Introducción a Microsoft .NET Framework

Bienvenido al curso de programación con Microsoft .NET Framework.
Antes de comenzar es importante recordar que para realizar este curso es importante tener conocimientos básicos de programación. Por ejemplo, entender y saber aplicar los conceptos de:

- Tipos de datos
- Variables y contantes
- Condicionales e iteraciones
- Procedimientos y funcionales
- Clase e intancias de una clase (objetos), métodos y atributos.
- Conceptos básicos de bases de datos (tablas, claves primarias y foráneas, relaciones, etc.)

El objetivo de estos textos en ayudarte a recapitular lo explicado y trabajado en clase pero no pretende reemplazar la clase presencial ni ser un manual de referencia. Por eso te recomendamos consultes los vínculos a la Web mencionados, realices las prácticas de laboratorio planteadas y examines las descargas disponibles para cada clase.

Bibliografía opcional

Existe una amplia bibliografía en .NET, con lo cual es realmente difícil recomendar un texto en particular. Por otro lado no es en necesario recurrir a ella para completar este curso con éxito. Sin embargo, si querés profundizar sobre los temas de éste curso o relacionados, más podés consultar a tu instructor que seguramente podrá orientarte al respecto.

Software necesario para realizar los laboratorios

En este curso utilizaremos Visual C# Express 2010 o Visual Basic 2010 Express, según sea el lenguaje elegido. Ambos productos son gratuitos y pueden descargase desde aquí: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express

Con respecto al motor de base de datos utilizaremos Microsoft SQL Server 2008 R2 RTM Express with Management Tools. Este producto también es gratuito y puede descargarse desde aquí: http://www.microsoft.com/en-us/download/details.aspx?id=23650

Consejos

- Si disponés de la versión Professional (o superior) de Visual Studio 2010, no es necesario que instales la versión Express. Solamente hay que tener en cuenta que las versiones Express no soportan soluciones en diferentes lenguajes, con lo cual en el aula sólo podrás visualizar tus soluciones desarrolladas en un único lenguaje (C# o VB.NET).
- Si disponés de alguna versión de SQL Server (2000 en adelante) no te recomendamos que realices el upgrade a 2008 R2 exclusivamente por el curso, salvo que tengas planeado hacerlo por otros motivos. Esto se debe a que los upgrades de SQL Server son especialmente problemáticos. En este caso simplemente tendrás dos versiones de la bases de datos que usaremos a lo largo del curso, una en tu versión y otra que funcionará en el aula, concretamente 2008 R2.
- Si preferís instalar la nueva versión de SQL Server 2012 Express debés tener en cuanta que

probablemente también tengas que mantener dos versiones diferentes de tus bases de datos.
- La versión de Visual Studio Express 2012 está disponible desde el 15/08/12. Consultá a tu instructor sobre la compatibilidad con la versión 2010.

Qué es Microsoft .NET Framework

Es el centro o core de la tecnología de programación que usarás en este curso. En pocas palabras el Framwork es un compilador, un entorno de ejecución y una biblioteca de clases.

El .NET el centro son las bibiotecas de clases y los lenguajes (C#, VB.NET, etc.) son transformados a un código intermedio llamado MSIL. Por lo tanto, cualquier preferencia sobre un lenguaje u otro está dada simplemente por sus estilos de sintaxis.

Para conocer la evolución de la plataforma .NET podés consultar:
http://es.wikipedia.org/wiki/Microsoft_.NET

Para saber más sobre la arquitectura interna del Framework, podés consultar:
http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

Por cierto, ¿sabías que existen muchos lenguajes .NET? Podés ver una lista aquí:
http://en.wikipedia.org/wiki/List_of_CLI_languages

El programa Hola Mundo

Tu instructor te habrá guiado para escribir este clásico con el comienzan todos los cursos de programación. El objetivo es que identifiques la estructura general de un programa de consola y sobre todo que te familiarices con la interfaz de Visual Studio 2010 Express.

