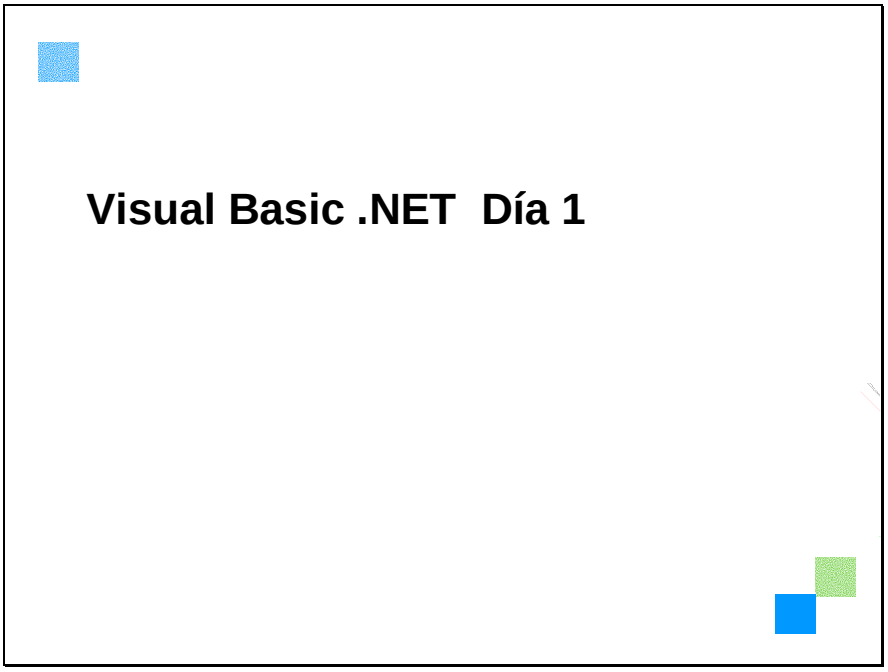
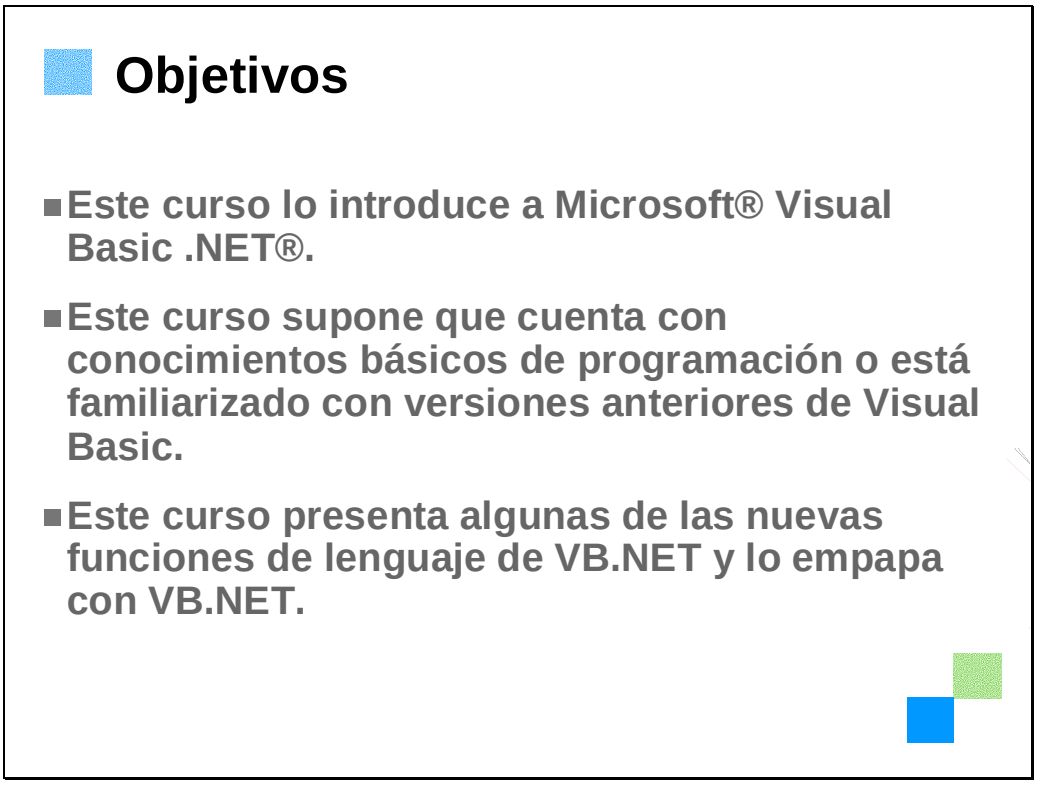
Diapositiva 1



Diapositiva 2



Diapositiva 3

Agenda

- Hoy aprenderemos:
  - ¿Qué es .NET?
    - .NET Framework
    - CLR, Biblioteca de clases
  - Funciones de lenguaje
    - Estructura del Programa VB.NET
    - Compilación condicional
    - División y combinación de instrucciones
    - Etiquetas
    - Caracteres especiales
    - Comentarios
    - Limitaciones
    - Tipos de datos
    - Operadores
    - Flujo de control
    - Bucles
    - Operaciones básicas de entrada/salida
    - Orientado completamente a objetos
    - Alcance
    - Manejo de errores
    - Cambios de sintaxis
  - Hello World!

Diapositiva 4

Requerimientos: Lo que necesitará

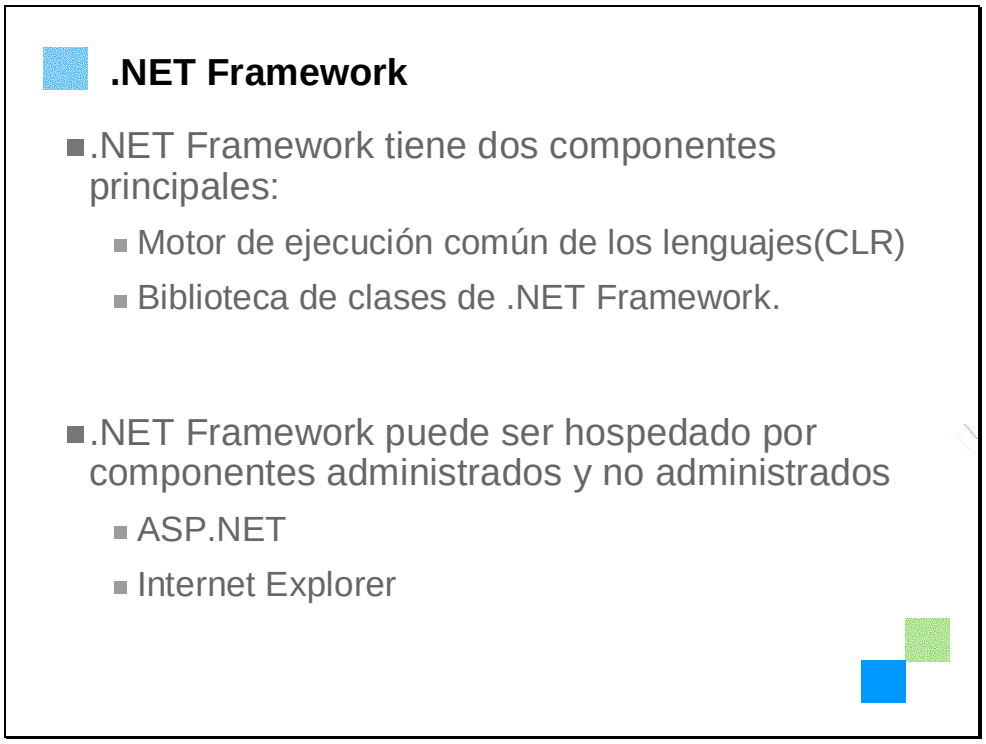
- **Se requiere el SDK de.NET Framework 1.1**
  - http://microsoft.com/downloads/details.aspx?FamilyId=9B3A2CA6-3647-4070-9F41-A333C6B9181D&displaylang=en
  - Matriz Web
    http://www.asp.net/webmatrix/download.aspx?tabindex=4
  - MSDE
    http://www.asp.net/msde/default.aspx?tabindex=0&tabid=1
  - MDAC 2.7
    http://www.microsoft.com/downloads/details.aspx?FamilyID=9ad000f2-cae7-493d-b0f3-ae36c570ade8&DisplayLang=en
- **Herramientas que se recomiendan**
  - Visual Studio .NET 2003
  - SQL Server
  - Internet Information Services 5.0 ó superior

Diapositiva 5

.NET Framework

- .NET Framework es una nueva plataforma de cómputo que simplifica el desarrollo de aplicaciones en el ambiente altamente distribuido de Internet.
  - Minimizar los conflictos de implementación y versiones del software.
  - Programación consistente orientada al objeto
  - Garantizar una ejecución segura del código
  - Elimina los problemas de rendimiento de los ambientes con secuencia de comandos o interpretados.
  - Experiencia consistente para el desarrollador a través de varios tipos de aplicaciones
    - Windows
    - Basado en el Web
  - Desarrollado sobre los estándares de la industria para asegurar que el código basado en .NET Framework pueda integrarse con cualquier otro.

Diapositiva 6



.NET Framework tiene dos componentes principales:

El Motor de ejecución común de los lenguajes (CLR)

El Motor de ejecución común de los lenguajes es el fundamento de .NET Framework. Usted puede pensar sobre el motor de ejecución como un agente que maneje código en el motor de ejecución , proporcionando servicios principales como la administración de memoria, administración de subprocesos y remota, aplicando al mismo tiempo una seguridad estricta de tipos y otras formas de precisión de códigos que garantiza la seguridad y la robustez. El concepto de administración de código es un principio fundamental del motor de ejecución . El código que se ejecuta en el motor de ejecución se conoce como código administrado, mientras que el código el que no lo hace, como código no administrado

Biblioteca de clases de .NET Framework

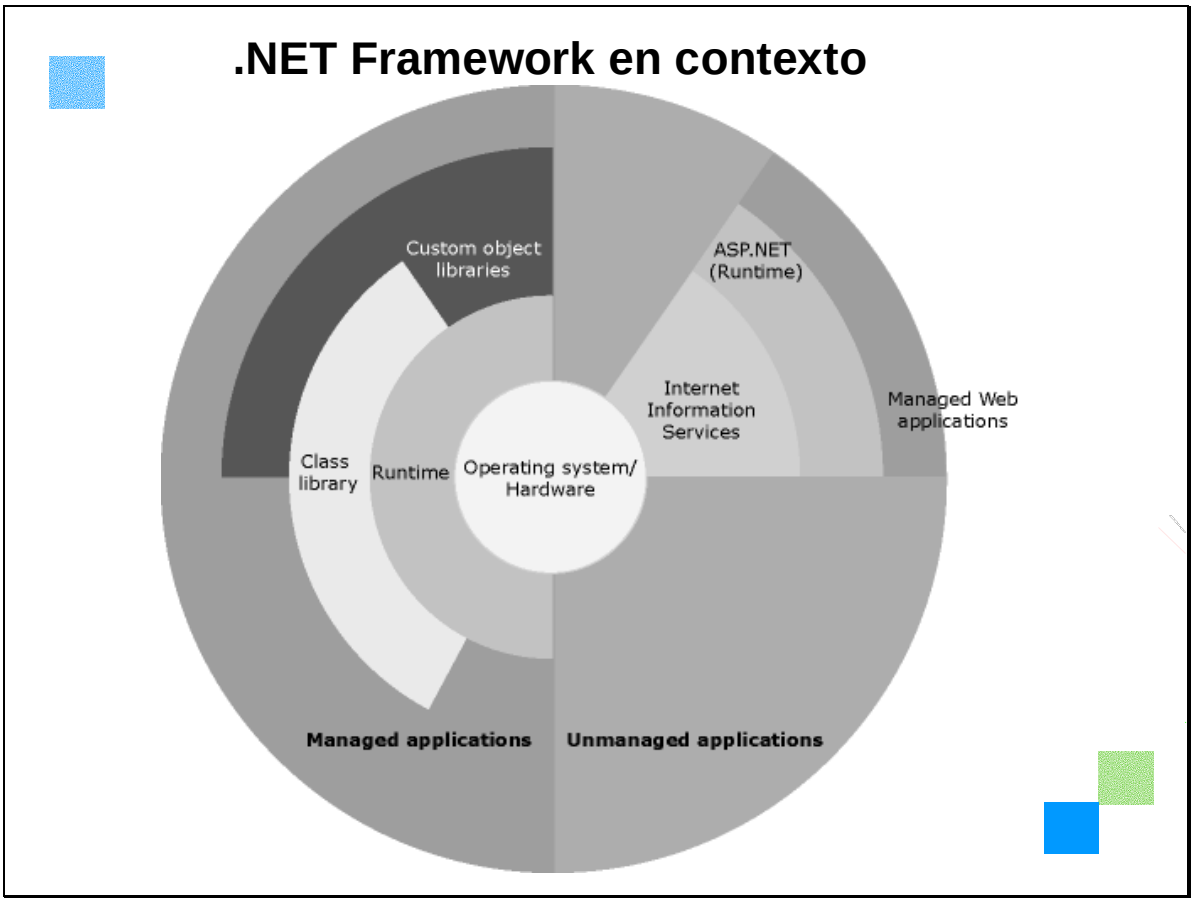
La biblioteca de clases, el otro componente principal de .NET Framework, es una colección completa y orientada a objetos de tipos reutilizables que puede utilizar para desarrollar aplicaciones que van desde aplicaciones tradicionales de línea de comandos o de interfaz gráfica (GUI) hasta aplicaciones basadas en las innovaciones más recientes proporcionadas por ASP.NET, tales como Web Forms y los servicios Web XML.

.NET Framework puede ser hospedado por componentes no administrados que cargan el Motor de ejecución común de los lenguajes en sus procesos e inician la ejecución de código administrado, creando por lo tanto un ambiente de software que puede explotar tanto funciones administradas como no administradas. .NET Framework no sólo proporciona varios *hosts* de motor de ejecución , sino también soporta el desarrollo de *hosts* de motor de ejecución de terceros.

Por ejemplo, ASP.NET alberga el motor de ejecución para proporcionar un ambiente escalable y del lado del servidor para el código administrado. ASP.NET funciona directamente con el motor de ejecución para activar las aplicaciones ASP.NET y los servicios Web de XML, mismos que se analizan más adelante en este tema.

