Conceptos básicos OO. PHP

Heike Bonilla Redondo

Índice

Contenido

1. Clases y Objetos¶

PHP sigue un paradigma de programación orientada a objetos (POO) basada en clases.

Un clase es un plantilla que define las propiedades y métodos para poder crear objetos. De este manera, un objeto es una instancia de una clase.

Tanto las propiedades como los métodos se definen con una visibilidad (quien puede acceder)

- private: Sólo puede acceder la propia clase.
- protected: Sólo puede acceder la propia clase o sus descendientes.
- public: Puede acceder cualquier otra clase.

Para declarar una clase, se utiliza la palabra clave class seguido del nombre de la clase. Para instanciar un objeto a partir de la clase, se utiliza new:

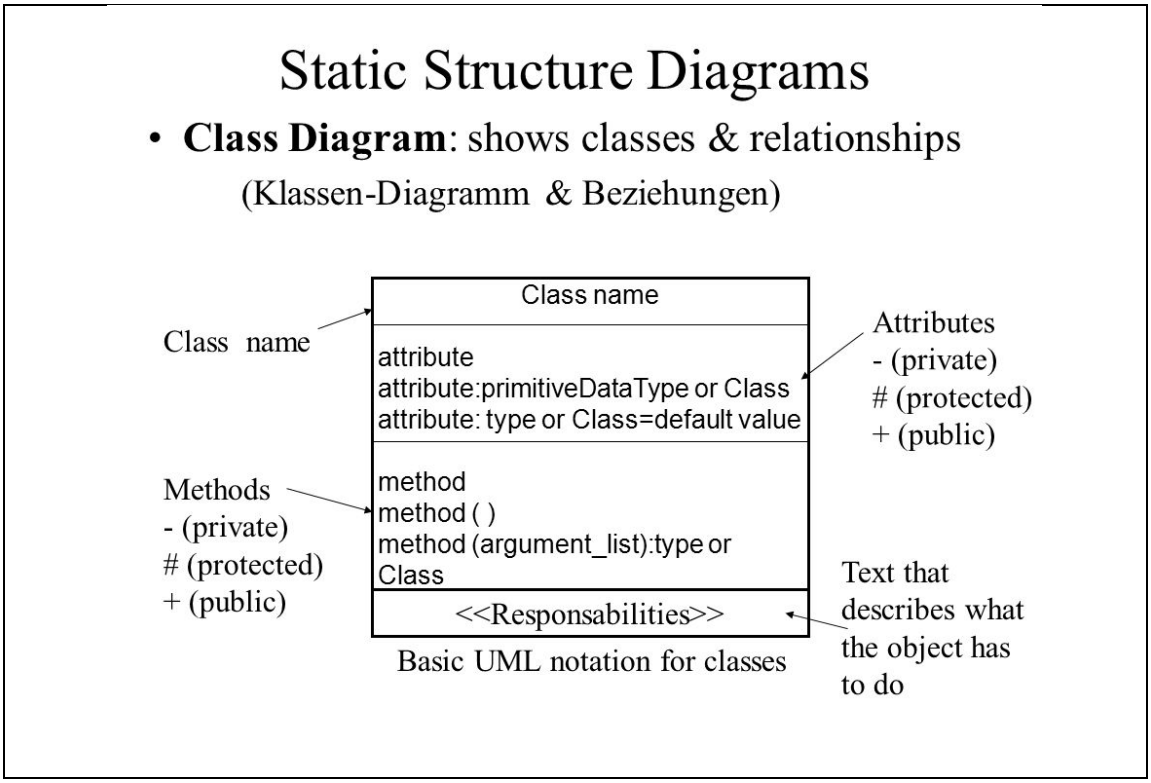
```
<?php
class NombreClase {
// propiedades
// y métodos
}
```

```
$ob = new NombreClase();
?>
```

Por convención, todas las clases empiezan por letra mayúscula.

Cada clase irá definida en un fichero diferente

Cuando un proyecto crece, es normal modelar las clases mediante UML.



*PHP **no utiliza la notación punto** para acceder a los miembros de una clase, sino la **notación flecha (->)**.*

$objeto->propiedad;

```
$objeto->método(parámetros);
```

Como ejemplo **(ejemplo_1.php)**, escribimos una clase persona como:

```
<?php
class Persona {
    private string $nombre;

    public function setNombre(string $nom) {
        $this->nombre=$nom;
    }

    public function imprimir(){
```

```
        echo $this->nombre;
        echo '<br>';
    }
}
//Creamos un objeto
$bruno = new Persona();
$bruno->setNombre("Pepe Pérez");

//Usamos el método mprimir
$bruno->imprimir();
?>
```

Aunque se pueden declarar varias clases en el mismo archivo, es una mala práctica. Así pues, cada fichero contedrá una sola clase, y se nombrará con el nombre de la clase.

