

# Tarea 4 DWES

4 de septiembre del 20XX

## 1. Repasa la POO y detalla sus conceptos y características principales.

Conceptos fundamentales:

- Clase

Una clase es un tipo de dato que nosotros definimos a nuestro antojo en función de las necesidades de nuestro programa. Puede estar compuesta por otros tipos de datos, tanto primitivos como por instancias de otras clases que hayamos creado

- Objeto

Es una instancia de clase, que tendrá todas las propiedades y métodos definidos en la clase. Cada objeto es independiente y puede tener un estado diferente a otros objetos del mismo tipo

- Método

Los métodos definen el comportamiento de nuestros objetos y los dotan de funcionalidad. Pueden recibir otros objetos como parámetro y pueden devolver o no algún valor

Los métodos pueden ser declarados como estáticos, estos métodos tienen como particularidad que pertenecen a la clase y no a los objetos, para acceder a estos métodos se hace a través de la clase y no necesitamos crear una instancia de esta. Un ejemplo típico de métodos estáticos lo podemos observar en la clase Math de java en la que todos sus métodos son estáticos

- Herencia

Permite añadir funcionalidad a una clase ya creada sin modificarla, simplemente creando una nueva clase y añadiendo la funcionalidad y propiedades que deseemos

---

## Características de POO:

- Abstracción

Cada clase representa en nuestro sistema una parte de nuestro problema de la vida real, con la abstracción podemos modelar en un lenguaje de programación nuestro problema

Un ejemplo sería la clase Pedido, Venta .... de un sistema de ventas de un comercio

- Encapsulamiento

Reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción

- Polimorfismo

Dos métodos pueden llamarse igual y tener la misma firma si son de objetos diferentes, su comportamiento será el definido en cada uno, que no tiene por que ser el mismo. Este concepto está relacionado con el de sobrecarga

El polimorfismo también está presente en otro nivel los tipos de dato mediante el uso de interfaces y herencia

- Herencia

Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo.

- Modularidad

Permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos.

---

- Principio de ocultación

Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una "interfaz" a otros objetos que detalla cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado

- Recolector de basura

Técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos.

## 2. Haz este tutorial desde aquí hasta aquí de POO; para ello, ejecuta todos sus ejemplos y explícalos.


```
<!DOCTYPE html>
<html>
<body>

<?php
class Fruit {
    // Properties
    public $name;
    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
    function set_color($color) {
        $this->color = $color;
    }
    function get_color() {
        return $this->color;
    }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>

</body>
</html>
```



Ejemplo de clase, no tiene mucho sentido poner las propiedades públicas y hacer getters y setters, uso del operador new para crear un objeto

```

<!DOCTYPE html>
<html>
<body>

<?php
class Fruit {
    // Properties
    public $name;
    public $color;

    // Methods
    function set_name($name) {
        $this->name = $name;
    }
    function get_name() {
        return $this->name;
    }
}

$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>

</body>
</html>

```

```
bool(true)
```

operator instanceof

```

<!DOCTYPE html>
<html>
<body>

<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    function get_name() {
        return $this->name;
    }
    function get_color() {
        return $this->color;
    }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>

</body>
</html>

```

```
Apple
red
```

## Uso del constructor con parámetros

```
<!DOCTYPE html>
<html>
<body>

<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name) {
        $this->name = $name;
    }
    function __destruct() {
        echo "The fruit is {$this->name}.";
    }
}

$apples = new Fruit("Apple");
?>

</body>
</html>
```

The fruit is Apple.

## Uso del destructor

```
<?php
class Fruit {
    public $name;
    protected $color;
    private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

modificadores de acceso public protected y private

```

<!DOCTYPE html>
<html>
<body>

<?php
class Fruit {
    public $name;
    public $color;
    public function __construct($name,
    $color) {
        $this->name = $name;
        $this->color = $color;
    }
    protected function intro() {
        echo "The fruit is {$this->name} and
the color is {$this->color}.";
    }
}

class Strawberry extends Fruit {
    public function message() {
        echo "Am I a fruit or a berry? <br>
";
        $this->intro();
    }
}

// Try to call all three methods from
outside class
$strawberry = new
Strawberry("Strawberry", "red"); // OK.
__construct() is public
$strawberry->message(); // OK. message()
is public
$strawberry->intro(); // ERROR. intro()
is protected

```

Am I a fruit or a berry?  
The fruit is Strawberry and the color is red.

Herencia con método protected

