


|   |            |  |             |     |       |        |  |
|---|------------|--|-------------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |  |

## Tema 1 – De Java a .Net

Tras un primer año de introducción a la programación en lenguaje Java, en este segundo curso se continuará la programación en dicho lenguaje en algunas de las asignaturas. Sin embargo en otras vamos a completar el perfil programador del alumno introduciéndonos en otra gran tecnología de programación ampliamente extendida hoy en día que es el Framework .Net de Microsoft.


La motivación es clara: es el punto de referencia para realizar la programación de cualquier sistema Microsoft: ordenadores, móviles, servidores Web, tablets, etc... que estén basadas en un SO Windows podrán ser programadas mediante este Framework.

Además MS está realizando un esfuerzo y uniéndose con la comunidad de software libre para convertir este Framework en realmente multiplataforma a través de proyectos como Mono y Xamarin.

Como ya conocemos un lenguaje, unas librerías y un entorno de desarrollo no partiremos de cero en esta nueva herramienta. Nos basaremos en lo aprendido el año pasado para facilitar en gran medida el salto de Java a C# que será el lenguaje .Net que trabajaremos (Además de ser el más parecido a Java, lo que es una ventaja).

Se puede ver en qué consiste el Framework .Net en el Apéndice II de este documento. Introduciremos en este capítulo en IDE Visual Studio Community y luego las bases del lenguaje C# de una forma muy ágil partir de las mismas bases de Java.



|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

## El IDE Visual Studio

En el mundo de la programación .Net existen varios IDEs, nosotros concretamente vamos a trabajar con el Visual Studio de Microsoft. Si accedemos a la web de MS para descargarlo:

<https://www.visualstudio.com/es-es/downloads/download-visual-studio-vs.aspx>

encontramos las siguientes posibilidades:

- Community Edition: Versión gratuita que permite desarrollar aplicaciones .Net en distintos lenguajes y distintos sistemas operativos. Realmente en sistemas Unix (linux y macOS) es una adaptación del proyecto Xamarin.
- Enterprise: Versión de pago con herramientas avanzadas para el desarrollo. Existe una versión intermedia (Professional) de pago pero más barata y con menos herramientas pensada para equipos pequeños.
- Code: Editor muy potente que tiene posibilidades de reconocimiento sintáctico, control de versiones, depuración y admite extensiones. Es multiplataforma y lo encontramos para Linux, Mac OS y Windows.

En <https://www.visualstudio.com/es/vs/compare/?rr=https%3A%2F%2Fwww.google.es%2F> se pueden ver esquemáticamente las diferencias.

El curso pasado en la asignatura Entornos de Desarrollo se profundizó en cierta medida en el manejo de aquellas herramientas que facilitaban la tarea del día a día al programador. Aún habiéndonos centrado en un IDE en concreto, todos los conceptos vistos son genéricos de cualquier IDE actual. Por tanto cuando arrancamos el Visual Studio nos vamos a encontrar con una aplicación que tiene un montón de opciones muy conocidas por nosotros. Esto hará que sea fácil adaptarnos a este IDE.

Se puede ver una comparativa de las distintas versiones de Visual Studio y su evolución histórica aquí:

[http://en.wikipedia.org/wiki/Microsoft\\_Visual\\_Studio#Editions\\_feature\\_grid](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio#Editions_feature_grid)

Como curiosidad en la siguiente Web se pueden ver los precios de la herramienta completa:


<https://www.microsoft.com/es-es/store/collections/visualstudio>

*Nota: En el momento de revisar este tema la versión de Visual Studio en distribución es la 2019*

## Uso básico

Veamos mediante un ejemplo guiado cómo es el ciclo de creación y depuración de una aplicación en Visual Studio .

1. Ejecuta la aplicación Visual Studio. Tras ejecutar el IDE nos sale una ventana principal en la que nos permite crear un proyecto nuevo o abrir uno reciente. Por supuesto esto se puede hacer también desde el menú de la aplicación.

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

2. Crea un nuevo proyecto. Al crear un nuevo proyecto nos sale la posibilidad de varios tipos de proyecto. Esto configura de forma automática el entorno para que tenga acceso a las librerías que podamos necesitar. Inicialmente trabajaremos con dos de estas posibilidades: Aplicación de consola para realizar programas ejecutables en línea de comandos (sin gráficos) y Aplicación de Windows Forms para realizar aplicaciones de escritorio. Más adelante veremos otras posibilidades según nos vayan haciendo falta.

En este punto se puede observar la amplitud de posibilidades del Visual Studio que permite trabajar realmente en una gran cantidad de lenguajes y de plataformas destino. También permite hacer programación de móviles ya no solo Windows Phone si no también de iOS y Android a través de Xamarin.

<https://docs.microsoft.com/es-es/xamarin/android/get-started/>

<https://blogs.msdn.microsoft.com/esmsdn/2017/05/09/guia-para-principiantes-de-xamarin/>

<https://thatcsharpguy.github.io/post/xamarin-como-empiezo/>


Incluso permite acceder a distintas plantillas y ejemplos a través de internet.

En muchos casos verás el mismo tipo de proyecto pero con una librería distinta. Por ejemplo si seleccionas proyectos de consola, verás que tienes la librería .Net Core y .Net Framework. La diferencia entre estas dos es que la Core es multiplataforma y la Framework es solo de Windows. La ventaja de esta última es que dispone de algunas librerías añadidas que pueden ser muy útiles en este sistema operativo ya que no tiene la obligación de ejecutarse en distintas plataformas.

3. Filtra por Windows y Desktop.
4. Selecciona Aplicación de Windows Forms en C# (Ojo porque existe también en VB) para este ejemplo. Dale como nombre "NavegadorWeb".

El manejo de elementos gráficos veremos que se nos hará muy cómodo debido al diseñador de formularios que no usábamos el año pasado. De hecho al empezar un proyecto ya vemos el formulario y nada de código. Tenemos a la derecha el Explorador de soluciones y a la izquierda el diseñador de formularios.

5. Pincha en el Explorador de Soluciones en Form1.cs. Se desplegará y podrás seleccionar en la barra de herramientas superior o en el menú contextual "Ver Código". Veremos, explicaremos y analizaremos más adelante todo el código generado por el diseñador.
6. A la izquierda está oculto el "Cuadro de herramientas". Pincha sobre el y déjalo fijo si te resulta más cómodo mediante el icono de la chincheta. Si se tiene seleccionado el diseño del formulario, en este cuadro se verán los distintos componentes que se pueden colocar en el formulario. Prueba a arrastrar algún botón o caja de edición.
7. Coloca una etiqueta (Label), una caja de texto (TextBox) y un botón (Button) en el formulario. Cada uno de estos elementos como ya sabes es un objeto con sus propiedades. Para cambiar dichas propiedades selecciona mediante el menú contextual la opción "Propiedades" para que salga un cuadro donde puedes cambiar de forma cómoda las propiedades del objeto. Si ya tienes dicha

|   |            |  |             |     |       |        |
|---|------------|--|-------------|-----|-------|--------|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |

ventana abierta con seleccionar el componente ya llega.

- Cambia el campo Text de la etiqueta (URL) y el del botón (Allá voy!).
- Coloca también un componente *WebBrowser*. Ajústalo de forma que ocupe la parte inferior del formulario y cambia la propiedad Anchor de forma que incorpore los valores Top, Left, Right y Bottom. Esto hace que al cambiar el tamaño del formulario también cambie el del componente pues se mantiene la distancia a cada punto de anclaje.
- Ejecuta pulsando el icono Play o F5 o en el menú *Depurar* → *Iniciar Depuración*. Verás que se ejecuta nuestro diseño aunque por ahora no haga nada: Hemos situado los componentes pero no hemos programado los Eventos a los que tienen que responder.

