

C#

Pablo Orduña Fernández (aka NcTrun)

Julio 2005



DotNetGroup



This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA

Introducción al cursillo

Introducción a . . .

Introducción a C#

Introducción breve . . .

Algunas novedades . . .

Referencias

Página www

Página de Abertura



Página 1 de 60

Regresar

Full Screen

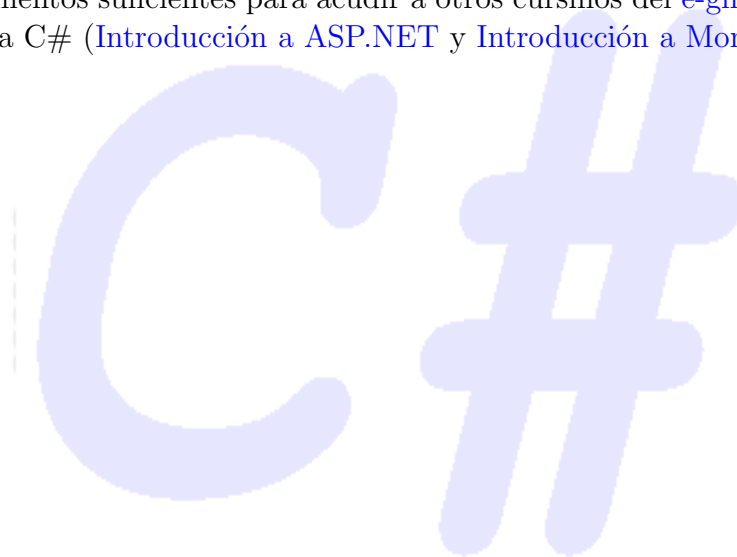
Cerrar

Abandonar

1. Introducción al cursillo

1.1. Objetivos

- Aprender a moverse con C#
- Tener conocimientos suficientes para acudir a otros cursillos del [e-ghost](#) y [DotNetGroup](#) en los que se necesita C# ([Introducción a ASP.NET](#) y [Introducción a Mono](#))

[Introducción al cursillo](#)[Introducción a ...](#)[Introducción a C#](#)[Introducción breve ...](#)[Algunas novedades ...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀◀](#) [▶▶](#)[◀](#) [▶](#)[Página 2 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

1.2. Qué se va a dar en este cursillo

- Conceptos básicos de .NET/Mono/...
- Introducción a C#
- Introducción breve a la FCL
- Novedades en C# 2.0



Introducción al cursillo

Introducción a . . .

Introducción a C#

Introducción breve . . .

Algunas novedades . . .

Referencias

Página www

Página de Abertura

◀▶

◀▶

Página 3 de 60

Regresar

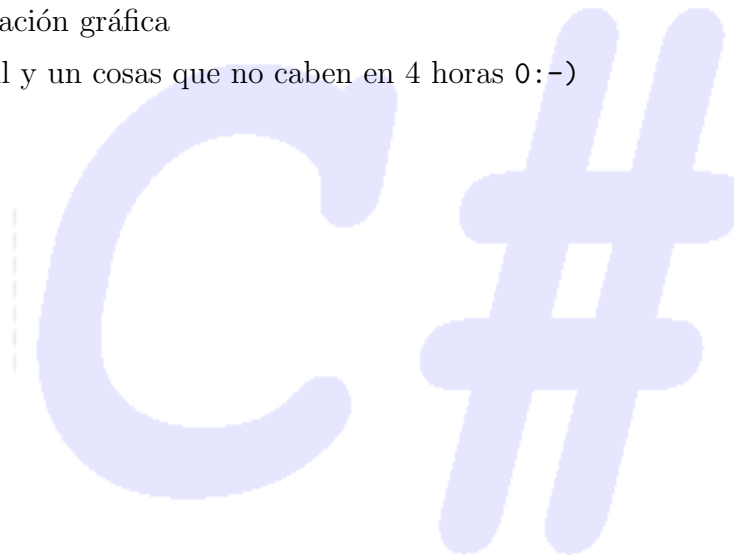
Full Screen

Cerrar

Abandonar

1.3. Qué queda en el tintero

- Un uso mínimamente detallado de la FCL, quedando fuera:
 - Hilos
 - Comunicaciones
 - Programación gráfica
 - Otras mil y un cosas que no caben en 4 horas 0:-)



Introducción al curso

Introducción a...

Introducción a C#

Introducción breve...

Algunas novedades...

Referencias

Página www

Página de Abertura

◀ ▶

◀ ▶

Página 4 de 60

Regresar

Full Screen

Cerrar

Abandonar

2. Introducción a Mono/.NET

2.1. ¿Qué es?

- El .NET Framework es una *plataforma de desarrollo de software*, enfocada en:
 - Desarrollo rápido y explotación de aplicaciones gestionadas (*managed*) y orientadas a objetos
 - Independencia del lenguaje
 - Independencia de la plataforma
 - Transparencia a través de la red
- Esta plataforma ofrece, entre otras cosas:
 - Nuevos lenguajes de programación modernos (C#, VB.NET,...)
 - Nuevos lenguajes de programación con mayor o menor compatibilidad con otros lenguajes (Managed C++, J#, ...)
 - La posibilidad de incluir nuevos lenguajes de programación
 - Integración multilenguaje, reutilización de componentes, herencia entre componentes desarrollados en diferentes lenguajes
 - Una extensa framework de librerías de clases independiente del lenguaje
 - Un sistema de ejecución de lenguaje común (CLR)
 - Un conjunto de servidores .NET
 - Nuevas formas de programación
 - * web: ASP.NET
 - * gráfica: Windows Forms

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[«](#) [»](#)[«](#) [»](#)[Página 5 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

- * de Servicios Web XML independientes de la plataforma vía SOAP y WSDL
- Conjunto de herramientas de desarrollo (Visual Studio .NET, ...)



