

- [UD 3 PHP OO](#)
 - [Teoria](#)
 - [Ventajas:](#)
 - [Programa 1](#)
 - [Programa 2](#)
 - [Programa 3](#)
 - [Programa 4](#)
 - [Programa 5](#)
 - [Programa 6](#)
 - [Programa 7](#)
 - [Programa 8](#)
 - [Programa 9](#)
 - [Programa 10](#)
 - [Programa12](#)
 - [Programa 13](#)
 - [Programa 14](#)
 - [SPL](#)
 - [Programa 15](#)
 - [Programa 16](#)

UD 3 PHP OO

Separar el código de presentación de la lógica de negocio en el desarrollo web

Reflexión "Los programadores experimentados ven en estas herramientas un apoyo que les permite concentrarse en las partes más complejas y creativas del desarrollo, mientras la IA se encarga de los detalles más tediosos. De esta manera, Copilot no reemplaza las habilidades del desarrollador, sino que las complementa."

[Articulo](#)

Teoria

La Programación Orientada a Objetos permite organizar el código de manera modular, escalable y reutilizable.

El paradigma orientado a objetos se utilizo por primera vez con PHP 4 de forma basica hasta que despues salio PHP 5 siendo este mejor. Desde PHP 7 y 8, el lenguaje ha incorporado tipado estricto, traits, namespaces y mejoras en el rendimiento.

Ventajas:

- Mantenibilidad
 - Permite realizar cambios en una capa sin que se modifique otra
- Reutilización de Código
 - Se puede reutilizar en diferentes interfaces
- Escalabilidad
 - Facilita la expansion de la aplicación
- Mejora en la Prueba y Depuración
 - La logica de negocio puede ser probada de forma aislada
- Facilita el Trabajo en Equipo
 - Permite que los desarrolladores trabajen en paralelo
- Flexibilidad en la Presentación
 - Permite que la capa de presentación se adapte a diferentes tecnologías
- Facilidad para Implementar APIs y Servicios Web
 - Con la logica de negocio desacoplada es facil exponerla a traves de una API
- Uso de Patrones de Diseño y Buenas Prácticas
 - Facilita la implementacion de modelo MVC

Programa 1

UD3 > Program1.php > ...

```
1  <?php
2      1 reference | 0 implementations
3  class ClaseSencilla
4  {
5      // Declaración de una propiedad
6      1 reference
7      public $var = 'un valor predeterminado';
8
9
10     // Declaración de un método
11    1 reference | 0 overrides
12    public function mostrarVar(): void {
13        echo $this->var;
14    }
15
16    // Crear un objeto de la clase
17    $obj = new ClaseSencilla();
18    $obj->mostrarVar(); // Muestra 'un valor predeterminado'
19
20    ?>
```

un valor predeterminado

Como podemos ver creamos un objeto una propiedad y un metodo

Programa 2

```

    * Programaz.php > ...
<?php
2 references | 0 implementations
class Persona {
    // Propiedades de la clase
    2 references
    public $nombre;
    2 references
    public $edad;

    // Constructor de la clase
    2 references | 0 overrides
    public function __construct($nombre, $edad) {
        $this->nombre = $nombre; // Asigna el valor del parámetro $nombre a la propiedad $nombre del objeto
        $this->edad = $edad; // Asigna el valor del parámetro $edad a la propiedad $edad del objeto
    }

    // Método para mostrar información de la persona
    2 references | 0 overrides
    public function mostrarInfo(): string {
        return "Nombre: $this->nombre, Edad: $this->edad";
    }
}

// Crear una instancia de la clase Persona usando el constructor
$persona1 = new Persona(nombre: "Juan", edad: 25);
echo $persona1->mostrarInfo(); // Salida: Nombre: Juan, Edad: 25

$persona2 = new Persona(nombre: "María", edad: 30);
echo $persona2->mostrarInfo(); // Salida: Nombre: María, Edad: 30
?>

