Domenico Valle

Appunti VB.NET 2008

L’ambiente di sviluppo

• La pagina iniziale

La pagina iniziale contiene 4 sezioni :

- Progetti recenti
- Guida introduttiva
- Articoli Visual Studio : è possibile lasciare commenti connettendosi all’area Feedback
- MSDN-Visual Basic : elenco di articoli su VB

• Menù

Finestra iniziale : File, Modifica, Visualizza, Strumenti, Finestra, Comunità, Guida
Quando si crea un’applicazione vengono aggiunti
Progetto, Genera/Compila, Debug, Dati

• Barra degli strumenti

Visualizzata la barra Standard

• Controllo lato Server : esplora server / esplora database

Permette di accedere e gestire le risorse di qualsiasi computere locale e remoto compresi i database

• La casella degli strumenti

- Barra menù -> Visualizza -> casella degli strumenti
- Icona da barra degli strumenti
- Tasti CTRL+ALT+X

• Le finestre : codice, esplora soluzioni, proprietà

Proprietà

- Barra menù -> Visualizza -> Finestra proprietà
- Icona da barra degli strumenti
- F4
- Tasto dx su oggetto e selezionare dal menù a tendina la voce Proprietà

• Creare un nuovo progetto

- Barra menù -> file -> nuovo progetto
- icona da barra degli strumenti

Una soluzione può contenere diversi progetti, scritti con linguaggi diversi e diverse tipologie di file; è un’applicazione.
La form

- selezionando la form dalla finestra *Esplora Soluzioni* è possibile cambiare le proprietà del file , mentre selezionando la form dalla finestra *finestra di progettazione* è possibile cambiare le proprità visive e quelle logiche

Posizionare un controllo nella form

- cliccare sul controllo nella casella degli strumenti
- spostare il puntatore sulla form : il puntatore assume la forma del mirino con accanto l'icona del controllo selezionato
- tenere premuto il puntatore del mouse nel punto in cui si desidera collocare l'angolo superiore sinistro e trascinare il puntatore nel punto in cui si desidera collocare l'angolo inferiore dx

in alternativa

- doppio click sul controllo nella barra degli strumenti
- il controllo viene disegnato automaticamente sulla form: trascinarlo nella posizione voluta

ridimensionare il controllo : agire sui quadratini bianchi disegnati sui lati e negli angoli del controllo

• Convenzioni

nome del controllo : è opportuno far precedere il nome da un prefisso che ne indica il tipo :

Button	btn	bottone
CheckBo	chk	casella di controllo
ComboBox	cbo	casella combinata
Form	frm	form
Horizontal scroll bar	hsb	barra di scorrimento orizzontale
Label	lbl	etichetta
ListBox	lst	casella di riepilogo
MainMenu	mnu	menù
PictureBox	pic	riquadro immagine
RadioButton	rdb	pulsante di opzione
TextBox	txt	casella di testo
Vertical scroll bar	vsb	barra di scorrimento verticale

Simboli identificativi di alcuni componenti di un progetto:

parallelepipedo viola : metodo
quadratino blu : membro
4 libri impilati : libreria
3 quadrati raggruppati :classe
chiave: protetto
lucchetto: privato

Il Framework.NET

• Il Framework.NET

Il Framework.Net è una piattaforma per lo sviluppo di applicazioni Windows, Web o per dispositivi mobili utilizzando uno o più linguaggi di programmazione (Visual Basic, C#,J# sono i linguaggi più comunemente utilizzati).
Il Framework.Net è in realtà una serie di classi denominate **classi base**; esse sono indipendenti dal linguaggio utilizzato.

Un programma che deve essere eseguito in ambiente .NET non viene compilato e tradotto in linguaggio macchina, come avveniva ad esempio per i programmi scritti in VB6, bensì viene compilato e tradotto in un linguaggio denominato **MSIL** (**Microsoft Intermediate Language**), indipendente dal processore(il linguaggio MSIL è l'analogo del byte code di Java); al momento dell'esecuzione il codice MSIL viene tradotto nel codice macchina del processore utilizzato: questa traduzione viene fatta dal componente di .NET che si chiama **CLR** (**Common Language Runtime**).
Il **CLR** permette di:

- **caricare ed eseguire** il codice: il programma, già tradotto in **MSIL**, viene caricato, tradotto in liguaggio macchina, ed eseguito. Il codice eseguito sotto il controllo di **CLR** viene detto codice **gestito**;
- **isolare** il programma dalle altre applicazioni: l'arresto del programma isolato non provoca l'arresto delle altre applicazioni;
- **evitare che l'applicazione abbia accesso a tutte le risorse** del computer: le applicazioni Windows hanno accesso a tutte le risorse del computer; le applicazioni .NET per poter accedere a delle risorse devono rispondere a dei criteri impostati dal programmatore o dall'amministratore ad es. la provenienza : da un computer locale, dalla rete aziendale, da Internet
- **gestire** in modo efficiente **le eccezioni**
- avere una buona **interoperabilità** : ad es. tutti i tipi di dati sono condivisi dai vari linguaggi; le applicazioni .NET possono comunicare con i componenti **COM** (**Component Object Model**). Alla base dell'interoperabilità ci sono i componenti **Common Type System** e **Common Language Specification.**
- **gestire la memoria**: il CLR crea gli oggetti e tramite il **GC** (**Garbage Collector**) rimuove l'oggetto e rende dosponibile la memoria quando l'oggetto non è più utilizzato cioè non vi sono più riferimenti espliciti all'oggetto. Il GC è in grado anche di deframmentare e compattare la memoria.

• Namespace

Il Framework.NET è un'ampia raccolta di classi (più di 3500) ; i namespace ragguppano classi simili; un namespce può contenere a sua volta altri namespace.
La maggioer parte delle classi si trova nel namespace System o nei namespace, contenuti a loro volta in System

- System.Data: classi per l'accesso ai database
- System.Xml: classi per la lettura/scrittura di file Xml
- System.Windows.Forms: classi per la gestione delle Form
- System.Net: classi per la comunicazione in rete

Oltre al namespace System un altro namespace che contiene diverse classi utili è il namespace My

- My.Application: classe contenente informazioni relative all'applicazione in esecuzione
- My.Computer: classe contenente informazioni relative al computer utilizzato
- My.Forms: classe contenente informazioni sui Forms dell'applicazione
- My.Resources: classe relativa ai file di risorsa dell'applicazione (se presenti)
- My.Settings: classe relativa al file di configurazione dell'applicazione (se presente)
- My.User: classe relativ all'utente corrente

Tutte le classi devono trovarsi in un namespace, anche quelle del nostro progetto; quale è tale namespace? dal menù Progetto -> Proprietà -> scheda Applicazione -> Spazio dei nomi radice (Root namespace)

• Istruzione Imports

Deve essere scritta prima di qualsiasi istruzione, comprese le dichiarazioni Class o Module, nel file del codice in cui si vuole utilizzare; serve per "importare" il namespace: permette di richiamare una classe specificandone solo il nome senza indicare tutto il percorso, ad es.:

```
Dim reader As System.IO.StreamReader = System.IO.File.OpenText(nomeFile)
```

Si può riscrivere :

```
Imports System.IO   '—in testa
..
..
..
Dim reader As StreamReader = File.OpenText(nomeFile)
```

Le variabili e i tipi di dati

• Le variabili

Una variabile rappresenta un'area di memoria, dove viene memorizzato un valore
Esempio di dichiarazione di variabile:

```
Dim a As Double
```

Variabili locali : sono dichiarate tramite l'istruzione ***Dim***

- variabili di blocco : all'interno di un blocco di codice come ***if then … else … endif*** o ***for…next;*** area di visibilità (ambito) : blocco - durata (lifetime) : procedura
- variabili di procedura: area di visibilità (ambito) e durata (lifetime) : procedura; se al posto di ***Dim*** si usa l'istruzione ***Static*** la durata è pari alla durata del modulo(*) che contiene la procedura
- variabili a livello di modulo(*): area di visibilità (ambito) e durata (lifetime) : modulo(*); in questo caso al posto di ***Dim*** è preferibile utilizzare l'istruzione ***Private***

Le variabili globali sono dichiarate come ***Public*** in un modulo(*): area di visibilità (ambito) e durata (lifetime) : tutta l'applicazione.
(*) per modulo si intende un modulo generico, un modulo WindowsForm,un modulo di classe

• Le costanti

I valori costanti, una volta definiti non possono essere variati.
Esempio di dichiarazione di costante :

```
Dim Const PI As Double = 3.142
```

• Convenzione sui nomi di variabili, costanti e metodi

VB è Case-Insensitive.
I nomi delle costanti sono in maiuscolo.
Per le variabili e i metodi Microsoft suggerisce :
Camel Notation per le variabili: ogni parola, dopo la prima, inizia con lettera maiuscola
Pascal Notation per le classi e i metodi: ogni parola inizia con lettera maiuscola

Esempio di **Camel Notation**

salary totalSalary
myMathsMarks isPaid

Esempio di **Pascal Notation**

GetTotal()
Start()
WriteLine()
LastIndexOf()

notazione Ungherese: per indicare i controlli ad es. txtNome, lstProdotti

non è raccomandato l'uso dell' '_' all'interno dei nomi di variabili/metodi.

- **Tipi di dati**

Tipi di dati	N.Byte	Descrizione	Dati rappresentati
Boolean	1	Memorizza valori dell'Algebra di Boole	**True o False**
Short	2	E' il vecchio integer di Visual Basic 6.0	numeri interi compresi tra –32768 e 32767.
Integer	4		numeri interi compresi tra –2.147.483.648 to +2.147.483.647
Long	8		numeri interi compresi tra - 9.223.372.036.854.775.808 a +9.223.372.036.854.775.807
Single	4		Range da ±1.5E-45 a ±3.4 E+38 con precisione di 7 cifre
Double	8		Range da ±5.0E-324 a ±1.7E+308 con con precisione di 16 cifre
Decimal	16		Range da +/-1.0 E-28 a +/-7,9228E+28. Supporta fino a 29 cifre significative. È particolarmente adatto per i calcoli, ad esempio finanziari, che richiedono un numero elevato di cifre ma non sono in grado di tollerare errori di arrotondamento.
Char	2		Un carattere Unicode
Object	4/8	Il tipo di dati Object sostituisce il tipo Variant. In VB.Net non tipizzando una variabile e/o costante, la variabile non sarà più di tipo Variant, come in VB6, ma di tipo Object	Una variabile dichiarata in questo modo potrà memorizzare qualunque tipo di informazioni (c.d. tipo di dati universale) ed occuperà in memoria 4 oppure 8 byte, a seconda della piattaforma dove sta (piattaforma a 32 oppure a 64 bit)
Date	8	Informazioni temporali, ossia date e ore	Saranno rappresentate tutte le date partendo dalle ore 0.00.00 (mezzanotte) del 1° gennaio 0001 alle 23.59.59 del 31 Dicembre 9999
String	Variabile		insieme di caratteri alfanumerici (ossia numeri e lettere). Visual Basic.Net, a differenza del linguaggio di programmazione VB6 gestisce con tale tipo di dato stringhe a lunghezza variabile da 0 a circa 2 Miliardi di caratteri Unicode

Raccordo tra i tipi di dati VB.NET e i tipi di dati del Framework .Net.

Tipi di dati VB.Net	**Tipi di dati del Framework .Net**
Boolean	Boolean
Short	Int16
Decimal	Decimal
Double	Double
Integer	Int32
Long	Int64
Single	Single
Object	Object(classe)
String	String(classe)
Byte	Byte
Char	Char
Date	DateTime

Vi sono due "cattive" caratteristiche in VB.Net derivate dalle vecchie versioni VB5 e VB6:

- si può dichiarare una variabile senza specificare il tipo: in tale case si assume il tipo System.Object (in VB5/VB6 tipo Variant)
- si può fare una conversione tra valori (oggetti) di tipo incompatibile ad es. una stringa in un intero

Perché tali possibilità sono "cattive" ? perché possono dar luogo a dei bug. Si può ovviare a tale inconveniente tramite
Option Explicit : tutte le variabili devono essere dichiarate
Option Strict: non permette la conversione tra tipi diversi

È possibile definire queste opzioni in due modi

- da progetto
- in un file (form o modulo), con validità solo nel modulo in cui è definita, prima di qualsiasi istruzione

Option Strict presuppone ***Option explicit***

• Gli operatori : aritmetici, logici, di confronto

Operatori aritmetici

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione(1) tra variabili reali
\	Divisione tra interi con risultato intero (quoziente intero)
MOD	Resto della divisione tra interi
^	Elevamento a potenza

Esempi:
A = A + 1 oppure: **A += 1**
X = X - 3 oppure: **X -= 3**

(1) Il risultato dell' operazione di divisione dipende dal tipo della variabile che lo contiene: se A è una variabile di tipo intero,l' istruzione A=14/3 assegna ad A il valore 5 (cioè l' intero più vicino al risultato esatto, senza resto); se A è di tipo reale, il valore assegnato sarà 4,666.

Operatori logici

And	prodotto logico
Or	somma logica
Not	negazione logica
XOR	OR esclusivo

Operatori di confronto

=	Operatore per effettuare il **test di ugualianza**
<>	Diverso
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale

Esistono delle funzioni che restituiscono informazioni sul tipo di dati della variabile e sono utili a controllare gli input dell utente:
IsNumeric (parametro): restituisce True se il parametro fornito è un numero
Esempio:

```
Dim eta As Byte
eta= InputBox ( Inserisci la tua età )
While Not IsNumeric(eta)
eta= InputBox ( Inserisci la tua età )
Loop
```

• Istruzione su più linee

Per continuare un'istruzione su più linee utilizzare l'underscore '_'
errore

```
Dim myName As String = "Il mio nome è
Rossi Mario"
```

Corretto

```
Dim myName As String = " Il mio nome è " & _
" Rossi Mario e " & _
"mi piace .NET ..."
```

- ## Statement & expression

Esempi di **statement (istruzioni) validi**:
Dichiarazione: `Dim x As Integer`
Assegnazione: `x = 10`
Chiamata di sub/function/method: `MessageBox.Show ( Hello world! )`
Nuovi oggetti: ……..
Se si scrive
`x + 5`
non succede nulla, perché `x+5` è un' espressione (**expression**) non uno statement !
Le expression sono usate per scrivere le condizioni da testare in un' istruzione di selezione (***If***) o di ciclo (***Do .. Loop***), infatti vengono valutate (***evaluate***) e il risultato sarà True o False.
Quindi vale quanto segue:

Statement can be executed .... expression can be evaluated !!!