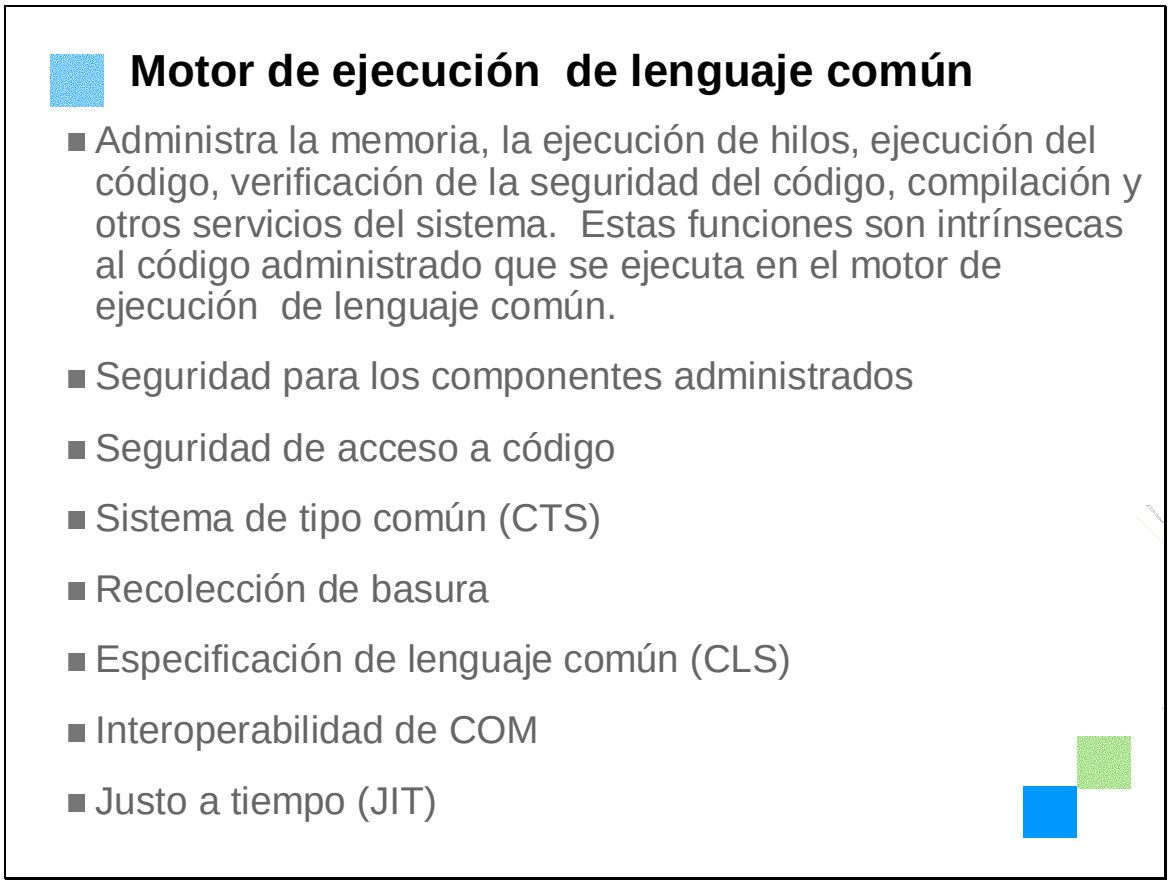
Internet Explorer es un ejemplo de una aplicación no administrada que alberga el motor de ejecución (en la forma de una extensión tipo MIME). Utilizar Internet Explorer para albergar el motor de ejecución le permite incrustar componentes administrados o controles de Windows Forms en los documentos HTML. Al hospedar el motor de ejecución de esta manera, se permite que el código móvil administrado (similar a los controles Microsoft® ActiveX®) sean posibles, pero con mejoras significativas que sólo el código administrado puede ofrecer, tales como la ejecución de semiconfianza y un almacenamiento seguro de archivos aislados.

Diapositiva 7



Esta ilustración muestra la relación del Motor de ejecución común de los lenguajes y la biblioteca de clases para sus aplicaciones y para el sistema en general. La ilustración también muestra cómo el código administrado opera dentro de una arquitectura más grande.

Diapositiva 8



Las siguientes secciones describen los componentes y las funciones principales de .NET Framework con más detalle.

Motor de ejecución de lenguaje común

El motor de ejecución común de los lenguajes administra la memoria, la administración de hilos, la ejecución del código, la verificación de la seguridad del código, compilación y otros servicios del sistema. Estas funciones son intrínsecas al código administrado que se ejecuta en el motor de ejecución de lenguaje común.

Con respecto a la seguridad, los componentes administrados están concientes de los diferentes grados de confianza, dependiendo del número de factores que incluyen su origen (tales como Internet, la red empresarial o la computadora local). Esto significa que un componente administrado puede o no realizar operaciones de acceso a los archivos, operaciones de acceso a los registros u otras funciones sensibles, incluso si se está utilizando en la misma aplicación activa.

El motor de ejecución permite la seguridad en el acceso al código. Por ejemplo, los usuarios pueden confiar en que un ejecutable incrustado en una página Web pueda reproducir una animación en pantalla o cantar una canción, pero no puede acceder a sus datos personales, al sistema de archivos o a la red. Por lo tanto, las funciones de seguridad del motor de ejecución permiten que el software legítimamente implementado por Internet sea excepcionalmente rico en funciones.

El motor de ejecución también permite la robustez del código al implementar una infraestructura de verificación estricta de tipo y código conocida como el sistema de tipo común (CTS). El CTS permite que todo el código administrado se describa a sí mismo. Los varios compiladores de lenguajes de Microsoft y de terceros generan código administrado que cumple con el CTS. Esto significa que el código administrado puede consumir otros tipos e instancias administrados, imponiendo estrictamente la fidelidad del tipo y la seguridad del tipo.

Recolección de residuos: Además, el ambiente administrado del motor de ejecución elimina muchos problemas comunes del software. Por ejemplo, el motor de ejecución maneja automáticamente el diseño del objeto y administra las referencias hacia los objetos, liberándolos cuando ya no se están utilizando. Esta administración automática de la memoria resuelve los dos errores más comunes de las aplicaciones, fugas de memoria y referencias inválidas de la misma.

CLS: El motor de ejecución también acelera la productividad del desarrollador. Por ejemplo, los programadores pueden escribir aplicaciones en su lenguaje de desarrollo de elección, y aprovechar al mismo tiempo el motor de ejecución , la biblioteca de clases y los componentes escritos en otros lenguajes por otros desarrolladores. Cualquier proveedor de compiladores que elija tener como objetivo el motor de ejecución puede hacerlo. Los compiladores de lenguaje que se enfocan en .NET Framework hacen que las funciones de .NET Framework estén disponibles para los códigos existentes escritos en ese lenguaje, facilitando en gran medida el proceso de migración para las aplicaciones existentes.

Para descubrir más acerca del CLS, refiérase a este vínculo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconwhatiscommonlanguagespecification.asp

Interoperabilidad de COM: Mientras que el motor de ejecución está diseñado para el software del futuro, también soporta el software de hoy y ayer. La interoperabilidad entre los

códigos administrados y no administrados permite a los desarrolladores seguir utilizando los componentes COM y las DLLs que son necesarios.

JIT: El motor de ejecución está diseñado para mejorar el rendimiento. Aunque el motor de lenguaje común proporciona muchos de los servicios estándar del motor de ejecución , el código administrado nunca se interpreta. Una función conocida como compilación justo a tiempo (JIT) permite que todo el código administrado se ejecute en el lenguaje nativo de la máquina del sistema en la cual se está ejecutando. Mientras tanto, el administrador de memoria elimina las posibilidades de memoria fragmentada e incrementa la localidad de referencia de la memoria para incrementar aún más el rendimiento.

Diapositiva 9

La biblioteca de clases de .NET Framework.
Es una colección de tipos reutilizables que se integran estrechamente con el motor de ejecución del lenguaje común. La biblioteca de clases está orientada a objetos proporcionando tipos a partir de los cuales sus propios códigos administrados pueden derivar funcionalidad. Esto no sólo permite que los tipos del .NET Framework sean fáciles de utilizar, sino que también reduce el tiempo asociado con aprender nuevas funciones del mismo. Además, los componentes de terceros pueden integrarse de manera transparente con las clases en .NET Framework.

Por ejemplo, las clases de recolección .NET Framework implementan un conjunto de interfaces que puede utilizar para desarrollar sus propias clases de recolección. Sus clases de recolección se unirán de manera transparente con las clases en .NET Framework.

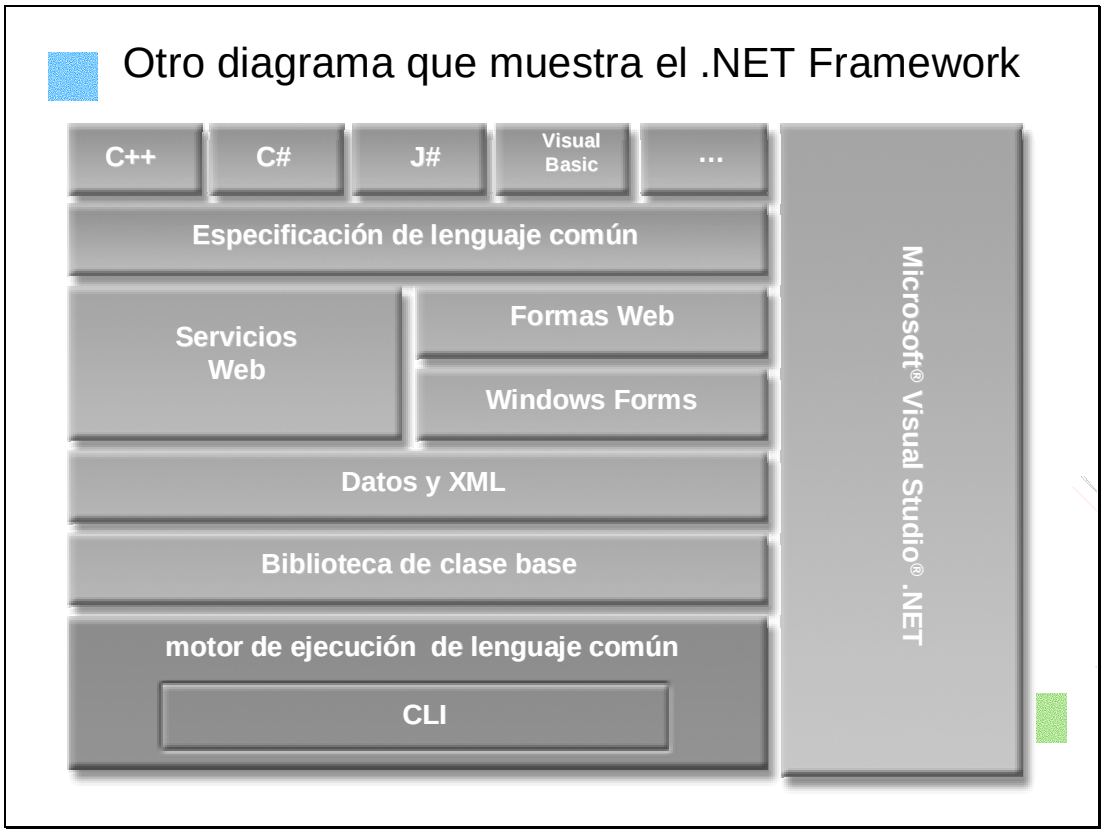
Como esperaría de una biblioteca de clases orientada a objetos, los tipos de .NET Framework le permiten lograr una amplia gama de tareas de programación comunes, incluyendo tareas tales como la administración de cadenas, recolección de datos, conectividad con bases de

datos y acceso a los archivos. Además de estas tareas comunes, la biblioteca de clases incluye tipos que soportan una amplia gama de escenarios especializados de desarrollo. Por ejemplo, puede utilizar .NET Framework para desarrollar los siguientes tipos de aplicaciones y servicios:

Aplicaciones de consola.
Aplicaciones de la GUI de Windows (Windows Forms).
Aplicaciones ASP.NET.
Servicios Web XML.
Servicios Windows

Por ejemplo, las clases de Windows Forms son un conjunto completo de tipos reutilizables que simplifican en gran medida el desarrollo de las GUI de Windows. Si escribe una aplicación Web Form de ASP.NET, puede utilizar las clases de Web Forms.

Diapositiva 10



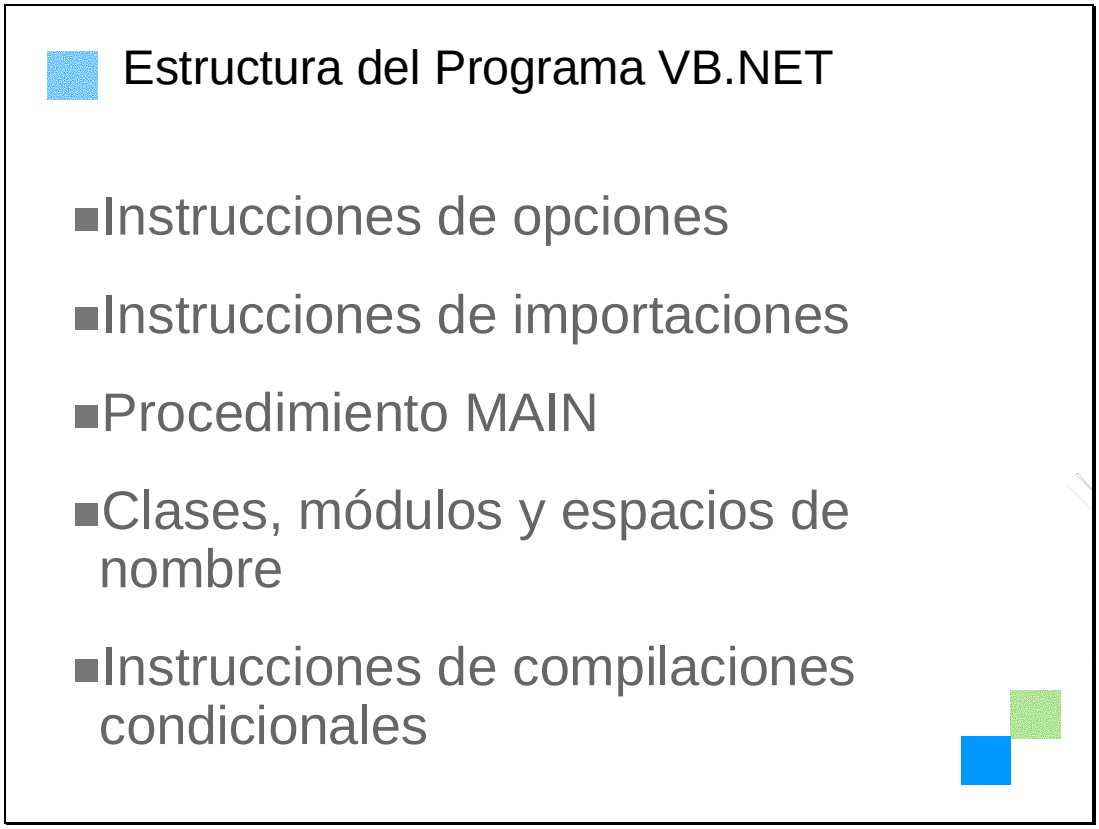
Aquí podemos ver cómo se apilan las cosas. Visual Studio es la herramienta de elección para crear los diferentes tipos de aplicaciones para .NET Framework con los diferentes lenguajes .NET gracias al **CLS** y al **CLR**.

Cuando compila una aplicación VB.NET o cualquier aplicación escrita en un lenguaje que cumple con CLS, la aplicación se compila en MSIL. Este MSIL luego se compila aún más en las instrucciones nativas del CPU cuando la aplicación es ejecutada por primera vez por el CLR. (En realidad, sólo las funciones invocadas se compilan la primera vez que son invocadas).

Veamos todo un poco más a detalle:
Usted compila utilizando el compilador VB.NET (vbc.exe) en un EXE.
El compilador VB.NET genera el código MSIL y un manifiesto hacia una parte de sólo lectura del EXE que cuenta con un encabezado PE (ejecutable portátil) estándar.

Cuando la aplicación se ejecuta, el sistema operativo carga el ejecutable, así como cualesquiera bibliotecas de vínculos dinámicos (DLLs) dependientes.
El cargador del sistema operativo luego brinca al punto de entrada dentro del ejecutable del programa, que es colocado ahí por el compilador VB.NET.
Debido a que el código MSIL no puede ejecutarse directamente, ya que no está en un formato ejecutable por la máquina, el CLR compila la MSIL utilizando un compilador justo a tiempo (JIT) en instrucciones nativas del CPU conforme procesa la MSIL. La compilación JIT ocurre sólo conforme se invocan los métodos en el programa. El código ejecutable compilado se coloca en la memoria caché de la máquina y es recompilado sólo si se realiza algún cambio al código fuente.

Diapositiva 11



Un programa Visual Basic tiene bloques de construcción estándar.

El código de Visual Basic se almacena en los módulos del proyecto. Los proyectos se componen de archivos, que son compilados en las aplicaciones. Cuando inicia un proyecto y abre el editor de códigos, ve algunos códigos que ya están colocados en sitio y en el orden correcto. Cualquier código que escriba deberá seguir esta secuencia:

Instrucciones de **opciones**

Instrucciones de **importación**

El procedimiento **Main**

Instrucciones de **clase**, **módulo** y **espacio de nombre**, de ser necesarias.

