UT7. Programación Orientada a Objetos

DAW - Desarrollo Web Entorno Servidor
Fernando Galindo

Contenidos

1. Introducción
2. Principios de POO
3. Clases en PHP
4. Patrones de diseño

1. Introducción

La programación orientada a objetos (*Programing Oriented Objects POO*), es una metodología que se apoya en los objetos

Un objeto es una estructura que contiene tanto el código, como los datos que maneja

PHP en las primeras versiones no daba soporte a objetos, no fue hasta PHP3 cuando se introdujeron las primeras características y fuertemente desarrollado en PHP5

2.Principios de POO

Recordemos que un objeto está definido en una clase en la que tenemos

- Métodos
- Atributos o propiedades

La creación de una clase se llama una instancia

```
$conexion = new mysqli("localhost", "r
```

Los pilares de POO son:

- Herencia
- Abstracción
- Polimorfismo
- Encapsulación

Veamos nuestra primera clase

```php
<?php
class alumno{
  protected $dni; // Las variables protected pueden ser accedidas por la clase y descendientes
  protected $nombre;
  private $nota;

  /*Podemos dar valores por defecto al definir nuestra clase*/
  public function __construct(string $dni="", string $nombre = "",int $nota=0) {
      $this->dni=$dni;
      $this->nombre=$nombre;
      $this->nota=$nota;
  }

  public function mostrar(){
      echo $this->dni." ".$this->nombre." ".$this->nota."<br> ";
  }
}
?>
```

3. Clases en PHP

Los atributos privados necesitan ser accedidos y definidos a través de métodos get y set

```
public function setNota($nueva_nota){
    $this->nota=$nueva_nota;
}

public function getNota(){
    return $this->nota;
}
```

En PHP disponemos de los métodos mágicos _ _get y _ _set que permiten trabajar con las propiedades de las variables

https://www.php.net/manual/en/language.oop5.magic.php

Métodos mágicos

Podremos acceder a cualquier propiedad independiente de su visibilidad

```
public function __get($atributo){
    if(property_exists($this, $atributo)) {
        return $this->$atributo;  //Fijaros que en este caso $atributo es una variable
    }
}
```

Podremos definir cualquier propiedad independiente de su visibilidad

```
public function __set($propiedad, $valor){
    if(property_exists($this, $propiedad)) {
        $this->$propiedad = $valor;
    }
}
```

Utilizando nuestra clase

```php
<?php
require_once 'alumno.php';

$a=new alumno("12","Juan",7);

$a->mostrar();

$a->setNota(9);

$a->mostrar();

echo $a->nombre;
?>
```

12 Juan 7
12 Juan 9

3. Clases en PHP

Las clases podemos definir constantes, normalmente se escriben con mayúsculas y accedemos con :: a su valor.

No debemos confundirlo con un método o un atributo estático

```php
<?php

class DB {
 const USUARIO = 'dwes';
 …
}
echo DB::USUARIO;
?>
```

```php
<?php

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

?>
```

3. Clases en PHP

Los atributos estáticos no pueden ser llamados desde fuera de la clase

```
$miProducto->num_producto;  //Dará un error.
```

Los métodos estáticos se llaman con el operador ::

```
Producto::nuevoProducto();
```

No es necesario instanciar la clase para poder llamar al método estático, tiene aplicaciones muy determinadas

3. Clases en PHP

Para crear una instancia

```
$miProducto = new Producto ();
```

Podemos comprobar el tipo de la variable si coincide con la clase, fijaros que **NO** es una función

```
if ($miProducto instanceof Producto) {
…
}
```

3. Clases en PHP

Resulta útil conocer algunas funciones para trabajar con objetos

- class_exists
- get_class
- get_object_vars
- method_exists
- property_exists

https://www.php.net/manual/en/ref.classobj.php

3. Clases en PHP

Clonación, resulta útil crear copias de objetos pero que cada uno tenga sus propias propiedades
La clonación se realiza con el método mágico _ _clone()

Cuando se clona un objeto, PHP llevará a cabo una copia superficial de las propiedades del objeto. Las propiedades que sean referencias a otras variables, mantendrán las referencias.

