Este curso forma parte de una serie de cursos cuyo objetivo principal es el de enseñar el lenguaje de programación PHP a personas con conocimientos básicos de programación, y prepararlos para llevar a cabo proyectos de gran envergadura. Para ello se enseñará PHP básico (este curso), PHP orientado a Objetos, las mejores y más novedosas practicas en PHP, Patrones de Diseño y el Zend Framework.

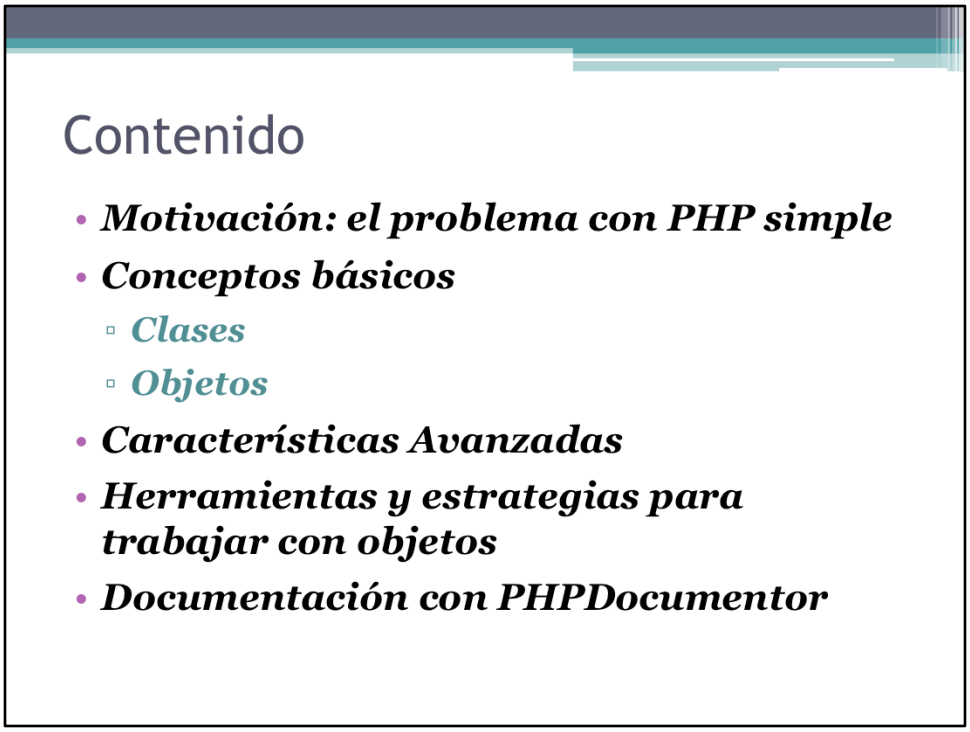
Acerca del Instructor

- Enrique Areyán
- Lic. Computación UCV
- Experiencia de aprox. 10 años (desde 2001) trabajando en aplicaciones web
- Certificado“Building PHP Applications using Zend Framework”
- Co-fundador www.estoesweb.com

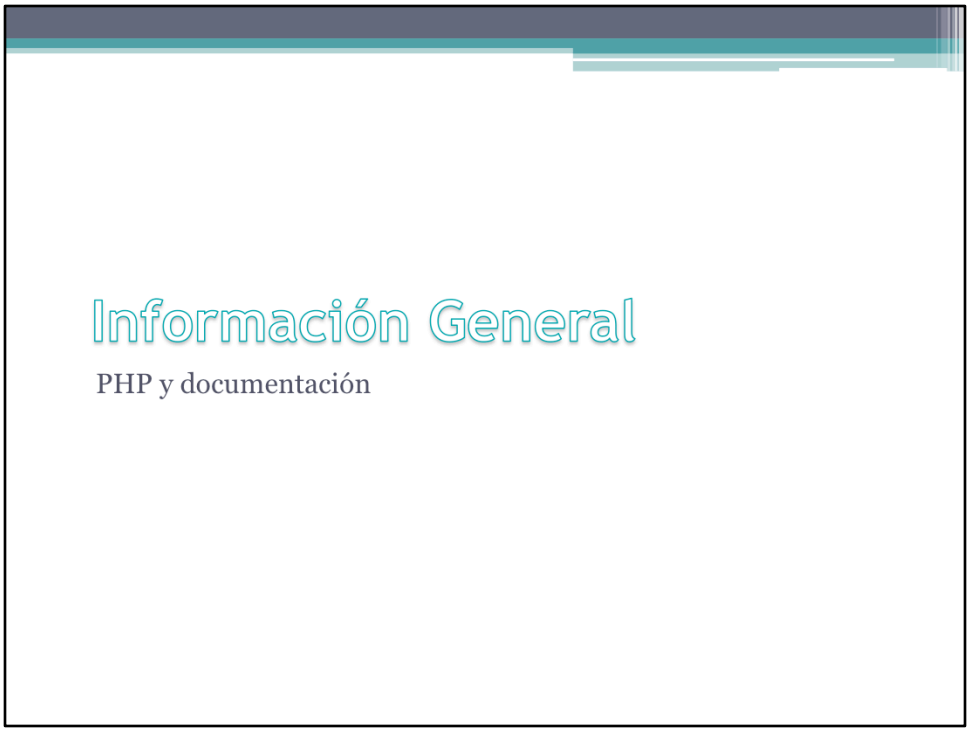
Pueden contactarme a través del correo electrónico enrique3@gmail.com

Antes de Comenzar

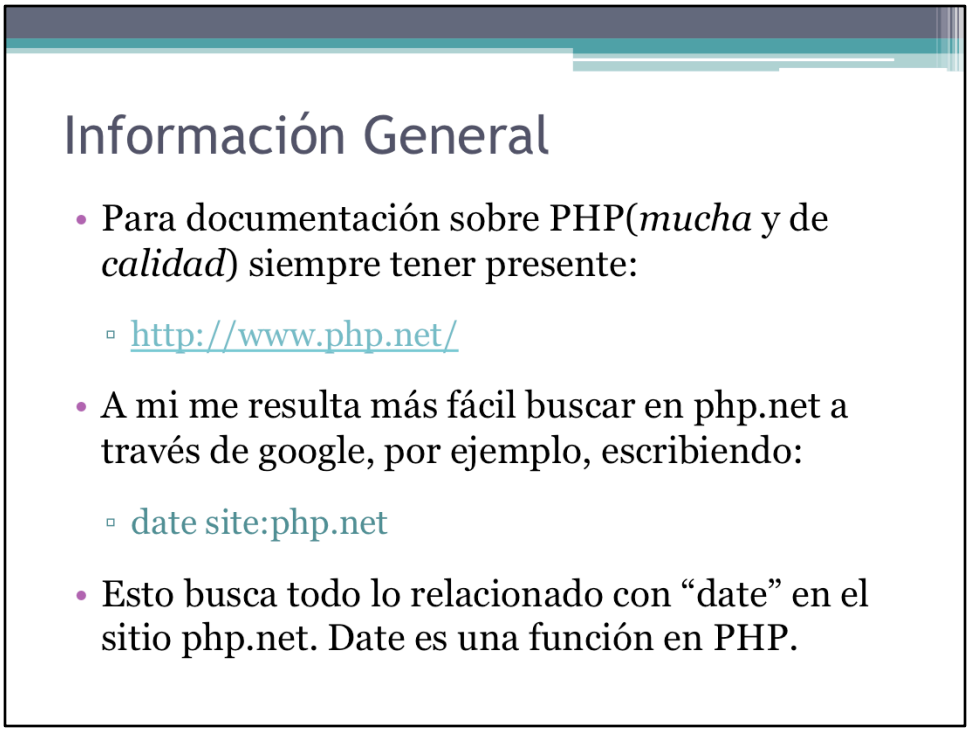
- Bajar (o usar pen drive) WAMP
- Instalarlo
- Mover las siguientes carpeta del pen drive a C:\wamp\www
  - Ejemplos
  - Ejercicios
  - Ejercicio Integral
- Esta presentación también la pueden descargar del pentdrive



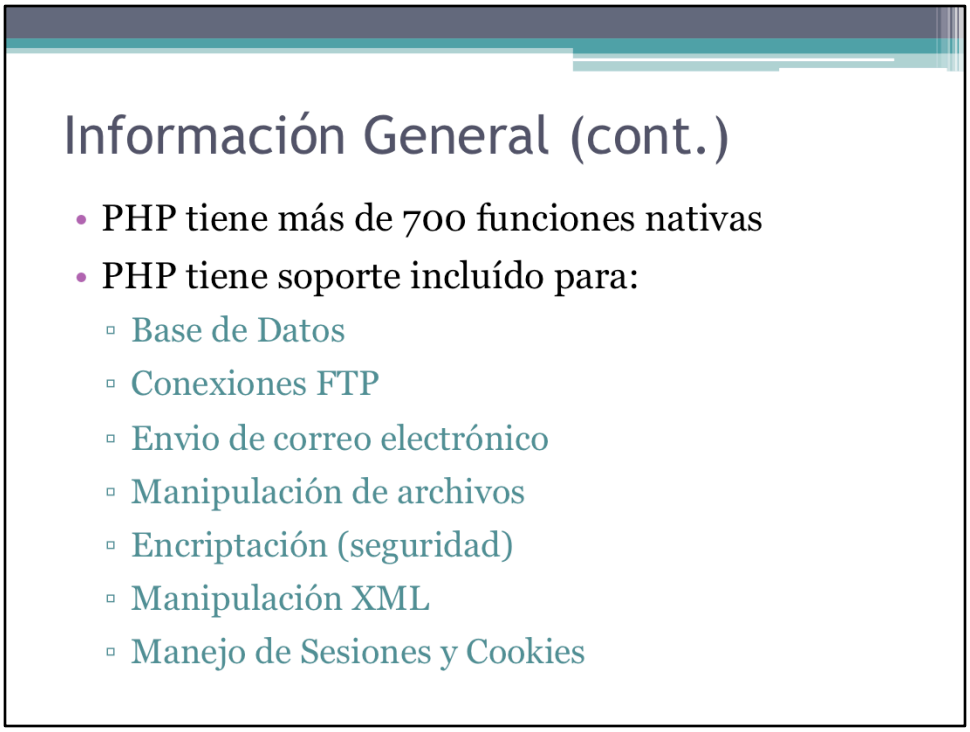
Este curso de PHP Orientado a Objetos pretender enseñar las herramientas básicas para desarrollar sistemas web bajo el paradigma de programación orientada a objetos en lenguaje PHP. Aquí se cubrirán los aspectos básicos más importantes de la programación orientada a objetos tanto desde el punto de vista teórico como práctico.
Se asume que el participante tiene conocimientos básicos de programación en PHP o que ha participado en el curso anterior a éste de PHP Básico.



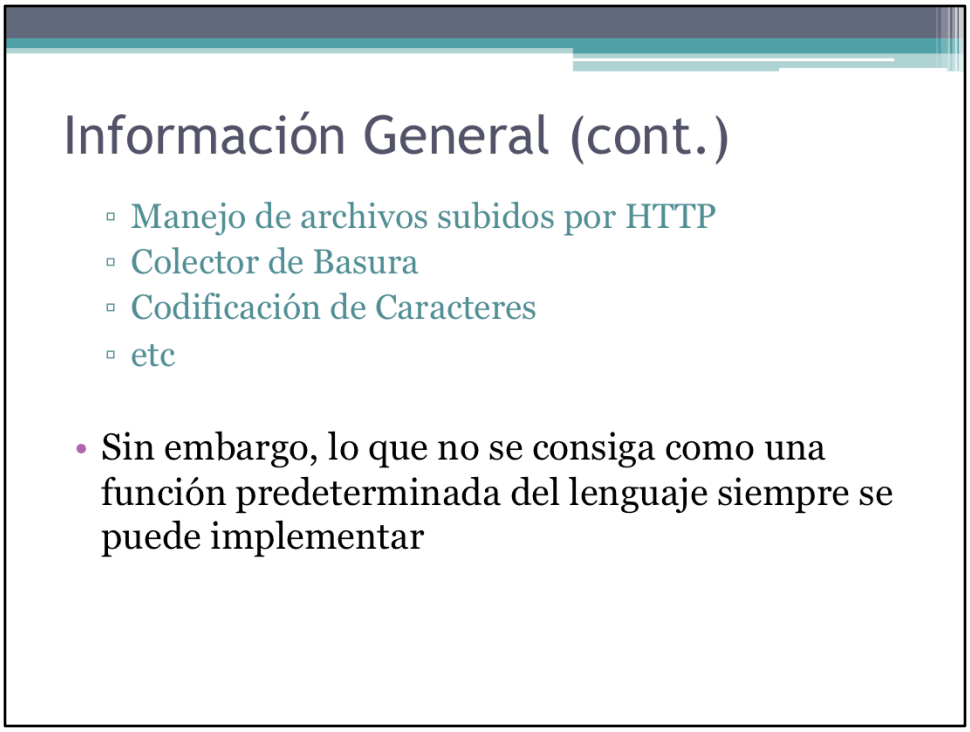
Una revisión general.



La documentación en línea sobre PHP es vasta y de calidad. Si algo no se consigue en el sitio de php, www.php.net, utilizando google se tienen altas probabilidades de solventar esta deficiencia.



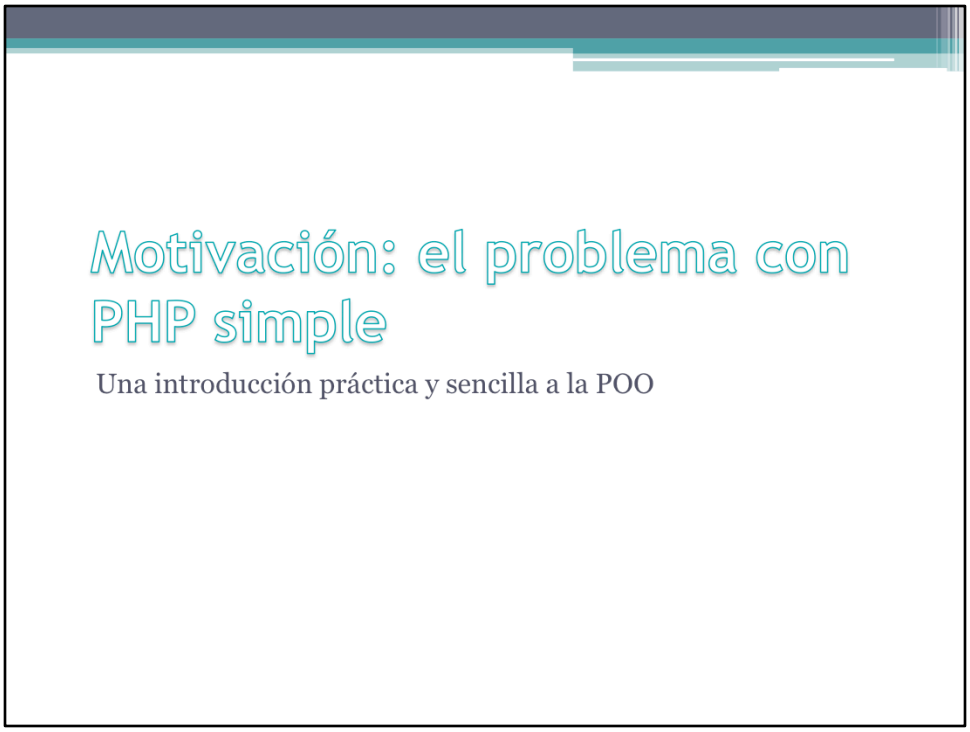
Una función nativa es aquélla que ya viene incluída como parte de las librerías internas del lenguaje y por lo tanto no debe ser implementada por el programador. Esto nos ahorra tiempo.



PHP se caracteriza por ser un lenguaje flexible. Esta característica tiene sus ventajas y desventajas, como veremos más adelante.

Glosario

- Usuario
  - Tu, el programador
- Cliente
  - La persona que visita la página web
- Interpretador o intérprete
  - El programa de PHP que se encarga de interpretar el código escrito en este lenguaje
- POO
  - Programación Orientada a Objetos



En este apartado se argumentarán algunas de las razones por las cuales la programación con PHP simple (no orientado a objetos) es insuficiente al momento de desarrollar sistemas complejos. Para tal fin, exploraremos las estrategias de programación de un ejemplo sencillo tanto en programación imperativa como en POO.