Además, un programa puede contener instrucciones condicionales de compilación. Éstas se pueden colocar en cualquier lugar en el módulo. Muchos programadores prefieren colocarlas al final.

Si registra instrucciones en un orden diferente, pueden resultar errores de compilación.

Instrucciones de opciones

Las instrucciones de **opciones** establecen las reglas básicas para el código subsiguiente, ayudando a evitar errores de sintaxis y de lógica. La instrucción **Option Explicit** asegura que todas las variables sean declaradas y deletreadas correctamente, lo cual recorta el tiempo que se invierte en depurar posteriormente. La instrucción de **Option Strict** ayuda a prevenir errores de lógica y pérdida de datos que pueden ocurrir cuando trabaja entre variables de diferentes tipos. La instrucción de **Option Compare** especifica la manera en que se comparan las cadenas entre sí, ya sea por su arreglo **Binario** o de **Texto**.

Instrucciones de importación

Las instrucciones de **importación** le permiten nombrar clases y otros tipos definidos dentro del espacio de nombre importado sin tener que calificarlos.

Procedimiento Main

El procedimiento **Main** es el "punto de inicio " para su aplicación, el primer procedimiento al cual accede cuando ejecuta su código. **Main** es el lugar donde colocaría el código que necesita ser accedido primero. En **Main**, puede determinar cual forma se carga primero cuando se inicia el programa, descubrir si una copia de su aplicación ya se está ejecutando en el sistema, establecer un conjunto de variables para su aplicación o abrir una base de datos que requiere la aplicación. Existen cuatro variables del **Main**

Sub Main()
Sub Main(ByVal CmdArgs() As String)
Function Main() As Integer
Function Main(ByVal CmdArgs() As String) As Integer

La variedad más común de este procedimiento es **Sub Main( )**.

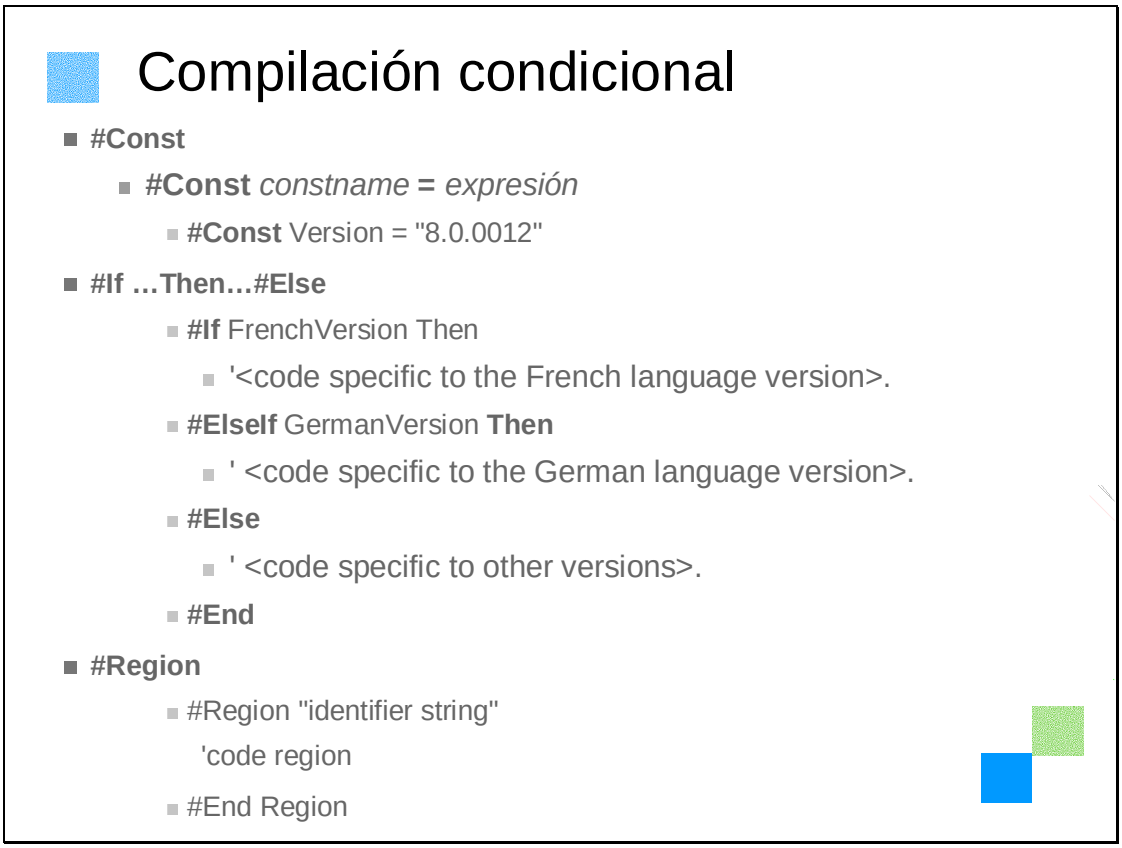
Instrucciones de clase, módulo y espacio de nombre

Las clases y los módulos conforman la mayor parte del código en su archivo fuente. Contienen la mayor parte del código que escribe, principalmente instrucciones **Sub**, **Function**, **Method** y **Event,** junto con declaraciones variables y otros códigos necesarios para que se pueda ejecutar su aplicación.

Instrucciones de compilación condicional

Las instrucciones de compilación condicional puede aparecer en cualquier lugar dentro del módulo. Se configuran para ejecutar si se cumplen ciertas condiciones dentro del motor de ejecución. También puede utilizarlas para depurar su aplicación, ya que el código condicional se ejecuta sólo en modo de depuración.

Diapositiva 12



Puede utilizar la compilación condicional para seleccionar secciones particulares de código para compilarlas, excluyendo al mismo tiempo otras secciones. Por ejemplo, tal vez desee escribir instrucciones de depuración que comparan la velocidad de diferentes enfoques con la misma tarea de programación, o tal vez desee localizar una aplicación para múltiples lenguajes. Las instrucciones de compilación condicional están diseñadas para ejecutarse durante el tiempo de compilación, no durante el motor de ejecución .

Declara un compilador condicional constante en el código con la directiva **#Const,** y denota bloques de código para que se compilen condicionalmente con la directiva **#If...Then...#Else**.

Puede establecer constantes de compilación condicional en una de tres maneras:
En el cuadro de diálogo Páginas de propiedades
En la línea de comandos al utilizar el compilador de línea de comandos
En su código

Las constantes de compilación condicional tienen un alcance especial y no puede acceder a éstas desde el código estándar. El alcance de una constante de compilación condicional depende de la manera en que se configura.

Para establecer constantes en el cuadro de diálogo Páginas de propiedades

Antes de crear su archivo ejecutable, establezca constantes en el cuadro de diálogo Páginas de propiedades al seguir los pasos proporcionados en Configurar propiedades del proyecto de Visual Basic.

Para establecer constantes en la línea de comandos

Utilice el interruptor **/d** para registrar las constantes de compilación condicional, como en el siguiente ejemplo: vbc MyProj.vb /d:conFrenchVersion=1:conANSI=0 No space is required between the **/d** switch and the first constant.

Las instrucciones de línea de comandos sobrepasan las instrucciones registradas en el cuadro de diálogo de Páginas de propiedades, pero no las borra. Los argumentos establecidos en las Páginas de propiedades permanecen en vigor para compilaciones subsiguientes.

Cuando se escriben constantes en el código mismo, no existen reglas estrictas en cuanto a su colocación, ya que su alcance es todo el módulo en el cual son declaradas.

Para establecer constantes en su código

Coloque las constantes en el bloque de declaración del modulo en el que se utilizan. Esto ayuda a mantener su código organizado y más fácil de leer.

Por ejemplo, para crear las versiones en los idiomas francés y alemán de la misma aplicación desde el mismo código fuente, incrusta segmentos de código específicos de la plataforma en instrucciones **#If...Then** utilizando las constantes predefinidas FrenchVersion y GermanVersion.

El siguiente ejemplo demuestra cómo:

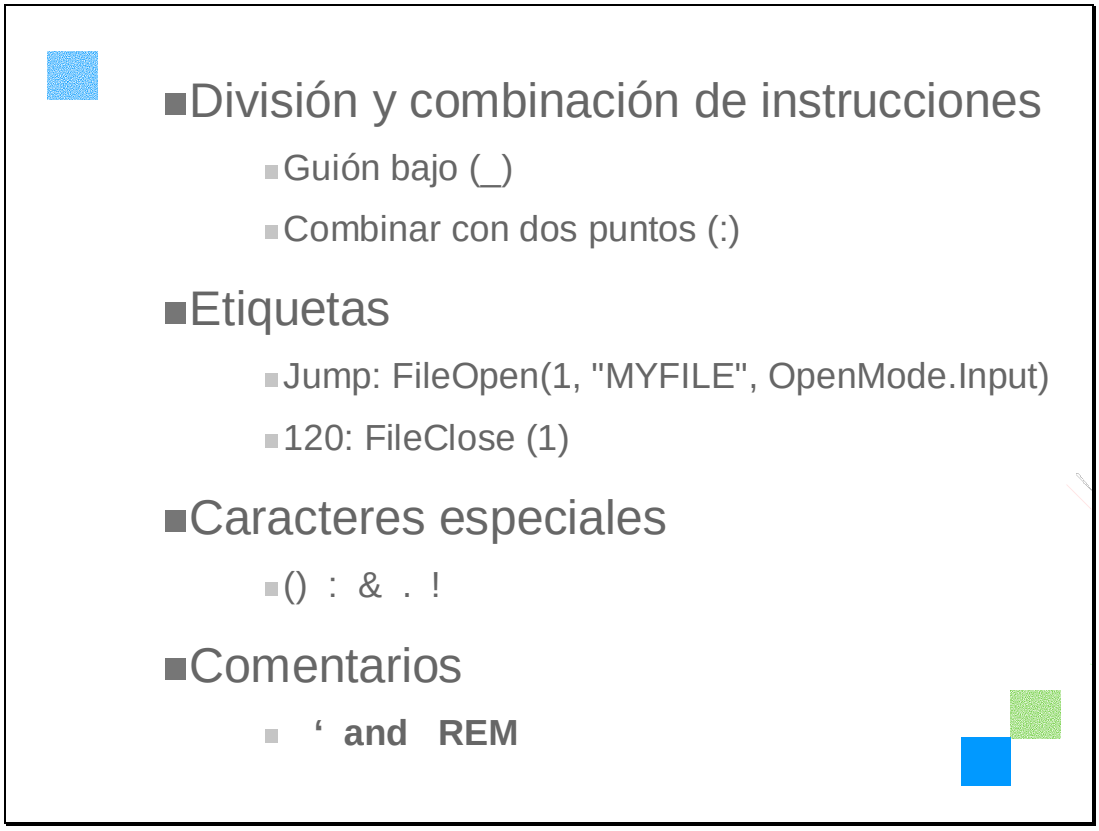
```
#If FrenchVersion Then
'<code specific to the French language version>.
#ElseIf GermanVersion Then
' <code specific to the German language version>.
#Else
' <code specific to other versions>.
#End
```

Si estable el valor de la constante FrenchVersion a **Verdadero** durante el tiempo de compilación, el código condicional para la versión en francés se compila. Si estable el valor de la constante GermanVersion a **Verdadero,** el compilador utiliza la versión en alemán. Si ninguno de los dos se establece a **Verdadero**, se ejecuta el código en el último bloque **Else**.
Nota El completado automático no funcionará al editar el código ni al utilizar las directivas de compilación condicional si el código no es parte de la rama actual.

La directiva **#Region** permite colapsar y ocultar secciones de código en los archivos de Visual Basic .NET. **#Region** le permite especificar un bloque de código que puede expandir o colapsar al utilizar la función Seleccionar y ocultar código del editor de Códigos de Visual Studio .NET. La capacidad de ocultar código selectivamente hace que sus archivos sean más administrables y más fáciles de leer.
Las directivas **#Region** soportan la semántica de bloque de códigos tales como **#If...#End If.** Esto significa que no pueden empezar en un bloque y terminar en otro; el inicio y el fin deben estar en el mismo bloque. Las directivas **#Region** no se soportan dentro de las funciones.

El bloque **#Region** se puede utilizar varias veces en un archivo de códigos; así, los usuarios pueden definir sus propios bloques de procedimientos y clases que pueden, a su vez, ser colapsados. Los bloques **#Region** también se pueden anidar dentro de otros bloques **#Region.**
Nota Los códigos ocultos no evitan que sean compilados, y no afectarán la instrucción **#If...#End If**.

Diapositiva 13



Al escribir su código, en algunas ocasiones puede crear instrucciones largas que necesitan desplazamiento horizontal en el Editor de códigos. Mientras que esto no afecta la manera en que se ejecuta el código, le dificulta a usted o a cualquier otra persona leer el código conforme aparece en el monitor. En tales casos, debe considerar dividir la instrucción larga en varias líneas.

En otros casos, tal vez desee consolidar instrucciones en una sola línea; por ejemplo, cuando cuenta con varias instrucciones muy cortas y desea conservar el espacio. Esta función también puede ser útil al organizar variables o comandos dentro de un módulo.

Para dividir una sola instrucción en varias líneas

Utilice la secuencia de continuación de línea, que es un espacio seguido por un guión largo (_), en el punto en el que desea dividir la línea. En el siguiente ejemplo, la instrucción se divide en cuatro líneas con secuencia de continuación de línea terminando todo excepto la última línea:

```
Data1.RecordSource = _
"SELECT * FROM Titles, Publishers" _
```

& "WHERE Publishers.PubId = Titles.PubID" _
& "AND Publishers.State = 'CA'"
Utilizar esta secuencia permite que su código sea más fácil de leer, tanto en línea como impreso.

Nota La secuencia de continuación de línea es necesariamente la última cosa en una línea. No puede seguirla con ninguna otra cosa en la misma línea. Existen algunas limitaciones en cuanto a donde se puede utilizar la secuencia de continuación de línea, tal como en medio de un nombre de argumento. Puede dividir una lista de argumento con la secuencia de continuación de línea, pero los nombres individuales de los argumentos deben permanecer intactos.

Nota No puede continuar un comentario utilizando una secuencia de continuación de línea. Una vez que empieza el comentario, el compilador no examina los caracteres en busca de un significado especial. Para un comentario de líneas múltiples, repita el símbolo de comentario (') en cada línea. Mientras que el método que se recomienda es colocar cada instrucción en una línea por separado, Visual Basic también le permite colocar múltiples instrucciones en la misma línea.

Para colocar múltiples instrucciones en la línea

Separe las instrucciones con dos puntos (**:**), como en el siguiente ejemplo:
Text1.Text = "Hello" : Red = 255 : Text1.BackColor = Red

Etiquetas:

Los bloques de instrucciones se conforman de líneas de código delimitadas por dos puntos. Se dice que las líneas de código que están precedidas de una cadena identificadota o un número entero están *etiquetadas*. Las etiquetas de instrucciones se utilizan para marcar una línea de código para identificarla para su uso con instrucciones tales como **On Error Goto**.

Nota La etiquetas se pueden utilizar sólo en instrucciones ejecutables dentro de los métodos.

Las siguientes líneas de códigos están etiquetadas con Jump y 120, respectivamente:
Jump: FileOpen(1, "MYFILE", OpenMode.Input)
120: FileClose (1)

Las etiquetas pueden ser ya sea identificadores Visual Basic .NET válidos, tales como aquellos que identifican los elementos de programación o literales de números enteros. Una etiqueta debe aparecer al inicio de una línea del código fuente y debe estar seguida por dos puntos, sin importar si es seguida por una instrucción en la misma línea.

El compilador identifica etiquetas al verificar si el inicio de la línea corresponde con cualquier identificador ya definido. Si no es así, el compilador supone que es una etiqueta.
Las etiquetas tienen su propio espacio de instrucción y no interfieren con otros identificadores. El alcance de una etiqueta es el cuerpo del método. La instrucción de la etiqueta toma precedencia en cualquier situación ambigua.

