



EBook Gratis

APRENDIZAJE

Visual Basic .NET

Language

Free unaffiliated eBook created from
Stack Overflow contributors.

[#vb.net](https://vb.net)

Tabla de contenido

| | |
|---|-----------|
| Acerca de..... | 1 |
| Capítulo 1: Comenzando con el lenguaje Visual Basic .NET..... | 2 |
| Observaciones..... | 2 |
| Versiones..... | 2 |
| Examples..... | 2 |
| Hola Mundo..... | 2 |
| Hola mundo en un cuadro de texto al hacer clic en un botón..... | 3 |
| Región..... | 4 |
| Creando una calculadora simple para familiarizarse con la interfaz y el código..... | 5 |
| Capítulo 2: Acceso a los datos..... | 14 |
| Examples..... | 14 |
| Leer campo de la base de datos..... | 14 |
| Función simple para leer de la base de datos y regresar como DataTable..... | 15 |
| Obtener datos escalares..... | 16 |
| Capítulo 3: Aleatorio..... | 17 |
| Introducción..... | 17 |
| Observaciones..... | 17 |
| Examples..... | 17 |
| Declarando una instancia..... | 17 |
| Generar un número aleatorio a partir de una instancia de Random..... | 18 |
| Capítulo 4: Bucle..... | 20 |
| Examples..... | 20 |
| Para ... siguiente..... | 20 |
| Para cada ... Siguiente bucle para recorrer la colección de elementos..... | 21 |
| Mientras bucle para iterar mientras alguna condición es verdadera..... | 21 |
| Hacer ... bucle..... | 22 |
| Cortocircuito..... | 23 |
| Bucle anidado..... | 25 |
| Capítulo 5: Características de Visual Basic 14.0..... | 26 |
| Introducción..... | 26 |

| | |
|---|-----------|
| Examples..... | 26 |
| Operador condicional nulo..... | 26 |
| Operador NameOf..... | 27 |
| Interpolación de cuerdas..... | 27 |
| Propiedades automáticas de solo lectura..... | 28 |
| Módulos parciales e interfaces..... | 29 |
| Literales multilínea de cuerda..... | 29 |
| ## Mejoras directivas regionales..... | 30 |
| Comentarios después de la continuación de la línea implícita..... | 30 |
| Manejo de excepciones..... | 31 |
| Capítulo 6: Compresión de archivos / carpetas..... | 33 |
| Examples..... | 33 |
| Creando un archivo zip desde el directorio..... | 33 |
| Extraer el archivo zip al directorio..... | 33 |
| Crear archivo zip dinámicamente..... | 33 |
| Agregando compresión de archivos a su proyecto..... | 33 |
| Capítulo 7: Condiciones..... | 35 |
| Examples..... | 35 |
| SI ... entonces ... si no..... | 35 |
| Si operador..... | 35 |
| Capítulo 8: Consola..... | 37 |
| Examples..... | 37 |
| Console.ReadLine ()..... | 37 |
| Console.WriteLine ()..... | 37 |
| Console.Write ()..... | 37 |
| Console.Read ()..... | 38 |
| Console.ReadKey ()..... | 38 |
| Prototipo de línea de comandos..... | 38 |
| Capítulo 9: Declarando variables..... | 40 |
| Sintaxis..... | 40 |
| Examples..... | 40 |
| Declarar y asignar una variable usando un tipo primitivo..... | 40 |

| | |
|--|-----------|
| Niveles de declaración - Variables locales y miembros..... | 43 |
| Ejemplo de modificadores de acceso..... | 44 |
| Capítulo 10: Depurando tu aplicación..... | 48 |
| Introducción..... | 48 |
| Examples..... | 48 |
| Depurar en la consola..... | 48 |
| Sangrando su salida de depuración..... | 48 |
| Depurar en un archivo de texto..... | 49 |
| Capítulo 11: Enhebrado..... | 51 |
| Examples..... | 51 |
| Realizar llamadas a prueba de subprocessos usando Control.Invoke ()..... | 51 |
| Realizar llamadas a prueba de subprocessos utilizando Async / Await..... | 51 |
| Capítulo 12: Enlace de datos WPF XAML..... | 53 |
| Introducción..... | 53 |
| Examples..... | 53 |
| Enlace de una cadena en el ViewModel a un TextBox en la Vista..... | 53 |
| Capítulo 13: Enumerar..... | 55 |
| Examples..... | 55 |
| Definición de enumeración..... | 55 |
| Inicialización de miembros..... | 55 |
| El atributo Flags..... | 55 |
| HasFlag ()..... | 56 |
| Análisis de cuerdas..... | 56 |
| GetNames ()..... | 57 |
| GetValues ()..... | 57 |
| Encadenar()..... | 58 |
| Determine si un Enum tiene FlagsAttribute especificado o no..... | 58 |
| Para cada bandera (iteración de bandera)..... | 59 |
| Determine la cantidad de banderas en una combinación de bandera..... | 60 |
| Encuentra el valor más cercano en un Enum..... | 60 |
| Capítulo 14: Excepcion de referencia nula..... | 62 |
| Observaciones..... | 62 |

| | |
|---|-----------|
| Examples..... | 62 |
| Variable no inicializada..... | 62 |
| Retorno vacio..... | 62 |
| Capítulo 15: Fecha..... | 64 |
| Examples..... | 64 |
| Convertir (analizar) una cadena en una fecha..... | 64 |
| Convertir una fecha en una cadena..... | 64 |
| Capítulo 16: Formación..... | 65 |
| Observaciones..... | 65 |
| Examples..... | 65 |
| Definición de matriz..... | 65 |
| De base cero..... | 65 |
| Declarar una matriz de dimensión única y establecer valores de elementos de matriz..... | 66 |
| Inicialización de matriz..... | 66 |
| Inicialización de matriz multidimensional..... | 66 |
| Inicialización de matriz irregular..... | 66 |
| Variables de matriz nula..... | 67 |
| Haciendo referencia a la misma matriz de dos variables..... | 67 |
| Límites inferiores distintos de cero..... | 67 |
| Capítulo 17: Funciones..... | 69 |
| Introducción..... | 69 |
| Examples..... | 69 |
| Definiendo una función..... | 69 |
| Definiendo una función # 2..... | 69 |
| Capítulo 18: GDI +..... | 70 |
| Examples..... | 70 |
| Crear objeto gráfico..... | 70 |
| Dibujar formas..... | 70 |
| Formas de relleno..... | 71 |
| Texto..... | 72 |
| Capítulo 19: Genéricos..... | 73 |
| Examples..... | 73 |

| | |
|---|-----------|
| Crear una clase genérica..... | 73 |
| Instancia de una clase genérica..... | 73 |
| Definir una clase 'genérica'..... | 73 |
| Usa una clase genérica..... | 73 |
| Limite los tipos posibles dados..... | 74 |
| Crear una nueva instancia del tipo dado..... | 74 |
| Capítulo 20: Google Maps en un formulario de Windows..... | 76 |
| Examples..... | 76 |
| Cómo utilizar un mapa de Google en un formulario de Windows..... | 76 |
| Capítulo 21: Introducción a la sintaxis..... | 87 |
| Examples..... | 87 |
| Comentarios..... | 87 |
| Ayudante Intellisense..... | 87 |
| Declarar una variable..... | 87 |
| Modificadores..... | 88 |
| Escribiendo una función..... | 89 |
| Inicializadores de objetos..... | 90 |
| Inicializador de colección..... | 91 |
| Capítulo 22: Las clases..... | 94 |
| Introducción..... | 94 |
| Examples..... | 94 |
| Creando clases..... | 94 |
| Clases abstractas..... | 94 |
| Capítulo 23: Leyendo el archivo de texto comprimido sobre la marcha..... | 96 |
| Examples..... | 96 |
| Leyendo archivo de texto .gz línea tras línea..... | 96 |
| Capítulo 24: LINQ..... | 97 |
| Introducción..... | 97 |
| Examples..... | 97 |
| Proyección..... | 97 |
| Seleccionando de matriz con condición simple..... | 97 |
| Mapeo de matriz por cláusula de selección..... | 97 |

| | |
|---|------------|
| Orden de salida..... | 98 |
| Generando diccionario desde IEnumerable..... | 98 |
| Obtención de valores distintos (utilizando el método Distinct)..... | 98 |
| Capítulo 25: Liza..... | 100 |
| Sintaxis..... | 100 |
| Examples..... | 100 |
| Crear una lista..... | 100 |
| Añadir elementos a una lista..... | 101 |
| Eliminar elementos de una lista..... | 101 |
| Recuperar elementos de una lista..... | 102 |
| Bucle a través de los elementos en la lista..... | 102 |
| Compruebe si el artículo existe en una lista..... | 103 |
| Capítulo 26: Los diccionarios..... | 104 |
| Introducción..... | 104 |
| Examples..... | 104 |
| Recorrer un diccionario e imprimir todas las entradas..... | 104 |
| Crear un diccionario lleno de valores..... | 104 |
| Obtener un valor de diccionario..... | 104 |
| Buscando clave ya en el diccionario - reducción de datos..... | 105 |
| Capítulo 27: Los operadores..... | 106 |
| Observaciones..... | 106 |
| Examples..... | 106 |
| Comparación..... | 106 |
| Asignación..... | 107 |
| Mates..... | 107 |
| Ampliación y estrechamiento..... | 109 |
| Sobrecarga del operador..... | 109 |
| Bitwise..... | 109 |
| Concatenación de cuerdas..... | 109 |
| Capítulo 28: Manejo de archivos..... | 111 |
| Sintaxis..... | 111 |
| Examples..... | 111 |

| | |
|---|------------|
| Escribir datos en un archivo..... | 111 |
| Leer todos los contenidos de un archivo..... | 111 |
| Escribir líneas individualmente en un archivo de texto usando StreamWriter..... | 111 |
| Capítulo 29: Manejo de errores..... | 113 |
| Examples..... | 113 |
| Intenta ... Atrapa ... Finalmente Declaración..... | 113 |
| Creando excepciones y lanzamientos personalizados..... | 113 |
| Intenta atrapar en la operación de base de datos..... | 114 |
| La excepción irrecoverable..... | 114 |
| Excepciones críticas..... | 115 |
| Capítulo 30: Manejo de la conexión..... | 116 |
| Examples..... | 116 |
| Propiedad de conexión pública..... | 116 |
| Capítulo 31: Metodos de extension..... | 117 |
| Observaciones..... | 117 |
| Examples..... | 117 |
| Creando un método de extensión..... | 117 |
| Haciendo el lenguaje más funcional con métodos de extensión..... | 118 |
| Numéricos de relleno..... | 118 |
| Obtención de la versión de montaje de nombre fuerte..... | 119 |
| Capítulo 32: Multihilo..... | 120 |
| Examples..... | 120 |
| Subprocesamiento múltiple utilizando Thread Class..... | 120 |
| Capítulo 33: Objetos desechables..... | 122 |
| Examples..... | 122 |
| Concepto básico de IDisposable..... | 122 |
| Declarando más objetos en uno usando..... | 123 |
| Capítulo 34: OOP Keywords..... | 124 |
| Examples..... | 124 |
| Definiendo una clase..... | 124 |
| Modificadores de herencia (en clases)..... | 124 |
| Hereda..... | 124 |

| | |
|--|------------|
| No Heredable | 124 |
| Debe Heredarse | 125 |
| Modificadores de herencia (en propiedades y métodos) | 125 |
| Anulable | 125 |
| Anulaciones | 125 |
| NotOverridable | 126 |
| MustOverride | 126 |
| MyBase | 127 |
| Yo vs myclass | 128 |
| Sobrecarga | 128 |
| Oscuridad | 129 |
| Interfaces | 130 |
| Capítulo 35: Opción Estricta | 132 |
| Sintaxis | 132 |
| Observaciones | 132 |
| Examples | 132 |
| ¿Por qué usarlo? | 132 |
| Cómo encenderlo | 133 |
| Capítulo 36: Opción explícita | 135 |
| Observaciones | 135 |
| Examples | 135 |
| ¿Qué es? | 135 |
| ¿Cómo encenderlo? | 135 |
| Capítulo 37: Opcion Inferir | 137 |
| Examples | 137 |
| ¿Qué es? | 137 |
| Cómo habilitarlo / deshabilitarlo | 137 |
| Cuándo usar inferencia de tipo | 138 |
| Capítulo 38: Operadores de cortocircuito (y también - orElse) | 140 |
| Sintaxis | 140 |
| Parámetros | 140 |

| | |
|---|------------|
| Observaciones..... | 140 |
| Examples..... | 140 |
| Y también uso..... | 140 |
| Uso del orse..... | 141 |
| Evitando NullReferenceException..... | 141 |
| Si no..... | 141 |
| Y también..... | 142 |
| Capítulo 39: Palabras clave ByVal y ByRef..... | 143 |
| Examples..... | 143 |
| Palabra clave byVal..... | 143 |
| ByRef palabra clave..... | 143 |
| Capítulo 40: Patrón asíncrono basado en tareas..... | 145 |
| Examples..... | 145 |
| Uso básico de Async / Await..... | 145 |
| Usando TAP con LINQ..... | 145 |
| Capítulo 41: Pruebas Unitarias en VB.NET..... | 147 |
| Observaciones..... | 147 |
| Examples..... | 147 |
| Pruebas unitarias para el cálculo de impuestos..... | 147 |
| Clase de empleado de prueba asignada y propiedades derivadas..... | 149 |
| Capítulo 42: Recursion..... | 153 |
| Examples..... | 153 |
| Calcular el número de Fibonacci..... | 153 |
| Capítulo 43: Reflexión..... | 154 |
| Examples..... | 154 |
| Recuperar propiedades para una instancia de una clase..... | 154 |
| Obtener los miembros de un tipo..... | 154 |
| Consigue un método e invocalo..... | 154 |
| Crear una instancia de un tipo genérico..... | 155 |
| Capítulo 44: Servidor FTP..... | 156 |
| Sintaxis..... | 156 |
| Examples..... | 156 |

| | |
|---|------------|
| Descargar el archivo desde el servidor FTP | 156 |
| Descargue el archivo desde el servidor FTP cuando sea necesario iniciar sesión..... | 156 |
| Subir archivo al servidor FTP..... | 156 |
| Cargue el archivo al servidor FTP cuando se requiera iniciar sesión..... | 156 |
| Capítulo 45: Tipo de conversión | 158 |
| Sintaxis..... | 158 |
| Parámetros..... | 158 |
| Examples..... | 158 |
| Convertir el texto del cuadro de texto en un entero..... | 158 |
| Capítulo 46: Trabajador de fondo | 160 |
| Examples..... | 160 |
| Usando BackgroundWorker..... | 160 |
| Accediendo a los componentes GUI en BackgroundWorker..... | 161 |
| Capítulo 47: Trabajar con formularios Windows Forms | 162 |
| Examples..... | 162 |
| Usando la instancia de formulario por defecto..... | 162 |
| Pasando datos de una forma a otra..... | 162 |
| Capítulo 48: Usando axWindowsMediaPlayer en VB.Net..... | 164 |
| Introducción..... | 164 |
| Examples..... | 164 |
| Añadiendo el axWindowsMediaPlayer..... | 164 |
| Reproducir un archivo multimedia..... | 166 |
| Capítulo 49: Usando BackgroundWorker | 168 |
| Examples..... | 168 |
| Implementación básica de la clase de trabajador de fondo..... | 168 |
| Capítulo 50: Utilizando la declaración | 169 |
| Sintaxis..... | 169 |
| Examples..... | 169 |
| Ver ejemplos en Objetos desechables..... | 169 |
| Capítulo 51: WinForms SpellCheckBox..... | 170 |
| Introducción..... | 170 |

| | |
|------------------------------|------------|
| Examples..... | 170 |
| ElementHost WPF TextBox..... | 170 |
| Credits..... | 174 |

Acerca de

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capítulo 1: Comenzando con el lenguaje Visual Basic .NET

Observaciones

Visual Basic .NET es el sucesor oficial del lenguaje de programación Visual Basic original de Microsoft. Visual Basic [.NET] parece tener similitudes con Python con la falta de punto y coma y corchetes, pero comparte con C ++ la estructura básica de las funciones. Las llaves no están en VB .NET, pero en cambio se reemplazan con frases como `End If` , `Next` y `End Sub` .

Versiones

| Versión VB.NET | Versión de Visual Studio | Versión de .NET Framework | Fecha de lanzamiento |
|----------------|--------------------------|---------------------------|----------------------|
| 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

Hola Mundo

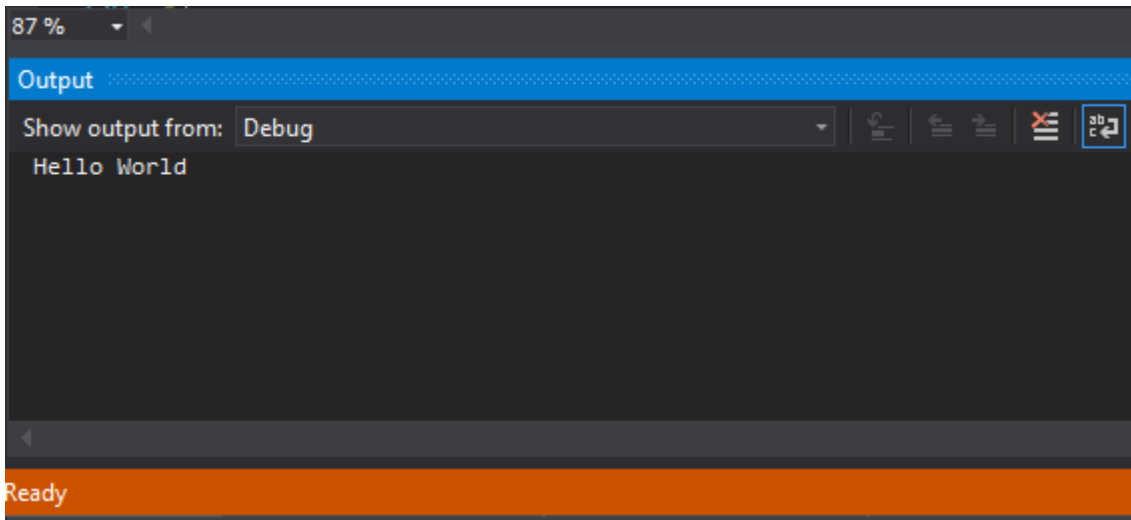
Primero, instale una versión de [Microsoft Visual Studio](#) , incluida la edición gratuita de la Comunidad. Luego, cree un proyecto de aplicación de consola de Visual Basic de tipo *Aplicación de consola* y el siguiente código imprimirá la cadena 'Hello World' en la consola:

```
Module Module1
    Sub Main()
```

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

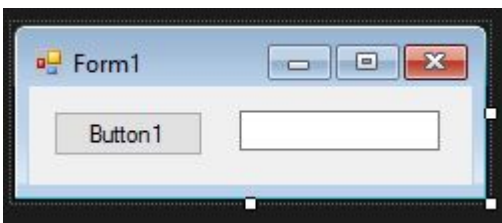
End Module
```

Luego, guarde y presione **F5** en el teclado (o vaya al menú *Depurar*, luego haga clic en *Ejecutar sin depurar* o *Ejecutar*) para compilar y ejecutar el programa. 'Hello World' debería aparecer en la ventana de la consola.



Hola mundo en un cuadro de texto al hacer clic en un botón

Arrastre 1 cuadro de texto y 1 botón



Haga doble clic en button1 y será transferido al `Button1_Click` event

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

    End Sub
End Class
```

Escriba el nombre del objeto que desea apuntar, en nuestro caso es el `textbox1.Text` es la propiedad que queremos usar si queremos poner un texto en él.

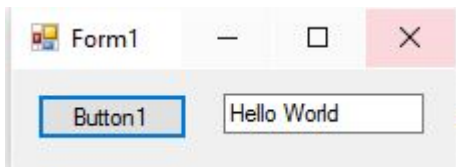
Property `Textbox.Text`, gets or sets the current text in the `TextBox`. Ahora, tenemos `Textbox1.Text`

Necesitamos establecer el valor de ese `Textbox1.Text` para que usemos el signo `=`. El valor que queremos poner en el `Textbox1.Text` es `Hello World`. En general, este es el código total para poner un valor de `Hello World` en `Textbox1.Text`

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

Agregando ese código al `clicked` event de `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
```



Región

En aras de la legibilidad, que será útil para los principiantes al leer el código VB, así como para que los desarrolladores a tiempo completo mantengan el código, podemos usar "Región" para establecer una región del mismo conjunto de eventos, funciones o variables:

```
#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
```

Este bloque de región podría colapsarse para obtener alguna ayuda visual cuando la fila de códigos pasa a 1000+. También es guardar sus esfuerzos de desplazamiento.


