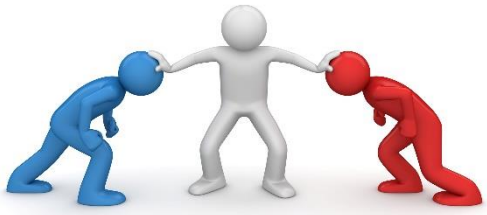
Florida Universitària

Bloque2: Programación con VS.NET (C#)

Características de C# vs JS



Tipos de Datos en C# (I)

Florida Universitària

C# → Lenguaje fuertemente tipado

Tabla de tipos integrados (Referencia de C#)

Predefinidos

Son los tipos que el lenguaje incorpora para almacenar valores comunes, entre ellos:

- `bool` para valores binarios, como un *sí* o *no*, o `1` o `0`
- `int` para valores enteros, que puede ir de `-2,147,483,648` a `2,147,483,647`
- `char` para caracteres como `a`, `@` o `%`
- `string` para una secuencia de caracteres

La lista la completan: `byte`, `decimal`, `float`, `long`, `sbyte`, `short`, `uint`, `ulong`, `ushort` y `object`.

A partir de estos tipos de dato se pueden crear otros tipos con la finalidad de satisfacer nuestras necesidades.

Compuestos

Los tipos compuestos están formados a partir de los tipos predefinidos y nos ayudarán a modelar de manera más real los problemas con los que nos encontremos. Estos se crean usando las palabras reservadas `interface`, `struct`, `enum` y `class`.

Tipos de Datos en C# (II)

Estructura	Detalles	Bits	Intervalo de valores	Tipo C#
Byte	Bytes sin signo	8	[0, 255]	byte
Int16	Enteros cortos con signo	16	[-32.768, 32.767]	short
UInt16	Enteros cortos sin signo	16	[0, 65.535]	ushort
Int32	Enteros normales	32	[-2.147.483.648, 2.147.483.647]	int
UInt32	Enteros normales sin signo	32	[0, 4.294.967.295]	uint
Int64	Enteros largos	64	[-9.223.372.036.854.775.808, 9.223.372.036.854.775.807]	long
UInt64	Enteros largos sin signo	64	[0, 18.446.744.073.709.551.615]	ulong
Single	Reales con 7 dígitos de precisión	32	$[1{,}5\times10^{-45}$, $3{,}4\times10^{38}]$	float
Double	Reales de 15-16 dígitos de precisión	64	$[5{,}0\times10^{-324}$, $1{,}7\times10^{308}]$	double
Decimal	Reales de 28-29 dígitos de precisión	128	$[1{,}0\times10^{-28}$, $7{,}9\times10^{28}]$	decimal
Boolean	Valores lógicos	32	true, false	bool
Char	Caracteres unicode	16	['\u0000', '\uFFFF']	char
String	Cadenas de caracteres	Variable	El permitido por la memoria	string
Object	Cualquier objeto	Variable	Cualquier objeto	object

Tabla de tipos integrados (Referencia de C#)

Declaración de variables:

```
int numero;
float porcent = 0.16;
char carácter;
bool salir = true;
string nombre;
double salario;
string dado = “seis”;
byte edad;
```

Palabras reservadas en C#

abstract	enum	long	stackalloc
as	event	namespace	static
base	explicit	new	string
bool	extern	null	struct
break	false	object	switch
byte	finally	operator	this
case	Fixed	out	throw
catch	float	override	true
char	for	params	try
checked	foreach	private	typeof
class	goto	protected	uint
const	if	public	ulong
continue	implicit	readonly	unchecked
decimal	in	ref	unsafe
default	int	return	ushort
delegate	interface	sbyte	using
do	internal	sealed	virtual
double	is	short	void
else	lock	sizeof	while