Desde la versión Community al ejecutar también aparece un monitor de recursos de la aplicación. No lo veremos por el momento, pero nos da una idea del consumo de memoria y CPU de nuestro programa en ejecución.

- Haz doble clic sobre el botón. Aparece la pantalla de código y en concreto aparece el método siguiente:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Este método está automáticamente enlazado con el evento clic del botón (veremos como se realiza este enlace más adelante). Por tanto el código que escribamos aquí se ejecutará al pulsar el botón.

- Pruébalo añadiendo esta línea de código:


```
webBrowser1.Navigate(textBox1.Text);
```

Ejecútalo y prueba que funciona correctamente.

- Vamos a ver la depuración. Para ello modificamos ligeramente el código anterior para hacerlo un poco más largo con el objetivo de establecer un punto de ruptura y ejecutar paso a paso. Sustituye el código anterior por el siguiente:

```
string url= textBox1.Text;
if (url.Trim() == "")
{
    url = "www.duckduckgo.com";
}
webBrowser1.Navigate(url);
this.Text = "Estamos en: " + url;
```

- Realiza doble clic sobre el margen izquierdo de la ventana de edición de código. Aparecerá un círculo granate que indica que ahí se ha puesto un punto de ruptura.
- Ejecuta de nuevo. En cuanto establezcas la URL y pulses el botón aparecerá el código. Una flecha amarilla en el margen indica la línea en la que estás. En la parte inferior se ven las variables locales. Si quieres ver tus variables u objetos puedes verlos en la ventana inspección.

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

Para ejecutar paso a paso se usa F11 y paso a paso sin entrar en los procedimientos con F10.

Puedes ver otras teclas y posibilidades en el menú Depuración.

- Finalmente para obtener la ayuda sobre algún componente (como los de Java son muy amplios en cuanto a métodos, propiedades, eventos, etc...) lo mejor es situarse sobre el con un clic simple y pulsar F1 lo que te lleva al MSDN que es la biblioteca de ayuda de Visual Studio .Net.

### Otros entornos/Frameworks

Debido a la popularización de .Net y de las grandes posibilidades que tiene han surgido otros entornos de desarrollo que conviene tener en cuenta:

**MonoDevelop:** Basado en el Proyecto Mono. Es software libre. Es multiplataforma (Linux, Mac y Windows) y no requiere del Framework de Microsoft ya que tiene el suyo propio. Por supuesto es compatible con el de MS. De hecho Xamarin está basado en Mono.

<http://monodevelop.com/>

**SharpDevelop:** Es exclusivamente un entorno de desarrollo, de hecho es necesario tener instalado el Framework .Net para poder programar. Es gratuito, libre y sólo para plataformas Windows.

<http://www.icsharpcode.net/OpenSource/SD/>

**Xamarin:** No es exactamente un entorno si no que es una implementación del Framework .Net multiplataforma para móviles Android e iOS. Tanto Visual Studio como los otros IDEs mencionados permiten desarrollo tanto con el Framework de MS como con Xamarin.

<https://www.xamarin.com>

### Code Snippets (Fragmentos de código).


Al igual que en otros IDEs había la posibilidad de escribir código de forma rápida en VS existe los Code Snippets por si alguien quiere escribir de forma rápida ciertos fragmentos usando XML.

Por ejemplo `cw` y doble `TAB` escribe `Console.WriteLine();`

Lo existentes puedes verlos en Tools→Code Snippets Manager. Si quieres diseñar los tuyos propios puedes leer más en:

<https://docs.microsoft.com/es-es/visualstudio/ide/walkthrough-creating-a-code-snippet?view=vs-2017>

<https://docs.microsoft.com/es-es/visualstudio/ide/code-snippets-schema-reference?view=vs-2017>

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

## Introducción a C# desde Java

Al igual que Java, C# es un lenguaje orientado a objetos y por tanto todo se distribuye en clases. Sin embargo hay un par de conceptos estructurales que están por encima de las clases que conviene conocer:

### Namespaces (Espacios de nombres):

Es una forma de agrupar los diferentes elementos (clases, interfaces, módulos, estructuras...) de forma lógica y jerarquizada. Una especie de Librería. Es equivalente a los packages de Java.

Por ejemplo en una aplicación de gestión que es un Namespace puedo tener dentro a su vez un Namespace donde se agrupan las clases de contabilidad y otro en el que se agrupan las de control de stocks.

Cuando queremos acceder a un namespace ajeno, se debe usar el nombre de dicha namespace, de forma similar a como accedemos a las propiedades o métodos de los objetos. O usar la sentencia *using* en la parte superior del código.

Un espacio de nombres puede estar repartido entre varios ejecutables o DLLs o incluso en un mismo programa se pueden establecer diversos espacios de nombres.


### Ensamblados

Son bloques ejecutables. Es decir son los clásicos .exe y .dll que componen una aplicación incluyendo además los recursos como imágenes, textos, etc... con la ventaja de no tener que cargarlos desde un soporte externo. Puedes verlo de forma equivalente a los .jar de Java.

Una DLL (Dinamic Link Library) es un módulo que tiene en su interior funciones ejecutables, clases, etc... y que pueden ser llamadas desde distintas aplicaciones. Por ejemplo las DirectX son un conjunto de dll que proporcionan clases y funciones para manejo de gráficos, audio, etc... Son como librerías pero ya compiladas y preparadas para ejecutar.

Puedes ver las funciones de las DLLs instaladas en tu Windows mediante programas como:

[http://www.nirsoft.net/utls/dll\\_export\\_viewer.html](http://www.nirsoft.net/utls/dll_export_viewer.html)

|   |            |  |        |     |       |        |
|---|------------|--|--------|-----|-------|--------|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |

### Estructura básica de un programa en C#

Al crear un nuevo proyecto de consola (.Net Framework) podemos ver la estructura mínima de un programa C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Bienvenido a la programación en C#");
            Console.ReadLine();
        }
    }
}
```

Los *using* iniciales son similares a los *import* de Java.

El namespace es equivalente al package pero con la salvedad de que tiene llaves de inicio y fin de forma que en un archivo podría tener varios namespaces.

La función principal de C# es la estática Main (fíjate en que empieza por mayúscula).

Las líneas dentro del Main realizan tareas ya conocidas: WriteLine escribe una línea en la consola terminándola en un retorno de carro y ReadLine es similar al nextLine() del Scanner de Java.


Por supuesto lo anterior lo genera automáticamente el VS (Visual Studio) y no todas las líneas son necesarias en todos los programas. Podríamos reducirlo aún más:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Bienvenido a la programación en C#");
        Console.ReadLine();
    }
}
```

De hecho si crearas un proyecto consola .Net Core aparecería así, pues las otras librerías son de Windows.

Hemos dejado sólo el using System ya que contiene la clase Console. Si quisiéramos quitar este using podríamos hacerlo metiendo el System en la línea de Console de la siguiente forma:

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

```
class Program
{
    static void Main()
    {
        System.Console.WriteLine("Bienvenido a la programación en C#");
        System.Console.ReadLine();
    }
}
```

También se ha modificado el Main ya que dispone de varias sobrecargas con y sin parámetros.

Teniendo estas cuestiones básicas en cuenta empezaremos a ver los distintos elementos del lenguaje diferenciándolos con el Java.

