



PHP - POO



Ciclo: DAW
Módulo: DWES
Curso: 2020-2021
Autor: César Guijarro Rosaleny



Introducción.....	3
Elementos de la POO.....	4
Clases.....	4
Objetos.....	5
La pseudovariable \$this.....	6
Constructores y destructores.....	7
Propiedades estáticas.....	9
Métodos estáticos.....	12
Herencia.....	18
Abstracción de clases.....	22
Interfaces.....	24
Palabra clave final.....	25
Bibliografía.....	26

Introducción

A partir de PHP 5, el modelo de objetos ha sido reescrito para tener en cuenta un mejor rendimiento y una mejor funcionalidad. Este fue un cambio importante a partir de PHP 4. PHP 5 tiene un modelo de objetos completo.

Entre las características de PHP 5 están la inclusión de la visibilidad, clases y métodos abstractos y finales, métodos mágicos adicionales, interfaces, clonación y determinación de tipos.

PHP trata los objetos de la misma manera que las referencias o manejadores, lo que significa que cada variable contiene una referencia a un objeto en lugar de una copia de todo el objeto.

Elementos de la POO

Clases

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 constantes, propiedades y métodos pertenecientes a dicha clase.

```
<?php

class ClassName
{
    //Constantes
    public const BAR = "BAR";

    //Propiedades
    public $var = "valor predeterminado";

    //Métodos
    public function funcionClase() {

    }
}
```

En la definición de las constantes, propiedades o métodos tenemos que definir la visibilidad usando una de las palabras reservadas **public**, **protected** o **private** (según la visibilidad sea pública, protegida o privada).

NOTA: Por defecto la visibilidad será pública.

Objetos

Para crear una instancia de una clase, se debe emplear la palabra reservada **new** seguida del nombre de la clase y dos paréntesis (en realidad se le pueden pasar valores a las clases dentro de esos paréntesis. Lo veremos más adelante cuando vemos los constructores).

Las clases deberían ser definidas antes de la instanciación (y en algunos casos esto es un requerimiento).

```
<?php

class ClaseA
{
    public $var1 = 5;

    public function hola() {
        echo "Hola mundo";
    }
}

$objeto1 = new ClaseA();
```

Una vez hemos creado el objeto, podemos acceder a sus propiedades y métodos mediante el operador flecha (→).

```
$objeto1 = new ClaseA();
echo($objeto1->var1);
$objeto1->hola();
```

La pseudovariable **\$this**

La pseudovariable **\$this** está disponible cuando un método es invocado dentro del contexto de un objeto. **\$this** es una referencia al objeto invocador (usualmente el objeto al cual el método pertenece, aunque puede que sea otro objeto si el método es llamado estáticamente desde el contexto de un objeto secundario).

```
<?php

class ClaseA
{
    public $var1 = 5;

    public function getVar1() {
        echo $this->var1;
    }
}

$objeto1 = new ClaseA();
$objeto1->getVar1();
```

En el ejemplo anterior, si queremos acceder a la variable `$var1` dentro del método `getVar1()` debemos utilizar **\$this** para indicarle a PHP que la variable que debe leer es la que está definida en la clase. Si cambiamos la línea por `echo $var1` nos dará un error, ya que la variable `$var1` no está definida en el método.

\$this no sólo se utiliza para acceder a las propiedades de la clase, también lo usaríamos para ejecutar los métodos de la misma.

Constructores y destructores

A partir de PHP 5 se pueden declarar métodos constructores para las clases. Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que lo hace idóneo para cualquier inicialización que el objeto pueda necesitar antes de ser usado.

