EBook Gratuito

APPRENDIMENTO

Visual Basic .NET Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#vb.net

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: visual-basic--net-language

It is an unofficial and free Visual Basic .NET Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Visual Basic .NET Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Introduzione al linguaggio Visual Basic .NET

Osservazioni

Visual Basic .NET è il successore ufficiale del linguaggio di programmazione Visual Basic originale di Microsoft. Visual Basic [.NET] sembra avere somiglianze con Python con la mancanza di punto e virgola e parentesi, ma condivide con C ++ la struttura di base delle funzioni. Le parentesi graffe sono assenti in VB .NET, ma vengono sostituite con frasi come `End If` , `Next` e `End Sub` .

Versioni

Versione VB.NET	Versione di Visual Studio	Versione di .NET Framework	Data di rilascio
7.0	2002	1.0	2002/02/13
7.1	2003	1.1	2003/04/24
8.0	2005	2.0 / 3.0	2005-10-18
9.0	2008	3.5	2007-11-19
10.0	2010	4.0	2010-04-12
11.0	2012	4.5	2012-08-15
12.0	2013	4.5.1 / 4.5.2	2013/10/17
14.0	2015	4.6.0 ~ 4.6.2	2015/07/20
15.0	2017	4.7	2017/03/07

Examples

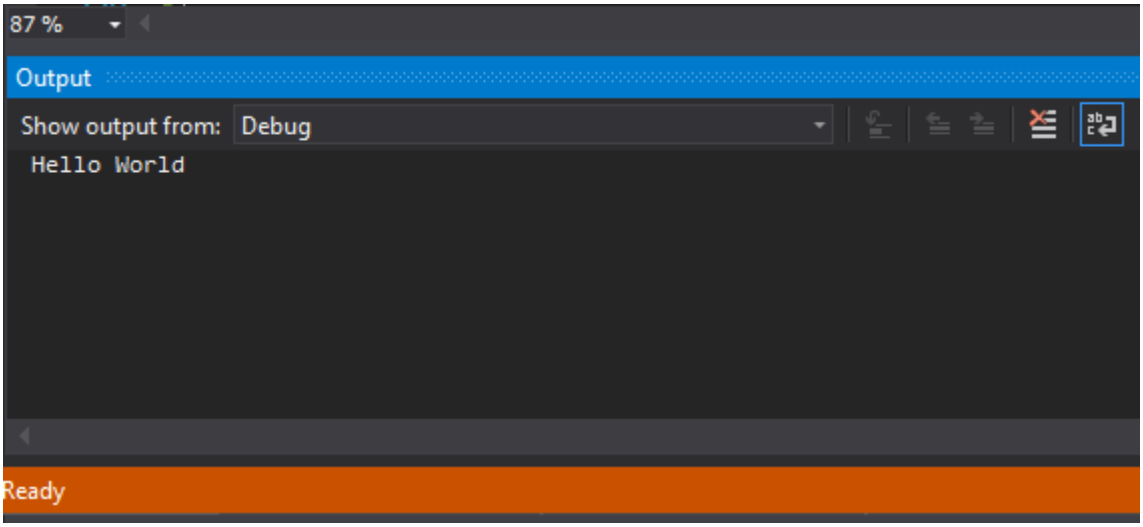
Ciao mondo

Innanzitutto, installa una versione di Microsoft Visual Studio , inclusa l'edizione Community gratuita. Quindi, creare un progetto Applicazione console di Visual Basic di tipo *Applicazione console* e il codice seguente stamperà la stringa `'Hello World'` sulla console:

```
Module Module1
```

```
    Sub Main()
        Console.WriteLine("Hello World")
    End Sub

End Module
```

Quindi, salva e premi `F5` sulla tastiera (o vai al menu *Debug* , quindi fai clic su *Esegui senza Debug* o *Esegui*) per compilare ed eseguire il programma. `'Hello World'` dovrebbe apparire nella finestra della console.



Ciao mondo su una casella di testo al clic di un pulsante

Trascina 1 casella di testo e 1 pulsante

Fai doppio clic sul pulsante 1 e verrai trasferito `Button1_Click event`

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

Digita il nome dell'oggetto che vuoi scegliere come target, nel nostro caso è la `textbox1` . `.Text` è la proprietà che vogliamo usare se vogliamo mettere un testo su di esso.

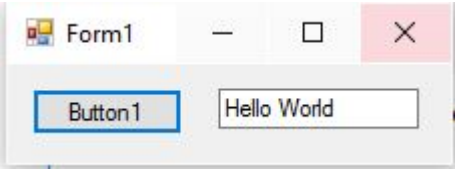
`Property Textbox.Text, gets or sets the current text in the TextBox` . Ora, abbiamo `Textbox1.Text`

Dobbiamo impostare il valore di tale `Textbox1.Text` modo da utilizzare il segno `=` . Il valore che vogliamo inserire in `Textbox1.Text` è `Hello World` . Complessivamente, questo è il codice totale per inserire un valore di `Hello World` in `Textbox1.Text`

```
TextBox1.Text = "Hello World"
```

Aggiunta di quel codice `clicked event` di `button1`

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        TextBox1.Text = "Hello World"
    End Sub
End Class
```

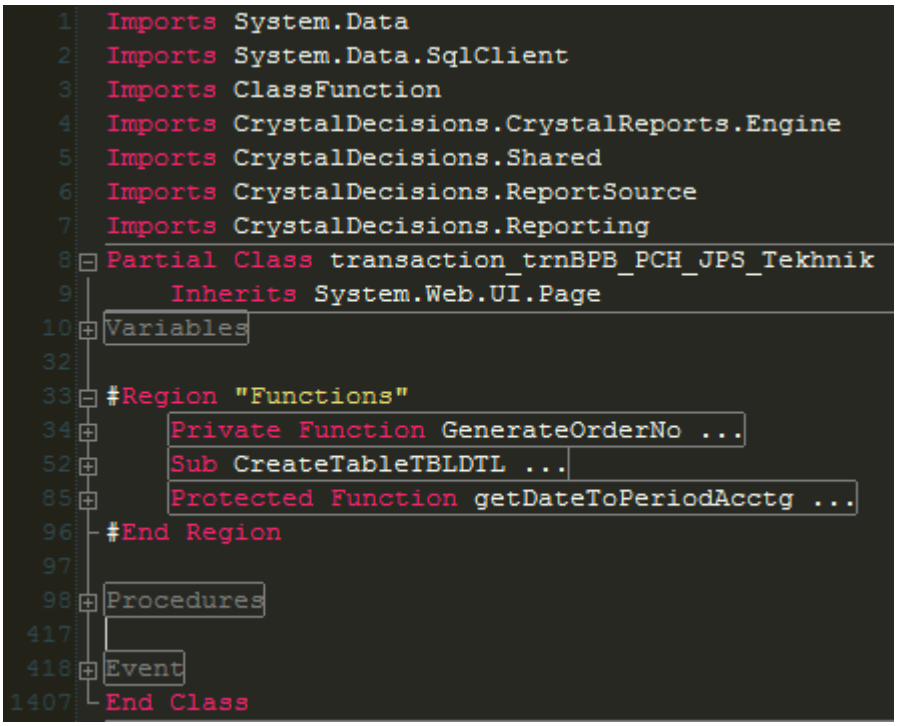


Regione

Per motivi di leggibilità, che sarà utile per i principianti durante la lettura del codice VB e per gli sviluppatori a tempo pieno per mantenere il codice, possiamo usare "Regione" per impostare una regione dello stesso insieme di eventi, funzioni o variabili:

```
#Region "Events"
    Protected Sub txtPrice_TextChanged(...) Handles txtPrice.TextChanged
        'Do the ops here...
    End Sub

    Protected Sub txtTotal_TextChanged(...) Handles txtTotal.TextChanged
        'Do the ops here...
    End Sub

    'Some other events....

#End Region
```

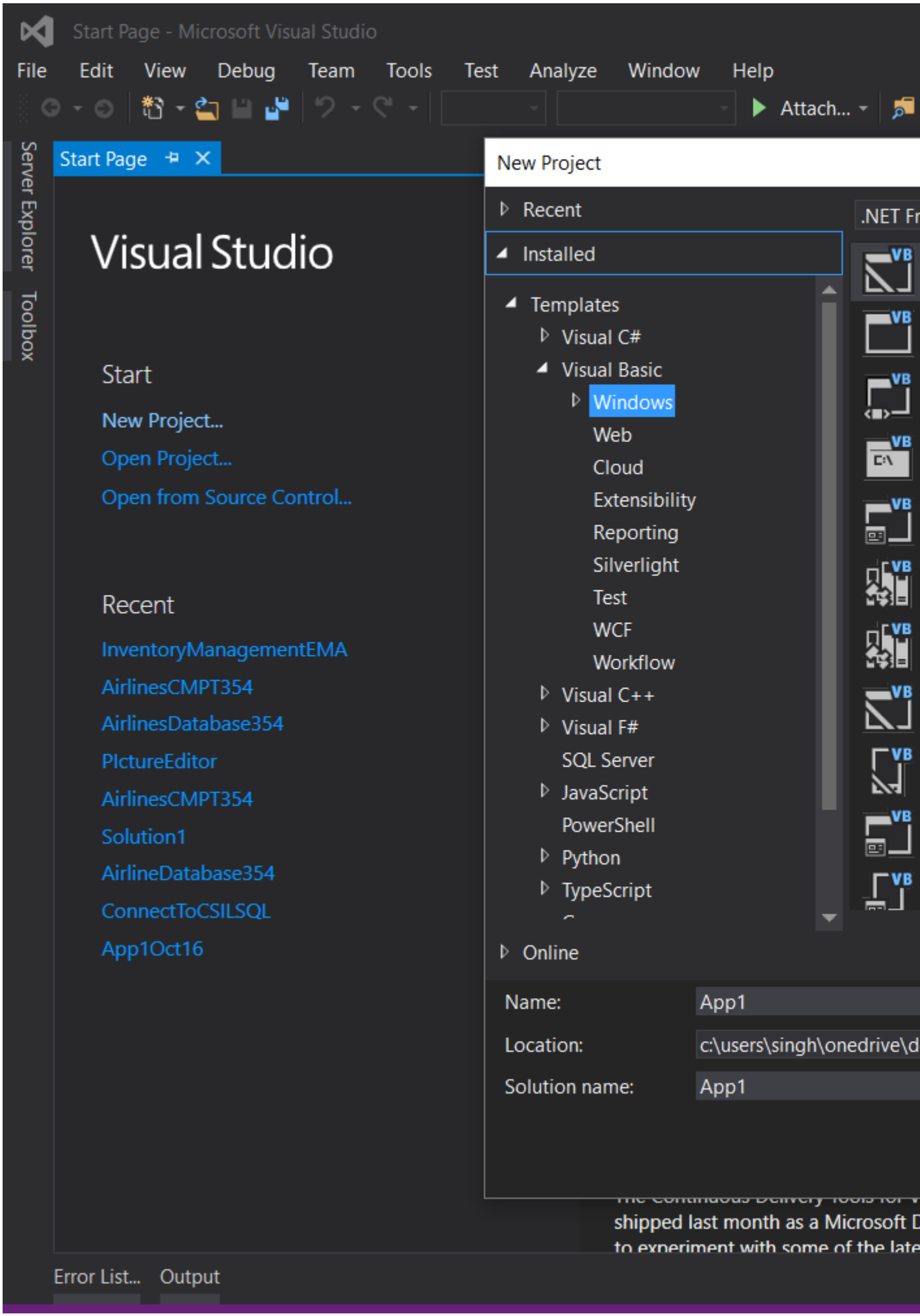
Questo blocco regione potrebbe essere compresso per ottenere un aiuto visivo quando la riga del codice diventa 1000+. Inoltre, salva i tuoi sforzi di scorrimento.



Testato su VS 2005, 2008 2010, 2015 e 2017.

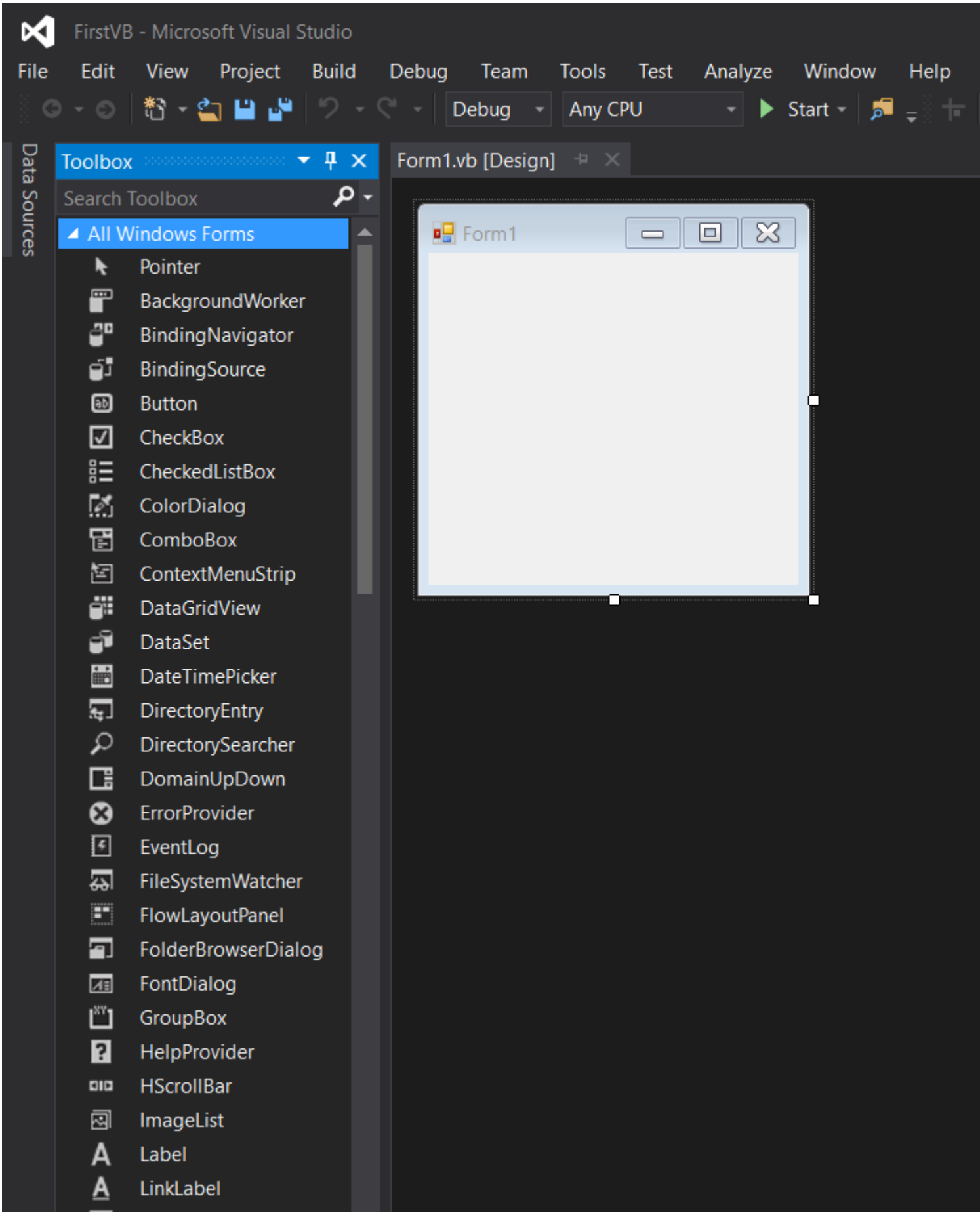
Creare una semplice calcolatrice per familiarizzare con l'interfaccia e il codice.

1. Una volta installato Visual Studio da https://www.visualstudio.com/downloads/ , avvia un nuovo progetto.

2. 
Start Page - Microsoft Visual Studio
File Edit View Debug Team Tools Test Analyze Window Help
Attach...
Server Explorer
Toolbox
Start Page
Visual Studio
Start
New Project...
Open Project...
Open from Source Control...
Recent
InventoryManagementEMA
AirlinesCMPT354
AirlinesDatabase354
PIctureEditor
AirlinesCMPT354
Solution1
AirlineDatabase354
ConnectToCSILSQL
App1Oct16
New Project
Recent
Installed
Templates
Visual C#
Visual Basic
Windows
Web
Cloud
Extensibility
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
SQL Server
JavaScript
PowerShell
Python
TypeScript
Online
Name: App1
Location: c:\users\singh\onedrive\d
Solution name: App1
shipped last month as a Microsoft D
Error List... Output

3. Seleziona "Applicazione Windows Form" dalla scheda Visual Basic. Puoi rinominarlo qui se necessario.

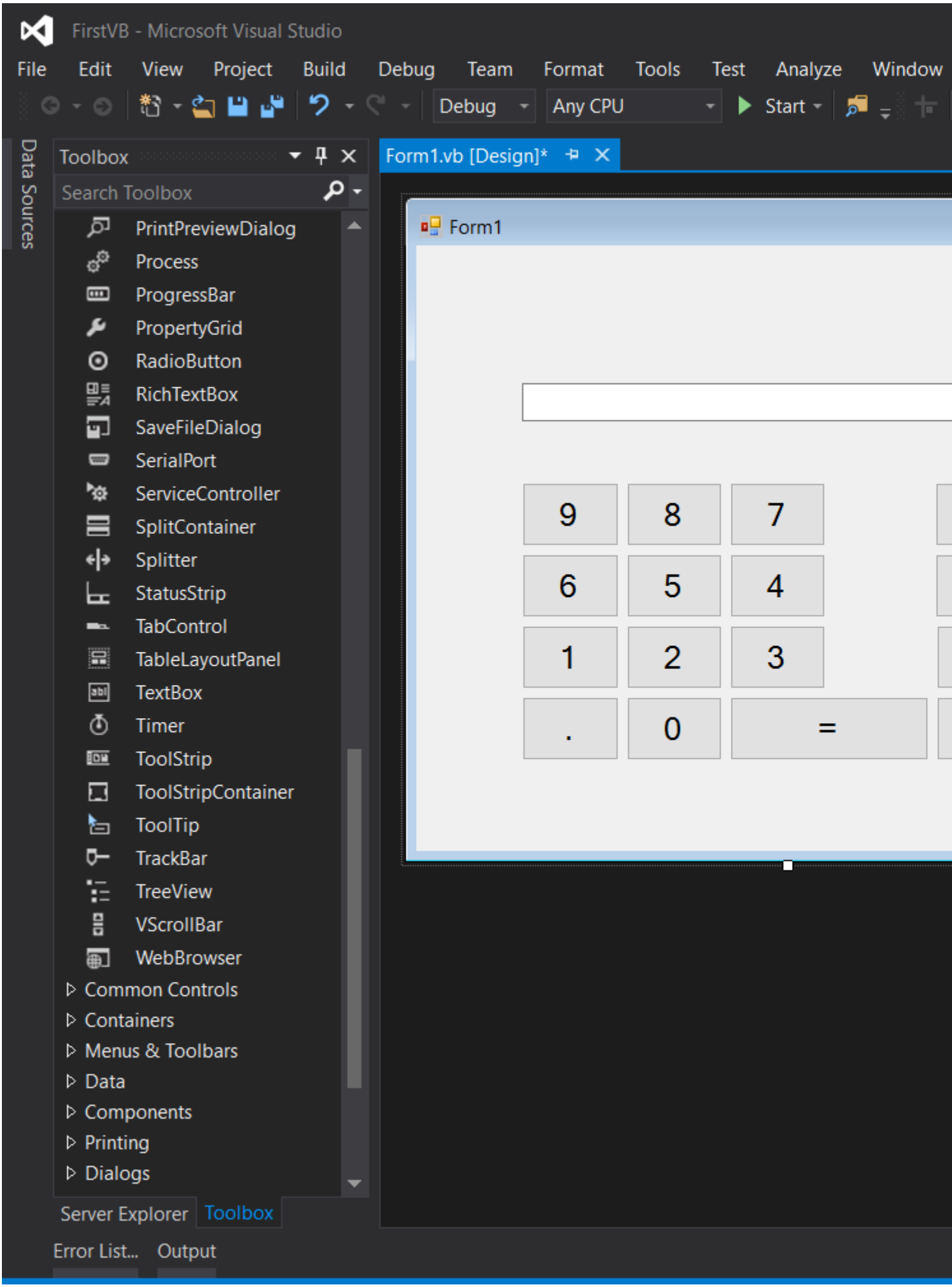
4. Dopo aver fatto clic su "OK", vedrai questa finestra:



5. Fai clic sulla scheda "Toolbox" sulla sinistra. La barra degli strumenti ha l'opzione 'Nascondi automaticamente' abilitata per impostazione predefinita. Per disabilitare questa opzione, fai

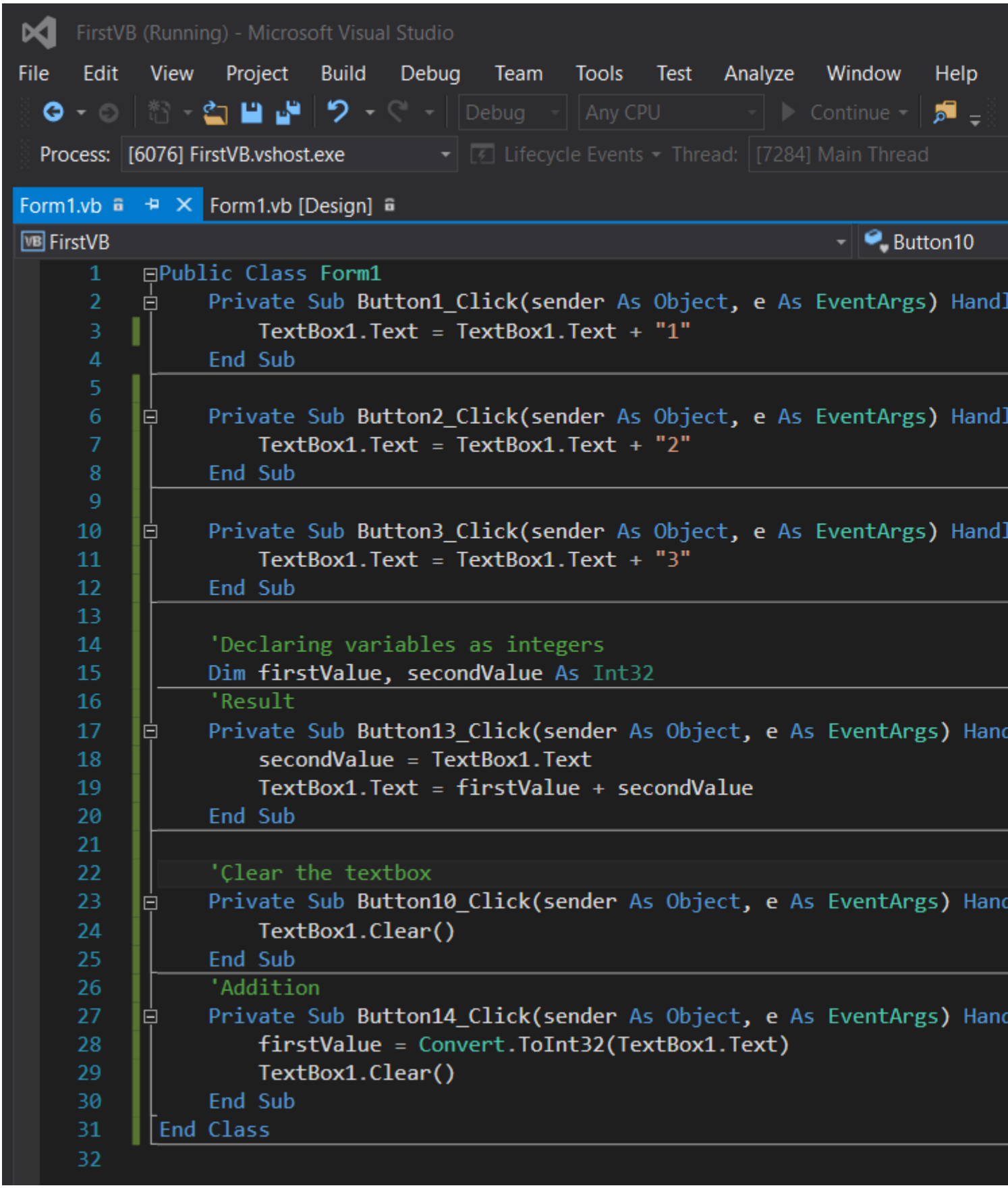
clic sul piccolo simbolo tra il simbolo "freccia giù" e il simbolo "x", nell'angolo in alto a destra della finestra Toolbox.

6. Prendi familiarità con gli strumenti forniti nella scatola. Ho realizzato un'interfaccia per calcolatrice usando i pulsanti e una casella di testo.


FirstVB - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test Analyze Window
Debug
Any CPU
Start
Data Sources
Toolbox
Search Toolbox
PrintPreviewDialog
Process
ProgressBar
PropertyGrid
RadioButton
RichTextBox
SaveFileDialog
SerialPort
ServiceController
SplitContainer
Splitter
StatusStrip
TabControl
TableLayoutPanel
TextBox
Timer
ToolStrip
ToolStripContainer
ToolTip
TrackBar
TreeView
VScrollBar
WebBrowser
Common Controls
Containers
Menus & Toolbars
Data
Components
Printing
Dialogs
Server Explorer
Toolbox
Error List...
Output
Form1.vb [Design]*
Form1
9
8
7
6
5
4
1
2
3
.
0
=

(si trova sul lato destro dell'editor). È possibile modificare la proprietà *Text* di un pulsante e la casella di testo per rinominarli. *La* proprietà *Font* può essere utilizzata per modificare il carattere dei controlli.

8. Per scrivere l'azione specifica per un evento (ad esempio facendo clic su un pulsante), fare doppio clic sul controllo. La finestra del codice si aprirà.



9. VB.Net è un linguaggio potente progettato per uno sviluppo veloce. L'alta incapsulamento e l'astrazione sono un costo per questo. Non è necessario aggiungere *punto* e *virgola* per indicare la fine di una dichiarazione, non ci sono parentesi e, nella maggior parte dei casi, corregge automaticamente il caso degli alfabeti.

10. Il codice fornito nell'immagine dovrebbe essere semplice da capire. *Dim* è la parola chiave utilizzata per inizializzare una variabile e *nuova* alloca la memoria. Tutto ciò che digiti nella casella di testo è di tipo *stringa* per impostazione predefinita. La fusione è necessaria per utilizzare il valore come un tipo diverso.

Goditi la tua prima creazione in VB.Net!

Leggi Introduzione al linguaggio Visual Basic .NET online: https://riptutorial.com/it/vb-net/topic/352/introduzione-al-linguaggio-visual-basic--net

Capitolo 2: Accesso ai dati

Examples

Leggi il campo dal Database

```
Public Function GetUserFirstName(UserName As String) As String
    Dim Firstname As String = ""

    'Specify the SQL that you want to use including a Parameter
    Dim SQL As String = "select firstname from users where username=@UserName"

    'Provide a Data Source
    Dim DBDSN As String = "Data Source=server.address;Initial Catalog=DatabaseName;Persist
Security Info=True;User ID=UserName;Password=UserPassword"

    Dim dbConn As New SqlConnection(DBDSN)

    Dim dbCommand As New SqlCommand(SQL, dbConn)

    'Provide one or more Parameters
    dbCommand.Parameters.AddWithValue("@UserName", UserName)

    'An optional Timeout
    dbCommand.CommandTimeout = 600

    Dim reader As SqlDataReader
    Dim previousConnectionState As ConnectionState = dbConn.State
    Try
        If dbConn.State = ConnectionState.Closed Then
            dbConn.Open()
        End If
        reader = dbCommand.ExecuteReader
        Using reader
            With reader
                If .HasRows Then
                    'Read the 1st Record
                    reader.Read()
                    'Read required field/s
                    Firstname = .Item("FirstName").ToString
                End If

            End With

        End Using

    Catch
        'Handle the error here
    Finally
        If previousConnectionState = ConnectionState.Closed Then
            dbConn.Close()
        End If
        dbConn.Dispose()
        dbCommand.Dispose()

    End Try
    'Pass the data back from the function
```

```
    Return Firstname

End Function
```

Utilizzare la funzione sopra è semplicemente:

```
   Dim UserFirstName as string=GetUserFirstName(UserName)
```

Funzione semplice da leggere dal database e restituire come DataTable

Questa semplice funzione eseguirà il comando Select SQL specificato e restituirà il risultato come set di dati.

```
Public Function ReadFromDatabase(ByVal DBConnectionString As String, ByVal SQL As String) As
DataTable
    Dim dtReturn As New DataTable
    Try
        'Open the connection using the connection string
        Using conn As New SqlClient.SqlConnection(DBConnectionString)
            conn.Open()

            Using cmd As New SqlClient.SqlCommand()
                cmd.Connection = conn
                cmd.CommandText = SQL
                Dim da As New SqlClient.SqlDataAdapter(cmd)
                da.Fill(dtReturn)
            End Using
        End Using
    Catch ex As Exception
        'Handle the exception
    End Try


    'Return the result data set
    Return dtReturn
End Function
```

Ora puoi eseguire la funzione di cui sopra dai seguenti codici

```
Private Sub MainFunction()
    Dim dtCustomers As New DataTable
    Dim dtEmployees As New DataTable
    Dim dtSuppliers As New DataTable


    dtCustomers = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Customers]")
    dtEmployees = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Employees]")
    dtSuppliers = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Suppliers]")

End Sub
```

L'esempio sopra prevede che l'istanza di SQL Express "SQLEXPRESS" sia attualmente installata

su "MYDEVPC" e il tuo database "MyDatabase" contenga le tabelle "Clienti", "Fornitori" e "Dipendenti" e la password utente "sa" sia "pwd22". Si prega di modificare questi valori secondo la configurazione per ottenere i risultati desiderati.

Ottieni dati scalari

Questa semplice funzione può essere utilizzata per ottenere il valore esattamente da un risultato di query del record di un campo uno

```
Public Function getDataScalar(ssql As String)
    openConnection()

    Try
        Dim q As New MySqlCommand

        q.Connection = db
        q.CommandText = ssql
        getDataScalar = q.ExecuteScalar

    Catch ex As Exception
        'Exception
    End Try
End Function
```

Come usarlo:

```
Dim userid as String = getDataScalar("select username from user where userid=99")
```

La variabile "nome utente" verrebbe riempita con il valore del nome utente del campo come risultato di quella query.

Leggi Accesso ai dati online: https://riptutorial.com/it/vb-net/topic/3113/accesso-ai-dati

Capitolo 3: Associazione dati XAML WPF

introduzione

Questo esempio mostra come creare un ViewModel e una Vista all'interno del pattern MVVM e WPF, e come associare i due insieme, in modo che ciascuno venga aggiornato ogni volta che viene modificato l'altro.

Examples

Associazione di una stringa in ViewModel a un controllo TextBox nella vista

SampleViewModel.vb

```
'Import classes related to WPF for simplicity
Imports System.Collections.ObjectModel
Imports System.ComponentModel

Public Class SampleViewModel
    Inherits DependencyObject
    'A class acting as a ViewModel must inherit from DependencyObject

    'A simple string property
    Public Property SampleString as String
        Get
            Return CType(GetValue(SampleStringProperty), String)
        End Get

        Set(ByVal value as String)
            SetValue(SampleStringProperty, value)
        End Set
    End Property

    'The DependencyProperty that makes databinding actually work
    'for the string above
    Public Shared ReadOnly SampleStringProperty As DependencyProperty = _
                           DependencyProperty.Register("SampleString", _
                           GetType(String), GetType(SampleViewModel), _
                           New PropertyMetadata(Nothing))

End Class
```

Una DependencyProperty può essere facilmente aggiunta utilizzando lo snippet di codice `wpfdp` (digita `wpfdp` , quindi premi due volte il tasto `TAB`), tuttavia, lo snippet di codice non è sicuro, e non verrà compilato in `Option Strict On` .

SampleWindow.xaml

```
<Window x:Class="SampleWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        xmlns:des="http://schemas.microsoft.com/expression/blend/2008"
        DataContext="{Binding}"
        Loaded="Window_Loaded">
    <Grid>
        <TextBox>
            <TextBox.Text>
                <Binding Path="SampleString" />
            </TextBox.Text>
        </TextBox>
    </Grid>
</Window>
```

SampleWindow.xaml.vb

```
Class SampleWindow

    Private WithEvents myViewModel As New SampleViewModel()

    Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Me.DataContext = myViewModel
    End Sub
End Class
```

Si noti che questo è un modo molto rudimentale per implementare MVVM e il databinding. Una pratica più solida sarebbe quella di utilizzare una piattaforma come Unity per "iniettare" ViewModel nella Vista.

Leggi Associazione dati XAML WPF online: https://riptutorial.com/it/vb-net/topic/8177/associazione-dati-xaml-wpf

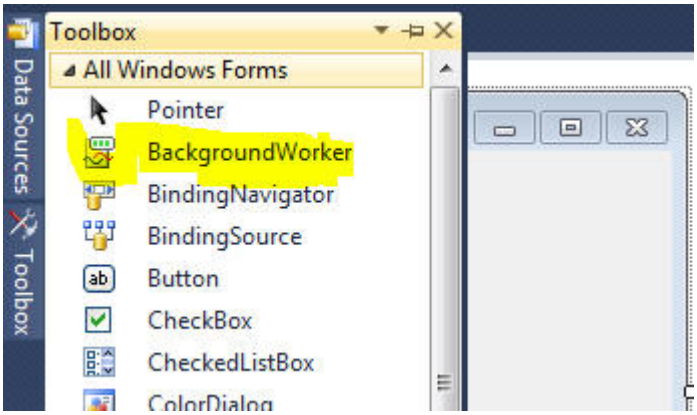
Capitolo 4: BackgroundWorker

Examples

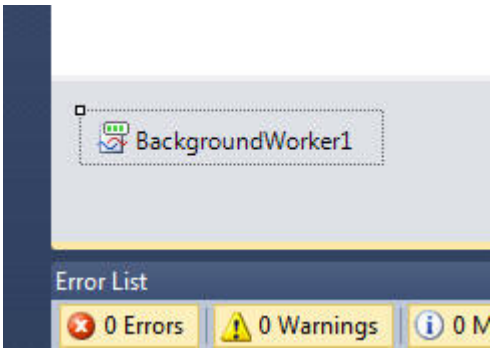
Utilizzando BackgroundWorker

Esecuzione di un'attività con l'operatore in background.

Fare doppio clic sul controllo `BackgroundWorker` dalla casella degli strumenti



Ecco come appare il BackgroundWorker dopo averlo aggiunto.



Fare doppio clic sul controllo aggiunto per ottenere l'evento `BackgroundWorker1_DoWork` e aggiungere il codice da eseguire quando viene chiamato BackgroundWorker. Qualcosa come questo:

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

    'Do the time consuming background task here

End Sub
```

La chiamata di BackgroundWorker per eseguire l'attività può essere eseguita in qualsiasi evento come `Button_Click` , `Textbox_TextChanged` , ecc. Come segue:

```
BackgroundWorker1.RunWorkerAsync()
```

Modificare l'evento `RunWorkerCompleted` per acquisire l'evento finito dell'attività di BackgroundWorker come segue:

```
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles
BackgroundWorker1.RunWorkerCompleted
    MsgBox("Done")
End Sub
```

Verrà visualizzata una finestra di messaggio che dice `Done` quando l'operatore termina l'attività assegnata ad esso.

