

# Programación Web .NET Core

Módulo 1

# Visibilidad y Sobreescritura

# Modificadores de visibilidad

Los modificadores de visibilidad permiten **especificar “desde dónde” se puede usar un tipo o un miembro del tipo.**

De esta forma, se puede gestionar la **encapsulación de los objetos** en las aplicaciones:

- Una clase definida en un **namespace** puede ser **public** o **internal**:
  - **public**: la clase se podrá utilizar por todos.
  - **internal**: la clase solo se podrá utilizar por los componentes del assembly que la contengan (un assembly es básicamente una dll que está compilada a código intermedio).
- Los miembros de una clase pueden ser **public**, **private**, **protected**, **internal** o **protected internal**.

Modificador de visibilidad	Un miembro del tipo T (por ejemplo T es una clase) definido en el assembly A es accesible...
public	desde cualquier lugar
private (por omisión)	sólo desde dentro de T (por omisión)
protected	desde T y los tipos derivados de T (es decir visible desde una clase y sus clases derivadas)
internal	desde los tipos incluidos en A (es decir visible desde todas las clases del mismo assembly)
protectedinternal	desde T, los tipos derivados de T y los tipos incluidos en A (es decir visible desde la clase, desde sus clases derivadas y desde todas las clases del mismo assembly)

# Herencia

Se hizo una breve introducción sobre el concepto de **herencia en POO**. Es momento de profundizar ese concepto.

El mecanismo de **herencia** es uno de los pilares fundamentales en los que se basa la programación orientada a objetos. Es un mecanismo que **permite definir nuevas clases a partir de otras ya definidas**. Si en la definición de una clase indicamos que ésta deriva de otra, entonces la primera -a la que se le suele llamar **clase hija** o **clase derivada**- será tratada por el compilador automáticamente como si su definición incluyese

la definición de la segunda -a la que se le suele llamar **clase padre** o **clase base**. Las clases que derivan de otras se definen con la sintaxis:

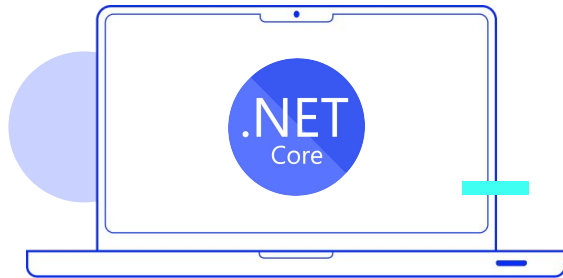
```
class<claseHija> :<clasePadre>
{
    <miembrosHija>
}
```



A los miembros definidos en la clase hija se le añadirán automáticamente los que se definieron en la clase padre: la clase derivada "hereda" de la clase base.

La palabra clave **base** se utiliza para obtener acceso a los miembros de la clase base desde una clase derivada.

**C# sólo permite herencia simple.**



## Redefinición (Sobreescritura) de métodos

Siempre que se redefina un método que existe en la clase base, hay que utilizar explícitamente la palabra reservada **override** y, de esta forma, se evitan redefiniciones accidentales (una fuente de errores en lenguajes como Java o C++).

Si en la clase **Point** se agrega el método llamado **ToString**, es decir se hereda de **System.Object** y se redefine, como se muestra a la derecha:

```
public class Point
{
    public override string ToString()
    { return( "[" + this.X + ", " + this.Y + "]" ) ;
      } ...
}
```



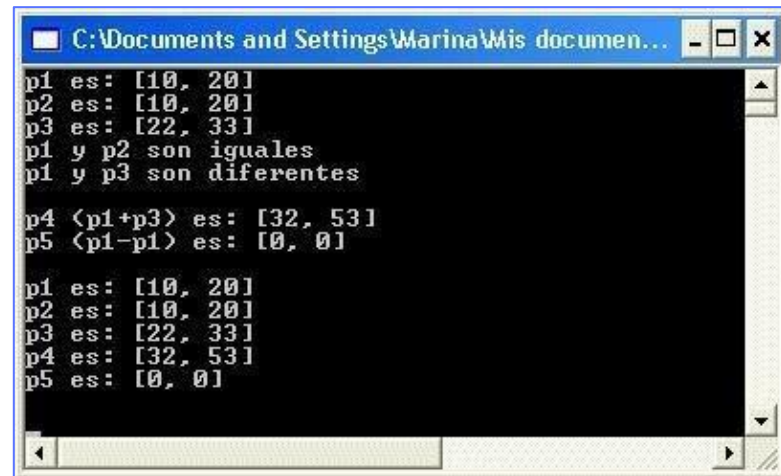
y se sustituyen las instrucciones del tipo:

```
Console.WriteLine("p1 es: ({0},{1})", p1.X, p1.Y);
```

por:

```
Console.WriteLine("p1 es: " + p1.ToString());
```

El resultado de la ejecución es:



```
C:\Documents and Settings\Marina\Mis document...  
p1 es: [10, 20]  
p2 es: [10, 20]  
p3 es: [22, 33]  
p1 y p2 son iguales  
p1 y p3 son diferentes  
  
p4 <p1+p3> es: [32, 53]  
p5 <p1-p1> es: [0, 0]  
  
p1 es: [10, 20]  
p2 es: [10, 20]  
p3 es: [22, 33]  
p4 es: [32, 53]  
p5 es: [0, 0]
```



# Métodos virtuales

Un método es **virtual** si puede redefinirse en una clase derivada. **Los métodos son no virtuales por omisión.**

- Los **métodos no virtuales** no son polimórficos (no pueden reemplazarse) ni pueden ser abstractos.
- Los **métodos virtuales** se definen en una clase base (usando la palabra reservada **virtual**) y pueden ser reemplazados (empleando la palabra reservada **override**) en las subclases (éstas proporcionan su propia -específica- implementación).

Generalmente, contendrán una implementación por omisión del método (si no, se deberían utilizar *métodos abstractos*).



**¡Sigamos  
trabajando!**