

Cisa 1, Arezzo 4-5 giugno 2005

# **Visual Basic .Net**

e

# **ADO .Net**

Massimo Piubelli

[massimo@piubelli.net](mailto:massimo@piubelli.net)

[www.piubelli.net](http://www.piubelli.net)

Il Framework	2
Prima novità: Completamente OOP (Object Oriented Programming)	3
Seconda novità: Applicazioni console	7
L'ambiente di sviluppo integrato VB/C#/ASP.Net	8
Nuove caratteristiche dell'ambiente di sviluppo	9
VB.Net, Le Variabili	10
Eccezioni:La gestione delle eccezioni strutturata	13
ADO.Net Introduzione	16
ADO.Net Oggetto Connection	16
ADO.Net Oggetto Command	18
ADO.Net Oggetto DataReader	19
ADO.Net Oggetto DataAdapter	20
ADO.Net Oggetto DataSet	22
Distribuire un'applicazione Windows	25

# Il Framework

Non pretendo di spiegare in due pagine che cosa sia il framework, per cui mi limiterò ad una brevissima descrizione, magari con un confronto forse non ottimale; pensiamo al framework.net come ad una Java Virtual Machine "evoluta", nel senso che il framework consente molte più cose che JVM non consente, ad esempio .net è multilinguaggio, quindi posso integrare in una applicazione codice VB e codice C#, ma veniamo nel dettaglio ad analizzare le sue principali caratteristiche che compongono il framework elencate nella seguente tabella che poi analizzerò partendo dalla base della stessa:

Visual Basic	C++	C#	Cobol.net	....
Common Language Specifications (CLS)				
Windows Forms		ASP.Net		
Data e XML				
Base Class Library (BLC)				
Common Language Runtime				
Windows API		COM+ Services		

## **Windows API e COM+ Services**

.Net mette a disposizione tutte le funzioni del sistema operativo, che non sono riscritte ma sono una serie di chiamate alle DLL di Windows, allo stesso livello troviamo anche COM+ che non è stato riscritto ma che ci offre la possibilità di utilizzare componenti precedentemente sviluppati con COM (Ado, Dao, Excel, Word), anche se questa tecnica ovviamente comporta un certo appesantimento delle prestazioni in quanto si parla di codice "non gestito"

## **Common Language Runtime(gestione memoria, debug, eccezioni, JIT compiler)**

Il CLR è il cuore del framework, in quanto si preoccupa di quasi tutta la gestione dell'applicazione, dalla compilazione JIT (Just in time) alla gestione delle eccezioni, alla gestione della memoria ecc.

## **Classi di base (tipi di dato, IO, string, collection, threading...)**

Questa parte del Framework gestisce tutti i tipi di dato partendo da Object (System.Object), passando dai numerici, alle date, ma queste classi sono quelle che ci consentono anche la gestione della scrittura su file (IO), la gestione delle stringhe (vedremo più avanti come le variabili stringa siano appunto delle classi), la gestione delle collection (insieme di oggetti, esempio controls, Items ecc.), la gestione dei thread (essendo che .net ha una discreta gestione del multithread)

### **Classi "Data" e "XML" (Ado.net, XML, XSLT, Xpath)**

Queste classi mi permettono di interagire con Ado.net e con i file XML (leggerli, scriverli e trasformarli con XSLT ad esempio)

Vedremo in seguito approfonditamente il funzionamento di Ado.net, purtroppo per problemi di tempo, non avremo tempo di accennare ad XML.

### **Windows Forms, ASP.Net**

Anche in questo caso, abbiamo all'interno del CLR tutto quello che ci serve per la gestione sia delle Windows Forms (compresi tutti i controlli) sia delle form web e dei web services (che rientrano in ASP.Net e non verranno trattati)

### **Common Language Specifications**

Queste sono le specifiche messe a disposizione da Microsoft per tutti i produttori di compilatori di linguaggi di programmazione, descrivono in pratica le caratteristiche minime che deve avere un linguaggio per essere conforme agli standard di .net

### **I linguaggi .net**

.Net, offre la possibilità di sviluppare in molti linguaggi anche all'interno di una stessa applicazione, quindi nel livello più alto troviamo appunto il linguaggio che verrà utilizzato dallo sviluppatore.

## **Prima novità: Completamente OOP (Object Oriented Programming)**

### **Proprietà, metodi ed eventi**

- Le proprietà sono le caratteristiche degli oggetti, es. Name, BackColor, Text; ci sono proprietà che si possono impostare e leggere sia a run-time che a design-time, mentre altre proprietà sono definite read only o write only
- I metodi sono le "azioni" che un oggetto è in grado di compiere, ad esempio per aprire una maschera uso il suo metodo Show (in vb.net non esiste, ma l'oggetto DoCmd di access è un oggetto che contiene solo metodi. Alcuni metodi possono ricevere dei parametri (Es, il metodo Add delle ListBox, vuole il valore da aggiungere), altri no (es. Show)
- Gli eventi sono situazioni che l'oggetto è in grado di riconoscere. In questo caso la sintassi è cambiata notevolmente, prendiamo l'esempio di una chiusura di una Form, in cui voglio annullare l'operazione se l'utente non conferma il messaggio che gli viene proposto
-

```

Private Sub FrmEventi_Closing(ByVal sender As Object, ByVal e As _
    System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
    'Dichiaro una variabile che contiene il risultato della funzione msgbox
    'in realtà l'enum che viene fatto è questo (visibile in object browser)
    'Public Enum MsgBoxResult As Integer

    Dim Risposta As MsgBoxResult

    Risposta = MessageBox.Show("Sei sicuro di voler chiudere", "Conferma chiusura",
    -
    MessageBoxButtons.YesNo, MessageBoxIcon.Question,
    MessageBoxDefaultButton.Button2)

    If Risposta = MsgBoxResult.No Then
        e.Cancel = True 'annullo la conseguenza dell'evento, in pratica la form non
si chiude
    End If

End Sub

```

### **Ereditarietà**

- E' una nuova caratteristica degli oggetti di Vb.Net, consente di creare una classe base, per poi ereditarne una più specializzata, esempio, una classe base potrebbe essere una classe "persona" tutti abbiamo delle proprietà (nome, cognome, data di nascita), dei metodi (cammina, mangia, dormi) e degli eventi (sento caldo, ho sete, ho sonno, ho fame) Posso quindi derivare da persona la classe impiegati, che in più ha una proprietà (stipendio) dei metodi (Lavora, LavoraTanto) e degli eventi (il Capo mi sta rimproverando)

```

Public Class CLConto
    Friend M_Saldo As Double 'il saldo non può essere scritto,
    'se non dalle classi derivate
    Dim M_Numero As Integer 'il numero di conto può essere impostato
    'solo quando creo l'oggetto
    Dim M_Correntista As String 'il nome e cognome del correntista

    Public Event SaldoInsufficiente()

    Public Property Correntista() As String
        Get
            Return M_Correntista
        End Get
        Set(ByVal Value As String)
            M_Correntista = Value
        End Set
    End Property

    Public ReadOnly Property Saldo()
        Get
            Return M_Saldo
        End Get
    End Property

    Public Overridable Sub Prelevamento(ByVal Importo As Double)
        'la parola chiave overridable verrà analizzata quando si
        'affronterà il tema del polimorfismo
        If Importo > M_Saldo Then
            M_Saldo -= Importo
        Else

```

```

        RaiseEvent SaldoInsufficiente()
    End If
End Sub

End Class

Public Class CLContoSportello
    Inherits CLConto

    Public Sub Versamento(ByVal Importo As Double)
        M_Saldo += Importo
    End Sub
End Class

```

### **Polimorfismo**

In Vb.Net è possibile usare questa tecnica di programmazione che consente di creare sub e function aventi lo stesso nome, ma richiamabili in modo diverso in fase di esecuzione.

