Introducción a la programación orientada a objeto con PHP

Maximiza la potencia de PHP.

Introducción a la programación orientada a objeto con PHP

Maximiza la potencia de PHP.

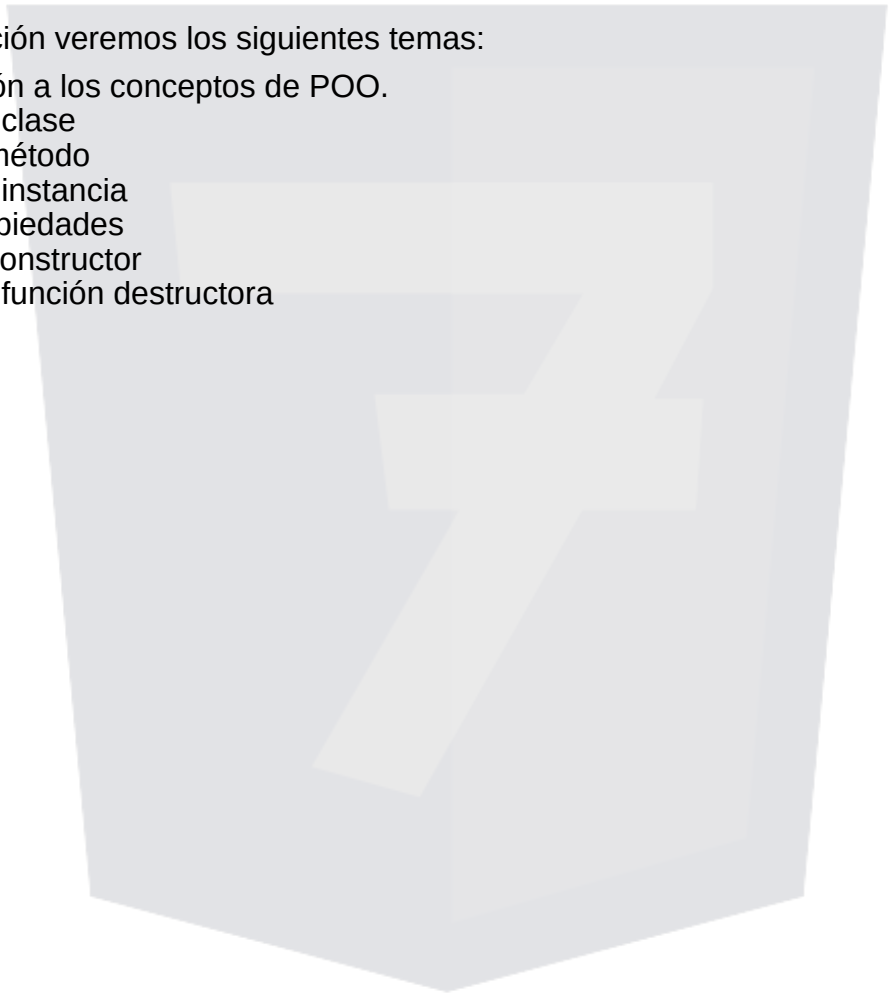
Capítulo 1: Introducción a la programación orientada a objetos en PHP

Objetivo: El alumno aprenderá los principios y fundamentos de la programación orientada a objetos en cualquier lenguaje

Introducción:

En esta sección veremos los siguientes temas:

1. Introducción a los conceptos de POO.
2. Crear una clase
3. Crear un método
4. Crear una instancia
5. Crear propiedades
6. Crear un constructor
7. Crear una función destructora



1.1. Principios generales de programación orientada a objetos

La **Programación orientada a objetos** u **Object Oriented Programming** (OOP) es una pieza fundamental para el desarrollo de gran magnitud. En este curso aprenderás a programar PHP con este paradigma.

