

POO en PHP

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

Conceptos básicos

- **Clase:** estructura (similar a una plantilla) de un objeto. En ella se definen los elementos que formarán parte de ella:
 - **Propiedades:** (equivalente a variables en programación estructurada).
 - Almacenan información sobre el objeto.
 - Su valor puede ser diferente para cada uno de los objetos de la misma clase.
 - **Métodos:** define el comportamiento de los objetos (equivalente a las funciones en programación estructurada).
 - Contienen código.
 - Pueden recibir parámetros y devolver valores.
- **Instanciar una clase:** crear un objeto basado en una clase. Se usa la palabra reservada “**new**”.
- **Instancia de una clase:** objeto obtenido al instanciar una clase.

Crear clase en PHP

- En PHP las clases se declaran con la palabra reservada “**class**” seguida del nombre que queramos darle (**siempre conviene buscar un nombre de clase que identifique lo mejor posible a lo que representa**)
- A continuación, entre llaves se definen todos sus atributos y métodos.
- Para definir los métodos, se utiliza la misma sintaxis que las funciones del lenguaje PHP.
- Sintaxis básica:

```
class [Nombre de la Clase] {  
    [atributos]  
    [métodos]  
}
```

Crear clase en PHP

Conviene:

- Definir los miembros de la clase de forma ordenada, primero los atributos y después los métodos.
- Definir cada clase en su propio fichero.
- Los nombres de las clases suelen comenzar por letra mayúscula para distinguirlos de los objetos.

Crear clase en PHP

- Ejemplo de definición de una clase

```
<?php
//fichero Coche.php
class Coche
    //declaración de una propiedad con
    public $tipo="turismo";
    public $color;

    public function mostrarTipo(){
        echo $this->tipo;
    }
?>
```

Observa:

- La clase está definida en su propio fichero.
- El nombre de la clase comienza por una letra mayúscula.
- Para acceder a los atributos dentro de los métodos de la clase necesitamos utilizar el la pseudovariable **\$this** seguida del operador flecha (->)
- **\$this** es una referencia al objeto que llamó al método.
- Si no accedemos a los atributos con **\$this->** estaremos creando una variable local y el algoritmo fallará.
- El nombre del atributo al ser referenciado no lleva el símbolo del dólar.

Crear clase en PHP


- Ejemplo instanciación de una clase

```
<?php
//fichero Coche.php
class Coche {
    //declaración de una propiedad c
    public $tipo="turismo";
    public $color;

    public function mostrarTipo(){
        echo $this->tipo;
    }
}
?>
```

Observa:

- Hay que incluir el archivo que tiene definida la clase antes de poder instanciar un objeto, si no da error.
- Para acceder a los atributos o métodos de un objeto creado, se usa el operador flecha. **Solo se pone el símbolo \$ delante del nombre del objeto, no delante de los nombres de los atributos (aunque en la definición de la clase si aparezca con \$), ni de los métodos.**
- En este caso, al ser los atributos de la clase de tipo **public** se les puede asignar valores desde fuera de la clase utilizando el nombre del objeto seguido del operador “->” y el nombre del atributo (**SIN EL SÍMBOLO \$**).



```
<?php
//fichero index.php
include "Coche.php";
$coche = new Coche();
$coche->color = "rojo";
?>
```

Crear clase en PHP

Visibilidad de las propiedades:

- Al declarar un atributo hay que indicar el nivel de acceso que tiene. Pueden ser:
 - **Public:** los atributos “public” pueden utilizarse directamente por los objetos instanciados de la clase. Es decir, a estos atributos se puede acceder desde donde sea.
 - **Private:** los atributos “private” solo pueden ser accedidos y modificados por los métodos definidos en la clase, no directamente por los objetos que se creen de la misma (los atributos normalmente son privados).
 - **Protected:** a este tipo de atributo solo se puede acceder y modificar desde la clase donde está definido o en clases que heredan de la clase donde se ha definido.

