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)

- Además las clases pueden ser **abstract**, **sealed** o **static**

Introducción a Visual C#

CLASES ABSTRACTAS

- Una clase abstracta es una clase que tenga algún miembro **abstract**

- Una clase abstracta no se puede instanciar, pero sí puede servir como clase base para heredar

- Los miembros abstractos no contienen ninguna implementación, debiendo proporcionar en la clase hija

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

CLASES SEALED

- Una clase sealed es aquella que no se permite heredar de ella.

- abstract o sealed son mutuamente excluyentes

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

CLASES STATIC

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

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 raiz 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#

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**

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.

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

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

Introducción a C#

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.

- 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, salvo que especifiquemos lo contrario

· 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 :

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

Introducción a C#

· 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** para el mismo fin

```
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 “