



Escuela de Sistemas y Tecnologías

Transparencias de ANALISTA DE SISTEMAS
Edición 2009 – Materia: POO en C#

TEMA: PROGRAMACION EN LENGUAJE C#



Plantel y Contactos

- **Coordinador Académico:**
 - Ing. Jorge Corral
 - Mail: jcorral@bios.edu.uy
- **Bedelía:**
 - Mail: bedeliasistemas@bios.edu.uy
- **Encargado de Sucursal:**
 - Pablo Castaño
 - Mail: pablocasta@bios.edu.uy



Recursos

- **Recursos Imprescindibles:**
 - Sitio Web: www.espaciobios.com/espaciob
(comunicarse con Bedelía por usuario/contraseña).
 - Transparencias del Curso.
 - Contar con el software necesario



Consideraciones

- Estas transparencias no tienen el objetivo de suplir las clases.
- Por tanto, serán complementadas con ejemplos, códigos, profundizaciones y comentarios por parte del docente.
- El orden de dictado de estos temas está sujeto a la consideración del docente.

Agenda (1)

- Introducción a Microsoft .Net
- .Net Framework
- Sentencias
- Bloques de código
- Comentarios
- Aplicaciones en Consola
- Tipos de datos
- Variables
- Constantes
- Operadores
- Expresiones Lógicas

Agenda (2)

- Sentencias de Selección
- Sentencias de Iteración
- Vectores
- Matrices
- Métodos
- Diccionario

Introducción a Microsoft .NET (1)

- Plataforma que engloba diferentes aplicaciones, servicios y conceptos que en conjunto permiten el desarrollo y ejecución de aplicaciones
- Características:
 - Plataforma de ejecución intermedia
 - 100% orientado a objetos
 - MultiLenguaje
 - Modelo de programación único para todo tipo de Aplicaciones

Introducción a Microsoft .NET (2)

Plataforma de Ejecución Intermedia



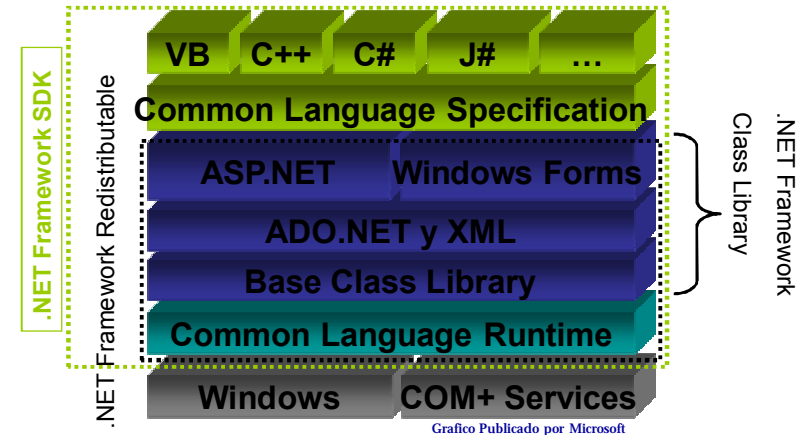
Grafico Publicado por Microsoft

.NET Framework (1)

- Paquete de software fundamental de la plataforma .NET
- Se distribuye de forma libre y gratuita
- Existen tres variantes principales:
 - .Net Framework Redistributable Pack
 - .Net Framework SDK
 - .Net Compact Framework

.NET Framework (2)

Arquitectura



.NET Framework (3)

Componentes Principales:

- CLR (Common Language Runtime)
- Class Library (NameSpaces)
- CLS (Common Language Specification)
- BCL (Base Class Library)
- ADO .NET (Advanced Data Objects)
- ASP .NET (Active Server Pages)
- WinForms (interfaz gráfica basada en formularios y ventanas Windows)

.NET Framework (4)

Ventajas Principales:

- Unifica modelos de programación.
- Simplifica el desarrollo.
- Entorno de ejecución robusto y seguro.
- Independiente del lenguaje de programación.
- Es extensible.
- Simplifica instalación y administración de aplicaciones.

