Parte III

Constructores

Destructores

Referencias

Propiedades

Constructores

Los constructores son unos métodos que nos sirven para resumir las tareas de inicialización de los objetos, en el instante en el que el objeto es instanciado.

El constructor no construye nada, sólo inicializa el objeto.

Como método que es, su sintaxis es la siguiente:

```
function __construct(){
 //Código que inicializa el objeto creado
}
```

Un ejemplo de constructor podría ser el siguiente, en este caso refiriéndonos a la clase Freelance que ya hemos definido.

```
function __construct($nombre, $precioHora){
 $this->nombre = $nombre;
 $this->precioHora = $precioHora;
 $this->ocupado = false;
}
```

Cada clase sólo puede tener un constructor, pero podría darse el caso de no tener ninguno. Esto es una diferencia con Java, ya que allí nos podíamos encontrar varios constructores en la misma clase, sólo diferenciándose en el número o tipo de parámetros.

Los parámetros no tienen porqué tener el mismo nombre que las propiedades que van a inicializar, pero se suele hacer así por claridad.

Compatibilidad de Constructores

- En PHP5 los constructores se llaman __construct().
- En PHP4 los constructores se llamaban con el mismo nombre de la clase en la que se encontraban.

Por motivos de compatibilidad hacia atrás:

Si en una clase de PHP no se encuentra __construct(), se intentará encontrar un método con el mismo nombre de la clase, el cual se tratará como si fuera el constructor.

Por esa razón, no es conveniente definir ningún método con el nombre de la clase.

A partir de ahora, todas las propiedades se declararán como private o protected, salvo las imprescindibles, y accederemos a ellas sólo desde dentro de la clase. Las podemos inicializar con el constructor.

Ejemplo

```php
<?php

class Persona
{
  private $nombre;
  private $fechaNacimiento;
  private $trabajo;

  public function presentate(){
    echo 'Hola, soy ' . $this->nombre . ' y nací el ' .
          $this->fechaNacimiento . ' y trabajo como ' .
          $this->trabajo;
  }
  function __construct($nombre,$fechaNac,$trabajo){
    $this->nombre = $nombre;
    $this->fechaNacimiento = $fechaNac;
    $this->trabajo = $trabajo;
  }
}

?>
```

Ejemplo cont.

Si ahora quisiéramos usar la clase persona, sólo tendríamos que crear un script nuevo y escribir lo siguiente:

```
<?php

include 'persona.class.php';

$pepe = new Persona('Pepe','24/12/1990','Informático');

$pepe->presentate();

?>
```

Sobrecarga de constructores

Como ya hemos dicho antes, sólo puede haber un constructor por clase.

Recordemos que la sobrecarga de métodos es el hecho de tener varios métodos todos ellos con el mismo nombre, pero diferenciándose en la cantidad de parámetros o en el tipo que tengan.

En PHP no hay sobrecarga "nativa" de métodos.

PHP, al igual que Javascript, son lenguajes poco estrictos, donde se puede invocar una función con cualquier número de parámetros, independientemente de su declaración.

Por tanto, eso impide que pueda haber métodos con el mismo nombre aunque tengan diferente cantidad de parámetros.

Ejemplo

Siguiendo con la clase persona, si el constructor lo definiésemos de esta manera:

```
function __construct($nombre = "Anónimo",
                     $fechaNac = "01/01/1990",
                     $trabajo = "En Paro"){
 $this->nombre = $nombre;
 $this->fechaNacimiento = $fechaNac;
 $this->trabajo = $trabajo;
}
```

Serían válidos los siguientes objetos:

```
$pepe = new Persona('Pepe','24/12/1990','Informático');
$juan = new Persona('Juan','29/02/1992');
$ines = new Persona('Inés');
$nadie = new Persona();
```

Pero no sería válido el objeto:

```
$alguien = new Persona('Yo',,'Informático');
```

Ejemplo: Migas de Pan

Vamos a hacer una clase para implementar "migas de pan"

Home > Noticias > Noticias Académicas

Esta clase se basará en la clase Enlace que debemos haber hecho en los ejercicios de la parte anterior.

```
class Enlace
{
    private $texto;
    private $url;

    public function __construct($texto, $url){
        $this->texto = $texto;
        $this->url = $url;
    }

    public function mostrar(){
        return "<a href=\"$this->url\">$this->texto</a>";
    }
}
```

