


# Feedback

## 6

David Herrero Estévez



## TABLA DE CONTENIDO

Enunciado .....	3
EJERCICIO CLASE ABSTRACTA .....	3
EJERCICIOS INTERFACES .....	3
Clase abstracta .....	5
Clase cuadrado implementando la clase Figura .....	5
Clase Círculo .....	6
Interfaz .....	6
Clases implementando la interfaz .....	7
Uso de las clases creadas .....	8

## ENUNCIADO

### EJERCICIO CLASE ABSTRACTA

1°. 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.

2°. 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

### 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 implementaran la interfaz creada.

Las clases se describen a continuación:

Clase Coche

Esta clase implementará la interfaz Vehiculo 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 Vehiculo 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:

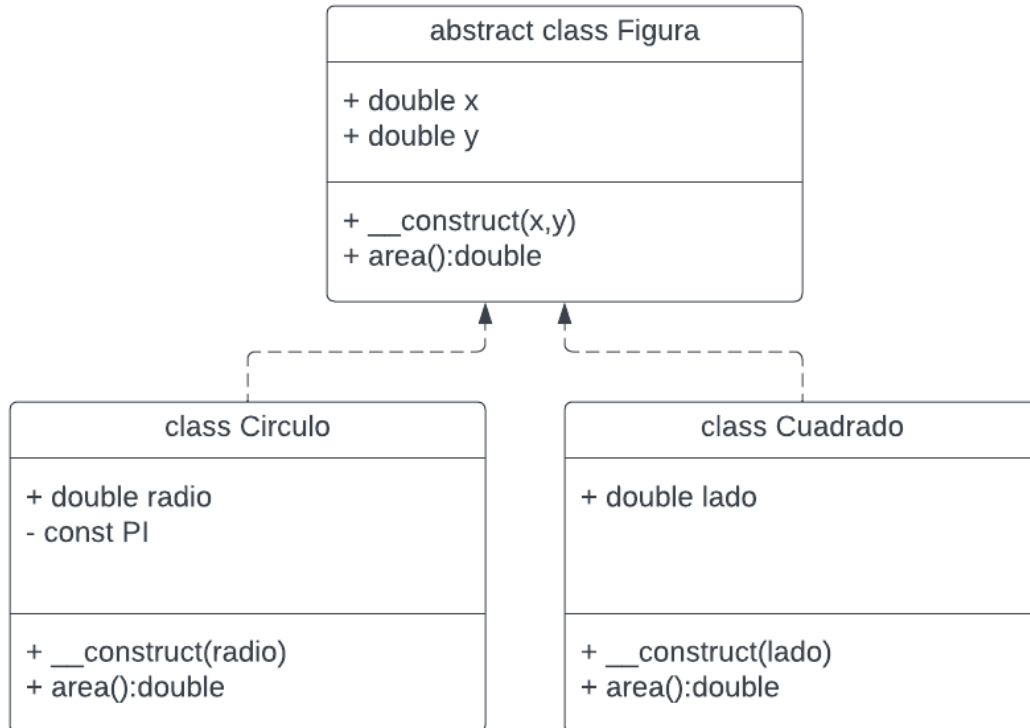
Creamos una matriz de tipo Vehiculo con dos cajones.

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

## CLASE ABSTRACTA

Este ejercicio no es apropiado para ser desarrollado en PHP, puesto que hace referencia a tipos de variables entero, double, cuando en PHP no existen estos tipos de datos.



```
abstract class Figura{
    protected $x;
    protected $y;
    public function __construct($x,$y){
        $this->x=$x;
        $this->y=$y;
    }
    abstract public function area():double;
}
```

