
# Feedback Modulo VI

DESARROLLO WEB EN ENTORNO SERVIDOR

**Vianca Tereza Franco Rodriguez**

# ENUNCIADO

## EJERCICIO CLASE ABSTRACTA

**1o. Vamos a crear una clase denominada Figura de la cual no se podrán crear objetos. Esta clase debe tener lo siguiente:**

- 2 variables de tipo double denominadas x e y.
- Constructor con argumentos que inicialicen x e y
- Método abstracto denominado área que retorne un double.

**2o. A continuación crearemos dos clases publicas denominadas Cuadrado y Circulo, las cuales heredarán de la clase Figura.**

**La clase Circulo tendrá lo siguiente:**

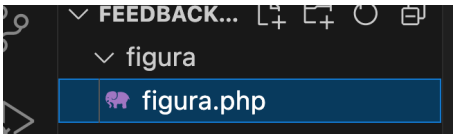
- Una variable de clase de tipo double denominada radio.
- Tendrá el constructor con argumentos.
- Desarrollará la lógica necesaria del método área de la clase padre.

**La clase Cuadrado tendrá lo siguiente:**

- Una variable de clase de tipo double denominada lado.
- Tendrá el constructor con argumentos.
- Desarrollará la lógica necesaria del método área de la clase padre.

# RESUMEN DEL PROGRAMA

**He creado una carpeta llamada Figura con el Scripts Figura.php**




Primero: defino la Clase abstracta Figura, y declaro las variables (x,y) de tipo float.

```php
// Clase abstracta Figura
abstract class Figura {
    // Variables de instancia
    // No hay double en php, pero hay float
    protected float $x;
    protected float $y;
```

Segundo: Creo el Constructor con argumentos, utilizando *this* para referenciar la variable de instancia, y luego declaró un método abstracto para calcular el área. Es importante destacar que la implantación del método calcularArea debe definirse en cada clase que la herede.

```php
    // Constructor con argumentos
    public function __construct($x, $y) {
        // Se utiliza this para referenciar la variable de instancia
        $this->x = $x;
        $this->y = $y;
    }

    // Método abstracto para calcular el área
    abstract public function calcularArea(): float;
}
```

Tercero: Defino la clase Cuadrado que hereda de clase Figura, creo una propiedad estática *$lado* que representa el lado del cuadrado y que es compartida entre todas las instancias de la clase.

```php
// Clase Cuadrado que hereda de la clase Figura
class Cuadrado extends Figura {
    // Variable de clase, es compartida entre todas las instancias de la clase porque es estatica
    private static float $lado;
```

Cuarto: Agregó un constructor que toma tres argumentos ( x, y, $lado), el cual llama al constructor de la clase padre, y asigna el valor de $lado a la propiedad estática.

```php
// Constructor con argumentos
public function __construct($x, $y, $lado) {
    // Se utiliza parent::__construct para llamar al constructor de la clase padre
    parent::__construct($x, $y);
    // Se utiliza self para referenciar la variable de clase
    self::$lado = $lado;
}
```

Quinto: Implementó el método abstracto CalcularArea, que me retorna el valor del área.

```php
// Implementación del método abstracto
public function calcularArea(): float {
    // Formula para calcular el area de un cuadrado
    return self::$lado ** 2;
}
```

sexto: Creo un objeto cuadrado, pasandole argumentos para las coordenadas (x,y,$lado), luego imprime por pantalla un mensaje concatenando el resultado de llamar al método calcularArea() del objeto $cuadrado.

```php
// Ejemplos
$cuadrado = new Cuadrado(0, 0, 4);
echo "Área del cuadrado: " . $cuadrado->calcularArea() . "<br>";
```

