UD4 – POO con PHP

2º CFGS
Desarrollo de Aplicaciones Web

2024-25

1.- Introducción a POO

El concepto de **Programación Orientada a Objetos** apareció con el lenguaje de programación **Simula** en **1962**, lanzado oficialmente en 1967.

Su auge fue gracias a **Java** (1995) y **C#** (2000) que introdujeron estos conceptos y metodologías.

En la POO todos los elementos que actúan en un programa se tratan como elementos de la vida real con sus características y comportamientos.

Estos elementos son **objetos** que se ajustan a unas definiciones llamadas **clases**.

1.- Introducción a POO

Ventajas que aporta la POO:

- **Modularidad**: permite dividir los programas en partes más pequeñas independientes entre sí. Se pueden usar esas partes en otros programas.
- **Extensibilidad**: para ampliar la funcionalidad de las clases solo hay que modificar su código mediante herencia.
- **Mantenimiento**: Gracias a la modularidad el mantenimiento es más sencillo. Es importante que cada clase esté en un archivo diferente.

Más información acerca de la POO.

1.- Introducción a POO

Características de la POO:

- **Herencia**: cuando se crea una clase a partir de otra se hereda su comportamiento y características.
- **Abstracción**: de cara al exterior la clase sólo muestra los métodos no el cómo hace las cosas.
- **Polimorfismo** y **sobrecarga**: los métodos pueden tener comportamientos diferentes según la forma de utilizarlos.
- **Encapsulación**: Están juntos los datos y el código que los usa.

1.- Introducción a POO

Clases:

- **Propiedades** (atributos): almacenan información acerca del estado del objeto al que pertenecen. Su valor puede ser diferente para diferentes objetos de una misma clase.

- **Métodos**: contienen código ejecutable y definen las acciones que realiza el objeto. Son como una función y pueden recibir parámetros y devolver valores.

- **Instancia**: cuando se tiene una clase definida y se crea un objeto de dicha clase se dice que se tiene una instancia de la clase (un elemento "real").

2.- POO en PHP

En un principio **PHP no se diseñó para la POO**.

En PHP 3 se introdujeron algunos conceptos de POO.

En PHP 4 se potenció su uso debido al auge de la POO gracias a Java y C#.

En **PHP 5 se reescribió el motor de PHP** que incluía un nuevo modelo de objetos.

Actualmente PHP soporta todas las características de la **POO excepto la herencia múltiple y la sobrecarga de métodos**.

3.- Clases en PHP

Aunque en PHP no es necesario que cada clase esté en su propio archivo .php, **es recomendable que cada clase que se declare esté en su archivo** propio cuyo nombre seguirá el patrón: **NombreClase.inc.php**.

Para la declaración de una clase se usa la palabra reservada **class**.

Los nombres de las clases deben comenzar por **mayúscula**.

Como buena práctica al programar, el código dentro de la clase debe estar bien organizado, por ello primero se indicarán las propiedades, luego los constructores y finalmente los métodos propios.

3.- Clases en PHP

Ejemplo de una clase Product → **Product.inc.php**:

```php
class Product {
    public $code;
    public $name;
    public $price;

    public function __construct($code='', $name='', $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }

    public function data() {
        return $this->name .' ('. $this->code .'): '. $this->price .' €';
    }
}
```

3.- Clases en PHP

Para acceder a las propiedades de la clase desde dentro del código de la clase siempre es necesario utilizar **$this->**

```php
public function __construct($code='', $name='', $price=0.0) {
    $this->code = $code;
    $this->name = $name;
    $this->price = $price;
}

public function data() {
    return $this->name .' ('. $this->code .'): '. $this->price .' €';
}
```

En los scripts PHP que se quiera usar la clase Product se incluirá su código:

```php
require_once($_SERVER['DOCUMENT_ROOT'] .'/includes/Product.inc.php');
```

3.- Clases en PHP

En las clases conviene tener un método que permita inicializar a los objetos, este método es el **constructor**.

Para definir un constructor se usa **__constructor** como se puede observar delante del nombre se usan dos caracteres _.

```php
public function __construct($code='', $name='', $price=0.0) {
    $this->code = $code;
    $this->name = $name;
    $this->price = $price;
}
```

3.- Clases en PHP

Como PHP no permite sobrecarga solo se puede definir un constructor.

