línea horizontal

## Producción web

Profesor: Fernando Gonzalo Gaitán  
Carrera: Analista de sistemas  
Año: 2024

Primera escuela de arte multimedial Da Vinci

Programación orientada a objetos con PHP - Parte 1

[Producción web](#_heading=h.gjdgxs) 1

[**¿Qué es un objeto en PHP?**](#_heading=h.zekg7rf06fr6) **1**

[¿Cómo crear una clase en PHP?](#_heading=h.8jxlutdt4z6d) 2

[¿Cómo instanciar un objeto en PHP?](#_heading=h.5mm589n5ezvs) 3

[**Constructores y destructores**](#_heading=h.ymwoj5kbqugj) **4**

[¿Qué es un constructor?](#_heading=h.c2ov9d7mr8mz) 4

[¿Qué es un destructor?](#_heading=h.jev4bymowdwr) 5

[**Visibilidad**](#_heading=h.u3g05bq2a841) **6**

[Métodos \_\_get() y \_\_set()](#_heading=h.q85wt0l1crt5) 8

[**Clases abstractas y herencia**](#_heading=h.41o3ccfhlczm) **11**

[Clase abstracta](#_heading=h.6cp3ok7jlqu9) 11

[Herencia](#_heading=h.sxhzithiuka6) 12

[Sobrescribir métodos padres](#_heading=h.lk2hmtfjpc8h) 14

# ¿Qué es un objeto en PHP?

En PHP se entiende a un objeto como a una entidad compuesta por propiedades (información del objeto) y métodos (comportamientos del mismo)

Un objeto es el resultado de la instancia de una clase, donde se definen las características de éste.

Por ejemplo, podríamos decir que una clase podría ser el plano de un arquitecto para diseñar casas, y el objeto justamente una casa creada en base a ese plano. Otro ejemplo podría ser un molde para hacer galletas, y los objetos las galletas que se elaboran con ese molde.

Los objetos tienen característica:

* Identidad: La identidad permite diferenciar a un objeto de otro. Podemos tener una clase Lámpara y dos instancias de ésta, por ejemplo en una instancia el color del foco es blanco y en otra de color naranja.
* Comportamiento: Tiene que ver con la funcionalidad de un objeto, y eventualmente esta operación puede modificar información propia del mismo objeto. Por ejemplo si tuviésemos una propiedad encendido con el valor true y un método apagar(), éste último cambiará el valor de la propiedad encendido a false.
* Estado: Es el conjunto de propiedades y sus valores en un momento dado. Siguiendo con el ejemplo de la clase Lámpara, podemos tener un objeto con las propiedades encendida = true y color = ”Blanco”, ese sería un estado, si luego apagamos la lámpara y le cambiamos el foco por otro de color naranja, entonces quedaría encendida = false y color “Naranja”, entonces decimos que el estado del objeto cambió.

## 

## ¿Cómo crear una clase en PHP?

Como mencionamos anteriormente una clase nos sirve para modelar objetos:

class Lampara

{

//Propiedades y métodos de la clase

}

Dentro de la clase debemos incluir las propiedades y métodos de la misma.

Las propiedades se escriben con el signo $ antes del nombre igual que cualquier variable de PHP, aunque con la palabra reservada public, protected o private, que corresponde a la visibilidad (más adelante veremos de qué se trata esto):

class Lampara

{

public string $marca;

public string $color;

public bool $encendida = false;

}

Los métodos se escriben igual que las funciones, aunque también tenemos que agregar la visibilidad:

class Lampara

{

public string $marca;

public string $color;

public bool $encendida = false;

public function encender():void

{

$this->encendida = true;

}

public function apagar():void

{

$this->encendida = false;

}

}

Notar que para acceder a una propiedad dentro de una clase, tenemos que usar la palabra reservada $this y una especie de flecha:

$this->encendida = true;

## ¿Cómo instanciar un objeto en PHP?

Un objeto se escribe como cualquier variable, con el signo $ y el nombre igual a la palabra reservada new y la clase:

$lampara = new Lampara();

Para llamar a una propiedad o a un método desde el objeto debemos usar la flecha:

$lampara = new Lampara();

$lampara->color = 'Blanca';

$lampara->encender();

if($lampara->encendida){

echo 'La lámpara está encendida y es de color: ' . $lampara->color;

