I.T.E.S. PITAGORA

Via Pupino 10/A - Taranto

MANUALE

VISUAL BASIC 6.0

Introduzione a Visual Basic 6

Visual Basic è un linguaggio di programmazione **WUI** (Windows User Interface) di casa Microsoft, abbastanza potente e molto semplice nella sintassi e nelle funzionalità, adatto quindi sia al neofita che allo sviluppatore professionista.

Visual Basic, da ora in poi semplicemente **VB**, nasce dal vecchio **Basic**, sempre di casa Microsoft, come versione avanzata con interfaccia visuale (da qui il suo nome attuale). Il vecchio Basic, infatti, nasce e muore in ambiente DOS. VB è attualmente e definitivamente arrivato alla versione 6.0: la versione successiva, in uso da già qualche anno, fa parte della piattaforma .NET e pur conservando la maggior parte delle caratteristiche, può definirsi un linguaggio a se stante: VB è quindi differente da VB.NET.

VB è un tipo di programmazione detta **event driven**, ovvero basata sugli eventi.

2. Interfaccia e ambiente di sviluppo

Una volta eseguita ed andata a buon fine l'installazione, per accedere all'ambiente di sviluppo di VB partire dal menu

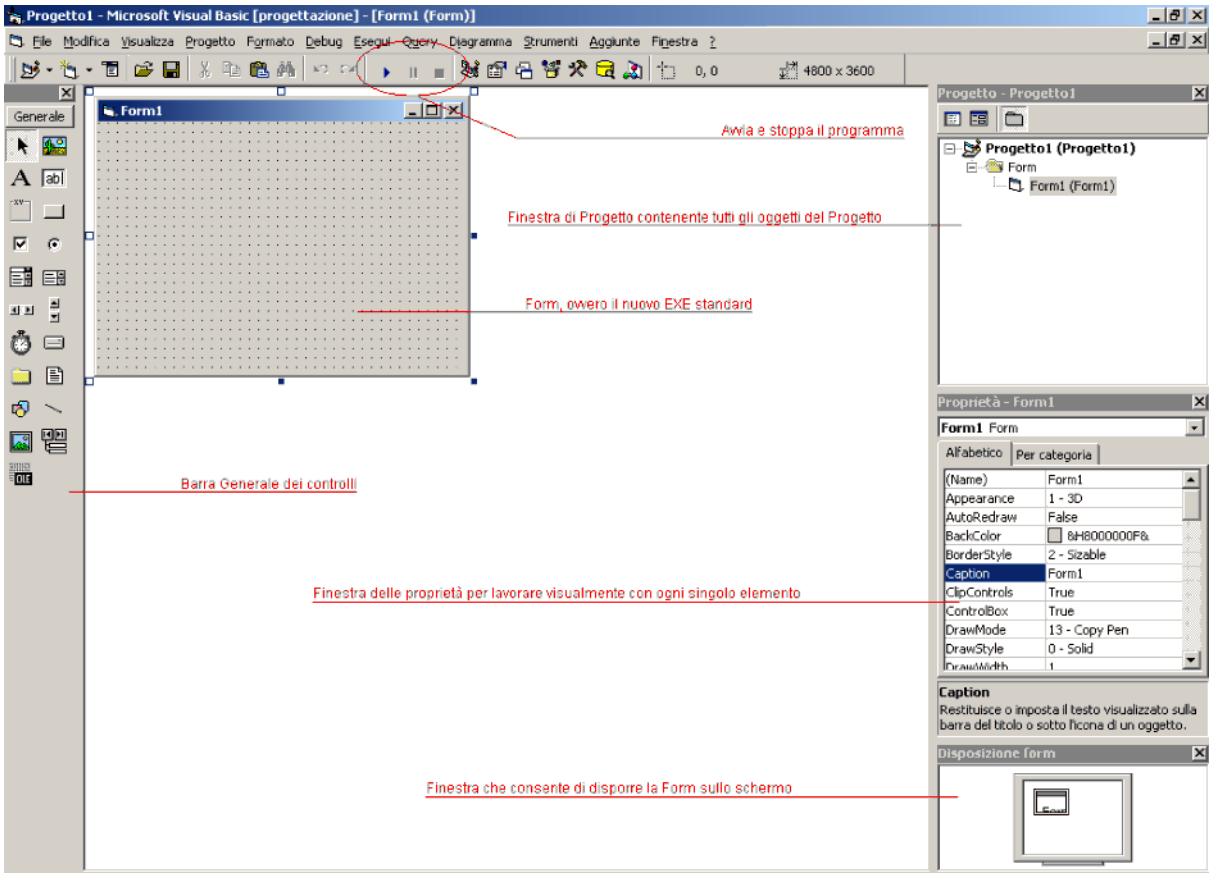
Start / Programmi / Microsoft Visual Studio 6.0 / Microsoft Visual Basic 6.0

la prima richiesta fatta da VB è di scegliere il tipo di progetto tramite la finestra documentata nell'immagine seguente:

La voce selezionata per default è **EXE standard** ovvero un classico progetto VB. Non preoccupatevi delle altre voci, si tratta in un modo o nell'altro di componenti avanzati.

Confermate quindi, cliccando il tasto **Apri** la scelta di un nuovo EXE standard.

L'interfaccia completa del programma, documentata con una serie di commenti aggiunti in rosso, è visualizzabile di seguito.



3. I Form e gli oggetti di un modulo VB

I **Form** sono gli elementi madre di un programma VB, o in generale di qualsiasi software di sviluppo e linguaggio di programmazione WUI. Si tratta di finestre molto comuni, classiche di Windows: un esempio è la finestra delle proprietà di Internet Explorer, oppure delle proprietà del Desktop, ecc...

Su di un Form vengono posizionati i vari elementi per la manipolazione dei dati o per la realizzazione di applicazioni non basate su di un database. In questo paragrafo vedremo i principali oggetti (che chiameremo prevalentemente *controlli*) per la realizzazione di un modulo VB.
Torniamo un attimo all'interfaccia di VB e guardiamo sulla sinistra dell'immagine la barra **Generale** in cui sono contenuti una serie di icone; da qui possiamo, con un doppio click o facendo il *drag' n' drop* (trascinamento) dell'icona sul Form stessa, inserire controlli da manipolare via codice. Facciamo un semplice esempio che vede coinvolti i tre controlli più utilizzati in VB, ovvero la **TextBox** (casella di testo per l'inserimento di dati), la **Label** (semplice area in cui inserire descrizioni) ed il **CommandButton** (bottone di comando per l'esecuzione di una routine o di uno script VB).

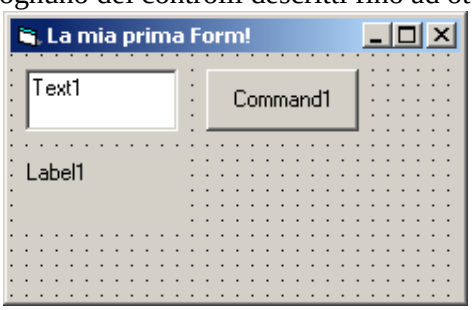
Si crei quindi un nuovo progetto EXE standard; dalla **Finestra delle Proprietà** sulla destra dell'IDE di sviluppo si vada a modificare la proprietà **Name** della Form e la si chiami **frmPrimoProg**. Prima di proseguire tengo a sottolineare la modalità di assegnazione dei nomi ai controlli dei Form ed ai Form stessi: si utilizza un suffisso in minuscolo e poi il nome da

assegnare con la maiuscola iniziale; prego inoltre di assegnare alle Form ed ai progetti nomi significativi, per due buoni motivi:

1. in fase di progettazione è difficile cambiare un nome ad un controllo per via dei riferimenti che VB crea automaticamente in una serie di file di configurazione
2. per non trovarvi in difficoltà quando non sapere un controllo a che scopo lo avete creato se non riuscite ad identificarlo facilmente e subito da un nome che abia un senso logico

Si modifichi inoltre la proprietà Caption (la scritta che compare sul titolo del Form) in "Il mio primo Form!".

Quelli che sono cerchiati in rosso nell'immagine precedente sono rispettivamente i controlli per l'inserimento di una Label, di una TextBox e di un CommandButton. Inserite sul Form uno per ognuno dei controlli descritti fino ad ottenere il seguente effetto in fase di sviluppo:

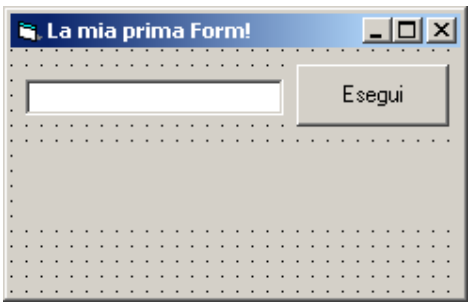


Cliccando una singola volta su un controllo sul Form compaiono attorno al controllo i puntini blu di selezione ed è possibile modificarne le dimensioni. Inoltre si assegnino, dalla Finestra delle Proprietà, i seguenti nomi e le seguenti diciture testuali ai tre controlli:

- TextBox
  - Name: txtTesto
  - Text: *nessun testo*

- Label
  - Name: lblRisultato
  - Caption: *nessun testo*
- CommandButton
  - Name: cmdEsegui
  - Caption: Esegui

fino ad ottenere il seguente risultato:



Eccoci finalmente di fronte alla spiegazione di programmazione **event driven**, ovvero basata (letteralmente guidata) sugli eventi. Al **click** sul CommandButton verrà eseguito uno script VB: il click è l'evento!

Si faccia doppio click sul CommandButton e si accederà al codice del bottone di comando, già composto dal seguente codice:

```
Private Sub cmdEsegui_Click()

End Sub
```

In VB questa è una **Sub** che tratteremo più avanti. All'interno di questa Sub, che comunque è l'elemento madre di un programma o di una singola fase di un programma VB, andremo ad inserire il codice che verrà eseguito al click sul CommadButton:

```
' Questo è un commento!
' VB non esegue tutto ciò che è preceduto
' dal singolo apice ma lo tratta come un
' promemoria per lo sviluppatore

Private Sub cmdEsegui_Click()
    If txtTesto.Text = "" Then
        lblRisultato.Caption = "Inserisci un testo"
    Else
        lblRisultato.Caption = txtTesto.Text
    End If
End Sub
```

Anche se non abbiamo ancora affrontato le istruzioni condizionali il programma ragionerà cosi: se la casella di testo contiene un valore vuoto allora scrivi nella Label un messaggio che indichi all'utente di inserire un testo, oppure scrivi il contenuto della casella di teso.

Per avviare il programma si faccia click sul pulsante Play cerchiato in rosso nell'interfaccia di VB, oppure si prema il tasto F5 sulla tastiera.

I tipi di dati

Innanzi tutto, vediamo di capire cosa sono le **costanti e le variabili**.

È possibile pensare ad esse come a contenitori in cui si trovano delle informazioni, cioè dei valori; più precisamente, costanti e variabili sono riferimenti a locazioni di memoria in cui sono salvati determinati valori. Non ci interessa sapere qual è l'esatto indirizzo della memoria che contiene questi valori: è Visual Basic che si occupa di andare a recuperare nella memoria il valore associato alla variabile o alla costante che stiamo utilizzando.

La differenza tra costanti e variabili è questa: le costanti, come dice il nome stesso, una volta impostate non sono più modificabili, mentre le variabili (anche in questo caso il nome è illuminante) possono essere modificati ogni volta che si desidera.

Ad esempio, se creiamo una costante di nome **NomeCostante** e impostiamo il suo valore su 10, in seguito non possiamo modificarla, quindi tale costante varrà 10 per tutta l'esecuzione del programma. Se, invece, abbiamo una variabile **NomeVariabile**, possiamo modificare il suo valore in ogni momento, quindi possiamo inizialmente assegnarli il valore 4, poi 9, poi 3, e così via.

Un altro punto nodale in qualsiasi linguaggio di programmazione è la distinzione dei **tipi di dato**. Come è facile intuire, un programma lavora con dati di tipo diverso, cioè stringhe (ovvero sequenze di caratteri) e numeri; questi ultimi, poi, si dividono ulteriormente a seconda che siano numeri interi, decimali, che indichino valute, etc.

Questa distinzione è molto importante, perché ogni tipo di dato ha una dimensione (cioè un'occupazione in memoria) diversa: ad esempio, un numero intero occupa meno memoria di un numero decimale a precisione doppia.

