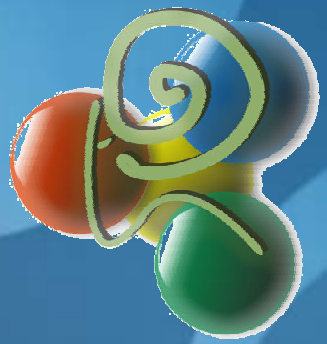
Célula Académica
UABC-Live .net



Universidad Autónoma de Baja California
Facultad de Ciencias Químicas e Ingeniería

http://uabc-live-net.spaces.live.com/

Sesión No. 4

Introducción a la Programación Orientada a Objetos en C# y Visual Basic.NET

Expositores: Jhania Issel Parra García
(x0620_003@hotmail.com)

José Luis Ruiz Mondragón
(sunfire45@msn.com)

Fecha: 12 de Octubre de 2006

Objetivo

Mostrar cómo aplicar los conceptos fundamentales de programación orientada a objetos utilizando los lenguajes Microsoft Visual C#.NET y Microsoft Visual Basic.NET

Prerrequisitos

- Poseer los conocimientos proporcionados en los siguientes módulos de la Estrella 0:
  - Fundamentos de Programación
  - Introducción a la Orientación a Objetos

Temas a Tratar

- Introducción a C# y VB.NET
- Sintaxis
  - Temas Generales
  - Definición e inicialización de variables
  - Operadores
  - Estructuras de Control
  - Clases y Objetos

Visual C#.NET

- Creado especialmente para .NET
- Estándar ECMA (2002)
- Orientado a Objetos y Componentes
- Sintaxis derivada de C, C++, Delphi y Java

Visual Basic.NET

- Rediseñado especialmente para .NET
- 100% Orientado a Objetos y Componentes
- Salvo por pequeñas excepciones, la sintaxis es la misma de VB6 y sus predecesores
- Tiene la misma potencia, robustez y capacidades de acceso a .NET que C#

Compiladores (1/2)

- Compilador de Línea de Comandos C#
  - Cumple con la especificación del CLS
  - <Win>\Microsoft.NET\Framework\<version>\csc.exe
  - Csc /out:XX /target:YY “Archivo1.cs” “Archivo2.cs”
- Compilador de Línea de Comandos VB.NET
  - Cumple con la especificación del CLS
  - <Win>\Microsoft.NET\Framework\<version>\vbc.exe
  - vbc /out:XX /target:YY “Archivo1.vb” “Archivo2.vb”

Compiladores (2/2)

- Algunas opciones útiles
  - /out:<file> — Nombre del archivo de salida
  - /target:exe/winexe/library — Consola/Windows/DLL
  - /reference:<file list> — Assemblies de referencia
  - /doc:<file> — Archivo de documentación
  - /debug[+|-] — Emitir info de DEBUG
  - /main:<type> — Determina la clase que posee el Entry Point (ignora los otros posibles)
  - /lib:<file list> — Directorios de librerías

Temas a Tratar

- Introducción a C# y VB.NET
- Sintaxis
  - Temas Generales
  - Definición e inicialización de variables
  - Operadores
  - Estructuras de Control
  - Clases y Objetos

VB.NET y C# - Terminación de línea

- C#: la línea finaliza con un “ ; ”

```
//Una linea con mas de un renglon
string nombre = primerNombre +
                apellido;
//El punto y coma indica FINAL de linea
```

- VB.NET: la línea finaliza con un salto de línea

```
'Una linea con mas de un renglon
Dim nombre As String = primerNombre & _
                       apellido
```

VB.NET y C# - Declaración de Bloques

- En C# los bloques se declaran entre llaves

```
class MainClass{
      public static void Main(string[] args) {
            Console.WriteLine("Hello World!");
      }
}
```

- En VB.NET cada bloque tiene su sentencia de apertura y su sentencia de cierre, que normalmente empieza con la palabra “End”

```
Class Main
      Public Shared Sub Main()
            Console.WriteLine("Hello World!")
      End Sub
End Class
```