### Comentarios

Existen al igual que en java los comentarios de línea mediante doble barra // y los de varias líneas mediante barra-asterisco /\* \*/

En el caso de los comentarios de documentación en el caso de C# se realizan con triple barra /// y se usan tags XML. Con programas como el SandCastle de Microsoft se puede generar documentación CHM o HTML a partir de estos comentarios. Probablemente veamos algo más adelante.

### Lista de palabras reservadas

No pueden ser usadas como identificadores


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

**Se podría usar** como identificador una palabra reservada si se le **antepone el carácter @**. Esto permite usar luego si es un nombre de una propiedad, acceder a través del objeto y dicho nombre sin @. Por ejemplo miObjeto.class si se definió la propiedad @class. Ojo, es una práctica que puede ser confusa y por ello no es recomendable.

### Literales

Los literales se manejan prácticamente igual que en Java. Los recordamos:



|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

*Enteros:* Simplemente se escribe la cifra. Se admite decimal y hexadecimal.

Si se desea escribir el número en hexadecimal, se antepone el prefijo 0x. Por ejemplo 0xFF.

Solo desde la versión 7 de C# se admite también en binario con el prefijo 0b, por ejemplo 0b1001 sería 9 en decimal.

Para negativos, se antepone el signo menos (–)

*Reales:* Admiten tanto notación habitual con punto como científica con E o e. Ejemplo 15.21, 3.02e10

*Lógicos:* true y false.

*Carácter:* Se encierran entre comillas simples: 'a', 'P',...

También se pueden representar mediante su código UNICODE. Para ello se usa la “secuencia de escape” \u. Es decir, si se pone entre comillas simples \u, el C# entiende que lo que va a continuación es el código hexadecimal del carácter que se desea escribir. Por ejemplo '\u002A' hace referencia al asterisco (de hecho es lo mismo que escribir '\*').

Se pueden ver los distintos valores UNICODE de los caracteres en el mapa de caracteres (charmap).

Además del uso del código UNICODE, algunos caracteres especiales y de control tienen su propia secuencia de escape al igual que en Java:

| Carácter              | Código de escape Unicode | Código de escape especial |
|-----------------------|--------------------------|---------------------------|
| Comilla simple        | \u0027                   | '                         |
| Comilla doble         | \u0022                   | "                         |
| Carácter nulo         | \u0000                   | \0                        |
| Alarma                | \u0007                   | \a                        |
| Retroceso             | \u0008                   | \b                        |
| Salto de página       | \u000C                   | \f                        |
| Nueva línea           | \u000A                   | \n                        |
| Retorno de carro      | \u000D                   | \r                        |
| Tabulación horizontal | \u0009                   | \t                        |
| Tabulación vertical   | \u000B                   | \v                        |
| Barra invertida       | \u005C                   | \\                        |

|                              |            |  |        |     |       |        |
|------------------------------|------------|--|--------|-----|-------|--------|
| COLEXIO<br><b>VIVAS</b> S.L. | RAMA:      | Informática  | CICLO: | DAM |       |        |
|                              | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |
|                              | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |
|                              | UNIDAD     | COMPETENCIA  |        |     |       |        |

Cadena:

Va entre comillas dobles ("esto es una cadena").

Si se desea que no se interpreten los caracteres de escape, se debe anteponer la arroba a la cadena. En general la arroba hace que la cadena sea todo lo que se encuentra entre la primera y la última comilla doble.

Por ejemplo:

```
Console.WriteLine(@"Un \ttabulador\nlínea distinta");
```

Escribirá en pantalla:

```
Un \ttabulador\nlínea distinta
```

*Nulo:*

Palabra reservada *null*. Se usará principalmente para inicializar objetos.

### Tipos de datos y variables

En C# todos los tipos de datos son objetos (herencia común de *object*). Todos. Es decir, así como en Java existían tipos primitivos (int, char, double, ...) estos no existen en C# de forma que cualquier valor o variable es un objeto y se podrá tratar como tal con sus propiedades y sus métodos. Por tanto el siguiente código es correcto:

```
string cadena = 10.ToString();
```

El propio número 10 es un objeto y por eso admite el operador punto.

Si declaras un char, al usar el operador punto ocurre lo siguiente:

```
string cadena = 10.ToString();
char a = 'w';
a.
Sy [CompareTo
Sy [Equals
[GetHashCode
[GetType
[GetTypeCode
[ToString]
teLine("Bienve
dLine();
```

se despliega las posibilidades que tengo para el *char*. Son pocas porque hereda directamente de la clase

|                              |            |  |        |       |
|------------------------------|------------|--|--------|-------|
| COLEXIO<br><b>VIVAS</b> S.L. | RAMA:      | Informática  | CICLO: | DAM   |
|                              | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |       |
|                              | PROTOCOLO: | Apuntes clases                                     | AVAL:  | DATA: |
|                              | UNIDAD     | COMPETENCIA  |        |       |

object. Por tanto en C# se puede guardar cualquier tipo en un *object* ya que todo hereda de este.

Sin embargo los tipos que considerábamos primitivos en Java, en C# siendo objetos realmente no se comportan como tales en cierto aspecto. Cuando asignas un entero a otro tenemos dos variables apuntando (referenciando) al mismo dato si no que se hace copias.

Esto conlleva a que se diferencia en C# dos tipos de variables: **variables valor y variables referencia**. Las variables valor se pueden manejar en cuanto a igualdad como los primitivos de java, directamente contienen el valor, no una referencia al mismo.

Por el contrario los tipos referencia sí contienen una referencia al dato. En cuanto a **tipos base** sólo hay dos por referencia: object y string y este último tiene sobrecargado el operador igual (sí, en C# se pueden sobrecargar operadores) de forma que funciona como en una variable por valor. También tiene sobrecargado el operador == para hacer comparaciones más cómodas.

Declaración de variables: Idéntica a Java. Ejemplos:


```
byte a;
char car = 'A';
int b, num1;
long num2 = 152, num3 = 998792;
```

En C# los tipos base realmente son alias (forma distinta de escribirlo. Se puede crear un alias con la directiva using) de tipos del CTS común a todos los lenguajes .Net. La tabla de equivalencias es la siguiente:

| Tipo CTS | Descripción                          | Bits     | Rango de valores  | Alias   |
|----------|--------------------------------------|----------|---|---------|
| SByte    | Bytes con signo                      | 8        | [-128, 127]   | sbyte   |
| 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       | [1,5×10 <sup>-45</sup> - 3,4×10 <sup>38</sup> ]         | float   |
| Double   | Reales de 15-16 dígitos de precisión | 64       | [5,0×10 <sup>-324</sup> - 1,7×10 <sup>308</sup> ]       | double  |
| Decimal  | Reales de 28-29 dígitos de precisión | 128      | [1,0×10 <sup>-28</sup> - 7,9×10 <sup>28</sup> ]         | 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  |

En C# se escribirá habitualmente el alias, pero el programador tendrá en cuenta el tipo CTS que está por debajo.