Accesso ai componenti della GUI in BackgroundWorker

Non è possibile accedere a componenti della GUI da BackgroudWorker. Ad esempio se provi a fare qualcosa di simile

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    TextBox1.Text = "Done"
End Sub
```

si riceverà un errore di runtime che dice che "Operazione cross-thread non valida: controllo 'TextBox1' accessibile da un thread diverso dal thread su cui è stato creato."

Questo perché BackgroundWorker esegue il codice su un altro thread in parallelo con il thread principale e i componenti della GUI non sono thread-safe. Devi impostare il tuo codice per essere eseguito sul thread principale usando il metodo `Invoke` , dandogli un delegato:

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    Me.Invoke(New MethodInvoker(Sub() Me.TextBox1.Text = "Done"))
End Sub
```

Oppure puoi utilizzare il metodo ReportProgress di BackgroundWorker:

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    Me.BackgroundWorker1.ReportProgress(0, "Done")
End Sub

Private Sub BackgroundWorker1_ProgressChanged(sender As Object, e As ProgressChangedEventArgs)
    Me.TextBox1.Text = DirectCast(e.UserState, String)
End Sub
```

Leggi BackgroundWorker online: https://riptutorial.com/it/vb-net/topic/6242/backgroundworker

Capitolo 5: Casuale

introduzione

La classe Random è usata per generare interi pseudo-casuali non negativi che non sono veramente casuali, ma sono per scopi generali abbastanza vicini.

La sequenza viene calcolata usando un numero iniziale (chiamato il **seme**) Nelle precedenti versioni di .net, questo numero di seme era lo stesso ogni volta che veniva eseguita un'applicazione. Quindi, ciò che accadrebbe era che avresti ottenuto la stessa sequenza di numeri pseudo-casuali ogni volta che l'applicazione è stata eseguita. Ora, il seme si basa sul momento in cui l'oggetto è dichiarato.

Osservazioni

Infine, una nota sulla randomizzazione. Come accennato in precedenza, quando si dichiara un'istanza di `Random` senza parametri, il costruttore utilizzerà l'ora corrente come parte del calcolo per creare il numero iniziale di seme. Normalmente questo è OK.

Però. Se si riscrivono nuove istanze in un intervallo di tempo molto breve, ogni volta che viene calcolato il numero di seme, il tempo potrebbe essere lo stesso. Considera questo codice.

```
For i As Integer = 1 To 100000
    Dim rnd As New Random
    x = rnd.Next
Next
```

Poiché i computer sono molto veloci in questi giorni, questo codice impiegherà una frazione di secondo per essere eseguito e su diverse iterazioni del ciclo, l'ora del sistema non sarà cambiata. Quindi, il numero seme non cambierà e il numero casuale sarà lo stesso. Se si desidera generare molti numeri casuali, dichiarare l'istanza di casuale al di fuori del ciclo in questo semplice esempio.

```
Dim rnd As New Random
For i As Integer = 1 To 100000
    x = rnd.Next
Next
```

La regola di base è non riattivare un generatore di numeri casuali su brevi periodi di tempo.

Examples

Dichiarazione di un'istanza

```
Dim rng As New Random()
```

Questo dichiara un'istanza della classe Random chiamata `rng` . In questo caso, l'ora corrente nel punto in cui viene creato l'oggetto viene utilizzata per calcolare il seme. Questo è l'uso più comune, ma ha i suoi problemi, come vedremo più avanti nelle osservazioni

Invece di consentire al programma di utilizzare l'ora corrente come parte del calcolo per il numero seme iniziale, è possibile specificare il numero seme iniziale. Questo può essere qualsiasi letterale intero a 32 bit, costante o variabile. Vedi sotto per gli esempi. Ciò significa che l'istanza genererà la stessa sequenza di numeri pseudo-casuali, che può essere utile in determinate situazioni.

```
Dim rng As New Random(43352)
```

o

```
Dim rng As New Random(x)
```

dove `x` è stato dichiarato altrove nel tuo programma come costante o variabile intera.

Genera un numero casuale da un'istanza di Random

L'esempio seguente dichiara una nuova istanza della classe Random e quindi utilizza il metodo `.Next` per generare il numero successivo nella sequenza di numeri pseudo-casuali.

```
Dim rnd As New Random
Dim x As Integer
x = rnd.Next
```

L'ultima riga sopra genererà il prossimo numero pseudo-casuale e lo assegnerà a `x` . Questo numero sarà compreso tra 0 e 2147483647. Tuttavia, è anche possibile specificare l'intervallo di numeri da generare come nell'esempio seguente.

```
x = rnd.Next(15, 200)
```

Si noti, tuttavia, che utilizzando questi parametri, l'intervallo di numeri sarà compreso tra 15 o sopra e 199 o inferiore.

Puoi anche generare numeri in virgola mobile del tipo Double usando `.NextDouble` es

```
Dim rnd As New Random
Dim y As Double
y = rnd.NextDouble()
```

Tuttavia, non è possibile specificare un intervallo per questo. Sarà sempre nell'intervallo da 0,0 a meno di 1,0.

Leggi Casuale online: https://riptutorial.com/it/vb-net/topic/10128/casuale

Capitolo 6: Classi

introduzione

Una classe raggruppa diverse funzioni, metodi, variabili e proprietà, che sono chiamati membri. Una classe incapsula i membri, a cui è possibile accedere da un'istanza della classe, chiamata oggetto. Le classi sono estremamente utili per il programmatore, poiché rendono l'attività comoda e veloce, con caratteristiche quali modularità, riutilizzabilità, manutenibilità e leggibilità del codice.

Le classi sono gli elementi costitutivi dei linguaggi di programmazione orientati agli oggetti.

Examples

Creare classi

Le classi forniscono un modo per creare i propri tipi all'interno del framework .NET. All'interno di una definizione di classe è possibile includere quanto segue:

- campi
- Proprietà
- metodi
- Costruttori
- eventi

Per dichiarare una classe si usa la seguente sintassi:

```
Public Class Vehicle
End Class
```

Altri tipi .NET possono essere incapsulati all'interno della classe ed esposti di conseguenza, come mostrato di seguito:

```
Public Class Vehicle
     Private Property _numberOfWheels As Integer
     Private Property _engineSize As Integer

     Public Sub New(engineSize As Integer, wheels As Integer)
         _numberOfWheels = wheels
         _engineSize = engineSize
     End Sub

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     End Function
End Class
```

Classi astratte

Se le classi condividono funzionalità comuni, puoi raggrupparle in una classe base o astratta. Le classi astratte possono contenere l'implementazione parziale o nessuna e consentire al tipo derivato di sovrascrivere l'implementazione di base.

Le classi astratte all'interno di VisualBasic.NET devono essere dichiarate come `MustInherit` e non possono essere istanziate.

```
Public MustInherit Class Vehicle
     Private Property _numberOfWheels As Integer
     Private Property _engineSize As Integer

     Public Sub New(engineSize As Integer, wheels As Integer)
         _numberOfWheels = wheels
         _engineSize = engineSize
     End Sub

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     End Function
End Class
```

Un sottotipo può quindi `inherit` questa classe astratta come mostrato di seguito:

```
Public Class Car
    Inherits Vehicle
End Class
```

L'auto erediterà tutti i tipi dichiarati all'interno del veicolo, ma potrà accedervi solo in base al modificatore di accesso sottostante.

```
Dim car As New Car()
car.DisplayWheelCount()
```

Nell'esempio precedente viene creata una nuova istanza di Car. Viene quindi richiamato il metodo `DisplayWheelCount()` che chiamerà l'implementazione `Vehicles` classe base.

Leggi Classi online: https://riptutorial.com/it/vb-net/topic/3522/classi

Capitolo 7: Compressione file / cartella

Examples

Creazione di un archivio zip dalla directory

```
System.IO.Compression.ZipFile.CreateFromDirectory("myfolder", "archive.zip")
```

Crea un file archive.zip contenente i file che sono in `myfolder` . Nei percorsi di esempio sono relativi alla directory di lavoro del programma. È possibile specificare percorsi assoluti.

Estrarre l'archivio zip nella directory

```
System.IO.Compression.ZipFile.ExtractToDirectory("archive.zip", "myfolder")
```

Estrae archive.zip nella directory myfolder. Nei percorsi di esempio sono relativi alla directory di lavoro del programma. È possibile specificare percorsi assoluti.

Crea dinamicamente l'archivio zip

```
' Create filestream to file
Using fileStream = New IO.FileStream("archive.zip", IO.FileMode.Create)
    ' open zip archive from stream
    Using archive = New System.IO.Compression.ZipArchive(fileStream,
IO.Compression.ZipArchiveMode.Create)
        ' create file_in_archive.txt in archive
        Dim zipfile = archive.CreateEntry("file_in_archive.txt")

        ' write Hello world to file_in_archive.txt in archive
        Using sw As New IO.StreamWriter(zipfile.Open())
            sw.WriteLine("Hello world")
        End Using

    End Using
End Using
```

Aggiunta della compressione dei file al progetto

1. In *Solution Explorer* vai al tuo progetto, fai clic destro su *Riferimenti* quindi *Aggiungi riferimento ...*
2. Cercare Compressione e selezionare *System.IO.Compression.FileSystem* quindi premere OK.
3. Aggiungi `Imports System.IO.Compression` all'inizio del file di codice (prima di qualsiasi classe o modulo, con le altre istruzioni `Imports`).

```
Option Explicit On
Option Strict On
```

```
Imports System.IO.Compression

Public Class Foo

    ...

End Class
```

Si noti che questa classe (ZipArchive) è disponibile solo da .NET versione 4.5 in poi

Leggi Compressione file / cartella online: https://riptutorial.com/it/vb-net/topic/4638/compressione-file---cartella

Capitolo 8: condizioni

Examples

If ... Then ... Else

```
Dim count As Integer = 0
Dim message As String

If count = 0 Then
    message = "There are no items."
ElseIf count = 1 Then
    message = "There is 1 item."
Else
    message = "There are " & count & " items."
End If
```

Se l'operatore

9.0

```
If(condition > value, "True", "False")
```

Possiamo usare l'operatore **If** invece di **If ... Then ... Else..End Se** blocchi di istruzioni.

Considera il seguente esempio:

```
If 10 > 9 Then
     MsgBox("True")
Else
     MsgBox("False")
End If
```

equivale a

```
MsgBox(If(10 > 9, "True", "False"))
```

`If()` usa la valutazione di *cortocircuito* , il che significa che valuterà solo gli argomenti che usa. Se la condizione è falsa (o un `Nullable` che non è `Nothing`), la prima alternativa non verrà valutata affatto e nessuno dei suoi effetti collaterali sarà osservato. Questo è effettivamente lo stesso dell'operatore ternario di C # sotto forma di `condition?a:b` .

Questo è particolarmente utile per evitare eccezioni:

```
Dim z As Integer = If(x = 0, 0, y/x)
```

Sappiamo tutti che dividendo per zero si genera un'eccezione, ma `If()` qui si difende da questo, cortocircuitando solo l'espressione che la condizione ha già assicurato è valida.

Un altro esempio:

```
Dim varDate as DateTime = If(varString <> "N/A", Convert.ToDateTime(varString), Now.Date)
```

Se `varString <> "N/A"` `varDate` `False` , assegnerà il valore di `Now.Date` come `Now.Date` senza valutare la prima espressione.

9.0

Le versioni precedenti di VB non hanno l'operatore `If()` e devono `IIf()` della funzione integrata `IIf()` . Come è una funzione, non un operatore, *non* è così corto circuito; vengono valutate tutte le espressioni, con tutti i possibili effetti collaterali, tra cui penalizzazioni delle prestazioni, modifica dello stato e eccezioni di lancio. (Entrambi i precedenti esempi che evitano eccezioni verrebbero lanciati se convertiti in `IIf` .) Se uno qualsiasi di questi effetti collaterali presenta un problema, non c'è modo di usare un condizionale in linea; invece, `If..Then` affidamento su `If..Then` blocca come al solito.

Leggi condizioni online: https://riptutorial.com/it/vb-net/topic/7484/condizioni

Capitolo 9: consolle

Examples

Console.ReadLine ()

```
Dim input as String = Console.ReadLine()
```

`Console.ReadLine()` leggerà l'input della console dall'utente, fino a quando non viene rilevato il nuovo fine riga (in genere premendo il tasto Invio o Invio). L'esecuzione del codice è sospesa nel thread corrente fino a quando non viene fornita una nuova riga. Successivamente, verrà eseguita la prossima riga di codice.

Console.WriteLine ()

```
Dim x As Int32 = 128
Console.WriteLine(x) ' Variable '
Console.WriteLine(3) ' Integer '
Console.WriteLine(3.14159) ' Floating-point number '
Console.WriteLine("Hello, world") ' String '
Console.WriteLine(myObject) ' Outputs the value from calling myObject.ToString()
```

Il metodo `Console.WriteLine()` stamperà gli argomenti specificati **con** una newline allegata alla fine. Questo stamperà qualsiasi oggetto fornito, incluso, ma non limitato a stringhe, numeri interi, variabili, numeri in virgola mobile.

Durante la scrittura di oggetti che non vengono esplicitamente richiamati dai vari overload di `WriteLine` (ovvero, si sta utilizzando il sovraccarico che prevede un valore di tipo `Object` , WriteLine utilizzerà il metodo `.ToString()` per generare una `String` da scrivere effettivamente. gli oggetti dovrebbero OverRide del metodo `.ToString` e produrre qualcosa di più significativo dell'implementazione predefinita (che in genere scrive solo il nome del tipo completo).

Console.Write ()

```
Dim x As Int32 = 128
Console.Write(x) ' Variable '
Console.Write(3) ' Integer '
Console.Write(3.14159) ' Floating-point number '
Console.Write("Hello, world") ' String '
```

Il metodo `Console.Write()` è identico al metodo `Console.WriteLine()` , tranne per il fatto che stampa gli argomenti specificati **senza** una newline allegata alla fine. Questo metodo può essere reso funzionalmente identico a `WriteLine` aggiungendo una stringa newline alla fine di qualsiasi argomento fornito:

```
Console.Write("this is the value" & Environment.NewLine)
```

Console.Read ()

```
Dim inputCode As Integer = Console.Read()
```

`Console.Read()` attende l'input dell'utente e, al ricevimento, restituisce un valore intero corrispondente al codice carattere del carattere inserito. Se il flusso di input è terminato in qualche modo prima che l'input possa essere ottenuto, viene invece restituito -1.

Console.ReadKey ()

```
Dim inputChar As ConsoleKeyInfo = Console.ReadKey()
```

`Console.ReadKey()` attende l'input dell'utente e, al ricevimento, restituisce un oggetto di classe `ConsoleKeyInfo` , che contiene informazioni rilevanti per il carattere che l'utente ha fornito come input. Per i dettagli relativi alle informazioni fornite, visitare la documentazione MSDN .

Prototipo di prompt della riga di comando

```
Module MainPrompt
Public Const PromptSymbol As String = "TLA > "
Public Const ApplicationTitle As String = GetType(Project.BaseClass).Assembly.FullName
REM Or you can use a custom string
REM Public Const ApplicationTitle As String = "Short name of the application"

Sub Main()
    Dim Statement As String
    Dim BrokenDownStatement As String()
    Dim Command As String
    Dim Args As String()
    Dim Result As String

    Console.ForegroundColor = ConsoleColor.Cyan
    Console.Title = ApplicationTitle & " command line console"

    Console.WriteLine("Welcome to " & ApplicationTitle & "console frontend")
    Console.WriteLine("This package is version " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString)
    Console.WriteLine()
    Console.Write(PromptSymbol)

    Do While True
        Statement = Console.ReadLine()
        BrokenDownStatement = Statement.Split(" ")
        ReDim Args(BrokenDownStatement.Length - 1)
        Command = BrokenDownStatement(0)

        For i = 1 To BrokenDownStatement.Length - 1
            Args(i - 1) = BrokenDownStatement(i)
        Next

        Select Case Command.ToLower
            Case "example"
                Result = DoSomething(Example)
            Case "exit", "quit"
```

```
                Exit Do
            Case "ver"
                Result = "This package is version " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString
            Case Else
                Result = "Command not acknowldged: -" & Command & "-"
        End Select
        Console.WriteLine(" " & Result)
        Console.Write(PromptSymbol)
    Loop

    Console.WriteLine("I am exiting, time is " & DateTime.Now.ToString("u"))
    Console.WriteLine("Goodbye")
    Environment.Exit(0)
End Sub
End Module
```

Questo prototipo genera un interprete di base della riga di comando.

Ottiene automaticamente il nome dell'applicazione e la versione per comunicare all'utente. Per ogni riga di input, riconosce il comando e una lista arbitraria di argomenti, tutti separati dallo spazio.

Come esempio di base, il codice a capire i comandi *di uscita ver, chiudere* e.

Il parametro *Project.BaseClass* è una classe del progetto in cui sono impostati i dettagli dell'Assieme.

Leggi consolle online: https://riptutorial.com/it/vb-net/topic/602/consolle

Capitolo 10: Data

Examples

Conversione (analisi) di una stringa in una data

Se conosci il formato della stringa che stai convertendo (analisi), dovresti usare `DateTime.ParseExact`

```
Dim dateString As String = "12.07.2003"
Dim dateFormat As String = "dd.MM.yyyy"
Dim dateValue As Date

dateValue = DateTime.ParseExact(dateString, dateFormat,
Globalization.CultureInfo.InvariantCulture)
```

Se non si è certi del formato della stringa, è possibile utilizzare `DateTime.TryParseExact` e verificare il risultato per vedere se analizzato o meno:

```
Dim dateString As String = "23-09-2013"
Dim dateFormat As String = "dd-MM-yyyy"
Dim dateValue As Date

If DateTime.TryParseExact(dateString, dateFormat, Globalization.CultureInfo.InvariantCulture,
DateTimeStyles.None, dateValue) Then
    'the parse worked and the dateValue variable now holds the datetime that was parsed as it
is passing in ByRef
Else
    'the parse failed
End If
```

Convertire una data in una stringa

Basta usare l'overload `.ToString` di un oggetto `DateTime` per ottenere il formato richiesto:

```
Dim dateValue As DateTime = New DateTime(2001, 03, 06)
Dim dateString As String = dateValue.ToString("yyyy-MM-dd") '2001-03-06
```

Leggi Data online: https://riptutorial.com/it/vb-net/topic/3727/data

Capitolo 11: Debug della tua applicazione

introduzione

Ogni volta che hai un problema nel tuo codice, è sempre una buona idea sapere cosa sta succedendo all'interno. La classe System.Diagnostics.Debug in .Net Framework ti aiuterà molto in questo compito.

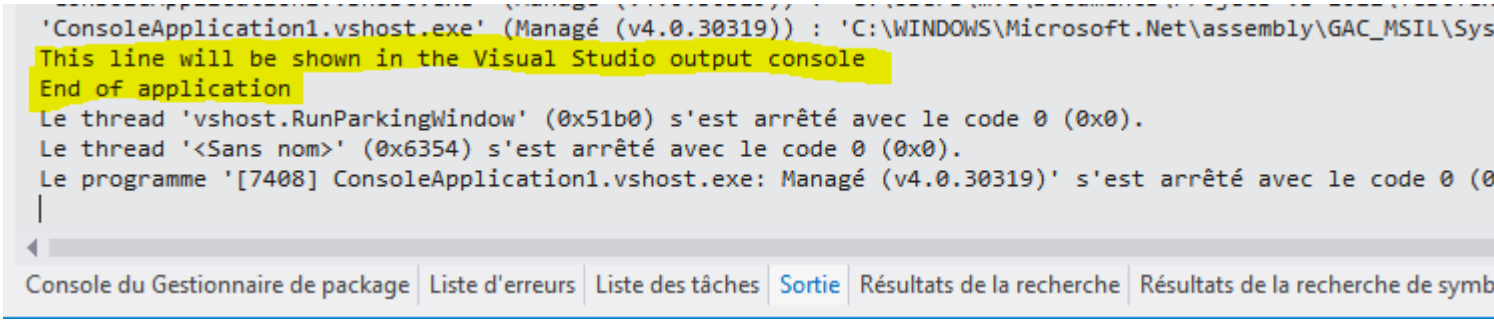
Il primo vantaggio della classe Debug è che produce codice solo se si crea l'applicazione in modalità Debug. Quando si crea l'applicazione in modalità di rilascio, non verrà generato alcun codice dalle chiamate di debug.

Examples

Debug nella console

```
Module Module1
    Sub Main()
        Debug.WriteLine("This line will be shown in the Visual Studio output console")

        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub
End Module
```

produrrà:



Indentazione del tuo output di debug

```
Module Module1

    Sub Main()
        Debug.WriteLine("Starting aplication")

        Debug.Indent()
        LoopAndDoStuff(5)
        Debug.Unindent()
```

```
        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub

    Sub LoopAndDoStuff(Iterations As Integer)
        Dim x As Integer = 0
        Debug.WriteLine("Starting loop")
        Debug.Indent()
        For i As Integer = 0 To Iterations - 1
            Debug.Write("Iteration " & (i + 1).ToString() & " of " & Iterations.ToString() &
": Value of X: ")
            x += (x + 1)
            Debug.WriteLine(x.ToString())
        Next
        Debug.Unindent()
        Debug.WriteLine("Loop is over")
    End Sub
End Module
```

produrrà:

```
'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Syster
Starting aplication
    Starting loop
        Iteration 1 of 5: Value of X: 1
        Iteration 2 of 5: Value of X: 3
        Iteration 3 of 5: Value of X: 7
        Iteration 4 of 5: Value of X: 15
        Iteration 5 of 5: Value of X: 31
    Loop is over
End of application
Le thread 'vshost.RunParkingWindow' (0x2764) s'est arrêté avec le code 0 (0x0).
Le thread '<Sans nom>' (0xe74) s'est arrêté avec le code 0 (0x0).
Le programme '[8316] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0x0
```

onsole du Gestionnaire de package | Liste d'erreurs | Liste des tâches | Sortie | Résultats de la recherche | Résultats de la recherche de symbole

Debug in un file di testo

All'inizio della tua applicazione, devi aggiungere un TextWriterTraceListener all'elenco Listeners della classe Debug.

```
Module Module1

    Sub Main()
        Debug.Listeners.Add(New TextWriterTraceListener("Debug of " & DateTime.Now.ToString()
& ".txt"))

        Debug.WriteLine("Starting aplication")

        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub
```

```
End Module
```

Tutto il codice di debug prodotto verrà riprodotto nella console di Visual Studio E nel file di testo che hai scelto.

Se il file è sempre lo stesso:

```
Debug.Listeners.Add(New TextWriterTraceListener("Debug.txt"))
```

L'output verrà aggiunto al file ogni volta che AND un nuovo file che inizia con un GUID, quindi verrà generato il nome del file.

Leggi Debug della tua applicazione online: https://riptutorial.com/it/vb-net/topic/8631/debug-della-tua-applicazione

Capitolo 12: Dichiarazione di variabili

Sintassi

- Contatore pubblico As Integer
- _Counter privato come numero intero
- Contatore dim Come numero intero

Examples

Dichiarare e assegnare una variabile usando un tipo primitivo

Le variabili in Visual Basic sono dichiarate usando la parola chiave `Dim` . Ad esempio, questo dichiara una nuova variabile chiamata `counter` con il tipo di dati `Integer` :

```
Dim counter As Integer
```

Una dichiarazione di variabili può anche includere un modificatore di accesso , come `Public` , `Protected` , `Friend` o `Private` . Questo funziona in congiunzione con l' ambito della variabile per determinarne l'accessibilità.

Modificatore d'accesso	Senso
Pubblico	Tutti i tipi che possono accedere al tipo di allegato
protetta	Solo la classe che racchiude e quelli che ne ereditano
Amico	Tutti i tipi nello stesso assembly che possono accedere al tipo di allegato
Amico protetto	La classe di inclusione e i suoi ereditari, *oi* tipi nello stesso assembly che possono accedere alla classe che li include
Privato	Solo il tipo di allegato
Statico	Solo su variabili locali e si inizializza solo una volta.

Come abbreviazione, la parola chiave `Dim` può essere sostituita con il modificatore di accesso nella dichiarazione della variabile:

```
Public TotalItems As Integer
Private counter As Integer
```

I tipi di dati supportati sono descritti nella tabella seguente:

genere	Alias	Allocazione della memoria	Esempio
SByte	N / A	1 byte	`Dim example As SByte = 10`
Int16	Corto	2 byte	`Dim example As Short = 10`
Int32	Numero intero	4 byte	`Dim example As Integer = 10`
Int64	Lungo	8 byte	`Dim example As Long = 10`
singolo	N / A	4 byte	`Dim example As Single = 10.95`
Doppio	N / A	8 byte	`Dim example As Double = 10.95`
Decimale	N / A	16 byte	`Dim example As Decimal = 10.95`
booleano	N / A	Dettato implementando la piattaforma	`Dim example As Boolean = True`
carbonizzare	N / A	2 byte	`Dim example As Char = "A"C`
Stringa	N / A	$20 + \left\lfloor \frac{length}{2} \right\rfloor * 4bytes$ fonte	`Dim example As String = "Stack Overflow"`
Appuntamento	Data	8 byte	`Dim example As Date = Date.Now`
Byte	N / A	1 byte	`Dim example As Byte = 10`
UInt16	USHORT	2 byte	`Dim example As UShort = 10`
UInt32	UInteger	4 byte	`Dim example As UInteger = 10`
UInt64	Ulong	8 byte	`Dim example As ULong = 10`
Oggetto	N / A	4 byte architettura a 32 bit, 8 byte architettura a 64 bit	`Dim example As Object = Nothing`

Esistono anche identificatori di dati e caratteri di tipo letterale utilizzabili in sostituzione del tipo testuale o per forzare il tipo letterale:

Digitare (o pseudonimo)	Carattere del tipo di identificatore	Carattere di tipo letterale
Corto	N / A	`example = 10S`

Digitare (o pseudonimo)	Carattere del tipo di identificatore	Carattere di tipo letterale
Numero intero	`Dim example%`	`example = 10%` O `example = 10I`
Lungo	`Dim example&`	`example = 10&` O `example = 10L`
singolo	`Dim example!`	`example = 10!` O `example = 10F`
Doppio	`Dim example#`	`example = 10#` O `example = 10R`
Decimale	`Dim example@`	`example = 10@` O `example = 10D`
carbonizzare	N / A	`example = "A"C`
Stringa	`Dim example$`	N / A
USHORT	N / A	`example = 10US`
UInteger	N / A	`example = 10UI`
Ulong	N / A	`example = 10UL`

I suffissi integrali sono anche utilizzabili con prefissi esadecimali (& H) o ottali (& O):
`example = &H8000S` O `example = &O77&`

Gli oggetti Date (Time) possono anche essere definiti usando la sintassi letterale:
`Dim example As Date = #7/26/2016 12:8 PM#`

Una volta che una variabile viene dichiarata esisterà entro la portata del tipo contenente, `Sub` o `Function` dichiarati, ad esempio:

```
Public Function IncrementCounter() As Integer
    Dim counter As Integer = 0
    counter += 1

    Return counter
End Function
```

La variabile di conteggio esisterà solo fino alla `End Function` e quindi sarà fuori portata. Se questa variabile contatore è necessaria al di fuori della funzione, sarà necessario definirla a livello di classe / struttura o modulo.

```
Public Class ExampleClass

    Private _counter As Integer

    Public Function IncrementCounter() As Integer
```

```
        _counter += 1
        Return _counter
    End Function

End Class
```

In alternativa, è possibile utilizzare il modificatore `Static` (da non confondere con `Shared`) per consentire a una variabile locale di mantenere il valore tra le chiamate del relativo metodo di inclusione:

```
Function IncrementCounter() As Integer
    Static counter As Integer = 0
    counter += 1

    Return counter
End Function
```

Livelli di dichiarazione - Variabili locali e membri

Variabili locali : quelle dichiarate all'interno di una procedura (subroutine o funzione) di una classe (o altra struttura). In questo esempio, `exampleLocalVariable` è una variabile locale dichiarata all'interno di `ExampleFunction()` :

```
Public Class ExampleClass1

    Public Function ExampleFunction() As Integer
        Dim exampleLocalVariable As Integer = 3
        Return exampleLocalVariable
    End Function

End Class
```

La parola chiave `Static` consente di mantenere una variabile locale e di mantenerne il valore dopo la terminazione (dove solitamente le variabili locali cessano di esistere al termine della procedura di contenimento).

In questo esempio, la console è `024` . Su ogni chiamata a `ExampleSub()` da `Main()` la variabile statica mantiene il valore che aveva alla fine della precedente chiamata:

```
Module Module1

    Sub Main()
        ExampleSub()
        ExampleSub()
        ExampleSub()
    End Sub

    Public Sub ExampleSub()
        Static exampleStaticLocalVariable As Integer = 0
        Console.Write(exampleStaticLocalVariable.ToString)
        exampleStaticLocalVariable += 2
    End Sub

End Module
```

Variabili membro - Dichiarate al di fuori di qualsiasi procedura, a livello di classe (o altra struttura). Possono essere **variabili di istanza** , in cui ogni istanza della classe contenitore ha una propria copia distinta di tale variabile, o **variabili** `Shared` , che esistono come una singola variabile associata alla classe stessa, indipendentemente da qualsiasi istanza.

Qui, `ExampleClass2` contiene due variabili membro. Ogni istanza di `ExampleClass2` ha una specifica `ExampleInstanceVariable` cui è possibile accedere tramite il riferimento alla classe. È tuttavia possibile accedere alla variabile condivisa `ExampleSharedVariable` utilizzando il nome della classe:

```
Module Module1

    Sub Main()

        Dim instance1 As ExampleClass4 = New ExampleClass4
        instance1.ExampleInstanceVariable = "Foo"

        Dim instance2 As ExampleClass4 = New ExampleClass4
        instance2.ExampleInstanceVariable = "Bar"

        Console.WriteLine(instance1.ExampleInstanceVariable)
        Console.WriteLine(instance2.ExampleInstanceVariable)
        Console.WriteLine(ExampleClass4.ExampleSharedVariable)

    End Sub

    Public Class ExampleClass4

        Public ExampleInstanceVariable As String
        Public Shared ExampleSharedVariable As String = "FizzBuzz"

    End Class

End Module
```

Esempio di modificatori di accesso

Nell'esempio seguente, si consideri una soluzione che ospita due progetti: **ConsoleApplication1** e **SampleClassLibrary** . Il primo progetto avrà le classi **SampleClass1** e **SampleClass2** . Il secondo avrà **SampleClass3** e **SampleClass4** . In altre parole, abbiamo due assembly con due classi ciascuno. **ConsoleApplication1** ha un riferimento a **SampleClassLibrary** .

Guarda come **SampleClass1.MethodA** interagisce con altre classi e metodi.

SampleClass1.vb:

```
Imports SampleClassLibrary

Public Class SampleClass1
    Public Sub MethodA()
        'MethodA can call any of the following methods because
        'they all are in the same scope.
        MethodB()
        MethodC()
        MethodD()
        MethodE()
```

```
    'Sample2 is defined as friend. It is accessible within
    'the type itself and all namespaces and code within the same assembly.
    Dim class2 As New SampleClass2()
    class2.MethodA()
    'class2.MethodB() 'SampleClass2.MethodB is not accessible because
                      'this method is private. SampleClass2.MethodB
                      'can only be called from SampleClass2.MethodA,
                      'SampleClass2.MethodC, SampleClass2.MethodD
                      'and SampleClass2.MethodE
    class2.MethodC()
    'class2.MethodD() 'SampleClass2.MethodD is not accessible because
                      'this method is protected. SampleClass2.MethodD
                      'can only be called from any class that inherits
                      'SampleClass2, SampleClass2.MethodA, SampleClass2.MethodC,
                      'SampleClass2.MethodD and SampleClass2.MethodE
    class2.MethodE()

    Dim class3 As New SampleClass3() 'SampleClass3 resides in other
                                     'assembly and is defined as public.
                                     'It is accessible anywhere.
    class3.MethodA()
    'class3.MethodB() 'SampleClass3.MethodB is not accessible because
                      'this method is private. SampleClass3.MethodB can
                      'only be called from SampleClass3.MethodA,
                      'SampleClass3.MethodC, SampleClass3.MethodD
                      'and SampleClass3.MethodE

    'class3.MethodC() 'SampleClass3.MethodC is not accessible because
                      'this method is friend and resides in another assembly.
                      'SampleClass3.MethodC can only be called anywhere from the
                      'same assembly, SampleClass3.MethodA, SampleClass3.MethodB,
                      'SampleClass3.MethodD and SampleClass3.MethodE

    'class4.MethodD() 'SampleClass3.MethodE is not accessible because
                      'this method is protected friend. SampleClass3.MethodD
                      'can only be called from any class that resides inside
                      'the same assembly and inherits SampleClass3,
                      'SampleClass3.MethodA, SampleClass3.MethodB,
                      'SampleClass3.MethodC and SampleClass3.MethodD

    'Dim class4 As New SampleClass4() 'SampleClass4 is not accessible because
                                      'it is defined as friend and resides in
                                      'other assembly.
End Sub

Private Sub MethodB()
    'Doing MethodB stuff...
End Sub

Friend Sub MethodC()
    'Doing MethodC stuff...
End Sub

Protected Sub MethodD()
    'Doing MethodD stuff...
End Sub

Protected Friend Sub MethodE()
    'Doing MethodE stuff...
End Sub
```

```
End Class
```

SampleClass2.vb:

```
Friend Class SampleClass2
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub

    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass3.vb:

```
Public Class SampleClass3
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub
    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass4.vb:

```
Friend Class SampleClass4
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub
    Private Sub MethodB()
        'Doing MethodB stuff...
```

```
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

Leggi Dichiarazione di variabili online: https://riptutorial.com/it/vb-net/topic/3366/dichiarazione-di-variabili

Capitolo 13: Digita la conversione

Sintassi

- CBool (espressione)
- CByte (espressione)
- CChar (espressione)
- CDate (espressione)
- CDbl (espressione)
- CDec (espressione)
- CInt (espressione)
- CLng (espressione)
- CObj (espressione)
- CSByte (espressione)
- CShort (espressione)
- CSng (espressione)
- CStr (espressione)
- CUInt (espressione)
- CULng (espressione)
- CUShort (espressione)

Parametri

Nome della funzione	Intervallo per argomento Espressione
CBool	Qualsiasi carattere Char o stringa o espressione numerica valida
CByte	Da 0 a 255 (senza segno); le parti frazionarie sono arrotondate.
CChar	Qualsiasi espressione Char o String valida; viene convertito solo il primo carattere di una stringa; il valore può essere compreso tra 0 e 65535 (senza segno).

