Desarrollo Web en Entorno Servidor

Programación en PHP

José A. Lara

Programación Orientada a Objetos

- Uno de los cambios más espectaculares y maravillosos que sufrió PHP es adaptarse a los lenguajes de programación modernos incluyendo dentro de su sintaxis el paradigma de Programación Orientada a Objetos.

- Muchos desarrolladores en PHP aún siguen programando con una metodología muy secuencial y que por lo tanto les provoca muchos problemas cuando se enfrentan al mantenimiento y actualización de su código.

Programación Orientada a Objetos

Cuando alguien se plantea programar con el lenguaje PHP nunca se plantea hacerlo de una manera orientada a objetos:

- El cambio se produjo a partir de la versión 5 de PHP
- Compatibilidad hacia atrás
- Los cambios de versiones en PHP no se dan con la misma velocidad que en otros lenguajes (versiones durante 5 o más años)

Programación Orientada a Objetos

La introducción de la orientación a objetos con PHP hace que podamos y debamos pensar en este paradigma para poder diseñar y programar nuestras aplicaciones con la arquitectura Orientada a Objetos.

Intentaremos:

- Aprovechar la potencia de la arquitectura POO para diseñar y desarrollar todas nuestras librerías de una forma más abstracta y que de esa forma sea mantenible en el tiempo y flexible.
- Aprovechar las opciones de desarrollo secuencial allá donde no haya más remedio (en las vistas principalmente).

Programación Orientada a Objetos

¿Qué es un objeto?

- Un objeto lo podemos entender como la representación mínima de cualquier resolución a un problema que tengamos a nivel de software.

Programación Orientada a Objetos

- Un objeto tiene dos partes claramente diferenciadas:
  - Propiedades/atributos, que hace referencia a las características del objeto
  - Métodos/funciones, que hace referencia a las acciones que puede desarrollar.

Programación Orientada a Objetos

¿Qué es una clase?

- La definición básica de una clase comienza con la palabra reservada class, seguida de un nombre de clase, y continuando con un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a dicha clase.

Programación Orientada a Objetos

El nombre de clase puede ser cualquier etiqueta válida, siempre que no sea una palabra reservada de PHP. Un nombre válido de clase comienza con una letra o un guión bajo, seguido de una cantidad arbitraria de letras, números o guiones bajos.

Programación Orientada a Objetos

Si hubiéramos definido el código sin clases, se vería así:

```php
<?php
    $color = 'Verde';
    $tipo = 'turismo';
    function mostrarColor() {
        echo '<br>';
        echo 'El color del coche es:';
        echo $color;
        echo '<br>';
    }
    function getColor(){
        return $color;
    }
?>
```

Programación Orientada a Objetos

Aquí no hay forma de agrupar qué información y cuáles acciones van juntas. También tenemos otros problemas:

- No es posible prevenir que el programador cambie el color o el tipo del vehículo. De hecho, y dado que PHP no es un lenguaje tipificado, cualquier programador que utilice nuestro código podría cambiar el tipo a uno diferente.
- Siempre tendremos que recordar qué argumentos pasarle a cada función y en cual orden y esto se vuelve difícil si una función depende de 3 o más parámetros.
- Si quisiéramos crear un segundo vehículo entonces ¿Tendríamos que crear las variables $color2, $tipo2? Y si queremos un tercer vehículo.

Programación Orientada a Objetos

Con las clases y con la POO resolvemos los anteriores problemas, pero es más, ganamos en abstracción, mantenimiento, seguridad y legibilidad de código. Vemos cómo se haría con una clase.

```php
<?php
    class ClaseCoche {
        // Declaración de una propiedad
        public $color = 'Verde';
        public $tipo = 'turismo';
        // Declaración de un método
        public function mostrarColor() {
            echo '<br>';
            echo 'El color del coche es:';
            echo $this->color; echo '<br>';
        }
         public function getColor(){
            return $this->color;
         }
         public function mostrarTipo() {
            echo $this->tipo;
         }
    }
?>
```

Programación Orientada a Objetos

Ejercicio

Tras iniciarse en la definición de clases, vamos a generar una primera clase que nos modele un equipo de baloncesto:

1) Generamos la estructura de ficheros dentro con una nueva carpeta y un documento php denominado **equipo.php**
2) Crearemos una clase utilizando la palabra reservada class, y denominando a la clase **Equipo**
3) Definiremos como única propiedad el **nombre** de equipo.