Pero a pesar de ser objetos, los tipos valor como int o float no admiten ser asignados a null. Esto en

|   |            |  |             |     |       |        |  |
|---|------------|--|-------------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |  |

ocasiones es un inconveniente (imagina una BD que no tiene un valor determinado para un entero, debería ser null). Para solventar esto último se puede usar los tipos “nullable” que es exactamente la misma definición que ya conocemos pero acabada en ?. Es decir int?, float?, bool?, etc. De esta forma se puede hacer:

```
int c = null; //Error!!!
int? a = null;
float? b = null;
```

### Constantes

Se definen de la siguiente forma

```
const <tipo_constante> <nombre_constante> = <valor>;
```

Por ejemplo

```
const double pi=3.14159265;
```

Cómo en Java hay que tener en cuenta el tipo base que se le asigna, para lo cual conviene usar en ciertos casos los subfijos:

```
const float pi=3.14F;
```

Subfijos enteros:


| Sufijo   | Tipo del literal entero |
|--|-------------------------|
| ninguno  | int                     |
| <b>L ó l (no recomendado, confusión con 1)</b> | <b>long</b>             |
| <b>U ó u</b>                                   | <b>int</b>              |
| <b>UL, UI, uL, uI, LU, Lu, IU ó lu</b>         | <b>ulong</b>            |

Subfijos reales:

| Sufijo                | Tipo del literal real |
|-----------------------|-----------------------|
| <b>F ó f</b>          | <b>float</b>          |
| ninguno, <b>D ó d</b> | <b>double</b>         |
| <b>M ó m</b>          | <b>decimal</b>        |

### Operadores

Son los mismo que en Java. Los resumimos:

|   |            |  |             |     |       |        |
|---|------------|--|-------------|-----|-------|--------|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |

#### Aritméticos:

```

+      suma
-      resta
*      multiplicación
/      división
%      resto
=      asignación

```

#### Lógicos:

```

&&: And (Se puede usar & que evalúa ambos resultados)
||: Or (Se puede usar | que evalúa ambos resultados)
!: not
^: xor
== "igual que"
!= "distinto de"
<,>,<=, >= menor, mayor, menor o igual, mayor o igual respectivamente

```

#### Bit a bit:

```

& and
| or
~ not (AltGr+4)
^ xor
<< desplazamiento a la izquierda
>> desplazamiento a la derecha

```

#### Asignación:

```

=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>= , ++, --

```

#### De concatenación de cadenas y caracteres:

```

+, +=

```

#### Conversión de tipos

El casting que aprendimos a usar en Java funciona también en C#:

```

int a;
long b = 13;
a = (int)b;

```


pero tenemos también métodos de conversión y comandos relacionados que vemos a continuación:

Una forma típica es usar la clase *Convert* de la siguiente forma:

```

Destino = Convert.ToXXXX(Origen);

```

|   |            |  |             |     |       |        |  |
|---|------------|--|-------------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |  |

donde XXXX representa el tipo destino. *Convert* contiene una serie de métodos estáticos para realizar conversiones. Por ejemplo

```
long b;
b = Convert.ToInt32("121");
```

Hay que conocer el tipo CTS para saber qué queremos convertir.

También se puede usar el método *Parse* que existe de forma estática en algunas de las clases:

```
long b;
b = long.Parse("121");
```

tiene el mismo resultado que el caso anterior.

La diferencia entre los dos casos es que el *Convert* primero comprueba que el string no sea null, si lo es devuelve 0. El *Parse* lanza una excepción *ArgumentNullException* si el string es nulo.

Para convertir cualquier tipo a cadena se usa el método *ToString()* que viene heredado ya de *object*. Es más, en nuestras clases se puede sobrescribir el método para que funcione según nuestros intereses.

Existe también (aunque lo usaremos menos) para hacer conversiones el operador *as*:

```
expresión_o_variable as tipo_de_dato
```

Esto devuelve la variable en el nuevo tipo. De todas formas esto solo vale para tipos por referencia (punteros), ya que no hace exactamente una conversión de tipo si no que se conserva el tipo original pero fuerza al programa a usarlo como el nuevo tipo. Esto hace que sea más eficiente, pero hay que ser más cuidadoso al usarlo. Es otra manera de realizar un casting. Se aplica en polimorfismo.

## Números aleatorios

Usaremos la clase *Random* para instanciar un objeto que nos permita generar números aleatorios:

```
Random generador = new Random();
```


El objeto generador dispone entre otros de los siguientes métodos de generación de números aleatorios:

*Next()*: Devuelve un número *int* aleatorio positivo.

*Next(int max)*: Devuelve un número aleatorio positivo menor que el valor máximo especificado.

*Next(int min, int max)*: Devuelve un número aleatorio que se encuentra en el intervalo especificado. Incluye *min* pero no *max*.

*NextDouble()*: Devuelve un número de punto flotante de doble precisión que es mayor o igual que 0.0 y

|   |            |  |        |     |       |        |  |
|---|------------|--|--------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |  |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |  |

menor que 1.0.

Por ejemplo:

```
Random generador = new Random();
double d;
int i;
d = generador.NextDouble(); //Nº real entre 0 y 1 (excluido)
i = generador.Next(1, 7); //Nº entero entre 1 y 7 (excluido, es decir, de 1 a 6)
```

Nunca debe meterse el `new Random()` en un bucle para generar varios números aleatorios pues usa como semilla el mismo instante temporal (por ser muy rápido) y siempre sacaría el mismo número.

### Declaración mediante var.

En c# se pueden hacer declaraciones implícitas no diciendo qué tipo es pero dejando que sea el compilador quién o decida usando el tipo *var*.

```
var i = 10; // tipado implícito
int i = 10; //tipado explícito
```

No la usaremos mucho pues es necesaria principalmente para gestionar lo que se denominan tipos anónimos que probablemente no veremos por falta de tiempo.

### E/S en Proyectos de Consola

Usaremos para leer datos `Console.ReadLine()` que es una función que nos devuelve un dato que introduce el usuario de forma similar al `nextLine` del `Scanner` de Java. Siempre devuelve un *string* por lo que se deben hacer las conversiones pertinentes.


Existe también `Console.ReadKey()` que espera a la pulsación de cualquier tecla y la devuelve como tipo `ConsoleKeyInfo` en el que no profundizaremos. Esto puede ser útil para establecer interacción con el usuario sin tener eco por pantalla.

Para escribir datos en pantalla usaremos básicamente `Console.WriteLine()` o `Console.Write()`. Este último no hace un retorno de carro.

Dentro de una cadena en un `Write` o `WriteLine` se pueden usar cualquiera de las secuencias de escape vistas, como por ejemplo el retorno de carro (`\n`).

Cuando queremos mostrar varias variables dentro de un texto podemos hacer lo siguiente:

- Concatenamos usando los operadores ya vistos y teniendo en cuenta las conversiones necesarias.
- Usamos modificadores con llaves: `{0} {1},...` insertadas dentro de la constante cadena. Esos modificadores corresponde con el mismo orden de colocación de las variables al final del `WriteLine` (similar al funcionamiento del `printf`).

|   |            |  |        |     |       |        |  |
|---|------------|--|--------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |  |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |  |

Ejemplo:

```
int a = 10, b = 3, totalsuma, totalresta;
string moneda;
```

```
totalsuma = a + b;
totalresta = a - b;
moneda = "Euros";
```

```
Console.WriteLine("El valor {0} sumado con el {1} resulta un total de {2}
{3}. Pero dicho valor {0} si se le resta {1} queda un total de {4} {3}\n", a, b,
totalsuma, moneda, totalresta);
```

```
Console.WriteLine("Para mostrar {llaves}, si no hay ninguna variable para
mostrar, no hay problema.");
```

```
Console.WriteLine("Si hay alguna, se usan dobles llaves\nLlaves:{{}} Moneda:
{0}", moneda);
```

Mediante estas, también se puede dar formato de números decimales y de espacio ocupado por una variable:

*{modificador, nºdigitos:.000}* Tantos 0 como cifras decimales

```
double pi = 3.14159265;
Console.WriteLine("{0,10:.000}", pi); // También es válido "{0,10:F3}"
```

Existe la función estática *String.Format* que devuelve la cadena con formato de la misma forma que trabaja *WriteLine*. Más información sobre formatos:

[http://msdn.microsoft.com/es-es/library/txafckwd\(v=vs.80\).aspx](http://msdn.microsoft.com/es-es/library/txafckwd(v=vs.80).aspx)

Finalmente comentar que la clase *Console* dispone de otros métodos que mejoran el manejo y aspecto de la consola. Por ejemplo *SetCursorPosition* permite colocar el cursor en cierta coordenada o *ForegroundColor* y *BackgroundColor* permite cambiar los colores del texto y fondo de la consola. Esto permite hacer aplicaciones más amigables con el usuario o incluso videojuegos sencillos en consola.

