Desarrollo Web en Entorno Servidor

Programación en PHP

José A. Lara

Programación Orientada a Objetos

__destruct()

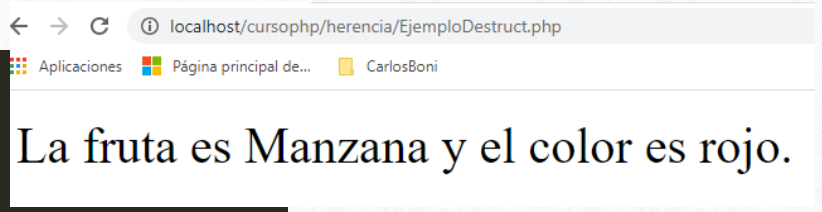
- Un destructor se va a invocar cuando el objeto se **destruya** o cuando se **pare** o salgamos del código de nuestra aplicación.

- Si creamos una función __destruct(), PHP llamará automáticamente a este código cuando se vaya a destruir el objeto.

Programación Orientada a Objetos

__destruct()

```php
<?php
    class Fruta {
      public $name;
      public $color;

      function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
      }
      function __destruct() {
        echo "La fruta es {$this->name} y el color es {$this->color}.";
      }
    }

    $apple = new Fruta("Manzana", "rojo");
?>
```



Programación Orientada a Objetos

Herencia

La **herencia** en POO = Cuando una clase deriva de otra clase.

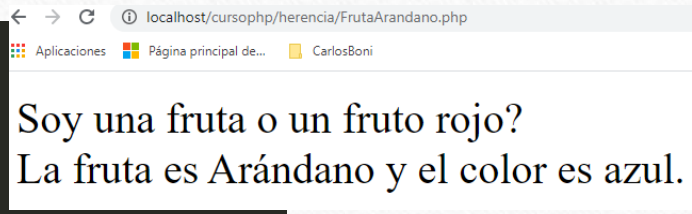
La clase hija **hereda** los métodos y atributos public y protected de la clase padre. Además puede tener sus propios atributos y métodos.

La clase que hereda se define usando la palabra reservada **extends**.

Programación Orientada a Objetos

Herencia

```php
<?php
  class Fruit {
    public $name;
    public $color;
    public function __construct($name, $color) {
      $this->name = $name;
      $this->color = $color;
    }
    protected function intro() {
      echo "La fruta es {$this->name} y el color es {$this->color}.";
    }
  }

  class Arandano extends Fruit {
    public function message() {
      echo "Soy una fruta o un fruto rojo? <br>";
      $this -> intro();
    }
  }

  $arandano = new Arandano("Arándano", "azul");
  $arandano->message();
?>
```



Programación Orientada a Objetos

Herencia

```php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "La fruta es {$this->name} y el color es {$this->color}.";
  }
}
```

```php
class Arandano extends Fruit {
  private $peso;

  public function __construct($name, $color, $gramos){
    $this->name = $name;
    $this->color = $color;
    $this->peso=$gramos;
  }

  public function intro() {
  echo "El fruto rojo es {$this->name}, el color es {$this->color}, y el peso es {$this->peso}
    gramos.";
  }
}
```

```php
$arandano = new Arandano("Arándano", "azul", 30);
$arandano->intro();
```

Programación Orientada a Objetos

Herencia

La palabra reservada **final** se usa para impedir que se herede de esa clase o para que no se pueda sobreescribir un método.

Programación Orientada a Objetos

Clases con constantes

Las constantes **no pueden ser modificadas** una vez que se declaran. Podemos usar clases con constantes para "centralizarlas".

Podemos definir las constantes con la palabra reservada **const**. Se recomienda escribir los nombres de las constantes en **mayúsculas**.

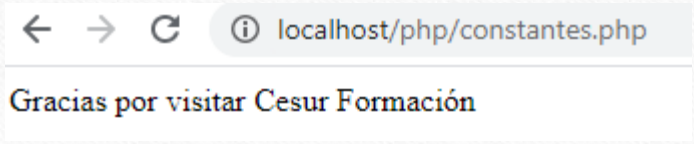
Podemos acceder a una constante desde fuera de esas clase usando el nombre de la clase seguido del operador :: seguido del nombre de la constante.

clase::constante

Programación Orientada a Objetos

Clases con constantes

```php
<?php
    class Adios {
      const MENSAJE = "Gracias por visitar Cesur Formación";
    }

    echo Adios::MENSAJE;
?>
```

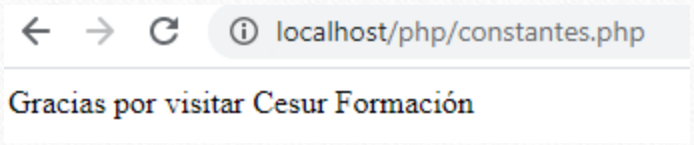


Programación Orientada a Objetos

Clases con constantes

También podemos acceder a una constante desde la misma clase pero usando la palabra reservada **self** seguido del operador **::** y del nombre de la constante.

```php
<?php
    class Adios {
      const MENSAJE = "Gracias por visitar Cesur Formación";

      public function despedida(){
        echo self::MENSAJE;
      }
    }

    $adios = new Adios();
    $adios->despedida();

?>
```



Programación Orientada a Objetos

Clases y métodos abstractos

Una **clase abstracta** es aquella de la cual no se puede instanciar un objeto.

Una **clase abstracta** es una clase que contiene al menos un método abstracto.

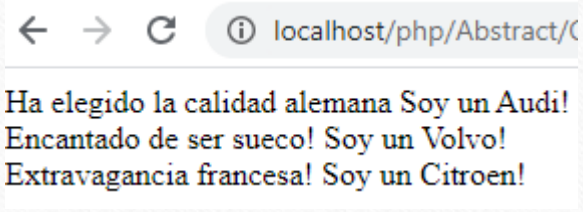
Un **método asbtracto** es un métdo que se declara en una clase abstracta pero no tiene implementado el cuerpo.

