Entrega 3: Programación Orientada a Objetos

> "Quienes aprovechan la IA de manera correcta podrán aprender más rápido y entregar mejores resultados".

Sinceramente estoy muy de acuerdo con esta frase ya que si entiendes de programación y si sabes pedirle a la IA algo específico, sabrás como usar la información que te genera

---

Teoría

La **Programación Orientada a Objetos (POO)** nació en Noruega en 1967 con el lenguaje Simula 67, creado por Kristen Nygaard y Ole-Johan Dahl . Este lenguaje introdujo por primera vez los conceptos de clases , subclases y corrutinas , que dieron origen al paradigma orientado a objetos moderno.

Antes de la POO, la programación era procedimental o imperativa , donde los datos y las instrucciones se trataban por separado. La POO unificó ambos aspectos al agrupar datos y comportamientos dentro de los objetos , permitiendo una forma más natural y organizada de representar el mundo real en el código.

La base de este paradigma se sostiene en **cuatro pilares fundamentales** :

Gracias a la POO en php se permite estructurar el código de forma modular, permitinedo hacerla reutilizable mediante clases y objetos. Mejorando asi la organización, escalabilidad y mantenimiento del software

Separar la lógica de negocio de la interfaz del usuario tiene varias ventajas:

- Facilita el mantenimiento, reutiliza código y la escalabilidad
- Permite realizar pruebas y depurar de manera independiente
- Mejora el trabajo entre frontend y backend
- Adapta la interfaz a diferentes plataformas
- Favorece el uso de patrones de diseño como Modelo Vista Controlador

---

Programa 1: Creaciones de clases

Si fuesen privados, no podriamos acceder a ellos desde fuera de la clase

---

Programa 2: Constructor

Al crear constructores podemos instanciar tantas clases como queramos con diferentes valores, en este caso vamos a crear varias personas y vamos a asignarle valores a través de su constructor

Dwes_Maria > UD3POO > Entrega 3 > Programa2.php > ...

```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: 26);
echo $persona1->mostrarInfo(); // Salida: Nombre: Juan, Edad: 26

$persona2 = new Persona(nombre: "María", edad: 22);
echo $persona2->mostrarInfo(); // Salida: Nombre: María, Edad: 22
?>
```

Nombre: Juan, Edad: 26Nombre: María, Edad: 22

El **destructor** (`__destruct`) se ejecuta automáticamente cuando el objeto se elimina o el script termina. Se utiliza principalmente para liberar recursos, cerrar conexiones o realizar limpieza de memoria.

Dwes_Maria > UD3POO > Entrega 3 > Programa2.php > PHP > Coche

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

    2 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.";
    }
}

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

echo $coche1-> obtenerInformacion();  // Salida: "Este es un coche de la marca Toyota, modelo Corolla."
echo $coche2-> obtenerInformacion(); // Salida: "Este es un coche de la marca Ford, modelo Focus."

?>
```

Nombre: Juan, Edad: 26Nombre: María, Edad: 22 Este es un coche de la marca Toyota, modelo Corolla.Este es un coche de la marca Ford, modelo Focus.

Hemos completado ahora el código que vemos justo arriba para que se ejecute correctamente

---

Programa 3: Destructor

Dwes_Maria > UD3POO > Entrega 3 > Programa3.php > PHP > Persona > saludar()

```php
<?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 <br>";
    }

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

$persona1 = new Persona(nombre: "Maria");
$persona2 = new Persona(nombre: "Juan");
$persona3 = new Persona(nombre: "Hugo");

