Programación orientada a objetos en PHP

Programación orientada a objetos

Definición de clases

- La declaración de una clase en PHP se hace utilizando la palabra class.
- A continuación y entre llaves, deben figurar los miembros de la clase.
- Conviene hacerlo de forma ordenada, primero las propiedades o atributos, y después los métodos, cada uno con su código respectivo.

```php
class Producto {
    private $codigo;
    public $nombre;
    public $pvp;
    public function muestra() {
        echo "<p>" . $this->codigo . "</p>";
    }
}
```

Programación orientada a objetos

Definición de clases

- Es preferible que cada clase figure en su propio fichero (producto.php).
- Los atributos de una clase son similares a las variables de PHP. Es posible indicar un valor en la declaración de la clase. En este caso, todos los objetos que se instancien a partir de esa clase, partirán con ese valor por defecto en el atributo.

Programación orientada a objetos

Definición de clases

- Se puede indicar en las funciones y métodos de qué clase deben ser los objetos que se pasen como parámetros así como el tipo del dato devuelto (caso que lo haya). Para ello, debes especificar el tipo antes del parámetro. Para el dato devuelto poner ":tipoDato", despues de la declaración de la función o el método y antes de las llaves.

```
public function precioProducto(Producto $p) :float {
      . . .
      return $precio;
}
```

Programación orientada a objetos

Creación de objetos

- Una vez definida la clase, podemos usar la palabra new para instanciar objetos de la siguiente forma:
    - $p = new Producto();
- Para que la línea anterior se ejecute sin error, previamente debemos haber declarado la clase. Para ello, en ese mismo fichero tendrás que incluir la clase poniendo algo como:
    - require_once('producto.php');

Programación orientada a objetos

Creación de objetos

- Para acceder desde un objeto a sus atributos o a los métodos de la clase, debes utilizar el operador flecha (sólo se pone el símbolo $ delante del nombre del objeto):
    - $p->nombre = 'Samsung Galaxy A6';
    - $p->muestra();

Programación orientada a objetos

Modificadores de acceso

- Se puede modificar el acceso a las propiedades y los métodos de una clase.
- Existen tres modificadores de acceso:
    - **public**: Los atributos/métodos declarados como public pueden utilizarse directamente por los objetos de la clase.
    - **protected**: Los atributos/métodos se pueden acceder desde la propia clase y desde las clases derivadas de la clase.
    - **private**: Los atributos/métodos sólo pueden ser accedidos y modificados por los métodos definidos en la clase.

Programación orientada a objetos

Constructores

- PHP permite declarar métodos constructores para las clases.
- Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que nos permite la inicialización que el objeto pueda necesitar antes de ser usado.
- Como PHP no admite sobrecarga de métodos sólo podremos crear un constructor por clase.
- Con el uso de las funciones “func_get_args()", “fun_get_arg()" y “func_num_arg()" podemos pasar distinto número de parámetros a un constructor "simulando" la sobrecarga del mismo.
- Otra posibilidad es usar el método mágico "__call" para capturar llamadas a métodos que no estén implementados.

Programación orientada a objetos

Constructores

```php
class Persona{
    public static $id;
    private $nombre;
    private $perfil;
    public function __construct($n, $p){
        $this->nombre=$n;
        $this->perfil=$p;
    }
}
// Creamos un objeto de la clase Persona e inicializamos sus atributos;
// Fíjate que ya NO podremos usar: $persona1=new Persona(); ya que el constructor espera dos parámetros.
$persona1=new Persona("Juan", "Privado");
//Podemos comprobarlo
var_dump($persona1);
```

Programación orientada a objetos

Destructores

- Es posible definir un método destructor, que debe llamarse "__destruct" y permite definir acciones que se ejecutarán cuando se elimine el objeto.

```php
class Producto {
    private static $num_productos = 0;
    private $codigo;
    public function __construct($codigo) {
        $this->$codigo = $codigo;
        self::$num_productos++;
    }
    public function __destruct() {
        self::$num_productos--;
    }
    …
}
$p = new Producto('GALAXYS');
```

Programación orientada a objetos

Funciones de utilidad

Función	Ejemplo	Significado
get_class()	`echo "La clase es: " . get_class($p);`	Devuelve el nombre de la clase del objeto.

class_exists	`if (class_exists('Producto') {`
	`  $p = new Producto();`
	`  . . .`
	`}`
		Devuelve **true** si la clase está definida o **false** en caso contrario.

get_declared_classes()	`print_r(get_declared_classes());`	Devuelve un array con los nombres de las clases definidas.
class_alias()	`class_alias('Producto', 'Articulo');` `$p = new Articulo();`	Crea un alias para una clase.