Tali particolari possono sembrare delle sottigliezze, però quando si sviluppano applicazioni di una certa complessità essi vengono ad assumere un'importanza rilevante. Vediamo ora i tipi di dati fondamentali in Visual Basic, ricordando che nella Guida in linea del linguaggio è possibile trovare altre informazioni su questo argomento:

Tipo di dato	Dimensione in memoria	Intervallo
Boolean	2 byte	true (-1) o False (0)
Integer (intero)	2 byte	Da -32.768 a 32.767
Long (intero lungo)	4 byte	Da -2.147.483.648 a 2.147.483.647
Single (virgola mobile a precisione semplice)	4 byte	Da -3,402823E38 a -1,401298E-45 per valori negativi;da 1,401298E-45 a 3,402823E38 per valori positivi
Double (virgola mobile a precisione doppia)	8 byte	Da -1,79769313486232E308 a -4,94065645841247E-324 pervalori negativi; da 4,94065645841247E-324 a 1,79769313486232E308 per valori positivi
String	10 byte + lunghezza stringa (10 byte + numero caratteri)	Da 0 a circa 2 miliardi

Le costanti si dichiarano in questo modo:

```
Const Nome [As Tipo] = valore
```

Const è una parola chiave riservata di VB che si usa per definire una costante. `Nome` è il nome che si sceglie di attribuire alla costante.

Nella scelta dei nomi (sia delle costanti, delle variabili, ma anche delle procedure, delle funzioni e dei controlli, che vedremo più avanti), è necessario seguire alcune regole. I nomi non devono essere più lunghi di 40 caratteri, non posso iniziare con un numero né contenere spazi e caratteri come `?`, `!`, `:`, `;`, `.` e `,`.

Visual Basic, al contrario di altri linguaggi come il C++ o Java, **non fa differenza tra maiuscole e minuscole**.

As è un parametro opzionale che indica il *Tipo* di dato contenuto nella costante; se non viene specificato, il compilatore lo determinerà sulla base del valore assegnato alla costante stessa. *valore* è il valore vero e proprio della costante. Ecco alcuni esempi di dichiarazioni di costanti:

```
Const PI = 3.14 Const Nome As String = "Marco"
```

Una sintassi analoga è quella che permette di dichiarare le variabili:

```
Dim nome [As Tipo]
```

In questo caso si usa la parola chiave **Dim** per indicare al compilatore che quella che si sta per definire è una variabile.

Le convenzioni per il nome sono le stesse che sono state accennate a proposito delle costanti. Anche per le variabili il parametro **As** è opzionale: se non viene specificato, la variabile verrà dichiarata di tipo `Variant`, un particolare tipo che può contenere dati di tutti i tipi. È sconsigliabile definire variabili di tipo `Variant`, se non espressamente necessario, dal momento che questo tipo di dato occupa molta memoria. Ecco alcuni esempi di dichiarazioni di variabili.

```
Dim Utenti As Integer
Dim Nome As String, Cognome As String
```

Per maggiori informazioni sugli argomenti trattati in questa lezione, cercare l'argomento Riepilogo dei tipi di dati nella Guida in linea di Visual Basic.

Le procedure e le funzioni

Dopo aver parlato di tipi di dati, costanti e variabili, dobbiamo illustrare brevemente le procedure e le funzioni. In entrambi i casi si tratti di una sorta di "raggruppamento" di istruzioni che svolgono una sequenza di operazioni. Praticamente tutto il codice di un programma Visual Basic è contenuto all'interno di funzioni e procedure (chiamate genericamente **routine**).

La differenza fondamentale tra procedure e funzioni è che le seconde restituiscono sempre almeno un valore, ad esempio il risultato di un'elaborazione oppure un valore di ritorno che determina se la routine ha avuto successo, mentre le procedure possono anche non restituire alcun valore.

Le procedure

Iniziamo esaminando la **dichiarazione di una procedura**:

```
Sub ([Parametro As <tipo>, ...])
...
End Sub
</t
```

Tutte le dichiarazioni di procedura iniziano con la parole chiave **Sub**. Può essere preceduta da Private o Public per indicare che può essere eseguita solo nel Form in cui è stata definita o in tutto il progetto.

Segue il nome della routine, che deve rispettare le convenzione analizzate precedente a proposito delle costanti. Il nome deve essere seguito da parentesi, al cui interno è possibile inserire i parametri (opzionali) richiesti della procedura; non c'è limite al numero di parametri che si possono definire. Tali parametri possono essere visti come variabili.

La scritta **End Sub** è composta di parole riservate di VB e indica la fine di una procedura.

Vediamo ora un esempio di procedura, anche per illustrare meglio l'utilizzo dei parametri. Supponiamo di dover calcolare l'area di un cerchio: la formula è sempre la stessa, quello che cambia è solo la misura del raggio. Per tale motivo, invece di riscrivere ogni volta la formula, possiamo scrivere una procedura che richieda come parametro proprio la lunghezza del raggio:

Sub AreaCerchio(Raggio As Double)
...
End Sub

Supponiamo ancora di voler scrivere un programma che chiede all'utente la lunghezza del raggio e sulla base di questa calcola l'area del cerchio. Dopo aver definito la procedura come sopra descritto, ci basterà richiamarla passandogli come argomento la lunghezza del raggio; ad esempio:

```
Sub AreaCerchio(Raggio As Double)
...
End Sub
```

Queste sono tre **chiamate alla procedura** con parametri diversi. Nel primo caso, Raggio varrà `5.4`, nel secondo `11` e nel terzo `6.9`. Ecco quindi come potrebbe risultare la procedura `AreaCerchio` completa:

```
Sub AreaCerchio(Raggio As Double)
  Msgbox 3.14*(raggio^2)
End Sub

AreaCerchio 5.4
AreaCerchio 11
AreaCerchio 6.9
```

In questo esempio è stata usata la funzione `MsgBox`, che visualizza un messaggio in una finestra di dialogo e attende che l'utente prema un tasto. A questo punto, utilizzando le tre chiamate sopra definite, otterremo questi risultati:

Le funzioni

Passiamo ora ad analizzare le funzioni, osservando che per esse vale la maggior parte delle considerazioni già fatte per le procedure. La **dichiarazione di una funzione** è questa:

```
Function ([Parametro As <tipo>, ...]) [As <tipo>]
...
End Function
```

Come si vede, in questo caso invece della parola chiave `Sub` si usa `Function`. La cosa nuova, cui si è già accennato, è che **le funzioni restituiscono sempre almeno un valore**. Nella dichiarazione, infatti, possiamo notare che, dopo l'elenco (opzionale) dei parametri c'è un ulteriore argomento opzionale, ancora una volta `As <tipo>`: esso indica il tipo di dato restituito dalla funzione.

Come per le variabili, se non viene specificato tale parametro il valore restituito sarà di tipo `Variant`.

Riprendiamo l'esempio di prima e trasformiamo la procedura `AreaCerchio` in una funzione:

```
Function AreaCerchio(Raggio As Double) As Double
  AreaCerchio = Raggio * Raggio * 3.14
End Function
```

Quando si richiama questa funzione, `AreaCerchio` contiene il valore dell'area del cerchio. Vediamo ora come si utilizzano le funzioni, basandoci come sempre sull'esempio.

```
Dim Area1 As Double, Area2 As Double, Area3 As Double

Area1 = AreaCerchio(5.4) 'Area1 vale 91,5624
Area2 = AreaCerchio(11)  'Area2 vale 379,94
Area3 = AreaCerchio(6.9) 'Area3 vale 149,4954
```

Anzitutto abbiamo dichiarato tre variabili, `Area1`, `Area2` e `Area3`, che conterranno i valori delle aree. Ad esse è stato poi assegnato il valore restituito dalla funzione `AreaCerchio`.

Di fianco ad ogni istruzione è stato posto un commento; per **inserire un commento in VB** è necessario digitare un apice (`'`): tutto quello che verrà scritto sulla stessa riga a destra dell'apice verrà considerato, appunto, come un commento, pertanto non verrà eseguito. Se adesso noi scrivessimo queste tre istruzioni:

```
MsgBox Area1
MsgBox Area2
MsgBox Area3
```

otterremo lo stesso risultato che è stato mostrato prima, cioè le tre finestre di messaggio contenenti rispettivamente `91,5624`, `379,94` e `149,4954`.

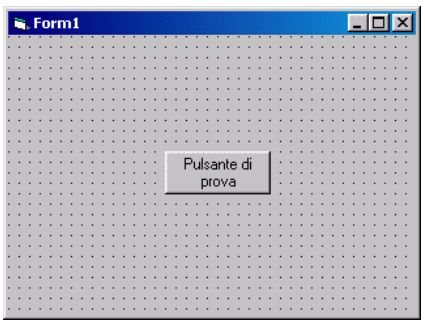
Prima di concludere questa panoramica, dobbiamo ancora presentare alcune istruzioni di VB che sono utilizzate praticamente in tutti i programmi: le **strutture iterative**, che saranno l'argomento della prossima lezione.

La Casella degli strumenti e la finestra delle Proprietà

Tutti i **controlli** che possono essere inseriti un form Visual Basic sono visualizzati sotto forma di icona nella Casella degli strumenti, una barra laterale che nell'impostazione predefinita è visualizzata sulla sinistra; ognuna di queste icone rappresenta un diverso controllo inseribile.

Per provare ad inserire un controllo nel form, fare doppio clic su un'icona contenuta nella casella degli strumenti: l'elemento selezionato verrà visualizzato al centro della finestra. Se ora si seleziona l'elemento appena aggiunto con il mouse, possiamo vedere che la finestra delle "Proprietà" (di solito sulla destra) conterrà tutte le proprietà dell'oggetto stesso.

Modificando queste proprietà è possibile cambiare l'aspetto e le caratteristiche del controllo. Facciamo subito una prova. Fate doppio clic sull'icona della Casella degli strumenti che rappresenta un pulsante; la potete identificare facilmente perché, tenendo il mouse fermo su di essa per qualche istante, verrà visualizzato un tooltip contenente il messaggio "CommandButton".
Un pulsante verrà aggiunto al centro del form; sopra di esso è scritto **Command1**: è, questa, la caption del pulsante. Per modificarla, selezionate il pulsante e fate clic a destra della scritta Caption, visualizzata nella finestra delle Proprietà.



Ora potete digitare la nuova etichetta del pulsante, che verrà modificata durante la digitazione. Ad esempio, provate a scrivere Pulsante di prova. Se avete seguito correttamente questi passaggi, dovreste ottenere un risultato simile a quello mostrato nella figura qui a lato.

C'è anche un altro modo per modificare le proprietà di un controllo inserito in un form: è possibile cambiare la proprietà da codice. In Visual Basic, per accedere da codice alle proprietà di un controllo, è necessario scrivere il nome del controllo stesso (che è il primo valore elencato nella finestra Proprietà), seguito da un punto (.) e dal nome della proprietà che si vuole modificare; poi si

deve digitare un uguale (=) e specificare finalmente il nuovo valore della proprietà. Ritornando al nostro esempio, per modificare la caption del pulsante da codice l'istruzione da scrivere è questa:

```
Command1.Caption = "Pulsante di prova"
```

Notate che, dopo aver digitato il punto, verrà visualizzato un elenco delle proprietà e dei metodi del controllo; mentre le proprietà consentono di impostare le caratteristiche dell'oggetto, i metodi sono azioni che il controllo può eseguire. Ad esempio, utilizzando il metodo Move, possiamo spostare il controllo in una qualsiasi posizione del form:

```
Command1.Move 0, 0
```

Questa istruzione sposta il pulsante nell'angolo in alto a sinistra del form. Come vedremo meglio nelle prossime lezione, ogni controllo dispone di proprietà e di metodi specifici.

Abbiamo così dato un rapido sguardo alla Casella degli strumenti e alla finestra delle "Proprietà" di Visual Basic. Chi volesse approfondire tale punto può selezionare la proprietà di cui vuole conoscere maggiori informazioni e premere il tasto F1: verrà così visualizzata la Guida in linea di Visual Basic relativa alla proprietà selezionata.

Visual Basic un linguaggio event-driven.
Gli **eventi**, cioè le azioni che vengono scatenate dall'utente oppure che sono generate da particolari condizioni (l'impostazione di un timer, la chiusura di Windows, ecc.) sono il **perno** attorno a cui ruota tutta l'attività di programmazione con VB.