VB.NET y C# - Comentarios

- C# soporta tres tipos de comentarios

```
string nombre = “Juan”; // Comentario de una sola línea
 /* Comentario con mas
   de una línea*/

/// <summary>
/// Documentación XML que puede ser consumida por otras
/// herramientas para mostrar ayuda dinámica o generar
/// documentación en varios formatos
/// </summary>
public class Matematica {
    /// <summary>
    /// Calcula la suma de dos números enteros
    /// </summary>
    /// <param name="x">El primer operando de la suma</param>
    /// <param name="y">El segundo operando de la suma</param>
    /// <returns> La suma entera de ambos operandos</returns>
    public int Sumar (int x, int y) {return x + y;}
}
```

VB.NET y C# - Comentarios

- VB.NET soporta dos tipos de comentarios

```
'Comentario simple
Dim nombre As String = “Juan”
```

```
''' <summary>
''' Documentación XML que describe un tipo y sus miembros
''' </summary>
''' <param name="x">El primer operando de la suma</param>
''' <param name="y">El segundo operando de la suma</param>
''' <returns> La suma entera de ambos operandos</returns>
Public Function Sumar (x as Integer, y as Integer) as Integer
       return x + y
End Function
```

VB.NET y C# - Case Sensitivity

- C# distingue entre mayúsuclas y minúsculas

```
system.console.writeline("HOLA");  INCORRECTO

System.Console.WriteLine("HOLA");  CORRECTO
```

- VB.NET no distingue entre mayúsuclas y minúsculas

```
system.console.writeline("HOLA")  CORRECTO

System.Console.WriteLine("HOLA")  CORRECTO
```

Temas a Tratar

- Introducción a C# y VB.NET
- Sintaxis
  - Temas Generales
  - Definición e inicialización de variables
  - Operadores
  - Estructuras de Control
  - Clases y Objetos

Tipos de Datos

Categoría	Clase	Descripción	C# Alias	VB.NET Alias
Enteros	Byte	Un entero sin signo (8-bit)	byte	Byte
	SByte	Un entero con signo (8-bit)	sbyte	Sbyte
	Int16	Un entero con signo (16-bit)	short	Short
	Int32	Un entero con signo (32-bit)	int	Integer
	Int64	Un entero con signo (64-bit)	long	Long
Punto Flotante	Single	Un número de punto flotante de simple precisión (32-bit)	float	Single
	Double	Un número de punto flotante de doble precisión (64-bit)	double	Double
	Decimal	Un número decimal de 96-bit	decimal	Decimal
Lógicos	Boolean	Un valor booleano (true o false)	bool	Boolean
Otros	Char	Un caracter Unicode (16-bit)	char	Char
	Object	La raíz de la jerarquía de objetos	object	Object
	String	Una cadena de caracteres unicode inmutable y de tamaño fijo	string	String

VB.NET y C# - Alcance de miembros

- **Miembro:** se refiere a los campos, propiedades, métodos, eventos, clases anidadas, etc.
- **C#:** todo miembro es declarado como PRIVATE por default
- **VB.NET:** todo miembro es declarado como PUBLIC por default
- Modificadores de acceso disponibles:

C#	VB.NET
public	Public
private	Private
internal	Friend
protected	Protected
protected internal	Protected Friend

VB.NET y C# - Declaración de Variables

- C#: el tipo de dato precede al identificador (nombre)

```
int x;
decimal y;
rectangle z;
Cliente cli;
```

- VB.NET: comienza con “Dim” o algún modificador de acceso (Public, Private, etc.) + identificador de la variable + “As” Tipo de Dato

```
Dim x As Integer        ‘Dim es = a Private por defecto
Dim y As Decimal
Dim z As Rectangle
Dim cli As Cliente
```

VB.NET y C# - Inicialización de Variables

- C#: toda variable debe ser inicializada EXPLICITAMENTE antes de ser usada

```
int tempBalance; //variable local
//ERROR: tempBalance NO ha sido inicializada
System.Console.WriteLine(tempBalance);
```

