VISUAL BASIC
Manuale Introduttivo

SOMMARIO

PER COMINCIARE

Per cominciare a programmare in Visual Basic occorrono alcune nozioni fondamentali di programmazione e soprattutto familiare con l'ambiente ad oggetti.
Cosa vuol dire programmare? Cosa vuol dire programmare ad oggetti? Queste sono le domande cui risponderemo nei prossimi capitoli.

Iniziare un progetto

In visual basic, un nuovo programma, in fase di creazione, prende il nome di progetto. Per avviare un nuovo progetto occorre scegliere il l'opzione nuovo dal menu file dell'interfaccia Visual Basic. Automaticamente al progetto viene assegnata l'estensione .VBP. Si può a questo punto scegliere tra ben nove tipi di progetto, anche se comunemente quello più utilizzato è il tipo .exe standard.
Nella finestra opzioni che compare sono presenti anche dei Wizard (Creazione guidata di…) che saranno utili soprattutto per i principianti alle prime prese con Visual Basic. Sconsigliamo però di utilizzare troppo questi meccanismi perché anche se da un lato facilitano i primi approcci, dall'altro non rendono pienamente comprensibili i meccanismi di funzionamento della programmazione ad oggetti.
Con i progetti si può lavorare in più modi, ad esempio, facendo riferimento alla modularità della programmazione, possiamo aggiungere un progetto ad un altro, mediante l'opzione Aggiungi progetti del menu File, riusando così del codice già scritto.
Per rimuovere invece il progetto utilizzeremo il comando rimuovi progetto dal menu file, dopo aver selezionato il progetto in questione nella finestra progetto.
Ovviamente per salvare un progetto si utilizzerà l'omonimo comando dal menu File. In questo modo verranno salvati anche i files ad esso collegati.

Programmazione ad oggetti significa utilizzare un ambiente di programmazione in cui le varie parti del codice sono riunite in varie classi che definiscono tutti i componenti che entrano in gioco nello sviluppo di un programma. Un programma oggi è infatti concepito in modo diverso da quanto avveniva in passato. Non abbiamo più i programmi costituiti da un elenco di istruzioni processate in sequenza, ma piuttosto i vari oggetti di un programma rispondono ad eventi causati dal sistema o dall'utente (es. click del mouse, apertura di un file) interagendo tra loro, ciascuno mediante un codice associato agli oggetti stessi in risposta all'evento verificatosi. Possiamo dunque defini il progetto come il "contenitore" di oggetti con i relativi codici che costituiscono il nostro programma.
Nel caso di Visual Basic, gli altri componenti (oggetti) che ritroviamo in un progetto sono:

- la FORM, ovvero la maschera, l'interfaccia grafica che permette all'utente di interagire con il programma, ovvero di scatenare eventi (click di un bottone, visualizzazione di un campo, ecc.
- MDI (Multiple Document Interface) che è l'interfaccia a documenti multipli (vedremo più avanti di cosa si tratta)
- Modulo, ovvero il codice che definisce una procedura o una funzione utilizzata nel programma
- Modulo di classe, cioè il codice associato ad una specifica classe
- Controllo utente, cioè controlli (bottoni, check box o quant'altro) personalizzati

- Documento, ovvero documenti personalizzati presenti nel progetto
- Data Report, report di progetto
- File esterni, che possono essere referenziati e quindi utilizzati dall'interno del programma
- Microsoft User Connection, per avere a disposizione una connessione a database per l'utilizzo di dati in esso contenuti
- DHTML page, pagine HTML dinamiche (pagine HTML usuali con controlli e campi)
- WebClass, classe orientata al Web (per applicazioni Web)
- ActiveX, perché nella finestra di progettazione si possono includere anche oggetti ActiveX

Tutti i comandi per l'introduzione in un progetto degli elementi che abbiamo elencato sono presenti nel menu File della finestra di progetto. Allo stesso modo, per rimuovere un oggetto da un progetto, basta selezionarne il nome dell'oggetto in questione e scegliere l'opzione rimuovi nel menu Progetto.

LA FINESTRA PROGETTO

Chiamata anche Gestione Progetti, questa finestra permette di organizzare e sviluppare tutti i componenti del progetto stesso, mostrati in un elenco gerarchico nella parte sinistra dello schermo. Nella finestra di progetto ritroviamo i seguenti elementi:

- Pulsante Visualizza Codice: apre la finestra che mostra il codice sorgente del progetto, in cui intervenire per apportare opportune modifiche
- Pulsante Visualizza Oggetto: apre la finestra relativa all'oggetto (form, controllo, ecc) selezionato
- Pulsante Espandi/Comprimi cartelle che permette di visualizzare in modo diverso la cartella che contiene gli elementi di un progetto
- Finestra di progetto: mostra tutti i progetti attualmente in uso/caricati e tutti gli elementi che li costituiscono. In particolare vi ritroviamo le voci:
  - Progetto: progetto ed elementi che lo compongono
  - Form: tutte le form (.frm) associati al progetto
  - Moduli: nomi dei moduli (.bas) utilizzati dal progetto
  - Moduli di classe: nomi dei moduli di classe (.cls) utilizzati all'interno del progetto
  - Controlli utente: tutti i controlli presenti nel progetto
  - Documenti utente (.dob) che sono inclusi nel progetto
  - Pagine delle proprietà: pagine di definizione delle proprietà presenti nel progetto (.pag)
  - Documenti correlati: documenti a cui è possibile/definito l'accesso dall'interno del progetto
  - Risorse: elenco di tutte le risorse disponibili per il progetto

Un progetto si salva mediante l'apposito comando dal menu FILE.

GESTIRE I FORM

I form MDI (Multiple Document Interface) permettono di migliorare sensibilmente l'aspetto dei propri programmi, permettendo di mantenere aperti contemporaneamente più documenti nell'applicazione. Traducendo la cosa in oggetti, si avrà un form principale da cui dipendono uno o più form secondari.

Creare un form

Visual Basic permette di creare progetti composti da più form, specialistici o generici. Per creare un nuovo form nel progetto corrente, si sceglie il comando Inserisci form dal menu Progetto. Apparirà una finestra in cui scegliere il tipo di form desiderato (con la Creazione guidata, si procede con l'aiuto dei wizard). I modelli di form proposti che appaiono nella finestra sono di seguito descritti:

- Finestra di dialogo **"Informazioni su",** che permette di inserire una finestra di informazioni sul progetto;
- Finestra di dialogo **"Accesso",** che è quelle che, quando richiamata, chiede all'utente il proprio identificativo e la password per accedere all'esecuzione del programma;
- Finestra di dialogo **"Opzioni",** che è suddivisa in sezioni e permette agli utenti del programma di accedere alla personalizzazione dell'applicazione;
- Videata **"Schermata Iniziale"**, che permette di creare una schermata iniziale contenente, per esempio, il nome del programmatore, il logo dell'azienda produttrice e altre informazioni varie ed eventuali;
- Videata **"Suggerimento del giorno",** che visualizzerà all'interno dell'applicazione
- Videata **"ODBC Login"** che presenta le opzioni per connettersi ad un database;
- **"Browser Web"** che permette di aggiungere all'applicazione una pagina web di Internet;
- **"Creazione guidata form dati VB"** per essere aiutati nella generazione di un form per elaborare o mostrare a video informazioni contenute in un database.

Quando in un progetto sono presenti più form dello stesso tipo, automaticamente verrà mostrato il primo. Per evitare ciò, occorre selezionare il comando *Proprietà di Nome progetto* dal menu *Progetto*. A questo punto si seleziona (nella finestra che appare) la selezione *Generale* e nell'elenco *Oggetto di avvio* si seleziona il nome corrispondente al form che si desidera venga avviato per primo. *OK* per confermare le impostazioni.