---

## CLASE CUADRADO IMPLEMENTANDO LA CLASE FIGURA

```
class Cuadrado extends Figura{
```

```
private $lado;

public function __construct($lado){
    parent::__construct($lado, $lado);
    $this->lado=$lado;
}

public function area():doubleval{
    return $this->lado * $this->lado;
}

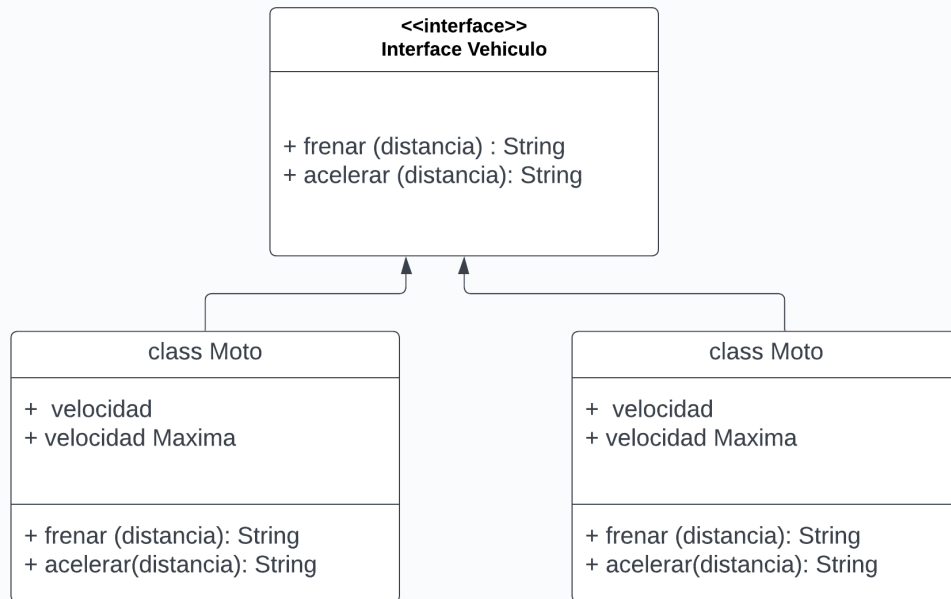
}
```

---

#### CLASE CÍRCULO

```
class Circulo extends Figura{
    private $radio;
    const PI =3.141592654;
    public function __construct($radio){
        parent::__construct($radio,0);
        $this->radio=$radio;
    }
    public function area():doubleval{
        return self::PI * pow($this->radio,2);
    }
}
```

#### INTERFAZ



```
interface Vehiculo{
    public function frenar( $distancia):String;
    public function acelerar( $distancia):String;
}
```

---

#### CLASES IMPLEMENTANDO LA INTERFAZ

```
class Coche implements Vehiculo{

    protected $velocidad=0;
    protected $velocidadMaxima=120;
    public function frenar ( $distancia):string{
        $this->velocidad-=$distancia;
        if ($this->velocidad<0)
            $this->velocidad=0;
        return "El coche ha frenado ya y va a " . $this->velocidad . " km
/ h";
    }
    public function acelerar ( $distancia):String{
        $this->velocidad+=$distancia;
    }
}
```

```

        if ($this->velocidad>$this->velocidadMaxima){
            $this->velocidad=$this->velocidadMaxima;
        }
        return "El coche ha acelerado ya y va a " . $this->velocidad . "
km / h";

    }
}

```

```

class Moto implements Vehiculo{

protected $velocidad=0;
protected $velocidadMaxima=120;
public function frenar ( $distancia):String{
    $this->velocidad-=$distancia;
    if ($this->velocidad<0)
        $this->velocidad=0;
    return "La moto ha frenado ya y va a " . $this->velocidad . " km /
h";
}
public function acelerar ( $distancia):String{
    $this->velocidad+=$distancia;
    if ($this->velocidad>$this->velocidadMaxima){
        $this->velocidad=$this->velocidadMaxima;
    }
    return "La moto ha acelerado y va a " . $this->velocidad . " km / h";
}
}

```



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

$vehiculos =["moto"=>new Moto(),
"coche" => new Coche()];
echo $vehiculos["moto"]->acelerar(20);
echo $vehiculos["coche"]->acelerar(30);

$aceleraciones = [10,20,15,10,5,10,10,10,50];
$decelaraciones = [50,50,50,50,50];

foreach ($aceleraciones as $ace){
    echo $vehiculos["moto"]->acelerar($ace) . "<br>";
    echo $vehiculos["coche"]->acelerar($ace) . "<br>";
}

foreach($decelaraciones as $dece){
    echo $vehiculos["moto"]->frenar($dece) . "<br>";
    echo $vehiculos["coche"]->frenar($dece) . "<br>";
}
?>
```