```

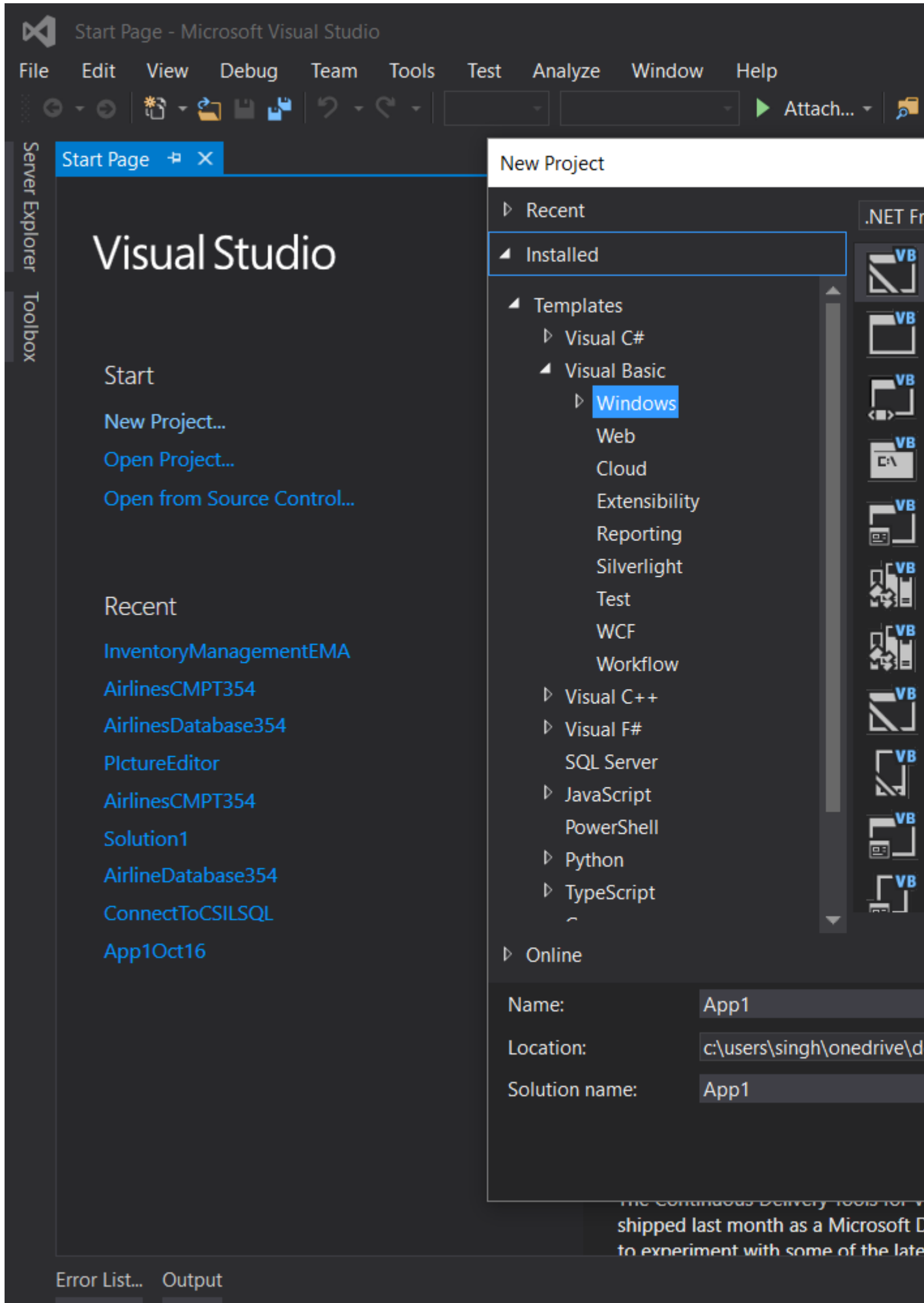
1 Imports System.Data
2 Imports System.Data.SqlClient
3 Imports ClassFunction
4 Imports CrystalDecisions.CrystalReports.Engine
5 Imports CrystalDecisions.Shared
6 Imports CrystalDecisions.ReportSource
7 Imports CrystalDecisions.Reporting
8 Partial Class transaction_trnBPB_PCH_JPS_Tekhnik
9     Inherits System.Web.UI.Page
10 Variables
32
33 #Region "Functions"
34 Private Function GenerateOrderNo ...
52 Sub CreateTableBLDTL ...
85 Protected Function getDateToPeriodAcctg ...
96 #End Region
97
98 Procedures
417
418 Event
1407 End Class

```

Probado en VS 2005, 2008 2010, 2015 y 2017.

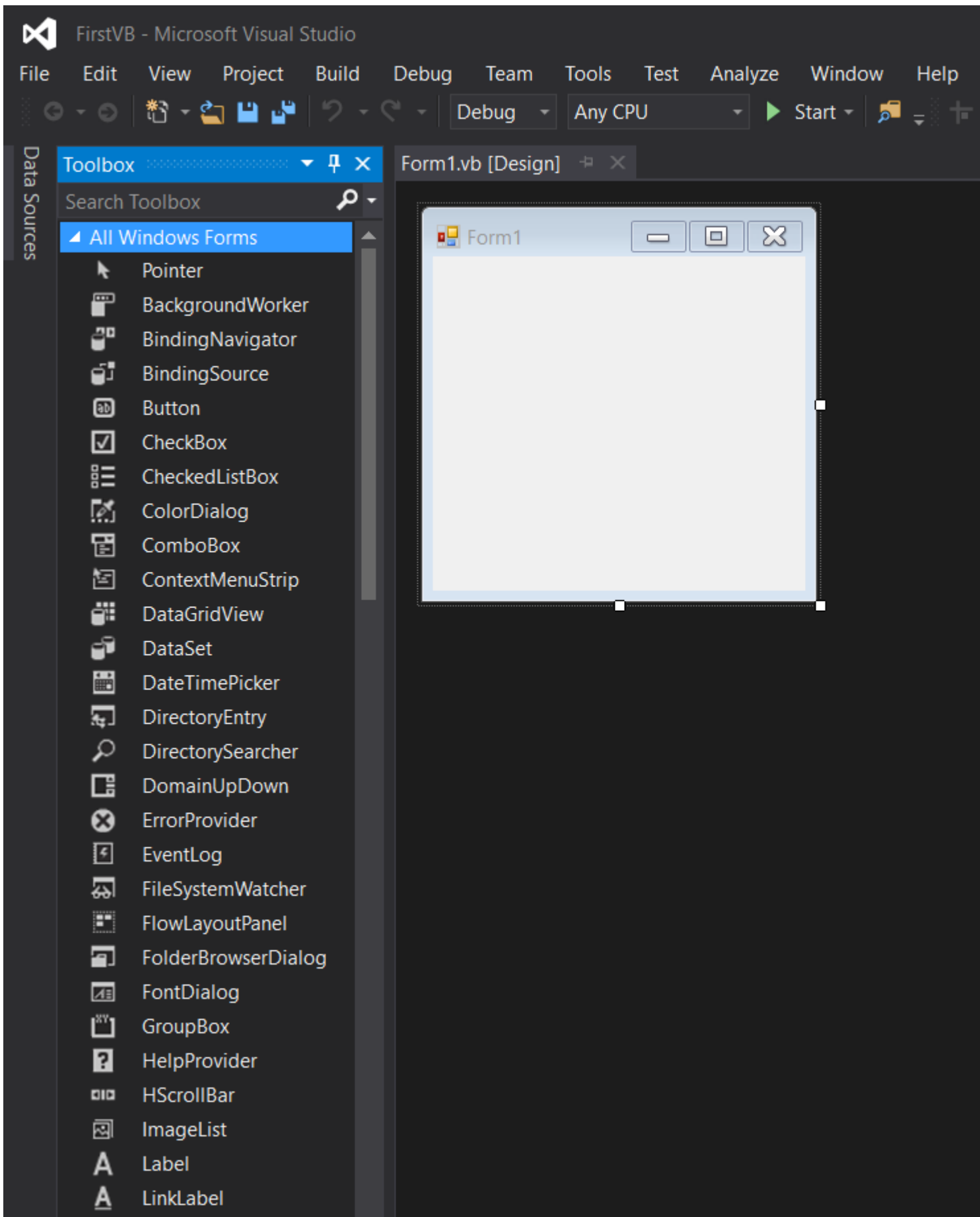
Creando una calculadora simple para familiarizarse con la interfaz y el código.

1. Una vez que haya instalado Visual Studio desde <https://www.visualstudio.com/downloads/> , comience un nuevo proyecto.



2.

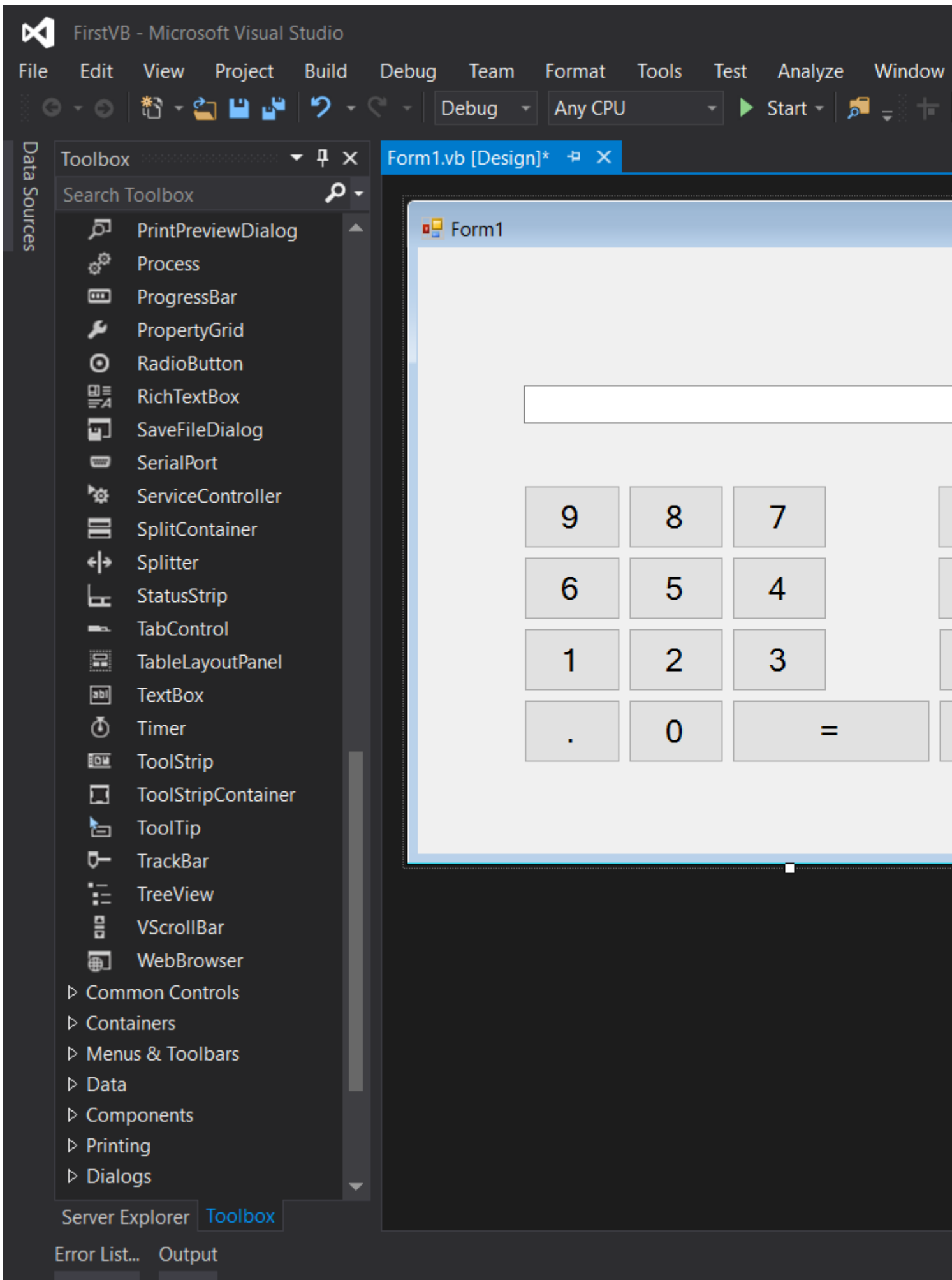
3. Seleccione 'Aplicación de Windows Forms' en la pestaña de Visual Basic. Puede cambiar el nombre aquí si lo necesita.
4. Una vez que haga clic en 'Aceptar', verá esta ventana:



5. Haga clic en la pestaña 'Caja de herramientas' a la izquierda. La barra de herramientas tiene la opción 'Ocultar automáticamente' habilitada de forma predeterminada. Para deshabilitar

esta opción, haga clic en el pequeño símbolo entre el símbolo de 'flecha hacia abajo' y el símbolo de 'x', en la esquina superior derecha de la ventana de la Caja de herramientas.

6. Familiarícese con las herramientas provistas en la caja. He hecho una interfaz de calculadora mediante el uso de botones y un cuadro de texto.



(está en el lado derecho del editor). Puede cambiar la propiedad de *texto* de un botón y el cuadro de texto para cambiarles el nombre. La propiedad de *fuentes* se puede utilizar para alterar la fuente de los controles.

8. Para escribir la acción específica para un evento (por ejemplo, hacer clic en un botón), haga doble clic en el control. Se abrirá la ventana de código.

```
1 Public Class Form1
2     Private Sub Button1_Click(sender As Object, e As EventArgs) Handl
3         TextBox1.Text = TextBox1.Text + "1"
4     End Sub
5
6     Private Sub Button2_Click(sender As Object, e As EventArgs) Handl
7         TextBox1.Text = TextBox1.Text + "2"
8     End Sub
9
10    Private Sub Button3_Click(sender As Object, e As EventArgs) Handl
11        TextBox1.Text = TextBox1.Text + "3"
12    End Sub
13
14    'Declaring variables as integers
15    Dim firstValue, secondValue As Int32
16    'Result
17    Private Sub Button13_Click(sender As Object, e As EventArgs) Handl
18        secondValue = TextBox1.Text
19        TextBox1.Text = firstValue + secondValue
20    End Sub
21
22    'Clear the textbox
23    Private Sub Button10_Click(sender As Object, e As EventArgs) Handl
24        TextBox1.Clear()
25    End Sub
26    'Addition
27    Private Sub Button14_Click(sender As Object, e As EventArgs) Handl
28        firstValue = Convert.ToInt32(TextBox1.Text)
29        TextBox1.Clear()
30    End Sub
31 End Class
32
```

9. VB.Net es un lenguaje poderoso diseñado para un rápido desarrollo. La alta encapsulación y la abstracción tienen un costo por ello. No es necesario que agregue un *punto y coma* para indicar el final de una declaración, no hay corchetes y, la mayoría de las veces, corrige automáticamente el caso de los alfabetos.

10. El código provisto en la imagen debe ser fácil de entender. *Dim* es la palabra clave utilizada para inicializar una variable y la *nueva* asigna memoria. Todo lo que escriba en el cuadro de texto es de tipo *cadena* de forma predeterminada. Se requiere la conversión para usar el valor como un tipo diferente.

¡Disfruta de tu primera creación en VB.Net!

Lea **Comenzando con el lenguaje Visual Basic .NET** en línea: <https://riptutorial.com/es/vb-net/topic/352/comenzando-con-el-lenguaje-visual-basic--net>

Capítulo 2: Acceso a los datos

Examples

Leer campo de la base de datos

```
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
```

Usar la función anterior es simplemente:

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

Función simple para leer de la base de datos y regresar como DataTable

Esta función simple ejecutará el comando Select SQL especificado y devolverá el resultado como conjunto de datos.

```
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
```

Ahora puede ejecutar la función anterior de los códigos siguientes

```
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
```

El ejemplo anterior espera que su instancia de SQL Express "SQLEXPRESS" esté actualmente

instalada en "MYDEVPC" y que su base de datos "MyDatabase" contenga las tablas "Clientes", "Proveedores" y "Empleados" y la contraseña de usuario "sa" sea "pwd22". Cambie estos valores según su configuración para obtener los resultados deseados.

Obtener datos escalares

Esta función simple se puede usar para obtener valor de exactamente un campo, un resultado de consulta de registro

```
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
```

Cómo usarlo:

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

La variable 'nombre de usuario' se llenaría con el valor del campo nombre de usuario como resultado de esa consulta.

Lea Acceso a los datos en línea: <https://riptutorial.com/es/vb-net/topic/3113/acceso-a-los-datos>

Capítulo 3: Aleatorio

Introducción

La clase aleatoria se utiliza para generar enteros pseudo-aleatorios no negativos que no son realmente aleatorios, pero que son para propósitos generales lo suficientemente cercanos.

La secuencia se calcula utilizando un número inicial (llamado la **semilla**) En versiones anteriores de .net, este número de semilla era el mismo cada vez que se ejecutaba una aplicación. Entonces, lo que sucedería sería que obtendría la misma secuencia de números pseudoaleatorios cada vez que se ejecutara la aplicación. Ahora, la semilla se basa en el momento en que se declara el objeto.

Observaciones

Finalmente, una nota sobre la aleatorización. Como se mencionó anteriormente, cuando declara una instancia de `Random` sin ningún parámetro, el constructor usará la hora actual como parte del cálculo para crear el número inicial. Normalmente esto está bien.

Sin embargo. Si vuelve a declarar las nuevas instancias en un espacio de tiempo muy corto, cada vez que se calcula el número inicial, el tiempo podría ser el mismo. Considere este código.

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

Debido a que las computadoras son muy rápidas en estos días, este código tardará una fracción de segundo en ejecutarse y en varias iteraciones secuenciales del bucle, la hora del sistema no habrá cambiado. Por lo tanto, el número inicial no cambiará y el número aleatorio será el mismo. Si desea generar muchos números aleatorios, declare la instancia de aleatorio fuera del bucle en este simple ejemplo.

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

La regla básica es no volver a crear una instancia del generador de números aleatorios en períodos cortos de tiempo.

Examples

Declarando una instancia

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

Esto declara una instancia de la clase aleatoria llamada `rng`. En este caso, la hora actual en el punto donde se crea el objeto se utiliza para calcular la semilla. Este es el uso más común, pero tiene sus propios problemas, como veremos más adelante en las observaciones.

En lugar de permitir que el programa utilice la hora actual como parte del cálculo para el número inicial inicial, puede especificar el número inicial inicial. Esto puede ser cualquier entero de 32 bits literal, constante o variable. Vea a continuación los ejemplos. Hacer esto significa que su instancia generará la misma secuencia de números pseudoaleatorios, lo que puede ser útil en ciertas situaciones.

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

o

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

donde `x` se ha declarado en otra parte de su programa como una constante o variable Integer.

Generar un número aleatorio a partir de una instancia de Random

El siguiente ejemplo declara una nueva instancia de la clase `Random` y luego usa el método `.Next` para generar el siguiente número en la secuencia de números pseudoaleatorios.

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

La última línea de arriba generará el siguiente número pseudoaleatorio y lo asignará a `x`. Este número estará en el rango de 0 a 2147483647. Sin embargo, también puede especificar el rango de números que se generarán como se muestra en el siguiente ejemplo.

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

Sin embargo, tenga en cuenta que al usar estos parámetros, el rango de números estará entre 15 o más y 199 o menos.

También puede generar números de punto flotante del tipo `Double` utilizando `.NextDouble` por ejemplo,

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

Sin embargo, no puede especificar un rango para esto. Siempre estará en el rango de 0.0 a menos de 1.0.

Lea Aleatorio en línea: <https://riptutorial.com/es/vb-net/topic/10128/aleatorio>

Capítulo 4: Bucle

Examples

Para ... siguiente

For ... El `Next` bucle se usa para repetir la misma acción un número finito de veces. Las declaraciones dentro del siguiente bucle se ejecutarán 11 veces. La primera vez, `i` tendrá el valor 0, la segunda vez que tendrá el valor 1, la última vez que tendrá el valor 10.

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

Se puede utilizar cualquier expresión entera para parametrizar el bucle. Se permite, pero no se requiere, que la variable de control (en este caso `i`) también se indique después de la `Next`. Se permite que la variable de control se declare por adelantado, en lugar de dentro de la instrucción

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
```

La posibilidad de definir los enteros de Inicio y Fin permite crear bucles que hacen referencia directamente a otros objetos, como:

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

Esto luego pasaría por cada fila en `DataGridView1` y realizaría la acción de escribir el valor de la Columna 1 en la Consola. (El -1 se debe a que la primera fila de las filas contadas sería 1, no 0)

También es posible definir cómo debe incrementarse la variable de control.

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

Esto produce:

1 3 5 7 9

También es posible disminuir la variable de control (cuenta atrás).

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

Esto produce:

10 9 8 7 6 5 4 3 2 1

No debe intentar usar (leer o actualizar) la variable de control fuera del bucle.

Para cada ... Siguiente bucle para recorrer la colección de elementos

Puede usar un bucle `For Each...Next` para iterar a través de cualquier tipo de `IEnumerable`. Esto incluye matrices, listas y cualquier otra cosa que pueda ser de tipo `IEnumerable` o que devuelva un `IEnumerable`.

Un ejemplo de bucle a través de la propiedad `Rows` de un `DataTable` se vería así:

```
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
```

Es importante tener en cuenta que la colección no debe modificarse en un bucle `For Each`. Si lo hace, provocará una `System.InvalidOperationException` con el mensaje:

La colección fue modificada; La operación de enumeración no puede ejecutarse.

Mientras bucle para iterar mientras alguna condición es verdadera

Un bucle `While` comienza evaluando una condición. Si es cierto, se ejecuta el cuerpo del bucle. Después de que se ejecuta el cuerpo del bucle, la condición `While` se evalúa nuevamente para determinar si se debe volver a ejecutar el cuerpo.

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

    iteration += 1
End While
```

Esto produce:

1 2 3 4 5 6 7 8 9 10

Advertencia: un bucle `While` puede llevar a un *bucle infinito*. Considere qué pasaría si se eliminara la línea de código que incrementa la `iteration`. En tal caso, la condición nunca sería verdadera y el bucle continuaría indefinidamente.

Hacer ... bucle

Use `Do...Loop` para repetir un bloque de declaraciones `While` o `Until` una condición sea verdadera, verificando la condición al principio o al final 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` se puede usar para saltar a la siguiente iteración del bucle:

```
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

Puede terminar el ciclo con `Exit Do`: tenga en cuenta que en este ejemplo, la falta de cualquier condición causaría un ciclo infinito:

```

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

```

0 1 2 3 4 5 6 7 8 9

Cortocircuito

Cualquier bucle se puede terminar o continuar antes en cualquier momento mediante el uso de las instrucciones `Exit` o `Continue`.

Saliendo

Puedes detener cualquier bucle saliendo antes. Para hacer esto, puede usar la palabra clave `Exit` junto con el nombre del bucle.

| Lazo | Declaración de salida |
|----------------|-------------------------|
| por | <code>Exit For</code> |
| Para cada | <code>Exit For</code> |
| Hacer mientras | <code>Exit Do</code> |
| Mientras | <code>Exit While</code> |

Salir temprano de un bucle es una excelente manera de mejorar el rendimiento, solo haciendo un bucle el número necesario de veces para satisfacer las necesidades de la aplicación. A continuación se muestra un ejemplo donde el bucle saldrá una vez que encuentre el número 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)

```

Continuo

Además de salir temprano, también puede decidir que necesita pasar a la siguiente iteración de bucle. Esto se hace fácilmente usando la instrucción `Continue`. Al igual que `Exit`, se procede con

el nombre del bucle.

| Lazo | Continuar declaración |
|----------------|-----------------------|
| por | Continue For |
| Para cada | Continue For |
| Hacer mientras | Continue Do |
| Mientras | Continue While |

Este es un ejemplo de cómo evitar que se agreguen números pares a la suma.

```
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
```

Consejos de uso

Hay dos técnicas alternativas que se pueden usar en lugar de usar `Exit` o `Continue`.

Puede declarar una nueva variable booleana, inicializándola a un valor y configurándola condicionalmente al otro valor dentro del bucle; luego utiliza una declaración condicional (por ejemplo, `If`) basada en esa variable para evitar la ejecución de las declaraciones dentro del bucle en las iteraciones posteriores.

```
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 de las objeciones a esta técnica es que puede ser ineficiente. Por ejemplo, si en el ejemplo anterior `N` es 1000000 y el primer elemento de la matriz `A` es igual a `SoughtValue`, el bucle repetirá 999999 veces más sin hacer nada útil. Sin embargo, esta técnica puede tener la ventaja de una mayor claridad en algunos casos.

Puede utilizar la instrucción `GoTo` para saltar fuera del bucle. Tenga en cuenta que no puede utilizar `GoTo` para saltar a un bucle.

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

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

Esta técnica a veces puede ser la mejor forma de saltar fuera del bucle y evitar una o más declaraciones que se ejecutan justo después del final natural del bucle.

Debe considerar todas las alternativas y utilizar la que mejor se ajuste a sus requisitos, considerando la eficiencia, la velocidad de escritura del código y la legibilidad (por lo tanto, la capacidad de mantenimiento).

No se desanime con `GoTo` en aquellas ocasiones en que sea la mejor alternativa.

Bucle anidado

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 estructura de un bucle anidado para el siguiente es:

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

    For counter2=startNumber to endNumber (Step increment)

        One or more VB statements

    Next counter2

Next counter1
```

Ejemplo:

```
For firstCounter = 1 to 5

    Print "First Loop of " + firstCounter

For secondCounter= 1 to 4

    Print "Second Loop of " + secondCounter

Next secondCounter

Next firstCounter
```

Lea Bucle en línea: <https://riptutorial.com/es/vb-net/topic/1639/bucle>

Capítulo 5: Características de Visual Basic

14.0

Introducción

Visual Basic 14 es la versión de Visual Basic que se envió como parte de Visual Studio 2015.

Esta versión fue reescrita desde cero en aproximadamente 1.3 millones de líneas de VB. Se agregaron muchas características para eliminar las irritaciones comunes y hacer que los patrones de codificación comunes sean más limpios.

El número de versión de Visual Basic fue directamente del 12 al 14, omitiendo 13. Esto se hizo para mantener VB en línea con la numeración de versión de Visual Studio.

Examples

Operador condicional nulo

Para evitar verificaciones nulas, el `?.` El operador ha sido introducido en el idioma.

La vieja sintaxis verbosa:

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

Puede ahora ser reemplazado por el conciso:

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

El `?.` El operador es particularmente poderoso cuando tienes una cadena de propiedades. Considera lo siguiente:

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

Normalmente tendrías que escribir algo como esto:

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

Pero con el `?.` operador esto puede ser reemplazado con solo:

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

Operador NameOf

El operador `NameOf` resuelve espacios de nombres, tipos, variables y nombres de miembros en tiempo de compilación y los reemplaza con el equivalente de cadena.

Uno de los casos de uso:

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

La sintaxis anterior expondrá el riesgo de cambiar el nombre de la variable y dejar la cadena codificada en el valor incorrecto.

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

Con `NameOf`, cambiar el nombre de la variable solo generará un error de compilación. Esto también permitirá que la herramienta de cambio de nombre cambie el nombre de ambos con un solo esfuerzo.

El operador `NameOf` solo usa el último componente de la referencia entre paréntesis. Esto es importante cuando se maneja algo así como espacios de nombres en el operador `NameOf`.

```
Imports System

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

El operador también utiliza el nombre de la referencia que se ingresa sin resolver ningún cambio de nombre de importación. Por ejemplo:

```
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
```

Interpolación de cuerdas

Esta nueva característica hace que la concatenación de cadenas sea más legible. Esta sintaxis se

compilará a su llamada `String.Format` equivalente.

Sin interpolación de cuerdas:

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

Con interpolación de cuerdas:

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

Las dos líneas son equivalentes y ambas se compilan en una llamada a `String.Format` .

Al igual que en `String.Format` , los corchetes pueden contener cualquier expresión (llamada a un método, propiedad, un operador de unión nula, etc.).

La interpolación de cadenas es el método preferido sobre `String.Format` porque evita que `String.Format` algunos errores de tiempo de ejecución. Considere la siguiente línea `String.Format` :

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

Esto se compilará, pero causará un error de tiempo de ejecución ya que el compilador no verifica que la cantidad de argumentos coincida con los marcadores de posición.

Propiedades automáticas de solo lectura

Las propiedades de solo lectura siempre fueron posibles en VB.NET en este 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 nueva versión de Visual Basic permite una mano corta para la declaración de propiedad así:

```
Public Class Foo

    Public ReadOnly Property MyProperty As String = "Bar"

End Class
```

La implementación real que genera el compilador es exactamente la misma para ambos ejemplos. El nuevo método para escribirlo es solo una mano corta. El compilador seguirá generando un campo privado con el formato: `_<PropertyName>` para respaldar la propiedad de solo lectura.

Módulos parciales e interfaces

Similar a las clases parciales, la nueva versión de Visual Basic ahora puede manejar módulos parciales e interfaces parciales. La sintaxis y el comportamiento son exactamente los mismos que para las clases parciales.