Form MDI

Un'applicazione di tipo MDI (Multiple Document Interface) permette di operare contemporaneamente con più documenti. (es. tutti i programmi Microsoft che attraverso il menu Windows permettono di gestire più documenti). Con Visual Basic si possono creare tali programmi. La particolarità di implementazione dei form in questo caso è che si dovrà avere solo un form MDI principale e almeno uno secondario (figlio). Il form principale viene indicato in Visual Basic come MDIForm. Un MDIForm si crea scegliendo l'opzione Inserisci form MDI dal menu Progetto.
I form secondari sono designati come MDIchild (cioè che hanno l'opzione omonima impostata a true nelle proprietà).

Se in un form secondario si prevede la presenza di menu diversi da quello principale, la barra dei menu cambierà automaticamente all'attivazione del form.

Gestire i form MDI

Per la gestione dei form MDI devono essere eseguite le seguenti operazioni:

- le variabili vanno adoperate a livello di form per gestire le informazioni correlate a form secondari;
- vanno utilizzati i gestori di evento per rispondere al sistema di menu che è parte inegrante del form MDI principale;
- eventi speciali come Load() e Unload() devono essere utilizzati per la gestione del caricamento e dello scaricamento del form MDI principale. Tali eventi permettono di inizializzare i form secondari e di scaricarli quando si chiude l'applicazione.

Ovviamente si possono anche gestire gli eventi associati ad operazioni effettuate con il mouse.

Aggiungere form MDI secondari

Abbiamo detto che un form MDI principale (MDIparent) può avere uno o più form figli che vengono definiti tali nella finestra Object Explorer. La finestre Object Explorer contiene sempre almeno un form figlio. Una implementazione fissa di form MDI principale con un numero prestabilito di form figli è di più semplice gestione ma limita le possibilità MDI. E' meglio dunque prevedere un numero di form secondari variabili e quindi lasciare all'utente la devisione di quanti form MDI secondari utilizzare.

Per gestire i form secondari di un progetto, occorre utilizzare a livello di form una delle varibili riportate di seguito:

- un oggetto di tipo Collection che memorizzi i form secondari come oggetti;
- un indice di form secondari che tenga traccia del numero totale dei form secondari creati via via.Il programma deve incrementare il valore di questo indice ogni volta che viene creato un nuovo form secondario e non lo decrementerà qundo verrà chiuso un form figlio, in modo tale da fornire sempre un identificativo univoco per ogni form secondario. E' consigliabile porre tale numero identificativo come parte dell'etichetta del form secondario.

Vediamo ora come aggiungere ad un progetto un form secondario.

1. Per prima cosa occorre dichiarare una variabile locale che crei un nuovo form secondario;
2. incrementare poi di 1 il valore dell'indice del form secondario;
3. caricare il form secondario mediante il comando Load object;
4. a questo punto visualizzare il nuovo form secondario attraverso l'uso del metodo Show( );
5. infine aggiungere il nuovo form secondario alla serie di oggetti servendosi del metodo Add(oggettoDaAggiungere).

Il codice che segue è un esempio di gestore di evento associato al comando new, che crea e visualizza un nuovo form secondario.

```
Private Sub mnuNew-Click( )
        Dim MDIchildForm as New_MDIchild
        'incrementa la variabile di livello del form
        MDIchildIndex = MDIchildIndex + 1
        Load MDIchildForm
        MDIchildForm.Caption = "MDI dipendente"
        Str(MDIchildIndex"
        MDIchildForm.Show
        ColMDIform.Add MDIchildForm
End Sub
```

Il codice dichiara una variabile locale (individuare qual è) che crea un nuovo form secondario, incrementa di 1 l'indice di livello del form secondario, carica il form, inserisce il valore desiderato nella caption, visualizza il nuovo form e lo aggiunge alla serie di oggetti relativi.

Arrange di form MDI dipendenti

Visual Basic permette di gestire form o icone all'interno di un oggetto MDIform utilizzando il metodo Arrange( ), la cui sintassi generale è:
MDIparentForm.Arrange tiposistemazione, in cui MDIparentForm è il nome del form MDI principale e tiposistemazione conrrisponde ad uno dei seguenti valori:

1. vbCascade (valore 0) se si vuole che i form secondari siano sovrapposti tra loro in cascata;
2. vbTileHorizontal (valore 1) per affincare orizzontalmente tutti i form dipendenti;
3. vbTile Vertical (valore 2) se si vuole affincarli verticalmente;
4. vbArrangeIcons (valore 3) se si vogliono ridurre i form ad icona sistemandoli in modo automatico.

Esmpio di istruzioni:

```
Private Sub mnuArrangeIcons_Click( )
        MDIparent.Arrange vbArrangeIcons
End Sub
```

Sistema le icone come nel caso 4.

Chiudere un form secondario attivo

Quando si desidera chiudere un form secondario attivo è necessario accedere alla proprietà AcriveForm del form MDI principale. Quando si usa un oggetto Collection per la gestione del form MDI dipendente, occorre utilizzare a tal fine una procedura che ricerchi il form attivo, scarichi il form MDI secondario e rimuova tale form dall'oggetto Collection.

Esempio:

```
Private Sub mnuClose_Click( )
        Dim I As Integer
        'ricerca il form MDI attivo
        For I = 1 To colMDIforms.Count
                'confronta le proprietà Caption
                If col.ColMDIforms.Item(I).Caption = MDIparent.ActiveForm.Caption
Then
                        'trovato il form MDI attivo, lo scarica
                        Unload colMDIforms.Item(I)
                        'rimuove il form dalla serie
                        colMDIform.Remove I
                        'esce dopo aver rimosso il form MDI secondario
                        Exit Sub
                End If
        Next I
End Sub
```

Chiudere tutti i form secondari

Se si utilizza, per la gestione, l'oggetto Collection, l'operazione è semplice: occorre scaricare i form secondari tramite l'Unload MDIchildForm e rimuoverli dall'oggetto Collection utilizzando il metodo Remove(indice)

Esempio:

```
Private Sub mnuClose_Click( )
        Dim I As Integer
        For I = colMDIforms.Count to 1 Step -1
                'scarica il form MDI
                Unload colMDIforms.Item(I)
                'rimuove il form MDI dalla serie dei form
                volMDIforms.Remove I
        Next I
End Sub
```

CONTROLLI SEMPLICI

All'interno di un form possiamo inserire vari tipi di controlli (bottoni, campi, comboBox, ecc…) In questo capitolo vedremo come si inseriscono e si gestiscono tali controlli.

Pulsante bitmap

Un Bitmap Button (Pulsante Bitmap) permette a chi sta sviluppando un progetto di inserire immagini bitmap (o icone) all'interno dei pulsanti di un'applicazione Visual Basic. Un Bitmap Button ha varie prorpietà che per la maggior parte possono essere anche impostate da programma utilizzando la sintassi (es.) cmdDisable.caption ="&Disabilita"

Command button

Il command button (Pulsante di comando) è il più utilizzato nelle applicazioni Windows. Serve infatti a far eseguire una data procedura. Anche il controllo Command Button ha alcune proprietà rilevanti, come il controllo illustrato nel paragrafo precedente e che vengono impostati dal programmatore. Gli eventi di più frequente attivazione del Command Button sono Click, LostFocus e GotFocus.

Check box

I controlli di tipo Check Box (casella di controllo) sono composti da un quadrato che mostra una x quando l'utente desidera selzionare l'ozione associata (etichetta). Ovviamente quando si disattiva la casella, la x scompare. (x vale si/vero, l'assenza della x vale no/falso). Più caselle di controllo possono essere attivate contemporaneamente.
Le proprietà più utilizzate in questo caso sono Value e Caption.
La prima ha la sintassi:
checkBoxControl.Value e indica lo stato di una casella di controllo (0 se è vuota, 1 se è selezionata). Viene utilizzata per controllare lo stato di una casella o per verificarne lo stato attivo.ù
La proprietà Caption ha invece la sintassi:
checkBoxControl.Caption che visualizza, sotto forma di stringa, il testo descrittivo dell'opzione associata ad una casella di controllo, in modo tale, ad esempio, di sapere quale opzione ha scelto l'utente.
In entrambi i casi CheckBoxContro sta per il nome del Check Box.