- Veamos el ejemplo https://www.php.net/manual/es/language.oop5.cloning.php

Comparación entre objetos

- == Comprueba que los atributos de los objetos son iguales
- === Comprueba que el objeto sea el mismo

- Veamos el ejemplo
  https://www.php.net/manual/es/language.oop5.object-comparison.php

Serialización

En ocasiones resulta útil almacenar la información de un objeto para que transferirlo o almacenarlo para recuperarlo posteriormente.

Para almacenar los objetos podemos utilizar el método serialize que convierte el objeto a un string

Podemos recuperar la información del objeto utilizando unserialize

Sin embargo al utilizar sesiones ya realiza la serialización y deserialización de forma automática

```php
require_once 'alumno.php';
session_start();
if (isset($_SESSION['alumno'])){
   echo "El alumno existe<br>";
   $b=unserialize($_SESSION['alumno']);
   $b->mostrar();
   var_dump($b);
}else{
   echo "Creo un alumno<br>";
   $a=new alumno("12","Juan",7);
   $_SESSION['alumno']=serialize($a);
}
```

3. Clases en PHP

La herencia es un recurso usado en programación y establecido en PHP que permite establecer relaciones entre objetos

Esto es útil para la definición y abstracción de la funcionalidad y permite la implementación de funcionalidad adicional en objetos similares sin la necesidad de reimplementar toda la funcionalidad compartida

```
class Animal{
   private $nombre;
   function __toString(){
       return "Hola soy un ".get_class($this)." y me
llamo $this->nombre <br>";
   }
   function __construct(String $nombre){
       $this->nombre=$nombre;
   }
}
```

3. Clases en PHP

```php
class Animal{
   private $nombre;
   function __toString(){
       return "Hola soy un ".get_class($this)." y
me llamo $this->nombre <br>";
   }
   function __construct(String $nombre){
       $this->nombre=$nombre;
   }
}
```

- Métodos mágicos __toString
- Llamada al constructor

```php
class Perro extends Animal{
   private $alimento;
   function come(String $comida){
       if($this->alimento=='$comida'){
           return "Gracias, muy rico";
       }else{
           return "No me gusta, sólo como $this-
>alimento<br>";
       }
   }
   function __construct(String $nombre,String $alimento){
       parent::__construct($nombre);
       $this->alimento=$alimento;
   }
}
$a=new Animal("Garfield");
$b=new Perro("Snoopy","carne");
echo $a;
echo $b;
```

Las propiedades privadas no son accesibles desde la clase hijo, sólo las public o protected

```php
class Perro extends Animal{
    …
    function saluda(){
        return "Me llamo $this->nombre<br>";
    }
}
echo $b->saluda();
```

Warning: Undefined property: Perro::$saluda in C:\xampp\htdocs\ut5\herencia.php on line *36*

Call Stack

#	Time	Memory	Function	Location
1	0.0008	408664	{main}( )	...\herencia.php:0

```php
class Animal{
    protected $nombre;
    …
}
```

Clases abstractas son aquellas que no se pueden instanciar

```php
abstract class Animal{
   protected $nombre;
   function __toString(){
       return "Hola soy un ".get_class($this)." y me llamo $this->nombre <br>";
   }
   function __construct(String $nombre){
       $this->nombre=$nombre;
   }
}
$a=new Animal("Garfield"); //Fatal error
$b=new Perro("Snoopy","carne");
```

Fatal error: Uncaught Error: Cannot instantiate abstract class Animal in C:\xampp\htdocs\ut5\herencia.php on line *31*

Error: Cannot instantiate abstract class Animal in C:\xampp\htdocs\ut5\herencia.php on line *31*

Call Stack

#	Time	Memory	Function	Location
1	0.0010	409360	{main}( )	...\herencia.php:0

Las interfaces de objetos permiten crear código con el cual especificar qué métodos deben ser implementados por una clase, sin tener que definir el código de los mismos

```php
interface Animal{
    function __toString();
    function come($alimento);
}
```

```php
class Perro implements Animal{
    protected $nombre;
    private $alimento;
    function come($alimento){
        if($this->alimento==$alimento){
            return "Gracias, muy rico";
        }else{
            return "No me gusta, sólo como $this->alimento<br>";
        }
    }
    function __construct(String $nombre,String $alimento){
        $this->nombre=$nombre;
        $this->alimento=$alimento;
    }
    function __toString(){
        return "Me llamo $this->nombre<br>";
    }
}
```

3. Clases en PHP

Hemos mencionado que los objetos se pueden serializar para poder almacenar información y recuperarla posteriormente

¿Podríamos transferir y recuperar dicha información por la Web?