Un ejemplo de módulo parcial:

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

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

Y una interfaz parcial:

```
Partial Interface Interfacel
    Sub Methodel()
End Interface

Partial Interface Interfacel
    Sub Methode2()
End Interface

Public Class Class1
    Implements Interfacel
    Public Sub Methodel() Implements Interfacel.Methodel
        Throw New NotImplementedException()
    End Sub

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

Al igual que para las clases parciales, las definiciones de los módulos e interfaces parciales deben ubicarse en el mismo espacio de nombres y el mismo ensamblado. Esto se debe a que las partes parciales de los módulos e interfaces se combinan durante la compilación y el ensamblaje compilado no contiene ninguna indicación de que la definición original del módulo o la interfaz se dividió.

Literales multilínea de cuerda

VB ahora permite cadenas literales que se dividen en varias líneas.

Sintaxis antigua:

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

Nueva sintaxis:

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

Mejoras directivas regionales

La directiva `#Region` ahora se puede colocar dentro de los métodos e incluso puede abarcar métodos, clases y módulos.

```
#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">[String] 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
```

Comentarios después de la continuación de la línea implícita

VB 14.0 introduce la posibilidad de agregar comentarios después de la continuación de línea implícita.

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

Manejo de excepciones

Durante la codificación, los errores inesperados surgen con la frecuencia suficiente, lo que requiere depuración y pruebas. Pero a veces los errores son realmente esperados y, para evitarlo, está el bloque `Try..Catch..Throw..Finally..End Try`.

Para administrar un error correctamente, el código se coloca en un bloque `Try..Catch`, por lo que el `Catch`, como su nombre indica, detectará todas las excepciones que surjan en este bloque.

Y en caso de excepción, tenemos la posibilidad de `Throw` el error, es decir, devolverlo para notificarlo al usuario o administrarlo internamente en el propio código.

El `Finally` parte es el código final que, cualquiera que sea el resultado, si hay una excepción o no, el código se ejecutará antes de salir del bloque.

En caso de que tengamos que salir del reloj, existe la instrucción `Exit Try` que se puede usar. Pero aquí también, el código en la sección `Finally` se ejecutará antes de terminar.

La sintaxis es simple;

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

donde solo es obligatorio el `Try` y el `End Try`. El resto se puede ignorar, pero como buena práctica, incluya la parte `Finally`, incluso si se deja en blanco.

Llegando a la excepción, hay diferentes tipos de excepción que pueden ser capturados. Están listas las excepciones disponibles desde .Net Framework, como se muestra a continuación;

| Clase de excepción | Breve descripción |
|--|--|
| <code>System.IO.IOException</code> | Maneja errores de E / S |
| <code>System.IndexOutOfRangeException</code> | Se refiere a un índice de matriz fuera de rango |
| <code>System.ArrayTypeMismatchException</code> | Cuando el tipo no coincide con el tipo de matriz |
| <code>System.NullReferenceException</code> | Maneja los errores generados al hacer referencia a un objeto nulo. |
| <code>System.DivideByZeroException</code> | Maneja los errores generados al dividir un dividendo con cero. |

| Clase de excepción | Breve descripción |
|-------------------------------|---|
| System.InvalidCastException | Maneja los errores generados durante el encasillado. |
| System.OutOfMemoryException | Maneja los errores generados desde la memoria libre insuficiente. |
| System.StackOverflowException | Maneja los errores generados por el desbordamiento de pila. |
| ----- | ----- |

Lea Características de Visual Basic 14.0 en línea: <https://riptutorial.com/es/vb-net/topic/1501/caracteristicas-de-visual-basic-14-0>

Capítulo 6: Compresión de archivos / carpetas

Examples

Creando un archivo zip desde el directorio

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

Cree el archivo `archive.zip` que contiene los archivos que están en mi `myfolder` . En el ejemplo, las rutas son relativas al directorio de trabajo del programa. Puede especificar rutas absolutas.

Extraer el archivo zip al directorio

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

Extrae `archive.zip` al directorio de mi carpeta. En el ejemplo, las rutas son relativas al directorio de trabajo del programa. Puede especificar rutas absolutas.

Crear archivo zip dinámicamente

```
' 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
```

Agregando compresión de archivos a su proyecto

1. En el *Explorador de soluciones*, vaya a su proyecto, haga clic derecho en *Referencias* y luego *Agregar referencia ...*
2. Busque *Compresión* y seleccione *System.IO.Compression.FileSystem* y luego presione OK.
3. Agregue `Imports System.IO.Compression` a la parte superior de su archivo de código (antes de cualquier clase o módulo, con las otras declaraciones de `Imports`).

```
Option Explicit On
```

```
Option Strict On

Imports System.IO.Compression

Public Class Foo

    ...

End Class
```

Tenga en cuenta que esta clase (ZipArchive) solo está disponible desde .NET version 4.5 en adelante

Lea [Compresión de archivos / carpetas en línea](https://riptutorial.com/es/vb-net/topic/4638/compresion-de-archivos---carpetas): <https://riptutorial.com/es/vb-net/topic/4638/compresion-de-archivos---carpetas>

Capítulo 7: Condiciones

Examples

Si ... entonces ... si no

```
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
```

Si operador

9.0

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

Podemos usar el operador **If en lugar de If ... Then ... Else..End If** instrucción de bloques.

Considere el siguiente ejemplo:

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

es lo mismo que

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

`If()` usa la evaluación de *cortocircuito*, lo que significa que solo evaluará los argumentos que usa. Si la condición es falsa (o un `Nullable` que es `Nothing`), la primera alternativa no se evaluará en absoluto, y no se observará ninguno de sus efectos secundarios. Esto es efectivamente el mismo que el [operador ternario de C #](#) en la forma de `condition?a:b`.

Esto es especialmente útil para evitar excepciones:

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

Todos sabemos que dividir por cero arrojará una excepción, pero `If()` aquí protege contra esto haciendo un cortocircuito a solo la expresión que la condición ya ha asegurado es válida.

Otro ejemplo:

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

Si `varString <> "N/A"` se evalúa como `False`, asignará `varDate` valor 's como `Now.Date` sin evaluar la primera expresión.

9.0

Las versiones anteriores de VB no tienen el operador `If()` y tienen que conformarse con la función integrada `IIf()`. Como se trata de una función, no un operador, *no* lo hace corto circuito; todas las expresiones se evalúan, con todos los efectos secundarios posibles, incluidas las penalizaciones de rendimiento, el cambio de estado y las excepciones de lanzamiento. (Los dos ejemplos anteriores que evitan las excepciones se generarían si se convirtieran a `IIf`). Si alguno de estos efectos secundarios presenta un problema, no hay forma de usar un condicional en línea; en su lugar, confíe en `If..Then` bloquea como de costumbre.

Lea Condiciones en línea: <https://riptutorial.com/es/vb-net/topic/7484/condiciones>

Capítulo 8: Consola

Examples

Console.ReadLine ()

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

`Console.ReadLine()` leerá la entrada de la consola del usuario, hasta que se detecte la próxima nueva línea (generalmente al presionar la tecla Intro o Retorno). La ejecución del código se detiene en el hilo actual hasta que se proporciona una nueva línea. Posteriormente, se ejecutará la siguiente línea de código.

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()
```

El método `Console.WriteLine()` imprimirá los argumentos dados **con** una nueva línea adjunta al final. Esto imprimirá cualquier objeto suministrado, incluyendo, pero no limitado a, cadenas, números enteros, variables, números de punto flotante.

Al escribir objetos que no son llamados explícitamente por las diversas sobrecargas de `WriteLine` (es decir, está utilizando la sobrecarga que espera un valor de tipo `Object` , `WriteLine` usará el método `.ToString()` para generar una `String` para escribir realmente. los objetos deben anular el método `.ToString` y producir algo más significativo que la implementación predeterminada (que normalmente solo escribe el nombre de tipo completamente calificado).

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 '
```

El método `Console.Write()` es idéntico al método `Console.WriteLine()` excepto que imprime los argumentos dados **sin** una nueva línea adjunta al final. Este método se puede hacer funcionalmente idéntico a `WriteLine` agregando una cadena de nueva línea al final de los argumentos proporcionados:

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

Console.Read ()

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

`Console.Read()` espera la entrada del usuario y, al recibirla, devuelve un valor entero correspondiente con el código de carácter del carácter ingresado. Si el flujo de entrada finaliza de alguna manera antes de poder obtener la entrada, se devuelve -1 en su lugar.

Console.ReadKey ()

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

`Console.ReadKey()` espera la entrada del usuario y, al recibirla, devuelve un objeto de la clase `ConsoleKeyInfo`, que contiene información relevante para el carácter que el usuario proporcionó como entrada. Para obtener detalles sobre la información proporcionada, visite la [documentación de MSDN](#).

Prototipo de línea de comandos

```
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 acknowledged: -" & 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

```

Este prototipo genera un intérprete de línea de comandos básico.

Obtiene automáticamente el nombre de la aplicación y la versión para comunicarse con el usuario. Para cada línea de entrada, reconoce el comando y una lista arbitraria de argumentos, todos separados por espacios.

Como ejemplo básico, este código comprende los comandos *ver*, *quit* y *exit*.

El parámetro *Project.BaseClass* es una clase de su proyecto donde se establecen los detalles del ensamblaje.

Lea Consola en línea: <https://riptutorial.com/es/vb-net/topic/602/consola>

Capítulo 9: Declarando variables

Sintaxis

- Contador público como entero
- Contador privado como entero
- Contador tenue como entero

Examples

Declarar y asignar una variable usando un tipo primitivo

Las variables en Visual Basic se declaran usando la palabra clave `Dim`. Por ejemplo, esto declara una nueva variable llamada `counter` con el tipo de datos `Integer`:

```
Dim counter As Integer
```

Una declaración de variable también puede incluir un [modificador de acceso](#), como `Public`, `Protected`, `Friend` o `Private`. Esto funciona en conjunto con el [alcance](#) de la variable para determinar su accesibilidad.

| Modificador de acceso | Sentido |
|---------------------------|--|
| Público | Todos los tipos que puedan acceder al tipo envolvente. |
| Protegido | Solo la clase envolvente y las que la heredan. |
| Amigo | Todos los tipos en el mismo ensamblaje que pueden acceder al tipo envolvente |
| Amigo protegido | La clase envolvente y sus herederos, o los tipos en el mismo ensamblaje que pueden acceder a la clase envolvente |
| Privado | Solo el tipo que lo encierra. |
| Estático | Solo en variables locales y solo inicializa una vez. |

Como una abreviatura, la palabra clave `Dim` se puede reemplazar con el modificador de acceso en la declaración de la variable:

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

Los tipos de datos admitidos se describen en la siguiente tabla:

| Tipo | Alias | Asignación de memoria | Ejemplo |
|--------------|----------|--|--|
| SByte | N / A | 1 byte | Dim example As SByte = 10 |
| Int16 | Corto | 2 bytes | Dim example As Short = 10 |
| Int32 | Entero | 4 bytes | Dim example As Integer = 10 |
| Int64 | Largo | 8 bytes | Dim example As Long = 10 |
| Soltero | N / A | 4 bytes | Dim example As Single = 10.95 |
| Doble | N / A | 8 bytes | Dim example As Double = 10.95 |
| Decimal | N / A | 16 bytes | Dim example As Decimal = 10.95 |
| Booleano | N / A | Dictado por plataforma implementadora. | Dim example As Boolean = True |
| Carbonizarse | N / A | 2 bytes | Dim example As Char = "A"C |
| Cuerda | N / A | $20 + \left\lceil \frac{length}{2} \right\rceil * 4bytes$ fuente | Dim example As String = "Stack Overflow" |
| Fecha y hora | Fecha | 8 bytes | Dim example As Date = Date.Now |
| Byte | N / A | 1 byte | Dim example As Byte = 10 |
| UInt16 | UShort | 2 bytes | Dim example As UShort = 10 |
| UInt32 | UInteger | 4 bytes | Dim example As UInteger = 10 |
| UInt64 | De largo | 8 bytes | Dim example As ULong = 10 |
| Objeto | N / A | Arquitectura de 4 bytes y 32 bits, arquitectura de 8 bytes y 64 bits. | Dim example As Object = Nothing |

También existen identificadores de datos y caracteres de tipo literal que se pueden usar en reemplazo del tipo de texto o para forzar el tipo de literal:

| Tipo (o alias) | Tipo de identificador de caracteres | Personaje de tipo literal |
|----------------|-------------------------------------|-------------------------------|
| Corto | N / A | example = 10S |
| Entero | Dim example% | example = 10% O example = 10I |

| Tipo (o alias) | Tipo de identificador de caracteres | Personaje de tipo literal |
|----------------|-------------------------------------|-------------------------------|
| Largo | Dim example& | example = 10& O example = 10L |
| Soltero | Dim example! | example = 10! O example = 10F |
| Doble | Dim example# | example = 10# O example = 10R |
| Decimal | Dim example@ | example = 10@ O example = 10D |
| Carbonizarse | N / A | example = "A"C |
| Cuerda | Dim example\$ | N / A |
| UShort | N / A | example = 10US |
| UInteger | N / A | example = 10UI |
| De largo | N / A | example = 10UL |

Los sufijos integrales también se pueden usar con prefijos hexadecimales (& H) u octales (& O):

```
example = &H8000S O example = &O77&
```

Los objetos de fecha (hora) también se pueden definir utilizando la sintaxis literal:

```
Dim example As Date = #7/26/2016 12:8 PM#
```

Una vez que se declara una variable, existirá dentro del **Ámbito** del tipo que contiene, `Sub` o `Function` declarada, como un ejemplo:

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

    Return counter
End Function
```

La variable de contador solo existirá hasta la `End Function` y luego quedará fuera de alcance. Si esta variable de contador es necesaria fuera de la función, tendrá que definirla a nivel de clase / estructura o módulo.

```
Public Class ExampleClass

    Private _counter As Integer

    Public Function IncrementCounter() As Integer
        _counter += 1
        Return _counter
    End Function

End Class
```

Alternativamente, puede usar el modificador `Static` (que no debe confundirse con `Shared`) para permitir que una variable local retenga su valor entre las llamadas de su método de cierre:

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

    Return counter
End Function
```

Niveles de declaración - Variables locales y miembros.

Variables locales : aquellas declaradas dentro de un procedimiento (subrutina o función) de una clase (u otra estructura). En este ejemplo, `exampleLocalVariable` es una variable local declarada dentro de `ExampleFunction()` :

```
Public Class ExampleClass1

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

End Class
```

La palabra clave `Static` permite que una variable local se retenga y mantenga su valor después de la terminación (donde generalmente, las variables locales dejan de existir cuando finaliza el procedimiento de contención).

En este ejemplo, la consola es `024` . En cada llamada a `ExampleSub()` desde `Main()` la variable estática retiene el valor que tenía al final de la llamada anterior:

```
Module Module1

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

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

End Module
```

Variables del miembro : declaradas fuera de cualquier procedimiento, a nivel de clase (u otra estructura). Pueden ser **variables de instancia** , en las que cada instancia de la clase que contiene tiene su propia copia distinta de esa variable, o **variables Shared** , que existen como una sola variable asociada con la clase en sí, independientemente de cualquier instancia.

Aquí, `ExampleClass2` contiene dos variables miembro. Cada instancia de la `ExampleClass2` tiene un individuo `ExampleInstanceVariable` que se puede acceder a través de la referencia de clase. La variable compartida `ExampleSharedVariable` sin embargo se accede utilizando el nombre de clase:

```

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

```

Ejemplo de modificadores de acceso

En el siguiente ejemplo, considere que tiene una solución que aloja dos proyectos: **ConsoleApplication1** y **SampleClassLibrary** . El primer proyecto tendrá las clases **SampleClass1** y **SampleClass2** . El segundo tendrá **SampleClass3** y **SampleClass4** . En otras palabras, tenemos dos ensamblajes con dos clases cada una. **ConsoleApplication1** tiene una referencia a **SampleClassLibrary** .

Vea cómo **SampleClass1.MethodA** interactúa con otras clases y métodos.

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
    End Sub
End Class

```



```

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
```

Lea Declarando variables en línea: <https://riptutorial.com/es/vb-net/topic/3366/declarando-variables>

Capítulo 10: Depurando tu aplicación

Introducción

Siempre que tenga un problema en su código, siempre es una buena idea saber qué ocurre dentro. La clase `System.Diagnostics.Debug` en .Net Framework te ayudará mucho en esta tarea.

La primera ventaja de la clase `Debug` es que produce código solo si compila su aplicación en modo `Debug`. Cuando compila su aplicación en el modo `Release`, no se generará ningún código a partir de las llamadas de depuración.

Examples

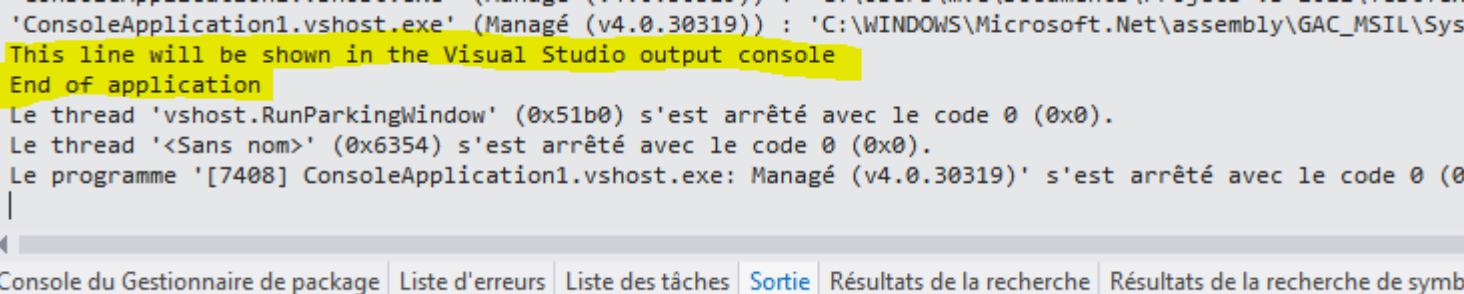
Depurar en la consola

```
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
```

Producirá:



The screenshot shows the Visual Studio output console with the following text: `'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Sys`, `This line will be shown in the Visual Studio output console`, `End of application`, `Le thread 'vshost.RunParkingWindow' (0x51b0) s'est arrêté avec le code 0 (0x0).`, `Le thread '<Sans nom>' (0x6354) s'est arrêté avec le code 0 (0x0).`, and `Le programme '[7408] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0`. The console also shows navigation buttons at the bottom: `Console du Gestionnaire de package`, `Liste d'erreurs`, `Liste des tâches`, `Sortie`, `Résultats de la recherche`, and `Résultats de la recherche de symb`.

Sangrando su salida de depuración

```
Module Module1

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

    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

```

Producirá:

```

'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System
Starting application
  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).

```

[Console du Gestionnaire de package](#) |
[Liste d'erreurs](#) |
[Liste des tâches](#) |
[Sortie](#) |
[Résultats de la recherche](#) |
[Résultats de la recherche de symbole](#)

Depurar en un archivo de texto

Al comienzo de su aplicación, debe agregar un [TextWriterTraceListener](#) a la lista de [escuchas](#) de la clase de depuración.

```

Module Module1

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

        Debug.WriteLine("Starting application")

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

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

```

Todo el código de depuración producido se emitirá en la consola de Visual Studio Y en el archivo de texto que elija.

Si el archivo es siempre el mismo:

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

La salida se adjuntará al archivo cada vez que Y un nuevo archivo comience con un GUID, luego se generará su nombre de archivo.

Lea **Depurando tu aplicación en línea**: <https://riptutorial.com/es/vb-net/topic/8631/depurando-tu-aplicacion>

Capítulo 11: Enhebrado

Examples

Realizar llamadas a prueba de subprocessos usando Control.Invoke ()

Usando el método `Control.Invoke()`, puede mover la ejecución de un método o función desde un hilo de fondo al hilo en el que se creó el control, que generalmente es el hilo de la interfaz de usuario. Al hacerlo, su código se pondrá en cola para ejecutarse en el subprocesso del control, lo que elimina la posibilidad de concurrencia.

La propiedad `Control.InvokeRequired` también debe verificarse para determinar si necesita invocar o si el código ya se está ejecutando en el mismo hilo que el control.

El método `Invoke()` toma un delegado como primer parámetro. Un delegado mantiene la referencia, la lista de parámetros y el tipo de retorno a otro método.

En Visual Basic 2010 (10.0) o superior, las *expresiones lambda* se pueden usar para crear un método delegado sobre la marcha:

```
If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(Sub() LogTextBox.AppendText("Check passed"))
Else
    LogTextBox.AppendText("Check passed")
End If
```

Mientras que en Visual Basic 2008 (9.0) o inferior, debes declarar al delegado por tu cuenta:

```
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
```

Realizar llamadas a prueba de subprocessos utilizando Async / Await

Si intentamos cambiar un objeto en el subprocesso de la interfaz de usuario desde un subprocesso diferente, obtendremos una excepción de operación entre subprocessos:

```
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
```

Antes de **VB 14.0** y **.NET 4.5**, la solución invocaba la asignación y el objeto que viven en el hilo de la interfaz de usuario:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the code on the UI thread:
    MyButton.Invoke(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Con **VB 14.0**, podemos ejecutar una `Task` en un subproceso diferente y luego restaurar el contexto una vez que se completa la ejecución y luego realizar la asignación con `Async / 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
```

Lea Enhebrado en línea: <https://riptutorial.com/es/vb-net/topic/1913/enhebrado>

Capítulo 12: Enlace de datos WPF XAML

Introducción

Este ejemplo muestra cómo crear un ViewModel y una Vista dentro del patrón MVVM y WPF, y cómo unir los dos, para que cada uno se actualice cada vez que se cambie el otro.

Examples

Enlace de una cadena en el ViewModel a un TextBox en la 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
```

Se puede agregar fácilmente una `wpf:dp` usando el fragmento de código `wpf:dp` (escriba `wpf:dp` , luego presione la tecla `TAB` dos veces), sin embargo, el fragmento de código no es seguro y no se compilará en `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
```

Tenga en cuenta que esta es una forma muy rudimentaria de implementar MVVM y enlace de datos. Una práctica más sólida sería usar una plataforma como Unity para "inyectar" el ViewModel en la Vista.

Lea [Enlace de datos WPF XAML en línea](https://riptutorial.com/es/vb-net/topic/8177/enlace-de-datos-wpf-xaml): <https://riptutorial.com/es/vb-net/topic/8177/enlace-de-datos-wpf-xaml>

Capítulo 13: Enumerar

Examples

Definición de enumeración

Una enumeración es un conjunto de constantes lógicamente relacionadas.

```
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
```

Inicialización de miembros

Cada uno de los miembros de la enumeración puede inicializarse con un valor. Si no se especifica un valor para un miembro, de manera predeterminada, se inicializa en 0 (si es el primer miembro de la lista de miembros) o en un valor mayor en 1 que el valor del miembro anterior.

```
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
```

El atributo Flags

Con el atributo `<Flags>` , la enumeración se convierte en un conjunto de indicadores. Este atributo permite asignar múltiples valores a una variable de enumeración. Los miembros de una enumeración de banderas deben inicializarse con potencias de 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 ()

El método `HasFlag()` se puede usar para verificar si un indicador está establecido.

```
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
```

Para obtener más información sobre el atributo `Flags` y cómo se debe usar, consulte [la documentación oficial de Microsoft](#) .

Análisis de cuerdas

Se puede crear una instancia de Enum analizando una representación de cadena del 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
```

Vea también: [Analizar una cadena a un valor Enum en VB.NET](#)

GetNames ()

Devuelve los nombres de constantes en el Enum especificado como una matriz de cadena:

```
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
```

Salida:

Pequeña

Medio

Grande

GetValues ()

'Este método es útil para iterar valores 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
```

Huellas dactilares:

1

2

4

Encadenar()

El método `ToString` en una enumeración devuelve el nombre de cadena de la enumeración. Por ejemplo:

```
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
```

Sin embargo, si se desea la representación de cadena del valor entero real de la enumeración, puede convertir la enumeración a un `Integer` y luego llamar a `ToString` :

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

Determine si un Enum tiene `FlagsAttribute` especificado o no

El siguiente ejemplo se puede usar para determinar si una enumeración tiene el [atributo `FlagsAttribute`](#) especificado. La metodología utilizada se basa en la [reflexión](#) .

Este ejemplo dará un resultado `True` :

```
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)
```

Este ejemplo dará un resultado `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)
```

Podemos diseñar un método de extensión de uso genérico como este:

```
<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
```

Ejemplo de uso:

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

Para cada bandera (iteración de bandera)

En algunos escenarios muy específicos, sentiríamos la necesidad de realizar una acción específica para cada marca de la enumeración de origen.