Il polimorfismo può essere ottenuto come in VB6 tramite le interfacce (tecnica che non illustrerò) oppure mediante l'ereditarietà (più utilizzato in tecniche di programmazione ad oggetti)

```

Public Class CLContoBancomat
    Inherits CLConto
    Public Shadows Event SaldoInsufficiente()

    Public Overrides Sub Prelevamento(ByVal Importo As Double)
        If Importo > M_Saldo Then
            M_Saldo -= Importo
        Else
            RaiseEvent SaldoInsufficiente()
        End If
    End Sub
End Class

```

### **Overload ed overrides**

Con questa tecnica di programmazione, legate in qualche modo al polimorfismo posso ottenere due risultati:

- Con l'overload posso creare sub e function aventi lo stesso nome, ma con "Firme" diverse (ovvero il numero e/o il tipo di parametri passati in ingresso). In questo esempio due funzioni con lo stesso nome, nelle quali la seconda funzione richiama la prima, arrotondandone il risultato

```

Public Function ConvertiEuro(ByVal Importo As Integer) As Double
    Return Importo / 1936.27
End Function

Public Function ConvertiEuro(ByVal Importo As Integer, ByVal Decimali _
As Byte) As Double
    Return (Math.Round(ConvertiEuro(Importo), Decimali))
End Function

```

- Con l'overrides invece, posso ridefinire un metodo di una classe base, nell'esempio che esporrò, cerco di costruire una classe che consenta di fare ciò che usualmente viene fatto con Ms Access e le ListBox/ComboBox, ovvero inserire nella la colonna associata il campo chiave primaria, per poi visualizzare altre informazioni. Questa cosa la svilupperò creando una classe che ha due variabili pubbliche (per

semplificare non utilizzerò le proprietà, ma renderò direttamente visibili i membri), per poi fare l'overrides del metodo ToString (metodo che viene cercato quando inserisco un oggetto in una combo/lista) in questo modo ho la possibilità di inserire nella lista tutti i valori, in modo da poter risalire sia alla descrizione (che verrà visualizzata come insieme di cognome e nome), sia alla chiave primaria (che non verrà visualizzata ma sarà memorizzata essendo che la lista contiene tutto l'oggetto)

```
Public Class ClPersona
    Public Nome As String 'membri pubblici
    Public Cognome As String
    Public ID As Integer

    Public Overrides Function ToString() As String
        Return Cognome & " - " & Nome
    End Function
End Class
```

Qui sopra viene rappresentata la classe che contiene i dati, come si può notare viene fatto l'overrides della funzione ToString, facendo leggere i valori di Cognome e Nome insieme. Nelle due Sub Sottostanti, la possibilità di aggiungere i valori all'interno della listBox e la successiva visualizzazione nelle caselle di testo.

```
Private Sub CmdAggiungi_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles CmdAggiungi.Click
    Dim NuovoValore As New ClPersona
    'TODO: Non viene effettuato nessun controllo, il campo accetta solo interi
    NuovoValore.ID = Me.TxtID.Text
    NuovoValore.Cognome = Me.TxtCognome.Text
    NuovoValore.Nome = Me.TxtNome.Text

    LstNominativi.Items.Add(NuovoValore)
    Call PulisciCaselle() 'procedura che cicla sui controlli e li pulisce
End Sub
Private Sub CmdAggiungi_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles CmdAggiungi.Click
    Dim NuovoValore As New ClPersona
    'TODO: Non viene effettuato nessun controllo, il campo accetta solo interi
    NuovoValore.ID = Me.TxtID.Text
    NuovoValore.Cognome = Me.TxtCognome.Text
    NuovoValore.Nome = Me.TxtNome.Text

    LstNominativi.Items.Add(NuovoValore)
    Call PulisciCaselle() 'procedura che cicla sui controlli e li pulisce
End Sub
```

### **Costruttori ed Overload dei Costruttori**

Un'altra grossa novità è rappresentata dal supporto dei costruttori in vb.net, il costruttore è il "codice" che viene eseguito ogni volta che creo un'istanza di una classe

(Es: Dim MiaForm as **New** FrmProva)

In vb.net il costruttore è la sub New, di cui posso fare l'overload, per passare dei parametri particolari in fase di apertura della maschera

## Seconda novità: Applicazioni console

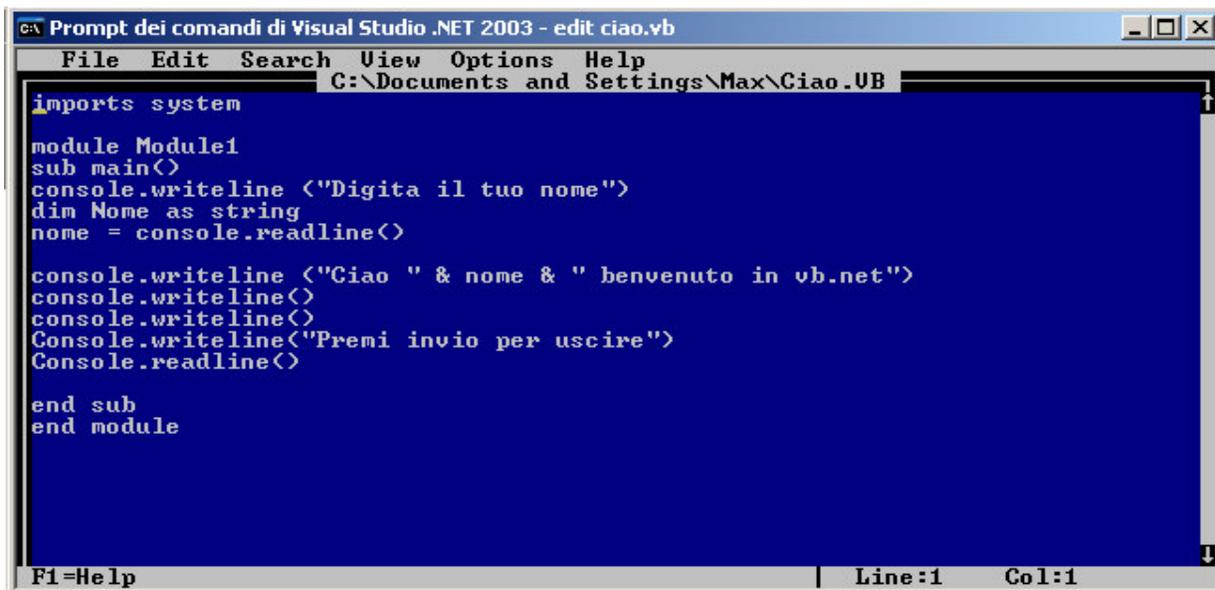
Non mi dilungherò molto sulle applicazioni console, in quanto non prevedo una loro fortissima espansione, tuttavia è bene sapere che .Net consente di realizzare completamente le applicazioni anche senza la presenza di Visual Studio, nell'esempio che riporto creo una semplice applicazione console, che chiede il nome dell'utente e quindi gli dà il benvenuto in vb.net.