```

Nombre: Juan, Edad: 25Nombre: María, Edad: 30

```

<?php
3 references | 0 implementations
class Coche {
    4 references
    public $marca;
    2 references
    public $modelo;

    3 references | 0 overrides
    public function __construct($marca, $modelo) {
        $this->marca = $marca;
        $this->modelo = $modelo;
    }

    2 references | 0 overrides
    public function obtenerInformacion(): string {
        return "...Este es un coche de la marca $this->marca, modelo $this->modelo." . "\n";
    }
}

$coche1 = new Coche(marca: "Toyota", modelo: "Corolla");
$coche2 = new Coche(marca: "Ford", modelo: "Focus");

echo $coche1->obtenerInformacion();
echo $coche2->obtenerInformacion();

?>

```

Programa 3

```
<?php
5 references | 0 implementations
class Persona {
    5 references
    private $nombre;

    5 references | 0 overrides
    public function __construct($nombre) {
        $this->nombre = $nombre;
        echo "Objeto creado con nombre: $this->nombre<br>";
    }

    3 references | 0 overrides
    public function saludar(): void {
        echo "Hola, soy $this->nombre";
    }

    0 references | 0 overrides
    public function __destruct() {
        echo "<br>Objeto destruido.";
    }
}

//Crea al menos 3 objetos de la clase persona
// y muestra un saludo para cada uno
$persona1 = new Persona(nombre: "Miguel");
$persona1->saludar();
$persona2 = new Persona(nombre: "Ana");
$persona2->saludar();
$persona3 = new Persona(nombre: "Luis");
$persona3->saludar();
?>
```

```
Objeto creado con nombre: Miguel
Hola, soy Miguel
Objeto creado con nombre: Ana
Hola, soy Ana
Objeto creado con nombre: Luis
Hola, soy Luis
Objeto destruido.
Objeto destruido.
Objeto destruido.
```

Programa 4

```
03 > 📁 Programa4.php > ...
1   <?php
2     2 references | 0 implementations
3   ✓ class MayorMenor {
4     2 references
5       private int $mayor;
6     2 references
7       private int $menor;
8
9
10      0 references | 0 overrides
11      public function setMayor(int $may) : void {
12        $this->mayor = $may;
13      }
14
15      0 references | 0 overrides
16      public function setMenor(int $men) : void {
17        $this->menor = $men;
18      }
19
20      1 reference | 0 overrides
21      public function getMayor() : int {
22        return $this->mayor;
23      }
24
25      1 reference | 0 overrides
26      public function getMenor() : int {
27        return $this->menor;
28      }
29    }
30
31    //crea una nueva función que devuelve un nuevo objeto con los valores mayor y menor que se le pasen
32    1 reference
33    ✓ function maymen(array $numeros) : ?MayorMenor {
34      $a = max(value_array: $numeros);
35      $b = min(value: $numeros);
36
37      $result = new MayorMenor();
38      //Establece el MAYOR y el MENOS
```

```
Mayor: 388
Menor: 1
```

Programa 5

```
3 > 🐾 Programa5.php > ⚒ B > ⚡ testB()
1   <?php
2     3 references | 0 implementations
3   class A
4   {
5     3 references | 0 overrides
6     function testThis(): void
7     ...
8     {
9       if (isset($this)) {
10         echo '$this está definida (';
11         echo get_class(object: $this);
12         echo ")\n";
13     } else {
14       echo "\$this no está definida.\n";
15     }
16   }
17
18   1 reference | 0 implementations
19   class B
20   {
21     1 reference | 0 overrides
22     function testB(): void
23     ...
24     {
25       // A::testThis();
26       $a= new A();
27       $a->testThis();
28     }
29
30   //Crea clase C con método estático que llame a testThis de A
31   1 reference | 0 implementations
32   class C {
33     1 reference | 0 overrides
34     public static function callTestThis(): void {
35       $a = new A();
36       $a->testThis();
37     }
38   }
```

\$this está definida (A) \$this está definida (A) \$this está definida (A)

Programa 6

```
> 🐘 Programa6.php > ...
<?php
1 reference | 0 implementations
class Foo
{
    1 reference
    public $bar = 'property';

    1 reference | 0 overrides
    public function bar(): string {
        return 'method';
    }
}

$obj = new Foo();
echo $obj->bar, PHP_EOL, $obj->bar(), PHP_EOL;
```