Combo box

Una Combo Box (casella combinata) è un controllo composto da una casella di elenco voci e da un'area di immissione. Un controllo di questo tipo permette all'utente sia di selezionare una voce presente nell'elenco proposto, si di inserire manualmente una nuova voce. Si consiglia di provare le varie opzioni che è possibile impostare dal menu proprietà della casella combinata.

Le proprietà possono essere impostate/verificate da programma mediante la sintassi generica:
cboBox.nomeproprietà, dove cboBox è il nome della Combo Box

Nella tabella che segue sono riportati invece i metodi che possono essere associati ad una combo box.

Metodo: sintassi	**Descrizione**	**Esempio**	**Commento**
AddItem: cboBox.AddItem voce, indice	Aggiunge una nuova voce all'elenco della casella	CboPippo.AddItem "pippo", 0	Aggiunge la striga "pippo" come prima voce nell'elenco
Clear: cboBox.Clear	Elimina le voci dall'elenco della casella	CboNames.Clear	Elimina tutte le voci dall'elenco della casella
RemoveItem: cboBox.RemoveItem indice	Rimuve la voce specificata mediante l'indice (la prima voce ha indice 0)	CboBox.RemoveItem 0	Rimuove la prima voce dell'elenco

In quest'altra tabella, invece, sono riportati gli eventi rilevanti che si possono associare ad una casella combinata.

Evento	**Descrizione**
Click	Si verifica quando si fa click all'interno dell'elenco
DblClick	Si verifica quando si effettua un doppio click nell'evento e quindi si seleziona una voce dell'elenco
Change	Si verifica quando il contenuto di una casella viene modificato
Scroll	Quando si fa scorrere il contenuto dell'elenco

Frame

Un controllo Frame (Riquadro) viene utilizzato per raggruppare elementi simili tra loro (esempio caselle di controllo o pulsanti di opzione), o per suddividere idealmente un form perché vi si svolgono operazioni distinte. Bisogna fare attenzione a verificare che il controllo Riquadro sia abilitato, poiché in caso contrario si disabilitano anche gli oggetti in esso racchiusi.

List Box

Il controllo List Box (Elenco di Voci) visualizza un elenco di voci dal quale l'utente potrà selezionarne una o più (selezione più MAIUSC per le voci contigue o CTRL per le voci non adiacenti), a seconda dello stile definito.
Anche le proprietà del controllo List Box possono essere impostate da programma in risposta, ad esempio, ad eventi, secondo la sintassi generale: **lstBox.nomeproprietà**, dove lstBox è il nome della List Box.
Nella tabella seguente vengono elencati i metodi associabili ad una List Box

Metodo: sintassi	**Descrizione**	**Esempio**	**Commento**
AddItem: lstBox.AddItem voce, indice	Inserisce un nuovo elemento nell'elenco di voci	MioElenco.AddItem "pippo", 0	Aggiunge la voce pippo come primo elemento dell'elenco
Clear: lstBox.Clear	Svuota l'elenco	LstNames.Clear	Cancella tutte le voci dall'elenco della list box Names
RemoveItem: lstBox.RemoveItem indice	Rimuove dall'elenco di voci l'elemento specificato dall'indice	MioElenco.Remove 0	Rimuove da MioElenco la prima voce

Nella successiva tabella vengono invece elencati gli eventi di risposta del controllo List Box più frequenti.

Evento	**Descrizione**
Click	Gestisce il click all'interno dell'elenco di voci
DblClick	Gestisce il doppio cli all'interno di un elenco di voci (corrisponde alla selezione di una voce)
Scroll	Gestisce lo scorrimento del contenuto dell'elenco di voci.

Option Button

L'Option Button (Pulsante Opzione, detto anche Radio Button) è un piccolo cerchietto che può apparire selezionato (annerito) o meno corrispondentemente ai valori si/vero, no/falso. Più Option Button non possono essere selezionati contemporaneamente ma sono in mutua esclusione. Le proprietà rilevanti del controllo sono Value (sintassi **OptionControl.Value** [=valore booleano - cioè vero o falso] e Caption (sintassi **OptionControl.Caption** [=Stringa]).

Scroll Bar

In Visual Basic è possibile anche creare Vertical Scroll Bar o Horizontal Scroll Bar, ovvero barre di scorrimento verticali o orizzontali. I controlli permettono all'utente di visualizzare e selezionare rapidamente un valore contenuto in ampi intervalli. Le proprietà dei controlli permettono molti settings e personalizzazioni. Si consiglia di provare le varie opzioni. (sintassi generale **scrBar.nomeproprietà**, dove ScrBar è il nome della Scroll Bar)

Nella tabella riportata di seguito, vengono elencati i principali eventi associabili alle barre di scorrimento:

Evento	**Descrizione**
Scroll	Gestisce lo scorrimento della barra
Change	Gestisce la modifica del valore della barra di scorrimento

Esempi: Private Sub miascroll_Scroll( )
Private Sub miascroll_Change( )

Label

Una Label (etichetta) è un testo statico, cioè non editabile direttamente dall'utente. Ovviamente anche in questo caso da programma possono essere modificate le impostazioni di una Label, utilizzando la sintassi generica **oggetto.nomeproprietà** dove oggetto sarà uguale al nome assegnato all'oggetto Label.

Text Box

Un Text Box (casella di testo) permette all'utente di immettere in un apposito spazio il testo desiderato. Anche in questo caso è possibile impostare da programma le proprietà del controllo secondo la sintassi generica: **oggetto.nomeproprietà**, dove oggetto è il nome della casella di testo.

CONTROLLI AVANZATI

Vediamo adesso i controlli avanzati di Visual Basic, cioè quelli che permettono qualsiasi operazione. Questi controlli presenti nella versione Professional o Enterprise di Visual Basic 6, permettono di accedere a molte delle interfacce standard di Windows tramite le quali si possono aprire file, visualizzare immagini ed eseguire molte altre operazioni.

DirListBox

Questo controllo consente di visualizzare nell'applicazione il contenuto ed il percorso di file contenuti in cartelle (directory) in base alla loro struttura gerarchica. Anche questo controllo ha molte proprietà (tra le principali ricordiamo: path, list, listCount, listIndex) per le quali vi invitiamo a condurre esperimenti, e che possono essere impostate da codice, mediante la sintassi generale **oggetto.nomeproprietà** , dove oggetto è il nome del controllo.

Ecco un esempio di codice che mostra il contenuto di una directory:

```
Sub ShowPath( )
        LblDrive.Caption = "Il percorso corrente è " + dirList.Path
End Sub
Private Sub DirList_Change( )
        ShowPath
End Sub
Private Sub drvDrive_Change( )
        DrvDrive = drvDrive.List(drvDrive.ListIndex)
        DirList.Path = drvDrive.Drive
        ShowPath
End Sub
Private Sub Form_Load( )
        ShowPath
End Sub
```

In questo codice, la procedura ShowPath( ) visualizza, trasferendone il valore alla proprietà Caption del controllo Etichetta, il percorso attivo.

DriveListBox

Il controllo Drive List Box (Casella di Riepilogo) permette all'utente, in fase di esecuzione del programma, di selezionare un disco. (viene utilizzato per esempio per visualizzare tutte le unità disco del sistema). Le proprietà possono essere impostate dalla finestra di dialogo apposita oppure da codice mediante l'istruzione generica: **oggetto.nomeproprietà**, dove oggetto è il nome del controllo.

