





Herencia

Las clases pueden ser extensiones de otras clases. La clase extendida puede usar todas las variables y funciones de la clase base, siempre y cuando no sean privados.

(esto es llamado 'herencia')





Extends

<?php

```
class Silla {  
    private $color;  
    private $resistenciaKG;  
  
    public function resiste($peso) {  
        return $this->resistenciaKG >= $peso;  
    }  
}
```

?>





Extends

<?php

```
class SillaDeRuedas extends Silla {  
    private $cantidadRuedas;
```

```
    public function setCantidadRuedas($cant) {  
        $this->cantidadRuedas = $cant;  
    }  
}
```

?>





Extends

¿Y que pasa con los scopes?

- ◇ public
- ◇ protected
- ◇ private





Interfaces

Las interfaces permiten crear contratos entre el implementador de la clase y el utilizador.

Se especifica qué métodos **deben** ser implementados por una clase, sin tener que definir cómo estos métodos son implementados.

Entre otras cosas, nos permiten hacer Type Hinting más complejo (y algo llamado Polimorfismo)



implements

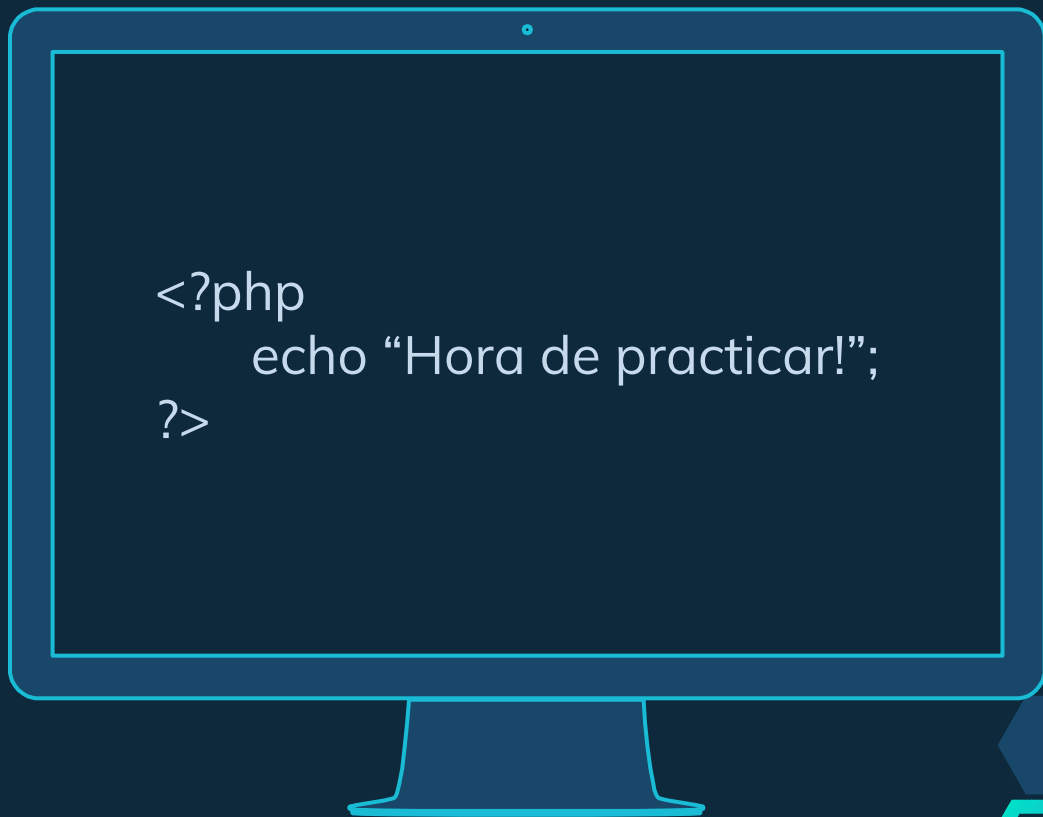
```
<?php
interface Animal {
    public function respirar();
    public function comer();
}

class Dog implements Animal{
    public function ladrar() {
        echo "Gua guau guau";
    }
    public function respirar() {
        echo "el perro está respirando...";
    }
    public function comer() { echo "el perro está comiendo...";}
}
?>
```



¡A practicar!

Ejercicios del 1 al 3





Clases abstractas

- ◇ Las clases abstractas no se pueden instanciar.
- ◇ Si heredamos de una clase abstracta, todos sus métodos abstractos deberán ser definidos (al igual que en Interfaces).
- ◇ Si el método abstracto está definido como protegido, la implementación de la función debe ser definida como protegida o pública, pero nunca como privada.