echo $persona1->saludar();
echo $persona2->saludar();
echo $persona3->saludar();
//Crea al menos 3 objetos de la clase persona
// y muestra un saludo para cada uno
?>
```

Objeto creado con nombre: Maria
Objeto creado con nombre: Juan
Objeto creado con nombre: Hugo
Hola, soy Maria
Hola, soy Juan
Hola, soy Hugo

Objeto destruido.
Objeto destruido.
Objeto destruido.

He creado a 3 personas con el constructor, las saludo y se destruyen porque el script ha terminado. En programas donde es script no termina, el objeto no se destruye. Para destruirlo usar unset($persona1)

Programa 4: Setter y getter

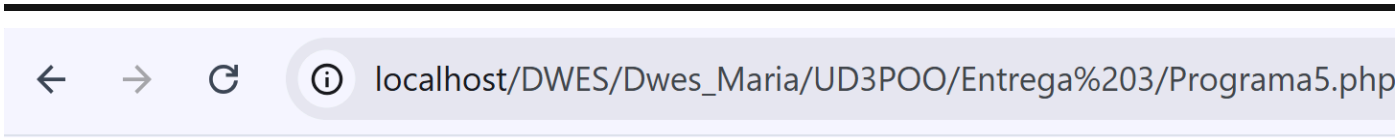
Mayor: 388
Menor: 1

Para que este programa funciones, debemos setearle a la clase tanto el menor como el mayor y unas vez ya tiene los valores, puede cogerlos con un get para así mostrarlos

---

Programa 5: THIS

```php
class A
{
    3 references | 0 overrides
    function testThis(): void
    {
        if (isset($this)) {
            echo '$this está definida (';
            echo get_class(object: $this);
            echo ")\n";
        } else {
            echo "\$this no está definida.\n";
        }
    }
}

1 reference | 0 implementations
class B
{
    1 reference | 0 overrides
    function testB(): void
    {
       // A::testThis();
        $a= new A();
         $a->testThis();
    }
}

//Crea clase C con método estático que llame a testThis de A
1 reference | 0 implementations
class C {
    1 reference | 0 overrides
    public static function callTestThis(): void {
        $a = new A();
        $a->testThis();
    }
}


$a = new A();
$a->testThis();

//A::testThis();

$b = new B();
$b->testB();

// B::bar();
C::callTestThis();
?>
```



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

En la clase C al ser una función estática podemos ver en linea 43 y 44 que no hace falta hacer una instancia de la clase para usarla

---

Programa 6: Propiedad y método

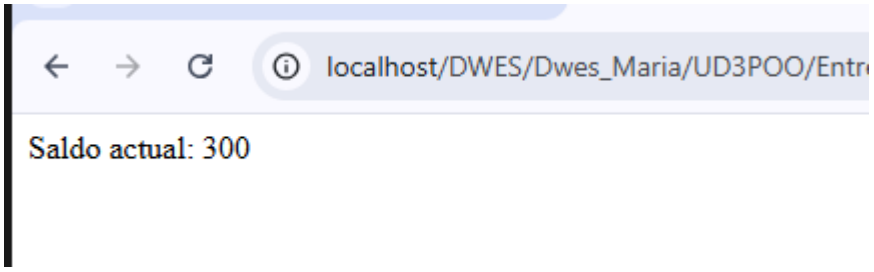
```php
<?php
1 reference | 0 implementations
class Foo
{
    1 reference
    public $bar = 'property';

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

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

property method

Programa 7: Private

Dwes_Maria > UD3POO > Entrega 3 > Programa7.php > ...

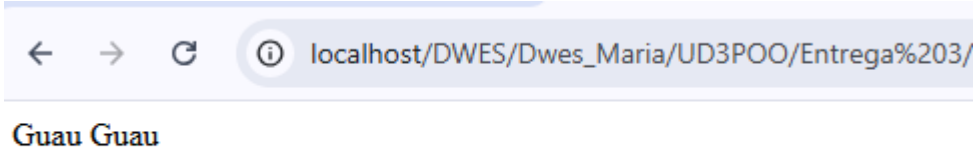
```php
<?php
1 reference | 0 implementations
class Cuenta {
    2 references
    private  $saldo = 0;

    2 references | 0 overrides
    public function depositar($cantidad): void {
        if ($cantidad > 0) {
            $this->saldo += $cantidad;
        }
    }

    1 reference | 0 overrides
    public function obtenerSaldo(): mixed {
        return $this->saldo;
    }
}

$cuenta = new Cuenta();
$cuenta->depositar(cantidad: 200);
$cuenta->depositar(cantidad: 100);
echo "Saldo actual: ". $cuenta->obtenerSaldo();
?>
```



Programa 8: Herencia

Dwes_Maria > UD3POO > Entrega 3 > Programa8.php > ...