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

Crear clase en PHP

Motivos para crear atributos privados:

- Su valor forma parte de la información interna del objeto.
- Permite mantener cierto control sobre sus posibles valores al solo poder ser accedidos mediante los métodos de la clase.
- Aunque no es obligatorio, el nombre del método que nos permite obtener el valor de un atributo suele empezar por **get** (cuando el método va devolver el valor de la propiedad), y el que nos permite modificarlo por **set** (cuando el método va a cambiar el valor de la propiedad), seguidos ambos por el nombre del atributo empezando por letra mayúscula.

```
<?php
//fichero Coche.php
class Coche {
    private $tipo="turismo";
    private $color;
    public function getTipo(){
        return $this->tipo;
    }
    public function setTipo($newTipo){
        $this->tipo=$newTipo;
    }
    public function getColor(){
        return $this->color;
    }
    public function setColor($newColor){
        $this->color=$newColor;
    }
}
?>
```


Crear clase en PHP

- Además de métodos y propiedades, en una clase podemos definir **constantes**.
- Los nombres de constantes no usan el símbolo dólar.
- Su visibilidad en versiones 7 o anteriores era siempre **public**, ahora puede modificarse.
- Las constantes están definidas a nivel de clase, no de objeto, por tanto **no es necesario crear ningún objeto de la clase para poder acceder a las constantes definidas en ella**.
- Para definir una constante se utiliza la palabra **const** antes del nombre que se le asigne(sin el símbolo \$) (**por convención este nombre se escribe en mayúsculas**), por ejemplo, **const PI = 3.1415;**
- Para acceder a una constante de una clase hay que utilizar el “**operador de resolución de ámbito**” que se representa con dos símbolos de dos puntos seguidos “**::**”
- Se indica el **nombre de la clase** seguido del operador de resolución de ámbito y el nombre de la constante: **echo Clase::PI**

Crear clase en PHP

- Ejemplo:

```
<?php
class Persona
{
    // Propiedades
    private $nombre    = null;
    private $apellidos = null;
    public $edad=null;

    // Constantes:
    const PERSONA_HOMBRE = "HOMBRE";
    const PERSONA_MUJER  = "MUJER";
}
```

```
<?php
require_once("Persona.php");

//accedo a la constante antes de crear un objeto
echo"Muestro la constante PERSONA_MUJER antes de crear instancia de la clase: ";
echo Persona::PERSONA_MUJER;
```

Crear clase en PHP

Static en clases

- Declarar propiedades o métodos de clases como estáticos los hacen accesibles **sin la necesidad de instanciar la clase**.
- Se definen utilizando la palabra reservada **static**.
- Los atributos estáticos **NO pueden ser accedidos desde un objeto instanciado usando el operador flecha ->** (la forma de acceder a ellos depende de su visibilidad).
- Un método estático sí puede ser accedido desde un objeto instanciado.
- No se puede acceder a propiedades **no estáticas** desde métodos estáticos ya que las propiedades no estáticas pertenecen solo a objetos instanciados donde cada objeto tiene un valor de la propiedad independiente.

Crear clase en PHP

Static en clases

- La forma de acceder a las **propiedades estáticas** depende de la visibilidad que tengan definida:
 - **Si el atributo estático es público**, se accede a él con el nombre de la clase y el operador de resolución de ámbito. `Clase::propiedad`
 - **Si el atributo estático es privado**, solo se puede acceder a él desde los métodos de la propia clase, y para ello se utiliza la palabra **self** seguido del operador de resolución de ámbito y el nombre del atributo (**\$this** hace referencia al objeto actual, mientras que **self** hace referencia a la clase actual). `self::propiedad`

Crear clase en PHP

Static en clases

- **Si el método estático es público**, se puede acceder a él con el nombre de la clase y el operador de resolución de ámbito o utilizando la instancia del objeto y el operador flecha.

```
Clase::método();
```

```
$obj->método();
```

- Como los métodos estáticos pueden ser llamados sin tener un objeto instanciado la pseudovariante `$this` no está disponible dentro de ellos para acceder a los atributos de la clase. Se usa en su lugar **self::**.