Ejemplo: Migas de Pan

```
class MigasPan
{
    private $separador;
    private $nodos = [];

    public function __construct($separador = ">"){
        $this->separador = $separador;
    }

    public function agregaNodo($texto, $ruta){
        $this->nodos[] = new Enlace($texto, $ruta);
    }

    public function mostrar(){
        $salida = '';
        foreach ($this->nodos as $indice => $enlace) {
            $salida .= $enlace->mostrar();
            if($indice != count($this->nodos) -1 ){
                $salida .= ' ' . $this->separador . ' ';
            }
        }
        return $salida;
    }
}
```

Ejemplo: Migas de Pan

Ahora, en el script, sólo necesitamos escribir:

```
$migas = new MigasPan();
$migas->agregaNodo("Home", "http://www.iescierva.net");
$migas->agregaNodo("Noticias", "http://www.iescierva.net/
noticias");
$migas->agregaNodo("Noticias Académicas", "http://
www.iescierva.net/noticias/academicas");
echo $migas->mostrar();
```

Destructores

Se llaman en el mismo momento en el que un objeto se queda sin ninguna referencia. Sirven para realizar aquellas tareas necesarias cuando un objeto deja de existir.

Su sintaxis es:

```
function __destruct(){
  //Código…
}
```

Por ejemplo, si un objeto necesita acceder a la base de datos, el constructor abriría la conexión y el destructor la cerraría.

Recordemos que cuando acaba un script todas las variables son eliminadas de la memoria y los objetos son destruidos, por lo que es en ese momento cuando, por defecto, se llama al destructor.

Destructores

Un ejemplo de destructor para la clase Freelance podría ser el siguiente:

```
protected function __destruct(){

 echo "Soy " .$this->nombre
 echo " y dejo de trabajar. Adios!!";

}
```

Destructores y exit()

exit() es una función que nos permite terminar de forma inmediata un script, independientemente de lo que pueda haber a continuación.

Si hacemos un **exit()** desde cualquier parte de PHP, se llamará a los destructores.

Si llamamos a un **exit()** dentro de un destructor, hará que se eviten las diversas finalizaciones de otros objetos. Es decir, al ejecutarse la función de salida dentro de un destructor, impiden que se ejecuten el resto de destructores de los demás objetos. A pesar de eso, al finalizar el script la memoria que ocupan es liberada, pero no se ejecutan las tareas finales que haya en el destructor.

Referencias

Una variable que contiene un objeto en realidad lo que tiene es una **referencia a un objeto**, lo que llamaríamos un apuntador o puntero.

```
$fr1 = new Freelance();
```

Cuando creamos un objeto con **new** lo que estamos haciendo es crear una referencia al objeto. La variable no es en sí el objeto sino un apuntador.

Referencias

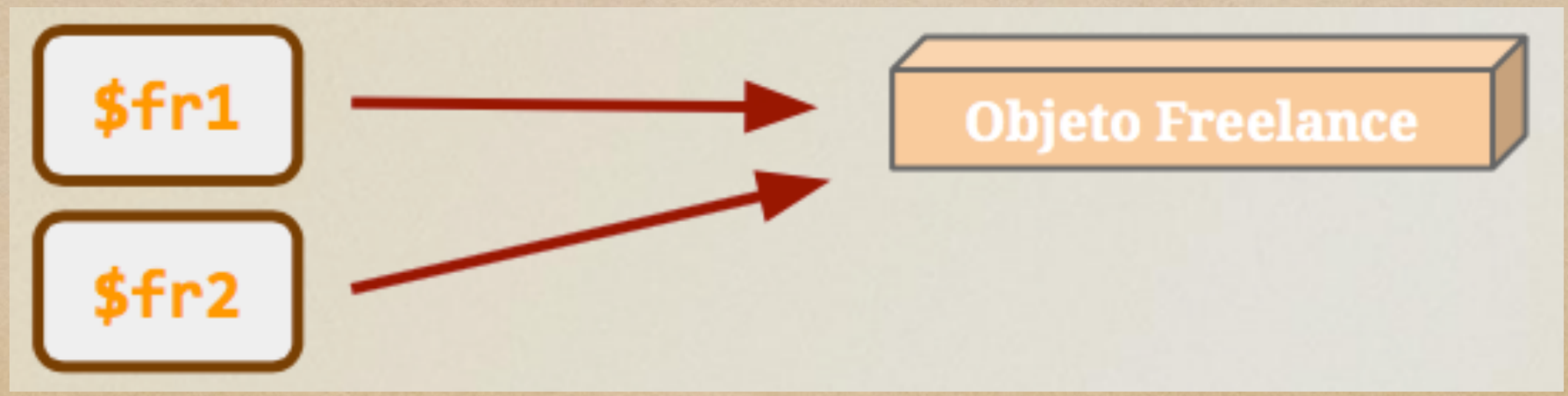
Asignar una variable a una referencia del objeto no clona el objeto. Lo que estamos creando son dos punteros al mismo lugar.