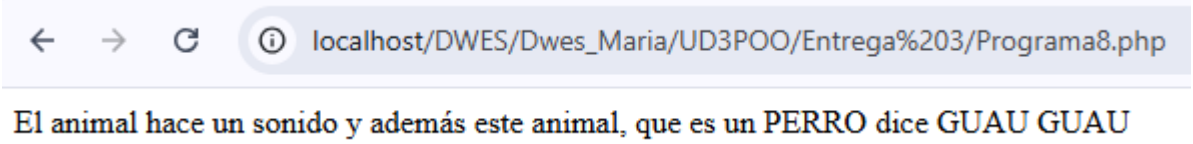
```php
<?php
1 reference | 1 implementation
class Animal {
    1 reference
    public $nombre;

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

    1 reference | 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 {
    1 reference | 0 overrides | prototype
    public function hacerSonido(): string {
    return "Guau Guau";
    }
}

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

?>
```



Lo modificamos ahora para que el lugar de sobreescribir el método del padre, lo use tal cual y encima que añada mñas información usando parent::

Dwes_Maria > UD3POO > Entrega 3 > Programa8.php > PHP > Perro

```php
<?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 {
       $padrehabla = parent::hacerSonido();
    return $padrehabla."y además este animal, que es un PERRO dice GUAU GUAU";
    }
}

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


?>
```

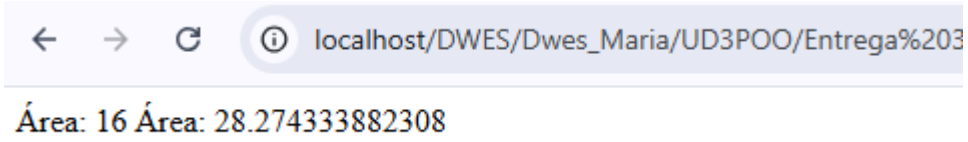


Programa 9: Polimorfismo

El polimorfismo es cuando nosotros llamamos a funciones de la misma forma pero hacen diferentes cosas, todo depende de la clase en la que se encuentren y la instancia que creemos. En el ejemplo, el array de figuras contiene objetos de distintos tipos pero todos son tratados como si fueran figuras, pero cuando se llama al método, en realidad está llamando al correspondiente a la clase real del objeto, por lo que si el objeto es un cuadrado, llamará al método de la clase del cuadrado en cambio si es un circulo lo hara con la del circulo

```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);
    }
}

// Polimorfismo en acción
$figuras = [new Cuadrado(lado: 4), new Circulo(radio: 3)];

foreach ($figuras as $figura) {
```

```
36      echo "Área: " . $figura->calcularArea() . "\n";
37  }
38  ?>
```



Área: 16 Área: 28.274333882308

Programa10: Abstract

Una clase abstracta significa que no ésta no se puede instanciar pero si que puede servir como base para una clase hija, las clases abstractas contienen métodos que las clases hijas deben usar

Dwes_Maria > UD3POO > Entrega 3 > Programa10.php > PHP > Coche > mover()

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

    0 references | 1 override
    public function __construct($velocidad) {
        $this->velocidad = $velocidad;
    }

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

3 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/hLa bicicleta se mueve a 20 km/h

Como podemos ver la clase abstracta transporte contiene un método abstracto mover(), tambien tenemos 2 clases (coche y bicicleta) que extienden de transporte, donde reescribimos la función mover(), luego instanciamos tanto un coche como una bicicleta y le seteamos una velocidad a cada uno

