci_Click() Handles cmdEsci.Click

        ' Chiude il form e libera l'area di memoria da esso occupata nel
computer:
        Me.Close()
        Me.Dispose()

    End Sub

End Class
```

Passiamo alla procedura che gestisce il caricamento del form: essa visualizza nella tabella tutti i dati disponibili nel database:

```
    Private Sub frmRicerche_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Me.Load

    cmdStampaSelezione.Enabled = False
    lblInformazioni.Text = "Per effettuare una ricerca, premi il tasto 'Seleziona
i dati in base a...', nella barra in alto."

    Titolo = "ELENCO DEGLI ALUNNI"
    Me.Text = Titolo

    ' Questa riga visualizza nella tabella tutti i dati disponibili:
    Me.AlunniTableAdapter.Fill(Me.AnagrafeDataSet.Alunni)

End Sub
```

Scriviamo ora nel codice la serie delle procedure che gestiscono gli eventi del mouse sul pulsante **cmdAvviaRicerca** e sulle voci del menu a tendina che discendono da questo pulsante:

```
Private Sub cmdAvviaRicerca_MouseDown(sender As Object, e As
System.Windows.Forms.MouseEventArgs) Handles cmdAvviaRicerca.MouseDown

    ' Quando il mouse si abbassa su questo pulsante, ripulisci le quattro
caselle di testo:
    txtNome.Text = ""
    txtNascita.Text = ""
    txtPeso.Text = ""
    txtAltezza.Text = ""

End Sub
```

```
Private Sub cmdAvviaRicerca_Mouseleave(sender As Object, e As
System.EventArgs) Handles cmdAvviaRicerca.MouseLeave

    ' Quando il mouse esce da questo pulsante, ripristina il messaggio
originale nella lblInformazioni:
    lblInformazioni.Text = "Per effettuare una ricerca, premi il tasto 'Avvia
una ricerca...', nella barra in alto."

End Sub
```

```
Private Sub CriterioNome_MouseEnter() Handles CriterioNome.MouseEnter

    lblInformazioni.Text = "Scrivi un nome o un gruppo di lettere, nella
casella grigia, poi premi il tasto INVIO."

End Sub
```

```
Private Sub CriterioNascita_mouseEnter() Handles CriterioNascita.MouseEnter

    lblInformazioni.Text = "Scrivi una provincia o uno stato, nella casella
grigia, poi premi il tasto INVIO."
```

```
End Sub
```

```
Private Sub CriterioPeso_mouseEnter() Handles CriterioPeso.MouseEnter

    lblInformazioni.Text = "Scrivi il peso minimo, in kg, nella casella
grigia, poi premi il tasto INVIO."

End Sub
```

```
Private Sub CriterioAltezza_mouseEnter() Handles CriterioAltezza.MouseEnter

    lblInformazioni.Text = "Scrivi l'altezza minima, in cm, nella casella
grigia, poi premi il tasto INVIO."

End Sub
```

```
Private Sub CriterioAltezza_MouseLeave() Handles CriterioAltezza.MouseLeave,
CriterioNascita.MouseLeave, CriterioNome.MouseLeave, CriterioPeso.MouseLeave

    lblInformazioni.Text = "Per effettuare una ricerca, premi il tasto 'Avvia
una ricerca...', nella barra in alto."

End Sub
```

```
Private Sub CriterioNome_Click() Handles CriterioNome.Click

    txtNome.Focus()

End Sub
```

```
Private Sub CriterioNascita_Click() Handles CriterioNascita.Click

    txtNascita.Focus()

End Sub
```

```
Private Sub CriterioPeso_Click() Handles CriterioPeso.Click

    txtPeso.Focus()

End Sub
```

```
Private Sub CriterioAltezza_Click() Handles CriterioAltezza.Click

    txtAltezza.Focus()

End Sub
```

Ed ecco le procedure che svolgono le funzioni più importanti all'interno di questo form: le procedure per la selezione dei dati e le procedure per la stampa dei dati.
La procedura per la selezione dei dati si attiva quando l'utente immette un parametro di ricerca in uno dei quattro TextBox e preme il pulsante INVIO.

Con la pressione del tasto INVIO si attiva una delle quattro *query* per la selezione dei dati che abbiamo creato nel **TableAdapter**:

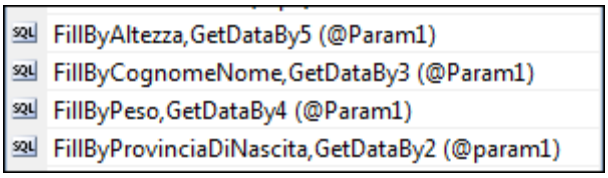


Figura 345: Le quattro query per la selezione dei dati, nel TableAdapter.

```
    Private Sub txtNome_KeyDown(sender As Object, e As
System.Windows.Forms.KeyEventArgs) Handles txtNome.KeyDown, txtNascita.KeyDown,
txtAltezza.KeyDown, txtPeso.KeyDown

    ' Controlla se nei due textbox numerici (peso e altezza) sono stati inseriti
solo numeri:
    Select Case sender.tag

        Case 2, 3
            NumeroValido = False
            If e.KeyCode > 47 And e.KeyCode < 58 Then NumeroValido = True

    End Select

    If e.KeyCode = 13 Then
        If sender.text = "" Then Exit Sub
        ' Se l'utente ha premuto il tasto INVIO in una delle quattro caselle di
testo,
        ' e se il testo non è vuoto, si avvia la ricerca dei dati:

        cmdStampaSelezione.Enabled = False

        ' La variabile Ricerca assume il valore dell'oggetto da cercare:
        Dim Ricerca As String = sender.text

        ' A seconda della casella di testo attiva, si avvia la query col
parametro digitato:

        Select Case sender.tag

            Case 0

Me.AlunniTableAdapter.FillByCognomeNome(Me.AnagrafeDataSet.Alunni, "%" + Ricerca
+ "%")
                Titolo = "ELENCO ALUNNI CON *" + Ricerca + "*  NEL NOMINATIVO"
                Me.Text = Titolo

            Case 1

Me.AlunniTableAdapter.FillByProvinciaDiNascita(Me.AnagrafeDataSet.Alunni,
Ricerca)
                Titolo = "ELENCO ALUNNI NATI IN PROVINCIA DI " + Ricerca
                Me.Text = Titolo

            Case 2
```

```
                Me.AlunniTableAdapter.FillByPeso(Me.AnagrafeDataSet.Alunni,
CInt(Ricerca))
                Titolo = "ELENCO ALUNNI CON PESO SUPERIORE A kg " + Ricerca
                Me.Text = Titolo

            Case 3

                Me.AlunniTableAdapter.FillByAltezza(Me.AnagrafeDataSet.Alunni,
CInt(Ricerca))
                Titolo = "ELENCO ALUNNI CON ALTEZZA SUPERIORE A cm " + Ricerca
                Me.Text = Titolo
        End Select

        ' Se la ricerca ha prodotto un risultato positivo, si attiva il pulsante
di stampa dei dati selezionati:

        If Me.AnagrafeDataSet.Alunni.Count > 0 Then cmdStampaSelezione.Enabled =
True

    End If

End Sub
```

```
Private Sub txtNome_KeyPressTesto(sender As Object, e As
System.Windows.Forms.KeyPressEventArgs)

    ' Questa procedura gestisce la scrittura del carattere corrispondente dei
tasti premuti nei TextBox:

    Select Case sender.tag

        Case 0, 1
            ' Trascrive in maiuscolo il parametro di ricerca scritto dall'utente:
            e.KeyChar = UCase(e.KeyChar)

        Case 2, 3
            ' se il carattere da scrivere nei due TextBox numerici non è un
numero, il programma finge
            ' di averlo già scritto e passa oltre, senza scrivere nulla:
            If NumeroValido = False Then e.Handled = True

    End Select

End Sub
```

Rileggiamo i comandi che attivano le quattro *query*:

```
Me.AlunniTableAdapter.FillByCognomeNome(Me.AnagrafeDataSet.Alunni, "%" + Ricerca
+ "%")
```

```
Me.AlunniTableAdapter.FillByProvinciaDiNascita(Me.AnagrafeDataSet.Alunni,
Ricerca)
```

```
Me.AlunniTableAdapter.FillByPeso(Me.AnagrafeDataSet.Alunni, CInt(Ricerca))
```

```
Me.AlunniTableAdapter.FillByAltezza(Me.AnagrafeDataSet.Alunni, CInt(Ricerca))
```

La variabile **Ricerca**, che corrisponde a ciò che l'utente ha scritto nel TextBox, viene utilizzata come parametro di ricerca per la selezione dei dati.
Nella prima di queste *query* (**FillByCognomeNome**), il comando aggiunge alla variabile **Ricerca**, in testa e in coda alla variabile, il simbolo percentuale **%.**
Con l'aggiunta di questo simbolo, l'operatore **LIKE**, che è l'operatore scelto per questa *query*, selezionerà tutti i nomi che **contengono** il parametro di ricerca.
Se, ad esempio, l'utente scrive nel primo TextBox le lettere "**anna**", il parametro di ricerca sarà **&anna&** e il risultato della ricerca sarà : Annalaura, Anna, Marianna, Annalisa...

Concludiamo la scrittura del codice con le procedure attivate dai pulsanti **cmdStampaSelezione** e **cmdStampaTutto**, che contengono le istruzioni dedicate alla stampa su carta dei risultati delle ricerche o dell'elenco completo degli alunni.
Queste procedure utilizzano due componenti per la stampa che non abbiamo ancora inserito nel form.
Si tratta dei componenti **PrintDocument** e **PrintPreviewDialog**: li preleviamo dalla Casella degli Strumenti e li inseriamo nel **frmRicerche**. Questi due componenti vanno a collocarsi in basso, nell'area esterna al form, assieme agli altri componenti che vi sono già presenti:

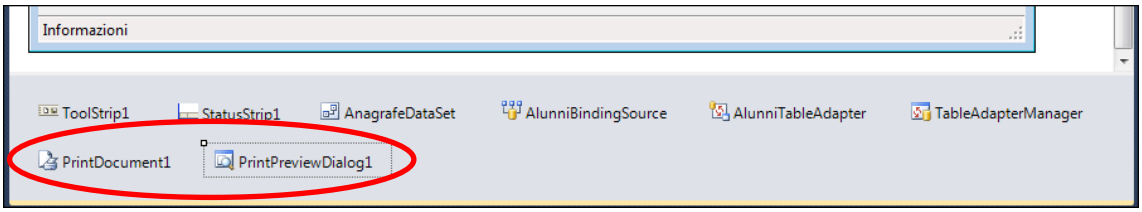


Figura 346: Inserimento dei componenti PrintDocument e PrintPreviewDialog nel frmRicerche.

La prima procedura di stampa si attiva con un *clic* del mouse sul pulsante **cmdStampaSelezione** o sul pulsante **cmdStampaTutto**. Essa verifica se l'utente vuole stampare una selezione di dati o tutto l'elenco, e quindi prepara il testo per la stampa, inserendovi i dati una riga alla volta.

```
    Private Sub cmdStampaSelezione_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles cmdStampaSelezione.Click, cmdStampaTutto.Click

        Dim NumeroRighe As Integer = 0
        Dim Linea As String
        ElencoDaStampare = New List(Of String)

        If sender.text = "Stampa tutti i dati" Then
            ' Se l'utente vuole stampare l'elenco completo, questa riga
visualizza TUTTI i dati nella tabella.
            Titolo = "ELENCO ALUNNI"
            Me.AlunniTableAdapter.Fill(Me.AnagrafeDataSet.Alunni)
        End If
```

```
        ' Prepara il testo da stampare aggiungendolo riga per riga alla lista
ElencoDaStampare:
        ElencoDaStampare.Add(Titolo)
        ElencoDaStampare.Add(vbCrLf)
        ElencoDaStampare.Add(vbCrLf)
        ElencoDaStampare.Add(vbCrLf)
        ElencoDaStampare.Add("Cognome e nome" & vbTab & "Indirizzo" & vbTab &
"Telefono")
        ElencoDaStampare.Add(vbCrLf)

        For Each Riga As DataGridViewRow In AlunniDataGridView.Rows
            Linea = ""
            NumeroRighe += 1
            ' Imposta una riga di stampa con un numero progressivo:
            Linea = NumeroRighe.ToString("###") & ". "
            ' Aggiunge il nome dell'alunna/o, che nella tabella del database si
trova alla colonna 2:
            Linea &= Riga.Cells("DataGridViewTextBoxColumn2").Value.ToString &
vbTab
            ' Aggiunge la residenza, che nella tabella del database  si trova
alla colonna 6:
            Linea &= Riga.Cells("DataGridViewTextBoxColumn6").Value.ToString & ",
"
            ' Aggiunge l'indirizzo, che nella tabella del database si trova alla
colonna 7:
            Linea &= Riga.Cells("DataGridViewTextBoxColumn7").Value.ToString &
vbTab
            ' Aggiunge il numero di telefono, che nella tabella del database  si
trova alla colonna 9:
            Linea &= Riga.Cells("DataGridViewTextBoxColumn9").Value.ToString

            ' Aggiunge la linea a ElencoDaStampare;
            ElencoDaStampare.Add(Linea)
        Next

        ' Conclude l'elenco delle righe da stampare inserendovi la data corrente:
        ElencoDaStampare.Add(vbCrLf)
        ElencoDaStampare.Add(vbCrLf)
        ElencoDaStampare.Add(Date.Today)

        ' L'elenco è così completato, si può procedere con la stampa:
        PrintPreviewDialog1.Document = PrintDocument1
        PrintPreviewDialog1.ShowDialog()

    End Sub
```

```
    Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e As
System.Drawing.Printing.PrintPageEventArgs) Handles PrintDocument1.PrintPage

        Dim ContaLinee As Integer

        ' misura l'altezza dei caratteri per l'allineamento
        Dim AltezzaFont = e.Graphics.MeasureString("W", Me.Font).Height

        ' misura l'altezza dell'area stampabile del foglio
        Dim AltgezzaFoglio = e.PageBounds.Height

        ' Il testo sarà scritto in tre colonne; i punti d'inizio della II e III
colonna sono fissati da queste tabulazioni:
        Dim Tabulazioni As Single() = {220, 220}
```

```
        ' Formattazione del testo su tre colonne, secondo le tabulazioni già
impostate:
        Dim Formato As New StringFormat
        Formato.SetTabStops(0, Tabulazioni)

        'comincia a stampare il contenuto dell'elenco
        For Contatore = 0 To ElencoDaStampare.Count - 1

            ' Stampa del testo, con il carattere del form, in nero:
            e.Graphics.DrawString(ElencoDaStampare(Contatore), Me.Font,
Brushes.Black, New PointF(50, 50 + CInt(ContaLinee * AltezzaFont)), Formato)

            ' Stampa una linea divisoria di colore grigio chiaro
            e.Graphics.DrawLine(Pens.LightGray, 45, 50 + (ContaLinee + 1) *
AltezzaFont, e.PageBounds.Width - 50, 50 + (ContaLinee + 1) * AltezzaFont)

            ContaLinee += 1

        Next

    End Sub
```

Ecco una anteprima di stampa dei dati ottenuta con queste procedure:

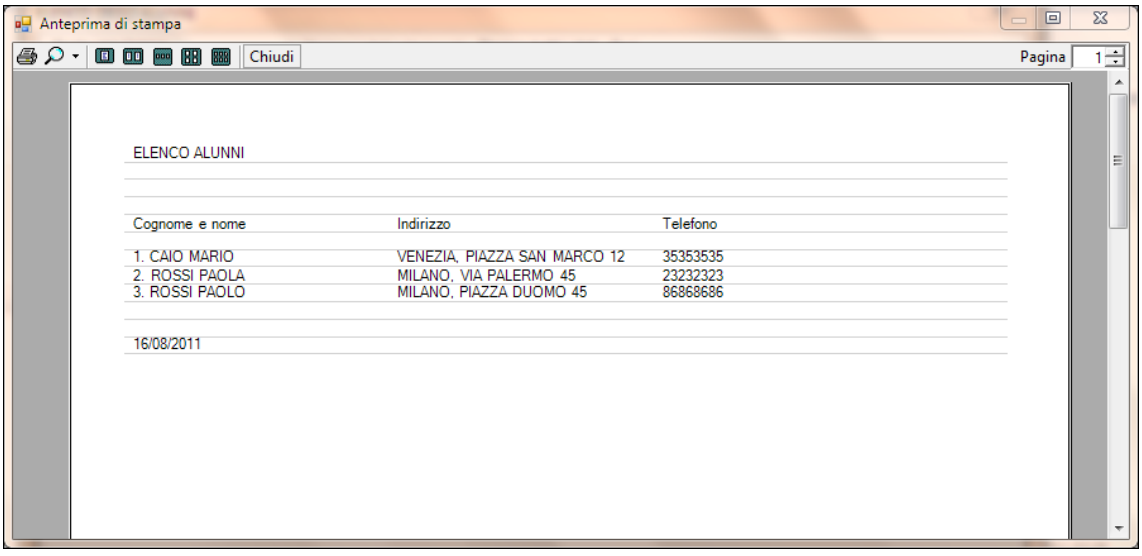


Figura 347: Anteprima di stampa di un elenco dati.

207: La finestra con gli auguri di buon compleanno.

Siamo giunti alla progettazione dell'ultima finestra del nostro programma; si tratta di una finestra con una funzione accessoria: la visualizzazione e la stampa di un biglietto di auguri di buon compleanno per gli alunni che compiono gli anni nel giorno corrente.
Come abbiamo visto nella scrittura del codice del **frmPrincipale**, all'avvio il programma controlla se nel giorno in cui il programma è aperto ricorre qualche compleanno.
In caso affermativo, le righe relative agli alunni da festeggiare si colorano di giallo e, nel componente **ToolStrip** in alto nel frmPrincipale, si attiva il pulsante **Stampa auguri**.
Con un *clic* su questo pulsante il programma apre una finestra particolare, che contiene un messaggio di auguri; questo messaggio può essere stampato.
Ma seguiamo passo per passo la messa a punto di questa nuova finestra.
Nella finestra **Esplora soluzioni**, facciamo un doppio *clic* sul form **frmAuguri.vb**.

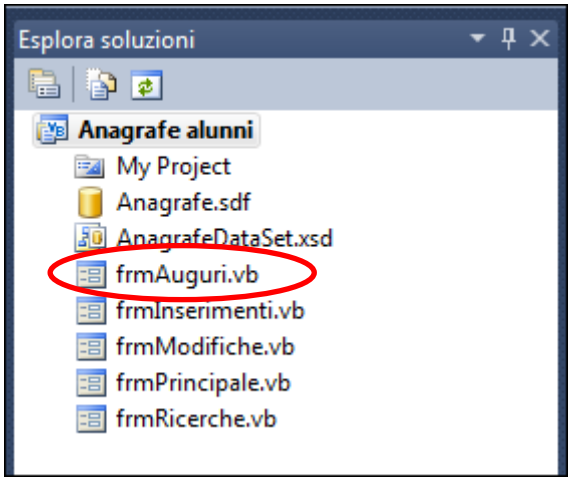


Figura 348: Il form frmAuguri.vb nella finestra Esplora soluzioni.

Impostiamo le proprietà del form in questo modo:

FormBorderStyle = FixedSingle	Imposta la visualizzazione tradizionale del form e della sua barra in alto.
MaximizeBox = False	L'utente non potrà allargare le dimensioni del form.
MinimizeBox = False	L'utente non potrà ridurre le dimensioni del form.
Showicon = False	Nessuna icona nella barra del testo del form, in alto a sinistra.

ShowInTaskbar = False	Nessuna icona, per questo programma, nella Barra delle applicazioni.
Size = 800; 570	Dimensioni del form.
StartPosition = CenterScreen	Colloca il form al centro dello schermo.
Text = Auguri di buon compleanno	Questo è il testo che compare nella barra del form, in alto a sinistra.

Come abbiano fatto per le altre fineste inseriamo nel form i due componenti che andranno a collocarsi in alto e in basso nel form un componente **ToolStrip** e un componente **StatusStrip**.

- Nel componente **ToolStrip** inseriamo quattro controlli: un **ComboBox**, un **DropDownButton** e due **Button**.
  Con un *clic* su ognuno di questi pulsanti, ne impostiamo queste proprietà:

Pulsante	**ComboBox1**	**DropDownButton1**	**Button1**	**Button2**
(Name)	cmdSceltaDestinatario	cmdSceltaImmagine	cmdStampa	cmdEsci
AutoSize	False	true	true	true
DisplayStyle	Text	Text	Text	Text
Text	Scegli il destinatario	Scegli l'immagine	Stampa	Esci
Size	200; 25			

Lasciamo il **ComboBox** vuoto, senza alcun nome. I nomi vi verranno aggiunti dal programma, quando vi saranno compleanni da festeggiare.

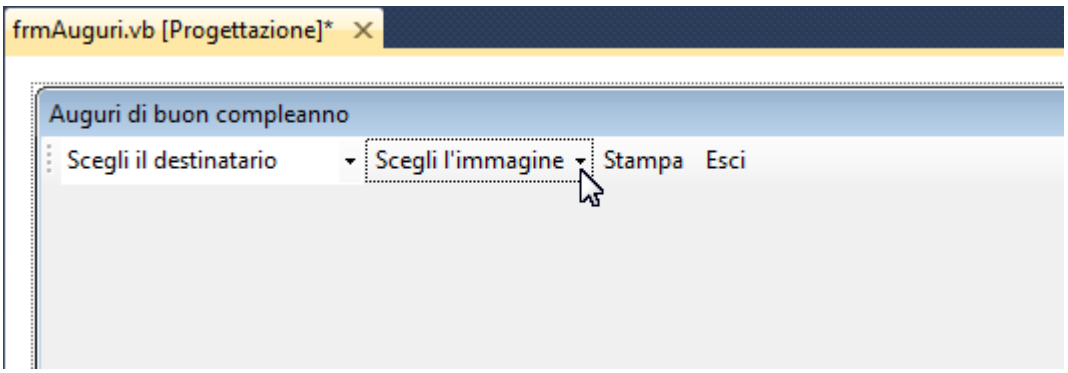


Figura 349: Inserimento di quattro controlli nel ToolStrip.

Inseriamo invece nel **DropDownButton1** (**Scegli l'immagine**) i nomi delle cinque immagini che troviamo nella cartella **Documenti / A scuola con VB 2010 / Immagini / Auguri**.
Facciamo un *clic* sulla freccina nera a destra del pulsante **Scegli l'immagine** e nel menu che si apre a tendina, mostrando la casella **Digitare qui**, scriviamo uno a uno i nomi delle cinque immagini:

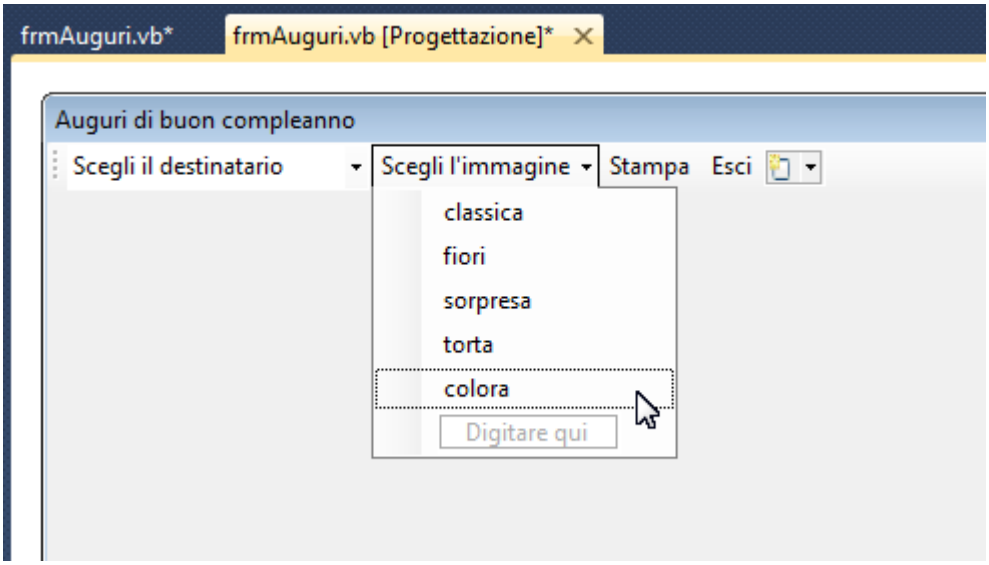


Figura 350: Inserimento di cinque item nel pulsante Scegli l'immagine.

Facendo un *clic* su ognuno di questi i cinque item inseriti in **Scegli l'immagine,** notiamo, nella **Finestra Proprietà**, che essi hanno preso in modo automatico questi nomi:

- ClassicaToolStripMenuItem1
- FioriToolStripMenuItem1
- SorpresaToolStripMenuItem1
- TortaToolStripMenuItem1
- ColoraToolStripMenuItem1

Per semplificare il nostro lavoro provvediamo a modificare i nomi come segue:

- **Classica**
- **Fiori**
- **Sorpresa**
- **Torta**
- **Colora**

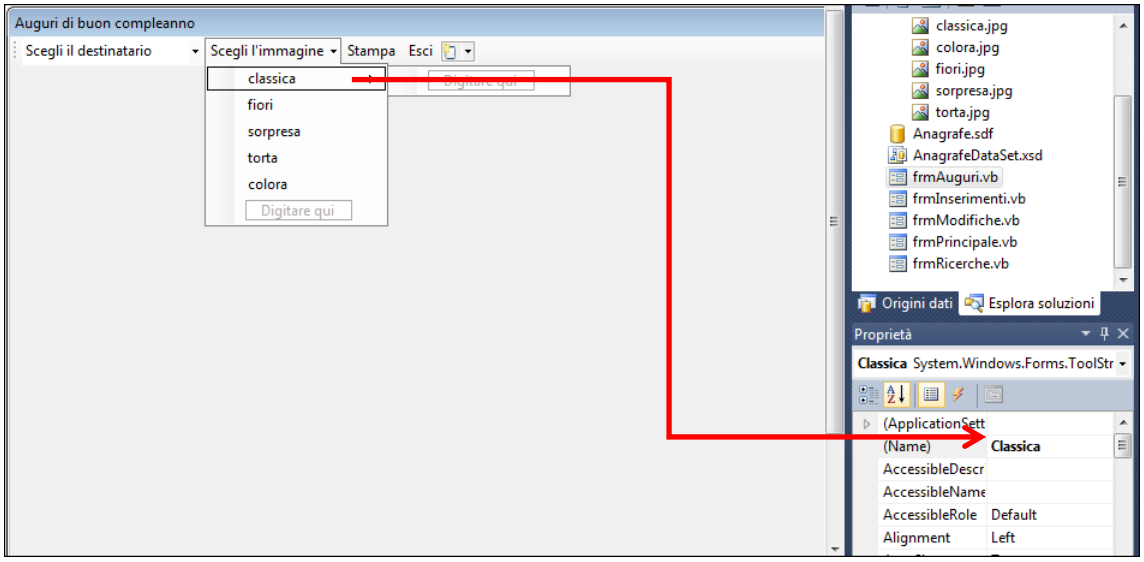


Figura 351: Denominazione dei cinque item del pulsante Scegli l'immagine.

Per integrare le cinque immagini nel programma, in modo da poterle distribuire più facilmente assieme ad esso, le collochiamo nelle risorse del programma:

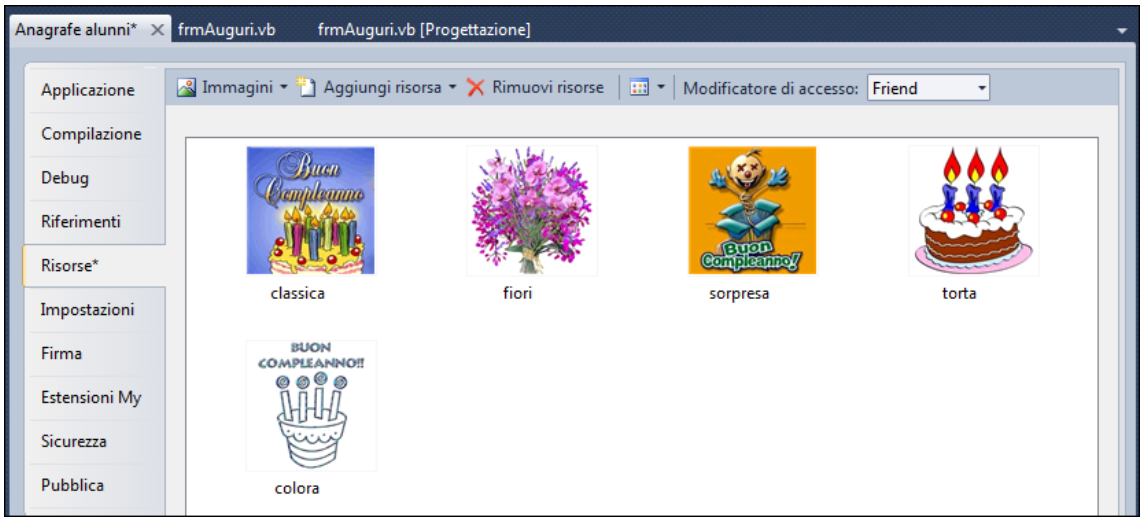


Figura 352: Inserimento di cinque immagini nelle risorse del programma.

Inseriamo nelle risorse del programma anche un file sonoro per abbinare una musica all'apertura di questo form. Preleviamo il file **Auguri.wav** dalla cartella **Documenti / Ascuola con VB 2010 / Audio**:

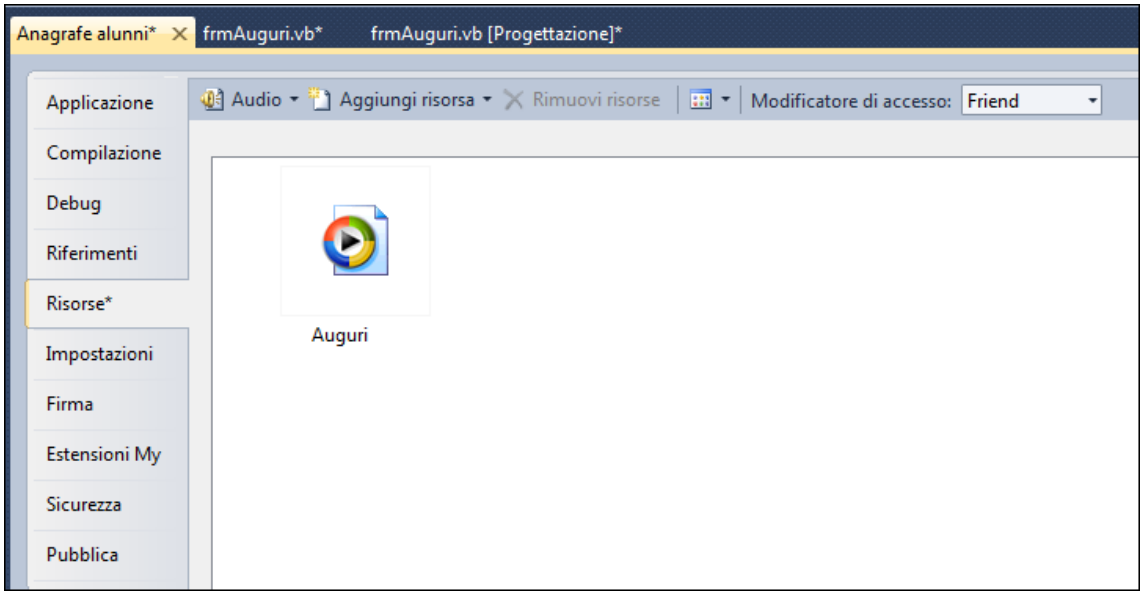


Figura 353: Inserimento di un file audio nelle risorse del programma.

Dopo l'inserimento di questi file nelle risorse del programma, ecco come si presenta la finestra **Esplora soluzioni**:

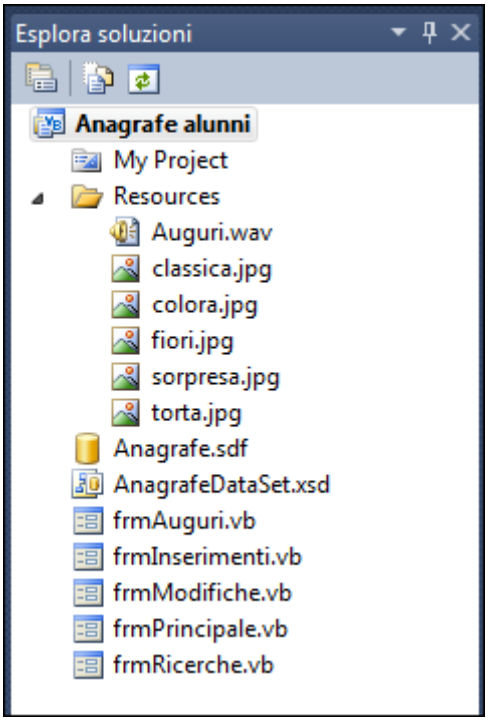


Figura 354: La finestra Esplora soluzioni con i file delle risorse del programma.

- Inseriamo ora nel componente **StatusStrip**, che si trova in basso nel form, un controllo **Label**, destinato a contenere informazioni sull'uso del programma. Proprietà di questa **Label**:
  (Name) = lblIstruzioni
  Text = Istruzioni

Inseriamo nella parte centrale del form tre controlli **Label** (rispettivamente: **Label1, Label2, Label3**) e un controllo **PictureBox**, come nell'immagine seguente.

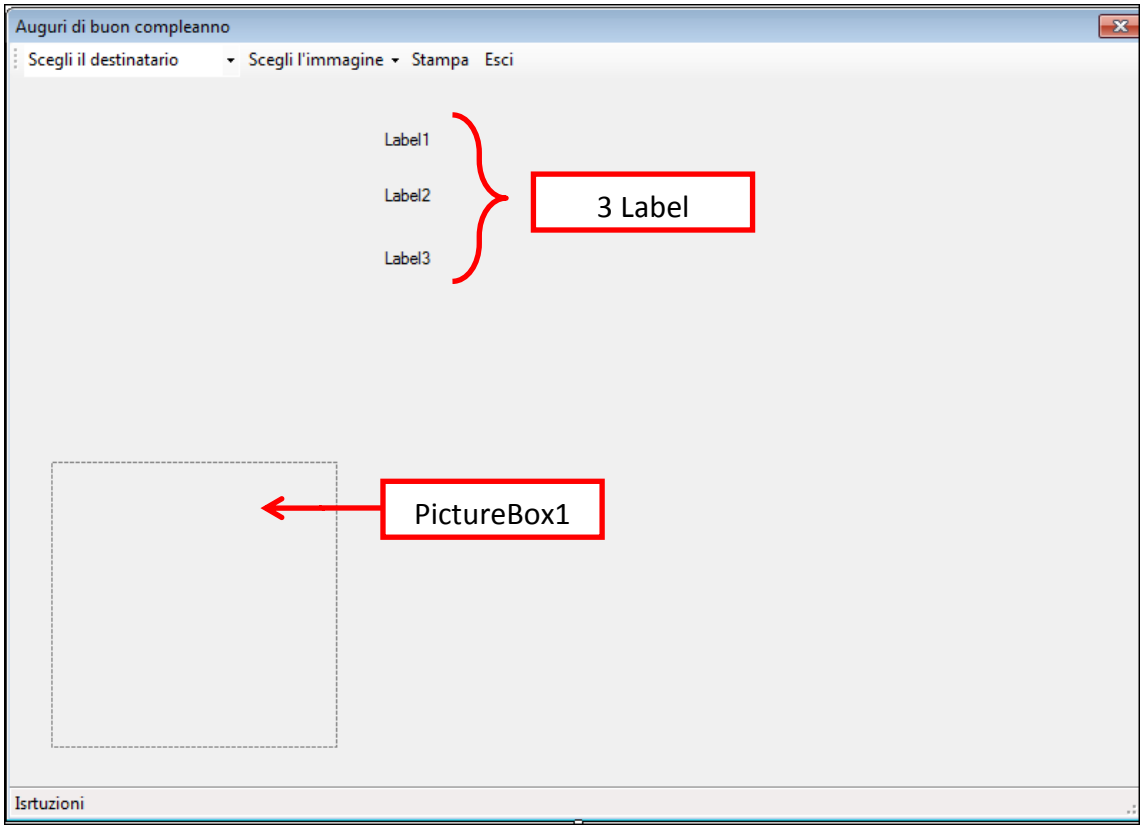


Figura 355: Completamento del frmAuguri.

I tre controlli **Label** sono destinati a contenere il testo del messaggio d'auguri, mentre il controllo **PictureBox** è destinato a contenere l'immagine da abbinare al messaggio.
Le proprietà di queste **Label** e del **PictureBox** saranno impostate dal codice, all'apertura del **frmAuguri**, per cui ora non è necessaria alcuna operazione.

Iniziamo la scrittura del codice partendo dalla procedura per la chiusura del form. Facciamo un doppio *clic* sul pulsante **cmdEsci** e accediamo alla **Finestra del Codice**.
In questo form non abbiamo l'esigenza di effettuare controlli di alcun tipo al momento della chiusura del form, per cui è sufficiente scrivere questa procedura:

```
Public Class frmAuguri

    Private Sub cmdEsci_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdEsci.Click
        ' Chiude la finestra
        Me.Close()
        ' Cancella dalla memoria tutti i riferimenti al frmAuguri:
        Me.Dispose()
    End Sub

End Class
```

Passiamo alla procedura che gestisce il caricamento del **frmAuguri**.
Questa procedura imposta le proprietà dei tre controlli **Label** e del controllo **PictureBox**, poi effettua queste operazioni:

- preleva le informazioni sui compleanni, memorizzate nella **ListBox1** nel **frmPrincipale**, e le copia nella lista del **ComboBox** di questo form;
- seleziona il primo nominativo presente nel **ComboBox cmdSceltaDestinatario**, facendo scattare l'evento **cmdSceltaDestinatario.SelectedIndexChanged**, la cui procedura è riportata più avanti.

```
Private Sub frmAuguri_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Imposta le proprietà dei controlli:
    Me.BackColor = Color.White

    With Label1
        .AutoSize = True
        .Font = New Font("Verdana", 48, FontStyle.Bold)
        .ForeColor = Color.Blue
        .Location = New Point(240, 40)
        .Text = ""
    End With

    With Label2
        .AutoSize = True
        .Font = New Font("Verdana", 24, FontStyle.Regular)
        .ForeColor = Color.Red
        .Location = New Point(248, 132)
        .Text = ""
    End With

    With Label3
        .AutoSize = True
        .Font = New Font("Verdana", 48, FontStyle.Bold)
        .ForeColor = Color.Green
        .Location = New Point(240, 416)
        .Text = ""
    End With
```

```
    With PictureBox1
        .Image = Nothing
        .Location = New Point(29, 293)
        .Size = New Size(200, 200)
    End With

    cmdSceltaDestinatario.Items.Clear()

    ' Inserisci nel ComboBox cmdSceltaDestinatario i nomi degli alunni che
compiono gli anni nel giorno corrente.
    ' (questi nomi sono copiati dal ListBox1 presente nel frmPrincipale)
    Dim Contatore As Integer
    For Contatore = 0 To frmPrincipale.ListBox1.Items.Count - 1

cmdSceltaDestinatario.Items.Add(frmPrincipale.ListBox1.Items.Item(Contatore))
    Next

    If frmPrincipale.ListBox1.Items.Count - 1 > 1 Then
        lblIstruzioni.Text = "Scegli il destinatario degli auguri, nel pulsante
in alto a sinistra."
    Else
        lblIstruzioni.Text = "Scegli un'immagine: clicca il pulsante nella barra
in alto."
    End If

    ' Seleziona l'alunna/o nella prima riga del ComboBox
    cmdSceltaDestinatario.SelectedIndex = 0

End Sub
```

La procedura collegata all'evento per la scelta del destinatario svolge i compiti più importanti:

- legge il dato cliccato dall'utente (in questo dato vi sono Cognome e nome dell'alunno che compie gli anni e la sua data di nascita, separati da un trattino);
- separa cognome e nome dalla data di nascita, spezzando la stringa in corrispondenza del trattino (-);
- estrapola dal dato cognome e nome il solo nome, tagliando la sequenza cognome nome in corrispondenza dell'ultimo spazio vuoto (si suppone che dopo l'ultimo spazio vuoto nella stringa cognome nome sia scritto il nome dell'alunno; es.: Rossi **Paolo**);
- calcola la differenza tra la data del giorno corrente e la data di nascita dell'alunno in anni, mesi, giorni, ore, minuti e secondi;
- scrive nelle proprietà Text dei tre controlli Label i valori così elaborati;
- infine, esegue il file audio 'Auguri' presente nelle risorse del programma.

```
    Private Sub cmdSceltaDestinatario_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
cmdSceltaDestinatario.SelectedIndexChanged

    ' Assegna alla variabile Selezione il testo (nominativo + data di
nascita) dell'item selezionato nel ComboBox:
    Dim Selezione As String = cmdSceltaDestinatario.SelectedItem
```

```
    ' Spezza la riga selezionata in corrispondenza del trattino (-), per
ricavarne due stringhe con
    ' il nome dell'alunno e la sua data di nascita
    Dim DividiSelezione As String() = Selezione.Split("-")
    ' la prima parte della stringa DividiSelezione, a sinistra del trattino
(-), contiene cognome e nome del festeggiato:
    Dim Festeggiato As String = DividiSelezione(0)
    ' la seconda parte della stringa DividiSelezione, a destra del trattino
(-), contiene la data di nascita:
    Dim Nascita As Date = CDate(DividiSelezione(1))

    ' Spezza la stringa con cognome dal nome del festeggiato in corripondenza
del carattere spazio vuoto:
    DividiSelezione = Festeggiato.Split(" ")

        ' Il nome è la seconda parte della stringa DividiSelezione, dopo il
cognome e dopo lo spazio vuoto:
        Dim Nome As String
        If DividiSelezione.Length > 1 Then
            Nome = DividiSelezione(1)
        Else
            Nome = DividiSelezione(0)
        End If

    ' Calcoli delle differenze tra la data odierna e la data di nascita del
festeggiato:

    Dim Anni As Integer = DateTime.Today.Year - Nascita.Year
    Dim DifferenzaData As TimeSpan = DateTime.Today.Subtract(Nascita)
    Dim Giorni As Integer = DifferenzaData.TotalDays
    Dim Ore As Integer = DifferenzaData.TotalHours
    Dim Minuti As Integer = DifferenzaData.TotalMinutes
    Dim Secondi As Integer = DifferenzaData.TotalSeconds

    ' Elaborazione dei testi per le tre Label:
    Label1.Text = "a " & LCase(Nome)

    Label2.Text = "per i suoi " + Anni.ToString + " anni" & vbCrLf & "e cioè:
" & vbCrLf
    Label2.Text &= (Anni * 12).ToString("###,###") + " mesi," & vbCrLf
    Label2.Text &= Giorni.ToString("###,###") & " giorni," & vbCrLf
    Label2.Text &= Ore.ToString("###,###,###") & " ore," & vbCrLf
    Label2.Text &= Minuti.ToString("###,###,###,###") & " minuti," & vbCrLf
    Label2.Text &= Secondi.ToString("###,###,###,###,###") + " secondi:"

    Label3.Text = "auguri!!!"

    ' Esecuzione del file audio 'Auguri' presente nelle risorse del
programma:
    My.Computer.Audio.Play(My.Resources.Auguri, AudioPlayMode.Background)

End Sub
```

Ecco la procedura per la scelta di un'immagine, tra le cinque disponibili, da inserire nel biglietto d'auguri:

```
    Private Sub Classica_Click(sender As Object, e As System.EventArgs) Handles
Classica.Click, Colora.Click, Fiori.Click, Sorpresa.Click, Torta.Click
    Dim NomeImmagine As String
    NomeImmagine = sender.text.trim
    PictureBox1.Image = My.Resources.ResourceManager.GetObject(NomeImmagine)

End Sub
```

L'eventuale stampa del biglietto viene effettuata con il metodo più semplice e diretto: il metodo **PrintForm**, che invia alla stampante, senza alcuna modifica, il contenuto di quanto è visualizzato nel form.
Per la precisione, nella procedura di stampa che vedremo tra poco il metodo **PrintForm** invia il contenuto del form non direttamente alla stampante, ma all'**anteprima di stampa** da dove l'utente, se lo desidera, potrà avviare il processo di stampa.
L'uso del metodo **PrintForm** richiede che nel form sia inserito l'omonimo componente **PrintForm**, che si trova nella **Casella degli Strumenti**, tra i controlli del gruppo **Visual Basic PowerPacks**:

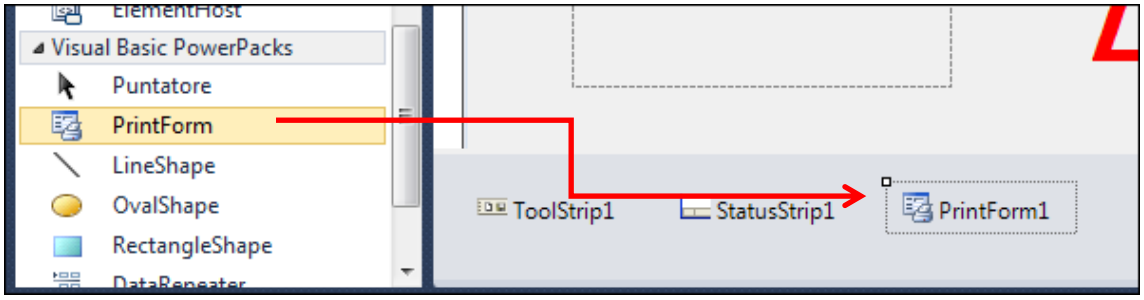


Figura 356: Completamento del frmAuguri con il componente PrintForm.

Ecco il codice della procedura di stampa:

```
Private Sub cmdStampa_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdStampa.Click

    ' Prima di stampare il contenuto del form nasconde i controlli che non devono
comparire nel biglietto:
    ToolStrip1.Visible = False
    StatusStrip1.Visible = False

    ' Esegui l'anteprima della stampa del form (dall'anteprima l'utente del
programma potrà passare alla stampa
    ' o potrà tornare a questo frmAuguri):

    PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
    PrintForm1.Print()

    ' Ripristina la visibilità dei controlli:
    ToolStrip1.Visible = True
    StatusStrip1.Visible = True