- VB.NET: inicializa automáticamente las variables en CERO o en Nothing

```
Dim tempBalance As Integer
'SIN ERROR: tempBalance vale CERO
System.Console.WriteLine(tempBalance)
```

VB.NET y C# - Conversiones de Tipos

- C# no permite conversiones implícitas de tipos
  - Si falla el cast se devuelve null o InvalidCastException

```
Cuenta cta = new CtaCte();
CtaCte cc = cta; //Error: puede que cta no sea una CtaCte
CtaCte cc = (CtaCte)cta; //Conversion explicita “CASTING”
CtaCte cc = cta as CtaCte; //Usando el operador “as”

if (cta is CtaCte) ... //Comp. con el operador “is”
```

- VB.NET usa las directivas del compilador Option Strict
  - Si falla el cast siempre se genera una InvalidCastException

```
Dim cta As Cuenta = New CtaCte()
Dim cc As CtaCte = cta ‘OK ⇔ Option Strict Off
Dim cc As CtaCte = CType(cta, CtaCte) ‘Option Strict On

If TypeOf cta Is CtaCte Then  ‘Comp. con TypeOf Is
```

VB.NET y C# - Arreglos

- C# utiliza corchetes [ ] para definición de arrays

```
string[] telefonos; //Definicion de un Arreglo de strings
telefonos = new string[3]; //De 3 elementos
telefonos[0] = "1245"; //Seteo del 1er elemento del arreglo

//Definicion y asignacion de una vez
telefonos = new string[] {"1","2","3"};
```

- VB.NET permite definir arrays de varias formas con ()

```
Dim telefonos As String()   ó   Dim telefonos() As String
Dim telefonos(3) As String 'Crea un array de 4 elementos
telefonos(0) = "1245" 'Seteo del 1er elemento del arreglo

'Definicion y asignacion de una vez
Dim telefonos() As String = {"1","2","3"}
```

Temas a Tratar

- Introducción a C# y VB.NET
- Sintaxis
  - Temas Generales
  - Definición e inicialización de variables
  - Operadores
  - Estructuras de Control
  - Clases y Objetos

VB.NET y C# - Operadores

Descripción	C#	VB.NET
Asignación	=	=
Adición	+	+
Sustracción	-	-
Multiplicación	*	*
División	/	/
Negación	!	not
Módulo (Parte entera de la división)	%	mod
Mayor	>	>
Menor	<	<
Mayor o Igual	>=	>=
Menor o Igual	<=	<=

VB.NET y C# - Operadores Lógicos

C#	VB.NET	Operador
&&	And	Operador logico Y
ll	Or	Operador logico O
!	Not	Negacion logica
==	=	Igual
!=	<>	Distinto

- En C# todas las evaluaciones se hacen por "cortocircuito"

```
//Si Hacer1() es True, entonces
//NO se evalua Hacer2()
if (Hacer1() || Hacer2())
{
}
```

```
//Si Hacer1() es False, entonces
//NO se evalua Hacer2()
if (Hacer1() && Hacer2())
{
}
```

- En VB.NET se debe utilizar los operadores AndAlso y OrElse

```
'Si Hacer1() es True, entonces
'NO se evalua Hacer2()
If Hacer1() OrElse Hacer2() Then
   ...
End If
```

```
'Si Hacer1() es False, entonces
'NO se evalua Hacer2()
If Hacer1() AndAlso Hacer2() Then
   ...
End If
```

Temas a Tratar

- Introducción a C# y VB.NET
- Sintaxis
  - Temas Generales
  - Definición e inicialización de variables
  - Operadores
  - Estructuras de Control
  - Clases y Objetos

VB.NET y C# - Sentencias condicionales

- C#: sentencia if con varios formatos

```
if (x > 10)
   HacerAlgo();
```

```
if (x < 10)
{
    Hacer1();
    Hacer2();
}
```

```
if (x < 10)
{
    Hacer1();
}
else
{
    Hacer2();
}
```

```
if (x < 10)
{
    Hacer1();
}
else if (x > 20)
{
    Hacer2();
}
else
{
    Hacer3();
}
```