Si declaramos:

```
$fr1 = new Freelance();
```

Y luego hacemos:

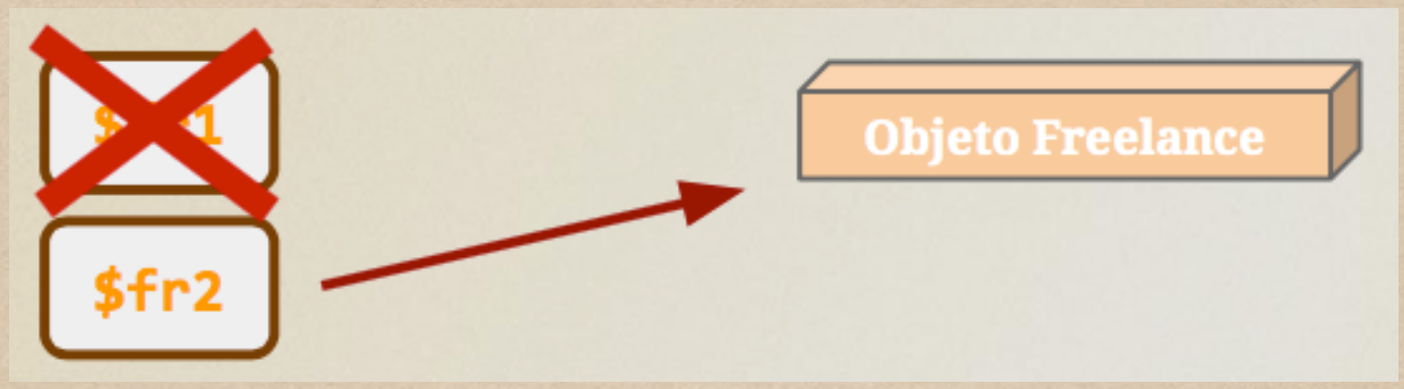
```
$fr2 = $fr1
```



Referencias

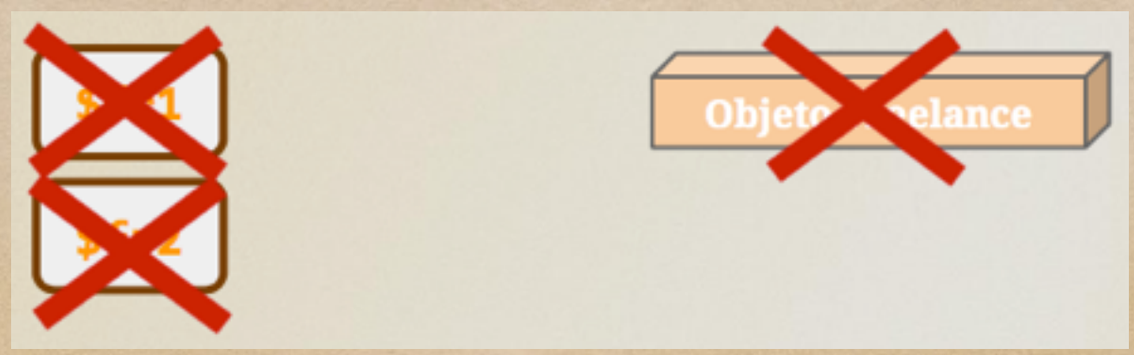
Si eliminamos una de las referencias (punteros), el objeto perdura, pues aun le apunta el otro.

```
unset($fr1);
```



Si eliminamos el segundo de los punteros, el objeto se queda sin nadie que lo referencie y entonces deja de existir.

