Lenguajes Interpretados en el Servidor

Objetos en PHP
(Primera parte)

Objetivos

- Dar a conocer los fundamentos de la Programación Orientada a Objetos.
- Tener claridad en cuanto al objetivo de utilizar el enfoque orientado a objetos de PHP.
- Identificar las diferencias fundamentales entre la Programación Orientada a Objetos y la programación funcional.
- Utilizar conceptos como abstracción de clases, polimorfismo e interfaces para resolver problemas prácticos.
- Programar utilizando clases y objetos con PHP.

Contenido

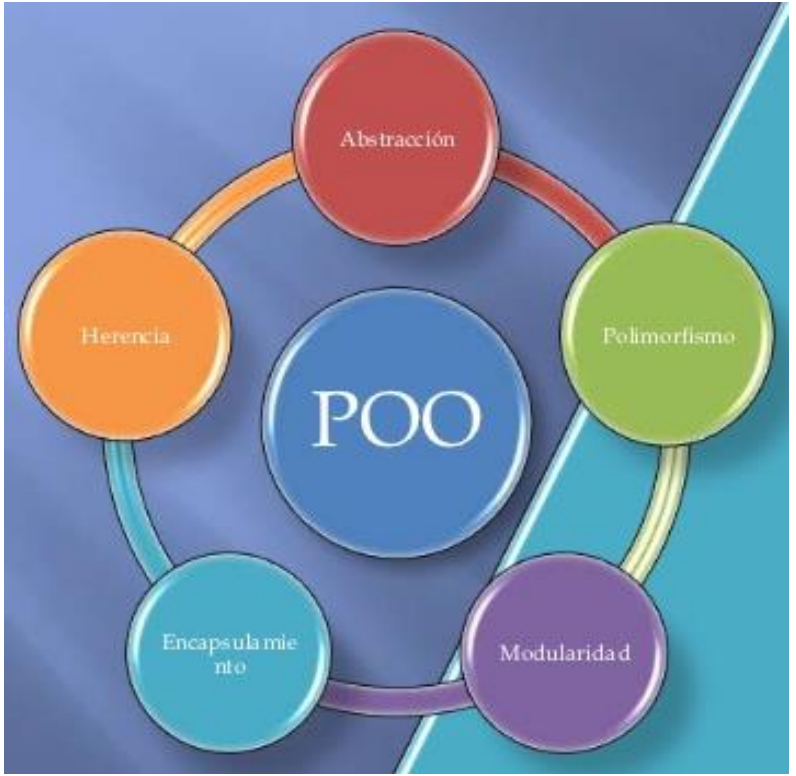
1. Introducción a la Programación Orientada a Objetos.
2. Fundamentos de la Programación Orientada a Objetos
3. Clases y objetos
4. Constructores y destructores
5. Diferencias en la creación de clases con PHP4 y PHP5
6. Control o nivel de acceso a los miembros de una clase
7. Definición de constantes en los objetos
8. Propiedades y métodos estáticos
9. Clonación de objetos
10. Autocarga de objetos

Contenido

11. Sobrecarga de propiedades y métodos.
12. Herencia.
13. Clases y métodos finales.
14. Clases y métodos abstractos.
15. Polimorfismo.
16. Interfaces.
17. Manejo de excepciones.

Introducción a la POO

- Hasta este punto en el curso, hemos desarrollado secuencias de comando en PHP con instrucciones secuenciales que utilizan estructuras de control y en las últimas clases con funciones que permiten desarrollar pequeños módulos de programa que pueden ser invocados desde el flujo principal del programa o desde otras funciones.



Introducción a la POO

- Este enfoque de programación es conocido como programación estructurada o, de acuerdo a su evolución histórica, como programación modular, en donde un programa demasiado complejo es dividido en módulos o subprogramas más legibles o manejables.
- La Programación Orientada a Objetos es un paradigma de programación que viene a sustituir el enfoque de la programación modular ofreciendo muchas más ventajas.
- Este nuevo paradigma de programación se basa en conceptos como herencia, abstracción, polimorfismo, encapsulamiento, entre otros.

Introducción a la POO

- PHP inició como un lenguaje para construir páginas web personales sencillas, en ese sentido se le conoció como un simple lenguaje de scripts.
- Desde la versión 3 y 4, pasó a ser un lenguaje serio para desarrollo web. De hecho, a partir de la versión 5.0 dispone de un soporte completo para el desarrollo orientado a objetos, gracias a la potencia del nuevo engine Zend 2.0.

Fundamentos de la POO (I)

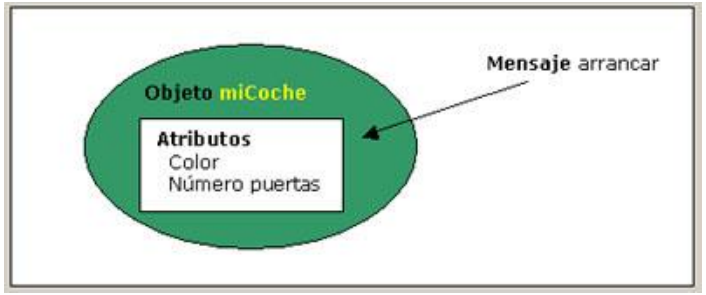
- Se considera a la Programación Orientada a Objetos (POO) como un método de desarrollo de software en el que se identifican las características y los comportamientos de los elementos, reales o abstractos, son modelados haciendo uso de clases y objetos.
- En la POO la creación de programas se basa en la definición de clases que constituyen modelos del mundo real y a partir de las cuales se crean objetos.

