

DAWS – 2º

POO en PHP

JESÚS MOLINA HERNÁNDEZ

jmolina@florida-uni.es

Índice

1. **Qué es la POO**
2. **Conceptos básicos – Clase y Objeto / Métodos y atributos**
3. **POO en PHP – palabras clave**
4. **Encapsulación**
5. **Elementos estáticos**
6. **Constructores y destructores**
7. **Herencia**
8. **Clases y métodos abstractos / Interfaces**

Qué es la POO

La **Programación Orientada a Objetos** se basa en la idea de representar **objetos del mundo real** dentro del código de un programa.

Cada objeto tiene

- **propiedades** (características o atributos) y
- **comportamientos** (acciones o métodos)

Por qué existe?

*La POO fue creada para resolver problemas relacionados con el manejo de la **complejidad**, la **organización** del código, la **reutilización** y la **flexibilidad**.*

Sus principales ventajas incluyen modularidad, encapsulamiento, reutilización del código, mantenibilidad y escalabilidad, y ha permitido desarrollar sistemas más grandes y complejos de manera más estructurada y eficiente.

Comparativa POO vs procedural

Quiero hacer una app para gestionar mis jugadores de liga fantasy.

- Para cada jugador tengo estas **características**: nombre, apellidos, edad, precio, puntuación media, puntuación ultima jornada.
- Y se puede hacer las siguientes **acciones**: cambiar precio, poner a la venta, actualizar puntuación ultima jornada (que automáticamente actualizara la puntuación media)

Ver en el ejemplo la comparativa del código

Conceptos básicos – Clase y Objeto / Métodos y atributos


Clase: Coche

Clase Coche
arrancar, ir, parar, girar
color, velocidad, carburante

← nombre de la clase
← métodos (funciones)
← atributos (datos)

Objeto: Ferrari

coche.ferrari ← nombre del objeto



← métodos
arrancar, ir, parar, girar
← datos
rojo, 280 km/h, lleno

```
class Coche {  
    // Atributos del coche  
    public $color;  
    public $velocidad;  
    public $carburante;  
    private $encendido = false;  
  
    // Constructor para inicializar el coche  
    public function __construct($color, $velocidad, $carburante) {  
        $this->color = $color;  
        $this->velocidad = $velocidad;  
        $this->carburante = $carburante;  
    }  
  
    // Método para arrancar el coche  
    public function arrancar() {  
        // código aquí  
    }  
  
    // Método para hacer que el coche se mueva  
    public function ir($distancia) {  
        // código aquí  
    }  
}
```

```
// Crear un objeto de la clase Coche
```

```
$coche1 = new Coche( color: "Rojo", velocidad: 120, carburante: 50);
```

```
ir el coche  
ar() {
```

POO en PHP – palabras clave

- Palabra clave 'new'
 - Crea una instancia de una clase → *new MyClass ();*
- Pseudo-variable \$this
 - Hace referencia al objeto actual, solo se usa dentro de la clase.
- Operador de objeto ->
 - Se usa cuando se llama a un método o se accede a una propiedad en una instancia de objeto. También se usa con *\$this*.

Encapsulación

Determina cómo se puede acceder a las propiedades/métodos de un objeto:

- **público**: se puede acceder desde cualquier lugar.
- **protegido**: solo puede acceder la clase y las subclases
- **privado**: solo se puede acceder a la clase.

```
1  <?php
2  class MyClass {
3      public $var1 = 'propiedad pública';
4      protected $var2 = 'propiedad protegida';
5      private $var3 = 'propiedad privada';
6
7      function printHello() {
8          echo $this->var1 . '<br>';
9          echo $this->var2 . '<br>';
10         echo $this->var3 . '<br>';
11     }
12 }
13
14 $obj = new MyClass();
15 echo $obj->var1 . '<br>'; // muestra la propiedad pública
16 $obj->printHello(); // muestra todas las propiedades
17
18 echo $obj->var2; // Error Fatal
19 echo $obj->var3; // Error Fatal
```

Elementos estáticos

- Un **método** o **atributo estático** es un elemento que **pertenece a la clase en sí misma** en lugar de a las instancias u objetos individuales de esa clase. Es decir, los métodos y atributos estáticos no están ligados a ningún objeto específico, sino que son compartidos por **todas las instancias** de la clase.

