Clases y objetos en el lenguaje PHP

Introducción

- Con PHP podemos utilizar dos estilos de programación: estructurada y orientada a objetos.
- Ejemplo conexión BBDD con mysqli:

```
// utilizando programación estructurada
$dwes = mysqli_connect(...);


// utilizando POO
$dwes = new mysqli();
$dwes->connect(...);
```

Evolución de la POO en PHP

- PHP no se diseñó con características de orientación a objetos, si no como un lenguaje de programación estructurado.
- En la versión 3 se empezaron a introducir algunos rasgos de POO.
- Se potenció en la versión 4, aunque todavía de forma muy rudimentaria.
  - Los objetos se pasan siempre por valor, no por referencia.
  - En una clase todos los miembros son públicos.
  - No existen los interfaces.
  - No existen métodos destructores.
- A partir de la versión 5, se reescribió y potenció el soporte de orientación a objetos del lenguaje.
- La versión 7 se ha reescrito de nuevo, mejora el rendimiento y añade ciertas funcionalidades interesantes como es el caso del **tipado** de las funciones.

Clases en PHP

- Similar a otros lenguajes como Java o C#.
- Se declaran con la palabra reservada **class**.
- **Entre llaves** se declaran los **atributos y métodos**, que pueden ser:
  - **Públicos** - Se pueden utilizar desde dentro y fuera de la clase.
  - **Privados** - Pueden emplearse desde la propia clase.
  - **Protegidos** - Se pueden utilizar dentro de la propia clase, las derivadas y las antecesoras.
- Por defecto son públicos.

Buenas prácticas (aunque no obligatorias)

- El nombre de la **clase** siempre **empezará por mayúsculas**.
- Siempre **definiremos cada clase en un fichero** que llamaremos NombreClase.php.
- Los **atributos siempre** serán **privados o protegidos** (crearemos **getters** y **setters** para acceder a ellos).

Instanciar objetos y acceder a sus atributos y métodos

- Al igual que en otros lenguajes usamos new para instanciar objetos:

```
$p = new Producto;
```

- Tendremos que hacer un **require del fichero de la clase**.
- Para acceder desde un objeto a sus atributos o métodos, utilizamos el operador flecha:

```
$p->nombre = 'Samsung Galaxy S';
$p->muestra(argumentos);
```

Tipado de objetos en funciones

- Podemos indicar en las funciones y métodos de qué clase deben ser los parámetros.

```
public function vendeProducto(Producto $p)
{
    …
}
```

El objeto $this

- Cuando desde un objeto se invoca un método de la clase, a éste se le pasa siempre una referencia al objeto que hizo la llamada.
- Esta referencia se almacena en la variable $this.
- Ejemplo:

```
print "<p>" . $this->codigo . "</p>";
```

Constantes de clase

- En una clase se pueden definir constantes, utilizando la palabra **const**.
- Para acceder a las constantes de una clase, **se debe utilizar el nombre de la clase** (o **self** si estamos dentro de la clase) y el operador de ámbito "**::**".

```
class DB {
    const USUARIO = 'php';
    public function muestraUsuario() {
        echo self::USUARIO; // desde dentro de la clase
    }
}
echo DB::USUARIO; // desde fuera de la clase
```

Atributos estáticos

- Una clase puede tener atributos o métodos estáticos, es decir, **atributos compartidos por todos los objeto** de esa clase.
- Se definen utilizando la palabra clave **static**.

```
class Producto
{
    private static $num_productos = 0;
    public static function nuevoProducto()
    {
        self::$num_productos++; // desde dentro de la clase
    }
}
PRODUCTO::nuevoProducto(); // desde fuera de la clase
```

Métodos mágicos

- Son métodos predefinidos que son **llamados automáticamente** en determinadas circunstancias.
- Los nombres de los métodos “__construct()”, “__destruct()”, “__call()”, “__callStatic()”, “__get()”, “__set()”, “__isset()”, “__unset()”, “__sleep()”, “__wakeup()”, “__toString()”, “__invoke()”, “__set_state()”, “**__clone()**” y “**__debugInfo()**” son mágicos en las clases PHP.
- No se puede tener métodos con estos nombres en ninguna clase a menos que se desee la funcionalidad mágica asociada a estos.

Constructor

- *void* __construct ([*mixed* $args = "" [, $...]])
  - Sólo puede haber **uno** y se llamará **__construct**.
  - Será invocada automáticamente al hacer **new**.
  - Es ideal **para inicializar** los datos del objeto antes de usarlo.

public *string* __toString(*void*)

- Permite a una clase decidir cómo comportarse cuando se le trata como un **string**.
- Por ejemplo, lo que mostraría “**echo** $obj”; debe devolver un string, si no, se emitirá error.
- No se puede lanzar una excepción desde dentro de __toString().

Herencia

- Para definir una clase que herede de otra usamos la palabra **extends**.
- La clase derivada tendrá los mismos atributos y métodos que la clase base y podrá añadir nuevos o sobrescribirlos.

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

Llamar a métodos de la clase base

- En PHP, si una clase heredada no tiene constructor propio, se llamará automáticamente al constructor de la clase base.
- Si la clase heredada define **su propio constructor**, habrá que realizar la llamada explícitamente.
- Utilizaremos la palabra reservada **parent** que hace referencia a la clase base de la clase actual (**parent::__construct()**).
- La palabra reservada **self** hace referencia a la **clase actual**.

Clases y métodos abstractos

- Para definir una clase o método abstracto utilizamos la palabra reservada **abstract**.
- **No se podrán instanciar objetos** de la clase, sólo podremos heredar de ella.

Interfaces

- Es como una clase vacía que **solamente contiene declaraciones de métodos**.
- Se definen utilizando la palabra **interface**.
- Si queremos que una clase implemente un interface, utilizaremos la palabra reservada **implements**.
- Esto obligará a que existan en la clase todos los métodos del interface.
- Una clase puede implementar más de un interface.
- Todos los **métodos** de un interface deben ser **públicos**.
- Un interface **puede incluir constantes, pero no atributos**.
- Un interface puede heredar de otro utilizando **extends**.
- PHP tiene una serie de interfaces ya definidos, por ejemplo, el interface **Countable**.