C#

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hola Mundo!");
        Console.WriteLine();
        Console.Write("Pulse una tecla para terminar...");
        Console.ReadKey();
    }
}
```

VB.NET

```
Module Program

    Sub Main()
        Console.WriteLine("Hola Mundo!")
        Console.WriteLine()
        Console.Write("Pulse una tecla para terminar...")
        Console.ReadKey()
    End Sub

End Module
```

En síntesis, se ha definido una clase con un método que se autojecuta al ejecutar la consola.

A tener en cuenta

- El "intelligence" (la lista desplegable que aparece cuando comienzas a escribir una instrucción) ayuda a seleccionar una opción válida sensible al contexto.
- En C# es importante comprender el uso de las llaves, los puntos y comas (que marcan el fin de una instrucción) y recordar que es "case sensitive", es decir, distingue el uso de mayúsculas y minúsculas. Los detalles sobre que significa "using" y "namespace" se verán más adelante.

Tipos de datos .NET

En .NET existen varios tipos de datos. Es importante conocerlos a fin de definir correctamente variables y contantes.

A continuación podemos ver algunos tipos de datos de uso frecuente:

Framework	C#	VB.NET	Se utiliza para representar…
System.Boolean	bool	Boolean	valores booleanos (verdadero/falso).
System.Int32	int	Integer	números enteros.
System.Double	double	Double	valores decimales en cálculos científicos.
System.Decimal	decimal	Decimal	valores decimales en cálculos financieros.
System.String	string	String	cadena de caracteres.
System.DateTime	datetime	DateTime	fechas con hora, minutos, segundos, etc.
System.Object	object	Object	cualquier objeto.

Cabe destacar que los tipos de datos en .NET son clases. Como tales poseen métodos y atributos.

Para ver todos los tipos de datos soportados en .NET consultá: http://msdn.microsoft.com/en-us/library/hfa3fa08(v=vs.110).aspx

Declaración y asignación de variables

En C# una variable se declara definiendo el tipo de dato y luego su nombre. Por ejemplo:

```
bool varBoolean;
```

En VB.NET se utiliza la instrucción Dim y a continuación el nombre de la variable seguido por el tipo de dato. Por ejemplo:

C#

```
bool Respuesta;
```

VB.NET

```
Dim Respuesta As Boolean
```

Buena práctica

Al declarar una variable asigná siempre un valor, incluso si dicho valor es su valor por defecto.

Constantes

Las constantes mantiene invariable su valor a lo largo de la ejecución del programa.
Se declaran de la siguiente forma:

C#

```
const string Empresa = "Acme";
```

VB.NET

```
Const Empresa As String = "Acme"
```

Conversiones entre tipos de datos

Una buena práctica en programación consiste en evitar las conversiones implícitas entre tipos de datos. Por ejemplo, la cadena de caracteres “1” no debería poder asignarse en forma directa a una variable numérica .

C# no permite prácticamente ninguna conversión en forma implícita. Pero VB.NET, por defecto, sí. Para evitar este comportamiento en VB.NET es necesario indicarlo a nivel de proyecto. Para ello abre las propiedades del proyecto y en la pestaña Compile selecciona Option strict = On.

Buena práctica

Siempre establecé la opción Option strict=On en tus proyectos VB.NET.

Operadores

Operadores aritméticos

Operador	C#	VB.NET
Suma	+	+
Resta	-	-
Multiplicación	*	*
División	/	/

Módulo	#	Mod

A tener en cuenta

En VB.NET existe un operador específico para la división entera: “\”.

Operadores de comparación

Operador	C#	VB.NET
Igual	==	=
Distinto	!=	<>
Mayor	>	>
Menor	<	<
Mayor o igual	>=	>=
Menor o igual	<=	<=

A tener en cuenta

En C# se distingue la igualdad (==) de la asignación (=).

Operadores lógicos

Operador	C#	VB.NET
Y	&	And
Y (corto)	&&	AndAlso
O	|	Or
O (corto)	||	OrElse
Negación	!=	Not

A tener en cuenta

El circuito corto puede proporcionar (tomando como carga de trabajo miles de comparaciones) un aumento en la velocidad. Sin embargo, como se puede ver en los ejemplos, en algunos casos puede tener efectos no deseados en tu código.

Asignación condicional

La asignación condicional funciona de la siguiente manera: si la condición entre parántesis es verdadera se asiga el valor que se encuentra después del signo de interrogación y antes de los dos puntos. En cambio si la condición entre paréntesis es falsa, se asigna el valor que está entre los dos puntos y el punto y coma.
En el siguiente ejemplo, si la variable Indice es mayor que 10 se asiga 10 a Resultado. Si no, se

asigna 0.

C#