- ## I commenti

Il commento si fa precedere dall apice (')
Le frasi poste dopo un apice (') servono per spiegare cosa viene scritto nel codice.
Esempio:
`'r è la variabile che contiene il valore del raggio`
VB offre la possibilità chiamata XML Document Comment per creare dei blocchi di commento per i metodi; posizionarsi sulla linea bianca che precede un metodo e digitare 3 apici consecutivi verranno inserite delle righe di commento, da completarsi a cura del programmatore, con riportati i nomi dei paramtri e l'indicazione dell'esistenza del valore di ritorno. Esempio :

```
''' <summary>
'''
''' </summary>
''' <param name="a"></param>
''' <param name="b"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function esempio(ByVal a As Integer, ByVal b As Integer) As String
    Dim x As String
    x = ""
    Return x
End Function
```

Le stringhe

Sequenza di caratteri.
Sono tipi reference anche se apparentemente si comportano come tipi value.
Sono immutabili e quindi ogni cambiamento a una stringa in effetti ne crea una nuova abbandonando la vecchia; esempio :

Dim s as String
s = "pippo"
s = s+" pluto" viene abbandonata la stringa "pippo" , pronta per essere cancellata dal **Garbage Collector** (**GC**), e resta in piedi la stringa "pippo pluto".

Operatori sulle stringhe

Addizione	+ o **&**	concatena 2 stringhe e ne crea una nuova
Eguaglianza	=	confronta 2 stringhe e restituisce True se sono uguali False se sono diverse
Diverso	<>	confronta 2 stringhe e restituisce True se sono diverse False se sono uguali
Assegnazione	=	Copia il contenuto di una stringa in una nuova stringa

Classe String (*'questa stringa'* sta ad indicare *stringa*.metodo(…..))

Proprietà o metodo	Descrizione
Length	Numero di caratteri di *'questa stringa'*
CompareTo(s As String)	Confronta *'questa stringa'* con la stringa *s* restituendo 0 se le stringhe sono uguali, < 0 se *'questa stringa'* è minore di *s* , > 0 se 'questa stringa' è maggiore di *s*
Compare(s1 As String, s2 As String)	(**shared**) Confronta *s1* con *s2*; restituisce un intero come CompareTo
Equals(s As String)	Restituisce TRUE se *'questa stringa'* è esattamente uguale a *s*, altrimenti restituisce FALSE
Concat(s As String)	Restituisce una nuova stringa che è la concatenazione di *'questa stringa'* con *s*
Insert(index As Integer, s As String)	Restituisce una nuova stringa risultante dall'inserimento della stringa *s* in *'questa stringa'* a partire dalla posizione index (base 0)
Copy(s As String)	(**shared**) Restituisce una nuova stringa copia di *s*
StartsWith(s As String)	Restituisce vero se *'questa stringa'* inizia con la stringa *s*
EndsWith(s As String)	Restituisce vero se *'questa stringa'* finisce con la stringa *s*
IndexOf(s As String) IndexOf(ch As Char)	Restituisce la posizione (base 0) della prima occorrenza della stringa *s* o del carattere *ch* (equivalente all' istruzione **Instr**)
LastIndexOf(s As String) LastIndexOf(ch As Char)	Restituisce la posizione (base 0) dell'ultima occorrenza della stringa *s* o del carattere *ch*
Replace(char,char) Replace(string, string)	Restituisce una nuova stringa sostituendo tutte le occorrenze del primo *char* con il secondo *char* o della prima *string* con la seconda *string*
Split(params As Char())	Restituisce un array costituito dalle sottostringhe di *'questa stringa'* delimitate da uno più caratteri specificati nell'array *Char()*
Substring(i1 As Integer) Substring(i2As Integer,i3 As Integer)	Restituisce una sottostringa di *'questa stringa'* a partire da *i1* (base 0) fino alla fine oppure a partire da *i2* per *i3* caratteri (equivalente all' istruzione **Mid**)
ToCharArray()	Restituisce un array dei caratteri di *'questa stringa'*
ToUpper()	Restituisce un copia di *'questa stringa'* tutta maiuscolo

ToLower()	Restituisce un copia di *'questa stringa'* tutta minuscolo
Trim()	Restituisce una nuova stringa dopo aver eliminato da *'questa stringa'*, in testa e in coda, i caratteri indicati (default spazio)
TrimStart()	Restituisce una nuova stringa dopo aver eliminato da *'questa stringa'*, in testa, i caratteri indicati (default spazio)
TrimEnd()	Restituisce una nuova stringa dopo aver eliminato da *'questa stringa'*, in coda, i caratteri indicati (default spazio)

Esempio

```
Sub Main()
        Dim s1 As String = "faraz"
        Dim s2 As String = "fraz"
        Dim s3 As String = "Faraz"
        Dim s4 As String = "VB.Net is a great programming language!"
        Dim s5 As String = " This is the target text "
        Console.WriteLine("Length of {0} is {1}", s1, s1.Length)
        Console.WriteLine("Comparision result for {0} with {1} is {2}", s1, s2, s1.CompareTo(s2))
        Console.WriteLine("Equality checking of {0} with {1} returns {2}", s1, s3, s1.Equals(s3))
        Console.WriteLine("Equality checking of {0} with lowercase {1} ({2}) returns {3}", _
            s1, s3, s3.ToLower(), s1.Equals(s3.ToLower()))
        Console.WriteLine("The index of a in {0} is {1}", s3, s3.IndexOf("a"))
        Console.WriteLine("The last index of a in {0} is {1}", s3, s3.LastIndexOf("a"))
        Console.WriteLine("The individual words of '{0}' are", s4)
        Dim words() As String = s4.Split(" "c)
        Dim word As String
        For Each word In words
            Console.WriteLine(" {0}", word)
        Next
        Console.WriteLine(vbCrLf & "The substring of " & vbCrLf & " '{0}' " & _
        vbCrLf & "from index 3 of length 10 is " & vbCrLf & " '{1}'", _
        s4, s4.Substring(3, 10))
        Console.WriteLine(vbCrLf & "The string " & vbCrLf & " '{0}'" & vbCrLf & _
        "after trimming is " & vbCrLf & " '{1}'", s5, s5.Trim())
End Sub
```

Output del programma

```
Length of faraz is 5
Comparision result for faraz with fraz is -1
Equality checking of faraz with Faraz returns False
Equality checking of faraz with lowercase Faraz (faraz) returns
True
The index of a in Faraz is 1
The last index of a in Faraz is 3
The individual words of 'VB.Net is a great programming language!'
are
VB.Net
is
a
great
programming
language!
The substring of
'VB.Net is a great programming language!'
from index 3 of length 10 is
'Net is a g'
The string
' This is the target text '
after trimming is
```

```
'This is the target text'
```

String.Format vedere più avanti la funzione Format

Le date

Proprietà delle date

Esempio

```
Dim dteData As Date
Dim str As String
dteData = Now
str = "Data: " & dteData.Date : scrive(str)
str = "Data (ToLongDateString): " & dteData.ToLongDateString : scrive(str)
str = "Data (ToShortDateString): " & dteData.ToShortDateString : scrive(str)
str = "Time (ToLongTimeString): " & dteData.ToLongTimeString : scrive(str)
str = "Time (ToShortTimeString): " & dteData.ToShortTimeString : scrive(str)
str = "-----------------" : scrive(str)
str = "Month: " & dteData.Month : scrive(str)
str = "Day: " & dteData.Day : scrive(str)
str = "Year: " & dteData.Year : scrive(str)
str = "Hour: " & dteData.Hour : scrive(str)
str = "Minute: " & dteData.Minute : scrive(str)
str = "Second: " & dteData.Second : scrive(str)
str = "Day of week: " & dteData.DayOfWeek : scrive(str) 'parte da Domenica=0
str = "Day of year: " & dteData.DayOfYear : scrive(str)
str = "Weekday name(dteData.ToString(""dddd"")) : " & dteData.ToString("dddd") : scrive(str)
str = "Month name(dteData.ToString(""MMMM"")) : " & dteData.ToString("MMMM") : scrive(str)
```

Output

Data: 04/01/2011
Data (ToLongDateString): martedì 4 gennaio 2011
Data (ToShortDateString): 04/01/2011
Time (ToLongTimeString): 11.16.00
Time (ToShortTimeString): 11.16

Month: 1
Day: 4
Year: 2011
Hour: 11
Minute: 16
Second: 0
Day of week: 2
Day of year: 4
Weekday name(dteData.ToString("dddd")) : martedì
Month name(dteData.ToString("MMMM")) : gennaio

Manipolare le date

Esempio

```
Dim str As String
Dim dteStartData As Date
Dim dteChangedData As Date
Dim differenzaData As Long

dteStartData = #2/28/2400#

str = "Data di partenza: " & dteStartData.ToLongDateString : scrive(str)
dteChangedData = dteStartData.AddDays(1)
str = "Nuova Data dopo AddDays(1): " & dteChangedData.ToLongDateString : scrive(str)
'
dteChangedData = dteStartData.AddMonths(6)
str = "Nuova Data dopo .AddMonths(6) " & dteChangedData.ToLongDateString : scrive(str)
'
dteChangedData = dteStartData.AddYears(-1)
str = "Nuova Data dopo AddYears(-1): " & dteChangedData.ToLongDateString : scrive(str)
'
differenzaData = DateDiff(DateInterval.Day, dteStartData, dteChangedData)
str = "differenza tra due date(DateDiff(DateInterval.Day, dteStartData, dteChangedData)): " &
differenzaData.ToString : scrive(str)
```

Output
Data di partenza: lunedì 28 febbraio 2400
Nuova Data dopo AddDays(1): martedì 29 febbraio 2400
Nuova Data dopo .AddMonths(6) lunedì 28 agosto 2400
Nuova Data dopo AddYears(-1): domenica 28 febbraio 2399
differenza tra due date(DateDiff(DateInterval.Day, dteStartData, dteChangedData)): -365

Formattare stringhe e date (tale formattazione è influenzata dalla versione locale del computer)

```
Dim TestDateTime As Date = #1/27/2001 5:04:23 PM#
'--- si deve usare il cancelletto (#) per delimitare la data letterale;
'--- si deve definire nel formato americano mese/giorno/anno indipendentemente
'--- dalle impostazioni locali del computer.

Dim TestStr As String
' Returns current system time in the system-defined long time format.
TestStr = Format(Now(), "Long Time")

' Returns current system date in the system-defined long date format.
TestStr = Format(Now(), "Long Date")

' Also returns current system date in the system-defined long date
' format, using the single letter code for the format.
TestStr = Format(Now(), "D")

' Returns the value of TestDateTime in user-defined date/time formats.
' Returns "5:4:23".
TestStr = Format(TestDateTime, "h:m:s")

' Returns "05:04:23 PM".
TestStr = Format(TestDateTime, "hh:mm:ss tt")

' Returns "Saturday, Jan 27 2001".
TestStr = Format(TestDateTime, "dddd, MMM d yyyy")

' Returns "27/01/2001".
TestStr = Format(TestDateTime, "dd/MM/yyyy")

' Returns "17:04:23".
TestStr = Format(TestDateTime, "HH:mm:ss")

' Returns "23".
TestStr = Format(23)

' User-defined numeric formats.
' Returns "5,459.40".
TestStr = Format(5459.4, "##,##0.00")

' Returns "334.90".
TestStr = Format(334.9, "###0.00")

' Returns "500.00%".
TestStr = Format(5, "0.00%")

' Returns "5.00".
TestStr = Format(5, "0.00")
```

Selezione e Iterazione

- ## If … then …. Else

```
If condizione Then
istruzioni1
[Else
istruzioni2 ]
End If
```

Una variante della dichiarazione I*f ... Then ... Else* è quella in cui si utilizza la parola chiave **ElseIf** per elencare più condizioni differenti:

```
If condizione1 Then
istruzioni1
ElseIf condizione2 Then
istruzioni2
ElseIf condizione3 Then
Istruzioni3
Else
                    Istruzioni4
End If
```

Esempio costrutto **ElseIf**:
Problema: *dato un punteggio visualizzare il corrispondente giudizio*

```
Private Sub Button1_Click(...) Handles Button1.Click
Dim punteggio As Integer
Dim esito As String
punteggio = InputBox("Inserisci il punteggio")
If punteggio < 50 Then
esito = "Non superato"
ElseIf punteggio >= 50 And punteggio < 75 Then
esito = "Superato"
ElseIf punteggio >= 75 And punteggio < 90 Then
esito = "Molto buono"
Else
esito = "Eccellente"
End If
MessageBox.Show(esito)
End Sub
```

- ## Select …. Case

```
Select Case VariabileControllo
Case valore_1
istruzioni_1
Case valore_2
istruzioni_2
.......
Case valore_n
istruzioni_n
[Case Else
Istruzioni]
End Select
```

Nota: si può anche scrivere l' istruzione sulla stessa riga mettendo **:** dopo il valore, esempio:

Case valore_n : istruzioni_n

Per inserire valori:

```
Case 1, 2, 5, 10 To 20
```

Per esprimere un intervallo:

```
Case 1 To N
```

Per esprimere maggiore/minore di ...

```
Case Is > N
Case Is < N
```

Esempio costrutto **Select Case**:
Problema (come ElseIf): *dato un punteggio visualizzare il corrispondente giudizio*

```
Private Sub Button2_Click(...) Handles Button2.Click
Dim punteggio As Integer
Dim esito As String
punteggio = InputBox("Inserisci il punteggio")
Select Case punteggio
Case Is < 50
esito = "Non superato"
Case 50 To 74
esito = "Superato"
Case 75 To 89
esito = "Molto buono"
Case Is >= 90
esito = "Eccellente"
End Select
MessageBox.Show(esito)
End Sub
```

- **Iterazione**

1) iterazione **precondizionale**

- **Do while …. Loop**

```
Do While condizione
istruzioni
Loop
```

cicla su condizione=True, esce quando diventa False

- **Do until …. Loop**

```
Do Until condizione
istruzioni
                    Loop
```

cicla su condizione=False, esce quando diventa True

2) iterazione **postcondizionale**

- **Do ….. Loop until**

```
Do
istruzioni
Loop Until condizione
```

cicla su condizione=False, esce quando diventa True

- **Do ….. Loop while**

```
Do
istruzioni
Loop While condizione
```

cicla su condizione=True, esce quando diventa False

3) **iterazione enumerativa**

- **For …. Next**

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

se Step *incremento* non viene indicato, si intende *incremento* = 1
La variabile **incremento** può contenere sia valori positivi che negativi. Se incremento è positivo **inizio** deve essere < di **fine**, in caso contrario le istruzioni interne al ciclo non vengono eseguite. Analogamente se **incremento** è negativo **inizio** deve essere > di **fine**

- **For each ….. next**

Esempi di iterazione

Problema: *somma di 5 numeri dati dall utente*
Struttura di sequenza:

```
Private Sub btnSeq_Click(...) Handles btnSeq.Click
Dim num1, num2, num3, num4, num5, somma As Integer
num1 = InputBox("Inserisci un numero")
num2 = InputBox("Inserisci un numero")
num3 = InputBox("Inserisci un numero")
num4 = InputBox("Inserisci un numero")
num5 = InputBox("Inserisci un numero")
somma = num1 + num2 + num3 + num4 + num5
```

```
MessageBox.Show("La somma dei numeri inseriti è: " & somma)
End Sub
```

Struttura iterativa con For

```
Private Sub btnFor_Click(...) Handles btnFor.Click
Dim num, somma, i As Integer
For i = 1 To 5
num = InputBox("Inserisci un numero")
somma = somma + num
Next
MessageBox.Show("La somma dei numeri inseriti è: " & somma)
End Sub
```

In questo esempio di codice la variabile **i** è il contatore del ciclo **For**.
Durante l' esecuzione del ciclo **For,** Visual Basic provvede a:
1) inizializzare **i** a 1
2) controllare se **i** è maggiore di **5**, in caso affermativo Visual Basic interrompe il ciclo ed esegue l' istruzione successiva a **Next** (nel nostro esempio l' istruzione **MessageBox**)
3) eseguire il blocco di istruzioni comprese tra le parole chiave **For** e **Next**
4) incrementare la variabile **i** di 1
5) ripetere i passaggi da 2. a 4.