Desventaja Principal:

- Disponible únicamente para Sistemas Operativos de la familia Windows®.



Entorno de Desarrollo (Visual Studio)

- **Lenguajes:**
 - Visual Basic .NET
 - C#
 - C++
 - J#
- **Tipos de Aplicaciones:**
 - de Consola
 - para Windows
 - Biblioteca de Clases (DLL)
 - Sitio Web

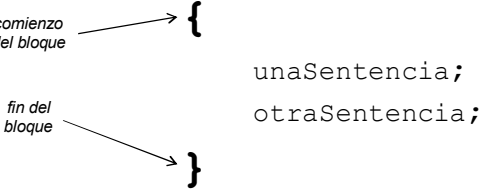


Sentencias en C#

- Al final de cada sentencia se debe colocar un punto y coma (;)
- El lenguaje C# distingue entre mayúsculas y minúsculas (*case sensitive*).
- Es altamente recomendable utilizar la indentación de sentencias anidadas.



Bloques de Código

- Un bloque de código (clase, método, estructura, sentencia) es una o más sentencias delimitadas por llaves:

- **Bloques y Variables:** Un bloque de código anidado dentro de otro no podrá tener variables con el mismo nombre que las existentes en el bloque de código contenedor.



Comentarios

- Un comentario es un texto que sirve de guía al programador para comentar el código.
- Los comentarios no se compilan.

```
// Comentario de una línea.
```

```
/* Comentario de  
   más de una línea.  
*/
```

```
///<summary>  
/// Comentario.  
///</summary>
```



Aplicaciones en Consola (1)

- Tipos de archivos básicos:
 - .sln → archivo de solución
 - .cs → archivo de código C#
- Lo primero que se correrá al ejecutar una aplicación en consola son las sentencias incluidas dentro del método llamado "main".
- Para incluir elementos .NET que se ubican dentro de cualquier NAMESPACE, se utiliza:

```
using namespace;
```

Ejemplo:

```
using System.Console;
```



Aplicaciones en Consola (2)

- La clase `Console` contiene las sentencias de entrada y salida estándar (teclado y pantalla respectivamente).
- Sólo es utilizable dentro de un proyecto de consola.
- Sentencias básicas de manejo de pantalla:

```
Console.Write ("texto");
```

→ escribe en la salida

```
Console.WriteLine ("texto");
```

→ escribe en la salida y agrega un retorno de carro al final

```
Console.Read();
```

→ lee el siguiente caracter

```
Console.ReadLine();
```

→ lee caracteres hasta que se genera un retorno de carro



Tipos de Datos (1)

- CTS (Common Type System) Define un conjunto de tipos de datos orientados a objetos. Todo lenguaje .Net debe implementarlos:
 - Tipo Valor: numéricos, booleanos, enumerados,
 - Tipo Referencia: clase, interfaz, array, string, DateTime
- Administra dos segmentos de memoria:
 - Stack
 - Heap



Tipos de Datos (2)

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

Gráfico Publicado por Microsoft



Variables (1)

- Toda variable es declarada Privada por defecto
- Algunos modificadores de acceso existentes:
 - public
 - private



Variables (2)

- Para declarar una variable en C# es obligatorio determinar su tipo de dato por ser un lenguaje fuertemente tipado.
- Sintaxis:

```
tipoAcceso tipoDato nombreVariable;
```
- Ejemplo:

```
int unNumero;  
public int unNumero;
```



Variables (3)

- Toda variable debe ser inicializada explícitamente antes de ser utilizada

Sintaxis:

```
tipoAcceso tipoDato nomVar = valor;  
  
tipoAcceso tipoDato nombreVariable;  
nombreVariable = valor;
```

Ejemplo:

```
int unNumero = 5;  
  
int unNumero;  
unNumero = 5;
```



Variables (4)

- La clase **Convert** posee variadas funcionalidades que permiten convertir tipos de datos básicos.
- Ejemplos:

```
int Numero = Convert.ToInt32("569");  
DateTime Fecha;  
Fecha = Convert.ToDateTime("24/12/2009");
```

Constantes

- Las constantes se declaran y definen igual que una variable, agregándole la palabra reservada **const**
- **Convención:** utilizar nombres en mayúsculas.