Motivación

```php
<?php
  function imprimir($automovil){
      echo "<p>Hola! soy un $automovil
  [marca] modelo $automovil[modelo]</p>";
  }

  $carro1 = array(
      'marca' => 'Toyota',
      'modelo' => 'Corolla'
  );
  $carro2 = array(
      'marca' => 'Volkswagen',
      'modelo' => 'Jetta'
  );

  imprimir($carro1);
  imprimir($carro2);
```

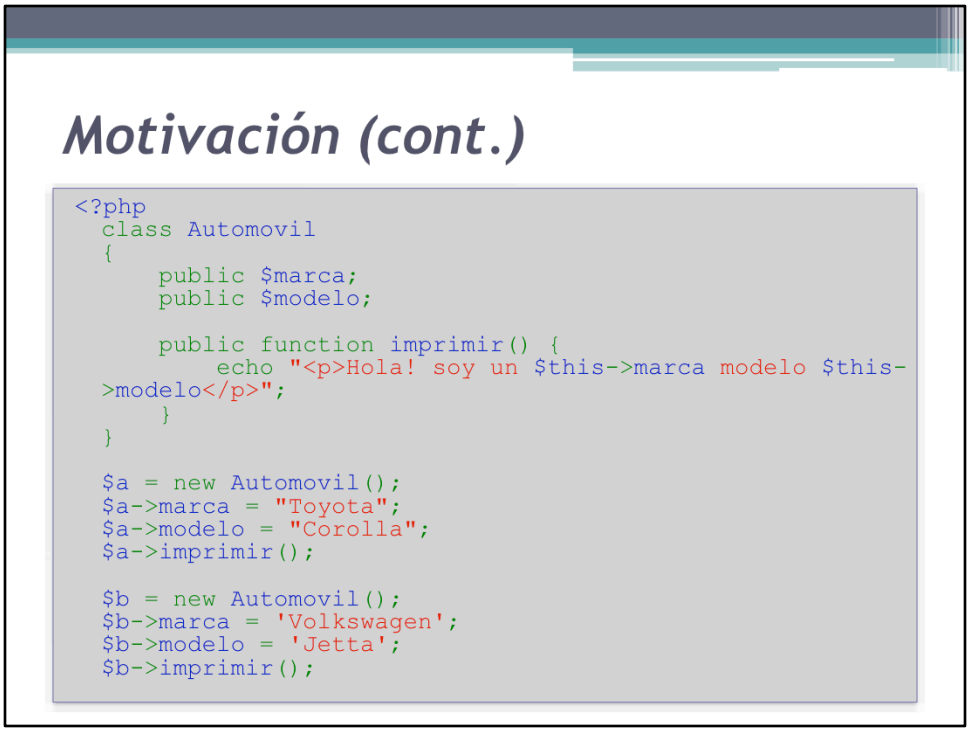
El código anterior busca encapsular, a través de una función, lógica propia a un sólo bloque de código. Sin embargo, esta estrategia falla por varias razones. Visto de otra forma, al ver éste código pregúntese lo siguiente:

•Qué pasa cuando se empiezan a tener muchas propiedades?

•O si se comienzan a tener muchas funciones?

•Qué pasa si queremos otro conjunto de rutinas y parte de los nombres de las funciones se repiten?

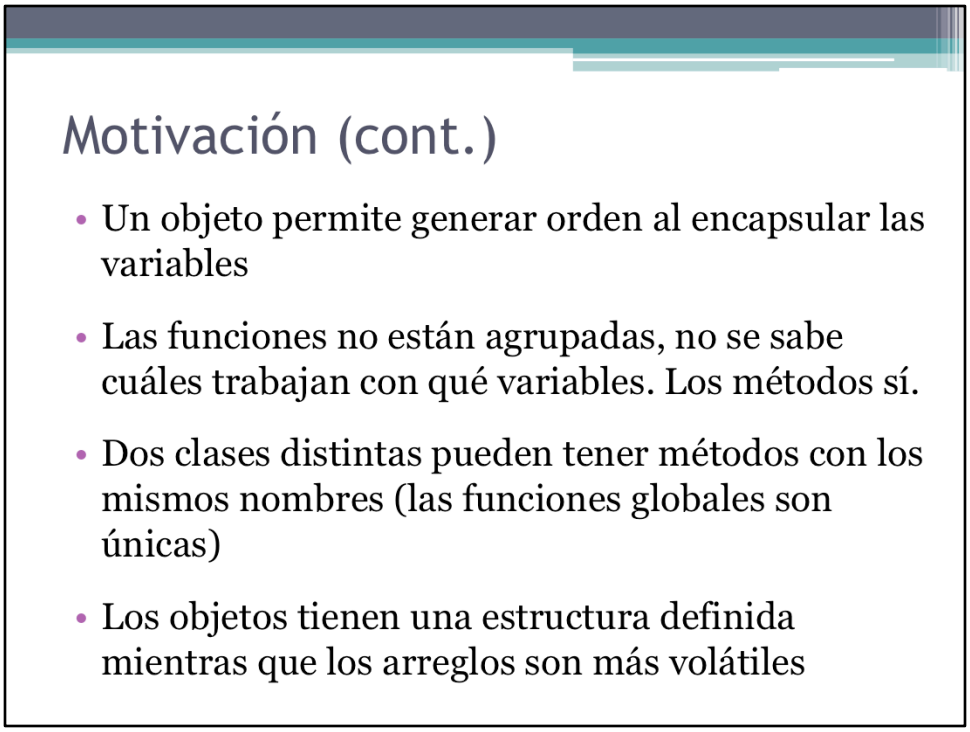
Ver archivo CodigoImperativo.php en la carpeta /Ejemplos



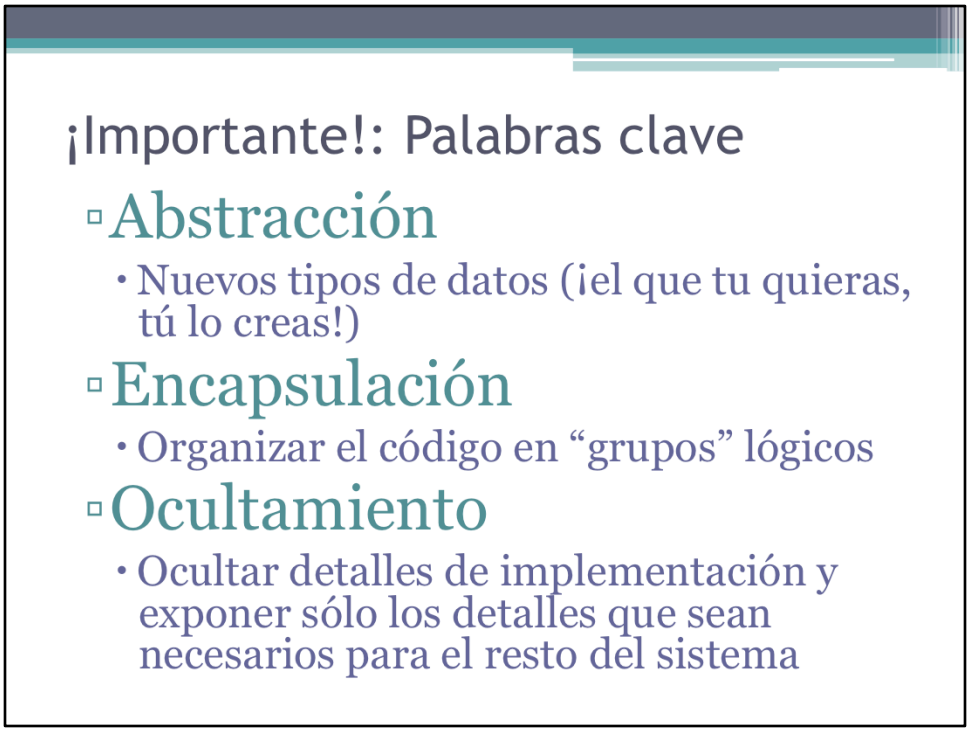
El mismo código del ejemplo anterior, pero en este caso orientado a objetos. En el código vemos una clase llamada Automóvil que posee dos propiedades y un método. Más adelante explicaremos con mayor detalle qué significan cada una de estas cosas. Por el momento es interesante observar como en un sólo bloque de código (agrupado entre las palabras **claves class Atuomovil { … }**) Tenemos agrupada la información que trabaja sobre un "Automóvil".

Más aún, al momento de instanciar, es decir, realizar copias de un automóvil, vemos como la POO nos hace la vida fácil. En el código anterior $a y $b son copias distintas, permitiendo así que $a tenga una propiedad 'modelo' que es suya propia y distinta de la propiedad 'modelo' de la variable $b.

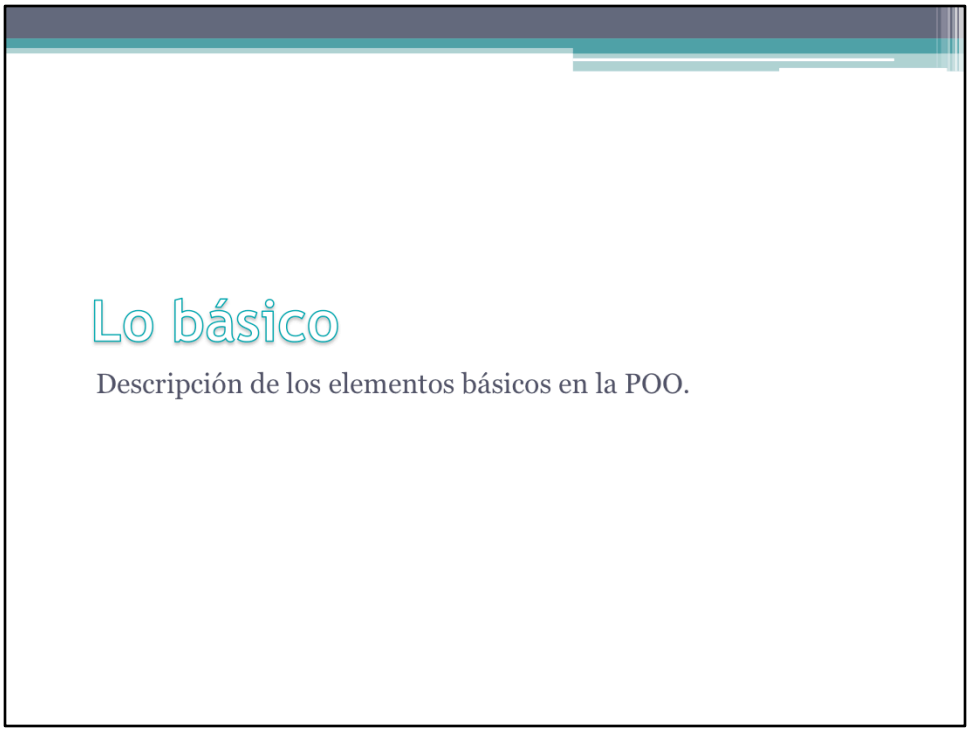
CodigoOrientadoAObjetos.php



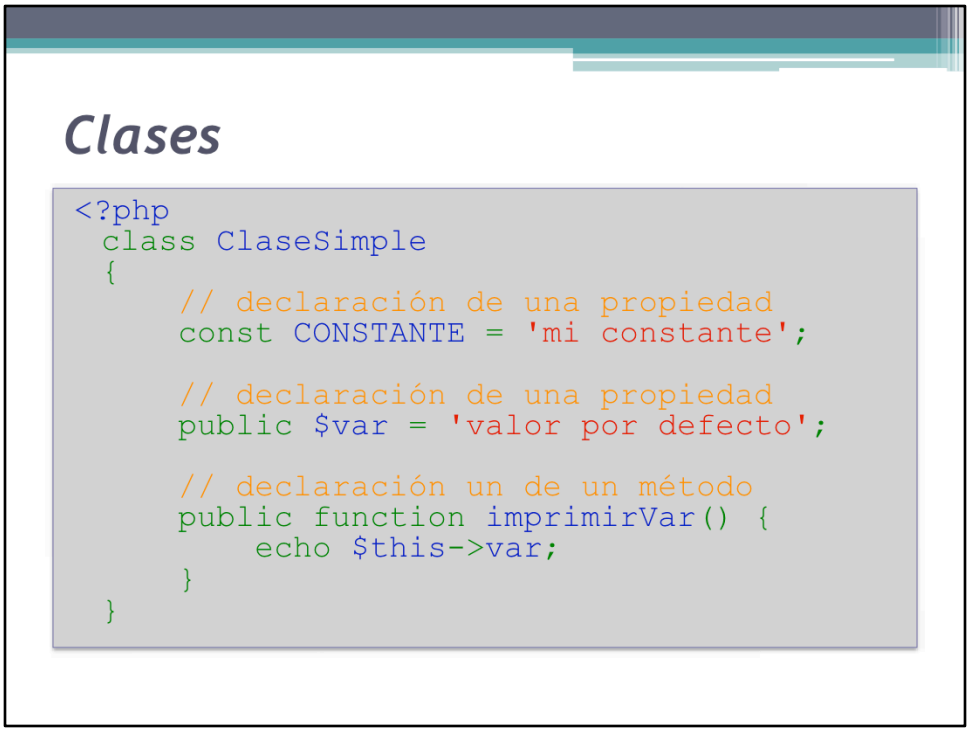
Algunas de las características técnicas básicas a resaltar de la POO en general.



Independientemente del lenguaje de programación que emplees, si éste soporta programación orientada a objetos entonces las tres características de arriba son fundamentales al momento de programar. Estas tres características no sólo cambian la forma de codificar algoritmos, sino que proveen al programador de toda una nueva estrategia al momento de enfrentar las etapas de análisis y desarrollo de sistemas. Dominar estas técnicas implícitas en la POO es fundamental para el éxito del proyecto.

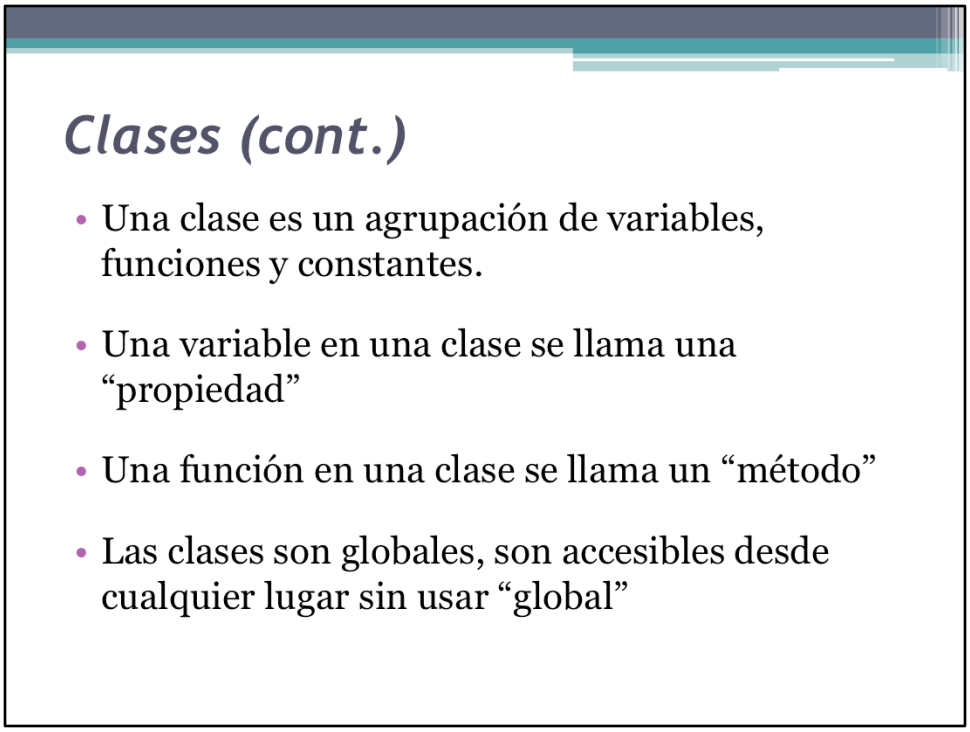


En este apartado exploraremos las herramientas y conceptos básicos de la POO en general pero siempre con ejemplos concretos en el lenguaje PHP.