## Control de flujo

En C# las sentencias de control de flujo son prácticamente idénticas a Java, por lo que las veremos por encima parándonos sólo en aquellos puntos diferenciadores.


Las sentencias selectivas **if**, **if-else** son idénticas a Java.

Las estructuras repetitivas **while**, **do-while** y **for** también son idénticas a las equivalentes de Java.

Las clausulas **break**, **return** y **continue** funcionan igual que en Java.

Existe un bucle **for** extendido (**foreach**) que veremos más adelante.



|   |            |  |        |     |       |        |  |
|---|------------|--|--------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |  |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |  |

### Sentencia switch

Es similar al switch de Java con la salvedad que además de break se pueden poner otras sentencias de ruptura del caso. Además dispone de más posibilidades. Estructura:

```
switch (<expresión>)
{
    case <valor1>:    <bloque1>
                    <siguienteAcción>
    case <valor2>:    <bloque2>
                    <siguienteAcción>
    ...
    default:         <bloqueDefault>
                    <siguienteAcción>
}
```

El manejo es igual que en Java, sólo cambia el elemento marcado como <siguienteAcción> colocado tras cada bloque de instrucciones indica qué es lo que ha de hacerse tras ejecutar las instrucciones del bloque que lo preceden.

Puede ser uno de estos tres tipos de instrucciones y es obligatorio ponerlas salvo que en el case no haya nada lo que lo haría pasar al siguiente:

```
goto case <valor_i>;
goto default;
break;
```

Si es un **goto case** indica que se ha de seguir ejecutando el bloque de instrucciones asociado en el switch a la rama del <valor\_i> indicado.

Si es un **goto default** indica que se ha de seguir ejecutando el bloque de instrucciones de la rama default.

Si es un **break** indica que se ha de seguir ejecutando la instrucción siguiente al switch, al igual que sucedía en Java.


Además como ocurría en la versión 7 de Java el switch de C# admite cadenas.

Ejemplo:

```
string nom;

Console.WriteLine("Por favor introduce tu nombre");
nom = Console.ReadLine();

switch (nom)
{
    case "adios":
    case "Adios":
    case "ADIOS":
        Console.WriteLine("Hasta la próxima");
        break;
    case "Curro":
```

|   |            |  |             |     |       |        |  |
|---|------------|--|-------------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |  |

```

        Console.WriteLine("Acceso denegado, cámbiate el nombre");
        goto case "ADIOS";
    default:
        Console.WriteLine("Hola {0}, bienvenido a este programa", nom);
        break;
}

```

Lo visto aquí es un ejemplo de la potencia de esta sentencia. Pero permite mucho más como aplicación directa de polimorfismo, usar sentencias case condicionales o incluso establecer rangos (desde C# 7). Puedes ver más ejemplos en:

<https://www.dotnetperls.com/switch>

<https://stackoverflow.com/questions/20147879/switch-case-can-i-use-a-range-instead-of-a-one-number>

## Métodos

La creación de métodos o funciones en C# es prácticamente igual a la vista el año pasado en Java. La principal diferencia se encuentra en el paso de parámetros y es donde nos vamos a centrar.

Veámos en Java que los parámetros es una forma de darle datos a la función para trabajar con ellos y, si es necesario, que la función devuelva algún dato mediante la sentencia return.

En C# además de esta posibilidad de uso de parámetros llamada paso de **parámetros por valor o parámetros de entrada**, existe otra posibilidad y es la de pasarle como parámetro una variable que sí sea modificada. A esto se le llama **parámetros de salida o parámetros por referencia**.

Para pasar un parámetro por referencia simplemente se especifica con la abreviación **ref** tanto en la definición de la función como en la llamada.


```

static void suma (int a, int b, ref int c){
    c=a+b;
}

static void Main()
{
    int res=0;
    suma(12, 34, ref res);
    Console.WriteLine("Resultado: {0}", res);
}

```

En el caso anterior es necesario inicializar *res* en el programa principal si no se provoca un error de compilación indicando que la variable no está inicializada. Esto se puede evitar usando una variante del parámetro por referencia denominado **out** que obliga a que el o los parámetros por referencia sean inicializados dentro de la función por lo que no es obligatorio inicializarlos antes. Modifica el programa de la

|   |            |  |        |     |       |        |  |
|---|------------|--|--------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |  |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |  |

siguiente forma y pruébalo:

```
static void suma (int a, int b, out int c){
    c=a+b;
}

static void Main()
{
    int res;
    suma(12, 34, out res);
    Console.WriteLine("Resultado: {0}", res);
}
```

Ahora no da error de compilación porque el *out* implica que *res* se inicializará dentro de la función.

### Parámetros con nombre y opcionales

En C# está permitido pasar los parámetros en cualquier orden si se especifica el nombre del parámetro.

También si se inicializa un parámetro con un valor, dicho parámetro se convierte en opcional. Veamos un ejemplo de ambas cosas. Sea la función:

```
static void suma10(int a, int b = 10)
{
    Console.WriteLine(a + b);
}
```

Puede ser llamada desde el Main de estas tres formas:

```
suma10(3, 4);
suma10(b:4,a:3);
suma10(3);
```

### Directivas del preprocesador (o del compilador)

En ocasiones conviene darle ciertas indicaciones al compilador para que varíe la forma en que realiza la compilación. Para ello se usan las directivas del preprocesador. Lo que hacen es “preprocesar” el código antes de ser compilado.


El formato de dar esas indicaciones es:

```
#<nombre de la directiva> <valor de la directiva>
```

Veamos algunos ejemplos que se ampliarán durante el curso de ser necesarios. La principal utilidad es indicar que partes del código serán compiladas y cuales no. Para ello se usan las siguientes directivas.

#### Directiva define:

El formato es:

|   |            |  |        |     |       |        |
|---|------------|--|--------|-----|-------|--------|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |

```
#define <identificador>
```

Esta directiva indica si cierto identificador está definido. Por ejemplo

```
#define PRUEBA
```

No es obligatorio pero es costumbre escribir los identificadores del preprocesador con mayúsculas.

Es obligatorio que las directivas **define** se encuentren **al principio del código**.