Programación Orientada a Objetos

include

include: La sentencia include incluye y evalúa el archivo especificado. Los archivos son incluidos con base en la ruta de acceso dada o, si ninguna es dada, el include_path especificado.

Si el archivo no se encuentra en el include_path, include finalmente verificará en el propio directorio del script que hace el llamado y en el directorio de trabajo actual, antes de fallar. El constructor include emitirá una advertencia si no puede encontrar un archivo, éste es un comportamiento diferente al de require, el cual emitirá un error fatal..

Programación Orientada a Objetos

include

Como vemos, la función include nos permite incluir el código definido en otro fichero, lo cual nos permite por lo tanto dividir nuestro código mediante librerías reutilizables a lo largo de nuestra aplicación.

vars.php

```php
<?php
    $color = 'verde';
    $fruta = 'manzana';
?>
```

```php
<?php
    echo "Una $fruta $color";
    include 'vars.php';
    echo "Una $fruta $color"; // Una manzana verde
?>
```

Programación Orientada a Objetos

include vs. require

Las cuatro estructuras que disponemos para poder incluir código desde otro fichero serían:

- **include**, incluye y evalúa el archivo especificado.
- **include_once**, la sentencia include_once incluye y evalúa el fichero especificado durante la ejecución del script. Tiene un comportamiento similar al de la sentencia include, siendo la única diferencia de que si el código del fichero ya ha sido incluido, **no se volverá a incluir**, e include_once devolverá TRUE. Como su nombre indica, el fichero será incluido **solamente una vez**.

Programación Orientada a Objetos

include vs. require

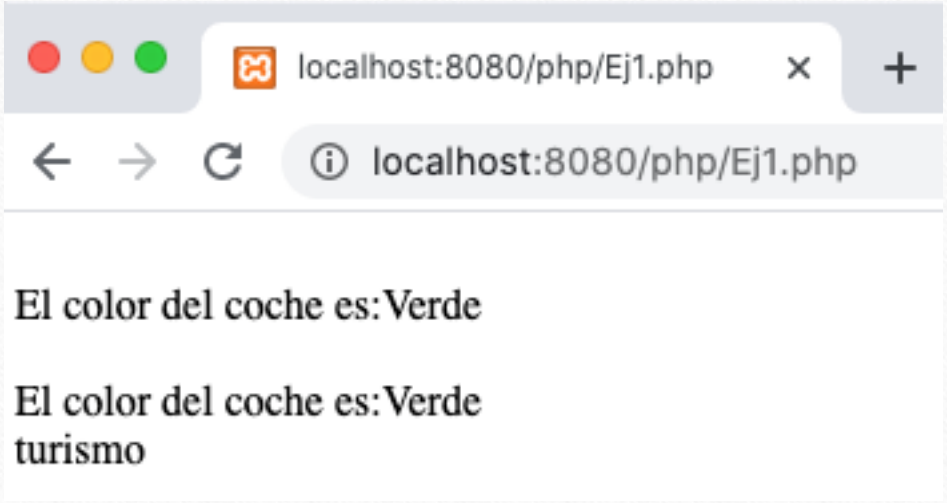
require:

- **require**, require es idéntico a include excepto que en caso de fallo producirá un error fatal de nivel E_COMPILE_ERROR. En otras palabras, éste **detiene el script** mientras que include sólo emitirá una advertencia (E_WARNING) lo cual permite continuar el script.
- **require_once**, la sentencia require_once es idéntica a require excepto que PHP verificará si el archivo ya ha sido incluido y si es así, no se incluye (require) de nuevo.

Programación Orientada a Objetos

Creación de objetos

Una vez que hemos creado nuestra clase, estamos preparados para poder usar y utilizar las clases.




```php
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <?php
            include 'ClaseCoche.php';
            $coche1 = new ClaseCoche();
            $coche1->getColor();
            $coche1->mostrarColor();
            $coche2 = new ClaseCoche();
            $coche2->mostrarColor();
            $coche2->mostrarTipo();
        ?>
    </body>
</html>
```

Programación Orientada a Objetos

Ejercicio

Continuaremos con el ejemplo del equipo de baloncesto antes definido:

1) Dentro de la estructura de ficheros antes generada, crearemos un segundo fichero denominado **liga.php**
2) Incluiremos la clase equipo.php mediante un **require**
3) Crearemos **dos equipos** mediante la palabra reservada new, con dos nuevos objetos que se denominen UNCBasket y RMDBasket