Viene riportato un esempio di codice per l'implementazione di questo controllo:

```
Sub ShowDrive( )
        LblDrive.Caption = "Il disco corrente è " + drvDrive.Drive
End Sub
Private Sub drvDrive_Change( )
        ShowDrive
End Sub
Private Sub Form_Load( )
        ShowDrive
End Sub
```

FileListBox

Permette di visualizzare, in fase di esecuzione, l'elenco dei file di un determinato percorso di memorizzazione. Le proprietà possono essere impostate sia dalla apposita finestra, sia da codice, mediante la sintassi generica: **oggetto.nomeproprietà**, dove oggetto è il nome assegnato al controllo.

Esempio di codice:

```
Sub ShowPath(BackSlash As String)
        LblDrive.Caption = "Il file corrente è "+ dirList.List(dirList.ListIndex) +
BackSlash + filList.Filename
End Sub
Private Sub dirList_Change( )
        FilList.Filename = dirList.Path
        ShowPath " "
End Sub
Private Sub drvDrive_Change( )
        DrvDrive.Drive = drvDrive.List(drvDrive.ListIndex)
        DirList.Path = drvDrive.Drive
        ShowPath " "
End Sub
Private Sub filList_Click( )
        ShowPath "\"
End Sub
Private Sub Form_Load( )
        ShowPath " "
End Sub
```

Image

Permette di visualizzare della grafica (oopure un'icona) all'interno di un form. Il formato dell'immagine deve essere un metafile, un enhanced metafile, jpeg oppure gif. Il controllo immagine usa un numero inferiore di risorse di sistema e viene ridisegnato più rapidamente rispetto ad un controllo Picture, anche se è in grado di gestire un numero inferiore di proprietà, eventi e metodi.

Le proprietà come al solito, secondo l'usuale sintassi, possono essere impostate sia dalla finestra apposita, sia da codice, mentre gli eventi più rilevanti sono Click, DoubleClick e quelli correlati alle operazioni eseguite con il mouse.

PictureBox

Un controllo PictureBox permette di visualizzare un'immagine grafica (bitmap, icona, metafile, enhanced metafile , jpeg o gif) ritagliando l'immagine stessa all'interno del controllo nel caso in cui le dimensioni siano superiori a quelle definite. Si può utilizzare il controllo PictureBox per controllare un gruppo di pulsanti di opzione e per visualizzare il risultato dei metodi per la visualizzazione delle immagini grafiche e del testo, gestiti dal metodo Print( ). Anche in questo caso, con le usuali modalità, è possibile impostare le proprietà dell'oggetto.

Shape

Il controllo Shape (Forma) è di tipo grafico. Il suo utilizzo permette di inserire rapidamente in un form un oggetto grafico (ellissi, cerchio, quadrato, ecc….) Gli eventi cui è sensibile questo oggetto sono Click, DbkClick e tutti quelli collegati all'uso del mouse, metre le proprietà si impostano nelle solite modalità. Nella tabella seguente vengono riportati i valori ammissibili per la proprietà Shape di un oggetto realizzato con il controllo Forma.

Costante	**Impostazione**	**Descrizione**
VbShapeRectangle	0	Crea un rettangolo (default)
VbShapeSquare	1	Crea un quadrato
VbShapeOval	2	Crea un'ellisse
VbShapeCircle	3	Crea un cerchio
VbShapeRoundedRectangle	4	Crea un rettangolo con bordi arrotondati
VbShapeRoundedSquare	5	Crea un quadrato con angoli arrotondati

Tabella dei valori ammessi per la proprietà FillStyle del controllo Forma

Costante	Impostazione	Descrizione
VbFSSolid	0	Appplica un riempimento pieno
VbFSTransparent	1	Applica un riempimento trasparente (default)
VbHorizontalLine	2	Applica un riempimento composto da linee orizzontali
VbVerticalLine	3	Applica un riempimento composto da linee verticali

VbUpwardDiagonal	4	Applica un riempimento con linee diagonali verso l'alto
VbDownwardDiagonal	5	Applica un riempimento con linee diagonali verso il basso
VbCross	6	Applica un riempimento a croci
VbDiagonalCross	7	Applica un riempimento a croci diagonali

Timer

Il controllo Timer (Temporizzatore) non è un controllo visibile, ma permette di eseguire del codice a intervalli di tempo regolari, servendosi dell'evento Timer. E' particolarmente utile per l'esecuzione di operazioni in background. La proprietà Enabled del controllo permette di attivarne o disattivarne l'operatività.
Mentre le prorpietà si impostano come per gli altri controlli, l'evento più utilizzato in questo caso è Timer.

CLASSI E TIPI DI DATI

Vedremo ora cosa vuol dire programmare ad oggetti, ovvero la creazione e l'uso delle classi e dei tipi di dati.

Tipi di dati personalizzati

In Visual Basic, così come in altri linguaggi, è possibile definire tipi di dati personalizzati, mediante l'utilizzo dell'enunciato Type. La sintassi generale per dichiarare un tipo di dato personalizzato è:

```
[Private ¦ Public] Type nomeTipo
        nomeElemento [([subscripts])] As tipo
        [nomeElemento[([subscripts])] As tipo]
        …
End Type
```

In cui:

- la parola Public (facoltativa) rende il tipo di dati disponibile a tutte le procedure di tutti i modulo del progetto;
- la parola chiave Private indica al contrario che il tipo di dati personalizzato è disponibile solo all'interno del modulo nel quale il tipo di dati è stato dichiarato;
- la voce nomeTipo corrisponde al nome assegnato al tipo di dati dal programmatore;
- la voce nomeElemento corrisponde al nome di un elemento del tipo di dati personalizzato,
- la parte subscripts specifica le dimensioni di un elemento matrice (array). Per dichiarare una matrice di dimensioni variabili si utilizzano solo le parentesi tonde senza specificare nulla all'interno di esse. L'elemento subscripts prevede la sintassi: [min To] max [,[min To] max] … (se si omette la parte min, il limite inferiore della matrice viene controllato dall'enunciato Option Base, che per default è 0);
- tipo specifica il tipo di dati dell'elemento.

Esempio:

```
Type DatiGioco
        IniziaGioco As Boolean
        NumeroIterazioni As Integer
        MaxIterazioni As Integer
        Segreto As Byte
        Indovina As Byte
        Messaggio As String
End Type
```

Esempio con matrici:

```
Type MiniFoglioCalcolo
        NumRighe As Byte
        NumCol As Byte
        Celle(50,50) As Double
        SommaCol(50) As Double
        SommaRighe(50) As Double
        SegnalaErrore As Boolean
        MessaggioErrore As String
```

End Type

Tipi enumerati personali

Con l'enunciato enum si può dichiarare un tipo di enumerazione. La sintassi generale è la seguente:

```
[Public ¦ Private] Enum nome
        nomeMembro [=espressioneCostante]
        nomeMembro [=espressioneCostante]
        ….
End Enum
```

Esempio:

```
Enum Giorni
        GiornoNonValido=0
        Lunedì
        Martedì
        Mercoledì
        Giovedì
        Venerdì
        Sabato
        Domenica
End Enum
```

Tipi di dati in Visual Basic

La tabella che segue riporta tutti i tipi di dati che è possibile utilizzare in Visual Basic.