### Directivas de compilación condicional

Son las directivas que nos permiten decidir qué se compila y qué no se compila en nuestro código (normalmente esto va asociado a que tenemos código de depuración que en algunas ocasiones interesa compilar y en otras no). La estructura es

```
#if <condición1>
    <código1>
#elif <condición2>
    <código2>
...
#else
    <códigoElse>
#endif
```

*elif* es equivalente a *else if*


Escribe y ejecuta el siguiente ejemplo:

```
#define PRUEBA

using System;

class A
{
    public static void Main()
    {
        #if PRUEBA
            Console.Write ("Esto es una prueba");
            #if TRAZA
                Console.Write(" con traza");
            #elif !TRAZA
                Console.Write(" sin traza");
            #endif
        #endif
    }
}
```

Prueba ahora a definir el identificador TRAZA y volver a ejecutarlo.

|   |            |  |        |     |       |        |
|---|------------|--|--------|-----|-------|--------|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |

Como veremos, el carácter ! Indica NOT. Otras posibilidades:

! para "not"

&& para "and"

|| para "or"

== operadores relacionales de igualdad

!= desigualdad

() paréntesis

true y false

Por ejemplo:

```
#if TRAZA // Se cumple si TRAZA esta definido.
#if TRAZA==true // Ídem al ejemplo anterior aunque con una sintaxis menos cómoda
#if !TRAZA // Sólo se cumple si TRAZA no está definido.
#if TRAZA==false //Ídem a al ejemplo anterior pero con una sintaxis menos cómoda
#if TRAZA == PRUEBA // Solo se cumple si tanto TRAZA como PRUEBA están
// definidos o si ninguno lo está.
#if TRAZA != PRUEBA // Solo se cumple si TRAZA esta definido y PRUEBA no o
// viceversa
#if TRAZA && PRUEBA // Solo se cumple si están definidos TRAZA y PRUEBA.
#if TRAZA || PRUEBA // Solo se cumple si están definidos TRAZA o PRUEBA.
#if false // Nunca se cumple (por lo que es absurdo ponerlo)
```

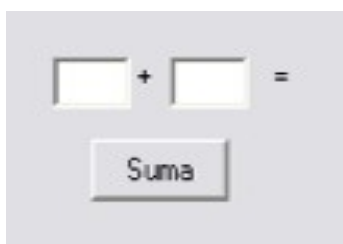
Existen más directivas que se verán si fueran necesarias.

|                              |            |  |        |       |
|------------------------------|------------|--|--------|-------|
| COLEXIO<br><b>VIVAS</b> S.L. | RAMA:      | Informática  | CICLO: | DAM   |
|                              | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |       |
|                              | PROTOCOLO: | Apuntes clases                                     | AVAL:  | DATA: |
|                              | UNIDAD     | COMPETENCIA  |        |       |

## Ejercicios

### Ejercicio 1

Crear un nuevo proyecto Windows y en el formulario colocar 2 cajas de texto y 2 etiquetas de forma que quede de la siguiente manera:



Funcionará de la siguiente forma: El usuario introducirá números en cada casilla, y al darle al botón, aparecerá la suma en la misma etiqueta donde está el =.

Se podrán sumar números Double.

Recuerda que la propiedad text de las TextBox es de tipo cadena.

**Opcional:** Realiza control de excepciones sabiendo que es prácticamente igual a como lo hacías en Java.

### Ejercicio 2

Realizar un programa en la consola que pida dos frases al usuario y que las muestre de las siguientes formas:

frase 1 [tabulación] frase 2

frase 1

frase 2


además si está definida la directiva de compilación FRASE no se verá lo anterior pero sí se verá:  
"frase 1" \frase2\

solo se permite un WriteLine en cada caso.

### Ejercicio 3

Mediante 3 TextBox establecer un Tragaperras con números. Existirá un botón y una etiqueta de información que indica si hay o no premio. Otra etiqueta indica el crédito disponible. Se parte de 50 euros. Una partida cuesta 2 euros. En cuanto se pulse el botón de juego en cada TextBox aparecerá un número aleatorio del 1 al 7. Si salen 3 números iguales se le indicará al usuario que hay premio de 20 euros, si salen 2 iguales hay premio de 5 euros. Un botón de crédito permite meter 10 euros más de crédito.

Para facilitar la depuración se establecerá mediante directivas de compilación que si salen 2 iguales se resten 5 euros en lugar de ganarlos.

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

#### Ejercicio 4

Realizar en modo consola un menú con 3 opciones (juegos) utilizando estructuración mediante funciones (al menos cada juego una función). Al final de cada opción se debe preguntar al usuario si desea volver a ejecutar dicha opción sin volver al menú inicial:

*Opción 1. Pedir un número al usuario del 1 al X, tirar diez dados mostrando el resultado de cada uno e informar al usuario de cuantas veces ha acertado. X es un parámetro que si no se pasa se inicializa por defecto a 6.*

*Opción 2. Juego adivina un número. El ordenador saca un número aleatorio entre 1 y 100. El usuario tiene 5 intentos para adivinarlo. Se informa mayor/menor en cada intento así como el número de intentos restantes.*

*Opción 3. Realizar una quiniela: Se deben dar 14 resultados aleatorios como 1, X ó 2 de forma ponderada. La probabilidad de sacar 1 sea del 60%, la de sacar X sea 25% y la de sacar un 2 sea un 15%. Se debe realizar al menos una función que devuelva un 1 una X o un 2 ponderado de la forma anterior, sacando un número de 1 a 100 y seleccionando el valor devuelto con un switch con clausula when.*

*Opción 4. Jugar a todos. El programa pasará por los 3 apartados anteriores. No se puede llamar simplemente a funciones, es necesario jugar con **goto case**.*

*Opción 5. Salir. Mientras no se seleccione esta opción, el ordenador se mantendrá en el programa.*


**Opcional:** El menú hazlo de forma que resalte la opción con un indicado o con video inverso y que se pueda seleccionar una u otra opción con los cursores (Usa ReadKey, SetCursorPosition, ForegroundColor,... ).

#### Ejercicio 5(Ver Apéndice I)

Realizar un pequeño programa que mediante un TextBox pida un texto al usuario. Tras eso, el usuario deberá pulsar un botón y se le preguntará mediante una MessageBox con Icono de Pregunta si el usuario desea poner ese texto como título del formulario principal y si contesta que sí, cambiar dicho título (Text=título). El texto introducido debe repetirse en la MessageBox como parte del mensaje (concatenación).

#### Ejercicio 6 (Opcional)

Programar un juego “mini-hundir-la-flota” contra el ordenador. Se debe generar una coordenada (de un tablero de 10x10 posiciones) aleatoria (sin mostrar nada en pantalla) y guardarla (barco del ordenador). Luego el usuario debe introducir una coordenada que el ordenador intentará averiguar (barco del usuario). A partir de ese momento y por turnos el usuario meterá coordenadas con la idea de localizar el barco del ordenador. Tras su turno será el ordenador el que diga una coordenada aleatoria y comprobará si coincide con la del barco del usuario. No es necesario comprobar si el ordenador repite coordenadas.

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

*Se debe hacer de una de estas dos formas: O el usuario mete coordenadas del estilo A4 y van apareciendo las tiradas marcadas en un tablero, o se puede realizar una versión más cómoda aún de forma que el usuario ponga el barco y marque las tiradas moviéndose por el tablero con los cursores. Usa control de excepciones.*


**Opcional:** mediante las funciones de consola dibuja la evolución en pantalla.

### **Ejercicio 7 (Opcional)**

*Se trata de hacer una calculadora similar a la de windows (modo gráfico). Se recomienda ir por pasos:*

- *Primero establecer botones de cifras, pantalla, una o dos operaciones y botón de cero. En principio no se trabajará con decimales por parte del usuario (sí para resultados).*
- *Añadir el resto de las operaciones.*
- *Añadir memorias parciales*
- *añadir tecla delete.*
- *Decimales*
- *...*



|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

## Apéndice I: MessageBox.

Componente .NET que sirve para mostrar distintos mensajes en pantalla. Muestra un cuadro de mensaje con el texto, el título, los botones, el icono y el botón predeterminado especificados.

```
public static DialogResult Show(
    string text,
    string caption,
    MessageBoxButtons buttons,
    MessageBoxIcon icon,
    MessageBoxDefaultButton defaultButton
);
```

### Parámetros

*text*

Texto que se va a mostrar en el cuadro de mensaje.