Lista de palabras reservadas en C#

Operadores y prioridades en C#

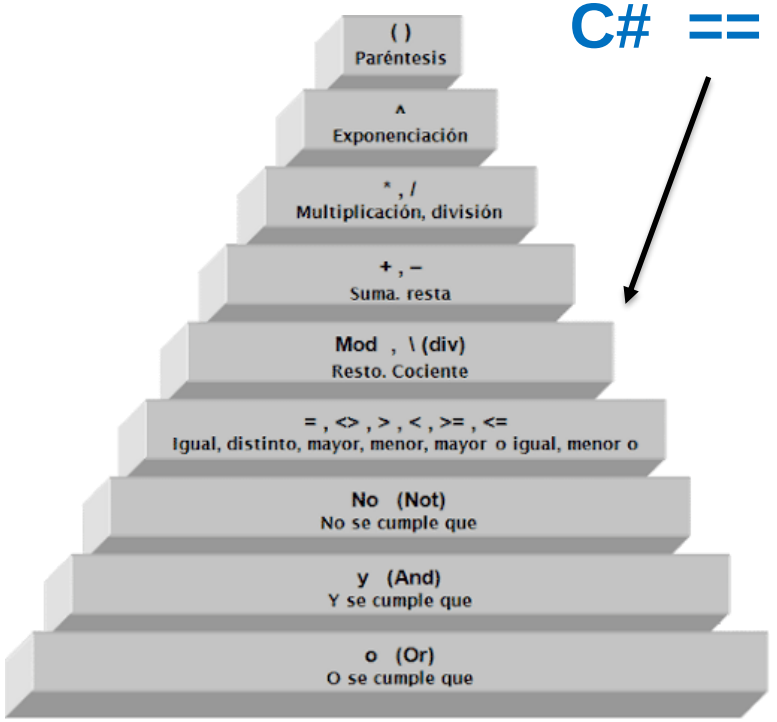
Florida Universitària

Operadores en C#

Categoría	Operadores
Aritméticos	`+` `-` `*` `/` `%`
Lógicos	`!` `&&` `||`
A nivel de bits	`&` `|` `^` `~`
Concatenación	`+`
Incremento, decremento	`++` `--`
Desplazamiento	`<<` `>>`
Relacional	`==` `!=` `<` `>` `<=` `>=`
Asignación	`=` `^=` `<<=` `>>=`
Acceso a miembro	`.`
Indexación	`[ ]`
Conversión	`( )`
Condicional	`? :` `??`
Creación de objeto	`new`
Información de tipo	`as` `is` `sizeof` `typeof`

Prioridades aritméticas

C# == JS



Conversión de tipos (I)

Conversiones de tipos en C#. …¿Cuándo es necesaria?... Ejemplos

- Si se declara una variable como entera y el valor entero viene representado como cadena.
- Cuando se necesita concatenar o imprimir un valor numérico en una cadena de texto
- Cuando dos valores numéricos necesitan concatenarse y no sumarse
- Cuando el resultado de una expresión es diferente al tipo de declarado de la variable a la cual se va a asignar
- Cuando una función envía un valor de retorno en un tipo de datos específico
- Etc….

Conversión de tipos (II)

Conversión implícita

- Se realiza con seguridad de tipos y **sin** pérdida de información
- No es necesaria una función/método para realizar la conversión

```
int n = 32767;  // tipo original
long nLargo = n;     // tipo destino
```

Tabla de conversiones numéricas implícitas (Referencia de C#)

Conversión Explícita → casting

- Se realiza con seguridad de tipos y **con** pérdida de información
- Si se necesita el valor perdido es conveniente usar una función de conversión

```
double valor = 16.78;   // tipo original
int valorEntero = (int) valor;   // tipo destino

// Resultado de valorEntero: 16
```

Tabla de conversiones numéricas explícitas (Referencia de C#)

Conversión de tipos (III)

Conversión definida por el usuario

- El programador decide qué tipo de datos desea convertir.
- El programador debe conocer la lista de conversiones implícitas y explícitas, y que además reconoce el tipo original y el de destino del dato.

Parse Method

```
string n1 = "20";
int numero1 = int.Parse(n1);

string n2 = "20.45";
double numero2 = double.Parse(n2);

string n3 = "56";
Int32 numero3 = Int32.Parse(n3);

string valor = "10/12/2018";
DateTime fecha = DateTime.Parse(valor);
```

Convert Class

Espacio de nombres: System

```
string n1 = "20";
int numero1 = Convert.ToInt16(n1);

string n2 = "20.45";
double numero2 = Convert.ToDouble(n2);

string valor = "10/12/2018";
DateTime fecha = Convert.ToDateTime(valor);
```

- ToBoolean
- ToByte
- ToChar
- ToDateTime
- ToDecimal
- ToDouble
- ToInt16
- ToInt32
- ToInt64
- ToSByte
- ToSingle
- ToString
- ToUInt16
- ToUInt32
- ToUInt64