**Para la clase Circulo:** Se realizan las misma acciones que en la clase Cuadrado, pero en este caso la propiedad estática definida es $radio. Luego, para calcular el área, utilizo la fórmula del área de un círculo: *La función pi()* × $r^2$ , donde el radio es obtenido de la propiedad estática $radio.

```php
// Clase Circulo que hereda de la clase Figura
class Circulo extends Figura {
    // Variable de clase, es compartida entre todas las instancias de la clase porque es estatica
    private static float $radio;

    // Constructor con argumentos
    public function __construct($x, $y, $radio) {
        // Se utiliza parent::__construct para llamar al constructor de la clase padre
        parent::__construct($x, $y);
        // Se utiliza self para referenciar la variable de clase
        self::$radio = $radio;
    }

    // Implementación del método abstracto
    public function calcularArea(): float {
        // Formula para calcular el area de un circulo
        return pi() * self::$radio ** 2;
    }
}
```

**Por último:** Creo un objeto círculo, pasandole argumentos para las coordenadas (x,y,$radio), luego imprime por pantalla un mensaje concatenando el resultado de llamar al método calcularArea() del objeto $círculo.

```php
$circulo = new Circulo(0, 0, 4);
echo "Área del circulo: " . $circulo->calcularArea() . "<br>";
```

**Pantallazo del Programa mostrado en pantalla**

Área del cuadrado: 16
Área del circulo: 50.265482457437

# EJERCICIOS INTERFACES

**Creamos un proyecto llamado EjercicioInterfaces crearemos una interface llamada Vehiculo que deberán implementar nuestras clases y que contendrá:**

- Dos métodos que retornarán un String llamados frenar y acelerar. Cada uno de los métodos tendrá un argumento de tipo entero llamado distancia.
- Después crearemos dos clases llamadas Coche y Moto que implementarán la interfaz creada.

**Las clases se describen a continuación:**

**Clase Coche**

Esta clase implementará la interfaz Vehículo y contará con una propiedad de clase denominada velocidad, de tipo entero, inicializada a cero.
El método **frenar** tiene que retornarnos un mensaje como este return "El coche ha frenado ya y va a "+velocidad+"km/hora";

El método **acelerar** tiene que comprobar que no superamos la velocidad máxima fijada en la interfaz y nos tiene que retornar al final el siguiente mensaje return "El coche ha acelerado y va a "+velocidad+"km/hora";

**Clase Moto**

Esta clase implementará la interfaz Vehículo y contará con una variable de clase denominada velocidad, de tipo entero, inicializada a cero.
El método frenar tiene que retornarnos un mensaje como este return "La moto ha frenado ya y va a " + velocidad + "km/hora";

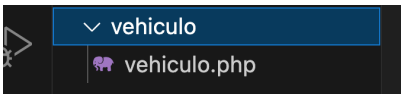
El método acelerar tiene que comprobar que no superamos la velocidad máxima fijada en la interfaz y nos tiene que retornar al final el siguiente mensaje return "La moto ha acelerado y va a " + velocidad + "km/hora";

**En el método main haremos lo siguiente:**

Creamos una matriz de tipo Vehículo con dos cajones.
En cada cajón metemos un objeto (Coche y Moto)
Recorremos la matriz de tipo Vehiculo mostrando por consola los datos de los métodos frenar y acelerar que hemos introducido a mano

# RESUMEN DEL PROGRAMA

**He creado una carpeta llamada Vehículo con el Scripts Vehículo.php**




Primero: Defino la interfaz Vehículo, luego agrego los Métodos frenar y acelerar.

```php
// Declaracion de la interfaz
interface Vehiculo {
    // Metodos con un parametro entero, devuelven un string
    public function frenar(int $distancia): string;
    public function acelerar(int $distancia): string;
}
```

Segundo: Declaro la Clase Coche que implementa a la interfaz Vehículo, agrego dos propiedades a la clase una llamada $velocidad para representar la velocidad actual del coche y otra propiedad llamada $velocidadMaxima para representar la velocidad máxima permitida, ambas inicializadas.

```php
// Clase Coche que implementa la interfaz Vehiculo
class Coche implements Vehiculo {
    // Propiedades de clase
    // Velocidad, al ser estática, es compartida entre todas las instancias de la clase
    private static int $velocidad = 0;
    private int $velocidadMaxima = 120;
```

Tercero: Implemente los métodos acelerar y frenar de la interfaz Vehículo.