*caption*

Texto que se va a mostrar en la barra de título del cuadro de mensaje.

*buttons*

Uno de los objetos [MessageBoxButtons](#) que especifica qué botones se mostrarán en el cuadro de mensaje.

*icon*

Uno de los valores de [MessageBoxIcon](#) que especifica qué icono se mostrará en el cuadro de mensaje.

*defaultButton*

Uno de los valores de [MessageBoxDefaultButton](#) que especifica cuál es el botón predeterminado del cuadro de mensaje.


Los valores de Iconos, Botones salen en cuanto se llega a ese apartado del *MessageBox*. Ya los muestra el propio IDE al escribir el nombre de la lista de posibilidades (*MessageBoxButtons* o *MessageBoxIcon*). Son enumerados.

### Valor devuelto

Uno de los valores de [DialogResult](#). Para verlos, se escribe en el editor *DialogResult* y un punto y aparece la lista de posibilidades. También se pueden ver en la ayuda. Estos valores están relacionados con uno de los botones pulsado: *OK*, *Cancel*, *Yes*, *No*, ...

```
DialogResult a;
```

```
a = MessageBox.Show("Salvar archivos", "El archivo ya existe ¿Desea Continuar?",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question,
    MessageBoxDefaultButton.Button2);
```

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

## Apéndice II: Estructura .Net

### .NET Framework

Es la plataforma sobre la que se constituye todo el sistema Microsoft.NET. De cara al programador le ofrece gran cantidad de herramientas y servicios para poder realizar su labor. En el siguiente esquema, aun no siendo el más actualizado, se expone de forma clara como se establece este framework por encima del sistema operativo.

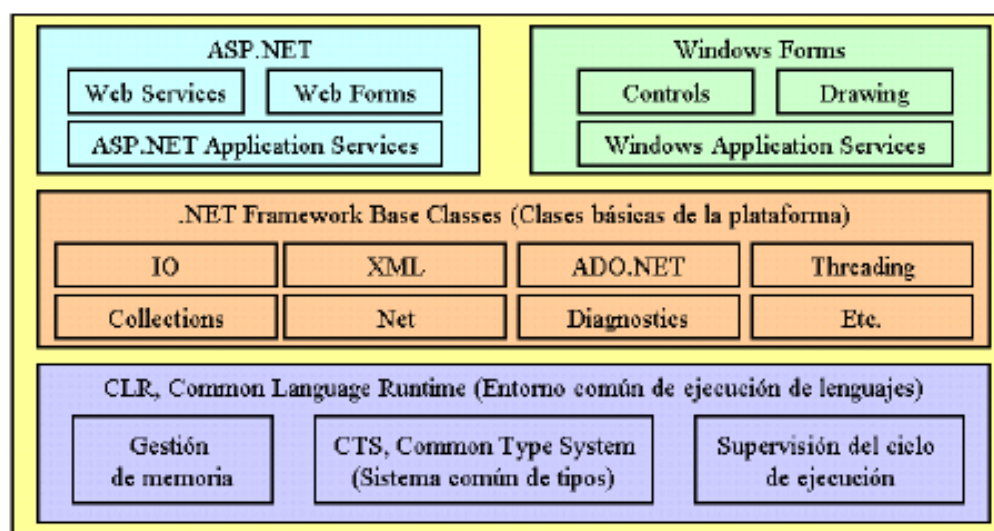


Figura 3. Esquema de componentes dentro de la plataforma .NET Framework.

**ASP.NET:** Clases necesarias para la programación de aplicaciones Web y de servicios Web


**Windows Forms:** Clases para la programación de aplicaciones en un entorno de ventanas. En principio Windows.

**Base Classes:** Clases básicas de .NET Framework. Contemplan clases genéricas para entrada salida, xml, conexión a bases de datos, threading, colecciones, redes,....

### CLR: Common Language Runtime.

Entorno común de ejecución de lenguajes. Se encarga de la ejecución del código intermedio (MSIL:MS Intermediate language) del lenguaje

Se permiten usar varios lenguajes como VB, C#, Jscript, C++ o Pascal (Delphi.Net). En concreto, los 3 primeros se ejecutan y son comprobados al mismo tiempo para evitar posibles errores. No ocurre esto con el C++, lo que provoca que pueda ser más arriesgado programar en este lenguaje pero a costa de obtener un promedio de un 10% de mejora en el rendimiento final de la aplicación.

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

Presenta un **sistema común de tipos básicos para los distintos lenguajes** .NET. Esto lo gestiona un módulo del CLR denominado CTS (Common Type System).

Esto forma parte del Common Language Specification (CLS) que es una especificación abierta que posibilita incorporar por parte de diversos fabricantes nuevos lenguajes al entorno .NET

Todos los tipos de datos van a ser realmente objetos.

El CLR además realiza una gestión eficiente de la memoria liberándola si no es necesaria. Lo que hace es que cuando detecta que la memoria reservada para una aplicación se está llenando, revisa todos los objetos que han sido liberados y libera dicha memoria y compacta el resto para mayor eficiencia.

### Compilación y ejecución

Como se comentó, una vez que compilamos aparece un ejecutable pero que realmente contiene código IL. Este se puede ver con el desensamblador que hay en Visual Studio.NET\FrameworkSDK\Bin

ILDASM

Para poder ejecutarlo se necesita el compilador JIT: Just in time. A diferencia de Java, a medida que se van necesitando partes del programa el JIT las compila a código nativo, y en sucesivas llamadas no se hace necesario volver a compilar (**compilación bajo demanda**)

|   |            |  |             |     |       |  |        |  |
|---|------------|--|-------------|-----|-------|--|--------|--|
| <div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div> | RAMA:      | Informática  | CICLO:      | DAM |       |  |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       |  | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |  |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |  |        |  |

## Apéndice III: Compilación en línea de comando

Extracto de “El lenguaje de programación C#” de José Antonio González Seco

Una vez escrito el código anterior con algún editor de textos –como el **Bloc de Notas** de Windows– y almacenado en formato de texto plano en un fichero `HolaMundo.cs` [1], para compilarlo basta abrir una ventana de consola (MS-DOS en Windows), colocarse en el directorio donde se encuentre y pasárselo como parámetro al compilador así:

```
csc HolaMundo.cs
```

**csc.exe** es el compilador de C# incluido en el .NET Framework SDK para Windows de Microsoft. Aunque en principio el programa de instalación del SDK lo añade automáticamente al path para poder llamarlo sin problemas desde cualquier directorio, si lo ha instalado a través de VS.NET esto no ocurrirá y deberá configurárselo ya sea manualmente, o bien ejecutando el fichero por lotes `Common7\Tools\vsvars32.bat` que VS.NET incluye bajo su directorio de instalación, o abriendo la ventana de consola desde el icono **Herramientas de Visual Studio.NET** → **Símbolo del sistema de Visual Studio.NET** correspondiente al grupo de programas de VS.NET en el menú Inicio de Windows que no hace más que abrir la ventana de consola y llamar automáticamente a `vsvars32.bat`. En cualquier caso, si usa otros compiladores de C# puede que varíe la forma de realizar la compilación, por lo que lo que aquí se explica en principio sólo será válido para los compiladores de C# de Microsoft para Windows.