```

<?php
class Fruit {
    final public function intro() {
        // some code
    }
}

class Strawberry extends Fruit {
    // will result in error
    public function intro() {
        // some code
    }
}
?>

```

Uso de final para impedir la sobreescritura de un método

---

```
<?php
final class Fruit {
    // some code
}

// will result in error
class Strawberry extends Fruit {
    // some code
}
?>
```

Uso de final para impedir la herencia

```
<?php
class Goodbye {
    const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Constantes



---

```
// Parent class
abstract class Car {
    public $name;
    public function __construct($name) {
        $this->name = $name;
    }
    abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
    public function intro() : string {
        return "Choose German quality! I'm an $this->name!";
    }
}

class Volvo extends Car {
    public function intro() : string {
        return "Proud to be Swedish! I'm a $this->name!";
    }
}

// Create objects from the child classes
$audi = new Audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new Volvo("Volvo");
echo $volvo->intro();
echo "<br>";
```

Clases y métodos abstractos

---

```
<?php
// Interface definition
interface Animal {
    public function makeSound();
}
// Class definitions
class Cat implements Animal {
    public function makeSound() {
        echo " Meow ";
    }
}
class Dog implements Animal {
    public function makeSound() {
        echo " Bark ";
    }
}
// Create a list of animals
$cat = new Cat();
$dog = new Dog();
$animals = array($cat, $dog);

// Tell the animals to make a sound
foreach($animals as $animal) {
    $animal->makeSound();
}
?>
```

Interfaz e implementación

---

```

trait message1 {
    public function msg1() {
        echo "OOP is fun! ";
    }
}

trait message2 {
    public function msg2() {
        echo "OOP reduces code duplication!";
    }
}

class Welcome {
    use message1;
}

class Welcome2 {
    use message1, message2;
}

$obj = new Welcome();
$obj->msg1();
echo "<br>";

$obj2 = new Welcome2();
$obj2->msg1();
$obj2->msg2();
?>

```

Trait y uso en varias clases

```

<?php
class greeting {
    public static function welcome() {
        echo "Hello World!";
    }
}

class SomeOtherClass {
    public function message() {
        greeting::welcome();
    }
}
?>

```

Métodos estáticos y uso

---

```
<?php
class pi {
    public static $value=3.14159;
}

class x extends pi {
    public function xStatic() {
        return parent::$value;
    }
}

// Get value of static property directly via child class
echo x::$value;

// or get value of static property via xStatic() method
$x = new x();
echo $x->xStatic();
?>
```

Propiedades estáticas y uso de parent

```
<?php
namespace Html;
class Table {
    public $title = "";
    public $numRows = 0;
    public function message() {
        echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
    }
}

$table = new Table();
$table->title = "My table";
$table->numRows = 5;
?>

<!DOCTYPE html>
<html>
<body>

<?php
$table->message();
?>

</body>
</html>
```

Creación de una clase en un espacio de nombres

---

```
<?php
include "Html.php";

$table = new Html\Table();
$table->title = "My table";
$table->numRows = 5;

$row = new Html\Row();
$row->numCells = 3;
?>

<html>
<body>

<?php $table->message(); ?>
<?php $row->message(); ?>
```

Acceso a clases de otro espacio de nombres

```
<?php
namespace Html;
include "Html.php";

$table = new Table();
$table->title = "My table";
$table->numRows = 5;

$row = new Row();
$row->numCells = 3;
?>

<html>
<body>

<?php $table->message(); ?>
<?php $row->message(); ?>
```

Acceso a clases de otro espacio de nombres sin el prefijo

```

<?php
// Create an Iterator
class MyIterator implements Iterator {
    private $items = [];
    private $pointer = 0;

    public function __construct($items) {
        // array_values() makes sure that the keys are numbers
        $this->items = array_values($items);
    }

    public function current() {
        return $this->items[$this->pointer];
    }

    public function key() {
        return $this->pointer;
    }

    public function next() {
        $this->pointer++;
    }

    public function rewind() {
        $this->pointer = 0;
    }

    public function valid() {
        // count() indicates how many items are in the list
        return $this->pointer < count($this->items);
    }
}

// A function that uses iterables
function printIterable(iterable $myIterable) {
    foreach($myIterable as $item) {
        echo $item;
    }
}

// Use the iterator as an iterable
$iterator = new MyIterator(["a", "b", "c"]);
printIterable($iterator);
?>