Para el método acelerar, utilizo un condicional que verifica si la velocidad actual + la distancia supera la velocidad máxima, si la supera se establece la velocidad actual igual a la velocidad máxima. De lo contrario, se agrega la distancia a la velocidad actual. El método me retorna un mensaje concatenando la velocidad actual.

```php
// Método para acelerar
public function acelerar(int $distancia): string {
    // Considero que la distancia es la cantidad de velocidad que va acelerar el coche
    // Verifica que no supere la velocidad máxima
    if (self::$velocidad + $distancia > $this->velocidadMaxima) {
        self::$velocidad = $this->velocidadMaxima;
    } else {
        self::$velocidad += $distancia;
    }
    return "El coche ha acelerado y va a " . self::$velocidad . " km/hora";
}
```

El método frenar reduce la velocidad del coche, en una cantidad que se determina mediante el parámetro *$distancia*. Después de realizar esta acción, la función devuelve una cadena que indica que el coche ha frenado y muestra la nueva velocidad del coche en kilómetros por hora.

```php
// Método para frenar
public function frenar(int $distancia): string {
  // Considero que la distancia es la cantidad de velocidad que va a frenar
  self::$velocidad -= $distancia;
  return "El coche ha frenado ya y va a " . self::$velocidad . " km/hora";
}
```

Por último, creo una nueva instancia de la clase Coche utilizando *$coche = new Coche()*, y hago un *echo* del objeto coche llamando a los métodos acelerar y frenar, para aumentar y disminuir la velocidad del objeto, mostrando el resultado por pantalla.

```php
// Ejemplo de uso
$coche = new Coche();
echo $coche->acelerar(100) . "<br>";
echo $coche->frenar(40) . "<br>";
```

**Para la clase MOTO , se realizan las mismas acciones que en la clase Coche, incluyo pantallazo del código.**

```php
// Clase Moto que implementa la interfaz Vehiculo
class Moto implements Vehiculo {
    // Propiedad de clase
    private static int $velocidad = 0;
    private int $velocidadMaxima = 120;
```

```php
// Método para acelerar
public function acelerar(int $distancia): string {
    // Considero que la distancia es la cantidad de velocidad que va acelerar la moto
    // Verifica que no supere la velocidad máxima
    if (self::$velocidad + $distancia > $this->velocidadMaxima) {
        self::$velocidad = $this->velocidadMaxima;
    } else {
        self::$velocidad += $distancia;
    }
    return "La moto ha acelerado y va a " . self::$velocidad . " km/hora";
}

// Método para frenar
public function frenar(int $distancia): string {
    // Considero que la distancia es la cantidad de velocidad que va a frenar
    self::$velocidad -= $distancia;
    return "La moto ha frenado ya y va a " . self::$velocidad . " km/hora";
}
```

```php
$moto = new Moto();
echo $moto->acelerar(70) . "<br>";
echo $moto->frenar(50) . "<br>";
```

**Pantallazo del Programa mostrado por pantalla**

El coche ha acelerado y va a 100 km/hora
El coche ha frenado ya y va a 60 km/hora
La moto ha acelerado y va a 70 km/hora
La moto ha frenado ya y va a 20 km/hora

**Nota importante:** He tratado de realizar el ejercicio con todas sus pautas, pero he tenido dificultades para entender el enunciado. Algunas dificultades serían que el tipo de dato Double no existe en PHP, en su lugar he utilizado Float. El método main no esta en PHP, no entendí donde debía usarlo.

Me gustaría si es posible, que me enviara la solución del último ejercicio, me ayudaría mucho a entender el enunciado en su totalidad.

Muchas gracias. Un saludo.