Para etiquetar una línea de código

Coloque un identificador, seguido por dos puntos, al inicio de la línea del código fuente.

Caracteres especiales:

En ocasiones, necesita utilizar caracteres especiales en su código; esto es, caracteres que no se encuentran en la lista alfanumérica estándar. La puntuación y los caracteres especiales en el conjunto de caracteres de Visual Basic .NET tienen varios usos, desde organizar el texto del programa hasta definir las tareas que el compilador o el programa compilado realiza. No especifican que se realice una operación.

Paréntesis

Utilice paréntesis al definir un procedimiento, tal como un **Sub o Function**. Debe encerrar todos los argumentos del procedimiento en paréntesis. También utiliza paréntesis para colocar variables o argumentos en grupos lógicos.

Separadores

Los separadores realizan lo que su nombre sugiere: separan secciones de código. En Visual Basic, el caracter del separador son los dos puntos (**:**). Utilice los separadores cuando desee colocar múltiples instrucciones en una sola línea en lugar de en líneas separadas, ahorrando espacio y mejorando la capacidad de lectura de su nombre. El siguiente ejemplo muestra tres instrucciones separadas por dos puntos (**:**):

```
a = 5: b = 10: c = 15
```

Concatenación

Utilice el operador "**&**" para la *concatenación,* o para unir cadenas. No lo confunda con el operador **+** que une valores numéricos. Utilizar el operador **"+"** para concatenar puede provocar resultados incorrectos al operar en dos valores numéricos. El siguiente código proporciona un ejemplo:

```
Var1 = "10.01"
Var2 = 11
Result = Var1 + Var2 ' Result = 21.01
Result = Var1 & Var2 ' Result = 10.0111
```

Operadores de acceso a miembros

Para acceder a un miembro de un tipo, utiliza el punto (**.**) o el operador del punto de exclamación (**!**) entre el nombre de tipo y el nombre del miembro. El tipo puede ser una enumeración, estructura, interfaz o clase y el miembro puede ser un campo, propiedad, evento o método.

Operador Punto (.)

Utilice el operador . como un operador de acceso al miembro para propiedades, eventos, campos y métodos, como en el siguiente ejemplo:

```
Dim MyForm As New System.Windows.Forms.Form
MyForm.Text = "This is my form" ' Accesses Text member (property) of Form class (on MyForm object). MyForm.CenterToScreen() ' Accesses CenterToScreen member (method) on MyForm.
```

Operador de símbolo de exclamación (!)

Utilice el operador **!** (sólo en una clase o interfaz) como un operador de acceso a diccionario. La clase o la interfaz debe tener una propiedad predeterminada que acepta un solo argumento de **Cadena**. El identificador que sigue inmediatamente al operador **!** se convierte en el argumento de cadena para la propiedad predeterminada, como en el siguiente ejemplo:

```
La Clase pública HasDefault ' expone una propiedad predeterminada.
Default Public ReadOnly Property Index(ByVal S As String) As Integer
Get
        Return 1000 + CInt(S)
End Get
End Property ' Index
End Class ' HasDefault
' ...
Public Class TestHasDefault
Public Sub CompareAccess()
Dim HD As HasDefault = New HasDefault()
Dim IndexStr As String = "5"
MsgBox("Traditional access returns " & HD.Index(IndexStr) & vbCrLf & _
        "Default property access returns " & HD(IndexStr) & vbCrLf & _
        "Dictionary access returns " & HD!IndexStr)
End Sub
```

```
End Class ' TestHasDefault
```

Las tres líneas de salidas de **MsgBox** muestran todas el valor 1005. La primera línea utiliza el acceso tradicional a Índice de propiedad, la segunda hace uso del hecho de que el Índice es la propiedad predeterminada de la clase HasDefault, y la tercera utiliza el acceso del diccionario a la clase.

Observe que el segundo operando del operador **!** debe ser un identificador o contraseña válida. Por lo tanto, el siguiente cambio a la última línea de la llamada **MsgBox** genera un error porque "5" no es ni un identificador ni una contraseña:

```
"Dictionary access returns " & HD!"5")
```

Nota Las referencias a las colecciones predeterminadas deben ser explícitas. En particular, no puede utilizar el operador **!** en una variable que se ha unido de manera tardía.

El caracter **!** también se utiliza como el caracter de tipo **Individual**.

Limitaciones

Las versiones anteriores de Visual Basic obligaban límites en el código, tales como la longitud de los nombres de variable, el número de variables permitidas en los módulos y el tamaño del mismo. En Visual Basic .NET, estas restricciones se han eliminado, ofreciéndolo una mayor libertad para escribir y arreglar su código. Siguen existiendo límites físicos, pero se configuran tan alto que no existe ninguna posibilidad de que los exceda y provoque errores de compilación dentro de su código.

Diapositiva 14

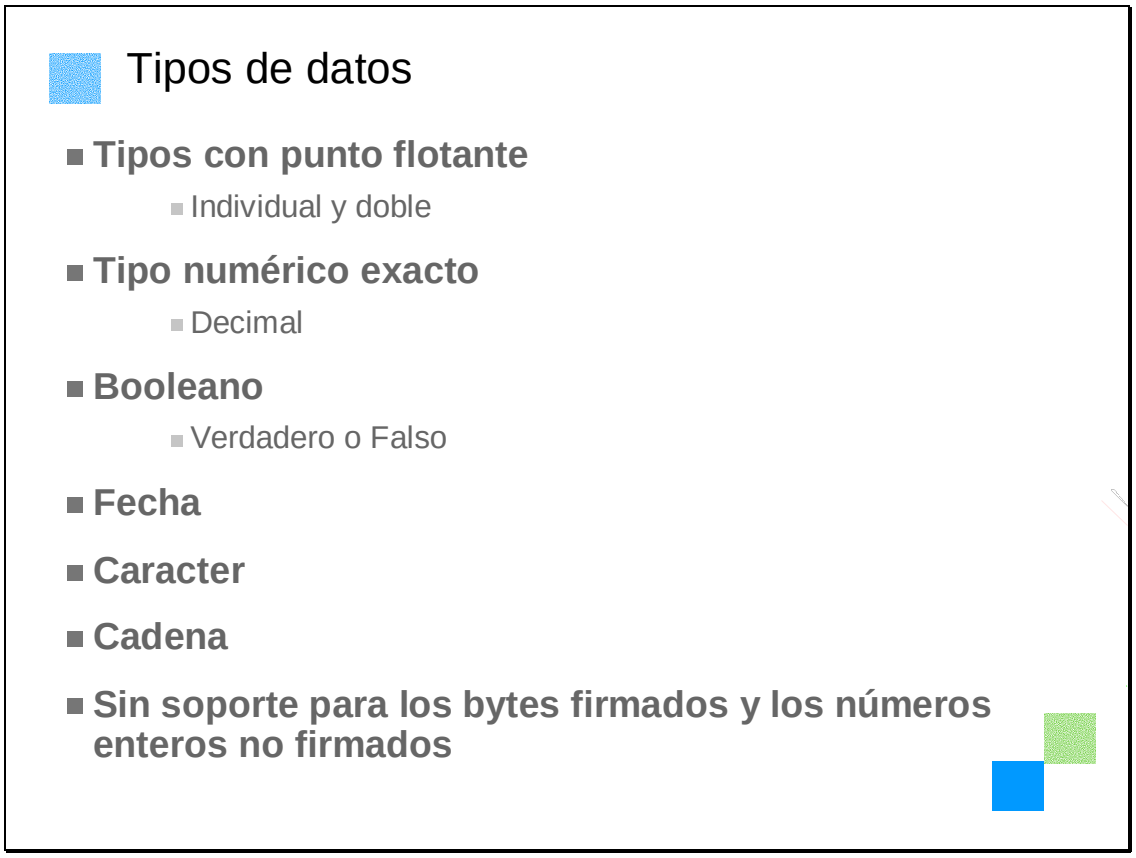
Tipos de datos

Tipos de números enteros

- Visual Basic.NET soporta tipos de números enteros que van desde 8 hasta 64 bits.

Tipos # enteros	VB 6	VB.NET	Tipo CLR
8 bits	**Byte**	**Byte**	**System.Byte**
16 bits	**Entero**	**Corto**	**System.Int16**
32 bits	**Largo**	**Entero**	**System.Int32**
64 bits	**n/a**	**Largo**	**System.Int64**

Diapositiva 15



Tipos de puntos de flotamiento

Existen dos tipos diferentes de puntos flotantes: Simple, que es un valor numérico de 4 bytes y Doble, que es un valor de 8 bytes. Ambos tipos son compatibles con IEEE y son idénticos con los tipos de Framework System.Single y System.Double, respectivamente.

Tipo numérico exacto

Para un cálculo exacto, existe un Decimal de tipo numérico, que puede retener 28 dígitos. Este tipo reemplaza el tipo *Currency*, que era menos preciso y menos flexible.

Booleano

El tipo Booleano representa uno de los dos estados “verdadero” o “falso.” Una variable Booleana puede configurar a través de las palabras claves *True* y *False* o del resultado de una expresión. Cuando otros tipos numéricos se convierten a valores Booleanos, 0 se convierte en Falso y todos los demás valores se convierten en Verdadero. Cuando los valores

Booleanos se convierten a otros tipos de datos, Falso se convierte en 0 y Verdadero se convierte en -1.

Fecha

Las variables de fecha se almacenan como números enteros largos (8 bytes) de 64 bits de IEEE que representan fechas que van desde el 1 de enero 1 CE (el año 1) hasta el 31 de diciembre 9999 y horas de 0:00:00 a 23:59:59.

Caracter

Las variables de caracteres se almacenan en números de 16 bits (2 bytes) que van en valor desde 0 hasta 65535. Cada número de 2 bytes almacena un carácter Unicode individual. Las conversiones implícitas entre el tipo de datos de Caracter y los tipos numéricos no son posibles, pero la conversión explícita entre el tipo de datos de Caracter y los tipos numéricos sí se soporta.

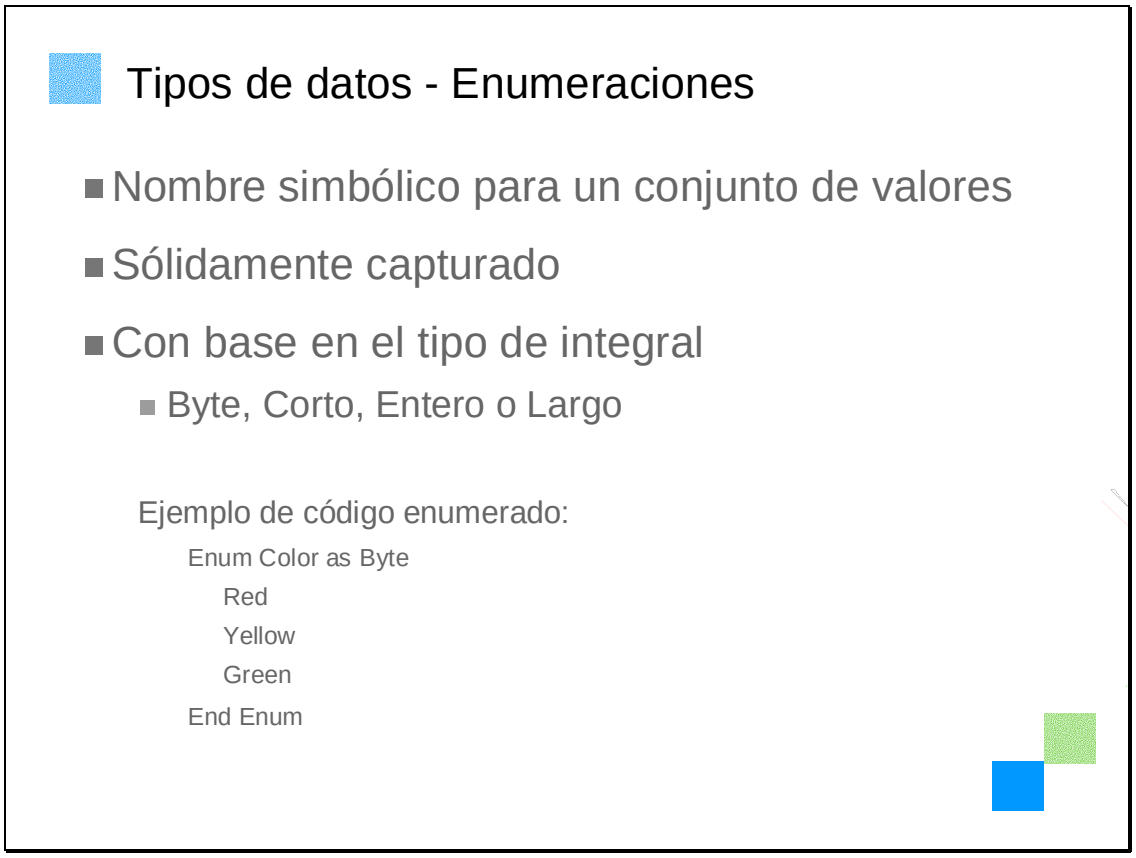
Cadena

Una cadena puede contener hasta aproximadamente 2,000 millones (2 ^ 31) de caracteres de Unicódigo. Parece natural que una estructura tan potencialmente larga sea un tipo de referencia.

Sin soporte para bytes con signo y números enteros sin signo

Es importante observar que los bytes con signo y los tipos de números enteros sin signo no son soportados en Visual Basic.NET. Como resultado, existen cuatro tipos de valores en el sistema de tipos que son ilegales de usar en un programa: *System.SByte, System.UInt16, System.UInt32 y System.UInt64. Cualquier referencia a estos tipos generará un error.*

Diapositiva 16



Enumeraciones

Nombre simbólico para un conjunto de valores

Una enumeración (Enum) es una forma especial de tipo de valor que se hereda de System.Enum y proporciona un nombre alternativo para un tipo primitivo subyacente. Un tipo Enum tiene un nombre, un tipo subyacente y un conjunto de campos. Los campos son campos literales estáticos, cada uno de los cuales representa una constante. A cada campo se le asigna un valor específico del tipo subyacente a través del compilador de lenguaje. Se puede asignar el mismo valor a múltiples campos. Cuando esto ocurre, el compilador marca exactamente uno de los valores Enum como un valor Enum “primario” para el valor, con fines de reflexión y conversión a cadenas.

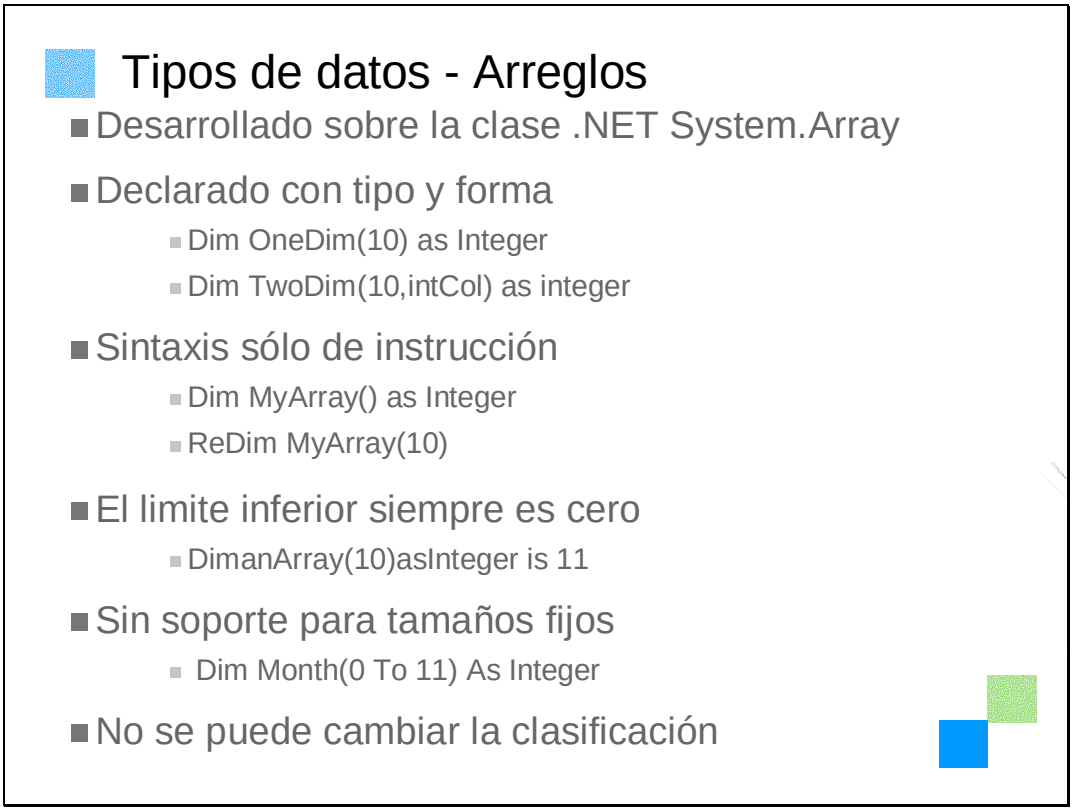
Sólidamente capturado

Las enumeraciones siempre se capturan de manera sólida y no son intercambiables con tipos de números enteros.