Crear clase en PHP

- Los atributos estáticos de una clase se suelen usar para guardar información general sobre la misma, como puede ser el número de objetos que se han instanciado. **Sólo existe un valor del atributo, que se almacena a nivel de clase.**
- Los métodos estáticos suelen realizar alguna tarea específica o devolver un objeto concreto. Por ejemplo, las clases matemáticas suelen tener métodos estáticos para realizar logaritmos o raíces cuadradas. No tiene sentido crear un objeto si lo único que queremos es realizar una operación matemática.

Crear clase en PHP

- Ejemplo propiedades y métodos estáticos

```
include "Coche.php";
echo "<h2>Sin crear objeto de la clase coche</h2>";
echo "Coche::RUEDAS (Constante ruedas): ".Coche::RUEDAS."<br/>";
echo "Coche::color: ".Coche::$color . "\n<br/>"; // Muestra "rojo"
echo "Coche::mostrarColor(): ".Coche::mostrarColor() . "\n<br/>"; // Muestra rojo
echo "Coche::dimeColorTecho(): ".Coche::dimeColorTecho() . "\n<br/>"; // Muestra negro
echo"<hr><br/>";
echo "<h2>Creamos objeto coche</h2>";
$miCoche = new Coche();
echo "Podemos acceder a un método estático con la instancia: \$miCoche->mostrarColor() :
.miCoche->mostrarColor() . "\n<br/>"; // Muestra "rojo"
/*
 * Una propiedad declarada como static no puede ser accedida desde una instancia,
 * si se puede desde un método estático
 */
echo "NO Podemos acceder a un atributo estático con la instancia:<br/>";
echo "\$miCoche->color:". $miCoche->color . "\n<br/>"; // Error, propiedad color no definida
echo"<hr><br/>";
```

```
<?php
class Coche {
    public static $color = 'rojo';
    private static $colorTecho = 'negro';
    const RUEDAS = 4;
    public static function mostrarColor()
    {
        return self::$color;
    }
    public static function dimeColorTecho()
    {
        return self::$colorTecho;
    }
}
```

Accedemos al atributo estático sin instanciar la clase con el nombre de la clase y el operador de ámbito de resolución.

Accedemos al método estático con el nombre de la clase y el operador de ámbito de resolución.

Accedemos al método estático mediante la instancia del objeto

Crear clase en PHP

Método constructor de una clase:

- Podemos crear métodos constructores de las clases.
- Si lo tienen definido, será invocado al crear el objeto, es el primer método que se ejecuta cuando se crea un objeto.
- Es invocado automáticamente.
- Se suelen utilizar para inicializar atributos que el objeto pueda necesitar antes de usarse.
- El constructor se llama **`__construct()`** (lleva dos guiones bajos antes de la palabra construct).
- También puede llamarse con el mismo nombre que la clase, pero solo puede haber uno.

```
public function __construct([parámetros])  
{  
    .....  
}
```

```
public function Clase([parámetros])  
{  
    .....  
}
```


Crear clase en PHP

Otras características de los constructores son:

- Solo puede haber un constructor en cada clase.
- Un constructor puede recibir parámetros que se utilizan normalmente para inicializar atributos, en cuyo caso se pasarán cuando se crea el objeto.
- El constructor puede llamar a otros métodos.

Crear clase en PHP

Ejemplo método constructor:

```
class Producto {  
    private static $num_productos = 0;  
    private $codigo;  
  
    public function __construct($codigo) {  
        $this->codigo = $codigo;  
        self::$num_productos++;  
    }  
}
```

En el ejemplo el constructor recibe un parámetro que utiliza para inicializar la propiedad código e incrementar en uno la propiedad estática \$num_productos.

```
<?php  
    $p = new Producto('refresco cola');  
?>
```

Crear clase en PHP

Método destructor de una clase:

- Al igual que podemos definir un método constructor, también podemos definir un método destructor (su uso es opcional).
- Debe llamarse **__destruct()** y no lleva argumentos.
- Permite hacer algún tipo de acción cuando el objeto va a ser destruido.
- Normalmente se ejecutará automáticamente al finalizar el script o función en el que se utilice un objeto, cuando este ya no vaya a ser referenciado más (por ejemplo cerrar una conexión a la BD).
- Podemos forzar a que se ejecute antes si liberamos la memoria del objeto con `unset($obj)`.