1.2. Tipo de datos: Enumeraciones

A partir de la versión 5.0 de PHP contamos con la palabra reservada "class" con la cual podemos hacer clases y con ellas la Programación Orientada a Objetos.

```
1  <?php
2  class Gatos{
3
4  }
5  /*
6  $clases = get_declared_classes();
7  foreach ($clases as $clase) {
8    print $clase."<br>";
9  }
10 */
11 if (class_exists("Gatos")) {
12   print "La clase Gatos si existe"."<br>";
13 } else {
14   print "La clase Gatos NO existe"."<br>";
15 }
16
17
18 ?>
```

Listado 1.2.1. crearClase.php

1.3. Crear un método de una clase con PHP

Otra pieza fundamental de la programación orientada a objetos en PHP (en cualquier lenguaje OOP) es la creación de los métodos, que no son otra cosa mas que nuestras conocidas funciones, pero con esteroides.

```
1  <?php
2  class Gato{
3    function maullar(){
4      print "El gato dice miau miau"."<br>";
5    }
6    function ronronear(){
7      print "El gato ronronea"."<br>";
8    }
9  }
10
11 $metodos = get_class_methods("Gato");
12 foreach ($metodos as $metodo) {
13   print $metodo."<br>";
14 }
15
16 if (method_exists("Gato", "maullar")) {
17   print "Los gatos pueden maullar"."<br>";
18 } else {
19   print "Los gatos NO pueden maullar"."<br>";
20 }
21 ?>
```

Listado 1.3.1. clasesMetodos.php

1.4. Instanciar un objeto de una clase con PHP

Una ***instancia*** es crear un objeto que podemos utilizar en base a la clase que hemos definido, por lo que tendrá los mismos métodos y propiedades iniciales que definimos en nuestro archivo de clase.

```
1  <?php
2  class Gato{
3    function maullar(){
4      return "miau, miau";
5    }
6  }
7
8  //Creamos las instancias
9  $cucho = new Gato();
10 $benito = new Gato();
11 $espanto = new Gato();
12
13 //detectar la clase de una instancia / objeto
14 print "Espanto pertenece a la clase ".get_class($espanto)."<br>";
15
16 //Verificar que un objeto pertenezca a una clase
17 print "Matute ";
18 if (is_a($matute,"Gato")) {
19   print "Si es un gato"."<br>";
20 } else {
21   print "No es un gato"."<br>";
22 }
23
24 //Llamar a un metodo
25 print "Cucho dice ".$cucho->maullar()."<br>";
26 print "Benito dice ".$benito->maullar()."<br>";
27 print "Espanto dice ".$espanto->maullar()."<br>";
28 ?>
```

Listado 1.4.1. claseInstancia.php

1.5. Crear propiedades de clase con PHP

Las propiedades las podemos ubicar como lo que le es "propio" a la instancia. No son otra cosa que variables que definen al objeto y que son establecidas en el archivo de clase.

```
1  <?php
2  class Gato{
3    var $nombre;
4    var $colorPelo;
5    var $corbata = "SI";
6
7    function maullar(){
8      return "miau, miau";
9    }
10
11   function tieneCorbata(){
12     return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
13   }
14 }
15 //instancias
16 $cucho = new Gato();
17
18 //Poblar las propiedades
19 $cucho->nombre = "Cucho";
20 $cucho->colorPelo = "rosa";
21 $cucho->corbata = "NO";
22
23 print $cucho->nombre." dice ".$cucho->maullar()."<br>";
24 print $cucho->tieneCorbata();
25 ?>
```

Listado 1.5.1. clasesPropiedades.php

1.6. Crear el constructor de la clase

Un ***constructor*** es la función que se ejecuta cuando el objeto es creado o instanciado, y nos servirá para recibir parámetros iniciales o para ejecutar acciones o métodos cuando es objeto es "construido".

```
1  <?php
2  class Gato{
3    var $nombre;
4    var $colorPelo;
5    var $corbata = "SI";
6
7    function __construct($nombre="", $pelo="negro"){
8      $this->nombre = $nombre;
9      $this->colorPelo = $pelo;
10   }
11
12   function maullar(){
13     return "miau, miau";
14   }
15
16   function tieneCorbata(){
17     return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
18   }
19
20   function saludo(){
21     $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
22     $cadena .= $this->colorPelo;
23     return $cadena;
24   }
25 }
26 //instancias
27 $cucho = new Gato("Cucho","rosa");
28
29 print $cucho->nombre." dice ".$cucho->maullar()."<br>";
30 print $cucho->tieneCorbata();
31 print $cucho->saludo();
32 ?>
```