End Sub
```

Notiamo, nella procedura, che prima di inviare il contenuto del form all'anteprima di stampa vengono nascosti i componenti ToolStrip1 e StatuStrip1, per evitare che siano stampati assieme al messaggio di auguri. I due componenti tornano a essere visibili subito dopo l'operazione di stampa.
Ecco un'immagine del programma in esecuzione, con l'anteprima di stampa di un messaggio di auguri:

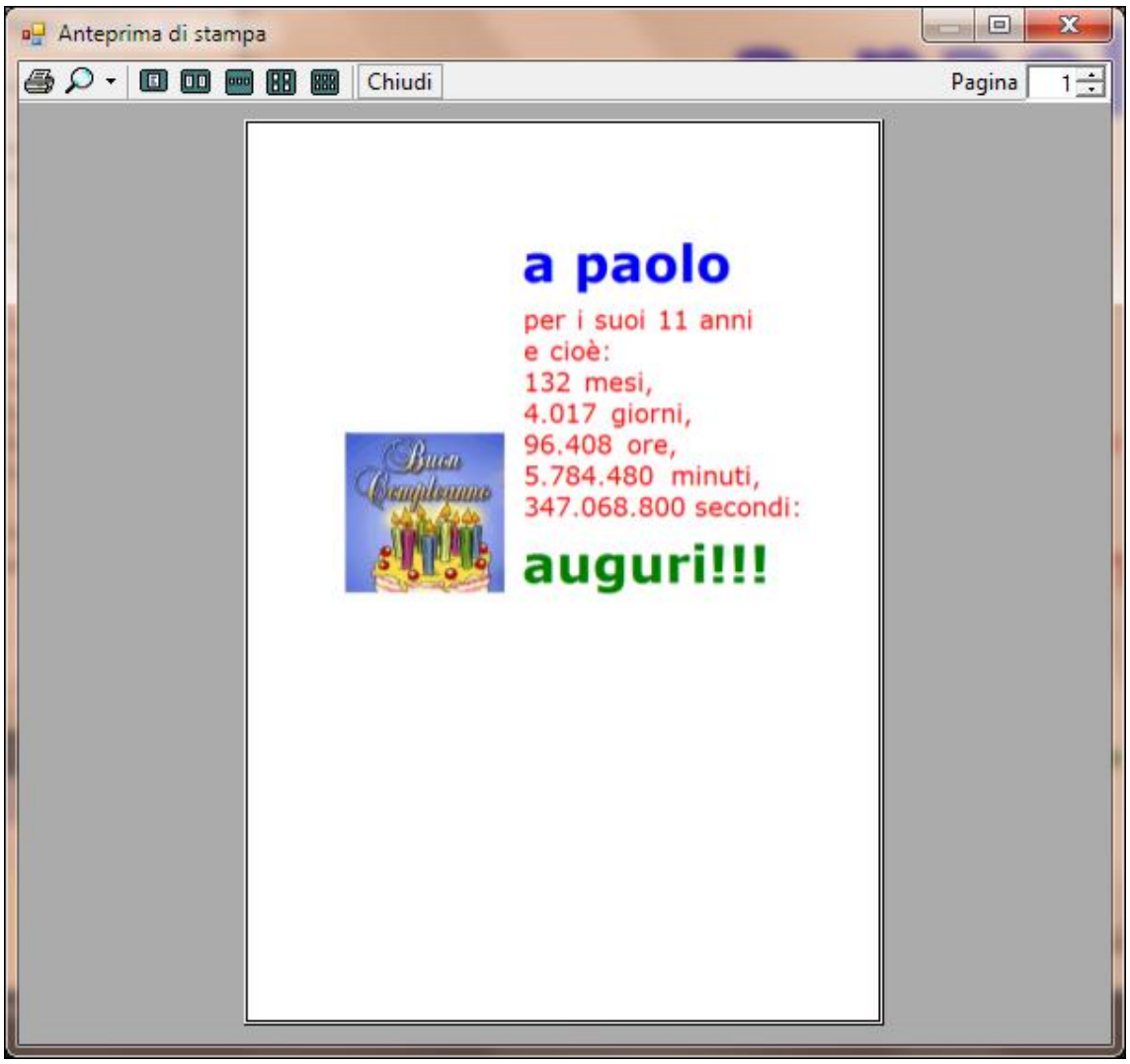


Figura 357: L'anteprima di stampa di un biglietto di auguri su un foglio di formato A4.

208: Prerequisiti per la distribuzione del programma.

Terminato il programma, si pone il problema della sua distribuzione in altri computer: a questo proposito rimandiamo la lettrice o il lettore all'ultimo capitolo, che è appunto dedicato alle operazioni di compilazione e distribuzione di un programma.
Perché questo particolare programma con database funzioni su altri computer senza problemi, possono essere necessarie alcune operazioni di aggiornamento del sistema operativo di quei computer.
Si tratta, in pratica, di verificare se i sistemi operativi dei computer ospiti hanno i requisiti necessari per garantire la piena funzionalità del programma e, in caso negativo, di installarvi i componenti mancanti.
Esaminiamo qui quali sono i prerequisiti di cui il programmatore deve garantirsi la presenza, prima di passare alla pubblicazione di un programma con database; le operazioni descritte in questo paragrafo sono dunque operazioni preliminari a quelle descritte nell'ultimo capitolo del manuale.

Facciamo un *clic* cul menu **Progetto**, nella barra dei menu di VB, e quindi sulla voce Proprietà di **Anagrafe alunni**:

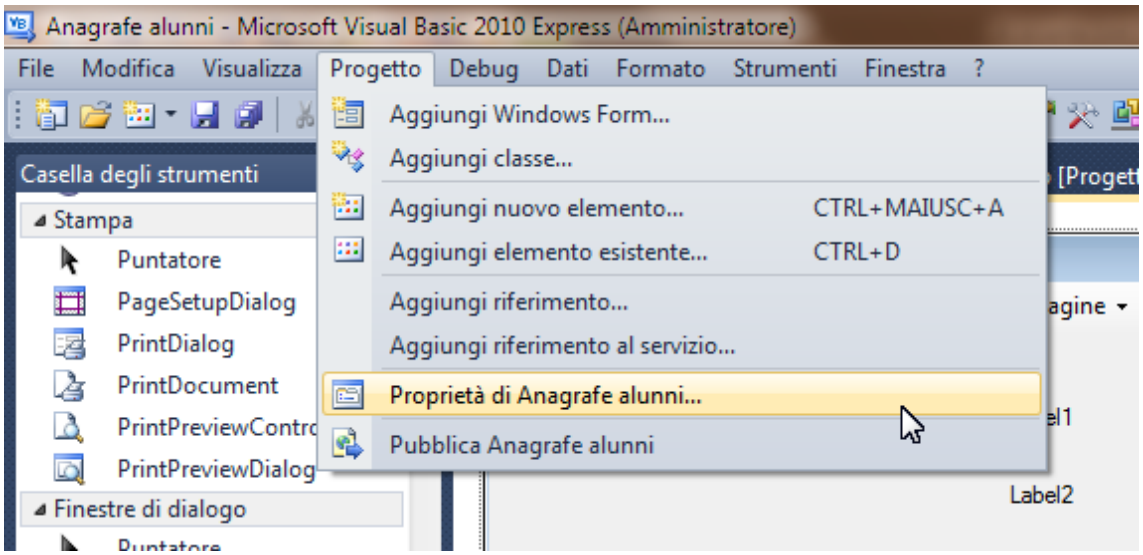


Figura 358: il menu Progetto / Proprietà di Anagrafe alunni.

Nella Finestra Proprietà del progetto, facciamo un *clic* sulla scheda **Pubblica** e quindi sul pulsante **Prerequisiti**:

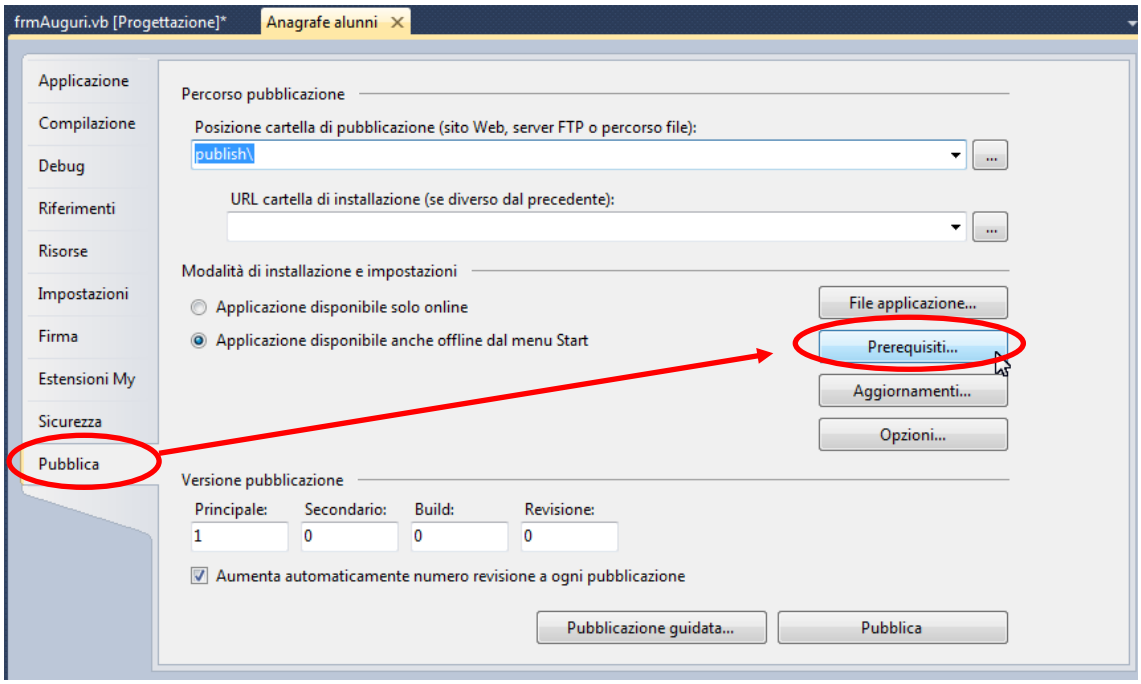


Figura 359: Apertura della scheda dei prerequisiti per la pubblicazione del programma.

Nella scheda dei prerequisiti da installare per la pubblicazione del programma, scegliamo queste opzioni e questi elementi:

- Crea programma di installazione per installare componenti dei prerequisiti;
- Microsoft .NET FrameWork 4 (versione completa, non la versione Client Profile);
- Microsoft Visual Basic PowerPacks 10.0 (è richiesto per la stampa degli auguri);
- SQL Server Compact 3.5 SP2 (gestisce gli oggetti del *database*, tabelle e *query*);
- Windows Installer 3.1;
- Scarica prerequisiti dallo stesso percorso dell'applicazione.

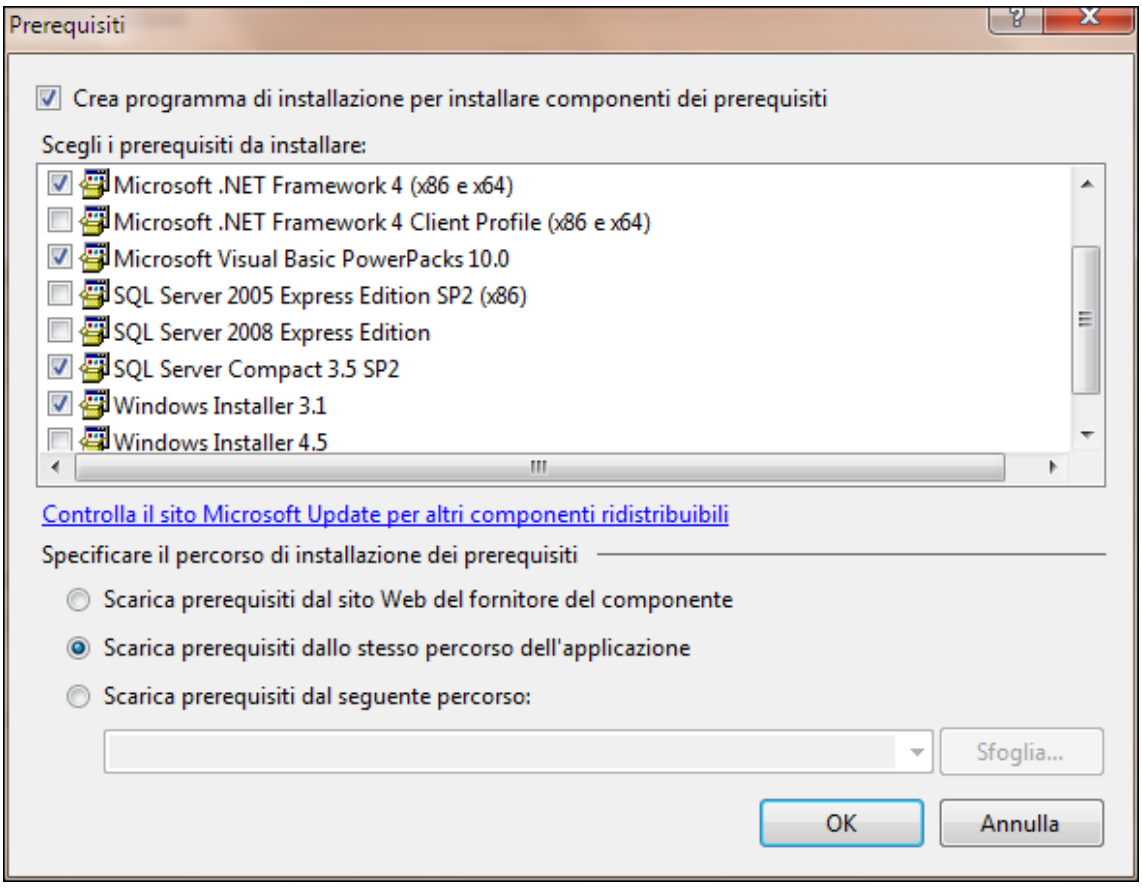


Figura 360: Scheda dei prerequisiti per la pubblicazione del programma.

PARTE VII: TERMINARE UN PROGETTO.

Questa parte del manuale è dedicata alle operazioni che concludono la creazione di un programma:

- *dotare il programma di strumenti di aiuto e di guida per l'utente;*
- *controllare che il programma non contenga errori e scorra senza incepparsi;*
- *preparare un pacchetto per la distribuzione del programma e per la sua installazione su altri computer, con tutti i file necessari per il suo funzionamento.*

Capitolo 37: CREAZIONE DI AIUTI E SUGGERIMENTI PER L'UTENTE.

Vi sono almeno quattro modi diversi per inserire in un programma informazioni o suggerimenti per l'utente; sta al programmatore valutare il modo (o la combinazione di modi) che meglio si adattano alla sua applicazione:

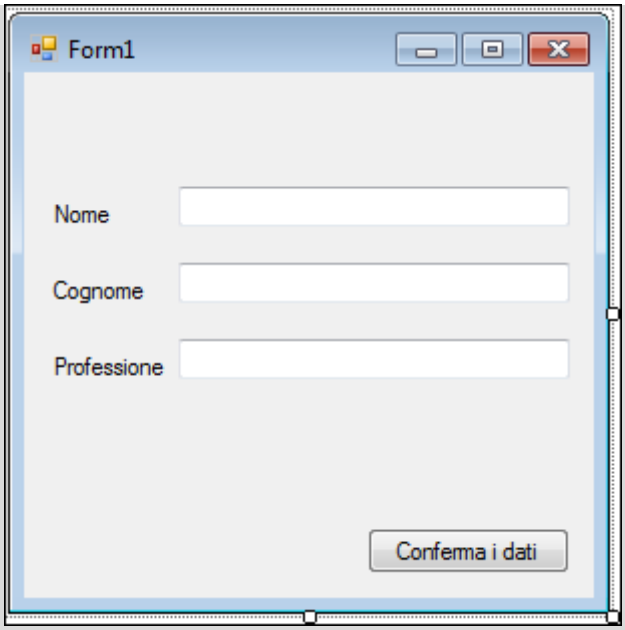
- il primo modo utilizza il componente **ToolTip** (strumento di suggerimento) associato ai controlli presenti nel form[91];
- il secondo modo consiste nella possibilità di mostrare all'interno di una **Label** testi di guida e di aiuto associati alla proprietà **.Tag** dei controlli presenti nel form;
- il terzo si avvale del componente **HelpProvider**[92] (visibile per l'utente nella barra in alto a destra) per mostrare testi di aiuto associati a un file di help esterno oppure associati a ogni singolo controllo presente nel form;
- quarto modo: è da considerare come uno strumento di guida per l'utente anche il componente **ErrorProvider**[93], che notifica in tempo reale eventuali errori commessi dall'utente.

Vedremo questi quattro modi, uno alla volta, negli esercizi che seguono.

Esercizio 137: Uso di ToolTip.

Apriamo un nuovo progetto e inseriamo nel form tre controlli TextBox, tre Label e un pulsante Button come in questa immagine:

[91] Per il componente **ToolTip** si veda anche anche il paragrafo43: Il componente ToolTip.198,
[92] Per il componente **HelpProvider** si veda anche il paragrafo 56, a pag. 273.
[93] Per il componente **ErrorProvider** si veda anche il paragrafo 55: Il componente ErrorProvider.268.



Impostiamo la proprietà **Text** di questi controlli.

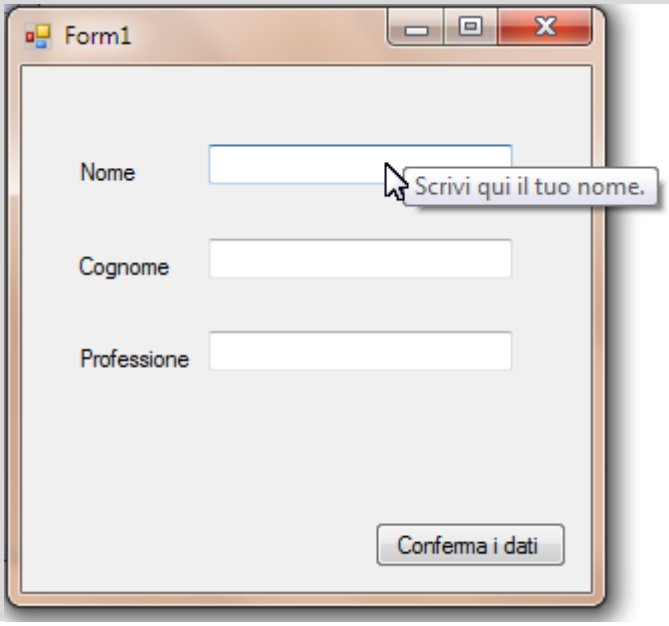
- Label1: **Text = Nome**
- Label2: **Text = Cognome**
- Label3: **Text = Professione**
- Button1: **Text = Conferma i dati**

Questo form, con questi controlli, servirà come base anche per gli altri esercizi contenuti in questi capitolo.

Inseriamo nel progetto un componente **ToolTip.**
Copiamo e incolliamo nella Finestra del Codice il listato che segue.
Notiamo che la procedura di avvio del programma (**Form1_Load**) crea quattro ToolTip di suggerimento per i tre TextBox e il Button1 presenti nel form; questi ToolTip sono visibili quando l'utente si sofferma con il mouse su uno dei controlli interessati.

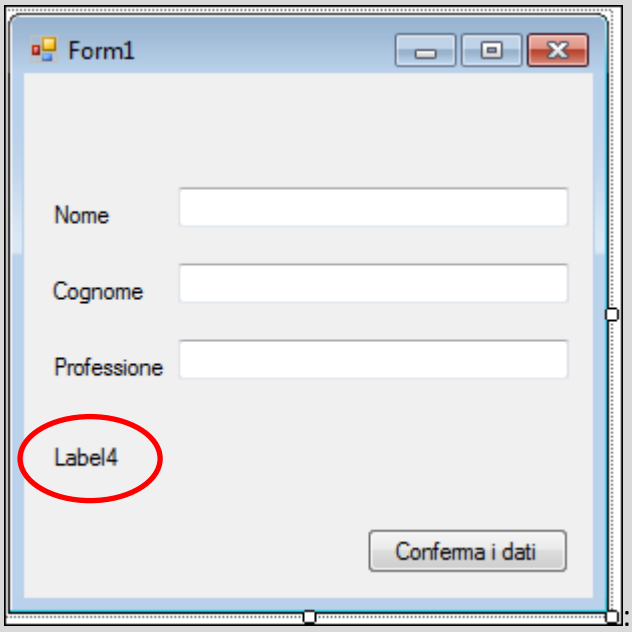
```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        ToolTip1.SetToolTip(TextBox1, "Scrivi qui il tuo nome.")
        ToolTip1.SetToolTip(TextBox2, "Scrivi qui il tuo cognome.")
        ToolTip1.SetToolTip(TextBox3, "Scrivi qui la tua professione.")
        ToolTip1.SetToolTip(Button1, "Clicca questo pulsante per confermare i
dati scritti.")

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:



Esercizio 138: Uso di una Label per fornire istruzioni sul programma.

Riprendiamo il form di base dell'esercizio precedente.
Inseriamo nel form un quarto controllo Label che servirà per visualizzare le informazioni sul programma:



Copiamo e incolliamo nella Finestra del Codice il listato che segue.

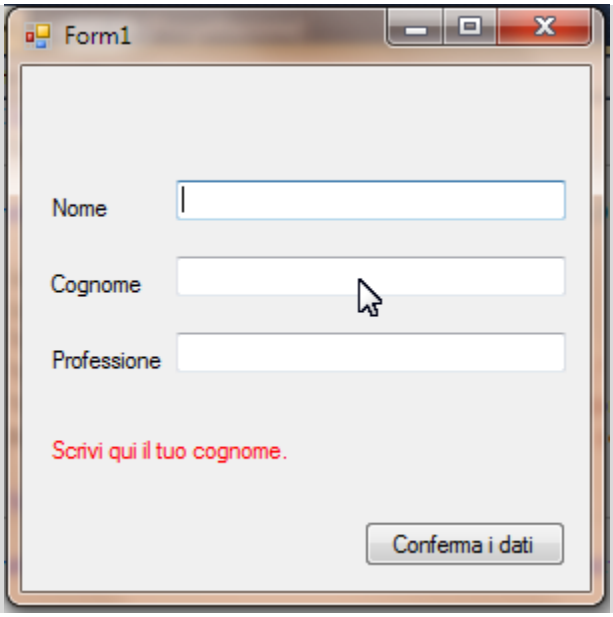
In questo caso, i brevi testi di aiuto sono collocati nella proprietà **.Tag** dei controlli interessati e sono visualizzati nella Label4 quando il puntatore del mouse entra nell'area di uno di questi controlli.

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Label4.Text = ""
        Label4.ForeColor = Color.Red
        TextBox1.Tag = "Scrivi qui il tuo nome."
        TextBox2.Tag = "Scrivi qui il tuo cognome."
        TextBox3.Tag = "Scrivi qui la tua professione."
        Button1.Tag = "Clicca questo pulsante per confermare i dati scritti."

    End Sub
```

```
    Private Sub TextBox1_MouseEnter(sender As Object, e As System.EventArgs)
Handles TextBox1.MouseEnter, TextBox2.MouseEnter, TextBox3.MouseEnter,
Button1.MouseEnter

        Label4.Text = sender.tag

    End Sub
```

```
    Private Sub TextBox1_MouseLeave(sender As Object, e As System.EventArgs)
Handles TextBox1.MouseLeave, TextBox2.MouseLeave, TextBox3.MouseLeave,
Button1.MouseLeave

        Label4.Text = ""

    End Sub

End Class
```

Ecco un'immagine del programma in esecuzione:

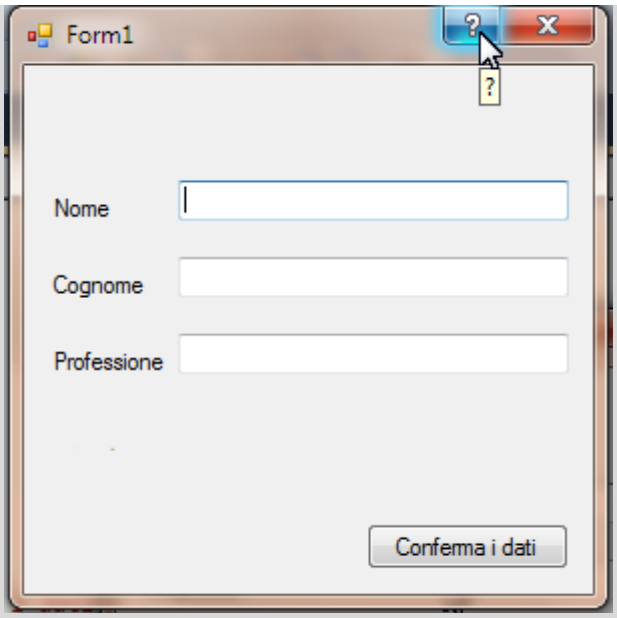


Esercizio 139: Il componente HelpProvider.

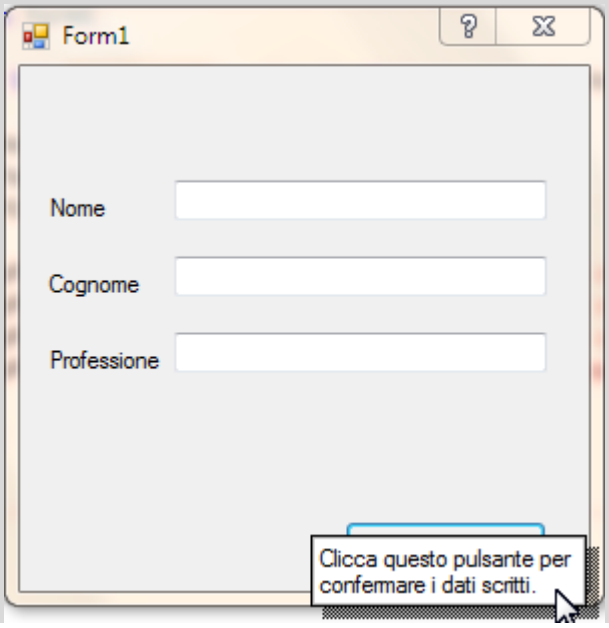
Riprendiamo il form dell'esercizio Esercizio 137, a pag. 993.
Inseriamo nel form un componente **HelpProvider**.
Copiamo e incolliamo nella Finestra del Codice il listato seguente.

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        Me.HelpButton = True
        Me.MaximizeBox = False
        Me.MinimizeBox = False

        HelpProvider1.SetHelpString(TextBox1, "Scrivi qui il tuo nome.")
        HelpProvider1.SetHelpString(TextBox2, "Scrivi qui il tuo cognome.")
        HelpProvider1.SetHelpString(TextBox3, "Scrivi qui la tua professione.")
        HelpProvider1.SetHelpString(Button1, "Clicca questo pulsante per
confermare i dati scritti.")

    End Sub

End Class
```

Mandando in esecuzione il programma, notiamo l'icona con il punto interrogativo nella barra del form, in alto a destra:



L'utente può cliccare questa icona e trascinare il punto interrogativo sul controllo che lo interessa; con un *clic* del mouse può visualizzare le informazioni disponibili per quel controllo.
Lo stesso effetto si ottiene cliccando un controllo con il mouse e premendo il tasto F1 sulla tastiera:



Esercizio 140: Il componente ErrorProvider.

Riprendiamo il form dell'esercizio Esercizio 137, a pag. 993.

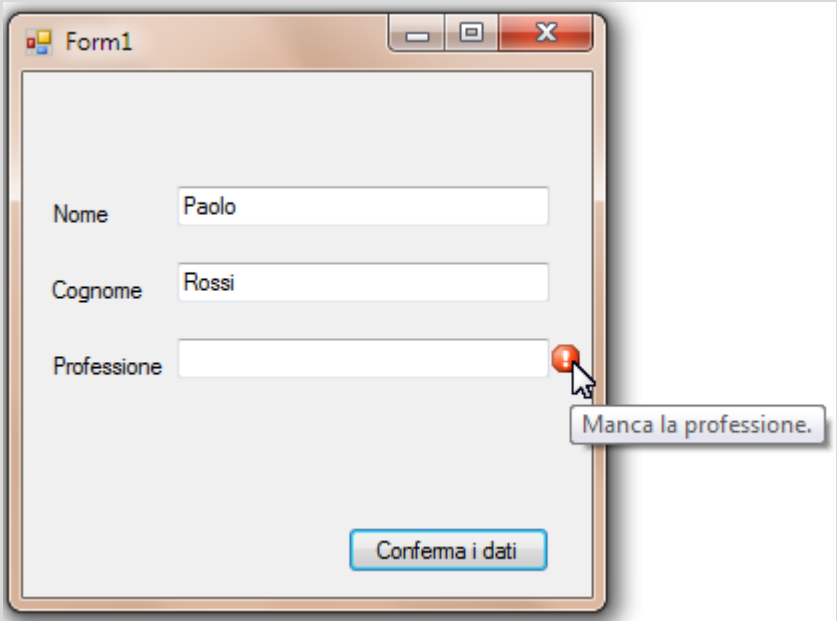
Inseriamo nel form tre componenti **ErrorProvider**, che saranno collegati, con il codice, ai tre TextBox.
Copiamo e incolliamo nella Finestra del Codice il listato seguente.

```
Public Class Form1

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        ' Associa gli ErrorProvider ai controlli, in posizione centrale e destra;
        ErrorProvider1.SetIconAlignment(TextBox1, ErrorIconAlignment.TopLeft)
        ErrorProvider2.SetIconAlignment(TextBox2, ErrorIconAlignment.TopLeft)
        ErrorProvider3.SetIconAlignment(TextBox3, ErrorIconAlignment.TopLeft)

    End Sub
```

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ' Cancella eventuali segnalazioni di errori precedenti:
        ErrorProvider1.Clear()
        ErrorProvider2.Clear()
        ErrorProvider3.Clear()

        ' Imposta i messaggi di errore associati ai tre TextBox e termina la
procedura:
        If TextBox1.Text = "" Then
            ErrorProvider1.SetError(TextBox1, "Manca il nome.")
            Exit Sub
        End If

        If TextBox2.Text = "" Then
            ErrorProvider2.SetError(TextBox2, "Manca il cognome.")
            Exit Sub
        End If

        If TextBox3.Text = "" Then
            ErrorProvider3.SetError(TextBox3, "Manca la professione.")
            Exit Sub
        End If

        ' Se non vi sono errori nei tre TextBox, la procedura continua:
        Dim Testo As String
        Testo &= TextBox1.Text & vbCrLf
        Testo &= TextBox2.Text & vbCrLf
        Testo &= TextBox3.Text & vbCrLf
        Testo &= "Confermi questi dati?"

        If MsgBox(Testo, MsgBoxStyle.YesNo + MsgBoxStyle.Question) =
MsgBoxResult.No Then
            TextBox1.Text = ""
            TextBox2.Text = ""
            TextBox3.Text = ""
        End If

    End Sub
```

```
End Class
```

Ecco un'immagine del programma in esecuzione: in questo esempio l'utente non ha completato tutti i dati, per cui il programma segnala il dato mancante con una icona che lampeggia per qualche istante:



Capitolo 38: INTERCETTAZIONE E CORREZIONE DEGLI ERRORI.

E' còmpito del programmatore provare in lungo e in largo il programma al quale sta lavorando, sforzandosi di mettersi nei panni dell'utente (che non sa nulla di come il programma funzioni), seguendo tutte le possibili varianti e tutti i possibili percorsi, forzando le opzioni ai livelli massimi o ai livelli minimi, immettendo dati numerici dove sono richiesti dati di testo, creando situazioni *assurde* o *impossibili*… in breve: il programmatore deve fare di tutto **per verificare che il suo programma non contenga errori o si blocchi durante il suo funzionamento**.

Quando il programma in esecuzione *gira* senza problemi è stato raggiunto un primo livello di rifinitura, in quanto sono già stati eliminati dal codice tutti gli errori di battitura o di sintassi.
Ma non è detto che il prodotto finale sia quello desiderato: il programma infatti potrebbe *girare* senza segnalare problemi, producendo tuttavia esiti molto diversi da quelli attesi.
Se si verificano esiti imprevisti, nel programma sono annidati comandi che producono effetti non voluti dal programmatore, oppure variabili i cui valori nel corso delle elaborazioni sfuggono a ciò che il programmatore aveva previsto, oppure, ancora, veri e propri errori di logica nella definizione del percorso che il programma dovrebbe seguire.

Gli errori nei quali può incorrere a un programmatore sono di due tipi:

- gli **errori di sintassi**, che VB è in grado di riconoscere e di segnalare in tempo reale al programmatore;
- gli **errori di programmazione** che consentono al programma di funzionare, ma che lo conducono a produrre effetti non voluti dal programmatore.

209: Errori di sintassi.

Sono errori di sintassi gli errori che si possono commettere scrivendo in modo sbagliato i termini del linguaggio di VB. Ecco alcuni esempi:

```
Diim Contatore As Integer
Dim Contatore As Interger
Foor Contatore = 0 To 12
```

Una scrittura prudente del codice e un uso accorto dei suggerimenti forniti di volta in volta da **IntelliSense** riducono al minimo il rischio di incorrere in questi errori; si tratta comunque di errori che non destano preoccupazioni, in quanto VB li riconosce immediatamente nella fase di scrittura del codice e li segnala al programmatore.
VB è in grado di segnalare in tempo reale anche altri tipi di errori, quali:

- la scrittura di righe di codice per un controllo che non è stato inserito nel form;
- la scrittura di cicli For… Next senza il Next conclusivo;
- la scrittura di comandi Select Case senza la riga conclusiva End Select;
- la scrittura di procedure Private Sub… senza la riga conclusiva End Sub;
- la scrittura di comandi If… Then senza la riga conclusiva End If;
- il rimando a linee di codice o a procedure inesistenti (ad esempio: `Call Correzione()`, quando la procedura Correzione non esiste o è denominata in altro modo).

Vediamo di seguito una panoramica degli errori più comuni, dei modi con i quali VB li segnala e delle modalità di correzione a disposizione del programmatore.

Zoom sul codice.

E' possibile in ogni momento ingrandire o ridurre la scrittura del codice, tenendo premuto il tasto CTRL e muovendo la rotellina del mouse.

Analizzare i colori del codice.

VB usa colori diversi per scrivere le diverse parti del codice.
Il programmatore ha la possibilità di modificare questi colori a piacere, dal menu **Strumenti \ Opzioni \ Ambiente \ Tipi di caratteri e colori**.
Se non sono sono state modificate le impostazioni predefinite, VB utilizza questi colori:

- il **colore blu** è riservato alle parole che compongono il linguaggio di VB, non modificabili dal programmatore;
- il **colore nero** è riservato alle variabili o alle procedure nelle parti modificabili dal programmatore;
- il **colore rosso** è riservato alle stringhe di testo scritte tra virgolette.

Il programmatore può quindi percepire la presenza di un errore vedendo un colore anomalo, come in questa immagine, in cui la parola Exit non compare colorata in blu, come normalmente dovrebbe essere, a causa di un errore di battitura:

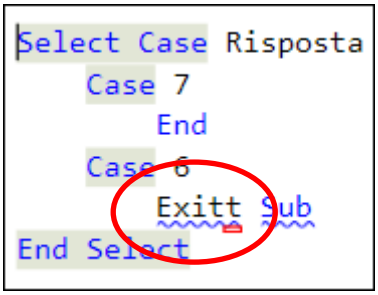


Figura 361: Segnalazione di un errore mediante il colore del codice.

Ancora sui colori del codice: quando si clicca il nome di un oggetto o di una variabile, in una procedura, tutte le occorrenze di questa parola sono evidenziate con il colore grigio. Il programmatore può dunque verificare se sono evidenziate tutte le parole che dovrebbero essere evidenziate, oppure se ne è rimasta esclusa qualcuna, scritta in modo diverso.

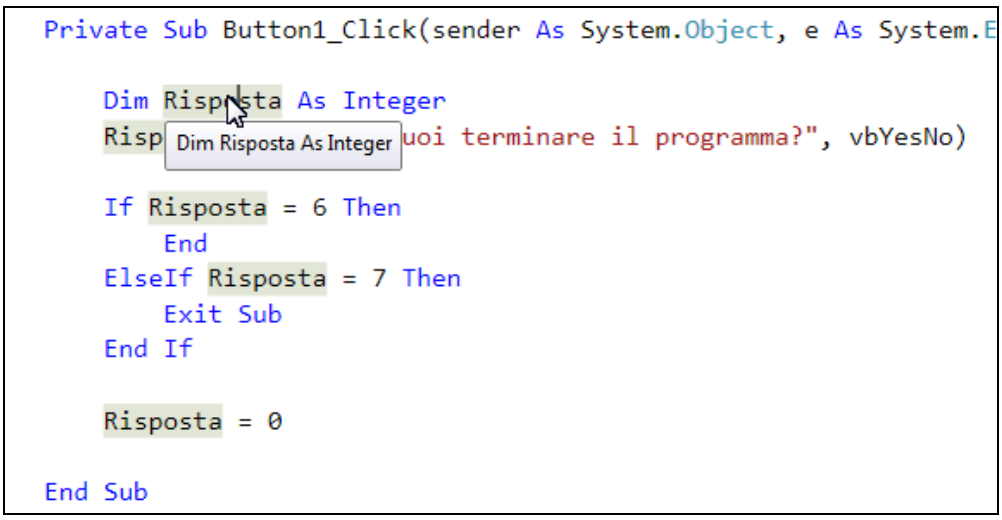
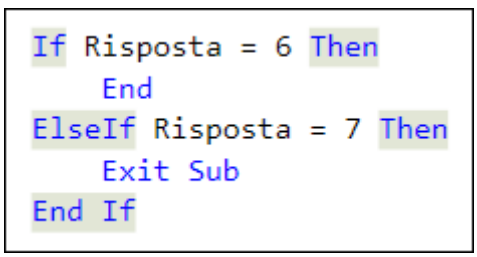


Figura 362: Evidenziazione di tutte le occorrenze della parola Risposta.

Allo stesso modo VB evidenzia tutte le parole cardine dei procedimenti decisionali basati su If… / End If e Select Case… / End Select:



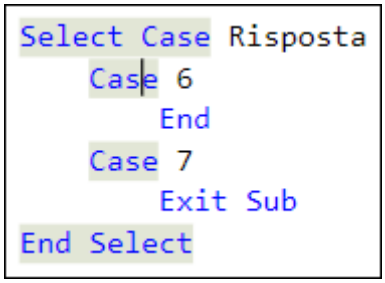


Figura 363: Evidenziazione dei procedimenti If... e Select Case.

Utilizzare le maiuscole.

Un metodo pratico, per evitare errori nella scrittura dei nomi delle variabili, è assegnare alle variabili nomi che contengano qualche lettera maiuscola.
Nel codice, il programmatore scrive questi nomi tutti in lettere minuscole: se IntelliSense riconosce le variabili provvede a correggere automaticamente il loro nome inserendo le lettere maiuscole dove occorre. Se la correzione automatica non avviene, ciò significa che IntelliSense non è in grado di riconoscere la variabile e dunque c'è un errore di battitura da parte del programmatore.
Vediamo un esempio: in un nuovo progetto, creiamo una variabile di tipo Integer con il nome **RisposteEsatte**.
Quando scriviamo questa variabile nel codice del programma con il nome **risposteesatte**, IntelliSense la riconosce e ne corregge automaticamente il nome in **RisposteEsatte**.
Se invece scriviamo il nome della variabile come **rispostesatte**, con una sola "e" al centro, IntelliSense non è in grado di riconoscerla e di inserirvi le lettere maiuscole: la mancanza di questa correzione da parte di IntelliSense segnala al programmatore l'errore di battitura.

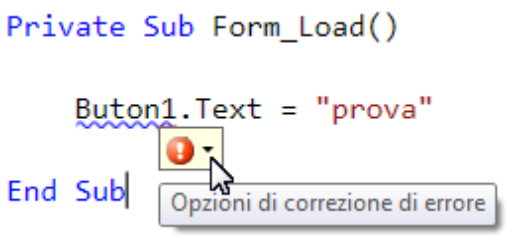
```
Public Class Form1

