Programación Orientada a Objetos en PHP

El crear una página o aplicación web haciendo uso de una correcta Programación Orientada a Objetos nos facilitará en gran medida la organización y reutilización del código y la depuración de errores, con lo cual nuestros proyectos web podrán ser mantenidos y ampliados de forma mucho más sencilla.

La programación orientada a objetos es una metodología de programación basada en objetos.

La estructura de los objetos se define en las clases. Las clases se componen de atributos y/o métodos.

**Métodos** un método es como una función, código que realiza una función determinada. Puede recibir parámetros y devolver valores.

**Atributos o propiedades**. Almacenan información del objeto.

Los pilares fundamentales de la POO son:

* **Herencia**. Es el proceso de crear una clase a partir de otra, heredando su comportamiento y características y pudiendo redefinirlos.
* **Abstracción**. Hace referencia a que cada clase oculta en su interior las peculiaridades de su implementación, y presenta al exterior una serie de métodos (interface) cuyo comportamiento está bien definido.
* **Polimorfismo**. Un mismo método puede tener comportamientos distintos en función del objeto con que se utilice.

El polimorfismo consiste en que dos clases diferentes, generalmente descendientes (que heredan de otra), tienen métodos iguales pero que en cada clase realizan acciones diferentes.

* **Encapsulación**. Tiene que ver con el ocultamiento de atributos y métodos.

La encapsulación es un mecanismo que consiste en organizar datos y métodos de una estructura, conciliando el modo en que el objeto se implementa, es decir, evitando el acceso a datos por cualquier otro medio distinto a los especificados. Por lo tanto, la encapsulación garantiza la integridad de los datos que contiene un objeto.

Ejercicio 1.-

<html>

<head>

<title> **P O O** </title>

<meta charset=**"utf-8"** />

</head>

<body>

<?php

**class** Persona {

**private** $nombre;

**private** $provincia;

**private** $sueldo;

**const** COMISION=256; // las constantes no usan el carácter $

//Los atributos constantes pueden almacenar: enteros, flotantes, cadenas, arrays.

//No pueden almacenar objetos

//Si accedemos desde dentro de la clase utilizaremos la palabra reservada self seguida de '::' y del nombre de la constante.

//A partir de PHP 5.3.0. nombre de la clase::nombre de la constante.

// Si accedemos desde fuera de la clase, ya que las constantes se consideran atributos públicos,

//utilizaremos el nombre de la clase seguido de '::' y el nombre de la constante.

//A partir de PHP 7.1.0 las constantes pueden tener visibilidad: pública (pero no se puede poner public),private y protected

**public** **function** **\_\_construct**($nom,$prov,$sueldo){

$this->nombre=$nom;

$this->provincia=$prov;

$this->sueldo=$sueldo;

}

**public** **function** visualizar()

{

**echo** $this->nombre." &nbsp;&nbsp;". $this->provincia;

$aumento=$this->subirSueldo(500);

**echo** " tiene un sueldo de: ".$aumento;

**echo** '<br>';

$this->sueldo=$aumento;

}

**public** **function** subirSueldo($aumento){

$totalSueldo=$this->sueldo+$aumento;

**return** $totalSueldo;

}

**public** **function** sueldoMasComision(){

**echo** "sueldo + comisión: " .(Persona::COMISION+$this->sueldo); //también self::COMISION

}

**public** **function** **\_\_destruct**() {//Es opcional. Sólo debemos crearlo si deseamos hacer alguna cosa cuando un objeto se elimine de la memoria.

**echo** '<br>';

**echo** "Borrando ". $this->nombre ;

}

}

$per1=**new** Persona('Jon','Bizkaia',1500.60);

$per1->visualizar();

$per2=**new** Persona('Ane','Bizkaia',1700.32);

$per2->visualizar();

$per2->sueldoMasComision();

?>

</body>

</html>

Ejercicio 2.-

<html>

<head>

<title>**Herencia**</title>

</head>

<body>

<?php

**class** Operacion {

**protected** $n1;

**protected** $n2;

**protected** $resultado;

**public** **function** valorN1($v)

{

$this->n1=$v;