I **metodi** invece, sono azioni “built in” ovvero già incapsulate in Visual Basic e richiamabili come una proprietà dell’oggetto. Ad esempio il metodo “setfocus” viene usato per rendere attivo un controllo nel form. Così per attivare il textbox TxtCognome si scriverà **TxtCognome.SetFocus**

Gli eventi fondamentali del mouse

Gli eventi principali che si possono generare con il mouse sono fondamentalmente 5:
Click, DlbClick, MouseDown, MouseUp e MouseMove.
L'evento **Click** si verifica quando un utente fa clic con il tasto sinistro del mouse (o destro, se è mancino) sopra un controllo, come un pulsante, una casella di testo, ecc.
L'evento **DlbClick**, invece, viene scatenato quando si fa doppio clic sul controllo, per convenzione viene usato quando si vuole sveltire un'operazione, facendo compiere all'utente contemporaneamente l'azione di scelta e quella di conferma.
E' importante notare che quando l'utente effettua un doppio clic su un controllo, viene eseguito il codice dell'evento Click e poi quello dell'evento **DblClick**.
Facciamo subito una prova per verificare quanto si è detto. Inserite un controllo pulsante (CommandButton) nel form, come abbiamo visto nella lezione precedente.
Ora fate doppio clic su di esso; verrà visualizzata questa routine:

```
Private Sub Command1_Click()
End Sub
```

E' qui che va inserito il codice che si vuole eseguire quando un utente fa clic sul pulsante. Ad esempio, scrivete:

MsgBox "E' stato fatto clic sul pulsante."

Abbiamo già incontrato in un esempio il comando MsgBox, quando abbiamo parlato di procedure e funzioni. Ora dobbiamo avviare l'applicazione; per fare questo ci sono tre modi: premete il tasto **F5**; fate clic sul menu **Esegui**, quindi sul comando **Avvia**; premete il pulsante **Avvia** nella barra degli strumenti. Apparirà il form con al centro il pulsante che abbiamo inserito; fate clic su di esso: se non avete commesso errori, verrà visualizzata una finestra.

A questo punto, per chiudere il form, fate clic sulla **X**, a destra nella barra del titolo (come in una normale applicazione per Windows).

Nella maggior parte dei casi, gli eventi Click e DblClick sono più che sufficienti per la gestione del mouse, ma in alcuni casi potrà essere necessario sapere quando si preme un pulsante del mouse, quando lo si rilascia oppure quando si sposta il mouse su un controllo: questi eventi sono chiamati, rispettivamente, MouseDown, MouseUp e MouseMove. Per spostarsi negli eventi MouseDown e MouseUp, fate doppio clic sul pulsante: verrà visualizzato il codice che abbiamo scritto precedentemente.

Ora fate clic sulla lista di sinistra per aprire l'elenco e selezionate, ad esempio, MouseDown: sarà ora possibile scrivere il codice che si vuole venga eseguito quando si verifica questo evento. Riprendiamo dunque l'esempio precedente e aggiungiamo alcune istruzioni che ci dicano quando un pulsante del mouse viene premuto e quando, invece, rilasciato:

```
Private Sub Command1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As
Single)
Me.Print "E' stato premuto un tasto del mouse."
End Sub

Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
Me.Print "E' stato rilasciato un tasto del mouse."
End Sub
```

In questo esempio ci sono due cosa da spiegare. La prima è la parola chiave Me, che indica il form corrente, cioè quello in cui si sta eseguendo l'operazione.
Dopo il punto, viene utilizzato il metodo **Print**, che in questo caso ha lo scopo di stampare del testo direttamente sopra il form. Ora eseguite il programma; in tal modo, oltre a verificare in prima persona come funzionano questi eventi, vedrete anche l'ordine in cui essi vengono generati: prima l'evento MouseDown, poi Click e infine MouseUp. Prima vedrete sul form la scritta E' stato premuto un tasto del mouse, subito dopo comparirà la finestra di messaggio E' stato fatto clic sul pulsante e, infine, di nuovo sul form, E' stato rilasciato un tasto del mouse:

Nella prossima lezione ci occuperemo dei principali eventi che vengono generati dalla tastiera, per poi passare finalmente ad analizzare con più attenzione i controlli di VB, a cominciare dal form.

Gli eventi fondamentali della tastiera

Gli eventi fondamentali della **tastiera** sono 4: Change, KeyPress, KeyDown e KeyUp. L'evento Change viene generato quando si modifica il contenuto di un controllo; nel controllo TextBox (nell'immagine a lato è evidenziata la sua posizione nella Casella degli strumenti), ad esempio, tale evento viene scatenato quando cambia il testo in esso contenuto.

Si deve fare attenzione all'uso di questo evento in quanto può innescare degli eventi a catena difficile da prevedere che posso mandare in loop l'applicazione. Ad esempio, se all'interno dell'evento Change di un TextBox si scrive del codice che modifica il testo del TextBox verrà generato di nuovo l'evento Change, e così via.

Quando un utente preme un tasto viene generato l'evento KeyDown, poi si genera l'evento KeyPress e, infine, KeyUp. E' importante notare che l'evento KeyPress viene generato solo se si premono i tasti alfanumerici, i tasti di funzione, e i tasti ESC, Backspace e Invio; l'evento non si verifica, ad esempio, se l'utente preme le frecce direzionali.

A questo punto possiamo creare un **piccolo progetto** di esempio per verificare quanto detto finora. Per le nostre prove utilizzeremo il controllo TextBox, di cui abbiamo parlato poc'anzi.

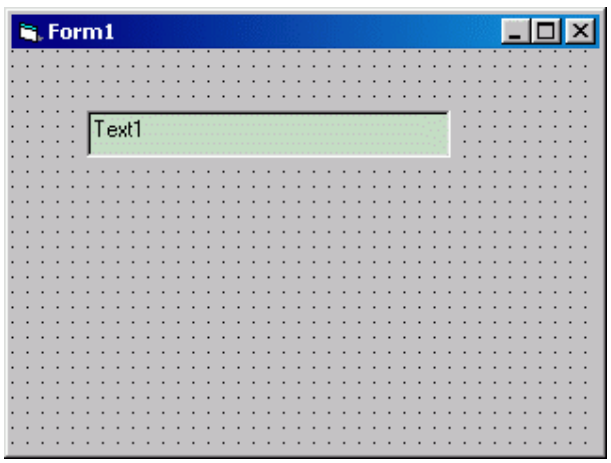
Fate clic sull'icona che lo rappresenta nella Casella degli strumenti, poi portatevi nel punto del form in cui lo volete inserire, fate clic con il tasto sinistro del mouse e, tenendolo premuto, create la casella con la forma che desiderate. Dopo aver rilasciato il tasto del mouse, la TextBox comparirà nel form. Vogliamo ora scrivere il codice per gestire gli eventi Change, KeyPress, KeyUp e KeyDown. Per fare questo, innanzi tutto fate doppio clic sul controllo: comparirà la solita finestra in cui scrivere il codice del programma L'evento in cui ci troviamo è proprio Change:

```
Private Sub Text1_Change()
End Sub
All'interno di questa routine scrivete semplicemente
Me.Print "Il contenuto della casella di testo è cambiato."
```

Il significato della parola Me e del metodo Print sono già stati accennati nella lezione precedente; la parola chiave Me sarà discussa più avanti. Ora premete F5 per avviare il programma e provate a digitare qualcosa nella casella di testo: noterete che, ogni volta che viene premuto un tasto, sul form compare la scritta Il contenuto della casella di testo è cambiato. Dopo aver chiuso il programma con un clic sulla **X** nella barra del titolo, completiamo l'esempio inserendo questo codice:

```
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
Me.Print "Evento KeyDown"
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
Me.Print "Evento KeyPress"
End Sub

Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
Me.Print "Evento KeyUp"
End Sub
```

Eseguite nuovamente il programma e, come prima, digitate qualcosa nella casella di testo; osservate quello che viene scritto sul form di volta in volta.



Abbiamo così analizzato gli eventi fondamentali generati dalla tastiera. Ora possiamo iniziare a parlare più dettagliatamente dell'elemento principale di un'applicazione: il form, l'oggetto della prossima lezione.

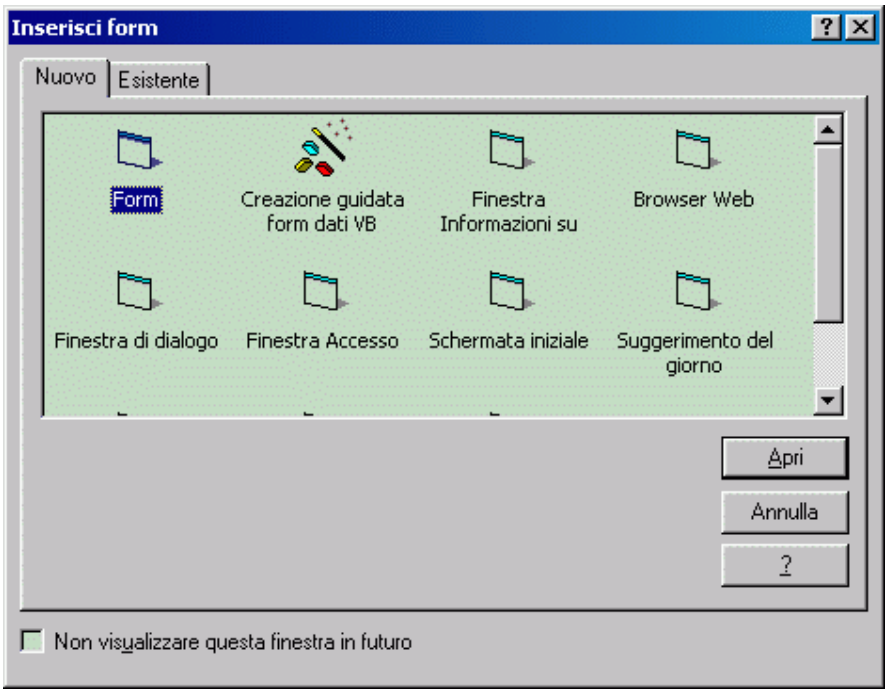
Il form, l'elemento centrale di un'applicazione

La parte fondamentale del disegno di una qualsiasi applicazione è la progettazione dell'interfaccia tra il computer e la persona che lo utilizza. I componenti fondamentali per la creazione dell'interfaccia sono i form e i controlli.

Il **form** (letteralmente "forma", "immagine", ma in questo caso indica genericamente la "finestra" di Windows) è l'elemento centrale di ogni applicazione; un form, infatti, costituisce la base dell'interfaccia utente: è in un form che vengono inseriti tutti gli altri controlli, siano essi pulsanti,

caselle di testo, timer, immagini, ecc. Un form è prima di tutto un oggetto e, come tale, dispone di proprietà che ne definiscono l'aspetto, di metodi e di eventi che permettono di determinare, tra le altre cose, l'interazione con l'utente.

La creazione di un nuovo form è diversa dalla procedura di inserimento di un controllo, che abbiamo visto nelle lezioni precedenti. Per inserire un nuovo form fate clic sul comando **Inserisci form** nel menu **Progetto** oppure fate clic sull'icona corrispondente nella barra degli strumenti (e visibile a lato). A questo punto comparirà la finestra di dialogo **Inserisci form**: per creare un form vuoto dovete fare clic sull'icona Form. In tale finestra potete inoltre notare che VB mette a disposizione diversi modelli di form predefiniti che si possono utilizzare come basi da personalizzare secondo le proprie esigenze.



Ora che sappiamo come aggiungere un nuovo form, possiamo vedere quali sono le proprietà fondamentali. Una delle più importanti è borderStyle, che permette di definire l'aspetto del bordo della finestra.

I valori che può assumere sono 5: None, Fixed Single, Sizable, Fixed Dialog, Fixed ToolWindow e Sizable ToolWindow. I valori più usati sono Sizable (predefinito, vengono visualizzati tutti i pulsanti, di riduzione ad icona, ingrandimento e chiusura, nonché la barra del titolo) e Fixed Dialog (form a dimensione fissa, vengono visualizzati il pulsante di chiusura e la barra del titolo). Potete provare gli altri stili della finestra, che qui sono stati solo accennati, portandovi nella finestra delle Proprietà e andando a modificare la proprietà borderStyle.