Con base en el tipo de número entero

El tipo subyacente debe ser uno de los tipos de números enteros integrados (Byte, Corto, Entero o Largo).

Diapositiva 17



Desarrollado sobre la clase .NET System.Array

Todos los arreglos declarados en Visual Basic.NET se basan en la clase System.Array de .NET Framework y, por lo tanto, tiene todas sus capacidades, tales como clasificar y buscar.

Declarado con tipo y forma

Cuando se declara un arreglo, deben especificarse el tipo subyacente y el rango.

Sintaxis sólo de instrucciones

Debido a que los arreglos en Visual Basic.NET son en realidad objetos y tipos de referencia, no se requiere que sus dimensiones sean especificadas al momento de la instrucción.

El limite inferior siempre es cero

Los arreglos en Visual Basic.NET son capturados de manera sólida y se basan en cero. Esto significa que el primer elemento de cada dimensión en el arreglo se puede encontrar en la posición 0 del índice. El número de elementos de un arreglo declarado con Dim anArray(10) como número entero es 11: el número dado se trata como el índice más alto.

Sin soporte para tamaños fijos

En Visual Basic 6.0, puede especificar el tamaño de un arreglo en su instrucción como en el siguiente ejemplo:

```
Dim Month(0 To 11) As Integer
```

Esto causa que el arreglo cuente con un tamaño fijo, que no se puede cambiar con la declaración ReDim. Esto ya no se soporta.

No se puede cambiar la clasificación

Aunque el tamaño de un arreglo se puede cambiar en Visual Basic.NET, se debe fijar el número de dimensiones.

Diapositiva 18

Operadores

Operadores aritméticos

- ^, *, /, \, Mod, +, -

Operadores de asignación

- =, ^=, *=, /=, \=, +=, -=, <<=, >>=, &=
  - ejemplo:

```
Dim var1 as Integer = 5
Dim var2 as Integer = 2
var1 *= var2      ‘ The value of var1 is now 10
```

Operadores de comparación

Operador:	Verdadero si	Falso si
< (Menor a)	expresión1 < expresión2	*expresión1* >= *expresión2*
<= (Menor o igual a)	*expresión1* <= *expresión2*	*expresión1* > *expresión2*
> (Mayor a)	*expresión1* > *expresión2*	*expresión1* <= *expresión2*
>= (Mayor o igual a)	*expresión1* >= *expresión2*	expresión1 < expresión2
= (Igual a)	*expresión1* = *expresión2*	*expresión1* <> *expresión2*
<> (No igual a)	*expresión1* <> *expresión2*	*expresión1* = *expresión2*

Operadores aritméticos

Estos operadores realizan operaciones aritméticas como multiplicación, división, resta, etc.

^, *, /, \, Mod, +, -

Operadores de asignación

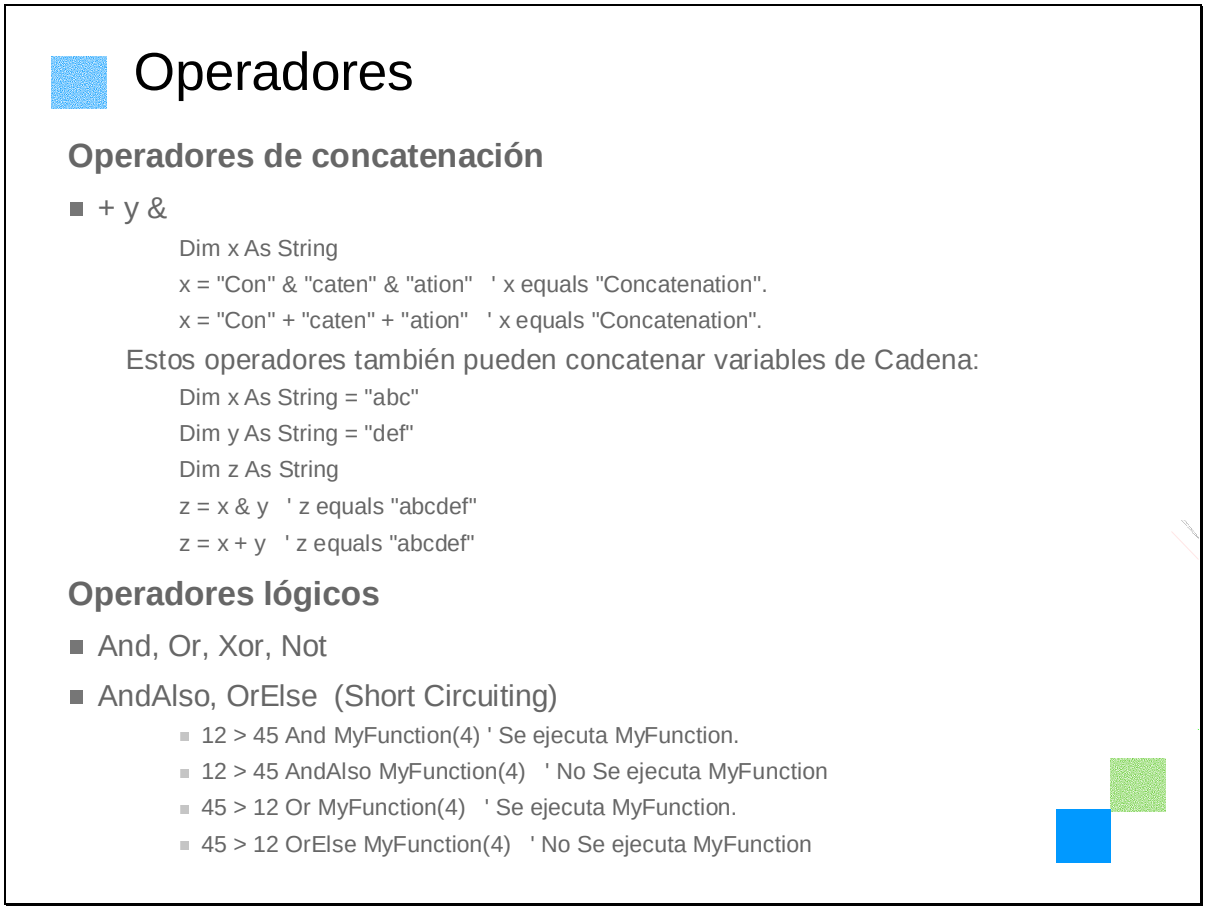
=, ^=, *=, /=, \=, +=, -=, <<=, >>=, &=

Los operadores, con excepción del operador “=” que sólo realiza una asignación, primero realizará la operación a la izquierda del signo “=” entre los dos lados del signo “=” y luego asignará el resultado a la variable en el lado izquierdo.

Por ejemplo:

```
Dim var1 as Integer = 5
Dim var2 as Integer = 2
var1 *= var2      ‘ El valor de Var1 ahora es 10
```

Diapositiva 19



Operadores de concatenación

Existen dos operadores de concatenación: + y &; ambos realizan la operación básica de concatenación, como se muestra a continuación:

```
Dim x As String
x = "Con" & "caten" & "ation"   ' x equals "Concatenation".
x = "Con" + "caten" + "ation"   ' x equals "Concatenation".
```

Estos operadores también pueden concatenar las variables de Secuencia de códigos, como en el siguiente ejemplo:

```
Dim x As String = "abc"
Dim y As String = "def"
Dim z As String
z = x & y   ' z equals "abcdef".
z = x + y   ' z equals "abcdef".
```

Operadores lógicos

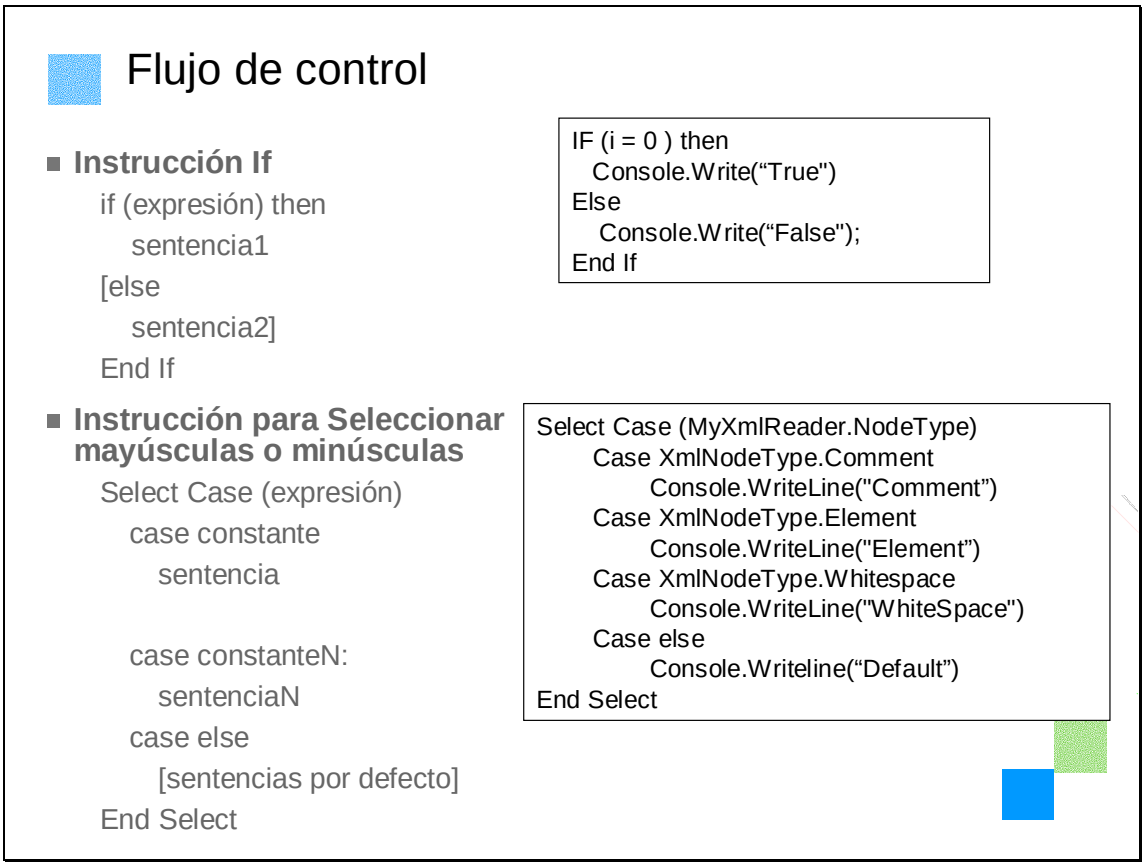
Los operadores lógicos comparan expresiones booleanas y regresan un resultado booleano. Los operadores And, Or, AndAlso, OrElse y Xor toman dos operandos, y el operador Not toma un solo operando.

El operador AndAlso y OrElse son muy similares al operador And y al operador Or , en cuanto a que también realiza una conjunción lógica en dos expresiones booleanas. La diferencia clave entre ellos es que AndAlso y OrElse exhiben un comportamiento de circuito corto. Si la primera expresión en una expresión AndAlso se evalúa a Falso, entonces la segunda expresión no es evaluada y Falso se regresa para la expresión AndAlso.

De manera similar, el operador OrElse realiza una disyunción lógica de circuito corto en dos expresiones booleanas. Si la primera expresión en una expresión OrElse evalúa a Verdadera, entonces la segunda expresión no es evaluada y Verdadero se regresa para la expresión OrElse. A continuación aparecen algunos ejemplos que ilustran la diferencia entre And, Or y sus contrapartes:

```
12 > 45 And MyFunction(4) ' MyFunction() is called.
12 > 45 AndAlso MyFunction(4)   ' MyFunction() is not called.
45 > 12 Or MyFunction(4)   ' MyFunction is called.
45 > 12 OrElse MyFunction(4)   ' MyFunction is not called
```

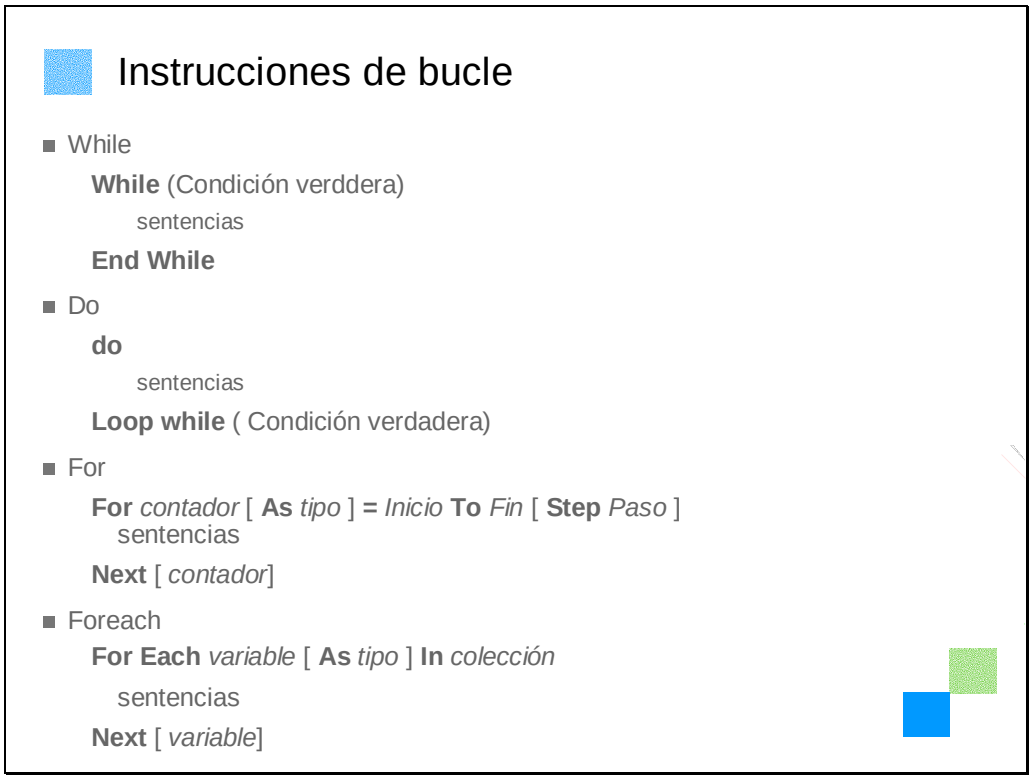
Diapositiva 20



Utiliza instrucciones de flujo de control para determinar qué código se debe ejecutar y cuándo se debe ejecutar. VB.NET tiene dos instrucciones que ayudan con esto: la instrucción if que ejecuta el código con base en una condición Booleana, y la instrucción select, utilizada para ejecutar el código con base en un valor. Las instrucción más comúnmente utilizada es la instrucción if.