Come potrete notare il listato è stato scritto con Edit di Dos, e compilato successivamente con l'istruzione

Vbc ciao.vb

Vbc è il compilatore di Visual Basic.Net e in questo caso ha creato un'applicazione console.

Lancio il programma da prompt di dos direttamente scrivendo Ciao.exe (è facoltativo il .exe)

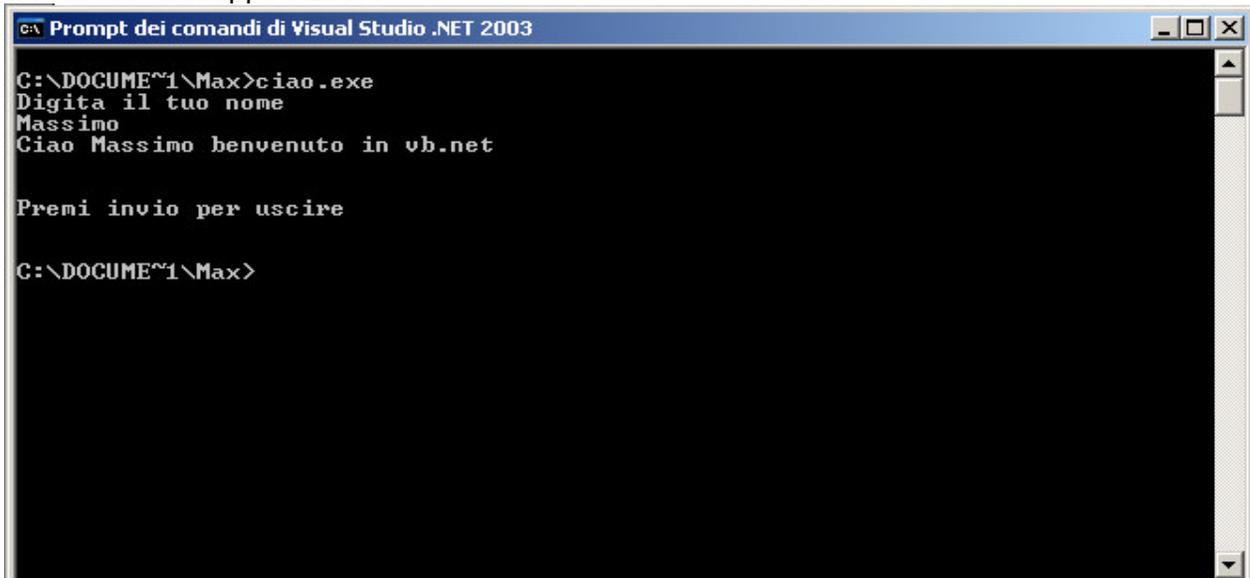


```
g:\ Prompt dei comandi di Visual Studio .NET 2003 - edit ciao.vb
File Edit Search View Options Help
C:\Documents and Settings\Max\Ciao.VB
imports system
module Module1
sub main()
console.writeline ("Digita il tuo nome")
dim Nome as string
nome = console.readline()

console.writeline ("Ciao " & nome & " benvenuto in vb.net")
console.writeline()
console.writeline()
Console.writeline("Premi invio per uscire")
Console.readline()

end sub
end module
F1=Help | Line:1 Col:1
```

Il risultato dell'applicazione



```
g:\ Prompt dei comandi di Visual Studio .NET 2003
C:\DOCUMENTE~1\Max>ciao.exe
Digita il tuo nome
Massimo
Ciao Massimo benvenuto in vb.net

Premi invio per uscire

C:\DOCUMENTE~1\Max>
```

# L'ambiente di sviluppo integrato VB/C#/ASP.NET

## I File principali

- Sln (è il file della soluzione)
- \*.vb (sono i file che contengono il codice, l'estensione indica il linguaggio utilizzato)
- vbproj (è il file di progetto)

## Cos'è un assembly

- Un assembly è immaginabile come un eseguibile, composto da un manifest, da moduli di codice e da eventuali file di risorse (html, jpg ecc) è la parte più piccola per poter gestire il "versioning" di un'applicazione
- L'assembly è scritto in linguaggio MSIL (Microsoft Intermediate Language) ed è quindi compilato successivamente dal JIT Compiler.

## I NameSpaces, definizione

Gli spazi dei nomi evitano ambiguità e semplificano i riferimenti quando si utilizzano gruppi di oggetti di grandi dimensioni, come le librerie di classi

## NameSpaces, creazione ed importazione

Nell'esempio viene costruito un Namespace NS1 che contiene un ulteriore Namespace NS2, il quale definisce una classe chiamata "ClasseA"

```
Namespace NS1    ' Dichiaro un namespace chiamato NS1
  Namespace NS2    ' Dichiaro un namespace chiamato N2 dentro N1.
    Class ClasseA    ' Dichiaro una classe "A" all'interno del NS1.NS2.
      Private M_NomeProprieta As String
      'codice della classe
      Public Sub ProcaSub()

      End Sub

      Public Property NomeProprieta() As String
      Get
        Return M_NomeProprieta
      End Get
      Set (ByVal Value As String)
        M_NomeProprieta = Value
      End Set
    End Property
  End Class
End Namespace
End NamespaceEnd Class
End Namespace
End Namespace
```

Per poter utilizzare la classeA, la dichiarazione della variabile (ipotizzando che non venga effettuato l'imports sarà:

```
Dim MiaVar As New NS1.NS2.ClasseA
```

## Nuove caratteristiche dell'ambiente di sviluppo

### Ambiente: Task List

Penso che la task List sia una delle aggiunte più interessanti che sono state inserite in vb.net, automaticamente la task list viene popolata con la segnalazione degli errori di sintassi, ma posso anche aggiungere manualmente delle attività da fare o posso fare in modo che riconosca determinati commenti che inserisco nel codice per aggiungerli in automatico.