Ver ejemplo_estáticos.php

Constructores y destructores

__construct ()

- es una función especial (mágica) que se llama automáticamente cuando se crea una instancia de objeto con la palabra clave 'new'.
- Un constructor puede tener cualquier número de parámetros definidos por el usuario. Los constructores se usan para inicializar el objeto
- El constructor de una super-clase puede ser llamado con `parent::__construct();`

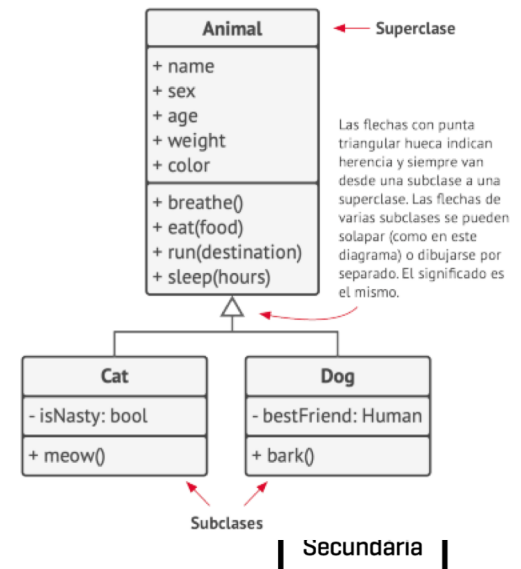
__destruct ()

- se llama automáticamente cuando el recolector de basura elimina el objeto de la memoria.

Herencia

- La herencia es el proceso de extender una clase existente (clase padre) a una nueva clase (subclase) usando la palabra clave 'extends'.
- Una subclase hereda todas las propiedades y métodos de su superclase (principal) excepto las privadas.
- La herencia se usa para la reutilización de códigos y en el polimorfismo.
- PHP no permite herencia múltiple (como máximo una superclase) pero si herencia multinivel.
- Una clase declarada con la palabra clave 'final' no se puede extender

Ver ejemplo_herencia.php



POO en PHP

Herencia > sobre-escritura de métodos

Sobre-escritura

Es el proceso en el que una subclase redefine un método de clase principal para cambiar su comportamiento.

- La declaración debe ser exactamente la misma.
- En caso de que queramos acceder a las funciones de nivel primario desde una subclase, utilizaremos 'parent::'

Palabra clave 'final'

- Una subclase no puede sobre-escribir un método declarado con la palabra clave 'final' en la super-clase

Clases y métodos abstractos

Una clase abstracta no puede ser instanciada. sirve como una **plantilla** o **base** para otras clases que heredan de ella.

- Proporcionan implementación abstracta de clase que debe ser ampliada para proporcionar un comportamiento específico
- Una definición de clase abstracta comienza con una palabra clave 'abstract'.

Los métodos abstractos son aquellos que se declaran inicialmente en una clase abstracta

- No incluyen código

Una subclase debe sobre-escribir los métodos abstractos.

Conceptos básicos – Interfaces

Diferencia entre clase abstracta e interfaz:

Aunque las clases abstractas y las interfaces tienen similitudes (ambas pueden definir métodos que deben ser implementados por las clases hijas), hay algunas diferencias clave:

Clase abstracta:

Puede tener tanto **métodos concretos** (con implementación) como **métodos abstractos**.

Puede tener atributos y estado compartido entre las subclases.

Las clases pueden heredar **solo una clase abstracta**.

Interfaz:

Solo puede tener **métodos abstractos** (sin implementación, hasta PHP 8 donde pueden tener métodos con cuerpo).

No puede tener atributos ni estado.

Una clase puede implementar **múltiples interfaces**

Claves:
interface nombre { }

class nombre implements xxx { }