Sintaxis:

```
tipoAcceso tipoDato const NOMBRE = valor;
```

Ejemplo:

```
public const IVA = 22;
```

Operadores (1)

Aritméticos:

Descripción	C#
Adición	+
Sustracción	-
Multiplicación	*
División	/
Módulo (Parte entera de la división)	%
División Entera	\

De Relación:

Descripción	C#
Mayor	>
Menor	<
Mayor o Igual	>=
Menor o Igual	<=

Operadores (2)

Lógicos:

Descripción	C#
Operador lógico Y	&&
Operador lógico O	
Negación lógica	!
Igualdad	==
Diferencia	!=

Varios:

Descripción	C#
Asignación	=
Sumar y Asignar	+=
Restar y Asignar	-=
Multiplicar y Asignar	*=
Dividir y Asignar	/=

Expresiones Lógicas

- Siempre se evalúan como verdaderas o falsas.

Ejemplos:

```
(unVariable <= 25)
(unVariable == otraVariable)
(unVariable > 10) && (unaVarLogica)
(unVariable >= 0) || (unVariable <= 100)
```

Sentencias de Selección (1)

- La estructura **if** permite realizar selecciones a partir de la evaluación de expresiones.

Sintaxis:

```
if (expresión)
{
    sentenciasPorEvaluaciónTrue;
}
else
{
    sentenciasPorEvaluaciónFalse;
}
```

Sentencias de Selección (2)

```
if (expresión1)
{
    sentenciasPorEvalTrueDeExp1;
}
else if (expresión2)
{
    sentenciasPorEvalTrueDeExp2;
}
else
{
    sentenciasPorEvalFalseDeTodas;
}
```

Sentencias de Selección (3)

Ejemplo

```
int edad = 18;
if (edad > 18)
{
    Console.WriteLine("Es mayor de edad");
}
else if (edad < 0)
{
    Console.WriteLine("Edad error");
}
else
{
    Console.WriteLine("Es menor de Edad");
}
```

Sentencias de Selección (4)

- La estructura **switch** permite realizar selecciones a partir del valor de una variable.

Sintaxis:

```
switch (variable) {
    case valor1: {
        sentencias;
        break;
    }
    case valor2: {
        sentencias;
        break;
    }
    default: {
        sentencias;
        break;
    }
}
```


Sentencias de Iteración (1)

- La estructura **while** se ejecuta mientras la expresión sea evaluada como verdadera.

Sintaxis:

```
while (expresión)
{
    sentencias;
}
```

Ejemplo:

```
int unNumero = 0;
while (unNumero <= 10)
{
    unNumero += 2;
}
```

Sentencias de Iteración (2)

Sintaxis

```
for (inicialización ; condición; actualización)
{
    sentencias;
}
```

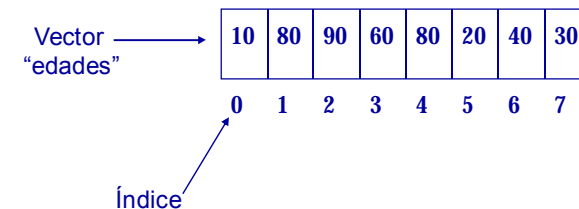
Ejemplo

```
for (int indice = 0 ; indice <= 10; indice++)
{
    Console.WriteLine(indice);
}
```

Vectores (1)

- Son variables que contienen una cantidad predefinida y fija de elementos del mismo tipo.
- Almacena sus elementos en posiciones de memoria contiguas.
- Permite acceso directo u aleatorio a sus elementos individuales.
- Se accede a cada elemento utilizando el nombre del vector y el índice específico.
- El índice del primer elemento del vector es el índice 0.