Análisis ejemplo

Namespace System

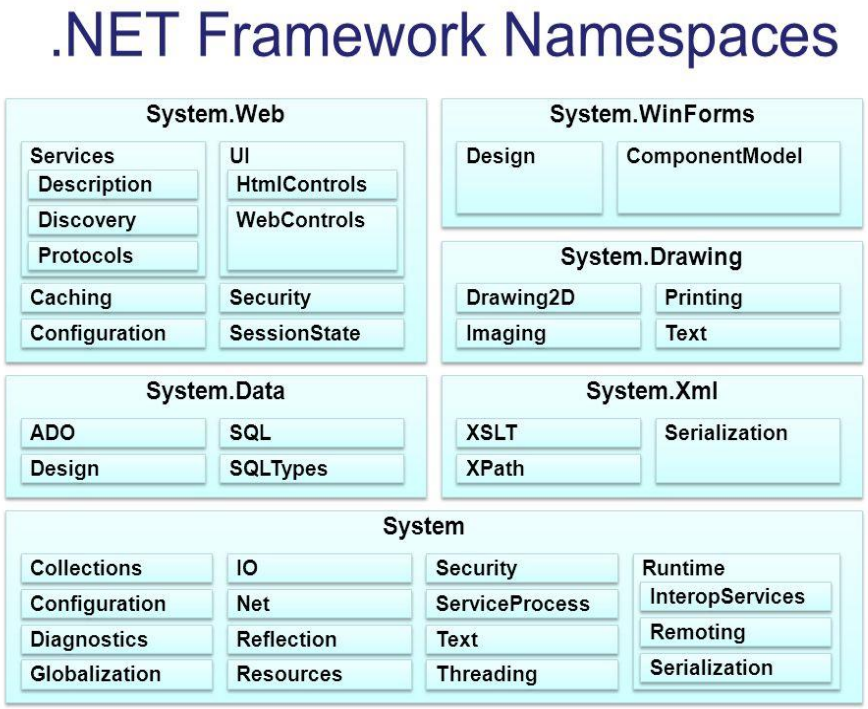
System Namespace

El espacio de nombres System contiene clases fundamentales y clases base que definen tipos de datos de valor y de referencia usados comúnmente, eventos y controladores de eventos, interfaces, atributos y excepciones de procesamiento.

Los **espacios de nombres** son un modo sencillo y eficaz de tener absolutamente todas las clases perfectamente organizadas, tanto las que proporciona el .NET Framework como las que cree un programador

Podemos verlo claro en examinador de objetos de Visual Studio.NET:

Ver → examinador de objetos (Ctrl+Alt+J)



Métodos Matemáticos (Math)

Math Class

Espacio de nombres: System

Proporciona constantes y métodos estáticos para operaciones trigonométricas, logarítmicas y otras funciones matemáticas comunes.

Random Class

Espacio de nombres: System

```
public static void Main()
{
   Random rnd = new Random();
   string[] malePetNames = { "Rufus", "Bear", "Dakota", "Fido",
                             "Vanya", "Samuel", "Koani", "Volodya",
                             "Prince", "Yiska" };
   string[] femalePetNames = { "Maggie", "Penny", "Saya", "Princess",
                               "Abby", "Laila", "Sadie", "Olivia",
                               "Starlight", "Talla" };

   // Generate random indexes for pet names.
   int mIndex = rnd.Next(malePetNames.Length);
   int fIndex = rnd.Next(femalePetNames.Length);
```

```
class MathTrapezoidSample
{
    private double m_longBase;
    private double m_shortBase;
    private double m_leftLeg;
    private double m_rightLeg;

    public MathTrapezoidSample(double longbase, double shortbase, double leftLeg
    {
        m_longBase = Math.Abs(longbase);
        m_shortBase = Math.Abs(shortbase);
        m_leftLeg = Math.Abs(leftLeg);
        m_rightLeg = Math.Abs(rightLeg);
    }

    private double GetRightSmallBase()
    {
        return (Math.Pow(m_rightLeg,2.0) - Math.Pow(m_leftLeg,2.0) + Math.Pow(m_
    }

    public double GetHeight()
    {
        double x = GetRightSmallBase();
        return Math.Sqrt(Math.Pow(m_rightLeg,2.0) - Math.Pow(x,2.0));
    }
```

Clase String

String Class

Espacio de nombres: System

Representa texto como una secuencia de unidades de código UTF-16.