    Dim RisposteEsatte As Integer

    Private Sub Form1_Load(sender As Object, e As System.EventArgs) Handles
Me.Load

        RisposteEsatte = 0

        rispostesatte = 0

    End Sub

End Class
```

Errori nella scrittura dei nomi degli oggetti.

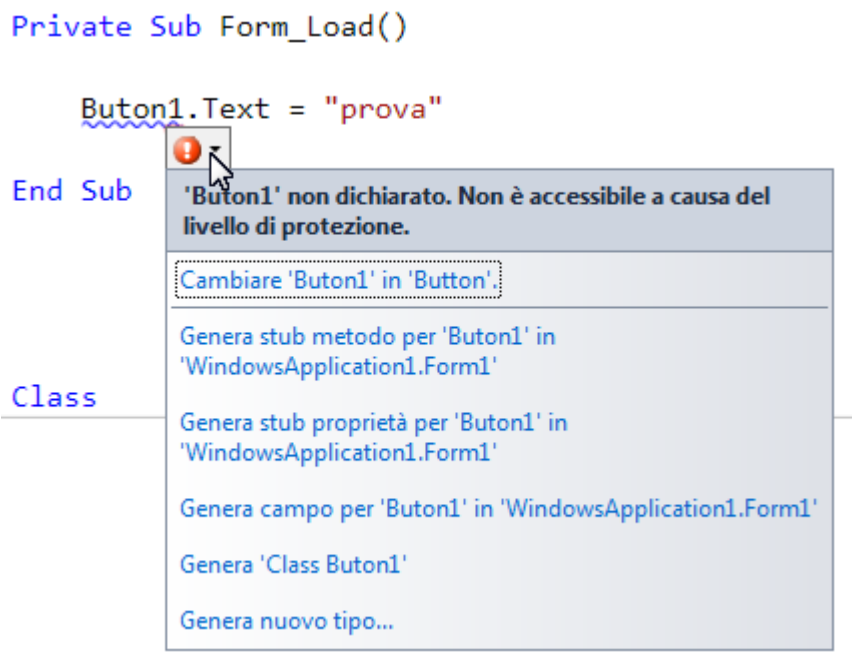
Un errore piuttosto comune è scrivere il nome di un controllo in modo sbagliato.
Supponiamo di avere inserito in un Form1 un pulsante Button al quale nella Finestra di Progettazione è stato dato il nome Button1.
Supponiamo ora che nella Finestra del Codice il programmatore scriva una procedura in cui il pulsante Button1 viene indicato con un errore di battitura:

```
Private Sub Form_Load()

    Buton1.Text = "prova"

End Sub
```

In questo caso IntelliSense, che non può capire a quale oggetto si riferisce il programmatore, segnala l'errore con la sottolineatura di una riga ondulata blu.
Questa segnalazione può essere sufficiente al programmatore per correggere l'errore.
In caso contrario, il programmatore può ricorrere al segno rosso che compare in coda alla linea ondulata blu: cliccando questo segno si accede alla finestra delle **Opzioni di correzione di errore**, fornite da VB:



Qui si possono trovare alcuni suggerimenti. Nel nostro esempio, il suggerimento contenuto nella prima riga - anche se non contiene la soluzione del problema - può fare capire al programmatore che ha commesso un errore di battitura scrivendo il nome del pulsante Button1:



Errori di punteggiatura.

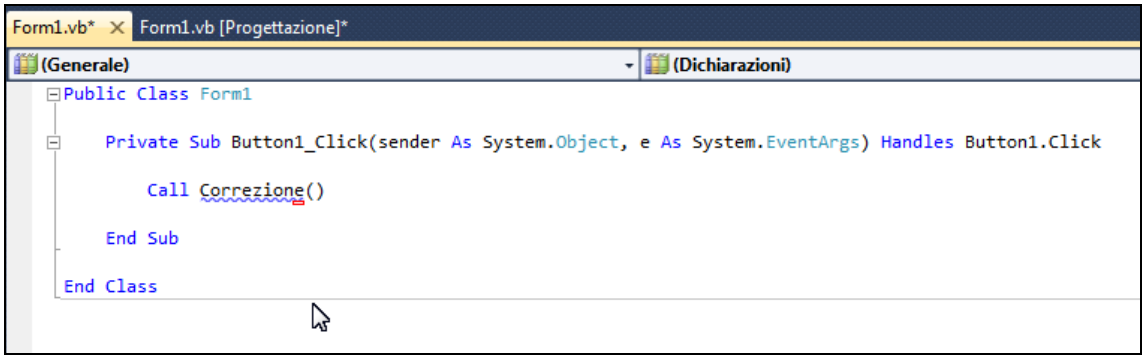
Un errore di sintassi può consistere nella dimenticanza del punto tra il nome di un oggetto e una sua proprietà:

```
Private Sub Form_Load()

    Button1Text = "Esci"

End Sub
```

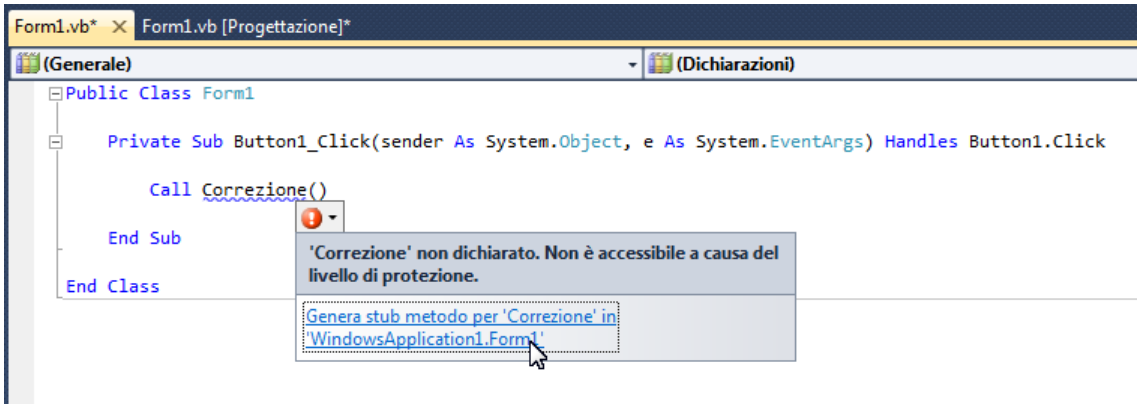
In questo caso VB considera la parola Button1Text, senza il punto separatore, come una parola unica e come il nome di un controllo non presente nel Form1: da qui, la segnalazione dell'errore con la linea blu ondulata.

Errori nei richiami tra procedure.

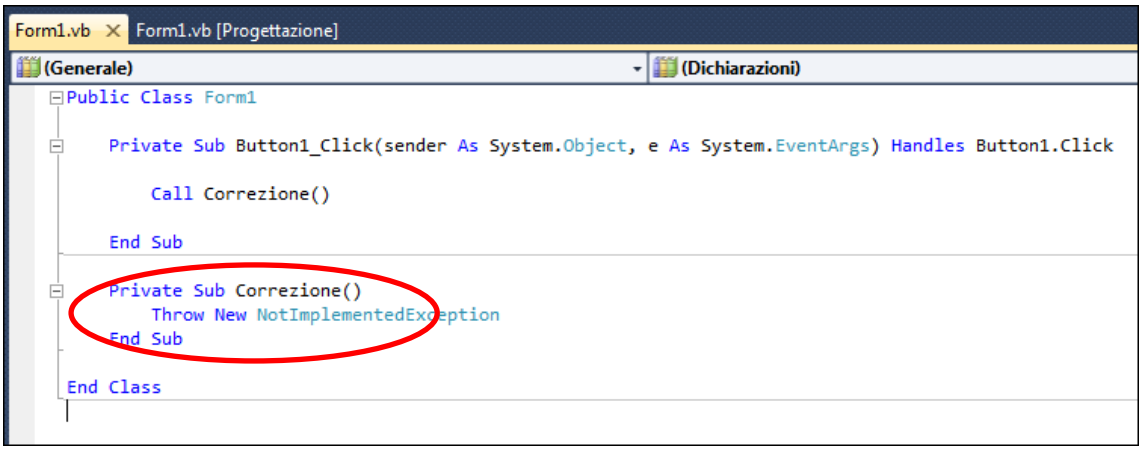
Un errore di sintassi può consiste nel richiamare una procedura che non esiste o che è stata denominata con un nome diverso. Nell'esempio che segue, l'evento *clic* sul Button1 attiva una procedura denominata "**Correzione**" che però nel codice non esiste.



In questo caso VB soccorre il programmatore proponendo di impostare in modo automatico la procedura che manca:



Accettando il suggerimento di VB, la procedura **Correzione** è impostata e aggiunta al codice automaticamente:



210: Errori di programmazione.

Per quanto riguarda gli errori di programmazione, invece, VB non è in grado di segnalare errori: l'ambiente di progettazione ovviamente non può conoscere le intenzioni del programmatore, ma è in grado di offrire strumenti per verificare quanto succede durante lo svolgimento di un programma.
In generale, per scoprire questi errori bisogna eseguire e analizzare il programma passo per passo, per controllare gli effetti che ogni procedura ha sugli oggetti e sulle variabili in uso nel programma.
VB offre numerose strumenti per effettuare questa analisi.
Di seguito ne vedremo tre, pratici ed efficaci, di uso comune:

- **inserire un punto di interruzione nel codice;**
- **effettuare il Debug con i tasti F8 o F10;**
- **usare la Finestra di controllo immediato.**

Ma, prima di ogni altra azione, può essere utile andare a controllare tutti i riferimenti a una determinata variabile esistente nel codice.
Facciamo un *clic* con il tasto destro del mouse sulla variabile che vogliamo controllare, poi un *clic* su **Trova tutti i riferimenti**. Si visualizza in questo modo una finestra con l'elenco delle righe del codice in cui la variabile è in azione.

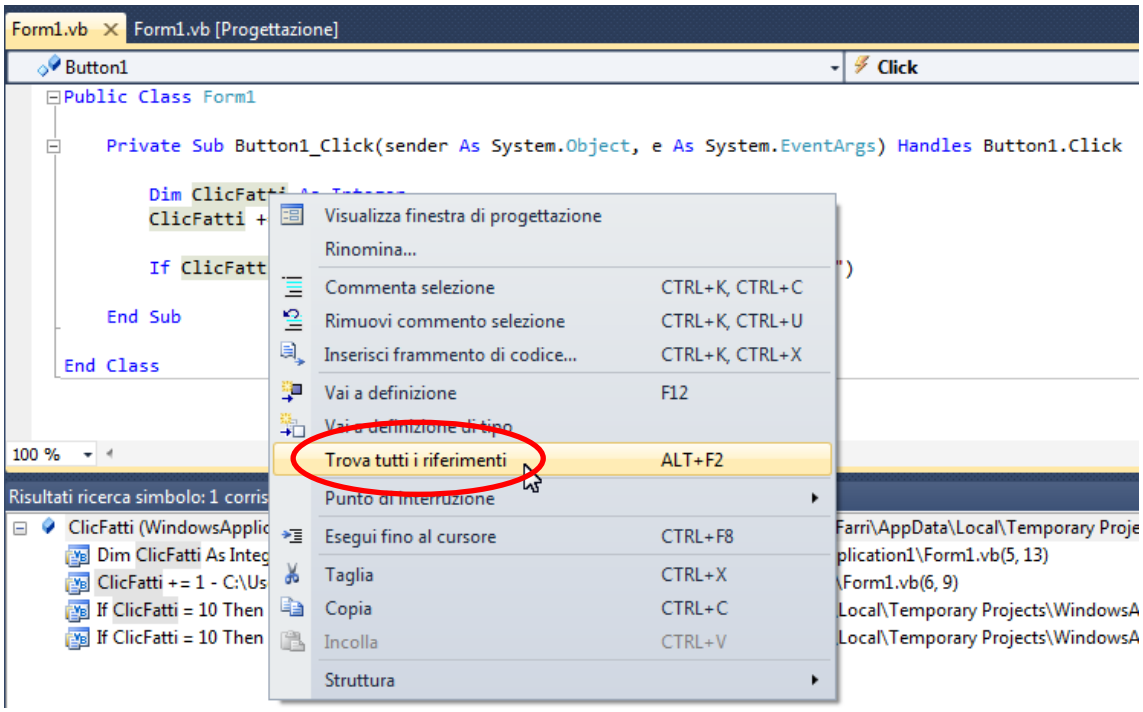


Figura 364: Visualizzazione di tutti i riferimenti a una variabile.

Qui il programmatore può avere una visione *panoramica* del campo di azione della variabile e farsi un'idea dei punti critici da tenere sotto osservazione.

Facendo un *clic* su una riga in questo elenco si accede direttamente alla riga corrispondente nel codice del programma dove, eventualmente, si può procedere con uno dei metodi presentati di seguito.

Inserire un punto di interruzione nel codice.

Un punto di interruzione è un segno (un pallino rosso) inserito nel codice a sinistra della riga di istruzioni che il programmatore vuole tenere sotto controllo:

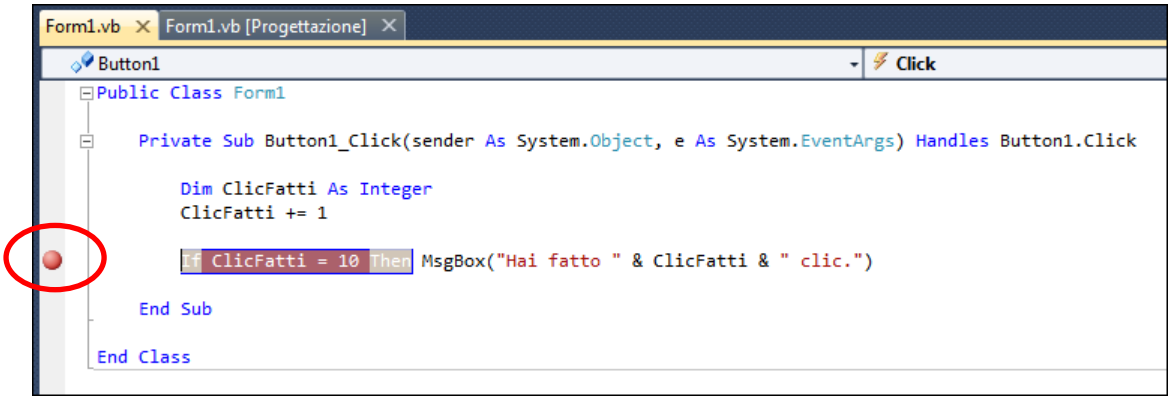


Figura 365: La collocazione di un punto di interruzione nel codice.

Nella fase di **Debug**, l'esecuzione del programma si interrompe quando giunge alla riga segnata dal punto di interruzione, e questo consente al programmatore di verificare la presenza di eventuali anomalìe.
Per collocare un punto di interruzione nel codice si può fare un *clic* con il mouse nel bordo a sinistra della riga interessata, oppure un *clic* sulla riga e un *clic* sul tasto **F9**. Le stesse operazioni eliminano un punto di interruzione già collocato nel codice.

Ecco un esempio di utilizzo di questa tecnica.
In un programma abbiamo un pulsante **Button1** al centro del Form1.
La sua proprietà **Text** è **Fai dieci clic**.
Funzionamento del programma: l'utente deve premere il pulsante Button1 con numerosi *clic* successivi. Quando i *clic* arrivano a dieci, il programma visualizza questo MsgBox:

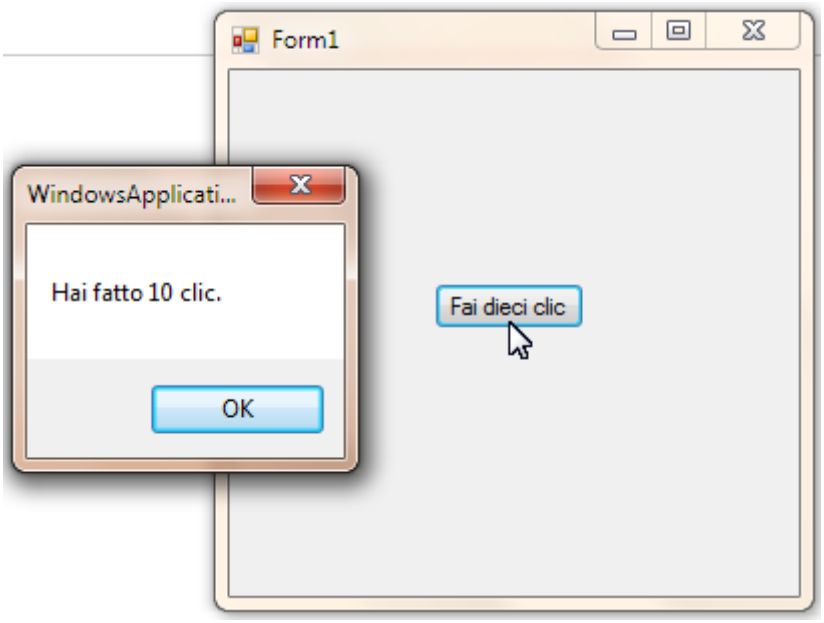


Figura 366: Un programma da testare con i punti di interruzione.

Ecco il listato del programma:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Dim ClicFatti As Integer
        ClicFatti += 1

        If ClicFatti = 10 Then MsgBox("Hai fatto " & ClicFatti & " clic.")

    End Sub

End Class
```

Mandiamo in esecuzione il programma con un clic sul pulsante **Avvia debug** o premendo il tasto **F5**. Notiamo che dopo avere fatto dieci *clic* con il mouse sul pulsante Button1… *non succede nulla* e, per quanti *clic* si facciano, continua a non succedere nulla.
Il listato è corretto dal punto di vista sintattico e il programma *gira* senza problemi, dunque siamo in presenza non di un errore di sintassi, ma di un errore di programmazione che VB non è in grado di intercettare.
Per scovare questo errore, collochiamo un punto di interruzione nel codice, a sinistra della riga in cui compare il procedimento If… Then… Quando sarà in esecuzione, giunto a questo punto il programma si fermerà e questo ci consentirà di controllare il contenuto della variabile ClicFatti:

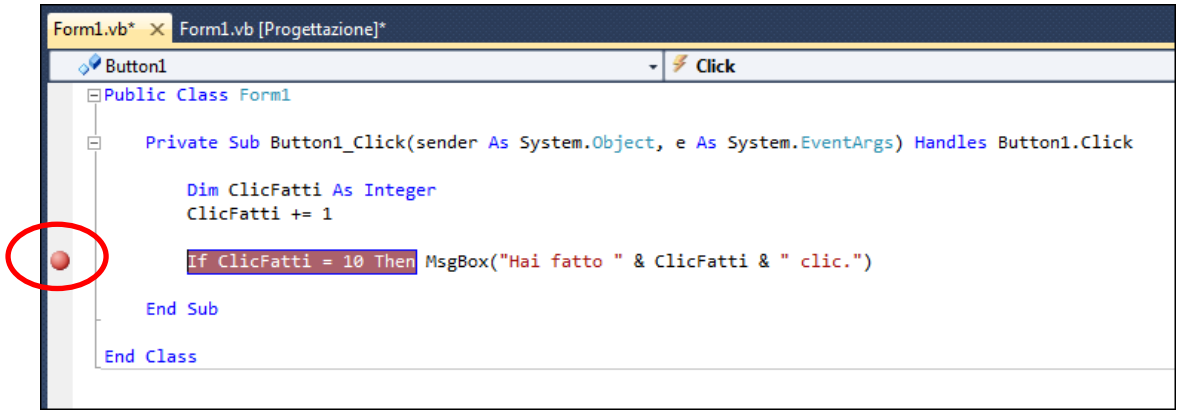


Figura 367: La collocazione di un punto di interruzione nel codice.

Mandiamo in esecuzione il programma e facciamo un clic sul pulsante Button1.
Quando il programma si interrompe perché è giunto al punto di interruzione, passiamo con il mouse sopra la variabile ClicFatti e leggiamo il suo valore.

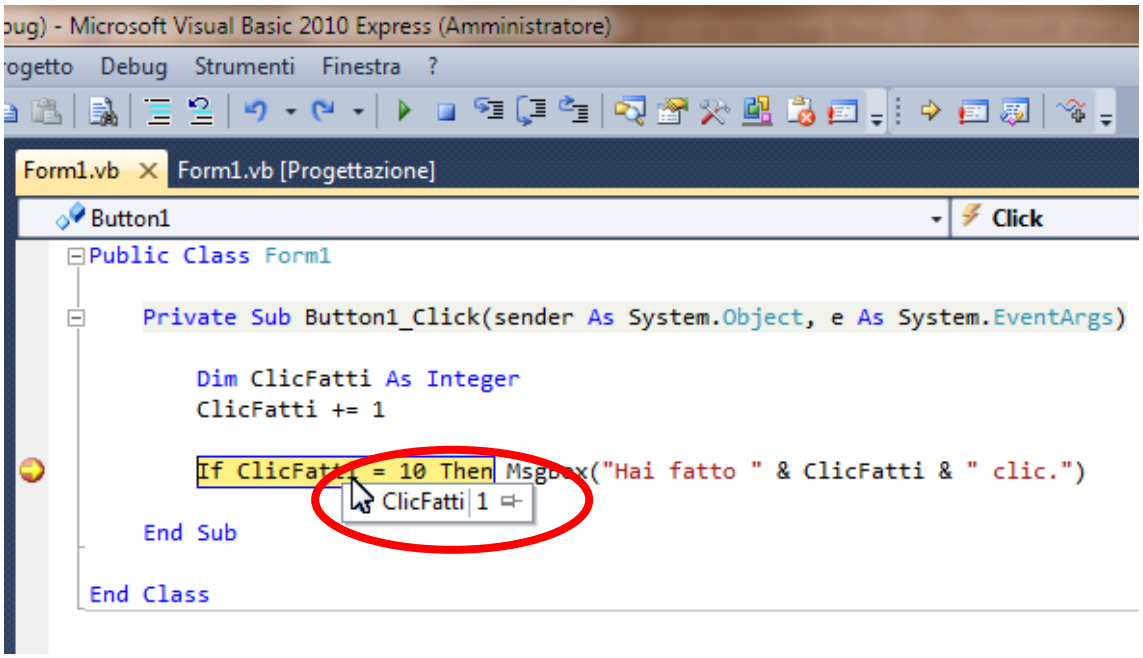


Figura 368: Lettura di una variabile durante l'interruzione di un programma.

Vediamo che la variabile ha il valore 1.
Facciamo proseguire il programma, con un clic sul pulsante **Continua**, nella barra delle icone, oppure premendo il tasto **F5**.
Facciamo un altro clic sul **Button1** e, alla successiva interruzione del programma, torniamo a controllare il contenuto della variabile ClicFatti: notiamo che esso non è aumentato di una unità, ma è ancora uguale a 1.
L'errore di programmazione dunque è qui: nel listato c'è *qualcosa* che re-inizializza la variabile ClicFatti in continuazione, riportandola sempre a zero.

Vediamo, infatti, che la creazione della variabile ClicFatti è contenuta nella procedura Button1.Click, per cui ad ogni *clic* del mouse questa variabile è *creata ex novo,* ogni volta con valore = 0.
Correggiamo l'errore spostando la creazione della variabile al di fuori della procedura, nell'àmbito di validità del Form1:

```
Public Class Form1

    Dim ClicFatti As Integer

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        ClicFatti += 1

        If ClicFatti = 10 Then MsgBox("Hai fatto " & ClicFatti & " clic.")

    End Sub

End Class
```

Togliamo il punto d'interruzione facendo un *clic* con il mouse sul pallino colorato, oppure con un clic sul menu **Debug \ Rimuovi punto di interruzione** o, ancora, un clic sulla riga e sul pulsante F9.
Mandando in esecuzione il programma, notiamo che il problema è stato risolto: a ogni *clic* del mouse il valore della variabile ClicFatti aumenta di una unità.

Effettuare il debug con il tasto F8 .