Fundamentos de la POO (II)

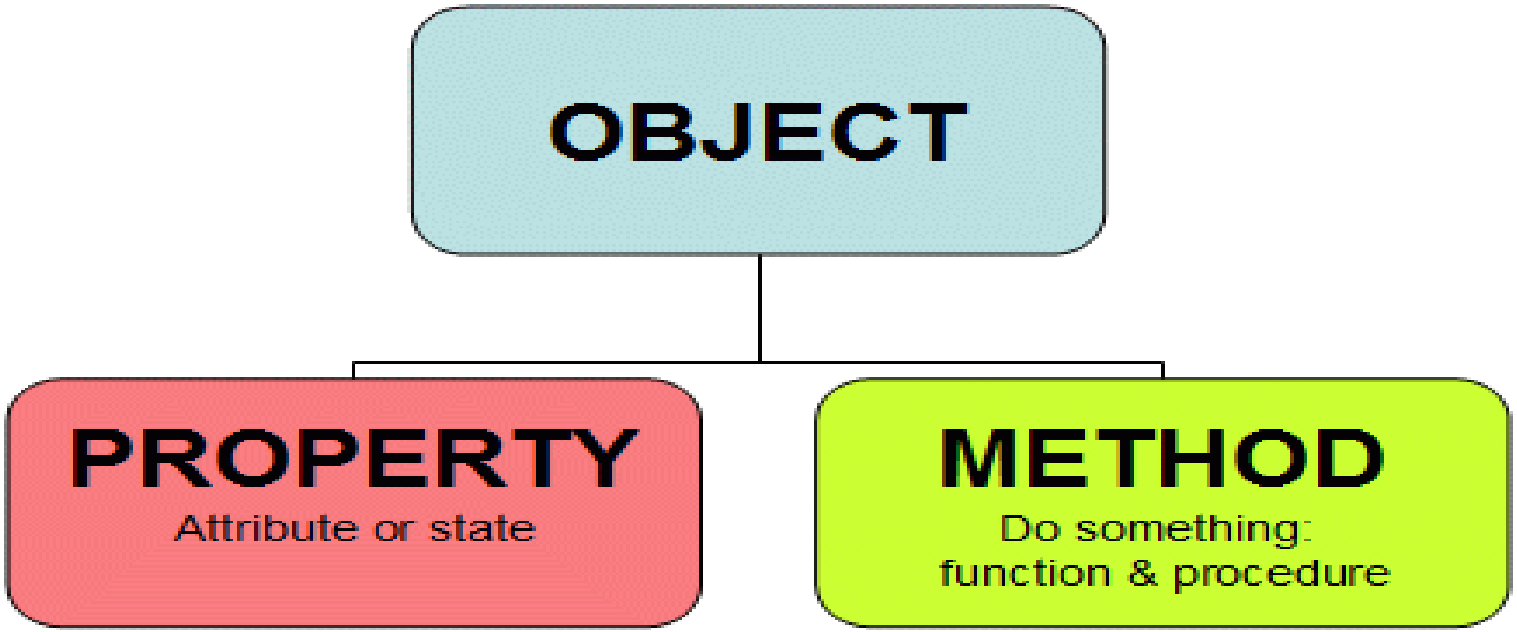
- Una **clase** es una **descripción genérica**, también llamada **modelo** o **plantilla.** Esta clase es tomada de un objeto del mundo real y modelada a través de **propiedades** y **métodos.**
- Un **objeto** es una instancia, muestra o ejemplar de la clase que posee dos características importantes que son: el **estado** y el **comportamiento.**
- La **relación** que existe **entre clase y objeto**, viene a ser la misma que hay entre el diseño de un automóvil (esta sería la clase o modelo) y el automóvil concreto construido en base a ese modelo, ya con un color específico, un número de puertas concreto, un tamaño de motor, etc.

Fundamentos de la POO (III)

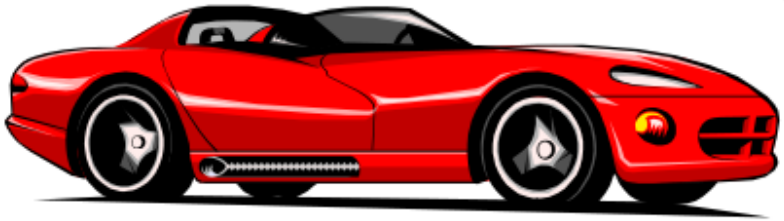
- El **estado** de un objeto se define mediante un conjunto de **propiedades** o **atributos** del objeto en si, que son las **características** que definen al objeto en cuestión.
- El **comportamiento** se indica mediante **métodos**, que son las **funciones**, **operaciones** o **acciones** que se pueden realizar con las propiedades o atributos del objeto, ya sea para modificar su comportamiento o para obtener algún efecto externo.