```
int Resultado = (Indice > 10) ? 10 : 0;
```

VB.NET

```
Dim Resultado As Integer = If(Indice > 10, 10, 0)
```

Condicionales (toma de decisiones)

Instrucción If

C#

```
if (Var1 == Var2)
{
    Console.WriteLine("El valor de ambas variables es el mismo.");
}
else
{
    Console.WriteLine("El valor de ambas variables es distinto.");
}
```

VB.NET

```
If Var1 = Var2 Then
    Console.WriteLine("El valor de ambas variables es el mismo.")
Else
    Console.WriteLine("El valor de ambas variables es distinto.")
End If
```

En ambos caso, la sección “else” es opcional.

Instrucción Switch/Select Case

C#

```
switch (Var3)
{
    case 1:
        Console.WriteLine("var3 = primavera.");
        break;
    case 2:
        Console.WriteLine("var3 = verano.");
        break;
    case 3:
        Console.WriteLine("var3 = otoño.");
        break;
    case 4:
        Console.WriteLine("var3 = invierno.");
        break;
    default: //si no es ninguno de los indicado arriba
        Console.WriteLine("var3 fuera de rango.");
        break;
}
```

VB.NET

```
Select Case Var3
    Case 1
        Console.WriteLine("Var3 = primavera.")
    Case 2
        Console.WriteLine("Var3 = verano.")
    Case 3
        Console.WriteLine("Var3 = otoño.")
    Case 4
        Console.WriteLine("var3 = invierno.")
    Case Else
        Console.WriteLine("Var3 fuera de rango.")
End Select
```

En C#, el soporte para rangos es limitado:

C#

```
switch (Var3)
{
    case 1:
    case 2:
    case 3:
        Console.WriteLine("var3 1, 2, o 3");
        break;
    case 4:
        Console.WriteLine("var3 4");
        break;
    default:
        Console.WriteLine("var3 fuera de rango.");
        break;
}
```

VB.NET

```
'Select case con rangos
Select Case Var3
    Case 1 To 3
        Console.WriteLine("Var3 1, 2, o 3")
    Case 4
        Console.WriteLine("Var3 4")
    Case Else
        Console.WriteLine("Var3 fuera de rango.")
End Select
```

Repeticiones (iteraciones)

Instrucción For

C#

```
for (int Var = 1; Var <= 10; Var++)
{
    Console.WriteLine("Var= {0}", Var);
}

for (int Var = 100; Var >= 10; Var = Var - 10)
{
    Console.WriteLine("Var= {0}", Var);
}
```

VB.NET

```
For Var = 1 To 10
    Console.WriteLine("Var= {0}", Var)
Next

For Var = 100 To 10 Step -10
    Console.WriteLine("Var= {0}", Var)
Next
```

Instrucción While

C#

```
while (Contador < 10)
{
    Console.WriteLine("contador = {0}", Contador);
    Contador = Contador + 1;
}
```

VB.NET

```
While Contador < 10
    Console.WriteLine("contador = {0}", Contador)
    Contador = Contador + 1
End While
```

Instrucción Do while/ Do Loop While

C#

```
do
{
    Console.WriteLine("contador = {0}", Contador);
    Contador = Contador + 1;
}
while (Contador < 10);
```

VB.NET