Relacionar clases

- Lo normal, cuando desarrollamos una aplicación web, es que tengamos varias clases y que algunas estén relacionadas entre sí. Tenemos varias formas de relacionar clases.
- Una forma sencilla de relacionar dos clases es instanciando una en otra.

```
<?php
require_once 'ClaseDos.php';
class ClaseUno {
    private $result;
    public function unMetodo($n1,$n2){
        $c12= new ClaseDos($n1,$n2);
        $this->result=$c12->multiplica();
        return $this->result;
    }
}
```

```
<?php

class ClaseDos {
    private $num1;
    private $num2;

    public function __construct($n1,$n2){
        $this->num1=$n1;
        $this->num2=$n2;
    }

    public function multiplica(){
        return $this->num1 * $this->num2;
    }
}
```

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            require "ClaseUno.php";
            $c11=new ClaseUno();
            echo $c11->unMetodo(2,5);
        ?>
    </body>
</html>
```

Relacionar clases

- Otra forma de relacionar clases es pasar como parámetro a un método de una clase un objeto de otra.

Observa que el parámetro que le llega es un objeto, por tanto tiene acceso a sus propiedades y métodos permitidos.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      require "ClaseDos.php";
      $c11=new ClaseDos();
      echo $c11->multiplica(new ClaseUno(5,2))."<br/>";
      echo $c11->multiplica(new ClaseUno(6,7))."<br/>";
    ?>
  </body>
</html>
```

```
<?php
require 'ClaseUno.php';
class ClaseDos {
    public function multiplica(ClaseUno $claseUno){
        return $claseUno->getNum1() * $claseUno->getNum2();
    }
}
```

```
<?php
class ClaseUno {
    private $num1;
    private $num2;
    //Constructor con el nombre de la clase
    public function ClaseUno($n1,$n2){
        $this->num1=$n1;
        $this->num2=$n2;
    }
    public function getNum1(){
        return $this->num1;
    }
    public function getNum2(){
        return $this->num2;
    }
}
```

Relacionar clases

- Otra manera de relacionar clases, es creando objetos de una en la definición de otra.

```
<?php
require_once('grafica.php');
require_once('ram.php');
require_once('pantalla.php');
require_once('hdd.php');

class Ordenador {
    private $tarjetaGrafica;
    private $memoriaRam;
    private $pantalla;
    private $discoDuro;

    public function __construct(){
        $this->pantalla = new Pantalla();
        $this->memoriaRam = new Ram();
        $this->discoDuro = new DiscoDuro();
        $this->tarjetaGrafica = new Grafica();
    }

    public function apagar(){
        $this->pantalla->apagar();
        $this->tarjetaGrafica->ventiladorOff();
        $this->discoDuro->PararDeGirar();
        $this->memoriaRam->limpiar();
    }
}

?>
```

Herencia

- La herencia permite definir clases en base a otras clases ya existentes.
- Las nuevas clases que “heredan” se conocen con el nombre de “**subclases**”.
- La clase de la que heredan se denomina “**clase padre**” o “**superclase**”.
- Una subclase hereda todos los atributos y métodos públicos y protegidos (no los privados) de la clase padre que mantendrán la funcionalidad original a menos que una subclase los sobrescriba.
- En una subclase se pueden definir atributos y métodos propios.
- Una clase solo puede heredar de una única clase (**no es posible realizar herencia múltiple**), pero sí es posible la **herencia multinivel** (unas clases pueden heredar de otras).
- Para indicar que una clase hereda de otra se utiliza la palabra clave “**extends**”.
- Ventaja de usar la herencia: optimización/reutilización de código.

Herencia

Sobreescritura de métodos:

- En una subclase, podemos volver a definir los métodos heredados de la clase padre.
- Al volver a escribir el método con el mismo nombre, sobrescribimos el comportamiento.
- Nuestro código detectará desde qué instancia se está invocando el método, y ejecutará un método u otro.
- En la subclase se puede crear un nuevo constructor que redefina el comportamiento de la clase padre (si existe).
- Podemos además hacer que en nuestro método sobrescrito se ejecute el correspondiente método de la clase padre. Para ello se usa el modificador **parent::** seguido del nombre del método.
- “**parent**” hace referencia a la clase padre de la actual.