Fundamentos de la POO (IV)



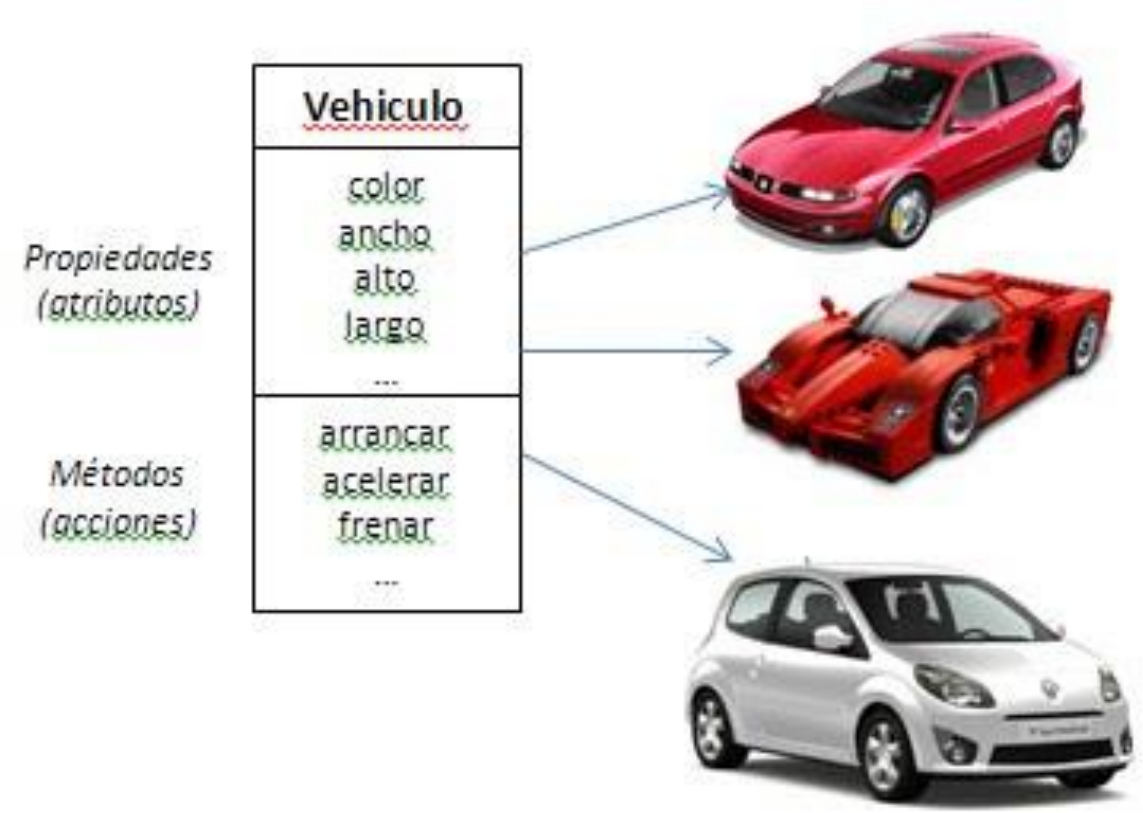
Fundamentos de la POO (V)



- Atributos:
  - color
  - velocidad
  - ruedas
  - motor

- Métodos:
  - arranca()
  - frena()
  - dobla()

Fundamentos de la POO (VI)



Fundamentos de la POO (VI)

- En conclusión, cuando se desarrollan aplicaciones orientadas a objetos, se crearán objetos a partir de cualquier elemento o concepto, pudiendo ser estos **objetos físicos** como una mesa, un carro, una cuenta bancaria o un cliente, o un **objeto conceptual** que solo existe a nivel de software, como un archivo, un formulario o página web.
- Los **objetos encapsulan** tanto **propiedades** como **métodos**. Las propiedades se utilizan para almacenar información de estado del objeto, en tanto que los métodos se ocupan de hacer modificaciones sobre dicho estado, definiendo así el comportamiento del mismo.

Fundamentos de la POO (VIII)

- El software orientado a objetos se diseña y construye como un conjunto de objetos independientes con **atributos** o **propiedades** y **operaciones** o **métodos** que interactúan para realizar las tareas que sean requeridas.
- La ventaja principal del software orientado a objetos es su capacidad de fomentar la encapsulación, también conocida como ocultación de datos. Esto significa que el acceso a las **propiedades** de un objeto, solamente es posible hacerlo a través de sus **métodos**.

Ilustración de clase y objeto (I)