Examples

Conversione del testo di Textbox in un numero intero

Da MSDN

> Utilizzare la funzione CInt per fornire conversioni da qualsiasi altro tipo di dati a un sottotipo Integer. Ad esempio, CInt forza l'aritmetica dei numeri interi quando si verifica normalmente l'aritmetica di valuta, precisione singola o doppia precisione.

Supponendo che tu abbia 1 pulsante e 2 caselle di testo. Se scrivi su textbox1.text `5.5` e su textbox2.text `10` .

Se hai questo codice:

```
Dim result = textbox1.text + textbox2.text
MsgBox("Result: " & result)
'It will output
5.510
```

Per aggiungere i valori delle 2 caselle di testo devi convertire i loro valori in `Int` usando la `CInt(expression)` .

```
Dim result = CInt(textbox1.text) + CInt(textbox2.text)
MsgBox("Result: " & result)
'It will output
16
```

> Nota: quando la parte frazionaria di un valore è esattamente 0.5, la funzione CInt arrotonda al numero pari più vicino. Ad esempio, da **0,5 a 0** , da **1,5 a 2 e da 3,5 a 4** . Lo scopo di arrotondare al numero pari più vicino è compensare una distorsione che potrebbe accumularsi quando molti numeri vengono aggiunti insieme.

Leggi Digita la conversione online: https://riptutorial.com/it/vb-net/topic/4681/digita-la-conversione

Capitolo 14: dizionari

introduzione

Un dizionario rappresenta una raccolta di chiavi e valori. Vedi la classe MSDN Dictionary (Tkey, TValue) .

Examples

Passa attraverso un dizionario e stampa tutte le voci

Ogni coppia nel dizionario è un'istanza di `KeyValuePair` con gli stessi parametri del dizionario. Quando esegui il looping del dizionario con `For Each` , ogni iterazione ti assegnerà una delle coppie di valori-chiave memorizzate nel dizionario.

```
For Each kvp As KeyValuePair(Of String, String) In currentDictionary
  Console.WriteLine("{0}: {1}", kvp.Key, kvp.Value)
Next
```

Crea un dizionario pieno di valori

```
Dim extensions As New Dictionary(Of String, String) _
  from { { "txt", "notepad" },
  { "bmp", "paint" },
  { "doc", "winword" } }
```

Questo crea un dizionario e lo riempie immediatamente con tre KeyValuePairs.

Puoi anche aggiungere nuovi valori in seguito utilizzando il metodo Aggiungi:

```
extensions.Add("png", "paint")
```

Si noti che la chiave (il primo parametro) deve essere univoca nel dizionario, altrimenti verrà generata un'eccezione.

Ottenere un valore di dizionario

Puoi ottenere il valore di una voce nel dizionario usando la proprietà 'Articolo':

```
Dim extensions As New Dictionary(Of String, String) From {
    { "txt", "notepad" },
    { "bmp", "paint" },
    { "doc", "winword" }
}

Dim program As String = extensions.Item("txt") 'will be "notepad"
```

```
' alternative syntax as Item is the default property (a.k.a. indexer)
Dim program As String = extensions("txt") 'will be "notepad"

' other alternative syntax using the (rare)
' dictionary member access operator (a.k.a. bang operator)
Dim program As String = extensions!txt 'will be "notepad"
```

Se la chiave non è presente nel dizionario, verrà lanciata una KeyNotFoundException.

Verifica della chiave già presente nel dizionario - riduzione dei dati

Il metodo `ConstainsKey` è il modo per sapere se una chiave esiste già nel Dizionario.

Questo è utile per la riduzione dei dati. Nell'esempio seguente, ogni volta che incontriamo una nuova parola, la aggiungiamo come chiave nel dizionario, altrimenti incrementiamo il contatore per questa parola specifica.

```
 Dim dic As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)

 Dim words As String() = Split(<big text source>," ", -1, CompareMethod.Binary)

 For Each str As String In words
     If dic.ContainsKey(str) Then
         dic(str) += 1
     Else
         dic.Add(str, 1)
     End If
 Next
```

Esempio di riduzione XML: acquisizione di tutti i nomi e l'occorrenza dei nodi figlio in un ramo di un documento XML

```
Dim nodes As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)
Dim xmlsrc = New XmlDocument()
xmlsrc.LoadXml(<any text stream source>)

For Each xn As XmlNode In xmlsrc.FirstChild.ChildNodes 'selects the proper parent
    If nodes.ContainsKey(xn.Name) Then
        nodes(xn.Name) += 1
    Else
        nodes.Add(xn.Name, 1)
    End If
Next
```

Leggi dizionari online: https://riptutorial.com/it/vb-net/topic/2165/dizionari

Capitolo 15: elenchi

Sintassi

- List.Add (item As Type)
- List.RemoveRange (index As Integer, count As Integer)
- List.Remove (index As Integer)
- List.AddRange (raccolta)
- List.Find (corrisponde a Predicate (di String))
- List.Insert (indice come numero intero, elemento come tipo)
- List.Contains (item as Type)

Examples

Crea una lista

Gli elenchi possono essere compilati con qualsiasi tipo di dati necessario, con il formato

```
Dim aList as New List(Of Type)
```

Per esempio:

Crea una nuova lista vuota di stringhe

```
Dim aList As New List(Of String)
```

Crea un nuovo elenco di stringhe e popola con alcuni dati

VB.NET 2005/2008:

```
Dim aList as New List(Of String)(New String() {"one", "two", "three"})
```

VB.NET 2010:

```
Dim aList as New List(Of String) From {"one", "two", "three"}
```

-

VB.NET 2015:

```
Dim aList as New List(Of String)(New String() {"one", "two", "three"})
```

NOTA:

Se si riceve quanto segue quando viene eseguito il codice:

Il riferimento non impostato su un'istanza di un oggetto.

Assicurati di dichiarare come `New` cioè `Dim aList as New List(Of String)` o se dichiari senza il `New` , assicurati di impostare l'elenco su un nuovo elenco - `Dim aList as List(Of String) = New List(Of String)`

Aggiungi elementi a un elenco

```
Dim aList as New List(Of Integer)
aList.Add(1)
aList.Add(10)
aList.Add(1001)
```

Aggiungi più di un elemento per volta usa **AddRange** . Aggiunge sempre alla fine della lista

```
Dim blist as New List(of Integer)
blist.AddRange(alist)


Dim aList as New List(of String)
alist.AddRange({"one", "two", "three"})
```

Per aggiungere elementi al centro dell'elenco, utilizzare **Inserisci**

Inserisci posizionerà l'elemento nell'indice e rinumererà gli elementi rimanenti

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("three")
alist(0) = "one"
alist(1) = "three"
alist.Insert(1,"two")
```

Nuova uscita:

```
alist(0) = "one"
alist(1) = "two"
alist(2) = "three"
```

Rimuovi elementi da una lista

```
Dim aList As New List(Of String)
aList.Add("Hello")
aList.Add("Delete Me!")
aList.Add("World")

'Remove the item from the list at index 1
aList.RemoveAt(1)

'Remove a range of items from a list, starting at index 0, for a count of 1)
'This will remove index 0, and 1!
aList.RemoveRange(0, 1)
```

```
'Clear the entire list
alist.Clear()
```

Recupera elementi da un elenco

```
Dim aList as New List(Of String)
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList(0)
```

`output` :

> Ciao mondo

Se non si conosce l'indice dell'elemento o si conosce solo parte della stringa, utilizzare il metodo **Trova** o **Trova tutto**

```
Dim aList as New List(Of String)
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList.Find(Function(x) x.StartWith("Hello"))
```

`output` :

> Ciao mondo

Il metodo **FindAll** restituisce una nuova lista (di String)

```
Dim aList as New List(Of String)
aList.Add("Hello, Test")
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList.FindAll(Function(x) x.Contains("Test"))
```

> output (0) = "Ciao, prova"
>
> output (1) = "Test"

Passa tra gli oggetti in lista

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("two")
aList.Add("three")

For Each str As String in aList
    System.Console.WriteLine(str)
Next
```

Produce il seguente risultato:

```
one
two
three
```

Un'altra opzione, sarebbe quella di scorrere attraverso l'uso dell'indice di ciascun elemento:

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("two")
aList.Add("three")

For i = 0 to aList.Count - 1 'We use "- 1" because a list uses 0 based indexing.
    System.Console.WriteLine(aList(i))
Next
```

Controlla se l'elemento esiste in una lista

```
    Sub Main()
        Dim People = New List(Of String)({"Bob Barker", "Ricky Bobby", "Jeff Bridges"})
        Console.WriteLine(People.Contains("Rick James"))
        Console.WriteLine(People.Contains("Ricky Bobby"))
        Console.WriteLine(People.Contains("Barker"))
        Console.Read
    End Sub
```

Produce il seguente risultato:

```
False
True
False
```

Leggi elenchi online: https://riptutorial.com/it/vb-net/topic/3636/elenchi

Capitolo 16: enum

Examples

Definizione di enum

Un enum è un insieme di costanti logicamente correlate.

```
Enum Size
    Small
    Medium
    Large
End Enum

Public Sub Order(shirtSize As Size)
    Select Case shirtSize
        Case Size.Small
            ' ...
        Case Size.Medium
            ' ...
        Case Size.Large
            ' ...
    End Select
End Sub
```

Inizializzazione del membro

Ciascun membro di enum può essere inizializzato con un valore. Se non si specifica un valore per un membro, per impostazione predefinita viene inizializzato su 0 (se è il primo membro nell'elenco dei membri) o su un valore maggiore di 1 rispetto al valore del membro precedente.

```
Module Module1

    Enum Size
        Small
        Medium = 3
        Large
    End Enum

    Sub Main()
        Console.WriteLine(Size.Small)    ' prints 0
        Console.WriteLine(Size.Medium)   ' prints 3
        Console.WriteLine(Size.Large)    ' prints 4

        ' Waits until user presses any key
        Console.ReadKey()
    End Sub

End Module
```

L'attributo Flags

Con l'attributo `<Flags>` , l'enumerazione diventa una serie di flag. Questo attributo consente di assegnare più valori a una variabile enum. I membri di un enumerazione di bandiere devono essere inizializzati con le potenze di 2 (1, 2, 4, 8 ...).

```
Module Module1

    <Flags>
    Enum Material
        Wood = 1
        Plastic = 2
        Metal = 4
        Stone = 8
    End Enum

    Sub Main()
        Dim houseMaterials As Material = Material.Wood Or Material.Stone
        Dim carMaterials as Material = Material.Plastic Or Material.Metal
        Dim knifeMaterials as Material = Material.Metal

        Console.WriteLine(houseMaterials.ToString()) 'Prints "Wood, Stone"
        Console.WriteLine(CType(carMaterials, Integer)) 'Prints 6
    End Sub

End Module
```

HasFlag ()

Il metodo `HasFlag()` può essere usato per verificare se è impostato un flag.

```
Module Module1

    <Flags>
    Enum Material
        Wood = 1
        Plastic = 2
        Metal = 4
        Stone = 8
    End Enum

    Sub Main()
        Dim houseMaterials As Material = Material.Wood Or Material.Stone

        If houseMaterials.HasFlag(Material.Stone) Then
            Console.WriteLine("the house is made of stone")
        Else
            Console.WriteLine("the house is not made of stone")
        End If
    End Sub

End Module
```

Per ulteriori informazioni sull'attributo Flags e su come dovrebbe essere utilizzato, consultare la documentazione ufficiale di Microsoft .

Analisi delle stringhe

Un'istanza Enum può essere creata analizzando una rappresentazione stringa dell'Enum.

```
Module Module1

    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size =  DirectCast([Enum].Parse(GetType(Size), "Medium"), Size)

        ' Prints 'Medium'
        Console.WriteLine(shirtSize.ToString())

        ' Waits until user presses any key
        Console.ReadKey()
    End Sub

End Module
```

Vedi anche: Analizza una stringa su un valore Enum in VB.NET

GetNames ()

Restituisce i nomi delle costanti nell'enumerazione specificata come array di stringhe:

```
Module Module1

  Enum Size
    Small
    Medium
    Large
  End Enum

  Sub Main()
    Dim sizes = [Enum].GetNames(GetType(Size))

    For Each size In sizes
      Console.WriteLine(size)
      Next
  End Sub

End Module
```

Produzione:

Piccolo

medio

Grande

GetValues ()

'Questo metodo è utile per iterare i valori di Enum'

```
Enum Animal
    Dog = 1
    Cat = 2
    Frog = 4
End Enum

Dim Animals = [Enum].GetValues(GetType(Animal))

For Each animal in Animals
    Console.WriteLine(animal)
Next
```

stampe:

1

2

4

Accordare()

Il metodo `ToString` su una enumerazione restituisce il nome della stringa dell'enumerazione. Per esempio:

```
Module Module1
    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size = Size.Medium
        Dim output As String = shirtSize.ToString()
        Console.WriteLine(output)   ' Writes "Medium"
    End Sub
End Module
```

Se, tuttavia, si desidera la rappresentazione della stringa del valore intero effettivo dell'enumerazione, è possibile eseguire il cast dell'enumerazione su un numero `Integer` e quindi chiamare `ToString` :

```
Dim shirtSize As Size = Size.Medium
Dim output As String = CInt(shirtSize).ToString()
Console.WriteLine(output)   ' Writes "1"
```

Determina se un Enum ha FlagsAttribute specificato o meno

Il prossimo esempio può essere usato per determinare se un enumerazione ha specificato **FlagsAttribute** . La metodologia utilizzata è basata su **Reflection** .

Questo esempio darà un `True` risultato:

```
Dim enu As [Enum] = New FileAttributes()
Dim hasFlags As Boolean = enu.GetType().GetCustomAttributes(GetType(FlagsAttribute),
inherit:=False).Any()
Console.WriteLine("{0} Enum has FlagsAttribute?: {1}", enu.GetType().Name, hasFlags)
```

Questo esempio darà un risultato `False` :

```
Dim enu As [Enum] = New ConsoleColor()
Dim hasFlags As Boolean = enu.GetType().GetCustomAttributes(GetType(FlagsAttribute),
inherit:=False).Any()
Console.WriteLine("{0} Enum has FlagsAttribute?: {1}", enu.GetType().Name, hasFlags)
```

Possiamo progettare un metodo di estensione di utilizzo generico come questo:

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Function HasFlagsAttribute(ByVal sender As [Enum]) As Boolean
    Return sender.GetType().GetCustomAttributes(GetType(FlagsAttribute), inherit:=False).Any()
End Function
```

Esempio di utilizzo:

```
Dim result As Boolean = (New FileAttributes).HasFlagsAttribute()
```

For-each flag (flag iteration)

In alcuni scenari molto specifici sentiremmo la necessità di eseguire un'azione specifica per ogni flag dell'enumerazione di origine.

Possiamo scrivere un semplice metodo di estensione *generico* per realizzare questo compito.

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Sub ForEachFlag(Of T)(ByVal sender As [Enum],
                             ByVal action As Action(Of T))

    For Each flag As T In sender.Flags(Of T)
        action.Invoke(flag)
    Next flag

End Sub
```

Esempio di utilizzo:

```
Dim flags As FileAttributes = (FileAttributes.ReadOnly Or FileAttributes.Hidden)

flags.ForEachFlag(Of FileAttributes)(
      Sub(ByVal x As FileAttributes)
```

```
          Console.WriteLine(x.ToString())
      End Sub)
```

Determina la quantità di flag in una combinazione di flag

L'esempio successivo è destinato a contare la quantità di flag nella combinazione di flag specificata.

L'esempio è fornito come metodo di estensione:

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Function CountFlags(ByVal sender As [Enum]) As Integer
    Return sender.ToString().Split(","c).Count()
End Function
```

Esempio di utilizzo:

```
Dim flags As FileAttributes = (FileAttributes.Archive Or FileAttributes.Compressed)
Dim count As Integer = flags.CountFlags()
Console.WriteLine(count)
```

Trova il valore più vicino in un Enum

Il prossimo codice illustra come trovare il valore più vicino di un **Enum** .

Per prima cosa definiamo questo **Enum** che servirà a specificare i criteri di ricerca (direzione della ricerca)

```
Public Enum EnumFindDirection As Integer
    Nearest = 0
    Less = 1
    LessOrEqual = 2
    Greater = 3
    GreaterOrEqual = 4
End Enum
```

E ora implementiamo l'algoritmo di ricerca:

```
<DebuggerStepThrough>
Public Shared Function FindNearestEnumValue(Of T)(ByVal value As Long,
                                                  ByVal direction As EnumFindDirection) As T

    Select Case direction

        Case EnumFindDirection.Nearest
            Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                    Order By Math.Abs(value - Convert.ToInt64(enumValue))
                    ).FirstOrDefault

        Case EnumFindDirection.Less
            If value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).First) Then
```

```
                Return [Enum].GetValues(GetType(T)).Cast(Of T).FirstOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) < value
                        ).LastOrDefault
            End If

        Case EnumFindDirection.LessOrEqual
            If value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).First) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).FirstOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) <= value
                        ).LastOrDefault
            End If

        Case EnumFindDirection.Greater
            If value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) > value
                        ).FirstOrDefault
            End If

        Case EnumFindDirection.GreaterOrEqual
            If value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) >= value
                        ).FirstOrDefault
            End If

    End Select

End Function
```

Esempio di utilizzo:

```
Public Enum Bitrate As Integer
    Kbps128 = 128
    Kbps192 = 192
    Kbps256 = 256
    Kbps320 = 320
End Enum

Dim nearestValue As Bitrate = FindNearestEnumValue(Of Bitrate)(224,
EnumFindDirection.GreaterOrEqual)
```

Leggi enum online: https://riptutorial.com/it/vb-net/topic/1809/enum

Capitolo 17: Funzionalità di Visual Basic 14.0

introduzione

Visual Basic 14 è la versione di Visual Basic fornita come parte di Visual Studio 2015.

Questa versione è stata riscritta da zero in circa 1,3 milioni di righe di VB. Sono state aggiunte molte funzionalità per rimuovere le irritazioni comuni e per rendere più comuni i modelli di codifica comuni.

Il numero di versione di Visual Basic passava direttamente da 12 a 14, saltando 13. Ciò è stato fatto per mantenere VB in linea con la numerazione della versione di Visual Studio stesso.

Examples

Operatore condizionale nullo

Per evitare il controllo null dettagliato, il `?.` operatore è stato introdotto nella lingua.

La vecchia sintassi verbosa:

```
If myObject IsNot Nothing AndAlso myObject.Value >= 10 Then
```

Può essere ora sostituito dal conciso:

```
If myObject?.Value >= 10 Then
```

Il `?` l'operatore è particolarmente potente quando si ha una catena di proprietà. Considera quanto segue:

```
Dim fooInstance As Foo = Nothing
Dim s As String
```

Normalmente dovresti scrivere qualcosa come questo:

```
If fooInstance IsNot Nothing AndAlso fooInstance.BarInstance IsNot Nothing Then
    s = fooInstance.BarInstance.Baz
Else
    s = Nothing
End If
```

Ma con il `?` l'operatore può essere sostituito solo con:

```
s = fooInstance?.BarInstance?.Baz
```

Nome dell'operatore

L'operatore `NameOf` risolve spazio dei nomi, tipi, variabili e nomi dei membri in fase di compilazione e li sostituisce con l'equivalente della stringa.

Uno dei casi d'uso:

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException("variable")
End Sub
```

La vecchia sintassi espone il rischio di rinominare la variabile e lasciare la stringa hardcoded al valore errato.

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException(NameOf(variable))
End Sub
```

Con `NameOf` , la sola rinomina della variabile solleverà un errore del compilatore. Ciò consentirà inoltre allo strumento di ridenominazione di rinominare entrambi con un solo sforzo.

L'operatore `NameOf` utilizza solo l'ultimo componente del riferimento tra parentesi. Questo è importante quando si gestisce qualcosa come namespace nell'operatore `NameOf` .

```
Imports System

Module Module1
    Sub WriteIO()
        Console.WriteLine(NameOf(IO)) 'displays "IO"
        Console.WriteLine(NameOf(System.IO)) 'displays "IO"
    End Sub
End Module
```

L'operatore usa anche il nome del riferimento che viene digitato senza risolvere alcun cambiamento di nome nelle importazioni. Per esempio:

```
Imports OldList = System.Collections.ArrayList

Module Module1
    Sub WriteList()
        Console.WriteLine(NameOf(OldList)) 'displays "OldList"
        Console.WriteLine(NameOf(System.Collections.ArrayList)) 'displays "ArrayList"
    End Sub
End Module
```

Interpolazione a stringa

Questa nuova funzione rende più leggibile la concatenazione di stringhe. Questa sintassi verrà compilata alla sua chiamata `String.Format` equivalente.

Senza interpolazione delle stringhe:

```
String.Format("Hello, {0}", name)
```

Con interpolazione delle stringhe:

```
$"Hello, {name}"
```

Le due righe sono equivalenti e vengono entrambi compilati in una chiamata a `String.Format` .

Come in `String.Format` , le parentesi possono contenere qualsiasi singola espressione (chiamata a un metodo, proprietà, un operatore coalescente null eccetera).

String Interpolation è il metodo preferito su `String.Format` perché impedisce che si verifichino alcuni errori di runtime. Si consideri la seguente riga `String.Format` :

```
String.Format("The number of people is {0}/{1}", numPeople)
```

Questo verrà compilato, ma causerà un errore di runtime in quanto il compilatore non controlla che il numero di argomenti corrisponda ai segnaposto.

Auto-Proprietà di sola lettura

Le proprietà di sola lettura erano sempre possibili in VB.NET in questo formato:

```
Public Class Foo

  Private _MyProperty As String = "Bar"

  Public ReadOnly Property MyProperty As String
    Get
      Return _MyProperty
    End Get
  End Property

End Class
```

La nuova versione di Visual Basic consente una breve mano per la dichiarazione delle proprietà in questo modo:

```
Public Class Foo

  Public ReadOnly Property MyProperty As String = "Bar"

End Class
```

L'effettiva implementazione generata dal compilatore è esattamente la stessa per entrambi gli esempi. Il nuovo metodo per scriverlo è solo una mano breve. Il compilatore genererà comunque un campo privato con il formato: `_<PropertyName>` per supportare la proprietà di sola lettura.

Moduli e interfacce parziali

Simile alle classi parziali, la nuova versione di Visual Basic ora è in grado di gestire moduli parziali e interfacce parziali. La sintassi e il comportamento sono esattamente gli stessi di quelli per le classi parziali.

Un esempio di modulo parziale:

```
Partial Module Module1
  Sub Main()
    Console.Write("Ping -> ")
    TestFunktion()
  End Sub
End Module

Partial Module Module1
  Private Sub TestFunktion()
    Console.WriteLine("Pong")
  End Sub
End Module
```

E un'interfaccia parziale:

```
Partial Interface Interface1
  Sub Methode1()
End Interface

Partial Interface Interface1
  Sub Methode2()
End Interface

Public Class Class1
  Implements Interface1
  Public Sub Methode1() Implements Interface1.Methode1
    Throw New NotImplementedException()
  End Sub

  Public Sub Methode2() Implements Interface1.Methode2
    Throw New NotImplementedException()
  End Sub
End Class
```

Proprio come per le classi parziali, le definizioni per i moduli e le interfacce parziali devono trovarsi nello stesso spazio dei nomi e nello stesso assembly. Questo perché le parti parziali dei moduli e delle interfacce vengono unite durante la compilazione e l'assembly compilato non contiene alcuna indicazione che la definizione originale del modulo o dell'interfaccia sia stata divisa.

Letterali stringa multilinea

VB ora consente di ottenere stringhe letterali suddivise su più righe.

Vecchia sintassi:

```
Dim text As String = "Line1" & Environment.NewLine & "Line2"
```

Nuova sintassi:

```
Dim text As String = "Line 1
Line 2"
```

#Region miglioramenti alla direttiva

La direttiva #Region può ora essere collocata all'interno di metodi e può estendersi su metodi, classi e moduli.

```
#Region "A Region Spanning A Class and Ending Inside Of A Method In A Module"
   Public Class FakeClass
    'Nothing to see here, just a fake class.
   End Class

   Module Extensions

    ''' <summary>
    ''' Checks the path of files or directories and returns [TRUE] if it exists.
    ''' </summary>
    ''' <param name="Path">[Sting] Path of file or directory to check.</param>
    ''' <returns>[Boolean]</returns>
    <Extension>
    Public Function PathExists(ByVal Path As String) As Boolean
        If My.Computer.FileSystem.FileExists(Path) Then Return True
        If My.Computer.FileSystem.DirectoryExists(Path) Then Return True
        Return False
    End Function

    ''' <summary>
    ''' Returns the version number from the specified assembly using the assembly's strong
name.
    ''' </summary>
    ''' <param name="Assy">[Assembly] Assembly to get the version info from.</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
#End Region
        Return Split(Split(Assy.FullName, ",")(1), "=")(1)
    End Function
End Module
```

Commenti dopo la continuazione della linea implicita

VB 14.0 introduce la possibilità di aggiungere commenti dopo la continuazione della linea implicita.

```
Dim number =
    From c As Char 'Comment
    In "dj58kwd92n4" 'Comment
    Where Char.IsNumber(c) 'Comment
    Select c 'Comment
```

La gestione delle eccezioni

Durante la codifica, gli errori imprevisti si verificano abbastanza frequentemente, il che richiede il debug e il test. Ma a volte gli errori sono effettivamente previsti e per aggirarlo, c'è il blocco

`Try..Catch..Throw..Finally..End Try` .

Per gestire correttamente un errore, il codice viene inserito in un blocco `Try..Catch` , in cui il `Catch` , come afferma il nome, catturerà tutte le eccezioni che si presentano in questo blocco.

E in caso di eccezione, abbiamo la possibilità di `Throw` l'errore, cioè restituirlo per avvisare l'utente o gestirlo internamente nel codice stesso.

La `Finally` parte è il codice finale che, qualunque sia il risultato sia, se v'è o meno un'eccezione, il codice verrà eseguito prima di uscire del blocco.

Nel caso in cui sia necessario uscire dall'orologio, è possibile utilizzare l'istruzione `Exit Try` . Ma anche qui, il codice nella sezione `Finally` verrà eseguito prima di terminare.

La sintassi è semplice;

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

dove solo la `Try` e `End Try` è obbligatoria. Il resto può essere ignorato, ma come una buona pratica, non comprendono la `Finally` parte, anche se sarebbe essere lasciato vuoto.

Venendo all'eccezione, ci sono diversi tipi di eccezioni che possono essere scoperti. Sono disponibili eccezioni pronte disponibili da .Net Framework, come di seguito;

Classe di eccezione	Breve descrizione
System.IO.IOException	Gestisce gli errori di I / O
System.IndexOutOfRangeException	Si riferisce a un indice di array fuori intervallo
System.ArrayTypeMismatchException	Quando il tipo non corrisponde al tipo di array
System.NullReferenceException	Gestisce gli errori generati dal riferimento a un oggetto nullo.
System.DivideByZeroException	Gestisce gli errori generati dalla divisione di un dividendo con zero.
System.InvalidCastException	Gestisce gli errori generati durante la tipizzazione.
System.OutOfMemoryException	Gestisce gli errori generati da una memoria insufficiente.

Classe di eccezione	Breve descrizione
System.StackOverflowException	Gestisce gli errori generati dallo stack overflow.

Leggi Funzionalità di Visual Basic 14.0 online: https://riptutorial.com/it/vb-net/topic/1501/funzionalita-di-visual-basic-14-0

Capitolo 18: funzioni

introduzione

La funzione è come sub. Ma la funzione restituisce un valore. Una funzione può accettare parametri singoli o multipli.

Examples

Definire una funzione

È davvero facile definire le funzioni.

```
Function GetAreaOfARectangle(ByVal Edge1 As Integer, ByVal Edge2 As Integer) As Integer
    Return Edge1 * Edge2
End Function
```

```
Dim Area As Integer = GetAreaOfARectangle(5, 8)
Console.Writeline(Area) 'Output: 40
```

Definizione di una funzione # 2

```
Function Age(ByVal YourAge As Integer) As String
    Select Case YourAge
        Case Is < 18
            Return("You are younger than 18! You are teen!")
        Case 18 to 64
            Return("You are older than 18 but younger than 65! You are adult!")
        Case Is >= 65
            Return("You are older than 65! You are old!")
    End Select
End Function
```

```
Console.WriteLine(Age(48)) 'Output: You are older than 18 but younger than 65! You are adult!
```

Leggi funzioni online: https://riptutorial.com/it/vb-net/topic/10088/funzioni

Capitolo 19: GDI +

Examples

Crea oggetto grafico

Esistono tre modi per creare un oggetto grafico

1. Dall'evento **Paint**

Ogni volta che il controllo viene ridisegnato (ridimensionato, aggiornato ...) viene chiamato questo evento, utilizzare in questo modo se si desidera che il controllo disegna costantemente sul controllo

```
'this will work on any object's paint event, not just the form
Private Sub Form1_Paint(sender as Object, e as PaintEventArgs) Handles Me.Paint
  Dim gra as Graphics
  gra = e.Graphics
End Sub
```

2. **Crea grafica**

Questo è più spesso utilizzato quando si desidera creare un grafico una volta sul controllo o non si desidera che il controllo si ridisegni

```
Dim btn as New Button
Dim g As Graphics = btn.CreateGraphics
```

3. Da una **grafica esistente**

Utilizzare questo metodo quando si desidera disegnare e modificare un'immagine esistente

```
'The existing image can be from a filename, stream or Drawing.Graphic
Dim image = New Bitmap("C:\TempBit.bmp")
Dim gr As Graphics = Graphics.FromImage(image)
```

Disegna forme

Per iniziare a disegnare una forma è necessario definire un oggetto penna. La `Pen` accetta due parametri:

1. Colore penna o pennello
2. Pen Width

L'oggetto penna viene utilizzato per creare una **struttura** dell'oggetto che si desidera disegnare

Dopo aver definito la penna, è possibile impostare proprietà penna specifiche

```
   Dim pens As New Pen(Color.Purple)
   pens.DashStyle = DashStyle.Dash 'pen will draw with a dashed line
   pens.EndCap = LineCap.ArrowAnchor 'the line will end in an arrow
   pens.StartCap = LineCap.Round 'The line draw will start rounded
   '*Notice* - the Start and End Caps will not show if you draw a closed shape
```

Quindi usa l'oggetto grafico che hai creato per disegnare la forma

```
  Private Sub GraphicForm_Paint(sender As Object, e As PaintEventArgs) Handles MyBase.Paint
    Dim pen As New Pen(Color.Blue, 15) 'Use a blue pen with a width of 15
    Dim point1 As New Point(5, 15) 'starting point of the line
    Dim point2 As New Point(30, 100) 'ending point of the line
    e.Graphics.DrawLine(pen, point1, point2)

    e.Graphics.DrawRectangle(pen, 60, 90, 200, 300) 'draw an outline of the rectangle
```

Per impostazione predefinita, la larghezza della penna è uguale a 1

```
    Dim pen2 as New Pen(Color.Orange) 'Use an orange pen with width of 1
    Dim origRect As New Rectangle(90, 30, 50, 60) 'Define bounds of arc
    e.Graphics.DrawArc(pen2, origRect, 20, 180) 'Draw arc in the rectangle bounds

End Sub
```

Riempi forme

Graphics.FillShapes disegna una forma e la riempie con il colore dato. Le forme di riempimento possono essere utilizzate

1. Strumento `Brush` : per riempire la forma con un colore solido

```
Dim rect As New Rectangle(50, 50, 50, 50)
e.Graphics.FillRectangle(Brushes.Green, rect) 'draws a rectangle that is filled with
green

e.Graphics.FillPie(Brushes.Silver, rect, 0, 180) 'draws a half circle that is filled with
silver
```

2. Strumento `HatchBrush` : per riempire la forma con un motivo

```
Dim hBrush As New HatchBrush(HatchStyle.ZigZag, Color.SkyBlue, Color.Gray)
'creates a HatchBrush Tool with a background color of blue, foreground color of gray,
'and will fill with a zigzag pattern
Dim rectan As New Rectangle(100, 100, 100, 100)
e.Graphics.FillRectangle(hBrush, rectan)
```

3. `LinearGradientBrush` - per riempire la forma con un gradiente

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed,
Color.PaleGreen)
 Dim rect As New Rectangle(50, 50, 200, 200)
 e.Graphics.FillRectangle(lBrush, rect)
```

4. `TextureBrush` : per riempire la forma con un'immagine

È possibile scegliere un'immagine da risorse, una bitmap già definita o da un nome file

```
   Dim textBrush As New TextureBrush(New Bitmap("C:\ColorPic.jpg"))
    Dim rect As New Rectangle(400, 400, 100, 100)
    e.Graphics.FillPie(textBrush, rect, 0, 360)
```

Sia `Hatch Brush Tool` che `LinearGradientBrush` importano la seguente istruzione: **Imports System.Drawing.Drawing2D**

Testo

Per disegnare il testo sul modulo, utilizzare il metodo `DrawString`

Quando si disegna una stringa è possibile utilizzare uno dei 4 pennelli elencati sopra

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
e.Graphics.DrawString("HELLO", New Font("Impact", 60, FontStyle.Bold), lBrush, New Point(40,
400))
'this will draw the word "Hello" at the given point, with a linearGradient Brush
```

Poiché non è possibile definire la larghezza o l'altezza del testo, utilizzare `Measure Text` per controllare le dimensioni del testo

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
Dim TextSize =  e.Graphics.MeasureString("HELLO", New Font("Impact", 60, FontStyle.Bold),
lBrush)
'Use the TextSize to determine where to place the string, or if the font needs to be smaller
```

> Es: devi disegnare la parola "Test" in cima al modulo. La larghezza del modulo è 120. Utilizzare questo ciclo per ridurre la dimensione del carattere finché non si adatta alla larghezza dei moduli

```
Dim FontSize as Integer = 80
Dim TextSize = e.graphics.measeString("Test", New Font("Impact",FontSize, FontStyle.Bold), new
Brush(colors.Blue, 10)
Do while TextSize.Width >120
FontSize = FontSize -1
TextSize = e.graphics.measeString("Test", New Font("Impact",FontSize, FontStyle.Bold), new
Brush(colors.Blue, 10)
Loop
```

Leggi GDI + online: https://riptutorial.com/it/vb-net/topic/5096/gdi-plus

Capitolo 20: Generics

Examples

Crea una classe generica

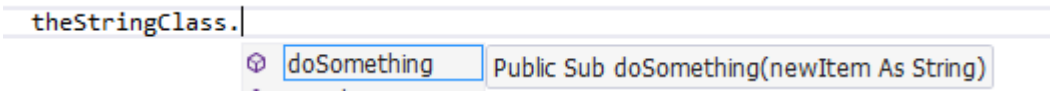
Viene creato un tipo generico per adattarsi in modo che la stessa funzionalità possa essere accessibile per diversi tipi di dati.