Tipo di dati	**Dimensione**	**Intervallo memorizzazione**
Byte	1 byte	Da 0 a 255
Boolean	2 byte	True o False
Integer	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 semplice)	4 byte	Da -3.402823E38 a -1.401298E45 per la precisione dei valori negativi e da 1.401298E45 a 3.402823E38 per la precisione dei numeri positivi
Double (virgola mobile a doppia precisione)	8 byte	-4.94065645841247E-324 per i valori negativi e 4.94065645841247E-324 per i valori positivi
Currency	8 byte	Da -922.337.203.685.477,5808 a 922.337.203.685.477,5808
Decimal	14 byte	+/- 79.288.162.514.264.337.593.543.9

		50.335 senza cifra decimale e +/- 7,9228162514264337593543950335 con 28 cifre decimali dopo la virgola. Il valoe minimo deiverso da zero è pari a +/- 0,0000000000000000000000000001
Date	8 byte	Dal 1 Gennaio 100 al 31 Dicembre 9999 (ottimisti!)
Object	4 byte	Qualsiasi riferimento a oggetto
String	10 byte + lunghezza stringa	Da 0 a, circa, 2 miliardi
Variant (con numero)	16 byte	Un qualsiasi valore numerico fino all'intervallo di Double
Variant	22 byte + lunghezza stringa	Da 0 a circa 2 miliardi
Personalizzato (definito da type)	Numero richiesto dagli elementi	L'intervallo di ogni elemento è identico a quello dell'intervallo per il tipo di dati.

Dichiarare una Classe

In Visual Basic è possibile dichiarare una classe che specifichi gli elementi di dati ed i relativi metodi applicabili. I progetti Visual Basic memorizzano le classi in file separati con l'estensione .cls. Per aggiungere un file classe occorre selezionare il comando *Inserisci Modulo Classe* dal menu *Progetto*.

Gli elementi dati che vengono specificati in una classe possono essere tipi di dati predefiniti, personalizzati, enumerati personali o altre classi.
La sintassi generale è:
[Private ¦ Public] elementoDati As Tipo

E' data anche la possibilità di dichiarare funzioni e procedure in modo che queste si comportino come metodi che siano in grado di manipolare elementi dati interni alla classe. Una routine può essere dichiarata Public o Private. Nel primo caso saranno accessibili ai componenti di altre classi o di altri moduli, mentre nel secondo caso potranno operare solo con altre routine della stessa classe.

Esempio di una classe che dichiara Point che prevede gli elementi dati mintXCoord di tipo Private Intefer che memorizza il valore della coordinata x e mintYCoord per la coordinata y:

```
Option Explicit
Private mintXCoord As Integer
Private mintYCoord As Integer

Public Sub MoveTo (X As Integer, Y As Integer)
        MintXCoord = X
        MintYCoord = Y
End Sub
```

```
Public Sub MoveBy (XDist As Integer, YDist As Integer)
        MintXCoord = mintXCoord + Xdist
        MintYCoord = mintYCoord + Ydist
End Sub

Function GetX( ) As Integer
        GetX = mintXCoord
End Function

Function GetY( ) As Integer
        GetY = mintYCoord
End Function

Public SubMove ToCoord (Coord As Cpoint)
        MintXCoord = Coord.GetX
        MintYCoord = Coord.GetY
End Sub
```

La classe dichiara anche la routine MoveTo( ), MoveBy( ), GetX( ), GetY( ) e MoveToCoord( ). La procedura MoveTo( ) copia i valori degli argomenti indicati negli elementi dati mintXCoord e mintYCoord. La procedura MoveBy( ) aggiunge i valori degli argomenti indicati agli elementi mintXCoord e MintYCoord. Le funzioni GetX( ) e GetY( ) restituiscono i valori degli elementi dati mintXCoord e mintYCoord. La procedura MoveToCoord( ) copia le coordinate dell'argomento Coord di tipo Cpoint (istanza della classe Cpoint) negli elementi dati mintXCoord e mintYCoord.

Gli operatori aritmetici

Per l'uso degli operatori aritmetici, fare riferimento alla tabella seguente.

Operatore	**Sintassi**	**Commenti**
+	Espressione1 + espressione2	Esegue la somma di due valori
-	Espressione1 - espressione2	Esegue la sottrazione di due valori
*	Espressione1*espressione2	Esegue la moltiplicazione di due valori
/	Espressione1/espressione2	Esegue la divisione tra due valori
\	Espressione1\espressione2	Esegue una divisione di due valori senza cifre decimali
^	Espressione1^espressione2	Eleva a potenza un numero in base al valore dell'esponente
Mod	Espressione1 Mod espressione2	Divide due valori interi e restituisce il resto

Le Classi

1. Il primo passo da fare è creare una nuova istanza della classe, ovvero adoperare la sintassi generale:
   dim istanzaClasse As New nomeClasse
2. a questo punto è necessario impostare i metodi Public della classe di appartenenza, ovvero:
   istanzaClasse.nomeMetodo elencoValoriArgomento

Esempio:
```
Private mintVal As Integer
Public Sub Store(X As integer
        MintVal = X
End Sub
Public Function Recall( ) As Integer
        Recall = mintVal
End Function
```

La classe dichiara l'elemento dati mintVal di tipo Private Integer e le routine Store( ) e Recall( ). Per dichiarare un'occorrenza della classe CmyInt si dovrà scrivere:
Dim objInt As New CmyInt
Per accedere ora all'istanza objInt si può scrivere, per esempio:
obj.Int.Store(12)
Debug.Print objInt.Recall e così via….
La classe può essere richiamata come argomento. In questo caso la sintassi generale è:
nomeArgomento As nomeClasse

Esempio:
```
Sub CopyInt(SourceInt As CmyInt, TargetInt As CmyInt)
        TergetInt.Store(SourceInt.Recall)
End Sub
```

Le costanti

Con Visual Basic è possibile dichiarare nomi cui associare valori fissi.
La sintassi generale per dichiarare una costante è:
[Private ¦ Public] Const nomeCostante [As tipo] = espressione

Esempi:
```
Const GIORNI_SETTIMANA = 7
Const MESI_ANNO As Byte = 12
Const COSTANTE_E As Double = 2.7183
Const Mois As String = "Mario Rossi"
```

Gli operatori logici

Per l'uso degli operatori logici, fare riferimento alla tabella di seguito riportata.

Operatore	Sintassi	Commento	Annotazioni
And	Booleano1 And booleano2	Esegue una congiunzione logica di due espressioni booleane (controlla se entrambe le espressioni sono vere)	
Or	Booleano1 Or booleano2	Esegue una separazione logica di due espressioni (controlla se almeno una delle due è vera)	
Xor	Booleano1 Xor booleano2	Esegue un'esclusione logica di due espressioni (controlla per verificare se solo una delle espressioni risulta vera)	
Not	Not booleano	Esegue la negazione logica di un'espressione	
Imp	Booleano1 Imp booleano2	Esegue un'implicazione logica su due espressioni….	
Eqv	Booleano1 Eqv booleano2	Equivalenza logica di due espressioni. (diverso da And, in quanto false and false risulta falsa, mentre false eqv false risulta vera).	