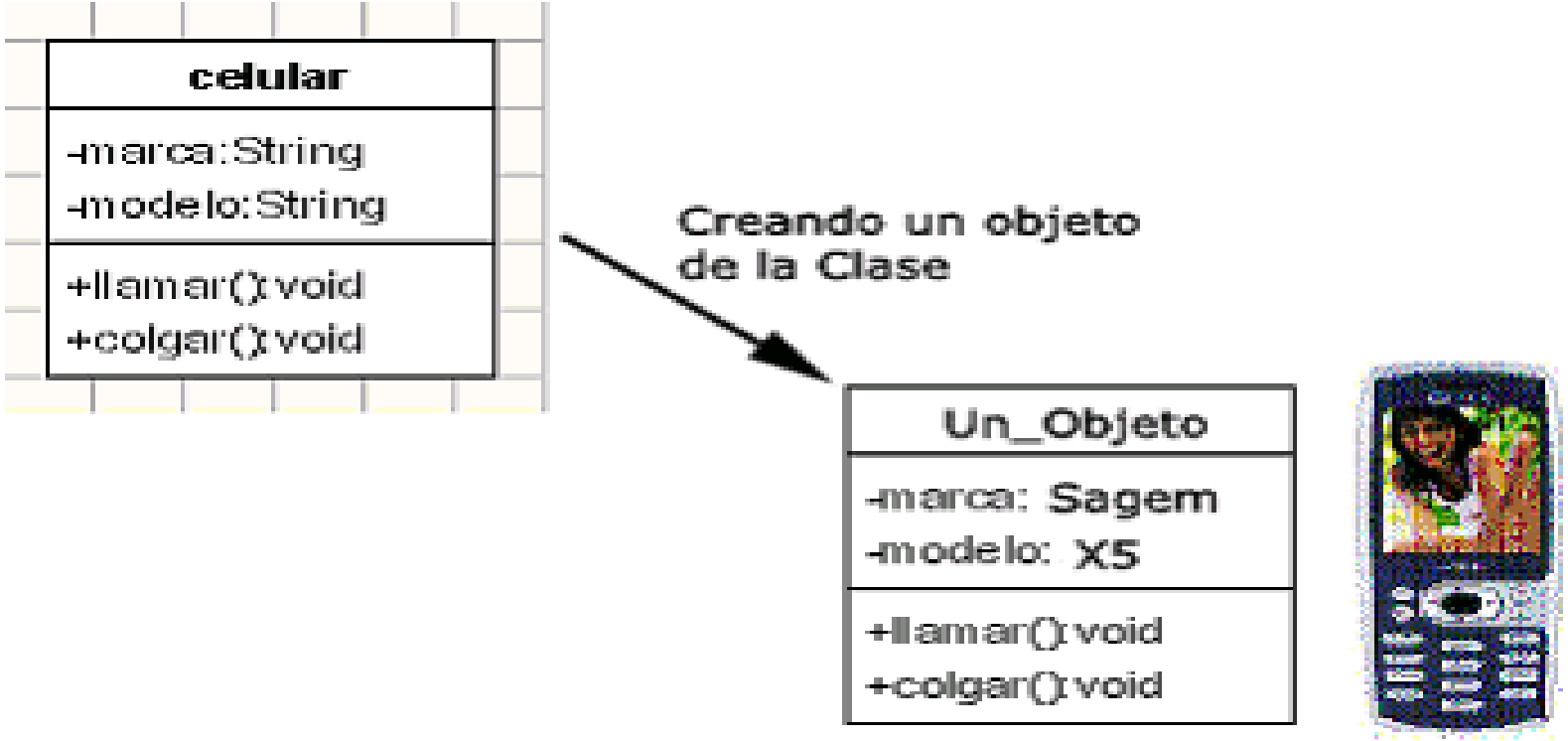


Ilustración de clase y objeto (II)

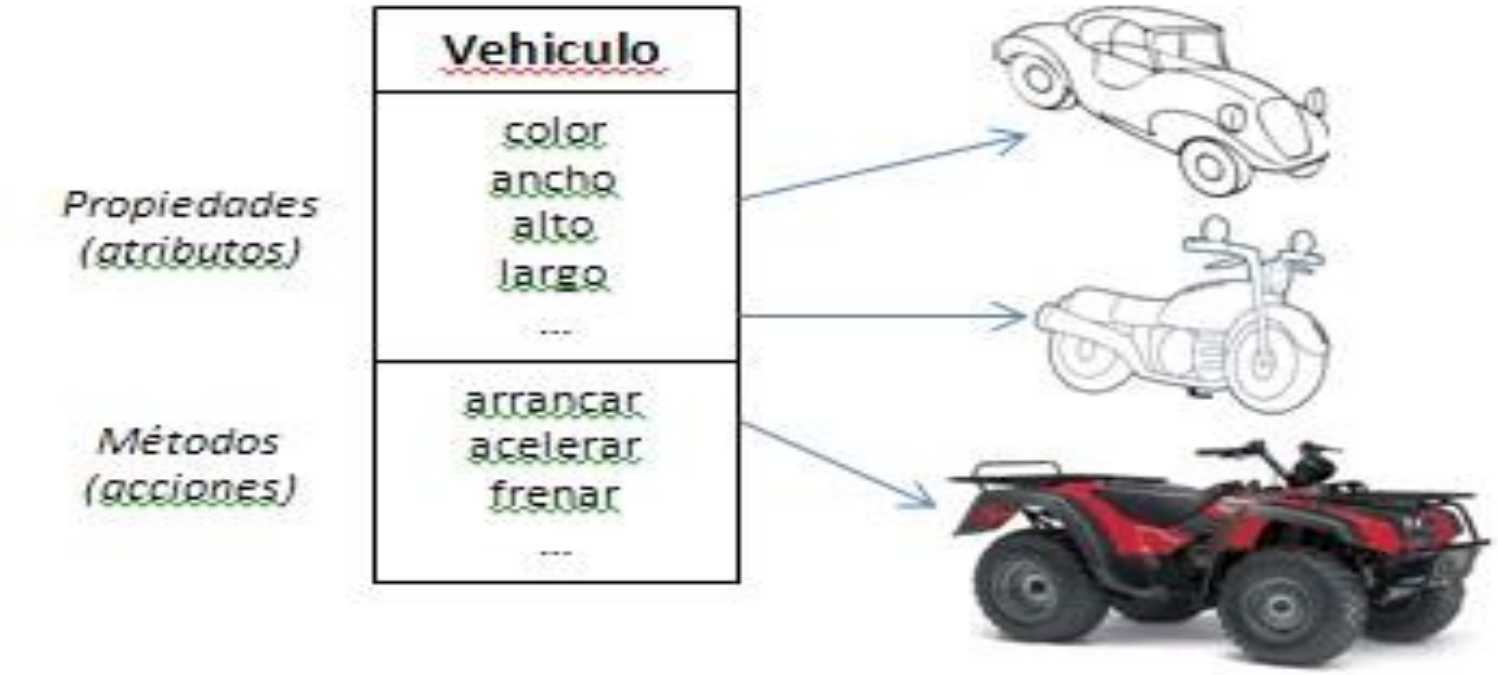


Ilustración de clase y objeto (III)