Podemos escribir un método de extensión *genérico* simple para realizar esta tarea.

```
<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
```

Ejemplo de uso:

```
Dim flags As FileAttributes = (FileAttributes.ReadOnly Or FileAttributes.Hidden)  
  
flags.ForEachFlag(Of FileAttributes) (  
    Sub(ByVal x As FileAttributes)
```

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

Determine la cantidad de banderas en una combinación de bandera

El siguiente ejemplo está destinado a contar la cantidad de banderas en la combinación de bandera especificada.

El ejemplo se proporciona como un método de extensión:

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

Ejemplo de uso:

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

Encuentra el valor más cercano en un Enum

El siguiente código ilustra cómo encontrar el valor más cercano de un **Enum**.

Primero definimos este **Enum** que servirá para especificar los criterios de búsqueda (dirección de búsqueda)

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

Y ahora implementamos el algoritmo de búsqueda:

```
<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
                ).FirstOrDefault
    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
                ).FirstOrDefault
    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

```

Ejemplo de uso:

```

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)

```

Lea Enumerar en línea: <https://riptutorial.com/es/vb-net/topic/1809/enumerar>

Capítulo 14: Excepcion de referencia nula

Observaciones

`NullReferenceException` se produce cuando una variable está vacía y se hace referencia a uno de sus métodos / propiedades. Para evitar esto, asegúrese de que todas las variables se inicialicen correctamente (`new` operador) y que todos los métodos devuelvan un valor no nulo.

Examples

Variable no inicializada

CÓDIGO MALO

```
Dim f As System.Windows.Forms.Form
f.ShowDialog()
```

BUEN CODIGO

```
Dim f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Dim f As New System.Windows.Forms.Form ' alternative syntax
f.ShowDialog()
```

INCLUSO MEJOR CÓDIGO (Asegurar la eliminación adecuada del objeto `IDisposable` [más información](#))

```
Using f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Using f As New System.Windows.Forms.Form ' alternative syntax
    f.ShowDialog()
End Using
```

Retorno vacio

```
Function TestFunction() As TestClass
    Return Nothing
End Function
```

CÓDIGO MALO

```
TestFunction().TestMethod()
```

BUEN CODIGO

```
Dim x = TestFunction()
If x IsNot Nothing Then x.TestMethod()
```

Operador Condicional Nulo

```
TestFunction()?.TestMethod()
```

Lea Excepcion de referencia nula en línea: <https://riptutorial.com/es/vb-net/topic/4076/excepcion-de-referencia-nula>

Capítulo 15: Fecha

Examples

Convertir (analizar) una cadena en una fecha

Si conoce el formato de la cadena que está convirtiendo (análisis), debe usar `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)
```

Si no está seguro del formato de la cadena, puede usar `DateTime.TryParseExact` y probar el resultado para ver si está analizado o no:

```
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
```

Convertir una fecha en una cadena

Simplemente use la sobrecarga `.ToString` de un objeto `DateTime` para obtener el formato que necesita:

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

Lea Fecha en línea: <https://riptutorial.com/es/vb-net/topic/3727/fecha>

Capítulo 16: Formación

Observaciones

```
Dim myArray(2) As Integer

someFunc(myArray)
```

Una matriz es una colección de objetos ordenada por índice. El tipo de objeto se define por el tipo dado en la declaración de matriz.

Las matrices en Visual Basic .NET suelen estar basadas en cero (0), lo que significa que el primer índice es 0. Una matriz de 10 elementos tendrá un rango de índice de 0-9. Al acceder a los elementos de la matriz, el índice máximo accesible es uno menos que el número total de elementos. Debido a esto, los bucles que acceden a los índices de matriz de forma incremental siempre deben hacer una verificación de rango donde el valor es menor que la longitud de la matriz.

Examples

Definición de matriz

```
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).
```

De base cero

Todas las matrices en VB.NET están basadas en cero. En otras palabras, el índice del primer elemento (el límite inferior) en una matriz VB.NET siempre es 0. Las versiones anteriores de VB, como VB6 y VBA, se basaban en una sola de forma predeterminada, pero proporcionaban una forma de anular los límites predeterminados. En esas versiones anteriores de VB, los límites inferior y superior podrían establecerse explícitamente (por ejemplo, `Dim array(5 To 10)`). En VB.NET, para mantener la compatibilidad con otros lenguajes .NET, se eliminó esa flexibilidad y

el límite inferior de 0 ahora siempre se aplica. Sin embargo, la sintaxis de To aún se puede usar en VB.NET, lo que puede aclarar el rango de manera explícita. Por ejemplo, los siguientes ejemplos son todos equivalentes a los enumerados anteriormente:

```
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)
```

Declaraciones de matriz anidadas

```
Dim myArray = {{1, 2}, {3, 4}}
```

Declarar una matriz de dimensión única y establecer valores de elementos de matriz

```
Dim array = New Integer() {1, 2, 3, 4}
```

o

```
Dim array As Int32() = {1, 2, 3, 4}
```

Inicialización de matriz

```
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.
```

Inicialización de matriz multidimensional

```
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
```

Inicialización de matriz irregular

Tenga en cuenta el paréntesis para distinguir entre una matriz dentada y una matriz multidimensional Las subreglas pueden tener una longitud diferente

```
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
```

Variables de matriz nula

Dado que las matrices son tipos de referencia, una variable de matriz puede ser nula. Para declarar una variable de matriz nula, debe declararla sin un tamaño:

```
Dim array() As Integer
```

O

```
Dim array As Integer()
```

Para verificar si una matriz es nula, compruebe si no `Is Nothing` :

```
Dim array() As Integer  
If array Is Nothing Then  
    array = {1, 2, 3}  
End If
```

Para establecer una variable de matriz existente en nulo, simplemente configúrela en `Nothing` :

```
Dim array() As Integer = {1, 2, 3}  
array = Nothing  
Console.WriteLine(array(0)) ' Throws a NullReferenceException
```

O usa `Erase` , que hace lo mismo:

```
Dim array() As Integer = {1, 2, 3}  
Erase array  
Console.WriteLine(array(0)) ' Throws a NullReferenceException
```

Haciendo referencia a la misma matriz de dos variables

Dado que las matrices son tipos de referencia, es posible tener varias variables que apuntan al mismo objeto de matriz.

```
Dim array1() As Integer = {1, 2, 3}  
Dim array2() As Integer = array1  
array1(0) = 4  
Console.WriteLine(String.Join(", ", array2)) ' Writes "4, 2, 3"
```

Límites inferiores distintos de cero.

Con `Option Strict On` , aunque .NET Framework permite la creación de matrices de una sola dimensión con límites inferiores a cero, no son "vectores" y, por lo tanto, no son compatibles con

las matrices de tipo VB.NET. Esto significa que solo se pueden ver como `Array` y, por lo tanto, no se pueden usar referencias de array (`index`) normales.

```
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
```

Además de usar `Option Strict Off`, puede recuperar la sintaxis (`index`) al tratar la matriz como un `IList`, pero luego no es una matriz, por lo que no puede usar `LBound` y `UBound` en ese nombre de variable (y todavía no evito el boxeo):

```
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
```

Las matrices de límites inferiores no nulos multidimensionales *son* compatibles con las matrices de tipos multidimensionales 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
```

Referencia de MSDN: [Array.CreateInstance](#)

Lea Formación en línea: <https://riptutorial.com/es/vb-net/topic/805/formacion>

Capítulo 17: Funciones

Introducción

La función es como sub. Pero la función devuelve un valor. Una función puede aceptar parámetros únicos o múltiples.

Examples

Definiendo una función

Es realmente fácil definir las funciones.

```
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
```

Definiendo una función # 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!
```

Lea Funciones en línea: <https://riptutorial.com/es/vb-net/topic/10088/funciones>

Capítulo 18: GDI +

Examples

Crear objeto gráfico

Hay tres formas de crear un objeto gráfico.

1. Desde el **evento de pintura**

Cada vez que se vuelve a dibujar el control (redimensionado, actualizado ...) se llama a este evento, use de esta manera si desea que el control se dibuje constantemente en el control

```
'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. **Crear gráfico**

Esto se usa con más frecuencia cuando desea crear un gráfico de una sola vez en el control o no desea que el control se vuelva a pintar.

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

3. De un **gráfico existente**

Utilice este método cuando desee dibujar y cambiar un gráfico existente

```
'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)
```

Dibujar formas

Para comenzar a dibujar una forma, debe definir un objeto de lápiz. El `Pen` acepta dos parámetros:

1. Color de la pluma o pincel
2. Ancho de la pluma

El objeto Pluma se utiliza para crear un **esquema** del objeto que desea dibujar

Después de definir la pluma, puede establecer propiedades específicas de la pluma

```
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
```

Luego usa el objeto gráfico que creaste para dibujar 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
```

Por defecto, el ancho del lápiz es igual 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
```

Formas de relleno

Graphics.FillShapes dibuja una forma y la llena con el color dado. Formas de relleno pueden usar

1. Herramienta de `Brush` - para rellenar la forma con un color sólido

```
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. Herramienta `HatchBrush` - para rellenar la forma con un patrón

```
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` - para rellenar la forma con un degradado

```
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` - para rellenar la forma con una imagen

Puede elegir una imagen de los recursos, un mapa de bits ya definido o de un nombre de archivo

```
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)
```

Tanto la `Hatch Brush Tool LinearGradientBrush` como `LinearGradientBrush` importan la siguiente declaración: **Imports System.Drawing.Drawing2D**

Texto

Para dibujar texto en el formulario use el método `DrawString`

Cuando dibuja una cadena puede usar cualquiera de los 4 pinceles enumerados anteriormente

```
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, 40))
'this will draw the word "Hello" at the given point, with a linearGradient Brush
```

Ya que no puede definir el ancho o alto del texto, use `Measure Text` para verificar el tamaño del texto

```
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
```

Ej: debe dibujar la palabra "Prueba" en la parte superior del formulario. El ancho del formulario es 120. Use este bucle para disminuir el tamaño de la fuente hasta que encaje en el ancho del formulario

```
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
```

Lea GDI + en línea: <https://riptutorial.com/es/vb-net/topic/5096/gdi-plus>

Capítulo 19: Genéricos

Examples

Crear una clase genérica

Se crea un tipo genérico para adaptarse de modo que la misma funcionalidad pueda ser accesible para diferentes tipos de datos.

```
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
```

Instancia de una clase genérica

Al crear una instancia de la misma clase con un tipo diferente dado, la interfaz de la clase cambia según el tipo dado.

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

theStringClass.

doSomething Public Sub doSomething(newItem As String)

Definir una clase 'genérica'

Una clase genérica es una clase que se adapta a un tipo dado posteriormente para que la misma funcionalidad se pueda ofrecer a diferentes tipos.

En este ejemplo básico se crea una clase genérica. Tiene un sub que usa el tipo genérico T. Al programar esta clase, no sabemos el tipo de T. En este caso, T tiene todas las características de Objeto.

```
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
```

Usa una clase genérica

En este ejemplo hay 2 instancias creadas de la Clase SomeClass. Dependiendo del tipo dado, las 2 instancias tienen una interfaz diferente:

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

theStringClass.

doSomething Public Sub doSomething(newItem As String)

theIntegerClass.

doSomething Public Sub doSomething(newItem As Integer)

La clase genérica más famosa es la lista (de)

Limite los tipos posibles dados

Los posibles tipos pasados a una nueva instancia de SomeClass deben heredar SomeBaseClass. Esto también puede ser una interfaz. Las características de SomeBaseClass son accesibles dentro de esta definición de clase.

```
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
```

Crear una nueva instancia del tipo dado

La creación de una nueva instancia de un tipo genérico se puede realizar / realizar en el momento de la compilación.

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

O con tipos limitados:

```
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 (si ninguna es un Objeto dado) debe tener un parámetro menos constructor.

Esto también se puede hacer en tiempo de ejecución a través de la [reflexión](#).

Lea Genéricos en línea: <https://riptutorial.com/es/vb-net/topic/6572/genericos>

Capítulo 20: Google Maps en un formulario de Windows

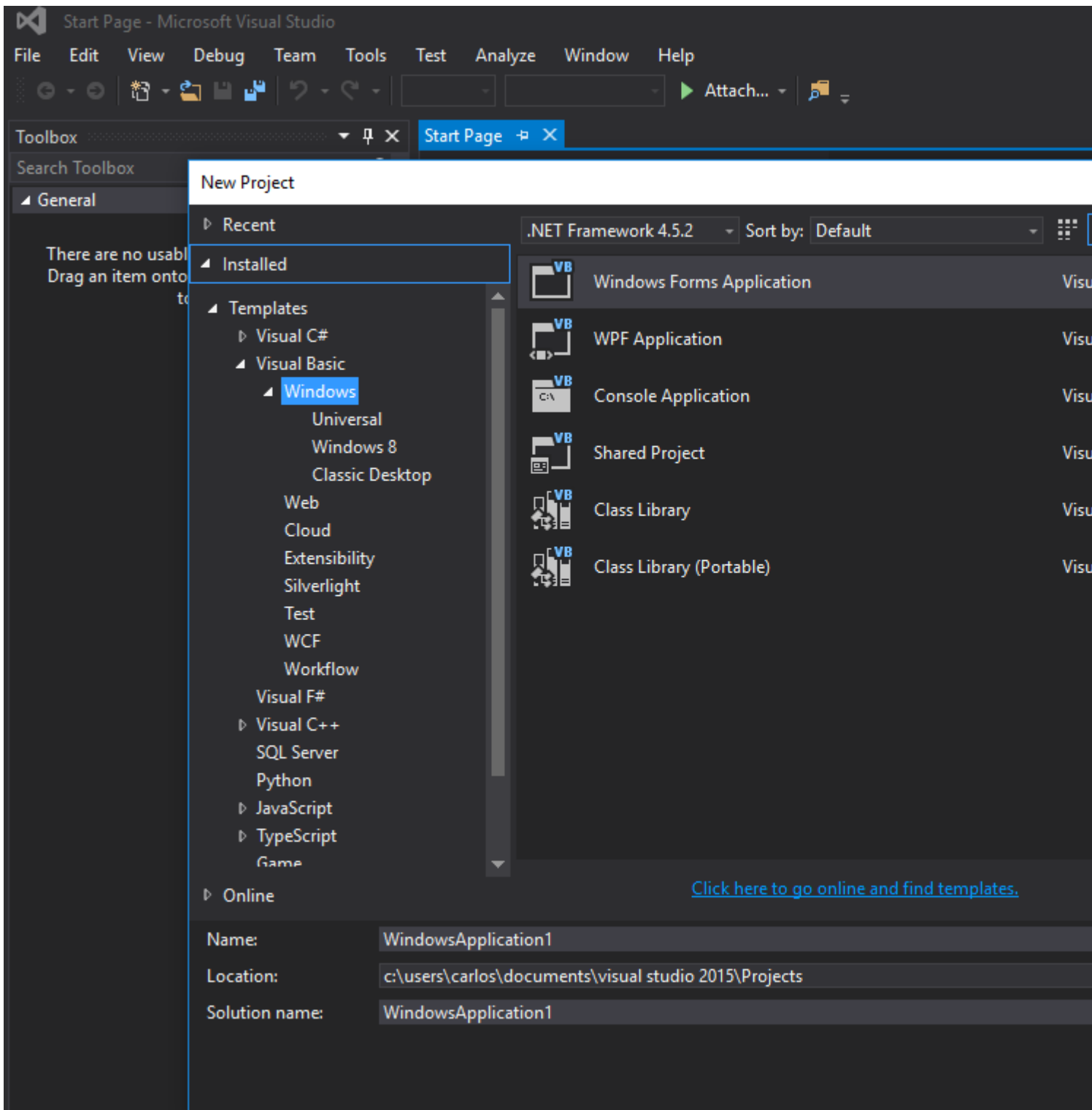
Examples

Cómo utilizar un mapa de Google en un formulario de Windows

La primera parte de este ejemplo explica cómo implementarlo. En el segundo, explicaré cómo funciona. Esto trata de ser un ejemplo general. La plantilla para el mapa (ver paso 3) y las funciones de ejemplo son completamente personalizables.

IMPLEMENTACIÓN
#####

Paso 1. En primer lugar, crea un nuevo proyecto y selecciona la aplicación Windows Form. Dejemos su nombre como "Form1".



Paso 2. Agregue un control WebBrowser (que contendrá su mapa) a su Form1. Llamémoslo "wbmap"

Paso 3. Crea un archivo .html llamado "googlemap_template.html" con tu editor de texto favorito y pega el siguiente código:

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;
    }
  }
}

```

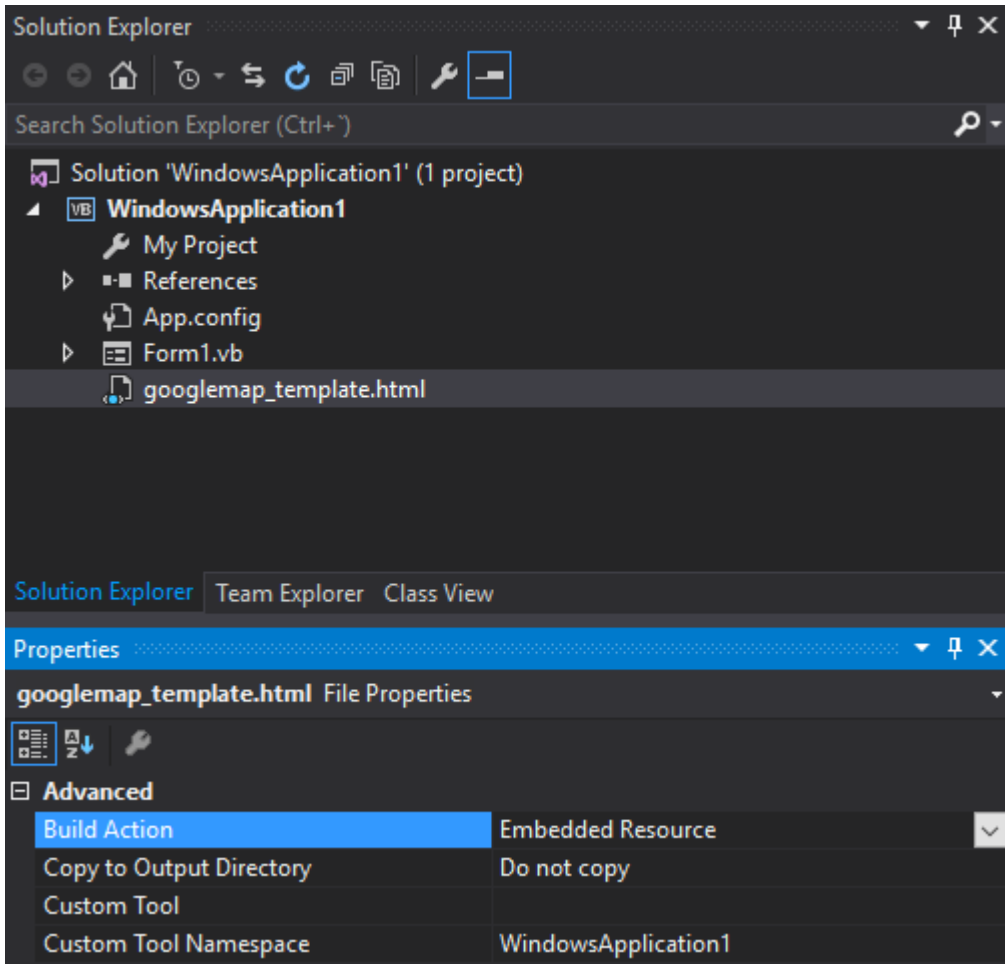
```
    google.maps.event.addDomListener(window, 'load', initialize);
</script>
<script type="text/javascript">
    // Function triggered from the WF with no arguments
    function showJavascriptHelloWorld() {
        alert("Hello world in HTML from WF");
    }
</script>
<script type="text/javascript">
    // 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>
```

Esto servirá como nuestra plantilla de mapa. Explicaré cómo funciona más tarde.

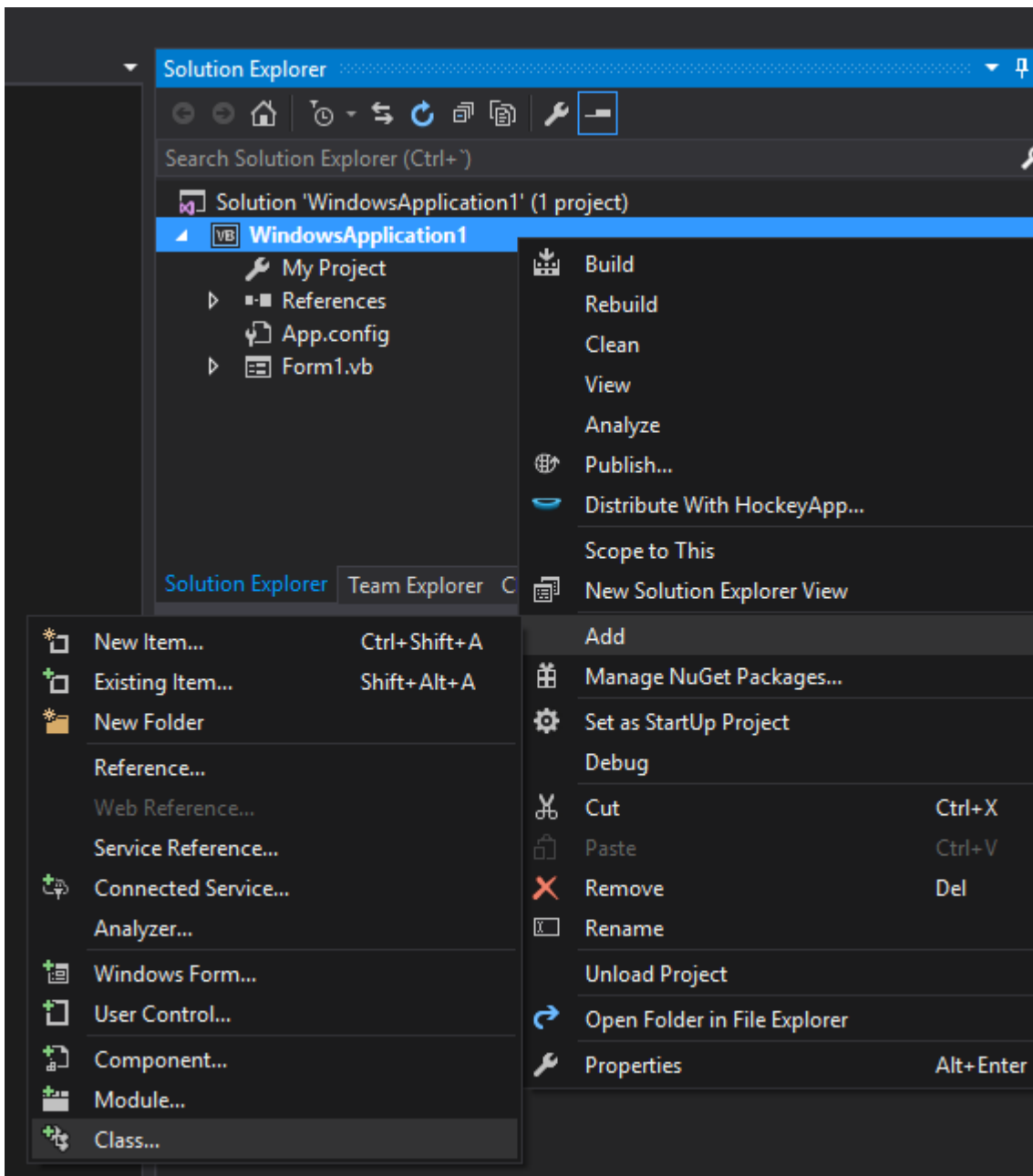
Paso 4. Agregue el archivo `googlemap_template.html` a su proyecto (haga clic derecho en su proyecto-> agregar-> elemento existente)

Paso 5. Una vez que aparezca en el Explorador de soluciones, establezca sus propiedades en:

- Acción de compilación -> Recurso incrustado
- Espacio de nombres de herramientas personalizado -> escriba el nombre del proyecto



Paso 6. Agregue una nueva clase (haga clic derecho en su proyecto-> agregar-> clase). En mi ejemplo lo llamaré GoogleMapHelper.



Paso 7. Pega el siguiente código en tu clase:

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) & "]"
        End If
    Next i
    mMarkerDataAsText += "]"
End Function

```

```

& mMarkerData(i, 2) & "]"
    ElseIf dataSize = 4 Then 'lat,lon,title and image
        mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & "," & mMarkerData(i, 2) & "," & 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) + "'") & vbCrLf & 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
```

