Parte V

Abstracción

Abstracción

En PHP tenemos clases y métodos abstractos. La abstracción se refiere a elementos que definen comportamientos independientemente de su concreción.

Por ejemplo, la clase mamífero puede tener como método abstracto "comunicar" ya que todos los mamíferos se comunican, lo que pasa es que no sabemos como implementarlo.

Cuando creamos la clase hija Perro podemos implementar ese método "comunicar" indicando algo como echo "guau guau", o la clase hija Gato con echo "miau miau".

Por eso se define como abstracto ese método. Todos las clases hijas lo van a implementar pero el padre no sabe como.

Clases y métodos abstractos

- Una clase que tiene uno o más métodos abstractos debe ser definida como abstracta.
- Una clase que es abstracta significa que no podemos instanciar objetos de esa clase.

```
abstract class Mamifero
{
 public abstract function comunicar();
}
```

Un método abstracto simplemente muestra su cabecera, no tiene código.

```
abstract class Persona
{
 public abstract function mear();
}
```

Abstracción y herencia

Una clase abstracta con métodos abstractos podría entenderse como una clase que dispone de métodos que no sabemos implementar a ese nivel, pero que todas las clases hijas que hereden de ésta si lo van a tener (y por tanto lo van a tener que redefinir o implementar).

Pero una clase abstracta puede disponer de métodos que si pueda implementar ya que no llevan a ningún tipo de ambigüedad en las clases hijas.

Por tanto, la abstracción cobra sentido en un esquema de herencia:

- Podemos heredar de clases abstractas.
- Para que la clase hija no sea abstracta, se deben definir todos los métodos abstractos.
- La visibilidad de los métodos abstractos, al implantarlos en la clase hija debe ser igual o menos restrictiva que la del padre.
- Si no definimos todos los métodos abstractos, no queda más remedio que definir la clase también como abstracta.

Ejemplo

```
class Hombre extends Persona
{
 public function mear(){
   echo "Me aproximo al urinario. ";
   echo "Apunto y me relajo para vaciar la vejiga";
 }
}
```

EJERCICIO

Terminar la clase Persona y crear dos clases hijas, Mujer y Hombre, que hereden de Persona y que implementen sus métodos abstractos.

Ejemplo Plantilla

El objetivo es renderizar una plantilla, sustituyendo los valores que vienen en el array de datos.

Origen: La plantilla tiene un contenido como

```
<b>Hola %nombre%</b>
<p>Estoy muy %animo%</p>
```

Datos: Cargamos desde un array

```
array("nombre" => "Pepe",
      "animo" => "contento");
```

Esta plantilla debe tener un método abstracto render(). Es abstracto porque las plantillas genéricas no saben donde se deben mostrar, si en la Web, si a un archivo de texto o pdf, si a un mail, un cvs, etc.

Crea plantillas especializadas que hereden de Plantilla, que implementen el método volcar() para generar la salida al medio especificado.

Clase Plantilla

```php
<?php
class Plantilla
{
    private $html; //Nuestra plantilla original
    private $htmlGenerado; //Nuestra plantilla con los valores sustituidos del array

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

    private function cargarDatos($diccionario){
        $this->htmlGenerado = $this->html;
        foreach ($diccionario as $key => $value) {
            $this->htmlGenerado = str_replace("%$key%", $value, $this->htmlGenerado);
        }
    }

    private function render(){
        echo $this->htmlGenerado;
    }

    public function renderData($diccionario){
        $this->cargarDatos($diccionario);
        $this->render();
    }
}
```

Archivo index.php

```
<?php
$html = '<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>mipagina con template</title>
</head>
<body>
    <b>Hola %nombre%</b>,
    <p>Estoy muy %animo%</p>
    <p>Vivo en %ciudad%</p>
</body>
</html>';

$diccionario = ["nombre" => "Pepe",
                "animo" => "contento",
                "ciudad" => "Murcia"];

$plantilla = new Plantilla($html);
$plantilla->renderData($diccionario);
?>
```

Observaciones

Esta primera versión de la clase Plantilla no dispone del método abstracto solicitado.

Esta clase que hemos producido sirve exclusivamente para generar código HTML. Es decir, sirve para generar la salida a una página Web.

Vamos ahora a modificar la clase para aproximarnos a las especificaciones solicitadas en el enunciado, una vez que sabemos que lo que hemos hecho funciona.

Nueva clase Plantilla

```
<?php
abstract class Plantilla
{
    private $html;
    protected $htmlGenerado;

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

    private function cargarDatos($diccionario){
        $this->htmlGenerado = $this->html;
        foreach ($diccionario as $key => $value) {
            $this->htmlGenerado = str_replace("%$key%", $value, $this-
>htmlGenerado);
        }
    }

    protected abstract function render();

    public function renderData($diccionario){
        $this->cargarDatos($diccionario);
        $this->render();
    }
}
```

Clases hijas

```
class PlantillaHTML extends Plantilla
{
    protected function render(){
        echo $this->htmlGenerado;
    }
}

class PlantillaEmail extends Plantilla
{
    private $subject;
    private $to;

    public function __construct($to, $subject, $body){
        parent::__construct($body);
        $this->to = $to;
        $this->subject = $subject;
    }

    protected function render(){
        mail($this->to, $this->subject, $this->htmlGenerado);
    }
}
```

Archivo index.php

```
<?php
$html = '<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>mipagina con template</title>
</head>
<body>
    <b>Hola %nombre%</b>,
    <p>Estoy muy %animo%</p>
    <p>Vivo en %ciudad%</p>
</body>
</html>';

$diccionario = ["nombre" => "Pepe","animo" => "contento","ciudad" => "Murcia"];

$plantilla = new PlantillaHTML($html);
$plantilla->renderData($diccionario);
$cuerpoEmail = 'Hola,
Gracias por registrarse en nuestro sitio, sus datos son:
Usuario: %user%
Clave: La que escribiste';
$plantillaEmail = new PlantillaEmail("iescierva.carlos@gmail.com", "Registro
completado", $cuerpoEmail);
$plantillaEmail->renderData(["user" => "carlos"]);
?>
```