Programación orientada a objetos (II)

1. VISIBILIDAD

Uno de los principios de la programación orientada a objetos es que la clase solo debe exponer los servicios que proporciona, no así su implementación.

Esto, que suena muy formal, se entiende mejor con la analogía del coche que estamos utilizando.

Cuando conduces un coche, lo único que necesitas es saber cómo frenar, acelerar, cambiar de marcha, etc.

No necesitas conocer cuál es el proceso que se sigue cuando pulsas el pedal del freno o del acelerador.

De hecho, si más tarde decides poner un sistema antibloqueo de frenos, seguro que no te gustaría que te cambiaran tu forma de frenar. Lo que quieres es seguir presionando el freno como siempre, independientemente de que se utilice un mecanismo diferente.

Este mismo enfoque es el que se sigue en la programación orientada a objetos.

En nuestra clase nosotros debemos mostrar al exterior que el coche puede acelerar, frenar o moverse, sin necesidad de conocer que cuando lo hace, lo que realmente ocurre es que cambia el valor de una o más propiedades.

PHP permite *"ocultar"* aquellas características de las clases que no queremos mostrar fuera de ella. Para ello, se utiliza la palabra **private**.

Normalmente, las propiedades deben ser privadas.

```
class Coche
{
     static $numruedas = 4;
     private $color;
     private $posx;
     private $posy;
     private $velocidad;
.....
}
```

La propiedad **numruedas** no es privada porque queremos poder utilizarla sin necesidad de crear un objeto para ello. Por lo tanto, debe ser pública o accesible desde fuera de la clase.

Copyright © Computer Aided Elearning, S.A.

Es una buena costumbre utilizar la palabra public para indicar el acceso público, ya que facilita la lectura del código de la clase.

Por otra parte, los métodos deberían ser públicos, ya que representan los servicios que la clase ofrece al exterior. Esto se consigue mediante la palabra **public** o simplemente no indicando nada, ya que el acceso predeterminado es público.

```
class Coche
{
      .....


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

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

}
```

Sin embargo, en muchas ocasiones desearemos poder acceder a algunas propiedades, tanto para conocer su valor como para poder modificarlo.

Si las definimos como privadas, esto no será posible. Por ello, para estas propiedades, deberás incluir un método específico para ello.

Estos métodos se conocen como métodos **get** y **set**, ya que obtienen y modifican las propiedades, respectivamente. Puedes utilizar cualquier nombre, pero es una buena costumbre incluir estas palabras inglesas.

Veámoslo para la propiedad **color**:

```
public function getColor()
{
     return $this->color;
}
public function setColor($valor)
{
     switch ($valor)
     {
           case "rojo":
                 $this->color = "rojo";
                 break;
           case "verde":
                 $this->color = "verde";
                 break;
           case "azul":
                 $this->color = "azul";
                 break;
           default:
                 $this->color = "ninguno";
     }
}
```

Vemos que en el caso del método **get**, lo único que hacemos es devolver el valor de la propiedad mediante la instrucción **return**.

En el método **set** incluimos el código necesario para establecer el valor de la propiedad. Dicho valor viene dado a través del parámetro **valor**.

En el ejemplo hacemos algo más, ya que validamos el color y, si no es uno de los permitidos (rojo, verde o azul), lo establecemos como "ninguno".

Esta es una de las principales ventajas de utilizar métodos para el acceso a las propiedades.

Si permitiéramos el acceso directo a la propiedad **color**, entonces se podría establecer a cualquier valor, incluso alguno que no fuera correcto.

Fíjate que el acceso a estos procedimientos tiene que ser **public** para poder utilizarlos desde fuera de la clase donde está definida la propiedad.

Utilizando los métodos **get** y **set** en lugar de las propiedades fuera de la clase, será muy sencilla cualquier modificación posterior, ya que solo la tendremos que hacer en estos métodos.

A continuación puedes ver el código de la clase **Coche**.

Copyright © Computer Aided Elearning, S.A.

```
class Coche
{
	static $numruedas = 4;
	private $color;
	private $posx;
	private $posy;
	private $velocidad;

	public function getColor()
	{
		return $this->color;
	}
	public function setColor($valor)
	{
		switch ($valor)
		{
			case "rojo":
				$this->color = "rojo";
				break;
			case "verde":
				$this->color = "verde";
				break;
			case "azul":
				$this->color = "azul";
				break;
			default:
				$this->color = "ninguno";
		}
	}

	public function getPosx()
	{
		return $this->posx;
	}
	public function setPosx($valor)
	{
		$this->posx = $valor;
	}

	public function getPosy()
	{
		return $this->posy;
	}
	public function setPosy($valor)
	{
		$this->posy = $valor;
	}

	public function getVelocidad()
	{
		return $this->velocidad;
	}
	public function setVelocidad($valor)
	{
		$this->velocidad = $valor;
	}
```

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

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

2. CREAR OBJETOS

Una vez hemos completado el código de nuestra sencilla clase **Coche**, vamos a utilizarla para poder crear y *"conducir"* coches.

Lo normal a la hora de crear las clases es hacerlo en un archivo independiente, por lo que será necesario incluir dicho archivo en las páginas donde vayamos a utilizar la clase.