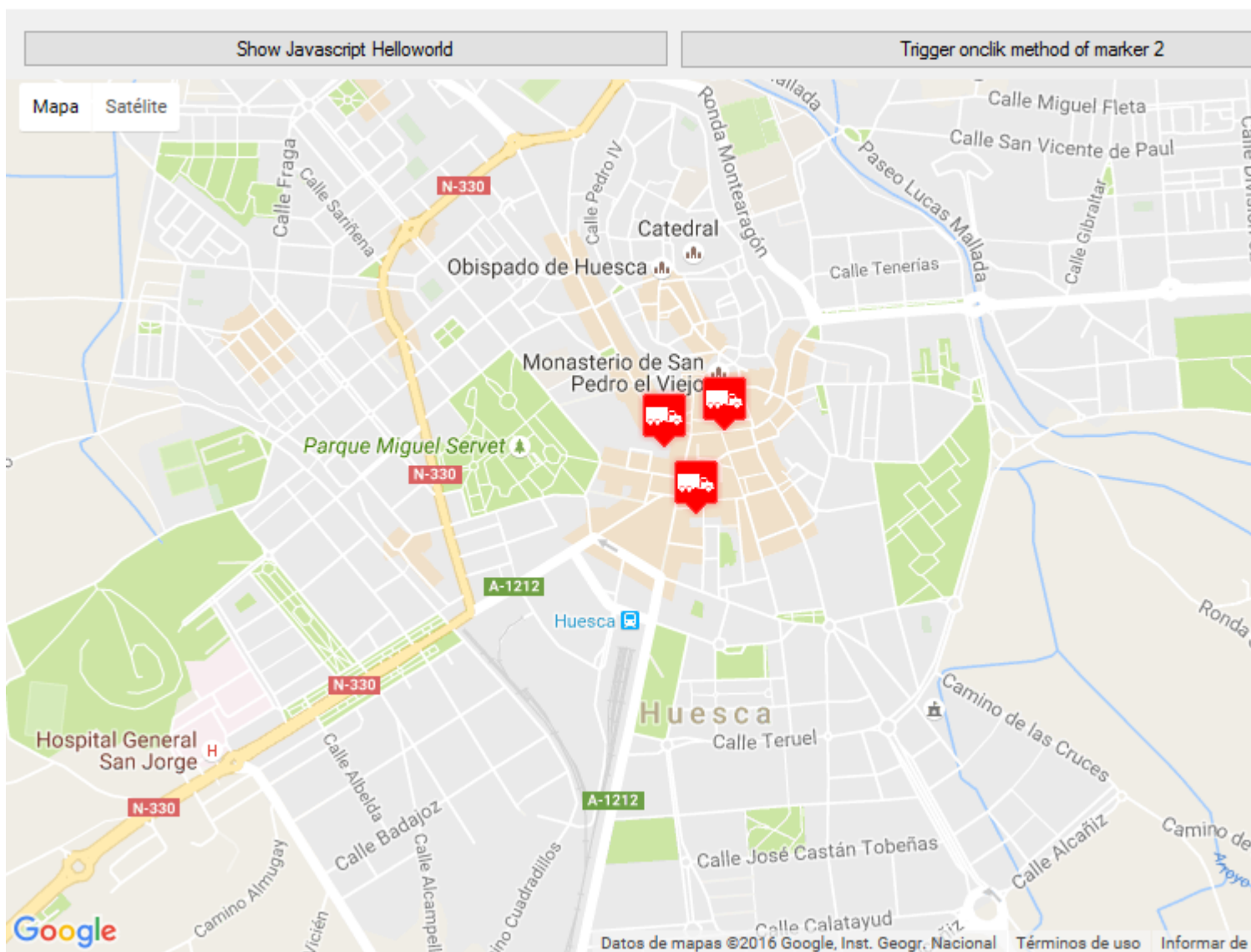
Nota: la constante MAP_TEMPLATE debe incluir el nombre de su proyecto

Paso 8. Ahora podemos usar nuestra clase GoogleMapHelper para cargar el mapa en nuestro navegador web simplemente creando e instancia y llamando a su método loadMap (). Cómo construyes tu markerData depende de ti. En este ejemplo, para aclarar, los escribo a mano. Hay 3 opciones para definir los datos del marcador (consulte los comentarios de la clase GoogleMapHelper). Tenga en cuenta que si usa la tercera opción (incluidos el título y los iconos), debe crear una carpeta llamada "marker_icons" (o lo que defina en la constante ICON_FOLDER de GoogleMapHelper) en su carpeta Debug / Release y colocar allí sus archivos .png. En mi caso:

C:\Users\Carlos\Documents\Visual Studio 2015\Projects\WindowsApplication1\WindowsApplication1\bin\Debug\marker_icons

Creé dos botones en mi Form1 para ilustrar cómo interactúan el mapa y el WF. Aquí está cómo se ve:

Form1



Y aquí está el código:

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: no olvide agregar estas líneas antes de la definición de su clase Form1:

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

Lo que hacen es decirle a .NET Framework que queremos plena confianza y hacer que la clase sea visible para COM, de modo que Form1 sea visible para JavaScript.

Además, no olvide esto en su función de carga Form1:

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

Expone su clase Form1 al JavaScript en la página googlemap_template.html.

Ahora puedes ejecutar y debería estar funcionando.

CÓMO FUNCIONA#####

Básicamente, lo que hace nuestra clase GoogleMapHelper es leer nuestro googlemap_template.html, hacer una copia temporal, reemplazar el código relacionado con los marcadores ([[MARKER_DATA]]) y ejecutar la página en el control del navegador web de nuestro formulario. Este html recorre todos los marcadores y asigna un detector de "clic" a cada uno. Esta función de clic es obviamente completamente personalizable. En el ejemplo, abre una ventana de información si el marcador tiene un título, centra el mapa en dicho marcador y llama a dos funciones externas que están definidas en nuestra clase Form1.

Por otro lado, podemos definir otras funciones de javascript (con o sin argumentos) en este html para llamar desde nuestro Formulario de Windows (usando wbmap.Document.InvokeScript).

Lea [Google Maps en un formulario de Windows en línea: https://riptutorial.com/es/vb-net/topic/5903/google-maps-en-un-formulario-de-windows](https://riptutorial.com/es/vb-net/topic/5903/google-maps-en-un-formulario-de-windows)

Capítulo 21: Introducción a la sintaxis

Examples

Comentarios

Lo primero que se debe saber es cómo escribir comentarios.

En VB .NET, escribe un comentario escribiendo un apóstrofe o escribiendo `REM`. Esto significa que el resto de la línea no será tomada en cuenta por el compilador.

```
'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
```

Ayudante Intellisense

Una cosa interesante es la posibilidad de agregar sus propios comentarios en Visual Studio Intellisense. Para que pueda hacer que sus propias funciones y clases escritas se expliquen por sí mismas. Para hacerlo, debe escribir el símbolo de comentario tres veces la línea sobre su función.

Una vez hecho esto, Visual Studio agregará automáticamente una documentación 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
```

Después de eso, si escribe su función de Prueba en algún lugar de su código, aparecerá esta pequeña ayuda:

```
Test(
  Test(Name As String) As String
  This function returns a hello to your name
  Name: Your Name
```

Declarar una variable

En VB.NET, todas las variables deben declararse antes de ser utilizadas (si `Option Explicit` está establecido en `On`). Hay dos formas de declarar variables:

- Dentro de una `Function` o un `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
```

Consulte [esta respuesta](#) para obtener detalles completos sobre `Option Explicit` , `Strict` and `Infer`

- Dentro de una `Class` o un `Module` :

Estas variables (también llamadas campos en este contexto) serán accesibles para cada instancia de la `Class` que se declaren. Podrían ser accesibles desde fuera de la `Class` declarada en función del modificador (`Public` , `Private` , `Protected` , `Protected Friend` o `Friend Protected 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
```

Estos campos también se pueden declarar con `Dim` pero el significado cambia según el tipo adjunto:

```
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
```

Modificadores

Los modificadores son una forma de indicar cómo los objetos externos pueden acceder a los datos de un objeto.

- Público

Significa que cualquier objeto puede acceder a esto sin restricciones.

- Privado

Significa que solo el objeto declarante puede acceder y ver esto

- Protegido

Significa que solo el objeto declarante y cualquier objeto que se hereda de él pueden acceder y ver esto.

- Amigo

Significa solo que el objeto delecado, cualquier objeto que se herede de él y cualquier objeto en el mismo espacio de nombres puede acceder y ver esto.

```
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
```

Escribiendo una función

Una función es un bloque de código que se llamará varias veces durante la ejecución. En lugar de escribir el mismo fragmento de código una y otra vez, uno puede escribir este código dentro de una función y llamar a esa función cuando sea necesario.

Una función :

- Debe ser declarado en una *clase* o en un *módulo*.
- Devuelve un valor (especificado por el tipo de retorno)
- Tiene un *modificador*
- Puede tomar parámetros para hacer su procesamiento.

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

Un Nombre de Función, podría ser usado como la declaración de retorno

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

    If rbSealBarType.Selected.Value = "" Then
        err = True
    End If

    Return err
End Function
```

es lo mismo que

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

    If rbSealBarType.Selected.Value = "" Then
```

```

        sealBarTypeValidation = True
    End If

End Function

```

Inicializadores de objetos

- Tipos nombrados

```

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

```

Es equivalente a

```

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

```

- Tipos anónimos (*la opción Inferir debe estar activada*)

```

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

```

Aunque una `anonymousInstance` similar no tiene el mismo tipo que `someInstance`

El nombre de miembro debe ser único en el tipo anónimo y se puede tomar de una variable u otro nombre de miembro de objeto

```

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

```

Cada miembro puede ir precedido por la palabra `Key` clave. Esos miembros serán `ReadOnly` propiedades, los que no se leerá propiedades / escritura

```

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

```

Dos instancias anónimas definidas con los mismos miembros (nombre, tipo, presencia de `Key` y orden) tendrán el mismo tipo anónimo.

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

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

Los tipos anónimos son estructuralmente comparables. Dos instancias de los mismos tipos anónimos que tengan al menos una propiedad de `Key` con los mismos valores de `Key` serán iguales. Tienes que usar el método `Equals` para probarlo, usando `=` no compilará y `Is` comparará la referencia del objeto.

```
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
```

Tanto el inicializador de tipos anónimo como el anónimo se pueden anidar y mezclar

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

Inicializador de colección

- Arrays

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

- Contenedores (`List(Of T)` , `Dictionary(Of TKey, TValue)` , etc.)

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

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

Es equivalente a

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

Los elementos pueden ser el resultado de un constructor, una llamada de método, un acceso de propiedad. También se puede mezclar con inicializador de objetos.

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

No es posible utilizar la sintaxis del inicializador de objetos **Y** la sintaxis del inicializador de colecciones para el mismo objeto al mismo tiempo. Por ejemplo, estos **no** funcionarán.

```
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 personalizado

También podemos permitir la sintaxis de inicialización de colección proporcionando un tipo personalizado.

Debe implementar `IEnumerable` y tener un método `Add` (método de instancia, compartido o incluso extensión) accesible y compatible mediante reglas de sobrecarga.

Ejemplo elaborado:

```
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
End Class
```



```

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"
    '...
}

```

Si quisiéramos agregar un objeto `Person` a una `List(Of Person)` simplemente colocando el nombre en el inicializador de la colección (pero no podemos modificar la clase `List(Of Person)`) podemos usar un método de Extensión

```

' 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"
}

```

Lea Introducción a la sintaxis en línea: <https://riptutorial.com/es/vb-net/topic/3997/introduccion-a-la-sintaxis>

Capítulo 22: Las clases

Introducción

Una clase agrupa diferentes funciones, métodos, variables y propiedades, que se llaman sus miembros. Una clase encapsula los miembros, a los que se puede acceder mediante una instancia de la clase, llamada un objeto. Las clases son extremadamente útiles para el programador, ya que hacen que la tarea sea cómoda y rápida, con características como modularidad, reutilización, facilidad de mantenimiento y legibilidad del código.

Las clases son los componentes básicos de los lenguajes de programación orientados a objetos.

Examples

Creando clases

Las clases proporcionan una forma de crear sus propios tipos dentro del marco .NET. Dentro de una definición de clase puede incluir lo siguiente:

- Campos
- Propiedades
- Métodos
- Constructores
- Eventos

Para declarar una clase usas la siguiente sintaxis:

```
Public Class Vehicle
End Class
```

Otros tipos de .NET se pueden encapsular dentro de la clase y exponerse en consecuencia, como se muestra a continuación:

```
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
```

Clases abstractas

Si las clases comparten una funcionalidad común, puede agrupar esto en una clase base o abstracta. Las clases abstractas pueden contener una implementación parcial o nula y permiten que el tipo derivado anule la implementación base.

Las clases abstractas dentro de VisualBasic.NET deben declararse como `MustInherit` y no se pueden crear instancias.

```
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 subtipo puede `inherit` esta clase abstracta como se muestra a continuación:

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

El automóvil heredará todos los tipos declarados dentro del vehículo, pero solo puede acceder a ellos según el modificador de acceso subyacente.

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

En el ejemplo anterior se crea una nueva instancia de `Car`. Luego se invoca el método `DisplayWheelCount()` que llamará a la implementación de `Vehicle` clase base.

Lea Las clases en línea: <https://riptutorial.com/es/vb-net/topic/3522/las-clases>

Capítulo 23: Leyendo el archivo de texto comprimido sobre la marcha

Examples

Leyendo archivo de texto .gz línea tras línea

Esta clase abre un archivo .gz (formato habitual de archivos de registro comprimidos) y devolverá una línea en cada llamada de `.NextLine()`

No hay uso de memoria para descompresión temporal, muy útil para archivos grandes.

```
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: no hay seguridad a prueba de fallos, para fines de legibilidad.

Lea [Leyendo el archivo de texto comprimido sobre la marcha en línea](https://riptutorial.com/es/vb-net/topic/6960/leyendo-el-archivo-de-texto-comprimido-sobre-la-marcha):

<https://riptutorial.com/es/vb-net/topic/6960/leyendo-el-archivo-de-texto-comprimido-sobre-la-marcha>

Capítulo 24: LINQ

Introducción

LINQ (Language Integrated Query) es una expresión que recupera datos de un origen de datos. LINQ simplifica esta situación al ofrecer un modelo consistente para trabajar con datos a través de varios tipos de fuentes de datos y formatos. En una consulta LINQ, siempre está trabajando con objetos. Utiliza los mismos patrones de codificación básicos para consultar y transformar datos en documentos XML, bases de datos SQL, conjuntos de datos ADO.NET, colecciones .NET y cualquier otro formato para el que esté disponible un proveedor LINQ.

Examples

Proyección

```
' 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)
```

Podemos proyectar múltiples resultados a la vez también

```
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})
```

Seleccionando de matriz con condición simple

```
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"
```

La consulta será un objeto enumerable que contiene `Stack Overflow` y `Super User`. `x` en la consulta es una variable iterativa donde se almacenará cada objeto verificado por la cláusula `Where`.

Mapeo de matriz por cláusula de selección

```
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

```

El resultado de la consulta será un objeto enumerable que contiene longitudes de cadenas en la matriz de entrada. En este ejemplo, estos serían los valores 14, 10, 10, 24. x en la consulta es una variable iterativa donde se almacenará cada objeto de la matriz de entrada.

Orden de salida

```

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 cláusula `OrderBy` ordena la salida por el valor devuelto por la cláusula. En este ejemplo es la longitud de cada cadena. El orden de salida predeterminado es ascendente. Si necesita descender puede especificar una palabra clave `Descending` después de la cláusula.

```

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

```

Generando diccionario desde 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

```

Obtención de valores distintos (utilizando el método 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"
```

Lea LINQ en línea: <https://riptutorial.com/es/vb-net/topic/3111/linq>

Capítulo 25: Liza

Sintaxis

- Lista.Agregar (artículo como tipo)
- List.RemoveRange (index As Integer, count As Integer)
- List.Remove (index As Integer)
- List.AddRange (colección)
- List.Find (coinciden como Predicado (de cadena))
- List.Insert (index como Integer, item como Type)
- List.Contains (artículo como tipo)

Examples

Crear una lista

Las listas se pueden rellenar con cualquier tipo de datos según sea necesario, con el formato

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

Por ejemplo:

Crear una nueva lista vacía de cadenas

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

Crear una nueva lista de cadenas y rellenar con algunos datos

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:

Si está recibiendo lo siguiente cuando se ejecuta el código:

Referencia a objeto no establecida como instancia de un objeto.

Asegúrese de declarar como `New` es decir, `Dim aList as New List(Of String)` o si declara sin el `New`, asegúrese de establecer la lista en una nueva lista - `Dim aList as List(Of String) = New List(Of String)`

Añadir elementos a una lista

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

Para agregar más de un elemento a la vez, use **AddRange**. Siempre se agrega al final de la lista.

```
Dim bList as New List(of Integer)
bList.AddRange(aList)

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

Para agregar elementos al centro de la lista, use **Insertar**

Insertar colocará el elemento en el índice y volverá a numerar los elementos restantes.

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

Nueva salida:

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

Eliminar elementos de 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()
```

Recuperar elementos de una lista

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

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

output :

Hola Mundo

Si no conoce el índice del elemento o solo conoce parte de la cadena, use el método **Find** o **FindAll**

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

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

output :

Hola Mundo

El método **FindAll** devuelve una nueva lista (de cadena)

```
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"))
```

salida (0) = "Hola, Prueba"

salida (1) = "Prueba"

Bucle a través de los elementos en la 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 la siguiente salida:

```
one
two
three
```

Otra opción, sería recorrer en bucle utilizando el índice de cada 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
```

Compruebe si el artículo existe en 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 la siguiente salida:

```
False
True
False
```

Lea Liza en línea: <https://riptutorial.com/es/vb-net/topic/3636/liza>

Capítulo 26: Los diccionarios

Introducción

Un diccionario representa una colección de claves y valores. Ver [Clase de diccionario MSDN \(Tkey, TValue\)](#) .

Examples

Recorrer un diccionario e imprimir todas las entradas

Cada par en el diccionario es una instancia de `KeyValuePair` con los mismos parámetros de tipo que el diccionario. Cuando recorre el diccionario con `ForEach` , cada iteración le proporcionará uno de los pares clave-valor almacenados en el diccionario.

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

Crear un diccionario lleno de valores.

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

Esto crea un diccionario y lo llena inmediatamente con tres `KeyValuePair`s.

También puede agregar nuevos valores más adelante usando el método `Agregar`:

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

Tenga en cuenta que la clave (el primer parámetro) debe ser única en el diccionario, de lo contrario se lanzará una excepción.

Obtener un valor de diccionario

Puede obtener el valor de una entrada en el diccionario usando la propiedad `'Artículo'`:

```
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"
```

Si la clave no está presente en el diccionario, se lanzará una excepción `KeyNotFoundException`.

Buscando clave ya en el diccionario - reducción de datos

El método `ContainsKey` es la forma de saber si ya existe una clave en el Diccionario.

Esto es útil para la reducción de datos. En el ejemplo siguiente, cada vez que encontramos una nueva palabra, la agregamos como clave en el diccionario, de lo contrario, incrementamos el contador para esta palabra específica.

```
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
```

Ejemplo de reducción de XML: obtener todos los nombres de los nodos secundarios y la aparición en una rama de 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
```

Lea Los diccionarios en línea: <https://riptutorial.com/es/vb-net/topic/2165/los-diccionarios>

Capítulo 27: Los operadores

Observaciones

Los operadores se utilizan para asignar o comparar valores. Consisten en un solo símbolo o palabra clave y, por lo general, están intercalados entre un valor izquierdo y uno derecho. Por ejemplo: `right = left` .

Los operadores son intrínsecos al lenguaje (como `=`) y no a funciones como las que proporciona `System.Math`.

Examples

Comparación

Los operadores de comparación comparan dos valores y le devuelven un valor booleano (`True` o `False`) como resultado.

Igualdad

- El signo igual `=` se usa tanto para la comparación como para la asignación de igualdad.

```
If leftValue = rightValue Then ...
```

Desigualdad

- El nido del soporte de ángulo izquierdo al soporte de ángulo derecho `<>` realiza una comparación desigual.

```
If leftValue <> rightValue Then ...
```

Mas grande que

- El corchete angular izquierdo `<` realiza una comparación mayor.

```
If leftValue < rightValue Then ...
```

Mayor que o igual

- El nido de signo igual al corchete de ángulo izquierdo `=>` realiza una comparación mayor o igual que.

```
If leftValue => rightValue Then ...
```

Menos que

- El corchete de ángulo recto `>` realiza una comparación menor.

```
If leftValue > rightValue Then ...
```

Menor o igual

- El nido del signo igual al corchete de ángulo derecho => realiza una comparación mayor o igual que.

```
If leftValue => rightValue Then ...
```

Me gusta

- El operador `Like` prueba la igualdad de una cadena y un patrón de búsqueda.
- El operador `Me gusta` confía en la [Opción Comparar Declaración](#)
- La siguiente tabla enumera los patrones disponibles. Fuente: <https://msdn.microsoft.com/en-us/library/swf8kaxw.aspx> (sección de comentarios)

| Personajes en el <i>patrón</i> | Partidos en la <i>cadena</i> |
|--------------------------------|---|
| ? | Cualquier caracter |
| * | Cero o mas personajes |
| # | Cualquier dígito (0 - 9) |
| [charlist] | Cualquier carácter individual en <i>charlist</i> |
| [! charlist] | Cualquier sola no carácter en <i>lista_caracteres</i> |

- Ver más información sobre [MSDN](#) en la sección de comentarios.

```
If string Like pattern Then ...
```

Asignación

Hay un solo operador de asignación en VB.

- El signo igual = se usa tanto para la comparación como para la asignación de igualdad.

```
Dim value = 5
```

Notas

Cuidado con la comparación de la asignación vs igualdad.

```
Dim result = leftValue = rightValue
```

En este ejemplo, puede ver el signo igual que se utiliza como operador de comparación y como operador de asignación, a diferencia de otros idiomas. En este caso, el `result` será de tipo `Boolean` y contendrá el valor de la comparación de igualdad entre `leftValue` y `rightValue`.

Relacionado: [Usar Option Strict On para declarar variables correctamente](#)

Mates

Si tienes las siguientes variables

```
Dim leftValue As Integer = 5
```

```
Dim rightValue As Integer = 2
Dim value As Integer = 0
```

Adición realizada por el signo más + .

```
value = leftValue + rightValue

'Output the following:
'7
```

Resta realizada por el signo menos - .

```
value = leftValue - rightValue

'Output the following:
'3
```

Multiplicación realizada por el símbolo de estrella * .

```
value = leftValue * rightValue

'Output the following:
'10
```

División realizada por el símbolo de barra diagonal / .

```
value = leftValue / rightValue

'Output the following:
'2.5
```

División entera Realizada por el símbolo de barra invertida \ .

```
value = leftValue \ rightValue

'Output the following:
'2
```

Módulo realizado por la palabra clave Mod .

```
value = leftValue Mod rightValue

'Output the following:
'1
```

Elevar a un Poder de Realizado por el símbolo ^ .

```
value = leftValue ^ rightValue

'Output the following:
'25
```


Ampliación y estrechamiento

Necesita edición.

Sobrecarga del operador

Necesita edición.

Bitwise

Estos son los operadores bitwise en VB.NET: Y, O, Xor, Not

Ejemplo de Y operación bitwise

```
Dim a as Integer
a = 3 And 5
```

El valor de a será 1. El resultado se obtiene después de comparar 3 y 5 en binario para. 3 en forma binaria es 011 y 5 en forma binaria es 101. El operador And coloca 1 si ambos bits son 1. Si cualquiera de los bits es 0, el valor será 0

```
3 And 5 will be  011
                  101
                  ---
                  001
```

Entonces, el resultado binario es 001 y cuando se convierte a decimal, la respuesta será 1.

O el operador coloca 1 si ambos o un bit es 1

```
3 Or 5 will be  011
                 101
                 ---
                 111
```

El operador Xor coloca 1 si solo uno de los bits es 1 (no ambos)

```
3 Xor 5 will be 011
                 101
                 ---
                 110
```

El operador no revierte los bits incluyendo el signo.

```
Not 5 will be - 010
```

Concatenación de cuerdas

La concatenación de cadenas se produce cuando se combinan dos o más cadenas en una sola

variable de cadena.

La concatenación de cadenas se realiza con el símbolo & .

```
Dim one As String = "Hello "  
Dim two As String = "there"  
Dim result As String = one & two
```

Los valores que no son de cadena se convertirán en cadena cuando se use & .