```
Public Class SomeClass(Of T)
    Public Sub doSomething(newItem As T)
        Dim tempItem As T
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Istanza di una classe generica

Creando un'istanza della stessa classe con un tipo diverso, l'interfaccia della classe cambia a seconda del tipo specificato.

```
Dim theStringClass As New SomeClass(Of String)
Dim theIntegerClass As New SomeClass(Of Integer)
```



Definire una classe 'generica'

Una classe generica è una classe che si adatta a un tipo successivo in modo che la stessa funzionalità possa essere offerta a tipi diversi.

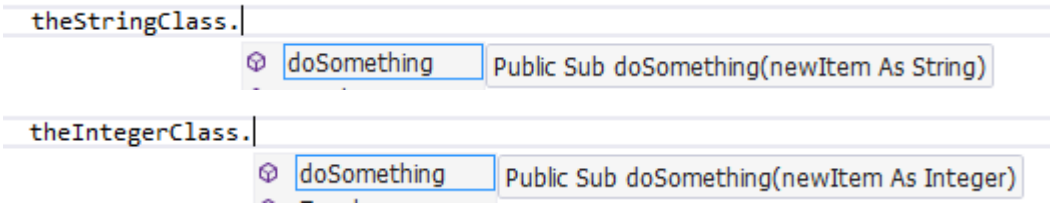
In questo esempio di base viene creata una classe generica. Ha un sottotitolo che usa il tipo generico T. Durante la programmazione di questa classe, non conosciamo il tipo di T. In questo caso T ha tutte le caratteristiche dell'oggetto.

```
Public Class SomeClass(Of T)
    Public Sub doSomething(newItem As T)
        Dim tempItem As T
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Utilizzare una classe generica

In questo esempio sono state create 2 istanze della classe SomeClass. A seconda del tipo dato le 2 istanze hanno un'interfaccia diversa:

```
Dim theStringClass As New SomeClass(Of String)
Dim theIntegerClass As New SomeClass(Of Integer)
```



La classe generica più famosa è Lista (di)

Limita i tipi possibili dati

I tipi possibili passati a una nuova istanza di SomeClass devono ereditare SomeBaseClass. Questa può anche essere un'interfaccia. Le caratteristiche di SomeBaseClass sono accessibili all'interno di questa definizione di classe.

```
Public Class SomeClass(Of T As SomeBaseClass)
    Public Sub DoSomething(newItem As T)
        newItem.DoSomethingElse()
        ' Insert code that processes an item of data type t.
    End Sub
End Class

Public Class SomeBaseClass
    Public Sub DoSomethingElse()
    End Sub
End Class
```

Crea una nuova istanza del tipo specificato

La creazione di una nuova intanza di un tipo generico può essere eseguita / checed in fase di compilazione.

```
Public Class SomeClass(Of T As {New})
    Public Function GetInstance() As T
        Return New T
    End Function
End Class
```

O con tipi limitati:

```
Public Class SomeClass(Of T As {New, SomeBaseClass})
    Public Function GetInstance() As T
        Return New T
    End Function
End Class

Public Class SomeBaseClass
End Class
```

La baseClass (se non è stata assegnata alcuna Object) deve avere un parametro meno

costruttore.

Questo può essere fatto anche in fase di esecuzione attraverso la riflessione

Leggi Generics online: https://riptutorial.com/it/vb-net/topic/6572/generics

Capitolo 21: Gestione degli errori

Examples

Prova ... Catch ... Finally Statement

Struttura:

```
Try
    'Your program will try to run the code in this block.
    'If any exceptions are thrown, the code in the Catch Block will be executed,
    'without executing the lines after the one which caused the exception.
Catch ex As System.IO.IOException
    'If an exception occurs when processing the Try block, each Catch statement
    'is examined in textual order to determine which handles the exception.
    'For example, this Catch block handles an IOException.
Catch ex As Exception
    'This catch block handles all Exception types.
    'Details of the exception, in this case, are in the "ex" variable.
    'You can show the error in a MessageBox with the below line.
    MessageBox.Show(ex.Message)
Finally
    'A finally block is always executed, regardless of if an Exception occurred.
End Try
```

Codice di esempio:

```
Try
    Dim obj = Nothing
    Dim prop = obj.Name 'This line will throw a NullReferenceException

    Console.WriteLine("Test.") ' This line will NOT be executed
Catch ex As System.IO.IOException
    ' Code that reacts to IOException.
Catch ex As NullReferenceException
    ' Code that reacts to a NullReferenceException
    Console.WriteLine("NullReferenceException: " & ex.Message)
    Console.WriteLine("Stack Trace: " & ex.StackTrace)
Catch ex As Exception
    ' Code that reacts to any other exception.
Finally
    ' This will always be run, regardless of if an exception is thrown.
    Console.WriteLine("Completed")
End Try
```

Creazione di eccezioni e lanci personalizzati

È possibile creare un'eccezione personalizzata e lanciarle durante l'esecuzione della propria funzione. Come pratica generale dovresti solo lanciare un'eccezione quando la tua funzione non è in grado di raggiungere la sua funzionalità definita.

```
Private Function OpenDatabase(Byval Server as String, Byval User as String, Byval Pwd as
```

```
String)
    if Server.trim="" then
        Throw new Exception("Server Name cannot be blank")
    elseif User.trim ="" then
        Throw new Exception("User name cannot be blank")
    elseif Pwd.trim="" then
        Throw new Exception("Password cannot be blank")
    endif

    'Here add codes for connecting to the server
End function
```

Prova Catch in Database Operation

È possibile utilizzare Try..Catch per eseguire il rollback dell'operazione del database posizionando l'istruzione rollback nel segmento Catch.

```
    Try
        'Do the database operation...
        xCmd.CommandText = "INSERT into ...."
        xCmd.ExecuteNonQuery()

        objTrans.Commit()
        conn.Close()
    Catch ex As Exception
        'Rollback action when something goes off
        objTrans.Rollback()
        conn.Close()
    End Try
```

L'eccezione non intercettabile

Anche se `Catch ex As Exception` afferma che può gestire tutte le eccezioni - c'è un'eccezione (nessun gioco di parole).

```
Imports System
Static Sub StackOverflow() ' Again no pun intended
    StackOverflow()
End Sub
Static Sub Main()
    Try
        StackOverflow()
    Catch ex As Exception
        Console.WriteLine("Exception caught!")
    Finally
        Console.WriteLine("Finally block")
    End Try
End Sub
```

Oops ... C'è un `System.StackOverflowException` non catturato mentre la console non ha nemmeno stampato nulla! Secondo MSDN ,

> A partire da .NET Framework 2.0, non è possibile catturare un oggetto StackOverflowException con un blocco try / catch e il processo corrispondente viene

terminato per impostazione predefinita. Di conseguenza, dovresti scrivere il tuo codice per rilevare e prevenire un overflow dello stack.

Quindi, `System.StackOverflowException` non è catchable. Attenzione a quello!

Eccezioni critiche

Generalmente la maggior parte delle eccezioni non è così importante, ma ci sono alcune eccezioni davvero serie che potresti non essere in grado di gestire, come la famosa `System.StackOverflowException` . Tuttavia, ce ne sono altri che potrebbero essere nascosti da `Catch ex As Exception` , come `System.OutOfMemoryException` , `System.BadImageFormatException` e `System.InvalidProgramException` . È una buona pratica di programmazione lasciarli fuori se non riesci a gestirli correttamente. Per filtrare queste eccezioni, abbiamo bisogno di un metodo di supporto:

```
Public Shared Function IsCritical(ex As Exception) As Boolean
    Return TypeOf ex Is OutOfMemoryException OrElse
           TypeOf ex Is AppDomainUnloadedException OrElse
           TypeOf ex Is AccessViolationException OrElse
           TypeOf ex Is BadImageFormatException OrElse
           TypeOf ex Is CannotUnloadAppDomainException OrElse
           TypeOf ex Is ExecutionEngineException OrElse ' Obsolete one, but better to include
           TypeOf ex Is InvalidProgramException OrElse
           TypeOf ex Is System.Threading.ThreadAbortException
End Function
```

Uso:

```
Try
    SomeMethod()
Catch ex As Exception When Not IsCritical(ex)
    Console.WriteLine("Exception caught: " & ex.Message)
End Try
```

Leggi Gestione degli errori online: https://riptutorial.com/it/vb-net/topic/4232/gestione-degli-errori

Capitolo 22: Gestione dei file

Sintassi

- `System.IO.File.ReadAllLines(path As String)`
- `System.IO.File.ReadAllText(path As String)`
- `System.IO.File.WriteAllText(path As String, contents As String)`
- `System.IO.File.WriteAllLines(path As String, contents() As String)`

Examples

Scrivi dati su un file

Per scrivere il contenuto di una stringa in un file:

```
Dim toWrite As String = "This will be written to the file."
System.IO.File.WriteAllText("filename.txt", toWrite)
```

`WriteAllText` aprirà il file specificato, scriverà i dati e quindi chiuderà il file. Se il file di destinazione esiste, viene sovrascritto. Se il file di destinazione non esiste, viene creato.

Per scrivere il contenuto di una matrice in un file:

```
Dim toWrite As String() = {"This", "Is", "A", "Test"}
System.IO.File.WriteAllLines("filename.txt", toWrite)
```

`WriteAllLines` aprirà il file specificato, scriverà ogni valore dell'array su una nuova riga e quindi chiuderà il file. Se il file di destinazione esiste, viene sovrascritto. Se il file di destinazione non esiste, viene creato.

Leggi tutti i contenuti di un file

Per leggere il contenuto in un file in una variabile stringa:

```
Dim fileContents As String = System.IO.File.ReadAllText("filename.txt")
```

`ReadAllText` aprirà il file specificato, leggerà i dati fino alla fine, quindi chiuderà il file.

Per leggere un file, separandolo in un elemento dell'array per ogni riga:

```
Dim fileLines As String() = System.IO.File.ReadAllLines("filename.txt")
```

`ReadAllLines` aprirà il file specificato, leggerà ogni riga del file in un nuovo indice in una matrice fino alla fine del file, quindi chiuderà il file.

Scrivere le righe singolarmente su un file di testo usando StreamWriter

```
Using sw As New System.IO.StreamWriter("path\to\file.txt")
    sw.WriteLine("Hello world")
End Using
```

L'uso di un blocco `Using` è una buona pratica raccomandata quando si utilizza un oggetto che implementa l' `IDisposable`

Leggi Gestione dei file online: https://riptutorial.com/it/vb-net/topic/2413/gestione-dei-file

Capitolo 23: Gestione delle connessioni

Examples

Proprietà di connessione pubblica

```
Imports System.Data.OleDb

Private WithEvents _connection As OleDbConnection
Private _connectionString As String = "myConnectionString"

Public ReadOnly Property Connection As OleDbConnection
    Get
        If _connection Is Nothing Then
            _connection = New OleDbConnection(_connectionString)
            _connection.Open()
        Else
            If _connection.State <> ConnectionState.Open Then
                _connection.Open()
            End If
        End If
        Return _connection
    End Get
End Property
```

Leggi Gestione delle connessioni online: https://riptutorial.com/it/vb-net/topic/6398/gestione-delle-connessioni

Capitolo 24: Google Maps in un Windows Form

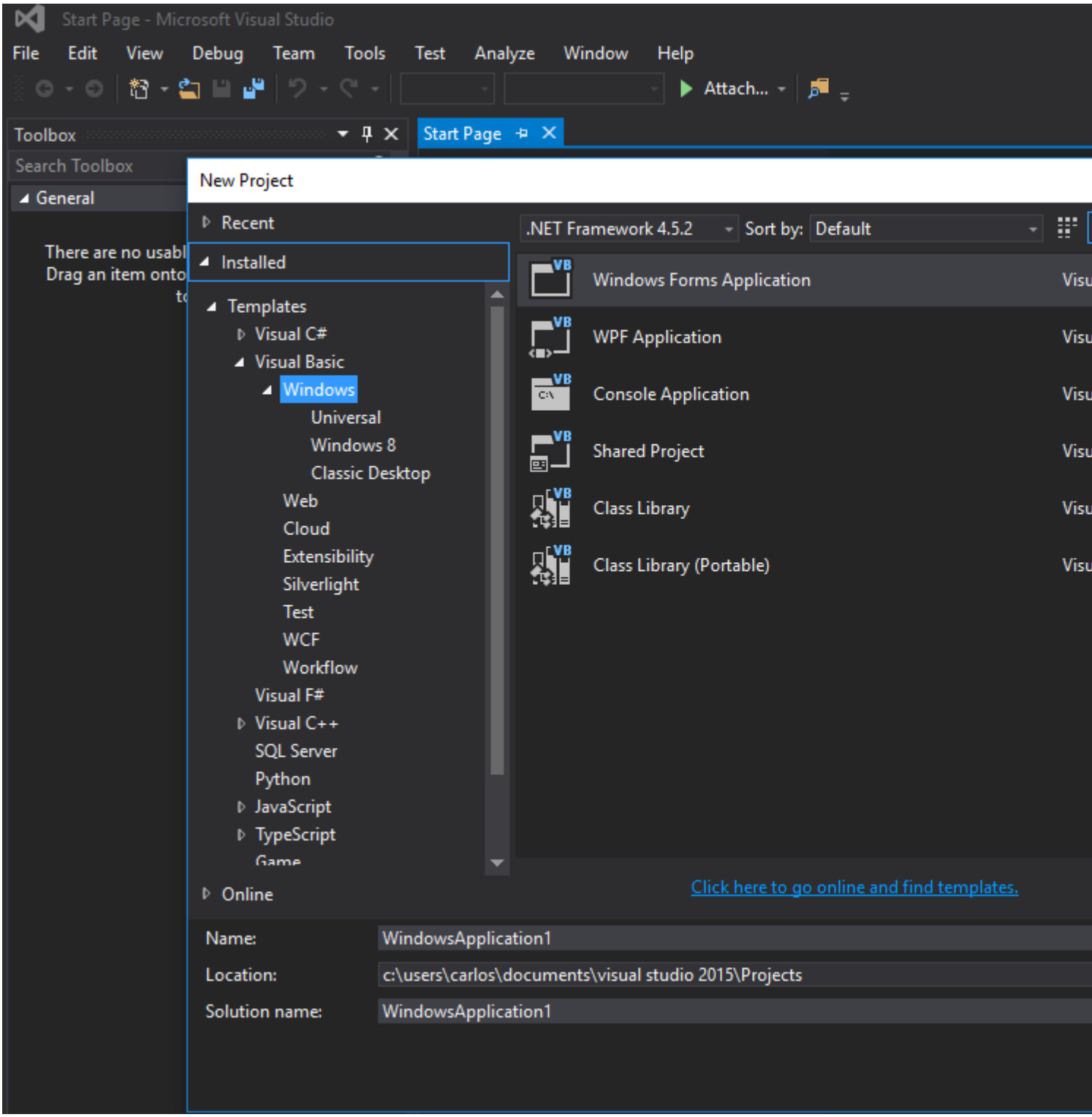
Examples

Come utilizzare una mappa di Google in un Windows Form

La prima parte di questo esempio spiega come implementarla. Nel secondo, spiegherò come funziona. Questo cerca di essere un esempio generale. Il modello per la mappa (vedere il passaggio 3) e le funzioni di esempio sono completamente personalizzabili.

################################## **IMPLEMENTAZIONE** ################## ##################

Passaggio 1. In primo luogo, creare un nuovo progetto e selezionare Applicazione Windows Form. Lasciamo il suo nome come "Form1".



Passaggio 2. Aggiungere un controllo WebBrowser (che manterrà la mappa) al Form1. Chiamiamolo "wbmap"

Passaggio 3. Creare un file .html denominato "googlemap_template.html" con il proprio editor di testo preferito e incollare il seguente codice:

googlemap_template.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
     <style type="text/css">
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
      }
      #gmap {
        height: 100%;
      }
     </style>
    <script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
    <script type="text/javascript">
        function initialize() {
            //Use window.X instead of var X to make a variable globally available
            window.markers = new Array();
            window.marker_data = [[MARKER_DATA]];
            window.gmap = new google.maps.Map(document.getElementById('gmap'), {
            zoom: 15,
            center: new google.maps.LatLng(marker_data[0][0], marker_data[0][1]),
            mapTypeId: google.maps.MapTypeId.ROADMAP
          });
          var infowindow = new google.maps.InfoWindow();
          var newmarker, i;
          for (i = 0; i < marker_data.length; i++) {
              if (marker_data[0].length == 2) {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap
                  });
              } else if (marker_data[0].length == 3) {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap,
                      title: (marker_data[i][2])
                  });
              } else {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap,
                      title: (marker_data[i][2]),
                      icon: (marker_data[i][3])
                  });
              }
            google.maps.event.addListener(newmarker, 'click', (function (newmarker, i) {
                return function () {
                    if (newmarker.title) {
                        infowindow.setContent(newmarker.title);
                        infowindow.open(gmap, newmarker);
                    }
                    gmap.setCenter(newmarker.getPosition());
                    // Calling functions written in the WF
                    window.external.showVbHelloWorld();
                    window.external.getMarkerDataFromJavascript(newmarker.title,i);
                }
            })(newmarker, i));
            markers[i] = newmarker;
          }
        }
```

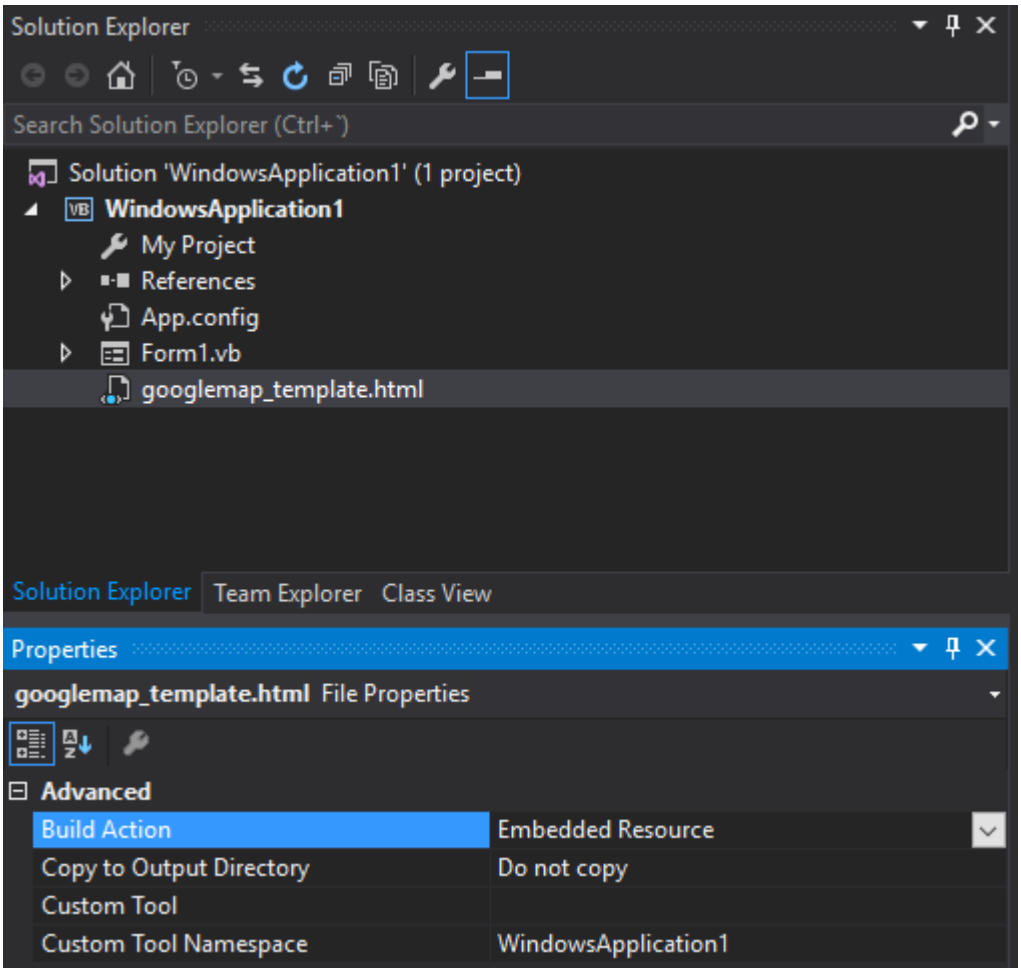
```
        google.maps.event.addDomListener(window, 'load', initialize);
    </script>
    <script type="text/javascript">
        // Function triggered from the WF with no arguments
        function showJavascriptHelloWorld() {
            alert("Hello world in HTML from WF");
        }
     </script>
      <script type="text/javascript">
        // Function triggered from the WF with a String argument
        function focusMarkerFromIdx(idx) {
            google.maps.event.trigger(markers[idx], 'click');
        }
      </script>
  </head>
  <body>
    <div id="gmap"></div>
  </body>
</html>
```

Questo servirà da modello di mappa. Spiegherò come funziona dopo.

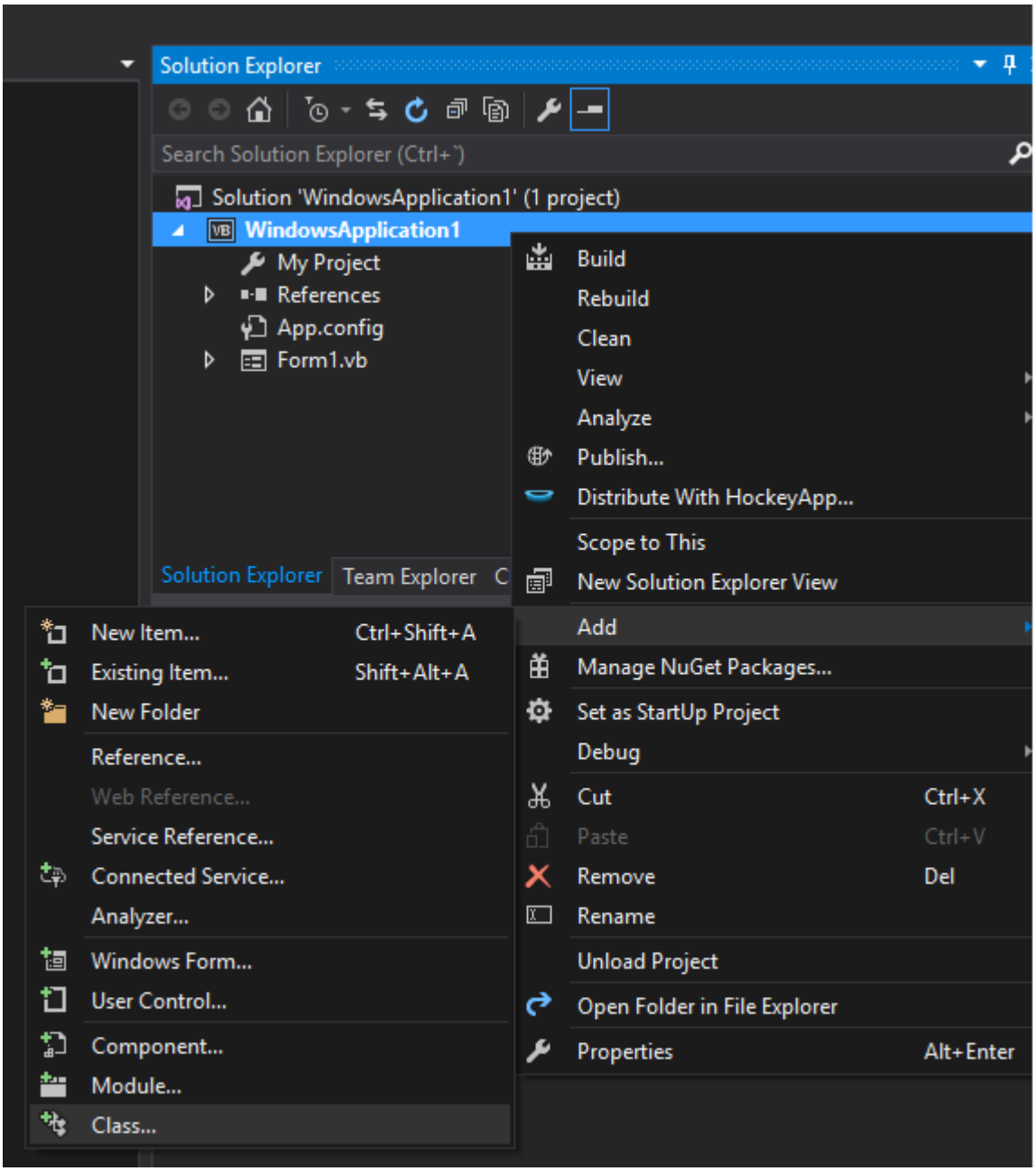
Passaggio 4. Aggiungi il file googlemap_template.hmtl al tuo progetto (fai clic con il tasto destro del mouse sul tuo progetto-> aggiungi-> elemento esistente)

Passaggio 5. Una volta visualizzato in Solution Explorer, impostarne le proprietà su:

- Build Action -> Embedded Resource
- Spazio dei nomi degli strumenti personalizzati -> scrivi il nome del progetto



Passaggio 6. Aggiungi una nuova classe (fai clic con il tasto destro del mouse sul progetto-> aggiungi-> classe). Nel mio esempio lo chiamerò GoogleMapHelper.



Passaggio 7. Incollare il codice seguente nella classe:

GoogleMapHelper.vb

```
    Imports System.IO
    Imports System.Reflection
    Imports System.Text

    Public Class GoogleMapHelper

    ' 1- googlemap_template.html must be copied in the main project folder
    ' 2- add the file into the Visual Studio Solution Explorer (add existing file)
    ' 3- set the properties of the file to:
    '                                  Build Action -> Embedded Resource
    '                                  Custom Tool Namespace -> write the name of the project

    Private Const ICON_FOLDER As String = "marker_icons/" 'images must be stored in a folder
inside  Debug/Release folder
    Private Const MAP_TEMPLATE As String = "WindowsApplication1.googlemap_template.html"
```

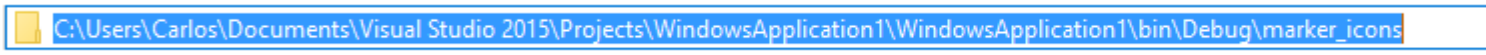
```
    Private Const TEXT_TO_REPLACE_MARKER_DATA As String = "[[MARKER_DATA]]"
    Private Const TMP_NAME As String = "tmp_map.html"


    Private mWebBrowser As WebBrowser

    'MARKER POSITIONS
    Private mPositions As Double(,) 'lat, lon
    ' marker data allows different formats to include lat,long and optionally title and icon:
    ' op1: mMarkerData = New String(N-1, 1) {{lat1, lon1}, {lat2, lon2}, {latN, lonN}}
    ' op2: mMarkerData = New String(N-1, 2) {{lat1, lon1,'title1'}, {lat2, lon2,'title2'},
{latN, lonN, 'titleN'}}
    ' op3: mMarkerData = New String(N-1, 3) {{lat1, lon1,'title1','image1.png'}, {lat2,
lon2,'title2','image2.png'}, {latN, lonN, 'titleN','imageN.png'}}
    Private mMarkerData As String(,) = Nothing


    Public Sub New(ByRef wb As WebBrowser, pos As Double(,))
        mWebBrowser = wb
        mPositions = pos
        mMarkerData = getMarkerDataFromPositions(pos)
    End Sub

    Public Sub New(ByRef wb As WebBrowser, md As String(,))
        mWebBrowser = wb
        mMarkerData = md
    End Sub

    Public Sub loadMap()
        mWebBrowser.Navigate(getMapTemplate())
    End Sub

    Private Function getMapTemplate() As String

        If mMarkerData Is Nothing Or mMarkerData.GetLength(1) > 4 Then
            MessageBox.Show("Marker data has not the proper size. It must have 2, 3 o 4
columns")
            Return Nothing
        End If

        Dim htmlTemplate As New StringBuilder()
        Dim tmpFolder As String = Environment.GetEnvironmentVariable("TEMP")
        Dim dataSize As Integer = mMarkerData.GetLength(1) 'number of columns
        Dim mMarkerDataAsText As String = String.Empty
        Dim myresourcePath As String = My.Resources.ResourceManager.BaseName
        Dim myresourcefullPath As String =
Path.GetFullPath(My.Resources.ResourceManager.BaseName)
        Dim localPath = myresourcefullPath.Replace(myresourcePath, "").Replace("\", "/") &
ICON_FOLDER

        htmlTemplate.AppendLine(getStringFromResources(MAP_TEMPLATE))
        mMarkerDataAsText = "["

        For i As Integer = 0 To mMarkerData.GetLength(0) - 1
            If i <> 0 Then
                mMarkerDataAsText += ","
            End If
            If dataSize = 2 Then 'lat,lon
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & "]"
            ElseIf dataSize = 3 Then 'lat,lon and title
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & ",'"
```

```
& mMarkerData(i, 2) & "']"
            ElseIf dataSize = 4 Then 'lat,lon,title and image
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & ",'"
& mMarkerData(i, 2) & "','" & localPath & mMarkerData(i, 3) & "']" 'Ojo a las comillas simples
en las columnas 3 y 4
            End If
        Next

        mMarkerDataAsText += "]"
        htmlTemplate.Replace(TEXT_TO_REPLACE_MARKER_DATA, mMarkerDataAsText)

        Dim tmpHtmlMapFile As String = (tmpFolder & Convert.ToString("\")) + TMP_NAME
        Dim existsMapFile As Boolean = False
        Try
            existsMapFile = createTxtFile(tmpHtmlMapFile, htmlTemplate)
        Catch ex As Exception
            MessageBox.Show("Error writing temporal file", "Writing Error",
MessageBoxButtons.OK, MessageBoxIcon.[Error])
        End Try

        If existsMapFile Then
            Return tmpHtmlMapFile
        Else
            Return Nothing
        End If
    End Function

    Private Function getMarkerDataFromPositions(pos As Double(,)) As String(,)
        Dim md As String(,) = New String(pos.GetLength(0) - 1, 1) {}
        For i As Integer = 0 To pos.GetLength(0) - 1
            md(i, 0) = pos(i, 0).ToString("g", New System.Globalization.CultureInfo("en-US"))
            md(i, 1) = pos(i, 1).ToString("g", New System.Globalization.CultureInfo("en-US"))
        Next
        Return md
    End Function

    Private Function getStringFromResources(resourceName As String) As String
        Dim assem As Assembly = Me.[GetType]().Assembly

        Using stream As Stream = assem.GetManifestResourceStream(resourceName)
            Try
                Using reader As New StreamReader(stream)
                    Return reader.ReadToEnd()
                End Using
            Catch e As Exception
                Throw New Exception((Convert.ToString("Error de acceso al Recurso '") &
resourceName) + "'" & vbCr & vbLf + e.ToString())
            End Try
        End Using
    End Function

    Private Function createTxtFile(mFile As String, content As StringBuilder) As Boolean
        Dim mPath As String = Path.GetDirectoryName(mFile)
        If Not Directory.Exists(mPath) Then
            Directory.CreateDirectory(mPath)
        End If
        If File.Exists(mFile) Then
            File.Delete(mFile)
        End If
        Dim sw As StreamWriter = File.CreateText(mFile)
        sw.Write(content.ToString())
```

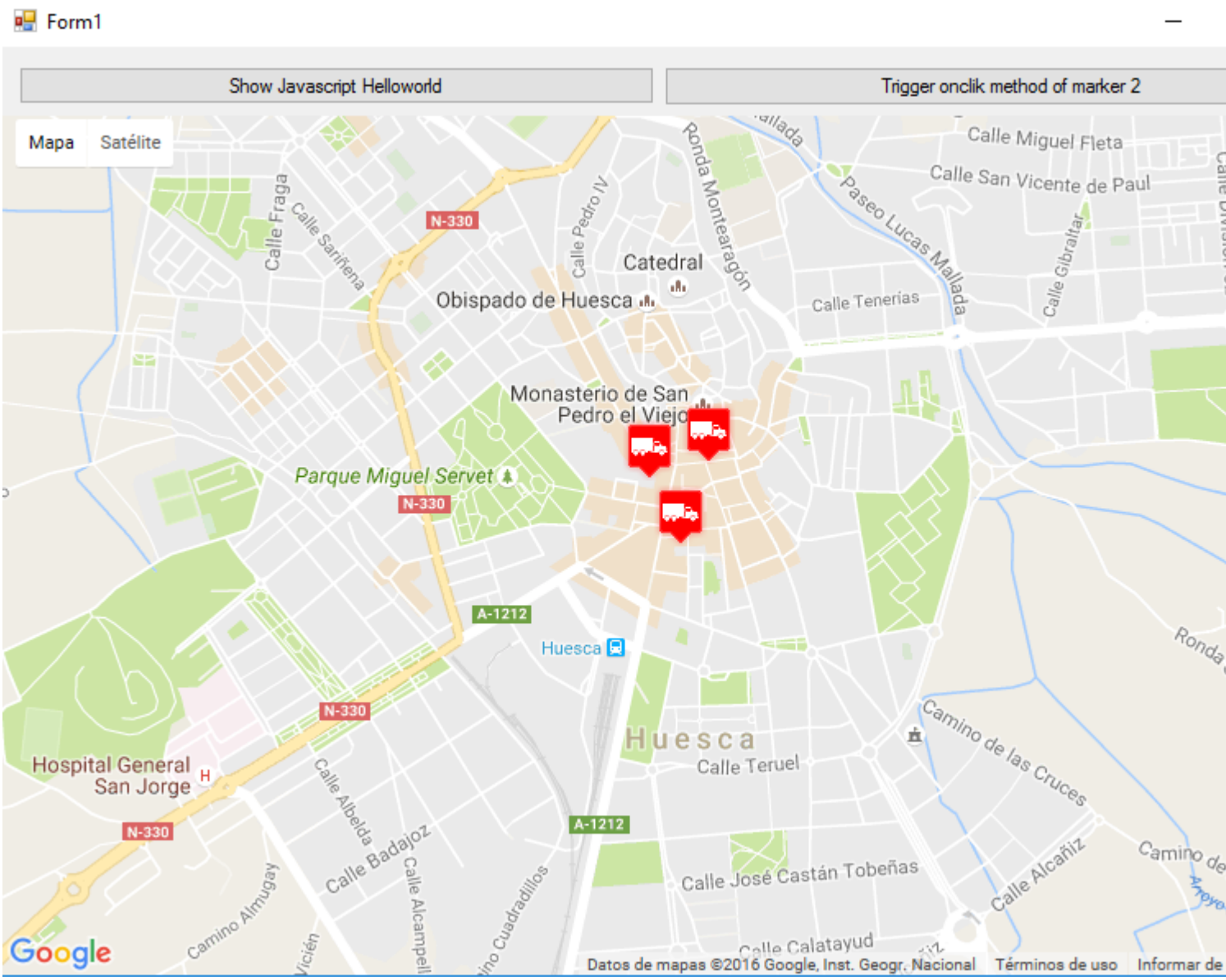
```
        sw.Close()
        Return True
    End Function
    End Class
```

Nota: la costante MAP_TEMPLATE deve includere il nome del progetto

Passaggio 8. Ora possiamo utilizzare la nostra classe GoogleMapHelper per caricare la mappa nel nostro browser semplicemente creando e istanziando il suo metodo loadMap (). Il modo in cui costruisci il tuo marcatore dipende da te. In questo esempio, per chiarimenti, li scrivo a mano. Ci sono 3 opzioni per definire i dati del marcatore (vedi commenti della classe GoogleMapHelper). Nota che se utilizzi la terza opzione (incluso titolo e icone) devi creare una cartella chiamata "marker_icons" (o qualsiasi cosa tu definisca nella costante di GoogleMapHelper ICON_FOLDER) nella cartella Debug / Release e posizionare lì i tuoi file .png. Nel mio caso:



Ho creato due pulsanti nel mio Form1 per illustrare come interagiscono la mappa e il WF. Ecco come appare:



Ed ecco il codice:

Form1.vb

```
Imports System.IO
Imports System.Reflection
Imports System.Security.Permissions
Imports System.Text
<PermissionSet(SecurityAction.Demand, Name:="FullTrust")>
<System.Runtime.InteropServices.ComVisible(True)>
Public Class Form1

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    Me.wbmap.ObjectForScripting = Me

    Dim onlyPositions As Double(,) = New Double(2, 1) {{42.13557, -0.40806}, {42.13684, -
0.40884}, {42.13716, -0.40729}}
    Dim positonAndTitles As String(,) = New String(2, 2) {{"42.13557", "-0.40806", "marker0"},
{"42.13684", "-0.40884", "marker1"}, {"42.13716", "-0.40729", "marker2"}}
    Dim positonTitlesAndIcons As String(,) = New String(2, 3) {{"42.13557", "-0.40806",
"marker0", "truck_red.png"}, {"42.13684", "-0.40884", "marker1", "truck_red.png"},
{"42.13716", "-0.40729", "marker2", "truck_red.png"}}

    'Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, onlyPositions)
    'Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, positonAndTitles)
    Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, positonTitlesAndIcons)
    gmh.loadMap()
End Sub

'############################## CALLING JAVASCRIPT METHODS ##############################
'This methods call methods written in googlemap_template.html
Private Sub callMapJavascript(sender As Object, e As EventArgs) Handles Button1.Click
    wbmap.Document.InvokeScript("showJavascriptHelloWorld")
End Sub

Private Sub callMapJavascriptWithArguments(sender As Object, e As EventArgs) Handles
Button2.Click
    wbmap.Document.InvokeScript("focusMarkerFromIdx", New String() {2})
End Sub
'########################################################################################

'############################## METHODS CALLED FROM JAVASCRIPT ##########################
'This methods are called by the javascript defined in googlemap_template.html when some events
are triggered
Public Sub getMarkerDataFromJavascript(title As String, idx As String)
    MsgBox("Title: " & title & " idx: " & idx)
End Sub

Public Sub showVbHelloWorld()
    MsgBox("Hello world in WF from HTML")
End Sub
End Class
```

IMPORTANTE: non dimenticare di aggiungere queste righe prima della definizione Form1 della classe:

```
<PermissionSet(SecurityAction.Demand, Name:="FullTrust")>
<System.Runtime.InteropServices.ComVisible(True)>
```

Quello che fanno è dire a .NET Framework che vogliamo fulltrust e rendere la classe visibile a COM in modo che Form1 sia visibile a JavaScript.