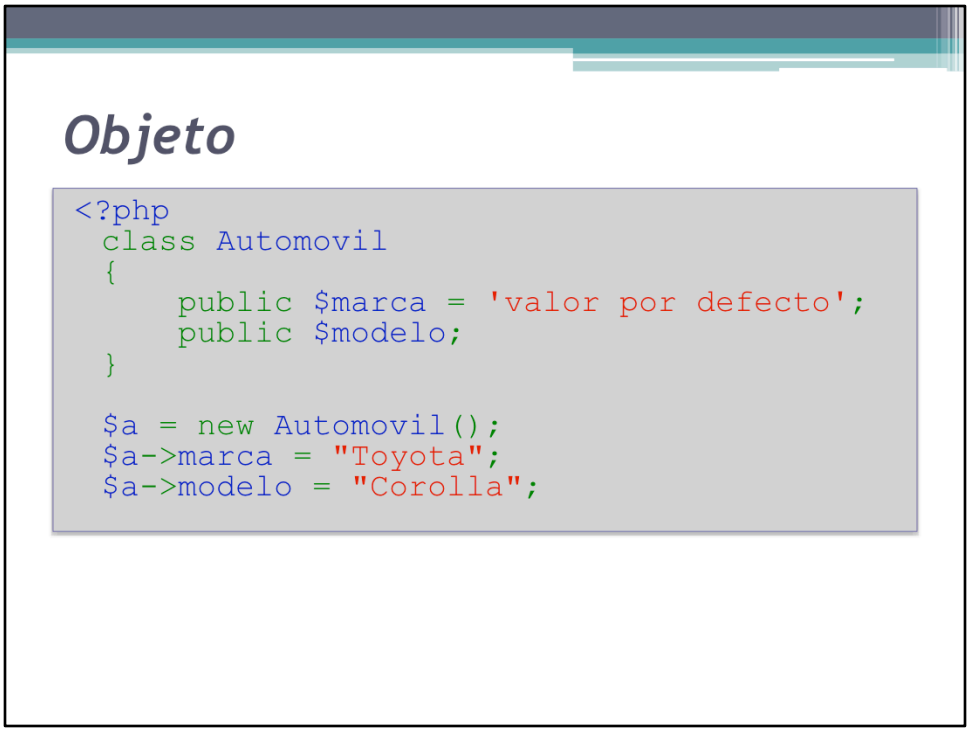


Como ya se ha reflejado antes, toda clase consta de la palabra clave **class** seguido del nombre de la clase y un bloque de código entre llaves.
Dentro del bloque de código se pueden crear sólo tres tipos de bloques básicos: constantes, variables y métodos. Una vez creadas dentro de las llaves, tanto la constante, como la variable, como la función pertenecen a la clase, y para ser utilizadas hay que acceder a través de la clase.

ClaseSimple.php



Aquí se muestran las carácterísitcas básicas de una clase. Una analogía apropiada (pero muy básica) al momento de pensar en una clase es pensar en un molde del cuál se van a extraer múltiples "copias" u "objetos" similares. A diferencia de un objeto físico, en este caso las copias serán dinámicas y pueden cambiar su comportamiento y estructura al momento de ejecutar un programa.

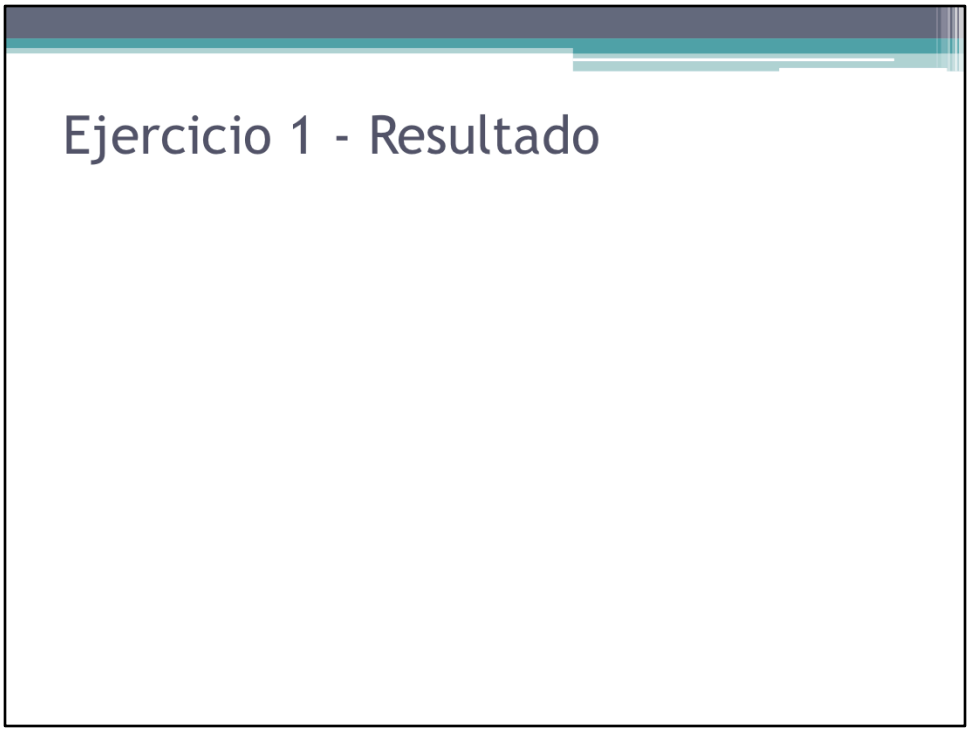


Aquí 'Automovil' es la clase y la variable $a es un objeto (instancia o copia personalizada) sobre la clase Automovil. La palabra clave new hace que Automovil junto a todas sus propiedades y funciones se copien a $a.

Objeto.php

Ejercicio 1 – Objeto

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con el objeto
- Especificaciones:
  - Crear una clase llamada fruta con las propiedades color y tamano
  - Crear una instancia sobre esta clase y asignarle valores a las propiedades
  - Imprimir los valores de estas propiedades



Resultado del ejercicio.

Ejercicio 1 - Solución

```php
<?php

 class Fruta{
   public $color;
   public $tamano;
 }

 $fruta = new Fruta;
 $fruta->color = "Amarillo";
 $fruta->tamano = "Mediano";

 echo "\n<br />$fruta->color";
 echo "\n<br />$fruta->tamano";
```

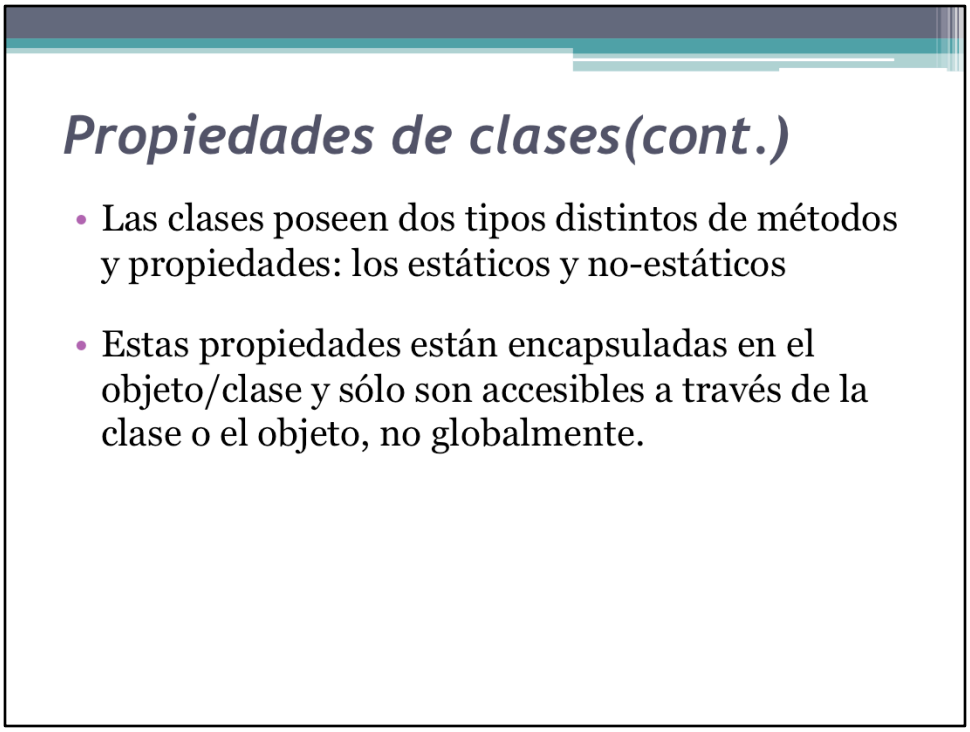
Objeto (cont.)

- Un objeto es una variable más (como los arreglos o enteros), sólo que actúa como una copia de la clase.
- Podemos tener todas las copias (objetos) que se deseen de la misma clase.
- Un objeto suele denominarse como una instancia de una clase

Propiedades de clases

```php
<?php
  class Automovil{

      //Modelo
      public $marca;

      //Marca
      public static $modelo;
  }

  Automovil::$modelo = 'Toyota';

  $a = new Automovil();
  $a->marca = '4Runner';
  $b = new Automovil();
  $b->marca = 'Corolla';

  echo "\n<br />" . Automovil::$modelo;
  echo "\n<br />" . $a->marca;
  echo "\n<br />" . $b->marca;
```

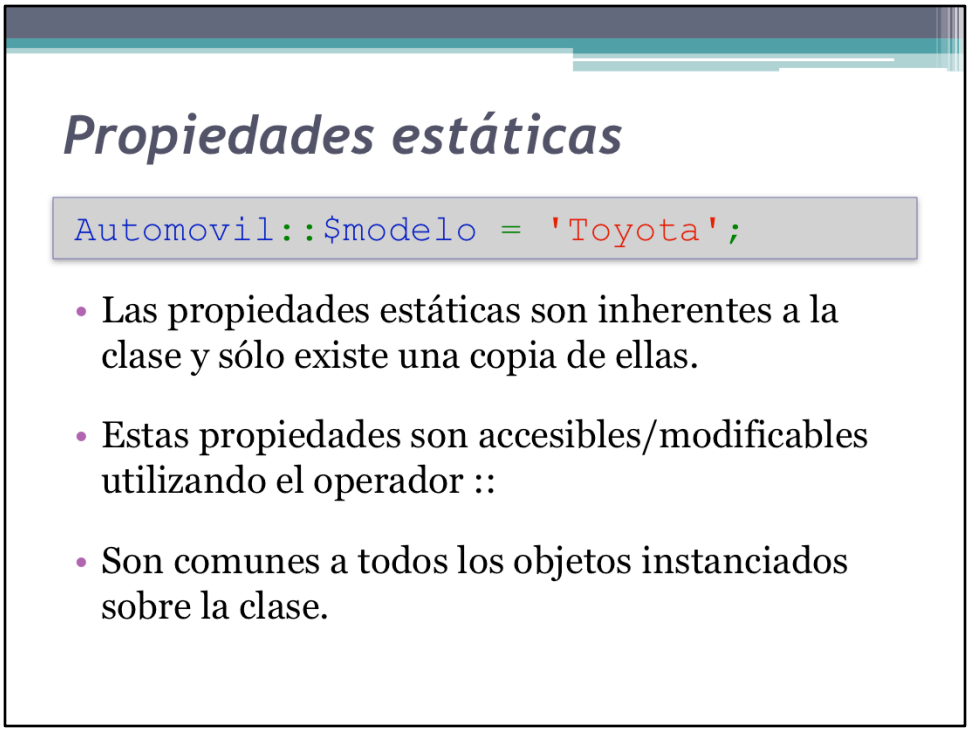
PropiedadesDeClases.php



Una de las diferencias entre las propiedades de una clase y las variables comunes es que las propiedades al ser declaradas dentro de la clase no pueden hacer referencia a otras variables. Fuera de la clase si se pueden asignar otros valores:

```
$miModelo = “Toyota”;
Class Automovil{
  //Correcto:
  public static $modelo;
  //Incorrecto:
  public static $modelo = $miModelo;
}
Automovil::$modelo = $miModelo;
```

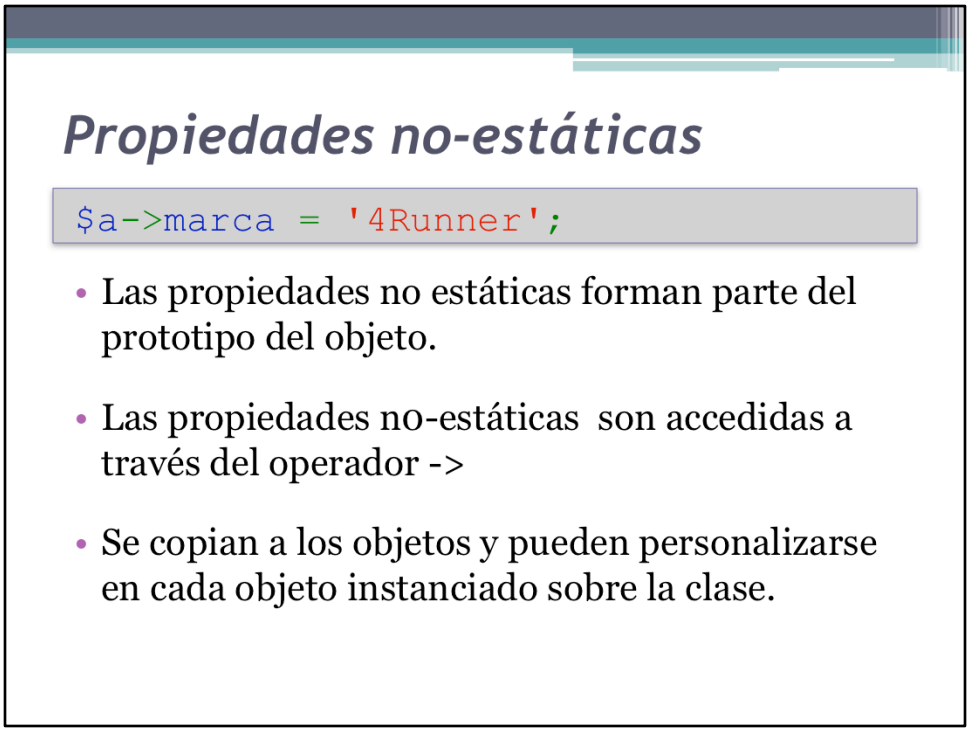
Esta limitación se asignación se debe a que el intérprete tiene que conocer cuál es el valor de todas las propiedades de una clase al momento de leer el código.



Una de las diferencias entre las propiedades de una clase y las variables comunes es que las propiedades al ser declaradas dentro de la clase no pueden hacer referencia a otras variables. Fuera de la clase si se pueden asignar otros valores:

```
$miModelo = “Toyota”;
Class Automovil{
  //Correcto:
  public static $modelo;
  //Incorrecto:
  public static $modelo = $miModelo;
}
Automovil::$modelo = $miModelo;
```

Esta limitación se asignación se debe a que el intérprete tiene que conocer cuál es el valor de todas las propiedades de una clase al momento de leer el código.



Una de las diferencias entre las propiedades de una clase y las variables comunes es que las propiedades al ser declaradas dentro de la clase no pueden hacer referencia a otras variables. Fuera de la clase si se pueden asignar otros valores:

```
$miModelo = “Toyota”;
Class Automovil{
  //Correcto:
  public static $modelo;
  //Incorrecto:
  public static $modelo = $miModelo;
}
Automovil::$modelo = $miModelo;
```

Esta limitación se asignación se debe a que el intérprete tiene que conocer cuál es el valor de todas las propiedades de una clase al momento de leer el código.

Métodos

- Tanto los métodos estáticos como no-estáticos disponen de la palabra clave 'self' que hace referencia a la misma clase
- Ambos tipos de métodos disponen adicionalmente de la palabra clave 'parent' que hace referencia a la clase padre (Herencia).

Ejemplo usando 'self'

```php
<?php
  class Automovil{

    public static $marca;

    public static function establecerMarca
  ($marca){
       self::$marca = $marca;
    }
  }

  Automovil::establecerMarca('Volkswagen');

  echo Automovil::$marca;
```

Una de las diferencias entre las propiedades de una clase y las variables comunes es que las propiedades al ser declaradas dentro de la clase no pueden hacer referencia a otras variables. Fuera de la clase si se pueden asignar otros valores:

```
$miModelo = "Toyota";
Class Automovil{
  //Correcto:
  public static $modelo;
  //Incorrecto:
  public static $modelo = $miModelo;
}
Automovil::$modelo = $miModelo;
```

Esta limitación se asignación se debe a que el intérprete tiene que conocer cuál es el valor de todas las propiedades de una clase al momento de leer el código.

EjemploSelf.php

Objeto y $this

```php
<?php
class Automovil
{
    public $marca;
    public $modelo;

    public function imprimir() {
        echo "<p>Hola! soy un $this->marca modelo $this->modelo</p>";
    }
}

$a = new Automovil();
$a->marca = 'Volkswagen';
$a->modelo = 'Jetta';
$a->imprimir();
```

$a y $b son copias distintas, permitiendo así que $a tenga una propiedad 'modelo' que es suya propia y distinta de la propiedad 'modelo' de la variable $b.