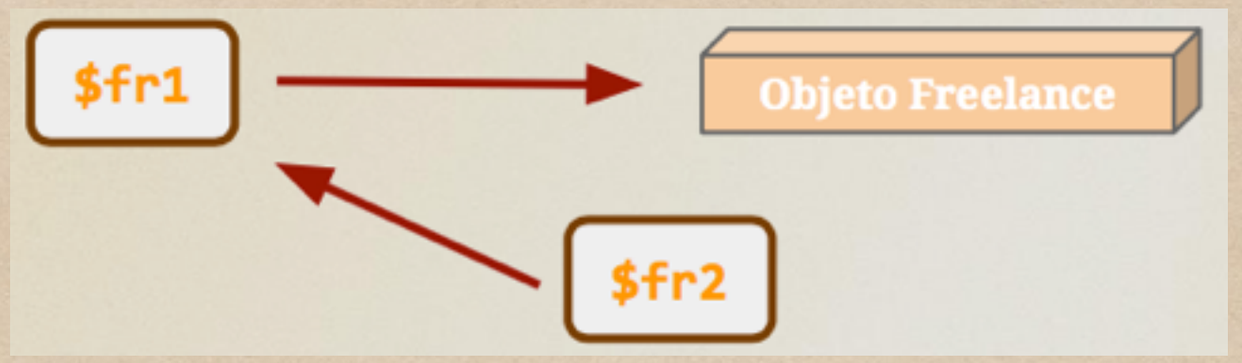
```
unset($fr2);
```



Referencias

Podemos asignar una variable por referencia. Así conseguiríamos que un puntero apunte al otro.

```
$fr1 = new Freelance();
$fr2 = & $fr1;
```

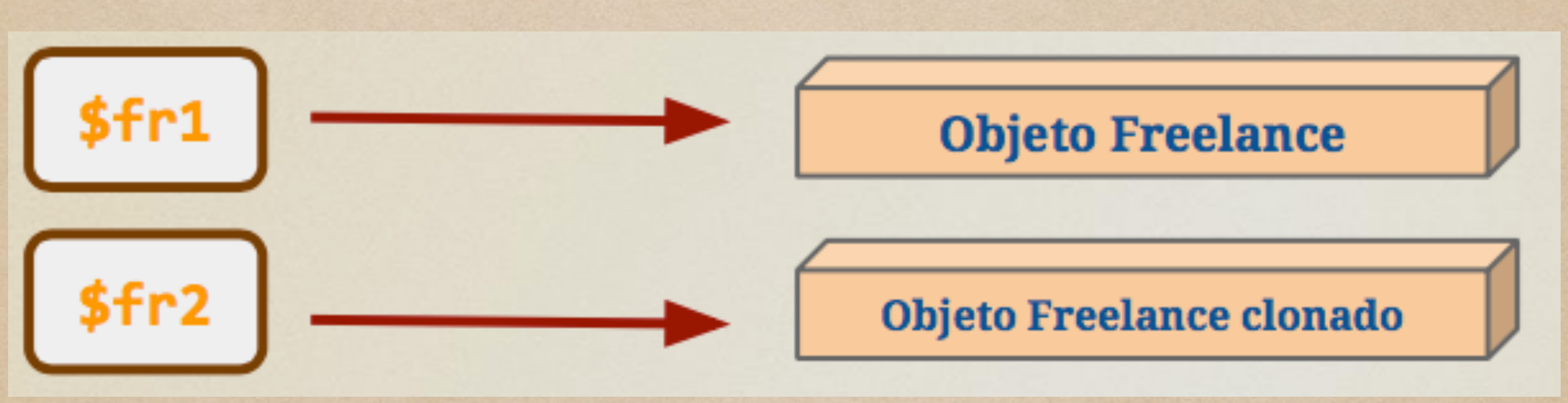


Si ahora igualáramos la referencia $fr1 a “null”, afectaría a ambas.

Referencias

Si nuestra intención no era crear una segunda referencia al objeto sino clonarlo (duplicarlo), tenemos la palabra reservada **clone** que nos permite hacerlo.

```
$fr2 = clone $fr1;
```



Más información en:

http://www.php.net/manual/es/language.oop5.cloning.php

Concepto "static"

static es otro modificador de visibilidad, al igual que los ya vistos **public**, **private** y **protected**.

Son propiedades y métodos "de clase". No están asociados a un objeto en particular, sino de manera global a la clase.

- No necesitamos ningún objeto para acceder a las **propiedades static** o invocar los **métodos static**. Es decir, podemos acceder a ellos sin tener ningún objeto instanciado.

- No tiene sentido acceder a **$this**.

```
class Freelance()
{
 public static $juegoCaracteres = "UTF8";
}
```

Ejemplo

Supongamos que estamos haciendo una clase llamada Círculo. Cualquier objeto de esa clase es un círculo, y entre sus propiedades tenemos el centro y el radio. Y como métodos podemos tener aquellos que me permiten dibujarlo, calcular su perímetro, su área, etc.

Pero existe una propiedad interna y común a todos los círculos, es decir, lo que llamaríamos una propiedad de clase, y es el número PI. Todos los círculos (objetos de la clase) tienen esa propiedad, y es común a todos ellos. Es lo que se conoce como una propiedad de clase.