Inoltre, non dimenticarlo nella funzione di caricamento di Form1:

```
Me.wbmap.ObjectForScripting = Me
```

Espone la tua classe Form1 al JavaScript nella pagina googlemap_template.hmtl.

Ora puoi eseguire e dovrebbe funzionare

################################## **COME FUNZIONA**############## ##################

Fondamentalmente, ciò che fa la nostra classe GoogleMapHelper è leggere il nostro googlemap_template.html, creare una copia temporale, sostituire il codice relativo ai marcatori ([[MARKER_DATA]]) ed eseguire la pagina nel controllo del browser web del nostro modulo. Questo html scorre tutti i marker e assegna un "click" listener a ciascuno di essi. Questa funzione di clic è ovviamente completamente personalizzabile. Nell'esempio apre una finestra di apertura se il marcatore ha un titolo, centra la mappa in tale marcatore e chiama due funzioni esterne che sono definite nella nostra classe Form1.

D'altra parte, possiamo definire altre funzioni javascript (con o senza argomenti) in questo html da chiamare dal nostro Windows Form (usando wbmap.Document.InvokeScript).

Leggi Google Maps in un Windows Form online: https://riptutorial.com/it/vb-net/topic/5903/google-maps-in-un-windows-form

Capitolo 25: Introduzione alla sintassi

Examples

Commenti

La prima cosa interessante da sapere è come scrivere commenti.

In VB .NET, scrivi un commento scrivendo un apostrofo 'o scrivendo `REM` . Ciò significa che il resto della linea non verrà preso in considerazione dal compilatore.

```
'This entire line is a comment
Dim x As Integer = 0 'This comment is here to say we give 0 value to x

REM There are no such things as multiline comments
'So we have to start everyline with the apostrophe or REM
```

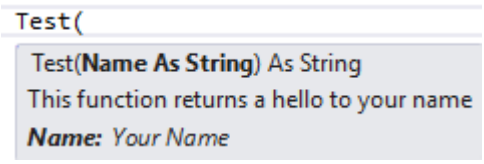
Helper Intellisense

Una cosa interessante è la possibilità di aggiungere i propri commenti in Visual Studio Intellisense. In questo modo puoi rendere le tue funzioni e classi scritte autoesplicative. Per fare ciò, è necessario digitare il simbolo di commento tre volte la riga sopra la funzione.

Una volta terminato, Visual Studio aggiungerà automaticamente una documentazione XML:

```
''' <summary>
''' This function returns a hello to your name
''' </summary>
''' <param name="Name">Your Name</param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Test(Name As String) As String
    Return "Hello " & Name
End Function
```

Dopodiché, se digiti la funzione Test da qualche parte nel tuo codice, questo piccolo aiuto apparirà:



Dichiarazione di una variabile

In VB.NET, ogni variabile deve essere dichiarata prima di essere utilizzata (se Option Explicit è impostata su **On**). Esistono due modi per dichiarare le variabili:

- All'interno di una `Function` o di una `Sub` :

```
Dim w 'Declares a variable named w of type Object (invalid if Option Strict is On)
Dim x As String 'Declares a variable named x of type String
Dim y As Long = 45 'Declares a variable named y of type Long and assigns it the value 45
Dim z = 45 'Declares a variable named z whose type is inferred
           'from the type of the assigned value (Integer here) (if Option Infer is On)
           'otherwise the type is Object (invalid if Option Strict is On)
           'and assigns that value (45) to it
```

Vedi questa risposta per tutti i dettagli su `Option Explicit` , `Strict` e `Infer` .

- All'interno di una `Class` o di un `Module` :

Queste variabili (chiamate anche campi in questo contesto) saranno accessibili per ogni istanza della `Class` cui sono dichiarate. Potrebbero essere accessibili dall'esterno della `Class` dichiarata a seconda del modificatore (`Public` , `Private` , `Protected` , `Protected Friend` , `Protected Friend` o `Friend`)

```
Private x 'Declares a private field named x of type Object (invalid if Option Strict is On)
Public y As String 'Declares a public field named y of type String
Friend z As Integer = 45 'Declares a friend field named z of type Integer and assigns it the
value 45
```

Questi campi possono anche essere dichiarati con `Dim` ma il significato cambia in base al tipo di allegato:

```
Class SomeClass
    Dim z As Integer = 45 ' Same meaning as Private z As Integer = 45
End Class

Structure SomeStructure
    Dim y As String ' Same meaning as Public y As String
End Structure
```

modificatori

I modificatori sono un modo per indicare come gli oggetti esterni possono accedere ai dati di un oggetto.

- Pubblico

Significa che qualsiasi oggetto può accedere a questo senza restrizioni

- Privato

Significa che solo l'oggetto dichiarante può accedere e visualizzarlo

- protetta

Significa che solo l'oggetto dichiarante e qualsiasi oggetto che ne eredita possono accedere e visualizzare questo.

- Amico

Indica solo l'oggetto delcaring, qualsiasi oggetto che ne eredita e qualsiasi oggetto nello stesso spazio dei nomi può accedere e visualizzarlo.

```
Public Class MyClass
    Private x As Integer

    Friend Property Hello As String

    Public Sub New()
    End Sub

    Protected Function Test() As Integer
        Return 0
    End Function
End Class
```

Scrivere una funzione

Una funzione è un blocco di codice che verrà chiamato più volte durante l'esecuzione. Invece di scrivere sempre lo stesso pezzo di codice, si può scrivere questo codice all'interno di una funzione e chiamare quella funzione ogni volta che è necessario.

Una funzione :

- Deve essere dichiarato in una *classe* o in un *modulo*
- Restituisce un valore (specificato dal tipo di reso)
- Ha un *modificatore*
- Può prendere parametri per fare la sua elaborazione

```
Private Function AddNumbers(X As Integer, Y As Integer) As Integer
    Return X + Y
End Function
```

Un nome funzione, potrebbe essere usato come dichiarazione di ritorno

```
Function sealBarTypeValidation() as Boolean
    Dim err As Boolean = False

    If rbSealBarType.SelectedValue = "" Then
        err = True
    End If

    Return err
End Function
```

è proprio come

```
Function sealBarTypeValidation() as Boolean
    sealBarTypeValidation = False

    If rbSealBarType.SelectedValue = "" Then
        sealBarTypeValidation = True
    End If
```

```
End Function
```

Inizializzatori di oggetti

- Tipi nominati

  ```
  Dim someInstance As New SomeClass(argument) With {
      .Member1 = value1,
      .Member2 = value2
      '...
  }
  ```

  È equivalente a

  ```
  Dim someInstance As New SomeClass(argument)
  someInstance.Member1 = value1
  someInstance.Member2 = value2
  '...
  ```

- Tipi anonimi *(opzione Inferente deve essere attiva)*

  ```
   Dim anonymousInstance = New With {
       .Member1 = value1,
       .Member2 = value2
       '...
   }
  ```

  Sebbene `anonymousInstance` simile non abbia lo stesso tipo di `someInstance`

  Il nome del membro deve essere univoco nel tipo anonimo e può essere preso da una variabile o da un altro nome membro dell'oggetto

  ```
   Dim anonymousInstance = New With {
       value1,
       value2,
       foo.value3
       '...
   }
   ' usage : anonymousInstance.value1 or anonymousInstance.value3
  ```

  Ogni membro può essere preceduto dalla parola chiave `Key` . Questi membri saranno proprietà `ReadOnly` , quelli senza proprietà di lettura / scrittura

  ```
   Dim anonymousInstance = New With {
       Key value1,
       .Member2 = value2,
       Key .Member3 = value3
       '...
   }
  ```

  Due istanze anonime definite con gli stessi membri (nome, tipo, presenza di `Key` e ordine)

avranno lo stesso tipo anonimo.

```
Dim anon1 = New With { Key .Value = 10 }
Dim anon2 = New With { Key .Value = 20 }

anon1.GetType Is anon2.GetType ' True
```

I tipi anonimi sono strutturalmente equi Due istanze degli stessi tipi anonimi con almeno una proprietà `Key` con gli stessi valori `Key` saranno uguali. Devi usare il metodo `Equals` per testarlo, usando `=` non verrà compilato e `Is` confronterà il riferimento all'oggetto.

```
Dim anon1 = New With { Key .Name = "Foo", Key .Age = 10, .Salary = 0 }
Dim anon2 = New With { Key .Name = "Bar", Key .Age = 20, .Salary = 0 }
Dim anon3 = New With { Key .Name = "Foo", Key .Age = 10, .Salary = 10000 }

anon1.Equals(anon2) ' False
anon1.Equals(anon3) ' True although non-Key Salary isn't the same
```

Sia l'inizializzatore Named che i tipi Anonymous possono essere nidificati e misti

```
Dim anonymousInstance = New With {
    value,
    Key .someInstance = New SomeClass(argument) With {
        .Member1 = value1,
        .Member2 = value2
        '...
    }
    '...
}
```

Inizializzatore di raccolta

- Array

```
Dim names = {"Foo", "Bar"} ' Inferred as String()
Dim numbers = {1, 5, 42} ' Inferred as Integer()
```

- Contenitori (`List(Of T)` , `Dictionary(Of TKey, TValue)` , ecc.)

```
Dim names As New List(Of String) From {
    "Foo",
    "Bar"
    '...
}

Dim indexedDays As New Dictionary(Of Integer, String) From {
    {0, "Sun"},
    {1, "Mon"}
    '...
}
```

È equivalente a

```
Dim indexedDays As New Dictionary(Of Integer, String)
indexedDays.Add(0, "Sun")
indexedDays.Add(1, "Mon")
'...
```

Gli articoli possono essere il risultato di un costruttore, una chiamata al metodo, un accesso alla proprietà. Può anche essere mescolato con l'inizializzatore di oggetti.

```
Dim someList As New List(Of SomeClass) From {
    New SomeClass(argument),
    New SomeClass With { .Member = value },
    otherClass.PropertyReturningSomeClass,
    FunctionReturningSomeClass(arguments)
    '...
}
```

Non è possibile utilizzare la sintassi dell'inizializzatore dell'oggetto **E la** sintassi di inizializzazione della raccolta per lo stesso oggetto allo stesso tempo. Ad esempio, questi **non** funzioneranno

```
Dim numbers As New List(Of Integer) With {.Capacity = 10} _
                                   From { 1, 5, 42 }

Dim numbers As New List(Of Integer) From {
    .Capacity = 10,
    1, 5, 42
}

Dim numbers As New List(Of Integer) With {
    .Capacity = 10,
    1, 5, 42
}
```

- Tipo personalizzato

  Possiamo anche consentire la sintassi di inizializzazione della raccolta fornendo un tipo personalizzato.
  Deve implementare `IEnumerable` e avere un accesso accessibile e compatibile con regole di overload `Add` metodo (istanza, metodo di estensione condiviso o pari)

  Esempio di esempio

```
Class Person
    Implements IEnumerable(Of Person) ' Inherits from IEnumerable

    Private ReadOnly relationships As List(Of Person)

    Public Sub New(name As String)
        relationships = New List(Of Person)
    End Sub

    Public Sub Add(relationName As String)
        relationships.Add(New Person(relationName))
    End Sub
```

```
    Public Iterator Function GetEnumerator() As IEnumerator(Of Person) _
        Implements IEnumerable(Of Person).GetEnumerator

        For Each relation In relationships
            Yield relation
        Next
    End Function

    Private Function IEnumerable_GetEnumerator() As IEnumerator _
        Implements IEnumerable.GetEnumerator

        Return GetEnumerator()
    End Function
End Class

' Usage
Dim somePerson As New Person("name") From {
    "FriendName",
    "CoWorkerName"
    '...
}
```

Se volessimo aggiungere l'oggetto Person ad una `List(Of Person)` semplicemente inserendo il nome nell'inizializzatore della raccolta (ma non possiamo modificare la classe List (Of Person)) possiamo usare un metodo Extension

```
' Inside a Module
<Runtime.CompilerServices.Extension>
Sub Add(target As List(Of Person), name As String)
    target.Add(New Person(name))
End Sub

' Usage
Dim people As New List(Of Person) From {
    "Name1", ' no need to create Person object here
    "Name2"
}
```

Leggi Introduzione alla sintassi online: https://riptutorial.com/it/vb-net/topic/3997/introduzione-alla-sintassi

Capitolo 26: Lavorare con Windows Forms

Examples

Utilizzando l'istanza di modulo predefinita

VB.NET offre istanze di modulo predefinite. Lo sviluppatore non ha bisogno di creare l'istanza poiché è creata dietro le quinte. Tuttavia, ***non*** *è preferibile* utilizzare l'istanza predefinita tutti, ma i programmi più semplici.

```
Public Class Form1

    Public Sub Foo()
        MessageBox.Show("Bar")
    End Sub

End Class

Module Module1

    Public Sub Main()
        ' Default instance
        Form1.Foo()
        ' New instance
        Dim myForm1 As Form1 = New Form1()
        myForm1.Foo()

    End Sub

End Module
```

Guarda anche:

- Devi creare in modo esplicito l'istanza del modulo in VB.NET?
- Perché esiste un'istanza predefinita di ogni modulo in VB.Net ma non in C #?

Passaggio di dati da una forma a un'altra

A volte potresti voler passare informazioni che sono state generate in un modulo, in un altro modulo per uso aggiuntivo. Questo è utile per i moduli che visualizzano uno strumento di ricerca o una pagina delle impostazioni tra molti altri usi.

Diciamo che vuoi passare un `DataTable` tra un modulo che è già aperto *(MainForm)* e un nuovo modulo *(NewForm)* :

In The Mainform:

```
  Private Sub Open_New_Form()
      Dim NewInstanceOfForm As New NewForm(DataTable1)
      NewInstanceOfForm.ShowDialog()
    End Sub
```

In The NewForm

```
Public Class NewForm
    Dim NewDataTable as Datatable

    Public Sub New(PassedDataTable As Datatable)
       InitializeComponent()
       NewDataTable= PassedDataTable
    End Sub

End Class
```

Ora, quando viene aperto *NewForm* , viene passato `DataTable1` da *MainForm* e memorizzato come `NewDataTable` in *NewForm* per essere utilizzato da tale modulo.

Questo può essere estremamente utile quando si tenta di passare grandi quantità di informazioni tra i moduli, specialmente quando si combinano tutte le informazioni in un singolo `ArrayList` e si passa `ArrayList` al nuovo modulo.

Leggi Lavorare con Windows Forms online: https://riptutorial.com/it/vb-net/topic/4636/lavorare-con-windows-forms

Capitolo 27: Leggere file di testo compresso al volo

Examples

Lettura di file di testo .gz riga dopo riga

Questa classe apre un file .gz (il solito formato di file di registro compresso) e restituirà una riga ad ogni chiamata di `.NextLine()`

Non esiste un utilizzo della memoria per la decompressione temporanea, molto utile per file di grandi dimensioni.

```
Imports System.IO

Class logread_gz

  Private ptr As FileStream
  Private UnGZPtr As Compression.GZipStream
  Private line_ptr As StreamReader
  Private spath As String

  Sub New(full_filename As String)
    spath = full_filename
  End Sub

  Sub Open()
     Me.ptr = File.OpenRead(spath)
     Me.UnGZPtr = New Compression.GZipStream(ptr, Compression.CompressionMode.Decompress)
     Me.line_ptr = New StreamReader(UnGZPtr)
  End Sub()

  Function NextLine() As String
    'will return Nothing if EOF
    Return Me.line_ptr.ReadLine()
  End Function

  Sub Close()
    Me.line_ptr.Close()
    Me.line_ptr.Dispose()
    Me.UnGZPtr.Close()
    Me.UnGZPtr.Dispose()
    Me.ptr.Close()
    Me.ptr.Dispose()
  End Sub

End Class
```

Nota: non esiste il failsafe, per scopi di leggibilità.

Leggi Leggere file di testo compresso al volo online: https://riptutorial.com/it/vb-net/topic/6960/leggere-file-di-testo-compresso-al-volo

Capitolo 28: LINQ

introduzione

LINQ (Language Integrated Query) è un'espressione che recupera i dati da un'origine dati. LINQ semplifica questa situazione offrendo un modello coerente per lavorare con i dati attraverso vari tipi di fonti e formati di dati. In una query LINQ, si lavora sempre con gli oggetti. Si utilizzano gli stessi schemi di codifica di base per interrogare e trasformare i dati in documenti XML, database SQL, set di dati ADO.NET, raccolte .NET e qualsiasi altro formato per il quale sia disponibile un provider LINQ.

Examples

Proiezione

```
' sample data
Dim sample = {1, 2, 3, 4, 5}

' using "query syntax"
Dim squares = From number In sample Select number * number

' same thing using "method syntax"
Dim squares = sample.Select (Function (number) number * number)
```

Possiamo anche proiettare più risultati contemporaneamente

```
Dim numbersAndSquares =
    From number In sample Select number, square = number * number

Dim numbersAndSquares =
    sample.Select (Function (number) New With {Key number, Key .square = number * number})
```

Selezione dall'array con condizioni semplici

```
Dim sites() As String = {"Stack Overflow", "Super User", "Ask Ubuntu", "Hardware
Recommendations"}
Dim query = From x In sites Where x.StartsWith("S")
' result = "Stack Overflow", "Super User"
```

Query sarà un oggetto enumerabile contenente `Stack Overflow` e `Super User` . `x` nella query sta iterando la variabile dove verrà archiviato ogni oggetto controllato dalla clausola `Where` .

Mappatura dell'array in base alla clausola Select

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
```

```
                          "Hardware Recommendations"}
Dim query = From x In sites Select x.Length
' result = 14, 10, 10, 24
```

Il risultato della query sarà un oggetto enumerabile contenente lunghezze di stringhe nell'array di input. In questo esempio, i valori 14, 10, 10, 24. x nella query sono variabili iterating in cui verrà archiviato ogni oggetto dall'array di input.

Ordinazione dell'output

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
                         "Hardware Recommendations"}

Dim query = From x In sites
            Order By x.Length

' result = "Super User", "Ask Ubuntu", "Stack Overflow", "Hardware Recommendations"
```

La clausola OrderBy ordina l'output in base al valore restituito dalla clausola. In questo esempio è Lunghezza di ogni stringa. L'ordine di output predefinito è crescente. Se è necessario scendere, è possibile specificare la parola chiave `Descending` dopo la clausola.

```
Dim query = From x In sites
            Order By x.Length Descending
```

Generazione del dizionario da IEnumerable

```
' Just setting up the example
Public Class A
    Public Property ID as integer
    Public Property Name as string
    Public Property OtherValue as Object
End Class

Public Sub Example()
    'Setup the list of items
    Dim originalList As New List(Of A)
    originalList.Add(New A() With {.ID = 1, .Name = "Item 1", .OtherValue = "Item 1 Value"})
    originalList.Add(New A() With {.ID = 2, .Name = "Item 2", .OtherValue = "Item 2 Value"})
    originalList.Add(New A() With {.ID = 3, .Name = "Item 3", .OtherValue = "Item 3 Value"})

    'Convert the list to a dictionary based on the ID
    Dim dict As Dictionary(Of Integer, A) = originalList.ToDictionary(function(c) c.ID,
function(c) c)

    'Access Values From The Dictionary
    console.Write(dict(1).Name) ' Prints "Item 1"
    console.Write(dict(1).OtherValue) ' Prints "Item 1 Value"
End Sub
```

Ottenere valori distinti (usando il metodo Distinct)

```
Dim duplicateFruits = New List(Of String) From {"Grape", "Apple", "Grape", "Apple", "Grape"}
'At this point, duplicateFruits.Length = 5

Dim uniqueFruits = duplicateFruits.Distinct();
'Now, uniqueFruits.Count() = 2
'If iterated over at this point, it will contain 1 each of "Grape" and "Apple"
```

Leggi LINQ online: https://riptutorial.com/it/vb-net/topic/3111/linq

Capitolo 29: looping

Examples

Per il prossimo

`For` ... `Next` ciclo `Next` viene utilizzato per ripetere la stessa azione per un numero finito di volte. Le istruzioni all'interno del seguente ciclo verranno eseguite 11 volte. La prima volta, `i` avrà il valore 0, la seconda volta avrà il valore 1, l'ultima volta che avrà il valore 10.

```
For i As Integer = 0 To 10
    'Execute the action
    Console.Writeline(i.ToString)
Next
```

Qualsiasi espressione intera può essere utilizzata per parametrizzare il ciclo. È consentito, ma non richiesto, che la variabile di controllo (in questo caso `i`) venga dichiarata anche dopo il `Next` . È consentito che la variabile di controllo sia dichiarata in anticipo, anziché all'interno dell'istruzione `For` .

```
Dim StartIndex As Integer = 3
Dim EndIndex As Integer = 7
Dim i As Integer

For i = StartIndex To EndIndex - 1
    'Execute the action
    Console.Writeline(i.ToString)
Next i
```

Essere in grado di definire gli interi Start ed End consente di creare loop che fanno riferimento direttamente ad altri oggetti, come ad esempio:

```
For i = 0 to DataGridView1.Rows.Count - 1
    Console.Writeline(DataGridView1.Rows(i).Cells(0).Value.ToString)
Next
```

Ciò quindi eseguirà un ciclo attraverso ogni riga in `DataGridView1` ed eseguirà l'azione di scrivere il valore di Column 1 sulla console. *(Il -1 è perché la prima riga delle righe contate sarebbe 1, non 0)*

È anche possibile definire come deve essere incrementata la variabile di controllo.

```
For i As Integer = 1 To 10 Step 2
    Console.Writeline(i.ToString)
Next
```

Questo produce:

    1 3 5 7 9

È anche possibile decrementare la variabile di controllo (conto alla rovescia).

```
For i As Integer = 10 To 1 Step -1
    Console.Writeline(i.ToString)
Next
```

Questo produce:

> 10 9 8 7 6 5 4 3 2 1

Non si dovrebbe tentare di utilizzare (leggere o aggiornare) la variabile di controllo al di fuori del ciclo.

Per ogni ... Ciclo successivo per il looping attraverso la raccolta di elementi

È possibile utilizzare un ciclo `For Each...Next` per eseguire iterazioni su qualsiasi tipo `IEnumerable` . Ciò include matrici, elenchi e qualsiasi altra cosa che può essere di tipo IEnumerable o restituisce un oggetto IEnumerable.

Un esempio di looping di una proprietà di DataTable's Rows sarebbe simile a questo:

```
For Each row As DataRow In DataTable1.Rows
    'Each time this loops, row will be the next item out of Rows
    'Here we print the first column's value from the row variable.
    Debug.Print(Row.Item(0))
Next
```

Una cosa importante da notare è che la collezione non deve essere modificata mentre si trova in un ciclo `For Each` . Ciò causerebbe un `System.InvalidOperationException` con il messaggio:

> La raccolta è stata modificata; l'operazione di enumerazione potrebbe non essere eseguita.

Durante il ciclo per iterare mentre alcune condizioni sono vere

Un ciclo `While` inizia valutando una condizione. Se è vero, viene eseguito il corpo del ciclo. Dopo che il corpo del ciclo è stato eseguito, la condizione `While` viene valutata di nuovo per determinare se rieseguire il corpo.

```
Dim iteration As Integer = 1
While iteration <= 10
  Console.Writeline(iteration.ToString() & " ")

  iteration += 1
End While
```

Questo produce:

> 1 2 3 4 5 6 7 8 9 10

Attenzione: un ciclo `While` può portare a un *loop infinito* . Considera cosa accadrebbe se venisse

rimossa la riga di codice che incrementa l' `iteration` . In tal caso la condizione non sarebbe mai Vero e il ciclo continuerebbe indefinitamente.

Do ... Loop

Usa `Do...Loop` per ripetere un blocco di istruzioni `While` o `Until` una condizione è vera, controllando la condizione all'inizio o alla fine del ciclo.

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop While x < 10
```

o

```
Dim x As Integer = 0
Do While x < 10
    Console.Write(x & " ")
    x += 1
Loop
```

0 1 2 3 4 5 6 7 8 9

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop Until x = 10
```

o

```
Dim x As Integer = 0
Do Until x = 10
    Console.Write(x & " ")
    x += 1
Loop
```

0 1 2 3 4 5 6 7 8 9

`Continue Do` può essere usato per saltare alla successiva iterazione del ciclo:

```
Dim x As Integer = 0
Do While x < 10
    x += 1
    If x Mod 2 = 0 Then
        Continue Do
    End If
    Console.Write(x & " ")
Loop
```

1 3 5 7 9

Puoi terminare il ciclo con `Exit Do` - nota che in questo esempio, la mancanza di qualsiasi condizione causerebbe altrimenti un loop infinito:

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
    If x = 10 Then
        Exit Do
    End If
Loop
```

0 1 2 3 4 5 6 7 8 9

Cortocircuito

Qualsiasi ciclo può essere terminato o continuato presto in qualsiasi momento utilizzando le istruzioni `Exit` o `Continue` .

Uscita

Puoi interrompere qualsiasi ciclo uscendo presto. Per fare ciò, puoi usare la parola chiave `Exit` insieme al nome del loop.

Ciclo continuo	Uscita Dichiarazione
Per	`Exit For`
Per ciascuno	`Exit For`
Fare mentre	`Exit Do`
Mentre	`Exit While`

L'uscita anticipata di un ciclo è un ottimo modo per aumentare le prestazioni, effettuando il ciclo solo del numero necessario di volte per soddisfare le esigenze dell'applicazione. Di seguito è riportato un esempio in cui il ciclo uscirà una volta trovato il numero `2` .

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SoughtValue As Integer = 2
Dim SoughtIndex
For Each i In Numbers
    If i = 2 Then
        SoughtIndex = i
        Exit For
    End If
Next
Debug.Print(SoughtIndex)
```

Proseguendo

Oltre a uscire presto, puoi anche decidere di passare alla prossima iterazione del ciclo. Questo è fatto facilmente usando l'istruzione `Continue` . Proprio come `Exit` , è preceduto dal nome del ciclo.

Ciclo continuo	Continua Dichiarazione
Per	`Continue For`
Per ciascuno	`Continue For`
Fare mentre	`Continue Do`
Mentre	`Continue While`

Ecco un esempio di come impedire l'aggiunta di numeri pari alla somma.

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SumOdd As Integer = 0
For Each i In Numbers
    If Numbers(i) \ 2 = 0 Then Continue For
    SumOdd += Numbers(i)
Next
```

Consigli d'uso

Esistono due tecniche alternative che possono essere utilizzate anziché utilizzare `Exit` o `Continue` .

È possibile dichiarare una nuova variabile booleana, inizializzandola su un valore e impostandola condizionalmente sull'altro valore all'interno del ciclo; quindi si utilizza un'istruzione condizionale (ad esempio `If`) basata su tale variabile per evitare l'esecuzione delle istruzioni all'interno del ciclo nelle iterazioni successive.

```
Dim Found As Boolean = False
Dim FoundIndex As Integer
For i As Integer = 0 To N - 1
    If Not Found AndAlso A(i) = SoughtValue Then
        FoundIndex = i
        Found = True
    End If
Next
```

Una delle obiezioni a questa tecnica è che potrebbe essere inefficiente. Ad esempio, se nell'esempio precedente `N` è 1000000 e il primo elemento dell'array `A` è uguale a `SoughtValue` , il ciclo eseguirà un'ulteriore ripetizione di 999999 volte senza fare nulla di utile. Tuttavia, in alcuni casi questa tecnica può avere il vantaggio di una maggiore chiarezza.

Puoi usare l'istruzione `GoTo` per saltare fuori dal giro. Nota che non puoi usare `GoTo` per saltare *in* loop.

```
    Dim FoundIndex As Integer
    For i As Integer = 0 To N - 1
        If A(i) = SoughtValue Then
```

```
            FoundIndex = i
            GoTo Found
        End If
    Next
    Debug.Print("Not found")
Found:
    Debug.Print(FoundIndex)
```

Questa tecnica a volte può essere il modo migliore per saltare fuori dal ciclo ed evitare una o più istruzioni che vengono eseguite subito dopo la fine naturale del ciclo.

Dovresti prendere in considerazione tutte le alternative e utilizzare quella che meglio si adatta alle tue esigenze, considerando l'efficienza, la velocità di scrittura del codice e la leggibilità (quindi la manutenibilità).

Non `GoTo` scoraggiare dall'uso di `GoTo` in quelle occasioni in cui è la migliore alternativa.

Ciclo annidato

```
A nested loop is a loop within a loop, an inner loop within the body of an outer one. How this
works is that the first pass of the outer loop triggers the inner loop, which executes to
completion. Then the second pass of the outer loop triggers the inner loop again. This repeats
until the outer loop finishes. a break within either the inner or outer loop would interrupt
this process.
```

La struttura di un ciclo annidato For For Next è:

```
For counter1=startNumber to endNumber (Step increment)

    For counter2=startNumber to endNumber (Step increment)

        One or more VB statements

    Next counter2

Next  counter1
```

Esempio :

```
    For  firstCounter = 1 to 5

      Print “First Loop of ” + firstCounter

    For   secondCounter= 1 to 4

      Print “Second Loop of ” + secondCounter

    Next secondCounter

  Next firstCounter
```

Leggi looping online: https://riptutorial.com/it/vb-net/topic/1639/looping

Capitolo 30: Metodi di estensione

Osservazioni

I metodi di estensione sono metodi (`Sub` o `Function`) che aggiungono funzionalità a un tipo (che può essere un tipo di riferimento o un tipo di valore). Questi tipi possono o non possono essere di tua proprietà.

Possono o meno essere nello stesso assieme del tipo che intendono modificare. Puoi consentire l'attivazione dei tuoi metodi di estensione isolandoli nel loro spazio dei nomi. Oppure, se preferisci, puoi renderli sempre disponibili includendoli nello stesso spazio dei nomi del tipo che modificano (assumendo che tutti i riferimenti dell'assembly siano a posto e corretti). Vedere il progetto Entity Framework Core 1.0 su GitHub per un buon esempio dello stile opt-in dei metodi di estensione.

I metodi di estensione in VB hanno alcuni requisiti:

- I metodi di estensione possono essere dichiarati solo in moduli.
- I metodi di estensione devono essere decorati con l'attributo `Extension()` .
- Lo spazio dei nomi ExtensionAttribute deve essere disponibile all'interno del modulo. `Imports System.Runtime.CompilerServices`
- Il primo parametro del metodo deve essere di un tipo a cui verrà collegato questo metodo.
- Il primo parametro del metodo rappresenterà l'istanza su cui questo metodo opera. (Equivalente a `Me` se questo fosse un metodo di istanza reale).
- Un metodo di estensione può essere chiamato come un metodo normale fornendo tutti i parametri se non richiamati sull'oggetto istanziato.

Examples

Creare un metodo di estensione

I metodi di estensione sono utili per estendere il comportamento delle librerie che non possediamo.

Sono usati come metodi di istanza grazie allo zucchero sintattico del compilatore:

```
Sub Main()
    Dim stringBuilder = new StringBuilder()

    'Extension called directly on the object.
    stringBuilder.AppendIf(true, "Condition was true")

    'Extension called as a regular method. This defeats the purpose
    'of an extension method but should be noted that it is possible.
    AppendIf(stringBuilder, true, "Condition was true")

End Sub

<Extension>
Public Function AppendIf(stringBuilder As StringBuilder, condition As Boolean, text As String)
```

```
As StringBuilder
    If(condition) Then stringBuilder.Append(text)

    Return stringBuilder
End Function
```

Per avere un metodo di estensione utilizzabile, il metodo richiede l'attributo `Extension` e deve essere dichiarato in un `Module` .

Rendere la lingua più funzionale con i metodi di estensione

Un buon uso del metodo di estensione è rendere la lingua più funzionale

```
Sub Main()
    Dim strings = { "One", "Two", "Three" }

    strings.Join(Environment.NewLine).Print()
End Sub

<Extension>
Public Function Join(strings As IEnumerable(Of String), separator As String) As String
    Return String.Join(separator, strings)
End Function

<Extension>
Public Sub Print(text As String)
    Console.WriteLine(text)
End Sub
```

Numeri di imbottitura

```
Public Module Usage
  Public Sub LikeThis()
    Dim iCount As Integer
    Dim sCount As String

    iCount = 245
    sCount = iCount.PadLeft(4, "0")

    Console.WriteLine(sCount)
    Console.ReadKey()
  End Sub
End Module



Public Module Padding
  <Extension>
  Public Function PadLeft(Value As Integer, Length As Integer) As String
    Return Value.PadLeft(Length, Space(Length))
  End Function



  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer) As String
```

```
    Return Value.PadRight(Length, Space(Length))
  End Function



  <Extension>
  Public Function PadLeft(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadLeft(Length, Character)
  End Function



  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadRight(Length, Character)
  End Function
End Module
```

Ottenere la versione dell'Assembly da un nome sicuro

Esempio di chiamata di un metodo di estensione come estensione e come metodo regolare.