EjemploThisCorrecto.php

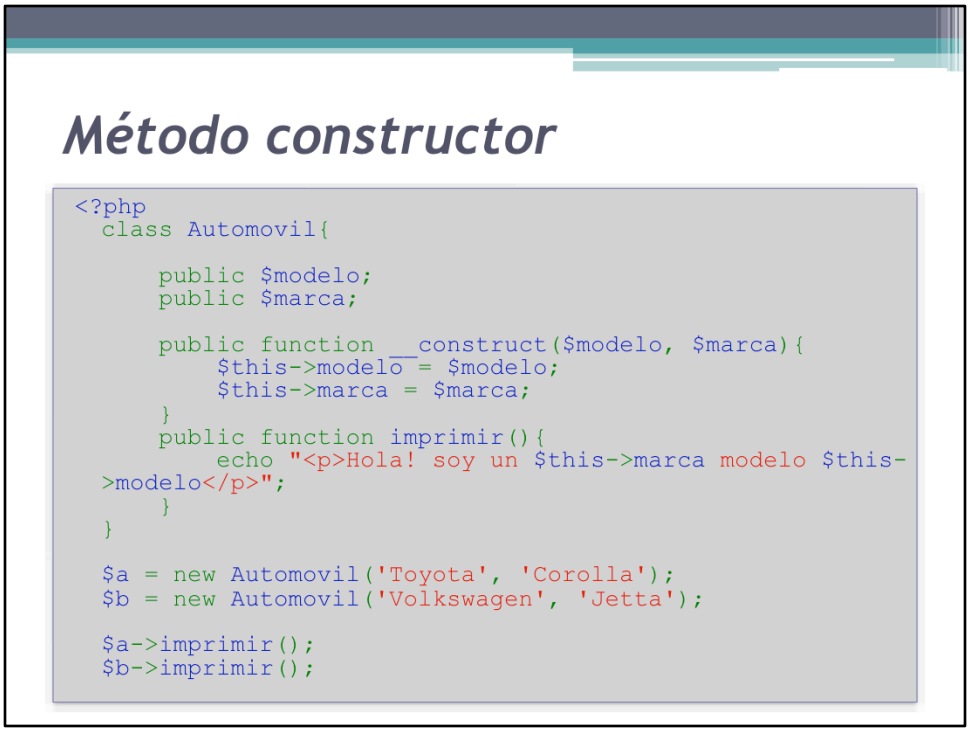
Objetos y $this (cont.)

- Los métodos no-estáticos disponen adicionalmente de la variable $this.
- Usando esta variable dentro del objeto, se puede acceder a otros métodos/propiedades del objeto.
- La variable $this no está disponible en las clases
- En el ejemplo anterior es equivalente usar $a desde afuera del objeto que $this desde adentro (para las propiedades protegidas esto es distinto).

Objeto y $this (uso incorrecto)

```php
<?php
  class Automovil{
      public $marca;

      public static function obtenerMarca
  ($marca){
          return $this->marca;
      }
  }
  $a = new Automovil();
  $a->marca = 'Volkswagen';
  $b = new Automovil();
  $b->marca = 'Toyota';

  //Volkswagen o Toyota? Fatal error
  echo Automovil::obtenerMarca();
```

EjemploThisIncorrecto.php



El constructor permite generar un estado inicial del objeto que se adapte a los requerimientos. Aquí nos interesa guardar los dos parámetros que se le pasan al constructor, posiblemente para utilizarlos después.

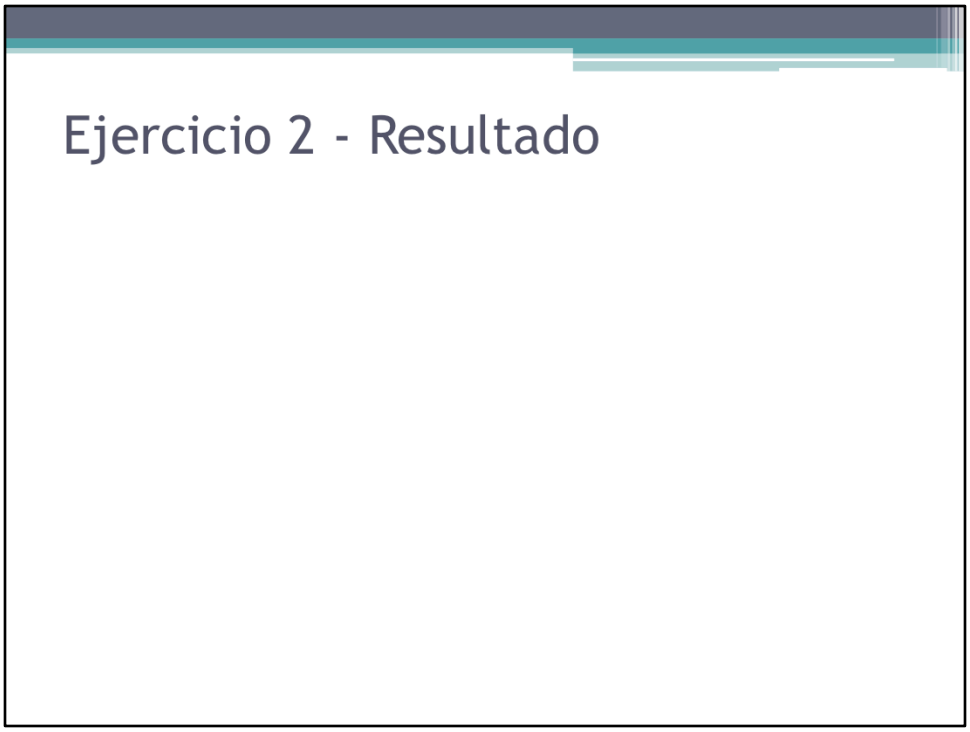
Constructor.php

Método constructor (cont.)

- El constructor se llama automáticamente al instanciarse el objeto.
- Se define como un procedimiento: esto significa que no devuelve ningún valor.
- No es llamado más veces durante la vida útil del objeto

Ejercicio 2 – Métodos

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con el uso de métodos en objetos
- Especificaciones:
  - A la clase Fruta agregarle un constructor que reciba las propiedades color y tamaño y las guarde
  - A la clase Fruta agregarle un método llamado imprimir que escriba "Esta es una fruta de color **<color>** de tamaño **<tamano>**"
  - El constructor debe llamar al método imprimir



Resultado del ejercicio.

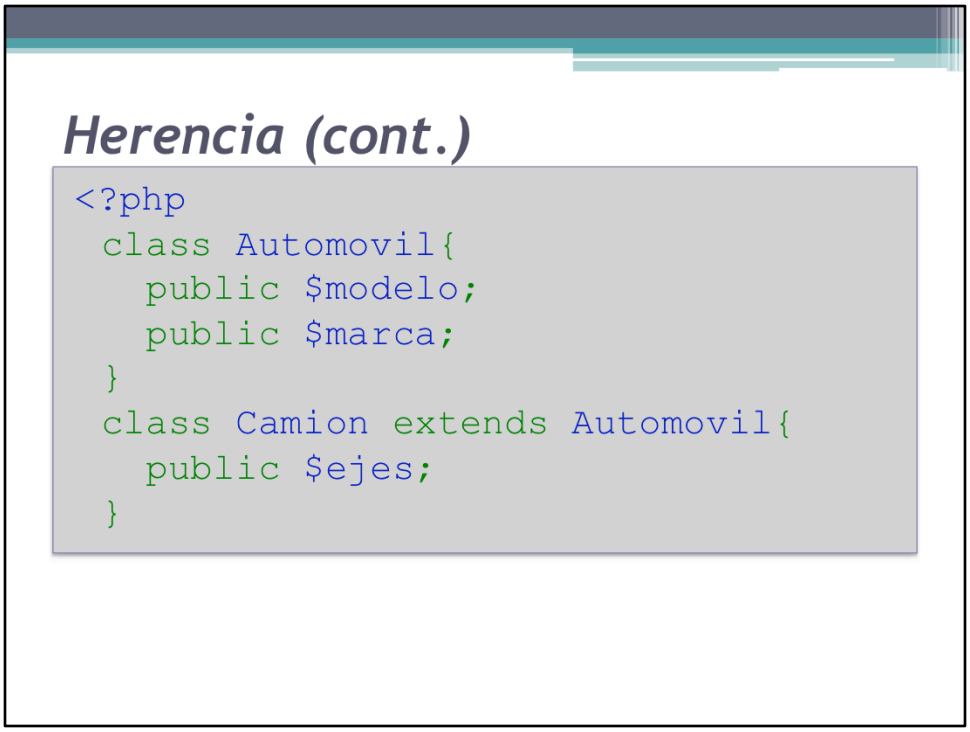
Ejercicio 2 - Solución

```
<?php

  class Fruta{
    public $color;
    public $tamano;

    public function __construct($color, $tamano){
      $this->color = $color;
      $this->tamano = $tamano;
      $this->imprimir();
    }

    public function imprimir(){
      echo "<p>Esta es una fruta de color $this-
  >color de tamaño $this->tamano</p>";
    }
  }

  $fruta = new Fruta('Amarillo', 'Mediano');
```

Herencia

La herencia de clases se define como la extensión de clases.

Una clase puede necesitar extender la funcionalidad de otra clase ya existente.

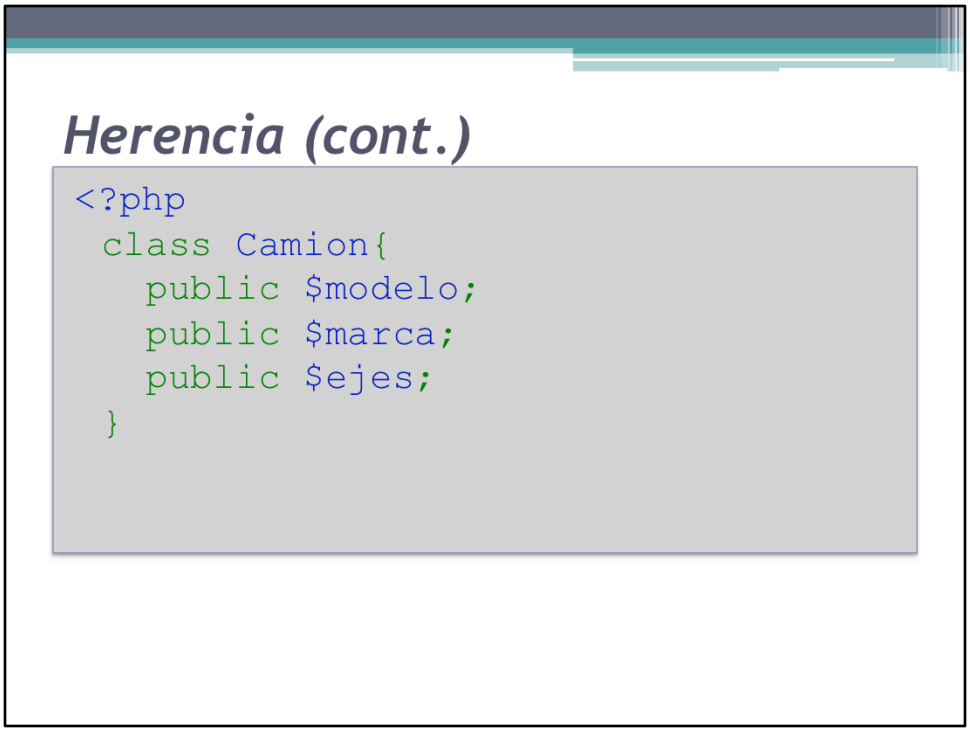


Ahora la clase Camion tiene las tres propiedades: $modelo, $marca y $ejes.

HerenciaBasico.php

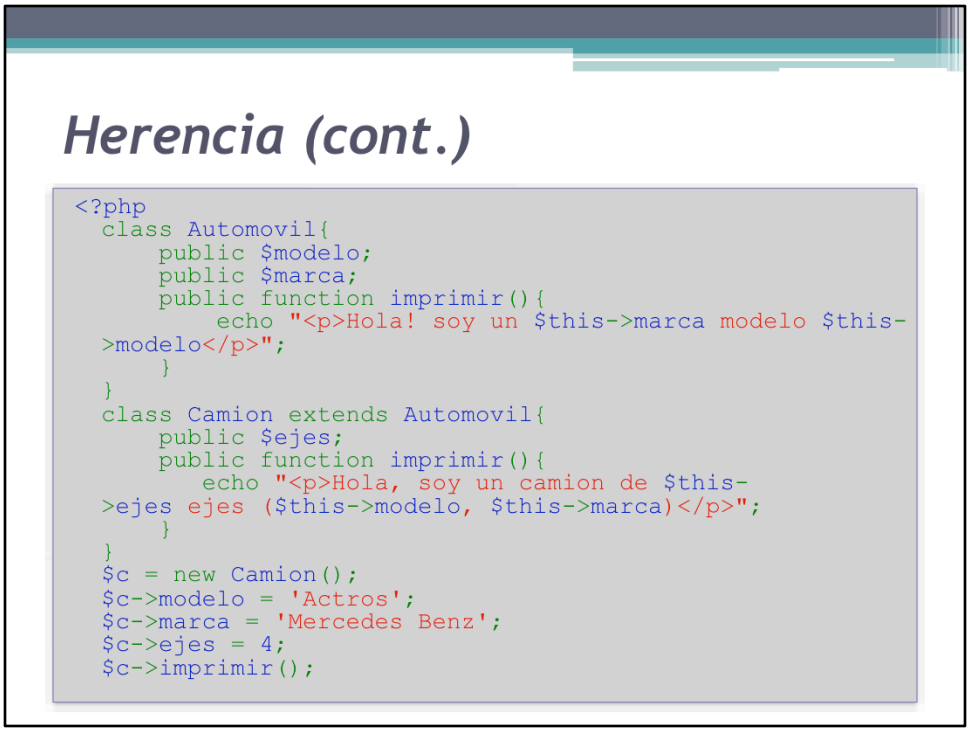
Herencia (cont.)

- Ahora tenemos una nueva clase llamada Camion
- Esta nueva clase además de la propiedad 'ejes' también hereda las propiedades 'modelo' y 'marca'



Esta es una declaración completamente equivalente a la anterior, sólo que no se extiende, sino que se crean directamente en la clase.

HerenciaBasicoExplicacion.php



HerenciaSobrescritura.php

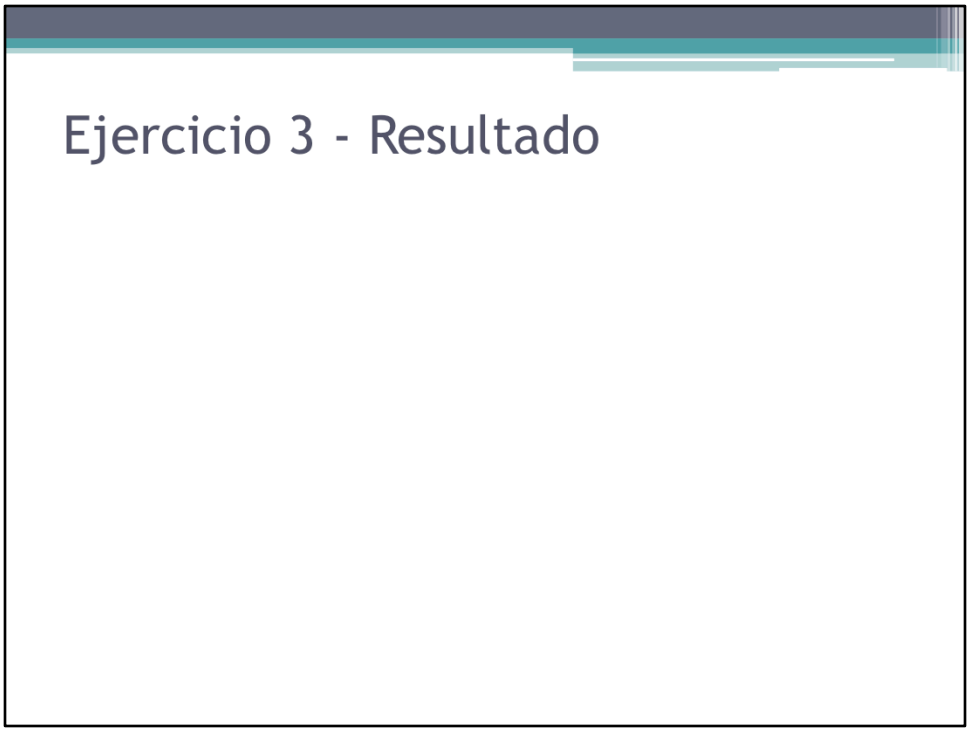
Herencia (cont.)

- Tanto propiedades, métodos como constantes pueden ser reemplazados en clases hijas.
- Se puede agregar toda la funcionalidad que se desee
- La clase que extiende es una agrupación de todos los métodos, propiedades y constantes que defina más los de la clase de la que extiende.

Cabe destacar que si se sobrescribe un método ya existente, el nuevo método debe tener la misma firma que la del padre (debe recibir los mismos parámetros). Si esto no ocurre, se genera un error de nivel E_STRICT. El único método para el cual esto no aplica es el constructor. El constructor puede redefinir los parámetros que requiere.

Ejercicio 3 – Herencia

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con la herencia
- Especificaciones:
  - Crear una clase Uva que herede de Fruta
  - Agregarle la propiedad 'tieneSemilla'
  - Agregarle el método 'tieneSemilla' que devuelva el valor de la propiedad 'tieneSemillá'
  - Imprimir si la fruta tiene semillas o no en función de lo que devuelva el método 'tieneSemilla'



Resultado del ejercicio.