Al utilizar la instrucción select, puede especificar una expresión que regresa un valor integral y una o más piezas de código que se ejecutarán dependiendo del resultado de la expresión. Es similar a utilizar múltiples instrucciones if/else, pero aunque puede especificar múltiples (posiblemente sin relación) instrucciones condicionales con múltiples instrucciones if/else, una instrucción select consiste en sólo una instrucción condicional seguida por todos los resultados que su código está preparado para manejar.

Diapositiva 21



En VB.NET las instrucciones while, do/while, for y foreach le permiten realizar bucles. En cada caso, una instrucción simple o compuesta especificada se ejecuta hasta que una expresión Booleana se resuelve a falso, excepto para el caso de la instrucción foreach, que se utiliza para recorrer una lista de objetos.

While
While (Condición verddera)
sentencias
End While

Ejemplo

```
While ( Not MyFile.EOF )
  Console.Write(MyFile.Read())
End While
```

Do

do

sentencias

Loop while (Condición verdadera)

Ejemplo

```
Do
    Number = Number - 1
    Counter = Counter + 1
Loop While Number > 6
```

For

For *contador* [**As** *tipo*] **=** *Inicio* **To** *Fin* [**Step** *Paso*] sentencias

Next [*contador*]

Ejemplo

```
For i=0 to 10 step 1
  Console.Write(i)
Next i
```

Foreach

For Each *variable* [**As** *tipo*] **In** *colección*

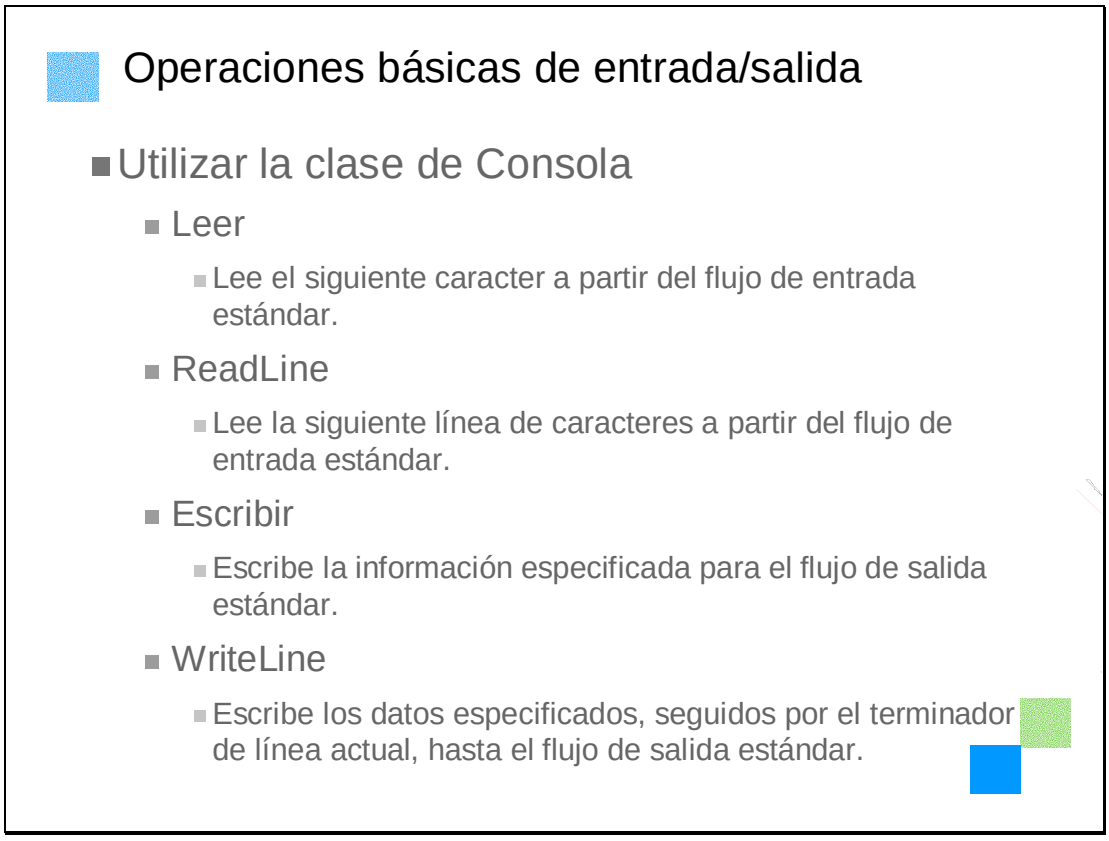
sentencias

Next [*variable*]

Ejemplo

```
Dim myControl As System.Windows.Forms.Control
For Each myControl In myForm.Controls
    myControl.BackColor = System.Drawing.Color.LightBlue
Next myControl
```

Diapositiva 22

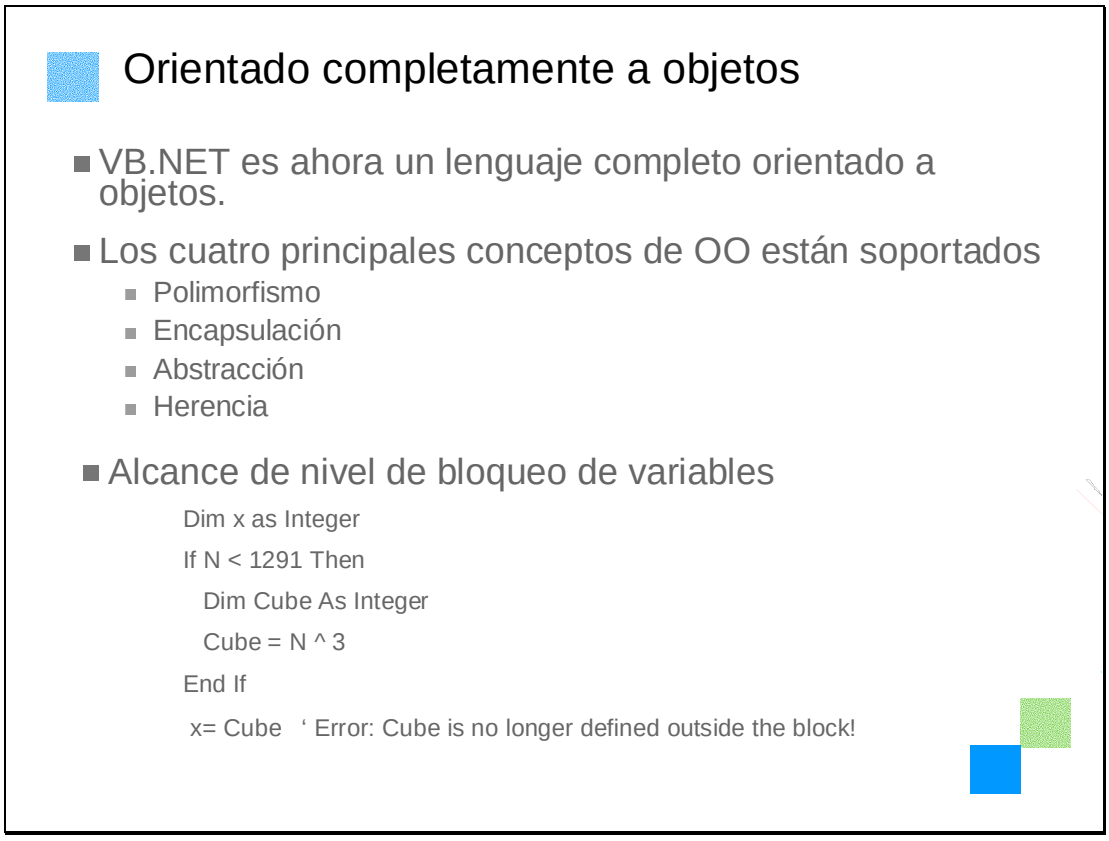


Representa los flujos de entrada, salida y de error estándar para las aplicaciones de la consola. Esta clase no se puede heredar.

La clase de Consola proporciona soporte básico para aplicaciones que leen caracteres desde y escriben caracteres hacia la consola. Si la consola no existe, como en una aplicación basada en Windows, las escrituras hacia la consola no aparecen y no se presenta excepción alguna.

Los datos de la consola se leen a partir del flujo de entrada estándar; los datos normales hacia la consola se escriben hacia el flujo de salida estándar; y los datos de error hacia la consola se escriben hacia el flujo de salida de errores estándar. Estos flujos se asocian automáticamente con la consola cuando inicia su aplicación y se le presentan como las propiedades **In**, **Out** y **Error**.

Diapositiva 23



VB.NET es ahora un lenguaje completo orientado a objetos. Los cuatro conceptos principales de OO están soportados: Polimorfismo, encapsulación, abstracción y herencia.

Polimorfismo

Polimorfismo significa que podemos tener dos clases con diferentes implementaciones o códigos, pero con un conjunto común de métodos y propiedades. La manera tradicional de utilizar el polimorfismo en Visual Basic era con interfaces. Las interfaces todavía se pueden utilizar para este propósito, pero ahora tiene la opción de utilizar la herencia para proporcionar polimorfismo. Como con otros aspectos orientados a objetos, existen ventajas tanto para las interfaces como para la herencia. Debe utilizar la herencia cuando desee crear funcionalidad básica que las clases derivadas pueden extender. Las interfaces son mejores en situaciones donde la funcionalidad similar necesita ser proporcionada por múltiples implementaciones que tienen poco en común.

Encapsulación

Tal vez el más importante de los conceptos orientados a objetos. La encapsulación es el concepto de que un objeto debe separar totalmente su interfaz de su implementación. Todos los datos y los códigos de implementación para un objeto deben ocultarse detrás de su interfaz. En otras palabras, un objeto debe ser como una “Caja negra”.

Abstracción

La abstracción es el proceso por medio del cual podemos pensar acerca de propiedades o comportamientos específicos sin pensar acerca de un objeto en particular que tiene esas propiedades o comportamientos. Es meramente la capacidad para un lenguaje de poder crear una “Caja negra” de código para tomar un concepto y crear una representación abstracta de ese concepto dentro de una aplicación.

Herencia

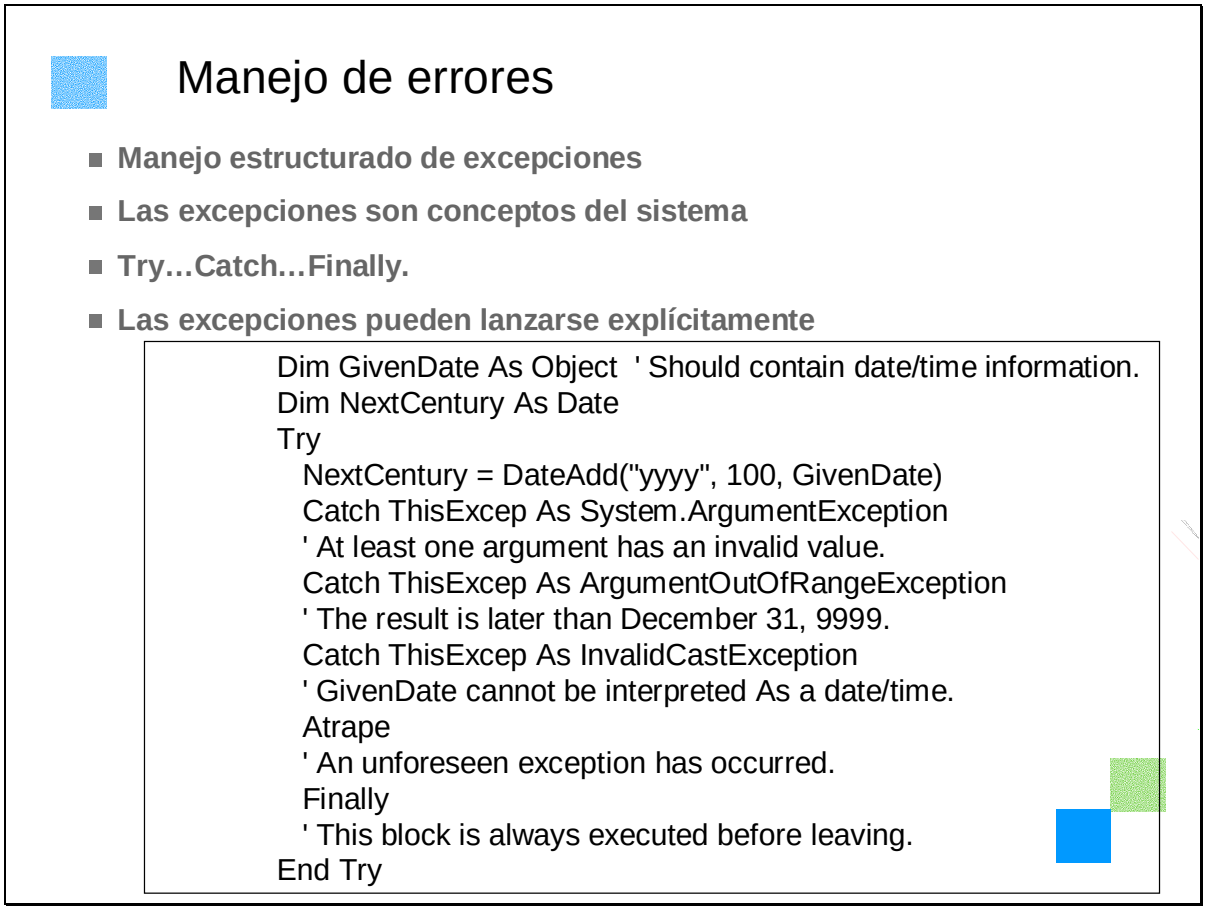
El concepto de herencia es la herramienta principal para lograr la reutilización del código por un lado y una estructura bien formada de un programa por otro. El concepto subyacente de una clase describe una unidad que consiste de datos y rutinas que funcionan en o con esos datos. Al declarar meramente una relación de herencia entre una clase y otra, la clase que hereda obtiene toda la funcionalidad que proporciona la clase que hereda gratuitamente. Y luego en la clase que hereda, la funcionalidad se puede agregar, cambiar o afinar para necesidades especificas.

Se dice que la clase heredada es la clase base; se dice que la clase que hereda se dice que es la clase *derivada*.

ALCANCE

VB.NET ahora soporta el alcance a nivel de bloque de variables. Un bloque es un conjunto de instrucciones terminadas por una instrucción End, Else, Loop o Next; por ejemplo, dentro de una construcción For...Next o If...Then...Else...End If. Un elemento declarado dentro de un bloque se puede utilizar sólo dentro de ese bloque. En el siguiente ejemplo, el alcance del Cubo de variables de enteros es el bloque entre If y End If, y ya no se puede hacer referencia a Cubo cuando la ejecución pasa fuera del bloque:

Diapositiva 24



Manejo estructurado de excepciones

Las excepciones son conceptos del sistema

Con un enfoque estructurado, el cual es implementado por el Motor de ejecución común de los lenguajes .NET, es posible atrapar una excepción en un nivel externo de la ejecución. Incluso es posible tener excepciones anidadas: si un manejador de errores por si mismo lanza una excepción, también puede ser atrapada y todavía se puede examinar la razón del error original.

Forma sintáctica de manejo

Un manejo de excepción se implementa utilizando la instrucción de bloque Try…Catch…Finally.

El motor de ejecución empieza a ejecutar el bloque Try. Cuando se lanza una excepción, el motor de ejecución examina la pila de llamadas, una lista de las llamadas de función creadas por el programa que se está ejecutando actualmente. Esta ruta de ejecución puede tener varias funciones. Por ejemplo, si Main() invoca FirstMethod() y ese método invoca

SecondMethod() y ese método invoca ThirdMethod(), todos éstos están en la pila de invocaciones.