```
public Class MyClass
  Sub Main()

        'Extension called directly on the object.
        Dim Version = Assembly.GetExecutingAssembly.GetVersionFromAssembly()

        'Called as a regular method.
        Dim Ver = GetVersionFromAssembly(SomeOtherAssembly)

    End Sub
End Class
```

Il metodo di estensione in un modulo. Rendi pubblico il modulo se le estensioni sono compilate in una DLL e saranno referenziate in un altro assembly.

```
Public Module Extensions
    ''' <summary>
    ''' Returns the version number from the specified assembly using the assembly's strong
name.
    ''' </summary>
    ''' <param name="Assy">[Assembly] Assembly to get the version info from.</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
        Return Split(Split(Assy.FullName,",")(1),"=")(1)
    End Function
End Module
```

Leggi Metodi di estensione online: https://riptutorial.com/it/vb-net/topic/1592/metodi-di-estensione

Capitolo 31: multithreading

Examples

Multithreading usando Thread Class

Questo esempio utilizza la classe `Thread` , ma le applicazioni multithread possono essere eseguite anche con `BackgroundWorker` . I `AddNumber` , `SubstractNumber` e `DivideNumber` funzioni saranno eseguite da thread separati:

Modifica: ora il thread dell'interfaccia utente attende che i thread figlio finiscano e mostri il risultato.

```
Module Module1
    'Declare the Thread and assign a sub to that
    Dim AddThread As New Threading.Thread(AddressOf AddNumber)
    Dim SubstractThread As New Threading.Thread(AddressOf SubstractNumber)
    Dim DivideThread As New Threading.Thread(AddressOf DivideNumber)

    'Declare the variable for holding the result
    Dim addResult As Integer
    Dim SubStractResult As Integer
    Dim DivisionResult As Double

    Dim bFinishAddition As Boolean = False
    Dim bFinishSubstration As Boolean = False
    Dim bFinishDivision As Boolean = False

    Dim bShownAdditionResult As Boolean = False
    Dim bShownDivisionResult As Boolean = False
    Dim bShownSubstractionResult As Boolean = False

    Sub Main()

        'Now start the trheads
        AddThread.Start()
        SubstractThread.Start()
        DivideThread.Start()

        'Wait and display the results in console
        Console.WriteLine("Waiting for threads to finish...")
        Console.WriteLine("")

        While bFinishAddition = False Or bFinishDivision = False Or bFinishSubstration = False
            Threading.Thread.Sleep(50)     'UI thread is sleeping
            If bFinishAddition And Not bShownAdditionResult Then
                Console.WriteLine("Addition Result : " & addResult)
                bShownAdditionResult = True
            End If

            If bFinishSubstration And Not bShownSubstractionResult Then
                Console.WriteLine("Substraction Result : " & SubStractResult)
                bShownSubstractionResult = True
            End If

            If bFinishDivision  And Not bShownDivisionResult Then
```

```
                Console.WriteLine("Division Result : " & DivisionResult)
                bShownDivisionResult = True
            End If

        End While

        Console.WriteLine("")
        Console.WriteLine("Finished all threads.")
        Console.ReadKey()
    End Sub


    Private Sub AddNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11

        For i As Integer = 0 To 100
            addResult = addResult + (n1 + n2)
            Threading.Thread.Sleep(50)      'sleeping Add thread
        Next
        bFinishAddition = True
    End Sub

    Private Sub SubstractNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11

        For i As Integer = 0 To 80
            SubStractResult = SubStractResult - (n1 - n2)
            Threading.Thread.Sleep(50)
        Next
        bFinishSubstration = True
    End Sub

    Private Sub DivideNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11
        For i As Integer = 0 To 60
            DivisionResult = DivisionResult + (n1 / n2)
            Threading.Thread.Sleep(50)
        Next
        bFinishDivision = True
    End Sub

End Module
```

Leggi multithreading online: https://riptutorial.com/it/vb-net/topic/6756/multithreading

Capitolo 32: NullReferenceException

Osservazioni

NullReferenceException viene generata ogni volta che una variabile è vuota e viene fatto riferimento a uno dei suoi metodi / proprietà. Per evitare ciò, assicurarsi che tutte le variabili siano inizializzate correttamente (`new` operatore) e che tutti i metodi restituiscano un valore non nullo.

Examples

Variabile non inizializzata

CODICE CATTIVO

```
Dim f As System.Windows.Forms.Form
f.ShowModal()
```

BUON CODICE

```
Dim f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Dim f As New System.Windows.Forms.Form ' alternative syntax
f.ShowModal()
```

CODICE ANCORA MIGLIORE (Garantire il corretto smaltimento dell'oggetto IDisposable maggiori informazioni)

```
Using f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Using f As New System.Windows.Forms.Form ' alternative syntax
    f.ShowModal()
End Using
```

Ritorno vuoto

```
Function TestFunction() As TestClass
    Return Nothing
End Function
```

CODICE CATTIVO

```
TestFunction().TestMethod()
```

BUON CODICE

```
Dim x = TestFunction()
If x IsNot Nothing Then x.TestMethod()
```

14.0

Operatore condizionale nullo

```
TestFunction()?.TestMethod()
```

Leggi NullReferenceException online: https://riptutorial.com/it/vb-net/topic/4076/nullreferenceexception

Capitolo 33: Oggetti usa e getta

Examples

Concetto di base di IDisposable

Ogni volta che istanziati una classe che implementa gli `IDisposable` , dovresti chiamare `.Dispose` [1] su quella classe quando hai finito di usarla. Ciò consente alla classe di eliminare tutte le dipendenze gestite o non gestite che potrebbe essere in uso. Non farlo potrebbe causare una perdita di memoria.

La parola chiave `Using` assicura che `.Dispose` chiamato `.Dispose` , senza che tu debba chiamarlo *esplicitamente* .

Ad esempio senza `Using` :

```
Dim sr As New StreamReader("C:\foo.txt")
Dim line = sr.ReadLine
sr.Dispose()
```

Ora con l' `Using` :

```
Using sr As New StreamReader("C:\foo.txt")
    Dim line = sr.ReadLine
End Using '.Dispose is called here for you
```

Uno dei principali vantaggi `Using` ha è quando viene generata un'eccezione, perché *assicura* `.Dispose` viene chiamato.

Considera quanto segue. Se viene generata un'eccezione, è necessario ricordare di chiamare .Dispositivo, ma potrebbe anche essere necessario controllare lo stato dell'oggetto per assicurarsi di non ottenere un errore di riferimento nullo, ecc.

```
    Dim sr As StreamReader = Nothing
    Try
        sr = New StreamReader("C:\foo.txt")
        Dim line = sr.ReadLine
    Catch ex As Exception
        'Handle the Exception
    Finally
        If sr IsNot Nothing Then sr.Dispose()
    End Try
```

Un blocco usando significa che non devi ricordarti di farlo e puoi dichiarare il tuo oggetto all'interno del `try` :

```
    Try
        Using sr As New StreamReader("C:\foo.txt")
            Dim line = sr.ReadLine
```

```
        End Using
    Catch ex As Exception
        'sr is disposed at this point
    End Try
```

---

[1] Devo sempre chiamare Dispose () sui miei oggetti DbContext? no

Dichiarare più oggetti in un solo utilizzo

A volte, devi creare due oggetti `Disposable` in fila. C'è un modo semplice per evitare l'annidamento `Using` blocchi.

Questo codice

```
Using File As New FileStream("MyFile", FileMode.Append)
    Using Writer As New BinaryWriter(File)
        'You code here
        Writer.Writer("Hello")
    End Using
End Using
```

può essere abbreviato in questo Il vantaggio principale è che ottieni un livello di indentazione:

```
Using File As New FileStream("MyFile", FileMode.Append), Writer As New BinaryWriter(File)
    'You code here
    Writer.Writer("Hello")
End Using
```

Leggi Oggetti usa e getta online: https://riptutorial.com/it/vb-net/topic/3204/oggetti-usa-e-getta

Capitolo 34: operatori

Osservazioni

Gli operatori sono utilizzati per assegnare o confrontare valori. Sono costituiti da un singolo simbolo o parola chiave e di solito sono racchiusi tra un valore di sinistra e uno di destra. Ad esempio: `right = left` .

Gli operatori sono intrinseci al linguaggio (come `=`) e non funzioni come quelle fornite da System.Math.

Examples

Confronto

Gli operatori di confronto confrontano due valori e restituiscono un valore booleano (`True` o `False`) come risultato.

Uguaglianza

- Il segno di uguale `=` viene usato sia per il confronto di uguaglianza che per l'assegnazione.
  `If leftValue = rightValue Then` ...

Disuguaglianza

- La parentesi angolare sinistra nella parentesi angolare destra `<>` esegue un confronto non uguale.
  `If leftValue <> rightValue Then` ...

Più grande di

- La parentesi angolare sinistra `<` esegue un valore maggiore del confronto.
  `If leftValue < rightValue Then` ...

Maggiore o uguale

- Il segno di uguaglianza annidato alla parentesi angolare sinistra `=>` esegue un confronto maggiore o uguale a.
  `If leftValue =< rightValue Then` ...

Meno di

- La parentesi angolare destra `>` esegue meno del confronto.
  `If leftValue > rightValue Then` ...

Meno o uguale

- Il segno di uguale nidificazione alla parentesi angolare `= =` esegue un confronto maggiore o uguale a.
  `If leftValue => rightValue Then ...`

Piace

- L'operatore `Like` verifica l'uguaglianza di una stringa e un modello di ricerca.
- L'operatore `Like` si basa sull'istruzione Option Compare
- La seguente tabella elenca i modelli disponibili. Fonte: https://msdn.microsoft.com/en-us/library/swf8kaxw.aspx (sezione Commenti)

Personaggi nel *modello*	Partite nella *stringa*
?	Qualsiasi singolo personaggio
*	Zero o più caratteri
#	Qualsiasi cifra singola (0 - 9)
[Charlist]	Qualsiasi singolo carattere in *charlist*
[! Charlist]	Qualsiasi singolo carattere non in *charlista*

- Vedi ulteriori informazioni su MSDN nella sezione commenti.
  `If string Like pattern Then ...`

assegnazione

Esiste un singolo operatore di assegnazione in VB.

- Il segno di uguale `=` viene usato sia per il confronto di uguaglianza che per l'assegnazione.
  `Dim value = 5`

Gli appunti
Fai attenzione al confronto tra assegnazione e uguaglianza.

```
Dim result = leftValue = rightValue
```

In questo esempio è possibile vedere il segno di uguale utilizzato come operatore di confronto e operatore di assegnazione, a differenza di altre lingue. In questo caso, il `result` sarà di tipo `Boolean` e conterrà il valore del confronto di uguaglianza tra `leftValue` e `rightValue` .

Correlato: usare Option Strict On per dichiarare correttamente le variabili

Matematica

Se hai le seguenti variabili

```
Dim leftValue As Integer = 5
```

```
Dim rightValue As Integer = 2
Dim value As Integer = 0
```

Aggiunta eseguita dal segno più `+` .

```
value  = leftValue + rightValue

'Output the following:
'7
```

Sottrazione Eseguita dal segno meno `-` .

```
value = leftValue - rightValue

'Output the following:
'3
```

Moltiplicazione eseguita dal simbolo asterisco `*` .

```
value = leftValue * rightValue

'Output the following:
'10
```

Divisione eseguita dal simbolo barra in avanti `/` .

```
value = leftValue / rightValue

'Output the following:
'2.5
```

Divisione intera Eseguita dal simbolo barra retroversa `\` .

```
value = leftValue \ rightValue

'Output the following:
'2
```

Modulo realizzato dalla parola chiave `Mod` .

```
value = leftValue Mod rightValue

'Output the following:
'1
```

Aumenta a un potere di Eseguito dal simbolo `^` .

```
value = leftValue ^ rightValue

'Output the following:
'25
```

Allargamento e restringimento

Ha bisogno di essere modificato.

Sovraccarico dell'operatore

Ha bisogno di essere modificato.

bitwise

Questi sono gli operatori bit a bit in VB.NET: And, Or, Xor, Not

Esempio di operazione And bitwise

```
Dim a as Integer
a = 3 And 5
```

Il valore di a sarà 1. Il risultato è ottenuto dopo aver confrontato 3 e 5 in binario per. 3 in forma binaria è 011 e 5 in forma binaria è 101. L'operatore And colloca 1 se entrambi i bit sono 1. Se uno qualsiasi dei bit è 0, il valore sarà 0

```
3 And 5 will be  011
                 101
                 ---
                 001
```

Quindi il risultato binario è 001 e quando viene convertito in decimale, la risposta sarà 1.

Oppure l'operatore piazza 1 se entrambi o un bit è 1

```
   3 Or 5 will be 011
                  101
                  ---
                  111
```

L'operatore Xo pone 1 se solo uno dei bit è 1 (non entrambi)

```
   3 Xor 5 will be  011
                    101
                    ---
                    110
```

L'operatore non ripristina i bit incluso il segno

```
Not 5 will be - 010
```

Concatenazione di stringhe

La concatenazione di stringhe si ha quando si combinano due o più stringhe in una variabile di

stringa singola.

La concatenazione delle stringhe viene eseguita con il simbolo `&` .

```
Dim one As String = "Hello "
Dim two As String = "there"
Dim result As String = one & two
```

I valori non stringa verranno convertiti in stringa quando si utilizza `&` .

```
Dim result as String = "2" & 10 ' result  = "210"
```

Usa sempre `&` (`e` commerciale) per eseguire concatenazioni di stringhe.

NON FARE QUESTO
Mentre è possibile, nel *più semplice* dei casi, usare il simbolo `+` per fare concatenazione di stringhe, non dovresti mai farlo. Se un lato del simbolo più non è una stringa, quando Option strict è off, il comportamento diventa non intuitivo, quando Option strict è attivo genererà un errore del compilatore. Tenere conto:

```
Dim value = "2" + 10    ' result = 12  (data type Double)
Dim value = "2" + "10"  ' result = "210"  (data type String)
Dim value = "2g" + 10   ' runtime error
```

Il problema qui è che se l'operatore `+` vede un operando che è un tipo numerico, presumerà che il programmatore volesse eseguire un'operazione aritmetica e tenterà di eseguire il cast dell'altra operando sul tipo numerico equivalente. Nei casi in cui l'altro operando è una stringa che contiene un numero (ad esempio, "10"), la stringa viene *convertita in un numero* e quindi *aritmeticamente* aggiunta all'altro operando. Se l'altro operando non può essere convertito in un numero (ad esempio "2g"), l'operazione si bloccherà a causa di un errore di conversione dei dati. L'operatore `+` eseguirà la concatenazione di stringhe solo se *entrambi gli* operandi sono di tipo `String` .

L'operatore `&` , tuttavia, è progettato per la concatenazione di stringhe e invierà tipi non string di stringhe.

Leggi operatori online: https://riptutorial.com/it/vb-net/topic/3257/operatori

Capitolo 35: Operatori a corto circuito (AndAlso - OrElse)

Sintassi

- risultato = espressione1 AndAlso espressione2
- risultato = espressione1 OrElse espressione2

Parametri

Parametro	Dettagli
risultato	Necessario. Qualsiasi espressione booleana. Il risultato è il risultato booleano del confronto tra le due espressioni.
espressione1	Necessario. Qualsiasi espressione booleana.
espressione2	Necessario. Qualsiasi espressione booleana.

Osservazioni

'AndAlso' e **'OrElse'** sono operatori **ShortCircuiting** che significa che l'esecuzione è più breve perché il compilatore non valuta tutte le espressioni in un confronto booleano se il primo fornisce il risultato desiderato.

Examples

E anche l'uso

```
' Sometimes we don't need to evaluate all the conditions in an if statement's boolean check.

' Let's suppose we have a list of strings:

Dim MyCollection as List(Of String) = New List(of String)()

' We want to evaluate the first value inside our list:

If MyCollection.Count > 0 And MyCollection(0).Equals("Somevalue")
    Console.WriteLine("Yes, I've found Somevalue in the collection!")
End If

' If MyCollection is empty, an exception will be thrown at runtime.
' This because it evaluates both first and second condition of the
' if statement regardless of the outcome of the first condition.

' Now let's apply the AndAlso operator
```

```
If MyCollection.Count > 0 AndAlso MyCollection(0).Equals("Somevalue")
    Console.WriteLine("Yes, I've found Somevalue in the collection!")
End If

' This won't throw any exception because the compiler evaluates just the first condition.
' If the first condition returns False, the second expression isn't evaluated at all.
```

Uso OrElse

```
' The OrElse operator is the homologous of AndAlso. It lets us perform a boolean
' comparison evaluating the second condition only if the first one is False

If testFunction(5) = True OrElse otherFunction(4) = True Then
    ' If testFunction(5) is True, otherFunction(4) is not called.
    ' Insert code to be executed.
End If
```

Evitare NullReferenceException

7.0

O altro

```
Sub Main()
    Dim elements As List(Of Integer) = Nothing

    Dim average As Double = AverageElementsOrElse(elements)
    Console.WriteLine(average) ' Writes 0 to Console

    Try
        'Throws ArgumentNullException
        average = AverageElementsOr(elements)
    Catch ex As ArgumentNullException
        Console.WriteLine(ex.Message)
    End Try
End Sub

Public Function AverageElementsOrElse(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is not called if elements is Nothing so it cannot crash
    If (elements Is Nothing OrElse elements.Count = 0) Then
        Return 0
    Else
        Return elements.Average()
    End If
End Function

Public Function AverageElementsOr(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is always called so it can crash if elements is Nothing
    If (elements Is Nothing Or elements.Count = 0) Then
        Return 0
    Else
        Return elements.Average()
    End If
End Function
```

7.0

E anche

```
Sub Main()
    Dim elements As List(Of Integer) = Nothing

    Dim average As Double = AverageElementsAndAlso(elements)
    Console.WriteLine(average) ' Writes 0 to Console

    Try
        'Throws ArgumentNullException
        average = AverageElementsAnd(elements)
    Catch ex As ArgumentNullException
        Console.WriteLine(ex.Message)
    End Try
End Sub

Public Function AverageElementsAndAlso(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is not called if elements is Nothing so it cannot crash
    If (Not elements Is Nothing AndAlso elements.Count > 0) Then
        Return elements.Average()
    Else
        Return 0
    End If
End Function

Public Function AverageElementsAnd(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is always called so it can crash if elements is Nothing
    If (Not elements Is Nothing And elements.Count > 0) Then
        Return elements.Average()
    Else
        Return 0
    End If
End Function
```

14.0

Visual Basic 14.0 ha introdotto l'operatore condizionale null , consentendo di riscrivere le funzioni in modo più pulito, imitando il comportamento della versione `AndAlso` dell'esempio.

Leggi Operatori a corto circuito (AndAlso - OrElse) online: https://riptutorial.com/it/vb-net/topic/2509/operatori-a-corto-circuito--andalso---orelse-

Capitolo 36: Opzione esplicita

Osservazioni

`Option Explicit On` è una buona pratica consigliata con Visual Basic .Net. Ti aiuta come sviluppatore a produrre codice più pulito, più stabile, più privo di bug e più manutenibile. In alcuni casi può anche aiutarti a scrivere programmi con prestazioni migliori!

con riferimento a https://support.microsoft.com/it-it/kb/311329#bookmark-3 è inoltre possibile utilizzare l'opzione strict al posto dell'opzione esplicita. Opzione strict eredita l'opzione esplicita.

Examples

Che cos'è?

Ti costringe a dichiarare esplicitamente tutte le variabili.

Qual è la differenza tra dichiarare esplicitamente e dichiarare implicitamente una variabile?

Dichiarare esplicitamente una variabile:

```
Dim anInteger As Integer = 1234
```

Dichiarare implicitamente una variabile:

```
'Did not declare aNumber using Dim
aNumber = 1234
```

Conclusione

Pertanto, dovresti sempre avere `Option Explicit On` quanto potresti perdere una variabile durante l'assegnazione, il che causa il comportamento imprevisto del programma.

Come accenderlo?

Livello di documento

È attivo per impostazione predefinita, ma puoi avere un ulteriore livello di protezione posizionando `Option Explicit On` nella parte superiore del file di codice. L'opzione si applicherà all'intero documento.

Livello del progetto

Puoi accenderlo tramite il menu in Visual Studio:

> Progetto> [Progetto] Proprietà> scheda Compila> Opzione esplicita

Scegli `On` nel menu a discesa. L'opzione si applicherà all'intero documento.

Tutti i nuovi progetti

Puoi attivarlo di default per tutti i nuovi progetti selezionando:

> Strumenti> Opzioni> Progetti e soluzioni> Valori predefiniti VB> Opzione Esplicita

Scegli `On` nel menu a discesa.

Leggi Opzione esplicita online: https://riptutorial.com/it/vb-net/topic/4725/opzione-esplicita

Capitolo 37: Opzione Inferiore

Examples

Che cos'è?

Abilita l'uso dell'inferenza di tipo locale nella dichiarazione delle variabili.

Cos'è l'inferenza di tipo?

È possibile dichiarare variabili locali senza dichiarare esplicitamente un tipo di dati. Il compilatore deduce il tipo di dati di una variabile dal tipo della sua espressione di inizializzazione.

Opzione Inferiore su :

```
Dim aString  = "1234" '--> Will be treated as String by the compiler
Dim aNumber  = 4711   '--> Will be treated as Integer by the compiler
```

vs. dichiarazione di tipo esplicita:

```
'State a type explicitly
Dim aString as String = "1234"
Dim aNumber as Integer = 4711
```

Opzione Inferiore Off:
Il comportamento del compilatore con `Option Infer Off` dipende dall'impostazione `Option Strict` che è già documentata qui .

- **Opzione Inferiore Off - Opzione Rimozione off**
  Tutte le variabili senza dichiarazioni di tipo esplicite sono dichiarate come `Object` .

  ```
  Dim aString  = "1234" '--> Will be treated as Object by the compiler
  ```

- **Opzione Inferiore - Opzione Rigorosa**
  Il compilatore non ti consente di dichiarare una variabile senza un tipo esplicito.

  ```
  'Dim aString  = "1234" '--> Will not compile due to missing type in declaration
  ```

Come abilitarlo / disabilitarlo

Livello di documento

È `Option Infer On|Off` per impostazione predefinita, ma è possibile impostarlo posizionando `Option Infer On|Off` nella parte superiore del file di codice. L'opzione si applicherà all'intero documento.

Livello del progetto

Puoi accenderlo / spegnerlo tramite il menu in Visual Studio:

> Progetto> [Progetto] Proprietà> Scheda Compila> Inferiore opzione

Scegli `On|Off` nel menu a discesa. L'opzione si applicherà all'intero documento.

Tutti i nuovi progetti

Puoi attivarlo di default per tutti i nuovi progetti selezionando:

> Strumenti> Opzioni> Progetti e soluzioni> Valori predefiniti VB> Opzione Inferiore

Scegli `On|Off` nel menu a discesa.

Quando utilizzare l'inferenza del tipo

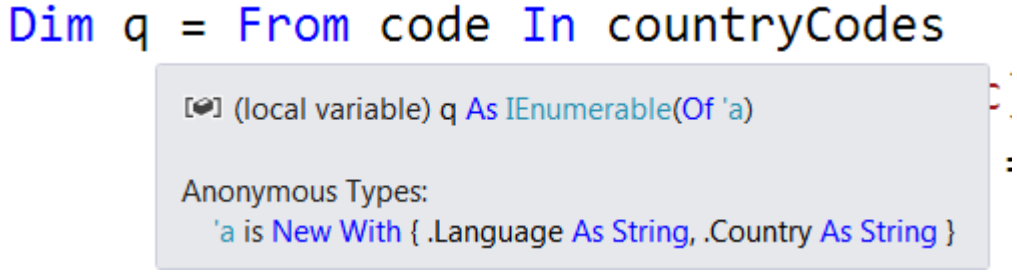
Fondamentalmente è possibile utilizzare l'inferenza di tipo ogni volta che è possibile.
Tuttavia, fai attenzione quando combini `Option Infer Off` e `Option Strict Off` , in quanto ciò può portare a un comportamento di runtime indesiderato:

```
Dim someVar = 5
someVar.GetType.ToString() '--> System.Int32
someVar = "abc"
someVar.GetType.ToString() '--> System.String
```

Tipo anonimo
I tipi anonimi possono essere dichiarati **solo** con `Option Infer On` .
Sono spesso usati quando si ha a che fare con LINQ :

```
Dim countryCodes = New List(Of String)
countryCodes.Add("en-US")
countryCodes.Add("en-GB")
countryCodes.Add("de-DE")
countryCodes.Add("de-AT")

Dim q = From code In countryCodes
        Let split = code.Split("-"c)
        Select New With {.Language = split(0), .Country = split(1)}
```

- **Opzione Inferiore**
  Il compilatore riconoscerà il tipo anonimo:



- **Opzione Inferiore**
  Il compilatore genererà un errore (con `Option Strict On`)

o considererà `q` come `object` tipo (con `Option Strict Off`).
Entrambi i casi produrranno il risultato che non è possibile utilizzare il tipo anonimo.

Doubles / decimali
Le variabili numeriche con cifre decimali verranno dedotte come `Double` per impostazione predefinita:

```
Dim aNumber = 44.11 '--> Will be treated as type `Double` by the compiler
```

Se si desidera un altro tipo come `Decimal` il valore inizializzato della variabile deve essere contrassegnato:

```
Dim mDecimal = 47.11D '--> Will be treated as type `Decimal` by the compiler
```

Leggi Opzione Inferiore online: https://riptutorial.com/it/vb-net/topic/5095/opzione-inferiore

Capitolo 38: Opzione rigorosa

Sintassi

- Option Strict {On | Off}

Osservazioni

`Option Strict On` è una buona pratica consigliata con Visual Basic .Net. Ti aiuta come sviluppatore a produrre codice più pulito, più stabile, più privo di bug e più manutenibile. In alcuni casi può anche aiutarti a scrivere programmi con prestazioni migliori, evitando cose come Conversione implicita.

`On` *non* è l'impostazione predefinita per una nuova installazione di Visual Studio. Dovrebbe essere una delle prime cose cambiate prima di iniziare la programmazione se si intende utilizzare VB.NET. La ragione per cui non è l'impostazione predefinita viene dalle prime edizioni di Visual Studio quando ci si aspettava che i programmatori stessero migrando i progetti da VB6.

Examples

Perché usarlo?

`Option Strict On` impedisce a tre cose di accadere:

1. Errori di conversione restringimento impliciti

Impedisce all'utente di assegnare a una variabile che ha *meno precisione o minore capacità* (una conversione di restringimento) senza un cast esplicito. Fare ciò comporterebbe una perdita di dati.

```
Dim d As Double = 123.4
Dim s As Single = d 'This line does not compile with Option Strict On
```

2. Chiamate in ritardo

L'associazione tardiva non è consentita. Questo serve a prevenire errori di battitura che potrebbero essere compilati, ma falliscono in fase di runtime

```
Dim obj As New Object
obj.Foo 'This line does not compile with Option Strict On
```

3. Errori del tipo di oggetto implicito

Ciò impedisce che la variabile venga dedotta come oggetto quando in realtà avrebbero dovuto essere dichiarati come un tipo

```
Dim something = Nothing. 'This line does not compile with Option Strict On
```

Conclusione

A meno che non sia necessario eseguire l'associazione tardiva, è sempre necessario `Option Strict On` poiché causerà errori di compilazione con errori di compilazione anziché eccezioni di runtime.

Se *avete* a che fare l'associazione tardiva, è *possibile*

- Racchiudere tutte le chiamate in ritardo in una classe / modulo e utilizzare `Option Strict Off` nella parte superiore del file di codice (questo è il metodo preferito in quanto riduce la probabilità di errori di battitura in altri file), *o*
- Specificare che Late Binding non causa un errore di compilazione (`Project Properties > Compile Tab > Warning Configuration`)

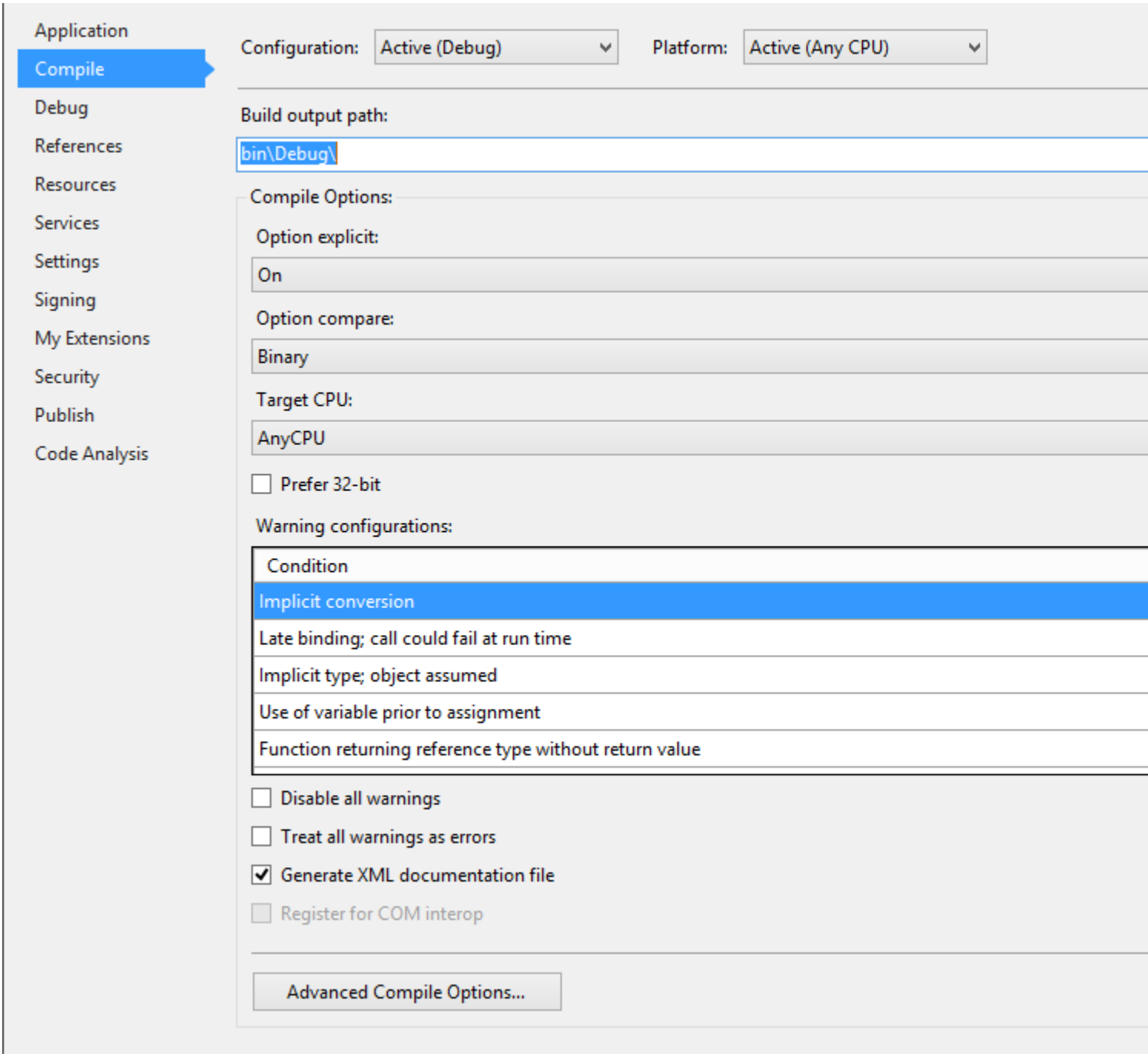
Come accenderlo

- È possibile accenderlo a livello di modulo / classe posizionando la direttiva nella parte superiore del file di codice.

  ```
  Option Strict On
  ```

- Puoi accenderlo a livello di progetto tramite il menu in Visual Studio

  Progetto> [Progetto] Proprietà> Scheda Compila> Opzione Rigorosa> Attiva



- Puoi attivarlo di default per tutti i nuovi progetti selezionando:

  Strumenti> Opzioni> Progetti e soluzioni> Valori predefiniti VB> Opzione rigorosa
  Impostalo su `On` .

Leggi Opzione rigorosa online: https://riptutorial.com/it/vb-net/topic/4022/opzione-rigorosa

Capitolo 39: Parole chiave ByVal e ByRef

Examples

Parola chiave ByVal

Parola chiave ByVal prima che il parametro method (o nessuna parola chiave come ByVal sia assunto per impostazione predefinita) dice che il parametro verrà inviato in un modo che **non** consente al metodo di modificare (assegnare un nuovo valore) la variabile sottostante il parametro.
Non impedisce il contenuto (o lo stato) dell'argomento da modificare se si tratta di una classe.