Un'altra proprietà importante è la proprietà Caption, che permette di specificare la stringa da visualizzare nella barra del titolo del form. Infine, la proprietà Icon consente di selezionare l'icona che contraddistinguerà il form. Per inserire un'icona, andate come di consueto nella finestra delle Proprietà e ciccate sul pulsante con i tre puntini che appare quando si seleziona la proprietà Icon: così facendo verrà mostrata la finestra di dialogo **Carica icona**, in cui selezionare il file dell'icona (estensione .ICO) desiderata.

Per ogni Form sono 3 gli eventi di base utilizzabili: Form_Load, Form_Activate, Form_Unload

Quando si chiede di mostrare un form, con l'istruzione NomeForm.Show, questo viene caricato in memoria e vengono mostrati i controlli che compongono la parte grafica. Se l'utente ha predisposto l'esecuzione di alcune istruzioni quando il form viene caricato in memoria, le inserisce nella procedura

```
Private Sub Form_Load()
'istruzioni da eseguire
End Sub
```

Ad esempio, posizionare alcuni oggetti, impostare valori di default in alcuni controlli, azzerare alcune variabili. ecc.

Se di deve attivare un controllo, ad esempio usare il metodo SetFocus, allora si deve inserire l'evento

```
Private Sub Form_Activate()
TxtCognome.SetFocus
End Sub
```

L'evento Form_Unload serve per poter controllare la chiusura di un form. Infatti se nel progetto ci sono più form e si chiude un form usando il pulsante X si chiude il form ma il programma è ancora in esecuzione.
L'evento Form_Unload utilizza una variabile di sistema, "Cancel" che se impostata al valore 0 consente la chiusura del form, se impostata a 1 non lo consente.
Se il progetto contiene il form FrmMenu e due Form, FrmUno, FrmDue, Nei form FrmUno e FrmDue si potrebbe impostare l'evento Form_Unload nel modo seguente:

```
Private Sub Form_Unload(Cancel as integer)
        Unload me  ' ovvero scaricami dalla memoria
        FrmMenu.Show
End Sub
```

Invece nel Form FrmMenu che è quello principale del progetto, si potrebbe impostare l'evento nel modo seguente:

```
Private Sub Form_Unload(Cancel as integer)
        Dim SiNo as integer
        SiNo=Msgbox("Vuoi uscire dal programma?",VbQUestion+VbYesNo,"ATTENZIONE")
        If SiNo=VbYes  then          ' Si è cliccato sul tasto "SI"
        End                          ' fine programma
Else
        Cancel=1  ' annulla richiesta di terminazione
End Sub
```

Sviluppando un'applicazione capita praticamente sempre di dover utilizzare più form; quando vogliamo visualizzare un secondo form, ad esempio una finestra contente le opzioni del programma, dobbiamo utilizzare l'istruzione Load, che ha proprio lo scopo di caricare in memoria un form. L'istruzione Show invece lo rende visibile sullo schermo. Ovviamente se si usa

l'istruzione From1.Show, il Form1 viene prima caricato in memoria e poi mostrato. Se invece si carica in memoria in un momento qualsiasi, e poi lo si mostra quando serve, si velocizza l'operazione, specialmente se nel form vi sono molti controlli.

Se vogliamo che un Form non resti visibile, ma non venga scaricato dalla memoria, si deve usare il comendo NomeForm.Hide (che significa Nascondi il form NomeForm). Un form può chiedere a Visual Basic di nasconderlo con l'istruzione Me.Hide

Ad esempio, supponiamo di avere un form principale e di volere che, alla pressione di un pulsante, venga visualizzato un altro form. Innanzi tutto, inseriamo un pulsante nel form; poi aggiungiamo un secondo form al progetto. Ora ci dobbiamo riportare nel primo form, fare doppio clic sul pulsante e, all'interno dell'evento Click, scrivere questo codice:

```
'Carica il form in memoria.
Load Form2
'Visualizza il form.
Form2.Show
```

Ora avviamo il progetto premendo il tasto F5. Verrà visualizzato il primo form; se facciamo clic sul pulsante, si aprirà la seconda finestra: Chiudiamo il secondo form e, quindi, il primo.

Da ultimo, se vogliamo, ad esempio, scaricare un form quando si preme un pulsante, dobbiamo utilizzare l'istruzione Unload:

```
Unload Form1
```

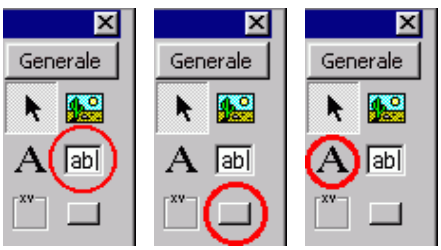
Eseguendo tale istruzione verrà generato l'evento Form_Unload, in cui è possibile annullare l'uscita. Provate ad inserire questa istruzione nell'evento Click di un pulsante, poi premete il tasto F5 per avviare il progetto e fate clic sul CommandButton: se non avete commesso errori, l'applicazione dovrebbe chiudersi.

Un'ultima nota: se, all'interno di un form, si vuol fare riferimento al form stesso, invece del suo nome è possibile utilizzare la parola chiave Me: Ad esempio, invece di scrivere Unload Form1, dal momento che la finestra che vogliamo chiudere è la stessa in cui viene richiamata la funzione Unload, sarebbe stato lecito (e di solito è la pratica più comune) scrivere Unload Me.

CommandButton, TextBox e Label

Analizziamo i controlli messi a disposizione da Visual Basic per costruire l'interfaccia utente. Iniziamo col descrivere i controlli CommandButton, TextBox e Label.

Il controllo CommandButton (pulsante di comando) è stato già descritto negli esempi precedenti.



In generale, il **CommandButton** è utilizzato con una didascalia (la caption) e, opzionalmente, un'immagine che fanno comprendere immediatamente all'utente l'azione che verrà eseguita quando il pulsante sarà premuto.

L'evento più utilizzato del CommandButton è il Click, ed è in esso che si dovrà scrivere il codice da eseguire alla pressione del pulsante. Per inserire un'immagine nel pulsante si deve modificare la proprietà Style in 1 - Graphical e quindi fare clic sulla proprietà Picture, in modo da far apparire la finestra di dialogo in cui selezionare l'immagine (di tipo bitmap, icona, metafile, GIF e JPG). Quest'ultima proprietà si può anche impostare in fase di esecuzione, utilizzando l'istruzione LoadPicture. Supponiamo di avere un'immagine di tipo bitmap nella cartella C:IconeApp.bmp; dopo aver settato la proprietà Style su Graphical, il comando da eseguire per associare l'immagine al pulsante è:

```
Command1.Picture = LoadPicture("C:IconeApp.bmp")
```

Impostando la proprietà Default su true è possibile definire un pulsante come predefinito, in modo che se l'utente preme il tasto Invio da un qualsiasi punto della finestra il controllo passi al pulsante attivando l'evento Click. In modo analogo, settando la proprietà Cancel su true, il pulsante verrà attivato quando viene premuto il tasto Esc (questo è il tipico caso di un pulsante Annulla).
Il controllo **TextBox** (casella di testo) offre all'utente la possibilità di visualizzare, modificare e immettere informazioni. Quando si inserisce un TextBox in un form, in esso viene visualizzato il nome del controllo; per modificarne il contenuto è sufficiente utilizzare la proprietà Text, disponibile nella finestra delle Proprietà e accessibile da codice:

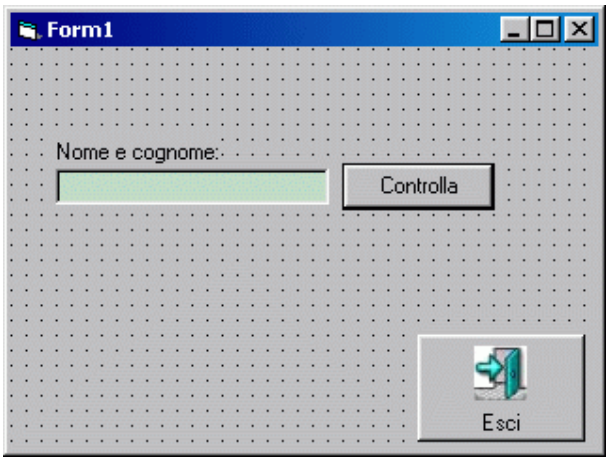
```
Text1.Text = "Testo di prova"
```

Per modificare le modalità di visualizzazione si possono usare le proprietà alignment e font. La prima consente di impostare l'allineamento del testo all'interno della casella è può assumere i seguenti valori:
0 - Left Justify (testo allineato a sinistra);
1 - Right Center (allineato a destra);
2 - Center (centrato).
La proprietà font, invece, è usata per modificare il carattere con cui viene visualizzato il testo: facendo clic su tale proprietà viene visualizzata una finestra di dialogo per selezionare il tipo di carattere tra quelli installati nel computer, la dimensione, lo stile e gli effetti. Se si desidera modificare il colore del testo è necessario modificare la proprietà ForeColor. Per impostazione predefinita una casella di testo non permette di andare a capo, ma il testo viene visualizzato su un'unica riga che scorre verso destra nel momento in cui si raggiunge il margine del controllo. Per consentire la digitazione su più righe è sufficiente impostare la proprietà MultiLine su true e la proprietà ScrollBars su 3 - Both, così da mostrare le barre di scorrimento che consentono di visualizzare un testo più lungo rispetto alle dimensioni del controllo. Infine, se si vuole limitare la lunghezza massima del testo che può essere digitato in una TextBox basta impostare la proprietà MaxLength sul numero massimo di caratteri che si possono immettere.

Il controllo **Label** (etichetta) è solitamente usato per rendere più esplicativa l'interfaccia, ad esempio come didascalia o commento di una casella di testo; a differenza di quest'ultima, l'utente non può modificare direttamente il testo visualizzato in una Label. Un'altra diversità è che per impostare il testo di un'etichetta si deve usare la proprietà Caption e non Text, che non è presente. Un aspetto comune tra i due controlli, invece, è che anche in una Label, appena inserita in un form, viene visualizzato il nome del controllo. E' possibile fare in modo che il controllo si ridimensioni

automaticamente per adattarsi alla lunghezza del testo impostando la proprietà Autosize su true; la proprietà WordWrap, infine, permette l'estensione del testo su più righe.

Realizziamo ora un piccolo esempio per mettere in pratica quanto abbiamo detto finora. Vogliamo creare una piccola applicazione con una casella di testo in cui si deve immettere il proprio nome e un pulsante che controlla la validità del testo immesso. Per rendere l'interfaccia più amichevole aggiungiamo anche un'etichetta per informare l'utente sul tipo di dati che deve immettere e un pulsante con un'icona per uscire dal programma.