C#

[Introducción al curso](#)

[Introducción a...](#)

[Introducción a C#](#)

[Introducción breve...](#)

[Algunas novedades...](#)

[Referencias](#)

[Página www](#)

[Página de Abertura](#)

◀▶

◀▶

Página 6 de 60

[Regresar](#)

[Full Screen](#)

[Cerrar](#)

[Abandonar](#)

2.2. Common Language Runtime

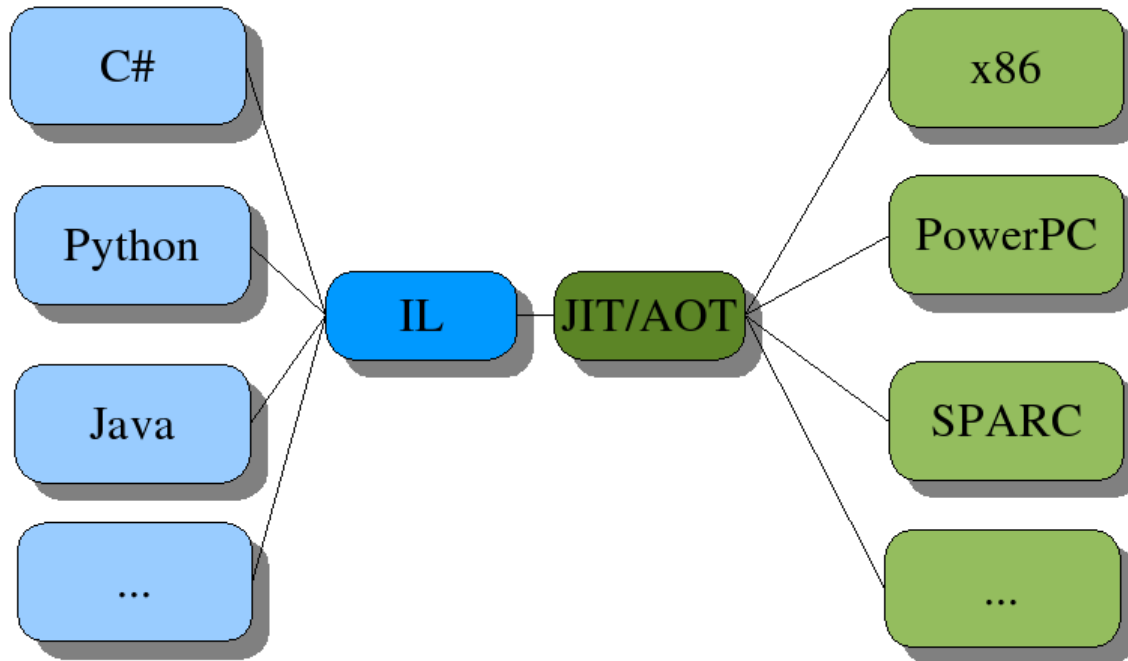
2.2.1. Características

- La CLR trae incorporadas las características de un runtime moderno:
 - Integración lenguajes
 - Garbage Collector
 - Multihilo
 - Interoperabilidad con código nativo
- El CLR es el encargado de ejecutar las aplicaciones .NET

2.2.2. Soporte multilinguaje

- La plataforma es independiente del lenguaje
- Cuenta con un lenguaje universal, el CIL
 - Common Intermediate Language, también llamado IL o MSIL
 - fácilmente compilable
 - Cada lenguaje tiene su compilador a CIL
- Luego, del CIL se genera el código nativo de la plataforma en la que se ejecute
 - compilador JIT (Just In Time) o AOT (Ahead Of Time) o intérprete
 - las diferentes implementaciones de .NET soportan diferentes plataformas

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 7 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)



- Gracias a esto:
 - Se puede desde un lenguaje utilizar componentes escritos en otro lenguaje
 - Dada una librería, su API es accesible a todos los lenguajes
 - Lo único que es necesario es que estos lenguajes tengan su compilador a CIL
 - Incluso se puede reutilizar compiladores hechos por terceros

2.2.3. Librerías

- Como hemos dicho, .NET trae una gran cantidad de librerías
- Todas estas librerías son accesibles a todo lenguaje que cuente con compilador para el CIL

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀◀](#) [▶▶](#)[◀](#) [▶](#)[Página 9 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

2.3. Estandarización de .NET

- Microsoft estandarizó parte del .NET Framework en el [ECMA](#)
 - En los estándares [ECMA 334](#) (C#) y [ECMA 335](#) (CLI: Common Language Infrastructure) Estandarizó el núcleo de .NET y C#
 - C# y la CLI además está estandarizado por la ISO en los estándares [ISO/IEC 23270](#) y [ISO/IEC 23271](#) respectivamente
- Pero otros componentes no están estandarizados:
 - ASP.NET
 - Windows Forms
 - ADO.NET
- [Más informacion](#) (págs 3 en adelante) y [referencias](#) (págs 5 y 6) 0:-)

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 10 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

2.4. Implementaciones

- Microsoft tiene lógicamente su implementación de .NET
- Pero hay más implementaciones, algunas Open Source, como:

- Mono
- DotGNU

- En el aula están instaladas:

- La implementación de Microsoft en Windows
 - * .NET 1.1
- Mono en Ubuntu, Debian y Windows
 - * Versiones 1.1.x en las tres

- La versión 1.1.x de Mono implementa parte de C# 2, que daremos en clase

Aunque todo lo que vayamos a dar sea estándar, en la versión instalada en el aula hay cosas que sólo funcionarían con Mono

3. Introducción a C#

3.1. Introducción

- Lenguaje sencillo
- Orientado a objetos
- El siguiente en lenguajes derivados de C, pasando por C++ y Java
 - Recoge muchas de las cosas que Java no había cogido de C++
 - Asimismo, incorpora varias novedades no presentes en ninguno
 - La # viene de que es C++++, donde ++ encima de ++ es #
- Permite tanto la programación **managed** como la **unsafe** (con punteros como en C++)
- Más simple que C++, pero tan poderoso y flexible

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 12 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

3.2. Comenzando con C#

3.2.1. Ficheros de código en C#

- Al igual que en C++, C# tiene:
 - uno o más ficheros (con extensión .cs)
 - el cual tiene o no espacios de nombres (namespaces)
 - que generan un único ejecutable o librería (.exe o .dll)

3.2.2. Hola mundo en C#

- Primer programa en C#:

```
class A{
    static void Main(){
        System.Console.WriteLine("hola mundo");
    }
}
```

- Hola mundo explicado:
 - class
 - * Al igual que en Java, *todo código aparecerá en métodos de clases*
 - Main
 - * El punto de entrada (**entrypoint**) del programa estará en un método especial llamado Main, que podrá ser declarado tal que:

```
static void Main();
static int Main();
static void Main(string [] args);
static int Main(string [] args);
```

- * Sólo podrá haber un Main por ensamblado ejecutable.
- * C# es *case-sensitive*.