Para definer una clase abstracta utilizaremos la palabra reservada **abstract**.

Programación Orientada a Objetos

Clases y métodos abstractos

```php
// Creamos objetos de las clases hija
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
```




```php
// Clase padre
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}
```

```php
// Clases hija
class Audi extends Car {
  public function intro() : string {
    return "Ha elegido la calidad alemana Soy un $this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "Encantado de ser sueco! Soy un $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "Extravagancia francesa! Soy un $this->name!";
  }
}
```

Programación Orientada a Objetos

Interfaces

Los Interfaces nos permiten especificar qué métodos debe implementar una clase.

Cuando una o más clases utilizan el mismo interface, se dice que hay polimorfismo.

Para declarar un Interface lo haremos con la palabra reservada interface

Programación Orientada a Objetos

Interfaces vs. Clases Abstractas

Los Interface son similares a las clases abstractas. Las diferencias son:

- Los Interfaces **no pueden tener atributos**
- Todos los métodos de un interface deben ser **public**, mientras que en las clases abstractas pueden ser public o protected
- Todos los **métodos** de un interface son **abstractos**
- Las clases pueden **implementar** un interface mientras están **heredando** de otra clase a la vez

Programación Orientada a Objetos

Interfaces

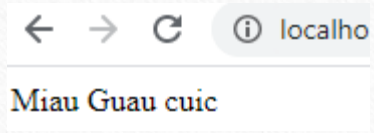
Para implementar un interface usamos la palabra reservada interface.

Una clase debe implementar todos los métodos de un interface.

```php
interface Animal {
  public function makeSound();
}
```

Programación Orientada a Objetos

Interfaces




```php
class Cat implements Animal {
  public function makeSound() {
    echo "Miau";
  }
}
class Dog implements Animal {
  public function makeSound() {
    echo " Guau ";
  }
}

class Mouse implements Animal {
  public function makeSound() {
    echo " Cuic ";
  }
}

// Creamos una lista de animales
$cat = new Cat();
$dog = new Dog();
$mouse = new Mouse();
$animals = array($cat, $dog, $mouse);

// Decimos a los animales que hagan ruido...
foreach($animals as $animal) {
  $animal->makeSound();
}
```

Programación Orientada a Objetos

Traits

PHP solo soporta herencia simple: una clase hija solo puede heredar de una sola clase padre, por lo que si una clase necesita heredar de más de una clase, cómo se hace?

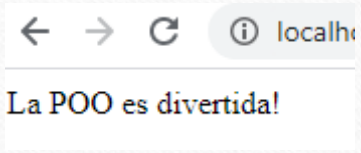
Los Traits se usan para **declarar métodos** que pueden ser **usados en casos multiples**. Pueden tener métodos y métodos abstractos que se pueden usar en muchas clases, y los métodos pueden tener cualquier modificador de acceso (public, private, or protected).

Programación Orientada a Objetos

Traits

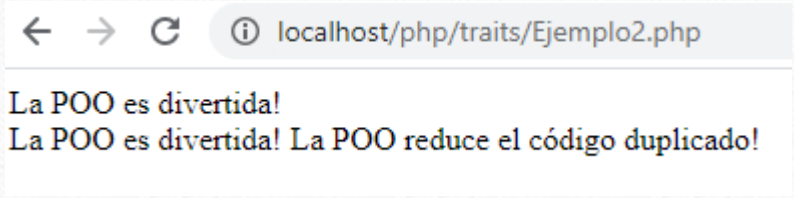
Los Traits se declaran con la palabra reservada **trait**

```php
trait message1 {
public function msg1() {
    echo "La POO es divertida! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
```



Programación Orientada a Objetos

Traits



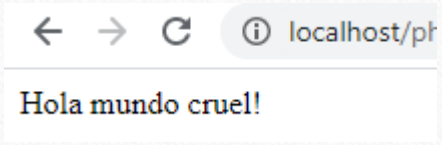

```php
trait message1 {
  public function msg1() {
    echo "La POO es divertida! ";
  }
}

trait message2 {
  public function msg2() {
    echo "La POO reduce el código duplicado!";
  }
}

class Welcome {
  use message1;
}

class Welcome2 {
  use message1, message2;
}

$obj = new Welcome();
$obj->msg1();
echo "<br>";

$obj2 = new Welcome2();
$obj2->msg1();
$obj2->msg2();
```

Programación Orientada a Objetos

Métodos estáticos

Los métodos estáticos se pueden llamar directamente sin crear una instancia de un objeto.

Se declaran con la palabra reservada **static**



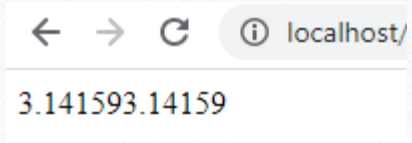

```php
class greeting {
  public static function welcome() {
    echo "Hola mundo cruel!";
  }
}

class SomeOtherClass {
  public function message() {
    greeting::welcome();
  }
}

$miclase = new SomeOtherClass();
$miclase->message();
```

Programación Orientada a Objetos

Atributos estáticos

Los atributos estáticos se pueden llamar directamente sin crear una instancia de un objeto.

Se declaran con la palabra reservada **static**




```php
class pi {
  public static $value=3.14159;
}

class x extends pi {
  public function xStatic() {
    return parent::$value;
  }
}

// Podemos obtener directamente el valor
echo x::$value;

// o bien mediante el método
$x = new x();
echo $x->xStatic();
```