Innanzi tutto inseriamo un controllo **TextBox** nel form; al suo interno verrà visualizzato il nome del controllo, in questo caso Text1. Eliminiamo questo testo modificando la proprietà Text. Ora a sinistra della casella di testo posizioniamo un pulsante e modifichiamo la sua proprietà Caption su Controlla; modifichiamo inoltre la sua proprietà Default settandola su true. Ora aggiungiamo l'etichetta esplicativa: creiamo una Label sopra la casella di testo e impostiamo la sua Caption su Nome e cognome:. Infine aggiungiamo un pulsante nell'angolo in basso a destra del form, modifichiamo il sua Caption su Esci, la proprietà Cancel su true e Style su 1 - Graphical. Ora fate clic sulla proprietà Picture del CommandButton e selezionate l'icona da visualizzare sul pulsante. Dopo queste operazioni il form dovrebbe risultare come quello mostrato a lato.
Ora dobbiamo scrivere il codice del programma. Vogliamo che, premendo il tasto Controlla, venga controllata la proprietà Text della casella di testo e venga visualizzato un messaggio diverso a seconda che in essa sia contenuto o no del testo. Infine, premendo il tasto Esci, l'applicazione deve terminare. Nella routine Command1_Click andremo a scrivere:

```
'Controlla la proprietà Text.
If Text1.Text = "" Then MsgBox "Digitare il nome e il cognome." Else MsgBox
"Valori corretti."
```

Questa semplice istruzione controlla il valore della proprietà Text; se è uguale a "" (stringa vuota), significa che non è stato digitato nulla, quindi visualizza un messaggio che chiede di inserire i dati richiesti; altrimenti, qualora sia stato digitato qualcosa, informa l'utente che i valori sono corretti.
L'ultima cosa che ci resta da fare è scrivere il codice per chiudere il programma quando si preme il tasto **Esci**.
Come abbiamo visto nella lezione precedente, per fare questo basta scrivere nell'evento Command2_Click l'istruzione Unload Me.
Adesso complichiamo il programma. Vogliamo fare in modo che, se l'utente preme il tasto **Esci** senza aver digitato nulla nella casella di testo, venga visualizzata una finestra di dialogo e l'uscita

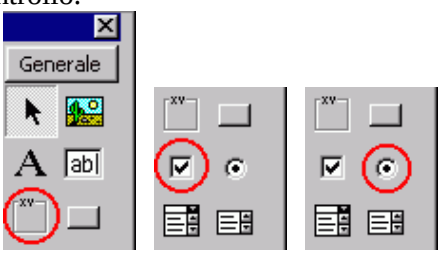
venga annullata. Possiamo raggiungere questo obiettivo scrivendo il codice di controllo nell'evento Form_QueryUnload e, se necessario, impostando il parametro Cancel su true per annullare l'uscita:

```
Private Sub Form_Unload(Cancel As Integer)
'Controlla se nella casella è stato digitato qualcosa.
If Text1.Text = "" Then
'Non è stato digitato nulla; visualizza un messaggio.
MsgBox "Immettere il nome e il cognome per uscire."
'Sposta lo stato attivo sul controllo TextBox.
Text1.SetFocus
'Annulla l'uscita.
Cancel = 1
End If
End Sub
```

L'unica istruzione che merita qualche commento è **Text1.SetFocus**. Il metodo SetFocus ha lo scopo di spostare lo stato attivo sul controllo, cioè, nel caso specifico, di visualizzare il cursore all'interno della TextBox, cosicché sia possibile digitare al suo interno.

I controlli Frame, CheckBox e OptionButton

Esaminiamo i controlli Frame, CheckBox e OptionButton.
Il controllo **Frame** (cornice) permette di raggruppare elementi dell'interfaccia logicamente legati fra loro; in un Frame, ad esempio, possiamo inserire tutte le opzioni relative al salvataggio dei file, oppure alla personalizzazione dell'interfaccia, ecc. La comodità del Frame è che tutti gli elementi inseriti in esso vengono trattati come un "blocco" unico; ad esempio, se si nasconde un Frame, verranno automaticamente nascosti anche tutti i controlli al suo interno. Per inserire un elemento in un Frame è necessario disegnarlo all'interno della cornice stessa; fatto questo, sarà possibile spostarlo solo entro i margini del Frame. trattandosi di un contenitore di altri oggetti, di solito non vengono gestiti i suoi eventi, ma ci si limita a modificare poche proprietà, innanzi tutto la Caption, per modificare l'etichetta del controllo.



Il controllo **CheckBox** (casella di controllo) è rappresentato graficamente da un'etichetta con a fianco una casella di spunta, nella quale viene visualizzata una crocetta quando viene selezionato. Solitamente è utilizzato per visualizzare una serie di opzioni tra cui selezionare quelle desiderate. Anche il CheckBox dispone della proprietà Caption, per modificarne l'etichetta. La proprietà più importante di questo controllo è la Value, che permette di impostare o recuperare lo stato del CheckBox, cioè di sapere se esso è selezionato oppure no. I valori che può assumere sono tre:
0 - Unchecked, il controllo non è selezionato;
1 - Checked, il controllo è selezionato;
2 - Grayed, il controllo è disabilitato nello stato di selezionato.
Per cambiare lo stato del controllo è possibile modificare la proprietà Value anche da codice:

```
Check1.Value = vbUnchecked 'Deseleziona il controllo.
Check1.Value = vbChecked 'Seleziona il controllo.
Check1.Value = vbGrayed 'Deseleziona il controllo nello stato di selezionato.
```

vbUnchecked, vbChecked, vbGrayed sono costanti definite da Visual Basic e valgono rispettivamente 0, 1 e 2; si possono quindi usare senza differenza le costanti, come mostrato sopra, oppure i loro valori numerici. L'evento più utilizzato del CheckBox è l'evento Click, che si verifica quando si modifica lo stato del controllo; di solito, in tale evento si inserisce il codice per attivare o disattivare altri controlli in accordo con la scelta effettuata. Come esempio, diamo un'occhiata a questa routine:

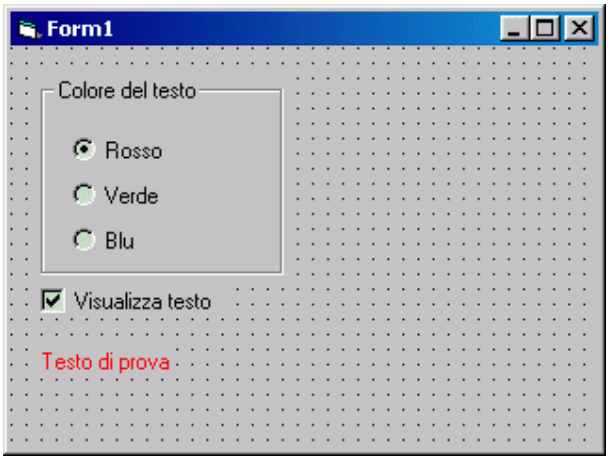
```
Private Sub Check1_Click()
If Check1.Value = vbChecked Then
'Il controllo è stato selezionato.
Check1.Caption = "Il controllo è stato selezionato."
Else
'Il controllo è stato deselezionato.
Check1.Caption = "Il controllo è stato deselezionato."
End If
End Sub
```

Quando si verifica l'evento Click, viene controllato se il controllo è stato selezionato o deselezionato e, quindi, cambia in modo opportuno la Caption del CheckBox. E' importante notare che l'evento Click viene generato anche quando si modifica la proprietà Value da codice.
Il controllo **OptionButton** (pulsante di opzione) è simile al CheckBox, ma con la differenza che in un gruppo di controlli OptionButton è possibile selezionare un solo elemento alla volta, mentre è possibile selezionare più controlli CheckBox contemporaneamente.
Anche per questo tipo di controlli le proprietà più utilizzate sono la Caption e la Value, che in tal caso assume i valori true (controllo selezionato o False (controllo non selezionato).

Solitamente gli OptionButton sono raggruppati in un altro controllo, come un Frame, perché VB presume che tutti i pulsanti di opzione presenti nello stesso form appartengano al medesimo gruppo; di conseguenza è possibile selezionare solo un OptionButton alla volta tra quelli presenti in uno stesso gruppo.

Anche l'OptionButton dispone dell'evento Click, che si verifica quando si modifica il suo stato, ovvero quando viene selezionato facendo clic su di esso.

Esempio:
Vogliamo creare un semplice programma che visualizzi, all'interno di un Frame, una serie di opzioni per modificare il colore del testo di una Label.
Una casella di controllo, poi, deve permettere di visualizzare o nascondere l'etichetta. Per semplicità, d'ora in poi indicheremo solo gli elementi che si vogliono inserire nel form e quali proprietà andranno modificate; è possibile fare riferimento all'immagine visualizzata a lato per avere un'idea della disposizione dei controlli.



Ora dobbiamo inserire il codice del nostro programma. Intuitivamente, vogliamo che quando l'utente esegue un clic su uno degli OptionButton, il colore del testo venga modificato. Per intercettare la pressione del tasto del mouse sopra il controllo sopra menzionato dobbiamo utilizzare l'evento Click; per modificare il colore del testo, invece, è necessario modificare la proprietà ForeColor dell'etichetta Label1. Di seguito il codice:

```
Private Sub Option1_Click()
'Si è scelto di visualizzare il testo in rosso.
Label1.ForeColor = vbRed
End Sub

Private Sub Option2_Click()
'Si è scelto di visualizzare il testo in verde.
Label1.ForeColor = vbGreen
End Sub

Private Sub Option3_Click()
'Si è scelto di visualizzare il testo in blu.
Label1.ForeColor = vbBlue
End Sub
```

Anche in questo caso sono state utilizzate le costanti definite da Visual Basic per i codici dei colori; nella Guida in linea è possibile trovare tutte le informazioni a riguardo. Ora però vogliamo completare l'esempio aggiungendo il codice che vogliamo venga eseguito quando si fa clic sul controllo Check1. In questo caso, sempre nell'evento Click, dobbiamo verificare lo stato del controllo (cioè se è selezionato oppure no) e, di conseguenza, visualizzare o nascondere la Caption:

```
Private Sub Check1_Click()
If Check1.Value = vbChecked Then
'La casella di controllo viene selezionata; visualizza la Caption.
Label1.Visible = true
Else
'La casella di controllo viene deselezionata; nasconde la Caption.
Label1.Visible = False
End If
End Sub
```

Ora non ci resta che premere il tasto F5 per avviare il progetto, che è anche disponibile per il download facendo clic qui.

I controlli ListBox e ComboBox

In questa lezione ci occuperemo di altri due controlli di VB molto usati: ListBox, e ComboBox. Il ListBox (cerchiato in blu) e il ComboBox (cerchiato in rosso) sono utilizzati con lo stesso scopo, cioè visualizzare una lista di opzioni selezionabili dall'utente; la differenza fondamentale tra i due controlli è il modo in cui la lista è presentata.
Nel ListBox, infatti, gli elementi della lista sono sempre visibili, mentre nel ComboBox la lista è "a scomparsa", ovvero è visibile soltanto se l'utente fa clic sulla freccia verso il basso a destra del controllo; nel ComboBox, inoltre, è possibile digitare un valore che non compare nella lista di quelli disponibili.

Per il resto, i due controlli si comportano nello stesso modo e molte proprietà e metodi funzionano allo stesso modo. Per aggiungere valori ad un ListBox o ad un ComboBox si possono seguire due strade: utilizzare la proprietà List disponibile nella Finestra delle proprietà oppure il metodo AddItem richiamabile da codice. Vediamo un esempio di questa seconda soluzione:

```
List1.AddItem "Marco"
List1.AddItem "Luigi"
List1.AddItem "Andrea"
```

Tali istruzioni aggiungono tre valori in una ListBox; lo stesso identico codice funziona con una ComboBox, basta cambiare il nome dell'oggetto. Per eliminare tutte le voci da un elenco si deve richiamare il metodo Clear:

```
List1.Clear
```

Se, invece, si desidera eliminare un ben preciso elemento, è necessario conoscerne la posizione all'interno della lista (il cosiddetto indice). A questo proposito, è importante ricordare che l'indice delle voci ha base 0, ovvero il primo elemento di una lista ha indice 0, il secondo ha indice 1, il terzo 2, e così via. Se, ad esempio, vogliamo rimuovere il sesto elemento contenuto in un controllo ListBox, l'istruzione da utilizzare è:

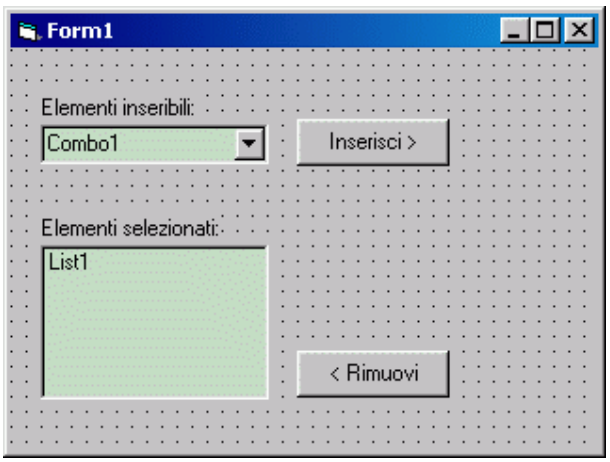
```
List1.RemoveItem 5
```

Se l'indice specificato non esiste (ad esempio si tenta di cancellare il quinto elemento di una lista che ha solo quattro elementi) verrà generato un errore.
Per recuperare l'indice della voce selezionata dall'utente si deve controllare la proprietà ListIndex, sia per le ListBox sia per le ComboBox; anche in questo caso il valore restituito parte da 0. Se nessun elemento della lista è stato selezionato, il valore restituito da ListIndex sarà -1. Questa

proprietà può anche essere utilizzata per selezionare da codice un particolare elemento della lista; ad esempio, se vogliamo selezionare il secondo elemento di una ListBox, è sufficiente scrivere:
List1.ListIndex = 1
Detto questo, è molto semplice rimuovere l'elemento selezionato in una lista; per compiere tale operazione, infatti, basta il comando:
List1.RemoveItem List1.ListIndex
La proprietà Text restituisce la stringa selezionata in un ListBox o visualizzata in un ComboBox.