Ejercicio 3 - Solución

- Ver Ejercicio3.php

Herencia y constructores

```
<?php

 class Automovil{
     public function __construct(){
         echo "Hola, soy un automovil!"
 ;
     }
 }
 class Camion extends Automovil{
     public function __construct(){
         echo "Hola, soy un camion!";
     }
 }
 $c = new Camion();
```

HerenciaConstructores.php

Herencia y constructores (cont.)

- El constructor a ejecutar será el de la clase que extiende
- El constructor del padre no será invocado a de no ser que sea solicitado explícitamente

Invocando métodos anulados

```php
<?php
  class Automovil{
      public $modelo;
      public $marca;
      public function __construct($marca, $modelo){
          $this->marca = $marca;
          $this->modelo = $modelo;
      }
  }

  class Camion extends Automovil{
      public $ejes;
      public function __construct
  ($marca, $modelo, $ejes){
          $this->ejes = $ejes;
          parent::__construct($marca, $modelo);
      }
  }

  $c = new Camion('Mercedes Benz', 'Actros', 4);
```

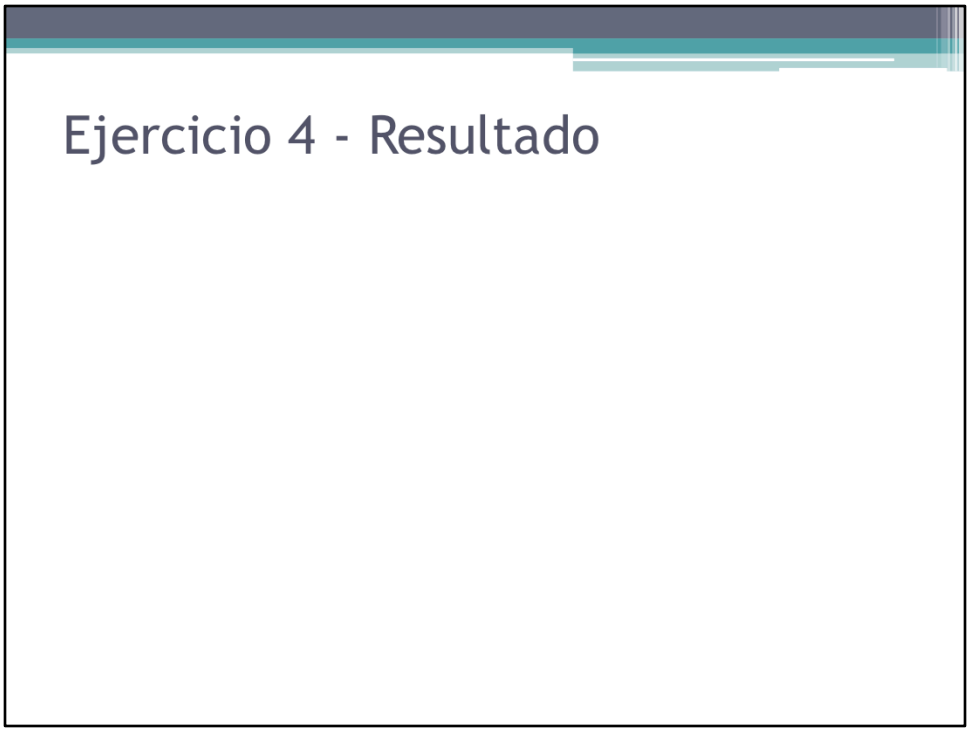
HerenciaInvocarAnulados.php

Invocando métodos anulados (cont.)

- La palabra clave 'parent' hace referencia a la clase de la cual se está heredando
- Al usar parent::metodo(); se llama a la función que se sobrescribió
- La instrucción parent::metodo(); se puede hacer tanto sobre objetos como sobre métodos estáticos de clases

Ejercicio 4 – Métodos anulados

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con el uso de métodos anulados
- Especificaciones:
  - Crear un nuevo constructor en la clase Uva
  - El constructor debe imprimir “Soy el constructor del hijo!” y luego llamar al constructor padre.



Resultado del ejercicio.

Ejercicio 4 - Solución

- Ver Ejercicio4.php

Restringiendo acceso

```php
<?php
 class Automovil{
     public $modelo = 'Toyota';
     protected $marca = 'Corolla';
 }
 $a = new Automovil();
 //Imprime Toyota
 echo $a->modelo;
 //Genera un fatal error
 echo $a->marca;
```

RestringirAcceso.php

Restringiendo acceso (cont.)

- La palabra clave 'public' permite que el método/ propiedad sea accesible desde afuera del objeto
- La palabra clave 'protected' no permite que estos métodos sean llamados desde afuera o que las propiedades sean leídas/modificadas desde afuera

Restringiendo acceso (cont.)

```
<?php
  class Automovil{
      protected $marca;
      protected $modelo;

      public function __construct($marca, $modelo){
          $this->marca = $marca;
          $this->modelo = $modelo;
      }
      public function imprimir(){
          echo "<p>Hola! soy un $this-
  >marca modelo $this->modelo</p>";
      }
  }
  $a = new Automovil('Toyota', 'Corolla');
  $a->imprimir();
```

RestringirAccesoUso.php

Restringiendo acceso (cont.)

- El objeto puede garantizar que las propiedades protegidas tienen valores que él colocó.
- Si un objeto depende absolutamente de el estado de sus propiedades, puede garantizar que funcionará correctamente si nadie las modifica directamente.

Restringiendo acceso (métodos)

```php
<?php
  class Automovil{
      protected $marca;
      protected $modelo;

      public function __construct($marca, $modelo){
          $this->marca = $marca;
          $this->modelo = $modelo;
          //Imprime los datos correctamente
          $this->imprimir();
      }
      protected function imprimir(){
          echo "<p>Hola! soy un $this-
  >marca modelo $this->modelo</p>";
      }
  }
  $a = new Automovil('Toyota', 'Corolla');
  //Genera un fatal error
  $a->imprimir();
```

RestringirAccesoMetodos.php

Restringiendo acceso (métodos)(cont.)

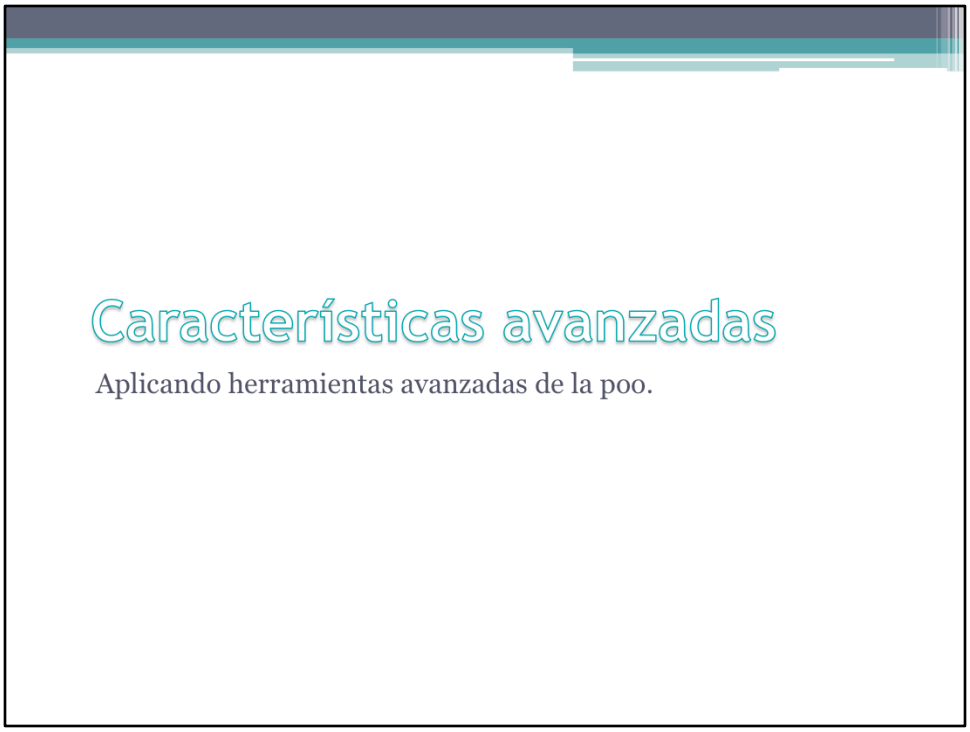
- Los métodos también pueden ser protegidos.
- Los métodos protegidos sólo pueden ser llamados dentro del objeto usando $this->metodo();
- Los métodos protegidos garantizan cierta funcionalidad sólo sea aplicada bajo un estado seguro que conoce el objeto

Restringiendo acceso (private)

```php
<?php
  class Automovil{
      private $marca = 'Toyota';
      public function imprimirAutomovil(){
          echo $this->marca;
      }
  }
  class Camion extends Automovil{
      public function imprimirCamion(){
          echo $this->marca;
      }
  }
  $c = new Camion();
  //Imprime Toyota
  $c->imprimirAutomovil();
  //No imprime nada
  $c->imprimirCamion();
```

Restringiendo acceso (private)(cont.)

- Los métodos/propiedades privados se diferencian de los protegidos sólo cuando hay herencia
- Los métodos/propiedades privados son visibles sólo a la clase/objeto padres. En los hijos actúan como si no existieran.
- Cuando una clase no ha sido extendida, los métodos/propiedades privados actúan idéntico a los protegidos.



instanceof

```php
<?php
 class Automovil{

 }
 class Camion extends Automovil{

 }
 $a = new Camion();
 var_dump($a instanceof Camion);
 var_dump($a instanceof Automovil);
```

Instanceof.php

instanceof

- Este es un operador booleano que actúa sobre un objeto y una clase.
- A la izquierda va el objeto y a la derecha va el nombre de la clase.
- El operador determinar si el objeto es instancia de esa clase.
- Un objeto es instancia de su clase y de todas las clases que esta hereda.

Instanceof (adicional)

- is_a funciona idéntico al operador 'instanceof'

```
is_a($a, 'Automovil');
```

- El siguiente ejemplo imprimiría 'Automovil'

```
echo get_class($a);
```

Clases abstractas

```php
<?php

  abstract class Filtro{

    public function filtrarMuchos
  ($elementos){
      $resp = array();
      foreach ($elementos as $id => $elem){
        $resp[$id] = $this->filtrar($elem);
      }
      return $resp;
    }


    abstract public function filtrar();
  }
```

ClasesAbstractas.php

Clases abstractas (cont.)

- El objetivo de una clase abstracta es definir un prototipo de comportamiento.
- Este prototipo es incompleto por definición y debe ser extendido para ser utilizado.
- Las clases abstractas no son instanciables, sólo las que heredan de ellas.

Clases abstractas (implement.)

```php
<?php

  class Filtro_Vocales extends Filtro{
    public function filtrar($cadena){
      return preg_replace('/[aeiou]/',
                          '', $cadena);
    }
  }

  $f = new Filtro_Vocales();
  $resp = $f->filtrarMuchos(array(
    'Palabra', 'Murcielago'
  ));

  echo '<pre>'.print_r($resp,true).'</pre>';
```

ClasesAbstractasFiltroLetras.php

Clases abstractas (implement.)

- La clase anterior funciona idéntico a como si hubiera heredado de una clase no abstracta
- Posee todos los métodos de la clase abstracta
- Adicionalmente está obligada a definir los métodos que fueron definidos como abstractos en la clase abstracta

```
//Fatal error! el metodo filtrar no fue extendido
  class Filtro_Vocales extends Filtro
  {

  }
```

Clases abstractas (utilidad)

- Al aplicar el operador 'instanceof' sobre un objeto, se puede saber si hereda de alguna clase abstracta específica
- Muchas veces la funcionalidad base es común para todas las implementaciones, pero parte del comportamiento varía según el contexto

Clases abstractas (implement.)

```php
<?php

  class Filtro_Pares extends Filtro{
    public function filtrar($numeros){
      foreach ($numeros as $clave => $n){
        if ($n % 2 == 0){
          unset($numeros[$clave]);
        }
      }
      return $numeros;
    }
  }

  $f = new Filtro_Pares();
  $resp = $f->filtrarMuchos(array(
    array(1, 2, 3, 4, 5), array(10, 501)
  ));
  echo '<pre>' . print_r($resp, true) . '</pre>';
```

ClasesAbstractasFiltroNumeros.php

Interfaces

- Son clases que no pueden ser instanciadas.
- Son especificaciones de métodos que debe contener una clase.
- Una clase puede implementar una interfaz para asegurarse que sigue unas especificaciones determinadas
- Las interfaces sólo contienen firmas de métodos y constantes.

Interfaces (cont.)

```php
<?php

  interface Filtro{
    public function filtrar($elem);
  }

  class Filtro_Vocales implements Filtro{
    public function filtrar($cadena){
      return preg_replace('/[aeiou]/',
                          '', $cadena);
    }
  }

  $a = new Filtro_Vocales();
  echo $a->filtrar('Dinosaurio');
```

Interfaces.php

Interfaces (cont.)

- La palabra clave 'implements' obliga a la clase a seguir las especificaciones de la interfaz.
- El operador 'instanceof' se puede aplicar entre objetos e interfaces también.
- Una clase que implementa una interfaz funciona idéntico a una clase común, sólo que al saber qué interfaz implementa, se sabe más de su funcionalidad.

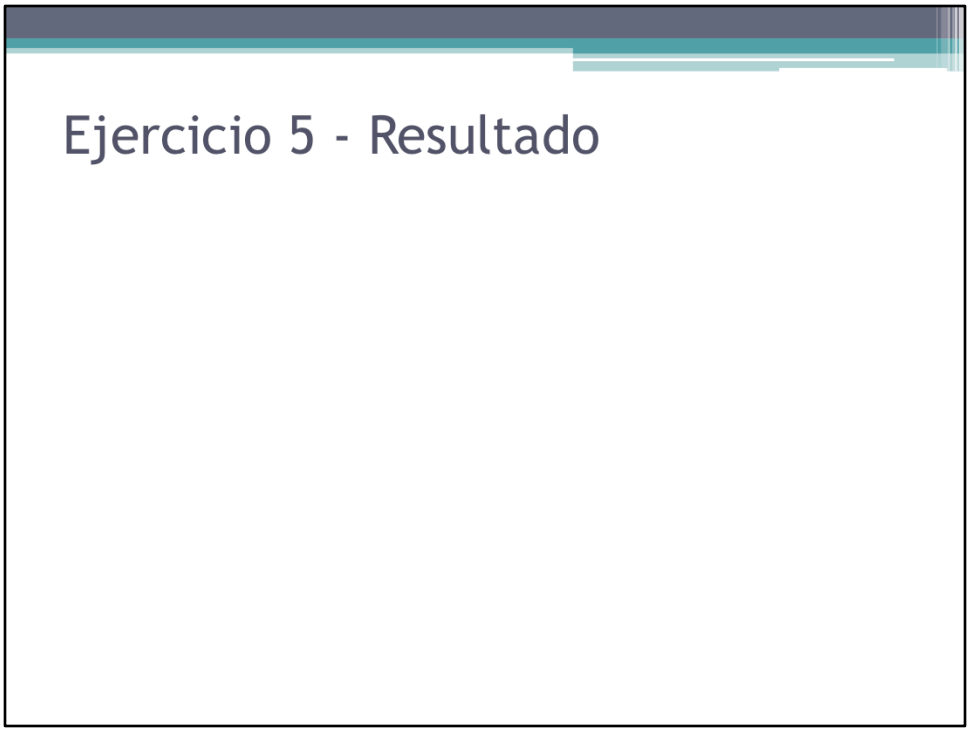
Interfaces (cont.)

- Una clase puede implementar varias interfaces mientras no estas interfaces no compartan métodos
- Una interfaz puede extender a otra utilizando el operador 'extends'

```php
<?php
  interface Carro{
    public function encender();
  }

  interface Camion extends Carro{
    public function cargar();
  }
```

Ejercicio 5 – Clases abstractas

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con las clases abstractas
- Especificaciones:
  - Convertir la clase Fruta en una clase abstracta
  - Al final del programa intentar instanciar la clase Fruta



Resultado del ejercicio.