```
Dim result as String = "2" & 10 ' result = "210"
```

Siempre use & (ampersand) para realizar la concatenación de cadenas.

No hagas esto

Si bien es posible, en el *más simple* de los casos, usar el símbolo + para hacer la concatenación de cadenas, nunca debe hacer esto. Si un lado del símbolo más no es una cadena, cuando la opción estricta está desactivada, el comportamiento no es intuitivo, cuando está activada la opción estricta producirá un error de compilación. Considerar:

```
Dim value = "2" + 10 ' result = 12 (data type Double)  
Dim value = "2" + "10" ' result = "210" (data type String)  
Dim value = "2g" + 10 ' runtime error
```

El problema aquí es que si el operador + ve cualquier operando que sea de tipo numérico, supondrá que el programador quería realizar una operación aritmética e intentar convertir el otro operando al tipo numérico equivalente. En los casos en que el otro operando es una cadena que contiene un número (por ejemplo, "10"), la cadena se *convierte en un número* y luego se agrega *aritméticamente* al otro operando. Si el otro operando no se puede convertir a un número (por ejemplo, "2g"), la operación se bloqueará debido a un error de conversión de datos. El operador + solo realizará la concatenación de cadenas si *ambos* operandos son de tipo `String` .

Sin embargo, el operador & está diseñado para la concatenación de cadenas y convertirá los tipos que no sean de cadenas en cadenas.

Lea Los operadores en línea: <https://riptutorial.com/es/vb-net/topic/3257/los-operadores>

Capítulo 28: Manejo de archivos

Sintaxis

- `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

Escribir datos en un archivo

Para escribir el contenido de una cadena en un archivo:

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

`WriteAllText` abrirá el archivo especificado, escribirá los datos y luego cerrará el archivo. Si el archivo de destino existe, se sobrescribe. Si el archivo de destino no existe, se crea.

Para escribir el contenido de una matriz en un archivo:

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

`WriteAllLines` abrirá el archivo especificado, escribirá cada valor de la matriz en una nueva línea y luego cerrará el archivo. Si el archivo de destino existe, se sobrescribe. Si el archivo de destino no existe, se crea.

Leer todos los contenidos de un archivo

Para leer el contenido de un archivo en una variable de cadena:

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

`ReadAllText` abrirá el archivo especificado, leerá los datos hasta el final y luego cerrará el archivo.

Para leer un archivo, separándolo en un elemento de matriz para cada línea:

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

`ReadAllLines` abrirá el archivo especificado, leerá cada línea del archivo en un nuevo índice en una matriz hasta el final del archivo, luego cerrará el archivo.

Escribir líneas individualmente en un archivo de texto usando `StreamWriter`

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

El uso de un bloque de `Using` es una buena práctica recomendada cuando se utiliza un objeto que implementa `IDisposable`

Lea Manejo de archivos en línea: <https://riptutorial.com/es/vb-net/topic/2413/manejo-de-archivos>

Capítulo 29: Manejo de errores

Examples

Intenta ... Atrapa ... Finalmente Declaración

Estructura:

```
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
```

Código de ejemplo:

```
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
```

Creando excepciones y lanzamientos personalizados.

Puede crear una excepción personalizada y lanzarlas durante la ejecución de su función. Como práctica general, solo debe lanzar una excepción cuando su función no pueda lograr su funcionalidad definida.

```
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
```

Intenta atrapar en la operación de base de datos

Puede usar Try..Catch para deshacer la operación de la base de datos colocando la instrucción de retrotracción en el segmento de captura.

```
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
```

La excepción irrecuperable

Aunque `Catch ex As Exception` afirma que puede manejar todas las excepciones, hay una excepción (no pretendía hacer un juego de palabras).

```
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
```

¡Vaya! Hay una excepción `System.StackOverflowException` capturada, ¡mientras que la consola ni siquiera imprimió nada! Según [MSDN](https://msdn.microsoft.com/en-us/library/ee417614.aspx),

A partir de .NET Framework 2.0, no puede capturar un objeto `StackOverflowException` con un bloque try / catch, y el proceso correspondiente finaliza de forma

predeterminada. En consecuencia, debe escribir su código para detectar y evitar un desbordamiento de pila.

Por lo tanto, `System.StackOverflowException` no se puede capturar. ¡Cuidado con eso!

Excepciones críticas

En general, la mayoría de las excepciones no son tan importantes, pero hay algunas excepciones realmente serias que tal vez no pueda manejar, como la famosa excepción

`System.StackOverflowException`. Sin embargo, hay otros que pueden quedar ocultos por `Catch ex As Exception`, como `System.OutOfMemoryException`, `System.BadImageFormatException` y

`System.InvalidProgramException`. Es una buena práctica de programación omitirlos si no puede manejarlos correctamente. Para filtrar estas excepciones, necesitamos un método auxiliar:

```
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
```

Lea Manejo de errores en línea: <https://riptutorial.com/es/vb-net/topic/4232/manejo-de-errores>

Capítulo 30: Manejo de la conexión

Examples

Propiedad de conexión pública

```
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
```

Lea Manejo de la conexión en línea: <https://riptutorial.com/es/vb-net/topic/6398/manejo-de-la-conexion>

Capítulo 31: Metodos de extension

Observaciones

Los métodos de extensión son métodos (`Sub` o `Function`) que agregan funcionalidad a un Tipo (que puede ser un Tipo de Referencia o un Tipo de Valor). Estos tipos pueden o no ser propiedad de usted.

Pueden o no estar en el mismo ensamblaje que el Tipo que están diseñados para modificar. Puede permitir una suscripción a sus métodos de extensión aislándolos en su propio espacio de nombres. O si lo prefiere, puede hacer que estén siempre disponibles incluyéndolos en el mismo espacio de nombres que el Tipo que modifican (suponiendo que todas las referencias de ensamblaje estén en su lugar y sean correctas). Vea el proyecto Entity Framework Core 1.0 en GitHub para ver un buen ejemplo del estilo opt-in de los métodos de extensión.

Los métodos de extensión en VB tienen algunos requisitos:

- Los métodos de extensión solo pueden ser declarados en módulos.
- Los métodos de extensión deben estar decorados con el atributo `Extension()` .
- El espacio de nombres del atributo de extensión debe estar disponible dentro de su módulo.
`Imports System.Runtime.CompilerServices`
- El primer parámetro del método debe ser de un tipo al que se adjuntará este método.
- El primer parámetro del método representará la instancia en la que este método opera. (Equivalente a `Me` si este fuera un método de instancia real).
- Se puede llamar a un método de extensión como un método regular al proporcionar todos los parámetros si no se llama en el objeto instanciado.

Examples

Creando un método de extensión

Los métodos de extensión son útiles para extender el comportamiento de las bibliotecas que no poseemos.

Se utilizan de forma similar a los métodos de instancia gracias al azúcar sintáctico del compilador:

```
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

```

Para tener un método de extensión utilizable, el método necesita el atributo `Extension` y debe declararse en un `Module` .

Haciendo el lenguaje más funcional con métodos de extensión.

Un buen uso del método de extensión es hacer que el lenguaje sea más funcional.

```

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

```

Numéricos de relleno

```

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

```

Obtención de la versión de montaje de nombre fuerte

Ejemplo de llamar a un método de extensión como una extensión y como un método regular.

```

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

```

El método de extensión en un módulo. Haga que el módulo sea público si las extensiones se compilan en una dll y se hará referencia en otro conjunto.

```

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

```

Lea Metodos de extension en línea: <https://riptutorial.com/es/vb-net/topic/1592/metodos-de-extension>

Capítulo 32: Multihilo

Examples

Subprocesamiento múltiple utilizando Thread Class

Este ejemplo utiliza la clase `Thread`, pero las aplicaciones multiproceso también se pueden hacer usando `BackgroundWorker`. Las `AddNumber`, `SubstractNumber` y `DivideNumber` se ejecutarán mediante subprocesos separados:

Edición: ahora el subproceso de la interfaz de usuario espera a que finalicen los subprocesos secundarios y muestra el resultado.

```
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
        End While
    End Sub
End Module
```

```

        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 SubtractNumber()
    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

```

Lea Multihilo en línea: <https://riptutorial.com/es/vb-net/topic/6756/multihilo>

Capítulo 33: Objetos desechables

Examples

Concepto básico de IDisposable.

Cada vez que crea una instancia de una clase que implemente `IDisposable`, debe llamar a `.Dispose` ¹ en esa clase cuando haya terminado de usarla. Esto permite que la clase limpie las dependencias administradas o no administradas que pueda estar usando. No hacer esto podría causar una pérdida de memoria.

La palabra clave `Using` garantiza que se `.Dispose`, sin que tenga que llamarlo *explícitamente*.

Por ejemplo, sin `Using`:

```
Dim sr As New StreamReader("C:\foo.txt")
Dim line = sr.ReadLine
sr.Dispose()
```

Ahora con el `Using`:

```
Using sr As New StreamReader("C:\foo.txt")
    Dim line = sr.ReadLine
End Using 'Dispose is called here for you
```

Una de las principales ventajas que `Using` tiene es cuando se lanza una excepción, porque *garantiza que se* `.Dispose`.

Considera lo siguiente. Si se produce una excepción, debe recordar llamar `.Dispose`, pero es posible que también deba verificar el estado del objeto para asegurarse de que no reciba un error de referencia nulo, etc.

```
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 bloque de uso significa que no tiene que recordar hacer esto y puede declarar su objeto dentro 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 [¿Siempre tengo que llamar a Dispose \(\) en mis objetos DbContext? No](#)

Declarando más objetos en uno usando

A veces, tienes que crear dos objetos `Disposable` en una fila. Hay una manera fácil de evitar el anidamiento `Using` bloques.

Este código

```
Using File As New FileStream("MyFile", FileMode.Append)
    Using Writer As New BinaryWriter(File)
        'You code here
        Writer.Writer("Hello")
    End Using
End Using
```

se puede acortar en este. La principal ventaja es que ganas un nivel de sangría:

```
Using File As New FileStream("MyFile", FileMode.Append), Writer As New BinaryWriter(File)
    'You code here
    Writer.Writer("Hello")
End Using
```

Lea [Objetos desechables en línea](https://riptutorial.com/es/vb-net/topic/3204/objetos-desechables): <https://riptutorial.com/es/vb-net/topic/3204/objetos-desechables>

Capítulo 34: OOP Keywords

Examples

Definiendo una clase

Las *clases* son aspectos vitales de la POO. Una clase es como el "plano" de un objeto. Un objeto tiene las propiedades de una clase, pero las características no están definidas dentro de la misma clase. Como cada objeto puede ser diferente, definen sus propias características.

```
Public Class Person
End Class

Public Class Customer
End Class
```

Una clase también puede contener *subclases*. Una subclase hereda las mismas propiedades y comportamientos que su clase primaria, pero puede tener sus propias propiedades y clases únicas.

Modificadores de herencia (en clases)

Hereda

Especifica la clase base (o padre)

```
Public Class Person
End Class

Public Class Customer
    Inherits Person

End Class

'One line notation
Public Class Student : Inherits Person
End Class
```

Posibles objetos:

```
Dim p As New Person
Dim c As New Customer
Dim s As New Student
```

No Heredable

Evita que los programadores utilicen la clase como clase base.

```
Public NotInheritable Class Person
End Class
```

Posibles objetos:

```
Dim p As New Person
```

Debe Heredarse

Especifica que la clase está destinada para ser utilizada como una clase base solamente. (Clase abstracta)

```
Public MustInherit Class Person
End Class

Public Class Customer
    Inherits Person
End Class
```

Posibles objetos:

```
Dim c As New Customer
```

Modificadores de herencia (en propiedades y métodos)

Anulable

Permite que una propiedad o un método en una clase se invalide en una clase derivada.

```
Public Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class
```

Anulaciones

Anula una propiedad o un método anulable definido en la clase base.

```
Public Class Customer
    Inherits Person

    'Base Class must be Overridable
    Public Overrides Sub DoSomething()
```

```
        Console.WriteLine("Customer")
    End Sub
End Class
```

NotOverridable

Evita que una propiedad o un método se sobrescriba en una clase heredada. Comportamiento por defecto. Solo se puede declarar en **métodos de anulaciones**

```
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
```

Ejemplo de uso:

```
Dim p As New Person
p.DoSomething()

Dim c As New Customer
c.DoSomething()

Dim d As New DetailedCustomer
d.DoSomething()
```

Salida:

```
Person
Customer
Customer
```

MustOverride

Requiere que una clase derivada anule la propiedad o el método.

Los métodos MustOverride se deben declarar en **las clases MustInherit**.

```
Public MustInherit Class Person

    Public MustOverride Sub DoSomething()
        'No method definition here
    End Sub

End Class

Public Class Customer
    Inherits Person

    'DoSomething must be overridden
    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Ejemplo de uso:

```
Dim c As New Customer
c.DoSomething()
```

Salida:

```
Customer
```

MyBase

La palabra clave MyBase se comporta como una variable de objeto que se refiere a la clase base de la instancia actual de una clase.

```
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
```

Ejemplo de uso:

```
Dim p As New Person
p.DoSomething()
```

```
Console.WriteLine("----")

Dim c As New Customer
c.DoSomething()
c.DoSomethingElse()
```

Salida:

```
Person
----
Person
Person
```

Yo vs myclass

Yo usa la instancia de objeto actual.

MyClass usa la definición de miembro en la clase donde se llama al miembro

```
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
```

Ejemplo de uso:

```
Dim c As New Customer
c.useMe()
c.useMyClass()
```

Salida:

```
Customer
Person
```

Sobrecarga

La sobrecarga es la creación de más de un procedimiento, constructor de instancia o propiedad en una clase con el mismo nombre pero diferentes tipos de argumentos.

```
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
```

Oscuridad

Se vuelve a declarar un miembro que no es anulable. Sólo se verán afectadas las llamadas a la instancia. El código dentro de las clases base no se verá afectado por esto.

```
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
```

Ejemplo de uso:

```
Dim p As New Person
Dim c As New Customer
p.UseMe()
c.UseMe()
Console.WriteLine("----")
p.DoSomething()
c.DoSomething()
```

Salida:

```
Person
Person
----
```

```
Person
Customer
```

Escollos :

Ejemplo 1, Creando un nuevo objeto a través de un genérico. ¿Qué función se utilizará?

```
Public Sub CreateAndDoSomething(Of T As {Person, New}) ()
    Dim obj As New T
    obj.DoSomething()
End Sub
```

ejemplo de uso:

```
Dim p As New Person
p.DoSomething()
Dim s As New Student
s.DoSomething()
Console.WriteLine("----")
CreateAndDoSomething(Of Person) ()
CreateAndDoSomething(Of Student) ()
```

Salida: Por intuición el resultado debe ser el mismo. Sin embargo, eso no es cierto.

```
Person
Student
----
Person
Person
```

Ejemplo 2:

```
Dim p As Person
Dim s As New Student
p = s
p.DoSomething()
s.DoSomething()
```

Salida: Por intuición, podrías pensar que p y s son iguales y se comportarán igual. Sin embargo, eso no es cierto.

```
Person
Student
```

En estos simples ejemplos, es fácil aprender el extraño comportamiento de las Sombras. Pero en la vida real trae muchas sorpresas. Es aconsejable evitar el uso de sombras. Uno debería usar otras alternativas tanto como sea posible (anulaciones, etc.)

Interfaces

```
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
```

Lea OOP Keywords en línea: <https://riptutorial.com/es/vb-net/topic/4273/oop-keywords>

Capítulo 35: Opción Estricta

Sintaxis

- Opción Estricta {On | Apagado }

Observaciones

`Option Strict On` es una buena práctica recomendada con Visual Basic .Net. Le ayuda como desarrollador a producir un código más limpio, más estable, más libre de errores y más fácil de mantener. En algunos casos, también puede ayudarlo a escribir programas con mejor rendimiento, evitando cosas como la conversión implícita.

`On no` es la configuración predeterminada para una nueva instalación de Visual Studio. Debe ser uno de los primeros cambios antes de comenzar la programación si va a utilizar VB.NET. La razón por la que no es la configuración predeterminada proviene de las primeras ediciones de Visual Studio cuando se esperaba que los programadores estuvieran migrando proyectos desde VB6.

Examples

¿Por qué usarlo?

`option Strict On` evita que ocurran tres cosas:

1. Errores de conversión de reducción implícita

Le impide asignar a una variable que tenga *menos precisión o menor capacidad* (una conversión de reducción) sin una conversión explícita. Hacerlo daría lugar a la pérdida de datos.

```
Dim d As Double = 123.4
Dim s As Single = d 'This line does not compile with Option Strict On
```

2. Últimas llamadas vinculantes

No se permite la vinculación tardía. Esto es para evitar errores tipográficos que se compilan, pero fallan en tiempo de ejecución

```
Dim obj As New Object
obj.Foo 'This line does not compile with Option Strict On
```

3. Errores de tipo de objeto implícito

Esto evita que la variable se infiera como un Objeto cuando de hecho deberían haberse declarado como un tipo


```
Dim something = Nothing. 'This line does not compile with Option Strict On
```

Conclusión

A menos que necesite realizar un enlace tardío, siempre debe tener la `Option Strict On` ya que hará que los errores mencionados generen errores de tiempo de compilación en lugar de excepciones de tiempo de ejecución.

Si usted *tiene* que hacer el enlace en *tiempo*, puede

- Envuelva todas sus llamadas de enlace tardías en una clase / módulo y use `Option Strict Off` en la parte superior del archivo de código (este es el método preferido ya que reduce la posibilidad de errores tipográficos en otros archivos), o
- Especifique que el enlace en tiempo de ejecución no provoca un error de compilación (`Project Properties > Compile Tab > Warning Configuration`)

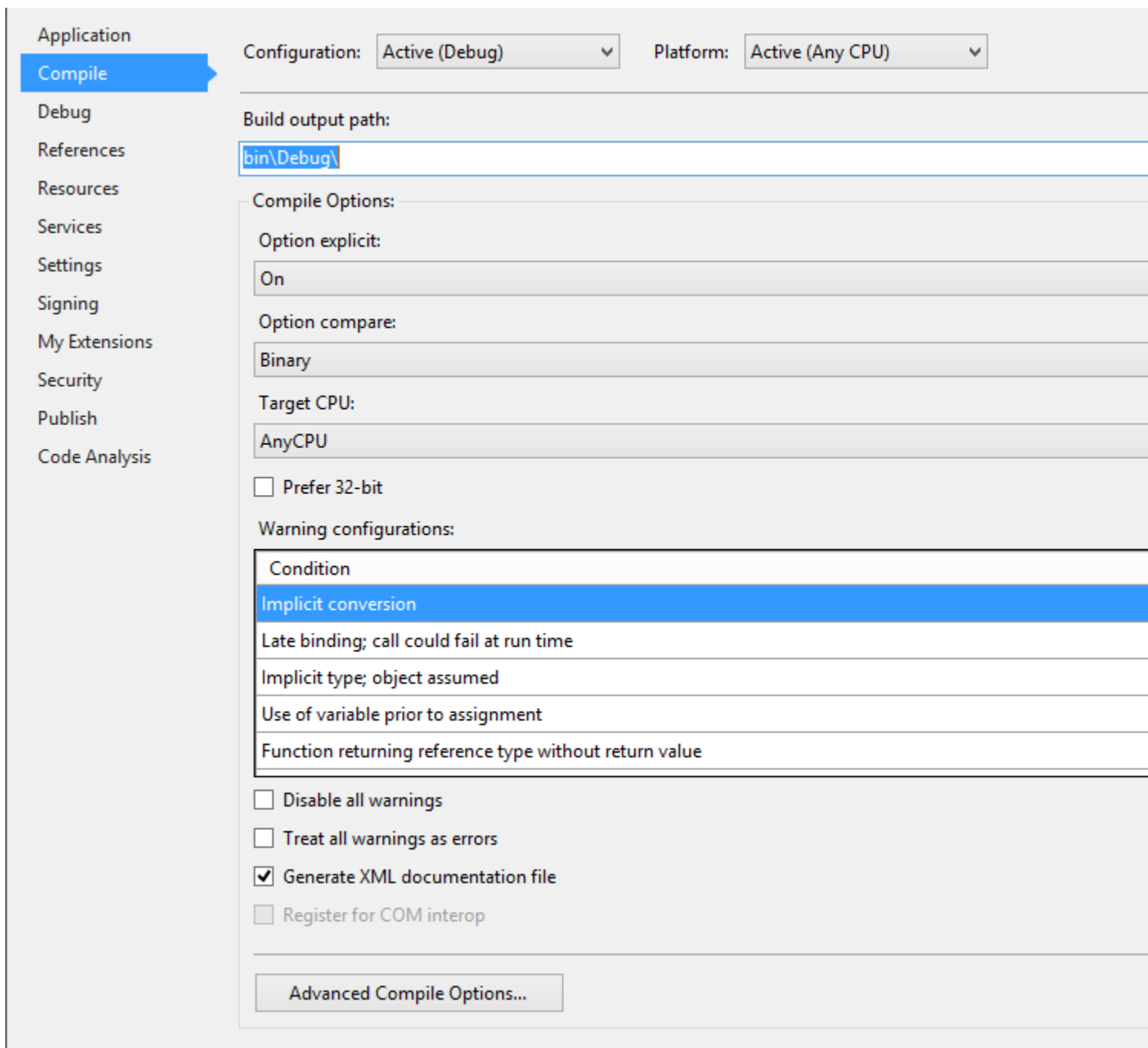
Cómo encenderlo

- Puede activarlo en el Módulo / Nivel de clase colocando la directiva en la parte superior del archivo de código.

```
Option Strict On
```

- Puede activarlo a nivel de proyecto a través del menú en Visual Studio

Proyecto > Propiedades de [Proyecto] > Ficha Compilar > Opción estricta > Activar



- Puede activarlo de forma predeterminada para todos los proyectos nuevos seleccionando:

Herramientas> Opciones> Proyectos y soluciones> Valores predeterminados de VB>
Opción estricta
Póngalo en On .

Lea Opción Estricta en línea: <https://riptutorial.com/es/vb-net/topic/4022/opcion-estricta>

Capítulo 36: Opción explícita

Observaciones

`Option Explicit On` es una buena práctica recomendada con Visual Basic .Net. Le ayuda como desarrollador a producir un código más limpio, más estable, más libre de errores y más fácil de mantener. En algunos casos, también puede ayudarlo a escribir programas con mejor rendimiento.

con ref a <https://support.microsoft.com/en-in/kb/311329#bookmark-3> también se puede utilizar la opción estricta en lugar de la opción explícita. Opción estricta hereda la opción explícita.

Examples

¿Qué es?

Te obliga a declarar explícitamente todas las variables.

¿Cuál es la diferencia entre declarar explícitamente y declarar implícitamente una variable?

Declarando explícitamente una variable:

```
Dim anInteger As Integer = 1234
```

Declarando implícitamente una variable:

```
'Did not declare aNumber using Dim  
aNumber = 1234
```

Conclusión

Por lo tanto, siempre debe tener `Option Explicit On` ya que podría escribir mal una variable durante la asignación, lo que hace que su programa se comporte de manera inesperada.

¿Cómo encenderlo?

Nivel de documento

Está `Option Explicit On` de forma predeterminada, pero puede tener una capa adicional de protección colocando `Option Explicit On` en la parte superior del archivo de código. La opción se aplicará a todo el documento.

Nivel de proyecto

Puede activarlo a través del menú en Visual Studio:

Proyecto> Propiedades de [Proyecto]> Ficha Compilar> Opción explícita

Elija en el menú desplegable. La opción se aplicará a todo el documento.

Todos los nuevos proyectos

Puede activarlo de forma predeterminada para todos los proyectos nuevos seleccionando:

Herramientas> Opciones> Proyectos y soluciones> Valores predeterminados de VB>
Opción explícita

Elija en el menú desplegable.

Lea **Opción explícita** en línea: <https://riptutorial.com/es/vb-net/topic/4725/opcion-explicita>

Capítulo 37: Opcion Inferir

Examples

¿Qué es?

Habilita el uso de inferencia de tipo local en la declaración de variables.

¿Qué es la inferencia de tipos?

Puede declarar variables locales sin indicar explícitamente un tipo de datos. El compilador infiere el tipo de datos de una variable del tipo de su expresión de inicialización.

Opción Inferir En :

```
Dim aString = "1234" '--> Will be treated as String by the compiler
Dim aNumber = 4711 '--> Will be treated as Integer by the compiler
```

Declaración de tipo explícita vs.

```
'State a type explicitly
Dim aString as String = "1234"
Dim aNumber as Integer = 4711
```

Opción Inferir Off:

El comportamiento del compilador con `Option Infer Off` depende de la configuración de `Option Strict` que ya está [documentada aquí](#) .

- **Opción Inferir desactivada - Opción estricta desactivada**

Todas las variables sin declaraciones explícitas de tipo se declaran como `Object` .

```
Dim aString = "1234" '--> Will be treated as Object by the compiler
```

- **Opción Inferir desactivada - Opción estricta activada**

El compilador no le permitirá declarar una variable sin un tipo explícito.