}

# Constructores y destructores

## ¿Qué es un constructor?

Un constructor es un método que se llama al crear un objeto. Convenientemente éste, al igual que cualquier método, puede recibir parámetros, lo que permite definir por ejemplo los valores iniciales de una o más propiedades:

class Lampara

{

public $marca;

public $color;

public $encendida;

public function \_\_construct($p\_marca, $p\_color, $p\_encendida)

{

$this->marca = $p\_marca;

$this->color = $p\_color;

$this->encendida = $p\_encendida;

}

}

Como vemos en el ejemplo usamos la palabra reservada “\_\_construct” para indicar que ése es el constructor:

Antiguamente en PHP se podía usar el nombre de la clase como constructor, al igual que pasa en otros lenguajes de programación como Java, pero esto ya no es posible desde la versión 7.

Si quisiéramos instanciar esta clase deberíamos hacerlo de la siguiente manera:

$lampara = new Lampara('Philips', 'Azul', true);

También podemos definir parámetros con valores por defecto, es decir, que podríamos hacer algo como esto:

public function \_\_construct($p\_marca, $p\_color, $p\_encendida=false)

{

$this->marca = $p\_marca;

$this->color = $p\_color;

$this->encendida = $p\_encendida;

}

El parámetro $p\_encendida por defecto es false, esto quiere decir que si no indicamos un valor, ese último parámetro será false:

Podemos hacer esto:

$lampara = new Lampara('Philips', 'Azul', true);

O bien esto:

$lampara = new Lampara('Philips', 'Azul');

Ignorando el último parámetro.

## 

## ¿Qué es un destructor?

A diferencia del constructor, podemos definir un método destructor el cual se va ejecutar al terminar el ciclo de vida del objeto.

class Lampara

{

public $marca;

public $color;

public $encendida;

public function \_\_construct($p\_marca, $p\_color, $p\_encendida=false)

{

$this->marca = $p\_marca;

$this->color = $p\_color;

$this->encendida = $p\_encendida;

}

public function \_\_destruct()

{

echo 'LA LÁMPARA SE ROMPIÓ';

}

}

# Visibilidad

Tanto las propiedades como los métodos tienen tres formas de definir su nivel de acceso:

* public: Se puede acceder desde cualquier lado.
* private: Sólo se podrá acceder dentro de la clase que la definió.
* protected: Sólo se podrá acceder dentro de la clase que la definió o clases que heredan de la misma (veremos más de esto en herencia)

Es una buena práctica definir las propiedades de una clase como privadas y crear un método getter que nos devuelva el valor de la misma y otro setter que nos permita modificarlo:

class Lampara

{

private $marca;

private $color;

private $encendida;

public function \_\_construct($p\_marca, $p\_color, $p\_encendida=false)

{

$this->marca = $p\_marca;

$this->color = $p\_color;

$this->encendida = $p\_encendida;

}

public function getColor()

{

return $this->color;

}

public function setColor($p\_color)

{

$this->color = $p\_color;

}

}

En primer lugar al ser las propiedades privadas, ya no podremos acceder a ellas; ni verlas, ni modificarlas desde el objeto:

$lampara = new Lampara('Philips', 'Azul');

$lampara->color = 'Rojo';

echo $lampara->color;

Este último código nos devolverá un error.

Si quisiéramos hacer esto tendríamos que usar los métodos getter y setter que creamos para ver y modificar el color de la lámpara:

$lampara = new Lampara('Philips', 'Azul');

$lampara->setColor('Rojo');

echo $lampara->getColor();

## 

## Métodos \_\_get() y \_\_set()

En PHP existe algo llamado métodos mágicos de los cuales ya vimos algunos como \_\_construct() y \_\_destruct()

Si te interesa saber más del tema podés revisar la siguiente info:

<https://www.php.net/manual/es/language.oop5.magic.php>

Entre estos métodos mágicos se encuentran \_\_get() y \_\_set() métodos que nos permitirán recuperar y escribir propiedades inaccesibles o inexistentes.

Por ejemplo:

class MiClase

{

private $propiedad\_privada;

public function \_\_get($name)

{

return $this->name;

}

public function \_\_set($name, $value)

{

$this->name = $value;

}

}

Y luego hacer algo como esto:

$producto = new MiClase();

$producto->privada = 'AAA <br />';

echo $producto->privada;

$producto->no\_existe = 'BBB <br />';

echo $producto->no\_existe;

En ciertas ocasiones puede ser que haya objetos con muchas propiedades, lo cual hace que crear métodos getter() y setter() para cada propiedad termine siendo una tarea tediosa.

