Unity

Máster en Programación de Videojuegos

Ignacio Martínez Rodríguez

Curso 2013-2014

Diferencias entre C# y C++

C# vs C++

- **Compilación**: C# compila a un lenguaje intermedio (IL) y luego en runtime se compila a código máquina mediante un proceso llamado JIT (just in time).

- **Memoria**: C# usa un recolector de basura como Java. Adiós a los delete ☺

- **Clases**: En C++, las clases y las estructuras son casi idénticas, mientras que en C# son bastante diferentes.

- **Herencia**: Las clases de C# pueden implementar cualquier número de interfaces, pero sólo pueden heredar de una clase base.

C# vs C++

- **Matrices**: En C++, una matriz es simplemente un puntero. En C#, las matrices son objetos que incluyen métodos y propiedades. Por ejemplo, el tamaño de una matriz se puede consultar mediante la propiedad Length.

- Las matrices de C# también emplean indizadores que comprueban cada uno de los índices utilizados para tener acceso a la matriz. La sintaxis para declarar matrices de C# es diferente de la que se utiliza para las matrices de C++: los símbolos "[]" aparecen después del tipo de matriz en C#, no la variable.

C# vs C++

Ejemplos de matrices:

```
int[] table; // al contrario que int table[] en C++;
int[] numbers; // number es un array de tamaño no definido
numbers = new int[10];  // numbers es una array con 10 elementos
int size = numbers.Length; // size vale 10
foreach(int i in numbers)
{
  System.Console.WriteLine(i);
}
```

Es importante notar que las matrices al ser objetos hay que instanciarlos con **new**

C# vs C++

- **Booleanos**: en C++, el tipo **bool** es esencialmente un entero. En C# no, por lo tanto no es posible realizar la conversión del tipo **bool** a otros tipos o viceversa.

```
int x = 123;
if (x)   // Error: "Cannot implicitly convert type 'int' to 'bool'"
{
    Console.Write("El valor no es cero.");
}
```

```
if (x != 0)   // Correcto
{
    Console.Write("El valor no es cero.");
}
```

C# vs C++

- **Parámetros**: en C++, todas las variables se pasan por valor a menos que se pasen explícitamente con un puntero o una referencia. En C#, las clases se pasan por referencia y las estructuras se pasan por valor, a menos que se pasen explícitamente por referencia con los modificadores de parámetro ref o out.

- **switch**: a diferencia de la instrucción **switch** de C++, C# no admite el paso explícito de una etiqueta case a otra:

```
switch (caseSwitch)
{
    // El siguiente switch produce un error
    case 1:
        Console.WriteLine("Case 1...");
        // poner un break aquí para que no de error
    case 2:
        Console.WriteLine("... and/or Case 2");
        break;
}
```

C# vs C++

- **Delegados**: los delegados de C# son similares a los punteros a función de C++, son seguros y proporcionan seguridad de tipos:

```
// declaro un tipo nuevo de delegado
public delegate void Del(string message);

// Crea un método para el delegado
public static void DelegateMethod(string message)
{
    System.Console.WriteLine(message);
}

// Instancia el delegado
Del handler = DelegateMethod;
// Llama al delegado
handler("Hello World");
```

C# vs C++

- **Cabeceras**: En C# no se utilizan archivos de encabezado.

- **Cadenas**: en C++, una cadena es simplemente una matriz de caracteres. En C#, las cadenas son objetos de tipo **string** que admiten métodos sólidos de búsqueda y manejo sencillo.

- **Globales**: en C# no se admiten métodos y variables globales. Los métodos y las variables deben estar contenidos en class o struct.

- **Punteros**: C# no usa punteros (aunque los soporta en modo unsafe por compatibilidad).

- **Destructores**: los destructores no se llaman de forma determinista en C# ya que dependen de cuando el recolector de basuara detecta que el objeto ya no tiene referencias.

C# vs C++

- **Objetos**: todo en C# es un objeto incluido los tipos primitivos

```
// guardar la referencia de un tipo valor (boxing)
object x = 7;
// tomar el valor de un objeto (unboxing)
int y = (int)x;
// Se puede llamar a los métodos de intr32 sobre un literal
string s = 425.ToString();
```

- **Propiedades**: C# añade el concepto de propiedades para sustituir a los Get() Set() para encapsular valores.

- **Enumeradores:** Están presentes en C# y son mucho más versátiles que sus equivalentes de C++, dado que son sintácticamente estructuras de pleno derecho, y soportan varias propiedades y métodos a través de la clase Enum.

C# vs C++

- Más información:

http://www.iesvelazquez.org/joomla/images/stories/marcoelio/Segundo/Primera_Eval/c_para_programadores_c.pdf