```
Class SomeClass
    Public Property Member As Integer
End Class

Module Program
    Sub Main()
        Dim someInstance As New SomeClass With {.Member = 42}

        Foo (someInstance)
        ' here someInstance is not Nothing (still the same object)
        ' but someInstance.Member is -42 (internal state can still be changed)

        Dim number As Integer = 42
        Foo(number)
        ' here number is still 42
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub

    Sub Foo(arg As Integer) ' No ByVal or ByRef keyword, ByVal is assumed
        arg = -arg
    End Sub
End Module
```

Parola chiave ByRef

La parola chiave ByRef prima del parametro method indica che il parametro verrà inviato in un modo che consente al metodo di modificare (assegnare un nuovo valore) la variabile sottostante il parametro.

```
Class SomeClass
    Public Property Member As Integer
End Class

Module Program
    Sub Main()
        Dim someInstance As New SomeClass With {.Member = 42}
```

```
        Foo (someInstance)
        ' here someInstance is not Nothing
        ' but someInstance.Member is -42

        Bar(someInstance)
        ' here someInstance is Nothing
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub

    Sub Bar(ByRef param As Integer)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub
End Module
```

Leggi Parole chiave ByVal e ByRef online: https://riptutorial.com/it/vb-net/topic/4653/parole-chiave-byval-e-byref

Capitolo 40: Parole chiave OOP

Examples

Definire una classe

Le lezioni sono aspetti vitali dell'OOP. Una classe è come il "progetto" di un oggetto. Un oggetto ha le proprietà di una classe, ma le caratteristiche non sono definite all'interno della classe stessa. Poiché ogni oggetto può essere diverso, definiscono le loro caratteristiche.

```
Public Class Person
End Class

Public Class Customer
End Class
```

Una classe può anche contenere *sottoclassi* . Una sottoclasse eredita le stesse proprietà e comportamenti della sua classe genitore, ma può avere le sue proprietà e classi uniche.

Modificatori dell'ereditarietà (sulle classi)

eredita

Specifica la classe base (o genitore)

```
Public Class Person
End Class

Public Class Customer
    Inherits Person

End Class

'One line notation
Public Class Student : Inherits Person
End Class
```

Oggetti possibili:

```
Dim p As New Person
Dim c As New Customer
Dim s As New Student
```

NotInheritable

Impedisce ai programmatori di utilizzare la classe come classe base.

```
Public NotInheritable Class Person
End Class
```

Oggetti possibili:

```
Dim p As New Person
```

MustInherit

Specifica che la classe è concepita per essere utilizzata solo come classe base. (Classe astratta)

```
Public MustInherit Class Person
End Class

Public Class Customer
    Inherits Person
End Class
```

Oggetti possibili:

```
Dim c As New Customer
```

Modificatori di ereditarietà (su proprietà e metodi)

Overridable

Permette di sovrascrivere una proprietà o un metodo in una classe in una classe derivata.

```
Public Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class
```

Sostituzioni

Sovrascrive una proprietà o un metodo Overridable definito nella classe base.

```
Public Class Customer
    Inherits Person

    'Base Class must be Overridable
    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub
End Class
```

NotOverridable

Impedisce che una proprietà o un metodo vengano sovrascritti in una classe ereditaria. Comportamento predefinito Può essere dichiarato solo sui **metodi di sostituzione**

```
Public Class Person

    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub

End Class

Public Class Customer
    Inherits Person

    Public NotOverridable Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class

Public Class DetailedCustomer
    Inherits Customer

    'DoSomething can't be overridden
End Class
```

Esempio di utilizzo:

```
Dim p As New Person
p.DoSomething()

Dim c As New Customer
c.DoSomething()

Dim d As New DetailedCustomer
d.DoSomething()
```

Produzione:

```
Person
Customer
Customer
```

MustOverride

Richiede che una classe derivata sostituisca la proprietà o il metodo.

I metodi MustOverride devono essere dichiarati nelle **classi MustInherit.**

```
Public MustInherit Class Person

    Public MustOverride Sub DoSomething()
    'No method definition here

End Class

Public Class Customer
    Inherits Person

    'DoSomething must be overridden
    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Esempio di utilizzo:

```
Dim c As New Customer
c.DoSomething()
```

Produzione:

```
Customer
```

MyBase

La parola chiave MyBase si comporta come una variabile oggetto che fa riferimento alla classe base dell'istanza corrente di una classe.

```
Public Class Person
    Public Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class

Public Class Customer
    Inherits Person

    Public Sub DoSomethingElse()
        MyBase.DoSomething()
    End Sub

End Class
```

Esempio di utilizzo:

```
Dim p As New Person
p.DoSomething()

Console.WriteLine("----")

Dim c As New Customer
c.DoSomething()
```

```
c.DoSomethingElse()
```

Produzione:

```
Person
----
Person
Person
```

Me vs MyClass

Io usa l'istanza dell'oggetto corrente.

MyClass utilizza memberdefinition nella classe in cui viene chiamato il membro

```
Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub

    Public Sub useMe()
        Me.DoSomething()
    End Sub

    Public Sub useMyClass()
        MyClass.DoSomething()
    End Sub
End Class

Class Customer
    Inherits Person

    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub
End Class
```

Esempio di utilizzo:

```
Dim c As New Customer
c.useMe()
c.useMyClass()
```

Produzione:

```
Customer
Person
```

Sovraccarico

Il sovraccarico è la creazione di più di una procedura, costruttore di istanze o proprietà in una classe con lo stesso nome ma tipi di argomenti diversi.

```
Class Person
    Overloads Sub Display(ByVal theChar As Char)
        ' Add code that displays Char data.
    End Sub

    Overloads Sub Display(ByVal theInteger As Integer)
        ' Add code that displays Integer data.
    End Sub

    Overloads Sub Display(ByVal theDouble As Double)
        ' Add code that displays Double data.
    End Sub
End Class
```

Shadows

Ricalcola un membro che non è sovrascrivibile. Saranno interessate solo le chiamate all'istanza. Il codice all'interno delle classi base non sarà influenzato da questo.

```
Public Class Person
    Public Sub DoSomething()
        Console.WriteLine("Person")
    End Sub


    Public Sub UseMe()
        Me.DoSomething()
    End Sub
End Class
Public Class Customer
    Inherits Person
    Public Shadows Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Esempio di utilizzo:

```
Dim p As New Person
Dim c As New Customer
p.UseMe()
c.UseMe()
Console.WriteLine("----")
p.DoSomething()
c.DoSomething()
```

Produzione:

```
Person
Person
----
Person
Customer
```

Insidie :

Esempio 1, Creazione di un nuovo oggetto tramite un generico. Quale funzione verrà utilizzata ??

```
Public Sub CreateAndDoSomething(Of T As {Person, New})()
    Dim obj As New T
    obj.DoSomething()
End Sub
```

esempio di utilizzo:

```
Dim p As New Person
p.DoSomething()
Dim s As New Student
s.DoSomething()
Console.WriteLine("----")
CreateAndDoSomething(Of Person)()
CreateAndDoSomething(Of Student)()
```

Uscita: per intuizione il risultato dovrebbe essere lo stesso. Eppure non è vero.

```
Person
Student
----
Person
Person
```

Esempio 2:

```
Dim p As Person
Dim s As New Student
p = s
p.DoSomething()
s.DoSomething()
```

Output: per intuizione si potrebbe pensare che p e s siano uguali e si comportino in modo uguale. Eppure non è vero.

```
Person
Student
```

In questi semplici esempi è facile imparare lo strano comportamento di Shadows. Ma nella vita reale porta molte sorprese. È consigliabile evitare l'uso di ombre. Uno dovrebbe usare altre alternative il più possibile (overrides ecc.)

interfacce

```
Public Interface IPerson
    Sub DoSomething()
End Interface

Public Class Customer
```

```
    Implements IPerson
    Public Sub DoSomething() Implements IPerson.DoSomething
        Console.WriteLine("Customer")
    End Sub

End Class
```

Leggi Parole chiave OOP online: https://riptutorial.com/it/vb-net/topic/4273/parole-chiave-oop

Capitolo 41: Pattern asincrono basato su attività

Examples

Utilizzo di base di Async / Attesa

È possibile avviare un processo lento in parallelo e quindi raccogliere i risultati quando vengono eseguiti:

```
Public Sub Main()
    Dim results = Task.WhenAll(SlowCalculation, AnotherSlowCalculation).Result

    For Each result In results
        Console.WriteLine(result)
    Next
End Sub

Async Function SlowCalculation() As Task(Of Integer)
     Await Task.Delay(2000)

     Return 40
End Function

Async Function AnotherSlowCalculation() As Task(Of Integer)
    Await Task.Delay(2000)

    Return 60
End Function
```

Dopo due secondi entrambi i risultati saranno disponibili.

Utilizzo di TAP con LINQ

È possibile creare un `IEnumerable` di `Task` passando `AddressOf AsyncMethod` al **LINQ** `Select` metodo e quindi avviare e attendere tutti i risultati con `Task.WhenAll`

Se il tuo metodo ha parametri corrispondenti alla precedente chiamata a catena **LINQ** , verranno automaticamente mappati.

```
Public Sub Main()
    Dim tasks = Enumerable.Range(0, 100).Select(AddressOf TurnSlowlyIntegerIntoString)

    Dim resultingStrings = Task.WhenAll(tasks).Result

    For Each value In resultingStrings
        Console.WriteLine(value)
    Next
End Sub

Async Function TurnSlowlyIntegerIntoString(input As Integer) As Task(Of String)
```

```
    Await Task.Delay(2000)

    Return input.ToString()
End Function
```

Per mappare diversi argomenti è possibile sostituire il `AddressOf Method` con un lambda:

```
Function(linqData As Integer) MyNonMatchingMethod(linqData, "Other parameter")
```

Leggi Pattern asincrono basato su attività online: https://riptutorial.com/it/vb-net/topic/2936/pattern-asincrono-basato-su-attivita

Capitolo 42: ricorsione

Examples

Calcola n numero di Fibonacci

Visual Basic.NET, come la maggior parte delle lingue, permette la ricorsione, un processo mediante il quale una funzione chiama *se stessa* in determinate condizioni.

Ecco una funzione di base in Visual Basic .NET per calcolare i numeri di Fibonacci .

```
''' <summary>
''' Gets the n'th Fibonacci number
''' </summary>
''' <param name="n">The 1-indexed ordinal number of the Fibonacci sequence that you wish to
receive. Precondition: Must be greater than or equal to 1.</param>
''' <returns>The nth Fibonacci number. Throws an exception if a precondition is
violated.</returns>
Public Shared Function Fibonacci(ByVal n as Integer) as Integer
    If n<1
        Throw New ArgumentOutOfRangeException("n must be greater than or equal to one.")
    End If
    If (n=1) or (n=2)
        ''Base case. The first two Fibonacci numbers (n=1 and n=2) are both 1, by definition.
        Return 1
    End If
    ''Recursive case.
    ''Get the two previous Fibonacci numbers via recursion, add them together, and return the
result.
    Return Fibonacci(n-1) + Fibonacci(n-2)
End Function
```

Questa funzione funziona controllando innanzitutto se la funzione è stata chiamata con il parametro `n` uguale a `1` o `2` . Per definizione, i primi due valori nella sequenza di Fibonacci sono 1 e 1, quindi non è necessario alcun ulteriore calcolo per determinarlo. Se `n` è maggiore di 2, non possiamo cercare facilmente il valore associato, ma sappiamo che qualsiasi numero di Fibonacci è uguale alla somma dei due numeri precedenti, quindi li richiediamo tramite *ricorsione* (chiamando la nostra funzione Fibonacci). Poiché le chiamate ricorsive successive vengono chiamate con numeri sempre più piccoli tramite decrementi di -1 e -2, sappiamo che alla fine raggiungeranno numeri inferiori a 2. Una volta che tali condizioni (chiamate *casi base*) vengono raggiunte, lo stack si svolge e noi ottieni il nostro risultato finale.

Leggi ricorsione online: https://riptutorial.com/it/vb-net/topic/7862/ricorsione

Capitolo 43: Riflessione

Examples

Recupera le proprietà per un'istanza di una classe

```
Imports System.Reflection

Public Class PropertyExample

    Public Function GetMyProperties() As PropertyInfo()
        Dim objProperties As PropertyInfo()
        objProperties = Me.GetType.GetProperties(BindingFlags.Public Or BindingFlags.Instance)
        Return objProperties
    End Function

    Public Property ThisWillBeRetrieved As String = "ThisWillBeRetrieved"

    Private Property ThisWillNot As String = "ThisWillNot"

    Public Shared Property NeitherWillThis As String = "NeitherWillThis"

    Public Overrides Function ToString() As String
        Return String.Join(",", GetMyProperties.Select(Function(pi) pi.Name).ToArray)
    End Function
End Class
```

Il parametro di GetProperties definisce quali tipi di proprietà verranno restituiti dalla funzione. Poiché passiamo Public e Instance, il metodo restituirà solo proprietà pubbliche e non condivise. Vedi l'attributo e le spiegazioni di The Flags su come combinare le enumerazioni.

Ottieni i membri di un tipo

```
Dim flags = BindingFlags.Static Or BindingFlags.Public Or BindingFlags.Instance
Dim members = GetType(String).GetMembers(flags)
For Each member In members
    Console.WriteLine($"{member.Name}, ({member.MemberType})")
Next
```

Ottieni un metodo e invocalo

Metodo statico:

```
Dim parseMethod = GetType(Integer).GetMethod("Parse",{GetType(String)})
Dim result = DirectCast(parseMethod.Invoke(Nothing,{"123"}), Integer)
```

Metodo di istanza:

```
 Dim instance = "hello".ToUpper
 Dim method = Gettype(String).GetMethod("ToUpper",{})
```

```
Dim result = method.Invoke(instance,{})
Console.WriteLine(result) 'HELLO
```

Crea un'istanza di un tipo generico

```
    Dim openListType = GetType(List(Of ))
    Dim typeParameters = {GetType(String)}
    Dim stringListType = openListType.MakeGenericType(typeParameters)
    Dim instance = DirectCast(Activator.CreateInstance(stringListType), List(Of String))
    instance.Add("Hello")
```

Leggi Riflessione online: https://riptutorial.com/it/vb-net/topic/1598/riflessione

Capitolo 44: schieramento

Osservazioni

```
Dim myArray(2) As Integer

someFunc(myArray)
```

Una matrice è una raccolta di oggetti ordinata per indice. Il tipo di oggetto è definito dal tipo indicato nella dichiarazione dell'array.

Le matrici in Visual Basic .NET sono più comunemente (e per impostazione predefinita) basate su zero (0), il che significa che il primo indice è 0. Un array di 10 elementi avrà un intervallo di indice compreso tra 0 e 9. Quando si accede agli elementi dell'array, l'indice accessibile massimo è inferiore al numero totale di elementi. Per questo motivo, i cicli che accedono agli indici di array in modo incrementale dovrebbero sempre eseguire un controllo di intervallo in cui il valore è inferiore alla lunghezza dell'array.

Examples

Definizione di matrice

```
Dim array(9) As Integer ' Defines an array variable with 10 Integer elements (0-9).

Dim array = New Integer(10) {} ' Defines an array variable with 11 Integer elements (0-10)
                               'using New.

Dim array As Integer() = {1, 2, 3, 4} ' Defines an Integer array variable and populate it
                                      'using an array literal. Populates the array with
                                      '4 elements.

ReDim Preserve array(10) ' Redefines the size of an existing array variable preserving any
                         'existing values in the array. The array will now have 11 Integer
                         'elements (0-10).

ReDim array(10) ' Redefines the size of an existing array variable discarding any
                'existing values in the array. The array will now have 11 Integer
                'elements (0-10).
```

Zero-Based

Tutti gli array in VB.NET sono basati su zero. In altre parole, l'indice del primo elemento (il limite inferiore) in un array VB.NET è sempre `0` . Le versioni precedenti di VB, come VB6 e VBA, erano basate su una sola impostazione predefinita, ma fornivano un modo per ignorare i limiti predefiniti. In quelle versioni precedenti di VB, i limiti inferiore e superiore potevano essere dichiarati esplicitamente (es. `Dim array(5 To 10)` .In VB.NET, per mantenere la compatibilità con altri linguaggi .NET, quella flessibilità era stata rimossa e il limite inferiore `0` viene ora sempre

applicato. Tuttavia, la sintassi `To` può ancora essere utilizzata in VB.NET, che può rendere l'intervallo più esplicitamente chiaro. Ad esempio, i seguenti esempi sono tutti equivalenti a quelli sopra elencati:

```
Dim array(0 To 9) As Integer

Dim array = New Integer(0 To 10) {}

ReDim Preserve array(0 To 10)

ReDim array(0 To 10)
```

Dichiarazioni di array annidati

```
Dim myArray = {{1, 2}, {3, 4}}
```

Dichiarare una matrice a dimensione singola e impostare i valori degli elementi dell'array

```
Dim array = New Integer() {1, 2, 3, 4}
```

o

```
Dim array As Int32() = {1, 2, 3, 4}
```

Inizializzazione di array

```
Dim array() As Integer = {2, 0, 1, 6}                   ''Initialize an array of four
Integers.
Dim strings() As String = {"this", "is", "an", "array"} ''Initialize an array of four Strings.
Dim floats() As Single = {56.2, 55.633, 1.2, 5.7743, 22.345}
              ''Initialize an array of five Singles, which are the same as floats in C#.
Dim miscellaneous() as Object = { New Object(), "Hello", New List(of String) }
              ''Initialize an array of three references to any reference type objects
              ''and point them to objects of three different types.
```

Inizializzazione di array multidimensionali

```
Dim array2D(,) As Integer = {{1, 2, 3}, {4, 5, 6}}
' array2D(0, 0) is 1 ; array2D(0, 1) is 2 ; array2D(1, 0) is 4

Dim array3D(,,) As Integer = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}}
' array3D(0, 0, 0) is 1 ; array3D(0, 0, 1) is 2
' array3D(0, 1, 0) is 4 ; array3D(1, 0, 0) is 7
```

Inizializzazione di Jagged Array

Nota la parentesi per distinguere tra una matrice frastagliata e una matrice multidimensionale I sottarray possono essere di lunghezza diversa

```
Dim jaggedArray()() As Integer = { ({1, 2, 3}), ({4, 5, 6}), ({7}) }
' jaggedArray(0) is {1, 2, 3} and so jaggedArray(0)(0) is 1
' jaggedArray(1) is {4, 5, 6} and so jaggedArray(1)(0) is 4
' jaggedArray(2) is {7} and so jaggedArray(2)(0) is 7
```

Null Array Variables

Poiché le matrici sono tipi di riferimento, una variabile di matrice può essere nullo. Per dichiarare una variabile di matrice nulla, devi dichiararla senza una dimensione:

```
Dim array() As Integer
```

O

```
Dim array As Integer()
```

Per verificare se un array è nullo, prova a vedere se non `Is Nothing` :

```
Dim array() As Integer
If array Is Nothing Then
    array = {1, 2, 3}
End If
```

Per impostare una variabile array esistente su null, è sufficiente impostarla su `Nothing` :

```
Dim array() As Integer = {1, 2, 3}
array = Nothing
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Oppure usa `Erase` , che fa la stessa cosa:

```
Dim array() As Integer = {1, 2, 3}
Erase array
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Riferimento alla stessa matrice da due variabili

Poiché gli array sono tipi di riferimento, è possibile avere più variabili che puntano allo stesso oggetto matrice.

```
Dim array1() As Integer = {1, 2, 3}
Dim array2() As Integer = array1
array1(0) = 4
Console.WriteLine(String.Join(", ", array2))  ' Writes "4, 2, 3"
```

Limiti inferiori diversi da zero

Con `Option Strict On` , sebbene .NET Framework consenta la creazione di matrici a dimensione singola con limiti inferiori non zero, non sono "vettori" e quindi non sono compatibili con gli array

tipizzati VB.NET. Ciò significa che possono essere visti solo come `Array` e quindi non possono utilizzare riferimenti di array `(index)` normali.

```
Dim a As Array = Array.CreateInstance(GetType(Integer), {4}, {-1})
For y = LBound(a) To UBound(a)
    a.SetValue(y * y, y)
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {a.GetValue(y)}")
Next
```

Oltre a utilizzare `Option Strict Off` , puoi recuperare la sintassi `(index)` trattando l'array come un `IList` , ma non è un array, quindi non puoi usare `LBound` e `UBound` su quel nome di variabile (e tu? non stiamo ancora evitando la boxe):

```
Dim nsz As IList = a
For y = LBound(a) To UBound(a)
    nsz(y) = 2 - CInt(nsz(y))
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {nsz(y)}")
Next
```

Le matrici con limite inferiore non-zero multidimensionale *sono* compatibili con gli array tipografici multidimensionali VB.NET:

```
Dim nza(,) As Integer = DirectCast(Array.CreateInstance(GetType(Integer),
                                    {4, 3}, {1, -1}), Integer(,))
For y = LBound(nza) To UBound(nza)
    For w = LBound(nza, 2) To UBound(nza, 2)
        nza(y, w) = -y * w + nza(UBound(nza) - y + LBound(nza),
                                 UBound(nza, 2) - w + LBound(nza, 2))
    Next
Next
For y = LBound(nza) To UBound(nza)
    Dim ly = y
    Console.WriteLine(String.Join(" ",
        Enumerable.Repeat(ly & ":", 1).Concat(
            Enumerable.Range(LBound(nza, 2), UBound(nza, 2) - LBound(nza, 2) + 1) _
            .Select(Function(w) CStr(nza(ly, w))))))
Next
```

Riferimento MSDN: Array.CreateInstance

Leggi schieramento online: https://riptutorial.com/it/vb-net/topic/805/schieramento

Capitolo 45: Server FTP

Sintassi

- My.Computer.Network.DownloadFile (file server come stringa, file locale come stringa)
- My.Computer.Network.DownloadFile (serverFile As String, localFile As String, user As String, password As String)
- My.Computer.Network.UploadFile (localFile As String, serverFile As String)
- My.Computer.Network.UploadFile (localFile As String, serverFile As String, user As String, password As String)

Examples

Scarica il file dal server FTP

```
My.Computer.Network.DownloadFile("ftp://server.my/myfile.txt", "donwloaded_file.txt")
```

Questo comando scarica il file `myfile.txt` dal server denominato `server.my` e lo salva come `donwloaded_file.txt` nella directory di lavoro. È possibile specificare il percorso assoluto per il file scaricato.

Scarica il file dal server FTP quando è richiesto il login

```
My.Computer.Network.DownloadFile("ftp://srv.my/myfile.txt", "donwload.txt", "Peter", "1234")
```

Questo download comando `myfile.txt` file dal server denominato `srv.my` e lo salva come `donwload.txt` nella directory di lavoro. È possibile specificare il percorso assoluto per il file scaricato. Il file è scaricato dall'utente Peter con password 1234.

Carica il file sul server FTP

```
My.Computer.Network.UploadFile("example.txt", "ftp://server.my/server_example.txt")
```

Questo comando carica il file `example.txt` dalla directory di lavoro (è possibile specificare il percorso assoluto se lo si desidera) sul server denominato `server.my` . Il file memorizzato sul server sarà denominato `server_example.txt` .

Carica il file sul server FTP quando è richiesto l'accesso

```
My.Computer.Network.UploadFile("doc.txt", "ftp://server.my/on_server.txt", "Peter", "1234")
```

Questo comando carica il file `doc.txt` dalla directory di lavoro (è possibile specificare il percorso assoluto se lo si desidera) sul server denominato `server.my` . Il file memorizzato sul server sarà

denominato `server_example.txt` . Il riempimento è inviato sul server dall'utente Peter e la password 1234.

Leggi Server FTP online: https://riptutorial.com/it/vb-net/topic/4078/server-ftp

Capitolo 46: Test unitario in VB.NET

Osservazioni

Questo è l'esempio più semplice ma descrittivo per la procedura di test unitario. Sentiti libero di aggiungere altri metodi per verificare diversi tipi di dati.

Examples

Test unitario per il calcolo delle tasse

Questo esempio è diviso in due pilastri

- **SalaryCalculation Class** : calcolo dello stipendio netto al netto delle detrazioni fiscali
- **Classe SalaryCalculationTests** : per testare il metodo che calcola il salario netto

Passaggio 1: creare la libreria di classi, denominarla libreria **salari** o qualsiasi nome appropriato. Quindi rinomina la classe in **SalaryCalculation**

" " " 'Classe per calcoli salariali' " Classe pubblica Calcolo salariale

```
    ''' <summary>
    ''' Employee Salary
    ''' </summary>
    Public Shared Salary As Double

    ''' <summary>
    ''' Tax fraction (0-1)
    ''' </summary>
    Public Shared Tax As Double

    ''' <summary>
    ''' Function to calculate Net Salary
    ''' </summary>
    ''' <returns></returns>
    Public Shared Function CalculateNetSalary()
        Return Salary - Salary * Tax
    End Function
End Class
```

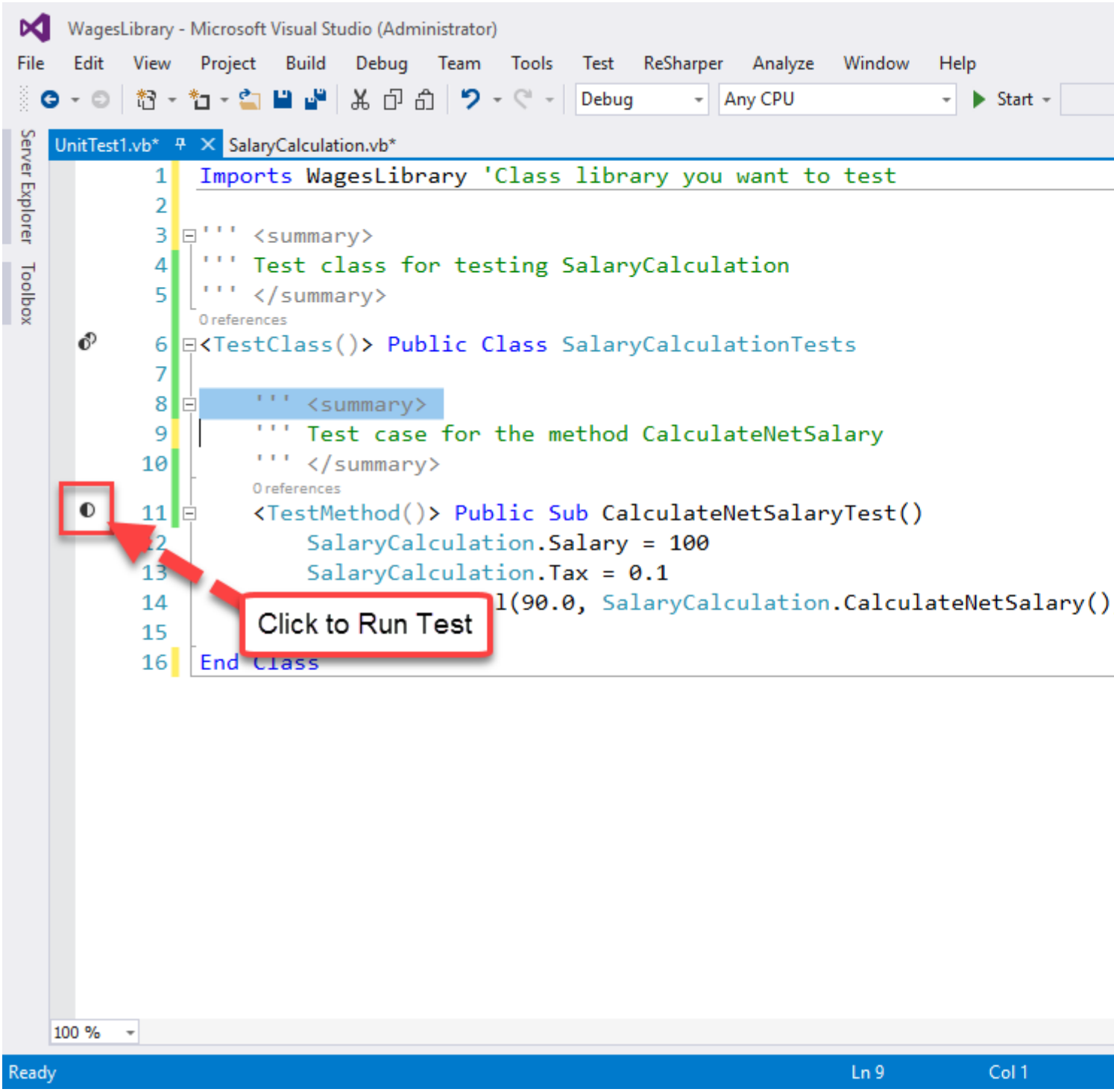
Passaggio 2 : creare un progetto di test unitario. Aggiungi riferimento alla libreria di classi creata e incolla il codice seguente

```
Imports WagesLibrary 'Class library you want to test

''' <summary>
''' Test class for testing SalaryCalculation
''' </summary>
<TestClass()> Public Class SalaryCalculationTests

    ''' <summary>
```

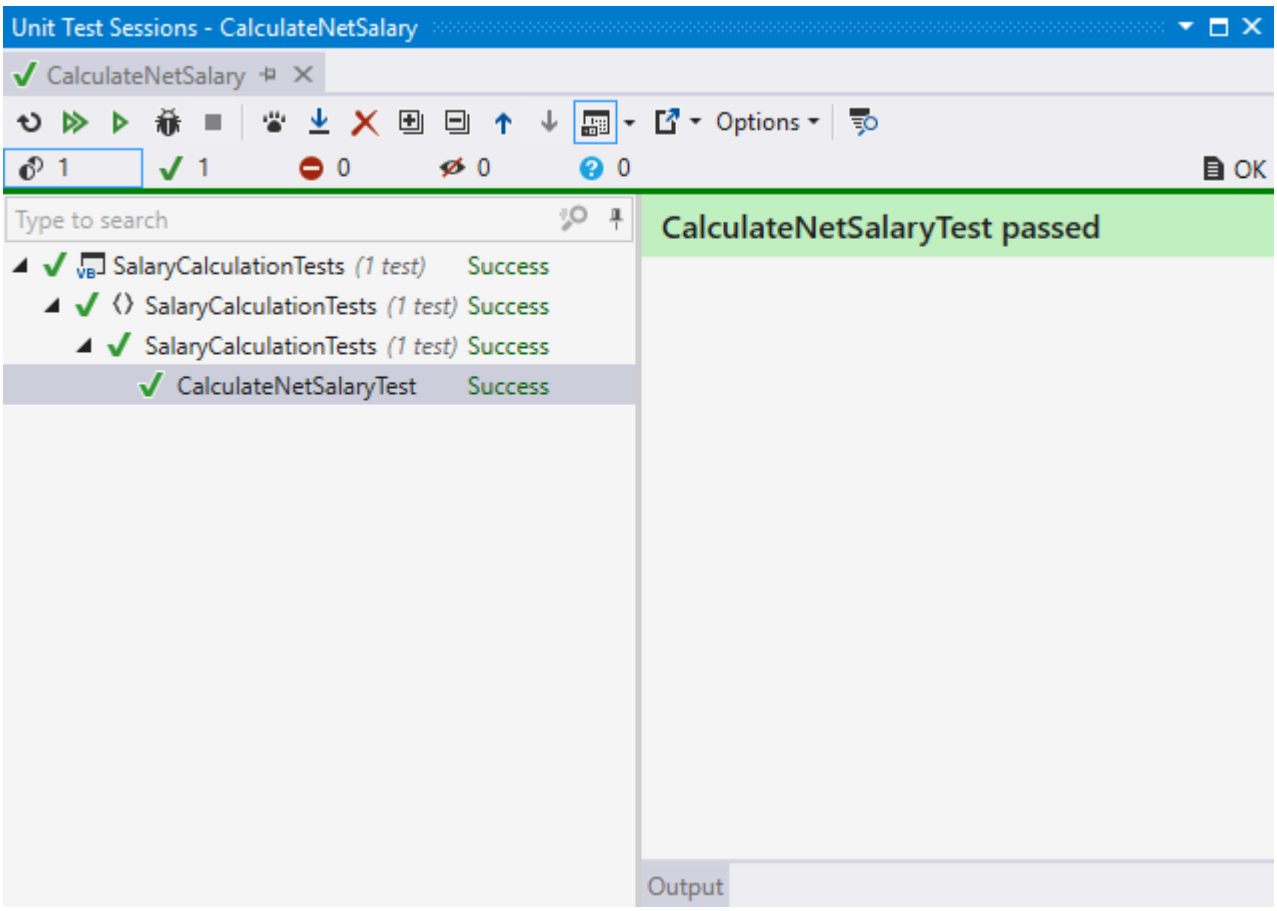
```
    ''' Test case for the method CalculateNetSalary
    ''' </summary>
    <TestMethod()> Public Sub CalculateNetSalaryTest()
        SalaryCalculation.Salary = 100
        SalaryCalculation.Tax = 0.1
        Assert.AreEqual(90.0, SalaryCalculation.CalculateNetSalary(), 0.1)
    End Sub
End Class
```

`Assert.Equal` controlla il valore previsto rispetto al valore calcolato effettivo. il valore `0.1` viene utilizzato per consentire la tolleranza o la variazione tra il risultato *previsto* e quello *effettivo* .

Passaggio 3: eseguire il test del metodo per vedere il risultato



Risultato del test



Test delle proprietà assegnate e derivate dalla classe dipendente

Questo esempio ha più test disponibili nei test unitari.

Employee.vb (Class Library)

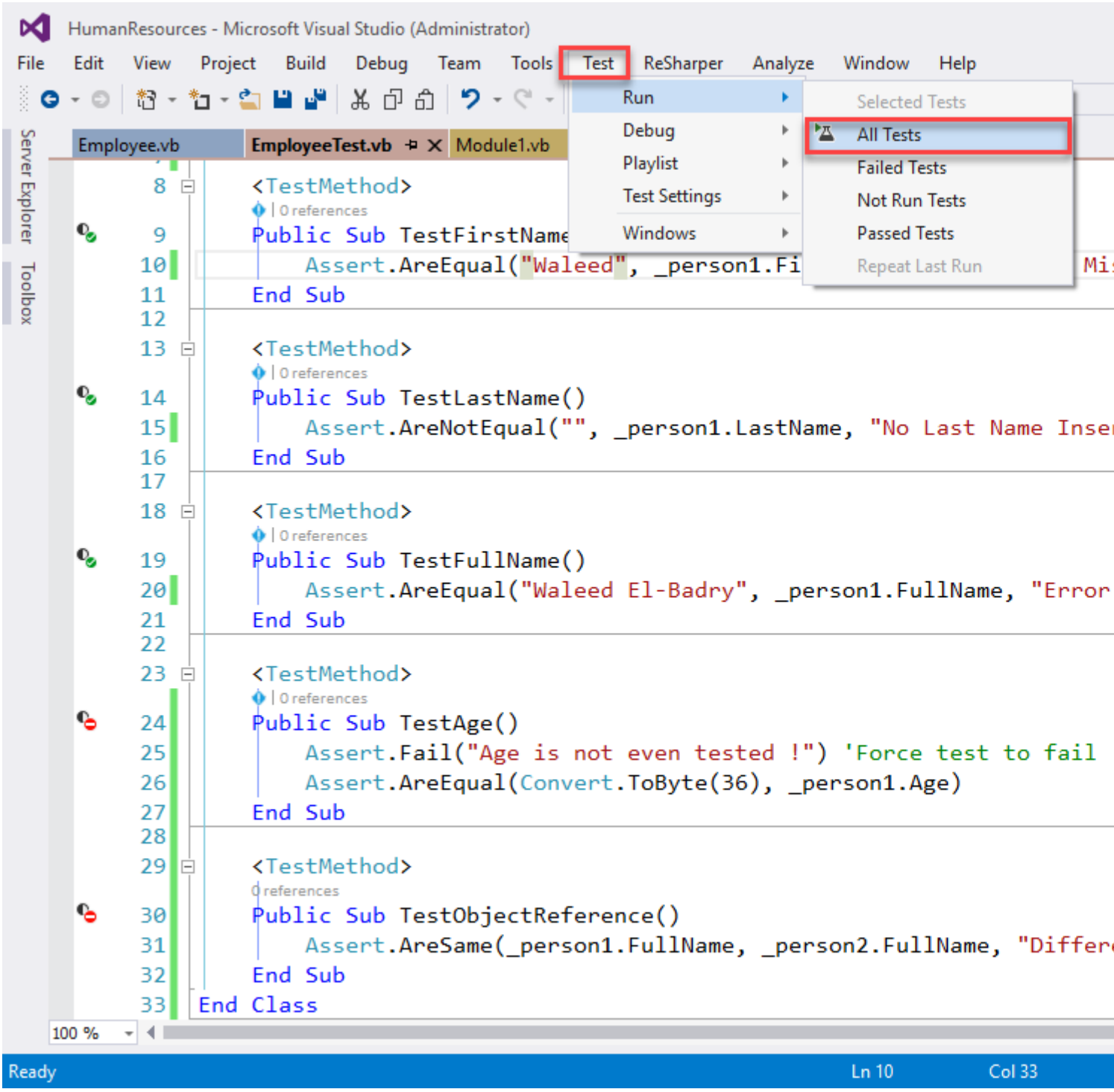
```
''' <summary>
''' Employee Class
''' </summary>
Public Class Employee