Herencia

- Si no queremos que la clase sea heredada se indica con la palabra reservada “**final**”.
- Si no queremos que un método pueda sobrescribirse se indica también con la palabra “**final**”.

```
final class Coche {  
    public function getColor()  
    {  
        echo "Rojo";  
    }  
}  
class CocheDeLujo extends Coche {  
    // Error fatal, clase no heredable  
}
```

```
class Coche {  
    final public function getColor()  
    {  
        echo "Rojo";  
    }  
}  
class CocheDeLujo extends Coche {  
    public function getColor()  
    {  
        echo "Azul";  
    }  
} // Dará un error fatal, getColor() no puede  
sobreescribirse
```

Herencia

```
<?php
class Coche {
    protected $color;
    //si no se declara static no se puede acceder con el operador parent::
    protected static $numRuedas;
    function __construct(){
        echo "Estoy en el constructor de la clase Coche<br/>";
    }
    public function setColor($color)
    {
        echo "Estoy en setColor de Coche y pongo el color $color. <br/>";
        $this->color = $color;
    }
    public function getColor()
    {
        return $this->color;
    }
    /* los argumentos de una función, si los tiene. tienen que ser los mismos
    * en la clase padre y en la heredada, si no se produce un error
    */
    public function printCaracteristicas()
    {
        echo " Color de printCaracteristicas de Coche: ". $this->getColor()."<br/>";
    }
}
```

Ejemplo de sobreescritura de métodos y uso de parent.

```
<?php
require 'Coche.php';
class CocheDeLujo extends Coche {
    protected $extras;
    public function __construct(){
        echo "Estoy en el construct de CocheDeLujo va a invocar al constructor "
        . "de la clase padre<br/>";
        //ejecuto el constructor de la clase padre
        parent::__construct();
        echo "Estoy otra vez en el construct de CocheDeLujo<br/>";
    }

    //Defino los método en la clase cocheLujo de la propiedad extras de esta clase
    public function setExtras($extras)
    {
        $this->extras = $extras;
    }
    public function getExtras()
    {
        return $this->extras;
    }

    //Sobreescribo el método setColor de la clase padre
    public function setColor($color){
        echo "Estoy en setColor de CocheDeLujo y pongo el color $color. <br/>";
        $this->color="$color";
    }

    public function printCaracteristicas()
    {
        echo "Estoy en printCaracteristicas de CocheDeLujo<br/>";
        //ejecuto el método de la clase padre
        parent::printCaracteristicas();
        echo 'Extras de printCaracteristicas de CocheDeLujo: ' . $this->extras."<br/>";
        /* A las propiedades y métodos estáticos no se puede acceder directamente desde el objeto
        * $numRuedas es una propiedad estática y protegida. Solo puedo acceder usando
        * self(porque la hereda) o parent(referenciando directamente la clase padre
        */
        parent::$numRuedas=4;
        echo "Numero de ruedas en prinCaracteristicas de cocheDeLujo: ".parent::$numRuedas."<br/>";
        echo "Numero de ruedas en prinCaracteristicas de cocheDeLujo: ".self::$numRuedas."<br/>";
        echo '<hr/>';
    }
}
```

Recomendaciones

- Si la relación entre dos clases responde a una pregunta de tipo “**Clase A es un/a Clase B**”, es posible que el tipo de relación entre ellas sea de herencia.
- En cambio, si la relación entre dos clases responde a una pregunta de tipo “**Clase A tiene un/a Clase B**”, quizá en lugar de implementar una herencia es mejor declarar en la Clase A un atributo de la Clase B (relación de colaboración).

Clases y métodos abstractos

```
abstract class Clase{  
    .....  
}
```

- Para definir una clase como abstracta se antepone el modificador abstract a la palabra class.
- Las clases abstractas **no pueden ser instanciadas, solo pueden ser heredadas.**
- No se puede declarar una clase como abstract y final simultáneamente. Una clase abstracta obliga a que se herede para que se pueda utilizar, mientras que final indica que no se podrá heredar.
- La forma de indicar que una clase hereda de una clase abstracta es exactamente igual que cuando se hereda de una clase no abstracta, utilizando el modificador extends.