Programación orientada a objetos

Funciones de utilidad

get_class_methods()	`print_r(get_class_methods('Producto'));`	Devuelve un array con los nombres de los métodos de una clase que son accesibles desde dónde se hace la llamada.

method_exists()	`if (method_exists('Producto', 'vende') {`
	`  …`
	`}`
		Devuelve **true** si existe el método en el objeto o la clase que se indica, o **false** en caso contrario, independientemente de si es accesible o no.

get_class_vars()	`print_r(get_class_vars('Producto'));`	Devuelve un array con los nombres de los atributos de una clase que son accesibles desde dónde se hace la llamada.
get_object_vars()	`print_r(get_object_vars($p));`	Devuelve un array con los nombres de los métodos de un objeto que son accesibles desde dónde se hace la llamada.

property_exists()	`if (property_exists('Producto', 'codigo') {`
	`  . . .`
	`}`
		Devuelve **true** si existe el atributo en el objeto o la clase que se indica, o **false** en caso contrario, independientemente de si es accesible o no.

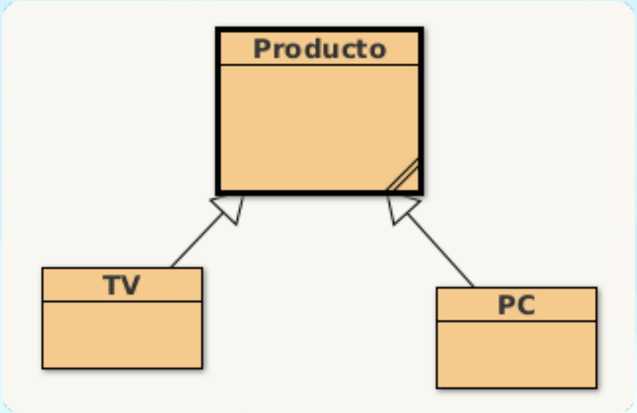
Programación orientada a objetos

Herencia

- La herencia es un mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente.
- Las nuevas clases que heredan también se conocen con el nombre de subclases.
- La clase de la que heredan se llama clase base o superclase.

Programación orientada a objetos

Herencia




```php
class Producto {
    public $codigo;
    public $nombre;
    public $nombre_corto;
    public $pvp;
    public function muestra() {
        echo "<p>" . $this->codigo . "</p>";
    }
}
```

```php
class TV extends Producto {
    public $pulgadas;
    public $tecnologia;
}
```

Programación orientada a objetos

Herencia

Funciones de utilidad en la herencia en PHP

Función	Ejemplo	Significado
`get_parent_class()`	`echo get_parent_class($t);`	Devuelve el nombre de la clase padre del objeto o la clase que se indica.
`is_subclass_of()`	`if(is_subclass_of($t,'Producto')`	Devuelve **true** si el objeto o la clase del primer parámetro, tiene como clase base a la que se indica en el segundo parámetro, o **false** en caso contrario.

Programación orientada a objetos

Constantes

- Además de métodos y propiedades, en una clase también se pueden definir constantes, utilizando la palabra const.
- Las constantes no pueden cambiar su valor (obviamente, de ahí su nombre), no usan el carácter $ y, además, su valor va siempre entre comillas y está asociado a la clase, es decir, no existe una copia del mismo en cada objeto.
- Para acceder a las constantes de una clase, se debe utilizar el nombre de la clase y el operador ::, llamado operador de resolución de ámbito (que se utiliza para acceder a los elementos de una clase).
- No es necesario que exista ningún objeto de una clase para poder acceder al valor de las constantes que defina. Además, sus nombres suelen escribirse en mayúsculas.