Ejercicio 5 - Solución

- Ver Ejercicio5.php

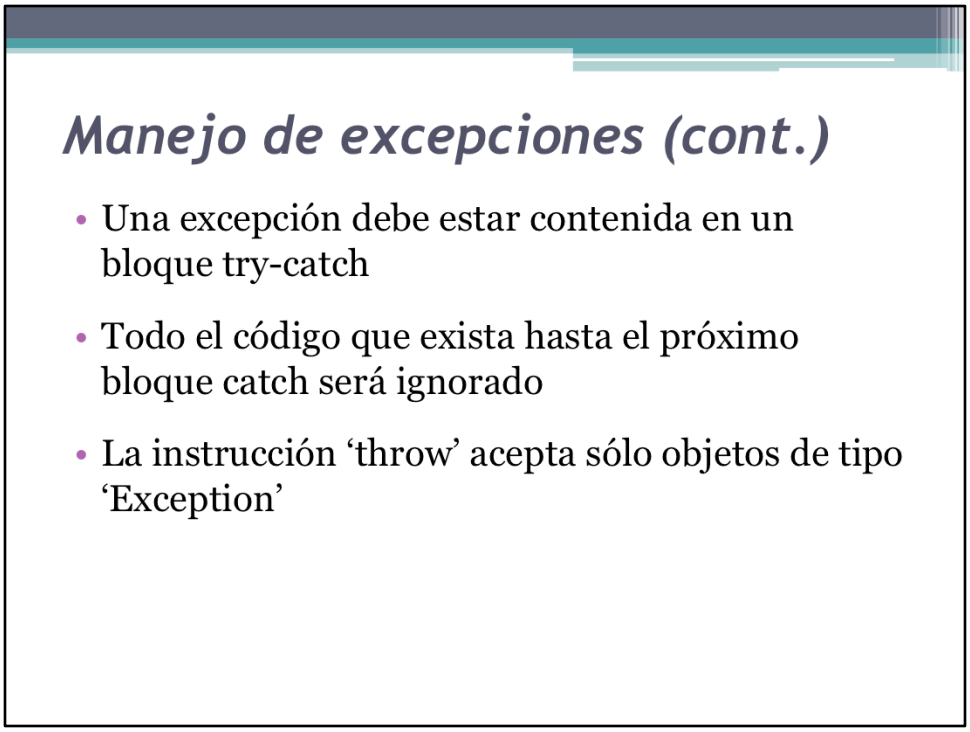
Manejo de excepciones

- PHP implementa el uso de excepciones para el manejo de errores.
- Una excepción es una situación bajo la cual el programa no puede continuar su ejecución normal.
- Cuando un programa detecta una situación inestable puede 'lanzar' una excepción que debe ser 'atrapada'.

Manejo de excepciones (cont.)

```php
<?php

 try{

   echo "<p>antes del error</p>";

   throw new Exception('Error..!');

   //Código que no se ejecuta
   echo "<p>despues del error</p>";

 }catch(Exception $e){
   //Recuperacion del error
 }
```

ExcepcionesIntroduccion.php



Hablar sobre el relanzado de excepciones (tercer parámetro)

Manejo de excepciones (cont.)

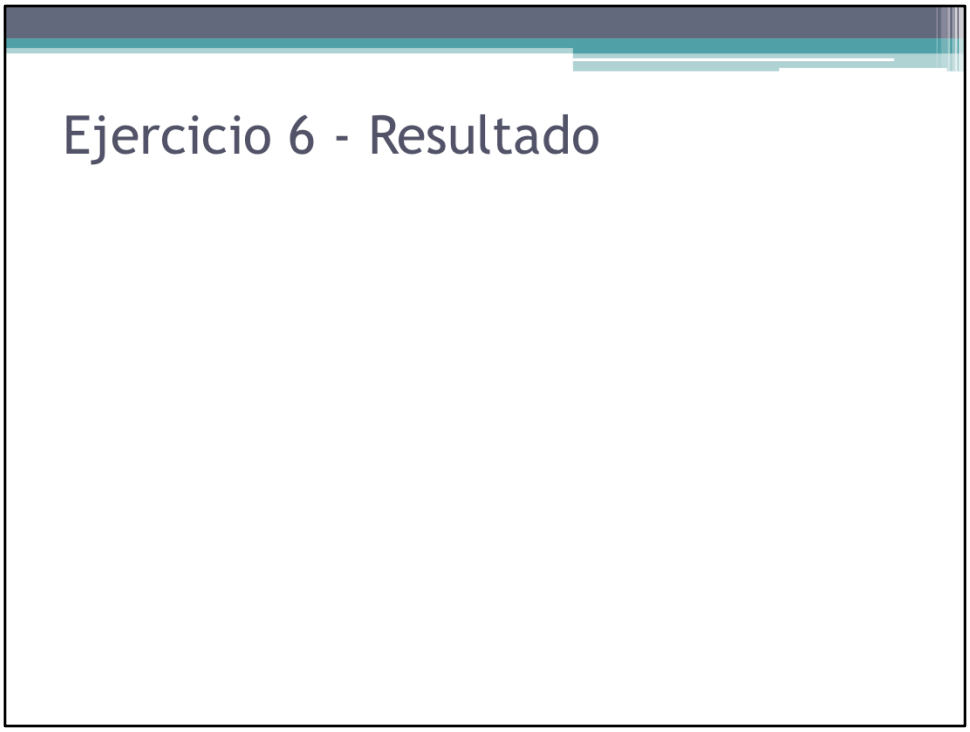
```php
<?php

  function imprimir($string){
    if (!is_string($string)){
      throw new Exception('Solo imprimimos cadenas', 700);
    }

    echo $string;
  }

  try{

    imprimir('<p>Esta llamada es correcta</p>');
    //Esta llamada es incorrecta
    imprimir(array('Carro', 'Camion'));


  }catch(Exception $e){

    echo '<p>Hubo un error: </p>';
    echo '<p>Mensaje: ' . $e->getMessage() . '</p>';
    echo '<p>Codigo: ' . $e->getCode() . '</p>';
  }
```

Las excepciones manejan un mensaje y un código de error que son propiedades del objeto Exception

ExcepcionesCodigo.php

Ejercicio 6 – Excepciones

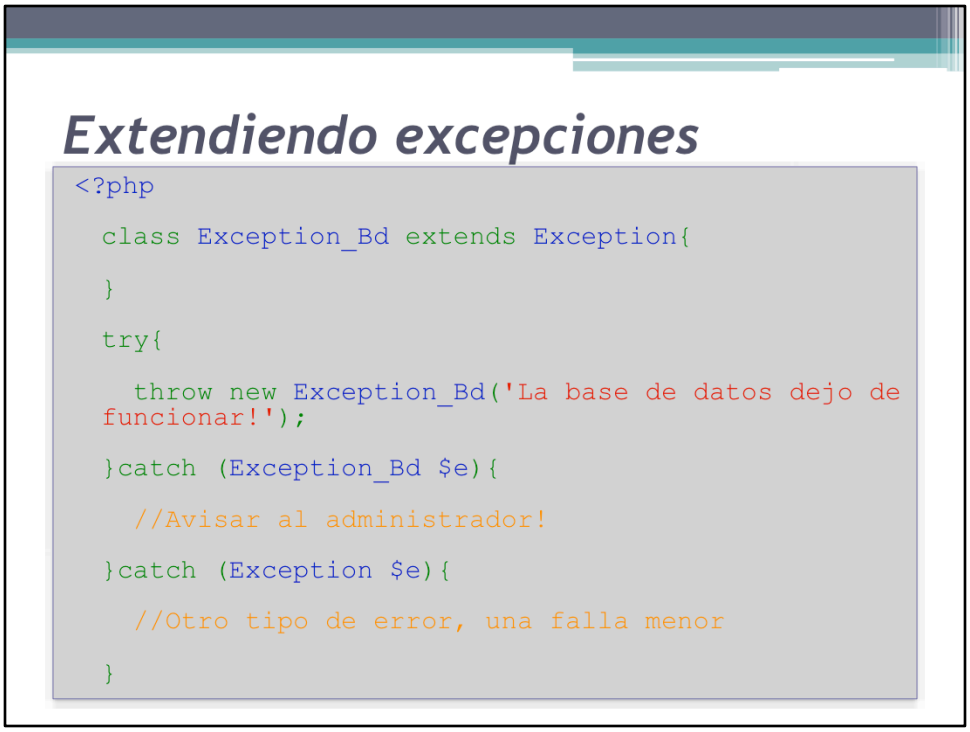
- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con las excepciones
- Especificaciones:
  - Quitar la instrucción que instancia a Fruta
  - Agregar una condición al constructor de la uva que lance una excepción si el color es naranja
  - Intentar crear una Uva naranja dentro de un bloque try-catch e imprimir el error generado



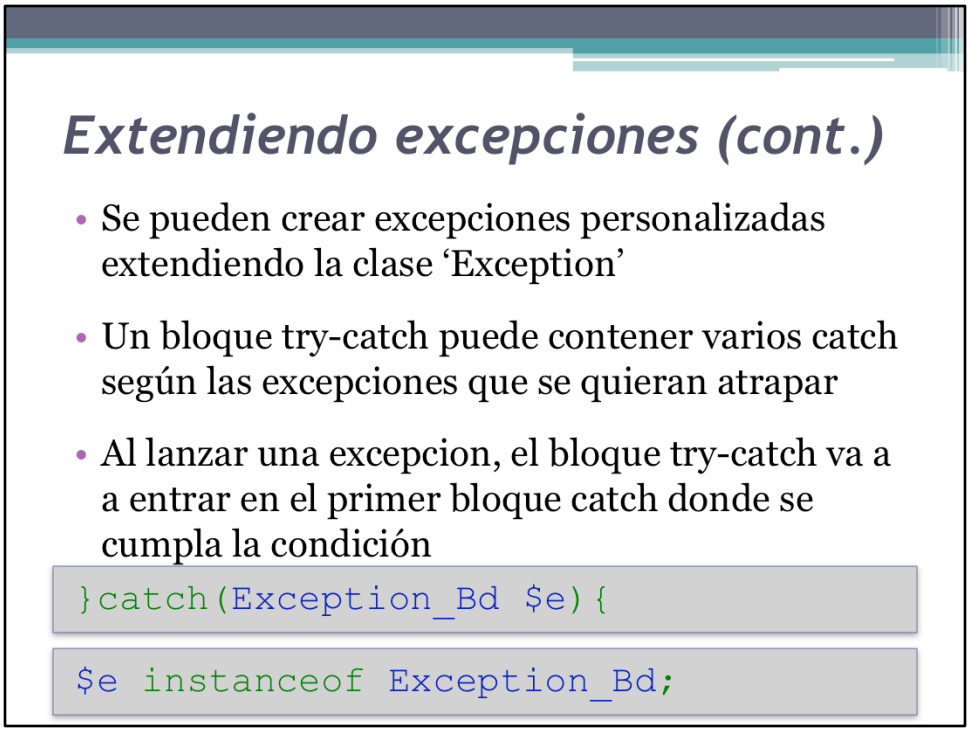
Resultado del ejercicio.

Ejercicio 6 - Solución

- Ver Ejercicio6.php



ExtendiendoExcepciones.php



Hablar sobre el relanzado de excepciones (tercer parámetro)

Extendiendo excepciones (cont.)

```php
<?php
  class Exception_Bd extends Exception{
  }
  try{
    throw new Exception_Bd
  ('La base de datos dejo de funcionar!');
  }catch (Exception $e){
    echo '<p>Soy una excepcion comun =)</p>';
  }catch (Exception_Bd $e){
    echo '<p>Avisamos al adminstrador!</p>';
  }
```

Explicar que se cumple instanceof Exception

AtrapandoExcepciones.php

Excepciones anidadas

```php
<?php

  class Exception_Bd extends Exception{
  }

  try{

    try{
      throw new Exception_Bd
  ('La base de datos dejo de funcionar!');
    }catch(Exception_Bd $e){
      echo '<p>Avisamos al administrador!</p>';
      //Relanzamos la excepcion
      throw $e;
    }

  }catch (Exception $e){
    echo '<p>Detenemos de forma limpia la ejecucion<
  /p>';
  }
```

En este ejemplo ambos bloques catch atrapan la excepción porque $e es instancia tanto de Exception_Bd como de Exception.

ExcepcionesAnidadas.php

Excepciones anidadas (cont.)

- Los bloques try-catch pueden estar anidados
- En un bloque catch se pueden lanzar más excepciones
- La misma excepción puede volver a ser lanzada, no hace falta crear una nueva excepción en cada 'throw'

Métodos finales

- Los métodos finales de una clase no se pueden sobrescribir en clases que hereden de ella.
- Pueden estar en cualquier tipo de clase
- Se pueden comportar idéntico a cualquier otro método

Métodos finales (cont.)

```php
<?php

  class Automovil{

    public function __construct(){
      //...
    }

    final public function imprimir(){
      //...
    }
  }

  class Camion extends Automovil{
    //Fatal error:
    //Cannot override final method Automovil::imprimir()
    public function imprimir(){
      //...
    }
  }
```

MetodosFinales.php

Clases finales

- Al igual que con los métodos, estas clases no pueden ser extendidas
- Pueden ser instanciadas igual a las demás

```php
<?php

  final class Automovil{

  }

  //Fatal error:
  //Class Camion may not inherit from final class (Automovil)
  class Camion extends Automovil{

  }
```

ClasesFinales.php

Métodos interceptores

- PHP ofrece un conjunto de métodos “mágicos”
- Estos métodos mágicos son llamados automáticamente por PHP cuando alguna acción concreta ocurre sobre un objeto/clase
- El constructor es un ejemplo específico de esto, es llamado cuando se crea el objeto para inicializarlo

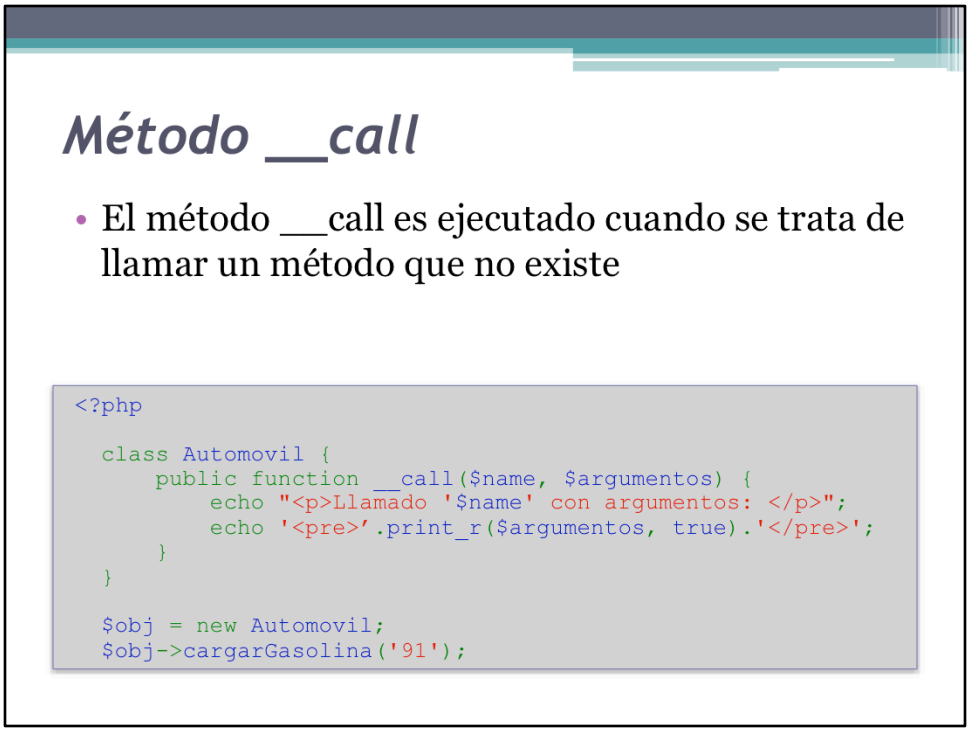
Método __destruct

- El método __destruct se define como un procedimiento a ser llamado únicamente cuando el objeto vaya a ser destruido
- Un objeto se destruye bajo dos eventos:
  - Se aplicó unset($obj)
  - El garbage collector detecta que no se tiene más acceso al objeto

Método __destruct (cont.)

```php
<?php

  class Bd{

    public function __construct(){
      echo '<p>Abrimos la base de datos!</p>';
    }

    public function __destruct(){
      echo '<p>Cerramos la base de datos!</p>';
    }
  }

  $a = new Bd();
```

MetodoDestruct.php



MetodoCall.php

Método __callStatic

- Al igual que con __call, __callStatic se invoca cuando se trata de acceder a un método estático que no existe

```php
<?php

  class Automovil {
      public function __callStatic($name, $argumentos) {
          echo "<p>Llamado '$name' con argumentos: </p>";
          echo '<pre>'.print_r($argumentos, true).'</pre>';
      }
  }

  Automovil::guardarPosiblesModelos(array(
      'Toyota', 'Volkswagen'
  ));
```