– System.Console

- * Al ejecutar `System.Console.WriteLine("hola mundo");` estamos:
 - llamando al método `WriteLine`
 - de la clase `Console`
 - del espacio de nombres `System`

- * Podríamos sustituirlo por:
`using System;`

```
class A{
    static void Main(){
        Console.WriteLine("hola mundo");
    }
}
```

- * De manera que podemos acceder a todas las clases, espacios de nombres, delegates, interfaces, y enumeraciones que cuelgan de `System`

3.2.3. Ejecutandolo

- Primero lo compilamos:

```
nctrun@ord3p:~/dev/cs$ mcs holamundo.cs
nctrun@ord3p:~/dev/cs$
```

- Y luego lo ejecutamos:
nctrun@ord3p:~/dev/cs\$ mono holamundo.exe
hola mundo
nctrun@ord3p:~/dev/cs\$



3.3. Fundamentos de C#

3.3.1. Tipos de datos

- En C# hay dos tipos de tipos de datos:
 - por valor
 - * Cada vez que son utilizados, se copia su área de memoria
 - * Se alojan en la **stack**
 - * Destruídos en cuanto se destruye el bloque en el que son declarados acaba.
 - por referencia
 - * Cada vez que son utilizados, se copia un puntero a su posición en memoria
 - * Se alojan en la **heap**
 - * Destruídos cuando el **Garbage Collector** detecta que la última referencia al dato ha sido destruida
- C# cuenta con varios tipos de datos **built-in**:
 - `byte, char, bool, sbyte, short, ushort, int, uint, float, double, decimal, long, ulong`
 - Ejemplos:

```
bool bln = true;
byte byt1 = 22;
char ch1='x', ch2='\u0066';
decimal dec1 = 1.23M;
double dbl1=1.23, dbl2=1.23D;
short sh = 22;
int i = 22;
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 16 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

```

long lng1 =22, lng2 =22L;
sbyte sb = 22;
float f=1.23F;
ushort us1=22;
uint ui1=22, ui2=22U;
ulong ul1 =22, ul2=22U, ul3=22L, ul4=2UL;

```

– Todos estos tipos de datos son por valor

- strings

– Los strings son tipos de datos por referencia

– Los caracteres escapados son los típicos de otros lenguajes de programación:

```
\', \", \\, \0, \a, \b, \f, \n, \r, \t, \v
```

– Para evitar caracteres escapados, basta con poner una @ por delante

```

string s = "hola";
s = "hola\n";
s = @"C:\Documents and Settings\";

```

3.3.2. Variables

- Las variables se asignan como en C++ o Java:

```

class Variables{
    static void Main(){
        int numero = 5;
        System.Console.WriteLine("El número es: {0}",numero);
    }
}

```

Introducción al curso

Introducción a...

Introducción a C#

Introducción breve...

Algunas novedades...

Referencias

Página www

Página de Abertura

◀ ▶

◀ ▶

Página 17 de 60

Regresar

Full Screen

Cerrar

Abandonar

- Al igual que en Java, un dato debe estar definido antes de ser usado:

```
class Variables{
    static void Main(){
        int numero;
        System.Console.WriteLine("El número es: {0}",numero);
    }
}
```

el compilador suelta:

```
nctrun@ord3p:~/dev/cs$ mcs definitivo.cs
definitivo.cs(4) error CS0165: Use of unassigned local variable 'numero'
Compilation failed: 1 error(s), 0 warnings
```

3.3.3. Constantes

- Las constantes son declaradas al estilo C++:

```
const constante = 50;
const constante2 = constante - 2;
```

3.3.4. Enumerados

- Los enumerados de C++ se han adaptado para que sean orientados a objetos

```
class Enumerados{
    enum ViernesToca {GhostKino,Cena,Fiestas};
    static void Main(){
        ViernesToca vt = ViernesToca.GhostKino; //yeah!
    }
}
```

[Introducción al curso](#)

[Introducción a...](#)

[Introducción a C#](#)

[Introducción breve...](#)

[Algunas novedades...](#)

[Referencias](#)

[Página www](#)

[Página de Abertura](#)

◀ ▶

◀ ▶

[Página 18 de 60](#)

[Regresar](#)

[Full Screen](#)

[Cerrar](#)

[Abandonar](#)

3.3.5. Control de flujo

- Condicionales

- if...else

```
int i = 15;
if(i == 5)
    System.Console.WriteLine("...un fmono diría algo de hincos...");
else if(i == 15)
    System.Console.WriteLine("es quince");
else
    System.Console.WriteLine("mariposa");
```

- switch

```
switch(queToca){
    case ViernesToca.GhostKino:
        Console.WriteLine("ghostkino y...");
        goto case ViernesToca.Fiestas;
    case ViernesToca.Cena:
        Console.WriteLine("cena");
        break;
    case ViernesToca.Fiestas:
        Console.WriteLine("fiestas");
        break;
}
```

- En línea, al estilo C++ y Java:

```
string ropa = haceFrio?"abrigo":"camiseta";
```

- Sentencias repetitivas

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 19 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

```
- do...while
int i = 6;
do{
    Console.WriteLine("i: {0}",--i);
while(i > 0);
Console.WriteLine(":-S");

- while
while(respuesta.Incorrecta())
    respuesta = Preguntar();

- for
for(int i = 0; i < 5; ++i)
    Console.WriteLine("Va por...{0}",i);

- foreach
string [] nombres = new string[]{"nombre1","nombre2"};
foreach(string n in nombres)
    Console.WriteLine(n);

- continue y break como en el resto de lenguajes

int n = 10;
while(n-- > 0){
    if(n % 2 == 0)
        continue; //No hagas nada con los pares
    int n2 = n;
    Console.WriteLine("n2 es: {0}",n2);
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 20 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

3.3.6. Directivas de compilador

- En C#, al igual que en C++, hay directivas de compilador:

```
#define DEBUG
```

```
#if DEBUG
    //código
#elif TEST
    //otro código
#else
    //otro
#endif
```

```
#undef DEBUG
```

```
#if DEBUG
    //va a ser que no
#endif
```

```
#region Mis cosillas
```

```
    //movidillas. Los IDEs permiten agrupar varios métodos, por ejemplo, para no verlos
```

```
#endregion
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 21 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

3.4. Clases

3.4.1. Definiendo clases

- Una clase es definida tal que:

```
public class MiClase{
    //por defecto: datos privados
    int atributo1;
    int atributo2;

    public MiClase(){ //constructor
        atributo1 = atributo2 = 3;
    }

    void Metodo(){ //por defecto: privado
        //codigo
    }

    public int Atributo1{//property
        get{
            return atributo1;
        }
    }
}
```

- C# tiene mejor trabajados los modificadores de acceso que sus lenguajes predecesores:
 - `public`. Todo el mundo puede acceder
 - `private`. Sólo la propia clase puede acceder.
 - `protected`. Sólo la propia clase y derivadas pueden acceder.

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 22 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

- `internal`. Sólo accesible desde las clases que están en el mismo ensamblado.
- `protected internal`. Sólo accesible desde las clases del mismo ensamblado o desde las hijas.

3.4.2. Construyendo objetos

- Por defecto, C# provee un constructor por defecto
 simplemente pone a cero los datos numéricos, a `false` los booleanos, a `'\0'` los `char`, a `0` los `enum` y a `null` las referencias
- Para crear una instancia, basta con:

```
MiClase mc = new MiClase(parametros);
```
- Al igual que en Java, C# tiene inicializadores:

```
private int num = 15;
```
- A diferencia de C++, C# no provee de ningún *constructor copy*
 - Hay que implementarlo en caso de que se desee
 - Existe el interfaz `System.ICloneable`

```
nctrun@ord3p:~$ /opt/mono-1.1.7/bin/monop System.ICloneable
public interface ICloneable {

    object Clone ();
}
```
- La palabra reservada `this` tiene el mismo valor que en sus predecesores
- C# introduce la palabra reservada `readonly`, que hace que un atributo pueda ser unicamente modificado en el constructor de la clase

Introducción al curso

Introducción a . . .

Introducción a C#

Introducción breve . . .

Algunas novedades . . .

Referencias

Página www

Página de Abertura

◀ ▶

◀ ▶

Página 23 de 60

Regresar

Full Screen

Cerrar

Abandonar

3.4.3. Métodos

- La definición de métodos en C# es:
`modificadorAcceso (static) tipoDatoDevuelto nombreFuncion(parametros){`
`}`
- Los modificadores de acceso ya hemos comentado, y el uso y definición de `static` y retorno es igual a sus predecesores
- Sin embargo, el paso de parámetros cambia bastante

- Por defecto, el paso de parámetros es, al igual que Java, siempre por valor:

```
void Metodo(MiClase mc){
    mc.Modificar(); //sin problemas
    mc = null; //sin problemas, pero no afecta fuera de la función
}
//...
variable.Metodo(mc);
```

- Sin embargo, C# permite el paso por referencia, pero de manera diferente a C++:

- * Obliga a poner la palabra `ref` antes del parámetro en la definición del método:

```
void Metodo(ref MiClase mc){
    mc = null; //afecta fuera también
}
```

- * Obliga a poner la misma palabra `ref` antes del parámetro en la llamada al método:
`variable.Metodo(ref mc);`

- * Como con el paso por valor, para llamarlo, la variable debe estar inicializada:

```
int numero;
Metodo(ref numero); //no compila
```

[Introducción al curso](#)
[Introducción a...](#)
[Introducción a C#](#)
[Introducción breve...](#)
[Algunas novedades...](#)
[Referencias](#)
[Página www](#)
[Página de Abertura](#)
[◀](#)
[▶](#)
[◀](#)
[▶](#)
[Página 24 de 60](#)
[Regresar](#)
[Full Screen](#)
[Cerrar](#)
[Abandonar](#)

- Para permitir que una variable sea inicializada dentro de un método, se utiliza la palabra `out`:

- * Obliga a poner la palabra `out` antes del parámetro en la definición del método.

```
void Metodo(out int dato){
    dato = 0;
}
```

- * También obliga a poner la misma palabra `out` antes del parámetro en la llamada al método:

```
int numero;
Metodo(out numero);
```

- * A la hora de pasar la variable por parámetro, da lo mismo si está o no inicializada
- * A la hora de salir del método, el compilador da por hecho que puede no estar inicializada y que debe ser inicializada dentro, ateniéndose a las mismas restricciones que si no estuviese inicializada desde el principio

- Además, C# permite el paso de una lista indefinida de parámetros con la palabra `params`:

```
void Metodo(params int [] numeros){
    foreach(int n in numeros)
        Console.WriteLine(n);
}
//...
Metodo();
Metodo(1);
Metodo(1,2);
Metodo(1,2,3);
```

3.4.4. Destructores

- En la clase `Object` .NET, de la que `todo` hereda, existe el método virtual `Finalize`, que es similar al `finalize` de Java, salvo que .NET te garantiza que va a ser llamado.

[Introducción al cursillo](#)
[Introducción a...](#)
[Introducción a C#](#)
[Introducción breve...](#)
[Algunas novedades...](#)
[Referencias](#)
[Página www](#)
[Página de Abertura](#)
[«](#)
[»](#)
[«](#)
[»](#)
[Página 25 de 60](#)
[Regresar](#)
[Full Screen](#)
[Cerrar](#)
[Abandonar](#)

- En C# puedes o bien sobrecargar el método:

```
protected override void Finalize(){
    Console.WriteLine("hola");
}
```

o bien, hay un atajo heredado de C++, que es escribir:

```
~MiClase(){
    Console.WriteLine("hola");
}
```

que el compilador de C# se encargará de transformar en `Finalize`.

- En caso de que no se deba esperar hasta que el Garbage Collector llame al destructor, se implementa el método `Dispose`:

- Se implementa el interfaz `System.IDisposable`
- Se puede bien controlar con `bools` que no se repita la llamada al método `Dispose`, para evitar que lo vuelva a llamar el destructor, o bien se puede evitar que el Garbage Collector llame al método `Finalize` con:

```
System.GC.SuppressFinalize(instancia);
```

- El uso de `Dispose` permite una sintaxis más sencilla y óptima con un nuevo uso de la palabra `using`:

```
//MiClase implementa System.IDisposable
using(MiClase mc = new MiClase(parametros)){
    mc.hacerAlgo();
} //llegado aquí, se llama al Dispose y la variable deja de existir
```

[Introducción al curso](#)

[Introducción a...](#)

[Introducción a C#](#)

[Introducción breve...](#)

[Algunas novedades...](#)

[Referencias](#)

[Página www](#)

[Página de Abertura](#)

◀ ▶

◀ ▶

[Página 26 de 60](#)

[Regresar](#)

[Full Screen](#)

[Cerrar](#)

[Abandonar](#)

3.4.5. Propiedades

- C# incorpora el uso de propiedades:
 - Una *property* es un método que ejecuta un código en una clase casi como otro cualquiera, pero que desde fuera es accedido como si fuese un atributo público
 - Evita el continuo uso de *getAlgos* y *setAlgos*, haciendo más legible el código

- Pueden ser declarados de sólo lectura:

```
public int Numero{
    get{
        Console.WriteLine("Devolviendo...{0}",numero);
        return numero;
    }
}
```

```
//...
```

```
int num = variable.Numero; //guay
variable.Numero = 5; //no compila
```

- o bien de sólo escritura:

```
public int Numero{
    set{
        Console.WriteLine("Asignando...{0}",value);
        numero = value; //value es el valor pasado por parámetro
    }
}
```

```
//...
```

```
int num = variable.Numero; //no compila  
variable.Numero = 5; //guay
```

- o bien de lectura y escritura:

```
public int Numero{  
    set{  
        Console.WriteLine("Asignando...{0}",value);  
        numero = value; //value es el valor pasado por parámetro  
    }  
    get{  
        Console.WriteLine("Devolviendo...{0}",numero);  
        return numero;  
    }  
}  
  
int num = variable.Numero; //guay  
variable.Numero = 5; //guay
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 28 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

3.5. Estructuras

3.5.1. Qué son

- Los **structs** son:
 - agrupamientos lógicos de datos y métodos (al igual que las clases)
 - no permiten ni herencia ni destructores
 - Son datos por valor

3.5.2. Definición

- La definición de un **struct** es tal que:

```
public struct Punto{
    public int X;
    public int Y;
    public Punto(int x, int y){
        X = x;
        Y = y;
    }
}
```

3.5.3. Creación

- La creación de **structs** es también similar a la de instancias de clases:

```
Punto p = new Punto(0,5);
p.X = 5;
```

*el que se use la palabra **new** no significa que sea por referencia*

- Por defecto, hay un constructor sin parámetros que inicializa todo. De hecho, podría crearse un struct de esta manera:
Punto p;
p.X = 5;

No se aconseja este código (mejor crear siempre con `new`, deja el código más claro)



3.6. Herencia y polimorfismo

3.6.1. Heredando

- Al igual que Java, C# no soporta herencia múltiple
- La sintaxis similar a C++, pero sin modificadores de acceso en la herencia, al estilo Java:

```
public class Mamifero : Animal
```

- Si la clase padre tiene un constructor sin parámetros, por defecto será llamado antes de ejecutar el código del constructor de la clase hija
- Si no, será necesario llamar a base:

```
public class Mamifero : Animal{
    public Animal(string nombre) : base(nombre){
        //...
    }
}
```

- Para redefinir un método, hace falta la palabra **new**:

```
class A{
    public void Metodo(){
        Console.WriteLine("A");
    }
}
class B : A{
    public new void Metodo(){
```

[Introducción al curso](#)
[Introducción a...](#)
[Introducción a C#](#)
[Introducción breve...](#)
[Algunas novedades...](#)
[Referencias](#)
[Página www](#)
[Página de Abertura](#)
[◀](#)
[▶](#)
[◀](#)
[▶](#)
[Página 31 de 60](#)
[Regresar](#)
[Full Screen](#)
[Cerrar](#)
[Abandonar](#)

```

        Console.WriteLine("B");
    }
}
class Test{
    public static void Main(){
        B b = new B();
        b.Metodo(); //mostrará B
        A a = b;
        a.Metodo(); //mostrará A
    }
}

```

- Para evitar que alguien pueda heredar de una clase determinada se utiliza la palabra `sealed` (similar al final de Java):

```

sealed class A{
//...
}
class B : A{ //no compila
}

```

3.6.2. Polimorfismo

- Para escribir métodos polimórficos hace falta las palabras `virtual` y `override`, al estilo C++:

```

class A{
    public virtual void Metodo(){
        Console.WriteLine("A");
    }
}

```

[Introducción al curso](#)
[Introducción a...](#)
[Introducción a C#](#)
[Introducción breve...](#)
[Algunas novedades...](#)
[Referencias](#)
[Página www](#)
[Página de Abertura](#)
[◀](#)
[▶](#)
[◀](#)
[▶](#)
[Página 32 de 60](#)
[Regresar](#)
[Full Screen](#)
[Cerrar](#)
[Abandonar](#)

```
class B : A{
    public override void Metodo(){
        Console.WriteLine("B");
    }
}
class Test{
    public static void Main(){
        B b = new B();
        b.Metodo(); //mostrará B
        A a = b;
        a.Metodo(); //mostrará B también :)
        A [] varios = new A[2];
        varios[0] = new B();
        varios[1] = new A();
        foreach(A nueva in varios)
            nueva.Metodo();
    }
}
```

3.6.3. Clases abstractas

- Una clase abstracta:
 - no puede ser instanciada
 - puede declarar una serie de métodos que obliga a toda hija clase no abstracta a definir
- Definiendo clases abstractas:
 - Se definen con la palabra reservada **abstract**:

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀▶](#)[◀▶](#)[Página 33 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

```
abstract public class A{
    protected int dato;
}
abstract public class B : A{
    abstract public void Metodo();
}
public class C : B{
    public override void Metodo(){
        Console.WriteLine();
    }
}
```

3.6.4. Relacionando tipos de datos por valor y por referencia

- **Todo** puede ser un object:
 - Incluso datos por valor

```
string s = 5.ToString();
```
- A este proceso se le llama *boxing* y al opuesto (pasar de un object a int *unboxing*).
- El proceso de *unboxing* debe ser lógicamente explícito:

```
object o = 5;
int i = (int)o;
```

3.6.5. Sobrecarga de operadores

- Al igual que en C++, C# puede sobrecargar operadores

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 34 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

- Sobrecargando un operador:

```
class A : B{  
    public static A operator+(A a, A b){  
        return a.X + b.X;  
    }  
}
```



3.7. Interfaces

3.7.1. Qué son

- Como hemos comentado antes, C#, al igual que Java, no tiene herencia múltiple
- Sin embargo, una clase sí puede *implementar* varios interfaces.
- El interfaz define varios métodos, que la clase que lo implemente deberá declarar como públicos
- Al igual que en Java, un interfaz puede heredar de otro

3.7.2. Definiendo interfaces

- Se definen tal que:

```
interface IDisposable{
    void Dispose();
}
interface IDestroyable : IDisposable{
    void Destroy();
    int Status{
        get;
    }
}
```

3.7.3. Is, as

- Para saber si una instancia implementa un interfaz o hereda de otra clase, se puede utilizar:
 - `is`
 - * Devuelve un `bool` informando de si es o no

– as

* Devuelve `null` si **no** lo implementa, o la instancia como otra cosa en caso de que sea



C#

Introducción al curso

Introducción a . . .

Introducción a C#

Introducción breve . . .

Algunas novedades . . .