Clases y métodos abstractos

- Los **métodos definidos como abstractos no definen la implementación, solo se declara.**
- **Cualquier clase que contenga la definición de un método abstracto debe ser definida como abstracta.**
- Los **métodos declarados como abstractos deben implementarse obligatoriamente en las clases que heredan** de la abstracta, en caso contrario se produce un error.
- De este modo, podemos obligar que en las clases que heredan de la clase abstracta implementen comportamientos.
- Para declarar un método abstracto se antepone el modificador **abstract** antes de la visibilidad.

Clases y métodos abstractos

- El método abstracto a implementar en la clase que hereda de la clase abstracta:
 - Debe tener el mismo nombre que el definido en la clase abstracta.
 - Debe tener el mismo o menor modificador de restricción de acceso, por ejemplo si en la clase abstracta está definido como `protected` en la clase que hereda debe ser `protected` o `public`.
 - El número de argumentos ha de ser el mismo, aunque la clase que hereda puede tener argumentos opcionales.

- Ejemplo métodos abstractos con parámetros opcionales.
- Fuente W3Shools

```
<?php
abstract class ParentClass {
    // Abstract method with an argument
    abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
    // The child class may define optional arguments that are not in the parent's abstract
    method
    public function prefixName($name, $separator = ".", $greet = "Dear") {
        if ($name == "John Doe") {
            $prefix = "Mr";
        } elseif ($name == "Jane Doe") {
            $prefix = "Mrs";
        } else {
            $prefix = "";
        }
        return "{$greet} {$prefix}{$separator} {$name}";
    }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Interfaces

- Un interface es como una clase vacía que solamente contiene declaraciones de métodos. Podemos imaginarlos como contratos que deben cumplir las clases que los implementen.
- Se definen utilizando la palabra **interface** en lugar de **class**.
- Al crear las subclases se indica con la palabra “**implements**” que implementa una interfaz.
- Todos los métodos declarados en el interfaz deben ser desarrollados dentro de la clase que la implemente.
- Todos los métodos que se declaren en un interface deben ser públicos.
- Pueden contener constantes, pero no atributos.

Interfaces

- Una de las dudas más comunes en POO, es qué utilizar: interfaces o clases abstractas.
- Ambas permiten definir reglas para las clases que los implementen o hereden respectivamente. Y ninguna permite instanciar objetos. Las diferencias principales entre ambas opciones son:
 - En las clases abstractas, los métodos pueden contener código. En las interfaces no, habría que repetir el código en todas las clases que lo implemente.
 - Las clases abstractas pueden contener atributos, y los interfaces no.
 - No se puede crear una clase que herede de dos clases abstractas, pero sí se puede crear una clase que implemente varios interfaces.

Parámetros tipo objeto

Objetos como parámetros en funciones y métodos:

- Cuando se pasan objetos como parámetros en funciones y métodos, se puede especificar de qué clase deben ser indicando antes del parámetro el tipo (el nombre de la clase).
- Si al invocar al método, el parámetro no es del tipo adecuado (aunque no se haya especificado el tipo), se produce un error que podrías capturar.
- Cuando se pasa un objeto como parámetro, se pasa **siempre** por referencia. Es decir los cambios que hagamos quedan reflejados en el objeto original.

Parámetros tipo objeto

- Ejemplo de paso de objeto como parámetro

```
class Persona { public $nombre; }
class Gato { public $nombre; }

// esta función solo puede recibir objetos de tipo Persona
function muestraNombre(Persona $persona) {
    echo $persona->nombre;
}

$persona = new Persona();
$persona->nombre = "Pepe";
$gato = new Gato();
$gato->nombre = "Rufus";
muestraNombre ($persona); // correcto
muestraNombre ($gato); // ERROR: un gato no es una persona
```

Parámetros opcionales

- Un parámetro de un método es opcional, si en su declaración le asignamos un valor por defecto. Recuerda que los parámetros opcionales deben ir al final de la lista de argumentos.
- Si llamamos al método sin enviarle ese parámetro, tomará el valor por defecto.
- Se pueden utilizar tanto en programación estructurada como en POO.

Parámetros opcionales