Para declarar un constructor, utilizamos el **método mágico `__construct()`** (los métodos mágicos los provee PHP y nos permiten realizar ciertas tareas OO. Se identifican por el uso de dos guiones bajos “`__`” como prefijo).

```
<?php

class ClaseA
{
    public $var1;
    public $var2;

    function __construct($var1, $var2) {
        $this->var1 = $var1;
        $this->var2 = $var2;
    }
}

$objeto1 = new ClaseA(5, 6);
echo "$objeto1->var1, $objeto1->var2";
```

También podemos definir **métodos destructores** con el **método mágico `__destruct()`**.

```
<?php

class ClaseA
{
    public $var1;
    public $var2;

    function __construct($var1, $var2) {
        $this->var1 = $var1;
        $this->var2 = $var2;
    }

    function __destruct() {
        echo "Objeto destruido";
    }
}

$objeto1 = new ClaseA(5, 6);
echo "$objeto1->var1, $objeto1->var2";
```

El método destructor será llamado **tan pronto como no hayan otras referencias a un objeto determinado**, o en cualquier otra circunstancia de finalización.

NOTA: Ésto último puede llevar a confusión. En el ejemplo anterior en ningún momento se borra el objeto ni se iguala a NULL ni nada parecido, sin embargo se ejecutará su método destructor después de mostrar por pantalla el valor de las variables, ya que no hay más referencias al objeto en el código del programa.

Propiedades estáticas

Declarar propiedades estáticas las hacen accesibles **sin la necesidad de instanciar una clase**. Una propiedad estática **no puede ser accedida con un objeto de clase instanciada** (aunque un método estático si lo puede hacer).

Para definir una propiedad como estática, anteponemos la palabra **static** al nombre de la variable.

```
<?php  
  
class ClaseA  
{  
    static public $var1 = 5;  
}  
  
echo ClaseA::$var1;
```

Para acceder al valor de la variable estática definida utilizamos el nombre de la clase seguido de doble dos puntos (::) y el nombre de la variable.

No se pueden acceder a las propiedades estáticas a través del objeto utilizando el operador flecha (→).

```
<?php

class ClaseA
{
    static public $var1 = 5;

    public function getVar1() {
        echo $this->var1;
    }
}

$objeto1 = new ClaseA();
$objeto1->getVar1();
```

(!) Notice: Accessing static property ClaseA::\$var1 as non static in /var/www/html/DWS/clases/T4/clase05.php on line 15				
Call Stack				
#	Time	Memory	Function	Location
1	0.1304	399456	{main}()	.../clase05.php:0
2	0.1304	399496	ClaseA->getVar1()	.../clase05.php:20

(!) Notice: Undefined property: ClaseA::\$var1 in /var/www/html/DWS/clases/T4/clase05.php on line 15				
Call Stack				
#	Time	Memory	Function	Location

A las propiedades estáticas se puede acceder utilizando la palabra reservada **self** seguido de doble dos puntos y el nombre de la variable.

```
<?php  
  
class ClaseA  
{  
    static public $var1 = 5;  
  
    public function getVar1() {  
        echo self::$var1;  
    }  
}  
  
$objeto1 = new ClaseA();  
$objeto1->getVar1();
```

← → ↻ ⓘ localhost:8080/DWS/clases/T4/clase05.php

5

Métodos estáticos

Como ocurre con las propiedades, podemos declarar métodos estáticos en nuestras clases añadiendo la palabra reservada **static** a la declaración del método.

```
<?php

class ClaseA
{
    static public function hola() {
        echo "Hola mundo";
    }
}

ClaseA::hola();
```

Debido a que los métodos estáticos se pueden invocar sin tener creada una instancia del objeto, **la pseudovariable `$this` no está disponible dentro de los métodos declarados como estáticos.**

```
<?php

class ClaseA
{
    public $var1 = 5;

    static public function getVar1() {
        echo $this->var1;
    }
}

ClaseA::getVar1();
```

(!) Fatal error: Uncaught Error: Using \$this when not in object context in /var/www/html/DWS/clases/T4/clase06.php on line 17				
(!) Error: Using \$this when not in object context in /var/www/html/DWS/clases/T4/clase06.php on line 17				
Call Stack				
#	Time	Memory	Function	Location
1	0.2350	399264	{main}()	.../clase06.php:0
2	0.2351	399264	ClaseA::getVar1()	.../clase06.php:21

Para acceder a la variable anterior deberíamos utilizar **self**, aunque **también deberíamos declarar la variable como estática**, ya que si no seguiría dando error.

```
<?php

class ClaseA
{
    static public $var1 = 5;

    static public function getVar1() {
        echo self::$var1;
    }
}

ClaseA::getVar1();
```

← → ↻ ⓘ localhost:8080/DWS/cla

5

Aunque en principio no se puede llamar a un método no estático de forma estática (y viceversa), PHP nos proporciona dos métodos mágicos que podemos aprovechar para poder hacerlo: `__callStatic()` y `__call()`.

