# Programación orientada a objetos (I)

## 1. INTRODUCCIÓN

Desde la versión 5 de PHP que se puede decir que se trata de un lenguaje de programación **orientado a objetos**.

Hoy en día este paradigma de programación es el más utilizado y prácticamente lo podemos encontrar en cualquier lenguaje de programación que se precie.

Pero, ¿qué es realmente la orientación a objetos?

En programación orientada a objetos, el sistema se divide en **objetos** o entidades que podemos encontrar en la realidad.

Veamos un ejemplo. Imagina que está realizando la simulación de un coche.

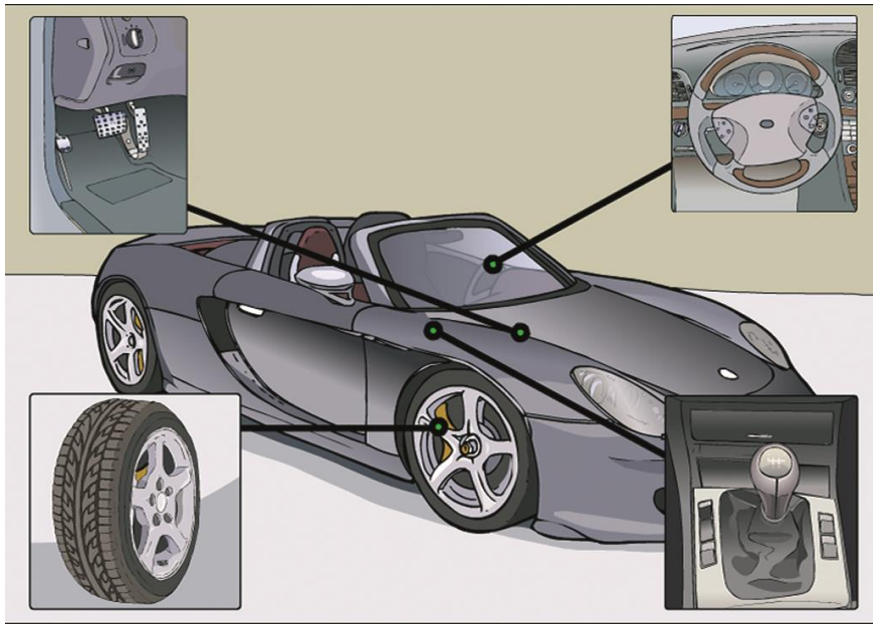
Realizando una descomposición **funcional** del coche, en la que no se utiliza un enfoque orientado a objetos, hablaríamos de:

- Arrancar y apagar.
- Girar a la izquierda y a la derecha.
- Cambiar de marcha.
- Moverse hacia delante.
- Moverse hacia atrás.
- Aumentar la velocidad.
- Disminuir la velocidad.
- Etc.

Sin embargo, si pensaras en objetos, hablarías de:

*El coche dispone de un motor que se puede arrancar y apagar utilizando para ello el contacto. Además, el coche dispone de un volante que permite que nos desplacemos en una determinada dirección: derecha o izquierda. Al aumentar la velocidad utilizando el acelerador, debemos cambiar a una marcha más alta. Si reducimos la velocidad con el freno, puede que sea conveniente cambiar a una marcha inferior...*

Observa cómo en la descomposición funcional nos hemos fijado en lo que debería hacer el coche, mientras que en la descomposición en objetos nos hemos centrado en qué elementos podemos encontrar: motor, volante, acelerador, freno, caja de cambios, etc.



Al pensar en objetos en lugar de en funciones, te acercas más a la realidad, ya que es la forma en que lo harías en cualquier otra situación cotidiana.

De esta forma, una vez encontrados los objetos que intervienen, podremos asignarles las tareas que tienen que realizar.

Puede que te estés preguntando para qué nos va a servir conocer los conceptos de orientación a objetos a la hora de desarrollar una aplicación web con PHP.

Bien, aunque no es obligatorio utilizar un enfoque orientado a objetos en PHP, es frecuente encontrarnos con paquetes o librerías creadas por terceros que añaden funcionalidad específica al lenguaje.

La mayoría de estas librerías están desarrolladas con un enfoque orientado a objetos, por lo que si las queremos utilizar, es necesario saber cómo. Incluso nosotros mismos podemos aplicar este enfoque a la hora de desarrollar nuestras aplicaciones.

Vamos a ver, por lo tanto, cómo se utiliza la programación orientada a objetos en PHP.

## 2. CLASES

El elemento clave en todo lenguaje orientado a objetos es la **clase**. Una clase es como una plantilla o modelo que define a los objetos.

Por ejemplo, en el caso del coche, piensa en la clase como en la idea genérica que tienes de un coche. Sin embargo, un **objeto** sería un coche en particular que estás viendo en la calle.