---

Programa 11: Interfaces

Dwes_Maria > UD3POO > Entrega 3 > Programa11.php > ...

```php
<?php
1 reference | 1 implementation
interface Operaciones {
    3 references | 2 overrides
    public function depositar($cantidad);
    1 reference | 1 override
    public function retirar($cantidad);
}

2 references | 0 implementations
class Cuenta implements Operaciones {
    6 references
    private $saldo = 0;

    3 references | 0 overrides
    public function depositar($cantidad): void {
        $this->saldo += $cantidad;
    }

    1 reference
    public f                    ad): void {
        if ($cantidad <= $this->saldo) {
            $this->saldo -= $cantidad;
        }
    }

    2 references | 0 overrides
    public function obtenerSaldo(): mixed {
        return $this->saldo;
    }
}

$cuenta = new Cuenta();
$cuenta->depositar(cantidad: 100);
$cuenta->retirar(cantidad: 30);
echo $cuenta->obtenerSaldo(); // Salida: 70
?>
```

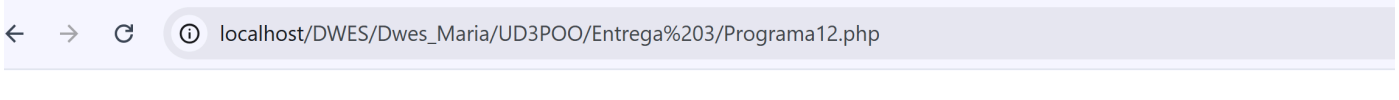
(parameter) $cantidad

<?php
$cantidad

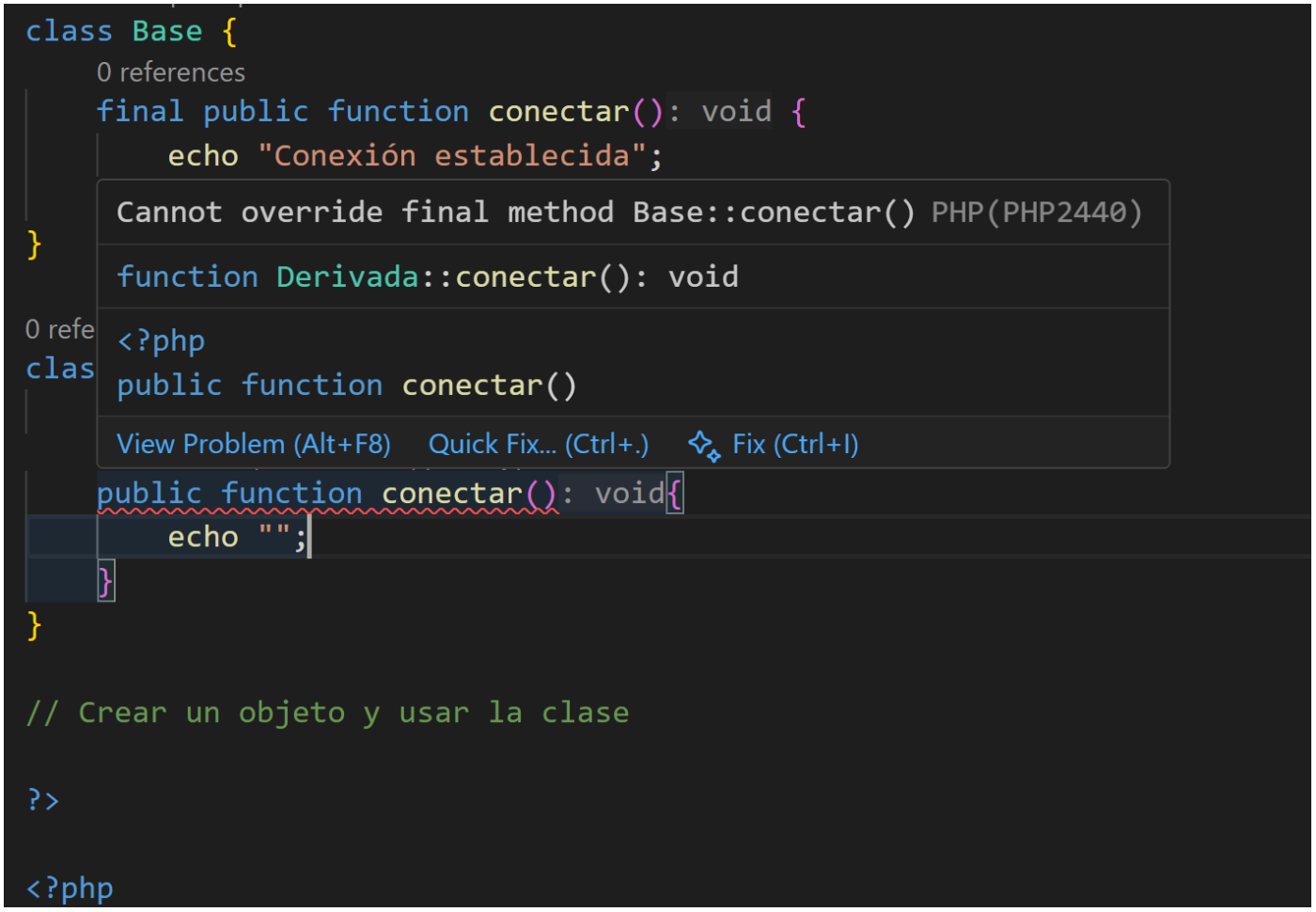
Programa 12: Final

Cuando una clase es Final, no se podrá heredar de ella.

```php
// Clase final: no se puede heredar de ella
2 references
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:
0 references | 0 implementations
class MiCalculadora extends Calculadora {}
?>
```



atal error: Class MiCalculadora cannot extend final class Calculadora in **C:\xampp\htdocs\DWES\Dwes_Maria\UD3POO\Entrega 3\Programa12.php** on line **20**



Tampoco podemos sobreescribir un método final

```php
final class Email
{
    5 references
    private string $value;

    2 references
    public function __construct(string $value)
    {
        // Validación simple