La función `file_get_contents` lee un fichero completo a una cadena, con la particularidad de que el fichero puede ser una URL

```
$p=file_get_contents("https://marca.com");
echo $p;
```

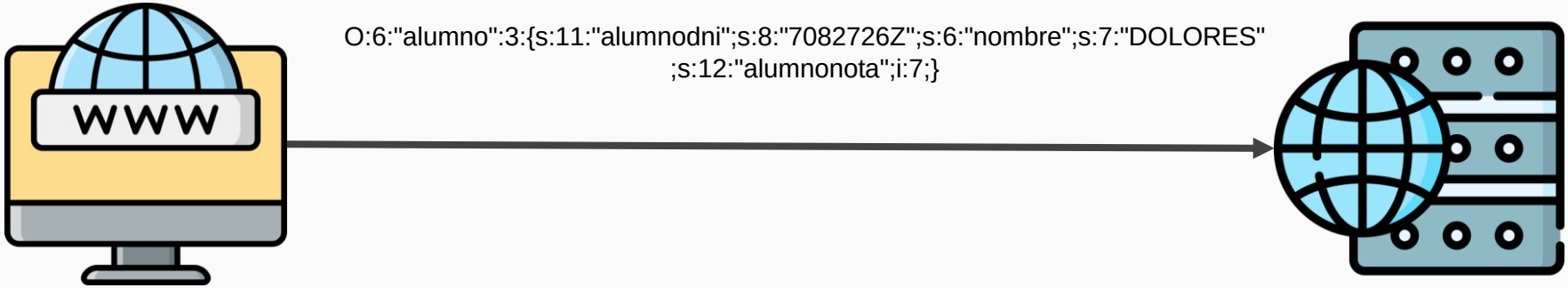
La función `file` lee un fichero completo a un array

```
$p=file("https://marca.com");
foreach ($p as $linea){
    echo $linea;
}
```

Utilizando está técnica podríamos crear una página que cree un objeto serializado y tener otra página que lo procese

serialize($objeto)

unserialize($web)



4. Patrón de diseño

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces

Un **patrón** de diseño resulta ser una s**olución a un problema de diseño**. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su **efectividad** resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser **reutilizable**, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

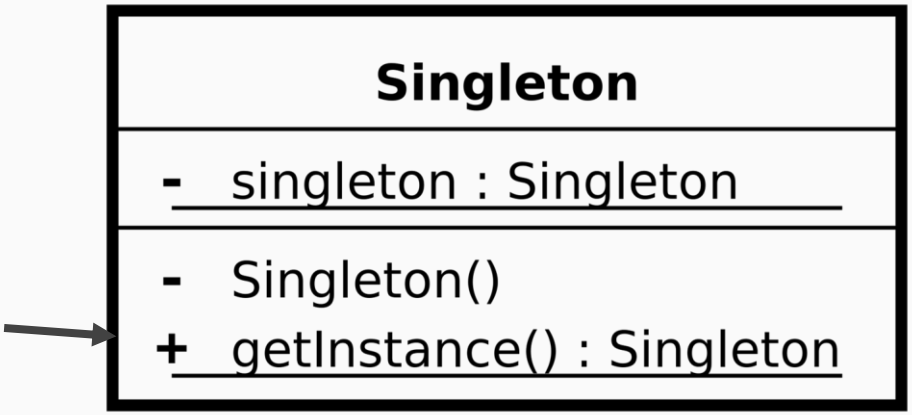
4. Patrón de diseño

Existen muchos tipos de patrones cada uno destino a diferentes ámbitos de diseño

- Singleton, nos asegura tener una única instancia de un objeto
- Observador, un cambio en el objeto notifica a todos sus dependientes
- Strategy, nos permite modificar el algoritmo sin necesidad de modificar el nombre
- Adaptador, nos permite crear un
- MVC Modelo Vista Controlador, es un patrón de diseño que nos permite dividir una aplicación en capas, donde cada capa tendrá una funcionalidad concreta
- Cadena de responsabilidad