Se, invece, vogliamo conoscere il valore di un particolare elemento di una lista, possiamo utilizzare la proprietà List, che richiede come parametro l'indice dell'elemento che si vuole recuperare. Ad esempio:
MsgBox List1.List(2)
'Visualizza una finestra di messaggio contenente il valore del terzo elemento della lista.

Come abbiamo già detto, le proprietà e i metodi sopra esposti sono comuni sia al controllo ListBox sia al controllo ComboBox. Qualche parola va però ancora spesa a proposito della proprietà Style del ComboBox. Essa può assumere tre valori: 0 - Dropdown combo (casella combinata a discesa, predefinita), comprende una casella di riepilogo a discesa e una casella di testo; l'utente potrà eseguire una selezione nella casella di riepilogo o digitare nella casella di testo; 1 - Simple combo (casella combinata semplice), comprende una casella di testo ed una casella di riepilogo non a discesa, cioè sempre visibile; l'utente potrà eseguire una selezione nella casella di riepilogo o digitare nella casella di testo.
Le dimensioni di una casella combinata semplice si riferiscono a entrambi i componenti. Per impostazione predefinita, la casella è dimensionata in modo che nessun elemento dell'elenco sia visualizzato; per visualizzare elementi dell'elenco, aumentare il valore assegnato alla proprietà Height; 2 - Dropdown list (casella di riepilogo a discesa), questo stile consente soltanto la selezione dall'elenco a discesa.



Ora che abbiamo visto come funzionano i controlli ListBox e ComboBox, facciamo un esempio pratico del loro utilizzo, così da fissare meglio i concetti che sono stati trattati.
La nostra applicazione sarà così strutturata: una ComboBox conterrà un elenco di elementi; selezionandone uno e facendo clic su un pulsante, la voce selezionata verrà inserita in una ListBox e rimossa dal ComboBox; un altro pulsante, poi, permetterà di rimuovere l'elemento dalla ListBox e di reinserirlo nella ComboBox. Come sempre, potete scaricare il form contenente solamente gli elementi dell'interfaccia nella giusta posizione, senza il codice.
Prima di tutto dobbiamo popolare il ComboBox con gli elementi che possono essere selezionati dall'utente; utilizzeremo il metodo AddItem nell'evento Form_Load, che, come abbiamo già detto, viene eseguito quando il form viene caricato in memoria. Inseriamo quindi un certo numero di voci:

```
Private Sub Form_Load()
Combo1.AddItem "Scheda madre"
Combo1.AddItem "Processore"
Combo1.AddItem "Monitor"
Combo1.AddItem "Videocamera"
Combo1.AddItem "Stampante"
Combo1.AddItem "Scanner"
'Seleziona il primo elemento.
Combo1.ListIndex = 0
End Sub
```

Ora vogliamo fare in modo che, premendo il pulsante Inserisci, l'elemento corrente venga inserito nel ListBox e, subito dopo, venga rimosso dal ComboBox:

```
Private Sub Command1_Click()
'Inserisce l'elemento selezionato nel ListBox.
List1.AddItem Combo1.Text
'Rimuove l'elemento dal ComboBox.
Combo1.RemoveItem Combo1.ListIndex
If Combo1.ListCount > 0 Then
'Se ci sono ancora elementi, seleziona il primo elemento del ComboBox.
Combo1.ListIndex = 0
Else
'Altrimenti, disattiva il pulsante "Inserisci".
Command1.Enabled = False
End If
End Sub
```

Ogni istruzione è commentata, è necessario spendere qualche parola solo sul ciclo If... Then... Else. Dopo l'inserimento di una voce, l'elemento viene rimosso dal ComboBox; a questo punto, la routine controlla quanti elementi sono rimasti, utilizzando la proprietà ListCount: se è maggiore di 0, cioè se sono presenti ancora voci nella lista, seleziona la prima, altrimenti disattiva il pulsante, poiché non possono essere selezionati e quindi inseriti altri elementi.
Ora dobbiamo inserire il codice che verrà eseguito premendo il tasto Rimuovi: facendo clic su tale pulsante l'elemento selezionato nella ListBox verrà rimosso e reinserito nella ComboBox:

```
Private Sub Command2_Click()
'Innanzi tutto, controlla se è stato selezionato un elemento nel ListBox.
If List1.ListIndex >= 0 Then
'Reinserisce la voce nel ComboBox.
Combo1.AddItem List1.Text
'Rimuove l'elemento dal ListBox.
List1.RemoveItem List1.ListIndex
'Riattiva il pulsante "Inserisci".
Command1.Enabled = true
Combo1.ListIndex = 0
End If
End Sub
```

Notiamo che il codice viene eseguito solo se la proprietà ListIndex del ListBox è maggiore o uguale a 0, ovvero se è stato selezionato un elemento nella lista.
Abbiamo così scritto tutto il codice necessario al funzionamento del nostro esempio, che come sempre potete scaricare facendo clic qui .

i controlli ImageBox e PictureBox

I controlli ImageBox (immagine - cerchiato in rosso) e PictureBox (casella immagine - cerchiato in blu) consentono di visualizzare immagini nella propria applicazione. Questi due oggetti sono per alcuni versi simili e dispongono di proprietà analoghe. Il controllo PictureBox, però dispone di numerose proprietà in più e può anche essere utilizzato come contenitore di altri controlli, alla stregua di un controllo Frame, di cui abbiamo parlato nella lezione 10 "I controlli Frame, CheckBox e OptionButton": allo scopo, anche in questo caso sarà sufficiente creare un oggetto all'interno di una PictureBox.
Per visualizzare un'immagine nel controllo PictureBox si usa la proprietà Picture, disponibile nella finestra delle proprietà del controllo: basta fare clic sui tre puntini a destra del campo Picture per visualizzare la finestra Carica immagine, in cui è possibile selezionare il file da caricare; i formati supportati sono BMP, GIF, JPG, WMF, EMF, ICO e CUR.

Per caricare un'immagine da codice, invece, si usa la funzione LoadPicture, come nel seguente esempio:

```
Picture1.Picture = LoadPicture("C:\DocumentiProva.bmp")
```

Questa istruzione carica il file Prova.bmp, contenuto nella cartella C:Documenti, e lo visualizza nel controllo Picture1. Una proprietà importante di questo controllo è Autosize, che consente di impostare il tipo di ridimensionamento da adottare: se Autosize è true, il controllo si ridimensiona automaticamente per adattarsi alle dimensioni dell'immagine caricata.

Come già accennato, il controllo Image è più semplice del PictureBox, infatti supporta solo alcune proprietà eventi e metodi del PictureBox e non può essere utilizzato come contenitore di altri controlli. Per visualizzare un'immagine, si possono usare la proprietà Picture o la funzione LoadPicture, in modo analogo al PictureBox.
Non esiste la proprietà Autosize, al cui posto si può usare la proprietà Stretch: se è False il controllo Image si ridimensiona adattandosi all'immagine caricata; se, invece, è true, l'immagine assume le dimensioni del controllo, quindi potrebbe risultare distorta.

Il principio di funzionamento di questi due controlli è semplice. Creiamo un progetto composto da un frame, al cui interno di trovano 4 OptionButton, e una CheckBox e un controllo Picture: vogliamo che, a seconda del pulsante di opzione selezionato, venga visualizzata una diversa immagine; la casella di controllo, poi, consente di adattare le dimensioni della PictureBox a quelle dell'immagine.

Abbiamo già realizzato una applicazione che utilizzava degli OptionButton per consentire all'utente di compiere determinate scelte; in questo caso vogliamo che, facendo clic su ciascuno di essi, venga visualizzata una diversa immagine nel controllo PictureBox. Ecco il codice che realizza quanto detto:

```
Private Sub Option1_Click()
Picture1.Picture = LoadPicture(App.Path & "Merlin.gif")
End Sub

Private Sub Option2_Click()
Picture1.Picture = LoadPicture(App.Path & "Genie.gif")
End Sub
```

```
Private Sub Option3_Click()
Picture1.Picture = LoadPicture(App.Path & "Robby.gif")
End Sub

Private Sub Option4_Click()
Picture1.Picture = LoadPicture(App.Path & "Peedy.gif")
End Sub
```

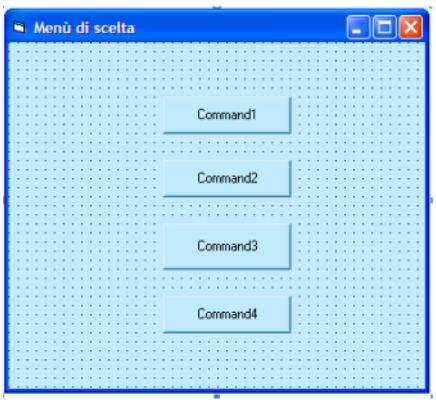
Notate che in questo codice è stata usata la proprietà App.Path, che restituisce il percorso completo in cui è memorizzato il progetto VB; ad esempio, se il file VBP del progetto è salvato nella cartella C:\DocumentiLavori, la proprietà App.Path restituirà proprio C:\DocumentiLavori. E' stato inoltre utilizzato l'operatore &, che ha lo scopo di concatenare, cioè unire, due stringhe distinte. Ora l'unica cosa che resta da fare è scrivere il codice per attivare la proprietà Autosize del controllo PictureBox; questo codice andrà inserito nell'evento Click del CheckBox:

```
Private Sub Check1_Click()
If Check1.Value = vbChecked Then Picture1.Autosize = true
End Sub
```

Avviare il programma: facendo clic su uno qualunque dei pulsanti di opzione verrà visualizzata un'immagine diversa; facendo clic su Autosize, infine, il controllo PictureBox verrà ridimensionato per adattarsi alle dimensioni dell'immagine. Fate clic qui per scaricare l'esempio completo (senza le immagini).

Matrici di controlli

Una matrice di controlli con è altro che un array di controlli, dello stesso tipo, che consentono di attivare una delle possibili alternative proposte dal programma.
Ad esempio se si deve scegliere quale Form attivare tra 4 possibili si può procedere ad inserire 4 CommandButton e dare a ciascuno un nome diverso. Facendo doppio click su ciascuno si attiva l'evento click proprio del pulsante:

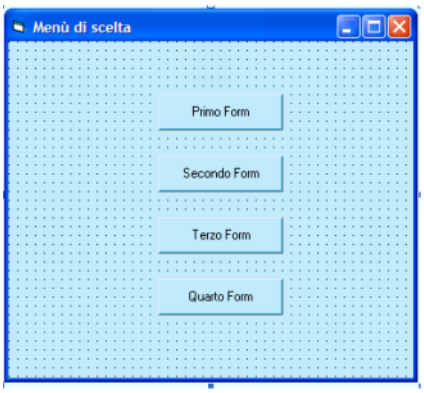



```
Private Sub Command1_Click()
Me.Hide
Form1.Show
End Sub

Private Sub Command2_Click()
Me.Hide
Form2.Show
End Sub

Private Sub Command3_Click()
Me.Hide
Form3.Show
End Sub

Private Sub Command4_Click()
Me.Hide
Form4.Show
End Sub
```

Oppure si può inserire un pulsante e chiamarlo ad esempio CmdX.
Quindi selezionarlo, copiarlo e incollarlo. Visual Basic rileverà che esiste già un controllo con quel nome e segnalerà all'utente se vuole creare una matrice di controlli. Rispondendo di SI, verrà creato un array: il primo pulsante si chiamerà CmdX(0), il secondo CmdX(1).
Incollando per altre due volte il pulsante verrà creato un array di controlli che differenzieranno per l'indice che li identifica, come gli elementi di un array.

Per attivare uno dei 4 form che costituiscono il progetto, non sarà più necessario scrivere 4 sottoprogrammi. Sarà sufficiente fare doppio click su uno qualunque dei pulsanti creati e verrà attivato un solo sottoprogramma che avrà come parametro l'indice del pulsante appena selezionato.