        if (!filter_var(value: $value, filter: FILTER_VALIDATE_EMAIL)) {
            throw new InvalidArgumentException(message: "Email no válido");
        }
        $this->value = strtolower(string: $value);
    }

    0 references
    public function value(): string
    {
        return $this->value;
    }

    // Comparación de valor (útil en tests)
    1 reference
    public function equals(Email $other): bool
    {
        // Comparar valores en minúsculas para ignorar mayúsculas/minúsculas
        return $this->value === $other->value;
    }

    0 references
    public function __toString(): string
    {
        return $this->value;
    }
}

// Uso
$e1 = new Email(value: "Alumno@IES.edu");
$e2 = new Email(value: "alumno@ies.edu");
var_dump(value: $e1->equals(other: $e2)); // bool(true)

// ERROR si intentas heredar de una clase final
 0 references | 0 implementations
class MiEmail extends Email {
    0 references
    private string $value2;
} // Fatal error:

?>
```

8
6 bool(true)
Fatal error: Class MiEmail cannot extend final class Email in **C:\xampp\htdocs\DWES\Dwes_Maria\UD3POO\Entrega 3\Programa12.php** on line **79**

Tampoco podemos extender de una clase que sea final

Programa 13 : Trait

Es como un objeto que dentro tiene diferentes métodos y mediante el use dentro de la clase, podemos inyectar esos bloques de métodos

Dwes_Maria > UD3POO > Entrega 3 > Programa13.php > ...

```php
<?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;


    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: "Maria");




$p = new Producto();
$p->crear(nombre: "tenedor ");
?>
```

localhost/DWES/Dwes_Maria/UD3POO/Entrega%203/Programa13.php

[2025-10-20 13:40:41] El usuario 'Maria' ha iniciado sesión.
[2025-10-20 13:40:41] Se ha creado el producto 'tenedor '.