Programa 7

```
<?php  
1 reference | 0 implementations  
class Cuenta {  
    2 references  
    private $saldo = 0;  
  
    1 reference | 0 overrides  
    public function depositar($cantidad): void  
        if ($cantidad > 0) {  
            $this->saldo += $cantidad;  
        }  
    }  
  
    0 references | 0 overrides  
    public function obtenerSaldo(): mixed {  
        return $this->saldo;  
    }  
}  
  
$cuenta = new Cuenta();  
$cuenta->depositar(cantidad: 100);  
echo "Saldo actual: " //Completa ???  
?>
```

Saldo actual:

Programa 8

```
1 <?php
2     1 reference | 1 implementation
3     class Animal {
4         1 reference
5         public $nombre;
6
7             0 references | 0 overrides
8             public function __construct($nombre) {
9                 $this->nombre = $nombre;
10            }
11
12             1 reference | 1 override
13             public function hacerSonido(): string {
14                 return "El animal hace un sonido";
15            }
16        }
17
18 // Clase Perro que hereda de Animal
19 1 reference | 0 implementations
20 class Perro extends Animal {
21     1 reference | 0 overrides | prototype
22     public function hacerSonido(): string {
23         return "Guau Guau";
24     }
25 }
26
27 $perro = new Perro(nombre: "Max");
28 echo $perro->hacerSonido(); // Salida: Guau Guau
29
30
31 ?>
```

Guau Guau

```
<?php
2 references | 1 implementation
< class Animal {
    1 reference
    public $nombre;

        0 references | 0 overrides
    < public function __construct($nombre) {
        $this->nombre = $nombre;
    }

        2 references | 1 override
    < public function hacerSonido(): string {
        return "El animal hace un sonido";
    }
}

// Clase Perro que hereda de Animal
1 reference | 0 implementations
< class Perro extends Animal {
    2 references | 0 overrides | prototype
    < public function hacerSonido(): string {
        return parent::hacerSonido();
    }
}

$perro = new Perro(nombre: "Max");
echo $perro->hacerSonido(); // Salida: Guau Guau

?>
```

Programa 9

```

Programas.php > ...
<?php
2 references | 2 implementations
class Figura {
    1 reference | 2 overrides
    public function calcularArea(): int {
        return 0;
    }
}

1 reference | 0 implementations
class Cuadrado extends Figura {
    3 references
    private $lado;

    1 reference | 0 overrides
    public function __construct($lado) {
        $this->lado = $lado;
    }

    1 reference | 0 overrides | prototype
    public function calcularArea(): float|int {
        return $this->lado * $this->lado;
    }
}

1 reference | 0 implementations
class Circulo extends Figura {
    2 references
    private $radio;

    1 reference | 0 overrides
    public function __construct($radio) {
        $this->radio = $radio;
    }

    1 reference | 0 overrides | prototype
    public function calcularArea(): float|int {
        return pi() * ($this->radio ** 2);
    }
}

```

Este código demuestra el **Polimorfismo** y la **Herencia** en la POO de PHP, permitiendo que diferentes figuras (**Cuadrado** y **Círculo**) hereden y **sobrescriban** el método **calcularArea()** de la clase base **Figura** para ejecutar el cálculo específico de cada forma.

Programa 10

```
> Programa10.php > ...
<?php
2 references | 2 implementations
✓ abstract class Transporte {
    3 references
    protected $velocidad;

    0 references | 2 overrides
    ✓ public function __construct($velocidad) {
        $this->velocidad = $velocidad;
    }

    // Método abstracto
    2 references | 2 overrides
    abstract public function mover();
}

4 references | 0 implementations
✓ class Coche extends Transporte {
    2 references | 0 overrides | prototype
    ✓ public function mover(): string {
        return "El coche se mueve a $this->velocidad km/h";
    }
}

1 reference | 0 implementations
✓ class Bicicleta extends Transporte {
    2 references | 0 overrides | prototype
    ✓ public function mover(): string {
        return "La bicicleta se mueve a $this->velocidad km/h";
    }
}

$coche = new Coche(marca: 120);
echo $coche->mover(); // Salida: El coche se mueve a 120 km/h

$bicicleta = new Bicicleta(velocidad: 20);
echo $bicicleta->mover(); // Salida: La bicicleta se mueve a 20 km/h
?>
```