Así pues, podríamos incluir la siguiente línea en el archivo **coches.php**, que es donde utilizaremos la clase anterior:

```
<?php
include_once("coche.inc.php");
…..
?>
```

Al incluir el archivo de la clase, podremos utilizarla perfectamente. Lo primero será crear algún objeto **Coche**:

```
$c = new Coche("rojo", 0, 0);
```

Copyright © Computer Aided Elearning, S.A.

Con esta línea creamos nuestro primer objeto **Coche**. Es decir, estaremos utilizando su constructor para ello.

Se utiliza una variable para guardar el nuevo objeto, la palabra clave **new**, el nombre de la clase y, entre paréntesis, los parámetros que necesite el constructor.

Si el constructor de la clase no necesita parámetros, será suficiente con incluir un par de paréntesis.

Una vez creado el objeto, podremos utilizar sus características (propiedades y métodos) públicas.

Por ejemplo, la expresión **`$c->mover(5, 78)`** es correcta porque estamos utilizando el método público mover del objeto; pero **`$c->color`** no lo sería porque estaríamos accediendo a una propiedad privada. En lugar de esto último, deberíamos utilizar el correspondiente método **get**: **`$c->getColor()`**.

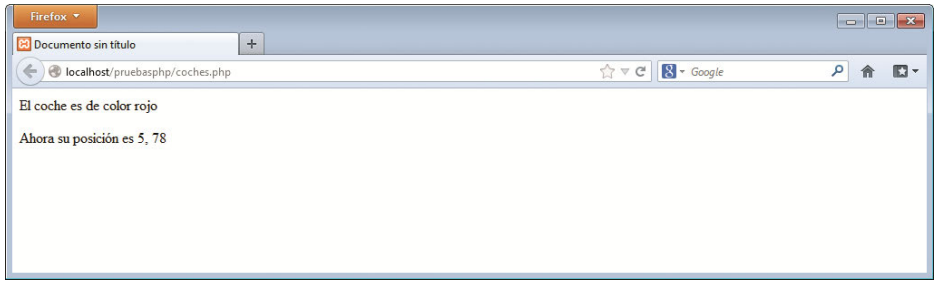
No siempre vas a necesitar un método **get** y otro método **set** para todas las propiedades. Por ejemplo, podrías encontrarte con propiedades que no permitirás modificar (solo lectura), por lo que sobraría su método **set**; o con propiedades que no permitirás consultar (solo escritura), por lo que no tendría sentido su método **get**.

Eres tú el que está diseñando la clase, así que tú decides lo que necesitas.

Fíjate cómo se utiliza el objeto en el siguiente código:

```
<body>
<?php
     include_once("coche.inc.php");
     $c = new Coche("rojo", 0, 0);
     echo "El coche es de color " . $c->getColor();
     $c->mover(5, 78);
     echo "<p>Ahora su posición es " . $c->getPosx() .
          ", " . $c->getPosy();
?>
</body>
```

Y ahora observa el resultado en el navegador.



3. DESTRUCTORES

Hemos podido comprobar que para trabajar con objetos en nuestro código, no necesitamos nada más que utilizar el constructor de la clase para crearlos y, a partir de ese momento, acceder a sus características públicas.

El objeto creado de esta forma existirá mientras se ejecute el código php y desaparecerá al finalizar este.

Es decir, que no necesitamos hacer nada para destruir el objeto, ya que esto ocurrirá cuando finalice el código en el que se ha creado.

Sin embargo, en ocasiones esto no es suficiente y tenemos que asegurarnos de que, al desaparecer el objeto, se cierren ciertos recursos, como archivos, conexiones con bases de datos, etc.

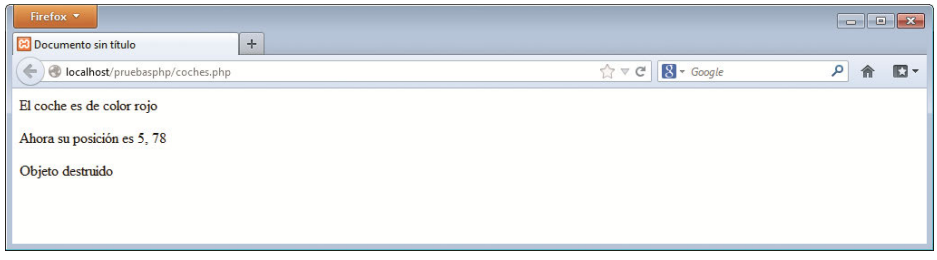
Podemos escribir un método especial llamado **destructor** para ello. Sin embargo, recuerda que solo necesitarás hacerlo cuando realmente tengas que llevar a cabo algún proceso al destruir el objeto.

```
public function __destruct()
{
   echo "<p>Objeto destruido</p>";
}
```

En este caso, el destructor simplemente imprimirá un texto en la página, pero nos servirá para comprobar que realmente se está ejecutando el código del destructor que ha creado.

Fíjate que el nombre es muy parecido al caso del constructor. Ahora se trata de dos subrayados y la palabra **destruct**.

Sin necesidad de llamar al destructor, se ejecutará su código porque se destruye automáticamente el objeto al finalizar la ejecución del código o al perder su ámbito.



Si necesitas destruir el objeto en un momento dado sin esperar a esto, puedes utilizar **unset()** y pasar el nombre del objeto como parámetro.

Copyright © Computer Aided Elearning, S.A.