Está claro que utilizarás un coche determinado (un objeto) y que no puedes hacer nada con la idea de coche que puedas tener (la clase).

Esta es la realidad. Traslademos ahora la idea al ambiente de programación. Normalmente las clases se escriben en archivos independientes para más tarde incluirlos allá donde necesitemos la funcionalidad que nos ofrecen.

PHP proporciona la palabra clave **class** para definir una clase:

```
class Coche
{

}
```

Dentro de las llaves incorporaremos el código que representa las características de la clase **Coche**. Para ello, debemos saber qué puede contener una clase.

Volvamos al ejemplo del coche. Estoy seguro de que si te piden que describas un coche, empezarías a indicar sus características físicas: tiene cuatro ruedas, puede ser de distintos colores, etc.

Estas características que definen el aspecto del coche son **propiedades** cuando se trasladan al ambiente de programación.

```
class Coche
{
     var $numruedas;
     var $color;
     var $posx;
     var $posy;
     var $velocidad;
}
```

Con las propiedades **posx** y **posy** indicaremos la posición del coche.

Siguiendo con la descripción de lo que es un coche, también indicarías que puede moverse en una determinada dirección, que puede tomar velocidad: aumentándola y reduciéndola...

Las características que definen lo que puede hacer el coche o, en definitiva, su comportamiento, son llamadas **métodos** en el ambiente de la programación.

```
function mover($x, $y)
{
      $posx = $x;
      $posy = $y;
}
```

Este es el aspecto que tiene un método. Recuerda que, gracias a los métodos, podemos indicar el comportamiento que pueden tener los objetos de la clase.

El método **mover** sirve para cambiar la posición del coche. También podemos tener otros métodos, como **acelerar** y **frenar** para variar su velocidad.

```
class Coche
{
      var $numruedas;
      var $color;
      var $posx;
      var $posy;
      var $velocidad;

      function mover($x, $y)
      {
            $posx = $x;
            $posy = $y;
      }
      function acelerar()
      {
            $velocidad = $velocidad + 10;
            return $velocidad;
      }
      function frenar()
      {
            if ($velocidad > 10)
                  $velocidad = $velocidad - 10;
            else
                  $velocidad = 0;
            return $velocidad;
      }
}
```

Observa lo directo que puede ser trasladar una idea que manejas en la realidad a un programa orientado a objetos.

## 3. PROPIEDADES

La parte de los objetos que permite describirlos detalladamente son las **propiedades**.

Las propiedades no siempre serán características físicas. Por ejemplo, podrías tener una propiedad que indicara la máxima velocidad que puede alcanzar un objeto **coche**.

Lógicamente, no todos los objetos **coche** van a tener la misma máxima velocidad.

Las propiedades se corresponden con variables en el interior de las clases. Así, en nuestra clase **Coche** hemos definido varias propiedades: **$numruedas**, **$color**, **$posx**, **$posy**, **$velocidad**...

Aunque no es necesario **declarar** previamente las propiedades, ya que son variables, es mejor hacerlo con la palabra **var** para entender mejor la estructura de la clase.

También podríamos haber asignado valores iniciales a estas propiedades en el mismo momento de su declaración. Esos valores iniciales tendrían que ser valores constantes.

Cada objeto se caracterizará por tener un determinado valor en dichas propiedades. Además, este valor podrá cambiar a lo largo de la *"vida"* del objeto.

Por ejemplo, un coche de color rojo y que está aparcado tendrá el valor rojo en la propiedad color y 0 en la de velocidad.

Sin embargo, un coche azul metalizado que va por la autopista podrá tener el valor azul metalizado en su propiedad color y 110 en su propiedad velocidad.

Observa cómo, en cualquier caso, ambos coches dispondrán del mismo número de ruedas: 4.

Esto nos hace ver que existen dos tipos de propiedades: aquellas que pueden tomar valores distintos en cada objeto y aquellas que son iguales para todos o que las comparten todos los objetos.

En este último caso se dice que son propiedades **compartidas** o **estáticas**. En PHP podrás indicar esta circunstancia antecediendo a su declaración la palabra **static**.

```
static $numruedas = 4;
```

De esta forma indicamos que el valor de la propiedad **numruedas** es compartido por todos los objetos que se creen de la clase **Coche**.

Esto quiere decir que si se modifica esta propiedad, se estará modificando para todos los objetos de esa clase.

En la mayoría de los casos tendrá sentido indicar el valor que van a tomar las propiedades compartidas en la propia declaración. En la línea de código anterior, la propiedad **$numruedas** toma el valor 4, por lo que todos los objetos "coche" se crearán con 4 ruedas.

Lo importante es entender que definimos las propiedades en la clase, pero que estas sólo tendrán sentido cuando creemos objetos.

## 4. MÉTODOS

Si las propiedades describen estructuralmente los objetos, los **métodos** hacen lo propio con su comportamiento.