Sottoprogrammi

Un problema, scomposto in sottoproblemi mediante l'analisi top-down, può essere implementato utilizzando:
un ***programma principale*** (main) per il problema generale
dei ***sottoprogrammi*** (sub / function / method /...) per la soluzione di ciascun sottoproblema.
Il programma principale richiama (tramite il suo nome) il sottoprogramma necessario in un certo punto. A sua volta il sottoprogramma può richiamare altri sottoprogrammi(struttura nidificata).
Il sottoprogramma può essere richiamato più volte, in punti diversi del programma, senza la necessità di dover ripetere le sue istruzioni.
L'uso dei sottoprogrammi risponde a tre esigenze fondamentali:
1. un blocco di istruzioni che viene eseguito più volte all'interno dello stesso programma, con dati diversi, può essere scritto una sola volta e poi richiamato ogni volta che serve passandogli i dati necessari per l'elaborazione;
2. spesso in programmi diversi possono essere utilizzati blocchi di istruzioni uguali che lavorano su dati diversi: conviene allora organizzare questi blocchi in modo tale da non doverli riscrivere ogni volta ;
3. l'uso delle procedure consente di scrivere il programma principale indicando semplicemente le parti fondamentali di cui si compone con un nome (nome della procedura): la procedura, che implementa il sottoproblema, viene realizzata a parte.
Questo rende il programma più leggibile, anche a distanza di tempo, e realizza nella pratica il metodo dei raffinamenti successivi (top-down).

I parametri
Usare molte variabili globali e lavorare direttamente su queste all'interno dei sottoprogrammi non è un buon metodo di programmazione. Il modo migliore di procedere è passare i dati che servono ai sottoprogrammi come *parametri* e ricevere i risultati allo stesso modo.
I parametri sono un elenco di dati che possono essere scambiati tra un sottoprogramma e il programma principale (o un altro sottoprogramma). I parametri che compaiono nella definizione del sottoprogramma sono chiamati **parametri formali** e sono utilizzati all' interno della procedura come una qualsiasi variabile locale. I dati che vengono sostituiti ai parametri formali al momento del richiamo del sottoprogramma vengono detti **parametri attuali**.
L'assegnazione dei valori, dai **parametri attuali** ai **parametri formali** avviene in base alla **posizione**: il primo parametro attuale è associato al primo parametro formale, il secondo parametro attuale è associato al secondo parametro formale, e così via; pertanto e importante che **i parametri attuali** e **i parametri formali** siano **uguali per numero e per tipo**!

Passaggio per valore **(by value ==> ByVal)**
Al **parametro formale** viene assegnato il **valore** del **parametro attuale**, ma i due dati sono distinti in quanto si trovano in due diverse zone di memoria.
In questo caso il parametro formale è una copia temporanea del dato fornito dal parametro attuale, quindi anche se viene modificato all interno del sottoprogramma, la modifica non ha alcun effetto sul parametro attuale (ossia sulla variabile del programma chiamante) una volta che il sottoprogramma termina.

Passaggio per riferimento o indirizzo **(by reference ==> ByRef)**
Il **parametro formale** e il **parametro attuale** fanno **riferimento alla stessa zona di memoria**.
Al sottoprogramma viene assegnato l' indirizzo di memoria del parametro (puntatore,reference), quindi il sottoprogramma accede direttamente alla variabile, di conseguenza qualsiasi modifica della variabile all'interno del sottoprogramma si ripercuote sul parametro attuale.
RICAPITOLANDO:

I parametri che devono essere modificati nel sottoprogramma devono essere passat**i by reference.**
I parametri che non devono essere modificati nel sottoprogramma devono essere passati **by value.**

Per default in VB il passaggio avviene **ByVal.**

NOTA: se il parametro è una struttura dati (array o file), o comunque un oggetto, il passaggio avviene sempre per reference nel senso che in ogni caso la modifica fatta nella procedura sulla struttura o l'oggetto si ripercuote sul dato originale.

Sottoprogrammi: le procedure (Subroutine o SUB)
Sintassi:
Private/Public Sub *nomeSubroutine* (ListaParametriFormali)
istruzioni
End Sub
Esempio: ordinare tre numeri letti tramite TextBox
Dichiarazione della procedura:

```
 Private Sub ordina2numeri(ByRef n1 As Integer, ByRef n2 As Integer)
[istruzioni per ordinare]
 End Sub
```

Richiamo della procedura nel programma principale:

```
Private Sub btnSort_Click(ByVal sender As _
System.Object,ByVal e As _
System.EventArgs) Handles btnSort.Click
Dim x, y, z As Integer
x = txt1.Text
y = txt2.Text
z = txt3.Text
ordina2numeri(x, y)
ordina2numeri(x, z)
ordina2numeri(y, z)
lblRis.Text = "Sorting: " & x & " " & y & " " & z
End Sub
```

Sottoprogrammi: le Funzioni (FUNCTION)
La funzione è un sottoprogramma che può avere un elenco di parametri, come la procedura, e che restituisce un valore associato al nome della funzione stessa. Quindi la funzione ha un **tipo** che è il tipo del valore che restituisce.
Quando una procedura restituisce un solo valore può essere realizzata come funzione.
Sintassi:

```
Private/Public Function nomeFunzione (ListaParametriFormali) As Tipo
      istruzioni
      Return valore
End Function
```

Nota: VB2008 accetta anche la sintassi delle versioni precedenti, dove l' istruzione :
Return *valoreFunzione*
era sostituita da:
nomeFunzione = *valore*
in cui il nome della funzione è usato come una variabile alla quale assegnare il valore da restituire.
La funzione viene richiamata in un programma principale, o in un altro sottoprogramma, usando il nome della funzione come fosse il nome di una variabile e quindi può comparire in un' istruzione di assegnazione (=) o all'interno di espressioni (ad esempio nel caso di una funzione che restituisce un valore di tipo boolean: **If *nomeFunzione* Then ...**)
Esempio:

calcolare l' area di un cerchio il cui raggio è letto tramite TextBox
Dichiarazione della funzione:

```
Private Function AreaCerchio(ByVal raggio As Single) As Single
Dim area As Single
area = raggio ^ 2 * Math.PI
Return area
End Function
```

Richiamo della funzione nel programma principale:

```
                Private Sub btnCalcola_Click(ByVal sender As System.Object,
                ByVal e As _
System.EventArgs) Handles btnCalcola.Click
lblRis.Text = "L'area del cerchio è: " & AreaCerchio(txtR.Text)
End Sub
```

Passaggio di parametri
Esempio con passaggio di parametri ByVal e ByRef:

```
Private Sub Incrementa(ByVal n1 As Integer, ByRef n2 As Integer)
n1 += 1
n2 += 1
lblProc.Text = n1 & " " & n2
End Sub

Private Sub btnCalcola_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnCalcola.Click
Dim a, b As Integer
a = 0
b = 0
Incrementa(a, b) richiamo della procedura Incrementa
lblMain.Text = a & " " & b
End Sub
```

Esempio con confronto tra Sub e Function che svolgono la stessa operazione (somma dei numeri compresi in un intervallo):

```
'Esempio con SUB
Private Sub Sub_SommaNumeri(ByVal a As Integer, _
ByVal b As Integer,ByRef somma As Integer)
Dim i As Integer
For i = a To b
somma += i
Next
End Sub

Private Sub bntS_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles bntS.Click
Dim x, y, s As Integer
x = txt1.Text
y = txt2.Text
Sub_SommaNumeri(x, y, s)
lblRis.Text = "La somma dei numeri compresi " & _
"nell'intervallo indicato è: " & s
End Sub
================================
```

```
'Esempio con FUNCTION
Private Function Fun_SommaNumeri(ByVal a As Integer, _
ByVal b As Integer) As Integer
Dim i, sum As Integer
For i = a To b
sum += i
Next
Return sum
End Function

Private Sub btnF_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnF.Click
Dim x, y, s As Integer
x = txt1.Text
y = txt2.Text
s = Fun_SommaNumeri(x, y)
lblRis.Text = "La somma dei numeri compresi " & _
"nell'intervallo indicato è: " & s
End Sub
```

La metodologia Top-Down e i sottoprogrammi

La metodologia top-down prevede la scomposizione di un problema in tanti sottoproblemi più semplici. Lo sviluppo di algoritmi con questa metodologia ha i vantaggi:

1. ***leggibilità***: il programma è organizzato in moduli semplici e autonomi; ogni modulo non dovrebbe superare cento istruzioni;
2. ***riutilizzabilità***: i vari moduli potrebbero essere utilizzati in programmi diversi;
3. ***affidabilità***: ogni sottoprogramma può essere sviluppato e testato facilmente;
4. ***revisionabilità***: è più facile revisionare e modificare il software in quanto è possibile isolare i moduli da revisionare senza dover andare a toccare gli altri; questo richiede, però, che ogni modulo sia sviluppato e funzioni autonomamente rispetto agli altri.

CONVIENE descrivere un'attività per mezzo di un sottoprogramma quando:

- è di interesse generale;
- non è di interesse generale, ma si presenta più volte all'interno di un programma;
- pur essendo di scarso interesse generale, permette una maggiore leggibilità del programma.

NON CONVIENE descrivere un'attività per mezzo di un sottoprogramma quando:

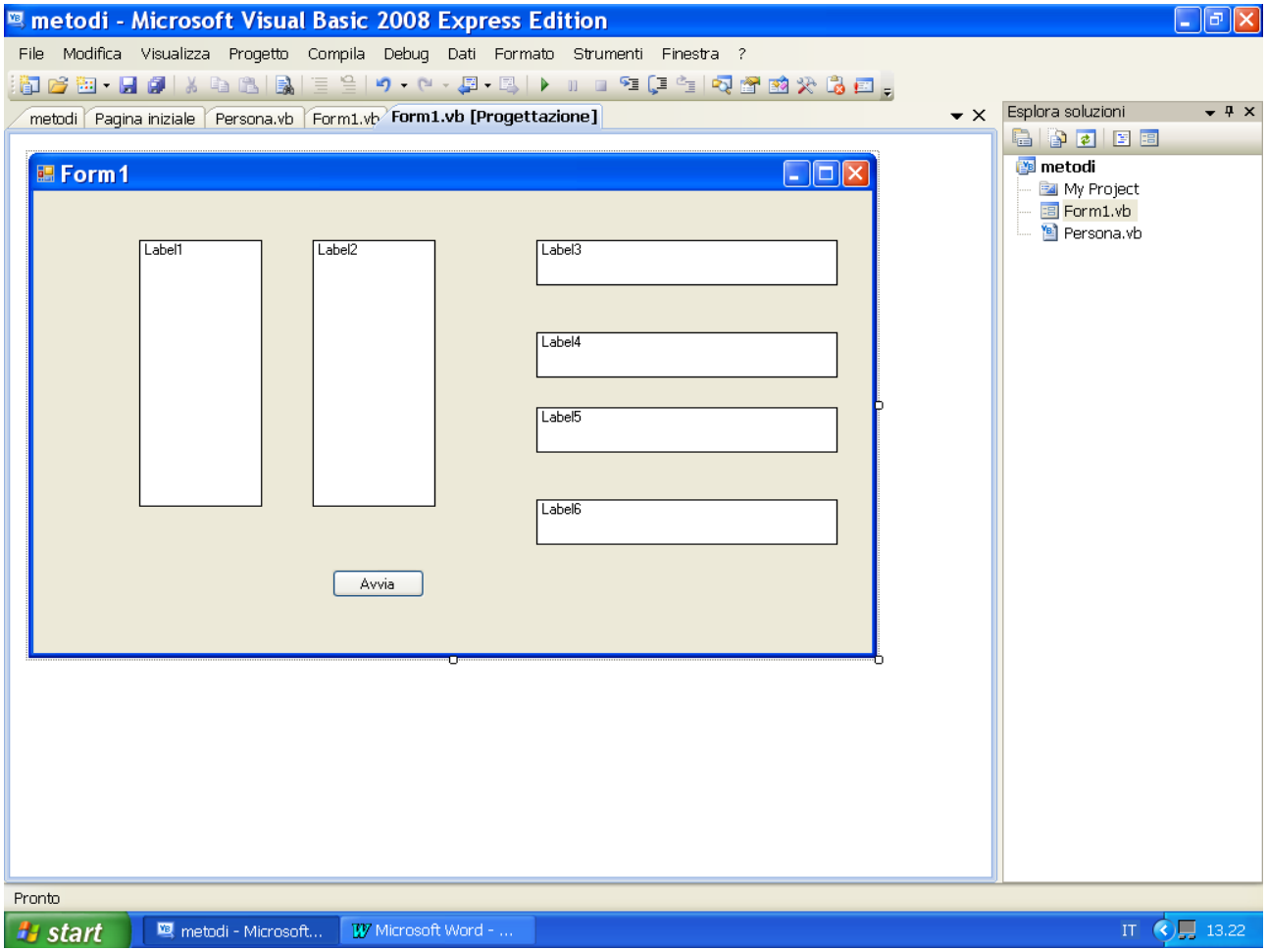
- è di scarso interesse generale;
- non migliora la leggibilità del programma o addirittura la complica;
- non garantisce un risparmio di tempo soprattutto se si tratta di un programma breve.