Si se necesita usar constructores con diferente número de parámetros se puede simular la sobrecarga declarando el constructor con parámetros con valor por defecto.

```php
public function __construct($code='', $name='', $price=0.0) {
    $this->code = $code;
    $this->name = $name;
    $this->price = $price;
}
```

3.- Clases en PHP

Para **instanciar** un objeto se usa la palabra **new**:

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$tvSamsung = new Product('Samsung65');
$iphone = new Product(name:'Apple-i14', price: 1009.0);
$tablet = new Product(name:'Galaxy Tab');
```

Se usa la notación **->** para acceder a las propiedades y los métodos de los objetos:

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$ps5->name = 'PS5';
$ps5->price = 459.0;
echo $ps5->data();
```

4.- Propiedades

Se puede definir el tipo de acceso para las propiedades y los métodos de una clase.

- **public**: se puede acceder a ellos de manera directa.
- **protected**: se puede acceder desde la propia clase y las que hereden de ella.
- **private**: solo se puede acceder a ellos desde dentro de la clase.

```php
class Product {
    private $code;
    private $name;
    private $price;
```

Error por tener las propiedades privadas:

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$ps5->name = 'PS5';
$ps5->price = 459.0;
echo $ps5->data();
```

4.- Propiedades

Lo más habitual es que las propiedades se declaren **private** y en ese caso será necesario crear los métodos **get** y **set** para poder acceder a ellas.

```php
class Product {
    private $code;
    private $name;
    private $price;

    public function __construct($code='', $name='', $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }

    public function getCode() {
        return $this->code;
    }

    public function setCode($code) {
        $this->code = $code;
    }

    public function data() {
        return $this->name .' ('. $this->code .'): '. $this->price .' €';
    }
}
```

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$ps5->setCode('sony-PS5');
echo $ps5->getCode();
```

5.- Métodos mágicos

En el modelo de objetos de PHP permite crear los denominados **métodos mágicos** que facilitan la creación de métodos que realizan tareas concretas.

Los métodos mágicos comienzan con dos caracteres _ seguidos.

__construct	__destruct	__call	__callStatic
__get	**__set**	__isset	__unset
__sleep	__wakeup	__serialize	__unserialize
__toString	__invoke	__set_state	__clone
__debugInfo			

Aunque dan más flexibilidad, los métodos mágicos suelen ser **más lentos que los normales**.

5.- Métodos mágicos

La mejor forma de comprender los métodos mágicos es viendo un ejemplo de **__get** y **__set**:

```php
class Product {
    private $code;
    private $name;
    private $price;

    public function __construct($code='', $name='', $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }

    public function __set ($property, $value) {
        $this->$property = $value;
    }

    public function __get ($property) {
        return $this->$property;
    }

    public function __toString() {
        return $this->name .' ('. $this->code .'): '. $this->price .' €';
    }
}
```

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$ps5->code = 'sony-PS5';
echo $ps5->name;
```

Es importante destacar que se usa la variable $property para el acceso a una propiedad determinada. La variable $property almacena el nombre de la propiedad a la que se accede.

5.- Métodos mágicos

Como en PHP no se declaran las variables, con __get y __set se pueden crear propiedades nuevas en las clases lo cual puede ser un problema de seguridad y se debe controlar.

```php
class Product {
    private $code;
    private $name;
    private $price;

    public function __construct($code='', $name='', $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }

    public function __set ($property, $value) {
        $this->$property = $value;
    }

    public function __get ($property) {
        return $this->$property;
    }

    public function __toString() {
        return $this->name .' ('. $this->code .'): '. $this->price .' €';
    }
}
```

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
//Se crea una nueva propiedad en el objeto
$ps5->newProperty = '2023';
echo $ps5->newProperty;
```

Desde PHP 8.2 esta funcionalidad aunque sigue operativa está obsoleta y muestra un aviso.

Deprecated: Creation of dynamic property

5.- Métodos mágicos

Por esta razón en los métodos mágicos __get y __set solo se debe acceder a las propiedades si la propiedad a la que se quiere acceder existe:

```php
public function __set ($property, $value) {
    if(isset($this->$property))
        $this->$property = $value;
}
public function __get ($property) {
    if(isset($this->$property))
        return $this->$property;
}
```

5.- Métodos mágicos

EL método __toString aunque está disponible no se suele utilizar debido a que no es recomendable mezclar el modelo de datos con la visualización de los mismos.

```php
class Product {
    private $code;
    private $name;
    private $price;

    public function __construct($code='', $name='', $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }

    public function __set ($property, $value) {
        if (isset($this->$property))
            $this->$property = $value;
    }

    public function __get ($property) {
        if (isset($this->$property))
            return $this->$property;
    }

    public function __toString() {
        return $this->name . ': '. $this->price .' €';
    }
}
```

5.- Métodos mágicos

Los métodos mágicos __get y __set dan mucha versatilidad y compactan mucho el código aunque ofrecen peor rendimiento.