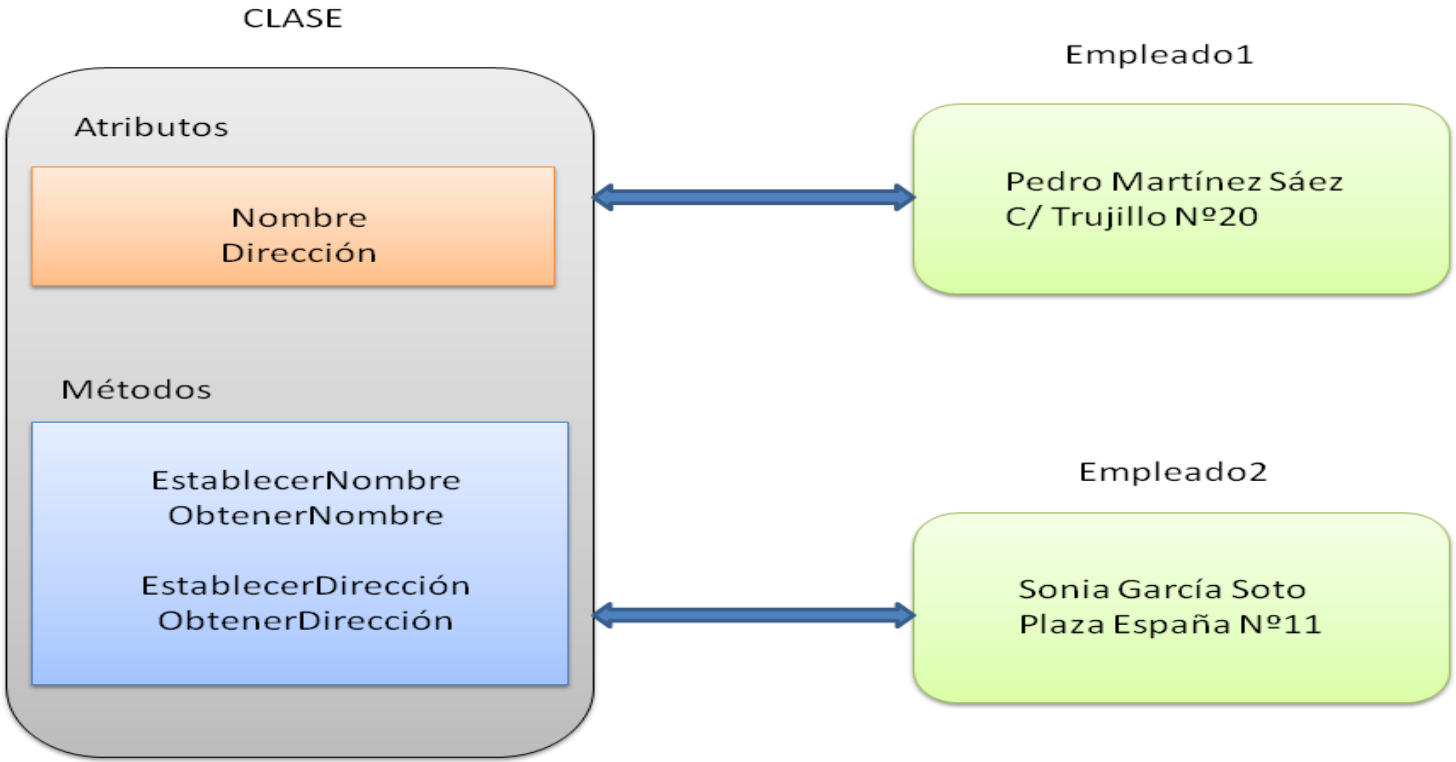
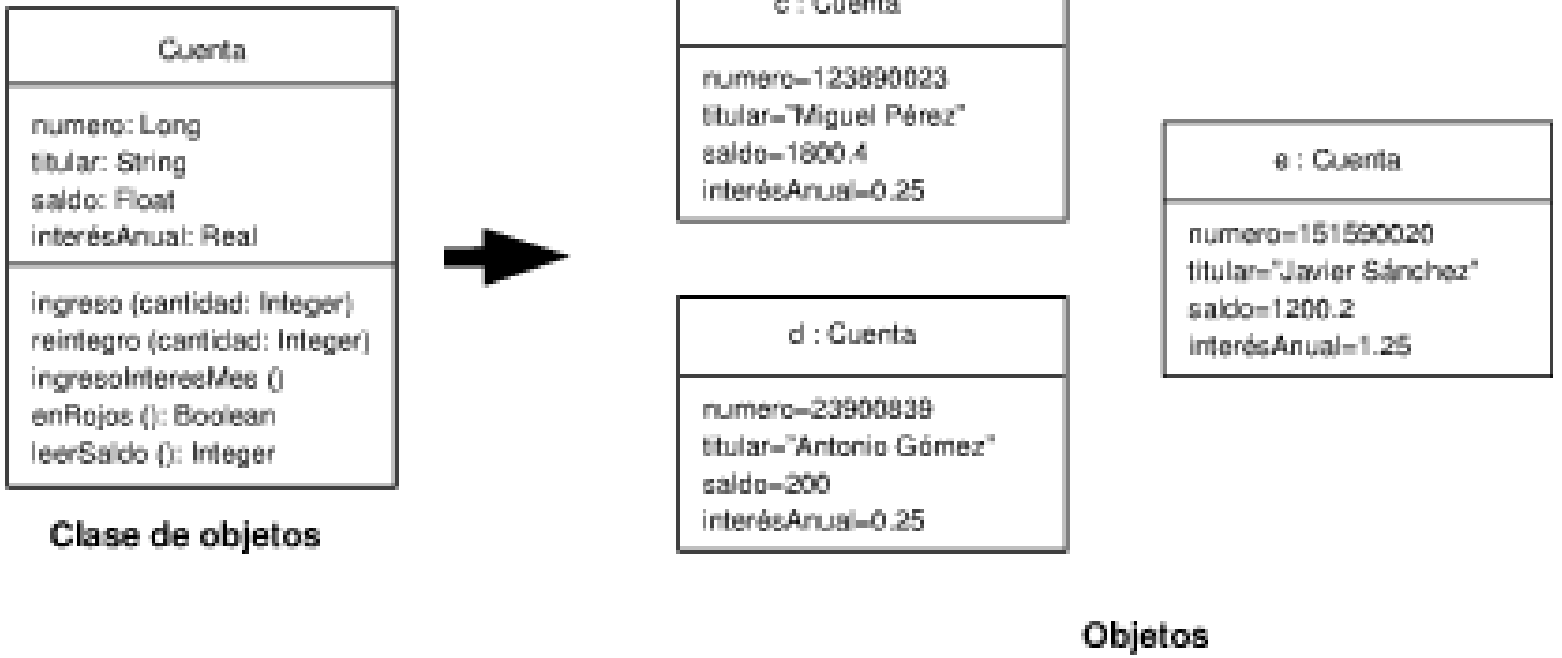