$this y ::parent

La variable **$this** se refiere siempre al objeto que está ejecutando el código, exactamente igual que en Java, Javascript y muchos otros lenguajes orientados a objeto.

A veces, cuando tenemos una jerarquía de clases y unas heredan de otras, necesitamos invocar algún método de la clase madre o superclase. En ese caso, usaremos la palabra **parent** seguida de la **notación cuatro puntos (::)**. Observa cómo se hace en este ejemplo **(ejemplo_2.php)** , en el que el constructor de la subclase invoca al constructor de la superclase:

```
<?php
class MiClase {
    private $var1;
    public function __construct($param) {
        $this->var1 = $param;
    }
}

class MiSubclase extends MiClase {
    private $var2;
    public function __construct($param1, $param2) {
        $this->var2 = $param2;
        // Llamada a un método de la superclase
        parent::__construct($param1);
    }
}
?>
```

2. Encapsulación¶

La encapsulación mantiene el estado de la clase como privado y proporcionamos un interfaz pública para manipular este estado interno.

Para esto, se añaden métodos públicos (*getter/setter*)

Las propiedades se definen como privadas o protegidas (si queremos que las clases heredadas puedan acceder).

GETTERS

Proporcionan un interfaz público para acceder a cada una de las propiedades de la clase.

- Crearemos uno por cada propiedad.
- Son públicos
- Tomarán el nombre getNombrePropiedad (camelCase)
- No les pasaremos atributos.
- Devolverá $this->nombrepropiedad

SETTERS

Proporcionan un interfaz público para modificar el estado interno de nuestra clase.

- Crearemos uno por propiedad
- Son públicos
- Tomarán el nombre setNombrePropiedad ($valor) (camelCase)
- Les pasaremos un atributo, el valor que queremos darle a la propiedad que corresponda.
- Devolverá o no $this

Siguiente ejemplo: **(ejemplo_3.php)**

```php
<?php
class MayorMenor {
    private int $mayor;
    private int $menor;
//SETTERS
    public function setMayor(int $may) {
        $this->mayor = $may;
        return $this;
    }

    public function setMenor(int $men) {
        $this->menor = $men;
        return $this;
    }
//GETTERS
    public function getMayor() : int {
        return $this->mayor;
    }

    public function getMenor() : int {
        return $this->menor;
    }
}

/*
```

```
Ejemplo de uso de la clase
$valor1=10;
$valor2=5;
$obj = new MayorMenor();
$obj->setMayor($valor1);
$obj->setMenor($valor2);

echo $obj->getMayor();
echo"<br>";
echo $obj->getMenor();
*/
?>
```

AMPLIACION

Los setters, en otros lenguajes, no devuelven nada. Sin embargo, en PHP es costumbre que los setters devuelvan el objeto completo, es decir, que terminen con un return $this. Así:

```
class MiClase {

    private $var1 = "Esto es un atributo privado";

    // Getter

    public function getVar1() {

        return $var1;

    }

    // Setter

    public function setVar1($value) {

        $var1 = $value;

        return $this;   // Devolvemos el objeto al terminar

    }

}
```

Si lo hacemos así, estaremos creando lo que se llama un fluent interface o interfaz fluido, podremos encadenar varias invocaciones a métodos del objeto en una sola instrucción, algo que permite que el código se vea más organizado y legible.

Un ejemplo. Imagina que la clase anterior tuviera más atributos ($var1, $var2, $var3, etc), cada uno con sus respectivos setters. La forma tradicional de invocarlos todos sería algo así:

$obj = new MiClase();

$obj->setVar1($valor1);

$obj->setVar2($valor2);

$obj->setVar3($valor3);

// etc.

En cambio, si los setters devuelven this podemos usar un fluent interface y escribirlo así:

$obj = new MiClase();

$obj->setVar1($valor1)

->setVar2($valor2)

->setVar3($valor2);

Puede parecer un cambio insignificante, pero cuando los objetos son muy complejos, el código fluent se hace mucho más legible que el código tradicional.

---

3. Constructor¶

El constructor de los objetos se define mediante el método mágico __construct. Puede o no tener parámetros, pero sólo puede haber un único constructor.