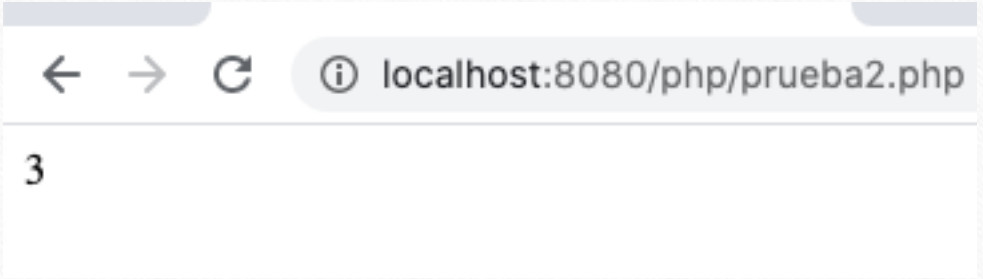
Programación Orientada a Objetos

Características de la POO en PHP – Propiedades o atributos

Las variables pertenecientes a una clase se llaman "propiedades", "atributos" o "campos".

```php
<?php
    class ClaseSencilla {
        public $var1 = 'hola ' . 'mundo';
        public $var2 = 1+2;
        public $var3 = array(true, false);
    }
?>
```

```php
<?php
    include 'clasesencilla.php';
    $o = new ClaseSencilla();
    echo $o->var2;
?>
```



Programación Orientada a Objetos

Características de la POO en PHP – Funciones o métodos

Una función, esté definida o no dentro de una clase, tiene la siguiente estructura:

```php
<?php
    function foo($arg_1, $arg_2, /* ..., */ $arg_n)
    {
        echo "Función de ejemplo.\n";
        return $valor_devuelto;
    }
?>
```

Programación Orientada a Objetos

Características de la POO en PHP – Funciones o métodos

Características también diferenciales que observamos en la creación y utilización de las funciones en php:

- No necesitamos identificar el **tipo** de los argumentos de entrada, al igual que el resto de variables dentro del lenguaje y tal y como hemos descrito anteriormente, PHP es un lenguaje muy poco tipificado.
- No necesitamos indicar si la función **devuelve o no** valores o resultados, por lo que también será opcional el uso de return.

Programación Orientada a Objetos

Ejercicio

Ampliaremos nuestro ejemplo de equipo de baloncesto añadiendo las siguientes propiedades y métodos a la clase:

1) Utilizaremos la clase antes definida **equipo.php**.
2) Añadiremos una nueva propiedad de tipo público denominado **posición**.
3) Añadiremos una nueva función denominada **mostrarEquipo** y que realice un echo de la propiedad nombre.

Programación Orientada a Objetos

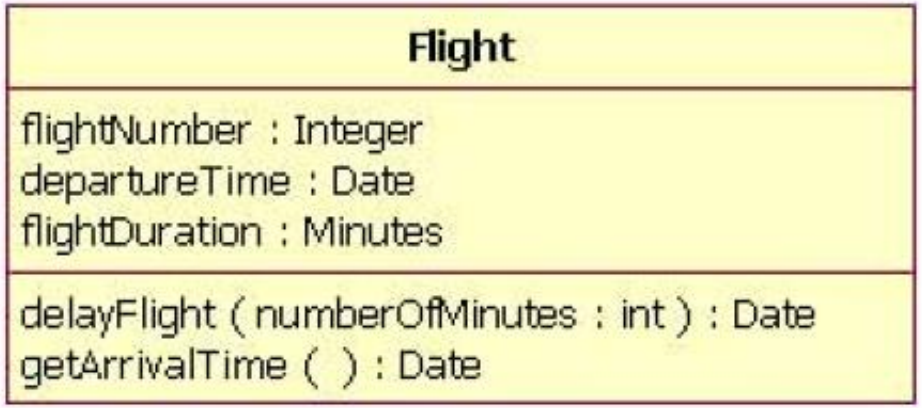
Ejercicio

4) Añadiremos una nueva función denominada **ponerEquipo** y que modifique la propiedad nombre de la clase a partir del parámetro introducido.

5) Por último, dentro de nuestro fichero liga.php, usaremos la función **ponerEquipo** para definir el nombre del equipo y **mostrarEquipo** para mostrarlo.