Invece di mandare in esecuzione un programma normalmente con un *clic* sul pulsante **Avvia debug** o con il tasto **F5**, è possibile fare un *clic* sul pulsante **Esegui istruzione**, oppure premere il tasto **F8**:

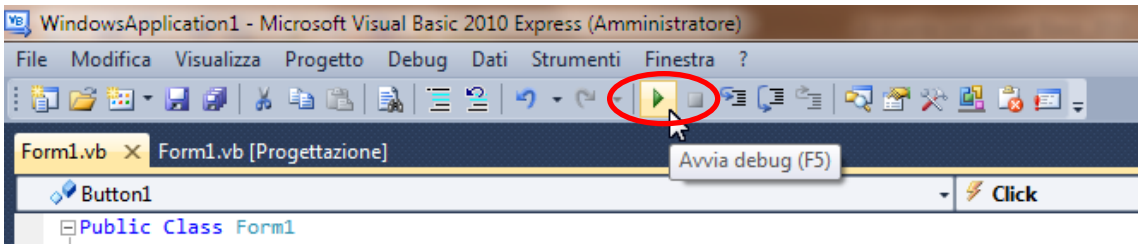


Figura 369: Il pulsante Avvia debug (F5).

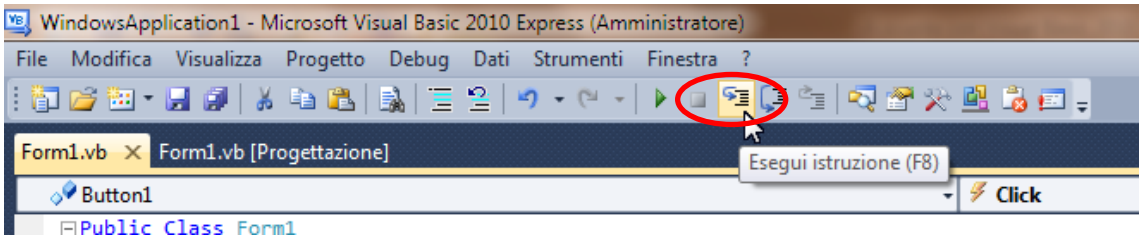


Figura 370: Il pulsante Esegui istruzione (F8).

Il pulsante **Esegui istruzione**, o il tasto **F8**, avviano l'esecuzione del codice **riga per riga**. Ad ogni clic su questo pulsante o ad ogni pressione del tasto **F8** il programma esegue una sola riga di istruzioni e si interrompe, anche senza la presenza di punti di interruzione.
Questo procedere riga per riga consente al programmatore di controllare attentamente lo stato di una variabile e i cambiamenti che esso subisce durante l'avanzamento del programma.
Premendo il tasto F10, invece, il programma limita l'esecuzione delle istruzioni riga per riga alla procedura corrente, in corso di esecuzione.

Usare la Finestra di controllo immediato.

Un altro strumento utile per verificare lo stato di una variabile durante la fase di debug di un programma è la **Finestra di controllo immediato**, che è visualizzata facendo un *clic* sul menu **Debug \ Finestre \ Controllo immediato**, oppure premendo i tasti **CTRL+G.**
In questa finestra possono essere visualizzate informazioni varie, sul funzionamento del programma, collocando nel codice, nei punti strategici, una riga di istruzioni con il comando **Debug.WriteLine()**.
I parametri del comando Debug.WriteLine(), da scrivere tra parentesi, sono due:
1. il nome della variabile da controllare;
2. un testo descrittivo definito dal programmatore

Ecco un esempio: il comando Debug.WriteLine inserito in questo listato scrive nella **Finestra di controllo immediato** il valore contenuto nella variabile ClicFatti ogni volta che l'utente fa un *clic* con il mouse sul pulsante Button1:

```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        Dim ClicFatti As Integer
        ClicFatti += 1

        Debug.WriteLine(ClicFatti, "Clic fatti")
```

```
        If ClicFatti = 10 Then MsgBox("Hai fatto " & ClicFatti & " clic.")

    End Sub

End Class
```

Facciamo un *clic* sul menu **Debug \ Finestre \ Controllo \ Controllo immediato**, oppure premiamo i tasti **CTRL+G** e mandiamo in esecuzione il programma.
Notiamo, nella **Finestra di controllo immediato** che il valore della variabile ClicFatti è sempre uguale a 1:

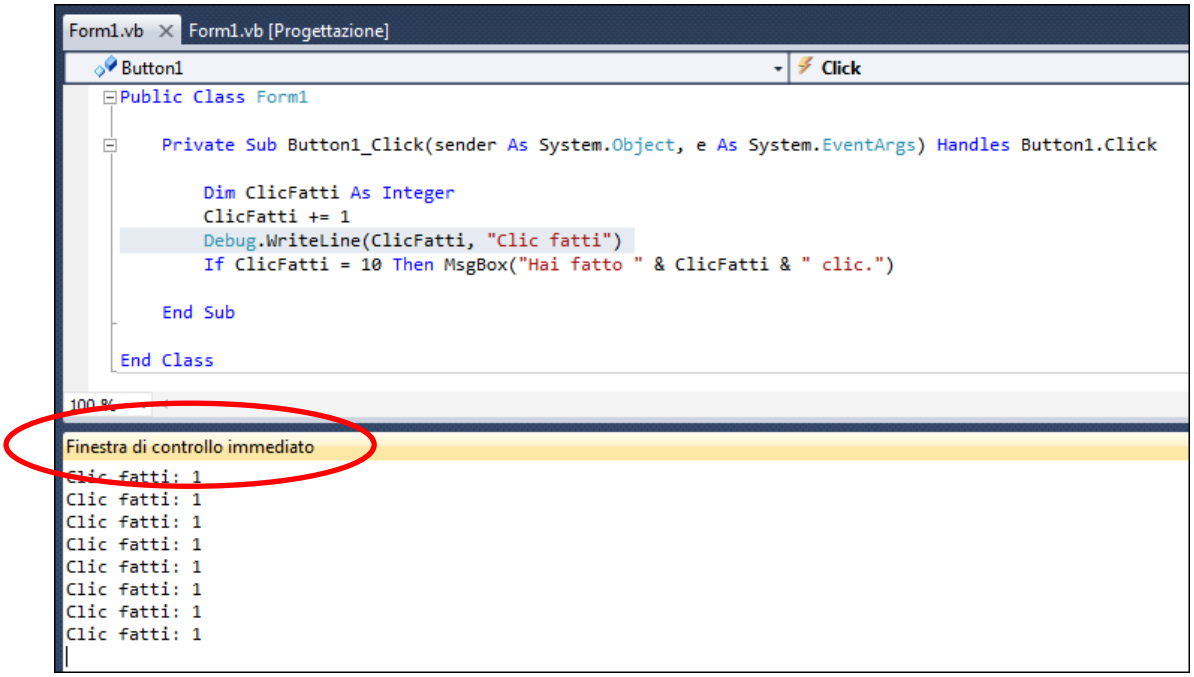


Figura 371; La Finestra di controllo immediato.

Una volta corretto l'errore spostando la creazione della variabile ClicFatti al di fuori della procedura Button1.Click, i suoi valori si susseguono nella **Finestra di controllo immediato** in modo corretto:

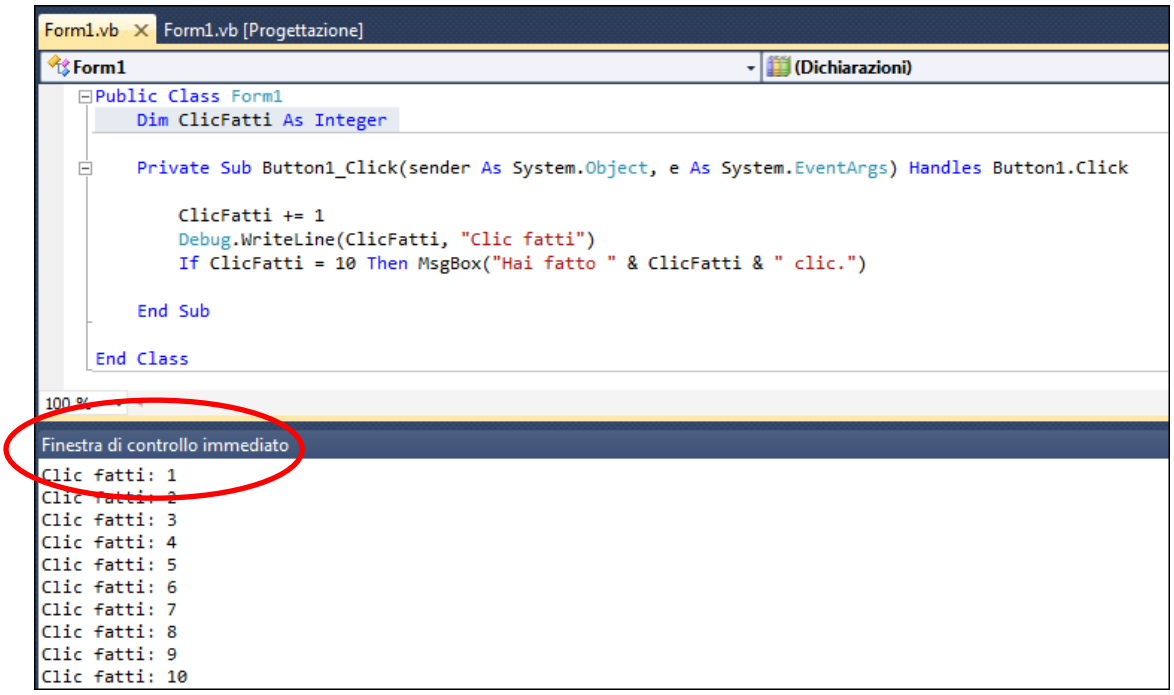


Figura 372; La Finestra di controllo immediato.

211: Metodi per evitare il blocco del programma.

Un errore imprevisto e dunque non gestito, all'interno di un programma, può portare al blocco del programma stesso e alla perdita dei dati creati dall'utente sino a quel momento.
Supponiamo, ad esempio, che in un programma in cui l'utente deve scrivere un elenco di indirizzi sia presente un errore nella procedura finale di salvataggio dei dati: in questo caso il programma si blocca e il lavoro effettuato dall'utente è perso.
Per prevenire questi rischi, il programmatore può inserire nel codice, nelle procedure che ritiene a rischio, i comandi

- **On Error Goto…** (in caso di errore continua il programma dal punto indicato);
- **On Error Resume Next** (in caso di errore continua il programma dalla prossima istruzione valida);

oppure un procedimento di tipo

- **Try… / Catch** (in caso di errore il programma visualizza un messaggio di errore e continua dalla prossima istruzione valida).

Si tratta di strumenti il cui uso è raccomandato nelle procedure che operano con **My.Computer.FileSystem** (cioè con i file presenti nel computer del programmatore o dell'utente): in questi casi un errore è sempre possibile, perché il programma si trova a operare in un ambiente non noto al programmatore.

Ecco un esempio di **On Error GoTo…** applicato a un programma con un controllo PictureBox e un pulsante Button inseriti nel form.
Premendo il pulsante Button1, il programma carica nel PictureBox l'immagine Gioconda.jpg:

```
    Private Sub Button1_Click() Handles Button1.Click

        PictureBox1.Image = Image.FromFile("C:/Gioconda.bmp")

    End Sub
```

Se l'immagine Gioconda.bmp non esiste nel percorso indicato ("C:/Gioconda.bmp"), il programma giunto a questo punto segnala l'errore e si blocca.
Si può eliminare questo rischio inserendo nella procedura i comandi **On Error Goto…** o **On Error Resume Next**:

```
    Private Sub Button1_Click(sender) Handles Button1.Click

        On Error GoTo Correzione

        PictureBox1.Image = Image.FromFile("C:/La Gioconda.bmp")

Correzione:

    End Sub
```

Oppure:

```
    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        On Error Resume Next

        PictureBox1.Image = Image.FromFile("C:/La Gioconda.bmp")

    End Sub
```

Try… Catch… End Try.

Il procedimento di tipo **Try… Catch… End Try** (letteralmente: *prova, cattura, termina* la prova) è un altro strumento utile per eliminare il rischio di un blocco del programma.
A differenza dei comandi **On Error** che abbiamo visto prima, con questo procedimento è possibile conoscere l'errore nel quale è incappato il programma ed è possibile informare l'utente, dopo di che il programma continua a funzionare dalla prossima istruzione valida.
Il procedimento si articola in questo modo:

```
        Try
            ' PROVA ad eseguire i comandi inseriti in questa parte del codice.

        Catch Errore As Exception

            ' CATTURA gli eventuali errori e visualizza il messaggio di errore.

        End Try
            ' TERMINA la PROVA e procedi.
```

Nell'esempio che segue vediamo un procedimento di tipo **Try… Catch… End Try** applicato allo stesso programma degli esempi precedenti (nel form sono inseriti un controllo PictureBox1 e un pulsante Button1):

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Try

            PictureBox1.Image = Image.FromFile("C:/Gioconda.bmp")

        Catch Errore As Exception

            MsgBox(Errore.Message, MsgBoxStyle.Critical + MsgBoxStyle.OkOnly)

        End Try

    End Sub

End Class
```

Ecco il messaggio di errore visualizzato dal programma:

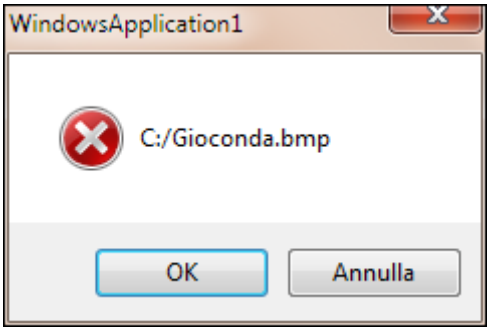


Figura 373: Un messaggio di errore prodotto da Try… Catch… End Try.

Il messaggio prodotto in modo automatico da VB può essere sostituito dal programmatore con un messaggio personalizzato:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Try

            PictureBox1.Image = Image.FromFile("C:/Gioconda.bmp")

        Catch Errore As Exception

            MsgBox("Si è verificato un errore, l'immagine richiesta non è
disponibile", MsgBoxStyle.Information + MsgBoxStyle.OkOnly)

        End Try

    End Sub

End Class
```

Ecco il messaggio di errore visualizzato dal programma:

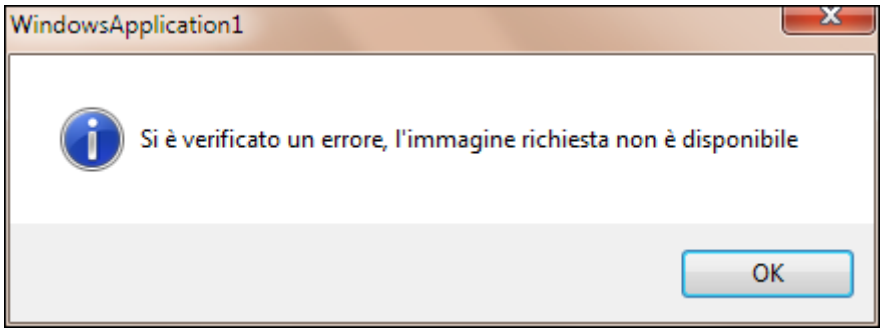


Figura 374: Un messaggio personalizzato di errore.

Con il procedimento Try… Catch… End Try è anche possibile inserire altri comandi nella procedura di gestione dell'errore.
In questo esempio, dopo avere visualizzato il messaggio di errore, il programma visualizza una croce nel controllo PictureBox oggetto dell'errore:

```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click

        Try

            PictureBox1.Image = Image.FromFile("C:/Gioconda.bmp")

        Catch Errore As Exception

            MsgBox("Si è verificato un errore, l'immagine richiesta non è
disponibile", MsgBoxStyle.Information + MsgBoxStyle.OkOnly)
            Dim SuperficieGrafica As Graphics = PictureBox1.CreateGraphics
            SuperficieGrafica.DrawLine(Pens.Red, New Point(0, 0), New Point(100,
100))
            SuperficieGrafica.DrawLine(Pens.Red, New Point(100, 0), New Point(0,
100))
        End Try


    End Sub

End Class
```

Ecco il risultato dell'azione eseguita dal programma dopo avere verificato la presenza di un errore:

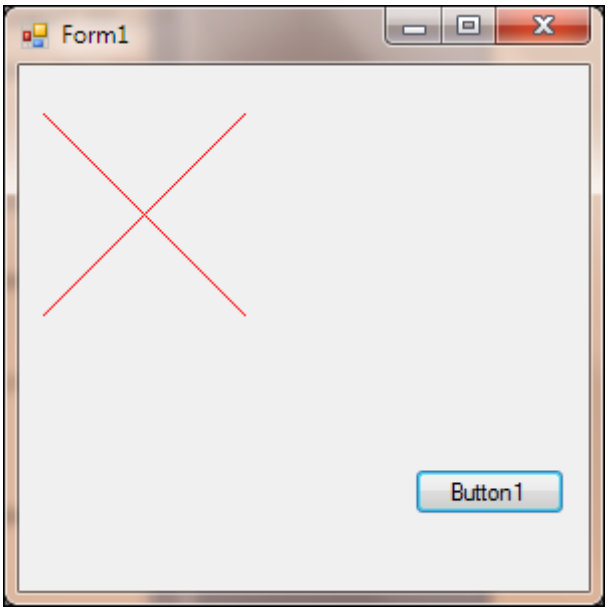


Figura 375: Inserimento di comandi in una procedura Try… Catch… End Try.

Liberare le risorse del computer.

Il blocco di un programma può essere causato anche da un impiego eccessivo delle risorse di memoria del computer.
Il codice può creare oggetti che richiedono risorse di memoria considerevoli, e, se non tenuti sotto controllo dal programmatore, questi possono superare la capienza della memoria disponibile nel computer e causare il blocco del programma o del computer.
Per evitare questo rischio, è buona norma **liberare le risorse** utilizzate, **eliminando** gli oggetti creati appena il loro uso è terminato:

- con il comando **Dispose**,
- o con il procedimento **Using… End Us**ing.

Dispose

Il comando per liberare direttamente le risorse di memoria occupate da oggetti è **Dispose**, che va riferito a ognuno degli oggetti da eliminare.
Ecco due esempi di creazione e successiva eliminazione di strumenti di grafica:

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        Dim Penna As New Pen(Color.Blue, 10)
        Dim Pennello As Brush = Brushes.Red

        e.Graphics.DrawArc(Penna, 10, 10, 200, 200, 180, 180)
        e.Graphics.FillEllipse(Pennello, 10, 10, 200, 200)

        ' Terminati i comandi grafici, le risorse di memoria vengono liberate:

        Penna.Dispose()
        Pennello.Dispose()

    End Sub

End Class
```

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Crea un'immagine virtuale
        Dim ImmagineVirtuale As Bitmap = Image.FromFile("C:\Triangolo.bmp")

        ' Altera l'immagine
```

```
        ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate90FlipX)

        ' Assegna l'immagine virtuale al Form1.
        e.Graphics.DrawImage(ImmagineVirtuale, New PointF(100, 100))

        ' Elimina l'immagine virtuale
        ImmagineVirtuale.Dispose()

    End Sub

End Class
```

Using… End Using

Un procedimento **Using… End Using** ha lo stesso effetto del comando **Dispose**.
Questo procedimento si apre con la creazione di un oggetto (**Using…** invece di **Dim**) e si conclude con la sua eliminazione (**End Using** invece di **Dispose**).
Vediamo i due esempi precedenti, riscritti con Using… End Using:

```
Public Class Form1

    Private Sub Form1_Paint(sender As Object, e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Crea un'immagine virtuale
        Using ImmagineVirtuale As Bitmap = Image.FromFile("C:\Triangolo.bmp")

            ' Altera l'immagine
            ImmagineVirtuale.RotateFlip(RotateFlipType.Rotate90FlipX)

            ' Assegna l'immagine virtuale al Form1.
            e.Graphics.DrawImage(ImmagineVirtuale, New PointF(100, 100))

            ' Elimina l'immagine virtuale
        End Using

    End Sub

End Class
```

```
Public Class Form1

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint

        ' Creazione di uno strumento Penna e di uno strumento Pennello:
        Using Penna As New Pen(Color.Blue, 10)
            Using Pennello As Brush = Brushes.Red

                e.Graphics.DrawArc(Penna, 10, 10, 200, 200, 180, 180)
```

```
                e.Graphics.FillEllipse(Pennello, 10, 10, 200, 200)

                ' Terminati i comandi grafici, le risorse di memoria vengono
liberate:

            End Using
        End Using

    End Sub

End Class
```

Capitolo 39: DISTRIBUIRE IL PRODOTTO FINITO.

Dopo avere provato il programma in tutte le sue opzioni e ramificazioni e dopo averne corretto gli eventuali errori, al programmatore restano da fare due operazioni:

- **compilare** il programma, per renderlo eseguibile anche al di fuori dell'ambiente di programmazione di Visual Basic;
- **creare un pacchetto di distribuzione** perché gli utenti possano ricevere, installare e usare il programma sui loro computer.

212: Compilare il programma.

Il linguaggio macchina[94] di un computer non è in grado di leggere il listato di un progetto scritto in VB, per cui è necessaria una **traduzione** da un linguaggio all'altro.
Lo strumento che si incarica di questa traduzione è il **programma compilatore**.
Il compilatore nel suo lavoro di traduzione rivede il listato e ne controlla la sintassi, le variabili, le procedure; si incarica inoltre di esaminare e collegare i file eventualmente necessari per il funzionamento del progetto.
In caso di errori, il compilatore interrompe il suo lavoro e segnala l'errore al programmatore.
Il prodotto finale del lavoro del compilatore è un file **eseguibile**, che ha il nome scelto dal programmatore e l'estensione .exe (ad esempio: Tabelline.exe).
Un progetto **compilato** non è più un progetto interno a VB, ma è un software finito e autonomo, eseguibile con un *clic* del mouse sulla sua icona anche in computer nei quali l'ambiente di progettazione di VB non è presente.

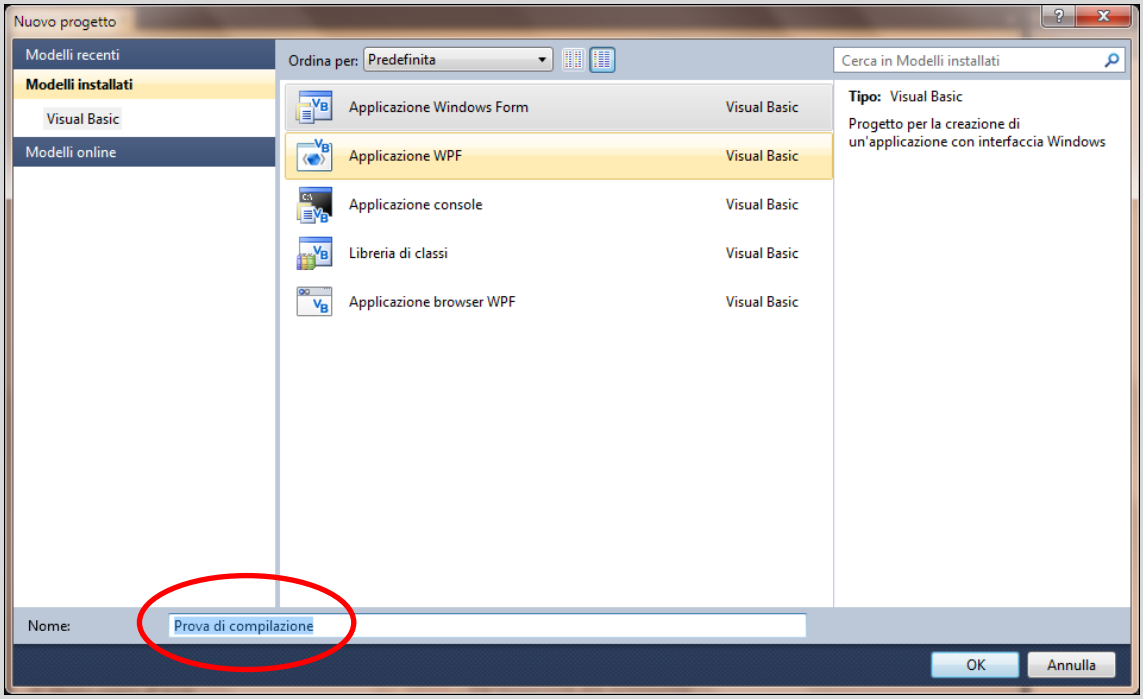
E' tuttavia richiesta, sul computer destinatario, la presenza della FrameWork alla quale VB fa riferimento, cioè l'archivio di software di base necessari per il funzionamento dei programmi in ambiente Windows.
Questa FrameWork è normalmente installata sui computer assieme al sistema operativo; nel caso non fosse presente nel computer ospite, l'utente del programma dovrà procurarsela e installarla (è liberamente scaricabile da internet).
Alcuni pacchetti di installazione provvedono automaticamente a verificare se la FrameWork esiste o no, e attivano la procedura per scaricarla da internet.

[94] Si veda il paragrafo 14: I linguaggi compilatori. 59.

Al momento di chiudere questo manuale la FrameWork più recente per i sistemi Windows è la **Microsoft.NET FrameWork 4**.
Nel prossimo esercizio vedremo un esempio di compilazione di un progetto.

Esercizio 141: Compilazione di un progetto.

Apriamo un nuovo progetto. Gli diamo il nome **Prova di compilazione**:



Proprietà del form:

- **BackgroundImage** = scegliamo l'immagine **orchestra e coro**, che si trova nella cartella **Documenti \ A scuola con VB \ Immagini.**
- **FormBorderStyle** = FixedToolWindow
- **Icon =** scegliamo l'icona **note**, che si trova nella cartella **Documenti \ A scuola con VB \ Immagini \ Icone**.
- **StartPosition** = CenterScreen
- **Text** = Prova di compilazione

Inseriamo nelle risorse del programma il file **alleluia.wav**, che si trova nella cartella **Documenti \ A scuola con VB \ Suoni**.
Inseriamo nel form un pulsante Button1, con queste proprietà:

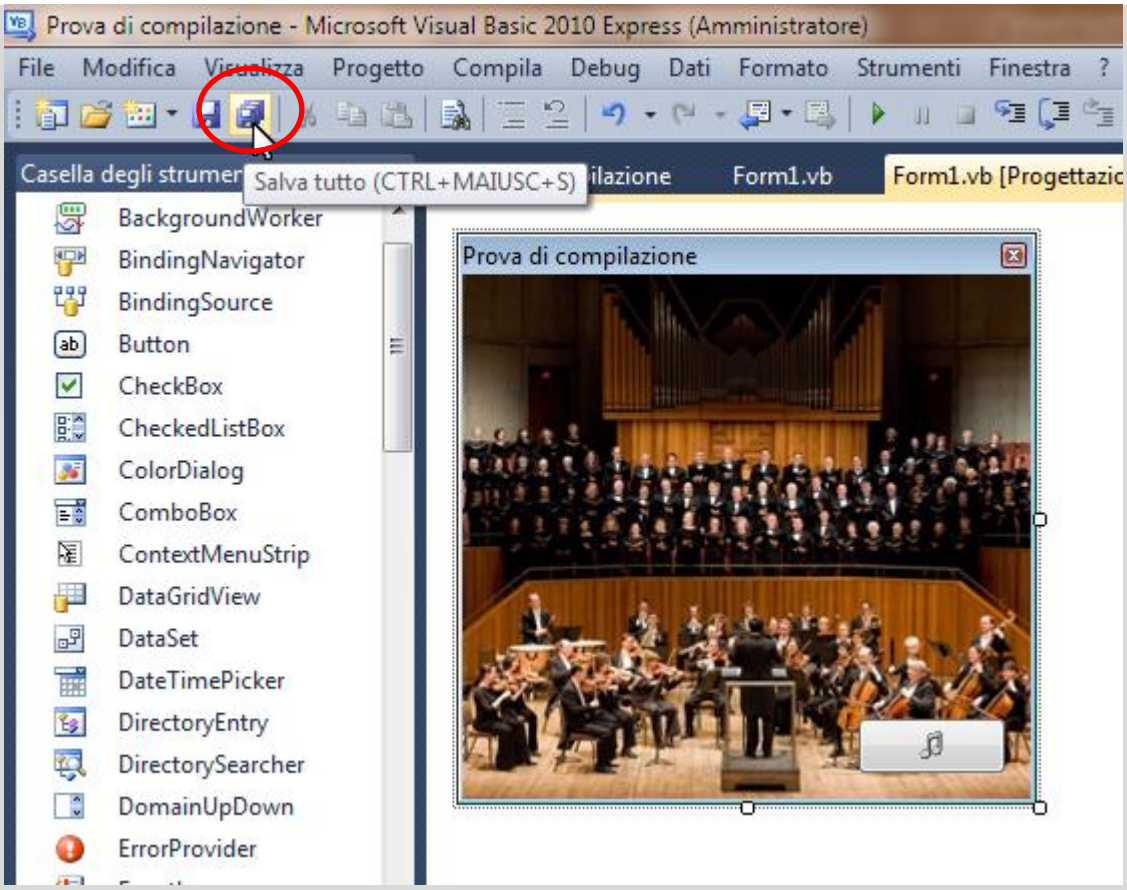
- **Cursor** = Hand
- **Text =** (nulla, cancellare la scritta Button1)
- **Immagine =** scegliamo la stessa icona (**note**) scelta per il form.