Referencias

Página www

Página de Abertura

◀▶

◀▶

Página 37 de 60

Regresar

Full Screen

Cerrar

Abandonar

3.8. Manejo de excepciones

3.8.1. Capturando excepciones

- Para capturar excepciones se utiliza `catch`:

```
try{
    int i = 10/0;
}catch(System.DivideByZeroException){
    Console.WriteLine("capturada");
}
```

- Si queremos hacer algo con la excepción, podemos capturar la instancia de la siguiente manera:

```
try{
    int i = 10/0;
}catch(System.DivideByZeroException dbze){
    Console.WriteLine(dbze.StackTrace)
}
```

- Si en cambio queremos capturar todas las excepciones, podemos hacer:

```
try{
    int i = 10/0;
}catch{
    Console.WriteLine("capturada");
}
```

- También podemos utilizar `finally` para que, tanto si se eleva una excepción como si no, se ejecute un código:

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 38 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

```
try{
    int i = 10/0;
}catch(System.DivideByZeroException dbze){
    Console.WriteLine("y aquí me he quedado :-");
}finally{
    Console.WriteLine("Esto SIEMPRE se ejecuta");
}
```

3.8.2. Lanzando excepciones

- Para lanzar una excepción, se utiliza la palabra reservada `throw`:

```
throw new Exception("excepción");
```

- Lo lanzado debe ser una instancia de una clase hija de `System.Exception` (o de `System.Exception`)
Se recomienda que las excepciones propias sean hijas de `System.ApplicationException` para diferenciar entre las excepciones del `Framework` y las de nuestra aplicación

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 39 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

3.9. Delegates

3.9.1. Qué son

- Son el equivalente a puntero a función de otros lenguajes
 - adaptados al modelo orientado a objetos
 - fáciles de usar

3.9.2. Definiendo delegates

- Un delegate se indica tal que:

```
delegate tipoDato funcion(parametros);
```

- Ejemplo:

```
delegate void Funcion(object o);
```

```
class A{  
    static void Main(){  
    }  
}
```

3.9.3. Utilizando delegates

- El uso es tan simple como:

```
delegate void Funcion();
```

```
class A{
```

```
void Hacer(Funcion f){
    f();
}

void HazAlgo(){
    Hacer(new Funcion(Saludar));
}

void Saludar(){
    System.Console.WriteLine("hola");
}

static void Main(){
    (new A()).HazAlgo();
}
}
```

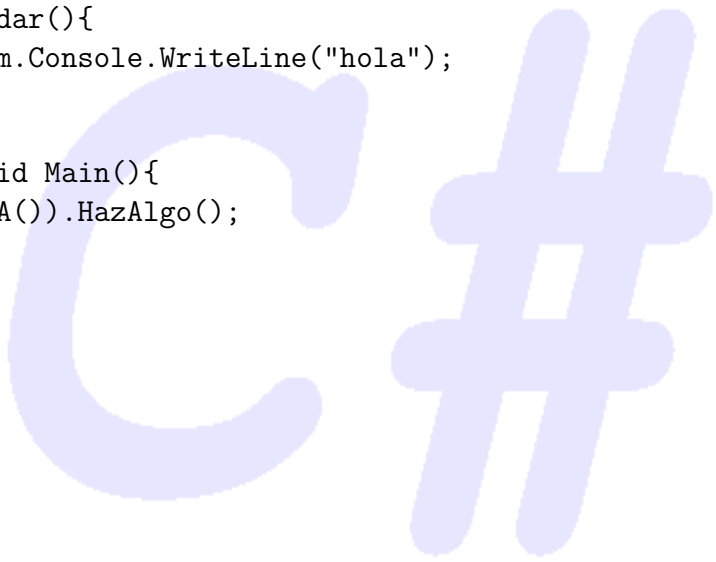
3.9.4. Utilizando multicasting

- Para que al llamar a un delegado, se llamen a varios métodos, sólo hay que ir añadiendo nuevos métodos con +=:

```
delegate void Funcion();
class A{
    void Hacer(Funcion f){
        f();
    }

    void HazAlgo(){
        Hacer(new Funcion(Saludar));
    }
}
```

```
    Funcion f = new Funcion(Saludar);  
    f += new Funcion(Saludar);  
    f += new Funcion(Saludar);  
    f();  
}  
  
void Saludar(){  
    System.Console.WriteLine("hola");  
}  
  
static void Main(){  
    (new A()).HazAlgo();  
}  
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 42 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

3.10. Atributos

3.10.1. Qué son

- Una forma de dar más información acerca de una clase, enumerado, atributo, método...
- Una explicación mucho más clara y profunda se puede encontrar [aquí](#)

3.10.2. Utilizando atributos

- Se pone el atributo entre corchetes antes de lo que se vaya a explicar
- Ejemplo:

```
using System;
using System.Diagnostics;

class A{
    static void Main(){
        Console.WriteLine("hola");
        Ejecuta();
        Console.WriteLine("adios");
    }
    [Conditional("EJECUTABLE")]
    static void Ejecuta(){
        Console.WriteLine("ejecutando");
    }
}
```

4. Introducción breve a la FCL

4.1. Qué es la FCL?

- Es la API de .NET
- La mayoría de clases y espacios de nombres cuelgan del espacio de nombres **System**
- Está disponible su documentación en: [esta web](#)
- Ejecutando mondoc también podemos acceder a la documentación
- En la web del [DotNetGroup](#) hay una sección **Documentación** en la que hay documentación de talleres que ha habido este curso en la que explica el uso de la FCL para comunicaciones con sockets, multihilo, etc. etc.

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 44 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

4.2. Clases básicas

- Echando un ojo al espacio de nombres `System`, podemos ver clases que ya conocemos realmente como:
 - `Console`: A la que ya estamos acostumbrados :-)
 - `Int32`, `Byte`, `String`: con todos los métodos y variables de todo `int`, `byte` o `string`, respectivamente
 - `Array`: Que es lo que hay cuando utilizamos arrays
- También podemos ver la documentación de las excepciones más frecuentes:
`ArithmeticException`, `DivideByZeroException`...
- Además podemos ver los espacios de nombres que hay definidos dentro de `System`, como por ejemplo:
 - `System.Collections`: colecciones
 - `System.IO`: manejo de entrada-salida
 - `System.Thread`: manejo de hilos
 - `System.Net`: redes
 - `System.Reflection`: reflection
 - `System.GC`: manejo del Garbage Collector
 - `System.XML`: procesamiento de XML
- y muchos muchos más

4.3. Colecciones

- En `System.Collections` hay una serie de clases e interfaces para usar pilas, colas, mapas...:
- Esto es, vienen definidas una serie de clases listas para ser utilizadas, como por ejemplo:
 - `ArrayList`: Estilo vector o `Vector`. Permite el ir añadiendo, eliminando y modificando elementos a la lista
 - `Hashtable`: Array asociativo. Permite añadir variables indexadas por otras variables
 - `Stack`, `Queue`: pilas y colas
 - `SortedList`: Listas ordenadas
- Así como una serie de interfaces que implementan estas clases, sus valores, etc.

`ICollection`, `ICollection`, `IDictionary`, `IComparer`...