Singleton

El patrón singleton restringe la creación de objetos a un único objeto y asegura que sólo hay una instancia de dicho objeto proporcionando un método de acceso único



podemos añadir más métodos

4. Patrón de diseño

```php
<?php
class Singleton{
    // Contenedor Instancia de la clase
    private static $instance = null;

    // Constructor privado, previene la creación de instancias vía new
    private function __construct() { }

    // Clonación no permitida
    private function __clone() { }

    // Método singleton
    public static function getInstance(){
        if (null === self::$instance) {
            self::$instance = new Singleton();
        }
        return self::$instance;
    }
}
?>
```

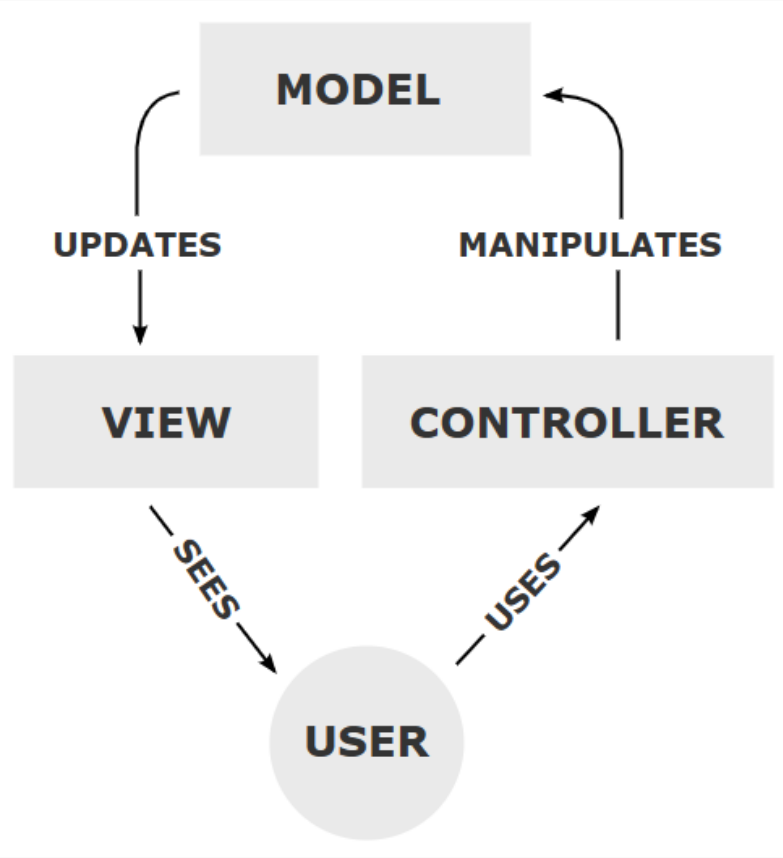
4. Patrón de diseño

Modelo Vista Controlador (MVC), es un patrón de diseño que se encarga de separar el módelo de negocio, de la capa de presentación a los usuarios

- Modelo, es la representación de la información y gestiona el acceso a los datos, consultas, actualizaciones, privilegios, etc.
- Vista, representa la información de forma adecuada a los usuarios
- Controlador, responde a los eventos del usuario e invoca peticiones al modelo

4. Patrón de diseño

Una página podrá tener diferentes VISTAS, MODELOS y CONTROLADORES



4. Patrón de diseño

Recordemos el ejemplo del Banco, con clientes y empleados

- Controlador, creará un array de los empleados (clientes), realizando una petición al modelo
- Vista, mostrará la tabla con los clientes del array pasado de controlador y los botones de acción, los botones de acción deberán realizar llamadas al controlador
- Modelo, realizará las consultas a la BD y devolverá la información pedida por el controlador