Ilustración de clase y objeto (IV)



Estructura de una clase en PHP (I)

- Para crear una clase en PHP, debe utilizar la palabra clave **`class`** y seguir la siguiente estructura sintáctica general:

```
class nombreClase [extends parentClass]{
    //propiedades;
    []
    //métodos;
}
```

Estructura de una clase en PHP (II)

- Para crear una clase en PHP, debe utilizar la palabra clave **`class`** y seguir la siguiente estructura sintáctica general:

```
class nombreClase [extends parentClass]{
   //propiedades;
   [public|private|protected] $property1;
   …
   //métodos;
   [public|private|protected] function name([$arg1,…]){}
}
```

Estructura de una clase en PHP 4.0 (III)

```
class nombreClase {
  //Propiedades
  var $propiedad1;
  var $propiedad2;
  var $propiedad3;
  …
  var $propiedadN;
  //Métodos
  function metodo1(){//instrucciones1;}
  function metodo2(){//instrucciones2;}
  function metodo3(){//instrucciones3;}
  …
  function metodoN(){//instruccionesN;}
}
```

Estructura de una clase en PHP 5.0 (IV)

```
class nombreClase {
   //Propiedades
   private|public|protected $propiedad1;
   private|public|protected $propiedad2;
   private|public|protected $propiedad3;
   …
   private|public|protected $propiedadN;
   //Métodos
   function metodo1([arg1[,arg2,…]]){//instrucciones1;}
   function metodo2([arg1[,arg2,…]]){//instrucciones2;}
   function metodo3([arg1[,arg2,…]]){//instrucciones3;}
   …
   function metodoN([arg1[,arg2,…]]){//instruccionesN;}
}
```

Clases y objetos en PHP (I)

- Una clase básica en PHP se puede representar utilizando un esquema de bloque como el siguiente:

Clase persona	
Propiedades	
Nombre	$nombre
Apellido	$apellido
Edad	$edad
Métodos	
Poner nombre	asignar_nombre($nombre, $apellido)
Dar nombre completo	dar_nombre_completo()

Clases y objetos en PHP (II)

- Implementando en código PHP la clase anterior:

```
class persona    {
   //Propiedades
   private $nombre;
   private $apellido;
   private $edad;
   //Métodos
   function asignar_nombre($nombre,$apellido){
     $this->nombre = $nombre;
     $this->apellido = $apellido;
   }
   function dar_nombre_completo(){
      return $this->nombre . "" . $this->apellido;
   }
}
```

Creación de objetos

- La creación de ejemplares de la clase es a lo que se le denomina instanciar a un objeto de la clase:

```
$persona1 = new persona(); //crea una instancia
$persona1 -> asignar_nombre("Raul","López");
$persona1 -> dar_nombre_completo();
```

- La primera instrucción crea una instancia de la clase persona; es decir, un objeto concreto o una persona concreta. Se utiliza la palabra reservada ***new*** para crear dicha instancia.
- La segunda y la tercera instrucción invocan los métodos del objeto persona. El método **`asignarnombre()`** asigna el nombre enviado a la persona y el método **`darnombre()`** muestra el nombre asignado a esta persona en la salida.
- Es importante comprender que el orden en que se invoquen estos métodos es relevante para obtener el resultado esperado.

Constructores y destructores (I)

- Un **constructor** es un método que se ejecuta automáticamente cuando se crea un ejemplar del objeto, sin necesidad de que sea invocado.
- Esto resulta muy útil si se desea que las propiedades del objeto posean un conjunto de características por defecto o, ejecutar una serie de acciones previas, conocidas como inicialización del objeto.
- Un nuevo objeto se crea en el momento en que se ejecuta la instrucción con la palabra reservada ***new*** seguida del nombre de la clase acompañado de unos paréntesis. Esta era la forma de hacerlo en PHP 4.0.
- Para PHP 5.0 se modificó completamente la implementación de la POO, definiendo como constructor el método especial **`__construct()`.**

Constructores y destructores

- Los constructores de PHP 5.0 o superior se declaran utilizando el método especial `__construct()`.
- Este es uno de los cambios en la versión 5.0 con respecto a la versión 4.0 de PHP. En la versión 4.0 los nombres de los constructores tenían el mismo nombre de la clase.
- Por motivos de compatibilidad inversa, si en una clase no se encuentra una función con el nombre `__construct()`, PHP buscará una función que tenga el mismo nombre que la clase.