Vamos ahora a quitar el trait para la clase producto y vamos usar el método de la clase sin inyectar el método del trait

Dwes_Maria > UD3POO > Entrega 3 > Programa13.php > PHP > Producto

```php
<?php
// Definición del trait
1 reference | 1 use
trait Registro {
    1 reference | 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 {


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


// Uso
$u = new Usuario();
$u->login(nombre: "Maria");




$p = new Producto();
$p->crear(nombre: "tenedor ");
?>
```

[2025-10-20 13:42:02] El usuario 'Maria' ha iniciado sesión.
Se ha creado el producto 'tenedor '.

Como vemos, no hemos añadido el mensaje inicial de la fecha y hora, ya que ese método forma parte del trait

Programa 14: Namespace

Son una forma de organizar el código y evitar conflictos entre nombres de clases, funciones o constantes cuando tienes multiples archivos o bibliotecas que podrían usar nombres similares. Se declaran en la primera linea

Dwes_Maria > UD3POO > Entrega 3 > Programa14 > Users > Admin.php > ...

```php
<?php
namespace Users;


2 references | 0 implementations
class Admin {
    1 reference | 0 overrides
    public function getRole(): string {
        return "Administrador";
    }
}


?>
```

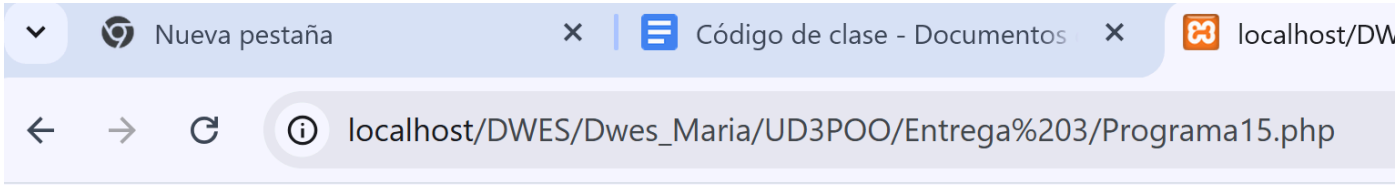
```php
<?php
require_once 'Users/Admin.php';
require_once 'Users/Guest.php';


use Users\Admin;
//use Users\Guest;


$admin = new Admin();
echo "<br> Rol del usuario: " . $admin->getRole(); // Output: Rol del usuario: Administrador


$guest = new Users\Guest();
echo "<br> Rol del usuario: " . $guest->getRole(); // Output: Rol del usuario: Invitado


?>
```

Programa 15: Stack (pila)

Al crear una pila, ya tiene métodos predefinidos, con push vamos añadiendo archivos y con pop los vamos sacando de la pila, pero desde atras hacia adelante

Dwes_Maria > UD3POO > Entrega 3 > Programa15.php

```php
<?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


?>