```
'Dim aString = "1234" '--> Will not compile due to missing type in declaration
```

Cómo habilitarlo / deshabilitarlo

Nivel de documento

Está activado de forma predeterminada, pero puede configurarlo colocando `Option Infer On|Off` en la parte superior del archivo de código. La opción se aplicará a todo el documento.

Nivel de proyecto

Puede activarlo / desactivarlo a través del menú en Visual Studio:

Proyecto > Propiedades de [Proyecto] > Ficha Compilar > Opción infer

Seleccione `On|Off` en el menú desplegable. La opción se aplicará a todo el documento.

Todos los nuevos proyectos

Puede activarlo de forma predeterminada para todos los proyectos nuevos seleccionando:

Herramientas > Opciones > Proyectos y soluciones > Valores predeterminados de VB > Opción Inferir

Seleccione `On|Off` en el menú desplegable.

Cuándo usar inferencia de tipo

Básicamente puedes usar la inferencia de tipos siempre que sea posible.

Sin embargo, tenga cuidado al combinar `Option Infer Off` y `Option Strict Off`, ya que esto puede llevar a un comportamiento de tiempo de ejecución no deseado:

```
Dim someVar = 5
someVar.GetType.ToString() '--> System.Int32
someVar = "abc"
someVar.GetType.ToString() '--> System.String
```

Tipo Anónimo

Los tipos anónimos **solo se** pueden declarar con `Option Infer On`.

Se usan a menudo cuando se trata de [LINQ](#):

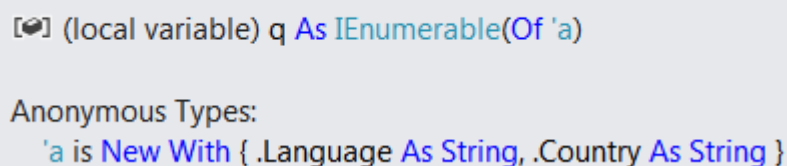
```
Dim countryCodes = New List(Of String)
countryCodes.Add("en-US")
countryCodes.Add("en-GB")
countryCodes.Add("de-DE")
countryCodes.Add("de-AT")

Dim q = From code In countryCodes
        Let split = code.Split("-"c)
        Select New With { .Language = split(0), .Country = split(1) }
```

- **Opción Inferir En**

El compilador reconocerá el tipo anónimo:

```
Dim q = From code In countryCodes
```



```
[?] (local variable) q As IEnumerable(Of 'a)
Anonymous Types:
'a is New With { .Language As String, .Country As String }
```

- **Opción Inferir Off**

El compilador lanzará un error (con la `Option Strict On`)
o considerará `q` como tipo de `object` (con la `Option Strict Off`).
Ambos casos producirán el resultado de que no puede utilizar el tipo anónimo.

Dobles / Decimales

Las variables numéricas con posiciones decimales se deducirán como `Double` por defecto:

```
Dim aNumber = 44.11 '--> Will be treated as type `Double` by the compiler
```

Si se desea otro tipo como `Decimal` el valor que inicializó la variable debe marcarse:

```
Dim mDecimal = 47.11D '--> Will be treated as type `Decimal` by the compiler
```

Lea Opcion Inferir en línea: <https://riptutorial.com/es/vb-net/topic/5095/opcion-inferir>

Capítulo 38: Operadores de cortocircuito (y también - orElse)

Sintaxis

- resultado = expresión1 y también expresión2
- result = expresión1 OrElse expresión2

Parámetros

| Parámetro | Detalles |
|------------|---|
| resultado | Necesario. Cualquier expresión booleana. El resultado es el resultado booleano de comparación de las dos expresiones. |
| expresión1 | Necesario. Cualquier expresión booleana. |
| expresión2 | Necesario. Cualquier expresión booleana. |

Observaciones

'**AndAlso**' y '**OrElse**' son operadores de **ShortCircuiting** que significan que la ejecución es más corta porque el compilador no evalúa todas las expresiones en una comparación booleana si la primera proporciona el resultado deseado.

Examples

Y también 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 del orse

' 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
```

Evitando NullReferenceException

7.0

Si no

```
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
```

Y también

```

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 introdujo el **operador condicional nulo**, lo que permite volver a escribir las funciones de una manera más limpia, imitando el comportamiento de la versión `AndAlso` del ejemplo.

Lea **Operadores de cortocircuito (y también - orElse)** en línea: <https://riptutorial.com/es/vb-net/topic/2509/operadores-de-cortocircuito--y-tambien---orelse->

Capítulo 39: Palabras clave ByVal y ByRef

Examples

Palabra clave byVal

La palabra clave ByVal antes del parámetro del método (o ninguna palabra clave como ByVal se supone de forma predeterminada) dice que el parámetro se enviará de una manera que **no** permita que el método cambie (asigne un nuevo valor) la variable subyacente al parámetro. No impide que se modifique el contenido (o el estado) del argumento si se trata de una clase.

```
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
```

ByRef palabra clave

La palabra clave ByRef antes del parámetro del método dice que el parámetro se enviará de manera que permita al método cambiar (asignar un nuevo valor) la variable subyacente al parámetro.

```
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
```

Lea Palabras clave ByVal y ByRef en línea: <https://riptutorial.com/es/vb-net/topic/4653/palabras-clave-byval-y-byref>

Capítulo 40: Patrón asíncrono basado en tareas

Examples

Uso básico de Async / Await

Puede iniciar un proceso lento en paralelo y luego recopilar los resultados cuando se realicen:

```
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
```

Después de dos segundos ambos resultados estarán disponibles.

Usando TAP con LINQ

Puede crear un `IEnumerable` de `Task` pasando `AddressOf AsyncMethod` al método **LINQ** `Select` y luego iniciar y esperar todos los resultados con `Task.WhenAll`

Si su método tiene parámetros que coinciden con la llamada de la cadena **LINQ** anterior, se asignarán automáticamente.

```
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
```

Para asignar diferentes argumentos, puede reemplazar el `AddressOf Method` con un lambda:

```
Function(linqData As Integer) MyNonMatchingMethod(linqData, "Other parameter")
```

Lea Patrón asíncrono basado en tareas en línea: <https://riptutorial.com/es/vb-net/topic/2936/patron-asincrono-basado-en-tareas>

Capítulo 41: Pruebas Unitarias en VB.NET

Observaciones

Este es el ejemplo más simple pero descriptivo para el procedimiento de prueba de unidad. Siéntase libre de agregar más métodos para verificar diferentes tipos de datos.

Examples

Pruebas unitarias para el cálculo de impuestos

Este ejemplo está dividido en dos pilares.

- **Clase de cálculo salarial:** cálculo del salario neto después de la deducción de impuestos
- **SalaryCalculationTests Class :** para probar el método que calcula el salario neto

Paso 1: Cree la biblioteca de clases, llámela **WagesLibrary** o cualquier nombre apropiado. Luego renombra la clase a **SalaryCalculation**

```
" " " Clase para cálculos salariales " 'Clase pública SalaryCalculation
```

```
''' <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
```

Paso 2 : Crear proyecto de prueba de unidad. Agregue una referencia a la biblioteca de clases creada y pegue el siguiente código

```
Imports WagesLibrary 'Class library you want to test

''' <summary>
''' Test class for testing SalaryCalculation
''' </summary>
<TestClass()> Public Class SalaryCalculationTests

    ''' <summary>
```

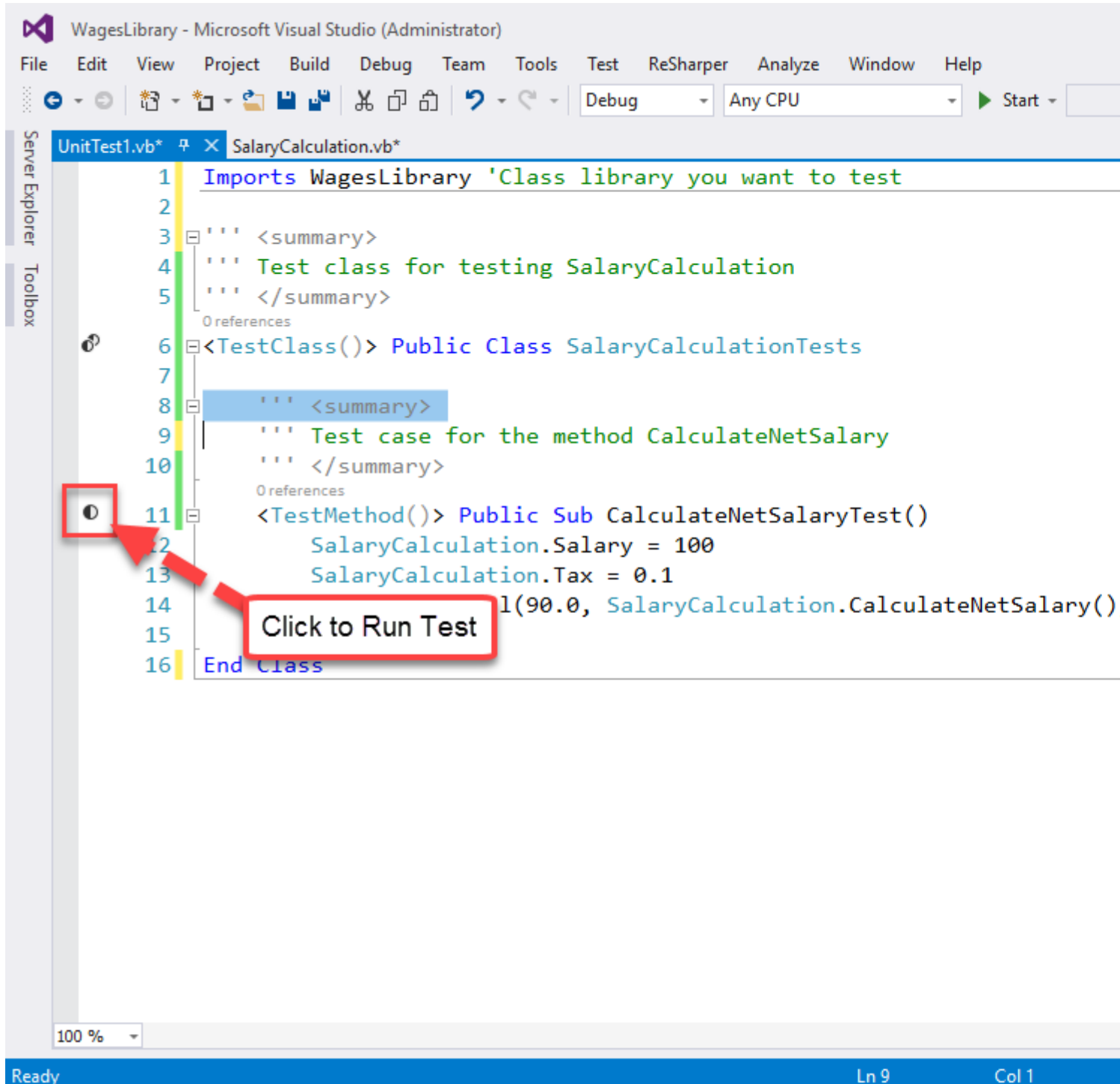
```

''' 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

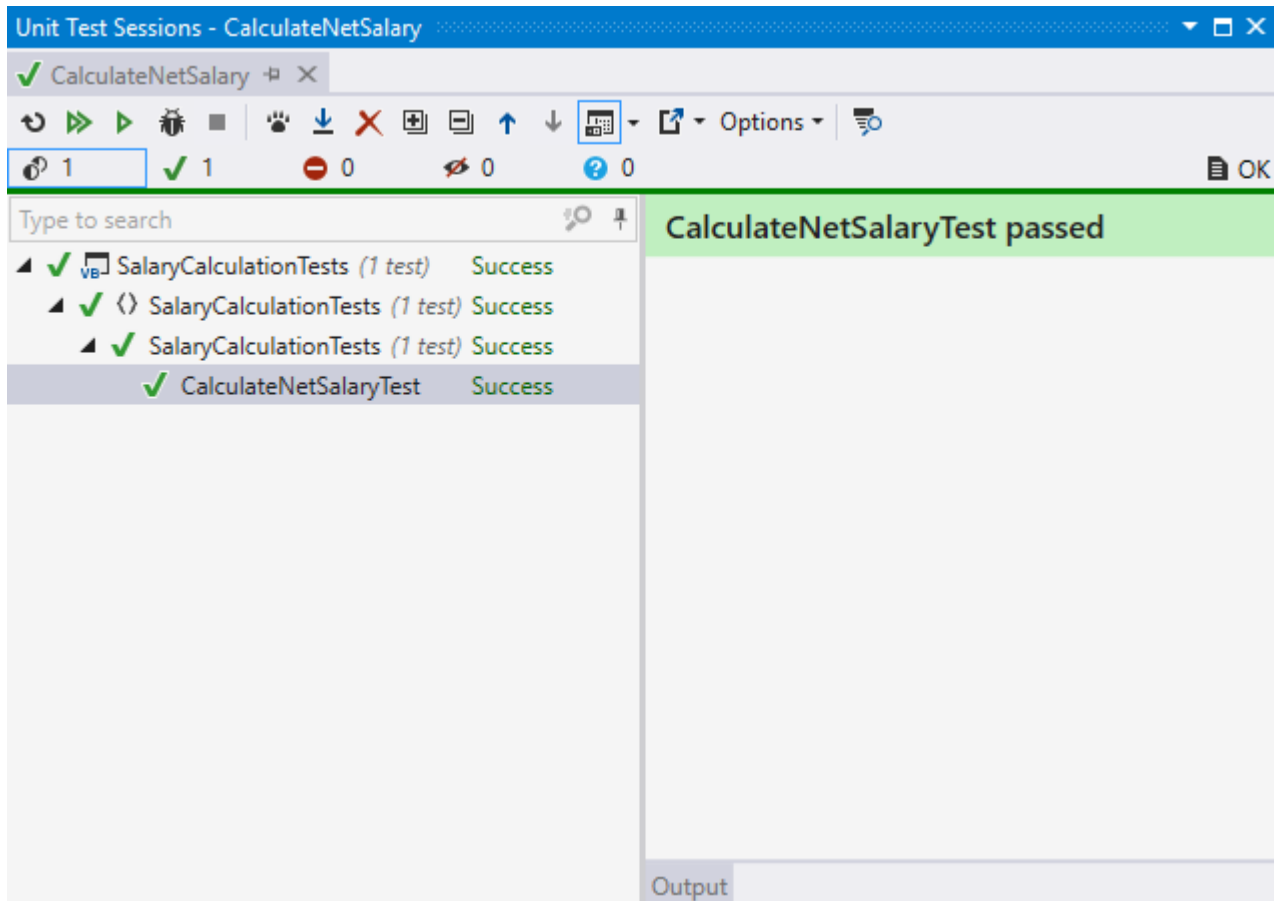
```

Assert.Equal verifica el valor esperado contra el valor real calculado. el valor 0.1 se utiliza para permitir la tolerancia o la variación entre el resultado esperado y el real .

Paso 3: Ejecutar la prueba del método para ver el resultado.



Resultado de la prueba



Clase de empleado de prueba asignada y propiedades derivadas

Este ejemplo tiene más pruebas disponibles en pruebas unitarias.

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 (Proyecto de prueba)

```

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

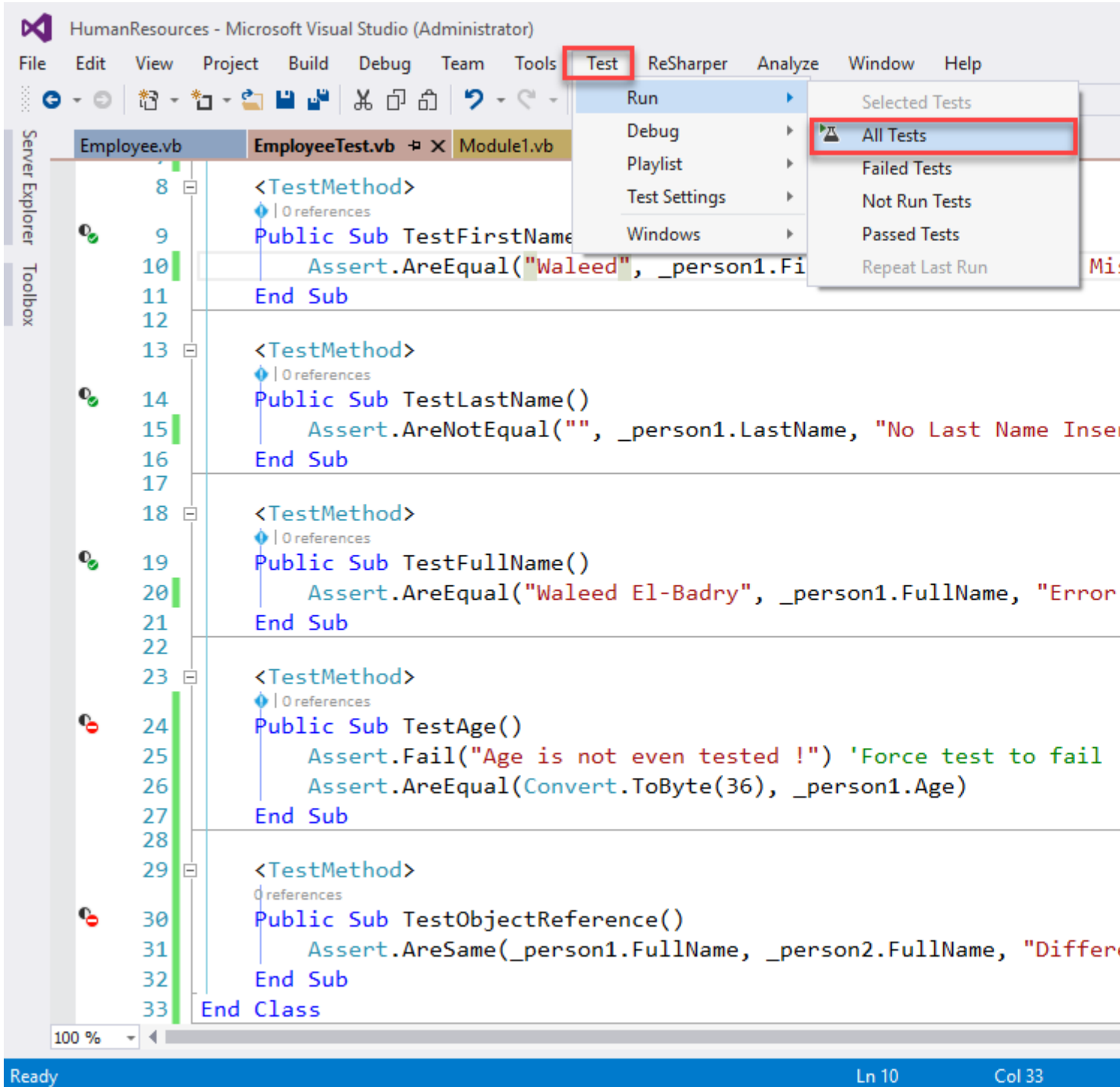
    <TestMethod>
    Public Sub TestFullName()
        Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error in concatenation of
names")
    End Sub

    <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

```

Resultado después de ejecutar pruebas



HumanResources - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Team Tools Test ReSharper Analyze Window Help

Debug Any CPU Start

Test Explorer

Configure continuous integration
Setup continuous integration(CI) builds to test continuously after every code change.
Don't show this again
Run All | Run... | Playlist: All Tests

Failed Tests (2)

- TestAge 5 ms
- TestObjectReference < 1 ms

Passed Tests (3)

- TestFirstName 3 ms
- TestFullName < 1 ms
- TestLastName < 1 ms

Summary

Last Test Run Failed (Total Run Time 0:00:00)

- 2 Tests Failed
- 3 Tests Passed

EmployeeTest.vb

```

8 <TestMethod>
9   | 0 references
9   Public Sub TestFirstName()
10  |   Assert.AreEqual("Waleed", _
11  |   End Sub
12
13 <TestMethod>
14 | 0 references
14 Public Sub TestLastName()
15 |   Assert.AreNotEqual("", _
16 |   End Sub
17
18 <TestMethod>
19 | 0 references
19 Public Sub TestFullName()
20 |   Assert.AreEqual("Waleed", _
21 |   End Sub
22
23 <TestMethod>
24 | 0 references
24 Public Sub TestAge()
25 |   Assert.Fail("Age is not
26 |   Assert.AreEqual(Convert.
27 |   End Sub
28
29 <TestMethod>
30 | 0 references
30 Public Sub TestObjectReferen
31 |   Assert.AreSame(_person1.
32 |   End Sub
33 End Class

```

Lea Pruebas Unitarias en VB.NET en línea: <https://riptutorial.com/es/vb-net/topic/6843/pruebas-unitarias-en-vb-net>

Capítulo 42: Recursion

Examples

Calcular el número de Fibonacci

Visual Basic.NET, como la mayoría de los lenguajes, permite la recursión, un proceso mediante el cual una función se llama a *sí misma* bajo ciertas condiciones.

Aquí hay una función básica en Visual Basic .NET para calcular los números de [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
```

Esta función funciona al verificar primero si la función se ha llamado con el parámetro n igual a 1 o 2 . Por definición, los dos primeros valores en la secuencia de Fibonacci son 1 y 1, por lo que no es necesario realizar ningún cálculo adicional para determinar esto. Si n es mayor que 2, no podemos buscar el valor asociado tan fácilmente, pero sabemos que cualquier número de Fibonacci es igual a la suma de los dos números anteriores, por lo que los solicitamos por medio de la *recursión* (llamando a nuestra propia función de Fibonacci). Dado que las llamadas recursivas sucesivas son llamadas con números cada vez más pequeños a través de decrementos de -1 y -2, sabemos que eventualmente alcanzarán números menores a 2. Una vez que se alcanzan esas condiciones (llamadas *casos base*), la pila se desenrolla y nosotros Consigue nuestro resultado final.

Lea Recursion en línea: <https://riptutorial.com/es/vb-net/topic/7862/recursion>

Capítulo 43: Reflexión

Examples

Recuperar propiedades para una instancia de una clase

```
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
```

El Parámetro de `GetProperties` define qué tipos de Propiedades devolverá la función. Desde que aprobamos `Public` e `Instance`, el método devolverá solo las propiedades que son tanto públicas como no compartidas. Consulte [el atributo The Flags](#) para obtener una explicación sobre cómo se pueden combinar Flag-enums.

Obtener los miembros de 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
```

Consigue un método e invocalo.

Método estático:

```
Dim parseMethod = GetType(Integer).GetMethod("Parse", {GetType(String)})
Dim result = DirectCast(parseMethod.Invoke(Nothing, {"123"}), Integer)
```

Método de instancia:

```
Dim instance = "hello".ToUpper
```

```
Dim method = GetType(String).GetMethod("ToUpper", {})  
Dim result = method.Invoke(instance, {})  
Console.WriteLine(result) 'HELLO
```

Crear una instancia de un tipo genérico.