Análisis de propiedades y métodos

```
string string1 = "This is a string created by assignment.";
Console.WriteLine(string1);
string string2a = "The path is C:\\PublicDocuments\\Report1.doc";
Console.WriteLine(string2a);
string string2b = @"The path is C:\PublicDocuments\Report1.doc";
Console.WriteLine(string2b);
// The example displays the following output:
//       This is a string created by assignment.
//       The path is C:\PublicDocuments\Report1.doc
//       The path is C:\PublicDocuments\Report1.doc
```

Mediante la asignación de un literal de cadena a un String variable

```
char[] chars = { 'w', 'o', 'r', 'd' };
sbyte[] bytes = { 0x41, 0x42, 0x43, 0x44, 0x45, 0x00 };

// Create a string from a character array.
string string1 = new string(chars);
Console.WriteLine(string1);

// Create a string that consists of a character repeated 20 times.
string string2 = new string('c', 20);
Console.WriteLine(string2);
```

Mediante llamada a un constructor de clase String

Arrays (Matrices unidimensionales) (I)

Florida Universitària

```
int[] valores; //valores sin inicializar
valores = new int[100];  //100 elementos
valores = new int[20];  //ahora contiene 20 elementos
```

Matrices unidimensionales (Guía de programación de C#)

Arreglos en C#

Inicialización

```
int[] valores = new int[10] {0,1,2,3,4,5,6,7,8,9};
string[] paises = new string[5] {"Argentina", "Bolivia",
"Peru","Chile","Colombia"};

//Inicializacion omitiendo el tamaño de la matriz
int[] valores = new int[] {0,1,2,3,4,5,6,7,8,9};
string[] paises = new string[] {"Argentina", "Bolivia", "Peru","Chile","Colombia"};

//Tambien podemos omitir el operador new

int[] valores =  {0,1,2,3,4,5,6,7,8,9};
string[] paises = {"Argentina", "Bolivia", "Peru","Chile","Colombia"};
```

Arrays (Matrices unidimensionales) (II)

```
static void Main(string[] args)
{

    int[] numero = new int[10];
    for(int i=0;i<10;i++)
    {
        Console.WriteLine("Introduce el número {0}: ", i+1);
        numero[i] = int.Parse(Console.ReadLine());
    }
}
```

Matrices unidimensionales (Guía de programación de C#)

Arreglos en C#

¡¡¡… En C# los Arrays deben tener un tamaño inicial… !!!

```
static void Main(string[] args)
{

    string[] nombre= new string[3] { "Salva", "Andrés", "Ana" };

    // Visualización con For tradicional
    for (int i=0;i<nombre.Length;i++)
    {
        Console.WriteLine("Persona: {0} ", nombre[i]);
    }

    // Visualización con foreach
    foreach(string elemento in nombre)
    {
        Console.WriteLine("Persona: {0} ", elemento);
    }

}
```

Florida Universitària

Redimensionado de un array (método Resize)

```
static void Main(string[] args)
{

    // Creamos un array con 3 elementos
    string[] nombre= new string[3] { "Salva", "Andrés", "Ana" };

    // redimensionamos el array para 5 elementos
    Array.Resize(ref nombre, 5);

    // Cargamos los dos últimos elementos
    nombre[3] = "Pedro"; nombre[4] = "Luis";

    // Visualización con For tradicional
    for (int i=0;i<nombre.Length;i++)
    {
        Console.WriteLine("Persona: {0} ", nombre[i]);
    }

}
```

Array Class

Espacio de nombres: System

Analizar

Analizar ejemplo

Proyecto Resize

Estructuras de decisión (I)

- if
- else

```
bool condition = true;

if (condition)
{
    Console.WriteLine("The variable is set to true.");
}
else
{
    Console.WriteLine("The variable is set to false.");
}
```

```
Console.Write("Enter a character: ");
char c = (char)Console.Read();
if (Char.IsLetter(c))
{
    if (Char.IsLower(c))
    {
        Console.WriteLine("The character is lowercase.");
    }
    else
    {
        Console.WriteLine("The character is uppercase.");
    }
}
else
{
    Console.WriteLine("The character isn't an alphabetic character.");
}
```

Estructuras de decisión (II)