El coche se mueve a 120 km/h
La bicicleta se mueve a 20 km/h

Programa12

```

<?php
// Clase final: no se puede heredar de ella
1 reference
final class Calculadora
{
    1 reference
    public function sumar($a, $b): mixed
    {
        return $a + $b;
    }

    1 reference
    public function restar($a, $b): float|int
    {
        return $a - $b;
    }
}

// Crear un objeto y usar la clase
$calc = new Calculadora();
echo $calc->sumar(a: 5, b: 3);    // Resultado: 8
echo "<br>";
echo $calc->restar(a: 10, b: 4); // Resultado: 6

// Esto provocaría un error fatal:
// class MiCalculadora extends Calculadora {}
?>

<?php //Métodos final
1 reference | 1 implementation
class Base

```

Programa 13

En vez de crear una clase se utiliza el trait y para usarlo se usa el use

```

<?php
// Definición del trait
2 references | 2 uses
trait Registro {
    2 references | 0 overrides
    public function registrarAccion($mensaje): void {
        $fecha = date(format: 'Y-m-d H:i:s');
        echo "[{$fecha}] {$mensaje}<br>";
    }
}

// Clase que usa el trait
1 reference | 0 implementations
class Usuario {
    use Registro; // "inyecta" los métodos del trait

    1 reference | 0 overrides
    public function login($nombre): void {
        $this->registrarAccion(mensaje: "El usuario '$nombre' ha iniciado sesión.");
    }
}

1 reference | 0 implementations
class Producto {
    use Registro; // Añadimos el trait Registro

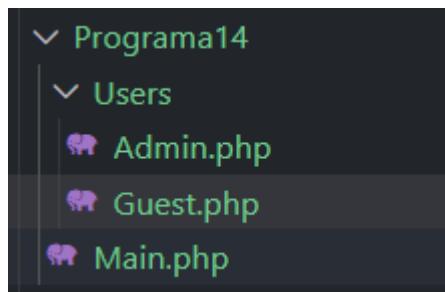
    1 reference | 0 overrides
    public function crear($nombre): void {
        $this->registrarAccion(mensaje: "Se ha creado el producto '$nombre'.");
    }
}

// Uso
$u = new Usuario();
$u->login(nombre: "Miguel"); // Ejemplo de uso del método login

$p = new Producto();
$p->crear(nombre: "Lantón"); // Ejemplo de uso del método crear

```

Programa 14



Rol del usuario: Administrador
 Rol del usuario: Invitado

SPL

SPL es la Libreria estandar de PHP que es un conjunto de clases e interfaces integradas que porporcionan funcionalidades

Programa 15

Se crea una Pila que esta doblemente enlazada y despues se le apaden valores

```
> 🐾 Programa15
<?php
// Crear una nueva pila
$stack = new SplStack();

// Añadir elementos a la pila
$stack->push(value: "Primer elemento");
$stack->push(value: "Segundo elemento");
$stack->push(value: "Tercer elemento");

// Desapilar elementos (se eliminarán en orden inverso)
echo $stack->pop(); // Output: Tercer elemento
echo "\n";
echo $stack->pop(); // Output: Segundo elemento
echo "\n";
echo $stack->pop(); // Output: Primer elemento

?>
```

```

$queue = new SplQueue();
$queue->enqueue(value: "Primer elemento");
$queue->enqueue(value: "Segundo elemento");
$queue->enqueue(value: "Tercer elemento");
echo $queue->dequeue(); // Output: Primer elemento
echo "<br>";
echo $queue->dequeue(); // Output: Segundo elemento
echo "<br>";
echo $queue->dequeue(); // Output: Tercer elemento
echo "<br>";

echo "<br><b> Clase SplFileObject: </b><br>";
$file = new SplFileObject(filename: 'monolog.txt', mode: 'r');

// iterate over its contents
while (!$file->eof()) {
    // get the current line
    $line = $file->fgets();
    echo "...LINEA: ". $line . "<br>";
    // trim it, and then check if its empty
    if (empty(trim(string: $line))) {
        // skips the current iteration
        continue;
    }
}

?>

```

Programa 16

Esto es un registro de carga automatico

- Programa16
 - clases
 - product.php
 - user.php
 - Index.php

```
3 > Programa16 > Index.php
1 <?php
2     spl_autoload_register(callback: function ($class_name): void {
3         include 'clases/' . strtolower(string: $class_name) . '.php';
4     });
5
6     $user = new user();
7     $product = new product();
8     ?>
...
```

Clase User cargada. Clase Product cargada.