Gli operatori Relazionali

Gli operatori relazionali consentono di confrontare I valori di due espressioni:

> >= < <= = <>(diverso)

Is (confronta due variabili di riferimento ad oggetti. Restituisce true se la variabile fa riferimento allo stesso oggetto, altrimenti false)

Like (restituisce true se la stringa soddisfa le specifiche fornite. Es. Like “W*D.exe”)

Tipi di dati personalizzati

Per usare un tipo di dati personalizzato è necessario dichiarare le variabili di tale tipo tramite l’enunciato Dim:
Dim nomevariabile As nomeTipo
Dopo aver dichiarato un tipo di dati personalizzato, si potrà accedere agli elementi dati servendosi dell’operatore . (punto)

Es. Dichiariamo innanzitutto il tipo di dati:

```
Type Punto
  Xloc As Integer
  Yloc As Integer
End Type
```

Per dichiarare ora la variabile di tipo Punto, dovremo scrivere:
Dim miaPosizione As Punto

Nel corso del listato, per accedere agli elementi dati della variabile miaPosizione, si dovrà scrivere:
MiaPosizione.Xloc = 12
MiaPosizione.Yloc = 22

Si ha anche la possibilità di utilizzare come argomento un tipo di dati personalizzato (come un puntatore di puntatore). La sintassi generale per questa operazione è:
NomeArgomento As nomeTipo

Ecco un esempio pratico:

```
Sub CopiaPunto(PuntoOrigine As Punto, PuntoDest As Punto)
    PuntoDest.Xloc = PuntoOrigine.Xloc
    PuntoDest.Yloc = PuntoOrigine.Yloc
End Sub
```

Tipi enumerati personali

Innanzitutto occorre dichiarare la variabile del tipo:
Dim nomevariabile As nomeTipo

Il tutto funziona come per il paragrafo precedente. La sintassi generale per accedere agli elementi dati di un tipo di dati enumerato personale è la seguente:
VarTipoPersonale [= valoreEnumerazione]

Vediamo un esempio per l’uso del tipo di dati personale enumerato FineSettimana:

```
Enum FineSettimana
   Sabato = 7
   Domenica
End Enum
```

Questo tipo di dato dichiara I valori Sabato e Domenica. Ora occorre dichiarare la variabile di tipo FineSettimana:

```
Dim mioGiorno As Weekend
```

Ora, per accedere ai valori della variabile mioGiorno si potranno scrivere enunciati simili al seguente:

```
MioGiorno = Domenica
If mioGiorno = Domenica then
  Debug.Print "Domenica"
Else
  Debug.Print "Sabato"
End If
```

Anche in questo caso il tipo di dati enumerato personale può essere usato come argomento.

Le variabili

Le variabili sono "contenitori" utilizzati per memorizzare dati o riferimenti a oggetti. Anche se Visual Basic è in grado di creare variabili temporanee, è buona norma di programmazione dichiararle sempre attraverso un enunciato Dim. L'enunciato Option Explicit fa sì che le variabili debbano essere dichiarate e tale enunciato dovrà essere inserito nel modulo generale di dichiarazione del progetto e dovrebbe, inoltre, apparire come uno dei primi enunciati di un programma.
L'enunciato Option Explicit indica al compilatore di emettere un messaggio di errore quando rileva la presenza di un testo che non è parte del linguaggio Visual Basic e quando tale testo non è stato precedentemente dichiarato. Option Explicit aiuta ad evitare di commettere errori di digitazione consentendo quindi di correggere il codice.

La sintassi generale per dichiarare le varibili è:
Dim [withEvents] nomeVar [([subscripts]] [As [New] tipo], …

La parola chiave facoltativa WithEvents specifica che nomeVar è una variabile oggetto utilizzato per rispondere ad eventi gestiti da un oggetto ActiveX; essa è valida solo nei moduli di classe.(con essa però non si possono creare matrici). Inoltre non si può utilizzare la parola chiave New in congiunzione con la parola chiave WithEvents.
L'argomento facoltativo subscripts specifica le dimensioni di una variabile matrice e Visual Basic, in una variabile matrice multidimensionale, consente di dichiarare fino a 60 dimensioni. L'argomento subscript ha la sintassi generale:
[min To] max [, [min To] max] …

La parola chiave New abilita la creazione implicita di un oggetto e l'argomento facoltativo tipo specifica il tipo di dati della variabile (Byte, Boolean, Integer, Long, Currency, Single, Double, String (stringa di lunghezza variabile), string * lunghezza (per una stringa di lunghezza fissa), Object, Variant o un tipo di dati personalizzato o un oggetto. E' necessario utilizzare As per ogni variabileche verrà dichiarata.

Funzioni e Procedure

Exit Function

Questo enunciato consente di uscire immediatamente dalla procedura function corrispondente. L'esecuzione del programma riprende con l'enunciato immediatamente successivo a quello che ha richiamato la procedura Function.
La sintassi generale dell'enunciato è:

Exit Function

L'enunciato prevede du tipi fondamentali di utilizzo:

- Presenza in una procedura Function prima di un eneuncato per la gestione degli errori della procedura stessa. Questo tipo di utilizzo evita che il programma esegua enunciati di gestione degli errori quando non se ne verifica alcuno.
- Inserimento all'interno di un enunciato If che esamina le condizioni per determinare se è necessario o meno uscire dalla procedura Function.

Segue un esempio del primo caso:

```
Function DivideInts (X As Integer, Y As Integer) As Integer
  Dim Z As Integer
  On Error GoTo ZeroDivide     'attiva il gestore di errori
  Z = X / Y
  On Error GoTo 0    'disattiva il gestore di errore
  DivideInts = Z
'esce dalla funzione
ZeroDivide:
  Y = 1
  Resume
End Function
```

L'esempio definisce la funzione DivideInts( ) che restituisce il risultato della divisione dei valori delle variabili X e Y di tipo intero. La funzione attiva un gestore di errore prima di dividere questi argomenti e quindi lo disattiva dopo aver effettuato la divisione. La funzione imposta il valore restituito come il risultato della divisione e richiama quindi l'enunciato Exit Function per uscire dalla funzione. Gli enunciati che seguono Exit Funtion sono dei gestori di errore che iniziano immediatamente dopo l'etichetta ZeroDivide.
Senza l'enunciato Exit Function, quando si richiama la funzione DivideInts( ) si avrebbe un errore di esecuzione.

Segue ora un esempio del secondo caso:

```
Function DivideNums (X As Integer, Y As Integer) As Integer
  Dim Z As Integer
DivideNums 0 0
  If y = 0 then Exit Function
  Z = X / Y
  DivideNums = Z
End Function
```

Questo esempio utilizza la funzione DivideNums( ) che restituisce il risultato della divisione dei suoi argomenti X e Y di tipo intero. La funzione assegna il valore restituito di default 0 e quindi utilizza un enunciato If per determinare se il valore dell'argomento Y è pari a 0. Quando questa condizione risulta vera, il programma esegue l'enunciato Exit Function per uscire dalla funzione. In caso contario la funzione divide gli argomenti, ne memorizza il risultato nella variabile Z e quindi utilizza il valore di tale variabile come valore restituito.

Exit Sub

L'enunciato permette di uscire immediatamente dalla procedura Sub corrispondente. L'esecuzione del programma riprende con l'enunciato che viene immediatamente dopo quello che ha richiamato la procedura Sub.
La sintassi generale è:

```
Exit Sub
```

L'Exit Sub si usa in generale in due casi:

- L'enunciato Exit Sub compare in una procedura Sub prima du un enunciato per la gestione degli errori contenuto nella procedura stessa. Questo tipo di utilizzo evita che il programma esegua gli enunciati per la gestione degli errori quando non si verifica alcun errore.
- Compare in un enunciato Of che esamina una condizione per determinare se sia o meno necessario uscire da una procedura Sub.

Funzioni

Per eseguire alcune operazioni e restituire un valore, Visual Basic si serve di funzioni, come del resto tutti I linguaggi di programmazione.
La sintassi generale per la definizione di una funzione è:

```
[Public ¦ Private ¦ Friend] [Static] Function nome [(elencoargomenti)] [As tipo]
[enunciati]
[nome = espressione]
[Exit Function]
[enunciati]
[nome = espressione]
[Exit Function]
```

La dichiarazione di una funzione semplice inizia con la parola chiave Function e deve concludersi con un enunciato che contenga comunque le parole End Function. La tabella che segue riporta schematicamente le parti facoltative che possono apparire prima della parola chiave Function.