Se pueden usar sin ningún problema, pero cuando hay que hacer validaciones para cada propiedad en los setters la lógica del código se puede volver caótica y en esos casos se recomienda no usar __set.

```php
public function __set ($property, $value) {
    if (isset($this->$property)) {
        if ($property==='code' && preg_match('/^[0-9]{5}$/', $value)) {
            $this->$property = $value;
        } else if ($property==='name' && preg_match('/^[\s\S]{5,20}$/', $value)) {
            $this->$property = $value;
        } else if ($property==='price' && preg_match('/^\d+(\.\d{2})?$/', $value)) {
            $this->$property = $value;
        }
    }
}
```

6.- Propiedades y métodos estáticos

Las **propiedades** y los **métodos estáticos**, también llamados **de clase**, se pueden utilizar sin instanciar ningún objeto.

Para definirlos se usa la palabra **static**.

Se utiliza la notación **::** para acceder a las propiedades o usar los métodos.

Para usarlos desde dentro de la clase se usa la notación **self::*propiedad***

Si son **public** se podrá acceder a ellos desde fuera de la clase usando el nombre de la clase y la notación ***NombreClase*::*propiedad***.

Si son **private** solo se podrá acceder desde dentro de la clase (declaración).

6.- Propiedades y métodos estáticos

```php
class Product {
    private static $quantityOfProducts = 0;
    public static $existsAnyproduct = false;
    private $code;
    private $name;
    private $price;

    public function __construct($code='', $name='', $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
        self::$quantityOfProducts++;
        self::$existsAnyproduct = true;
    }
}
```

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
echo 'cantidad: '. Product::$existsAnyproduct;
```

7.- Uso de objetos

El uso de clases y objetos en PHP es muy similar a su uso en Java.

Si se quiere comprobar si un objeto es de una clase determinada se puede usar la instrucción **instanceof** igual que en Java.

```php
if ($consola instanceof Product) {

}
```

PHP también hay una serie de funciones útiles para el uso de objetos:

get_class class_exists get_class_methods …

En la documentación se pueden ver todas todas.

7.- Uso de objetos

Igual que ocurre con las funciones, se puede indicar el tipo de dato de los parámetros y el tipo de dato de retorno tanto en las propiedades como en los métodos.

Se aconseja que a la hora de definir clases se utilicen estas técnicas ya que así los scripts son más estrictos y se evitan posibles errores.

En los métodos mágicos **__get y __set no se debe indicar el tipo de dato ya que si se indica no funcionarán correctamente**.

7.- Uso de objetos

Las **variables que guardan objetos realmente almacenan la dirección de memoria** donde se guardan los datos del objeto.

Por esa razón cuando se realizan las siguientes instrucciones no se está realizando una copia del objeto si no que las dos variables son lo mismo.

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$ps4 = $ps5;
$ps4->name = 'PS4';
echo $ps5->name .' - '. $ps4->name;
// Se mostrará: PS4 - PS4
```

Para realizar una copia de un objeto es necesario utilizar la función **clone**.

```php
$ps4 = clone($ps5);
```

7.- Uso de objetos

Para comparar objetos se usan el operador **==** para saber si sus propiedades tienen los mismos valores y **===** para saber si son el mismo objeto.

```php
$ps5 = new Product('sony-ps5', 'PS5', 499.99);
$ps4 = clone($ps5);
```

El resultado de:

$ps4 == $ps5 es **true** porque son dos copias idénticas.

$ps4 === $ps5 es **false** porque son dos objetos diferentes.

Práctica

Actividad 1:

Login.

8.- Herencia

La herencia permite definir clases en base a otras ya existentes creando así una jerarquía de clases.

Al definir clases nuevas en base a otras se puede ampliar su funcionalidad.

A las nuevas clases se les llama **subclases** y a las clases que sirven de base se les llama **superclases**.

Al utilizar la herencia, además de los modificadores de visibilidad **public** y **private**, se puede hacer uso del modificador **protected**.

8.- Herencia

Si todos los productos tuvieran solo las propiedades código, nombre y precio la siguiente clase sería suficiente.

Al incorporar productos como televisores, altavoces, móviles… estas propiedades son comunes pero cada tipo tiene además otras propiedades interesantes.

```php
class Product {
    private $code;
    private $name;
    private $price;

    public function __construct(string $code='', string $name='', float $price=0.0) {
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }

    public function __toString() {
        return $this->name .' ('. $this->code .'): '. $this->price .' €';
    }
}
```

8.- Herencia

Mediante la palabra reservada **extends** se definen subclases.