Vectores (2)



- En el ejemplo anterior se tiene un vector con las siguientes características:
 - El nombre de la variable es "edades".
 - Contiene elementos numéricos.
 - Posee 8 elementos, los cuales se acceden desde la posición 0 hasta la 7.

Vectores (3)

➤ Declaración de un Vector

Sintaxis: `tipoDato[] vector = new tipodato[largo];`

Ejemplo: `int[] edades = new int[8];`

➤ Asignar un valor a una posición específica

Sintaxis: `vectorNom[indice] = valor;`

Ejemplo: `edades[2] = 90;`

➤ Utilizar el valor de una posición específica

Sintaxis: `vectorNom[indice]`

Ejemplo: `Console.WriteLine(edades[2]);`

Vectores (4)

➤ Ejemplo de cómo recorrer un vector:

```
for (int i=1; i<=edades.Length; i++)
{
    Console.WriteLine("La edad en lugar" + i + "es" + edades[i]);
}
```

Vectores (5)

➤ Algoritmos de Ordenación:

- Selection Sort (orden por selección) → orden n^2
- Buble Sort (orden burbuja) → orden n^2
- Insertion Sort (orden por inserción) → orden n^2

➤ Algoritmos de Búsqueda:

- Lineal Search (búsqueda lineal)
 - Realiza la búsqueda en forma secuencial.
- Binary Search (búsqueda binaria)
 - Es mas optima
 - El vector debe estar previamente ordenado.
 - Realiza la búsqueda por partición.

Matrices (1)

➤ Una matriz es un vector de dos dimensiones; lo cual hace necesario tener dos índices para acceder a sus elementos.

➤ Declaración de una Matriz

`tipoDato[,] matrizNom = new tipoDato[largoF, largoC];`

➤ Asignar un valor a una posición específica

`matrizNom[indiceF, indiceC] = valor;`

➤ Utilizar el valor de una posición específica

`matrizNom[indiceF, indiceC]`

Matrices (2)

Matriz "números" →

ÍndiceF →

9	43	48	15	74
8	36	31	60	99
7	25	56	29	97
6	3	81	72	67
4	95	52	63	39

0 1 2 3 4

↑ ÍndiceC

Métodos (1)

- Un método es un trozo de código que realiza una tarea concreta y puede ser invocado repetidas veces.
- Ventajas de su uso:
 - Hay ahorro de líneas de código; ya que una sola tarea se escribe una sola vez
 - Mayor facilidad para corrección de errores
 - Mayor facilidad para futuras modificaciones
 - Mayor Claridad

Métodos (2)

- Hay dos tipos:
 - Procedimiento: proceso sin devolución de datos
 - Función: proceso que debe devolver un dato
- Los modificadores de acceso tienen el mismo alcance que los vistos en el capítulo de "Variables".

Métodos (3)

- La información que se desea enviar a un método para su proceso, se pasa como parámetro.
- Éstos están determinados por un nombre y el tipo de datos que contienen.
- Son posicionales (importa el orden).
- Hay tres tipos básicos:
 - Por Valor
 - Por Referencia
 - Solo de salida



Métodos (4)

➤ Sintaxis básica:

```
modifAcceso tipoDato nomMetodo (tipoDato nParam)
{
    sentencias;
    return (valor);
    sentencias;
}
```

- **void:** palabra reservada que indica que un método no tiene tipo de dato de retorno (es decir se comporta como un procedimiento).



Diccionario

Aplicación en Consola	Console Application
Bloques	Blocks
Comentarios	Comments
Espacio de Nombres	NameSpace
Método	Method
Sentencia	Statement
Sentencias Anidadas	Nested Statements
Vector	Array



FIN
PROGRAMACIÓN EN LENGUAJE C#