Ecco come si presenta l'interfaccia del programma:

Ora copiamo e incolliamo nella Finestra del Codice questo listato:

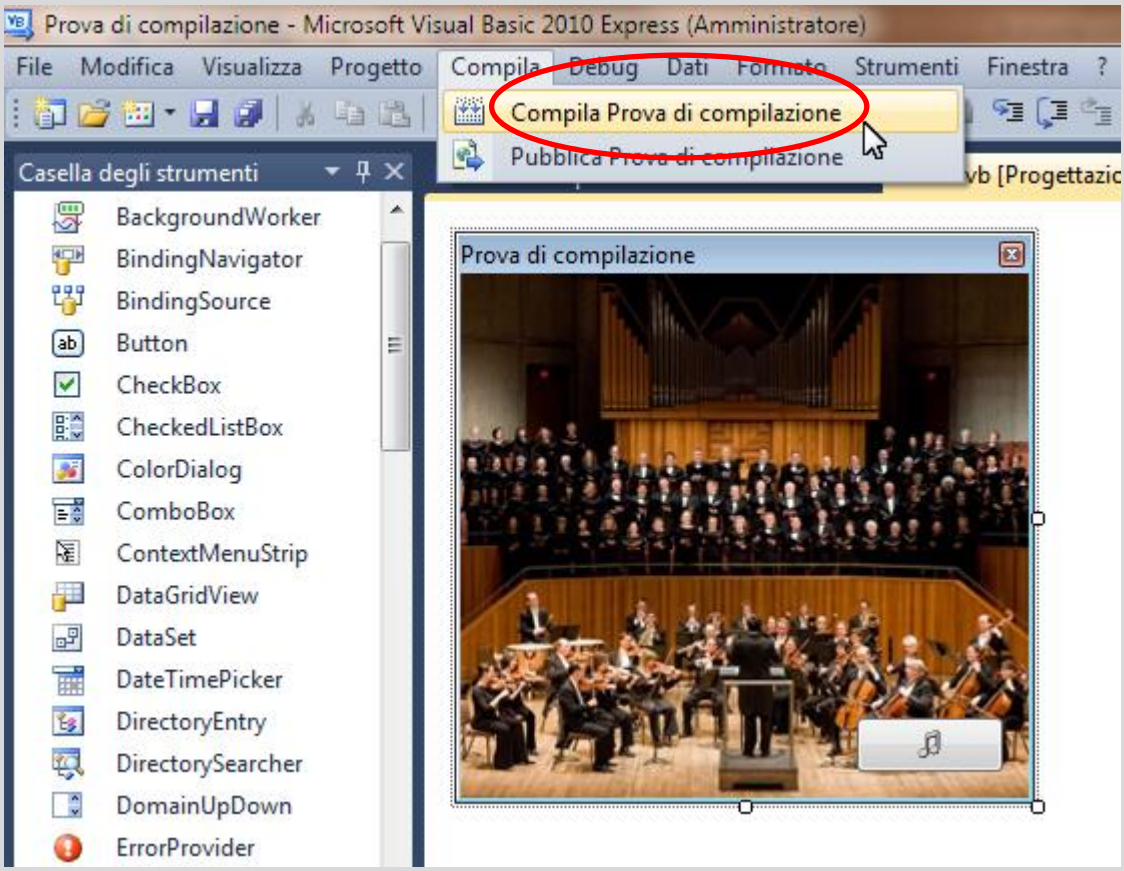
```
Public Class Form1

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
Handles Button1.Click

        My.Computer.Audio.Play(My.Resources.alleluia, AudioPlayMode.Background)

    End Sub

End Class
```

Mandiamo in esecuzione il programma: con un *clic* sul pulsante, l'utente può ascoltare le prime note dell'Alleluia dal Messia di G. F. Handel.
Salviamo il progetto con un *clic* sull'icona **Salva tutto**, nella barra delle icone di VB:

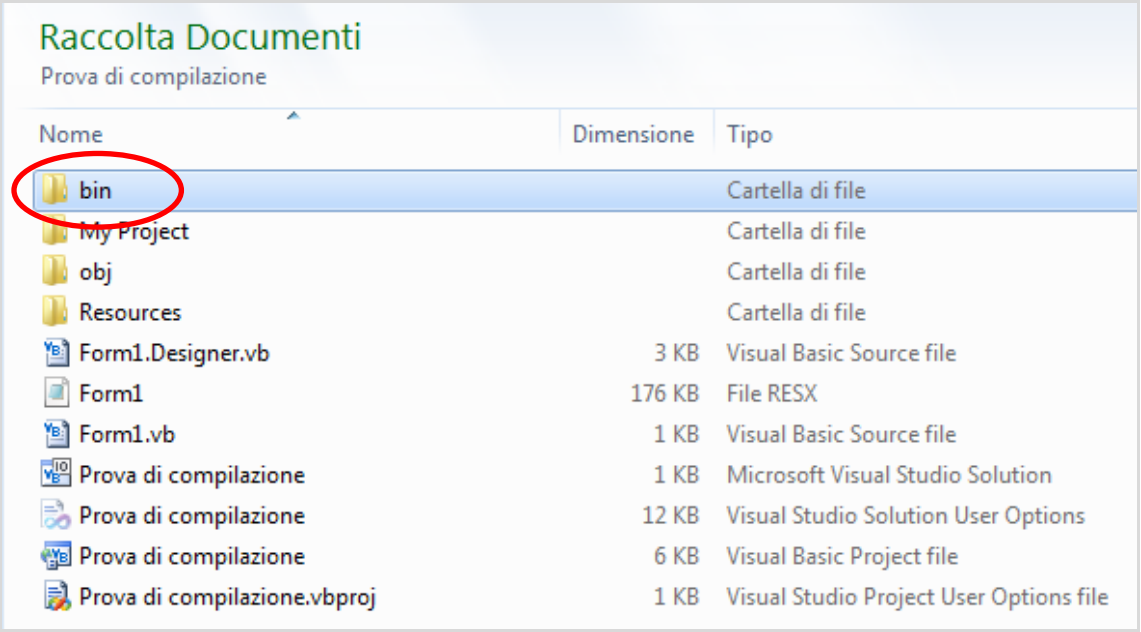


Se non modifichiamo le impostazioni predefinite di VB, il progetto è salvato nella cartella **Documenti \ Visual Studio 2010 \ Projects \ Prova di compilazione**. Procediamo ora alla sua compilazione, cioè alla creazione di un file eseguibile che potrà essere trasferito ed eseguito su altri computer.

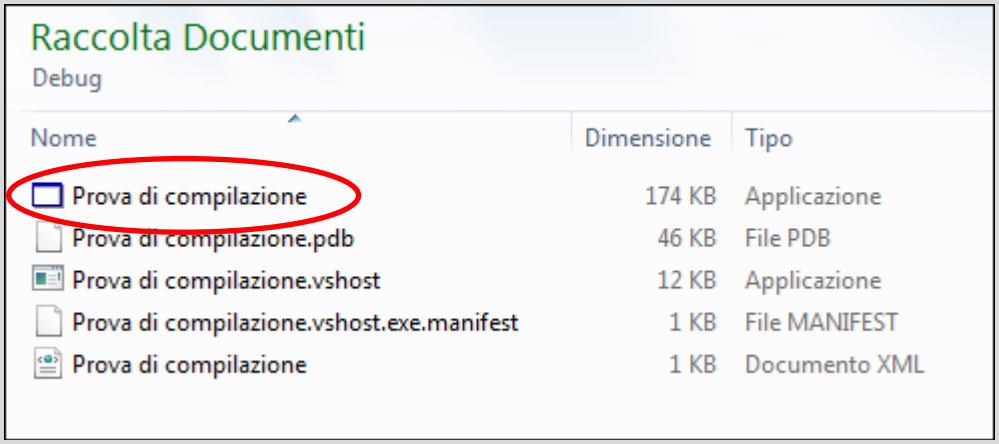
Nella striscia dei menu di VB, facciamo un *clic* su **Compila** e poi su **Compila Prova di compilazione**. Notiamo che nello stesso menu è presente il comando **Pubblica Prova di compilazione**, che servirà nella prossima fase della creazione del pacchetto di distribuzione:



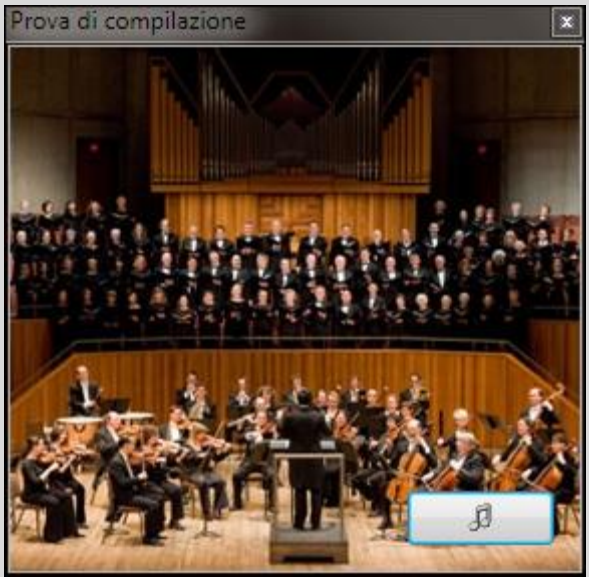
Eseguita la compilazione, scorriamo la cartella in cui è salvato il progetto **(Documenti \ Visual Studio 2010 \ Projects \ Prova di compilazione)**:



Nella sottocartella **bin \ Debug** troviamo il file eseguibile **Prova di applicazione.exe**. Questo è il programma che incorpora i file immagine e il file audio del progetto, e può essere eseguito su altri computer:



Possiamo provarlo: lo mandiamo in esecuzione direttamente con un doppio *clic* del mouse sul suo nome:



Notiamo che il programma così compilato funziona in modo autonomo da VB. Anche dopo avere chiuso l'ambiente di progettazione di VB, vediamo che con un doppio *clic* sul file **Prova di applicazione.exe** il programma funziona normalmente, senza visualizzare la Finestra di Progettazione, la Finestra del Codice, la finestra Esplora soluzione e le altre risorse di VB.

213: Distribuire il programma.

Dopo avere creato, provato, corretto e compilato il programma è giunto il momento di **creare un pacchetto di distribuzione** del file agli utenti.
Creare un pacchetto di distribuzione di un programma significa svolgere queste operazioni:

- **comprimere** il programma in modo che occupi la minore quantità possibile di memoria;
- **creare un file installatore**, che trasferirà nel computer dell'utente il programma e tutti i files ad esso collegati.

VB offre la possibilità di creare in modo automatico queste operazioni con il procedimento **ClickOnce** (clicca una volta sola) che, oltre a essere di uso molto semplice, offre la certezza che ogni file necessario al funzionamento del programma sia inserito nel pacchetto di distribuzione.
Con **ClickOnce** il programmatore può scegliere fra tre modi diversi di distribuzione e installazione del suo programma:

1. **Installazione da internet:**
   L'applicazione verrà collocata su un computer remoto che si trova in rete e da qui gli utenti potranno scaricarla, cliccando un link sulla pagina di un sito internet. Con questo metodo l'applicazione è scaricata, installata e avviata automaticamente sul computer dell'utente.

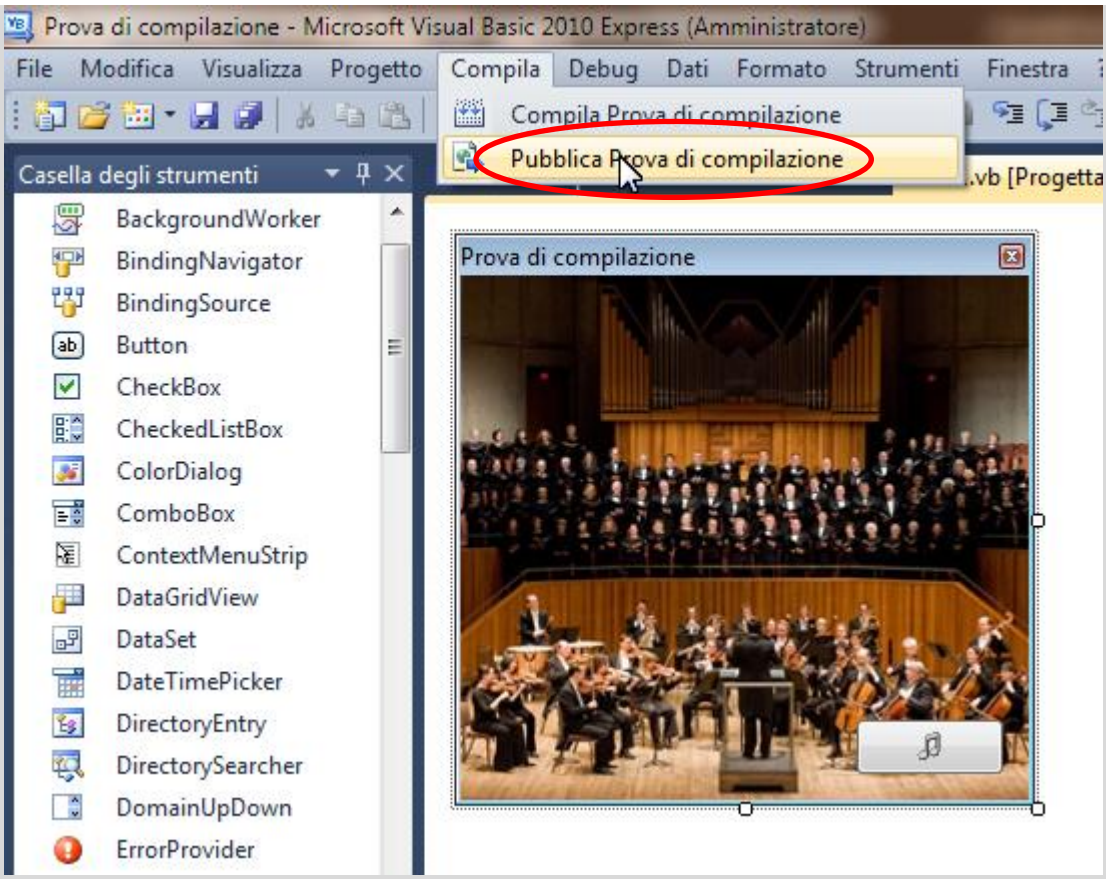
2. **Utilizzo dell'applicazione da internet:**
   L'utente può **usare** l'applicazione cliccando un link su una pagina di un sito internet, ma quando il programma viene chiuso esso non è più disponibile nel suo computer. Con questo metodo l'applicazione è scaricata e collocata in una cartella temporanea nel computer dell'utente; alla chiusura dell'applicazione la cartella temporanea è cancellata.

3. **Installazione da CD-ROM o DVD-ROM:**
   L'utente riceve un CD-ROM o DVD-ROM che contiene il file installatore della applicazione. Facendo un doppio clic su file, che in molti casi ha il nome **setup.exe**, si ottiene l'installazione dell'applicazione in questione nel computer dell'utente. Al termine dell'operazione l'utente ha nel suo computer, in modo stabile, l'applicazione che lo interessa, con i collegamenti sul desktop e nel menu Start \ Programmi per avviarla.

Le prime due modalità richiedono che il programmatore abbia accesso al server di un sito internet. La terza modalità è invece a portata di tutti.
Nel prossimo esercizio vedremo un esempio di questa terza modalità, per creare un pacchetto di distribuzione con **ClickOnce.**

Esercizio 142: Creazione di un pacchetto di distribuzione.

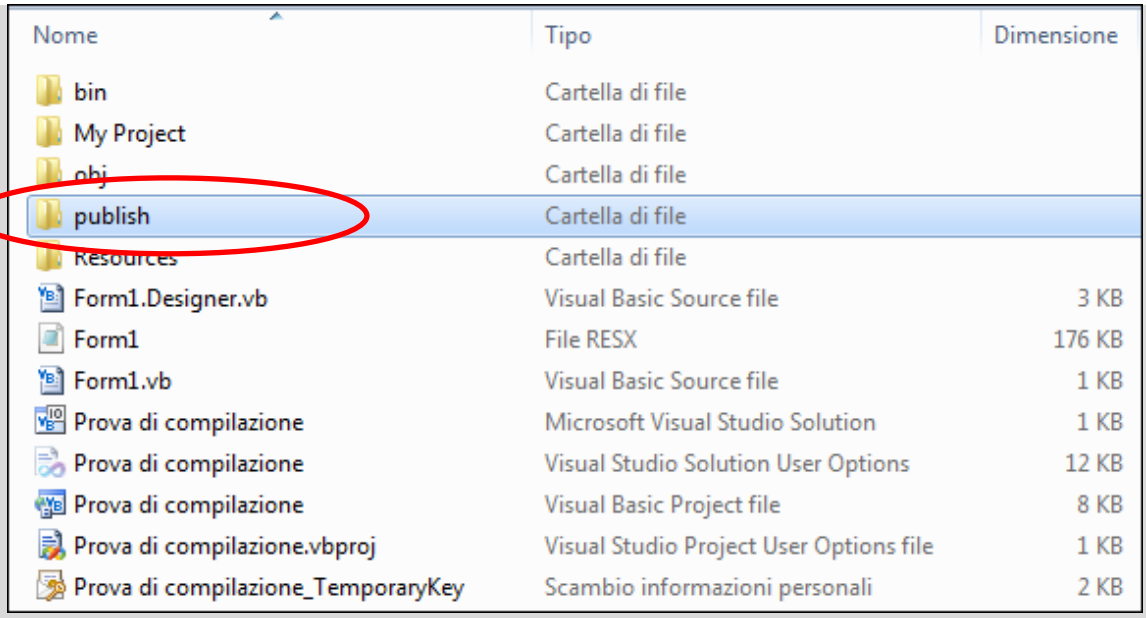
Riprendiamo il programma dell'esercizio precedente.
Nella barra dei menu testuali, in alto, facciamo un *clic* su **Compila / Pubblica Prova di compilazione**:



Accettiamo, senza modificarle, le opzioni che vengono offerte dalle finestre che si susseguono, per cui:

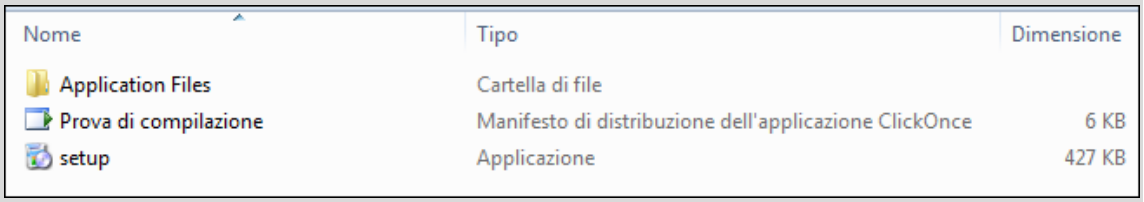
- l'applicazione verrà pubblicata nella sottocartella **publish**;
- il programma sarà installato da CD-ROM o DVD-ROM;
- non sarà richiesta la ricerca di aggiornamenti.

Terminata la sequenza di queste finestre, possiamo verificare nella cartella **Documenti \ Visual Studio 2010 \ Projects \ Prova di compilazione** la presenza di una nuova sottocartella di nome **publish:**



Nome	Tipo	Dimensione
bin	Cartella di file	
My Project	Cartella di file	
obj	Cartella di file	
publish	Cartella di file	
Resources	Cartella di file	
Form1.Designer.vb	Visual Basic Source file	3 KB
Form1	File RESX	176 KB
Form1.vb	Visual Basic Source file	1 KB
Prova di compilazione	Microsoft Visual Studio Solution	1 KB
Prova di compilazione	Visual Studio Solution User Options	12 KB
Prova di compilazione	Visual Basic Project file	8 KB
Prova di compilazione.vbproj	Visual Studio Project User Options file	1 KB
Prova di compilazione_TemporaryKey	Scambio informazioni personali	2 KB

All'interno della cartella **publish** troviamo questi file che, ai fini della distribuzione del programma, sono da copiare su un CD-ROM o un DVD-ROM:



Nome	Tipo	Dimensione
Application Files	Cartella di file	
Prova di compilazione	Manifesto di distribuzione dell'applicazione ClickOnce	6 KB
setup	Applicazione	427 KB

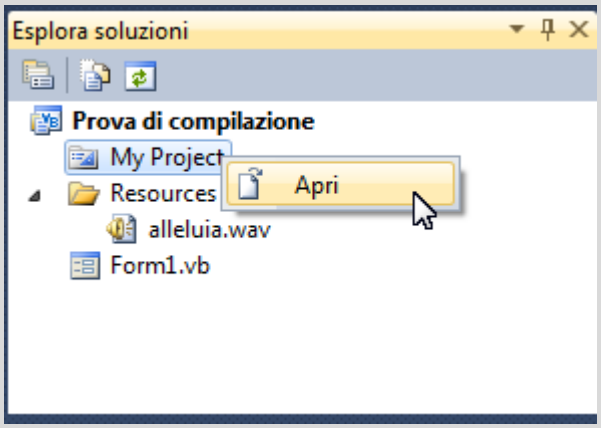
Cliccando il file **setup**, si procede alla installazione del programma sul computer ospite. Il file **setup** verifica innanzitutto che in questo computer sia presente l'archivio di software di base FrameWork. Se questo non è presente, il file di setup segnala l'esigenza di scaricarlo da internet e di installarlo, per poi completare l'installazione del programma.
A installazione ultimata, un collegamento a **Prova di compilazione** è aggiunto al menu **Avvio \ Programmi \ Microsoft** e il programma compare tra le applicazioni elencate nel Pannello di Controllo, da dove potrà eventualmente essere disinstallato con la procedura di disintallazione offerta dal sistema operativo.

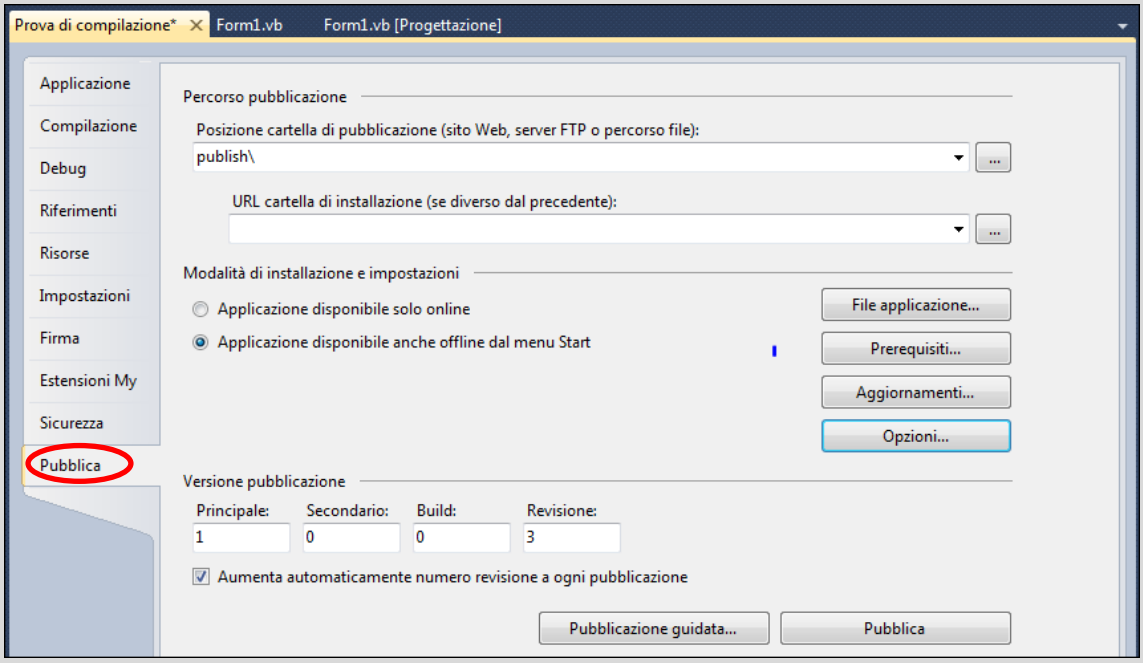
Una caratteristica apprezzabile di questo tipo di installazione consiste nel fatto che il file eseguibile non è collocato nella cartella Programmi del computer dell'utente, ma in un'area separata direttamente accessibile; il sistema operativo, dunque, non rileva problemi di protezione o eccezioni, anche se l'utente che effettua l'installazione non è l'amministratore del computer.

Nel prossimo esercizio ripeteremo la creazione di un pacchetto di distribuzione, sfruttando alcune opzioni per la sua personalizzazione.

Esercizio 143: Creazione di un pacchetto di distribuzione personalizzato.