Listado 1.6.1. clasesConstructor.php

1.7. Crear una función destruct para los objetos de nuestra clase

Una función destructora o "***destruct***" se ejecuta cuando el objeto deja de existir, ya sea porque se termina la página o porque le damos una sentencia "***unset***".

```
1  <?php
2  class Gato{
3    var $nombre;
4    var $colorPelo;
5    var $corbata = "SI";
6
7    function __construct($nombre="", $pelo="negro"){
8      $this->nombre = $nombre;
9      $this->colorPelo = $pelo;
10   }
11
12   function __destruct(){
13     print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14   }
15
16   function maullar(){
17     return "miau, miau";
18   }
19
20   function tieneCorbata(){
21     return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
22   }
23
24   function saludo(){
25     $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
26     $cadena .= $this->colorPelo;
27     return $cadena;
28   }
29 }
30 //instancias
31 $cucho = new Gato("Cucho","rosa");
32
33 print $cucho->nombre." dice ".$cucho->maullar()."<br>";
34 print $cucho->tieneCorbata();
35 print $cucho->saludo()."<br>";
36
37 unset($cucho);
38 ?>
```

Listado 1.7.1. clasesDestructor.php

Introducción a la programación orientada a objeto con PHP

Maximiza la potencia de PHP.

Capítulo 2: Fundamentos de la programación orientada a objetos

Objetivo: El alumno comprenderá los principios de herencia, setters, getters, métodos y propiedades estáticas

Introducción:

En esta sección veremos los siguientes temas:

1. Principios de herencia en PHP
2. Modificadores de acceso
3. Crear setters y getters
4. Métodos mágicos _get y _set
5. Métodos y propiedades estáticas
6. Sobreescribir un método en una clase
7. El operador de resolución de alcance (self, parent)
8. Clonar objetos en PHP

2.1. Principios de herencia en PHP

- El valor de la ***constante*** es la misma para todas las instancias, no se pueden modificar.
- El valor de una ***constante*** no puede ser una variable, función o expresión.
- Por omisión, las variables de clase son ***públicas***.
- Las constantes se diferencian de las variables en que ***no es necesario el símbolo de pesos o dola***r para definirlas ni para utilizarlas.

```
1  <?php
2  class Gato{
3  var $nombre;
4  var $colorPelo;
5  var $corbata = "SI";
6
7  function __construct($nombre="", $pelo="negro"){
8  $this->nombre = $nombre;
9  $this->colorPelo = $pelo;
10 }
11
12 function __destruct(){
13 print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14 }
15
16 function maullar(){
17 return "miau, miau";
18 }
19
20 function tieneCorbata(){
21 return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$this->
22 }
23
24 function saludo(){
25 $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
26 $cadena .= $this->colorPelo;
27 return $cadena;
28 }
29 }
30
31 ?>
```

Listado 2.1.1. claseGato.php


```
1  <?php
2  class Gato{
3    var $nombre;
4    var $colorPelo;
5    var $corbata = "SI";
6
7    function __construct($nombre="", $pelo="negro"){
```

```
8       $this->nombre = $nombre;
9       $this->colorPelo = $pelo;
10    }
11
12    function __destruct(){
13      print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14    }
15
16    function maullar(){
17      return "miau, miau";
18    }
19
20    function tieneCorbata(){
21      return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
22    }
23
24    function saludo(){
25      $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
26      $cadena .= $this->colorPelo;
27      return $cadena;
28    }
29  }
30
31  class GatoVolador extends Gato{
32
33  }
34
35  $cucho = new Gato("Cucho", "rosa");
36  $benito = new GatoVolador("Benito","azul");
37
38  print $cucho->saludo()."<br>";
39  print $benito->saludo()."<br>";
40
41  unset($cucho);
42  unset($benito);
43
44  print "El pariente de la clase Gato es ".get_parent_class("Gato")."<br>";
45  print "El pariente de la clase GatoVolador es ".get_parent_class("GatoVolador")."<br>";
46  print "<br>";
47  print is_subclass_of("Gato", "GatoVolador")?"Si":"No";
48  print "<br>";
49  print is_subclass_of("GatoVolador", "Gato")?"Si":"No";
50  print "<br>";
51  ?>
```