Vamos a ver un ejemplo. Crea un fichero llamado `estatico.php` en la carpeta `apache_htdocs/DWS/ejemplos/T4` con el siguiente contenido:

```
<?php


class ClaseA
{
    private $var1;

    public function setVar1($var1) {
        $this->var1 = $var1;
    }

    public function getVar1() {
        return $this->var1;
    }
}

ClaseA::setVar1(3);
echo "El valor de la variable es " . ClaseA::getVar1();
```

Obviamente, si intentamos ejecutar el programa nos dará un error, ya que estamos llamando a dos métodos no estáticos de forma estática.


 Deprecated: Non-static method ClaseA::setVar1() should not be called statically in /var/www/html/DWS/ejemplos/T4/estatico.php on line 17				
Call Stack				
#	Time	Memory	Function	Location

Ahora vamos a añadir el método mágico `__callStatic()`. Este método acepta dos parámetros, el nombre del método no estático llamado y sus argumentos. En principio nos vamos a olvidar de ellos (después veremos como utilizarlos).

```
<?php

public static function __callStatic($name, $arguments) {
    echo "Llamando a método no estático de forma estática";
}
```

El método recién creado se ejecutará siempre que llamemos a un método no estático de forma estática, con lo que en principio debería mostrarnos por pantalla la frase “Llamando a método no estático de forma estática” 2 veces. Pero si ejecutas el programa te seguirá dando el mismo error:

 Deprecated: Non-static method ClaseA::setVar1() should not be called statically in /var/www/html/DWS/ejemplos/T4/estatico.php on line 17				
Call Stack				
#	Time	Memory	Function	Location

El problema es que los métodos `setVar1()` y `getVar1()` son públicos, y al llamarlos no se ejecuta el método mágico, ya que están accesibles directamente. Cambia la visibilidad de ambos métodos a privado y comprueba el resultado:

```
Llamando a método no estático de forma estática
Llamando a método no estático de forma estática
El valor de la variable es
```

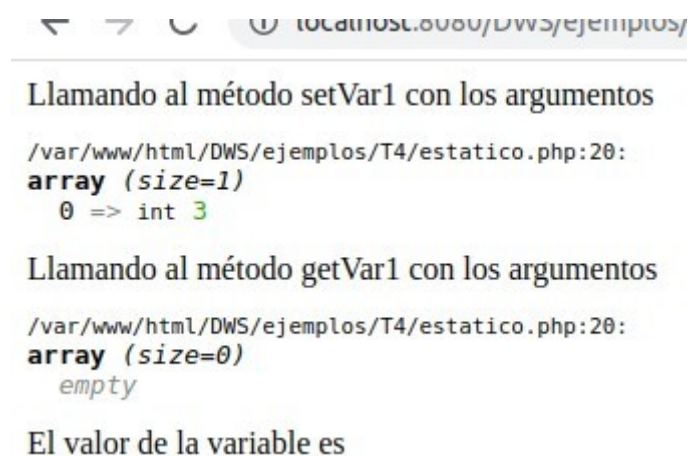
Ahora sí que se ejecuta el método mágico sin problemas.