Riprendiamo il programma utilizzato negli esercizi precedenti.
Per inserire alcuni elementi di personalizzazione nel pacchetto di installazione, invece di fare un *clic* sul menu **Compila / Pubblica Prova di compilazione**, apriamo la Finestra Proprietà del progetto:

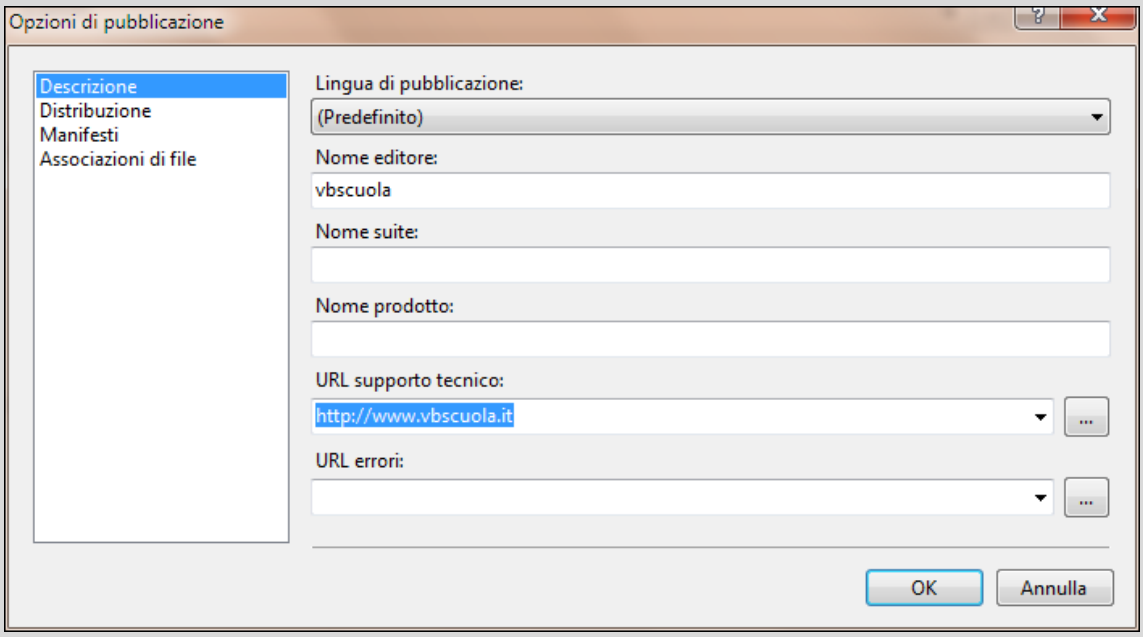




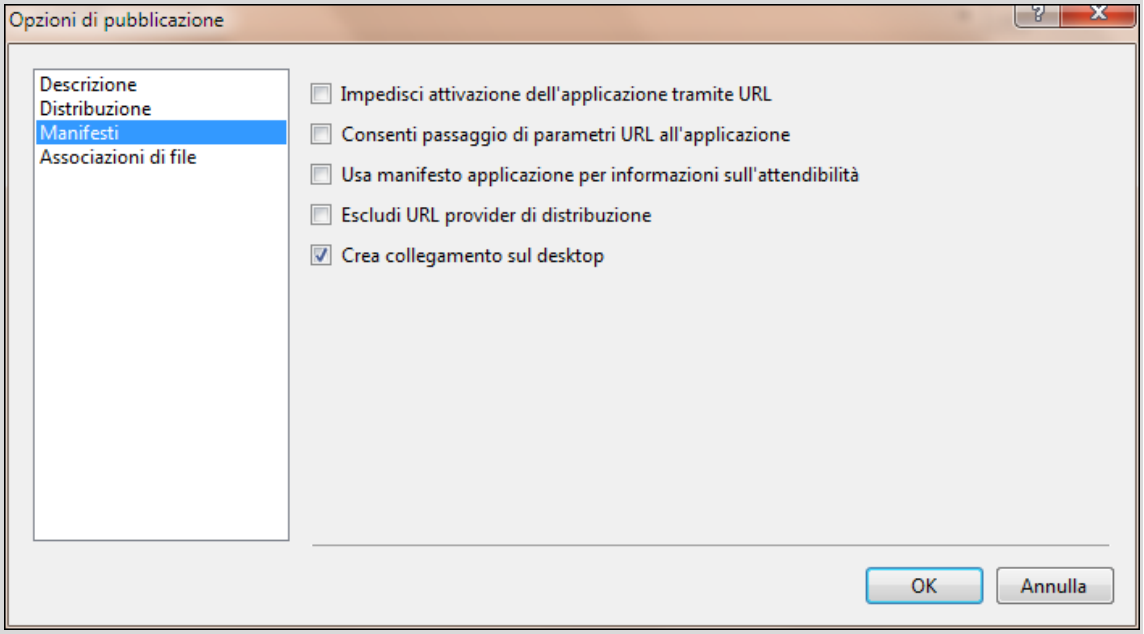
In questa finestra, apriamo la scheda **Pubblica**, ultima scheda disponibile nel menu, in basso a sinistra. Qui è possibile configurare alcune opzioni di **ClickOnce**, prima di avviare il processo di pubblicazione.

La prima parte di questa scheda rispecchia le prime due finestre della Pubblicazione guidata e consente la scelta dei percorsi di pubblicazione e installazione.

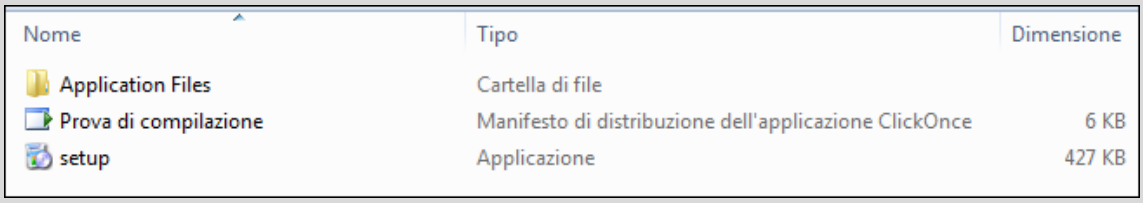
La parte intermedia **(Modalità di installazione e impostazioni)** consente di specificare se l'applicazione sarà installata sul computer ospite o sarà disponibile solo da internet. Cliccando il pulsante **Opzioni...** si accede alla finestra **Opzioni di pubblicazione**, dove è possibile inserire il nome dell'editore o dell'autore del programma e l'indirizzo di una pagina internet per l'assistenza agli utenti:



Qui è anche possibile indicare se si desidera un collegamento al programma sul desktop:



La parte in basso nella scheda **Pubblica** consente infine di impostare la versione dell'applicazione (è consigliabile mantenere l'opzione di numerazione progressiva automatica).
Terminate queste operazioni, con un *clic* sul pulsante **Pubblica** si crea il pacchetto di distribuzione: all'interno della cartella **publish** (o della cartella indicata nelle opzioni) troviamo i file che, ai fini della distribuzione del programma, sono da copiare su un CD-ROM o un DVD-ROM:



Quando l'utente riceve il CD-ROM o il DVD-ROM, con un *clic* sul file **setup** installa il programma sul suo computer, con le impostazioni personali definite dal programmatore.
In particolare, il **collegamento** a Prova di compilazione si troverà nel menu Avvio / Programmi \ Nome del programma.
Il programma avrà una icona di collegamento sul desktop e comparirà tra le applicazioni elencate nel Pannello di Controllo, da dove potrà eventualmente essere disinstallato con la procedura di disintallazione delle applicazioni offerta dal sistema operativo.

Capitolo 40: CREAZIONE DI UN QUIZ MATEMATICO.

In questo ultimo capitolo del manuale metteremo a frutto le conoscenze acquisite sino a ora per la creazione di un programma completo, ripercorrendo tutte le tappe della sua realizzazione, dalla ideazione alla distribuzione.
Il programma che abbiamo scelto è una delle quattro applicazioni che si trovano nella pagina iniziale di VB, sotto il menu **Formazione:**

- Creazione di un visualizzatore di immagini;
- Creazione di un labirinto;
- Creazione di un quiz matematico;
- Creazione di un gioco di abbinamenti.

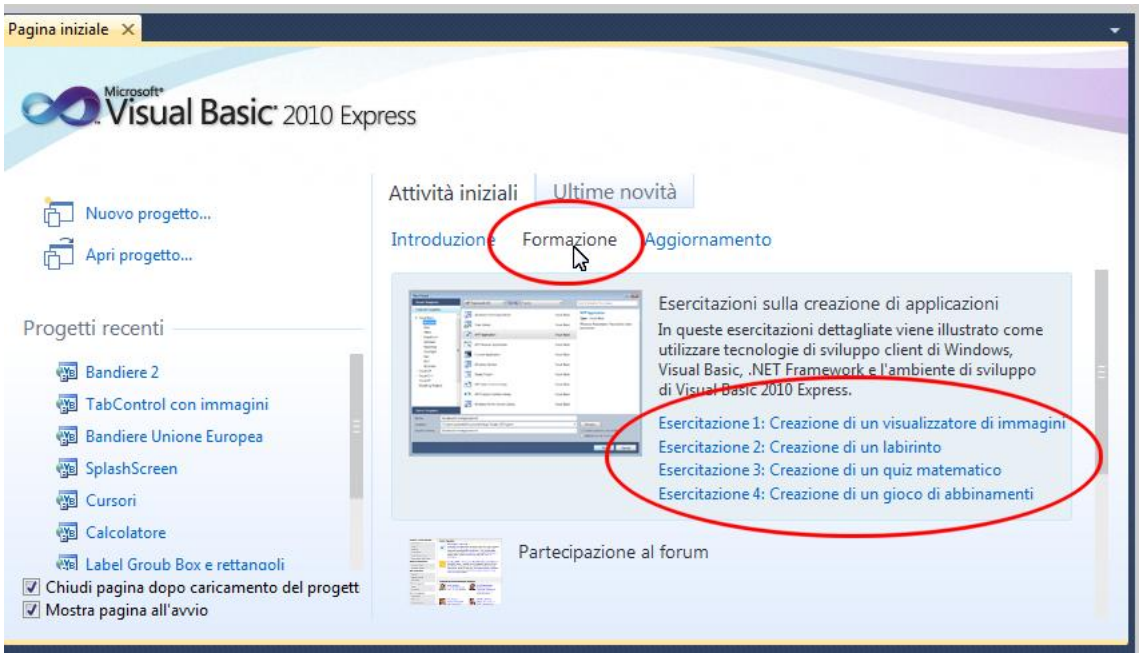


Figura 376: Il menu "Formazione" della pagina iniziale di VB.

La creazione guidata di queste quattro applicazioni è raccomandabile, per consolidare e ampliare le proprie conoscenze.
Il programma **Quiz matematico** è qui proposto con un percorso più chiaro e calibrato sulle conoscenze delle lettrici e dei lettori, con alcune integrazioni che abbiamo ritenuto utili per rendere più efficace il programma o per mostrare nuovi esempi di programmazione.

Il capitolo non contiene esercizi, ma il suo contenuto è in pratica un unico esercizio, al termine del quale le lettrici e i lettori avranno sperimentato in successione tutte le operazioni necessarie per creare, compilare e distribuire un programma.

L'applicazione **Quiz matematico**, completa, con i sorgenti, è disponibile nella cartella **Documenti \ A scuola con VB \ Applicazioni.**

Il progetto consiste in questo: vogliamo creare un programma di matematica con esercizi di calcolo mentale:

- addizioni e sottrazioni entro limiti accettabili per bambini di 9-10 anni, e
- esercitazioni sulle tabelline, presentate come operazioni di moltiplicazione e divisione.

Il programma dovrà essere uno strumento semplice e piacevole, che gli alunni potranno usare senza difficoltà per esercitare le loro capacità di calcolo mentale e ripassare le sequenze dei multipli.

214: L'interfaccia.

Apriamo un nuovo progetto. Gli diamo il nome **Quiz matematico**.
Impostiamo le proprietà del Form1:

- **Text** = Quiz matematico
- **Size** = 494; 394

Per impedire all'utente di modificare le dimensioni del form, impostiamo queste proprietà:

- **FormBorderStyle** = Fixed3D
- **MaximizeBox** = False

Inseriamo nel Form1 un controllo Label. Questa label verrà visualizzata come una casella nell'angolo superiore destro del form, essa è destinata a visualizzare il conto alla rovescia del numero di secondi disponibili per il quiz.
Le diamo queste proprietà:

- (**Name**) = lblTempo
- **AutoSize** = False
- **BorderStyle** = FixedSingle
- **Size** = 200;30
- **Location** = 272; 14
- **Text** = (nulla, cancellare la scritta Label1)
- **Font** = Microsoft Sans Serif; 16 punti

Inseriamo nel Form1 un altro controllo Label, con queste proprietà:

- **Text** = Tempo rimanente
- **Font** = Microsoft Sans Serif; 16 punti
- **Location** = 87; 14

Inseriamo ora nel form i controlli che visualizzeranno i calcoli con addizioni.
Inseriamo una nuova label con queste proprietà:

- (**Name**) = lblAddizioneSx
- **AutoSize** = False
- **Font** = Microsoft Sans Serif; 18 punti
- **Location** = 75; 75
- **Size** = 60, 50
- **Text** = **?**
- **TextAlign** = MiddleCenter

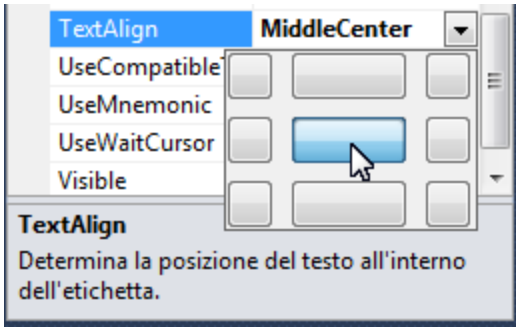


Figura 377: Impostazione della proprietà TextAlign = MidleCenter.

Selezioniamo con un *clic* del mouse la label **lblAddizioneSx**; la copiamo premendo i tasti CTRL+C oppure cliccando **Copia** dal menu **Modifica**.
Effettuiamo quindi le operazioni seguenti:

- incolliamo tre volte la label **lblAddizioneSx** premendo i tasti CTRL+V oppure scegliendo **Incolla** dal menu **Modifica**;
- disponiamo le tre nuove etichette in modo che si trovino in riga con la prima etichettale, utilizzando le linee colorate offerte da VB per allinearle;
- impostiamo la proprietà **Text** della seconda etichetta su **+** (segno di addizione);
- impostiamo la proprietà **(Name)** della terza etichetta su **lblAddizioneDx**;
- impostiamo la proprietà **Text** della quarta etichetta su **=** (segno di uguale).

Inseriamo ora nel Form1, sulla stessa riga delle label dell'addizione, un controllo **NumericUpDown** con queste proprietà:

- **(Name)** = Somma
- **Font** = Microsoft Sans Serif; 18 punti
- **Location**= 372; 93
- **Size** = 100;35

La proprietà **Maximum** (numero massimo inseribile) è impostata automaticamente uguale a 100; non la modifichiamo perché si tratta di un limite idoneo allo scopo del programma.

Il Form1, allo stato attuale della progettazione, mostra la prima riga di controlli, dedicati alla addizione, completa:

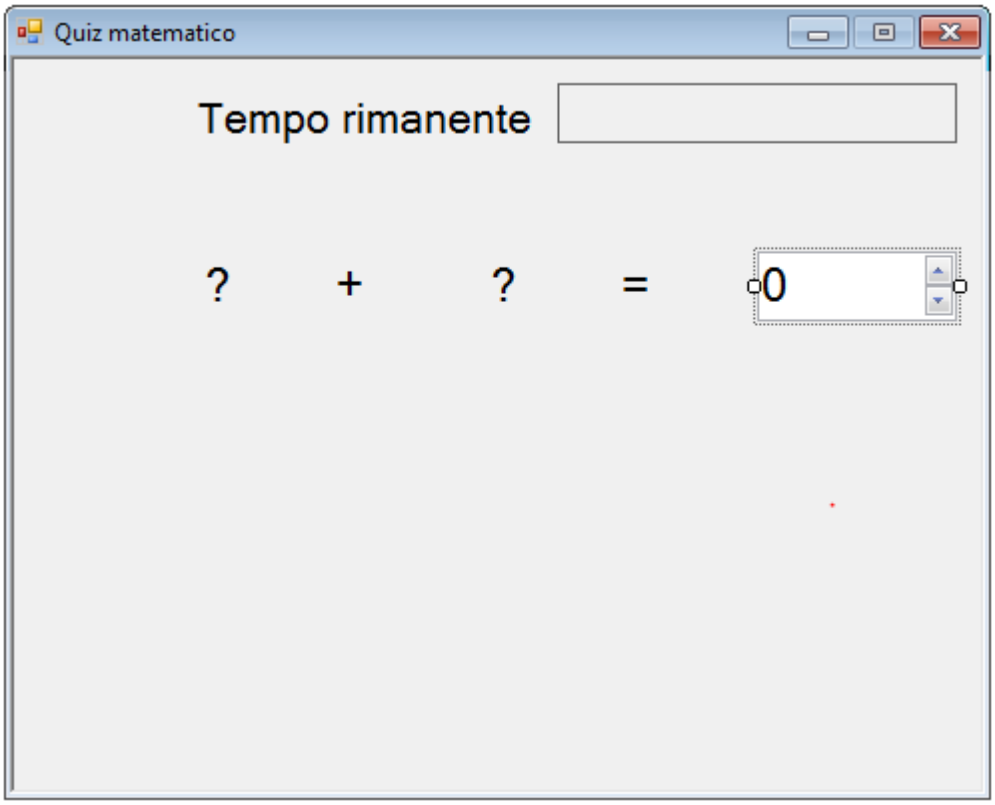


Figura 378: La riga dei controlli per l'addizione.

Ora, trascinando l'oggetto puntatore con il mouse, tracciamo un rettangolo attorno ai cinque controlli della riga della addizione, per selezionarli.
Rilasciando il mouse troviamo i cinque controlli selezionati ed evidenziati in questo modo:

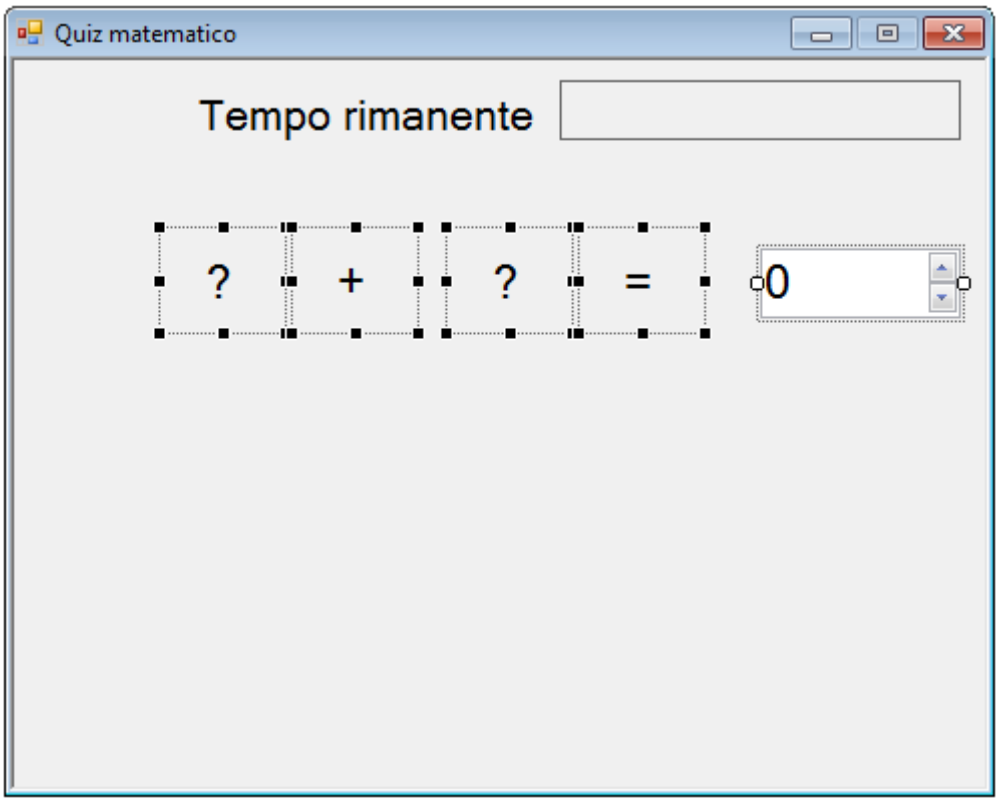


Figura 379: Selezione dei cinque controlli per l'addizione.

Li copiamo e li incolliamo, per creare la riga dei controlli che saranno dedicati alla sottrazione:

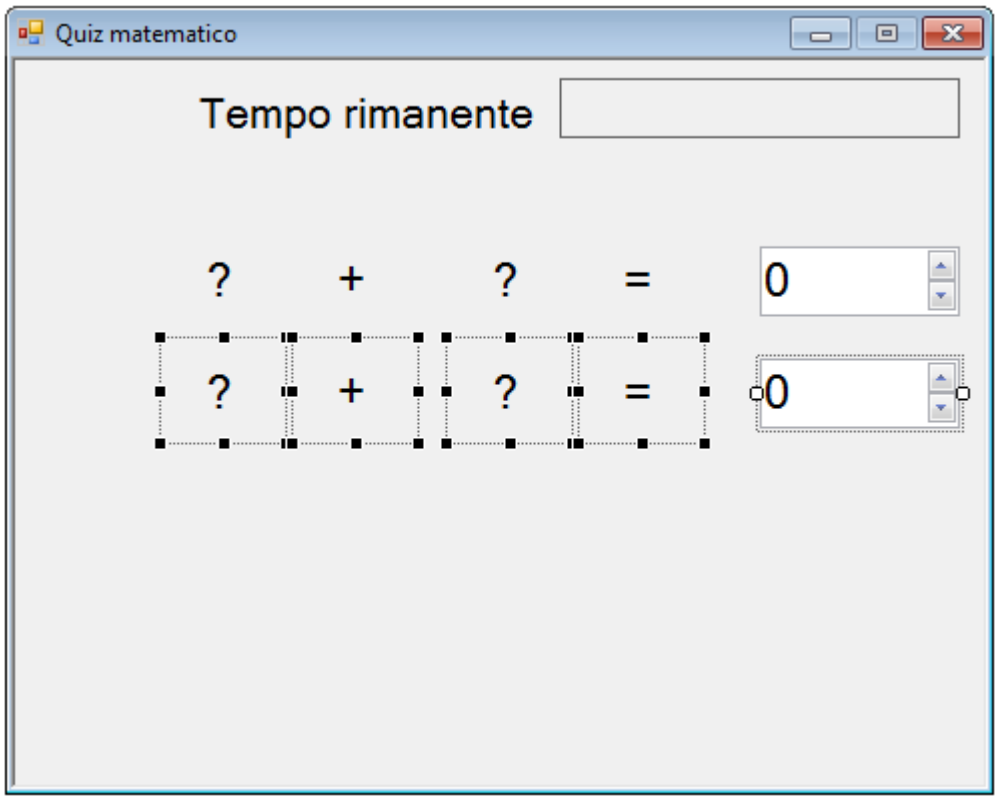


Figura 380: Inserimento dei controlli per la sottrazione.

Sistemiamo opportunamente la nuova riga di controlli sotto la riga dell'addizione, poi ripetiamo l'operazione incolla ancora due volte, per creare le righe dei controlli dedicati alla moltiplicazione e alla divisione, che sistemeremo nel Form1 in questo modo, modificando le proprietà Text delle label che riportano i simboli delle operazioni:

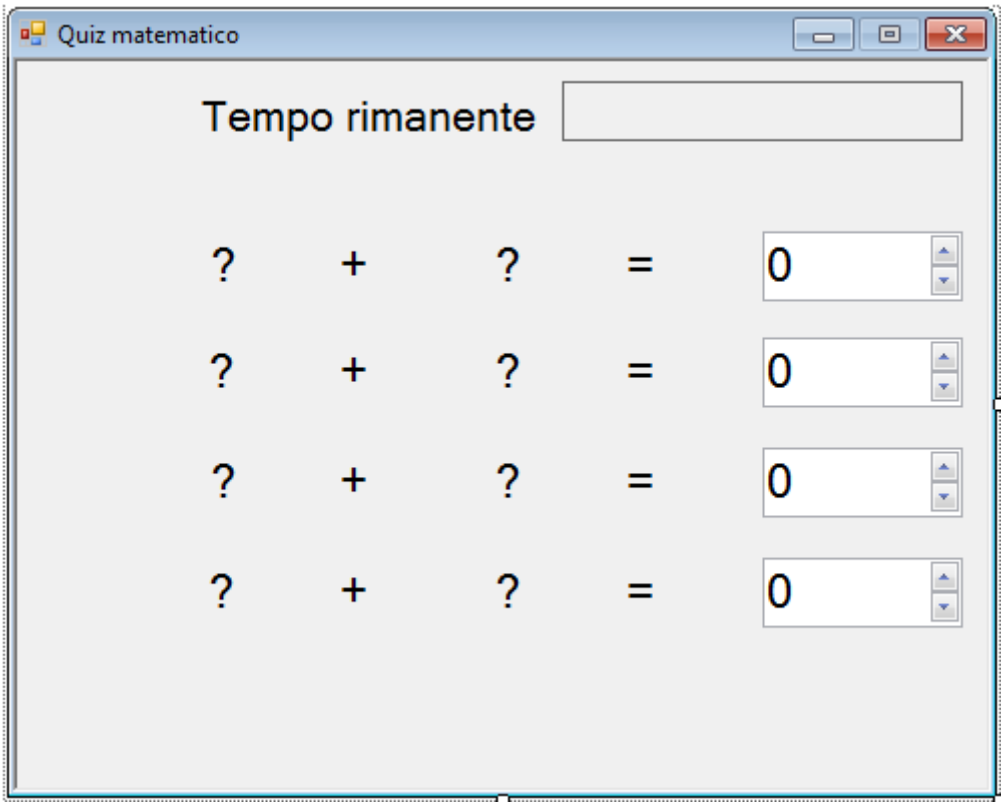


Figura 381: I controlli per le quattro operazioni.

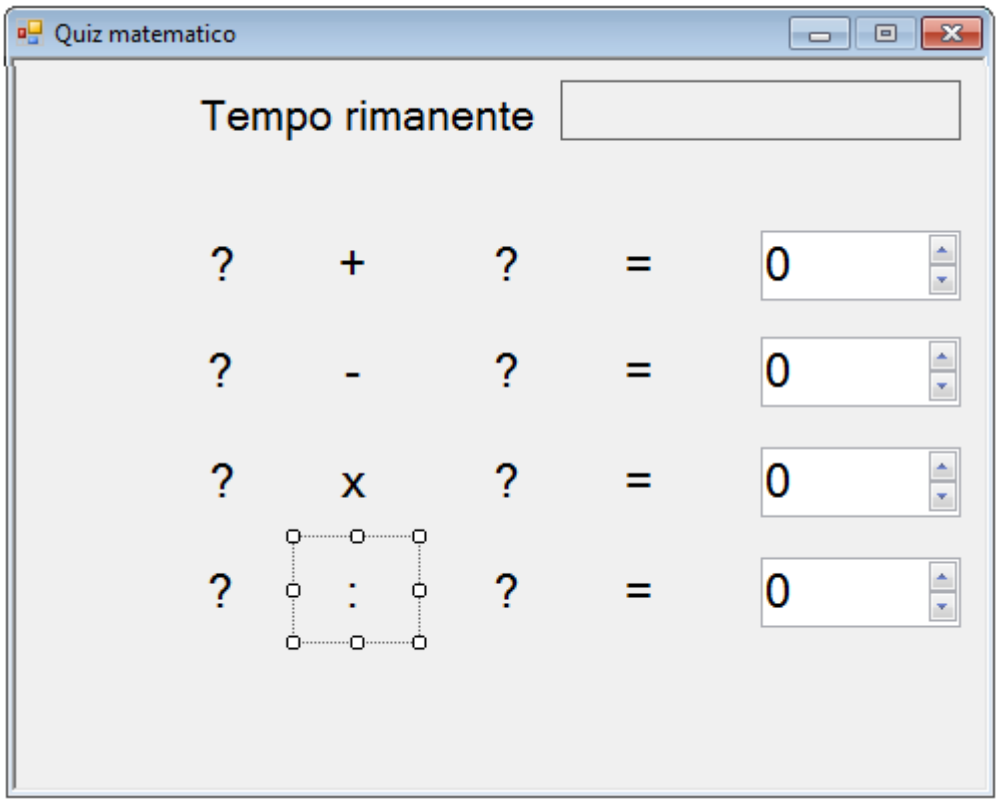


Figura 382: Impostazione dei simboli delle operazioni.