- Permette cliccando su un elemento, di posizionarsi subito nella corrispondente linea di codice
- Vengono aggiunti automaticamente gli errori di sintassi
- Si può utilizzare anche come lista di attività da svolgere (es: todo)

The screenshot displays the Visual Studio IDE. The top portion shows the code editor with three subroutines: `CmdAggiungi_Click`, `PulisciCaselle`, and `FrmOverrides_Load`. The `CmdAggiungi_Click` subroutine includes a green comment: `'TODO: Non viene effettuato nessun controllo, il campo accetta solo interi`. Below the code, the Task List window is visible, titled "elenco attività - 2 attività". It contains a table with the following data:

<input checked="" type="checkbox"/>	Descrizione	File	Riga
<input type="checkbox"/>	Fare clic qui per aggiungere una nuova attività		
	Impossibile convertire il valore di tipo "System.Windows.Forms.TextBox" in "St D:\Formazione\VbNet\CisaNet\FrmOverrides.vb	D:\Formazione\VbNet\CisaNet\FrmOverrides.vb	147
	TODO: Non viene effettuato nessun controllo, il campo accetta solo interi	D:\Formazione\VbNet\CisaNet\FrmOverrides.vb	145

At the bottom of the Task List window, there are several icons and labels:  Elenco attività,  Finestra di comando,  Output,  Risultati ricerca simbolo, and  Punti di interruzione.

### **Ambiente: Help Dinamico**

L'help dinamico ricerca automaticamente i collegamenti della guida in linea man mano che digito nella finestra di codice. Può risultare utile per un veloce reperimento di informazioni (premesso che la guida in linea è sensibile come in vba, quindi se premo F1 sopra una determinata voce viene automaticamente ricercato l'argomento); anche se, personalmente, mi da un certo fastidio vedere il suo continuo ricercare gli argomenti man mano che sto digitando il codice

### **Ambiente: Finestra di controllo e Immed - Debug.WriteLine**

Come in VBA, posso interrogare le variabili di un'applicazione, oppure impostarne il loro valore, tuttavia la finestra "immediata" è leggermente cambiata come funzionalità, in pratica alle vecchie possibilità, ora ne vengono aggiunte di nuove, in modalità comando, posso ad esempio dare istruzioni anche all'ambiente (es: avviare o stoppare il debug)

I comandi Debug. Sono più ampi, posso usare il debug.WriteLine (corrispondente dell'ex debug.print) un Debug.WriteIF (che consente di scrivere sulla output windows solo se una certa condizione si verifica ecc.)

## **VB.Net, Le Variabili**

### **Variabili: Tipi di dato Differenze VBA/VB6 - Vb.Net**

Una delle più grosse novità con Vb.Net è che tutte le variabili vengono trattate come oggetti, nella tabella qui sotto riporto le principali differenze tra i tipi di dato utilizzabili in VBA VB6 con VB.Net

Alcune altre considerazioni sulle variabili:

- Tutte le variabili sono oggetti, per cui hanno anche metodi e proprietà (es. per le variabili numeriche un MaxValue e un MinValue)
- Date memorizzate in un campo specifico
- Possibilità di assegnare un valore nel momento della dichiarazione

### **Variabili: Variabili Value-Type e Reference-Type Considerazioni**

- Le variabili reference-type puntano ad un'area di memoria, mentre le variabili value-type contengono una copia del valore
- Se una variabile non è dichiarata (con option explicit OFF) o se non è specificato il suo tipo di dato, le viene assegnato il tipo di dato Object, e ciò rallenta l'esecuzione del programma, oltre che non consentire al compilatore la verifica dei tipi
- Se non ho bisogno di decimali in variabili numeriche, le variabili di tipo integer sono le migliori.

### **Riepilogo tipi di dato:**

<b>Vb.Net</b>	<b>B</b>	<b>VB6 / VBA</b>	<b>Intervallo valori</b>
Boolean (System.Boolean)	2	Boolean	True / False
Byte (System.Byte)	1	Byte	Da 0 a 255
Char (System.Char)	2		Da 0 a 65535
Short (System.Int16)	2	Integer	Da -32.768 a + 32.767
Integer (System.Int32)	4	Long	Da -2 a +2 Miliardi ...
Long (System.Int64)	8		Da -9.223.372.036.854.775.808
Single (System.Single)	4	Single	Precisione Singola
Double (System.Double)	8	Double	Precisione Doppia
Decimal (System.Decimal)	16		
Date (System.DateTime)	8	Date	01.01.0001-31.12.9999
String (System.String) Classe	*	Sting	Circa 2 miliardi di caratteri
Object (System.Object) Classe	4	Object	Qualsiasi tipo
Strutture (Ereditate da System.ValueType)	*		Ciascun membro della struttura presente un suo intervallo
- non supportato		Variant	
- non supportato		Currency	

\* Dipende dalla piattaforma

### **Istruzione CType**

- Come Cstr, Cint, Cdbl, consente di convertire un valore da un tipo all'altro  
In realtà fa anche qualcosa di più, in quanto (come si può notare nell'esempio della listBox)

### **Variabili: Dichiarazione ed assegnazione**

Per quello che riguarda la dichiarazione delle variabili, le novità sono principalmente due: La prima è che nel momento della dichiarazione alle variabili può essere subito assegnato un valore.

### **Array: Descrizione di base**

Per quanto riguarda le matrici, ci sono alcune piccole differenze.  
Prima di tutto, le matrici sono tutte dichiarate su base 0 (non esiste più ne option base 1, ne il vecchio dim Matrice (5 to 10) as...)

### **Variabili: Scope e variabili di blocco**

Una novità viene introdotta con le variabili di blocco, che esistono solamente dentro specifici costrutti di programmazione

```
Private Sub CmdVarBlocco_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CmdVarBlocco.Click
    Dim i As Integer

    For i = 0 To 6
        Dim y As Integer

        y = y + 1 * 2
    Next

    y = y + 1 ' y qui non è più visibile e viene generato un errore di compilazione
End Sub
```

### **Strutture**

Le strutture sono paragonabili alle "vecchie" classi di Vb6 / VBA, all'interno di una struttura io posso creare sub e function, posso dichiarare variabili private e pubbliche, ma non ho possibilità di usare i costruttori, non approfondisco ulteriormente questa sezione in quanto superata dalla gestione delle classi

### **Operatori di assegnazione**

- Sono stati introdotti nuovi operatori di assegnazione +=, -=, \*= ecc.