MetodoCallStatic.php

Método __get

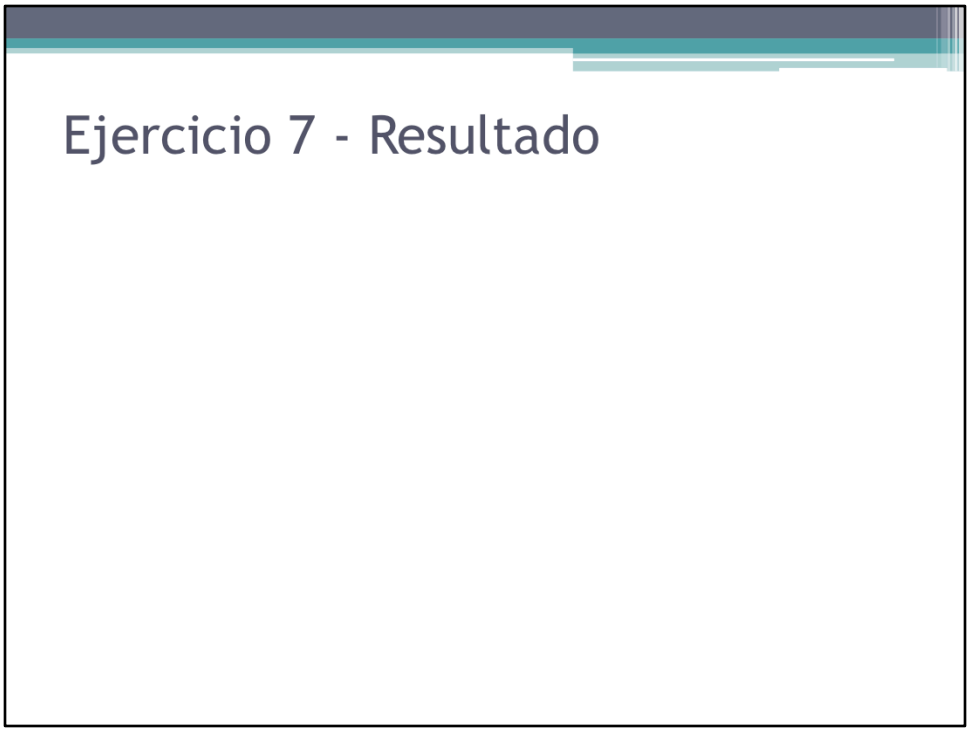
- Este método es llamado automáticamente cuando se trata de obtener una propiedad que no existe o es privada.

```php
<?php

  class Automovil{
    public function __get($name){
      return "Valor para '$name'";
    }
  }
  $obj = new Automovil;
  echo $obj->miPropiedad;
```

MetodoGet.php

Ejercicio 7 – Método __get

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse el método __get
- Especificaciones:
  - Crear una clase llamada MiClase
  - Crear el método __get, que si recibe de parámetro "mensaje" debe devolver "Propiedad que no existe (__get)"
  - Instanciar MiClase e imprimir su propiedad "mensaje"



Resultado del ejercicio.

Ejercicio 7 - Solución

```php
<?php

  class MiClase{
    public function __get($propiedad){
      if ('mensaje' == $propiedad){
        return 'Propiedad que no existe
  (__get)';
      }
    }
  }

  $o = new MiClase();

  echo "<p>$o->mensaje</p>";
```

Método __set

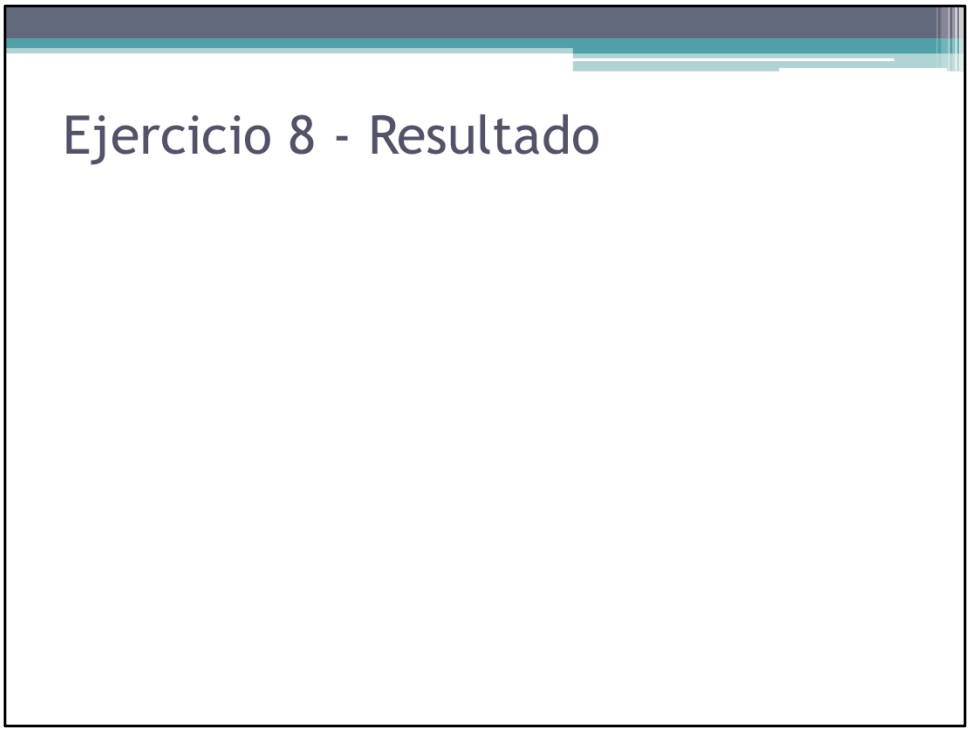
- Este método es llamado automáticamente cuando se trata de establecer una propiedad que no existe o es privada.

```php
<?php

  class Automovil{
    public function __set($nombre, $valor){
      echo "Guardando valor '$valor' para '$nombre'.";
    }
  }
  $obj = new Automovil;
  $obj->marca = 'Volkswagen';
```

MetodoSet.php

Ejercicio 8 – Método __set

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse el método __set
- Especificaciones:
  - Agregar una propiedad a MiClase $data (pública) que por defecto sea un arreglo
  - Agregar el método __set que automáticamente cree las propiedades como elementos del arreglo $data
  - Asignar al objeto la propiedad “mes” con el valor mayo
  - Leer de la propiedad data la clave “mes”



Resultado del ejercicio.

Ejercicio 8 - Solución

```php
<?php

  class MiClase{
    public $data = array();
    public function __set($propiedad, $valor){
      $this->data[$propiedad] = $valor;
    }
    public function __get($propiedad){
      if ('mensaje' == $propiedad){
        return 'Propiedad que no existe (__get)';
      }
    }
  }

  $o = new MiClase();
  echo "<p>$o->mensaje</p>";

  $o->mes = 'Mayo';
  echo "<p>{$o->data['mes']}</p>";
```

Método __isset

- Este método es llamado automáticamente cuando se llama 'isset' o 'empty' sobre propiedad que no existe o es privada.

```php
<?php

  class Automovil{
    public function __isset($nombre){
      echo "<p>La propiedad '$nombre'no existe..</p>";
      return false;
    }
  }
  $obj = new Automovil;
  if (isset($obj->prop1)){
    echo '<p>La propiedad p1 existe</p>';
  }
  if (empty($obj->prop2)){
    echo '<p>La propiedad p2 esta vacia</p>';
  }
```

MetodoIsset.php

Método __unset

- Este método es llamado automáticamente cuando se llama 'unset' sobre propiedad que no existe o es privada.

```php
<?php

  class Automovil{
    public function __unset($nombre){
      echo "<p>Eliminando propiedad '$nombre'..</p>";
    }
  }
  $obj = new Automovil;
  unset($obj->marca);
```

MetodoUnset.php

Representación de objetos por string

- Es posible la representación de un objeto por string usando la función serialize
- Este string puede guardarse en una base de datos y luego volver a convertirse en objeto a partir del string

Representación de objetos por string (cont.)

```php
<?php
  class Automovil{
    public $data = array();
  }

  $obj = new Automovil;
  $obj->data = array('marca' => 'Ford');
  $serializado = serialize($obj);

  echo $serializado;

  $obj = unserialize($serializado);
  echo '<pre>' .
       print_r($obj->data, true) .
       '</pre>';
```

Serialize.php

Método __sleep

- Al serializar un objeto, automáticamente se llama al método __sleep
- Nótese la diferencia con __destruct, ya que luego de serializar un objeto aún se puede llamar a __destruct
- Este método (si existe) debe devolver un arreglo con los nombres de las propiedades a serializar

MetodoSleep.php

Método __wakeup

- Al unserializar un objeto, automáticamente se llama al método __wakeup
- Este método a diferencia del __construct debe llevar el objeto a un estado que no necesariamente es el inicial
- No debe devolver ningún valor

Método __wakeup (cont.)

```php
<?php

 class Automovil{
   public function __wakeup(){
     echo 'Ya me desperte!';
   }
 }

 $a = new Automovil;
 $str = serialize($a);

 $a = unserialize($str);
```

MetodoWakeup.php

Método __toString

- Es llamado cuando un objeto es utilizado en un contexto donde representa un string
- El valor devuelto por este método es el que se usará en lugar del objeto:
  - Echo $obj
  - “Hola! “. $obj
  - $obj .= ‘ ahora $obj es un string’;

Método __toString (cont.)

```php
<?php

  class Automovil{
    public $marca;
    public $modelo;

    public function __toString(){
      return "Soy un carro modelo
        '$this->modelo' marca '$this->marca'.";
    }
  }

  $c = new Automovil;
  $c->marca = 'Toyota';
  $c->modelo = 'Corolla';

  echo $c;
```

MetodoToString.php

Método __invoke

- Es llamado cuando un objeto es utilizado de función
- Todos los parámetros que se le pasen al objeto como función se le pasarán también a la función invoke.
- Lo que devuelva la función invoke es lo que devolverá la llamada al objeto como función

Método __invoke (cont.)

```php
<?php

  class Clase{

    public function __invoke(){
      $param = func_get_args();
      echo '<pre>' .
           print_r($param, true).
           '</pre>';
    }
  }

  $a = new Clase();
  $a('Llamado', 'al', 'metodo', 'invoke');
```

Método __invoke (cont.)

- En caso de contener el método '__invoke', la siguiente instrucción devuelve 'true'

```
is_callable($obj);
```

var_export

- Dada una variable que tenemos en memoria, var_export devuelve el código necesario para poder generar esa variable.

```php
<?php

 echo var_export(array(
   'clave' => 'valor'
 ));
```

- Genera el código:

```
array ( 'clave' => 'valor',)
```

Método __set_state

- En caso de que se haga var_export a un objeto, este es el método que se va a escribir en el código
- __set_state recibirá un arreglo que tiene de clave el nombre de la propiedad y de valor el valor de la propiedad
- __set_state es un método estático que debe inclusive crear el objeto

Método __set_state (cont.)

```php
<?php

  class Ejemplo{
    public $public = 'valor1';
    protected $protected = 'valor2';
    private $private = 'valor3';

    public static function __set_state($param){
      $obj = new self;
      foreach($param as $nombre => $valor){
        $obj->$nombre = $valor;
      }
      return $obj;
    }
  }
  $obj = new Ejemplo;
  echo  '$obj = ' . var_export($obj);
```

MetodoSetState.php

Método __set_state (cont.)

- Al ejecutar el código generado por var_export, se obtendrá un $obj equivalente al que se usó para generar el código
- Es responsabilidad de __set_state lograr este estado.

```
$obj = Ejemplo::__set_state(array(
   'public' => 'valor1',
   'protected' => 'valor2',
   'private' => 'valor3',
))
```

Objetos y referencias

- Cuando a una variable se le asigna un objeto, se le está asignando una referencia
- Cuando una variable se copia el objeto de otra variable usando el operador igual '=', en realidad se está copiando la referencia
- Modificar cualquier referencia al objeto es modificar el objeto directamente

Objetos y referencias (ejemplo)

```php
<?php

  class Automovil{
    public $modelo;
  }
  $a = new Automovil;
  $a->modelo = 'Toyota';
  $b = $a;
  $b->modelo = 'Volkswagen';
  echo $a->modelo;
```

```
Volkswagen
```

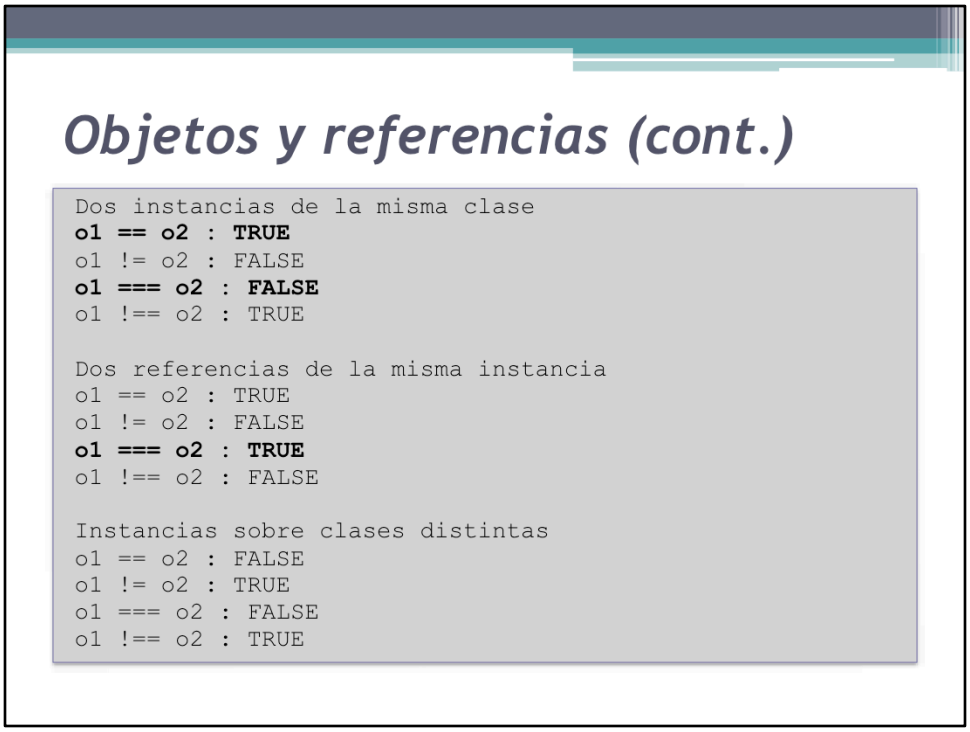
Funciones que usaremos

```php
<?php
  function bool2str($bool)
  {
      if ($bool === false) {
          return 'FALSE';
      } else {
          return 'TRUE';
      }
  }

  function comparar($o1, $o2)
  {
      echo 'o1 == o2 : ' . bool2str($o1 == $o2) . "\n";
      echo 'o1 != o2 : ' . bool2str($o1 != $o2) . "\n";
      echo 'o1 === o2 : ' . bool2str($o1 === $o2) . "\n";
      echo 'o1 !== o2 : ' . bool2str($o1 !== $o2) . "\n";
  }
```

Objetos y referencias (cont.)

```php
<?php

  class Ejemplo
  {
  }

  class OtroEjemplo
  {
  }

  $o = new Ejemplo();
  $p = new Ejemplo();
  $q = $o;
  $r = new OtroEjemplo();

  echo "Dos instancias de la misma clase\n";
  comparar($o, $p);

  echo "\nDos referencias de la misma instancia\n";
  comparar($o, $q);

  echo "\nInstancias sobre clases distintas\n";
  comparar($o, $r);
```

TiposReferencias.php


Objetos y referencias (cont.)
Dos instancias de la misma clase
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : FALSE
o1 !== o2 : TRUE
Dos referencias de la misma instancia
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE
Instancias sobre clases distintas
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE

Clonación de objetos

- Hay veces en las que nos interesa “copiar” un objeto. No queremos que dos variables hagan referencia al mismo objeto.

```
$objetoCopiado = clone $objeto;
```

- La palabra clave ‘clone’ nos permite hacer esto.
- Una copia de una variable usando ‘clone’ nos permite tener dos instancias distintas con las mismas propiedades.