Ejemplo_4.php

```php
<?php
class Persona {
    private string $nombre;

    public function __construct(string $nom) {
        $this->nombre = $nom;
    }

    public function setNombre(string $nom) {
        $this->nombre=$nom;
    }
    public function imprimir(){
      echo $this->nombre;
      echo '<br>';
    }
}

$bruno = new Persona("Pepe Pérez");
```

```php
$bruno->imprimir();
?>
```

Orden de los elementos dentro de una clase

A la hora de codificar el orden de los elementos debe ser:

```php
<?php

class NombreClase {
    // propiedades

    // constructor

    // getters - setters

    // resto de métodos
}
?>
```

4. Métodos estáticos¶

Los métodos estáticos en PHP se usan cuando una clase no tiene estado (es decir, no tiene atributos), o bien cuando ese método no tiene nada que ver con el estado de los objetos, sino que responde exactamente igual para todas las instancias.

Los métodos estáticos se declaran así:

```php
class MiClase {
    // Esto es un método estático
    public static function miMetodo() {
        ...
    }
}
```

Para invocar un método estático, no es necesario instanciar ningún objeto.
De hecho, si intentamos invocarlo a través de un objeto, fallará.
Estos métodos se invocan a través del nombre de la clase directamente, usando la notación cuatro puntos (::).

```php
// Esto invocará el método estático del ejemplo anterior
MiClase::miMetodo();
```

Ejemplo **(ejemplo_5.php)**:

```php
<?php
//Ejemplo uso método estático y constantes en clases
class Producto {
    const IVA = 0.23;
private static $numProductos = 0;
```

```
    public static function nuevoProducto() {
        self::$numProductos++;
    }

}

Producto::nuevoProducto();

//Podemos acceder a la constante también como si fuese estática
$impuesto = Producto::IVA;
echo $impuesto;
?>
```

Otro ejemplo **(ejemplo_6.php)**

```
<?php
class TokenGenerator {
    private const KEY = 'clave_secreta';

    // Método estático para generar un token utilizando uniqid
    public static function generarToken() {
        // Generar un identificador único y combinarlo con la clave secreta
        $token = md5(uniqid(self::SECRET_KEY, true));

        return $token;
    }
}

// Uso del método estático para generar un token
$token = TokenGenerator::generarToken();

echo "Token generado: $token";
```

5. Funciones de PHP

Al trabajar con clases y objetos, existen un conjunto de funciones ya definidas por el lenguaje que permiten obtener información sobre los objetos:

- instanceof: permite comprobar si un objeto es de una determinada clase
- get_class: devuelve el nombre de la clase
- get_declared_class: devuelve un array con los nombres de las clases definidas
- class_alias: crea un alias
- class_exists / method_exists / property_exists: true si la clase / método / propiedad está definida
- get_class_methods / get_class_vars / get_object_vars: Devuelve un array con los nombres de los métodos / propiedades de una clase / propiedades de un objeto que son accesibles desde dónde se hace la llamada.

Un ejemplo de estas funciones puede ser el siguiente (**ejemplo7.php**):

```
<?php
class Producto {
    const IVA = 0.23;
    private $numProductos = 0;
    private $codigo;
```

```
    public function __construct(string $cod) {
        $this->numProductos++;
        $this->codigo = $cod;
    }

    public function mostrarResumen() : string {
        return "El producto ".$this->codigo." es el número ".$this-
>numProductos;
    }
}

$prod1 = new Producto("PS5");
$prod2 = new Producto("XBOX Series X");
$prod3 = new Producto("Nintendo Switch");
echo $prod3->mostrarResumen();

$p = new Producto("PS5");
if ($p instanceof Producto) {
    echo "<br>Es un producto";
    echo "<br>La clase es ".get_class($p);

    class_alias("Producto", "Articulo");
    $c = new Articulo("Nintendo Switch");
    echo "<br>Un articulo es un ".get_class($c);
    echo"<br>";
    print_r(get_class_methods("Producto"));
    echo"<br>";
    //Solo saldrán las públicas
    print_r(get_class_vars("Producto"));
    echo"<br>";
    print_r(get_object_vars($p));
    echo"<br>";

    if (method_exists($p, "mostrarResumen")) {
        $p->mostrarResumen();
    }
}
```

6. Herencia¶

PHP soporta herencia simple, de manera que una clase solo puede heredar de otra, no de dos clases a la vez. Para ello se utiliza la palabra clave extends. Si queremos que la clase A hereda de la clase B haremos:

class A extends B

El hijo hereda los atributos y métodos públicos y protegidos.

Cada clase en un archivo

Debemos colocar cada clase en un archivo diferente para posteriormente utilizarlo mediante include. En los siguiente ejemplo los hemos colocado junto para facilitar su legibilidad.