}

**public** **function** valorN2($v)

{

$this->n2=$v;

}

**public** **function** visualizar()

{

**echo** $this->resultado.'<br>';

}

}

**class** Suma **extends** Operacion{

**public** **function** operar()

{

$this->resultado=$this->n1+$this->n2;

}

}

**class** Resta **extends** Operacion{

**public** **function** operar()

{

$this->resultado=$this->n1-$this->n2;

}

}

$suma=**new** Suma();

$suma->valorN1(10);

$suma->valorN2(10);

$suma->operar();

**echo** 'El resultado de la suma de 10+10 es:';

$suma->visualizar();

$resta=**new** Resta();

$resta->valorN1(10);

$resta->valorN2(5);

$resta->operar();

**echo** 'El resultado de la diferencia de 10-5 es:';

$resta->visualizar();

?>

</body>

</html>

Ejercicio 3.-

<html>

<head>

<title>**Ejercicio**</title>

</head>

<body>

<?php

**class** Persona {

**protected** $nombre;

**protected** $edad;

**public** **function** **\_\_construct**($nombre,$edad)

{

$this->nombre=$nombre;

$this->edad=$edad;

}

**public** **function** visualizar()

{

**echo** 'Nombre:'.$this->nombre.'<br>';

**echo** 'Edad:'.$this->edad.'<br>';

}

}

**class** Empleado **extends** Persona{

**private** $sueldo;

**public** **function** **\_\_construct**($nombre,$edad,$sueldo)

{

**parent**::**\_\_construct**($nombre, $edad);

$this->sueldo=$sueldo;

}

**public** **function** visualizar(){

**parent**::visualizar();

**echo** 'Sueldo:'.$this->sueldo.'<br>';

}

}

$persona1=**new** Persona('Jon Arrieta',34);

**echo** 'Datos personales de la persona:<br>';

$persona1->visualizar();

$empleado1=**New** Empleado('Argiñe Bengoa',22,3000);

**echo** 'Datos personales y sueldo del empleado:<br>';

$empleado1->visualizar();

?>

</body>

</html>

Resultado:

Datos personales de la persona:  
Nombre:Jon Arrieta  
Edad:34  
Datos personales y sueldo del empleado:  
Nombre:Argiñe Bengoa  
Edad:22  
Sueldo:3000

Ejercicio 4.-

Una clase abstracta tiene por objetivo agrupar atributos y métodos que luego serán heredados por otras subclases.

La ventaja de definir clases abstractas es que se producirá un error en tiempo de ejecución si queremos definir un objeto de dicha clase.

Por ejemplo en el Ejercicio 2.- clases Operación, Suma y Resta. No tiene sentido definir objetos de la clase Operación, por lo tanto ésta clase se puede declarar abstracta para no poder crear instancias de Operación.

abstract class Operacion {

…

…

}

Ejercicio 5.-

Si queremos que las subclases implementen comportamientos obligatoriamente podemos definir métodos abstractos.

Un método abstracto se declara en una clase pero no se implementa.

En nuestra clase Operacion tiene sentido declarar un método abstracto: operar(). Esto hará que todas las clases que hereden de la clase Operación deban implementar el método operar().

abstract class Operacion {

…

…

public abstract function operar();

}

Ejercicio 6.-

Clase final no se puede heredar.

final class Suma extends Operacion{ …..}

Método final ninguna subclase puede sobreescribir dicho método.

public final function visualizar() {

echo $this->resultado.'<br>';

}

Ejercicio 7.-

Los atributos estáticos de una clase se utilizan para guardar información general sobre la misma.

Las clases matemáticas suelen tener métodos estáticos para realizar logaritmos o raíces cuadradas. No tiene sentido crear un objeto si lo único que queremos es realizar una operación matemática.