Clonación de objetos (cont.)

```php
<?php

  class Ejemplo
  {
  }

  $o = new Ejemplo();
  $p = clone $o;

  echo "Instancia clonada\n";
  comparar($o, $p);
```

```
Instancia clonada
o1 == o2 : TRUE //Puede ser falso (veremos luego)
o1 != o2 : FALSE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

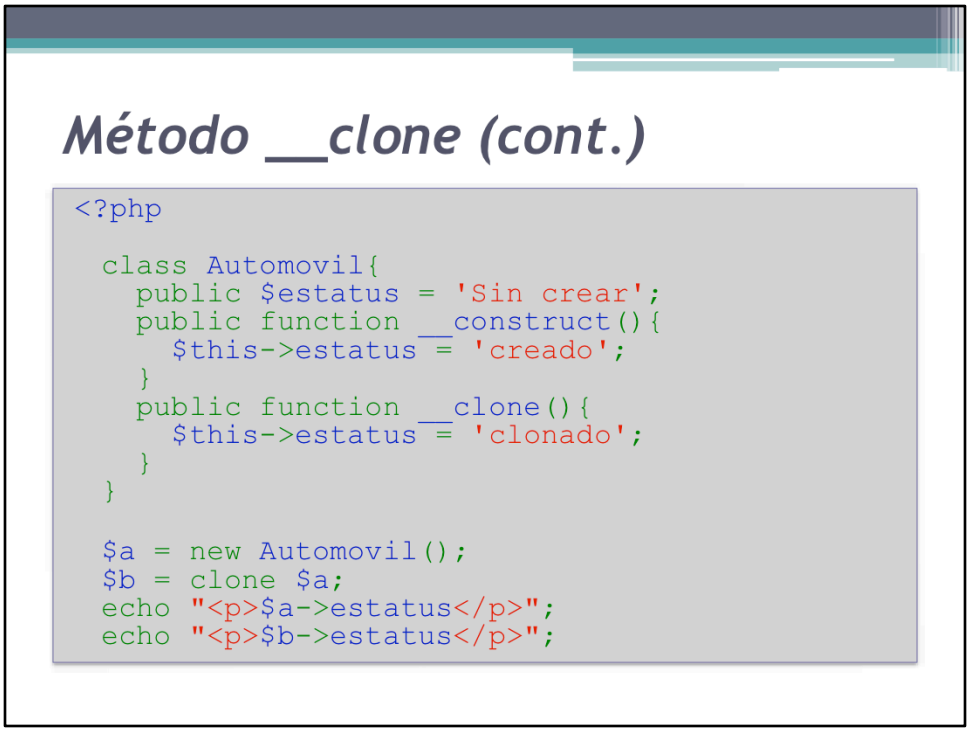
ClonarObjetosReferencias.php

Clonación de objetos (cont.)

- Esta clonación copia valores y referencias
- Al copiar un valor, se tiene el mismo valor copiado en los dos objetos
- Al copiar una referencia, se tiene la misma referencia copiada en los dos objetos:
  - Esto significa que si el objeto original tenía una propiedad que apuntaba al objeto B, el objeto clonado va a a puntar al objeto B también.

Método __clone

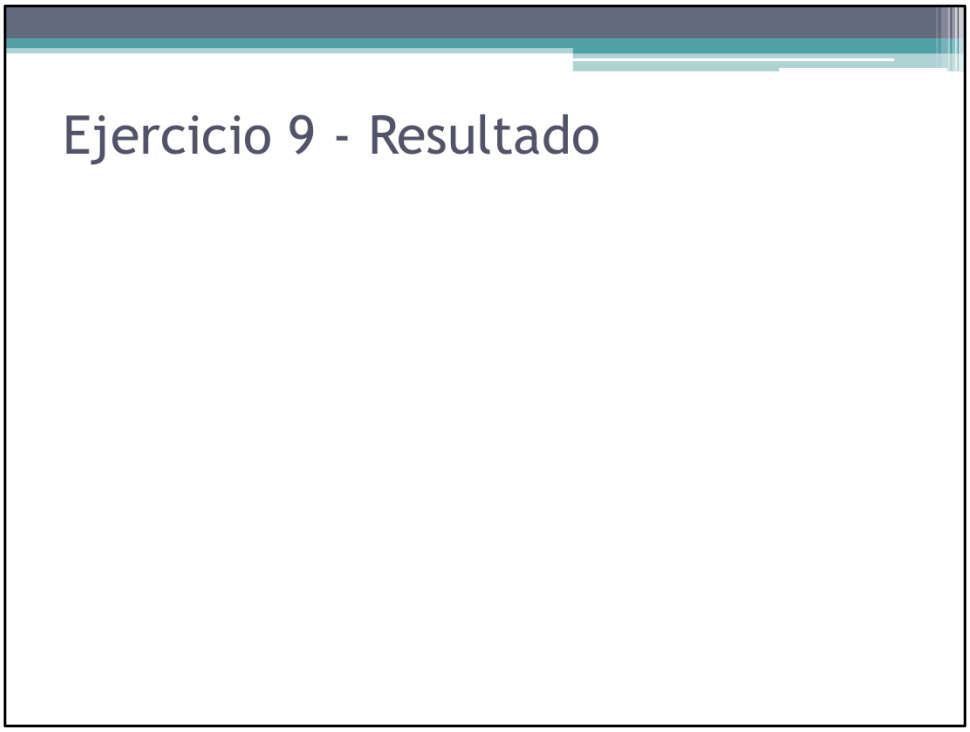
- Este métodos es llamado automáticamente cuando se clona un objeto.
- PHP copia todas las propiedades del objeto original al nuevo y luego ejecuta __clone (de existir)
- __clone no devuelve nada
- Para prohibir la clonación de objetos, __clone debe lanzar una excepción



MetodoClone.php

Ejercicio 9 – Método __clone

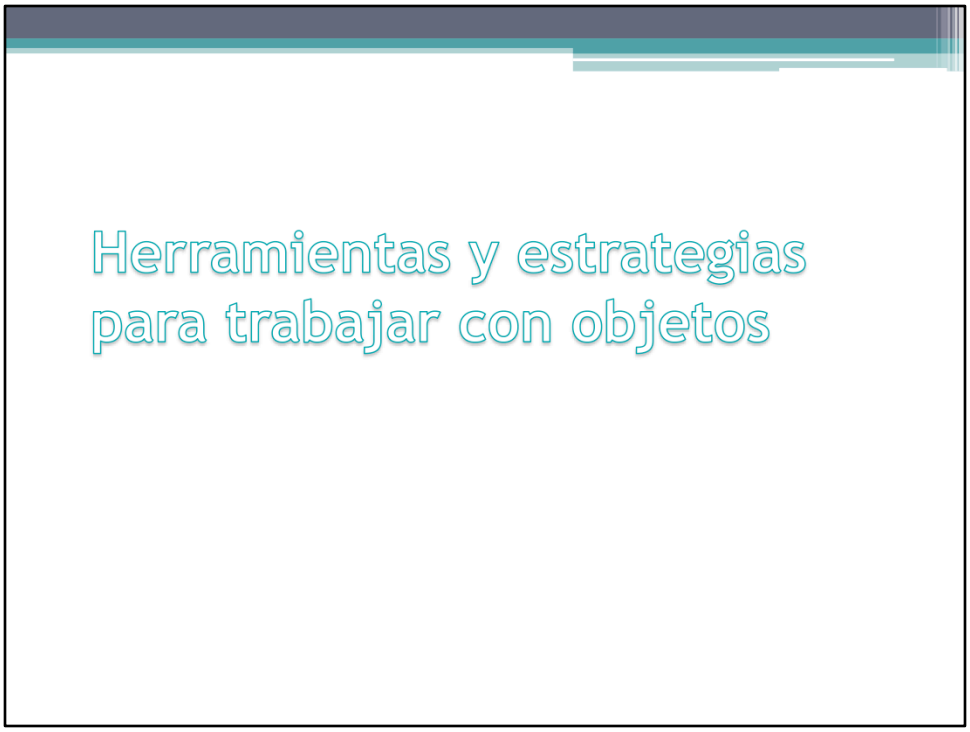
- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse el método __clone
- Especificaciones:
  - Agregar una propiedad estatica a MiClase llamada $instancias con el valor por defecto 0.
  - Agregar el constructor y hacer que cada vez que sea invocado aumente en 1 el valor de $instancias.
  - Agregar el método __clone que aumente en 1 el valor de $instancias.
  - Clonar el objeto que ya se tenga
  - Imprimir la propiedad estática $instancias



Resultado del ejercicio.

Ejercicio 9 - Solución

- Ver Ejercicio9.php



Separando el código

- Si un código es muy grande, es posible que sea más cómodo separarlo en dos archivos
- Para ejecutar el código de dos archivos en un mismo programa, se usan las instrucciones 'include' y 'require'
- 'include' y 'require' tienen el mismo efecto, sólo que require detiene la ejecución del programa si falla

Instrucción require

```
<?php
 $nombre = 'Pepito';

 require 'printer.php';
```

Archivo printer.php:

```
<?php

 echo "<p>Hola $nombre!</p>";
```

MetodoClone.php

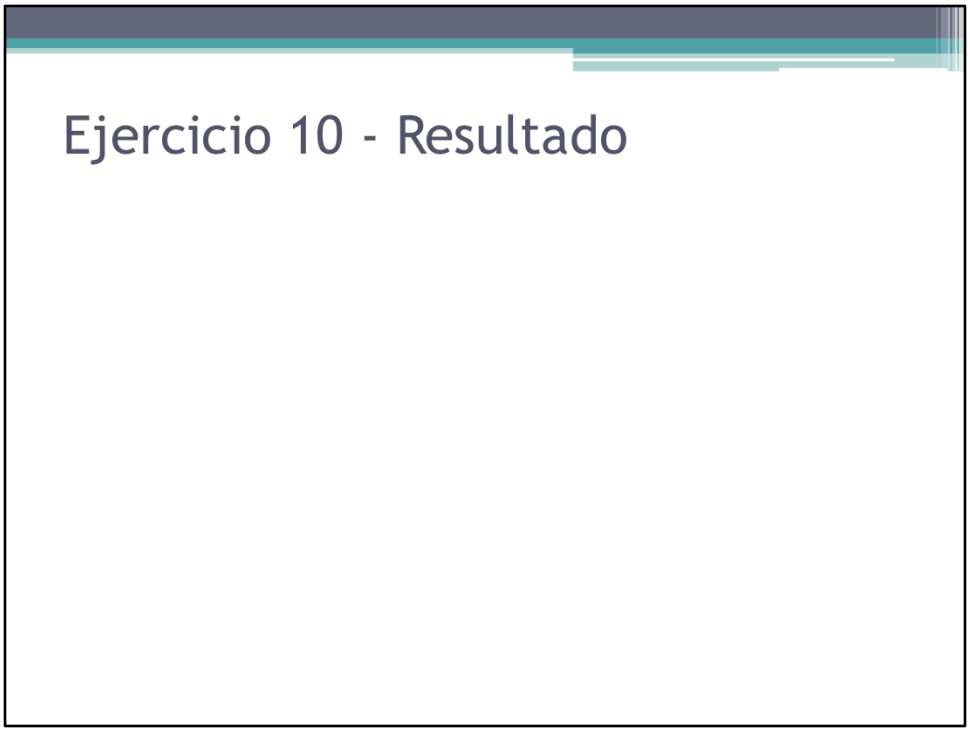
Código equivalente

```php
<?php
 $nombre = 'Pepito';

 echo "<p>Hola $nombre!</p>";
```

- El código anterior es equivalente a este en cuanto a funcionalidad
- Los include/require permiten picar archivos en varios pedazos

Ejercicio 10 – require

- Duración: de 5 a 10 minutos
- Objetivos:
  - Familiarizarse con la inclusión de archivos
- Especificaciones:
  - Mover la clase MiClase a un archivo llamado MiClase.php
  - Incluir MiClase.php en el archivo Ejercicio10.php al comienzo de la ejecución



Resultado del ejercicio.

Ejercicio 10 - Solución

```php
<?php

 require 'MiClase.php';

 $o = new MiClase();
 echo "<p>$o->mensaje</p>";

 $o->mes = 'Mayo';
 echo "<p>{$o->data['mes']}</p>";

 $b = clone $o;

 echo '<p>Cantidad de instancias: ' . M
 iClase::$instancias . '</p>';
```

Instrucción require_once

```
<?php
  $nombre = 'Pepito';

  require_once 'printer.php';
  require_once 'printer.php';
```

- Sólo imprime una vez “Hola Pepito!” ya que la instrucción require_once se asegura de no incluir el mismo archivo dos veces
- La instrucción include_once también está disponible

MetodoClone.php

Rutas

- El directorio punto (.) es el directorio donde se está ejecutando el script. Así si un script está en el mismo directorio que otro, puede incluirlo sin necesidad de especificar más nada
- Una dirección absoluta es una dirección que comienza desde la raíz del sistema de archivos:

```
/usr/local/lib/printer.php (LINUX)
```

```
C:\User\Pepito\www\printer.php (WINDOWS)
```

Rutas (cont.)

- Una ruta relativa, es una ruta a la que le falta especificar desde donde se lee:

```
lib/printer.php
```

- Se lee desde

```
/usr/local/
```

- o:

```
C:\User\Pepito\www\lib\printer.php
```

Rutas (cont.)

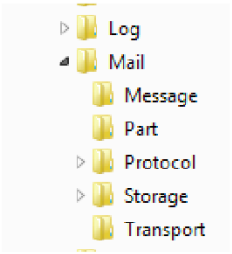
- La diferencia entre una ruta relativa y una absoluta radica en que la relativa comienza con slash (/) si estamos en linux o en la unidad que esté el archivo (C:\) si estamos en windows.

Paquetes

- Un paquete es un conjunto de clases que están relacionadas entre sí
- Los paquetes forman librerías
- Una vez que se forman librerías grandes de clases, es importante mantener el orden.

Paquetes (buenas prácticas)

- Cada clase debe ir en un archivo
- Todas las clases que estén relacionadas deben ir en el mismo directorio
- Un buen paquete es un directorio que puede o no depender de otros paquetes



Paquetes (buenas prácticas)

- Para seleccionar el nombre de una clase en función de su ubicación, existe una convención:

Archivo

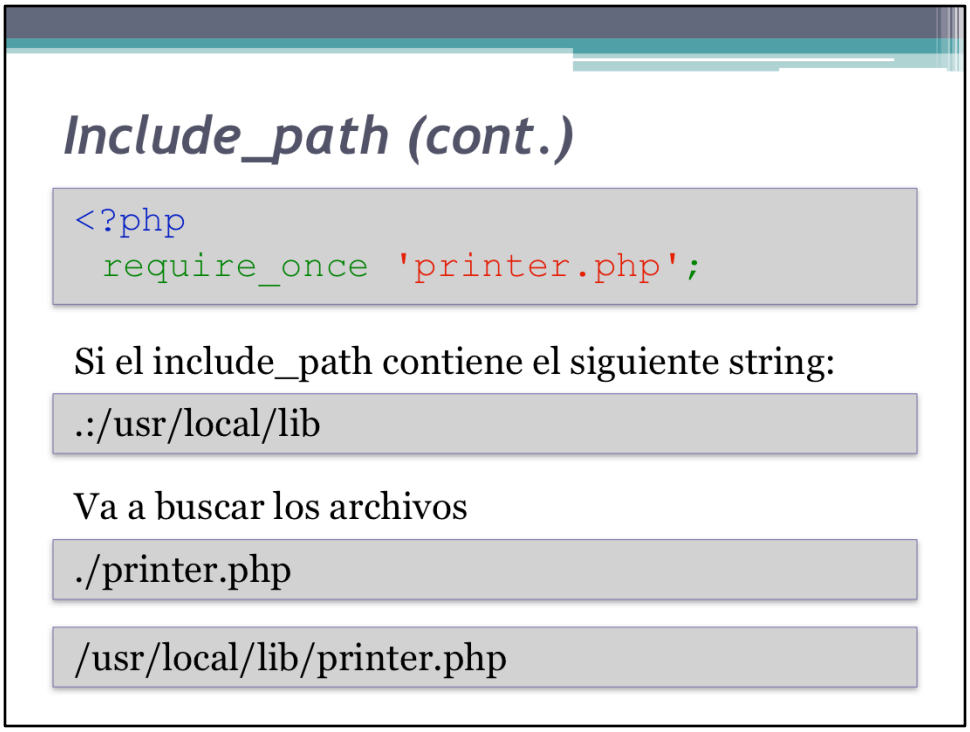
Filtro/Numero.php

class Filtro_Numero { ... }

- La ruta desde que comienza la librería hasta el archivo se usa como nombre.

Include_path

- Cuando se van a incluir archivos, el servidor tiene que saber dónde buscarlos.
- Para buscar los archivos, el servidor utiliza una directiva de configuración llamada include_path
- El include_path es un conjunto de directorios separados por ':' (linux) o ';' (windows)



MetodoClone.php

Include_path (cont.)

- El primer directorio en el que consiga el archivo que está buscando para la búsqueda e incluye al archivo
- Si PHP recorre todos los directorios especificados en el include_path y no consigue el archivo especificado, falla.
- En caso de falla, el include/include_once sólo da un warning
- En caso de falla, el require/require_once detiene la ejecución del programa

Paquetes e Include_path

- Para poder trabajar un sistema de archivos y con paquetes, se recomienda usar el include_path
- Podemos agregar tantos directorios como hagan falta al include_path

```
.:/usr/local/lib:/home/usuario/www/lib
```

Paquetes e Include_path

- En nuestro ejemplo del filtro

```
.:/usr/local/lib:/home/usuario/www/lib
```

- Al colocar en el include_path la dirección de la librería, PHP tratará de buscar el paquete en esa ruta:

```
/home/usuario/www/lib/Filtro/Numero.php
```