Si se quiere llamar a un método de la superclase se debe usar la palabra reservada **parent** seguida del operador **::**

```php
class TV extends Product {
    public $inches;
    public $tecnology;

    public function __construct(string $code, string $name, float $price, float $inches, string $tecnology) {
        parent::__construct($code, $name, $price);
        $this->inches = $inches;
        $this->tecnology = $tecnology;
    }

    public function __toString(){
        return parent::__toString() .' '.
                $this->inches .'" '.
                $this->tecnology;
    }
}
```

```php
class Speaker extends Product {
    public $power;
    public $chanels;

    public function __construct(string $code, string $name, float $price, int $power, float $chanels) {
        parent::__construct($code, $name, $price);
        $this->power = $power;
        $this->chanels = $chanels;
    }

    public function __toString(){
        return parent::__toString() .' '.
                $this->power .'W '.
                $this->chanels .'canales';
    }
}
```

```php
$tv = new TV('Sm13734', 'Samsung', 759.0, 65, 'QLED');

$soundBar = new Speaker('LG soundBar', 'SN8Y', 300.0, 120, 2.1);
```

8.- Herencia

En la documentación se pueden consultar las funciones para trabajar con objetos.

Entre esas funciones se encuentran las siguientes dos que facilitan el trabajo cuando se define herencia:

get_parent_class → obtiene la clase padre del objeto.

is_subclass_of → indica si es objeto es una subclase de otra clase.

9.- Sobrecarga

La **sobrecarga** en POO permite definir **métodos con el mismo nombre** pero con diferente cantidad de parámetros y tipos de los mismos.

Al realizar la llamada a un método sobrecargado dependiendo de los parámetros se ejecutara una "versión" u otra del método.

PHP no permite la sobrecarga, para simular la sobrecarga se puede utilizar alguna de las técnicas estudiadas en la unidad anterior:

- Argumentos con nombre.
- Cantidad de argumentos variables.

10.- Clases y métodos abstractos

Las clases **abstractas** son aquellas que no permiten instanciar objetos de ellas, así que su función es la de servir de superclase.

Los métodos de una clase también se pueden definir como abstractos, en ese caso el método no contendrá código y será obligado declararlos y definirlos en las subclases.

```php
abstract class Shape {
    protected $color;

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

    abstract public function draw();
    abstract public function getArea();
}
```

```php
class Square extends Shape {
    private $side;

    public function __construct($color, $side) {
        parent::__construct($color);
        $this->side = $side;
    }

    public function draw() {
        // instrucciones
    }

    public function getArea(): float {
        return pow($this->side, 2);
    }
}
```

10.- Clases y métodos abstractos

Si no se quiere permitir la herencia se debe usar la palabra **final** que sirve tanto para clases como para métodos.

```
final class Car {
    ...
}
```

10.- Interfaces

Las **interfaces** son clases que solo contienen declaraciones de métodos sin definir su código.

Las clases definidas como interfaces se utilizan como plantillas para otras clases, de esta manera estas clases deberán definir todos los métodos declarados en la interfaz.

```php
interface ShowData {
    public function show();
}

class Laptop extends Product implements ShowData {
    public function show() {
        // Instrucciones
    }
}
```

11.- Traits

Una de las características típicas de la herencia es poder reutilizar código ya codificado en la superclase.

En PHP no se permite la herencia múltiple (pocos lenguajes lo permiten).

PHP dispone de los **trait** que permiten reutilizar código reduciendo las limitaciones de la herencia simple.

Los **traits** son similares a las clases pero **no se permite instanciar objetos** y **solo permiten agrupar funcionalidades**.

11.- Traits

```php
trait Greetings {
    public function helloWord() {
        echo 'Hola mundo!';
    }
    public function greetingObiWan() {
        echo 'Hello there!';
    }
}

trait Goodbye {
    public function goodbye() {
        echo 'Hasta la vista baby!';
    }
}
```

```php
class Jedi {
    use Greetings;
    use Goodbye;
}

$obiwan = new Jedi; // también: new Jedi()
echo $obiwan->greetingObiWan();
echo '<br>';
echo $obiwan->goodbye();
```

12.- Excepciones

PHP dispone de un modelo de excepciones (que son clases) similar al de otros lenguajes de programación mediante bloques:

try – catch – finally

Un bloque **try** debe tener un bloque **catch** y/o un bloque **finally**.

Un bloque **try** puede tener varios bloques **catch**.

Se pueden lanzar excepciones con **throw new *TIPO_EXCEPCION***. (Tipos)

También se pueden crear subclases de las excepciones para crear excepciones propias.

12.- Excepciones

```php
$num1 = 13;
$num2 = 0;

try {
    $res = $num1 .'/'. $num2 .'='. $num1/$num2;
} catch (DivisionByZeroError $e) {
    echo 'Se ha producido un error: ' . $e->getMessage();
}
```

Práctica

Actividad 2:

Inventario héroe.