- VB.NET: la sentencia If requiere de la palabra Then

```
If x > 10 Then Hacer()
```

```
If x < 10 Then
   Hacer1()
   Hacer2()
End If
```

```
If x < 10 Then
   Hacer1()
Else
   Hacer2()
End If
```

```
If x < 10 Then
    Hacer1()
ElseIf x > 20 Then
    Hacer2()
Else
    Hacer3()
End If
```

VB.NET y C# - Sentencias condicionales

- C#: sentencia case

```
int a = 0;
switch(a) {
        case 1:{//CODIGO 1
          break;
        }
        case 2: {//CODIGO 2
          break;
        }
        default:{//CODIGO DEFAULT
          break;
        }
}
```

- VB.NET: sentencia case

```
Dim a As Integer = 0
Select a
        Case 1
          'Código 1
        Case 2
          'Código 2
        Case Else
          'Código Default
End Select
```

VB.NET y C# - Sentencia for

- C#: la sentencia for consta de tres partes

```
//Partes: declaración, prueba, acción
for (int i=1; i < 10; i++)
{
}
```

- VB.NET usa las palabras claves For, To, Next y Step

```
Dim i As Integer

For i = 1 To 100
  'i se incrementa en 1
Next
```

```
For i = 1 To 100 Step 2
    'i se incrementa en 2
Next
```

VB.NET y C# - Sentencia for/each

- For/Each permite recorrer arreglos y colecciones
- C#: usa la palabra foreach

```
string[] nombres = new string[5];
foreach(string auxNombre in nombres)
{
    //auxNombre es de SOLO LECTURA
}
```

- VB.NET usa las palabra For Each

```
Dim nombres(5) As String
Dim auxNombre As String
For Each auxNombre In nombres
    'auxNombre NO es de SOLO LECTURA
Next
```

VB.NET y C# - Sentencia while

- C#:

```
bool condicion = true;
while (condicion == true){
       //En algún momento poner condicion = false
}
```

- VB.NET:

```
Dim condicion As Boolean = True
While condicion = True
   'Poner condicion=false en algún momento
End While
```

Temas a Tratar

- Introducción a C# y VB.NET
- Sintaxis
  - Temas Generales
  - Definición e inicialización de variables
  - Operadores
  - Estructuras de Control
  - Clases y Objetos

VB.NET y C# - Clases

- C#: las clases son declaradas mediante class

```
//Definicion de la clase CuentaBancaria
class CuentaBancaria
{
      //Definicion de miembros
}
```

- VB.NET usa las palabras Class / End Class

```
'Definicion de la clase CuentaBancaria
Class CuentaBancaria
      'Definicion de miembros
End Class
```

Sintaxis – Entry Point

- C#:

```
public class HolaMundo {
        public static void Main(string[] args){
        //Punto de entrada de la aplicación
        }
}
```

- VB.NET

```
Public Class HolaMundo
    Public Shared Sub Main(ByVal args() As String)
       'Punto de entrada de la aplicación
    End Sub
End Class
```

VB.NET y C# - Métodos

- Métodos: acciones que un objeto puede llevar a cabo.
- En C# todo método es una función

```
public void HacerDeposito(int importe) //No devuelve valor
{
}

public int ObtenerInventario(int codArticulo) //Devuelve un entero
{
}
```

- VB.NET usa procedimientos Sub y funciones Function

```
Public Sub HacerDeposito(ByVal importe As Integer)
'No devuelve valor
End Sub

Public Function Inventario(ByVal codArt As Integer) As Integer
'Devuelve un entero
End Function
```

VB.NET y C# - Constructores

- Constructor: métodos dentro de la clase que son llamados automáticamente cuando se crea una instancia de dicha clase.
- En C# tienen el mismo nombre de la clase

```
class CtaCte
{
      public CtaCte(){...}          //Const. por default
      public CtaCte(int i){...}     //Const. con un parametro
}
```

- VB.NET usa un procedimiento Sub New

```
Class CtaCte
     Sub New()
     End Sub
     Sub New(ByVal i As Integer)
     End Sub
End Class
```