Tras la compilación se obtendría un ejecutable llamado `HolaMundo.exe` cuya ejecución produciría la siguiente salida por la ventana de consola:

```
¡Hola Mundo!
```

Si la aplicación que se vaya a compilar no utilizase la ventana de consola para mostrar su salida sino una interfaz gráfica de ventanas, entonces habría que compilarla pasando al compilador la opción `/t` con el valor `winexe` antes del nombre del fichero a compilar. Si no se hiciese así se abriría la ventana de consola cada vez que ejecutase la aplicación de ventanas, lo que suele ser indeseable en este tipo de aplicaciones. Así, para compilar `Ventanas.cs` como ejecutable de ventanas sería conveniente escribir:

```
csc /t:winexe Ventanas.cs
```

Nótese que aunque el nombre `winexe` dé la sensación de que este valor para la opción `/t` sólo permite generar ejecutables de ventanas, en realidad lo que permite es generar ejecutables sin ventana de consola asociada. Por tanto, también puede usarse para generar ejecutables que no tengan ninguna interfaz asociada, ni de consola ni gráfica.

|   |            |  |        |     |       |        |
|---|------------|--|--------|-----|-------|--------|
|  | RAMA:      | Informática  | CICLO: | DAM |       |        |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:  |     | DATA: |        |
|   | UNIDAD     | COMPETENCIA  |        |     |       |        |

Si en lugar de un ejecutable -ya sea de consola o de ventanas- se desea obtener una librería, entonces al compilar hay que pasar al compilador la opción **/t** con el valor **library**. Por ejemplo, siguiendo con el ejemplo inicial habría que escribir:

```
csc /t:library HolaMundo.cs
```

En este caso se generaría un fichero `HolaMundo.dll` cuyos tipos de datos podrían utilizarse desde otros fuentes pasando al compilador una referencia a los mismos mediante la opción **/r**. Por ejemplo, para compilar como ejecutable un fuente `A.cs` que use la clase `HolaMundo` de la librería `HolaMundo.dll` se escribiría:

```
csc /r:HolaMundo.dll A.cs
```

En general **/r** permite referenciar a tipos definidos en cualquier ensamblado, por lo que el valor que se le indique también puede ser el nombre de un ejecutable. Además, en cada compilación es posible referenciar múltiples ensamblados ya sea incluyendo la opción **/r** una vez por cada uno o incluyendo múltiples referencias en una única opción **/r** usando comas o puntos y comas como separadores. Por ejemplo, las siguientes tres llamadas al compilador son equivalentes:

```
csc /r:HolaMundo.dll;Otro.dll;OtroMás.exe A.cs
```

```
csc /r:HolaMundo.dll,Otro.dll,OtroMás.exe A.cs
```


```
csc /t:HolaMundo.dll /r:Otro.dll /r:OtroMás.exe A.cs
```

Hay que señalar que aunque no se indique nada, en toda compilación siempre se referencia por defecto a la librería **mscorlib.dll** de la BCL, que incluye los tipos de uso más frecuente. Si se usan tipos de la BCL no incluidos en ella habrá que incluir al compilar referencias a las librerías donde estén definidos (en la documentación del SDK sobre cada tipo de la BCL puede encontrar información sobre donde se definió)

Tanto las librerías como los ejecutables son ensamblados. Para generar un módulo de código que no forme parte de ningún ensamblado sino que contenga definiciones de tipos que puedan añadirse a ensamblados que se compilen posteriormente, el valor que ha de darse al compilar a la opción **/t** es **module**. Por ejemplo:

```
csc /t:module HolaMundo.cs
```

Con la instrucción anterior se generaría un módulo llamado `HolaMundo.netmodule` que podría ser añadido a compilaciones de ensamblados incluyéndolo como valor de la opción **/addmodule**. Por ejemplo, para añadir el módulo anterior a la compilación del fuente librería `Lib.cs` como librería se escribiría:

|   |            |  |             |     |       |        |  |
|---|------------|--|-------------|-----|-------|--------|--|
|  | RAMA:      | Informática  | CICLO:      | DAM |       |        |  |
|   | MÓDULO     | Desarrollo de Interfaces/Programación de Servicios |             |     |       | CURSO: |  |
|   | PROTOCOLO: | Apuntes clases                                     | AVAL:       |     | DATA: |        |  |
|   | UNIDAD     |  | COMPETENCIA |     |       |        |  |

```
csc /t:library /addmodule:HolaMundo.netmodule Lib.cs
```

Aunque hasta ahora todas las compilaciones de ejemplo se han realizado utilizando un único fichero de código fuente, en realidad nada impide que se puedan utilizar más. Por ejemplo, para compilar los ficheros `A.cs` y `B.cs` en una librería `A.dll` se ejecutaría:

```
csc /t:library A.cs B.cs
```

Nótese que el nombre que por defecto se dé al ejecutable generado siempre es igual al del primer fuente especificado pero con la extensión propia del tipo de compilación realizada (`.exe` para ejecutables, `.dll` para librerías y `.netmodule` para módulos) Sin embargo, puede especificarse como valor en la opción `/out` del compilador cualquier otro tal y como muestra el siguiente ejemplo que compila el fichero `A.cs` como una librería de nombre `Lib.exe`:

```
csc /t:library /out:Lib.exe A.cs
```


Véase que aunque se haya dado un nombre terminado en `.exe` al fichero resultante, éste sigue siendo una librería y no un ejecutable e intentar ejecutarlo produciría un mensaje de error. Obviamente no tiene mucho sentido darle esa extensión, y sólo se le ha dado en este ejemplo para demostrar que, aunque recomendable, la extensión del fichero no tiene porqué corresponderse realmente con el tipo de fichero del que se trate.

A la hora de especificar ficheros a compilar también se pueden utilizar los caracteres de comodín típicos del sistema operativo. Por ejemplo, para compilar todos los ficheros con extensión `.cs` del directorio actual en una librería llamada `Varios.dll` se haría:

```
csc /t:library /out:varios.dll *.cs
```

Con lo que hay que tener cuidado, y en especial al compilar varios fuentes, es con que no se compilen a la vez más de un tipo de dato con punto de entrada, pues entonces el compilador no sabría cuál usar como inicio de la aplicación. Para orientarlo, puede especificarse como valor de la opción `/main` el nombre del tipo que contenga el `Main()` ha usar como punto de entrada. Así, para compilar los ficheros `A.cs` y `B.cs` en un ejecutable cuyo punto de entrada sea el definido en el tipo `Principal`, habría que escribir:

```
csc /main:Principal A.cs B.cs
```

|   |                    |  |        |     |       |        |
|---|--------------------|--|--------|-----|-------|--------|
|  | RAMA:              | Informática  | CICLO: | DAM |       |        |
|   | MÓDULO             | Desarrollo de Interfaces/Programación de Servicios |        |     |       | CURSO: |
|   | PROTOCOLO:         | Apuntes clases                                     | AVAL:  |     | DATA: |        |
|   | UNIDAD COMPETENCIA |  |        |     |       |        |

Lógicamente, para que esto funcione `A.cs` o `B.cs` tiene que contener alguna definición de algún tipo llamado Principal con un único método válido como punto de entrada (obviamente, si contiene varios se volvería a tener el problema de no saber cuál utilizar)

---

[1] El nombre que se dé al fichero puede ser cualquiera, aunque se recomienda darle la extensión `.cs` ya que es la utilizada por convenio