```

Clase iterable con foreach mediante la implementación de la interfaz Iterator

### 3. Resuelve estos diez ejercicios sobre clases usando POO en PHP.

```

<?php

/*

    1. Crea una clase llamada Punto con dos propiedades/atributos

        denominados x e y, con constructor y con cuatro métodos (getter y
        setter),

```

---

```
    uno para obtener x, otro para obtener y, otro para modificar x y otro
método
```

```
    para modificar y. Crea 3 instancias/objetos de la clase Punto y
ejecuta
```

```
    en ellos los cuatro métodos creados.
```

```
*/
```

```
class Punto {

    private $x;

    private $y;

    public function __construct($x,$y){

        $this->x=$x;

        $this->y=$y;

    }

    public function getX(){

        return $this->x;

    }

    public function getY(){

        return $this->y;

    }

    public function setX($x){

        $this->x=$x;

    }

    public function setY($y){

        $this->y=$y;
```

```
}

public function getPosicion() {

    return array("x"=>$this->x,"y"=>$this->y);

}

public function desplazarX($longitud) {

    $this->x+=$longitud;

}

public function desplazarY($longitud) {

    $this->y+=$longitud;

}

public function setPosicion($x,$y) {

    $this->x=$x;

    $this->y=$y;

}

public function toString() {

    return "x= " . $this->x . " y= " . $this->y;

}

}

/* comentado por las importaciones de esta clase

$punto1=new Punto(0,0);

$punto2=new Punto(1,1);

$punto3=new Punto(2,2);

$punto1->setX(2);

echo $punto1->toString() . "<br>";
```



```
$punto2->desplazarX(2);

echo $punto2->toString() . "<br>";

$punto3->desplazarY(-4);

echo $punto3->toString();

*/

?>
```

```
<?php

/**

 *

 * 2. Crea una clase llamada Linea con cuatro propiedades/atributos d

 * enominados x1, x2, y1 e y2, con constructor y con un método que obtenga

 * el punto medio del segmento usando dichas propiedades/atributos.

 * Crea 3 instancias/objetos de la clase Linea y ejecuta en ellos el

método creado.

 *

 */

require_once "../ejercicio1.php";

class Linea{

    private $x1;

    private $y1;

    private $x2;

    private $y2;

    public function __construct($x1,$y1,$x2,$y2){
```