VB.NET y C# - Sobrecarga de Métodos

- Sobrecarga: varios métodos con el mismo nombre pero diferente “firma”.
- C#

```
public void HacerDeposito(int importe)
{
}

public void HacerDeposito(int importe, bool acreditar)
{
}
```

- VB.NET

```
Public Sub HacerDeposito(ByVal imp As Integer)
End Sub

Public Sub HacerDeposito(ByVal imp As Integer, ByVal acreditar As Boolean)
End Sub
```

VB.NET y C# - Métodos estáticos

- Miembros que no requieren de una instancia para ser invocados. Se los llama métodos “de clase”
- C#

```
public static void HacerDeposito(int importe)
{

}
```

- VB.NET

```
Public Shared Sub HacerDeposito(ByVal imp As Integer)

End Sub
```

VB.NET y C# - Propiedades

- Propiedad: característica o atributo de un objeto

- C#

```
class CtaCte
{
   int balance;

   public int Balance
   {
      get
      {
         return balance;
      }
      set
      {
         balance = value;
      }
   }
}

CtaCte cc = new CtaCte();
cc.Balance = 100; //Asignación
Mostrar(cc.Balance); //Obtención
```

- VB.NET

```
Class CtaCte
   Dim _balance As Integer

   Property Balance() As Integer
      Get
         Return _balance
      End Get
      Set (ByVal value As Integer)
         _balance = value
      End Set
   End Property
End Class

Dim cc As New CtaCte()
cc.Balance = 100 ‘Asignación
Mostrar(cc.Balance) ‘Obtención
```

VB.NET y C# - Herencia

- En C# la herencia se define:

```
class Cuenta                    //Clase Base
{
}
class CtaCte : Cuenta           //Clase Derivada
{
}
```

- VB.NET usa la palabra clave Inherits

```
Class Cuenta                    'Clase Base
End Class

Class CtaCte                    'Clase Derivada
      Inherits Cuenta
End Class
```

En .NET solo se permite Herencia Simple

VB.NET y C# - Herencia (Cont.)

- En C#

```
public sealed class Cuenta{
//No se puede heredar de esta clase “sellada”
}
public abstract class Cuenta{
//No se pueden crear instancias de esta clase, sólo
//de sus derivadas
}
```

- VB.NET

```
Public NotInheritable Class Cuenta
‘No se puede heredar de esta clase
End Class

Public MustInherit Class Cuenta
‘No se pueden crear instancias de esta clase, sólo de sus
‘derivadas
End Class
```

VB.NET y C# - Namespaces

- C#

```
namespace BancoARG
{
   namespace Gestion
   {
      public class CtaCte
      {
      }
      public class CajaAhorro
      {
      }
   }
}

//Referencia "full"
BancoARG.Gestion.CtaCte;
BancoARG.Gestion.CajaAhorro;

//Referencia "corta"
using BancoARG.Gestion;
CtaCte cc = new CtaCte();
CajaAhorro ca = new CajaAhorro();
```

- VB.NET

```
Namespace BancoARG
   Namespace Gestion
      Public Class CtaCte

      End Class

      Public Class CajaAhorro

      End Class
   End Namespace
End Namespace

'Referencia "full"
BancoARG.Gestion.CtaCte
BancoARG.Gestion.CajaAhorro

'Referencia a un namespace
Imports BancoARG.Gestion
Dim cc As New CtaCte()
Dim ca As New CajaAhorro()
```

VB.NET y C# - Admin. De Excepciones

- Excepción: condición anómala de funcionamiento de una aplicación

- C#: usa las palabras try/catch/finally

```
try
{
   int resultado = x/y;
}
catch(DivideByZeroException e)
{
   //Error division por cero
}
catch
{
   //Otro error
}
finally
{
   //Siempre pasa por aca
}
```

- VB.NET usa las palabras Try/Catch/Finally

```
Try
   Dim resultado As Integer
   resultado = x/y
Catch e As DivideByZeroException
   'Error division por cero
Catch
   'Otro error
Finally
   'Siempre pasa por aca
End Try
```

Microsoft®

© 2005 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.