Utilizzando poi l'istruzione Select Case Index si potrà scorrere le varie possibilità attivando il form corrispondente all'indice del pulsante.




```
Private Sub CmdX_Click(Index As Integer)
Me.Hide
Select Case Index
    Case 0
    Form1.Show
    Case 1
    Form2.Show
    Case 2
    form3.Show
    Case 3
    form4.Show
End Select
End Sub
```

I controlli DriveListBox, DirListBox e FileListBox

I controlli DriveListBox (cerchiato in rosso), DirListBox (blu), FileListBox (verde), e consentono di accedere ai dischi installati nel sistema e alle informazioni in essi contenute. Di solito vengono utilizzati insieme, allo scopo di fornire un sistema di navigazione tra le risorse del sistema; grazie alle proprietà di cui dispongono, infatti, è semplice sincronizzare questi tre controlli.

Il controllo DriveListBox (casella di riepilogo dei drive) visualizza le unità di memorizzazione presenti nel sistema (floppy, dischi rigidi, lettori di CD-ROM, ecc.).
Per cambiare l'unità selezionata si usa la proprietà Drive, che può essere modificata solo da codice (cioè disponibile solo in fase di esecuzione):

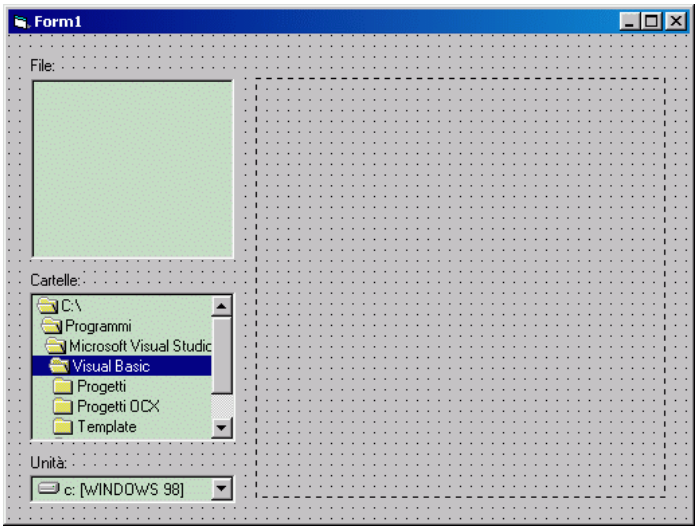
Drive1.Drive = "D:"

Specificando una lettera di unità non valida o non disponibile, verrà generato un errore. Quando si modifica l'unità selezionata nel controllo, viene generato l'evento **Change**.

Il controllo DirListBox (casella di riepilogo delle directory) visualizza le cartelle presenti nell'unità selezionata; facendo doppio clic sul nome di una directory è possibile spostarsi al suo interno.

La cartella corrente può essere modificata utilizzando la proprietà Path, che come la proprietà Drive del DriveListBox è disponibile solo in fase di esecuzione:

Dir1.Path = "C:\Documenti"



Analogamente al DriveListBox, se si specifica una cartella non valida verrà generato un errore; la modifica della directory corrente genera l'evento Change.

Infine, il controllo FileListBox (casella di riepilogo dei file) visualizza i file che si trovano nella cartella corrente. Esso dispone di alcune proprietà che consentono di selezionare quali file visualizzare. Innanzi tutto, con la proprietà Pattern è possibile stabilire i nomi dei file visualizzati; è possibile utilizzare i caratteri jolly * e ? e, inoltre, si possono specificare più criteri separandoli con un punto e virgola (senza spazi tra un criterio e l'altro).

Ad esempio, win*.exe restituirà l'elenco di tutti i file eseguibili il cui nome inizia con "win", mentre *.gif;*.jpg restituirà l'elenco di tutti i file GIF e di quelli JPG. Le proprietà Archive, Hidden, Normal, ReadOnly e System consentono di stabilire se il controllo FileListBox deve visualizzare file con gli attributi, rispettivamente, di Archivio, Nascosto, Normale, Sola lettura e File di sistema. Anche il controllo FileListBox dispone della proprietà Path, con la quale è possibile modificare la cartella di cui si vuole visualizzare il contenuto. Quando si seleziona un file viene generato l'evento Click; il nome del file corrente è conservato nella proprietà FileName del controllo FileListBox; ad esempio, se vogliamo che, quando si seleziona un file, venga visualizzata una finestra di messaggio contenente il nome del file stesso, basterà scrivere:

```
Private Sub File1_Click()
MsgBox File1.FileName
End Sub
```

Con un esempio cerchiamo ora di capire come si possono utilizzare insieme questi tre controlli; l'applicazione che vogliamo realizzare è un classico visualizzatore di immagini. Per questo avremo bisogno di un DriveListBox, un DirListBox, un FileListBox e un controllo Image; utilizzeremo anche alcune Label per rendere l'utilizzo dell'applicazione più immediato. Cominciamo innanzi tutto col costruire l'interfaccia del programma.

Ricordando quanto detto prima, iniziamo a scrivere il codice. Selezionando una unità nel controllo **DriveListBox** viene generato l'evento Change, che dobbiamo utilizzare per aggiornare la proprietà Path del DirListBox, in modo che quest'ultimo visualizzi le cartelle dell'unità selezionata. Per raggiungere lo scopo è sufficiente scrivere:

```
Private Sub Drive1_Change()
Dir1.Path = Drive1.Drive
End Sub
```

Notate che, selezionando una unità non disponibile verrà generato un messaggio di errore. Ora dobbiamo collegare i controlli DirListBox e FileListBox, in modo che quest'ultimo visualizzi i file della cartella selezionata; l'evento che dobbiamo utilizzare è il Change del DirListBox:

```
Private Sub Dir1_Change()
File1.Path = Dir1.Path
End Sub
```

Con queste semplici istruzioni abbiamo collegato tra loro i tre controlli; se volete verificarne il funzionamento, premete F5 e provate a navigare tra le risorse del sistema, selezionando unità e cartelle diverse. A questo punto dobbiamo ancora scrivere le istruzioni per visualizzare nell'**ImageBox** l'immagine selezionata; il codice necessario è questo:

```
Private Sub File1_Click()
Dim FileSelezionato As String

FileSelezionato = File1.Path & "" & File1.FileName
Image1.Picture = LoadPicture(FileSelezionato)
End Sub
```

Innanzi tutto dichiariamo una nuova variabile, di tipo String, in cui memorizzeremo il nome e il percorso completo del file selezionato; per fare questo, usiamo la proprietà Path di File1, ad essa aggiungiamo il carattere "" con l'operatore di concatenazione tra stringhe (&), e infine recuperiamo il nome del file con la proprietà FileName. L'ultima istruzione carica nel controllo Image1 il file dell'immagine che è stata selezionata.

L'applicazione è pronta: premete F5 e verificatene il corretto funzionamento.

il controllo Timer

Il controllo Timer consente di gestire azioni collegate al trascorrere del tempo; il suo funzionamento non dipende dall'utente e può essere programmato per l'esecuzione di operazioni a intervalli regolari. Un tipico utilizzo del Timer è il controllo dell'orologio di sistema per verificare se è il momento di eseguire una qualche attività; i timer sono inoltre molto utili per altri tipi di operazioni in background.

Il controllo Timer si differenzia dai controlli esaminati finora perché, in fase di esecuzione, è invisibile; dispone di un solo evento, l'evento Timer, che viene generato quando è trascorso l'intervallo di tempo (espresso in millisecondi) specificato nella proprietà Interval. Quest'ultima può assumere valori compresi tra 0, che equivale a disattivare il timer, e 65.535, cioè poco più di un minuto.
E' buona norma impostare nella proprietà Interval un valore corrispondente a circa la metà del tempo che vogliamo trascorra effettivamente tra due eventi Timer: ad esempio, se vogliamo che l'evento Timer venga generato ogni secondo (cioè 1000 millisecondi), conviene impostare la proprietà Interval su 500, per evitare ritardi dovuti ad altre elaborazioni del computer che

avvengono nello stesso momento. Un'altra proprietà importante è la proprietà Enable, che consente di stabilire o di impostare se il Timer è attivo oppure no.

Per comprende il funzionamento del controllo Timer realizzeremo un'applicazione classica, un orologio digitale. L'interfaccia, che potete scaricare facendo clic qui , è molto semplice e comprende, oltre al Timer, solo una Label.
Notate che la proprietà Interval è stata impostata su 500 millisecondi per la ragione sopra indicata. L'unica riga di codice che dobbiamo scrivere va inserita nell'evento Timer:

```
Private Sub Timer1_Timer()
Label1.Caption = Time
End Sub
```

Per recuperare l'ora di sistema è stata usata la funzione **Time**. Tutto qui: il programma è finito, premete F5 per osservare il suo funzionamento.

la gestione degli errori in Visual Basic

Quando si esegue un programma, anche il più semplice, c'è sempre la possibilità che accada qualcosa che non è stato previsto durante la progettazione, cioè che si verifichi un errore, una situazione inaspettata che l'applicazione non è in grado di gestire.
Quando si presenta un problema del genere, il programma visualizza un messaggio e, subito dopo, termina la sua esecuzione. Un errore si verifica, ad esempio, se si cerca di effettuare una divisione per zero, se si tenta di caricare in una PictureBox un file immagine inesistente, ecc.
Un programma ben sviluppato dovrebbe prevedere delle procedure in grado di risolvere gli errori più comuni; continuando gli esempi precedenti, se si effettua una divisione per zero dovrebbe apparire un messaggio che informa che l'operazione non è valida e, allo stesso modo, se ci cerca di aprire un file inesistente, l'utente dovrebbe essere avvisato e avere la possibilità di modificare la sua scelta.

In VB, la gestione degli errori si effettua utilizzando i cosiddetti gestori di errori. Il loro utilizzo è molto semplice. Per installare un gestore degli errori, è sufficiente scrivere (di solito come prima istruzione all'interno di una routine):

```
On Error GoTo <Etichetta>
```

Dove è una sorta di "segnalibro" in corrispondenza del quale inizia il codice incaricato di gestire gli errori. A questo punto se, all'interno della routine, si verifica un errore (chiamato errore di run-time), l'esecuzione passa immediatamente alla parte di codice che si trova in corrispondenza dell'etichetta definita con On Error... Provate, ad esempio, a scrivere queste istruzioni nell'evento Form_Load:

```
Dim A As Long, B As Long, C As Long
A = 5
B = 0
C = A / B 'Questa istruzione causa un errore di divisione per zero.
```

```
MsgBox "Il risultato della divisione è: " & C
```
figura

Ora premete F5: quando VB cercherà di eseguire l'istruzione C = A / B, mostrerà una finestra di errore, contenente due tipi di informazioni, il numero dell'errore e una breve descrizione dello stesso. Premendo il tasto Fine l'esecuzione del programma verrà terminata; facendo clic su Debug VB evidenzierà l'istruzione che ha causato l'errore, consentendo la sua modifica; il pulsante ?, infine, visualizza la Guida in linea relativa all'errore che si è appena verificato. Il pulsante Continua, attivo solo in certe situazioni, permette di continuare l'esecuzione del programma ignorando l'errore.
figura

Cerchiamo di modificare il codice in modo da gestire l'errore. Come abbiamo detto, per prima cosa dobbiamo creare un gestore degli errori con l'istruzione On Error GoTo <Etichetta>, dopodiché dobbiamo scrivere il codice di gestione vero e proprio. Ecco come possiamo modificare l'esempio:

```
Private Sub Form_Load()
Crea il gestore degli errori.
On Error GoTo GestoreErrori
Dim A As Long, B As Long, C As Long
A = 5
B = 0
C = A / B Questa istruzione causa un errore di divisione per zero.
MsgBox “Il risultato della divisione è: “ & C
Exit Sub

GestoreErrori:
‘Queste istruzioni vengono eseguite quando si verifica un errore.
If Err.Number = 11 Then
‘Divisione per zero.
MsgBox "Si è verificato un errore di divisione per zero. Controllare i valori digitati e riprovare."
End If
End Sub
```

Analizziamo quanto abbiamo scritto. L'istruzione **On Error Goto GestoreErrori** crea il gestore degli errori, cioè dice al programma che, in caso di errori, deve passare ad eseguire le istruzioni scritte sotto l'etichetta GestoreErrori. L'istruzione Exit Sub ha lo scopo di uscire dalla routine senza eseguire le istruzioni seguenti; notate che, se invece di una routine fossimo stati in una funzione, l'istruzione per uscire da essa sarebbe stata Exit Function. All'interno del gestore degli errori viene usato l'oggetto Err, che contiene informazioni relative agli errori di run-time; in particolare, in questo esempio controlliamo il valore di Err.Number, che restituisce il numero dell'ultimo errore verificatosi. Alcune volte si usa la proprietà Err.Description, la quale contiene la descrizione dell'errore. Ora provate a premere F5 per eseguire il codice; osserverete che non comparirà più la finestra di errore di VB, ma la MessageBox che abbiamo definito noi:

Un'altra istruzione importante è l'istruzione Resume, che riprende l'esecuzione dopo il completamento di una routine di gestione degli errori. Solitamente è usata in due modi: Resume, riprende l'esecuzione dalla stessa istruzione che ha causato l'errore; Resume Next, riprende l'esecuzione dall'istruzione successiva a quella che ha provocato l'errore. Per esempio, sostituite questo pezzo di codice

```
GestoreErrori:
'Queste istruzioni vengono eseguite quando si verifica un errore.
If Err.Number = 11 Then
'Divisione per zero.
MsgBox "Si è verificato un errore di divisione per zero. Controllare i valori digitati e riprovare."
End If
```

Con

```
GestoreErrori:
'Queste istruzioni vengono eseguite quando si verifica un errore.
If Err.Number = 11 Then
'Divisione per zero.
MsgBox "Si è verificato un errore di divisione per zero. Controllare i valori digitati e riprovare."
B = 1
Resume
End If
```

Il nuovo codice, dopo aver visualizzato il messaggio di errore, cambio il valore di B da 0 a 1 e infine, con un Resume, torna all'istruzione che aveva provocato l'errore, cioè C = A / B. Ora la divisione (che è diventata 5 / 1) viene eseguita correttamente, pertanto otterremo:

Adesso provate a sostituire l'istruzione Resume che abbiamo appena scritto con Resume Next: come abbiamo detto, con essa l'esecuzione salta all'istruzione successiva a quella che ha provocato l'errore, quindi C = B / A non verrà più eseguita, ma il programma passerà subito all'istruzione MsgBox "Il risultato della divisione è: " & C. Modificate il codice come suggerito e osservate il risultato.
L'istruzione Resume Next può anche essere usata insieme all'istruzione On Error: in questo modo si dice al programma che, ogni volta che si verifica un errore, il problema deve essere ignorato e l'esecuzione deve passare all'istruzione successiva. Per fare questo è sufficiente scrivere:

```
On Error Resume Next
```

Di solito è sconsigliabile seguire questa strada, perché, non gestendo gli errori, si possono verificare delle situazioni imprevedibili.

Dopo aver parlato della gestione degli errori in VB, possiamo aggiornare il visualizzatore di immagini che abbiamo realizzato nella lezione "I controlli DriveListBox, DirListBox e FileListBox", in modo da correggere quegli errori che si verificano, ad esempio, quando si seleziona un'unità non disponibile. Cominciamo proprio da questo punto. Avviate il programma e, nel DriveListBox, selezionate l'unità A: senza che nessun dischetto sia inserito: verrà generato l'errore di runtime 68, indicante una "periferica non disponibile". Dobbiamo aggiungere la gestione di questo errore nell'evento Drive1_Change, che viene eseguito quando si seleziona un'unità. Il codice originale

```
Private Sub Drive1_Change()
Dir1.Path = Drive1.Drive
End Sub
```

Va modificato in questo modo:

```
Private Sub Drive1_Change()
On Error GoTo GestoreErrori
Dir1.Path = Drive1.Drive
Exit Sub

GestoreErrori:

If Err.Number = 68 Then
'Periferica non disponibile.
MsgBox "Impossibile accedere all'unità specificata. Selezionare un'unità diversa e riprovare."
Drive1.Drive = Dir1.Path
End If
End Sub
```

Come abbiamo detto, l'errore che dobbiamo gestire è il 68, quindi, come prima cosa, nella routine di gestione utilizziamo la proprietà Err.Number per controllare il numero dell'errore: se il problema è dovuto ad una periferica non valida, visualizziamo un messaggio di errore e, con l'istruzione Drive1.Drive = Dir1.Path, facciamo sì che l'unità selezionata corrisponda a quella di cui si stanno visualizzando le cartelle.

Un'altra situazione di errore nel programma si può avere quando, nel FileListBox, si seleziona un file immagine non valido, ad esempio perché danneggiato oppure perché, nonostante l'estensione, è un file di un altro tipo. In questo caso, selezionando tale file, verrà generato l'errore di runtime 481, cioè "immagine non valida". Ora il codice da modificare è quello contenuto nell'evento File1_Click; ecco la nuova routine:

```
Private Sub File1_Click()
On Error GoTo GestoreErrori
Dim FileSelezionato As String

FileSelezionato = File1.Path & "" & File1.FileName
Image1.Picture = LoadPicture(FileSelezionato)
Exit Sub

GestoreErrori:

If Err.Number = 481 Then
'Immagine non valida.
MsgBox "L'immagine selezionata non è valida."
End If
End Sub
```

Dopo quello che abbiamo detto il significato del codice appena scritto dovrebbe essere chiaro; come prima, in caso di errore, viene visualizzata una finestra di messaggio che informa sul problema.

C'è ancora un errore che dobbiamo gestire e che, a differenza di quelli visti finora, è un po' più difficile da identificare, perché si verifica solo in una determina situazione. Consideriamo la riga di codice

```
FileSelezionato = File1.Path & "" & File1.FileName
```

Essa salva nella variabile FileSelezionato il nome e il percorso completo del file; per fare questo, recupera il percorso del file, vi aggiunge un back-slash ("\") e, infine, inserisce il nome del file. Questa procedura funziona bene se la cartella in cui ci si trova non è quella principale del disco (quindi non è C:, A: e così via). Se invece, siamo, ad esempio, in C:, la proprietà File1.Path contiene già il back-slash, perciò, dopo l'esecuzione dell'istruzione sopra riportata la variabile FileSelezionato conterrà un valore di questo tipo: C:File.bmp. Stando così le cose, l'istruzione Image1.Picture = LoadPicture(FileSelezionato) causerà l'errore di runtime 76, cioè l'errore "impossibile trovare il percorso".

Adesso che abbiamo imparato a scrivere il codice per gestire gli errori potremmo ampliare la routine già scritta, ma in questo caso è più conveniente prevenire il problema, piuttosto che correggerlo una volta che si è verificato. L'idea che sta alla base della soluzione è questa: per evitare l'errore, dovremmo fare in modo che il back-slash venga aggiunto solo se non è già l'ultimo carattere della proprietà File1.Path. A tale scopo possiamo utilizzare la funzione Right$(Stringa, n), che restituisce gli ultimi n caratteri della stringa specificata. Ad esempio, l'istruzione Right$("Prova", 2) restituisce la stringa "va". Vediamo ora il codice vero e proprio:

```
Private Sub File1_Click()
On Error GoTo GestoreErrori
Dim FileSelezionato As String

If Right$(File1.Path, 1) = "\" Then
FileSelezionato = File1.Path & File1.FileName
Else
FileSelezionato = File1.Path & "\" & File1.FileName
End If
Image1.Picture = LoadPicture(FileSelezionato)
Exit Sub

GestoreErrori:
If Err.Number = 481 Then
'Immagine non valida.
MsgBox "L'immagine selezionata non è valida."
End If
End Sub
```

Come abbiamo anticipato, la modifica introdotta controlla se la proprietà File1.Path comprende già il back-slash come ultimo carattere e solo in caso negativo lo aggiunge alla stringa FileSelezionato. Potete scaricare la nuova versione del visualizzatore immagini che abbiamo realizzato in questa lezione facendo clic qui.

Aggiungere un controllo OCX al progetto

Finora abbiamo analizzato i controlli standard di Visual Basic, quelli sempre presenti nella Casella degli strumenti e che, quindi, possono essere utilizzati in tutti i programmi. In VB, come in tutti i linguaggi di programmazione, è possibile aggiungere nuovi controlli al progetto, rendendo così disponibili nuovi oggetti da inserire nel programma, nuove funzioni, ecc.

Tali controlli vengono chiamati in due modi: controlli OCX (o più brevemente OCX), dal momento che .ocx è l'estensione del file che li contiene, oppure controlli ActiveX.
Una volta inseriti in un progetto, gli OCX possono essere trattati come un qualunque controllo standard, dal momento che mettono a disposizione proprietà, metodi ed eventi. Per inserire un controllo ActiveX in un progetto Visual Basic, è necessario fare clic sul comando Componenti (Components) del menu Progetto (Project); nella finestra di dialogo che si aprirà è possibile scegliere un ActiveX dall'elenco selezionando la relativa casella di controllo. Con un clic su OK o su Applica, esso verrà inserito nel progetto e sarà disponibile nella Casella degli strumenti insiemi agli altri componenti standard.

Proviamo subito ad utilizzare uno dei controlli OCX aggiuntivi forniti con VB, quello che permette di visualizzare le finestre di dialogo Apri, Salva con nome, Carattere, ecc. comuni a tutte le applicazioni Windows.
Aprite la finestra Componenti seguendo le istruzioni riportate sopra e scorrete l'elenco fino a trovare Microsoft Common Dialog Control; selezionate questo controllo OCX con un clic sulla casella di spunta visibile a lato del nome e chiudete la finestra di dialogo con un clic sul pulsante OK.

Ora nella Casella degli strumenti apparirà una nuova icona, corrispondente ad un nuovo controllo che è possibile utilizzare nella propria applicazione. Inseritelo nel form.
Nella finestra delle Proprietà del controllo potete notarne una chiamata (personalizzate): selezionatela e fate clic sul pulsante con i tre puntini visibile a destra: si aprirà una nuova finestra in cui è possibile modificare le proprietà del controllo.
Ora premete Annulla. Impostate la proprietà CancelError su true, in modo che venga generato un errore di runtime se l'utente preme il tasto Annulla nelle finestre di dialogo (ci servirà per l'esempio che andremo a realizzare).
figura

Per provare il controllo vogliamo richiamare la finestra di dialogo Apri per selezionare un file. In questo contesto ci interessa solo analizzare il comportamento dell'OCX, quindi per richiamarlo andrà benissimo un semplice pulsante.
Inserite dunque un CommandButton nel form e modificate la sua Caption, ad esempio, in "Apri file". Il Microsoft Common Dialog Control dispone di cinque metodi fondamentali, ShowOpen, ShowSave, Showfont, ShowColor e ShowPrinter, per visualizzare le finestre di dialogo rispettivamente per l'apertura di un file, per il salvataggio, per la selezione del carattere, per la selezione del colore, per l'impostazione della stampante.
figura

Ad essi va aggiunto il metodo ShowHelp, che visualizza la Guida in linea. Proviamo, ad esempio, ad utilizzare il metodo ShowOpen:

```
Private Sub Command1_Click()
'Visualizza la finestra di dialogo per l'apertura di un file.
CommonDialog1.ShowOpen
'Visualizza una MessageBox contenente il nome del file selezionato.
MsgBox "Il file selezionato è: " & CommonDialog1.FileName
End Sub
```

In questo codice, come detto, viene utilizzato il metodo ShowOpen, dopodiché si usa la proprietà FileName per recuperare il nome completo del file selezionato.

Dopo aver fatto qualche prova per verificare il funzionamento di questa routine, provate ad aprire la finestra ma, invece di selezionare un file, premete il tasto Annulla; se come suggerito avete impostato la proprietà CancelError su true, a questo punto verrà generato l'errore di runtime 32755. Ecco quindi un'altra situazione in cui dobbiamo prevedere una gestione degli errori. Dopo quello che si è detto nelle precedenti lezioni l'aggiunta non dovrebbe essere difficile da realizzare, ecco come dovrà apparire il codice dopo la modifica:

```
Private Sub Command1_Click()
On Error GoTo GestoreErrori
'Visualizza la finestra di dialogo per l'apertura di un file.
CommonDialog1.ShowOpen
'Visualizza una MessageBox contenente il nome del file selezionato.
MsgBox "Il file selezionato è: " & CommonDialog1.FileName
Exit Sub

GestoreErrori:

If Err.Number = 32755 Then
'E' stato premuto Annulla.
MsgBox "E' stato premuto il pulsante 'Annulla'."
End If
End Sub
```

Per maggiori informazioni su questo e, in generale, su tutti gli altri controlli OCX utilizzabili con VB, si raccomanda di consultare la Guida in linea che li accompagna.