

# Introducción a C#

## 4.2.- DEFINIENDO CLASES

- Definición de clases → palabra clave **class**

```
class MyClass  
{  
    // Class members.  
}
```

```
internal class MyClass  
{  
    // Class members.  
}
```

```
public class MyClass  
{  
    // Class members.  
}
```

- Por defecto las clases son **internal**
  - Únicamente el código del proyecto (unidad de compilación) actual puede instanciar una clase
  - Es equivalente a no poner nada.
- Otras palabras claves de accesibilidad : **public**
  - Puede ser instanciado desde cualquier proyecto (unidad de compilación)

# Introducción a C#

- Además las clases pueden ser **abstract**
  - No pueden ser instanciadas, únicamente heredadas
  - Deben tener como mínimo un miembro abstract
- **sealed**
  - No pueden ser heredadas

abstract o sealed son mutuamente excluyentes

- **static** : Una clase static es la que tiene todos sus miembros definidos como static

```
public abstract class MyClass
{
    // Class members, may be abstract.
}
```

```
public sealed class MyClass
{
    // Class members.
}
```

# Introducción a C#

## · HERENCIA

- Para especificar la herencia entre clases se utilizan los dos puntos ( : )

```
public class MyClass : MyBase
{
    // Class members.
}
```

- Únicamente se permite una clase como clase base
- Si se hereda de una clase abstract se deben implementar todos los métodos abstractos, salvo que la clase derivada sea abstracta
- El compilador no permite tener una clase derivada con mayor accesibilidad que la clase base
- Si no se especifica la clase base, entonces la clase hereda directamente de la clase System.Object. Todas las clases en C# tienen la clase Object como la raíz de toda la jerarquía de herencia

# Introducción a C#

- También se puede heredar de ( implementar ) un interfaz :

```
public class MyClass : IMyInterface
{
    // Class members.
}
```

- En este caso todos los métodos del interfaz deben ser implementados, aunque sea con un método vacío
- Si se hereda de interfaces y de clase, la clase debe estar situada en primer lugar, seguidos de los interfaces separados por comas, siendo posibles múltiples interfaces

```
public class MyClass : MyBase, IMyInterface, IMySecondInterface
{
    // Class members.
}
```

- Las clases abstractas también pueden implementar interfaces

# Introducción a C#

## DEFINICIÓN DE INTERFACES

- Los interfaces se definen de la misma forma que las clases, pero con la palabra clave **interface**

```
interface IMyInterface
{
    // Interface members.
}
```

Los modificadores de acceso son : **public e internal**

- La herencia múltiple es posible en los interfaces :

```
public interface IMyInterface : IMyBaseInterface, IMyBaseInterface2
{
    // Interface members.
}
```

Los interfaces no son clases por lo que no heredan de Object

# Introducción a C#

## SYSTEM.OBJECT

- Todas las clases heredan en última instancia de System.Object (ver ayuda de la clase System.Object)

*Ejemplo : método GetType() devuelve un objeto System.Type y operador typeof ( operador C# que convierte un nombre de clase en un objeto System.Type) (polimorfismo)*

*Ejemplo : método ToString() nos devuelve la representación tipo String de un objeto, suele ser útil sobrescribirlo*

*Otros métodos a tener en cuenta .*

- Equals → Comparar objetos
- ReferenceEquals → Comprueba si son referencias a la misma instancia
- MemberwiseClone() (se ve luego)
- GetHashCode → Utilizada como función hash del objeto. Devuelve un valor que identifica el estado del objeto de forma única

# Introducción a C#

## CONSTRUCTORES y DESTRUCTORES

- Cuando se define una clase no es necesario ni definir un constructor ni un destructor, el compilador lo suministra de forma por defecto.
- Constructor por defecto → método que tiene el mismo nombre que la clase y sin parámetros

Puede ser **public** (por defecto) o **private**

```
class MyClass
{
    public MyClass()
    {
        // Constructor code.
    }
}
```

```
class MyClass
{
    private MyClass()
    {
        // Constructor code.
    }
}
```

# Introducción a C#

- Si el constructor es **private**, una instancia de esa clase no puede ser instanciada utilizando ese constructor; si no se suministra un constructor distinto la clase sería no instanciable.

Los constructores que no son el de por defecto tienen el mismo nombre, pero distintos parámetros.

- Se pueden suministrar un número ilimitado de constructores
- Si se suministra un constructor que no sea el de por defecto y no se especifica explícitamente el constructor por defecto, éste ya no se puede utilizar

Nota : La tarea de reservar y situar memoria para el objeto se realiza por defecto, el constructor se suele utilizar para inicializar propiedades y campos.



# Introducción a C#

- El destructor se declara con una sintaxis distinta :

```
class MyClass
{
    ~MyClass()
    {
        // Destructor body.
    }
}
```

- El código del destructor se ejecuta cuando el recolector de basura (garbage collection) tiene lugar.
- Después que el destructor de la clase ha sido llamado, se llaman a los destructores de las clases de las que hereda, acabando en última instancia en el destructor de la clase Object.

# Introducción a C#

## SECUENCIA DE EJECUCIÓN DE CONSTRUCTORES

- Para que una clase derivada sea instanciada, todos los constructores desde Object hasta el de la clase derivada son llamados
- Independientemente del constructor de la clase derivada que sea llamado, se ejecutan los constructores por defecto de las clases base

```
public class MyBaseClass
{
    public MyBaseClass()
    {
    }
    public MyBaseClass(int i)
    {
    }
}
public class MyDerivedClass : MyBaseClass
{
    public MyDerivedClass()
    {
    }
    public MyDerivedClass(int i)
    {
    }
    public MyDerivedClass(int i, int j)
    {
    }
}
```

# Introducción a C#

- Si ejecutamos : `MyDerivedClass myObj = new MyDerivedClass();`  
Se ejecutan los siguientes constructores :  
`System.Object.Object() → MyBaseClass.MyBaseClass() → MyDerivedClass.MyDerivedClass()`
- Si ejecutamos : `MyDerivedClass myObj = new MyDerivedClass(4);`  
`System.Object.Object() → MyBaseClass.MyBaseClass() → MyDerivedClass.MyDerivedClass(int i)`
- Si ejecutamos : `MyDerivedClass myObj = new MyDerivedClass(4,5);`  
`System.Object.Object() → MyBaseClass.MyBaseClass() → MyDerivedClass.MyDerivedClass(int i, int j)`

Si queremos que se ejecute un constructor que no sea el de por defecto de las clases bases deberemos utilizar la palabra clave **base** de la siguiente forma :

# Introducción a C#

```
public class MyDerivedClass : MyBaseClass
{
    ...
    public MyDerivedClass(int i, int j) : base(i)
    {
    }
}
```

- La palabra clave **base** dirige el proceso de instanciación .NET para utilizar el constructor de la clase base que tiene dichos parámetros ( se pueden utilizar valores literales)

Se puede utilizar también la palabra clave **this**

→ se utiliza para dirigir el proceso de instanciación .NET para una clase para utilizar un constructor que no sea el de por defecto antes de que el constructor para el que se definió se ejecute

# Introducción a C#

```
public class MyDerivedClass : MyBaseClass
{
    public MyDerivedClass() : this(5, 6)
    {
    }
    ...
    public MyDerivedClass(int i, int j) : base(i)
    {
    }
}
```

- `base(...)` y `this(...)` se denominan “inicializador de constructor” (constructor initializer)

- Para un constructor, únicamente se puede utilizar un “inicializador de constructor” (constructor initializer)