Programación Orientada a Objetos

Diagrama de clases



Programación Orientada a Objetos

$this

Esta pseudovariable se utiliza dentro de cualquier clase/objeto para hacer **referencia a la propia clase** (decimos clase ya que está definida en la clase, pero sólo tiene sentido cuando se crea el objeto)

```php
<?php
class ClaseSencilla{
    public $var = 'un valor';

    public function mostrarVar(){
        echo $this->var;
    }

}
?>
```

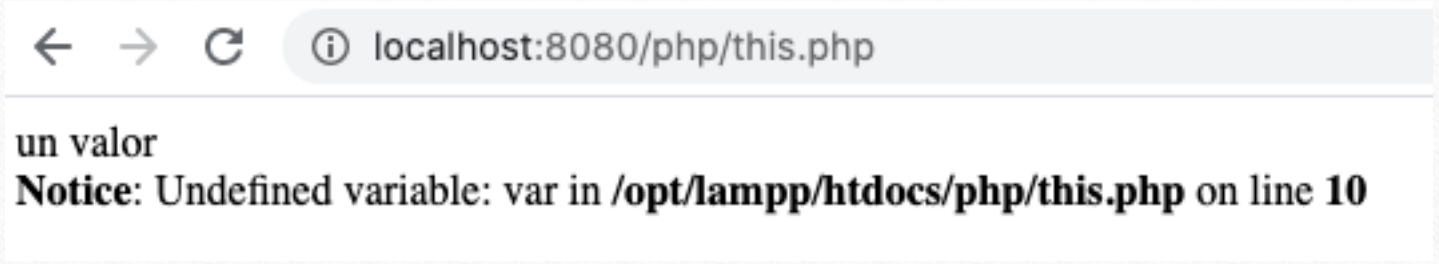
Programación Orientada a Objetos

Diagrama de clases

```php
<?php
class ClaseSencilla{
    public $var = 'un valor';

    public function mostrarVar(){
        echo $this->var;
    }

    public function mostrarVar2(){
        echo $var;
    }
}
?>
```

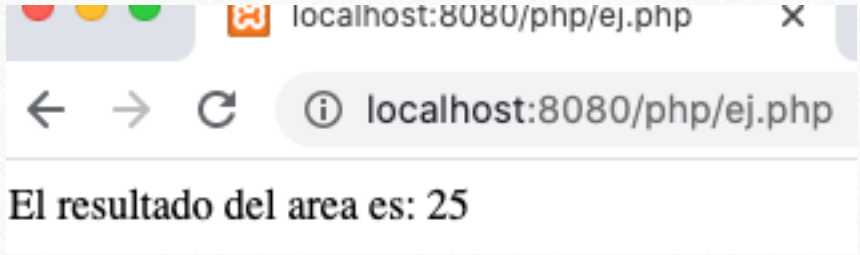


Programación Orientada a Objetos

Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos.

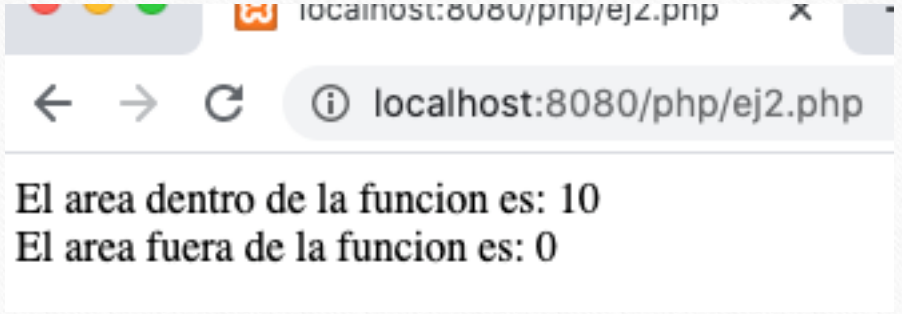
```php
<?php
//variable de calculo de areas
    $area=0;
    $lado=5;
?>
<div class="resultado">
    <?php //area del cuadrado
    $area=$lado*$lado; ?>
<?="El resultado del area es: ".$area?>
```



Programación Orientada a Objetos

Ámbito de las variables

```php
<?php
    //variable de calculo de areas
    $area=0;
    $lado=5;
    //Definimos el calculo de areas
    function calcularAreaCuadrado(){
        $area=10;
        echo "El area dentro de la funcion es: ".$area."<br>";
    }
    calcularAreaCuadrado();
    echo "El area fuera de la funcion es: ".$area;
?>
```



Programación Orientada a Objetos

Visibilidad dentro de las clases

La visibilidad nos permite definir dónde y quien puede usar un atributo o un método. La visibilidad es un concepto dentro de orientación a objetos que permite emplear u "ocultar" variables y funciones de objetos dependiendo de las necesidades del desarrollo.