```



Tercer elemento Segundo elemento Primer elemento

```php
echo $stack->top();
```

Tercer elementoTercer elemento Segundo elemento Primer elemento

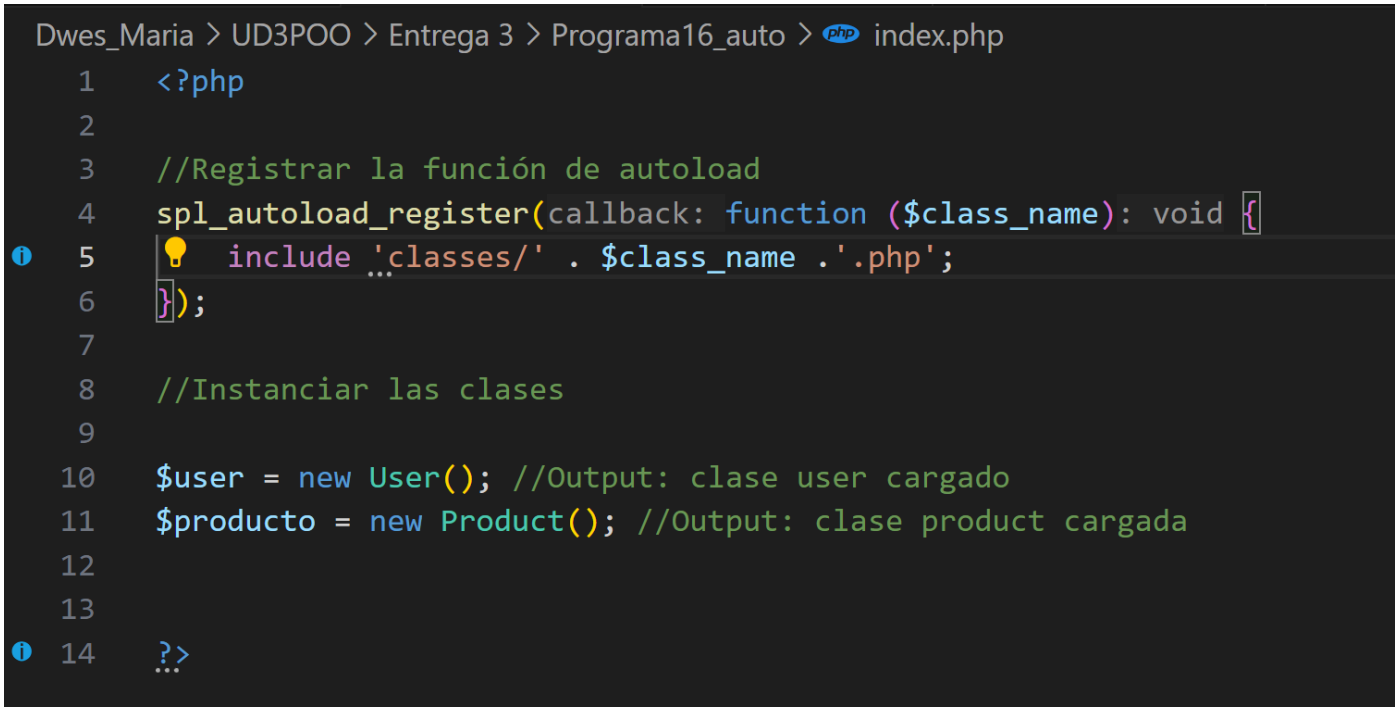
Como investigacion personal, he investigado sobre la función top(), lo que hace es mostrar el ultimo elemento de la pila, pero solo para ver su información sin eliminarlo

```php
$file = new SplFileObject('monolog.php', 'r');

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

este método es muy interesante, ya que le decimos que escoja el archivo monolog.php, que esta en modo lectura con el 'r', y continua leyendo lineas e imprimiendola

---

Programa 16: Autoload



Dwes_Maria > UD3POO > Entrega 3 > Programa16_auto > index.php

```php
<?php

//Registrar la función de autoload
spl_autoload_register(callback: function ($class_name): void {
    include 'classes/' . $class_name .'.php';
});

//Instanciar las clases

$user = new User(); //Output: clase user cargado
$producto = new Product(); //Output: clase product cargada


?>
```

Dwes_Maria > UD3POO > Entrega 3 > Programa16_auto > classes > Product.php > ...

```php
<?php
1 reference | 0 implementations
class Product {
    1 reference | 0 overrides
    public function __construct() {
        echo "Clase Product cargada.\n";
    }
}
?>

```

Dwes_Maria > UD3POO > Entrega 3 > Programa16_auto > classes > User.php > ...

```php
<?php
1 reference | 0 implementations
class User {
    1 reference | 0 overrides
    public function __construct() {
        echo "Clase User cargada.\n";
    }
}


?>
```

Clase User cargada. Clase Product cargada.