Parte	**Descrizione**
Public	Indica che la funzione sarà accessibile a tutte le altre procedure di tutti I moduli. Se Public si utilizza in un modulo contenente Option Private, la funzione non sarà disponibile dall'esterno del progetto.
Private	Indica che la funzione sarà accessibile solo alle altre procedure del modulo nella quale la funzione è dichiarata.
Friend	Utilizzato solo in un modulo classe. Indica che la funzione sarà visibile dall'interno del progetto ma non dal controllore di un'istanza di un oggetto.
Static	Indica che le variabili locali della funzione vengono conservate fra una chiamata e l'altra. L'attributo Static non influisce sulle variabili che vengono dichiarate al di fuori della funzione, anche nel caso in cui queste vengano utilizzate all'interno della funzione stessa.

La voce nome corrisponde al nome della funzione e deve attenersi alle convenzioni standard per l'assegnazione dei nomi in Visual Basic.
Elencoargomenti è facoltativo (sebbene spesso utilizzato) e rappresenta un elenco di argomenti tra di loro separati da virgole e che viene spesso passato alla funzione quando questa verrà richiamata. L'argomento elencoargomenti prevede la sintassi:

[Optional] [ByVal ¦ ByRef] [ParamArray] nomeVar[( )] [As tipo] [= valoreDefault]

Il termine argomento viene spesso indicato in altri linguaggi con il termine parametro.
La seguente tabella riporta le parti per l'indicazione di un argomento.

Parte	**Descrizione**
Optional	Indica che un argomento non viene espressamentte richiesto. Se utilizzato, tutti gli argomenti successivi contenuti in elencoargomenti devono essere considerati facoltativi e dichiaati utilizzando la parola chiave Optional. La parola chiave Optional non deve essere utilizzata per alcun argomento nel caso in cui venga utilizzato anche ParamArray.
ByVal	Indica che l'argomento viene passato in base al proprio valore.
ByRef	Indica che l'argomento viene passato in base al proprio riferimento (default).
ParamArray	Utilizzato solo come ultimo argomento di elencoargomenti per indicare che l'argomento finale è una matrice di tipo Optional di elementi Variant. La parola chiave ParamArray consente di fornire un numero arbitrario di argomenti e non deve essere usato in presenza di ByVal ByRef o Optional.
NomeVar	Corrisponde al nome della variabile che rappresenta l'argomento e deve seguire le convenzioni standard per l'assegnazione dei nomi in Visual Basic.
Tipo	Definisce il tipo di dati dell'argomento passato alla procedura; può essere Byte, Boolean…ecc…
ValoreDefault	Una qualsiasi costante o espressione costante. Questo elemento è valido per I parametri Optional e se il tipo è Object, sarà possibile solo assegnare il valore di default Nothing.

Una funzione non può restituire una matrice ma può restituire un valore Variant contenente una matrice. Per restituire un valore, gli enunciati della funzione devono contenere una istruzione che assegni un valore al nome della funzione. Se non si fa ciò, vengono assegnati I valori di default 0, se la funzione è numerica, stringa vuota o di lunghezza zero (“”) se di tipo stringa. Una funzione Variant, in questo caso restituirà Empty. Una funzione che restituisce il riferimento ad un oggetto restituirà Nothing.
L’enunciato Exit Function determina l’uscita prematura dalla funzione e in una funzione possono apparire più enunciati Exit Function.

Procedure

Una procedura è un componente di un’applicazione che esegue una o più operazioni all’interno di un programma o in contesti più estesi. Si può pensare ad una procedura come ad un miniprogramma contenuto in un programma. Una procedura si richiama passando I valori necessari. La sintassi generale è:

```
[Private ¦ Public ¦ Friend] [Static] Sub nome
[(elencoargomenti)]
  [enunciati]
  [Exit Sub]
  [enunciati]
End Sub
```

La dichiarazione di una procedura semplice inizia con la parola chiave Sub e termina con le parole End Sub.
In tabella sono riportati gli elementi facoltativi e le relative descrizioni:

Parte	**Descrizione**
Public	Indica che la procedura sarà accessibile a tutte le altre procedure di tutti I moduli. Se Public si utilizza in un modulo contenente Option Private, la procedura non sarà disponibile dall’esterno del progetto.
Private	Indica che la procedura sarà accessibile solo alle altre procedure del modulo da cui la procedura è dichiarata.
Friend	Utilizzato solo in un modulo classe. Indica che la procedura sarà visibile dall’interno del progetto ma non dal controllore di un’istanza di un oggetto.
Static	Indica che le variabili locali della procedura vengono conservate fra una chiamata e l’altra. L’attributo Static non influisce sulle variabili che vengono dichiarate al di fuori della procedura, anche nel caso in cui queste vengano utilizzate all’interno della procedura stessa.

Per tutti gli altri parametri (nome, elencoargomenti) valgono le stesse regole enunciate per le funzioni. Stessa cosa per l’enunciato Exit Sub.

Le stringhe

Concatenazione

Attraverso l'uso dell'operatore "+" è possibile concatenare (collegare) due o più stringhe. La sintassi è:
stinga1 + stringa2 + ….
Con stringha intendiamo espressioni di tipo stringa cioè contententi variabili stringa, elementi di testo (quelli racchiusi tra doppi apici).
Esempio:

```
….
Dim Str1,Str2,Str3 As String
…
Str1="Buongiorno"
Str2="a tutti!"
Str3=Str1+" "+Str2
…
```

Str3 conterrà la frase "Buongiorno a tutti!"

Il tipo Variant

In Visual Basic il tipo Variant permete di eseguire conversioni automatiche fra dati stringa e dati numerici. Ciò significa che il tipo variant si comporterà come una stringa in un'espressione stringa, come numerico in una numerica, come booleano in una logica, ecc..

La funzione InStr( )

Permette di cercare una stringa all'interno di un'altra.

Sintassi:
Instr([iniziale,]stringa1, stringa2[, confronto])

INIZIALE è il numero che indica la posizione iniziale della ricerca. Se si omette l'argomento, la ricerca inizia dal primo carattere. Attenzione: se utilizzate l'argomento CONFRONTO, dovete inserire obbligatoriamente anche INIZIO.
Stringa1 è la stringa da cercare. Confronto è il tipo di confronto da effettuare: può essere omesso (in questo caso per default si avrà il valore 0, che corrisponde ad un confronto binario), oppure si può assegnare il valore 1 (confronto testuale), 2 (viene utilizzato per Access ed effettua un confronto sulla base delle inormazioni contenute nel database).

La funzione InstrRev( )

Molto simile alla precedente, effettua però la ricerca al contratio, cioè a partire dall'ultimo carattere fino ad arrivare al primo.
Sintassi:
InstrRev (stringa1, stringa2 [,inizio] [,confronto])

Per inizio e confronto valgono le stesse considerazioni fatte per Instr( ). Stringa1 è la stringa in cui effettuare la ricerca, mentre Stringa2 è la stringa da ricercare.

La funzione Left( )

Permette di estrarre il numero di caratteri indicati come parametro dalla parte sinistra di una stringa.
Sintassi:
Left(stringa, lunghezza)
Se ad esempio scrivete
Stringa=”paperino”
Left(stringa,3)
Il risultato sarà “pap”

La funzione Right( )

Permette di estrarre il numero di caratteri indicati come parametro dalla parte destra di una stringa.
Sintassi:
Right(stringa, lunghezza)
Se ad esempio scrivete
Stringa=”paperino”
Right(stringa,3)
Il risultato sarà “ino”

La funzione Mid( )

Permette di estrarre il numero di caratteri indicati come parametro dalla parte centrale di una stringa, indicando il punto da cui deve cominciare il conteggio.
Sintassi:
Mid(stringa, inizio [,lunghezza])
Se ad esempio scrivete
Stringa=”paperino”
Right(stringa,3,3)
Il risultato sarà “ngh”

La funzione Lcase( )

Con questa funzione potete convertire i caratteri di una stringa da maiuscolo a minuscolo.
Sintassi:
Lcase(stringa)

Le funzioni LTrim( ), RTrim( ), Trim( )