Por ejemplo, tenemos una clase Producto y una Tv que hereda de Producto **(ejemplo_8.php)**:

```php
<?php
class Producto {
    public $codigo;
    public $nombre;
    public $nombreCorto;
    public $PVP;

    public function mostrarResumen() {
        echo "<p>Prod:".$this->codigo."</p>";
    }
}

/*
Esta clase iria en otro fichero
Tendriamos que hacer previamente in include del fichero que incluye Producto
*/

class Tv extends Producto {
    public $pulgadas;
    public $tecnologia;
}
?>
```

Podemos utilizar las siguientes funciones para averiguar si hay relación entre dos clases:

- get_parent_class(object): string
- is_subclass_of(object, string): bool

```php
/*ejemplo_8.php
Ejemplo de uso y ver relación entre las clases
*/
$t = new Tv();
$t->codigo = 33;
if ($t instanceof Producto) {
    echo $t->mostrarResumen();
} else {
    echo"No muestro resumen";
}

$padre = get_parent_class($t);
echo "<br>La clase padre es: " . $padre;
$objetoPadre = new $padre;
echo $objetoPadre->mostrarResumen();

if (is_subclass_of($t, 'Producto')) {
    echo "<br>Soy un hijo de Producto";
}
```

Sobreescribir métodos¶

Podemos crear métodos en los hijos con el mismo nombre que el padre, cambiando su comportamiento. Para invocar a los métodos del padre -> parent::nombreMetodo()

Ejemplo (**ejemplo_9.php)**

```php
<?php
include("ejemplo_7.php");
class Tv extends Producto {
   public $pulgadas;
   public $tecnologia;

   public function mostrarResumen() {
      parent::mostrarResumen();
      echo "<p>TV ".$this->tecnologia." de ".$this->pulgadas."</p>";
   }
}
?>
```

Constructor en hijos¶

En los hijos no se crea ningún constructor de manera automática. Por lo que si no lo hay, se invoca automáticamente al del padre.

En cambio, si lo definimos en el hijo y necesitamos ejecutar el constructor del padre, hemos de invocar al del padre de manera explícita.

Ejemplo (**ejemplo_10.php**)

```php
<?php
class Producto {
    public string $codigo;

    public function __construct(string $codigo) {
        $this->codigo = $codigo;
    }

    public function mostrarResumen() {
        echo "<p>Prod:".$this->codigo."</p>";
    }
}

class Tv extends Producto {
    public $pulgadas;
    public $tecnologia;

    public function __construct(string $codigo, int $pulgadas, string
$tecnologia) {
//Invocamos el constructor del padre
        parent::__construct($codigo);
        $this->pulgadas = $pulgadas;
        $this->tecnologia = $tecnologia;
    }
```

```
    public function mostrarResumen() {
        parent::mostrarResumen();
        echo "<p>TV ".$this->tecnologia." de ".$this->pulgadas."</p>";
    }
}
$t = new Tv(12,33,"LED");
echo $t->mostrarResumen();
```

7. Métodos mágicos¶

Todas las clases PHP ofrecen un conjunto de métodos, también conocidos como *magic methods* que se pueden sobreescribir para sustituir su comportamiento. Algunos de ellos ya los hemos utilizado.

Ante cualquier duda, es conveniente consultar la documentación oficial.

Los más destacables son:

- __construct()
- __destruct() → se invoca al perder la referencia. Se utiliza para cerrar una conexión a la BD, cerrar un fichero, ...
- __toString() → representación del objeto como cadena. Es decir, cuando hacemos echo $objeto se ejecuta automáticamente este método.
- __get(propiedad), __set(propiedad, valor) → Permitiría acceder a las propiedad privadas, aunque siempre es más legible/mantenible codificar los *getter/setter*.
- __isset(propiedad), __unset(propiedad) → Permite averiguar o quitar el valor a una propiedad.
- __sleep(), __wakeup() → Se ejecutan al recuperar (*unserialize*^) o almacenar un objeto que se serializa (*serialize*), y se utilizan para permite definir qué propiedades se serializan.
- __call(), __callStatic() → Se ejecutan al llamar a un método que no es público. Permiten sobrecargan métodos.

Clonado

Al copiar dos objetos con el signo de asignación (=) no se crea una copia referenciada al primero. De manera que si modificamos el inicial se modificará el copiado.

Si queremos una copia, hay que clonarlo mediante el método clone(object) : object

Si queremos modificar el clonado por defecto, hay que definir el método mágico __clone() que se llamará después de copiar todas las propiedades.

Más información en https://www.php.net/manual/es/language.oop5.cloning.php