```
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")
```

Lea Reflexión en línea: <https://riptutorial.com/es/vb-net/topic/1598/reflexion>

Capítulo 44: Servidor FTP

Sintaxis

- `My.Computer.Network.DownloadFile (serverFile As String, localFile As String)`
- `My.Computer.Network.DownloadFile (serverFile As String, localFile As String, usuario As String, contraseña como String)`
- `My.Computer.Network.UploadFile (localFile As String, serverFile As String)`
- `My.Computer.Network.UploadFile (localFile As String, serverFile As String, usuario As String, contraseña como String)`

Examples

Descargar el archivo desde el servidor FTP

```
My.Computer.Network.DownloadFile("ftp://server.my/myfile.txt", "downloaded_file.txt")
```

Este comando descarga el archivo `myfile.txt` del servidor llamado `server.my` y lo guarda como `downloaded_file.txt` en el directorio de trabajo. Puede especificar la ruta absoluta para el archivo descargado.

Descargue el archivo desde el servidor FTP cuando sea necesario iniciar sesión

```
My.Computer.Network.DownloadFile("ftp://srv.my/myfile.txt", "download.txt", "Peter", "1234")
```

Este comando descarga el archivo `myfile.txt` del servidor llamado `srv.my` y lo guarda como `download.txt` en el directorio de trabajo. Puede especificar la ruta absoluta para el archivo descargado. El archivo es descargado por el usuario `Peter` con la contraseña `1234`.

Subir archivo al servidor FTP

```
My.Computer.Network.UploadFile("example.txt", "ftp://server.my/server_example.txt")
```

Este comando carga el archivo `example.txt` desde el directorio de trabajo (puede especificar la ruta absoluta si lo desea) al servidor llamado `server.my`. El archivo almacenado en el servidor se llamará `server_example.txt`.

Cargue el archivo al servidor FTP cuando se requiera iniciar sesión

```
My.Computer.Network.UploadFile("doc.txt", "ftp://server.my/on_server.txt", "Peter", "1234")
```

Este comando carga el archivo `doc.txt` desde el directorio de trabajo (puede especificar la ruta

absoluta si lo desea) al servidor llamado `server.my` . El archivo almacenado en el servidor se llamará `server_example.txt` . El relleno se envía en el servidor por el usuario Peter y la contraseña 1234.

Lea Servidor FTP en línea: <https://riptutorial.com/es/vb-net/topic/4078/servidor-ftp>

Capítulo 45: Tipo de conversión

Sintaxis

- CBool (expresión)
- CByte (expresión)
- CChar (expresión)
- CDate (expresión)
- CDb1 (expresión)
- CDec (expresión)
- CInt (expresión)
- CLng (expresión)
- CObj (expresión)
- CSByte (expresión)
- CShort (expresión)
- CSng (expresión)
- CStr (expresión)
- CUInt (expresión)
- CULng (expresión)
- CUShort (expresión)

Parámetros

| Nombre de la función | Rango para el argumento de expresión |
|----------------------|--|
| CBool | Cualquier Char o String válido o expresión numérica |
| CByte | 0 a 255 (sin firmar); Las partes fraccionarias son redondeadas. |
| CChar | Cualquier expresión válida de Char o String; solo se convierte el primer carácter de una cadena; el valor puede ser de 0 a 65535 (sin firmar). |

Examples

Convertir el texto del cuadro de texto en un entero

Desde [MSDN](#)

Utilice la función CInt para proporcionar conversiones de cualquier otro tipo de datos a un subtipo de entero. Por ejemplo, CInt fuerza la aritmética de enteros cuando normalmente ocurre una aritmética de moneda, precisión simple o precisión doble.

Suponiendo que tiene 1 botón y 2 cuadros de texto. Si escribe en `textbox1.text` 5.5 y en

textbox2.text 10 .

Si tienes este código:

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

Para agregar los valores de los 2 cuadros de texto, debe convertir sus valores a `Int` usando el `CInt(expression)` .

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

Nota: cuando la parte fraccionaria de un valor es exactamente 0.5, la función `CInt` se redondea al número par más cercano. Por ejemplo, **0.5 redondea a 0** , mientras que **1.5 redondea a 2** , y **3.5 redondea a 4** . El propósito de redondear al número par más cercano es compensar un sesgo que podría acumularse cuando se suman muchos números.

Lea Tipo de conversión en línea: <https://riptutorial.com/es/vb-net/topic/4681/tipo-de-conversion>

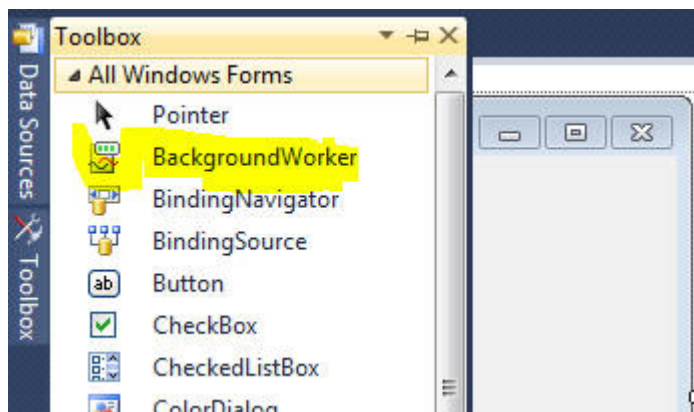
Capítulo 46: Trabajador de fondo

Examples

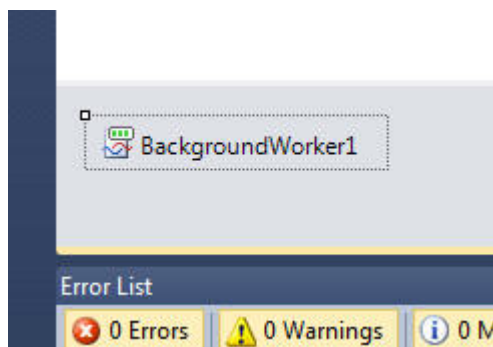
Usando BackgroundWorker

Ejecutando una tarea con el trabajador de fondo.

Haga doble clic en el control `BackgroundWorker` de la caja de herramientas



Así es como aparece BackgroundWorker después de agregarlo.



Haga doble clic en el control agregado para obtener el evento `BackgroundWorker1_DoWork` y agregue el código que se ejecutará cuando se llame a `BackgroundWorker`. Algo como esto:

```
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
```

Se puede llamar a `BackgroundWorker` para realizar la tarea en cualquier evento como `Button_Click`, `Textbox_TextChanged`, etc. de la siguiente manera:

```
BackgroundWorker1.RunWorkerAsync()
```

Modifique el evento `RunWorkerCompleted` para capturar el evento de tarea terminada del `BackgroundWorker` de la siguiente manera:

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

Esto mostrará un cuadro de mensaje que dice `Done` cuando el trabajador finalice la tarea asignada.

Accediendo a los componentes GUI en `BackgroundWorker`

No puede acceder a ningún componente GUI desde `BackgroundWorker`. Por ejemplo, si intentas hacer algo como esto.

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

recibirá un error de tiempo de ejecución que dice que "La operación de subprocessos no es válida: control 'TextBox1' al que se accede desde un subprocesso que no sea el que se creó".

Esto se debe a que `BackgroundWorker` ejecuta su código en otro subprocesso en paralelo con el subprocesso principal, y los componentes de la GUI no son seguros para subprocessos. Debe configurar su código para que se ejecute en el subprocesso principal utilizando el método `Invoke`, dándole un delegado:

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

O puede usar el método `ReportProgress` de `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
```

Lea **Trabajador de fondo en línea**: <https://riptutorial.com/es/vb-net/topic/6242/trabajador-de-fondo>

Capítulo 47: Trabajar con formularios

Windows Forms

Examples

Usando la instancia de formulario por defecto

VB.NET ofrece instancias de formulario por defecto. El desarrollador no necesita crear la instancia, ya que se crea entre bambalinas. Sin embargo, **no** es *preferible* utilizar la instancia predeterminada todos, pero los programas más simples.

```
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
```

Ver también:

- [¿Tienes que crear explícitamente la instancia de formulario en VB.NET?](#)
- [¿Por qué hay una instancia predeterminada de cada formulario en VB.Net pero no en C #?](#)

Pasando datos de una forma a otra

A veces es posible que desee pasar información que se ha generado de una forma, a otra forma para uso adicional. Esto es útil para formularios que muestran una herramienta de búsqueda o una página de configuración entre muchos otros usos.

Digamos que desea pasar un `DataTable` entre un formulario que ya está abierto (*MainForm*) y un nuevo formulario (*NewForm*):

En el MainForm:

```
Private Sub Open_New_Form()
```

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

En el nuevo formulario

```
Public Class NewForm
    Dim NewDataTable as Datatable

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

End Class
```

Ahora, cuando se abre *NewForm* , se le pasa `DataTable1` de *MainForm* y se almacena como `NewDataTable` en *NewForm* para que lo use ese formulario.

Esto puede ser extremadamente útil cuando se trata de pasar grandes cantidades de información entre formularios, especialmente cuando se combina toda la información en un solo `ArrayList` y se pasa el `ArrayList` al nuevo formulario.

Lea **Trabajar con formularios Windows Forms en línea**: <https://riptutorial.com/es/vb-net/topic/4636/trabajar-con-formularios-windows-forms>

Capítulo 48: Usando axWindowsMediaPlayer en VB.Net

Introducción

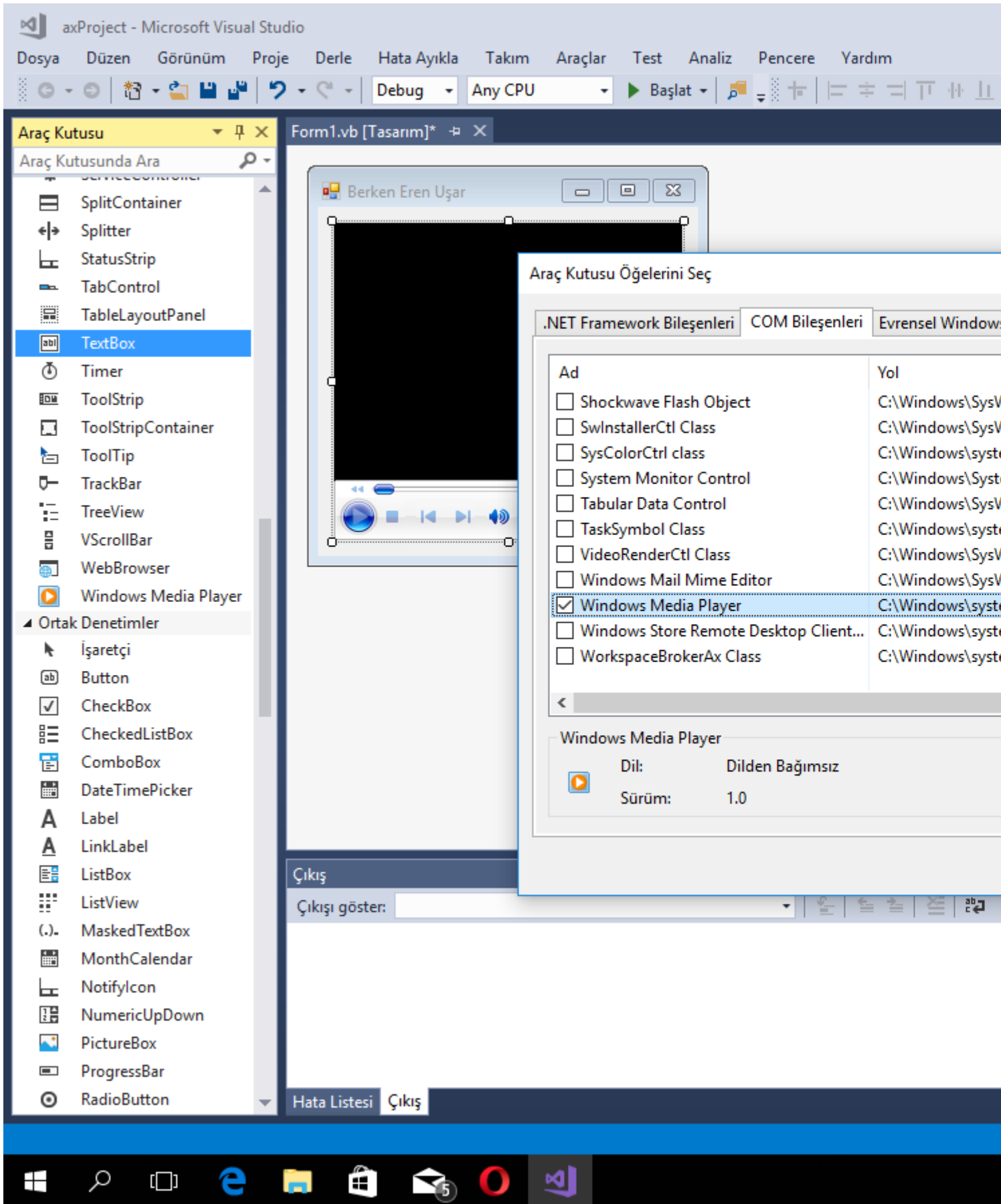
axWindowsMediaPlayer es el control para la reproducción de archivos multimedia como videos y música.

Examples

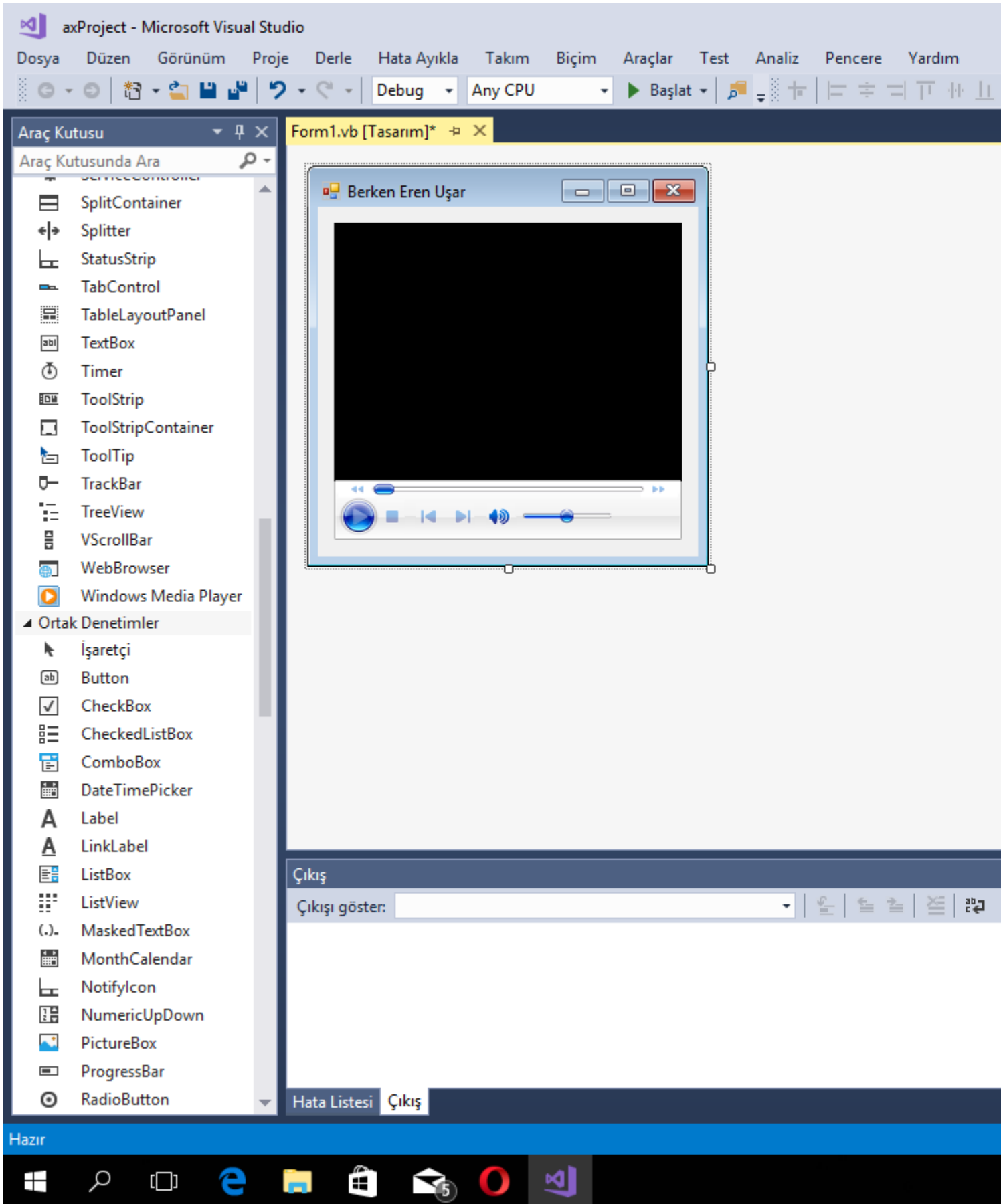
Añadiendo el axWindowsMediaPlayer

- Haga clic derecho en la caja de herramientas, luego haga clic en "Elegir elementos".
- Seleccione la pestaña Componentes COM y luego verifique Windows Media Player.
- axWindowsMediaPlayer se agregará a Toolbox.

Seleccione esta casilla de verificación para usar axWindowsMediaPlayer



Entonces puedes usar axWindowsMediaPlayer :)



Reproducir un archivo multimedia

```
AxWindowsMediaPlayer1.URL = "C:\My Files\Movies\Avatar.mp4"
```

```
AxWindowsMediaPlayer1.Ctlcontrols.play()
```

Este código jugará a Avatar en el axWindowsMediaPlayer.

Lea Usando axWindowsMediaPlayer en VB.Net en línea: <https://riptutorial.com/es/vb-net/topic/10096/usando-axwindowsmediaplayer-en-vb-net>

Capítulo 49: Usando BackgroundWorker

Examples

Implementación básica de la clase de trabajador de fondo

Necesitas importar System.ComponentModel para usar el trabajador de fondo

```
Imports System.ComponentModel
```

Entonces declara una variable privada

```
Private bgWorker As New BackgroundWorker
```

Debe crear dos métodos para los eventos DoWork y RunWorkerCompleted del trabajador en segundo plano y asignarlos.

```
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
```

El siguiente sub se ejecutará cuando el trabajador termine el trabajo

```
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
```

Luego, dentro de su código, agregue las líneas siguientes para iniciar el trabajador de fondo

```
bgWorker = New BackgroundWorker
AddHandler bgWorker.DoWork, AddressOf MyWorker_DoWork
AddHandler bgWorker.RunWorkerCompleted, AddressOf MyWorker_RunWorkerCompleted
bgWorker.RunWorkerAsync()
```

Cuando llame a la función RunWorkerAsync (), se ejecutará MyWorker_DoWork.

Lea Usando BackgroundWorker en línea: <https://riptutorial.com/es/vb-net/topic/6401/usando-backgroundworker>

Capítulo 50: Utilizando la declaración

Sintaxis

- Usando `a = Nueva DisposableClass [, b = ...]`
...
Terminar de usar
- Usando `a = GetDisposable (...) [, b = ...]`
...
Terminar de usar

Examples

Ver ejemplos en Objetos desechables.

[Concepto básico de IDisposable.](#)

Lea [Utilizando la declaración en línea](https://riptutorial.com/es/vb-net/topic/7965/utilizando-la-declaracion): <https://riptutorial.com/es/vb-net/topic/7965/utilizando-la-declaracion>

Capítulo 51: WinForms SpellCheckBox

Introducción

Ejemplo sobre cómo agregar una casilla de verificación de ortografía a una aplicación de WindowsForms. Este ejemplo NO requiere que Word esté instalado ni utiliza Word de ninguna manera.

Utiliza WPF Interop utilizando el control ElementHost para crear un UserControl de WPF desde un TextBox de WPF. WPF TextBox tiene una función incorporada para la revisión ortográfica. Vamos a aprovechar esta función incorporada en lugar de confiar en un programa externo.

Examples

ElementHost WPF TextBox

Este ejemplo fue modelado después de un ejemplo que encontré en internet. No puedo encontrar el enlace o le daría crédito al autor. Tomé la muestra que encontré y la modifiqué para que funcione para mi aplicación.

1. Agregue las siguientes referencias:

System.Xaml, PresentationCore, PresentationFramework, WindowsBase y WindowsFormsIntegration

2. Crea una nueva clase y pasa este código

```
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. Reconstruye la solución.
4. Añadir un nuevo formulario.
5. Busca en la caja de herramientas el nombre de tu clase. Este ejemplo es "SpellCheck". Debería aparecer en la lista debajo de los componentes de 'YourSoulutionName'.
6. Arrastra el nuevo control a tu formulario.
7. Establecer cualquiera de las propiedades asignadas en el evento de carga de formularios

```

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. Lo último que debe hacer es cambiar el conocimiento de DPI de su aplicación. Esto se debe a que está utilizando la aplicación WinForms. Por defecto, todas las aplicaciones de WinForms son DPI UNAWARE. Una vez que ejecute un control que tenga un elemento host (interoperabilidad WPF), la aplicación ahora se convertirá en DPI AWARE. Esto puede o no desordenar con sus elementos UI. La solución a esto es FORZAR la aplicación para que se convierta en DPI UNAWARE. Hay 2 formas de hacer esto. El primero es a través del archivo de manifiesto y el segundo es codificarlo en su programa. Si está utilizando OneClick para implementar su aplicación, debe codificarla, no usar el archivo de manifiesto o los errores serán inevitables.

Los dos ejemplos siguientes se pueden encontrar en los siguientes: [Escala de WinForms en la configuración de DPI grande: ¿es posible incluso?](#) Gracias a Telerik.com por la gran explicación sobre DPI.

Código codificado duro Ejemplo de código Aware. Esto DEBE ejecutarse antes de que se inicialice la primera forma. Siempre coloco esto en el archivo ApplicationEvents.vb. Puede acceder a este archivo haciendo clic derecho en el nombre de su proyecto en el explorador de soluciones y seleccionando "Abrir". Luego, elija la pestaña de la aplicación a la izquierda y luego haga clic en "Ver eventos de la aplicación" en la esquina inferior derecha al lado del menú desplegable de la pantalla de bienvenida.

```

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

```

Ejemplo de manifiesto

```

<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>

```

Lea WinForms SpellCheckBox en línea: <https://riptutorial.com/es/vb-net/topic/8624/winforms-spellcheckbox>

Creditos

| S. No | Capítulos | Contributors |
|-------|--|---|
| 1 | Comenzando con el lenguaje 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 | Acceso a los datos | MatVAD , Mike Robertson , Nico Agusta |
| 3 | Aleatorio | David Wilson |
| 4 | Bucle | CiccioRocca , debater , Imran Ali Khan , Mark , MatVAD , Sam Axe , Scott Mitchell , SilverShotBee , TyCobb , vbnet3d , void |
| 5 | Características de 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 , yumypasta |
| 6 | Compresión de archivos / carpetas | Cody Gray , MatVAD , Misaz , vbnet3d |
| 7 | Condiciones | Allen Binuya , Chetan Sanghani , Nathan Tuggy , Robert Columbia |
| 8 | Consola | Bugs , InteXX , lucamauri , Martin Soles , Matthew Whited , Sam Axe , Slava Auer , StardustGogeta , vbnet3d , VortexDev |
| 9 | Declarando variables | Cody Gray , Darren Davies , Fütemire , glaubergft , keronconk , LogicalFlaps , MatVAD , RamenChef , Sehnsucht |
| 10 | Depurando tu aplicación | Martin Verjans |
| 11 | Enhebrado | BiscuitBaker , Stefano d'Antonio , Visual Vincent |
| 12 | Enlace de datos WPF XAML | Milliron X |
| 13 | Enumerar | 4444 , CiccioRocca , David Sdot , dju , ElektroStudios , kodkod , LogicalFlaps , Shog9 , Steven Doggart , void |
| 14 | Excepcion de referencia nula | Alessandro Mascolo , RamenChef , Sehnsucht |

| | | |
|----|--|--|
| 15 | Fecha | Matt Wilko , Misaz |
| 16 | Formación | BunkerMentality , dju , Drarig29 , Luke Sheppard , Mark Hurd , MatVAD , Robert Columbia , Ryan Thomas , Sam Axe , Sehnsucht , Steven Daggart , TuxCopter , vbnet3d , VortixDev , zyabin101 |
| 17 | Funciones | Berken Usar |
| 18 | GDI + | Dman |
| 19 | Genéricos | JDC |
| 20 | Google Maps en un formulario de Windows | anonymous , Carlos Borau |
| 21 | Introducción a la sintaxis | Bugs , Mark Hurd , Martin Verjans , mnoronha , Nat G. , Nico Agusta , Sehnsucht |
| 22 | Las clases | Darren Davies , Harjot |
| 23 | Leyendo el archivo de texto comprimido sobre la marcha | Proger_Cbsk |
| 24 | LINQ | Dan Drews , Daz , Derek Tomes , Fütemire , H. Pauwelyn , Mark Hurd , Misaz , Sam Axe , Sehnsucht , Zev Spitz |
| 25 | Liza | Dman , DrDonut , Fütemire , Luke Sheppard , Robert Columbia , Seandk |
| 26 | Los diccionarios | DrDonut , Nathan , Proger_Cbsk , Sehnsucht , void |
| 27 | Los operadores | Bjørn-Roger Kringsjå , Cary Bondoc , MatVAD , Mike Robertson , Pasilda , Robert Columbia , RoyalPotato , Sam Axe , Sree , varocarbas , void |
| 28 | Manejo de archivos | Dan Granger , Luke Sheppard , Matt Wilko , Misaz , Shayan Toqraee , vbnet3d |
| 29 | Manejo de errores | Adam Zuckerman , Bjørn-Roger Kringsjå , HappyPig375 , Luke Sheppard , MatVAD , Nico Agusta , Vishal |
| 30 | Manejo de la conexión | Jonas_Hess |
| 31 | Metodos de extension | Fütemire , InteXX , Matt Wilko , Sam Axe , Stefano d'Antonio , void |
| 32 | Multihilo | 4444 , Daz , MatVAD |
| 33 | Objetos desechables | KE0GSD , Mark Hurd , Martin Verjans , Matt Wilko , Misaz , |

| | | |
|----|--|--|
| | | Mithrandir , Sam Axe |
| 34 | OOP Keywords | 4444 , David , JDC , Matt Wilko , Nat G. |
| 35 | Opción Estricta | Andrew Morton , Cary Bondoc , Matt Wilko , RussAwesome , Sam Axe , vbnet3d |
| 36 | Opción explícita | HappyPig375 , Matt Wilko , sansknowledge |
| 37 | Opcion Inferir | Alex B. , LogicalFlaps , Mark Hurd |
| 38 | Operadores de cortocircuito (y también - orElse) | Bart Jolling , CiccioRocca , Kendra , Sam Axe |
| 39 | Palabras clave ByVal y ByRef | Adam Zuckerman , Misaz , Sehnsucht |
| 40 | Patrón asíncrono basado en tareas | Stefano d'Antonio |
| 41 | Pruebas Unitarias en VB.NET | wbdry |
| 42 | Recursion | Robert Columbia |
| 43 | Reflexión | Axarydax , Matt , Sam Axe , void |
| 44 | Servidor FTP | Misaz |
| 45 | Tipo de conversión | Cary Bondoc , LogicalFlaps , vbnet3d |
| 46 | Trabajador de fondo | Jones Joseph , Shayan Toqraee |
| 47 | Trabajar con formularios Windows Forms | djv , SilverShotBee , vbnet3d |
| 48 | Usando axWindowsMediaPlayer en VB.Net | Berken Usar |
| 49 | Usando BackgroundWorker | MatVAD |
| 50 | Utilizando la declaración | VV5198722 |
| 51 | WinForms SpellCheckBox | Nathan |