- switch
- case
- default

```
public static void Main()
{
   int caseSwitch = 1;

   switch (caseSwitch)
   {
       case 1:
           Console.WriteLine("Case 1");
           break;
       case 2:
           Console.WriteLine("Case 2");
           break;
       default:
           Console.WriteLine("Default case");
           break;
   }
}
```

```
using System;

public class Example
{
   public static void Main()
   {
      Random rnd = new Random();
      int caseSwitch = rnd.Next(1,4);

      switch (caseSwitch)
      {
          case 1:
              Console.WriteLine("Case 1");
              break;
          case 2:
          case 3:
              Console.WriteLine($"Case {caseSwitch}");
              break;
          default:
              Console.WriteLine($"An unexpected value ({caseSwitch})");
              break;
      }
   }
}
// The example displays output like the following:
//       Case 1
```

Estructuras de decisión (III)

Florida Universitària

```
using System;

public enum Color { Red, Green, Blue }

public class Example
{
   public static void Main()
   {
      Color c = (Color) (new Random()).Next(0, 3);
      switch (c)
      {
         case Color.Red:
            Console.WriteLine("The color is red");
            break;
         case Color.Green:
            Console.WriteLine("The color is green");
            break;
         case Color.Blue:
            Console.WriteLine("The color is blue");
            break;
         default:
            Console.WriteLine("The color is unknown.");
            break;
      }
   }
}
```

Análisis de ejemplos resueltos de forma diferente

```
using System;

public enum Color { Red, Green, Blue }

public class Example
{
   public static void Main()
   {
      Color c = (Color) (new Random()).Next(0, 3);
      if (c == Color.Red)
         Console.WriteLine("The color is red");
      else if (c == Color.Green)
         Console.WriteLine("The color is green");
      else if (c == Color.Blue)
         Console.WriteLine("The color is blue");
      else
         Console.WriteLine("The color is unknown.");
   }
}
// The example displays the following output:
//       The color is red
```

Estructuras de repetición

- do
- for
- foreach, in
- while

```
int n = 0;
do
{
    Console.WriteLine(n);
    n++;
} while (n < 5);
```

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(i);
}
```

```
int n = 0;
while (n < 5)
{
    Console.WriteLine(n);
    n++;
}
```

```
var fibNumbers = new List<int> { 0, 1, 1, 2, 3, 5, 8, 13 };
int count = 0;
foreach (int element in fibNumbers)
{
    count++;
    Console.WriteLine($"Element #{count}: {element}");
}
Console.WriteLine($"Number of elements: {count}");
```

El siguiente ejemplo muestra el uso de la instrucción `foreach` con una instancia del tipo List<T> que implementa la interfaz IEnumerable<T>

Funciones con y sin devolución (retorno) - Análisis

```
class Program
{
    static void primeraFuncion()
    {
        Console.WriteLine("Hola otra vez");
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Hola");
        primeraFuncion();
        Console.ReadKey();
    }
}
```

```
//Función que devuelve un string (cadena)
static string devuelveTexto()
{
    return "Hola otra vez";
}

static void Main(string[] args)
{
    //Escribo el texto que devuelve la función
    Console.WriteLine(devuelveTexto());

    Console.ReadKey();
}
```

Programación modular (funciones - II)

```
namespace ejemplo
{
    class MainClass
    {
        public static void Main (string[] args)
        {

            int a = 5;
            int b = 10;

            int suma = f_sumador (a,b); //llamada funcion

            Console.WriteLine ("(Funcion) La suma es "+suma);

            m_sumador (a,b); //Llamada metodo

            Console.ReadLine ();
```

```
        //Declaracion Funcion
        public static int f_sumador(int a, int b){

            int suma = a + b;

            return suma;

        }

        //Declaracion Metodo
        public static void m_sumador(int a, int b){

            Console.WriteLine ( "(Metodo) La suma es "+ (a+b) );

        }
    }
}
```

Florida Universitària

ACTIVIDAD PRÁCTICA 7 (AP7)

Título

Ejercicios en C# (Programación estructurada)

Objetivos

- Realizar aplicaciones de consola básicas en C# utilizando programación estructurada.
- Dominar las diferentes formas de conversión de tipos utilizadas en lenguaje C#.
- Dominar el uso de estructuras de decisión, estructuras de repetición y programación modular utilizando el lenguaje C#.
- Utilizar métodos matemáticos (Math) y diversos métodos y propiedades de las clases String y Array.
- Saber gestionar y solventar de forma adecuada errores de compilación en el desarrollo de programas.