### **Sub e Function (differenze)**

- Come in VB6/Vba, una sub non può restituire valori, una function si
- ByVal viene messo di default (permette di passare una "copia" di una variabile e non il suo diretto riferimento all'area di memoria come sarebbe stato con ByVal, default in VBA/VB6)
- Ora posso far ritornare ad una function anche un tipo, per cui più di un valore
- Istruzione Return (function) corrisponde a valorizzare la funzione ed ad interromperla

### **Sub e Function: Overload / Optional**

- Oltre che all'uso dei parametri opzionali (optional) posso anche effettuare overload delle sub/function, usando lo stesso nome ma con "firme" (=Parametri) diversi

### **Proprietà: Cambiamenti con le proprietà di default**

- Non posso più omettere il nome della proprietà di default
- In alcuni casi posso farlo, in questo caso ho quasi sicuramente delle collection e ometto il nome della proprietà (es. "items")

## **Eccezioni:La gestione delle eccezioni strutturata**

In vb.net viene ancora supportato il "vecchio" on error Goto / Resume, ma il suo utilizzo può essere sostituito dall'attuale gestione degli errori "strutturata", tale supporto non esiste ovviamente per gli altri linguaggi di programmazione .net (tipo c#) personalmente consiglio di abituarsi ad usare la nuova gestione strutturata delle eccezioni.

### **Try...Catch...Finally**

- Considerazione generale: gli errori (o meglio usare il termine eccezioni) sono "pesanti" in quanto a consumo di memoria, pertanto è bene (ove sia possibile) testare i valori che posso controllare al fine di ridurre le volte in cui dover utilizzare un gestore degli errori
- Questo è il nuovo blocco, di fatto viene utilizzata una variabile ("ex" nell'esempio) che contiene una istanza dell'eccezione che è stata scatenata
- Quindi, sentiremo parlare di "sollevare un'eccezione" che vuol dire in termini pratici che quando si verifica un'eccezione, viene passato al chiamante un oggetto Exception, del quale posso analizzarne proprietà, o invocarne i metodi, come potrei anche gestire tranquillamente la situazione senza notificare nulla all'utente, in questo esempio molto semplice, viene sollevata un'eccezione, ovvero verrà eseguito il codice dopo il try, e, se si solleva un'eccezione, il codice verrà passato al blocco Catch

```
Dim x As Integer
Dim y As Integer

Try
    x = 5 / y
    MessageBox.Show("il risultato è " & x)
Catch ex As Exception
    MessageBox.Show(ex.ToString)
End Try
```

## **Filtrare le eccezioni**

E' una tecnica che equivale al vecchio Select Case err.number, ovvero gestisco in maniera diversa ogni singola tipologia di errore. In realtà in presenza di un errore il codice viene spostato al primo blocco Catch, quindi viene valutato se la tipologia di errore è quella corretta, se è così il codice viene eseguito, altrimenti si passa al successivo blocco catch, fino a che non si trova un catch valido. Se non si trova un catch valido l'eccezione risulta non gestita.

L'ultimo blocco che ho inserito in questo esempio gestisce tutte le tipologie di eccezioni, è importante che questo catch venga inserito alla fine degli altri blocchi, altrimenti verrebbe sicuramente eseguito facendo diminuire l'efficacia del filtro

```
Dim x As Integer
Dim y As Integer

Try
    x = 5 / y
Catch ex As OverflowException
    'esegue questa istruzione nel caso si stia verificando un overflow
    MessageBox.Show("Attenzione, si è verificato un overflow")
Catch ex As Exception
    'esegue questa istruzione nel caso si stia verificando un'altra eccezione
    MessageBox.Show(ex.ToString)
End Try
```

## **L'istruzione Finally**

Può essere inserita come no, serve nel caso abbia bisogno di eseguire del codice sia che ci sia, sia che non ci sia stato un errore, per eseguire per esempio una chiusura di una connessione o la chiusura di eventuali file aperti.

Solitamente si usa una variabile di tipo Boolean o Byte per sapere se si è scatenato un'eccezione e magari impostare un comportamento diverso del finally nel caso ci sia stato o meno

```
Try
    'mi connetto al database
    'eseguo un'istruzione di update errata
Catch ex As OverflowException
    'gestisco l'errore
Catch ex As Exception
    'gestisco eventuali altri errori
Finally
    'chiudo la connessione al database
End Try
```

## **Scatenare (throw) eccezioni**

Come nelle precedenti versioni, posso scatenare eccezioni, ovvero sollevare eccezioni all'interno di classi da me create, nell'esempio, mi passo un valore di tipo "voto" da registrare, quindi accettando valori tra 18 e 30 me lo faccio passare in un Byte, tuttavia mi viene passato il valore 12, posso scrivere la seguente istruzione

La classe che ho sviluppato è la seguente:

```
Class ClStudente
  Public Sub RegistraVoto(ByVal Voto As Byte)
    If Voto < 18 Or Voto > 30 Then
      Throw New ArgumentOutOfRangeException(Voto, _
        "Il voto non è valido inserire un valore tra 18 e 30")
    Else
      'salva il voto nel database
    End If
  End Sub
End Class
```

Mentre la utilizzo in questo modo

```
Private Sub CmdThrow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CmdThrow.Click
  Dim MioStudente As New ClStudente

  Try
    MioStudente.RegistraVoto(2)

  Catch ex As ArgumentOutOfRangeException
    MessageBox.Show("Il voto " & ex.ParamName & " non è consentito ")

  Catch ex As Exception
    MessageBox.Show(ex.ToString)

  End Try
End Sub
```

Il messaggio che mi appare è "Il voto 2 non è consentito"

# ADO.Net

## **ADO.Net: Introduzione**

Ado.net ha stravolto abbastanza l'idea di accesso ai database. In particolare per due motivi: primo, a differenza di Ado / DAO, principalmente lavora disconnesso, ovvero vengono inviati una serie di dati al client, si fanno le dovute elaborazioni, quindi i dati vengono ri-spediti al server.

Secondo: posso ri-costruire la struttura del database anche sul client, creando le tabelle e le relazioni, le chiavi primarie ed i vincoli.

Ovviamente esiste anche la possibilità di lavorare connessi, anche se risulta più difficile per il fatto di dover gestire molte operazioni di aggiornamento via codice.

La principale differenza nella gestione del recordset in rispetto ad ADO/DAO è che, mentre una volta nei cicli dovevo prestare molta attenzione a non dimenticarmi il moveNext (pena loop continuo) ora posso usare anche nuovi costrutti tipo il for each, su un oggetto DataRow (la riga di una tabella).

Ora il metodo Read, legge la riga e si sposta, se ritorna false, vuol dire che non è riuscito a leggere quindi sono già alla fine del "recordset" (il vecchio EOF per capirsi).

Ultima cosa, Ado è già inserito all'interno del framework, per cui non ho da distribuire altre dll (tipo gli MDAC, piuttosto che la stessa libreria di ado/dao) evitando quindi il famigerato dll hell (inferno dll)

## **Ado.Net: Oggetto Connection**

L'oggetto Connection è l'oggetto incaricato di stabilire una connessione tra l'applicazione (Web o Windows), a differenza che nella precedente versioni di ADO, esistono diversi oggetti connessione, che definirò XXXConnection.