```
Do
    Console.WriteLine("contador = {0}", Contador)
    Contador = Contador + 1
Loop While Contador < 10
```

Es importante recordar que a diferencia de while/While, do while/Do Loop While siempre realizan al menos un ciclo independientemente de la condición de salida.

Métodos (procedimientos y funciones)

Características

- Se denominan métodos tanto a los procedimientos cómo a las funciones.
- Un procedimiento es un conjunto de instrucciones agrupadas bajo un nombre que no devuelven un valor.
- Una función es un conjunto de instrucciones agrupadas bajo un nombre que devuelven un valor.
- Los métodos son capaces de recibir parámetros de entrada.
- En el caso de las funciones es indispensable indicar el tipo de dato que devolverá.

Procedimientos

C#

```
void Imprimir()
{
    //código...
}

void Imprimir(string paramEncabezado, string paramCuerpo, string paramPieDePagina)
{
    //código...
}
```

VB.NET

```
Private Sub Imprimir(paramEncabezado As String, paramCuerpo As String,
                                                paramPieDePagina As String)
    'Codigo...
End Sub
```

Funciones

C#

```
bool Funcion() //bool: retorna un valor del tipo boolean
{
    //Codigo...
    return true; //o false, dependiendo del código
}

double SuperficieCirculo(double paramRadio) //pase de parámetro
{
   return Math.PI * Math.Pow(paramRadio, 2);
}
```

VB.NET

```
Function Funcion() As Boolean 'As Boolean: retorna un valor del tipo Boolean
    'Codigo...
    Return True 'o False, dependiendo del código
End Function

Private Function SuperficieCirculo(paramRadio As Double) As Double
    Return Math.PI * Math.Pow(paramRadio, 2)
End Function
```

Paramétros opcionales

Los parámetros opcionales se definen en C# indicando un valor por defecto.
En VB.NET con el prefijo Optional.

C#

```
static void ParamOp (int param1, int param2, int param3 = 3, int param4 = 4)
{
    //...
}


ParamOp(1, 1, param4: 1);
```

VB.NET

```
Private Sub ParamOp(param1 As Integer, param2 As Integer,
                    Optional param3 As Integer = 3,
                    Optional param4 As Integer = 4)

    '...

End Sub

ParamOp(1, 1, , 1)
```

Parámetros por valor y parámetros por referencia

Por defecto, los parámetros se pasan aun método por valor, es decir, se pasa una copia del valor original. Esto implica que las modificaciones realizadas dentro del método no afectan a dicho valor fuera del mismo.
En cambio, si se especifica que el parámetro se pasa por referencia, lo que se pasa al método es un puntero a la dirección de memoria del valor. Esto implica que las modificaciones realizadas dentro del método afectan a dicho valor fuera del mismo.

C#

```
void Sumar(int Numero1, ref int Numero2)
{
    Numero1 = Numero1 + 1;
    Numero2 = Numero2 + 1;
}
```

VB.NET