Ejemplo

```
abstract class Fruta {  
    public function comer() { //Masticar }  
}
```

```
class Manzana extends Fruta {  
    public function comer() { //Masticar hasta llegar al Centro }  
}
```

```
class Naranja extends Fruta {  
    public function comer() { //Pelar la Naranja //Masticar }  
}
```



Ejemplo

```
$manzana = new Manzana();  
$manzana->comer(); // Tiene gusto a... Manzana!!
```

```
$fruta = new Fruta();  
$fruta->comer(); // Tiene gusto a ... MMM... fruta???
```

A decorative graphic on the left side of the slide. It features a large, solid blue hexagon in the center. Surrounding it are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There are also some abstract shapes like a network of dots and a speech bubble.

Parent

Al utilizar **parent** hacemos referencia al objeto padre de nuestra instancia. Esto permite llamar a métodos de esa clase para ejecutar su comportamiento, por más de que hayamos sobrescrito el método.

Parent

<?php

```
class Persona {  
    protected $nombre;  
    public function __construct ($nombre)  
    {  
        $this->nombre = $nombre;  
    }  
}
```

```
class Usuario extends Persona {  
    protected $email;  
    public function __construct ($nombre, $email)  
    {  
        parent::__construct($nombre);  
        $this->email = $email;  
    }  
}
```

```
$usuario = new Usuario('Roberto', 'roberto@roberto.com');
```

?>

A decorative graphic on the left side of the slide consists of a cluster of hexagons in various shades of blue and cyan. Some hexagons contain white icons: a lightbulb, a thumbs-up, a network node, a smartphone, a magnifying glass, and a gear. A large, solid cyan hexagon is positioned in the center of this cluster.

Parent

El ejemplo anterior muestra uno de los usos más comunes para el uso de **parent**. Pero también se puede utilizar en cualquier método para llamar a un método del padre.

No es necesario que el método llamante sea el que sobrescribe al padre.



instanceof

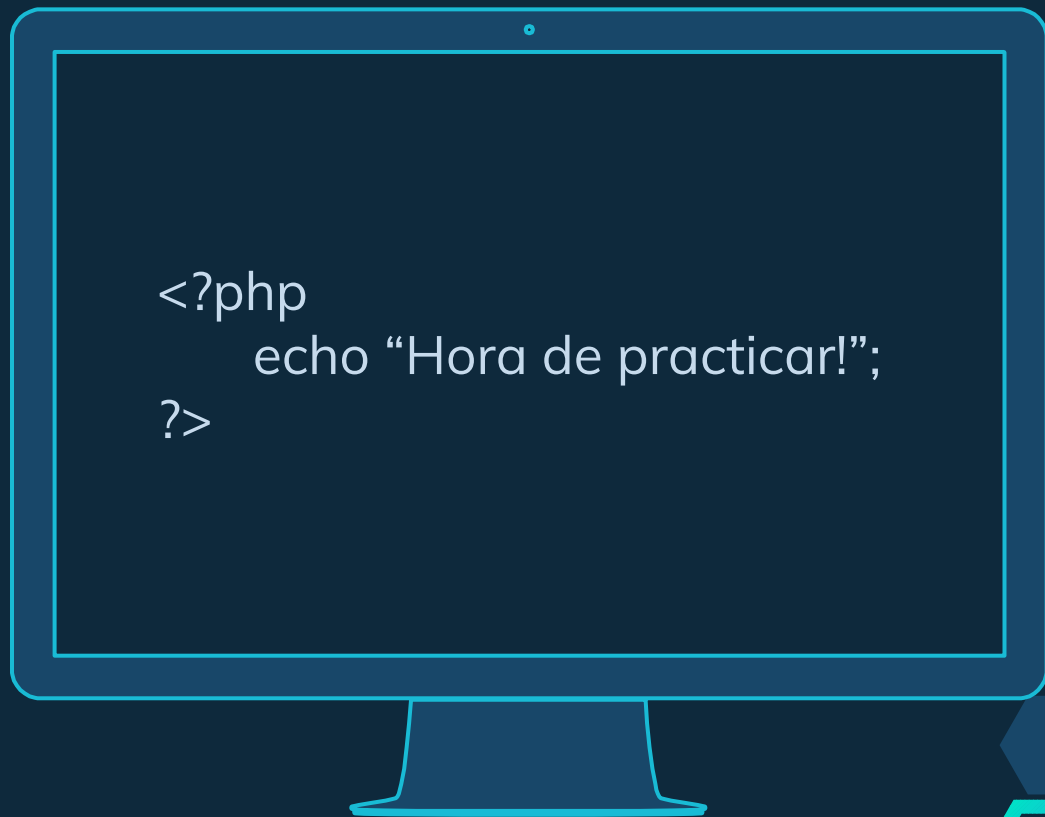
Es una función de PHP que nos permite preguntar si un objeto es de un tipo particular, es decir, cuál es su Clase.

```
<?php
if($manzana instanceof Fruta)
{
    echo "La manzana es una fruta";
}
?>
```



¡A practicar!