Ejemplo de constructor

- En el ejemplo de la clase persona, se crea un nuevo objeto cuando se ejecuta la siguiente instrucción:
  ```
  $juan = new persona();
  ```
- Se ejecuta un constructor por defecto o un constructor definido por el programador dentro de la clase.
- En PHP 4 los constructores se crean mediante una función con el mismo nombre de la clase. PHP 4 no dispone de destructores.
- En PHP 5 se incorpora una función constructora unificada denominada ***__construct()***, que puede o no llevar parámetros. Los constructores no devuelven valores.

Destructor

- PHP 5 incorpora el destructor, algo que no existía en las versiones de la generación 4. La utilidad de un destructor en PHP es limitada, ya que PHP siempre libera los recursos utilizados al finalizar la ejecución del *script*.
- Un apunte importante en cuanto al constructor y destructor, es que cuando se implementa un constructor o destructor en una clase hija, el constructor de la clase padre ya no puede ser invocado de forma directa e inmediata.

Destructor

- Para llamar al constructor de una clase padre, si se ha redefinido un constructor en la clase hija, debe emplear una instrucción como la siguiente:

```
parent::__construct();
parent::__destruct();
```

- El destructor de un objeto se llama justo antes de eliminar el objeto. Esto puede suceder en dos casos:
  - Porque han desaparecido todas las referencias al objeto por haber utilizado la función unset() en el objeto.
  - Porque el script ha terminado, ya sea de forma natural o porque se ha producido algún tipo de error.

Ejemplo de destructor

```
<?php
class person {
   public function save(){
      echo "Salvando los datos en la base de datos.";
   }
   public function __destruct(){
      $this->save();
   }
}
//Creando un objeto persona
$persona = new person();
unset($persona);
$otrapersona = new person();
die("Hemos finalizado el script de forma forzada");
?>
```

Diferencias entre PHP 4 y PHP 5 en cuanto a creación de objetos (I)

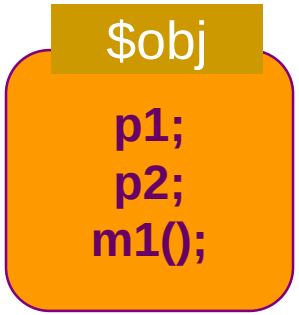
- La primera gran diferencia es que el modelo de objetos en PHP 5 ha sido completamente reescrito lo cual ha permitido mejorar el desempeño e incorporar muchísimas más características.
- En PHP 4 cuando se crean nuevos objetos con la sentencia **new** se devuelve el objeto mismo y no una referencia a este.
- PHP 4 únicamente disponía de constructores, no de destructores. PHP 5 incorpora ambos conceptos.

Diferencias entre PHP 4 y PHP 5 en cuanto a creación de objetos (II)

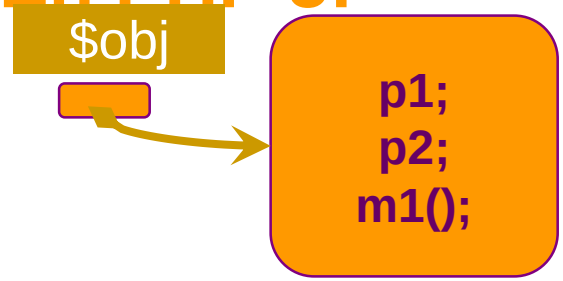
- En PHP 4 los constructores debían tener el mismo nombre que la clase. PHP 5 ha incorporado los métodos unificados ***__construct()***, como constructor, y ***__desctruct()***, como destructor.
- PHP 4 no daba soporte para el control de acceso de las propiedades de un objeto. PHP 5 incorpora las palabras reservadas ***private***, ***protected*** y ***public***, al igual que los lenguajes orientados a objetos más reconocidos, para definir el control de acceso a los miembros de una clase.

Creación de objetos en PHP 4 y PHP 5

- `$obj = new clase();`
- **En PHP 4:**



- **En PHP 5:**



- En una instrucción como:
- $objx = $obj
- En PHP 4 se crea una copia en un objeto totalmente independiente, mientras que en PHP 5 el nuevo objeto es una referencia que apunta, al igual que el objeto original, al objeto en sí.

Niveles de acceso a los miembros de una clase (I)

PHP proporciona tres niveles de acceso para los miembros declarados en una clase:

1. Público (public).
2. Privado (private).
3. Protegido (protected).

ESPECIFICADOR DE ACCESO	ACCESIBLE DESDE SU PROPIA CLASE	ACCESIBLE DESDE CLASE DERIVADA	ACCESIBLE DESDE OBJETOS FUERA DE LA CLASE
public	si	si	si
protected	si	si	no
private	si	no	no

Niveles de acceso a los miembros de una clase (II)

public: significa que el miembro de la clase es público y que, por tanto, se puede acceder a este desde cualquier parte del *script*. Estos miembros pueden llamarse o modificarse internamente dentro del objeto o fuera de este.

private: a estos miembros de clase sólo se puede acceder desde dentro de una instancia de dicha clase utilizando la palabra reservada ***$this***, que hace referencia al objeto mismo, seguido del operador de instancia de objeto -> y, a continuación el nombre de la propiedad.

protected: es similar a **private**, con la diferencia de que además de la instancia de la clase, también puede tener acceso a este miembro cualquier clase hija.