```
<?php
class Cabecera {
    private $titulo;
    private $ubicacion;
    private $colorFuente;
    private $colorFondo;
    public function __construct($tit,$ubi='center',$colorFuen='#ffffff',$colorFon='#000000')
    {
        $this->titulo=$tit;
        $this->ubicacion=$ubi;
        $this->colorFuente=$colorFuen;
        $this->colorFondo=$colorFon;
    }

    public function mostrar()
    {
        echo '<div style="font-size:40px;text-align:'.$this->ubicacion.';color:';
        echo $this->colorFuente.';background-color:'.$this->colorFondo.'">';
        echo $this->titulo;
        echo '</div><br><hr>';
    }
}
```

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            include 'Cabecera.php';
            $cab1=new Cabecera('Parámetros opcionales');
            $cab1->mostrar();

            $cab2=new Cabecera('Parámetros opcionales','left');
            $cab2->mostrar();

            $cab3=new Cabecera('Parámetros opcionales','right','#ff0000');
            $cab3->mostrar();

            $cab4=new Cabecera('Parámetros opcionales','center','DEEPPINK','pink');
            $cab4->mostrar();

        ?>
    </body>
</html>
```

Métodos mágicos

- En PHP5 se introdujeron los llamados “métodos mágicos”.
- Son aquellos cuyo nombre comienza por dos guiones bajos.
- Ningún método de una clase puede llamarse como un método mágico a no ser que se vayan a utilizar estas funcionalidades
- Ya hemos visto alguno de ellos, como `__construct()`, y `__destruct()`, aunque hay más:
- `__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` y `__debugInfo()`

Métodos mágicos

- El programador será el encargado de escribir el comportamiento de estos métodos.
- Su característica más importante es que estos métodos (si están definidos) **nunca serán llamados directamente por el programador. PHP se encarga de llamarlos en el momento adecuado.** Y es por eso por lo que reciben el nombre de método mágicos.

Métodos mágicos

- **__construct()**: es llamado automáticamente por PHP cuando va a crear un objeto. Su principal uso es la inicialización de atributos.
- **__destruct()**: es llamado automáticamente al destruir un objeto. Un ejemplo de uso habitual de este método es el cierre de una conexión a una base de datos.

Métodos mágicos

`__get()` y `__set()`:

- Normalmente las propiedades de las clases están definidas como `protected` o `private` por lo que es necesario tener definidos unos métodos `get()` y `set()` que nos permitan acceder a ellas.
- Con los métodos mágicos `__set()` y `__get()` podemos implementar un funcionamiento para poder modificar o acceder a los distintos atributos de la clase evitando así tener que crear un método para cada uno de ellos.
- Una vez declarados estos métodos, si se intenta acceder a un atributo (como si fuera público), PHP llamará automáticamente a `__get()`. Y si asignamos un valor a un atributo, llamará a `__set()`.
- Evidentemente, `__set()` necesita dos parámetros de entrada: nombre del atributo y el valor a asignar. Y `__get()` solo necesita el nombre del atributo a obtener su valor.

```

<?php
class MiClase
{
    private $id;
    private $nombre;
    private $email;

    function __construct($id, $nombre, $email) {
        $this->id = $id;
        $this->nombre = $nombre;
        $this->email = $email;
    }

    public function __set($propiedad, $valor) {
        //mira si existe la propiedad $var en la clase, y si es así le asigna
        //se pasa por parámetro
        //__CLASS__ es una constante predefinida en PHP que contiene el nombre
        //echo "CLASE:".__CLASS__;
        if (property_exists(__CLASS__, $propiedad)) {
            $this->$propiedad = $valor;
        } else {
            //podemos devolver un mensaje indicando que no existe.
            return "método __set() No existe el atributo $propiedad.";
        }
    }

    public function __get($propiedad) {
        if (property_exists(__CLASS__, $propiedad)) {
            return $this->$propiedad;
        }
        else
            return "método __get() No existe el atributo ".$propiedad;
    }
}

```

Ejemplo __get() y __set():

Observa que las propiedades de la clase están definidas como “private”,

__CLASS__ es una constante predefinida en PHP que contiene el nombre de la clase.

El método mágico **__set()** comprueba si existe la propiedad a la que desea asignarle un valor en la clase. Si es así, realiza la asignación, si no, muestra un mensaje indicando que la propiedad no existe.