---

```
        $this->x1=$x1;

        $this->y1=$y1;

        $this->x2=$x2;

        $this->y2=$y2;

    }

    public function getX1(){

        return $this->x1;

    }

    public function getY1(){

        return $this->y1;

    }

    public function setX1($x){

        $this->x1=$x;

    }

    public function setY1($y){

        $this->y1=$y;

    }

    public function getX2(){

        return $this->x2;

    }

    public function getY2(){

        return $this->y2;

    }

    public function setX2($x){
```

```
        $this->x2=$x;

    }

    public function setY2($y){

        $this->y2=$y;

    }

    public function puntoMedio(){

        $xMedio=($this->x2+$this->x1)/2;

        $yMedio=($this->y2+$this->y1)/2;

        return new Punto($xMedio,$yMedio);

    }

}

$linea1=new Linea(0,0,10,10);

$linea2=new Linea(-10,-10,0,0);

$linea3=new Linea(20,20,-20,20);

echo $linea1->puntoMedio()->toString()."<br>";

echo $linea2->puntoMedio()->toString()."<br>";

echo $linea3->puntoMedio()->toString();
```

```
<?php

/**
 * Crea una clase llamada Rectangulo con dos propiedades/atributos
 * denominados longitud y ancho, con constructor y con un método que
 * calcule el area del rectángulo usando dichas propiedades/atributos.
 * Crea 3 instancias/objetos de la clase Rectangulo y ejecuta en ellos el
método creado.
 */

class Rectangulo{

    private $longitud;

    private $ancho;

    public function __construct($longitud,$ancho)

    {

        $this->longitud=$longitud;

        $this->ancho=$ancho;

    }

    public function area(){

        return $this->longitud*$this->ancho;

    }

}

$r1=new Rectangulo(10,2);
```

```
$r2=new Rectangulo(5,2);

$r3=new Rectangulo(2,2);

echo "R1= " . $r1->area() . "m^2<br>" . "R2= " . $r2->area() . "m^2<br>"
. "R3= " . $r3->area() . "m^2<br>";

<?php

/**
 * Crea una clase llamada Circulo con una propiedad/atributo
 * denominado radio, con constructor y con dos métodos que
 * calculen el area del círculo y la circunferencia del círculo
 * usando dichas propiedades/atributos. Crea 3 instancias/objetos
 * de la clase Circulo y ejecuta en ellos los dos métodos creados.
 */

class Circulo{

    private $radio;

    public function __construct($radio)

    {

        $this->radio=$radio;

    }

    public function area(){

        return round(pow($this->radio,2)*M_PI*100)/100;

    }

}
```

```

        public function circunferencia(){

            return round($this->radio*M_PI*2*100)/100;

        }

    }

    $r1=new Circulo(10,2);

    $r2=new Circulo(5,2);

    $r3=new Circulo(2,2);

    echo "Areas<br>";

    echo "C1= " . $r1->area() . "m^2<br>" . "C2= " . $r2->area() . "m^2<br>" .
    "C3= " . $r3->area() . "m^2<br>";

    echo "Circunferencias<br>";

    echo "C1= " . $r1->circunferencia() . "m<br>" . "C2= " .
    $r2->circunferencia() . "m<br>" . "C3= " . $r3->circunferencia() .
    "m<br>";

<?php

/**

 * Crea una clase llamada Estudiante con dos propiedades/atributos

 * denominados nombre y notas (array/lista), con constructor y con

 * métodos que obtenga el nombre, modifique el nombre, obtenga las

 * notas, modifique las notas y, por último, que obtenga la media

 * de esas notas y las muestre. Crea 3 instancias/objetos de la

 * clase Estudiante y ejecuta en ellos el método creado.

 */

class Estudiante{