```
Private Sub Sumar(Numero1 As Integer, ByRef Numero2 As Integer)
    Numero1 = Numero1 + 1
    Numero2 = Numero2 + 1
End Sub
```

En ambos ejemplos, el valor de Numero 1 sigue siendo 0 al salir del método. En cambio, el valor de Numero2 es 1, ya que se ha modificado directamente su valor dentro del método.

Uso de enumeraciones

Las enumeraciones se utilizan principalmente para pasar parámetros a un método de tal forma que estos sean significativos para el programador que utliza el método.
Se aplican a parámetros que tiene una serie de valores posibles finita y estable.

C#

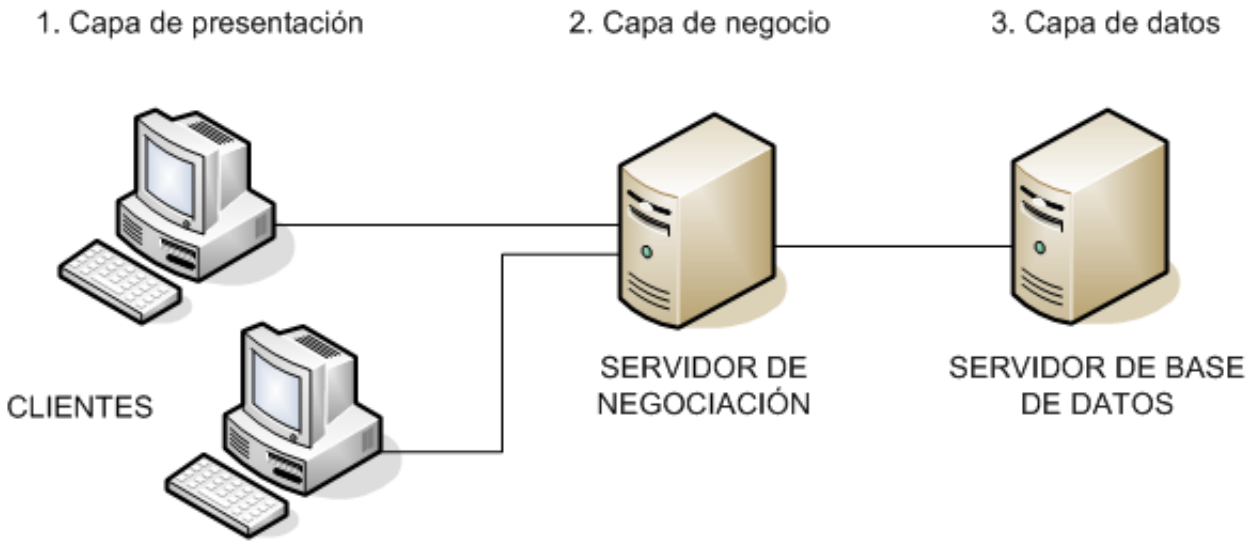
```
public enum EstadoCivil
{
    Soltero,
    Casado,
    Viudo,
    Separado,
    Divorciado,
    Conviviente
}
```

VB.NET

```
Public Enum EstadoCivil
    Soltero
    Casado
    Viudo
    Separado
    Divorciado
    Conviviente
End Enum
```

Arquitectura en tres capas

La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.



La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos.

Capa de presentación

Presenta el sistema al usuario, le comunica información y captura los datos que el usuario envía al sistema. También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario.

Esta capa se comunica únicamente con la capa de negocio.

Capa de negocio

En esta capa se define todo el código que define las reglas de negocio (procesos, cálculos, validaciones, etc.). Por esto se denomina capa de negocio (o lógica del negocio).

Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados.

Capa de datos

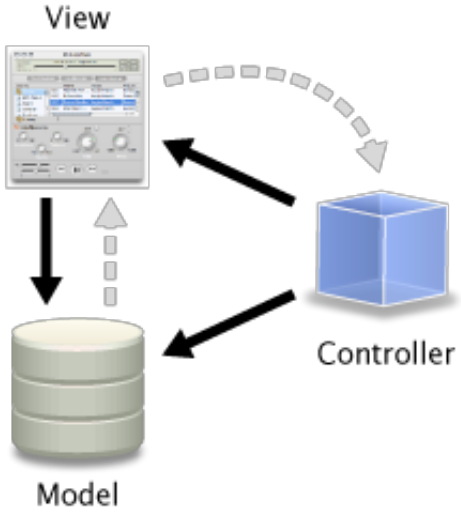
En esta capa se define el código que permite acceder a las fuentes de datos.
Normalmente se encuentran en esta capa codificadas al menos las cuatro operaciones básicas, llamadas CRUD (por Create-Retrieve-Update y Delete.

Diferencia entre capas y niveles

En una arquitectura de tres niveles, los términos "capas" y "niveles" no significan lo mismo ni son similares.
El término capa hace referencia a la forma como una solución es segmentada desde el punto de vista lógico y es la que acabamos de definir arriba..
En cambio, el término nivel corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física, por ejemplo, cliente y servidor.

Arquitectura Modelo-Vista-Controlador (MVC)

Es una implementación más estricta de la arquitectura por capas. Actualmente se aplica en el ámbito .NET a aplicaciones Web, aunque existen propuestas para aplicarla, por ejemplo, a Windows Forms y otras tecnologías.


View
Controller
Model