Questo significa che la connessione è gestita da un "provider gestito" ovvero la SqlConnection, sa già che si connette ad un SqlServer (6.5 e seguenti), per tutto quello che non è SqlServer possiamo utilizzare il provider OleDbConnection, piuttosto che il ODBCConnection (anche se personalmente lo sconsiglierei vista la sua particolare lentezza)

### **Proprietà Principali**

- **ConnectionString:** Permette di definire in che modo la connessione deve essere effettuata, quale provider usare, il nome del server/file, l'eventuale password di accesso piuttosto che la scelta della autenticazione di Windows
- **State:** Proprietà Read-Only, restituisce lo stato corrente della connessione (Aperta, Chiusa)

### **Metodi Principali**

- Open (apre la connessione la connection string deve essere stata impostata)
- Close (chiude la connessione)

### **Eventi Principali**

- StateChange (a seguito di un metodo open o close fornisce informazioni sullo stato attuale e precedente della connessione)

Nell'esempio che segue riporto la modalità per connettersi sia a SqlServer sia ad Access, l'esempio è fatto sul database di esempio northwind.

A titolo di esempio, ho impostato le connessioni in due modi diversi, la connessione a SqlServer, dichiarata a livello di Classe, con eventi. Mentre la connessione di Access è stata dichiarata privata per la Sub, nella connessione ad Access uso anche il costruttore per passare direttamente la ConnectionString nel momento in cui la connessione viene dichiarata.

```
Public Class FrmCnn
    Inherits System.Windows.Forms.Form
    Dim WithEvents CnnSql As New SqlConnection

    Private Sub FrmCnn_Load(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) Handles MyBase.Load

        Dim CnnAccess As New OleDb.OleDbConnection("Provider = Microsoft.Jet.Oledb.4.0;
"& _
        "Data Source = C:\Program Files\Microsoft
Office\Office\Samples\Northwind.mdb")

        CnnSql.ConnectionString = "Data Source = PCMAX; initial catalog = Northwind; "
& _
        "integrated security = SSPI"

        Try
            CnnSql.Open()
            CnnAccess.Open()
        Catch ex As Exception
            MessageBox.Show(ex.ToString)
        End Try

        'chiudo e distruggo la connessione di access
        CnnAccess.Close()
        CnnAccess = Nothing

    End Sub

    Private Sub CnnSql_StateChange(ByVal sender As Object, ByVal e _
        As System.Data.StateChangeEventArgs) Handles CnnSql.StateChange
        Debug.WriteLine("Stato precedente connessione: " & e.OriginalState)
        Debug.WriteLine("Stato attuale connessione: " & e.CurrentState)
    End Sub

End Class
```

## Ado.Net: Oggetto Command

Come la Connessione, anche l'oggetto command ora è di tipo specifico, ovvero esiste il SqlCommand e l'OleDbCommand, che ovviamente vanno utilizzati con i rispettivi XXXConnection.

L'oggetto command consente di fare le principali operazioni con i dati e la struttura degli oggetti del database.

Istruzioni DML (Operazioni con i dati): SELECT, INSERT, UPDATE e DELETE

Istruzioni DDL (Operazioni con gli oggetti): CREATE, ALTER e DROP

Istruzioni DCL (Imposto o cambio le autorizzazioni): GRANT, DENY e REVOKE

### **Proprietà Principali**

- Connection: Permette specificare quale connessione va utilizzata
- CommandText: Specifica l'istruzione SQL che verrà eseguita dall'oggetto command

### **Metodi Principali**

- ExecuteNonQuery: Viene utilizzato quando l'istruzione Sql non legge dati ma ne modifica, restituisce il numero di record modificati dall'istruzione
- ExecuteReader: Viene utilizzato con le istruzioni di Select, restituisce un oggetto DataReader (serve per leggere i dati, non sono possibili operazioni di modifica/inserimento/cancellazione)
- ExecuteScalar: Viene utilizzato per eseguire istruzioni che restituiscono solitamente una riga ed una colonna ("select count(\*) from TblClienti") ritorna un intero

Nell'esempio che segue riporto il codice per fare un'istruzione di update, si noti l'utilizzo di ExecuteNonQuery che valorizza direttamente un intero. L'istruzione viene eseguita in un try, per evitare che eventuali operazioni non consentite (riga commentata) provochino errori non gestiti.

```
Private Sub CmdCommand1_Click(ByVal sender As System.Object, ByVal e _  
    As System.EventArgs) Handles CmdCommand1.Click  
    Dim ObjCommand As New SqlClient.SqlCommand 'dichiaro la variabile  
    Dim Risultato As Integer  
    'imposto la connessione da utilizzare  
    ObjCommand.Connection = CnnSql  
  
    'imposto l'istruzione da eseguire  
    ObjCommand.CommandText = "update customers set CompanyName= 'Massimo Piubelli'  
" & _  
    "where customerid = 'alfki'"  
    'stessa istruzione ma che andrà in errore in quanto il campo non accetta valori  
null  
    'ObjCommand.CommandText = "update customers set CompanyName= null where  
customerid = 'alfki'"  
  
    'testo che la connessione sia aperta  
    If CnnSql.State = ConnectionState.Closed Then CnnSql.Open()  
  
    Try  
        Risultato = ObjCommand.ExecuteNonQuery  
    Catch ex As Exception  
        MessageBox.Show(ex.ToString)  
    Finally
```

```

    If Risultato = 0 Then
        MessageBox.Show("Non sono state modificate righe")
    Else
        MessageBox.Show("Sono state modificate " & Risultato & " righe")
    End If
End Try

ObjCommand = Nothing
End Sub

```

In questo secondo esempio utilizzo praticamente la stessa sintassi, solo che in questo caso il risultato è il numero di righe esistenti in tabella

```

Private Sub CmdCommand2_Click(ByVal sender As System.Object, ByVal _
    e As System.EventArgs) Handles CmdCommand2.Click

    Dim ObjCommand As New SqlClient.SqlCommand 'dichiaro la variabile
    Dim Risultato As Integer
    'imposto la connessione da utilizzare
    ObjCommand.Connection = CnnSql

    'imposto l'istruzione da eseguire
    ObjCommand.CommandText = "Select count(*) from customers"

    'testo che la connessione sia aperta
    If CnnSql.State = ConnectionState.Closed Then CnnSql.Open()

    Try
        Risultato = ObjCommand.ExecuteScalar
    Catch ex As Exception
        MessageBox.Show(ex.ToString)
    Finally
        If Risultato = 0 Then
            MessageBox.Show("Non presenti righe")
        Else
            MessageBox.Show("Sono in tabella ci sono " & Risultato & " clienti")
        End If
    End Try

    ObjCommand = Nothing
End Sub

```

Un esempio con un'istruzione di Select verrà fatto successivamente, nell'analisi del DataReader

## Ado.Net: Oggetto DataReader

Il data reader è lo strumento ideale quando devo accedere, in modalità connessa, ai dati in sola lettura.

Ha un metodo principale Read, che praticamente fa tutto. Legge e si sposta.