```

```
private $nombre;

private $notas=[];


public function __construct($nombre,$notas)

{

    $this->nombre=$nombre;

    $this->notas=$notas;

}


public function getNombre(){

    return $this->nombre;

}

public function setNombre($nombre){

    $this->nombre=$nombre;

}

public function getNotas(){

    return $this->nombre;

}

public function setNotas($notas){

    $this->notas=$notas;

}

public function addNota($nota){

    array_push($this->notas,$nota);

}
```

```
public function notaMedia(){

    $total=0;

    foreach ( $this->notas as $key => $value) {

        $total+=$value;

    }

    return round($total/count($this->notas)*100)/100;

}

}

/* comentado por las importaciones de esta clase

$r1=new Estudiante("Paco",[2,6,9]);

$r2=new Estudiante("Maria",[8,5,7]);

$r3=new Estudiante("Luis",[8,8,5]);

echo "Estudiantes<br>";

echo $r1->getNombre() . "= " . $r1->notaMedia() . "<br>" .
$r2->getNombre() . "= " . $r2->notaMedia() . "<br>" . $r3->getNombre() .
"= " . $r3->notaMedia() . "<br>";

*/

<?php

/**

 * Crea una función que reciba dos parámetros de entrada
 * de tipo clase Punto (realizado en ejercicio 01) y que
 * devuelva la distancia euclídea entre esos dos puntos.
 * Ejecuta 3 llamadas de ejemplo de la función creada.
 */
```



```
declare(strict_types=1); // strict requirement

require_once "../ejercicio1.php";

function distancia(Punto $a, Punto $b) : float {

    return sqrt( pow(($b->getX()-$a->getX()),2)+

                 pow(($b->getY()-$a->getY()),2)

                );

}

$p1=new Punto(4,2);

$p2=new Punto(6,2);

echo "P1: ".$p1->toString()."<br>";

echo "P2: ".$p2->toString()."<br>";

echo "Distancia = " . distancia($p1,$p2 ) . "m";

<?php

/**

 * Crea una clase llamada Linea2D con dos propiedades/atributos

 * denominados p1 y p2 de tipo clase Punto (realizado en ejercicio 01)

 * y con dos métodos, uno que obtenga el punto medio del segmento y

 * otro que obtenga la distancia euclídea, ambos usando dichas

 * propiedades/atributos. Crea 3 instancias/objetos de la clase Linea2D

 * y ejecuta en ellos los dos métodos creado.

 */

declare(strict_types=1);
```

```
require_once "../ejercicio1.php";

class Linea2D{

    private Punto $p1;

    private Punto $p2;


    public function __construct(Punto $p1,Punto $p2)

    {

        $this->p1=$p1;

        $this->p2=$p2;

    }


    public function puntoMedio(): Punto{

        $xMedio=($this->p1->getX()+$this->p2->getX())/2;

        $yMedio=($this->p1->getY()+$this->p2->getY())/2;

        return new Punto($xMedio,$yMedio);

    }


    function distancia(Punto $a, Punto $b) : float {

        return round(sqrt( pow(($b->getX()-$a->getX()),2)+

                            pow(($b->getY()-$a->getY()),2)

                            )*100)/100;

    }

}
```

```
}

$P1=new Punto(4,2);

$P2=new Punto(6,2);

$L1=new Linea2D($P1,$P2);

echo "P1: " . $P1->toString() . " P2: " . $P2->toString() . "<br>";

echo "Punto medio de L1: " . $L1->puntoMedio()->toString() . "<br>";

echo "Distancia de L1: " . $L1->distancia($P1,$P2) . "<br>";

$P3=new Punto(2,2);

$P4=new Punto(6,2);

$L2=new Linea2D($P3,$P4);

echo "P3: " . $P3->toString() . " P4: " . $P4->toString() . "<br>";

echo "Punto medio de L2: " . $L2->puntoMedio()->toString() . "<br>";

echo "Distancia de L2: " . $L2->distancia($P3,$P4) . "<br>";

$P5=new Punto(1,1);

$P6=new Punto(3,2);

$L3=new Linea2D($P5,$P6);

echo "P5: " . $P5->toString() . " P6: " . $P6->toString() . "<br>";

echo "Punto medio de L3: " . $L3->puntoMedio()->toString() . "<br>";

echo "Distancia de L3: " . $L3->distancia($P5,$P6) . "<br>";

<?php

/**

 * Crea una clase llamada Forma con una propiedad/atributo denominada
```

```
* centro de tipo clase Punto y un método que se llame area y que
* devuelva un número, por ejemplo 0. A continuación, crea dos clases
* llamadas Rectangulo y Circulo (realizados en ejercicios 03 y 04)
* que hereden de la clase Forma ya creada. Crea 3 instancias/objetos
* de las clases Rectangulo, Circulo, de la clase que hereda Forma y
ejecuta sus métodos.
```

```
*/
```

```
declare(strict_types=1);
```

```
require_once "../ejercicio1.php";
```

```
class Forma{

    private Punto $centro;

    public function __construct(Punto $p)

    {

        $this->centro=$p;

    }

    public function getCentro(){

        return $this->centro;

    }


    public function setCentro(Punto $centro){

        $this->centro=$centro;

    }

}
```

```
        public function area():float{

            return 0;

        }

    }

}

class Rectangulo extends Forma{

    private $longitud;

    private $ancho;

    public function __construct($longitud,$ancho)

    {

        $this->longitud=$longitud;

        $this->ancho=$ancho;

        parent::setCentro($this->centro()); //por defecto

    }

    public function area():float{

        return round($this->longitud*$this->ancho*100)/100;

    }

    private function centro():Punto{

        return new Punto($this->longitud/2,$this->ancho/2);

    }

}
```

```
class Circulo extends Forma{

    private $radio;

    public function __construct($radio)

    {

        $this->radio=$radio;

        parent::setCentro($this->centro());

    }

    public function area():float{

        return round(pow($this->radio,2)*M_PI*100)/100;

    }

    public function circunferencia(){

        return round($this->radio*M_PI*2*100)/100;

    }

    private function centro():Punto{

        return new Punto(0,0); //por defecto

    }

}

$r1=new Rectangulo(10,2);

$r2=new Rectangulo(5,2);

$r3=new Rectangulo(2,2);
```

```
echo "R1= " . $r1->area() . "m^2<br>" . "R2= " . $r2->area() . "m^2<br>" .  
"R3= " . $r3->area() . "m^2<br>";
```

```
$r1=new Circulo(10,2);
```

```
$r2=new Circulo(5,2);
```

```
$r3=new Circulo(2,2);
```

```
echo "Areas<br>";
```

```
echo "C1= " . $r1->area() . "m^2<br>" . "C2= " . $r2->area() . "m^2<br>" .  
"C3= " . $r3->area() . "m^2<br>";
```

```
echo "Circunferencias<br>";
```

```
echo "C1= " . $r1->circunferencia() . "m<br>" . "C2= " .  
$r2->circunferencia() . "m<br>" . "C3= " . $r3->circunferencia() .  
"m<br>";
```

```
<?php
```

```
/**
```

```
 * Crea una función que reciba un parámetro de entrada de tipo  
array/lista,
```

```
 * con identificador grupos, de tamaño 3 cuyos elementos sean de tipo  
array/lista
```

```
 * de clase Estudiante (realizado en ejercicio 05). La función tiene que  
devolver
```

```
 * el índice del array/lista grupos cuyo promedio de notas del grupo de  
estudiantes
```

```
 * sea el más alto. Ejecuta 1 llamada de ejemplo de la función creada.
```

```
 *
```

```
 * Por ejemplo, hay tres grupos de Bachillerato con 25 alumnos cada uno.
```