Programación orientada a objetos

Constantes

```php
class DB {
    const USUARIO = 'gestor';
    …
}
echo DB::USUARIO;
```

Programación orientada a objetos

Interfaces

- Un interface es como una clase vacía que solamente contiene declaraciones de métodos. Se definen utilizando la palabra interface.
- Por ejemplo, si quieres asegurarte de que todos los tipos de productos tengan un método muestra, puedes crear un interface como el siguiente.

```php
interface iMuestra {
    public function muestra();
}
```

Programación orientada a objetos

Interfaces

- Y cuando crees las subclases deberás indicar con la palabra **implements** que tienen que implementar los métodos declarados en este interface.

```
class TV extends Producto implements iMuestra {
    . . .
    public function muestra() {
        echo "<p>" . $this->pulgadas . " pulgadas</p>";
    }
    . . .
}
```

- Todos los métodos que se declaren en un interface deben ser públicos. Además de métodos, los interfaces podrán contener constantes pero no atributos.

Programación orientada a objetos

Interfaces

- Una interface es como un contrato que la clase debe cumplir. Al implementar todos los métodos declarados en el interface se asegura la interoperabilidad entre clases.
- Si sabes que una clase implementa un interface determinado, sabes qué nombre tienen sus métodos, qué parámetros les debes pasar y, probablemente, podrás averiguar fácilmente con qué objetivo han sido escritos.

Programación orientada a objetos

Interfaces

- En PHP una clase sólo puede heredar de otra, ya que no existe la herencia múltiple. Sin embargo, **sí es posible crear clases que implementen varios interfaces**, simplemente separando la lista de interfaces por comas después de la palabra “implements”.
- La única restricción es que los nombres de los métodos que se deban implementar en los distintos interfaces no coincidan.

```
class TV extends Producto implements iMuestra, Countable {
    . . .
}
```

Programación orientada a objetos

Clases y métodos abstractos

- Una de las dudas más comunes en POO, es qué solución adoptar en algunas situaciones: 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. Si van a existir varias subclases con un comportamiento común, se podría programar en los métodos de la clase abstracta. Si se opta por un interface, 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.

Programación orientada a objetos

Métodos y atributos estáticos

- Una clase puede tener atributos o métodos estáticos, también llamados a veces atributos o métodos de clase.
- Se definen utilizando la palabra clave static.

```
class Producto {
    private static $num_productos = 0;
    public static function nuevoProducto() {
        self::$num_productos++;
    }
    …
}
```

Programación orientada a objetos

Métodos y atributos estáticos

- Los atributos y métodos estáticos no pueden ser llamados desde un objeto de la clase utilizando el operador “->".
- Si el método o atributo es público, deberá accederse utilizando el nombre de la clase y el operador de resolución de ámbito.
    - Producto::nuevoProducto();
- Si es privado, como el atributo $num_productos en el ejemplo anterior, sólo se podrá acceder a él desde los métodos de la propia clase, utilizando la palabra self. De la misma forma que $this hace referencia al objeto actual, self hace referencia a la clase actual.
    - self::$num_productos ++;

Programación orientada a objetos

Serialización

- Para almacenar los objetos en la sesión del usuario, hace falta convertirlos a un formato estándar. Este proceso se llama serialización.
- En PHP, para serializar un objeto se utiliza la función serialize(). El resultado obtenido es un string que contiene un flujo de bytes, en el que se encuentran definidos todos los valores del objeto.

```php
$p = new Producto();
$a = serialize($p);
```

Programación orientada a objetos

Serialización

- Esta cadena se puede almacenar en cualquier parte, como puede ser la sesión del usuario, o una base de datos. A partir de ella, es posible reconstruir el objeto original utilizando la función unserialize().

```php
$p = unserialize($a);
```

- Si simplemente queremos almacenar un objeto en la sesión del usuario, deberíamos hacer por tanto:

```php
session_start();
$_SESSION['producto'] = serialize($p);
```

Programación orientada a objetos

Serialización

- Pero en PHP esto aún es más fácil. Los objetos que se añadan a la sesión del usuario son serializados automáticamente. Por tanto, no es necesario usar serialize() ni unserialize().

```php
session_start();
$_SESSION['producto'] = $p;
```

- Para poder deserializar un objeto, debe estar definida su clase. Al igual que antes, si lo recuperamos de la información almacenada en la sesión del usuario, no será necesario utilizar la función unserialize.

```php
session_start();
$p = $_SESSION['producto'];
```