El método mágico **__get()** comprueba si existe la propiedad solicitada como argumento, y si es así devuelve su valor.

```
<?php
    require_once("MiClase.php");

    $obj = new miClase(1, "objeto1", "prueba1@ejemplo.com");

    echo "ID:". $obj->id."<br/>";           // muestra 1
    echo "Nombre:". $obj->nombre."<br/>";    //muestra objeto1
    $obj->nombre = "he cambiado el nombre";
    echo "Nombre:". $obj->nombre."<br/>";    //muestra He cambiado el nombre
    echo "Apellidos:". $obj->apellidos."<br/>"; //muestra el mensaje de error definido en el método __get
    $obj->apellidos = "Tengo apellidos";    //intentamos crearla
    echo "Apellidos:". $obj->apellidos."<br/>"; //muestra el mensaje de error definido en el método __get
?>
```

Ejemplo __get() y __set():

Los atributos de la clase están definidos como private, pero accedemos a ellos como si tuvieran visibilidad pública mediante el operador flecha, gracias a los métodos mágicos.

Al tener definidos los métodos mágicos __set() y __get() PHP se encarga de ejecutarlos automáticamente, pero para poder hacerlo, deben estar definidos en la clase.

Métodos mágicos

`__get()` y `__set()`:

- El uso de estos métodos a veces no es recomendado por varias razones: son más lentos, hacen el mantenimiento más largo, se pierden las ventajas de la encapsulación (sería como haber declarado públicas las propiedades y eso normalmente no es recomendable, hay que mantenerlas protegidas).
- Por ejemplo, imaginemos que tenemos una clase en la que calculamos el importe de una factura aplicando el IVA. El IVA es el mismo siempre y está marcado por ley, por lo que no queremos que nadie pueda modificarlo de forma accidental. Si usáramos los métodos mágicos estaríamos permitiendo su acceso.

Métodos mágicos

- El método mágico **__toString()** es invocado automáticamente (si existe) cuando se realiza un echo o un print del objeto.
- Devuelve un string si no se produce un error.
- De esta forma podemos, por ejemplo devolver el contenido de las propiedades del objeto como si fuera un string.

```
class MiClase
{
    private $id;
    private $nombre;
    private $email;

    function __construct($id, $nombre, $email) {
        $this->id = $id;
        $this->nombre = $nombre;
        $this->email = $email;
    }

    public function __toString(){
        return $this->nombre." ".$this->email;
    }
}
```

```
<?php
    require_once("MiClase.php");

    $obj = new miClase(1, "objeto1", "prueba1@ejemplo.com");
    echo $obj;|
```

objeto1 prueba1@ejemplo.com

Al hacer un “echo” del objeto se invoca el método mágico **__toString** que nos devuelve una cadena con el contenido de las propiedades nombre y email

Bibliografía

- <https://diego.com.es/certificacion-php>
- <http://www.tutorialesprogramacionya.com/phpya/poo/>
- <https://www.php.net/manual/es/language.oop5.php>
- <https://informaticapc.com/tutorial-php/clases-objetos-herencia.php>
- <https://www.mmfilesi.com/blog/2-php-orientado-a-objetos-ocultacion-getter-y-setter/>
- <https://www.mmfilesi.com/blog/php-orientado-a-objetos-8-clases-abstractas-y-finales/>
- <https://diego.com.es/clases-abstractas-en-php>
- <https://diego.com.es/interfaces-en-php>
- <https://diego.com.es/patrones-de-diseno-en-php#ModelViewControllerPattern>
- <https://diego.com.es/metodos-magicos-en-php>