CONSIGLI di progettazione delle procedure con parametri:

- le procedure devono comunicare all esterno solo tramite parametri;
- il numero dei parametri non deve essere eccessivo (rivedere la progettazione);
- le procedure non devono usare variabili globali e soprattutto non devono modificarle;
- le procedure che risolvono un certo problema non devono usare istruzioni di input/output, ma comunicare i dati solo tramite parametri, (ad esempio: si deve realizzare una procedura per determinare il massimo tra due numeri, la richiesta del valore dei numeri (input) non deve essere

fatta all'interno della procedura, bensì prima del suo richiamo e i valori ottenuti passati come parametro alla procedura);

- è buona regola di programmazione gestire l'input e l'output in apposite procedure, in tal modo se si decide di cambiare modalità di input (ad esempio si vuole usare una TextBox al posto di una finestra InputBox), si deve revisionare solo una procedura.




```
'--------- provare con Option Strict
Public Class Form1
Private Sub btnAvvia_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnAvvia.Click
        Dim n(9) As Integer
        Dim a As Integer, b As Integer
        Dim i As Integer
        Dim str1 As String
        Dim str2 As String
        Dim p As Persona
        p = New Persona


        Label1.Text = ""
        Label2.Text = ""

        a = 10
        b = 20
        str1 = "prima stringa"
        str2 = "seconda stringa"


        carica1(n)
        ordina1(a, b, str1, str2)
        For i = 0 To 9
            Label1.Text = Label1.Text & n(i) & vbCrLf
```

```
        Next
        Label3.Text = " " & a & " " & b & vbCrLf & str1 & " - " & str2
        valorizzaPersona1(p)
        Label5.Text = p.cognome & " " & p.nome & " " & p.eta
        '---------------------------------------
        a = 10
        b = 20
        str1 = "prima stringa"
        str2 = "seconda stringa"

        For i = 0 To 9
            n(i) = 0
        Next
        carica2(n)
        ordina2(a, b, str1, str2)
        For i = 0 To 9
            Label2.Text = Label2.Text & n(i) & vbCrLf
        Next
        Label4.Text = " " & a & " " & b & vbCrLf & str1 & " - " & str2
        valorizzaPersona2(p)
        Label6.Text = p.cognome & " " & p.nome & " " & p.eta

    End Sub
Public Sub carica1(ByVal x() As Integer)
        For i As Integer = 0 To 9
            x(i) = i + 1
        Next
    End Sub
Public Sub carica2(ByRef x() As Integer)
        For i As Integer = 0 To 9
            x(i) = i + 10
        Next
    End Sub
Public Sub ordina1(ByVal x1 As Integer, ByVal x2 As Integer, ByVal s1 As String, ByVal s2 As String)
        Dim i As Integer
        Dim h As String
        h = s1
        i = x1
        x1 = x2
        x2 = i
        s1 = s2
        s2 = h
    End Sub
Public Sub ordina2(ByRef x1 As Integer, ByRef x2 As Integer, ByRef s1 As String, ByRef s2 As String)
        Dim i As Integer
        Dim h As String
        h = s1
        i = x1
        x1 = x2
        x2 = i
        s1 = s2
        s2 = h
    End Sub
Public Sub valorizzaPersona1(ByVal q As Persona)
        q.cognome = "rossi"
        q.nome = "enzo"
        q.dataNascita = "25/12/1950"
    End Sub
Public Sub valorizzaPersona2(ByRef q As Persona)
        q.cognome = "rossi"
        q.nome = "enzo"
        q.dataNascita = #12/25/1950#
    End Sub
End Class
'-----------------------------------------------
Public Class Persona
    Private _cognome As String
    Private _nome As String
    Private _dataNascita As Date
    Public Property cognome() As String
        Get
            Return _cognome
        End Get
        Set(ByVal value As String)
            _cognome = value
        End Set
    End Property
    Public Property nome() As String
```

```
        Get
            Return _nome
        End Get
        Set(ByVal value As String)
            _nome = value
        End Set
    End Property
    Public Property dataNascita() As Date
        Get
            Return _dataNascita
        End Get
        Set(ByVal value As Date)
            _dataNascita = value
        End Set
    End Property
    Public Function eta() As Integer
        Dim calcoloEta As Integer
        calcoloEta = DateDiff(DateInterval.Year, dataNascita, Now)
        Return calcoloEta
    End Function

End Class

---------------------- Sub per caricare un vettore con numeri casuali e ordinarlo --------------
Public Sub carica1(ByVal x() As Integer)
        Dim r As New Random

        For i As Integer = 0 To 9
x(i) = r.Next(100) '-----numero casuale compreso tra 0 e 99
        Next
        Array.Sort(x)
    End Sub
```

Esercizi sui sottoprogrammi

1) dare in input n valori interi stampare i valori primi
2) dare in input n valori interi, rappresentanti degli anni, dire per ognuno di essi se si tratta di un anno bisestile
3) caricare degli array di valori interi, ordinarli e stamparli

La programmazione orientata agli oggetti

Un programma non manipola solo numeri, caratteri, stringhe ma dati più complessi quali ad esempio conti correnti bancari, informazioni sui clienti, sugli alunni, sugli impiegati, forme grafiche, etc.. ; tutti questi tipi di dati possono essere definiti *oggetti*, pertanto possiamo dire che un

oggetto è un'entità che possiede delle caratteristiche (**attributi o proprietà**) e la capacità di fare qualcosa (**metodi**)
una
classe è la decrizione di un oggetto o meglio di tutti i possibili oggetti con quelle caratteristiche e quelle capacità (attributi e metodi)

Caratteristiche della programmazione ad oggetti

- **incapsulamento**: è il processo che nasconde i dati di un oggetto (proprietà -> variabili istanza); questi dati sono accessibili solo attraverso dei metodi:in pratica gli attributi o proprietà devono essere dichiarati privati
- **ereditarietà**: possibilità di una classe (classe derivata o sottoclasse) di ereditare proprietà e metodi da un'altra classe (classe base o superclasse)

public Class SportsCar
Inherits *Car*
……
end Class

la classe ***SportsCar*** eredita (**Inherits**) metodi e proprietà dalla classe ***Car***; può inoltre avere suoi metodi e proprietà.

- **polimorfismo**: la stessa elaborazione funziona per oggetti di tipo diverso e si adatta alla natura degli oggetti o meglio classi diverse hanno metodi con lo stesso nome con comportamenti diversi ad esempio TextBox, ComboBox, ListBox hanno metodi e proprietà con lo stesso nome ma comportamenti diversi a seconda del controllo.
- **overloading**: possibilità, all'interno della stessa classe di definire metdodi con lo stesso nome ma diversi per numero e/o tipo di parametri.
- **costruttore**: è un metodo particolare il cui scopo è quello di inizializzare gli attributi; si definisce come

Sub New (eventuali parametri) - possibilità di overload come in Java
……

end sub

se non presente viene utilizzato il costruttore di default per cui si assegnano alle proprietà di un nuovo oggetto i valori di default a seconda del tipo

Esempio di classe

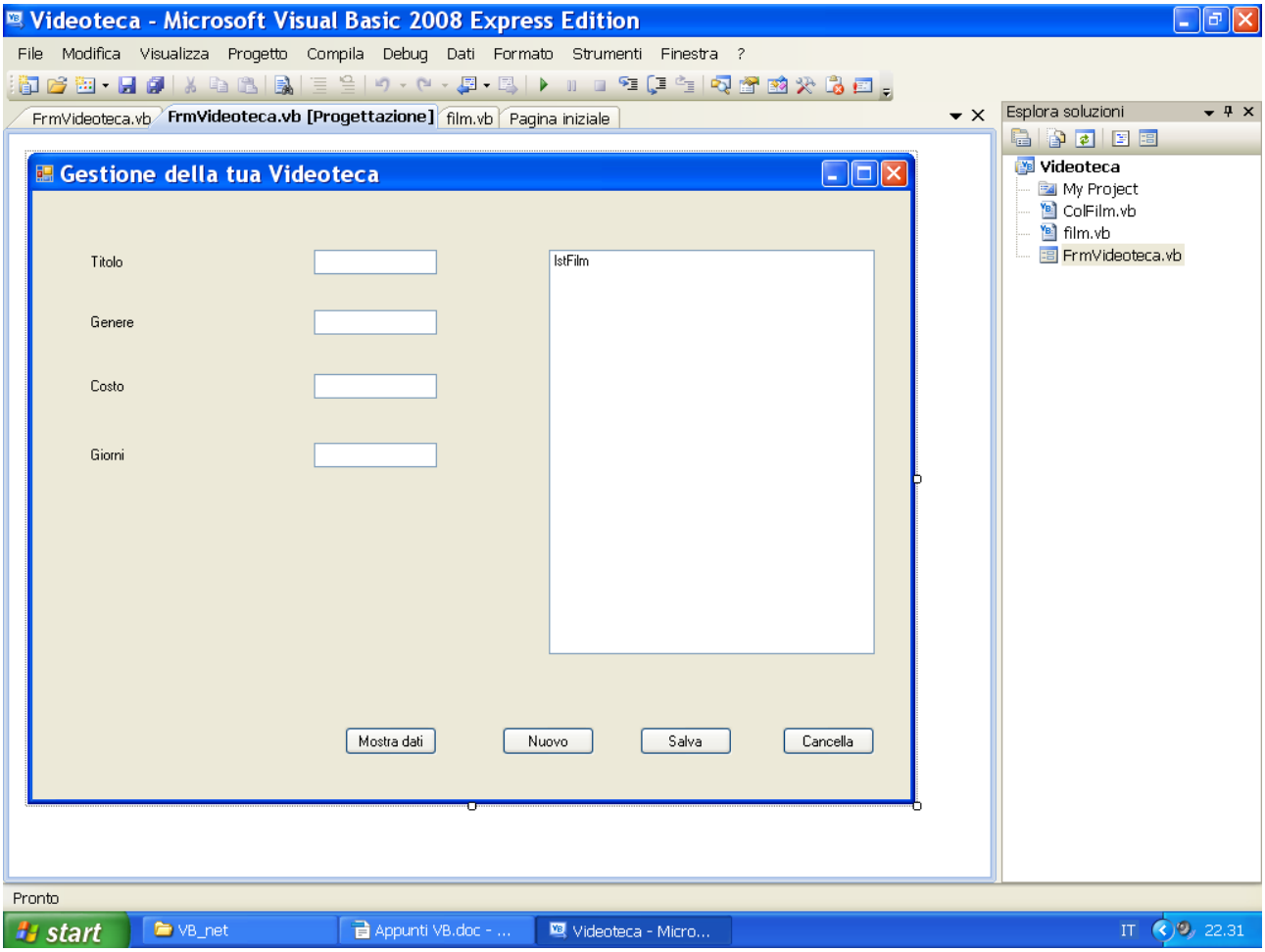
```
Public Class film
    '--- proprietà ------------
    Private mTitolo As String
    Private mGenere As String
    Private mCosto As Decimal
    '
    Property Titolo() As String
        Get
            Return mTitolo
```

```
        End Get
        Set(ByVal value As String)
            mTitolo = value
        End Set
    End Property
    Property Genere() As String
        Get
            Return mGenere
        End Get
        Set(ByVal value As String)
            mGenere = value
        End Set
    End Property
    Property Costo() As Decimal
        Get
            Return mCosto
        End Get
        Set(ByVal value As Decimal)
            mCosto = value
        End Set
    End Property
    Public Function calcolaCosto(ByVal giorni As Integer) As Decimal
        calcolaCosto = Costo * giorni
    End Function

End Class
```

- **variabile oggetto**

Esempio



```
Private Sub btnMostraDati_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnMostraDati.Click
        Dim costoTotale As Decimal
        Dim objFilm As film     ' dichiarazione  oppure Dim  objFilm As  new film
        Dim messaggio As String
        objFilm = New film      'creazione
        objFilm.Titolo = TxtTitolo.Text
        objFilm.Genere = txtGenere.Text
        objFilm.Costo = CDec(txtCosto.Text)
        costoTotale = objFilm.calcolaCosto(CInt(txtGiorni.Text))
        messaggio = "importo dovuto per il noleggio del film: " & _
        objFilm.Titolo & " costo totale:" & costoTotale
        MessageBox.Show(messaggio)
        objFilm = Nothing ' rilascio
    End Sub
```

Console Application

Un' applicazione di tipo Console è un applicazione a riga di comando, cioè senza interfaccia grafica, che deve essere eseguita in una finestra a caratteri (CUI, Character User Interface).
In un' applicazione console le operazioni di input avvengono tramite la tastiera mentre l' output è visualizzato sottoforma di testo sul monitor.
In Windows le operazioni sulla console sono effettuate nella finestra **Prompt dei comandi** .
Quando si apre un progetto console viene visualizzata una finestra denominata Module1.vb in cui scrivere il codice del nostro programma:
All interno del modulo possono essere presenti più sottoprogrammi: quello fondamentale è denominato **Main** : è quello che viene eseguito all avvio dell applicazione, quello, cioè, che contiene la prima istruzione che dovrà essere eseguita.

Istruzioni di Input/Output per la console

Comando	**Utilizzo**	**Descrizione**	**Esempio**
Console.Write	OUTPUT	visualizza sul monitor una stringa (racchiusa tra "") o il contenuto di una variabile	`Console.Write ( "Hello World!" )`
Console.WriteLine	OUTPUT	come Write, in più introduce un ritorno a capo	
Console.Read	INPUT	raccoglie tutti i dati digitati da un utente in una linea della console e li assegna ad una variabile di tipo stringa	`Dim s As String` `s = Console.Read()`
Console.ReadLine	INPUT	come Read, in più introduce un ritorno a capo	`Dim s As String` `s =Console.ReadLine()`

Esempio:

```
Sub Main()
Dim x As String
x = "Visual Basic 2008"
Console.Write("prima lezione di: " & x)
Console.ReadLine()
End Sub
```

L' istruzione Console.ReadLine, alla fine del programma, permette di fermare l' esecuzione dell'applicazione così visualizzare la finestra del *Prompt di comandi* finché non si digita un carattere qualsiasi per terminare l' applicazione.