    ''' <summary>
    ''' First name of employee
    ''' </summary>
    Public Property FirstName As String = ""

    ''' <summary>
    ''' Last name of employee
    ''' </summary>
    Public Property LastName As String = ""

    ''' <summary>
    ''' Full name of employee
    ''' </summary>
    Public ReadOnly Property FullName As String = ""

    ''' <summary>
    ''' Employee's age
    ''' </summary>
```

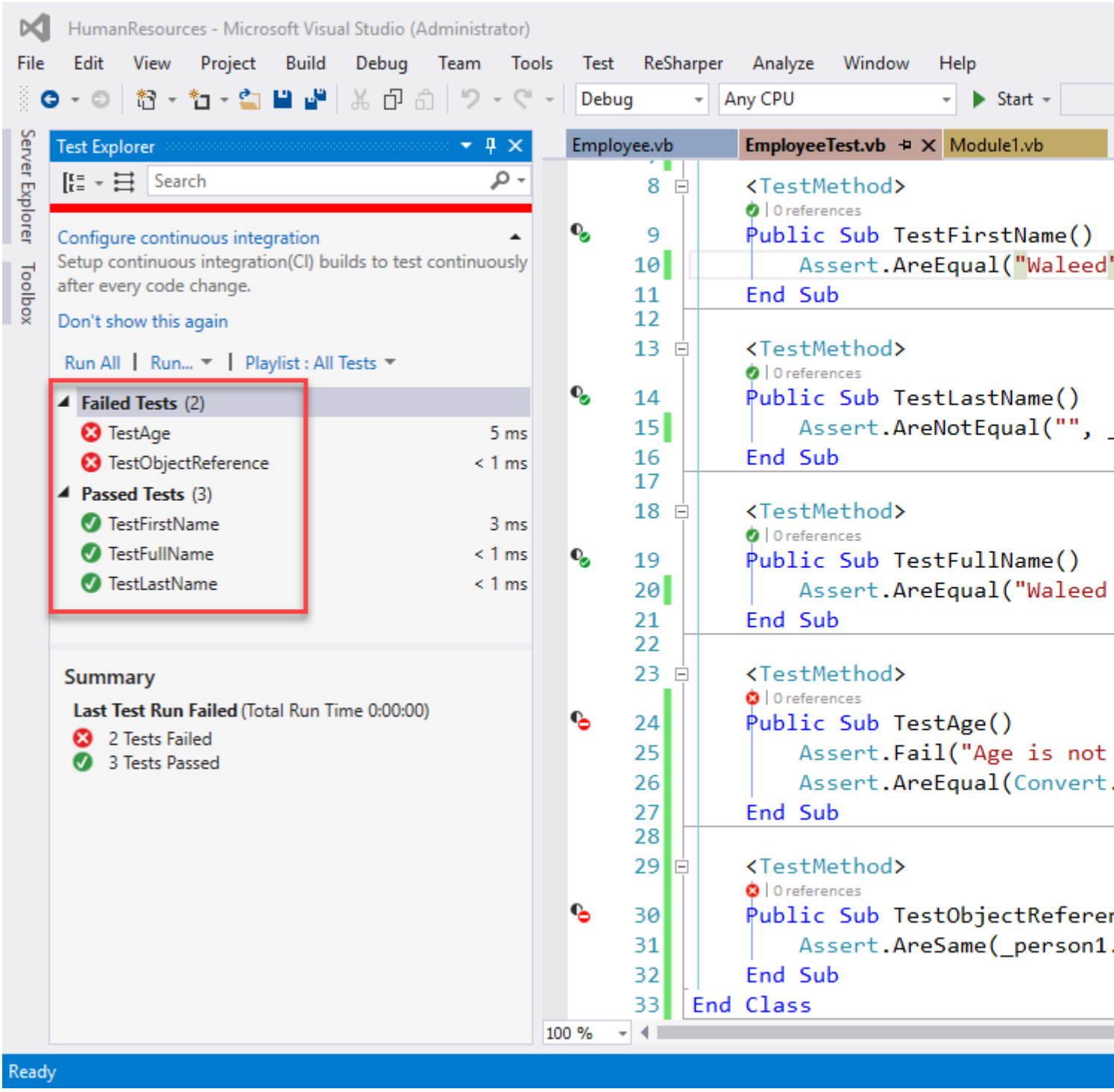
```
    Public Property Age As Byte

    ''' <summary>
    ''' Instantiate new instance of employee
    ''' </summary>
    ''' <param name="firstName">Employee first name</param>
    ''' <param name="lastName">Employee last name</param>
    Public Sub New(firstName As String, lastName As String, dateofbirth As Date)
        Me.FirstName = firstName
        Me.LastName = lastName
        FullName = Me.FirstName + " " + Me.LastName
        Age = Convert.ToByte(Date.Now.Year - dateofbirth.Year)
    End Sub
End Class
```

EmployeeTest.vb (Test Project)

```
Imports HumanResources

<TestClass()>
Public Class EmployeeTests
    ReadOnly _person1 As New Employee("Waleed", "El-Badry", New DateTime(1980, 8, 22))
    ReadOnly _person2 As New Employee("Waleed", "El-Badry", New DateTime(1980, 8, 22))

    <TestMethod>
    Public Sub TestFirstName()
        Assert.AreEqual("Waleed", _person1.FirstName, "First Name Mismatch")
    End Sub

    <TestMethod>
    Public Sub TestLastName()
        Assert.AreNotEqual("", _person1.LastName, "No Last Name Inserted!")
    End Sub

    <TestMethod>
    Public Sub TestFullName()
        Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error in concatination of
names")
    End Sub

    <TestMethod>
    Public Sub TestAge()
        Assert.Fail("Age is not even tested !") 'Force test to fail !
        Assert.AreEqual(Convert.ToByte(36), _person1.Age)
    End Sub

    <TestMethod>
    Public Sub TestObjectReference()
        Assert.AreSame(_person1.FullName, _person2.FullName, "Different objects with same
data")
    End Sub
End Class
```

Risultato dopo aver eseguito i test


HumanResources - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Tools Test ReSharper Analyze Window Help
Run
Debug
Playlist
Test Settings
Windows
Selected Tests
All Tests
Failed Tests
Not Run Tests
Passed Tests
Repeat Last Run
Server Explorer
Toolbox
Employee.vb
EmployeeTest.vb
Module1.vb
<TestMethod>
0 references
Public Sub TestFirstName
Assert.AreEqual("Waleed", _person1.Fi
End Sub
<TestMethod>
0 references
Public Sub TestLastName()
Assert.AreNotEqual("", _person1.LastName, "No Last Name Inse
End Sub
<TestMethod>
0 references
Public Sub TestFullName()
Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error
End Sub
<TestMethod>
0 references
Public Sub TestAge()
Assert.Fail("Age is not even tested !") 'Force test to fail
Assert.AreEqual(Convert.ToByte(36), _person1.Age)
End Sub
<TestMethod>
0 references
Public Sub TestObjectReference()
Assert.AreSame(_person1.FullName, _person2.FullName, "Differ
End Sub
End Class
100 %
Ready
Ln 10
Col 33



Leggi Test unitario in VB.NET online: https://riptutorial.com/it/vb-net/topic/6843/test-unitario-in-vb-net

Capitolo 47: threading

Examples

Esecuzione di chiamate thread-safe tramite Control.Invoke ()

Utilizzando il metodo `Control.Invoke()` è possibile spostare l'esecuzione di un metodo o una funzione da un thread in background al thread su cui è stato creato il controllo, che di solito è il thread dell'interfaccia utente (User Interface). In questo modo il codice verrà messo in coda per l'esecuzione sul thread del controllo, che rimuoverà la possibilità di concorrenza.

La proprietà `Control.InvokeRequired` dovrebbe inoltre essere controllata per determinare se è necessario richiamare o se il codice è già in esecuzione sullo stesso thread del controllo.

Il metodo `Invoke()` accetta un delegato come primo parametro. Un delegato mantiene il riferimento, l'elenco dei parametri e il tipo restituito con un altro metodo.

In Visual Basic 2010 (10.0) o versioni successive, è possibile utilizzare *espressioni lambda* per creare al volo un metodo delegato:

```
If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(Sub() LogTextBox.AppendText("Check passed"))
Else
    LogTextBox.AppendText("Check passed")
End If
```

Considerando che in Visual Basic 2008 (9.0) o inferiore, è necessario dichiarare il delegato per conto proprio:

```
Delegate Sub AddLogText(ByVal Text As String)

If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(New AddLogText(AddressOf UpdateLog), "Check passed")
Else
    UpdateLog("Check passed")
End If

Sub UpdateLog(ByVal Text As String)
    LogTextBox.AppendText(Text)
End Sub
```

Esecuzione di chiamate thread-safe usando Async / Attesa

Se proviamo a cambiare un oggetto sul thread dell'interfaccia utente da un thread diverso, otterremo un'eccezione di operazione cross-thread:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' Cross thread-operation exception as the assignment is executed on a different thread
    ' from the UI one:
```

```
    Task.Run(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Prima di **VB 14.0** e **.NET 4.5** la soluzione stava invocando l'assegnazione on e object living sul thread dell'interfaccia utente:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the conde on the UI thread:
    MyButton.Invoke(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Con **VB 14.0,** possiamo eseguire `Task` su un thread diverso e quindi sfruttare al contesto ripristinata quando l'esecuzione è completa e quindi eseguire l'assegnazione con asincrono / Await:

```
Private Async Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the code on a different thread then the context is restored
    ' so the assignment happens on the UI thread:
    MyButton.Text = Await Task.Run(Function() Thread.CurrentThread.ManagedThreadId)
End Sub
```

Leggi threading online: https://riptutorial.com/it/vb-net/topic/1913/threading

Capitolo 48: Utilizzando axWindowsMediaPlayer in VB.Net

introduzione

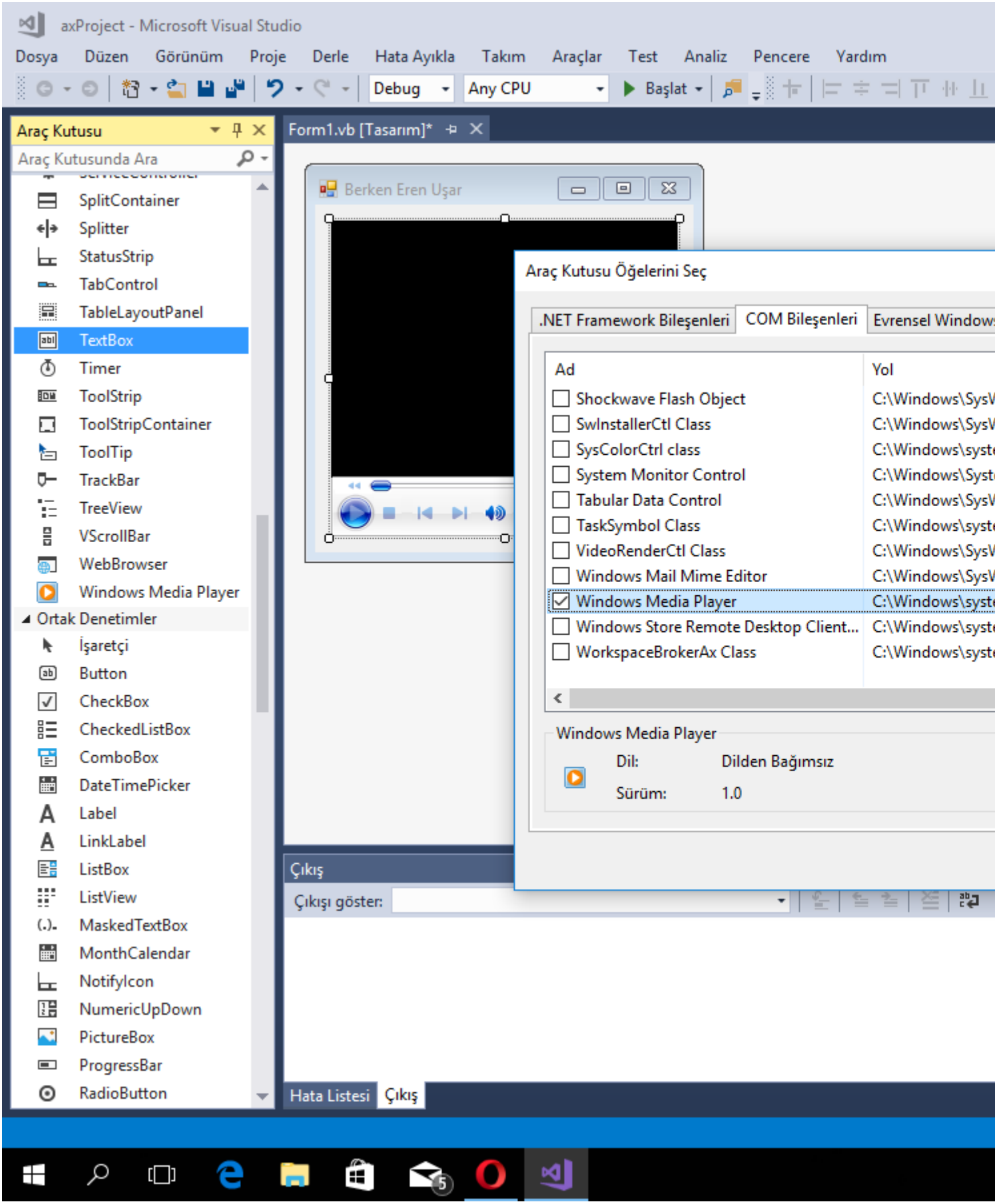
axWindowsMediaPlayer è il controllo per la riproduzione di file multimediali come video e musica.

Examples

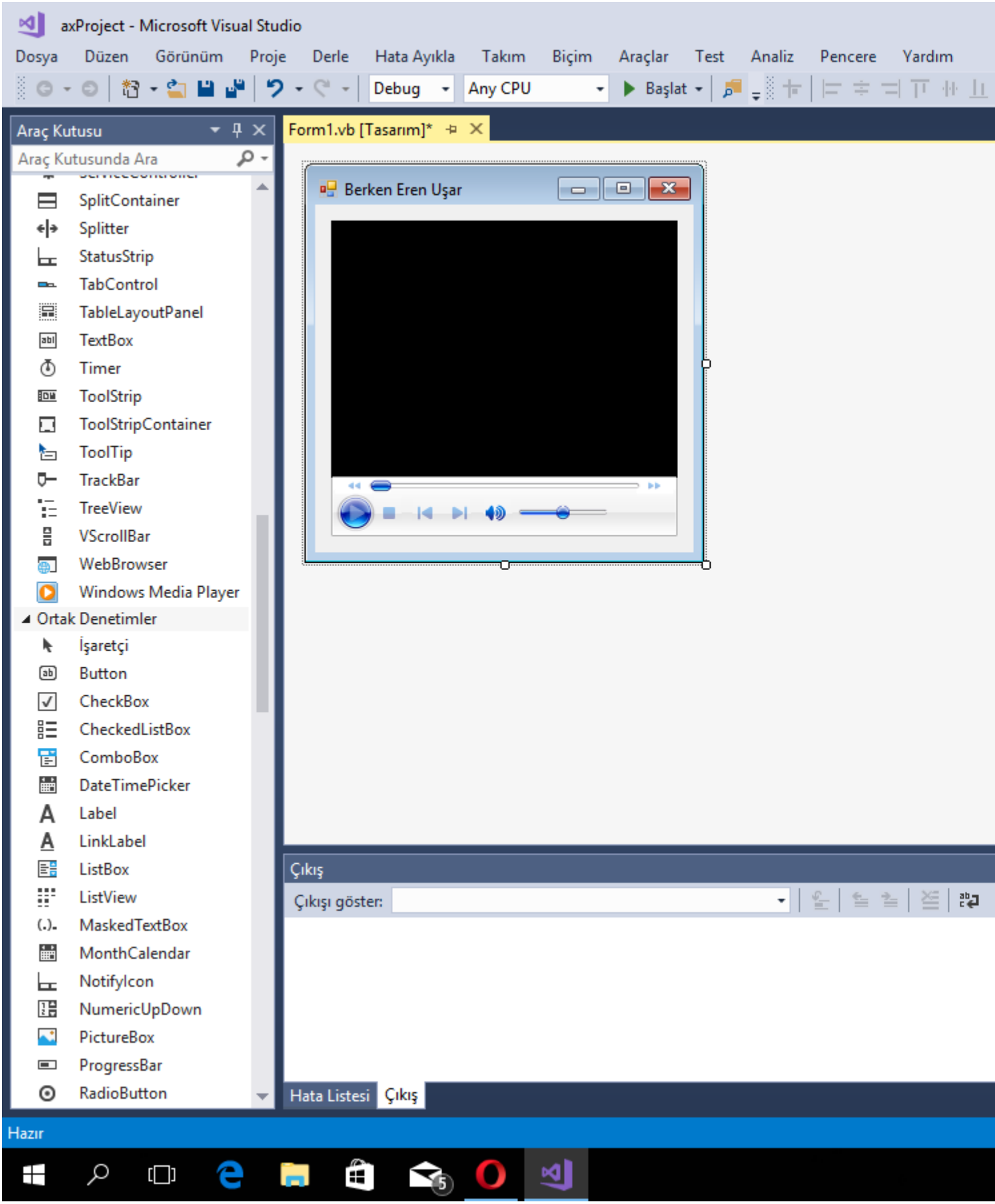
Aggiunta di axWindowsMediaPlayer

- Fai clic con il pulsante destro del mouse sulla casella degli strumenti, quindi fai clic su "Scegli elementi".
- Selezionare la scheda Componenti COM e quindi controllare Windows Media Player.
- axWindowsMediaPlayer verrà aggiunto a Toolbox.

Seleziona questa casella di controllo per utilizzare axWindowsMediaPlayer



Quindi puoi usare axWindowsMediaPlayer :)



Riproduci un file multimediale

```
AxWindowsMediaPlayer1.URL = "C:\My Files\Movies\Avatar.mp4"
```

```
AxWindowsMediaPlayer1.Ctlcontrols.play()
```

Questo codice giocherà Avatar in axWindowsMediaPlayer.

Leggi Utilizzando axWindowsMediaPlayer in VB.Net online: https://riptutorial.com/it/vb-net/topic/10096/utilizzando-axwindowsmediaplayer-in-vb-net

Capitolo 49: Utilizzando BackgroundWorker

Examples

Implementazione di base della classe worker Background

È necessario importare System.ComponentModel per l'utilizzo di background worker

```
Imports System.ComponentModel
```

Quindi dichiara una variabile privata

```
Private bgWorker As New BackgroundWorker
```

È necessario creare due metodi per gli eventi DoWork ed RunWorkerCompleted di Background worker e assegnarli.

```
Private Sub MyWorker_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs)
  'Add your codes here for the worker to execute

End Sub
```

Il sotto sotto verrà eseguito quando l'operatore termina il lavoro

```
Private Sub MyWorker_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs)
    'Add your codes for the worker to execute after finishing the work.

End Sub
```

Quindi all'interno del codice aggiungere le righe seguenti per avviare l'operatore in background

```
    bgWorker = New BackgroundWorker
    AddHandler bgWorker.DoWork, AddressOf MyWorker_DoWork
    AddHandler bgWorker.RunWorkerCompleted, AddressOf MyWorker_RunWorkerCompleted
    bgWorker.RunWorkerAsync()
```

Quando chiami la funzione RunWorkerAsync (), MyWorker_DoWork verrà eseguito.

Leggi Utilizzando BackgroundWorker online: https://riptutorial.com/it/vb-net/topic/6401/utilizzando-backgroundworker

Capitolo 50: Utilizzando la dichiarazione

Sintassi

- Usando *a* = New *DisposableClass* [, *b* = ...]
  ...
  Fine Uso
- Utilizzando *a* = *GetDisposable (...)* [, *b* = ...]
  ...
  Fine Uso

Examples

Vedi esempi sotto Oggetti usa e getta

Concetto di base di IDisposable

Leggi Utilizzando la dichiarazione online: https://riptutorial.com/it/vb-net/topic/7965/utilizzando-la-dichiarazione

Capitolo 51: WinForms SpellCheckBox

introduzione

Esempio su come aggiungere una casella di controllo ortografico a un'applicazione WindowsForms. Questo esempio NON richiede l'installazione di Word né utilizza Word in alcun modo.

Usa Interoperabilità WPF usando il controllo ElementHost per creare un UserControl WPF da un TextBox WPF. WPF TextBox ha una funzione integrata per il controllo ortografico. Utilizzeremo questa funzione incorporata piuttosto che fare affidamento su un programma esterno.

Examples

ElementHost WPF TextBox

Questo esempio è stato modellato su un esempio che ho trovato su internet. Non riesco a trovare il link o darei credito all'autore. Ho preso il campione che ho trovato e modificato per funzionare per la mia applicazione.

1. Aggiungi i seguenti riferimenti:

   System.Xaml, PresentationCore, PresentationFramework, WindowsBase e WindowsFormsIntegration

2. Crea una nuova classe e oltre questo codice

```
Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design.Serialization
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Forms.Integration
Imports System.Windows.Forms.Design

<Designer(GetType(ControlDesigner))> _
Class SpellCheckBox
Inherits ElementHost

Private box As TextBox

Public Sub New()
    box = New TextBox()
    MyBase.Child = box
    AddHandler box.TextChanged, AddressOf box_TextChanged
    box.SpellCheck.IsEnabled = True
    box.VerticalScrollBarVisibility = ScrollBarVisibility.Auto
    Me.Size = New System.Drawing.Size(100, 20)
End Sub

Private Sub box_TextChanged(ByVal sender As Object, ByVal e As EventArgs)
```

```
    OnTextChanged(EventArgs.Empty)
End Sub

<DefaultValue("")> _
Public Overrides Property Text() As String
    Get
        Return box.Text
    End Get
    Set(ByVal value As String)
        box.Text = value
    End Set
End Property

<DefaultValue(True)> _
Public Property MultiLine() As Boolean
    Get
        Return box.AcceptsReturn
    End Get
    Set(ByVal value As Boolean)
        box.AcceptsReturn = value
    End Set
End Property

<DefaultValue(True)> _
Public Property WordWrap() As Boolean
    Get
        Return box.TextWrapping <> TextWrapping.Wrap
    End Get
    Set(ByVal value As Boolean)
        If value Then
            box.TextWrapping = TextWrapping.Wrap
        Else
            box.TextWrapping = TextWrapping.NoWrap
        End If
    End Set
End Property

<DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
Public Shadows Property Child() As System.Windows.UIElement
    Get
        Return MyBase.Child
    End Get
    Set(ByVal value As System.Windows.UIElement)
        '' Do nothing to solve a problem with the serializer !!
    End Set
End Property

End Class
```

3. Ricostruisci la soluzione.
4. Aggiungi un nuovo modulo.
5. Cerca nella casella degli strumenti il nome della tua classe. Questo esempio è "SpellCheck". Dovrebbe essere elencato sotto Componenti 'YourSoulutionName'.
6. Trascina il nuovo controllo nel modulo
7. Impostare una qualsiasi delle proprietà mappate nell'evento di caricamento dei moduli

```
Private Sub form1_Load(sender As Object, e As EventArgs) Handles Me.Load
    spellcheckbox.WordWrap = True
    spellcheckbox.MultiLin = True
    'Add any other property modifiers here...
End Sub
```

7. L'ultima cosa che devi fare è cambiare la consapevolezza DPI della tua applicazione. Questo perché stai usando l'applicazione WinForms. Di default tutte le applicazioni WinForms sono DPI UNAWARE. Una volta eseguito un controllo con un host di elementi (WPF Interop), l'applicazione diventerà DPI AWARE. Questo può o non può incasinare con i tuoi Elementi dell'interfaccia utente. La soluzione a questo è forzare l'applicazione a diventare DPI UNAWARE. Ci sono 2 modi per farlo. Il primo è attraverso il file manifest e il secondo è quello di inserire il codice nel tuo programma. Se si utilizza OneClick per distribuire l'applicazione, è necessario codificarlo, non utilizzare il file manifest o gli errori saranno inevitabili.

Entrambi i seguenti esempi possono essere trovati al seguente: WinForms Scaling a grandi impostazioni DPI - È anche possibile? Grazie a Telerik.com per l'ottima spiegazione su DPI.

> Esempio di codice DPI a codice hard codificato. Questo DEVE essere eseguito prima che il primo modulo sia inizializzato. Lo posto sempre nel file ApplicationEvents.vb. È possibile accedere a questo file facendo clic con il pulsante destro del mouse sul nome del progetto in Solution Explorer e scegliendo "Apri". Quindi selezionare la scheda dell'applicazione a sinistra e quindi fare clic su "Visualizza eventi applicazione" in basso a destra accanto al menu a discesa della schermata iniziale.

```
Namespace My

    ' The following events are available for MyApplication:
    '
    ' Startup: Raised when the application starts, before the startup form is created.
    ' Shutdown: Raised after all application forms are closed.  This event is not raised if
the application terminates abnormally.
    ' UnhandledException: Raised if the application encounters an unhandled exception.
    ' StartupNextInstance: Raised when launching a single-instance application and the
application is already active.
    ' NetworkAvailabilityChanged: Raised when the network connection is connected or
disconnected.
    Partial Friend Class MyApplication

    Private Enum PROCESS_DPI_AWARENESS
        Process_DPI_Unaware = 0
        Process_System_DPI_Aware = 1
        Process_Per_Monitor_DPI_Aware = 2
    End Enum

    Private Declare Function SetProcessDpiAwareness Lib "shcore.dll" (ByVal Value As
PROCESS_DPI_AWARENESS) As Long

    Private Sub SetDPI()
        'Results from SetProcessDPIAwareness
        'Const S_OK = &H0&
        'Const E_INVALIDARG = &H80070057
        'Const E_ACCESSDENIED = &H80070005
```

```
        Dim lngResult As Long

        lngResult = SetProcessDpiAwareness(PROCESS_DPI_AWARENESS.Process_DPI_Unaware)

    End Sub

    Private Sub MyApplication_Startup(sender As Object, e As
ApplicationServices.StartupEventArgs) Handles Me.Startup
        SetDPI()
    End Sub

End Namespace
```

Esempio manifesto

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3" >
   <asmv3:application>
        <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
             <dpiAware>true</dpiAware>
        </asmv3:windowsSettings>
   </asmv3:application>
</assembly>
```

Leggi WinForms SpellCheckBox online: https://riptutorial.com/it/vb-net/topic/8624/winforms-spellcheckbox

Titoli di coda

S. No	Capitoli	Contributors
1	Introduzione al linguaggio Visual Basic .NET	Bjørn-Roger Kringsjå, Cary Bondoc, Community, Happypig375, Harjot, Jonathan Nixon, Martin Verjans, MDTech.us_MAN, Misaz, Natalie Orsi, Nico Agusta, Ryan Thomas, StardustGogeta, Virtual Anomaly
2	Accesso ai dati	MatVAD, Mike Robertson, Nico Agusta
3	Associazione dati XAML WPF	Milliron X
4	BackgroundWorker	Jones Joseph, Shayan Toqraee
5	Casuale	David Wilson
6	Classi	Darren Davies, Harjot
7	Compressione file / cartella	Cody Gray, MatVAD, Misaz, vbnet3d
8	condizioni	Allen Binuya, Chetan Sanghani, Nathan Tuggy, Robert Columbia
9	consolle	Bugs, InteXX, lucamauri, Martin Soles, Matthew Whited, Sam Axe, Slava Auer, StardustGogeta, vbnet3d, VortixDev
10	Data	Matt Wilko, Misaz
11	Debug della tua applicazione	Martin Verjans
12	Dichiarazione di variabili	Cody Gray, Darren Davies, Fütemire, glaubergft, keronconk, LogicalFlaps, MatVAD, RamenChef, Sehnsucht
13	Digita la conversione	Cary Bondoc, LogicalFlaps, vbnet3d
14	dizionari	DrDonut, Nathan, Proger_Cbsk, Sehnsucht, void
15	elenchi	Dman, DrDonut, Fütemire, Luke Sheppard, Robert Columbia, Seandk
16	enum	4444, CiccioRocca, David Sdot, djv, ElektroStudios, kodkod, LogicalFlaps, Shog9, Steven Doggart, void

17	Funzionalità di Visual Basic 14.0	Adam Zuckerman, Bjørn-Roger Kringsjå, Blackwood, Crazy Britt, Drake, Fütemire, Gridly, jColeson, liserdarts, Matt Wilko, Nadeem_MK, Nitram, Sam Axe, Stefano d'Antonio, yummypasta
18	funzioni	Berken Usar
19	GDI +	Dman
20	Generics	JDC
21	Gestione degli errori	Adam Zuckerman, Bjørn-Roger Kringsjå, Happypig375, Luke Sheppard, MatVAD, Nico Agusta, Vishal
22	Gestione dei file	Dan Granger, Luke Sheppard, Matt Wilko, Misaz, Shayan Toqraee, vbnet3d
23	Gestione delle connessioni	Jonas_Hess
24	Google Maps in un Windows Form	anonymous, Carlos Borau
25	Introduzione alla sintassi	Bugs, Mark Hurd, Martin Verjans, mnoronha, Nat G., Nico Agusta, Sehnsucht
26	Lavorare con Windows Forms	djv, SilverShotBee, vbnet3d
27	Leggere file di testo compresso al volo	Proger_Cbsk
28	LINQ	Dan Drews, Daz, Derek Tomes, Fütemire, H. Pauwelyn, Mark Hurd, Misaz, Sam Axe, Sehnsucht, Zev Spitz
29	looping	CiccioRocca, debater, Imran Ali Khan, Mark, MatVAD, Sam Axe, Scott Mitchell, SilverShotBee, TyCobb, vbnet3d, void
30	Metodi di estensione	Fütemire, InteXX, Matt Wilko, Sam Axe, Stefano d'Antonio, void
31	multithreading	4444, Daz, MatVAD
32	NullReferenceException	Alessandro Mascolo, RamenChef, Sehnsucht
33	Oggetti usa e getta	KE0GSD, Mark Hurd, Martin Verjans, Matt Wilko, Misaz, Mithrandir, Sam Axe
34	operatori	Bjørn-Roger Kringsjå, Cary Bondoc, MatVAD, Mike Robertson, Pasilda, Robert Columbia, RoyalPotato, Sam Axe

		, Sree, varocarbas, void
35	Operatori a corto circuito (AndAlso - OrElse)	Bart Jolling, CiccioRocca, Kendra, Sam Axe
36	Opzione esplicita	Happypig375, Matt Wilko, sansknwoledge
37	Opzione Inferiore	Alex B., LogicalFlaps, Mark Hurd
38	Opzione rigorosa	Andrew Morton, Cary Bondoc, Matt Wilko, RussAwesome, Sam Axe, vbnet3d
39	Parole chiave ByVal e ByRef	Adam Zuckerman, Misaz, Sehnsucht
40	Parole chiave OOP	4444, David, JDC, Matt Wilko, Nat G.
41	Pattern asincrono basato su attività	Stefano d'Antonio
42	ricorsione	Robert Columbia
43	Riflessione	Axarydax, Matt, Sam Axe, void
44	schieramento	BunkerMentality, djv, Drarig29, Luke Sheppard, Mark Hurd, MatVAD, Robert Columbia, Ryan Thomas, Sam Axe, Sehnsucht, Steven Doggart, TuxCopter, vbnet3d, VortixDev, zyabin101
45	Server FTP	Misaz
46	Test unitario in VB.NET	wbadry
47	threading	BiscuitBaker, Stefano d'Antonio, Visual Vincent
48	Utilizzando axWindowsMediaPlayer in VB.Net	Berken Usar
49	Utilizzando BackgroundWorker	MatVAD
50	Utilizzando la dichiarazione	VV5198722
51	WinForms SpellCheckBox	Nathan