```
using System;
using System.Collections;

class A{
    public static void Main(){
        ArrayList al = new ArrayList();
        string s;
        do{
            Console.WriteLine("Dame un algo (salir para salir): ");
            s = Console.ReadLine();
            al.Add(s);
        }while(s.ToLower() != "salir");
        string [] todasLasFrasas = (string[])al.ToArray(typeof(string));
        foreach(string frase in todasLasFrasas)
```

Introducción al curso

Introducción a...

Introducción a C#

Introducción breve...

Algunas novedades...

Referencias

Página www

Página de Abertura

◀ ▶

◀ ▶

Página 46 de 60

Regresar

Full Screen

Cerrar

Abandonar

```
}  
    Console.WriteLine(frase);  
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 47 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

4.4. Ficheros

- Ejemplo:

```
using System;
using System.IO;

class A{
    static void Main(){
        string nombreFich = Console.ReadLine();
        StreamReader fichero = null;
        try{
            fichero = new StreamReader(nombreFich);
            string linea = null;
            do{
                linea = fichero.ReadLine();
                if(linea != null)
                    Console.WriteLine(linea);
            }while(linea != null);
        }catch(IOException ioe){
            Console.WriteLine("maaaal: {0}",ioe);
        }finally{
            if(fichero != null)
                fichero.Close();
        }
    }
}
```

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 48 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

4.5. Hilos

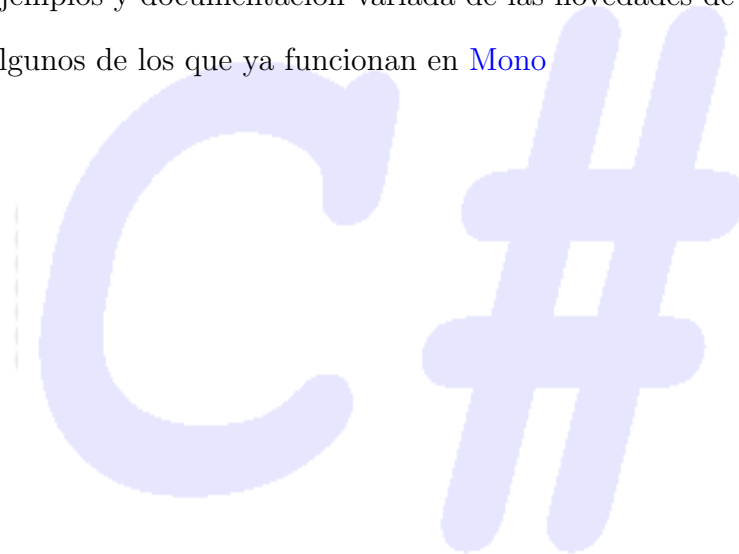
Aquí vamos a utilizar los [apuntes de hilos](#) que hay en la sección Documentación del DotNetGroup :)



5. Algunas novedades en C# 2.0

5.1. C# 2.0

- Actualmente la implementación de C# de Microsoft está en Beta
- Pero ya hay ejemplos y documentación variada de las novedades de C# 2.0, por ejemplo [este](#)
- Aquí vemos algunos de los que ya funcionan en [Mono](#)



[Introducción al curso](#)

[Introducción a ...](#)

[Introducción a C#](#)

[Introducción breve ...](#)

[Algunas novedades ...](#)

[Referencias](#)

[Página www](#)

[Página de Abertura](#)

◀▶

◀▶

Página 50 de 60

[Regresar](#)

[Full Screen](#)

[Cerrar](#)

[Abandonar](#)

5.2. Novedades

5.2.1. Iteradores

- Se introduce la palabra reservada `yield`, para creación de iteradores:

```
int [] aulas = {112,113,114};
System.Collections.IEnumerator GetEnumerator(){
    foreach(int aula in aulas)
        yield return aula;
}
```

- Cada vez que llamemos a `MoveNext`, volverá a la función y la ejecutará hasta el siguiente `yield`:

```
using System;
using System.Collections;

public class A{
    static int [] aulas = {113,114,115};

    static IEnumerator GetEnumerator(){
        Console.WriteLine("Entro en GetEnumerator");
        foreach(int aula in aulas){
            Console.WriteLine("una iteración en el foreach");
            yield return aula;
            Console.WriteLine("termina la iteración en el foreach");
        }
        Console.WriteLine("Salgo de GetEnumerator");
    }
}
```

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 51 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

```
public static void Main(){
    IEnumerator ie = GetEnumerator();
    while(ie.MoveNext())
        Console.WriteLine("Viendo siguiente {0}",ie.Current);
}
}
```

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[Página 52 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

5.2.2. Clases estáticas

- Una `static class` simplemente es una clase `sealed` que exige que todos sus atributos y métodos sean estáticos, y que no puede ser instanciada.

```
using System;

static class A{
    static int dato;

    static void Main(){
        Console.WriteLine("hola mundo");
    }
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 53 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

5.2.3. Métodos anónimos

- Un método anónimo es un delegado que es definido en línea.
- Tiene acceso a variables locales del método donde es definido.

```
using System;

delegate void Funcion();

class A{
    static void Main(){
        int numero = 0;
        Funcion f = delegate{
            ++numero;
        };
        for(int i = 0; i < 5; ++i)
            f();
        Console.WriteLine(numero);
    }
}
```

[Introducción al cursillo](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 54 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

5.2.4. Clases parciales

- Permiten la definición de una clase en bloques separados:

```
public partial class A{
    int numero;
    public A(){
        numero = 5;
    }
}

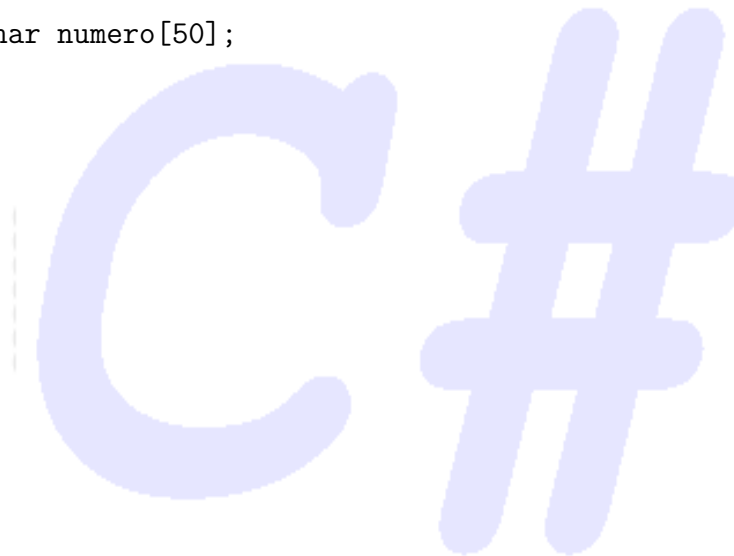
public partial class A{
    public void Mostrar(){
        System.Console.WriteLine(numero);
    }
    public static void Main(){
        new A().Mostrar();
    }
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 55 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