Le Windows Form

- **Creare**

- Barra menù -> progetto -> aggiungi Windows Form
- icona da barra degli strumenti
- finestra esplora soluzioni -> tasto dx su progetto -> aggiungi -> Windows Form

- **Salvare**

- Barra menù -> file -> salva … o salva tutto
- icona da barra degli strumenti
- finestra esplora soluzioni -> tasto dx su progetto -> aggiungi -> Windows Form

- **Eliminare**

- finestra esplora soluzioni -> tasto dx su windows form da eliminare -> elimina

- **Scelta della Form di avvio**

- Barra menù -> progetto -> proprietà -> casella form di avvio -> selezionare
- finestra esplora soluzioni -> tasto dx su progetto -> proprietà -> casella form di avvio -> selezionare

- **Le proprietà, gli eventi, i metodi**

Proprietà :

- *FormBorder Style* : aspetto e comportamento del bordo e della barra del titolo del form; può assumere i valori:
- *None* : non viene visualizzato nessun bordo e nemmeno la barra del titolo; la form non può essere spostata, ridimensionata, ridotta a icona.
- *Fixed Single* : non può essere ridimensionata
- *Fixed3D*: come *FixedSingle* con bordo tridimensionale
- *FixedDialog*: come *FixedSingle*; manca la casella di controllo
- *Sizable*: è il valore di default vengono visualizzati tutti i pulsanti: casella di controllo, barra del titolo, riduzione a icona, ingrandimento, chiusura
- *FixedToolWindows*: visualizzati solo barra del titolo, e pulsante di chiusura; la finestra può essere spostata ma non ridimensionata
- *SizableToolWindow*: visualizzati solo barra del titolo, e pulsante di chiusura; la finestra può essere spostata e ridimensionata
- *Text*: testo visualizzao nella barra del titolo
- *ControlBox (True/False)* : casella di controllo SI/NO
- *MaximizeBox (True/False)*: casella ingrandimento SI/NO
- *MinimizeBox (True/False)* : casella riduzione a icona SI/NO
- *Icon* : eventuale icona visualizzata nell'angolo superiore sinistro (casella di controllo) o quando la form viene ridotta ad icona
- *WindowState*: stato di visualizzazione iniziale della form
- *Normal*: valore di default; form visualizzata con valori normali
- *Minimized*: form ridotta ad icona

- *Maximized* : form a tutto schermo
- *StartPosition*: posizione iniziale della form
- *Manual*: nessuna posizione particolare
- *CenterScreen*: al centro dello schermo
- *CenterParent*: al centro rispetto al form padre
- *WindowsDefaultLocation*: nella posizioni predefinita di Windows
- *WindowsDefaultBounds*: nella posizione predefinita di Windows con limitazione sulle dimensioni
- *AutoScroll(True/False)*: comparsa SI/NO delle barre di scorrimento verticale/orizzontale se la form contiene controlli non contenuti nell'area non visualizzata della form

Eventi

Gli eventi scatenati in una form dal momento della sua creazione al momento in cui viene chiusa

- *Move*: scatenato quando si muove la form.Di default, anche se l'utente non muove la form, viene scatenato prima dell'evento Load, quando la form viene istanziata e lanciata.
- *Load*: l'evento viene scatenato quando la form viene creata e caricata anche se non è ancora visibile
- *VisibleChanged*: scatenato quando cambia il valore della proprietà Visible
- *Activate*: viene scatenato ogni volta che l'utente seleziona la form.
- *Activated*: viene scatenato nel momento in cui la form diventa attiva, subito dopo l'evento Paint
- *Deactivate*: viene scatenato quando la form perde il focus
- *Paint* : viene scatenato immediatamente prima che la form diventi visibile ed ogni volata che la form viene ridisegnata (ingrandita, ridotta ad icona, ripristinata)
- *Shown* : l'evento viene scatenato quando la form diventa visibile: è stato chiamato il metodo form.Show oppure si è asseganata la proprietà form.Visible = true; se si nasconde la form (form.Hide) e dopo viene di nuovo visualizzata (form.Show/ form.Visible = true) l'evento *Shown* viene nuovamente scatenato ma non l'evento *Load*
- *Closing*: viene scatenato alla chiusura della form. Se si annulla l'evento (proprietà Cancel = true passata al gestore di eventi) la form resta aperta; normalmente viene utilizzato per chiedere conferma se salvare o meno dei dati.
- *Closed*: viene scatenato quando la form è chiusa.

Sequenza degli eventi
Eventi attivati prima che la Form venga visualizzata:
1 - Move
2 - Load
3 - VisibleChanged
4 - Activated

Eventi attivati alla visualizzazione della Form
5 -Shown
6 - Paint

Attivato quando si passa ad un'altra applicazione/form
7 – Deactivate

Attivati quando riguadagna il focus

3 - Activated
6 - Paint

Attivati quando la Form viene chiusa
8 - FormClosing
7 - FormClosed
9 - Deactivate

Metodi

- Show :
- Frm1.show : finestra modeless (non modale)
- Frm1.showDialog : finestra modal (modale: a scelta obbligata)
- Hide

• Interfaccia a documenti multipli (MDI : Multiple-Document Interface)

L' interfaccia a documenti multipli è costituita da una form "padre" che la cui area centrale ("client area") è destinata a contenere altre form : le cosiddette form figlie.
Qualsiasi form può diventare una form padre mettendo a true la proprietà IsMdiContainer; la form figlia deve indicare nella proprietà MdiParent la form padre

I controlli

I controlli sono componenti che possono esssere utilizzati in un progetto VB.
La casella degli strumenti (ToolBox) contiene l'elenco dei controlli.

• I controlli (più comuni)

- *Puntatore*: non è un controllo ma il mouse dalla modalità di "inserimento controllo" torna alla forma "puntatore"
- *Button*: bottone, pulsante
- *CheckBox*: consente di selezionare l'opzione associata
- *ComboBox*: TextBox+ListBox : casella di testo modificabile con elenco di valori selezionabili
- *GroupBox*: contenitore di altri controlli
- *Label*: etichetta
- *LinkLabel*: etichetta con collegamenti ipertestuale
- *ListBox*: visualizza un elenco da cui l'utente può selezionare deglki elementi
- *PictureBox*: visualizza un'immagine
- *RadioButton*: consente di selezionare una e una sola opzione tra più opzioni possibili (altri RadioButton)
- *TextBox*: casella di testo

Proprietà comuni

Aspetto

BackColor: colore di sfondo
ForeColor: colore di primo piano
Font: font del carattere
Text: testo associato al controllo

Comportamento

Enabled: abilitato (True/False) a interagire con l'utente
TabStop: (True/False) il controllo riceve/non riceve il controllo tramite il tasto TAB
Tabindex: ordine di tabulazione del controllo (ordine con cui viene assegnato il focus (campo attivo)) all'interno del relativo contenitore; è possibile visualizzare l'ordine di tabulazione da menù: *Visualizza*(*View*) -> *Ordine di tabulazione* (*Tab Order*).
Visible: (True/False) il controllo è visibile/non visibile

Layout (dimensione e posizione)

Anchor: bordi del contenitore a cui il controllo è ancorato; la posizione del controllo rispetto ai bordi del contenitore non varia quando questi ultimi vengono spostati
Dock: bordo/i del controllo coincide/ono con bordo/i del contenitore;
Location: coordinate X e Y dell'angolo superiore sinistro del controllo rispetto all'angolo superiore sinistro del contenitore
Size: dimensione del controllo in altezza e larghezza, in pixel

Progettazione

Locked: (True/False) è/non è possibile spostare o ridimensionare il controllo
Name: nome del controllo

Metodi comuni

Focus: assegna il focus al controllo
Hide: nasconde il controllo
Show: mostra il controllo

Eventi comuni

Attivazione

GotFocus: il controllo riceve il focus
LostFocus: il controllo perde il focus

Tastiera e Mouse

Click, *DoubleClick*: click e doppio click del mouse
MouseDown: si preme un pulsante del mouse mentre il puntatore è posizionato sul componente
MouseEnter: il mouse entra nell'area del controllo,
MouseHover: il mouse resta fermo sul componente per un certo tempo
MouseLeave: il mouse abbandona l'area del controllo
MouseMove: il puntatore del mouse viene spostato sul controllo
MouseUp: viene rilasciato un pulsante del mouse mentre il puntatore è sul controllo
MouseWheel: viene spostata la rotellina del mouse mentre il punttaore è sul controllo
*KeyDown***:** viene generato quando un utente preme un tasto fisico
KeyPress: viene generato quando il tasto o i tasti premuti corrispondono a un carattere. Ad esempio, se un utente preme i tasti MAIUSC e "a" minuscola, il risultato sarà la lettera "A" maiuscola. Viene generato dopo *KeyPress*
KeyUp: viene generato quando un utente rilascia un tasto fisico

Aspetto

Move: si verifica quando il controllo viene spostato
Paint: viene generato quando il controllo viene ridisegnato
Resize: si verifica quando il controllo viene ridimensionato

Alcuni controlli

• Label

Proprietà

AutoSize: se True consente di ridimensionare la label in modo da contenere testo di dimensioni variabili (utile quando il testo subisce modifiche in fase di esecuzione)

• TextBox

Proprietà

AutoCompleteMode : modalità di autocompletamento. Possibilità di "suggerire" all'utente cosa digitare nel caso le prime lettere premute corrispondano all'inizio di una delle parole che il pr ogramma ha già elaborato.

Sono possibili quattro valori:

- *None* (assente)
- *Suggest* (viene suggerita la parola facendo apparire sotto la textbox un menù a discesa con tutte le possibili varianti)
- *Append* (viene suggerita la parola accodando alle lettere digitate il pezzo mancante evidenziato il blu)
- *AppendSuggest* (un'unione di Suggest e Append)

AutoCompleteSource : fonte dalla quale prelevare le parole dell'autocompletamento. Sono possibili i valori :

- risorse di sistema, quali la cronologia (HistoryList, nel caso, ad esempio, la textbox funga da contenitore di indirizzi internet)
- le cartelle (FileSystemDirectories, ad esempio, per facilitare l'immissione di un percorso da tastiera)
- i file (FileSystem),
- i file o i programmi aperti di recente (RecentlyUsedList)
- oppure tutti questi insieme (AllSystemResources).

Se impostato su CustomSource, sarà la proprietà AutoCompleteCustomSource a determinare la fonte da cui attingere informazioni;

CharacterCasing : indica il casing delle lettere:

- None (tutte le lettere vengono lasciate così come sono)
- Upper (tutte le lettere sono convertite in maiuscole)
- Lower (tutte in minuscole);

Lines : restituisce un array di str inghe rappresentanti tutte le righe di testo della textbox , nel caso di una tex tbox Multiline;

MaxLength*:* numero massimo di caratteri che possono essere immessi nella casella: 0 = nessuna limitazione (il massimo è di 32.000 caratteri).

Multiline*:* se impostato a **True** permette di inserire fino a 64kB di testo (il default è **False** e consente di inserire fino a 2048 caratteri) .

PasswordChar: Restituisce o assegna il carattere che serve per mascherare il testo digitato nella TextBox.

ReadOnly : determina se l'utente può modificare il testo della textbox .

ScrollBars :Barre di scorrimento

- None (nessuna scrollbar)
- Vertical (solo ver ticale)
- Horizontal (solo orizzontale)

- Both (entrambe)

Metodi
AppendText(testo): aggiunge del testo (*testo*) a quello già eventualmente presente nella casella
Copy: copia il testo selezionato nella TextBox negli Appunti
Cut: taglia il testo selezionato nella TextBox e lo copia negli Appunti
Paste: incolla nella TextBox il testo presente negli Appunti

• ListBox

Proprietà

FormatString : possibilità di formattare numeri e date
FormatEnabled : determina se è abilitata la formattazione degli elementi tramite FormatString
Integr alHeight : quando attiva, questa proprietà forza la lista ad assumere un valore di altezza (Size.Height) che sia un multiplo di ItemHeight, in modo tale che gli elementi siano sempre visibili interamente. Se disattivata, gli elementi possono anche venire "tagliati" : sono visualizzati parzialmente
Items : insieme delle voci contenute nella listBox
Items.Count: numero delle voci presenti nell'elenco
Items.Item(index): restituisce o assegna un valore alla voce nella posizione index
MultiColumn: se vale *True* gli elementi sono visualizzati su più colonne
SelectedIndex: restituisce un valore Integer che corrisponde alla posizione dell'elemento selezionato nella ListBox (*nota*: il conteggio delle voci inizia da 0):
se non è selezionata alcuna voce restituisce **-1**
se è selezionata la prima voce il valore è impostato a **0**
se è selezionata la seconda voce il valore è impostato a **1**
. .....
SelectedItem: restituisce la voce stessa (solitamente un valore stringa)
Sorted: determina se l'elenco viene ordinato

Metodi

***Items.Add(value)* :** aggiunge una voce (*value*) all'elenco
***Items.Clear()* :** cancella tutte le voci dall'elenco
Items.Insert (index, value): aggiunge una voce (*value*) all' elenco nella posizione (*index*)
Items.Remove (value): cancella una voce (*value*) dall elenco
Items.RemoveAt (index): cancella dall'elenco la voce (*value*) nella posizione (*index*)

Esempio:

```
ListBox1.Items.Clear()
ListBox1.Items.Add("pippo")
ListBox1.Items.Add("pluto")
ListBox1.Items.Insert(1, "carlo")
ListBox1.Items.Remove("gino")
ListBox1.Items.RemoveAt(2)
```

• ComboBox

Molti metodi e proprietà sono uguali a quelli di ListBox
Altre proprietà
AutoComplete... : come TextBox
DropDownHeight : determina l'altezza, in pixel, della casella di riepilogo a discesa

DropDownStyle : determina l'aspetto e la funzionalità della casella combinata.
Sono possibili tre valori :

- Simple : la ComboBox può essere assimilata a una ListBox; l'elenco a discesa è sempre visibile
- DropDown : stile normale con la possibilità di modificare il testo dell'elemento selezionato scrivendo entro la casella
- DropDownList : stile normale ma non è possibile modificare il testo dell'elemento selezionato

FlatStyle : determina la visualizzazione del controllo.
Può assumere quattro valori:

- Flat o Popup: la ComboBox è grigia e schiacciata, senza contorni 3D
- System o Professional: la ComboBox è azzurra e rilevata, con contorni 3D

MaxDropDownItems : il numero massimo di elementi visualizzabili nell'elenco a discesa
MaxLength : determina il massimo numero di caratteri di testo che possono essere inseriti nella casella della ComboBox . Questa proprietà ha senso solo se DropDownStyle non è impostata su DropDownList, poichè tale stile impedisce di modificare il contenuto della ComboBox tramite tastiera.