Y para definir una propiedad o método de clase se usa la palabra **static**.

Y si no necesitamos instanciar un objeto para acceder a estos elementos de clase, ¿cómo accedemos?

Acceso a prop. static, Operador ::

- Acceso desde fuera de la clase:

  A. Nombre de la clase y operador ::

```
echo "<b>" . Freelance::$juegoCaracteres . "</b>";
```

- Acceso desde dentro de la clase:

  A. Nombre de la clase y operador ::

  B. Con la palabra **self** y el operador ::

```
echo "<b>" . self::$juegoCaracteres . "</b>";
```

Es importante remarcar que la palabra reservada **self** sólo la podemos utilizar si estamos dentro del código de la clase.

Ejemplo

```
public function desarrollar(){

 echo "<br>Soy " . $this->nombre;

 echo " y comienzo a trabajar en ";

 echo self::juegoCaracteres;

 $this->ocupado = true;

 $this->comienzoTrabajo = time();

}
```

Métodos static

```
public static function dias_trabajo(){

 if($invierno){
  return array("Lunes", "Martes", "Miércoles",
               "Jueves", "Viernes",
               "Sábado");
 }

 return ["Lunes", "Martes", "Miércoles",
               "Jueves", "Viernes"];
}
```

Acceso a métodos static

- Desde fuera de la clase:

```
Freelance::dias_trabajo();
```

- Desde dentro de la clase:

```
Freelance::dias_trabajo();
self::dias_trabajo();
```

Visibilidad de static

Para la visibilidad, se pueden anteponer las palabras reservadas **public**, **private** y **protected**, y el significado es el mismo que si no se escribiera **static**.

Si no se indica nada, se entiende que es **public**.

Constantes de Clase

Son constantes que pertenecen a la clase.

Son como propiedades static, pero no se les puede cambiar su valor.

```
const ODIAMOS = "Internet Explorer";
//Ojo!!! no se pone el $
```

Por convención las constantes se escriben en mayúsculas, aunque no es necesario para la sintaxis de PHP.

Acceso a constantes de Clase

- Desde fuera de la clase

```
Freelance::ODIAMOS
```

- Desde dentro de la clase

```
public function ocupado(){
  if($this->ocupado){
    echo "<br>Estoy ocupado y además odio " .
          self::ODIAMOS;
  }else{
    echo "<br>Descansando, pero odiando a " .
          self::ODIAMOS;
  }
}
```

Ejemplo: Detectar Móvil

Vamos a crear una clase con un método static que detecte si el usuario está accediendo desde un dispositivo móvil.

Podemos usar http://detectmobilebrowsers.com/

En esta página web nos encontramos el código necesario en PHP para poder crear el método solicitado.

Ver ejemplo MobileHelper.php

Ejemplo: Imagen responsive

Crear una clase ImagenResponsive que tenga un método público mostrar() y que, dependiendo de si es un móvil o no se muestre una imagen en diferentes resoluciones.

Nota: Detectar el móvil una sola vez y se almacena en algún sitio. Así, no tenemos que hacer la detección el resto de veces que se muestra esa imagen o cualquier otra imagen responsive que tengamos.

Ver ejemplo ImagenResponsive.php y responsive.php

Funciones de Objetos

PHP dispone de un conjunto de funciones que podemos utilizar con objetos.

http://php.net/manual/es/ref.classobj.php

También PHP dispone de un conjunto de funciones para trabajar con funciones.

http://www.php.net/manual/es/ref.funchand.php

Ejercicios

- Añadir unos constructores que tengan sentido a las clases de los ejercicios de la Parte II.

- Crear una clase Fecha. Esta clase debe almacenar la fecha con un timestamp. El constructor debe admitir un timestamp o, si no se le envía ningún parámetro, la creará con la fecha actual. La clase debe ser capaz de representar la fecha en formato español y en formato inglés. Además, debe crearse una variable estática para definir el idioma en el que se mostrarán las fechas. Se supone que en una página las fechas son todas representadas con el mismo formato (por eso es estática).

- Objeto que conecta con una BD. Este objeto tiene que construirse enviando los parámetros de conexión. El cierre de la BD debe realizarse en el destructor, para no dejar nunca la conexión abierta. También se pide poder abrir y cerrar la conexión por medio de métodos normales. El destructor no debe cerrar la BD si ya lo está.

- Vacaciones de freelances. Realizar los cambios necesarios en la clase Freelance para poder crear un método Vacaciones() que hace que todos los freelances creados durante la vida de la aplicación paren de trabajar.