Consentono di eliminare da una stringa gli spazi bianchi in eccesso.
Sintassi:
Ltrim(stringa) rimuove gli spazi bianchi in eccesso a sinistra della stringa
Rtrim(stringa) rimuove i caratteri bianchi in eccesso a destra della stinga

Trim(stringa) rimuove i caratteri bianchi in eccesso sia sulla destra della stringa sia sulla sinistra (è una combinazione degli effetti di Ltrim( ) e Rtrim( )

La funzione Replace( )

Con questa funzione sostituite in una stringa, la stringa da sostituire con una nuova.
Sintassi:
Replace(stringa, ricerca, sostituzione [, inizio [, numero [, confronto]]])
Nella sintassi:
stringa – è la stringa in cui effettuare la sostituzione
ricerca – sono i caratteri da cercare e sostituire
sostituzione – sono i caratteri con cui sostituire quelli ricercati
inizio, se specificato, indica la posizione del caratterre da cui effettuare la ricerca (se non è indicato, si comincia dal primo)
numero, se indicato, indica il massimo numero di sostituzioni da effettuare. Se non è specificato la sostituzione viene effettuata su tutto il testo

La funzione StrReverse( )

Inverte tutti i caratteri di una stringa.
Sintassi:
StrReverse(stringa)
Esempio:
Stringa=”paperino”
StrReverse(stringa)
Otterrete il risultato “onirepap”

La funzione Ucase( )

E’ il contrario di Lcase( ) e dunque trasforma i caratteri di una stringa da minuscoli a maiuscoli.
Sintassi:
Ucase(stringa)

Gestione degli errori

Err

Err è l'oggetto che contiene tutte le infomazioni relative ad errori che si verificano quando si manda in esecuzione un'applicazione scritta in Visual Basic. E' dunque conveniente imparare ad utilizzare Err per poter monitorare ciò che accade all'interno dell'applicazione sviluppata. Attraverso il metodo Raise( ) è possibile gestire le informazioni contenute nell'oggetto Err. La sintassi del metodo Raise ( ) è:
Err.Raise numero, origine, descrizione, fileGuida, contestoGuida
Gli argomenti che potete assegnare al metodo Raise( ) sono le proprietà stesse di Err, che potete trovare nella descrizione dell'oggetto nella guida on line di Visual Basic.

On Error

E' un enunciato che consente di gestire gli errori di esecuzione.
La sintassi è:
On Error GoTo [etichetta | linea_di_codice]

Quando si verifica un errore, questo enunciato permette di passare all'esecuzione del codice a partire dalla posizione contrassegnata da un'etichetta o a partire da un numero di linea di codice indicato. Sia l'etichetta che il numero di linea devono essere nella stessa procedura che contiene On Error, altrimenti nel compilare il codice, vi verrà notificato il messaggio di errore: "On Error Resume Next", che in pratica vi comunica che l'esecuzione del codice passerà in caso di errore alla riga successiva a quella in cui si è verificato l'errore stesso.
Se invece scrivete "On Error GoTo 0" state comunicando al codice di disabilitare qualsiasi gestione di errore nella procedura.
In generale è preferibile porre gli enunciati di gestiode degli errori alla fine di ogni procedura e si consigli di inserire un "Exit Sub" oppure "Exit Function" o "Exit Property".

If-Then-Else
Permette di effettuare delle scelte condizionate da valori logici o da espressioni booleane.
La sintassi è:
If condizione Then [enunciati] [Else enunciatiElse]

La sintassi, nel caso in cui si vogliano introdurre altre condizioni, è invece la seguente, in cui si utilizza ElseIf:

```
If condizione Then
        [ enunciati]
        [ ElseIf condizione Then
                [condizioniElseIf]
                …
        [Else
                condizioniElse]]
End If
```

Chiaramente, indentando il codice, la lettura delle condizioni è più semplice e pulita.

Resume

Resume comunica all'applicazione come recuperare l'esecuzione al termine di una routine per la gestione degli errori.
La sintassi è:
Resume [0]
Rsume fa in modo che l'esecuzione venga rispresa dall'enunciato che ha causato l'errore. L'argomento 0, facoltatico, è stato introdotto solo per compatibilità con GWBasic e Basic.
Resume Next fa invece in modo che l'esecuzione riprenda dall'enunciato immediatamente successivo a quello che ha generato l'errore.
Resume Numero_di_Linea fa invece in modo che l'esecuzione riprenda dal numero di linea di codice indicata.

Select Case

Serve per prendere una decisione condizionata, scegliendo tra più opzioni. La sintassi è:

```
Select Case Espressione_da_verificare
        [Case elenco_di_espressioni
                [enunciati]]
                …
        [Case Else
                [enunciati_CaseElse]]
End Select
```

Espressione_da_verificare è un'espressione numerica o stringa.
Elenco_di_espressioni è un elenco di valori, separati da virgole, che vengono indicati dopo la clausola "Case". Quando uno di questi valori è verificato, viene eseguito un'enunciato riportato di seguito. Lo stesso vale per Case Else e gli enunciati che seguono.
Esempio:

```
…
Select Case giorno
        Case is "lunedì"
        Debug.Print "il primo giorno della settimana"
        Case is "martedì"
        Debug.Print "il secondo giorno della settimana"
Case Else
        Debug.Print "non è il primo giorno della settimana"
End Select
```

I cicli

I cicli permettono di eseguire, sotto determinate condizioni, più istruzioni in sequenza per un numero di volte predefinito o controllato dalla verifica di un'espressione. Vediamo ora come si utilizzano i cicli.

Do Until

Avvia l'esecuzione di una serie di istruzioni che si ferma quando risulta vera la condizione posta all'inizio del ciclo. Il cislo si ferma non appena la condizione risulta vera.
Sintassi:

```
Do Until condizione
        Istruzioni
Loop
```

Do While

Avvia l'esecuzione di una serie di istruzioni che continua finché è vera la condizione posta all'inizio del ciclo. Il ciclo si arresta, dnque, non appena la condizione risulta falsa.
Sintassi:

```
Do While condizione
        Istruzioni
Loop
```

Do – Loop Until

Le istruzioni vengono ripetute fino a quando la condizione posta alla fine del ciclo risulta vera.
Sintassi:

```
Do
        Istruzioni
Loop Until condizione
```

Do – Loop While

Esegue le istruzioni mentre la condizione posta alla fine del ciclo è vera.
Sintassi:

```
Do
        Istruzioni
Loop While condizione
```

Exit Do

Si usa quando si vuole uscire da uno dei cicli descritti precedentemente senza completare l'esecuzione, ad esempio perché si è verificata una condizione (di errore, oppure condizionata da un IF) in seguito alla quale si desidera uscire dal ciclo.
La sintassi è semplicemente: Exit Do

For

E' un ciclo che esegue le istruzioni per un numero finito di volte. Si inizializza una variabile e la si usa come contatore del ciclo. Si può assegnare un passo alla variabile, cioè, per esempio, farla scandire di due in due.
Sintassi:

```
For contatore =inizio To fine [Step passo]
        [istruzioni]
        [Exit For
        [istruzioni]
Next [contatore]
```

Esempio:

```
For i=1 to 20 step 2
        Debug.Print "sono arrivato al numero" & I
Next i
```

For Each

Permette di ripetere più espressioni per ogni elemento di una matrice o collezione di oggetti in generale.
Sintassi:

For Each elemento *In* gruppo
        [istruzioni]
        [*Exit For*]
        [istruzioni]
Next [elemento]

Exit For

Si usa quando si vuole uscire dal ciclo For o dal ciclo For Each, descritti precedentemente, senza completare l'esecuzione, ad esempio perché si è verificata una condizione (di errore, oppure condizionata da un IF) in seguito alla quale si desidera uscire dal ciclo.
La sintassi è semplicemente: Exit For