• CheckBox

Casella di controllo che si può segnare con un segno di spunta (tick)
Proprietà
Appearance : determina l'aspetto. Due valori:

- Normal: casellina di spunta con testo a fianco;
- Button: visualizzata come un bottone; se selezionata il bottone appare premuto

AutoCheck : determina se la checkbox cambia automaticamente stato (ossia da spuntata a non spuntata e viceversa) quando viene cliccata. Se False, l'unico modo per cambiare stato è tramite codice
AutoEllipsis : se True possibilità di visualizzare il testo che eventualmente con può essere visualizzato sul controllo
CheckAlign : se Appear ance = Normal, posizione della casellina di spunta
Checked : indica se la CheckBox è spuntata oppure no
CheckState : per le checkbox a tre stati, indica lo stato corrente (Checked,Unchecked,Indeterminate)
FlatStyle : aspetto del controllo come nella ComboBox
TextAlign : allineamento del testo
TextImageRelation : se presente un'immagine posizione del testo relativamente all'immagine
ThreeState : determina se la checkbox supporta i tre stati (Checked,Unchecked,Indeterminate) anzichè due

• RadioButton

Solo un RadioButton può esser e spuntato in un dato contenitore. Ad esempio,in una finestra che contenga tre di questi controlli, spuntando il primo, sarà impossibile spuntare il secondo e il terzo. Gode di tutte le propr ietà di CheckBox , tranne ovviamente ThreeState e CheckState, e rappr esenta visualmente il legame Xor tra più condizioni.

• GroupBox

Tra tutti i contenitori disponibili, GroupBox è il più semplice. La sua funzione consiste unicamente nel raggruppare in uno spazio solo più controlli uniti da un qualche legame logico.

I menu

Controllo MenuStrip

Consente di raggruppare i comandi di un'applicazione fornendo un facile accesso ad essi. È una gerarchia di selezioni: associata a ogni voce principale si trova una lista di opzioni, ciascuna delle quali può, a sua volta contenere un'altra lista di opzioni e così via.
Quando l'utente fa click su una voce principale del menu (per esempio la voce File, in un classico menù di un programma Office) viene visualizzato un elenco delle operazioni effettuabili, selezionando una di queste si genera un evento Click : il codice della sub relativa a questo evento si scrive in modo del tutto analogo a quello dell evento Click di un controllo Button. Non si scrive quindi codice per gestire l'evento Click sulle voci principali del menu, in quanto automaticamente viene aperto il sottomenu, a meno che non si voglia associare al Click su una voce principale un comportamento diverso.

Tasto di accesso : Alt+lettera sottolineata della voce di menù : è utile per muoversi tra le varie voci del menù con la tastiera

Tasti di scelta rapida (shortcutKeys) : Ctrl+simbolo(alfanumerico o tasto funzione) : sono utili per selezionare la voce anche senza visualizzare il menù contenente la voce

Segni di spunta : indica un elemento attualmente attivo

Separatore : linea di separazione tra le varie voci di un menù : (- nella modifica della voce di menu o selezionere in
ModificaDropDownItems -> Aggiungi -> Separatore

Menù standard con le voci : File Modifica Strumenti ?
Tasto destro : **Inserisci elementi Standard**

Modifica voci di menù

- menù principale: tasto destro su barra dei menu → modifica elementi
- menù secondari: tasto destro su voce principale → modifica DropDownItems

Proprietà principali:
Aspetto

- **Checked**: quando vale **True** si visualizza un segno di spunta a sinistra della voce di menù selezionata, per indicare se l'opzione è stata attivata o meno
- **Image:** permette di associare un'immagine alla voce di menù
- **Text**: testo visualizzato per la voce di menù (la lettera che identifica il tasto di accesso deve essere preceduta da **&**)

Progettazione

- **(Name)** : nome della voce di nome

Varie

- **ShortcutKeys :** tasto d scelta rapida
- **ShowShortcutKeys**: (True/Fale) visualizza/non visualizza il tasto di scelta rapida

Controllo ToolStrip

cosente di creare una barra degli strumenti; come già per controllo MenuStrip è possibile:

- inserire elementi standard
  tasto destro su barra degli strumenti → **Inserisci elementi Standard**
- modificare le voci
  tasto destro su barra degli strumenti → modifica elementi
  oppure
  tasto destro su barra degli strumenti → proprietà → items

Controllo ContextMenuStrip

Si può creare solo un menù di primo livello;si possono inserire/modificare/eliminare le voci in modo analogo a come si fa in un MenuStrip. Per associarlo a un controllo si deve selezionare il controllo e impostare la sua proprietà **ContextMenu** sul menu contestuale: ogni volta che si farà click con il tasto desto del mouse sul controllo verrà visualizzato il menù contestuale.
Un menù contestuale può essere associato a più controlli: nell'esempio che segue lo stesso menù contestuale è associato alle due caselle di testo TxtTesto e TxtTesto1; per sapere quale dei due controlli ha richiamato il menù contestuale si fa ricorso al test

```
If ActiveControl.Name = TxtTesto.Name Then
```

```
Private Sub mnuContCopia_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
mnuContCopia.Click
        If ActiveControl.Name = TxtTesto.Name Then
            TxtTesto.Copy()
        Else
            Txttesto1.Copy()
        End If
    End Sub

Private Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
        TxtPaste.Text = ""
        TxtPaste.Paste()
    End Sub
```

Finestre di dialogo

MessageBox

Permette di visualizzare dei messaggi e di compiere delle scelte
Può essere personalizzata utilizzando una grande varietà di icone e pulsanti; da non confondere con l'istruzione MsgBox.
Le varie forme sono:

risposta = MessageBox.show(*testo*)
risposta = MessageBox.show(*testo, titolo*)
risposta = MessageBox.show(*testo, titolo, pulsanti*)
risposta = MessageBox.show(*testo, titolo, pulsanti, icona*)
risposta = MessageBox.show(*testo, titolo, pulsanti, icona, pulsante di default*)
risposta = MessageBox.show(*testo, titolo, pulsanti, icona, pulsante di default, altre opzioni*)

testo(obbligatorio): testo visualizzato
titolo: testo visualizzato nella barra del titolo
pulsanti: quale tra i possibili pulsanti deve essere visualizzato, se manca viene visualizzato il pulsante OK
icona: quale icona deve essere visualizzata
pulsante di default: quale tra gli eventuali pulsanti visualizzati è il pulsante predefinito
altre opzioni: allineamento testo, dove visualizzare la finestra,….
risposta: valore restituito; può essere un intero o del tipo enumerativo DialogResult; restituisce il valore corrispondente al bottone premuto:

Abort	**3**
Cancel	**2**
Ignore	**5**
No	**7**
None	**0**
OK	**1**
Retry	**4**
Yes	**6**

Esempio

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles Button1.Click
Dim risposta As Integer
' con l'istruzione seguente viene visualizzato il messaggio indicato, la
finestra di messaggio avrà il nome "Uscita, saranno presenti due pulsanti: Sì e
No, un'icona con il punto interrogativo e il secondo pulsante (No) sarà quello
predefinito
risposta = MessageBox.Show("Vuoi veramente uscire?", "Uscita", _
MessageBoxButtons.YesNo, MessageBoxIcon.Question, _
MessageBoxDefaultButton.Button2)
If risposta = 6 Then ' risposta = 6 equivale a Si
End
Else                    'risposta = 7 equivale a No
MessageBox.Show("Uscita annullata dall'utente.", "Annullato", _
MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
End Sub
```

Le seguenti finestre di dialogo possono essere utilizzate inserendole in una form dalla casella degli strumenti come un qualsiasi altro controllo.

OpenFileDialog

Finestra per aprire un file; alcune proprietà e qualche metodo
Proprietà

DefaultExt: estensione predefinita per i file
FileName: (di sola lettura) restituisce il nome del file selezionato
Filter: stringa di filtro del nome file: opzioni che compaiono nella casella Files of type; ad es. : "Text files (*.text) | **.txt | Tutti i file (*.*) | *.*"**
FilterIndex: l'indice del filtro attualmente selezionato
InitialDirectory: directory di partenza
Title: titolo

Metodi:

ShowDialog: visualizza la finestra di dialogo

SaveFileDialog

Finestra per salvare un file; ricalca **OpenFileDialog**
Altre proprietà

OverWritePrompt: avvertimento se l'utente specifica il nome di un file che già esiste

FontDialog

Permette di selezionare un font o di visualizzare i font installati
Proprietà

Color: colore del carattere selezionato
Font: carattere selezionato
MaxSize e **MinSize**: dimensioni max e min, in pixel, selezionabili da un utente
ShowColor: indica se la finestra visualizza una scelta di colore
ShowEffects: indica se la finestra visualizza una scelta degli effetti (barrato, sottolineato)

Metodi:

ShowDialog: visualizza la finestra di dialogo

ColorDialog

Permette di selezionare un colore.
Proprietà

Color: colore selezionato

Metodi:

ShowDialog: visualizza la finestra di dialogo

FolderBrowserDialog

Permette di selezionare una cartella anziché un file.
Proprietà

SelectedPath : cartella scelta dall'utente

Metodi:

ShowDialog: visualizza la finestra di dialogo

Il metodo **ShowDialog,** comune a tutte le finestre di dialogo, restituisce un valore che rappresenta il tasto premuto dall'utente; il valore restituito, come già detto per la MessageBox, può essere un intero o del tipo enumerativo DialogResult; i valori che può assumere sono :

Ok	1
Cancel	2

Negli esempi *risposta* è una variabile che contiene il valore restituito dal metodo **ShowDialog**

es. (risposta dichiarata integer)

```
dim risposta as integer
risposta =  FontDialog1.ShowDialog()
if risposta = 1 then
       msgBox (" applicare il font")
       label1.font = fontDialog1.font
endif
if risposta = 2 then
       msgBox (" NON applicare il font")
endif
```

es. (risposta dichiarata DialogResult)

```
 Dim risposta As DialogResult
 If risposta = DialogResult.OK Then
           MsgBox(" applicare il font")
           Label1.Font = FontDialog1.Font
 End If
 If risposta = DialogResult.Cancel Then
           MsgBox(" NON applicare il font")
 End If
```

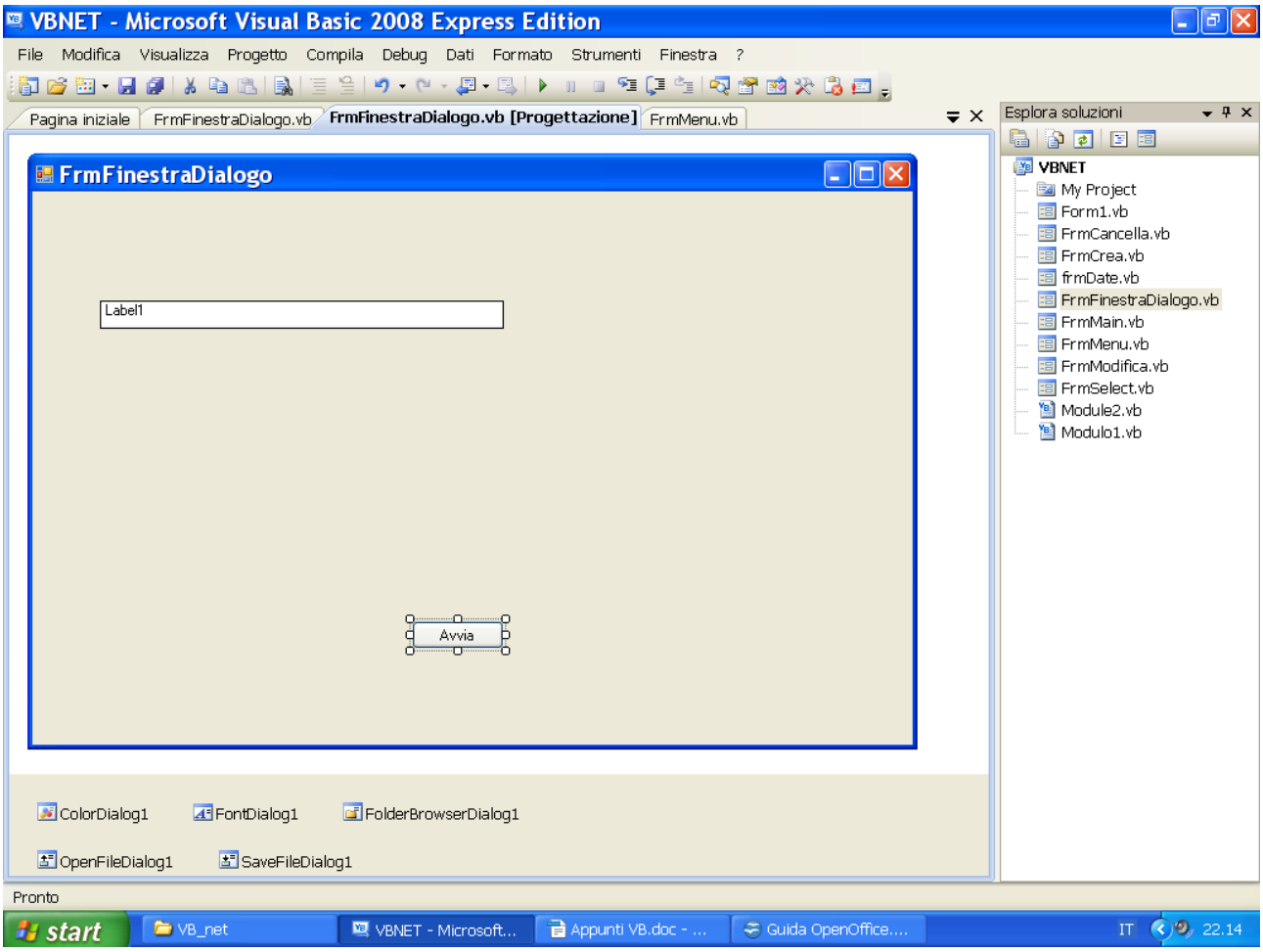
per le finestre di dialogo **OpenFileDialog** e **SaveFileDialog** permettono di selezionare il nome del file da aprire/salvare ma l'apertura e il salvataggio del file devono essere realizzate con opportune procedure:
Esempio : casella di testo multiline

```
 MsgBox("---- OpenFileDialog --------")
 risposta = OpenFileDialog1.ShowDialog()
 If risposta = DialogResult.OK Then
     Label1.Text = OpenFileDialog1.FileName
     AperturaFile(OpenFileDialog1.FileName)
 End If
 '
 MsgBox("---- SaveFileDialog --------")
 risposta = SaveFileDialog1.ShowDialog()
 If risposta = DialogResult.OK Then
     Label1.Text = SaveFileDialog1.FileName
     SaveFile(SaveFileDialog1.FileName)
 End If

Private Sub AperturaFile(ByVal nomeFile As String)
     Dim reader As System.IO.StreamReader = System.IO.File.OpenText(nomeFile)
     TextBox1.Text = reader.ReadToEnd
     reader.Close()
End Sub
Private Sub SaveFile(ByVal nomeFile As String)
     Dim writer As System.IO.StreamWriter = System.IO.File.CreateText(nomeFile)
     writer.Write(TextBox1.Text)
     writer.Close()
End Sub
```

Esercizi

1) Form con un controllo Label, un controllo Button per ogni finestra di dialogo, un controllo per ogni finestra di dialogo; visualizzare sulla label i file/directory aperte; cambiare il colore e il font della label in base al colore e al font prescelto.