Una vez comprobado que se ejecuta el método mágico, cambia su código por el siguiente (Lo que hace la última línea de código es utilizar enlaces estáticos en tiempo de ejecución para ejecutar los métodos llamados, aunque no te preocupes si no entiendes del todo el código):

```
<?php

public static function __callStatic($name, $arguments) {
    echo "Llamando al método $name con los argumentos";
    var_dump($arguments);
    return (new static)->$name(...$arguments);
}
```

Si ejecutas el código ahora deberías ver algo parecido a esto:



```

Llamando al método setVar1 con los argumentos
/var/www/html/DWS/ejemplos/T4/estatico.php:20:
array (size=1)
    0 => int 3

Llamando al método getVar1 con los argumentos
/var/www/html/DWS/ejemplos/T4/estatico.php:20:
array (size=0)
    empty

El valor de la variable es
```

Como ves, ya no da ningún error y se ejecuta los métodos `setVar1()` y `getVar1()` de forma estática.

El problema es que sigue sin mostrarnos el valor de la variable. Eso se debe a que está definida de forma no estática, con lo que no podemos acceder a ella de forma estática.

Para que funcione todo correctamente, debemos cambiar la variable a estática.

```
<?php

class ClaseA
{
    static private $var1;

    private function setVar1($var1) {
        self::$var1 = $var1;
    }

    private function getVar1() {
        return self::$var1;
    }

    public static function __callStatic($name, $arguments) {
        echo "Llamando al método $name con los argumentos";
        var_dump($arguments);
        return (new static)->$name(...$arguments);
    }
}

ClaseA::setVar1(3);
echo "El valor de la variable es " . ClaseA::getVar1();
```

Si ejecutas el programa ahora deberías ver el valor de la variable sin ningún problema:

Llamando al método setVar1 con los argumentos

```
/var/www/html/DWS/ejemplos/T4/estatico.php:21:
array (size=1)
    0 => int 3
```

Llamando al método getVar1 con los argumentos

```
/var/www/html/DWS/ejemplos/T4/estatico.php:21:
array (size=0)
    empty
```

El valor de la variable es 3

Herencia

La **herencia** es un principio de programación bien establecido y PHP hace uso de él en su modelado de objetos. Este principio afectará la manera en que muchas clases y objetos se relacionan unas con otras.

Por ejemplo, cuando se extiende una clase, la subclase **hereda todos los métodos públicos y protegidos de la clase padre**. A menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original.

Para indicar que una clase hereda de otra, se utiliza la palabra reservada **extends**.

```
<?php

class ClaseA {
    public $var1 = 5;
}

class ClaseB extends ClaseA {
    public $var2 = 10;
}

$objeto1 = new ClaseB();
echo "Var1 = $objeto1->var1, Var2 = $objeto1->var2";
```

Una cosa a tener en cuenta cuando usamos herencia es que los constructores padres (y los destructores padres) no son llamados implícitamente **si la clase hija define un constructor**.

Si el hijo no define un constructor, entonces se puede heredar de la clase madre como un método de clase normal (si no fue declarada como privada).

Por ejemplo, crea el archivo *herencia* en la carpeta *apache_htdocs/DWS/ejemplos/T4* y copia el siguiente contenido:

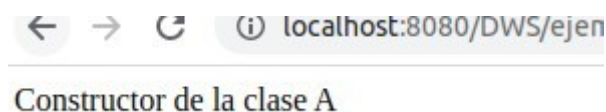
```
<?php

class ClaseA
{
    function __construct() {
        echo "Constructor de la clase A<br>";
    }
}

class ClaseB extends ClaseA
{
}

$objeto1 = new ClaseB();
```

Si lo ejecutas, la salida mostrará:



← → ↻ ⓘ localhost:8080/DWS/ejen

Constructor de la clase A

Ahora define un constructor en la clase B:

```
<?php

class ClaseA
{
    function __construct() {
        echo "Constructor de la clase A<br>";
    }
}

class ClaseB extends ClaseA
{
    function __construct() {
        echo "Constructor de la clase B";
    }
}

$objeto1 = new ClaseB();
```

Al abrir el archivo en un navegador verás que sólo ejecuta el constructor de la clase B:



Para ejecutar un constructor padre, se requiere invocar a **parent::__construct()** desde el constructor hijo.

