Parte II

Sintaxis básica de Programación Orientada a Objetos en PHP

Programación

El 90% de lo ya aprendido de PHP nos sirve para la Programación Orientada a Objetos.

Básicamente, aprenderemos unas nuevas palabras reservadas para definir las clases, la visibilidad de sus atributos y sus métodos…

Lo más complicado puede ser el cambio mental que debemos hacer para pasar de una programación estructurada a otra orientada a objetos.

Ese cambio no debe suponer ningún problema para los que ya sabéis manejar objetos de Java, porque básicamente es lo mismo.

Síntaxis PHP

- Inspirada en C, C++, Java y JavaScript
- Añade algunos operadores propios
  - Concatenar Cadenas “.”
  - Operador de objeto “->”
- Trata de manera diferente las cadenas encerradas entre comillas simples y dobles.

Creación de una clase

```
class Freelance
{

 //código de la clase…

}
```

Por convenio, el nombre de la clase se escribe con la primera letra en mayúsculas. Y también, la llave de apertura de la clase se escribe en la línea siguiente, y no a continuación, como ocurría hasta ahora en todo lo que hemos visto.

También es habitual guardar en un archivo cada clase, siendo el nombre del archivo el propio de la clase. En este caso, crearíamos el archivo freelance.php o también freelance.class.php para distinguir rápidamente que es una clase.

Definición de atributos

```
class Freelance
{
 public $nombre;
 protected $ocupado;
 private $precioHora = 10;
 private $comienzoTrabajo;

}
```

Definición de métodos

```
class Freelance
{

 public function desarrollar(){
  echo “Comienzo a trabajar”;

}
```

Instanciación (creación)

La instanciación (o creación) es el proceso por el cual se crea un ejemplar (objeto) a partir de una clase.

- Se pueden crear N objetos a partir de una clase.
- Durante el proceso de instanciación se puede (y generalmente es así) realizar una invocación a un método especial denominado “constructor”.
- Para la creación del objeto usamos la palabra reservada **new**

```
$freelance = new Freelance();
```

$freelance es una variable con una referencia a un objeto de la clase Freelance.

Notación

La notación PHP para el acceso a propiedades o métodos:

Operador ->

Recordad que en Java era un punto: .

Ejemplos:

```
$freelance->nombre = “Pepe”;

$freelance->desarrollar();
```

La primera línea permite acceder a una propiedad (nombre) del objeto $freelance, y la segunda línea es un mensaje que le lanzamos al objeto, el cual nos permite acceder a una funcionalidad del mismo.

Variable $this

Pero, después de lo anterior, la pregunta que nos podemos hacer es: ¿Cómo accedemos a las propiedades y métodos dentro de la propia clase?

La variable **$this** referencia al objeto que recibió el mensaje cuyo método se está programando.

```
class Persona
{
 private $nombre;

 public function presentate(){
   echo "Hola, soy " . $this->nombre;
 }
}
```

Así pues, la variable **$this** sólo tiene sentido dentro de la definición de la propia clase.

Ejemplo

```php
<?php

class Persona
{
 public $nombre;

 public function presentate(){
   echo "Hola, soy " . $this->nombre;
 }
}

$pepe = new Persona();
$pepe->nombre = "Pepe";
$pepe->presentate();

?>
```

Ejemplo: Clase Freelance

```
class Freelance
{
    private $nombre;
    protected $ocupado;
    private $precioHora = 10;
    private $comienzoTrabajo;

    public function desarrollar(){
        echo "<br>Soy ". $this->nombre . " y comienzo a trabajar";
        $this->ocupado = true;
        $this->comienzoTrabajo = time();
    }

    public function parar(){
        $this->ocupado = false;
        $horas_trabajadas = ceil((time() - $this->comienzoTrabajo)
/ 3600);
        echo "Terminé de trabajar. Facturo " . ($horas_trabajadas
* $this->precioHora);
    }
}
```

Vista pública/privada

La encapsulación hace que de una clase podamos exponer unas cosas u otras para que sean conocidas en el exterior.

La **vista pública** son aquellos datos u operaciones que la clase expone para que los demás trabajen con ella.

La **vista privada** es la implementación de esas operaciones y los datos que necesitamos para poder realizarlas.

Cuando describimos una clase, tenemos que declarar tanto la vista pública como la privada.

La **vista pública** dice lo que podremos hacer.

La **vista privada** es como hacemos las cosas, que está encapsulada.

Recordemos que la encapsulación es el proceso por el que se ocultan (deliberadamente) las características de algo.

Visibilidad

El modificador de visibilidad indica desde qué partes del código se permite, o no, acceder a los atributos o métodos de los objetos.

Public: Indica que podremos acceder desde cualquier parte del código. Es decir, tanto desde dentro como desde fuera de la clase. Si no indicamos visibilidad se entiende que es "public".

Private: Indica que podremos acceder sólo en la declaración de la clase. Si tratamos de acceder desde fuera de la clase, recibimos un error fatal.

Protected: Igual que private menos para el acceso desde sus clases hijas, que si se puede.

Acceso a una propiedad que no existe

PHP tiene como caso especial que, instanciado un objeto de una clase, podemos crearle una propiedad "al vuelo". Lo único es que esa propiedad sólo estará disponible para ese objeto de la clase, y no para todas las instancias de la clase. Es decir, la propiedad la crea para el objeto, no para la clase.

La propiedad creada así tiene por defecto visibilidad **public**

```
$freelance = new Freelance("Anastasio", 15);
$freelance->mote = "Tasio";
```

NOTA: Con "métodos mágicos" (los veremos dentro de unas cuantas diapositivas) podemos controlar que cosas hacer cuando pasan estos casos.

Ejercicios

- Crear una clase Enlace de HTML (etiqueta a). Pensar en que atributos debería tener el enlace. Implementar un método mostrar() para esa clase enlace, que produzca una salida en HTML para que ese enlace se visualice en la página. Instanciar algunos enlaces y lanzar el mensaje para que se impriman.

- Crear la clase Select. Será para implementar un campo SELECT de un formulario de HTML. Es parecido el mecanismo al ejercicio anterior del enlace, sólo que en este caso la etiqueta a producir es un poco más compleja. Pensar que atributos pueden ser necesarios y crear el método imprimete() para que el campo SELECT se muestre en la página.