La excepción asciende por la pila de invocaciones a cada bloque de excepciones que los abarca. Debido a que la pila no es modificada, cada bloque de excepción recibe la oportunidad de manejar la excepción. Si la excepción puede ser resuelta por una de las instrucciones Catch, ejecuta la instrucción de manejo. Si la excepción no concuerda con ninguna de las instrucciones Catch en el bloque de excepción, la pila está nuevamente sin modificación y la excepción se presenta al siguiente método. Si la excepción recorre todo el camino hacia el inicio del programa (por ejemplo, a Main()) y no ha sido manejada, el motor de ejecución muestra un cuadro de diálogo que declara que ha ocurrido una excepción no manejada.

Observe que el desenrollado de pila sólo va en una dirección. Debido a que la pila no está dañada, se destruyen los objetos. Una vez que se maneja la excepción, el programa continúa después del bloque Try de la instrucción Catch que manejó la excepción.

Si el bloque Try ha sido ejecutado sin haber lanzado una excepción, o un bloque Catch (un manejador de excepciones) ha sido ejecutado, se ejecuta el bloque Finally correspondiente al bloque Try o Catch que acaba de terminar. Este bloque normalmente se limpia después del algoritmo.

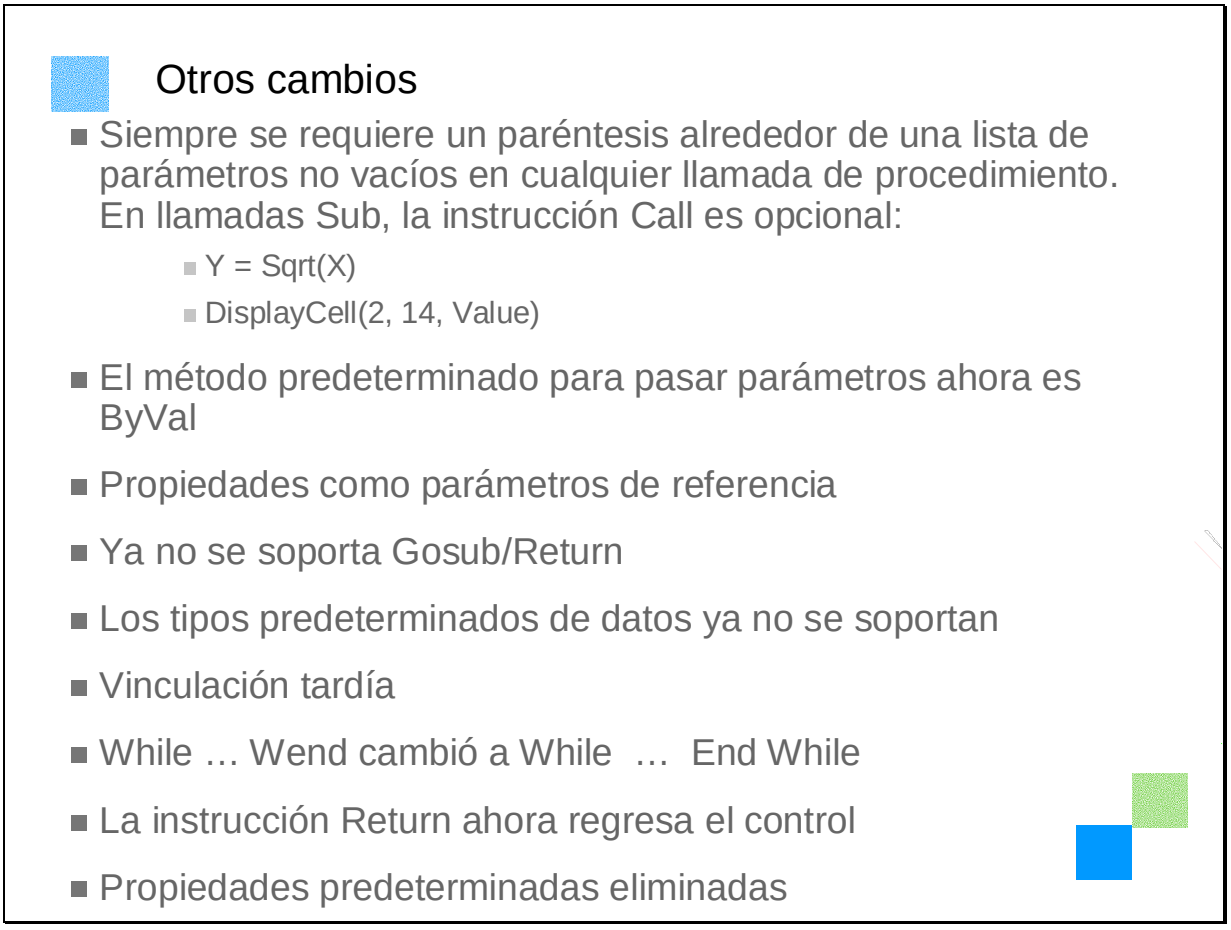
Las excepciones pueden lanzarse explícitamente

Las excepciones normalmente ocurren cuando una rutina del sistema detecta un error. Pero puede proporcionar excepciones propias al declarar una clase que hereda de una de las clase de excepción predefinidas. Y puede lanzar una excepción manualmente en cualquier momento utilizando la contraseña Throw.

```
Dim GivenDate As Object  ' Debe contener información de fecha y hora.
    Dim NextCentury As Date
    Try
      NextCentury = DateAdd("yyyy", 100, GivenDate)
    Catch ThisExcep As System.ArgumentException
      ' al menos un argumento tiene un valor inválido.
    Catch ThisExcep As ArgumentOutOfRangeException
      ' El resultado es posterior al 31 de diciembre de 9999.
    Catch ThisExcep As InvalidCastException
      ' GivenDate no puede interpretarse como fecha y hora válidos.
```

```
Catch
   ' Excepción no considerada.
Finally
   ' Esto se ejecuta antes de continuar, en cualquier caso.
End Try
```

Diapositiva 25



Otros cambios

Paréntesis alrededor de listas de parámetros no vacías

Siempre se requiere un paréntesis alrededor de una lista de parámetros no vacía en cualquier llamada de procedimiento. En llamadas Sub, la instrucción Call es opcional:

Y = Sqrt(X)

DisplayCell(2, 14, Value)

Si está invocando un procedimiento sin proporcionar ningún parámetro, puede incluir paréntesis vacíos o dejarlos fuera totalmente.

El método predeterminado para pasar parámetros ahora es ByVal

En Visual Basic.NET, los valores predeterminados de mecanismos que pasan a ByVal para cada parámetro cuando declara un procedimiento. Esto protege los parámetros contra modificaciones.

Propiedades como parámetros de referencias

Un parámetro de propiedad que pasa ByRef se copia tanto dentro como fuera del procedimiento. Esto es en contraste con el comportamiento de Visual Basic 6, en donde dichos parámetros fueron tratados como ByVal.

Ya no se soporta Gosub/Return

Los subprocedimientos invocados por Gosub ya no están soportados, porque ya no son considerados como estructurados. En cambio se deben utilizar los Subs regulares.

Los tipos predeterminados de datos ya no se soportan

Las instrucciones de tipos de datos predeterminados (DefInt, DefStr, etc) ya no están soportados. Esto se realizó para mejorar la claridad de las instrucciones.

Unión tardía

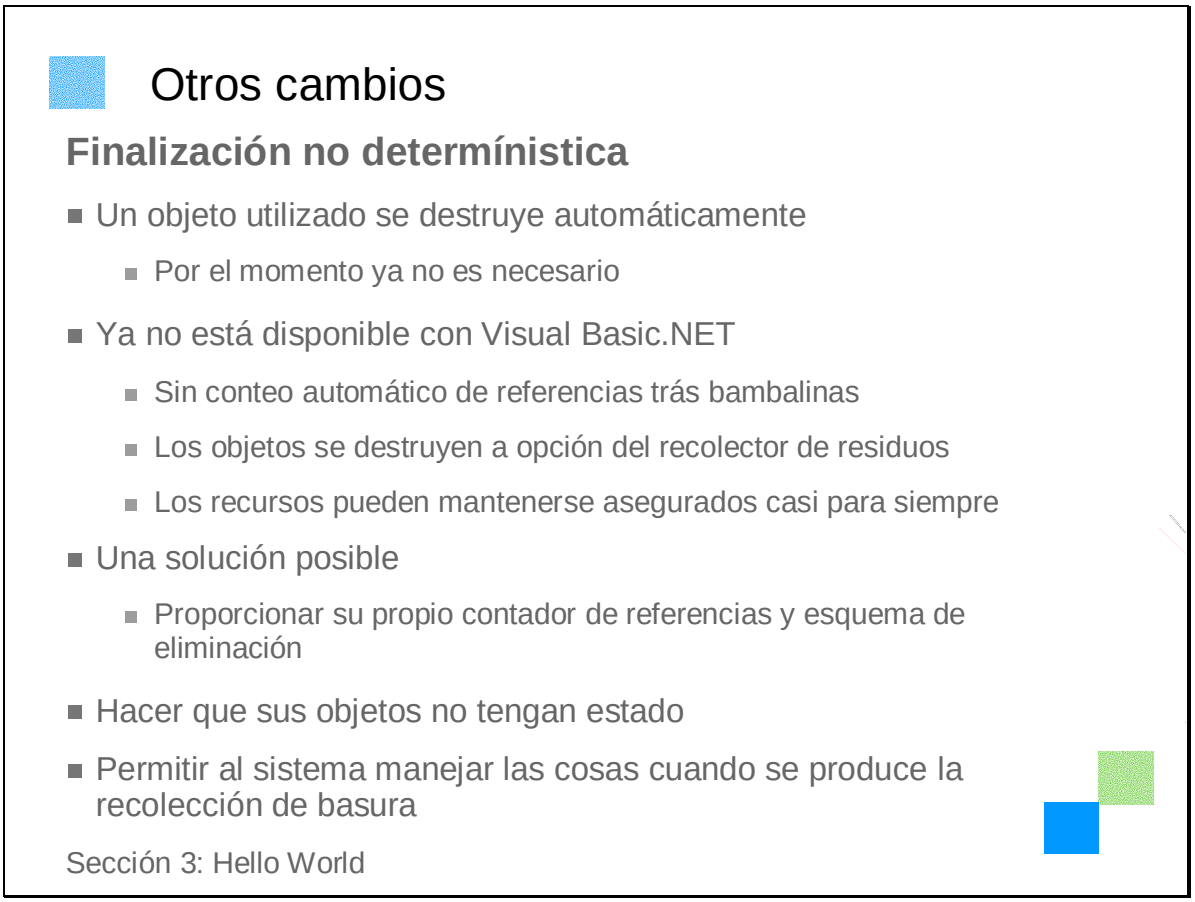
En Visual Basic 6.0, puede asignar una referencia de interfaz a una variable de Objeto tipo, permitiendo un acceso de unión tardía a los métodos y las propiedades de la interfaz. Este comportamiento está limitado a invocaciones en proceso en el mismo hilado. Visual Basic.NET permite unión tardía sólo a los miembros públicos de una clase y no a los miembros de la interfaz.

Retorno

La instrucción de retorno ahora regresa el control al que invoca la función. Si la función es de un cierto tipo, puede retornar el valor de nuevo al que invoca.

Las propiedades predeterminadas para cualquier objeto ya no están soportadas.

Diapositiva 26



Un objeto utilizado para destruirse automáticamente

En versiones anteriores de Visual Basic, los objetos se destruían tan pronto como se liberaba la última referencia de este objeto, y todas las referencias que se retenían a otros objetos también se liberaban. Establecer la última referencia a Nothing o simplemente salir de enfoque inmediatamente activaba la destrucción del objeto y liberaba todos sus recursos.

Ya no está disponible con Visual Basic.NET

Todo este esquema no está disponible en Visual Basic.NET. Debido a que la administración de memoria es realizada por .NET Framework, el cual hace uso de un recolector de basura, liberando así los recursos, esto se realiza cuando el motor de ejecución elige hacerlo así, y eso es básicamente cuando ya no hay nada más que hacer o no hay más recursos disponibles. (La manera en que funciona realmente el recolector de basura se analiza en el alcance de .NET Framework.)

Por ningún motivo el desarrollador se puede basar en el motor de ejecución que libera un recurso tan pronto se corta la referencia. Una forma que mantiene una conexión de base de

datos que se está terminando no garantiza que la conexión de la misma se libere inmediatamente.

Algunos programas existentes se basan en la finalización determinística. Estos programas enfrentarán problemas en el motor de ejecución .

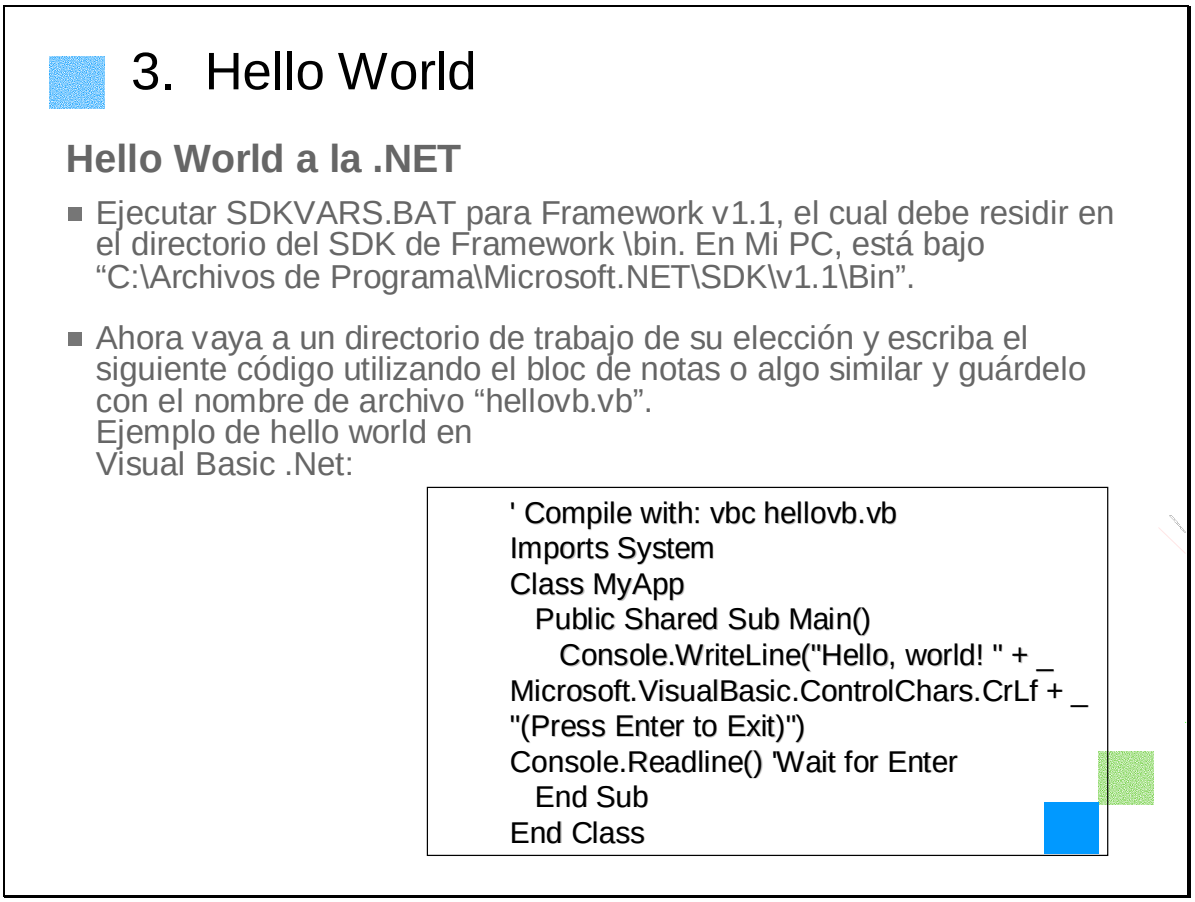
Una solución posible

Una solución posible es agregar un esquema de contador de referencia propio e invocar un método de limpieza después de que no necesite más el objeto o sus servicios. Mientras que ésta es una solución viable, este enfoque agrega una gran complejidad a su código.