Los métodos se implementan en PHP a través de **funciones**.

Veamos como ejemplo el método **mover**. Tras el nombre de la función, aparece, entre paréntesis, una lista de parámetros.

Un parámetro sirve para comunicar el método con el resto del código que lo utiliza. En este caso, el método **mover** permite desplazar el coche de una posición a otra. Lógicamente, necesita conocer la posición final, que se le indica a través de los parámetros **x** e **y**.

```
function mover($x, $y)
{
      $posx = $x;
      $posy = $y;
}
```

**Al igual que en el caso de las propiedades, podrías tener métodos compartidos, es decir, que tienen sentido solo cuando se piensa en la clase y no cuando estamos trabajando con un objeto en particular. Para indicarlo, utilizarías la palabra static delante de function.**

También puede darse el caso de que el método no utilice parámetros, con lo que la lista aparecerá vacía (como el método **acelerar**).

En el caso de que el método tenga que devolver un valor, utilizaremos la palabra clave **return** y lo indicaremos. Por ejemplo, vemos que el método **acelerar** devuelve la velocidad final del coche, es decir, tras acelerar.

```
function acelerar()
{
      $velocidad = $velocidad + 10;
      return $velocidad;
}
```

Existe un método especial, que permite crear objetos de la clase. Este método se llama **constructor**.

```
function __construct($color, $posx, $posy)
{
     $this->color = $color;
     $this->posx = $posx;
     $this->posy = $posy;
     $this->velocidad = 0;
}
```

En las versiones anteriores de PHP 5, los métodos constructores no se llamaban **__construct**, sino que tenían el mismo nombre que la clase.

Para mantener la compatibilidad con el código escrito antes de PHP 5, cuando se crea un objeto, el intérprete busca primero un método **__construct** y, si no lo encuentra, un método con el mismo nombre de la clase.

Fíjate en el nombre especial de este método. Se utilizan dos subrayados delante de la palabra **construct** para indicar que es el constructor de la clase.

Solo puede haber un constructor por clase y no es obligatorio escribirlo. Lo harás cuando quieras inicializar las propiedades al crear el objeto o por cualquier otro proceso que tenga que ejecutarse en ese momento.

En el método constructor es donde podemos establecer el valor que tienen las propiedades al crear el objeto.

Por ejemplo, en este caso vamos a crear un objeto con un determinado color y posición inicial. La velocidad inicial será siempre de 0 kilómetros por hora.

La forma de acceder a una determinada propiedad del objeto es a través de la sintaxis: **nombre_objeto->nombre_propiedad**.

**No utilices this fuera de la clase, ya que obtendrás un error.**

En el constructor se utiliza **$this**. La palabra clave **this** indica que estamos haciendo referencia al propio objeto que se está creando. Así, damos el valor del parámetro **$color** a la propiedad **color** del objeto.

Siempre que quieras acceder a alguna propiedad o método de un objeto debes indicar el nombre del objeto o, al menos, **$this** si es en la propia definición de la clase.

Por lo tanto, el código que habíamos escrito para los métodos no es correcto porque no se accede de forma adecuada a las propiedades del objeto.

Por ejemplo, en el método **mover** vemos que la intención es modificar el valor de las propiedades **posx** y **posy** del objeto sobre el que se llama el método. El código correcto sería:

```
function mover($x, $y)
{
      $this->posx = $x;
      $this->posy = $y;
}
```

Por lo tanto, para acceder a la propiedad **posx**, utilizamos la expresión **$this->posx**. Lo mismo ocurriría en el resto del código siempre que queramos acceder a una propiedad o método.

Archivo **coche.inc.php**:

```
<?php
class Coche
{
      static $numruedas = 4;
      var $color;
      var $posx;
      var $posy;
      var $velocidad;

      function __construct($color, $posx, $posy)
      {
            $this->color = $color;
            $this->posx = $posx;
            $this->posy = $posy;
            $this->velocidad = 0;
      }

      function mover($x, $y)
      {
            $this->posx = $x;
            $this->posy = $y;
      }
      function acelerar()
      {
            $this->velocidad = $this->velocidad + 10;
            return $this->velocidad;
      }
      function frenar()
      {
            if ($this->velocidad > 10)
                $this->velocidad = $this->velocidad - 10;
            else
                $this->velocidad = 0;
            return $this->velocidad;
      }
}
?>
```

Volviendo al método constructor, la idea es que cuando creemos objetos en el código que utiliza la clase, indiquemos el valor de las propiedades.

Observa que no es necesario indicar el valor de la propiedad **numruedas** ya que este valor se indica en la propia declaración de la propiedad al ser una propiedad compartida.

Por lo tanto, ya sabemos que podremos utilizar el método constructor para crear objetos y, además, hemos visto cómo podemos acceder a las propiedades de los objetos.

En la lección siguiente aprenderemos a crear objetos y a utilizar sus métodos.