```
Public Class FrmFinestraDialogo
Private Sub BtnAvvia_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
BtnAvvia.Click
        MsgBox("---- FolderBrowserDialog --------")
        FolderBrowserDialog1.ShowDialog()
        Label1.Text = FolderBrowserDialog1.SelectedPath
        '
        MsgBox("---- OpenFileDialog --------")
        OpenFileDialog1.ShowDialog()
        Label1.Text = OpenFileDialog1.FileName
        '
        MsgBox("---- FontDialog --------")
        FontDialog1.ShowDialog()
        Label1.Font = FontDialog1.Font
        '
        MsgBox("---- ColorDialog --------")
        ColorDialog1.ShowDialog()
        Label1.BackColor = ColorDialog1.Color
    End Sub
End Class
```

Strutture Dati

Array

```
Dim v(9) as integer
For i = 0 To 9
     x = x & " " & v(i)
     x = x & ControlChars.CrLf
Next
TextBox1.Text = x
```

Matrici

```
Dim m(3, 3) As Integer
For i = 0 To 3
   For j = 0 To 3
       x = x & " " & m(i, j)
   Next
   x = x & ControlChars.CrLf
Next
TextBox1.Text = x
```

Arraylist

E' una classe per la gestione di liste di elementi. All'inizio della sua "vita" non contiene elementi. Quando viene dichiarato un array

Dim A(100) As Integer

il programma riserva 101 celle di memoria, ciascuna di 4 byte, per memorizzare l'array. Quando viene dichiarato un ArrayList non vengono riservate celle per la memorizzazione degli elementi: questa memoria verrà occupata man mano che verranno aggiunti gli elementi.

```
Dim listaClienti As New ArrayList   'dichiarazione dell'ArrayList
listaClienti.Clear()                'pulisce l' ArrayList
listaClienti.Add(value)             'aggiunge un elemento(value)
listaClienti.Insert(index,value)    'aggiunge un elemento(value) nella posizione
'(index)
n = listaClienti.Count              'restituisce il numero di elementi presenti
'nell'ArrayList
value = listaClienti.Item(index)    'restituisce il dato memorizzato in
'posizione(index)
listaClienti.Remove(value)          'rimuove l'elemento (value)
listaClienti.RemoveAt(index)        'rimuove l'elemento in posizione(index)
index=listaClienti.IndexOf(value)   'restituisce l'indice dell'elemento(value)
x = listaClienti.Contains(value)          'restituisce TRUE se l'elemento(value) è
'contenuto nell'ArrayList, FALSE altrimenti
```

Hashtable

L'Hashtable è simile all'ArrayList per quanto riguarda allocazione di memoria ma è concettualmente differente per quanto riguarda l'utilizzo. Nell'arryList gli elementi sono identificati da un indice come in Array, nell'Hashtable gli elemnti sono identificati da una chiave, quindi mentre nell'ArrayList si lavora con la coppia (index,value) nell'hashtable lavoriamo con la coppia (key,value) : qaundo si vuoe aggiungere / ritrovare una valore bisogna indicare la coppia (chiave, valore)

```
Dim indirizziMail As New Hashtable  'dichiarazione dell'Hashtable
indirizziMail.Add(key,value)        'aggiunge un nuovo elememto
```

```
indirizziMail.Clear()                ‘pulisce l’ Hashtable


x = indirizziMail.Contains(key)           ‘restituisce TRUE se la chiave(key) è
contenuta nell’hashtable, FALSE altrimenti
indirizziMail.Remove(key)            ‘rimuove l’elemento di chiave(key)

For each name as object in indirizziMail.Values       ‘stampa tutti i valori
      Console.WriteLine(name)                         ‘presenti nell’hashtable
Next name

For each chiave as object in indirizziMail.Keys       ‘stampa tutte le chiavi
      Console.WriteLine(chiave)                       ‘presenti nell’hashtable
Next name
```

SortedList

Si compor ta esattamente come un Hashtable, solo che gli elementi vengono mantenuti sempre in ordine secondo la chiave.

VB.NET e i FILE

Si considera solo la scrittura e la lettura di un file di testo.

Namespace : System.IO

```
Imports System.IO
Public nomeFile As String = "c:\agenda.txt"
```

Lettura

```
        Public Sub letturafile()
         Dim str as string
         Dim reader As StreamReader = File.OpenText(nomeFile)
         Do While Not (reader.EndOfStream)
             str = reader.ReadLine
                ……..
 Loop
         reader.Close()
    End Sub
```

Scrittura

```
    Public Sub scritturafile()
        Dim writer As StreamWriter = File.CreateText(nomeFile)
        Dim str As String

        str = "…….    "
        writer.WriteLine(str) ' - in questo caso il testo scritto viene chiuso da un carattere CRLF
                              ' - (ODOA)
         ……..

writer.Close()
    End Sub
```

Nell'esempio seguente si legge tutto il file(di testo) e si scrive un testo (completo) senza l'aggiunta di CRLF

```
  Private Sub AperturaFile(ByVal nomeFile As String)
        Dim reader As System.IO.StreamReader = System.IO.File.OpenText(nomeFile)
        TextBox1.Text = reader.ReadToEnd
        reader.Close()
    End Sub
    Private Sub SaveFile(ByVal nomeFile As String)
        Dim writer As System.IO.StreamWriter = System.IO.File.CreateText(nomeFile)
        writer.Write(TextBox1.Text)
        writer.Close()
    End Sub
```

VB.NET e i DATABASE

ADO.NET è la tecnologia utilizzata per accedere ai dati. Le classi di ADO.NET si trovano nel namespace **System.Data** il quale a sua volta contiene altri namespace specifici (i cosiddetti ***provider di dati***), da utilizzare a seconda della sorgente dati a cui ci si vuole connettere:

- **System.Data.OleDb**: utilizzato per connettersi a database Access
- **System.Data.SqlClient**: utilizzato per connettersi a database SQL Server
- **System.Data.OracleClient**: utilizzato per connettersi a database SQL Oracle
- **System.Data.Odbc**: utilizzato per connettersi a database per cui esiste un driver ODBC

Nel seguito si farà riferimento al provider **System.Data.SqlClient** e alla cosiddetta **modalità connessa**.
Per lavorare con un database bisogna fare i seguenti passi:

1) connettersi al database

- creare un oggetto **SqlConnection**
- impostare la ***stringa di connessione***
- invocare il metodo **Open**

```
Public connectionString As String
Public objConnection As SqlConnection
ConnectionString="server = MIMMO-B9F76DD30\SQLEXPRESS;database = pippo;Integrated Security=true;"
objConnection = New SqlConnection(connectionString)
        objConnection.Open()
```

2) fornire il comando per l'azione che si vuole fare sul database (inserire,modificare,cancellare,interrogare)

- creare l'oggetto **SqlCommand**
- associare all'oggetto **SqlCommand** (proprietà **Connection**)l'oggetto **SqlConnection** creato al punto 1)
- assegnare alla proprietà **CommandText** dell'oggetto **SqlCommand** la query (di azione o di selezione) da eseguire sul database
- la proprietà **CommandType** indica come interpretare la query impostata in **CommandText**; valori possibili: ***Text***, ***StoredProcedure***
- utilizzare uno dei metodi *Execute* dell'oggetto **SqlCommand** per fare eseguire la query :
- **ExecuteNonQuery**: permette di eseguire una query di azione (*Insert*, *Update*, *Delete*), restituisce il numero di record elaborati
- **ExecuteReader**: esegue una query di selezione; restituisce un oggetto **SqlDataReader** contenente le tuple estratte (*resultset*)
- **ExecuteScalar**: esegue una query di selezione, da utilizzare quando il risultato è un singolo valore come nelle funzioni di aggregazione *sum*, *count*,…
- l'oggetto **SqlDataReader** restituito dal metoto **ExecuteReader** ha diversi metodi e proprietà tra cui:
- **Close**: chiude l' oggetto e rilascia le risorse
- **GetName(*index)***: restituisce il nome della colonna di indice ***index***
- **GetValue(*index*)**: restituisce il valore contenunoto nella colonna di indice ***index***;
- **HasRows**: restituisce *True* se il resultset contiene delle righe, *False* nel caso sia vuoto
- **Read**: sposta sul record successivo: restituisce *False* se si è arrivati alla fine del resultset
- **oggetto(*index*)**: valore contenuto nella colonna di indice index

- **oggetto(“*nomeCampo*”)**:valore contenuto nella colonna “nomecampo”

Esempio di ExecuteReader

```
Dim cmd As New SqlCommand()
        Dim dr As SqlDataReader
        cmd.Connection = objConnection
               cmd.CommandType = CommandType.Text
               sql1 = " select * from sys.all_objects  where type_desc = 'USER_TABLE' order by name"
cmd.CommandText = sql1
 Try
            dr = cmd.ExecuteReader()
            If dr.HasRows Then
               Do While (dr.Read())
                    tabelle(i) = dr("name")
                    i = i + 1
                Loop
            End If
            GoTo fine0
         Catch err As SqlException
            str1 = ""
            str1 = String.Concat(err.Number, "   ", err.Message)
            MessageBox.Show(str1)
            str1 = ""
         End Try
    fine0:
dr.close
```

Esempio di ExecuteNonQuery

```
objCommand = New SqlCommand()
       objCommand.Connection = objConnection
objCommand.CommandType = CommandType.Text
sql1 = "update ditte set data_inizio_campagna =  '" & datainizioca & _
               "', data_fine_campagna =  '" & datafineca & _
               "', data_modifica = '" & datainizioca & "'"

        objCommand.CommandText = sql1
        objCommand.ExecuteNonQuery()