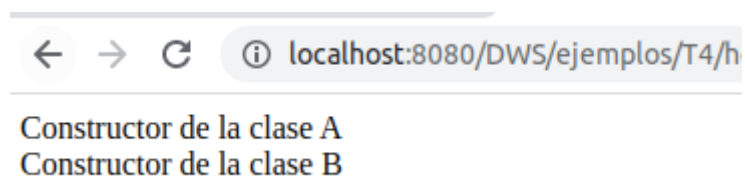
```
<?php

class ClaseA
{
    function __construct() {
        echo "Constructor de la clase A<br>";
    }
}

class ClaseB extends ClaseA
{
    function __construct() {
        parent::__construct();
        echo "Constructor de la clase B";
    }
}

$objeto1 = new ClaseB();
```

Si compruebas la salida ahora, ves que al crear el objeto ya se ejecutan los dos constructores:



Abstracción de clases

A partir de PHP 5 podemos definir **clases y métodos abstractos**. Las clases definidas como abstractas **no se pueden instanciar y cualquier clase que contiene al menos un método abstracto debe ser definida como tal**. Los métodos definidos como abstractos simplemente declaran la firma del método, pero **no pueden definir la implementación**.

Cuando se hereda de una clase abstracta, **todos los métodos definidos como abstractos en la declaración de la clase madre deben ser definidos en la clase hija (siempre que esta, a su vez, no sea abstracta)**; además, estos métodos deben ser definidos con la misma (o con una menos restrictiva) visibilidad.

Por ejemplo, **si el método abstracto está definido como protegido, la implementación de la función debe ser definida como protegida o pública, pero nunca como privada**.

Por otra parte, **las firmas de los métodos tienen que coincidir**, es decir, la declaración de tipos y el número de argumentos requeridos deben ser los mismos.

Por ejemplo, si la clase derivada define un argumento opcional y la firma del método abstracto no lo hace, no habría conflicto con la firma. Esto también se aplica a los constructores a partir de PHP 5.4. Antes de PHP 5.4, las firmas del constructor podían ser diferentes.

Para definir una clase (o método) como abstracta, se utiliza la palabra reservada **abstract**.

```
<?php

abstract class ClaseA
{
    abstract public function hola();

    public function hola2() {
        echo "Hola universo<br>";
    }
}

class ClaseB extends ClaseA
{
    public function hola() {
        echo "Hola mundo<br>";
    }
}

$objeto1 = new ClaseB();
$objeto1->hola();
$objeto1->hola2();
```

Hola mundo
Hola Universo

Interfaces

En PHP también podemos definir interfaces. Las interfaces se definen de la misma manera que una clase, aunque reemplazando la palabra reservada **class** por la palabra reservada **interface** y sin que ninguno de sus métodos tenga su contenido definido.

Todos los métodos declarados en una interfaz deben ser públicos, ya que ésta es la naturaleza de una interfaz.

Para implementar una interfaz, se utiliza el operador **implements**. Las clases pueden implementar más de una interfaz si se deseara, separándolas cada una por una coma.

```
<?php

interface Interface1 {
    public function hola();
    public function hola2();
}

class ClaseA implements Interface1 {
    public function hola() {
        echo "Hola mundo";
    }

    public function hola2() {
        echo "Hola universo";
    }
}

$objeto1 = new ClaseA();
$objeto1->hola();
$objeto1->hola2();
```


Palabra clave final

PHP 5 introdujo la nueva palabra clave **final**, que impide que las clases hijas sobrescriban un método,. Si la propia clase se define como final, entonces no se podrá heredar de ella.

```
<?php

class ClaseA
{
    final public function hola() {
        echo "Hola mundo";
    }
}

class ClaseB extends ClaseA
{
    public function hola() {
        echo "Hola mundo<br>";
    }
}

$objeto1 = new ClaseB();
$objeto1->hola();
```

(!) Fatal error: Cannot override final method ClaseA::hola() in /var/www/html/DWS/clases/T4/clase10.php on line 12

Bibliografía

<https://www.php.net/manual/es/index.php>