4. Patrón de diseño

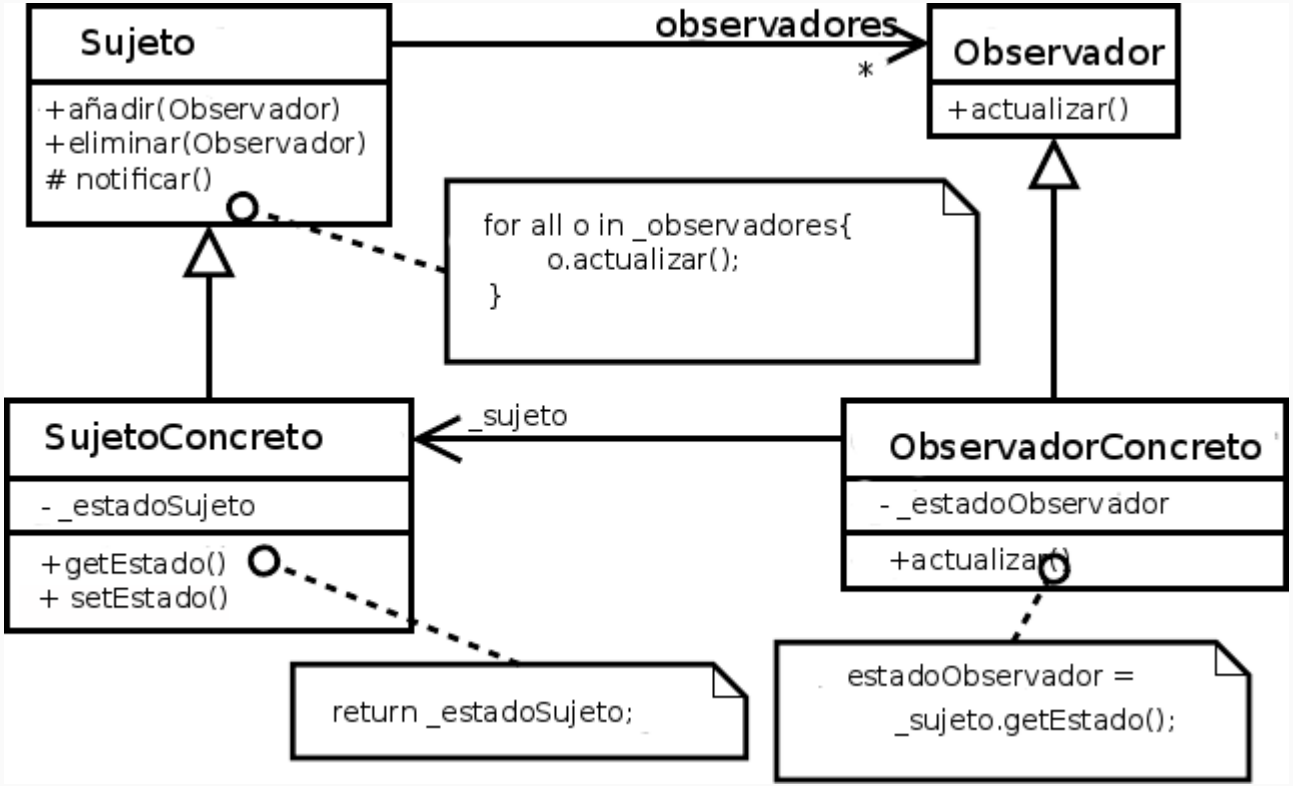
Patrón Observador, define una dependencia entre un objeto (sujeto) y diferentes objetos (observadores) que deben ser notificados cada vez que hay un cambio en el sujeto

Imagina que tienes dos tipos de objetos: un objeto Alumno y un objeto Profesor. El Alumno está muy interesado en las notas del módulo que estarán muy pronto disponibles.

El Alumno puede dirigirse al centro y preguntar al Profesor cada día para comprobar si las notas están disponibles. Lo más seguro es que la mayoría de los días las notas no estén disponibles y por tanto los viajes serán en vano.

Por otro lado, el Profesor puede avisar a todos los alumnos del curso sobre las notas, pero ciertos Alumnos bien no están matriculados en dicho módulo o bien no quieren recibir las notas.

4. Patrón de diseño



PHP dispone de interfaces que nos permiten definir la estructura que deben implementar los sujetos y observadores

`SplObserver`

`SplSubject`