Ejercicios del 4 al 13





Static

- ◆ Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase.
- ◆ Una propiedad declarada como static no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).





Propiedades estáticas

```
<?php
    class Colectivo
    {
        public static $precios = [6.00, 6.25, 6.50];
    }

    var_dump(Colectivo::$precios);
?>
```

Metodos estáticos

```
<?php
class Colectivo {
    public static $precios = [6.00, 6.25, 6.50];
    public static function getPrecio($distancia) {
        if ($distancia < 5) {
            return Colectivo::$precios[0];
        }
        elseif ($distancia < 10) {
            return Colectivo::$precios[1];
        }
        return Colectivo::$precios[2];
    }
}

Colectivo::getPrecio(20);
?>
```

A decorative graphic on the left side of the slide. It features a large cyan hexagon in the center. Surrounding it are several smaller hexagons in various shades of blue and cyan. Some of these hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, a gear, and a speech bubble. There is also a network-like icon with a central node and connecting lines.

Self

Al utilizar **self** hacemos referencia a la misma clase donde esté escrito pudiendo acceder a los métodos y propiedades estáticas de la misma.



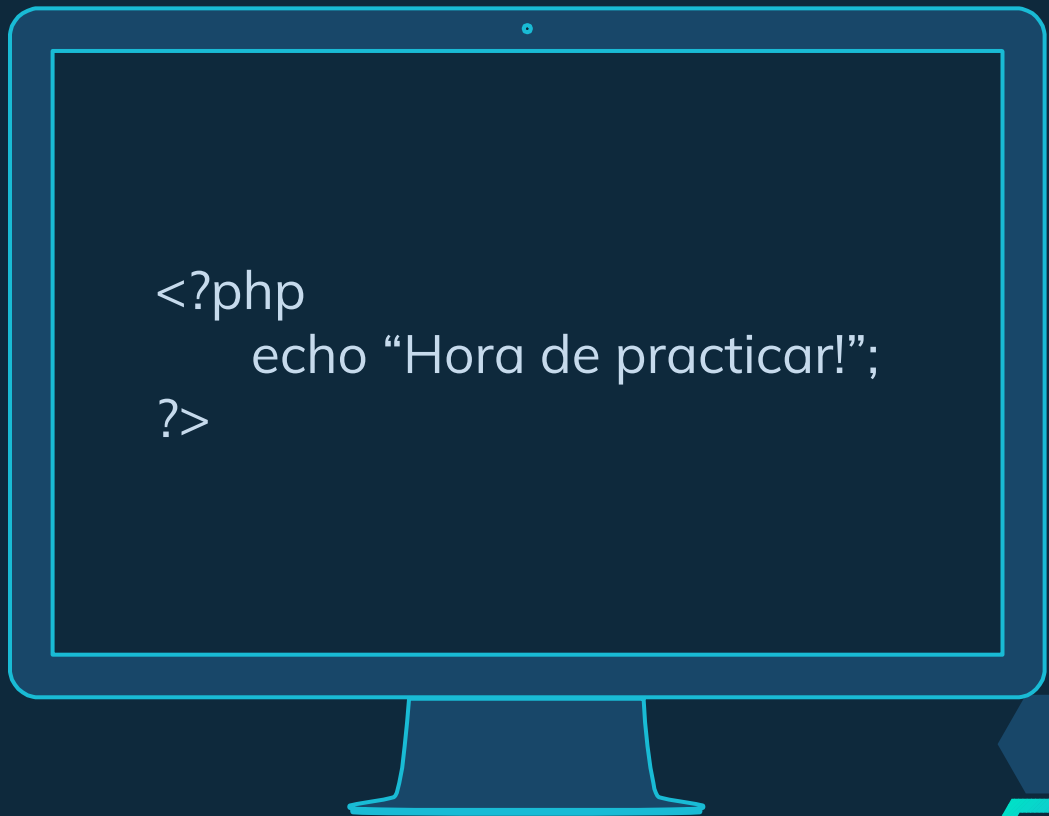
Self

```
<?php
    class Colectivo {
        public static $precios = [6.25, 6.50];
        public static function getPrecio($distancia) {
            if ($distancia < 10) {
                return self::$precios[0];
            }
            return self::$precios[1];
        }
    }

    Colectivo::getPrecio(20);
?>
```



¡A practicar!





Gracias!

¿Preguntas?