5.2.5. Buffers de tamaño fijo

- Permite definir el tamaño de un array en tiempo de compilación
- Debe declararse con la palabra reservada `fixed`, y ser en un `struct`:

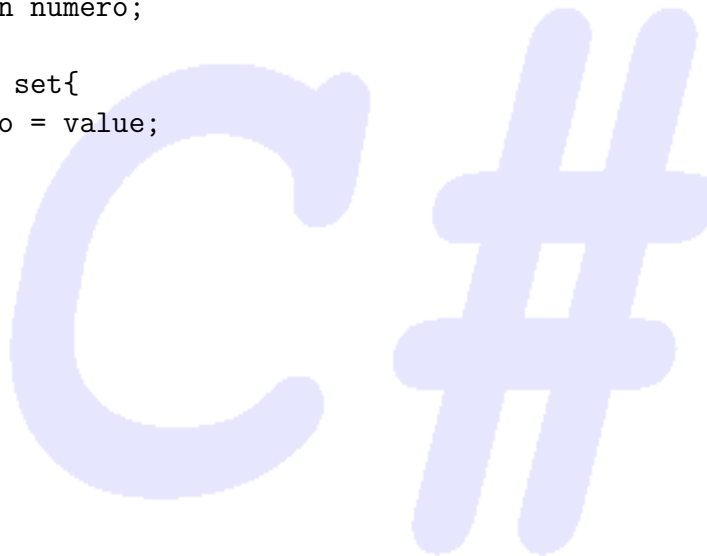
```
struct B{  
    fixed char numero[50];  
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 56 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

5.2.6. Nuevos niveles de accesibilidad

- Permite asignar diferentes niveles de accesibilidad a un get y un set en una propiedad

```
public int Numero{  
    get{//public  
        return numero;  
    }  
    protected set{  
        numero = value;  
    }  
}
```

[Introducción al curso](#)[Introducción a...](#)[Introducción a C#](#)[Introducción breve...](#)[Algunas novedades...](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 57 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

5.2.7. pragma

- Permite habilitar y deshabilitar warnings:
- Tenemos esto:

```
class A{
    static void Main(){
        int i;
    }
}
```

- Al compilarlo, nos da un warning:

```
nctrun@ord3p:~/dev/cs/cursilloc$ mcs pragma.cs
pragma.cs(3) warning CS0168: The variable 'i' is declared but never used
Compilation succeeded - 1 warning(s)
```

- Para eliminar los warnings 0168:

```
#pragma warning disable 0168
class A{
    static void Main(){
        int i;
    }
}
```

- Ahora no nos da ese warning:

```
nctrun@ord3p:~/dev/cs/cursilloc$ mcs pragma.cs
nctrun@ord3p:~/dev/cs/cursilloc$
```

6. Referencias

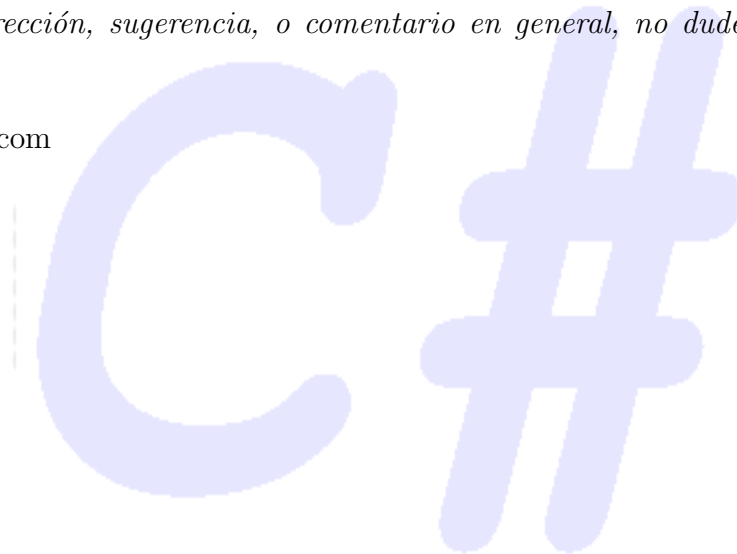
- Toda la documentación que hay en [MSDN](#)
- Más documentación que hay en el [DotNetGroup](#) de la [Universidad de Deusto](#), incluyendo su documentación y su [Biblioteca](#)
- Los [apuntes de .NET](#) de [Diego López de Ipiña](#)
- Los [tutoriales](#) de [monohispano](#)
- Un [wikilibro](#) de C#
- Otro [manual](#) de C#
- [Programming C#](#), de la editorial O'Reilly
- [Mono: A Developer's Notebook](#), de la editorial O'Reilly

[Introducción al cursillo](#)[Introducción a . . .](#)[Introducción a C#](#)[Introducción breve . . .](#)[Algunas novedades . . .](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀◀](#) [▶▶](#)[◀](#) [▶](#)[Página 59 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)

Este documento está escrito por [Pablo Orduña](#) para el cursillo de Introducción a C# del grupo de .NET de la Universidad de Deusto, el [DotNetGroup](#). Puede encontrarse junto con los ejemplos y las fuentes \LaTeX en la misma web. Probablemente las actualizaciones, dado que todas las herramientas utilizadas en el cursillo son [software libre](#), las cuelgue en [mi hueco web](#) en el grupo de software libre, el [e-ghost](#).

Para cualquier corrección, sugerencia, o comentario en general, no dudes en ponerte en contacto conmigo en:

pablo @ ordunya . com

[Introducción al cursillo](#)[Introducción a . . .](#)[Introducción a C#](#)[Introducción breve . . .](#)[Algunas novedades . . .](#)[Referencias](#)[Página www](#)[Página de Abertura](#)[◀](#) [▶](#)[◀](#) [▶](#)[Página 60 de 60](#)[Regresar](#)[Full Screen](#)[Cerrar](#)[Abandonar](#)