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 actual puede instanciar una clase

· Otras palabras claves de accesibilidad : public
  · Puede ser accedido desde cualquier parte

Introducción a C#

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

- O sealed
  - No pueden ser heredadas

abstract o sealed son mutuamente excluyentes

```
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 :

```
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#

TABLE 9-1: Access Modifiers for Class Definitions

MODIFIER	DESCRIPTION
none or `internal`	Class is accessible only from within the current project
`public`	Class is accessible from anywhere
`abstract` or `internal abstract`	Class is accessible only from within the current project, and cannot be instantiated, only derived from
`public abstract`	Class is accessible from anywhere, and cannot be instantiated, only derived from
`sealed` or `internal sealed`	Class is accessible only from within the current project, and cannot be derived from, only instantiated
`public sealed`	Class is accessible from anywhere, and cannot be derived from, only instantiated

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

Ejemplo 1

Introducción a C#

SYSTEM.OBJECT

- Todas las clases heredan en última instancia de System.Object (ver ayuda de la clase 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)*
- *GetHasCode → 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 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 (no por defecto) la clase sería no instanciable.

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

· Se pueden suministrar un número ilimitado de constructores

· Si se suministra un constructor no 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 no 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 no 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)

Introducción a C#

TIPOS STRUCTS

· Structs y Clases son similares (lo veremos más adelante)

· Structs son tipos por valor, mientras que las Clases son tipos por referencia

· Para poder copiar un objeto a otro por valor (como las structs) se puede utilizar un método de System.Object → MemberwiseClone()

→ MemberwiseClone es un metodo protected dentro de System.Object se podría implementar un método publico en la clase que utilizara el método protected

(realiza una copia incompleta; únicamente tiene en cuenta los miembros por valor. Los miembros por referencia los mantiene apuntando a donde apunte el original)

Ejemplo 2, Ejemplo3

Introducción a C#

- Alternativa : Implementar el interfaz IClonable e implementar su método Clone() pudiendo realizar las tareas necesarias para hacer una copia completa, partiendo por ejemplo de una copia incompleta

Ejemplo4

Introducción a C#

INICIALIZADORES DE OBJETOS DE CLASES

- Los objetos se inicializan tipícamente desde el constructor, pasándole los valores como parámetros

- Como alternativa se pueden suministrar valores para los campos/propiedades de acceso público mediante pares nombre/valor según la siguiente sintaxis :

```
<ClassName> <variableName> = new <ClassName>
{
    <propertyOrField1> = <value1>,
    <propertyOrField2> = <value2>,
    ...
    <propertyOrFieldN> = <valueN>
};
```

Introducción a C#

- Antes de esta inicialización se ejecuta el constructor por defecto.

- También se podría llamar a un constructor no por defecto si se hubiera colocado paréntesis antes de las llaves { } junto con los posibles parámetros. Se ejecuta antes de esta inicialización, por lo que la inicialización podría cambiar los valores inicializados en el constructor.

- Si el campo a inicializar es más complejo que un tipo simple,se podría utilizar un inicializar dentro de este

→ inicializador anidado

O inicializar el objeto interno mediante un constructor no por defecto

Introducción a C#

Nota :

- Los inicializadores NO son un sustituto de los constructores, ya que no siempre es posible conocer hasta que no se ejecuta los valores a inicializar, por lo que para ese tipo de situaciones no son válidos

Ejemplo5
]