Programación orientada a objetos y clases

En este tutorial veremos cómo implementamos la realidad a nuestras aplicaciones, para ello utilizaremos la programación orientada a objetos, también llamada POO.

Índice

1 - Qué es la programación orientad a objetos?

Es un paradigma que convierte objetos reales a entidades de código llamadas clases. Esto puede parecer muy complejo, pero no lo es para nada.

Imaginémonos un objeto del mundo real como puede ser una moto

Como podemos observar en la imagen, disponemos de dos entidades. `Piloto` y `Moto` .

Dependerá de la lógica de negocio saber si es la moto la que contiene al piloto, o es el piloto el que contiene la moto. Para nuestro ejemplo, la moto contendrá al piloto.

Cuando convertimos la moto a un objeto, debemos pesar en sus características, como marca, modelo, número de ruedas, cilindrada, etc. Estas características serán nuestras propiedades.

Además de eso, en la moto realizas acciones, como son arrancar, acelerar, frenar, cambiar de marcha, todas estas acciones serán los métodos.

Por lo tanto, una clase es un tipo específico que sirve para crear objetos.

Como recordamos de tutoriales anteriores, tenemos tipos primitivos como `int` , `decimal` , etc. Las clases son un tipo más, pero personalizado a nuestro gusto.

2 - Crear una clase

Para crear una clase debemos indicar la palabra registrada `class` y posteriormente el nombre que queremos utilizar. Como en todos los bloques de código, definiremos cuando empieza y cuando acaba utilizando los las llaves `{` y `}` , dentro del bloque de código, introduciremos nuestras propiedades y nuestros métodos.

```
class Moto
{
    public decimal VelocidadMaxima { get; set; }
    public int NumeroRuedas { get; set; }
    public Motorista Piloto { get; set; }

    public Acelerar(){
        //Código aquí
    }
    public Arrancar(){
        //Código aquí
    }
}
```

Como vemos además utilizamos la palabra reservada `public` la cual es un modificador de acceso, que podemos ver en el curso de los modificadores de acceso.

2.1 - Crear una instancia de una clase

Un objeto es la representación en memoria de una calase y puedes crear tantos objetos de una clase como quieras, para ello utilizaremos la palabra reservada `new` .

```
Moto aprilia = new Moto();
```

Posteriormente le podemos dar valores a sus propiedades, así como llamar a sus métodos. Por supuesto, al disponer de `Piloto` dentro del tipo `Moto` , también le podemos dar valores

```
aprilia.VelocidadMaxima = 320;
aprilia.NumeroRuedas = 2;
aprilia.Acelerar();

aprilia.Piloto.Nacionalidad = "ESP";
```

3 - Constructores

Como hemos observado, cuando creamos la instancia de la clase utilizamos `new Moto()` bien, esta sentencia lo que hace es llamar al constructor por defecto dentro de `System.Object` , pero claro, que pasa si no queremos utilizar el constructor por defecto, ya que el constructor por defecto nos inicializa todas las variables a null.

Para ello podemos crear un constructor personalizado por nosotros mismos, donde podremos asignar valores manualmente, como podemos observar en el ejemplo. Ahora cuando realizamos `Moto aprilia = new Moto();` nos creara por defecto una `VelocidadMaxima` de 320 y un `NumeroRuedas` de 2;

```
class Moto
{
    public decimal VelocidadMaxima { get; set; }
    public int NumeroRuedas { get; set; }
    public Motorista Piloto { get; set; }
    public string Marca { get; set; }
    public string Modelo { get; set; }

    public Moto(){
        VelocidadMaxima = 320;
        NumeroRuedas = 2;
    }

    public Acelerar(){
        //Código aquí
    }
    public Arrancar(){
        //Código aquí
    }
}
```

Pero ¿qué pasa si tenemos múltiples motos?

En nuestro ejemplo al tener los valores asignados en el constructor por defecto se nos crearían todas las motos con los mismos valores, esto quiere decir que en las siguientes sentencias

```
Moto aprilia = new Moto();
Moto Ducati = new Moto();
```

Ambas motos tendrán una `VeloidadMaxima` de 320 y un valor de 2 en `NumeroRuedas`, por lo que los valores podrían no ser correctos.

Para arreglar este problema, lo que hacemos es añadir un constructor que acepte valores por parámetro. Utilizando Sobrecarga de operadores podemos crear tantos constructores como deseemos.

```
class Moto
{
    public decimal VelocidadMaxima { get; set; }
    public int NumeroRuedas { get; set; }
    public Motorista Piloto { get; set; }
    public string Marca { get; set; }
    public string Modelo { get; set; }

    public Moto(decimal velocidadMaxima, int NumeroRuedas){
        VelocidadMaxima = velocidadMaxima;
        NumeroRuedas = NumeroRuedas;
    }

    public Moto(string marca, string modelo){
        Marca = marca;
        modelo = modelo;
    }

    public Moto(){
        VelocidadMaxima = 320;
        NumeroRuedas = 2;
    }

    public Acelerar(){
        //Código aquí
    }
    public Arrancar(){
        //Código aquí
    }
}
```

Como podemos comprobar, ambos constructores funcionan a la perfección

```
Moto aprilia = new Moto("Aprilia", "RX");
Moto motoSinMarca = new Moto(210,2);
```

Otra característica muy importante dentro de la programación orientada a objetos es la herencia. La cual nos permite heredar información de una clase padre a su hijo.

En el ejemplo anterior de la moto, supongamos que tenemos también un coche, ambos son vehículos, por lo que ambos tienen características comunes, como pueden ser marca, modelo, numero de ruedas, y otras diferentes como pueden ser la cilindrada para las motos o la tracción para los coches. El uso de la Herencia nos permite ahorrar multitud de líneas de código y complejidad dentro de nuestros programas.

```
class Vehiculo
{
    public decimal VelocidadMaxima { get; set; }
    public int NumeroRuedas { get; set; }
    public string Marca { get; set; }
    public string Modelo { get; set; }

    public Vehiculo(decimal velocidadMaxima, int NumeroRuedas){
        VelocidadMaxima = velocidadMaxima;
        NumeroRuedas = NumeroRuedas;
    }

    public Vehiculo(string marca, string modelo){
        Marca = marca;
        Modelo = modelo
    }

    public Acelerar(){
        //Código aquí
    }

    public Arrancar(){
        //Código aquí
    }
}

class Moto : Vehiculo
{
    public int Cilindrada { get; set; }

    public Moto(decimal velocidadMaxima, int NumeroRuedas, int cilindrada) : base(
       Cilindrada = cilindrada;
    }

    public Moto(string marca, string modelo, int cilindrada) : base(marca, modelo)
        Cilindrada  = cilindrada;
    }
```

```
    public void HacerCaballito(){
        //codigo
    }
}

class Coche : Vehiculo
{
    public string Traccion { get; set; }

    public Coche(string marca, string modelo, string Traccion) : base(marca, model

    }

    public bool CerrarPuertas(){

    }

}
```

Como podemos observar tanto `Coche` como `Moto` heredan, utilizando `:` , de la clase `Vehiculo` , esto quiere decir que desde ambas clases podemos acceder a los métodos y propiedades de la clase Vehiculo, además de a las suyas propias.

Otro detalle interesante es que como podemos observar en el constructor, desde la clase hijo se llama al constructor de la clase padre utilizando la palabra reservada `: Base()` y mandando los parámetros que ese constructor necesita.