

# PROGRAMACIÓN EN C#

## 8. LAS CLASES

1

## LAS CLASES.

### PROGRAMACIÓN ORIENTADA A OBJETOS.

Técnicas de programación:

#### ESTRUCTURADO:

- Hemos usado hasta ahora.
- Se ejecutan las líneas de código de forma secuencial.
- Limitaciones y problemas.
- Mantenimiento del código más complejo.

#### ORIENTADO A OBJETOS:

- Nueva forma de organizar el código.
- Código agrupado en elementos que se conocen como objetos.

2

# LAS CLASES.

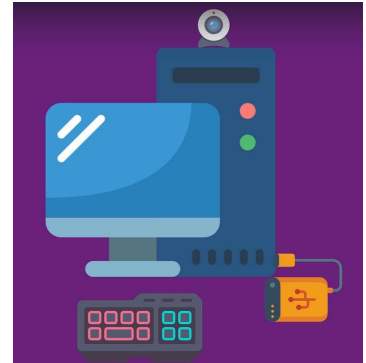
## EL OBJETO Y LAS CLASES.

**Objeto:** Elementos de la vida real, ej: monitor.

- Datos (**campos**): tamaño, color, material...
- Comportamiento (**métodos**): encender, apagar, cambiar de canal...

**Sistema:** Los objetos forman parte de un sistema, ej: el monitor forma parte del ordenador.

**Clase:** plantilla a partir de la cual se crean objetos.



3

3

# LAS CLASES.

## MIEMBROS DE LA CLASE.

**Datos:**

- Campos.
- Constantes.

**Comportamiento:**

- Propiedades.
- Métodos.
- Constructores.
- Finalizadores.



4

4

# LAS CLASES.

## DECLARACIÓN DE UNA CLASE.

```
//[modificador de acceso][class][identificador]
public class Automovil
{
    //Campos
    public string color, combustible, modelo;
    public byte año, numPuertas;
    public int ccMotor;

    //Métodos
    public bool Acelerar(){
        bool acelerar = true;
        Console.WriteLine("Acelerar");
        return acelerar;
    }
    ...
}
```

```
...
public bool Frenar(){
    bool frenar = true;
    Console.WriteLine("Frenar");
    return frenar;
}

public void Velocidad(ref byte velocidadPa){
    velocidadPa++;
    Console.WriteLine("Cambio velocidad");
}

}

} //class
```



5

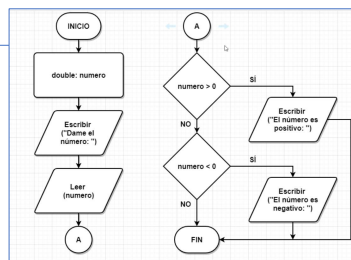
5

# LAS CLASES.

## DIAGRAMAS UML.

### PROGRAMACIÓN ESTRUCTURADA:

- Algoritmo.
- Pseudocódigo.
- Diagrama de flujo



### PROGRAMACIÓN ORIENTADA A OBJETOS:

- Diagramas UML.
- Unified Modeling Language.
- Lenguaje visual o gráfico.
- Permite crear diagramas o modelado de sistemas.



6

# LAS CLASES.

## CREANDO UN OBJETO O INSTANCIA DE CLASE.

```
using System;
...

namespace Seccion7{
    class Program {
        static void Main (string[] args){
            //Instancia de clase Automovil
            Automovil automovil1 = new Automovil();
        }
    }
    Public class Automovil{
        ...
    }
}
```



7

7

# LAS CLASES.

## ACCEDER A LOS CAMPOS DE CLASE.

```
class Program {
    static void Main (string[] args){
        Automovil automovil1 = new Automovil();
        automovil1.color = "rojo";
        Console.WriteLine("El color es: {0}", automovil1.color);
    }
}

public class Automovil{
    public string color, modelo, combustible;
    ...
}
```



10

10

# LAS CLASES.

## ACCEDER A LOS MÉTODOS DE CLASE.

```
class Program {
    static void Main (string[] args){
        bool acelerar;
        Automovil automovil1 = new Automovil();
        acelerar = automovil1.Acelerar();
        If (acelerar){
            Console.WriteLine("Acelerando correctamente");
        }
    }
}

public class Automovil{
    ...
    public bool Acelerar(){
        ....
    }
}
```



11

11

# LAS CLASES.

## MODIFICADOR STATIC.

Hay dos tipos de miembros en las clases:

- Miembros estáticos.
  - Pertenecen a la clase.
  - No es necesario instanciar a la clase para usarlos.
  - Métodos estáticos. Desventajas:
    - No puede llamar a otros métodos, a no ser que sean estáticos.
    - No se puede usar el operador this.
- Miembros de instancia.
  - Pertenecen al objeto.
  - Necesario instanciar a la clase para usarlos.



12

12

# LAS CLASES.

## MODIFICADOR STATIC.

```
class Program {
    static void Main (string[] args){
        Automovil automovil 1 = new Automovil();
        Automovil.Prueba(); //Llamada por medio de la propia clase
        automovil1.Prueba();//ERROR, el objeto no accede a métodos estáticos
    }
}
public class Automovil{
    public bool Acelerar(){
        ...
        Prueba();//Llamada desde la propia clase
    }
    public static void Prueba(){
        Console.WriteLine("Método estático");
    }
}
```



13

13

# LAS CLASES.

## CREANDO PROPIEDADES.

**Propiedad es un miembro de clase proporciona un método flexible para acceder a campos privados.**

Características:

- Permite a la clase mostrar al exterior de una manera pública los campos privados.
- Existen dos descriptores de acceso: get y set.




14

14

# LAS CLASES.

## DESCRIPTOR DE ACCESO GET.



```
class Program {
    static void Main (string[] args){
        Automovil automovil1 = new Automovil();
        Console.WriteLine(automovil1.color); //ERROR, no accesible por ser privado
        Console.WriteLine(automovil1.Color);
    }
}


public class Automovil{
    private string color = "rojo", modelo, combustible;

    //[acceso][tipo][Nombre]
    public string Color{
        //descriptor de acceso get o accessors
        get { return color; }
    }
}
```

15

# LAS CLASES.

## DESCRIPTOR DE ACCESO SET.



```
class Program {
    static void Main (string[] args){
        Automovil automovil1 = new Automovil();
        automovil1.combustible = "Diesel"; //ERROR, no accesible por ser privado
        automovil1.Combustible = "Diesel";
        Console.WriteLine("El combustible es:{0}", automovil1.Combustible);
    }
}

public class Automovil{
    private string color = "rojo", modelo, combustible;
    //[acceso][tipo][Nombre]
    public string Combustible{
        //descriptor de acceso get
        get { return combustible;}
        //descriptor de acceso set o accessors
        set { combustible = value; }
    }
}
```

16

# LAS CLASES.

## MÉTODO TOSTRING() Y MODIFICADOR OVERRIDE.

```
class Program {  
    static void Main (string[] args){  
        Automovil automovil1 = new Automovil();  
        Console.WriteLine(automovil1.ToString());  
    }  
}  
public class Automovil{  
    ...  
    public override string ToString(){  
        string mensaje;  
        mensaje = "Color: " + color + " Modelo: " + modelo + " Año: " + año;  
        return mensaje;  
    }  
}
```