Programación Orientada a Objetos

Visibilidad dentro de las clases

La visibilidad de una propiedad, un método o (a partir de PHP 7.1.0) una constante se puede definir anteponiendo a su declaración una de las palabras reservadas **public**, **protected** o **private**.

A los miembros de clase declarados como '**public**' se puede acceder desde donde sea; a los miembros declarados como '**protected**', solo desde la misma clase o mediante clases heredadas. A los miembros declarados como '**private**' únicamente se puede acceder desde la clase que los definió.

Programación Orientada a Objetos

Getter

Una función de tipo get, es aquella que sólo devuelve un parámetro normalmente un atributo y que no recibe ningún parámetro. Y digo normalmente ya que conforme vayamos avanzando en nuestros desarrollos, los métodos y sus casuísticas se complican.

Programación Orientada a Objetos

Setter

En este caso la función o método recibe una única variable cuyo cometido es cambiar el contenido de una propiedad del objeto.

Programación Orientada a Objetos

Ejercicio

Ampliaremos nuestro ejemplo de equipo de baloncesto añadiendo las siguientes propiedades y métodos a la clase:

1) Utilizaremos la clase antes definida equipo.php.
2) Definiremos el **getter** y el **setter** de la propiedad **posición**.
3) Comprobaremos la correcta funcionalidad de estos métodos dentro del fichero liga.php

Programación Orientada a Objetos

Visibilidad y setters

El uso de private dentro de un objeto tiene mucho sentido cuando estamos "setteando" una variable, ya que si una variable es pública no podemos controlar al 100% qué valor le va a colocar el usuario, mientras que cuando la variable es privada y su acceso se realiza mediante el setter, el control es total.

Programación Orientada a Objetos

Visibilidad y funciones

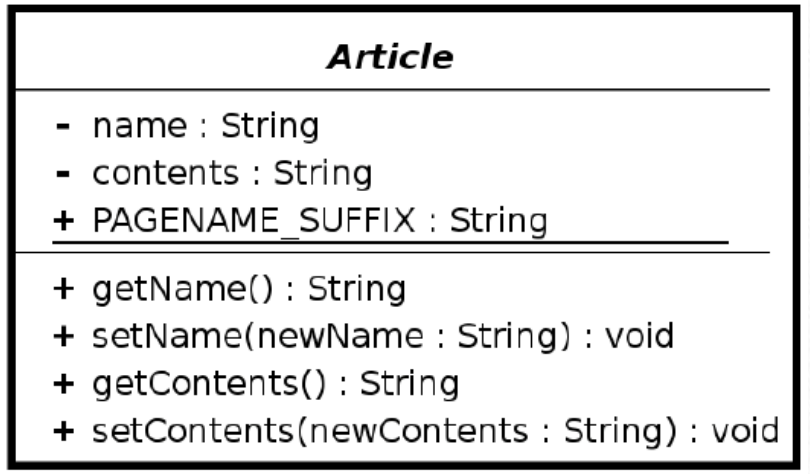
De igual forma que hemos visto que en las propiedades de una clase podemos utilizar los calificadores de visibilidad public y private, podemos aplicarlo a la definición de los métodos o funciones. Pero la pregunta en este caso es ¿para qué definiríamos un método privado?

Programación Orientada a Objetos

Diagrama de clases y visibilidad

Una vez vistos los modificadores de visibilidad dentro de las clases, es interesante ver cómo se pueden representar y utilizar con los diagramas de clase UML ya que es muy sencillo y nos proporciona una visualización rápida de la definición de la visibilidad de propiedades y métodos.

Símbolo	Descripción
+	Público
-	Privado
#	Protegido



Programación Orientada a Objetos

Constructor

En cualquier lenguaje y dentro de la definición de las clases existe una función especial y sumamente importante que se denomina constructor. Decimos que es un método/función especial ya que por un lado es un método, pero por otro no sólo se puede denominar de una forma concreta.

```php
<?php
    class BaseClass {
        function __construct() {
            print "En el constructor BaseClass\n";
        }
    }
    // En el constructor BaseClass
    $obj = new BaseClass(); //Sacará por pantalla, En el constructor BaseClass
?>
```

Programación Orientada a Objetos

Ejercicio

Por último definiremos el **constructor** en nuestro ejemplo usado a lo largo de la unidad:

1) Utilizaremos la clase antes definida equipo.php.
2) Definiremos el constructor **inicializando** las propiedades nombre a "Equipo sin nombre" y posición a 0
3) Comprobaremos la **correcta funcionalidad del constructor** a través de los getter dentro del fichero liga.php