E modifichiamo le proprietà **(Name)** dei controlli aggiunti come mostrato in questa immagine:

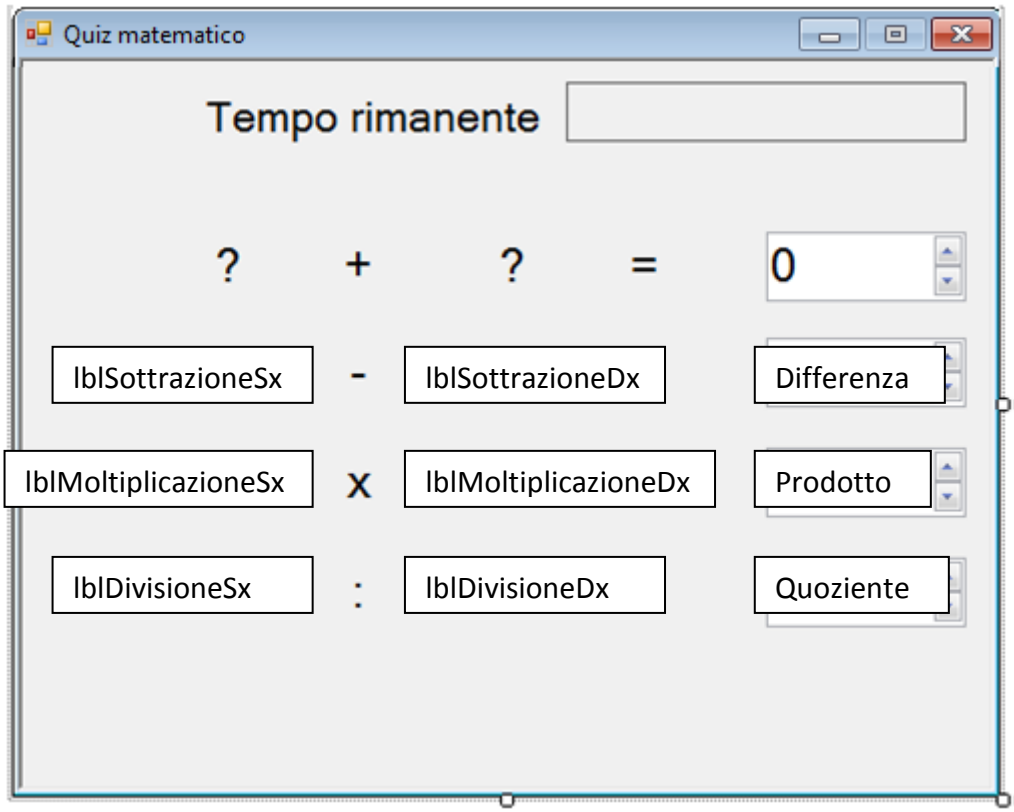


Figura 383: Assegnazione dei nomi ai controlli.

Inseriamo nel Form1 un controllo Button1, necessario per avviare il quiz, con queste proprietà:

- **AutoSize = True**, in modo che il pulsante venga ridimensionato automaticamente per adattarsi alle modifiche del testo in esso contenuto**;**
- **Font** = Microsoft Sans Serif; 14 punti
- **Text** = Avvia il quiz
- **Location** = 354; 316

Inseriamo ora nel Form1, in basso a sinistra, un controllo **ComboBox**, mediante il quale aggiungeremo al programma la possibilità di stabilire e modificare un tempo massimo per risolvere i calcoli; con questa funzione, il programma diventerà più avvincente e più flessibile, perché potrà essere adattato alle capacità degli utenti.
Proprietà del **ComboBox**:

- **Font**: Microsoft Sans Serif; 18 punti
- **Text** = Tempo massimo
- **Location** = 14; 316

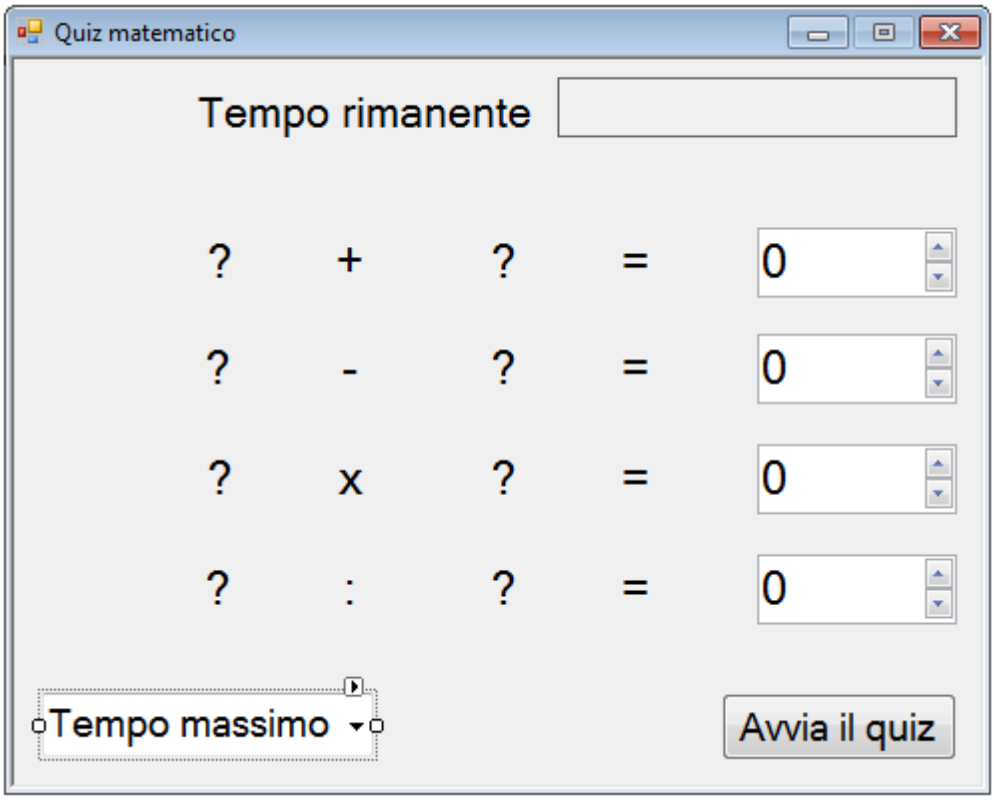


Figura 384: Inserimento di un ComboBox.

Facciamo un *clic* sul pulsante con i tre punti che si trova nella riga della proprietà **Items** del ComboBox e accediamo **all'Editor dell'insieme String**, dove scriviamo gli Items 60, 90, 120, 150, 180, andando ogni volta a capo:

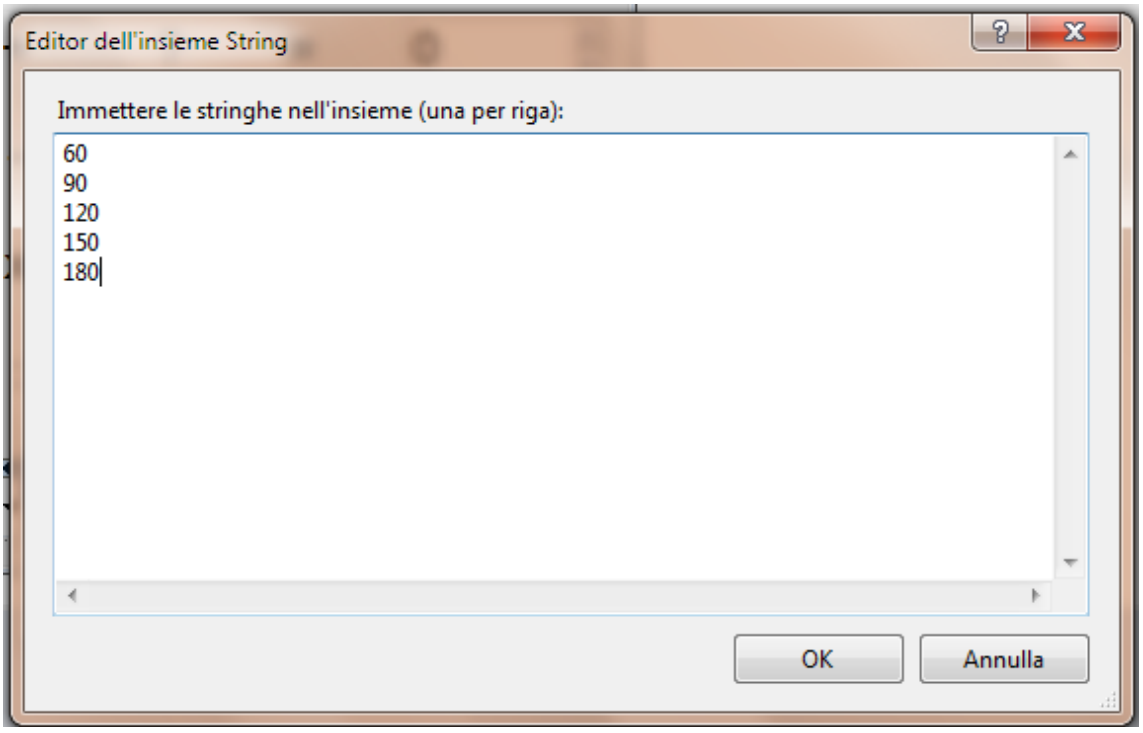


Figura 385: Inserimento di items nel ComboBox.

Gli items inseriti rappresentano quantità di tempo (quantità di secondi) che durante l'esecuzione del programma potranno essere selezionate per adeguare il programma alle capacità di ciascun alunno.
Manca ancora un componente: il Timer che controllerà il trascorrere del tempo, facendo il conto a ritroso dei secondi disponibili.
Inseriamo nel progetto il **Timer1**, con queste proprietà:

- **Enabled = False** (il timer è disattivato all'avvio del programma)
- **Interval = 1000** (il timer emette un tic al trascorrere di 1000 millisecondi, vale a dire ogni secondo).

Il Timer1 non servirà solo a controllare il tempo rimasto: al battere di ogni mezzo secondo, esso attiverà la procedura di controllo dei risultati, per verificare se i quattro numeri presenti nei quattro controlli NumericUpDown sono le risposte esatte.

I numeri nei quattro controlli NumericUpDown posso essere modificati premendo i due tasti con le freccine che si trovano in ognuno di essi, sino al raggiungimento del limite massimo, (proprietà **Maximum = 100**). In alternativa, i numeri possono essere scritti direttamente con la tastiera, come si scrive in una casella di testo. In questo caso, se il numero scritto supera il limite massimo consentito, il controllo NumericUpDown interessato provvede a riportarlo a 100.

Con questo, abbiamo terminato la progettazione dell'interfaccia e possiamo passare alla scritture del codice.
Notiamo che la sistemazione degli oggetti è funzionale a consentire agli alunni di risolvere con rapidità i calcoli; in particolare gli elementi sui quali agirà l'alunno per

l'avvio del quiz e la scrittura dei risultati sono vicini, allineati a destra e incolonnati per essere facilmente raggiungibili con il mouse.

215: La scrittura del codice.

Definiamo innanzitutto alcune variabili che avranno validità per tutte le procedure.

```
Public Class Form1

    ' Crea l'oggetto NumeroCasuale per generare e memorizzare numeri casuali.
    Dim NumeroCasuale As New Random

    ' Crea le variabili di numeri interi per memorizzare gli addendi
dell'addizione.
    Dim Addendo1 As Integer
    Dim Addendo2 As Integer

    ' Crea le variabili di numeri interi per memorizzare i termini della
sottrazione.
    Dim Minuendo As Integer
    Dim Sottraendo As Integer

    ' Crea le variabili di numeri interi per memorizzare i fattori della
moltiplicazione.
    Dim Moltiplicando As Integer
    Dim Moltiplicatore As Integer

    ' Crea le variabili di numeri interi per memorizzare i termini della
divisione.
    Dim Dividendo As Integer
    Dim Divisore As Integer

    ' Crea una variabile per memorizzare il tempo rimanente in secondi.
    Dim TempoRimasto As Integer

End Class
```

Il quiz inizia con un *clic* sul pulsante Button1.
La procedura che gestisce questo evento compie tutte queste operazioni:

- Assegna alla variabile TempoRimasto il numero di secondi che compare all'interno del controllo ComboBox1. Se il ComboBox non è stato cliccato e vi compare ancora la scritta "Tempo massimo", allora alla variabile TempoRimasto sono assegnati 60 secondi.
- Sorteggia gli 8 numeri che verranno utilizzati nelle quattro operazioni.
- Attiva il Timer1 che conta i secondi e controlla le risposte.

L'elaborazione casuale dei numeri per le quattro operazioni avviene utilizzando di volta in volta la variabile NumeroCasuale.
L'operazione materiale del sorteggio degli otto numeri utilizzati nei calcoli avviene con questi otto comandi:

```
        Addendo1 = NumeroCasuale.Next(31)
        Addendo2 = NumeroCasuale.Next(31)
        Minuendo = NumeroCasuale.Next(10, 31)
        Sottraendo = NumeroCasuale.Next(Minuendo)
        Moltiplicando = NumeroCasuale.Next(1, 11)
        Moltiplicatore = NumeroCasuale.Next(1, 11)
        Divisore = NumeroCasuale.Next(2, 11)
        Dividendo = Divisore * NumeroCasuale.Next(2, 11)
```

Notiamo la sintassi del comando che genera un numero casuale:

```
NumeroCasuale.Next()
```

Tra le parentesi vengono indicati il numero minimo e il numero massimo entro i quali deve avvenire il sorteggio. Se tra le parentesi è presente un solo numero, questo è il numero massimo; il numero minimo in questo caso è sottinteso e si intende uguale a 0.
Ecco il codice completo di questa procedura relativa al *clic* sul Button1:

```
    Private Sub Button1_Click() Handles Button1.Click

        My.Computer.Audio.PlaySystemSound(System.Media.SystemSounds.Asterisk)

        ' assegna alla variabile Temporimasto il numero ricavato dal ComboBox1,
se il ComboBox contiene un numero.
        If ComboBox1.Text <> "Tempo massimo" Then
            TempoRimasto = CInt(ComboBox1.Text)
        Else
            TempoRimasto = 90
        End If

        ' disattiva il Button1 e il ComboBox, sino alla fine dell'esercizio:
        Button1.Enabled = False
        ComboBox1.Enabled = False

        ' Sorteggia i due addendi dell'addizione e li inserisce nelle relative
label:
        Addendo1 = NumeroCasuale.Next(21)
        Addendo2 = NumeroCasuale.Next(21)
        lblAddizionesx.Text = Addendo1
        lblAddizioneDx.Text = Addendo2
        'porta a 0 il valore del controllo NumericUpDown che si trova nella riga
della addizione:
        Somma.Value = 0

        ' Sorteggia i due termini della sottrazione e li inserisce nelle relative
label:
        Minuendo = NumeroCasuale.Next(10, 31)
        Sottraendo = NumeroCasuale.Next(Minuendo)
        lblSottrazioneSx.Text = Minuendo
        lblSottrazioneDx.Text = Sottraendo
        'porta a 0 il valore del controllo NumericUpDown che si trova nella riga
della sottrazione:
        Differenza.Value = 0
```

```
        ' Sorteggia i due fattori della moltiplicazione e li inserisce nelle
relative label:
        Moltiplicando = NumeroCasuale.Next(1, 11)
        Moltiplicatore = NumeroCasuale.Next(1, 11)
        lblMoltiplicazioneSx.Text = Moltiplicando
        lblMoltiplicazioneDx.Text = Moltiplicatore
        'porta a 0 il valore del controllo NumericUpDown che si trova nella riga
della moltiplicazione:
        Prodotto.Value = 0

        ' Sorteggia i due termini della divisione e li inserisce nelle relative
label:
        Divisore = NumeroCasuale.Next(2, 11)
        Dividendo = Divisore * NumeroCasuale.Next(2, 11)
        lblDivisioneSx.Text = Dividendo
        lblDivisioneDx.Text = Divisore
        'porta a 0 il valore del controllo NumericUpDown che si trova nella riga
della divisione:
        Quoziente.Value = 0

        ' Avvia il timer contasecondi
        Timer1.Start()

    End Sub
```

Nell'analisi della procedura, ricordiamo che VB usa l'asterisco * come simbolo della moltiplicazione.
Una volta attivato con il comando `Timer1.Start`, il Timer1 crea a ogni secondo un evento che possiamo pensare simile al *tic* di un orologio.
Il controllo delle risposte avviene allo scadere di ogni secondo.
Quando compaiono i quattro risultati corretti, il programma blocca il timer e visualizza il messaggio di congratulazioni.
In caso contrario, il tempo continua a scorrere nell'attesa di un esito positivo, fino allo scadere dell'ultimo dei secondi prestabiliti.
La procedura seguente gestisce l'evento tic del Timer1 ed esegue queste operazioni:

- controlla se i risultati delle quattro operazioni sono corretti;
- se i risultati sono corretti emette un segnale sonoro positivo e visualizza un messaggio di congratulazioni;
- se non sono corretti, diminuisce il tempo disponibile di un secondo;
- se il tempo è finito, emette un segnale sonoro negativo, visualizza i quattro risultati corretti e riporta il programma allo stato iniziale, prontro ad avviare in nuovo quiz.

```
   Private Sub Timer1_Tick() Handles Timer1.Tick

        ' Ad ogni tic del Timer avviene il controllo dei risultati contenuti nei
4 NumericUpDowwn:

        ' Se i quattro calcoli sono corretti, ferma il Timer e visualizza il
messaggio di congratulazioni.
        If (Addendo1 + Addendo2 = Somma.Value) And
            (Minuendo - Sottraendo = Differenza.Value) And
            (Moltiplicando * Moltiplicatore = Prodotto.Value) And
            (Dividendo / Divisore = Quoziente.Value) = True Then
```

```
            Timer1.Stop()

My.Computer.Audio.PlaySystemSound(System.Media.SystemSounds.Exclamation)
            MessageBox.Show("Tutti i calcoli sono esatti!", "Congratulazioni!")

            ' Riattiva il Button1 e il ComboBox, per il prossimo esercizio:
            Button1.Enabled = True
            ComboBox1.Enabled = True

        ElseIf TempoRimasto > 0 Then
            ' Altrimenti, se i risultati non sono esatti e il tempo NON è
scaduto,
            ' diminuisce di un secondo il tempo a disposizione e mostra il tempo
rimanente nella lblTempo.
            TempoRimasto = TempoRimasto - 1
            If TempoRimasto < 10 Then lblTempo.ForeColor = Color.Red
            lblTempo.Text = TempoRimasto & " secondi"

        Else
            ' ALTRIMENTI, se i risultati NON sono esatti e il tempo è scaduto
            ' mostra un messaggio di disappunto e mostra i risultati corretti.
            Timer1.Stop()
            lblTempo.Text = "Tempo scaduto!"
            MessageBox.Show("Il tempo a disposizione è terminato.",
"Accipicchia!")
            Somma.Value = Addendo1 + Addendo2
            Differenza.Value = Minuendo - Sottraendo
            Prodotto.Value = Moltiplicando * Moltiplicatore
            Quoziente.Value = Dividendo / Divisore

            ' Riattiva il Button1 e il ComboBox, per il prossimo esercizio:
            Button1.Enabled = True
            ComboBox1.Enabled = True

        End If

    End Sub
```

L'immagine che segue mostra il programma in esecuzione.

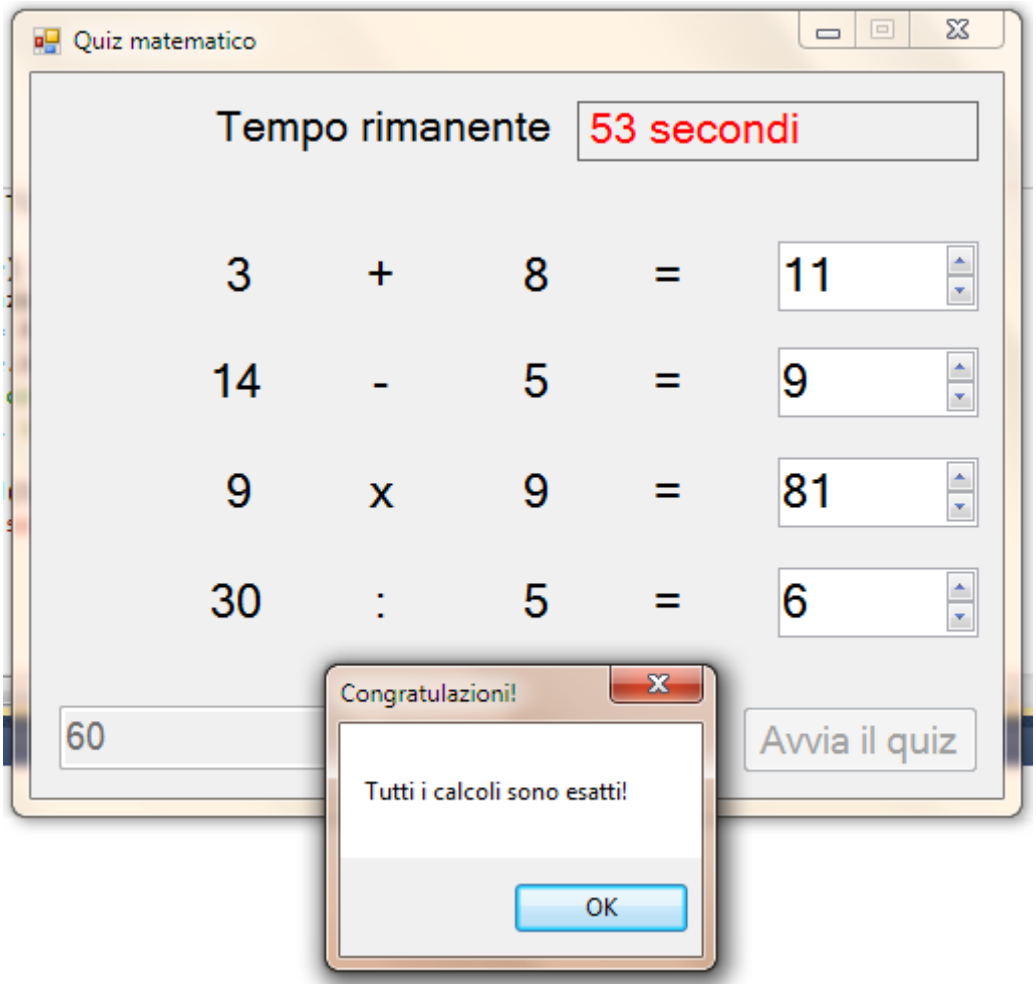


Figura 386: Il Quiz matematico in esecuzione.

Terminato il programma, non dimentichiamo di salvarlo con un clic sul pulsante **Salva tutto**. Se non abbiamo modificato le impostazioni di base di VB, il programma è salvato nella cartella **Documenti \ A scuola con VB 2010 \ Projects \ Quiz matematico**.

216: Compilazione del programma.

Passiamo alla fase della **compilazione** del programma.
Nella striscia dei menu di VB, facciamo un *clic* su **Compila** e poi su **Compila Prova di compilazione**.
Il prodotto finale di questa operazione è un file di Windows, autonomo rispetto a VB, eseguibile con un *clic* del mouse anche in computer nei quali l'ambiente di progettazione di VB non è presente.
Terminata la compilazione, scorriamo la cartella in cui è stato salvato il progetto: nella sottocartella **bin \ Debug** troviamo il file eseguibile **Quiz matematico.exe**.

Questo è il file che dovremo distribuire con il pacchetto di installazione che vedremo nel prossimo paragrafo.

217: Distribuzione del programma.

Nella finestra Esplora soluzioni, facciamo un *clic* con il tasto destro del mouse sul progetto Quiz matematico e, nel menu che si apre, facciamo un *clic* sull'ultima voce: **Proprietà**.
Accediamo così alla finestra delle proprietà del progetto. In questa finestra, apriamo la scheda **Pubblica**: l'ultima scheda disponibile nel menu, in basso a sinistra.
Qui configuriamo alcune proprietà del progetto, prima di avviare il processo di pubblicazione:

- In alto, lasciamo senza modifiche i percorsi di pubblicazione e installazione.
- Nella parte con le **Modalità di installazione e impostazioni** facciamo un *clic* su **Applicazione disponibile anche offline dal menu Start**, poi un clic sul pulsante pulsante **Opzioni...**
- Nella finestra **Opzioni di pubblicazione**, nella scheda **Descrizione** inseriamo i dati sull'autore del programma e nella scheda **Manifesti** facciamo un clic sull'opzione **Crea collegamento sul desktop**:

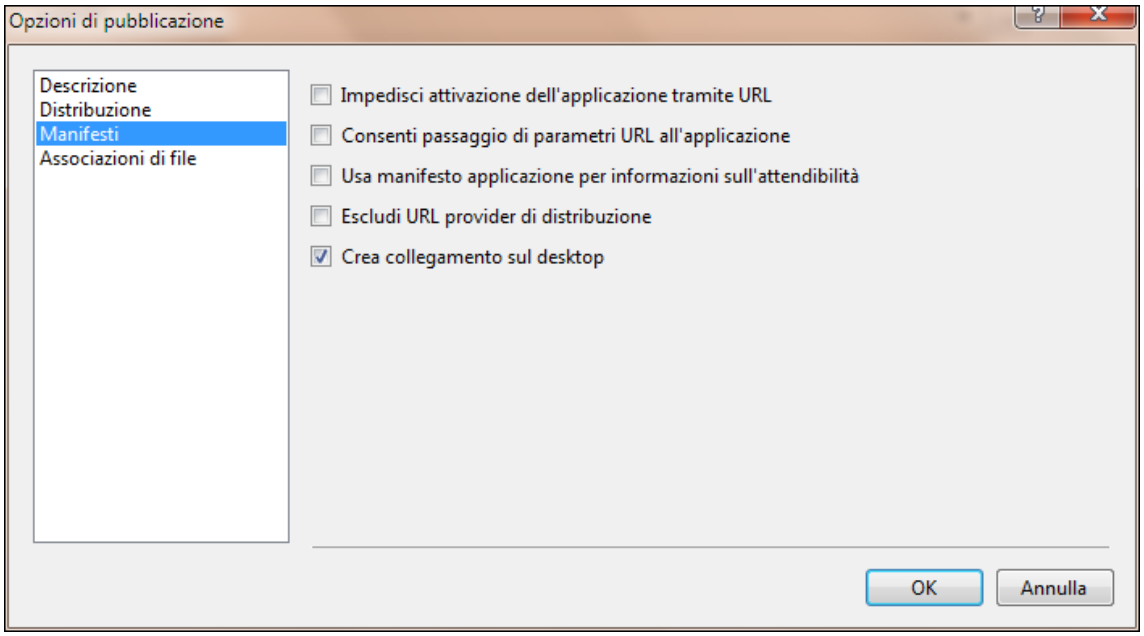


Figura 387: Creazione di un collegamento sul desktop.

Terminate queste operazione, facciamo un *clic* sul pulsante **Pubblica.**

Ora all'interno della sottocartella **publish** del progetto troviamo i file da copiare su un CD-ROM o su una chiavetta USB, o in un sito internet, per consentire la distribuzione del **Quiz matematico** agli utenti:

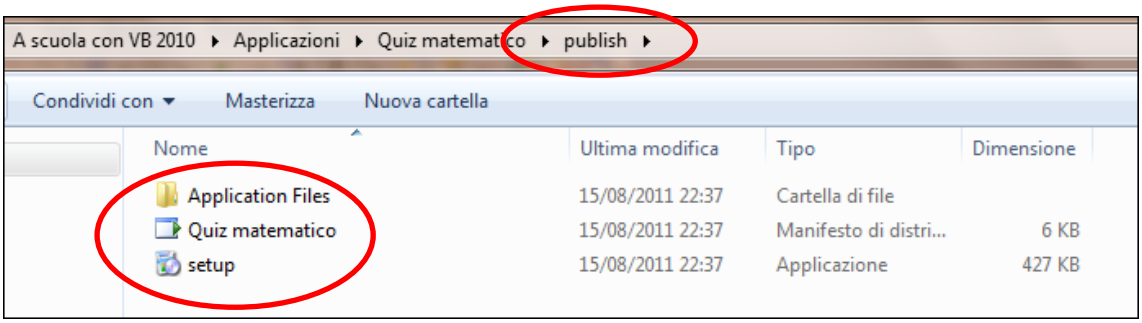


Figura 388: Il pacchetto di file per l'installazione del Quiz matematico.

Un *clic* sul file **setup** installa il programma sul computer ospite, con un collegamento al programma dal menu Avvio / Programmi e dal desktop, dall'icona Quiz matematico.
Il programma comparirà tra le applicazioni elencate nel Pannello di Controllo, da dove sarà possibile disinstallarlo con la procedura comune alle altre applicazioni.