---

```
* Cada grupo será un array/lista de 25 estudiantes (25 objetos/instancias
de la
* clase Estudiante) que se añadirá al array/lista grupos inicialmente
vacío.

* Cada estudiante tiene su media final, pero lo que queremos es la media
de todo
* ese grupo de estudiantes y compararlos con los otros grupos. Lo que
buscamos
* finalmente es conocer qué grupo de bachillerato tiene los alumnos con
mejor
* promedio de nota. Si se va a usar este ejemplo en el ejercicio, no es
necesario
* tantos alumnos (5 por grupo sería más que suficiente).

*/
```

```
declare(strict_types=1);
```

```
require_once "../ejercicio5.php";
```

```
class Grupo
```

```
{
```

```
    private array $estudiantes;
```

```
    public function __construct(array $estudiantes)
```

```
    {
```

```
        $this->estudiantes = $estudiantes;
```

```
    }
```



```
public function mediaGrupo(): float
{
    $total = 0;

    foreach ($this->estudiantes as $key => $value) {
        $total += $value->notaMedia();
    }

    return round($total / count($this->estudiantes) * 100) / 100;
}

function grupoTop(array $grupos): int
{
    $keyMax = 0;

    $mediaAnterior = 0;

    foreach ($grupos as $key => $value) {
        if ($value->mediaGrupo() >= $mediaAnterior) {
            $mediaAnterior = $value->mediaGrupo();
            $keyMax = $key;
        }
    }

    return $keyMax;
}

//grupo1
```

```
$r1 = new Estudiante("Paco", [2, 6, 9]);
$r2 = new Estudiante("Maria", [8, 5, 7]);
$r3 = new Estudiante("Luis", [8, 8, 5]);
$g1 = new Grupo([$r1, $r2, $r3]);

//grupo2

$r3 = new Estudiante("Pac2", [7, 9, 9]);
$r4 = new Estudiante("Mari2", [8, 9, 6]);
$r5 = new Estudiante("Lui2", [7, 8, 9]);
$g2 = new Grupo([$r3, $r4, $r5]);

//grupo1

$r6 = new Estudiante("Pac3", [4, 6, 9]);
$r7 = new Estudiante("Mari3", [6, 5, 7]);
$r8 = new Estudiante("Lui3", [5, 2, 5]);
$g3 = new Grupo([$r6, $r7, $r8]);

echo "Indice de grupo con mayor nota= " . grupoTop([$g1, $g2, $g3]);

<?php

/**
 * Crea una clase C que herede de una clase B y que la clase B herede de A.
 *
 * La clase C heredará todos los métodos y atributos de B y B de A.
 *
 * Como mínimo, una función de A, de B y de C tienen que tener el
 *
 * mismo nombre pero que hagan cosas distintas. Crea 3 instancias/objetos
 *
 * de las clases A, B y C y ejecuta todos los métodos que hayas creado.
```

```
*/

declare(strict_types=1);

class A
{
    private int $num;

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

    public function getNum(): int
    {
        return $this->num;
    }

    public function setNum(int $num): void
    {
        $this->num = $num;
    }

    public function toString(): string
    {
        return "Soy la clase " . get_class() . " propiedad num= " .
$this->num;
    }
}
```

```
class B extends A
{
    private int $num2;

    public function __construct(int $num, int $num2)
    {
        $this->num2 = $num2;

        parent::__construct($num);
    }

    public function getNum2(): int
    {
        return $this->num2;
    }

    public function setNum2(int $num2): void
    {
        $this->num2 = $num2;
    }

    public function toString(): string
    {
        return "Soy la clase " . get_class() . " propiedad num= " .
parent::getNum() . " propiedad num2= " . $this->num2;
    }
}

class C extends B
```

```
{

    private int $num3;

    public function __construct(int $num, int $num2, int $num3)

    {

        $this->num3 = $num3;

        parent::__construct($num, $num2);

    }

    public function getNum3(): int

    {

        return $this->num3;

    }

    public function setNum3(int $num3): void

    {

        $this->num3 = $num3;

    }

    public function toString(): string

    {

        return "Soy la clase " . get_class() . " propiedad num= "

            . parent::getNum() . " propiedad num2= " . parent::getNum2() .

" propiedad num3= " . $this->num3;

    }

}

$a = new A(1);

echo $a->toString();
```

```
echo "<br>";  
  
$b = new b(6, 3);  
  
echo $b->toString();  
  
echo "<br>";  
  
$c = new C(2, 3, 4);  
  
echo $c->toString();
```