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();

<html>

<head>

<title>**Métodos Estáticos**</title>

<meta charset=**"utf-8"** />

</head>

<body>

<?php

**class** Cadena {

**public** **static** **function** largo($cad)

{

**return** strlen($cad);

}

**public** **static** **function** mayusculas($cad)

{

**return** **strtoupper**($cad);

}

**public** **static** **function** minusculas($cad)

{

**return** **strtolower**($cad);

}

}

$c='Hola';

**echo** 'Cadena original:'.$c;

**echo** '<br>';

**echo** 'Largo:'.Cadena::largo($c);

**echo** '<br>';

**echo** 'En mayúsculas:'.Cadena::mayusculas($c);

**echo** '<br>';

**echo** 'En minúsculas:'.Cadena::minusculas($c);

?>

</body>

</html>

Ejercicio 8.-

Interfaces.- Un interface es como una clase vacía que solamente contiene cabeceras de métodos (siempre públicos), también puede contener constantes pero no variables.

interface MiInterfaz{

public function visualizar();

}

Si queremos asegurarnos que una clase contenga un método concreto crearemos una interface.

Una clase sólo puede heredar de una clase, mientras que en una misma clase se pueden implementar varias interfaces.

class MiClase implements MiInterfaz{

}

Separados por comas podemos poner más interfaces.

Un interface también puede heredar de otro interface con la palabra extends.

Ejercicio 9.-

* Crear una Interface con los métodos getters y setters para nombre, apellidos y género. Los setters recibirán el parámetro correspondiente.Y los getters retornaran el parámetro correspondiente.
* En otro .php crear una clase Persona que implemente la interface anterior.(Este .php debe llegar el include o require del .php anterior)
* En otro .php crear una instancia de Persona y llamar a los métodos getters y setters. .(Este .php debe llegar el include o require del .php anterior)

Ejercicio 10:

Crear la bd llamada: estudiantes con la tabla estudiantes: id, nombre y telefono.

\*\*\*PHP Procedimental\*\*\*\*

<!DOCTYPE html><html>

<head>

<meta charset=**"utf-8"**>

</head>

<body>

<?php

$conexion=**mysqli\_connect**("localhost","root","", "estudiantes") **or** **die**("Error en la conexion");

$est=**array**();

$sql="select \* from estudiantes";

$res=**mysqli\_query**($conexion,$sql);

**while**($reg =**mysqli\_fetch\_assoc**($res)){

//vamos a guardar cada registro como un array asociativo

//mysqli\_fetch\_assoc($res) es lo mismo que mysqli\_fetch\_array($res,MYSQLI\_ASSOC)

$est[]=$reg;

}

**for**($i=0; $i<**sizeof**($est);$i++){

$id=$est[$i]['id'];

$nom=$est[$i]['nombre'];

$telefono=$est[$i]['telefono'];

**echo** <<<visualizacion

Identificación: <input type='text' value='**$id**'/>

<br />

Nombre Completo: <input type='text' value='**$nom**'/>

<br />

Teléfono: <input type='text' value='**$telefono**'/>

<br />

<hr />

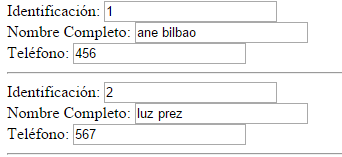
visualizacion;

}

?>

</body>

</html>



Ejercicio 11:

Sustituir el for por foreach.

Ejercicio 12:

Utilizando 2 foreach y para las leyendas(Identificación:,Nombre Completo: y Teléfono: ) crear un array

Ejercicio 13:

Modificar el ejercicio 10 de la siguiente manera:

1.-

En una carpeta llamada clases codificar: conectar.php ,estudiantes.php y vista.php. Fuera de la carpeta clases codificar index.php.

2.-

conectar.php.-Contiene la clase Conectar que consta de 2 funciones estáticas, una llamada conexión() que contiene la función mysqli\_connect() y retorna la referencia de la conexión y otra función para cerrar la conexión con la base de datos y liberar los recursos que utiliza.

3.-

estudiantes.php.-

Se codifica la clase Estudiantes.

Se declarará el atributo privado $est, siendo en el constructor donde se creará dicha variable de tipo array (En este array vamos a guardar los resultados de la consulta).

En la función get\_estudiante()se obtienen todos los estudiantes de la bd y se almacenan en el array creado en el constructor, retornando el resultado.

Cerrar la b.d.

4.-

vista.php

Clase Vista con la función visualizar

5.-

index.php.-

Codifica el controlador.