 objCommand = Nothing
```

3) disconnettersi da database

```
objConnection.Close()
```

Il Debug e la gestione degli errori

Errori

- di sintassi . scrittura errata del nome di una variabile, di una istruzione, etc..
- errori di run-time (esecuzione): un divisore assume il valore 0, input di un valore errato, etc..
- errori logici:

Debug

Visualizza -> Elenco errori (View -> Error List)
Quando si verifica un errore viene visualizzata la finestra "Exception Assistant" che riporta un elenco di suggerimenti e azioni che possono essere intrapresi per capire/correggere l'errore; tra le azioni suggerite "visualizza Dettagli" permette di avere ulteriori dettagli sull'errore.
Breakpoint (punto di arresto)

Comando Debug

- esegui istruzione F8 (Step Into) : esegue il codice passo-passo
- esegui istruzione/routine MAIUSC+F8 (Step Over) : come Step Into ma passa "sopra" le Sub/Function
- esci da istruzione/routine CTRL+MAIUSC+F8 (Step Out): salta alla fine della Sub/Function che si sta eseguendo
- esegui fino al cursore CTRL+F8 (Run To Cursor) : posizionato il cursore su una linea successiva al punto di interruzione consente di eseguire le istruzioni comprese tra il breakpoint e il cursore.

Quando il programma si interrompe posizionando il mouse su una variabile viene visualizzato un ***data tip*** che consente di visualizzare ed eventualmente modificare il valore della variabile.
Finestre

Variabili locali : visualizza/modifica le variabili locali al modulo in esecuzione
Controllo immediato: è possibile visualizzare il valore di una variabile tramite il comando ?(print) nomeVariabile; in tale finestra è possibile assegnare alla variabile un nuovo valore

Gestione degli errori

Per gestione degli errori si deve intendere tutto quell'insieme di istruzioni che servono per intercettare l'errore e dare una risposta quando l'errore si verifica.

- **gestione non strutturata**: all'inizio del codice viene inserita l'istruzione ***on error goto*** tramite la quale al verificarsi di un errore (***on error***) rimanda (***goto***) a una porzione di codice che ha il compito di gestire i vari errori.

- **gestione strutturata**: si basa sul concetto di *eccezioni* che possono essere generate e che vengono *catturate* nella fase di gestione dell'errore; viene implementata tramite il blocco

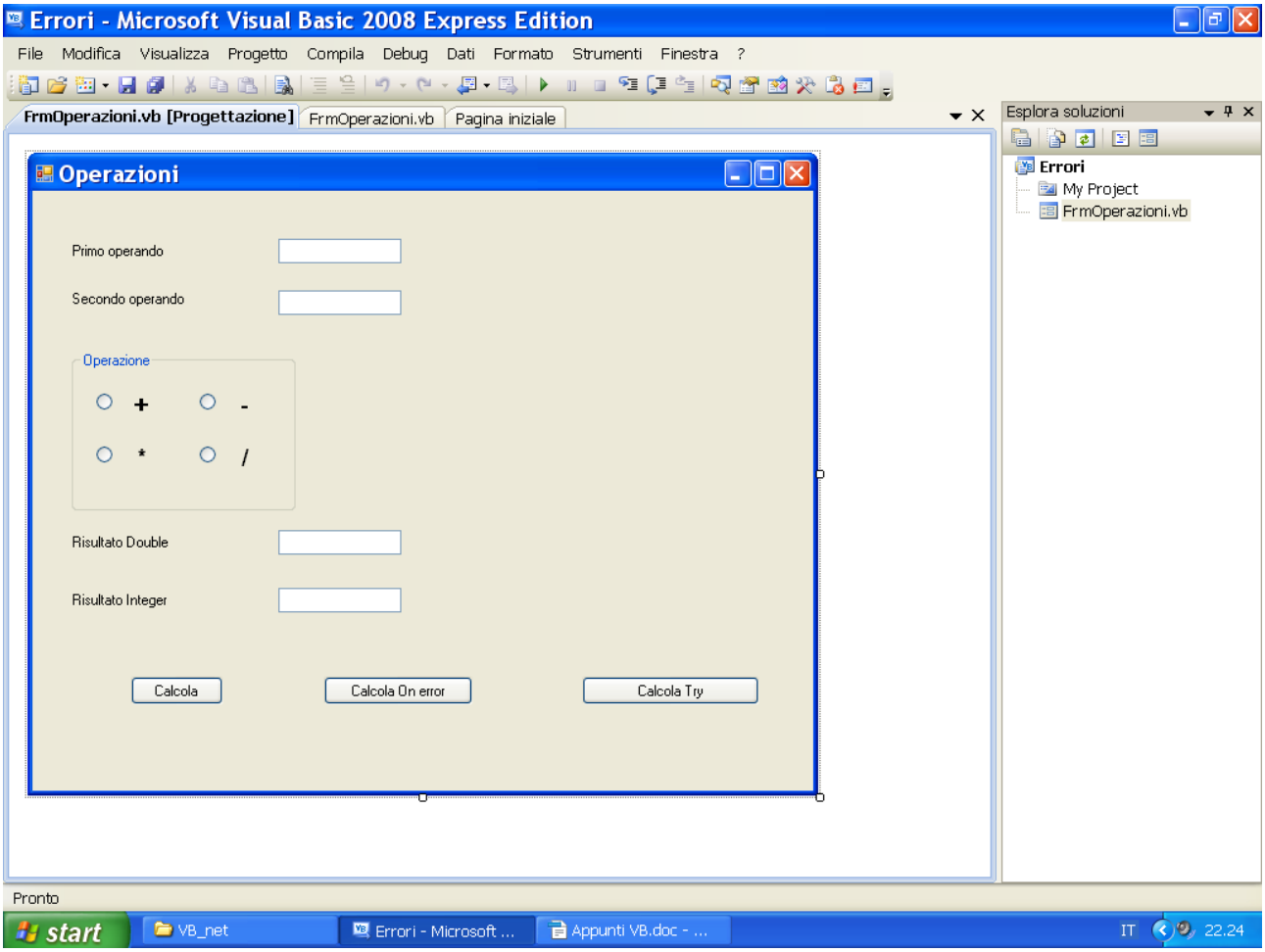
Try …….. ***Catch*** …….. ***Finally***

Try

Blocco di istruzioni in cui potrebbe essere generata un'eccezione

Catch variabileEccezione ***As*** tipoEccezione
Istruzioni specifiche per questa eccezione
Catch …… ***As*** ……..
Istruzioni specifiche per questa eccezione
……
Finally
Istruzioni che vengono eseguite sempre, anche se non si verifica alcuna eccezione.Molte volte il blocco Finally viene omesso.
End Try

All 'interno di un blocco ***Try ….. End Try*** si possono avere più ***Catch … As*** ognuno dedicato a una specifica eccezione : è importante l'ordine in cui sono disposti: dapprima devono trovarsi quelli relativi ad eccezioni specifiche e poi quelle più generiche; l'ultima ***Catch*** dovrebbe catturare ***System.Exception*** che cattura qualsiasi tipo di eccezione.



In questo esempio vengono intercettati, in modalità On Error e tramite Try, i due errori:

- I due operandi non vengono forniti
- Il secondo operando è 0 quindi si ha un errore nella divisione

```
Public Class FrmOperazioni
Private Sub btnCalcola_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCalcola.Click
        Dim op1 As Double
        Dim op2 As Double
        Dim risultato As Double
        Dim risIntero As Integer
        Dim operazione As Integer
        op1 = CDbl(txtPrimo.Text)
        op2 = CDbl(txtSecondo.Text)
        If rdbAddiziona.Checked Then operazione = 1
        If rdbSottrae.Checked Then operazione = 2
        If rdbMoltiplica.Checked Then operazione = 3
        If rdbDivide.Checked Then operazione = 4
        Select Case operazione
            Case 1
                risultato = op1 + op2
            Case 2
                risultato = op1 - op2
            Case 3
                risultato = op1 * op2
            Case 4
                risultato = op1 / op2
                risIntero = op1 / op2
        End Select
        txtRisultato.Text = risultato
        txtRisultatoInteger.Text = risIntero
    End Sub
-----------------------------------------------------------------------------------------
Private Sub btnOnError_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnOnError.Click
        Dim op1 As Double
        Dim op2 As Double
        Dim risultato As Double
        Dim risIntero As Integer
        Dim operazione As Integer
        On Error GoTo gestErr
Inizio:
        op1 = CDbl(txtPrimo.Text)
        op2 = CDbl(txtSecondo.Text)
        If rdbAddiziona.Checked Then operazione = 1
        If rdbSottrae.Checked Then operazione = 2
        If rdbMoltiplica.Checked Then operazione = 3
        If rdbDivide.Checked Then operazione = 4
Ricalcola:
        Select Case operazione
            Case 1
                risultato = op1 + op2
            Case 2
                risultato = op1 - op2
            Case 3
                risultato = op1 * op2
            Case 4
                risultato = op1 / op2
                risIntero = op1 / op2
        End Select
        '
        txtRisultato.Text = risultato
        txtRisultatoInteger.Text = risIntero
        Exit Sub
gestErr:
        MsgBox(Err.Number & "  " & Err.Description)
        If Err.Number = 13 Then
            txtPrimo.Text = ""
            txtSecondo.Text = ""
            txtPrimo.Focus()
            Exit Sub
        End If
        '
        If Err.Number = 6 Then op2 = 1 : Resume Ricalcola
        '
```

```
        Exit Sub
    End Sub
-------------------------------------------------------------------------------------------------
Private Sub BtnTry_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles BtnTry.Click
        Dim op1 As Double
        Dim op2 As Double
        Dim risultato As Double
        Dim risIntero As Integer
        Dim operazione As Integer
        Try
            op1 = CDbl(txtPrimo.Text)
            op2 = CDbl(txtSecondo.Text)
            If rdbAddiziona.Checked Then operazione = 1
            If rdbSottrae.Checked Then operazione = 2
            If rdbMoltiplica.Checked Then operazione = 3
            If rdbDivide.Checked Then operazione = 4
            Select Case operazione
                Case 1
                    risultato = op1 + op2
                Case 2
                    risultato = op1 - op2
                Case 3
                    risultato = op1 * op2
                Case 4
                    risultato = op1 / op2
                    risIntero = op1 / op2
            End Select
            txtRisultato.Text = risultato
            txtRisultatoInteger.Text = risIntero
            '
        Catch eccezione As InvalidCastException
            MsgBox(eccezione.Message)
            txtPrimo.Text = ""
            txtSecondo.Text = ""
            txtPrimo.Focus()
        Catch eccezione As OverflowException
            MsgBox(eccezione.Message)
            op2 = 1
            risultato = op1 / op2
            risIntero = op1 / op2
            txtRisultato.Text = risultato
            txtRisultatoInteger.Text = risIntero
        Finally
            MsgBox("BLOCCO Finally : questo blocco viene sempre eseguito")
        End Try
    End Sub
End Class
```

Bibliografia

Thearon Willis,Bryan Newsome – Visual Basic 2005 Guida per lo sviluppatore – Hoepli Informatica – 2006

Luigi Buono – VB.NET 2008 – Edizioni Master – 2008

SITOGRAFIA

Sepelang - INTRODUCTION TO VB.NET MANUAL – 2008 : con Google cercare “VB.Net Main Training Manua Sepelangl”

Guida VB.NET v3 by TOTEM : http://totemslair.org/guide/vb.php

Glossario

Control (*controllo*): è uno strumento utilizzato per creare oggetti in un programma Visual Basic (nella maggior parte dei casi gli oggetti vengono creati in un form). Gli strumenti vengono selezionati nella barra degli strumenti (*Toolbox*) e utilizzati per disegnare oggetti con il mouse in un form. Normalmente i controlli servono a creare elementi dell interfaccia utente come pulsanti (**Button**), caselle di testo (**TextBox**), etichette (**Label**), ecc.

ControlChars.CrLf

IsNumeric

Keyword (*parola chiave*): è una parola riservata del linguaggio Visual Basic, riconosciuta dal compilatore Visual Basic, che esegue attività utili. Ad esempio la parola chiave **End** termina l'esecuzione del programma. Le parole chiave rappresentano i blocchi base per la creazione di istruzioni di programma. La maggior parte delle parole chiave sono Proprietà scritte con carattere blu nell editor di codice.

Instruction (*istruzione*): è una riga di codice di un programma Visual Basic. Le istruzioni di programma possono avere lunghezze diverse, ma seguono tutte le regole di sintassi definite e imposte dal compilatore di Visual Basic.

Method (*metodo*): è un istruzione speciale che esegue un azione o un servizio per un determinato oggetto in un programma. Nel codice la notazione per l utilizzo di un metodo è:
`Object.Method (Value)`
Esempio: `ListBox1.Items.Add ( Genny )`
`MessageBox.Show( Hello! ),`
`ListBox1 e MessageBox` sono oggetti, `Add e Show` sono metodi, `Genny` e `Hello!` sono i valori

Namespace (*spazio dei nomi*): è una libreria gerarchica di classi strutturata con un nome unico, come *System.Windows*. Per accedere alle classi e agli oggetti sottostanti in un namespace, è necessario inserire un istruzione **Imports** all inizio del codice.

Object (*oggetto*): è in elemento creato in un programma Visual Basic con un controllo nella barra degli strumenti. In Visual Basic il form stesso rappresenta un oggetto.Tecnicamente gli oggetti sono istanze di una classe che supporta proprietà, metodi ed eventi.

Option Explicit

Property (*proprietà*): è un valore o una caratteristica di un oggetto. Ad esempio l'oggetto *Button* possiede la property **Text** per specificare il testo visualizzato sul pulsante, oppure *Width* o *ForeColor* stanno rispettivamente ad indicare la larghezza e il colore di primo piano di un oggetto.
Per le proprietà principali di ogni controllo sono definiti dei valori di **default** (ossia dei valori validi in mancanza di indicazioni specifiche). I valori delle proprietà di un oggetto possono essere modificati:
in fase di progettazione del layout, usando la *finestra delle proprietà*
in fase di esecuzione del programma.
Nel codice il formato dell impostazione di una proprietà è:
`Object.Property = Value`
Esempio: `TextBox1.Text = Hello! ,`

`TextBox1` è l'oggetto, `Text` è la proprietà, `Hello!` è il valore

Subroutine evento: un insieme di istruzioni che vengono eseguite quando l'utente o l'applicazione stessa intervengono su un controllo presente nel layout o su un oggetto. Le subroutine evento di solito valutano e impostano le proprietà e utilizzano altre istruzioni per eseguire le attività del programma

Esempio: `Sub btnCalcola_Click (...) ...`

`btnCalcola` è il pulsante che quando viene selezionato con il mouse attiva la procedura evento `btnCalcola_Click`

Esercizi

Dividere un numero (dividendo) per un altro numero (divisore) fino a che il quoziente non è < 1

- 2 textBox : txtDividendo, txtDivisore
- 2 label : lblDividendo, lblDivisore (proprietà *Text* : Dividendo, Divisore)
- 1 ListBox: lstRisultati
- 1 Button : btnDividi

Private sub button……

```
Dim risultato as double
Dim dividendo as double
Dim divisore as double
Dividendo = CDbl(txtDividendo.text)
Divisore = CDbl(txtDivisore.text)
risultato = dividendo
Do while risultato > 1
    risultato = risultato/divisore
    lstRisultati.items.add( dividendo & " / " & Divisore & " = " &risultato
    dividendo = risultato
loop
```

Se divisore = 1 e dividendo >= 1 si ha un ciclo infinito allora si può avere un'uscita forzata mettendo la condizione

```
If divisore = 1 the exit Do
```

The following table identifies the predefined numeric format names. These may be used by name as the style argument for the Format function:

Format name Description
General Number, G, or g
 Displays number with no thousand separator.

Currency, C, or c
 Displays number with thousand separator, if appropriate; displays two digits to the right of the decimal separator. Output
 is based on system locale settings.

Fixed, F, or f
 Displays at least one digit to the left and two digits to the right of the decimal separator.

Standard, N, or n
 Displays number with thousand separator, at least one digit to the left and two digits to the right of the decimal separator.

Percent
 Displays number multiplied by 100 with a percent sign (%) appended immediately to the right; always displays two digits to the right of the decimal separator.

P, or p
 Displays number with thousandths separator multiplied by 100 with a percent sign (%) appended to the right and separated by a single space; always displays two digits to the right of the decimal separator.

Scientific
 Uses standard scientific notation, providing two significant digits.

E, or e
 Uses standard scientific notation, providing six significant digits.

D, or d
 Displays number as a string that contains the value of the number in Decimal (base 10) format. This option is supported for integral types (Byte, Short, Integer, Long) only.

X, or x
 Displays number as a string that contains the value of the number in Hexadecimal (base 16) format. This option is supported for integral types (Byte, Short, Integer, Long) only.

Yes/No

Displays No if number is 0; otherwise, displays Yes.

True/False
Displays False if number is 0; otherwise, displays True.

On/Off
Displays Off if number is 0; otherwise, displays On.

Smart Device Developer Notes
The Yes/No, True/False, and On/Off formats are not supported.

Requirements
Namespace: Microsoft.VisualBasic

Module: Strings

Assembly: Visual Basic Runtime Library (in Microsoft.VisualBasic.dll)

See Also
Reference
String Manipulation Summary
Conversion Summary
Format Function
Predefined Date/Time Formats (Format Function)
User-Defined Numeric Formats (Format Function)

To make a suggestion or report a bug about Help or another feature of this product, go to the feedback site.

INDICE