Aún así, nunca es una buena idea que nuestras propiedades sean públicas, así como que todas las propiedades tengan un método para mostrar y sobreescribir su valor, sin importar su función.

Una posible solución podría ser la siguiente:

<?php

class Lampara

{

private $data;

public function \_\_construct($p\_marca, $p\_color, $p\_encendida=false)

{

$this->data = array(

'marca' => $p\_marca,

'color' => $p\_color,

'encendida' => $p\_encendida

);

}

public function \_\_get($name)

{

return $this->data[$name];

}

public function \_\_set($name, $value)

{

if(array\_key\_exists($name, $this->data))

{

$this->data[$name] = $value;

}

}

}

Podremos acceder a propiedades privadas como si fuesen públicas, tanto para verlas como para escribirlas:

$lampara = new Lampara('Philips', 'Azul');

echo $lampara->marca . '<br />';

¿Pero qué pasa si tenemos una propiedad a la cual no queremos permitir que pueda ser seteada desde el objeto? Por ejemplo la propiedad marca, no se puede modificar la marca de una lámpara:

$lampara = new Lampara('Philips', 'Azul');

echo $lampara->marca . '<br />';

$lampara->marca = 'Sony';

echo $lampara->marca . '<br />';

En este caso podríamos modificar el método \_\_set de la siguiente forma:

public function \_\_set($name, $value)

{

if(array\_key\_exists($name, $this->data) and $name != 'marca')

{

$this->data[$name] = $value;

}

else

{

throw new Error("No se puede acceder a la propiedad {$name}");

}

}

En este caso al intentar acceder a una propiedad que no es accesible desde el objeto, nos devolverá un error.

# Clases abstractas y herencia

## Clase abstracta

Una clase abstracta es una clase que no se puede instanciar:

abstract class Mascota {

//

}

Como vemos utiliza la palabra reservada abstract delante y al intentar crear un objeto de esta clase nos va a devolver un error:

$mascota = new Mascota();

## Herencia

Una clase puede extender de otra, esto hace que la subclase herede de la clase padre todas sus propiedades y todos sus métodos.

Por ejemplo supongamos que tenemos dos clases, una clase Perro y una clase Gato. Ambas clases tienen algo en común: son mascotas, por tanto tienen la misma información que tiene cualquier mascota: tienen un nombre, una fecha de nacimiento, peso, la lista de vacunas obligatorias; y además comportamientos como dormir, comer, jugar:

abstract class Mascota

{

private $nombre;

public function \_\_construct($p\_nombre)

{

$this->nombre = $p\_nombre;

}

public function jugar()

{

return $this->nombre . ' está jugando <br />';

}

}

class Gato extends Mascota

{

public function maullar()

{

return 'Miau <br />';

}

}

class Perro extends Mascota {

public function ladrar()

{

return 'Guau <br />';

}

}

Podemos crear una instancia del objeto gato:

$gato = new Gato('Lalo');

echo $gato->maullar();

echo $gato->jugar();

Como vemos en el ejemplo, el objeto Gato llama al método maullar(), el cual se define en la clase Gato, pero también al método jugar() que no le pertenece a esa clase, pero sí a la clase Mascota, de la cual hereda.

## 

## Sobrescribir métodos padres

Una subclase puede sobrescribir los métodos de la clase padre. Por ejemplo:

abstract class Mascota

{

private $nombre;

private $vacunas\_obligatorias = [];

public function \_\_construct($p\_nombre, $p\_vacunas\_obligatorias)

{

$this->nombre = $p\_nombre;

$this->vacunas\_obligatorias = $p\_vacunas\_obligatorias;

}

}

class Gato extends Mascota

{

public function \_\_construct($p\_nombre)

{

parent::\_\_construct($p\_nombre, [

'Antirrábica',

'Triple felina'

]);

}

}

Como vemos en el ejemplo, la clase Mascota define en su constructor un parámetro para indicar el nombre y otro para indicar cuáles son las vacunas obligatorias.

La clase gato tiene su propio constructor, con un solo parámetro, el cual reutiliza el constructor padre con la palabra reservada parent y define la lista de vacunas obligatorias de un gato.