Haga que sus objetos no tengan estado

Una mejor solución es evitar objetos que tengan estado. Esto también significa evitar objetos globales que, por ejemplo, mantienen una conexión viva de base de datos durante toda la vida del programa. En dicho caso, es mejor abrir la conexión cuando se necesita, cerrarla inmediatamente después de utilizarla y confiar en la memoria caché de conexiones que realiza el motor de ejecución.

Diapositiva 27



Hello World ala .NET

Muy bien. Primero lo primero. Vaya al indicador de comando y ejecute corvars.bat para Framework v1.0 ó SDKVARS.BAT para Framework v1.1 que debe estar en el directorio \bin del SDK de Framework. En mi computadora, está abajo "C:\Archivos de Programa\Microsoft Visual Studio .NET\FrameworkSDK\Bin " para Framework v1.0. Esto configurará las rutas del ambiente para que el compilador funcione.

Ahora vaya a un directorio de trabajo de su elección y escriba el siguiente código utilizando el bloc de notas o algo similar y guárdelo con el nombre de archivo "hellovb.vb".

Ejemplo de hello world en Visual Basic .Net:

```
' Compile with: vbc hellovb.vb
Imports System
Class MyApp
  Public Shared Sub Main()
```

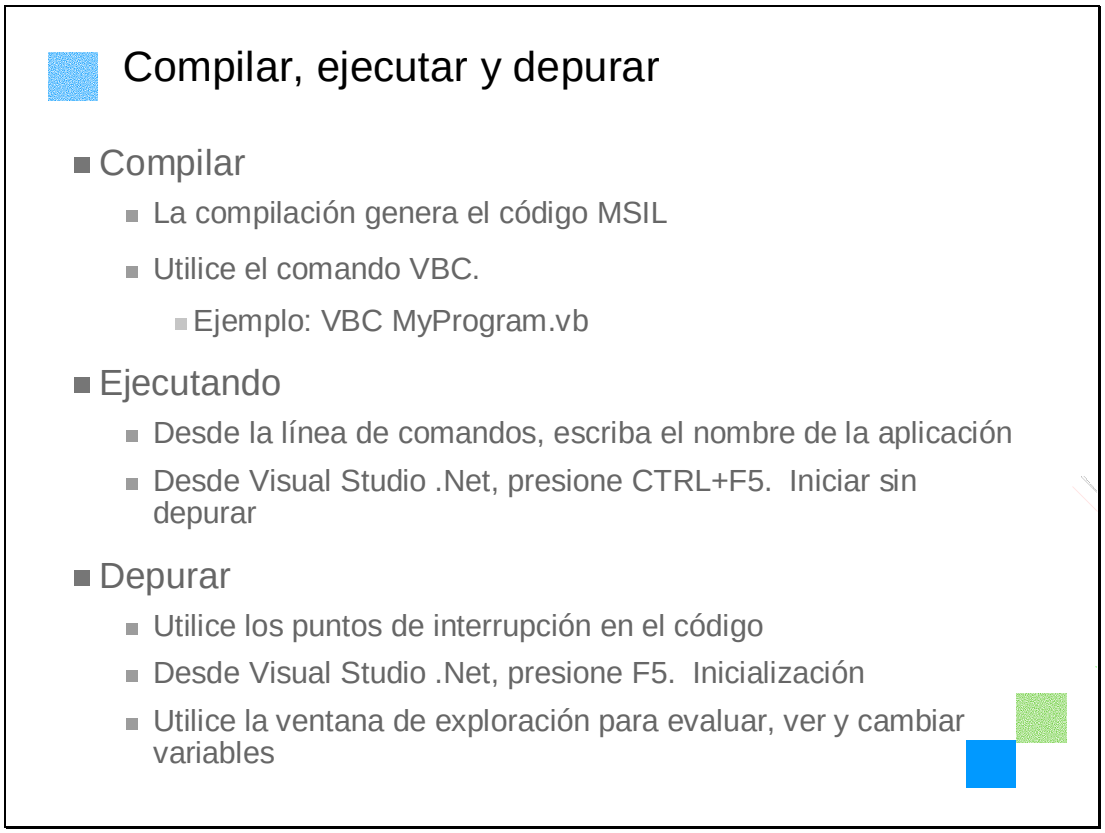
```
      Console.WriteLine("Hello, world! " + _
Microsoft.VisualBasic.ControlChars.CrLf + _
"(Press Enter to Exit)")
Console.Readline() 'Wait for Enter
   End Sub
End Class
```

Observará que el código utiliza la instrucción de importación para permitir al compilador saber donde encontrar la clase de consola; de otra forma, tendríamos que decir al compilador donde encontrar la clase incluyéndola en la llamada de clase como en el caso de utilizar la clase de caracteres de control. También creamos nuestra propia clase MyApp en lugar de un Módulo. La diferencia es que los módulos no puede crear una instancia de un módulo como lo puede hacer con una clase.

Compilación y ejecución de los programas:
Ahora escriba lo siguiente en la ventana del indicador de comandos asegurándose que esté en el mismo directorio que el archivo hellovb.vb creado con el bloc de notas.

```
      vbc hellovb.vb
   hellovb
```

Diapositiva 28



Interruptores de compilador comunes

Puede especificar un número de interruptores para el compilador VB.NET utilizando el comando VBC. Los siguientes elementos describen los interruptores más comunes.

/? /help

Muestra las opciones del compilador en la salida estándar.

/reference: or /r:

Provoca que el compilador cree información de tipo en los ensambles específicos disponibles para el proyecto que está compilando actualmente.

/reference:file[,file2] -or-

/r:*file[,file2]*

/out

Especifica el nombre del ejecutable.

/main

Especifica la clase que contiene el método Main (si más de una clase en la aplicación incluye el método Main).

/optimize

Activa y desactiva el optimizador de códigos.

/warn

Establece el nivel de advertencia del compilador.

/warnaserror

Trata todas las advertencias como errores que abortan la compilación.

/target

Especifica el tipo de aplicación generada.

/checked

Indica si el sobreflujo aritmético generará una excepción de motor de ejecución .

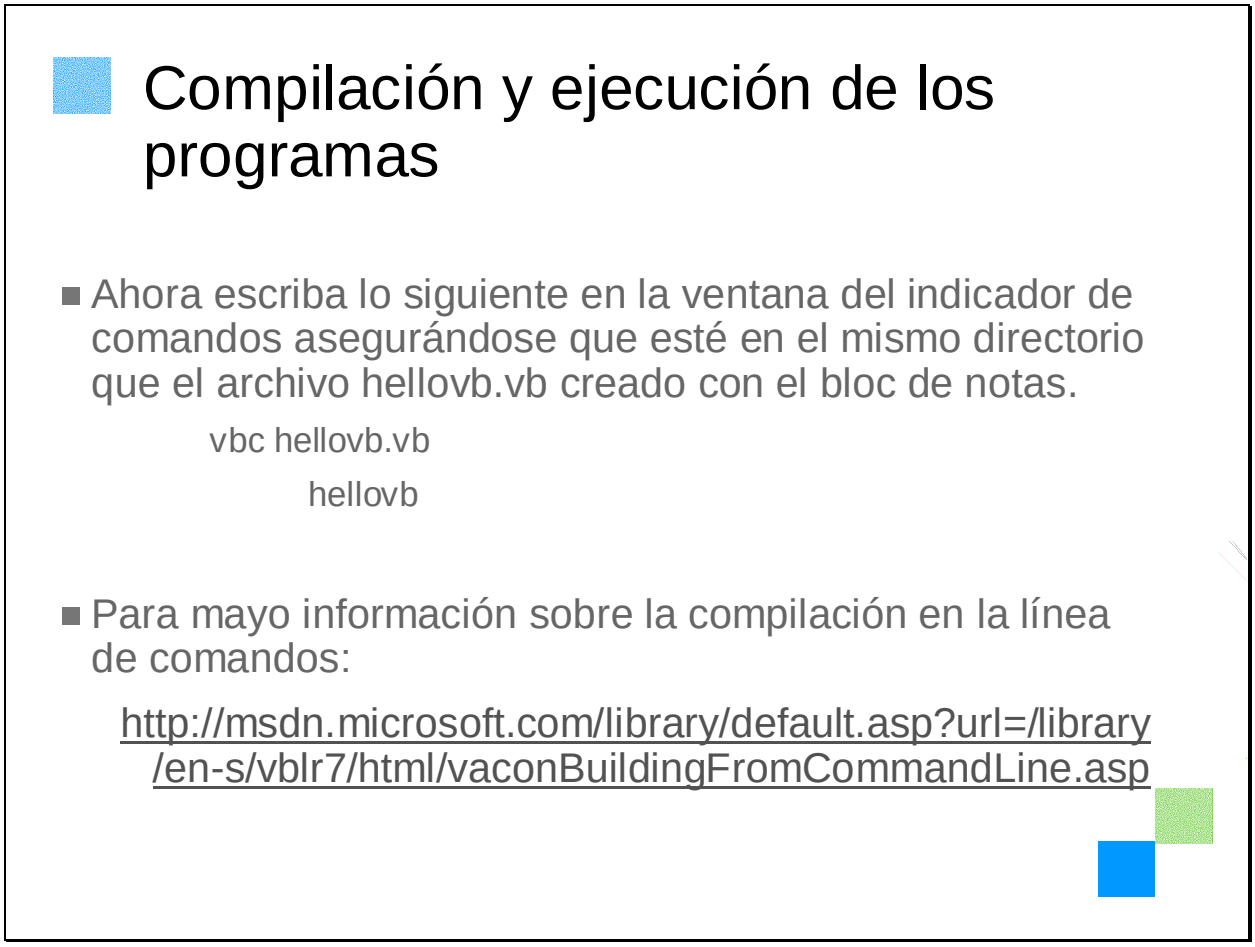
/doc

Procesa los comentarios de la documentación para producir un archivo XML.

/debug

Genera información de depuración.

Diapositiva 29



Para mayores informes sobre la compilación de la línea de comandos:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vblr7/html/vaconBuildingFromCommandLine.asp

Diapositiva 30

- **Lectura recomendada**
  - MacKinsey, Duncan. *Aprenda Visual Basic.NET en 21 días usted sólo.* Indianapolis, IN: Sams, 2001.
  - Beres, Evjen. *Biblia de Visual Basic .NET* Wiley, 2002
  - Reynolds-Haertle. *OOP con Microsoft Visual Basic .NET y Microsoft Visual C# .NET paso por paso.* Microsoft Press, 2002
  - Olsen, Allison, Speer. Manual de diseño de clases Visual Basic .NET. Wrox, 2002.
  - Balena. *Programar Microsoft Visual Basic .NET (Referencia básica).* Microsoft Press, 2002
- **Vínculos que se recomiendan para lectura y aprendizaje ulteriores**
  - WWW.ASP.NET
  - WWW.DOTNETJUNKIES.COM
  - http://msdn.microsoft.com/vbasic/using/understanding/default.aspx
  - http://support.microsoft.com/common/canned.aspx?R=d&H=Visual%20Basic%20.NET%20How%20To%20Articles&LL=kbvbnetsearch&Sz=kbhowtomaster
  - http://www.microsoft.com/seminar/mmcfeed/mmcdisplayfeed.asp?Lang=en&Product=103364

WWW.ASP.NET

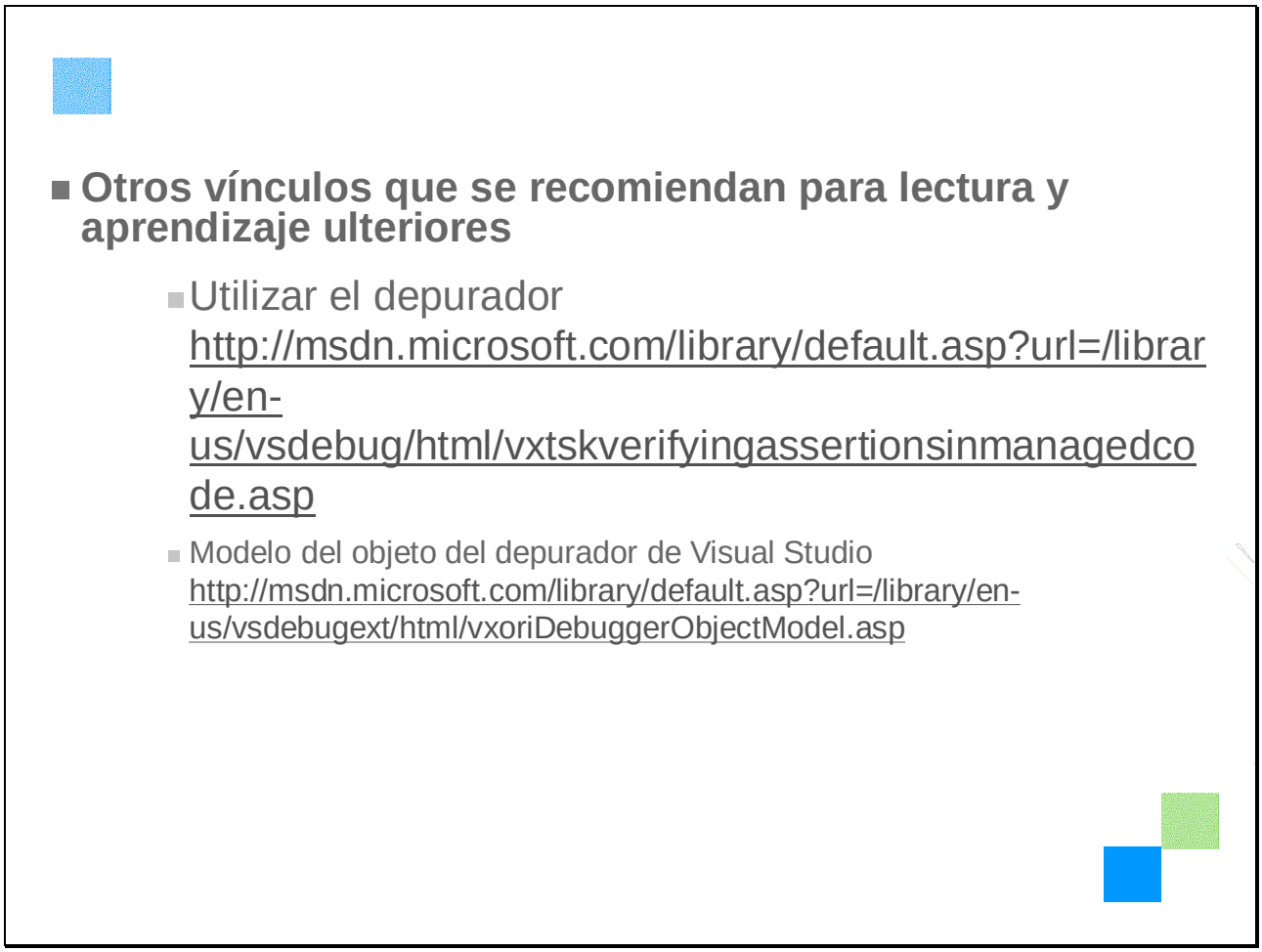
WWW.DOTNETJUNKIES.COM

http://msdn.microsoft.com/vbasic/using/understanding/default.aspx

http://support.microsoft.com/common/canned.aspx?R=d&H=Visual%20Basic%20.NET%20How%20To%20Articles&LL=kbvbnetsearch&Sz=kbhowtomaster

http://www.microsoft.com/seminar/mmcfeed/mmcdisplayfeed.asp?Lang=en&Product=103364

Diapositiva 31



Modelo del objeto del depurador de Visual Studio

http://msdn.microsoft.com/library/default.asp?url=/library/enus/vsdebugext/html/vxoriDebuggerObjectModel.asp