Il dataReader fa parte degli oggetti di provider specifici, quindi ci troveremo degli XXXDataReader.

Il data reader viene di fatto costruito dopo la chiamata del metodo ExecuteReader dell'oggetto command, tant'è che per questo oggetto nel momento della dichiarazione non uso l'istruzione new per crearne l'istanza.

Nell'esempio che segue si utilizza un dataReader per ciclare la tabella prodotti di Northwind, usando la tecnica della classe per memorizzare 2 valori (idProdotto e NomeProdotto) all'interno della lista.

```
Private Sub CmdCaricaProdotti_Click(ByVal sender As System.Object, ByVal _
    e As System.EventArgs) Handles CmdCaricaProdotti.Click
    Dim ObjCommand As New SqlClient.SqlCommand
    Dim ObjReader As SqlClient.SqlDataReader 'dichiaro il data reader
    Dim Prodotto As ClMioTipo

    'non posso usare il new con il data reader in quanto viene caricato
    direttamente
    'dall'oggetto command

    ObjCommand.CommandText = "Select * from Products" 'non imposto l'ordinamento,
    lo farò in seguito

    ObjCommand.Connection = CnnSql

    ObjReader = ObjCommand.ExecuteReader

    LstProdotti.Items.Clear()

    Do While ObjReader.Read
        Prodotto = New ClMioTipo
        Prodotto.ID = ObjReader.GetValue(0)
        Prodotto.Nome = ObjReader.GetValue(1)
        LstProdotti.Items.Add(Prodotto)
        'non serve il movenext, in quanto il metodo read lo fa già automaticamente
    Loop

    LstProdotti.Sorted = True 'ordino la lista, i dati resteranno coerenti

    ObjReader.Close() 'devo ricordarmi di farlo, altrimenti la volta successiva
    andrà in errore
    ObjReader = Nothing
    ObjCommand = Nothing
```

## Ado.Net: Oggetto DataAdapter

Il dataAdapter è il primo degli oggetti che ci servono per il funzionamento disconnesso di ADO.NET, in particolare è l'oggetto che compie la parte più grossa del lavoro, ovvero, si preoccupa di leggere i dati dal database e, eventualmente fosse necessario, di gestire le operazioni di insert, update e delete.

Tra gli oggetti disconnessi è l'unico ad essere ancora legato ad uno specifico provider, per cui avremo degli XXXDataAdapter.

La sua proprietà principale è la proprietà SelectCommand, alla quale viene passato un oggetto command che eseguirà un'istruzione di select (la connessione in questo caso è la stessa che viene usata dall'oggetto command, usando il dataAdapter, non ho bisogno di usare il metodo open della connessione; il DataAdapter, infatti, se trova la connessione non aperta la aprirà per il tempo necessario per leggere i dati, quindi la chiuderà)

## **Proprietà Principali**

- SelectCommand: E' l'oggetto command che viene passato per leggere i dati
- UpdateCommand, DeleteCommand, InsertCommand: Queste tre proprietà servono per permettere al DataAdapter di poter eseguire l'update dei dati modificati all'interno del database, verranno trattate più in seguito

## **Metodi Principali**

Il dataAdapter dispone di due metodi principali

- Fill: è il metodo che legge i dati dal db, solitamente al metodo fill viene passato un dataSet da popolare e un nome di una tabella
- Update: questo metodo ha la possibilità di aggiornare le righe modificate all'interno del dataSet rendendo definitive le modifiche nel database

Nell'esempio che segue, un semplice DataAdapter carica i dati all'interno di un DataSet. Non vengono gestite operazioni di aggiornamento dei dati. Il dataSet è il prossimo argomento di discussione per cui non verrà approfondito, cito solo che "aggancio" una DataGridView al mio dataSet per permettere la visualizzazione dei dati. Vorrei far notare anche l'imports fatto per semplificare la scrittura del codice.

```
Imports System.Data.OleDb

Public Class FrmDisc
    Inherits System.Windows.Forms.Form
    Dim WithEvents CnnAccess As New OleDb.OleDbConnection("Provider =
Microsoft.Jet.Oledb.4.0; " & _
    "Data Source = C:\Program Files\Microsoft
Office\Office\Samples\Northwind.mdb")
    Dim ObjDataReader As New OleDbDataAdapter
    Dim ObjDataSet As New DataSet

    Private Sub CnnAccess_StateChange(ByVal sender As Object, ByVal e As _
        System.Data.StateChangeEventArgs) Handles CnnAccess.StateChange
        Debug.WriteLine("Stato precedente connessione: " & e.OriginalState)
        Debug.WriteLine("Stato attuale connessione: " & e.CurrentState)
    End Sub

    Private Sub FrmDisc_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ObjCommand As New OleDbCommand
        ObjCommand.Connection = CnnAccess
        ObjCommand.CommandText = "Select * from Clienti"
        ObjDataReader.SelectCommand = ObjCommand

        ObjDataReader.Fill(ObjDataSet, "Clienti")
        GrdClienti.DataSource = ObjDataSet.Tables("Clienti")

    End Sub
End Class
```

## Ado.Net: Oggetto DataSet

Il DataSet è la vera novità di ADO.NET, in quanto permette di mantenere in memoria una serie di dati provenienti sia dallo stesso database, sia da database diversi (anche di diverso tipo, ovvero io potrei caricare all'interno di uno stesso dataset una tabella da Access, una da SQL Server e metterle anche in relazione tra loro - relazione a livello di applicazione ovviamente - )

Il DataSet può quindi essere visto come un piccolo database relazionale memorizzato nella memoria del client, con possibilità di creare tabelle, relazioni e vincoli (chiave primarie, valori unique e anche campi calcolati)

Il DataSet è un oggetto "generico", ovvero non associato a nessun particolare provider.

All'interno del DataSet esiste una collection di Tables, ad una tabella posso accedere o tramite il suo nome `ObjDs.Tables("Clienti")` oppure tramite il suo indice `ObjDs.Tables(0)`

### Ado.Net: Oggetto CommandBuilder

L'oggetto CommandBuilder è in grado, una volta "agganciato" ad un dataAdapter, di fornire al dataAdapter stesso le istruzioni di Insert, Update e Delete, partendo appunto dall'istruzione di Select di cui disponeva il DataAdapter.

### Ado.Net: Oggetto DataRelation

Come dice il nome, questo oggetto consente di creare una relazione all'interno del mio DataSet.

Al punto che ho una relazione, posso scegliere come comportarsi con i record nel caso di aggiornamento o di eliminazione. Posso scegliere di applicare o non applicare l'integrità referenziale e/o di disattivarla temporaneamente per velocizzare ad esempio operazioni di caricamento dei dati.

Nell'esempio di seguito riportato, creo una relazione tra due tabelle, carico tutti i dati nel DataSet, poi dal DataSet, popolo la prima lista con gli ordini.

Quando seleziono un ordine vado a vedermi il dettaglio delle sue righe e lo carico sulla seconda lista.