Niveles de acceso a los miembros de una clase (III)

- En PHP 4 no existía el concepto de nivel de acceso y, por defecto, todas las propiedades eran públicas. Para definir las propiedades se hacía uso de la palabra reservada **`var`**.
- Con PHP 5 se incorporaron los niveles de acceso por lo que ahora si se pueden establecer miembros privados (**private**) y protegidos (**protected**), además de miembros públicos (**public**).
- Los que ya tienen mucha experiencia en programación acostumbran a crear clases donde las propiedades son declaradas con ámbito privado, mientras que los métodos se declaran con ámbito público.

Niveles de acceso a los miembros de una clase (IV)

- Ejemplo con propiedades privadas y métodos públicos:

```
class celda {
  private $colorfondo;
  function ponercolorfondo($color){
    if($color=='Orange' || $color=='Red' || $color=='Maroon'){
       $this->colorfondo = $color;
    }
    else{
    die("ERROR: No se permite ".$color." como color de fondo");
    }
  }
}
$fila = new celda();
//$fila->colorfondo = 'fuchsia';  //provocará un error fatal
$fila->ponercolorfondo('orange');
```

Definición de constantes en las clases (I)

- Esta es una nueva característica de PHP 5 que permite definir valores constantes dentro de una clase.
- Para definir una constante dentro de la clase se utiliza la palabra reservada ***const*** seguida por el identificador de la constante a definir y el valor asignado.
- Ejemplo:

```
const PI = 3.1415926535
```

Definición de constantes en las clases (II)

- A este valor constante se puede acceder desde cualquier parte de la clase anteponiendo la palabra reservada ***self*** seguida del operador de resolución de contexto (::) y, a continuación, del identificador de la constante.
- Sin embargo, para acceder a este valor desde fuera del ámbito de la clase debe hacer uso del nombre de la clase junto con el operador de resolución (::), seguido por el identificador de la constante.
- `echo constAreas::PI;`

Propiedades y métodos estáticos (I)

- Una propiedad o método estático pertenece a la clase en la que está definida(o), no a los objetos creados a partir de esa clase.
- Lo anterior significa que estos atributos y métodos pueden utilizarse directamente desde la clase, sin necesidad de crear un objeto. Sin embargo, una propiedad o método estático puede ser llamado desde fuera del contexto de un objeto.
- La declaración de una propiedad o método de este tipo se realiza empleando la palabra reservada ***static*** que debe colocarse justo después de la declaración de visibilidad (***public***, ***private*** o ***protected***) en caso de que esta exista.

Propiedades y métodos estáticos (II)

- El hecho que las propiedades y métodos estáticos pertenezcan a la clase y no a los objetos, hace que para poder acceder a estas sea necesario utilizar el nombre de clase reservado: *self* y, a continuación, el operador de resolución de ámbito "::".
- *self* permite hacer referencia a la clase actual en la que se encuentra. Se utiliza principalmente para acceder a propiedades y métodos estáticos, así como a constantes de la propia clase.

```
class empleado {
   private static $idempleado = 0;
   private $nombre;
   private $apellido;
   public __construct($nombre, $apellido){
      self::$idempleado++;
      $this->nombre = $nombre;
      $this->apellido = $apellido;
   }
}
```

Clonación de objetos (I)

- En PHP 4, cuando un objeto se asigna a una variable, se crea una copia exacta del objeto; es decir, se hace una copia de dicho objeto y los cambios que se hacen sobre las propiedades de este objeto sólo afectan a la copia. Este comportamiento no se ajusta muchas veces a lo que se desea hacer, ni a la forma en que la mayoría de lenguajes orientados a objeto abordan este problema.
- En PHP 5 cuando se asigna un objeto a otro objeto directamente, siempre se estará creando una referencia al objeto original, de modo que siempre que se haga alguna modificación en una propiedad, se estará alterando la propiedad en ambos objetos.

```
<?php
  $unaPersona = new persona();
  $otraPersona = $unaPersona;
?>
```

Clonación de objetos (II)

- En el ejemplo anterior tanto $unaPersona como $otraPersona representan el mismo objeto. De modo que, cualquier modificación realizada en una de las instancias del objeto afectará también a la otra.
- En PHP 5 para crear una copia exacta del objeto en una instancia completamente independiente, debe utilizar la palabra reservada ***clone***. Esta palabra llama al método ***__clone()*** del objeto. Este método no puede ser llamado directamente.

```
$unaPersona = new persona();
$otraPersona = clone $unaPersona;
```

Auto carga de objetos en PHP (I)

- Es habitual que muchos programadores creen un script PHP por cada definición de clase que realicen. Esto implica después, tener que incluir una larga lista de sentencias **`include`** o **`require`** necesarias al inicio del script para poder utilizar las clases definidas.
- PHP 5 evita esta molestia definiendo una función ***__autoload()*** la cual es llamada automáticamente en caso que se intente utilizar una clase que no ha sido definida aún en la secuencia de comandos. Al llamarse a esta función PHP da una última oportunidad al *script* de cargar la clase antes de interrumpirse y enviar un error fatal.

Auto carga de objetos en PHP (II)

- Ejemplo:

```
<?php
  //Definición de la función de autocarga
  function __autoload($class_name){
     require_once $class_name . ".php"; //o ".inc.php"
  }
  //Intento de crear objetos de una clase que
  //aún no se ha definido en el script
  $obj = new MyClass1();
  $objalt =  new MyClass2();
?>
```

- En el ejemplo anterior, el script intentará cargar las clases MyClass1 y MyClass2 desde los archivos MyClass1.php y MyClass2.php, respectivamente, antes de interrumpir con un error fatal el script.

F I N

(1a parte)