#### 4. Define programación por capas y sus capas como BLL (lógica de negocio) o DAL y detalla la three-tier.

- **Programación por capas:**

Es una arquitectura cliente-servidor en la que las funciones de presentación, procesamiento de aplicaciones y gestión de datos están separadas físicamente.

- **Lógica de negocio:**

Es la parte del programa que codifica las reglas comerciales del mundo real que determinan cómo se pueden crear, almacenar y cambiar los datos

- **Capa de acceso a datos:**

Es una capa de un programa de computadora que proporciona acceso simplificado a los datos almacenados en un almacenamiento persistente de algún tipo, como una base de datos relacional de entidades

- **Arquitectura de tres niveles:**

La arquitectura de tres niveles es un patrón de arquitectura de software cliente-servidor en el que la interfaz de usuario (presentación), la lógica del proceso funcional ("reglas comerciales"), el almacenamiento de datos informáticos y el acceso a los datos se desarrollan y mantienen como módulos independientes , con mayor frecuencia en plataformas separadas

---

## 5. Lee exhaustivamente y explica con tus propias palabras lo que es un patrón de diseño software.

Son soluciones estándar para problemas frecuentes en el desarrollo de una aplicación. Aplicándolos conseguimos que nuestro programa sea fácil de mantener y escalable

## 6. Lee exhaustivamente y explica con tus propias palabras lo que es el patrón de diseño software MVC.

Divide la parte lógica del programa en tres elementos Modelo Vista y Controlador.

**El Modelo** es el encargado de mapear la BBDD generalmente con un ORM (Hibernate, EclipseLink...) mediante las entidades:

**El Controlador** recibe las peticiones del cliente (Vista) realiza las operaciones necesarias puede o no modificar el modelo y proporciona una respuesta (Vista) al cliente

**La Vista** es la página que recibe el navegador con la que interacciona el cliente, puede contener lógica normalmente en script de js

## 7. Mira este vídeo de principio a fin, resúmelo y explica el flujo de MVC empezando por el usuario.

Explicado en el punto 6

## 8. Lee bien este y este artículo y ejecuta, modifica y explica exhaustivamente esta plantilla MVC básica.

En la plantilla MVC proporcionada todas las peticiones van dirigidas a index?... Esta página crea las instancias de la clase Controlador, Vista y Modelo, en función de la acción recibida en la petición ejecuta uno u otro método de la clase Controlador, la clase controlador hace las operaciones necesarias y ejecuta el método de la clase Vista que tiene la plantilla html solicitada

## 9. Lee exhaustivamente y explica con tus propias palabras lo que es un framework web.

Es una herramienta que proporciona una forma estándar de crear un sitio web, nos proporciona multitud de bibliotecas para facilitar y agilizar el desarrollo, seguridad, sesiones, acceso a bbdd...

---

## **10. Lee exhaustivamente y explica con tus propias palabras lo que es Laravel.**

Es un framework de desarrollo web gratuito y de código abierto, basado en el patrón MVC. Posee administrador de dependencias, acceso a bbdd relacionales, ...