Da notare che questa tecnica prevede che venga inserita una chiave primaria per il corretto funzionamento del metodo Find della collection Rows (il quale agisce necessariamente su una chiave primaria)

```
Imports System.Data.OleDb
```

```
Public Class FrmRelazioni
```

```
    Inherits System.Windows.Forms.Form
```

```
    Dim WithEvents CnnAccess As New OleDb.OleDbConnection("Provider =  
Microsoft.Jet.Oledb.4.0; " & _
```

```
        "Data Source = C:\Program Files\Microsoft  
Office\Office\Samples\Northwind.mdb")
```

```
    Dim ObjDataAdapter As New OleDbDataAdapter 'qui vengono caricati gli ordini (join  
clienti)
```

```
    Dim ObjDaDettaglio As New OleDbDataAdapter 'qui viene caricato il dettaglio (join  
articoli)
```

```

Dim ObjDataSet As New DataSet 'dataset contiene sia ordini che dettaglio

Private Sub FrmRelazioni_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim ObjOrdini As New OleDbCommand 'oggetto command per ordini
    Dim ObjDettaglio As New OleDbCommand 'oggetto command per dettaglio
    Dim ObjRiga As DataRow 'riga per il ciclo all'interno degli ordini
    Dim NuovoOrdine As ClOrdine 'oggetto che mi consente di caricare i dati
dell'ordine nella lista
    Dim PKOrdine(0) As DataColumn 'matrice per la costruzione della chiave primaria
di ordini
    Dim PKDettaglio(1) As DataColumn 'matrice per la chiave primaria di dettaglio

    'imposto le proprietà dell command di ordini
    ObjOrdini.Connection = CnnAccess
    ObjOrdini.CommandText = "Select IDOrdine, NomeSocietà, DataOrdine from Ordini
inner join Clienti on Ordini.IDCliente = Clienti.IDCliente"
    ObjDataAdapter.SelectCommand = ObjOrdini

    'imposto le proprietà dell command di dettaglio
    ObjDettaglio.Connection = CnnAccess
    ObjDettaglio.CommandText = "Select IDOrdine, do.idprodotto, NomeProdotto,
QuantitàOrdinata from [dettagli ordini] as do inner join prodotti on do.IDProdotto
= prodotti.idprodotto"
    ObjDaDettaglio.SelectCommand = ObjDettaglio

    'riempio il data set con i dati di ordini
    ObjDataAdapter.Fill(ObjDataSet, "Ordini")
    'riempio ancora il data set con i dati di dettaglio
    ObjDaDettaglio.Fill(ObjDataSet, "Dettaglio")

    'ciclo sulle righe di ordini
    For Each ObjRiga In ObjDataSet.Tables("Ordini").Rows
        NuovoOrdine = New ClOrdine
        NuovoOrdine.Data = ObjRiga.Item("DataOrdine")
        NuovoOrdine.ID = ObjRiga.Item("IDOrdine")
        NuovoOrdine.Cliente = ObjRiga.Item("NomeSocietà")

        LstOrdini.Items.Add(NuovoOrdine)
    Next

    'creo la chiave primaria di ordini - necessaria per poter fare il find
    PKOrdine(0) = ObjDataSet.Tables("Ordini").Columns("IDOrdine")
    ObjDataSet.Tables("Ordini").PrimaryKey = PKOrdine

    'creo la chiave primaria di dettaglio, questa non è fondamentale, ma per vedere
anche la chiave doppia è interessante
    PKDettaglio(0) = ObjDataSet.Tables("Dettaglio").Columns("IDOrdine")
    PKDettaglio(1) = ObjDataSet.Tables("Dettaglio").Columns("IDProdotto")
    ObjDataSet.Tables("dettaglio").PrimaryKey = PKDettaglio

    'costruisco la relazione, le do il nome e specifico le colonne che vanno in
chiave
    ObjDataSet.Relations.Add("FK_Ordini_Dettaglio",
ObjDataSet.Tables("Ordini").Columns("IDOrdine"),
ObjDataSet.Tables("Dettaglio").Columns("IDOrdine"))
End Sub

Private Sub LstOrdini_SelectedIndexChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles LstOrdini.SelectedIndexChanged
    Dim ObjRiga As DataRow

```

```

Dim ObjRigheDett() As DataRow
Dim ObjRigaDett As DataRow
Dim ObjDetttaglio As ClDetttaglio

'prima di tutto svuoto la lista di dettaglio
LstDetttaglio.Items.Clear()

'cerco la riga che è selezionata nella lista
ObjRiga = ObjDataSet.Tables("Ordini").Rows.Find(CType(LstOrdini.SelectedItem,
ClOrdine).ID)

'una volta trovata, mi faccio dire "navigando" la relazione, quali sono le
righe di dettaglio di questa riga
ObjRigheDett = ObjRiga.GetChildRows("FK_Ordini_Detttaglio")

'faccio il ciclo sulle righe di dettaglio, per poterle caricare sulla lista
detttaglio
For Each ObjRigaDett In ObjRigheDett
    ObjDetttaglio = New ClDetttaglio
    ObjDetttaglio.Articolo = ObjRigaDett.Item("NomeProdotto")
    ObjDetttaglio.IDOrdine = ObjRigaDett.Item("IDOrdine")
    ObjDetttaglio.IDProdotto = ObjRigaDett.Item("IDProdotto")
    ObjDetttaglio.Quantita = ObjRigaDett.Item("QuantitàOrdinata")
    LstDetttaglio.Items.Add(ObjDetttaglio)
Next

End Sub
End Class

Public Class ClOrdine
    Public Cliente As String
    Public ID As Integer
    Public Data As Date 'per ordini carica la data

    Public Overrides Function ToString() As String
        Return ID & " - " & Cliente & " " & Data
    End Function
End Class

End Class

Public Class ClDetttaglio
    Public IDProdotto As Integer
    Public IDOrdine As Integer
    Public Articolo As String
    Public Quantita As Integer

    Public Overrides Function ToString() As String
        Return Articolo & " - " & Quantita
    End Function
End Class

End Class

```

# **Distribuire un'applicazione Windows**

Finisco con la cosa più semplice, distribuire un'applicazione con .Net è un gioco da ragazzi, basta copiare l'eseguibile nel computer e cliccarci sopra, ovvero posso fare la distribuzione CTRL+C e CTRL+V.

Condizione indispensabile per il funzionamento di un'applicazione .net è che il Framework sia installato sulla macchina, anche un eventuale pacchetto di setup creato con Visual Studio non è in grado di installare il Framework (in quanto il setup.exe che viene generato è anch'esso un eseguibile .net)

**FINE**

Ringrazio tutti per l'attenzione,

Massimo Piubelli

Per contattarmi:

[massimo@piubelli.net](mailto:massimo@piubelli.net)

Il presente documento è scaricabile anche presso il mio sito

[www.piubelli.net](http://www.piubelli.net)