Listado 2.1.2. herencia.php

2.2. Modificadores de acceso

- Otro de los conceptos básicos de la programación orientada a objetos es el ***encapsulamiento***, con lo cual el usuario (otro programador u otro código) sólo podrá ver aquello que nosotros deseemos y lo restante estará " ***encapsulado***" o ***restringido***.
- Los ***modificadores de acceso*** son el pilar en el que se basa el encapsulamiento, ya que con estos ***modificadores*** permitimos que el "usuario" vea o no las propiedades y métodos de nuestras clases.
  - ***public***: acceso al recurso. Valor por omisión.
  - ***private***: sólo tiene acceso dentro de la clase.
  - ***protected***: acceso sólo dentro de la clase y de las clases heredadas.

```
1  <?php
2  class Gato{
3  var $nombre;
4  var $colorPelo;
5  var $corbata = "SI";
6
7  function __construct($nombre="", $pelo="negro"){
8  $this->nombre = $nombre;
9  $this->colorPelo = $pelo;
10 }
11
12 function __destruct(){
13 print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14 }
15
16 function maullar(){
17 return "miau, miau";
18 }
19
20 function tieneCorbata(){
21 return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$this->
22 }
23
24 function saludo(){
25 $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
26 $cadena .= $this->colorPelo;
27 return $cadena;
28 }
29 }
30
31 ?>
```

Listado 2.2.1. claseGato.php

```
1  <?php
2  class Gato{
3  protected $nombre;
4  private $colorPelo;
5  private $corbata = "SI";
```

```
6
7  public function __construct($nombre="", $pelo="negro"){
8  $this->nombre = $nombre;
9  $this->colorPelo = $pelo;
10 }
11
12 function __destruct(){
13 print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14 }
15
16 function maullar(){
17 return "miau, miau";
18 }
19
20 function tieneCorbata(){
21 return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$this->
22 }
23
24 function saludo(){
25 $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
26 $cadena .= $this->colorPelo;
27 return $cadena;
28 }
29 }
30
31 class GatoVolador extends Gato{
32 function nombreGatoVolador(){
33 return $this->nombre;
34 }
35 }
36
37 $cucho = new Gato("Cucho", "rosa");
38 $benito = new GatoVolador("Benito","azul");
39
40 print $cucho->saludo()."<br>";
41 print $benito->saludo()."<br>";
42
43 print "El nombre del gato volador es: ".$benito->nombreGatoVolador()."<br>";
44
45 ?>
```

Listado 2.2.2. encapsulamiento.php

2.3. Crear setters y getters

Otra herramienta que nos permitirá manejar el encapsulamiento de nuestras clases es realizar las funciones "getters" y "setters".

```
1  <?php
2  class Gato{
3  protected $nombre;
4  private $colorPelo;
5  private $corbata = "SI";
6
7  function __construct($nombre="", $pelo="negro"){
8  $this->nombre = $nombre;
9  $this->colorPelo = $pelo;
10 }
11
12 function __destruct(){
13 print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14 }
15
16 function setCorbata($c="SI"){
17 if($c!="SI"){
18 $corbata = "NO";
19 }
20 $this->corbata = $c;
21 }
22
23 function getCorbata(){
24 return $this->corbata;
25 }
26
27 function maullar(){
28 return "miau, miau";
29 }
30
31 function tieneCorbata(){
32 return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$this->
33 }
34
35 function saludo(){
36 $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
37 $cadena .= $this->colorPelo;
38 return $cadena;
39 }
40 }
41
42 class GatoVolador extends Gato{
43 function nombreGatoVolador(){
44 return $this->nombre;
```

```
45 }
46 }
47
48 $cucho = new Gato("Cucho", "rosa");
49 $benito = new GatoVolador("Benito","azul");
50
51 print $cucho->saludo()."<br>";
52 print $benito->saludo()."<br>";
53
54 print "El nombre del gato volador es: ".$benito->nombreGatoVolador()."<br>";
55
56 $cucho->setCorbata("NO");
57
58 print $cucho->tieneCorbata();
59 print $benito->tieneCorbata();
60
61 ?>
```

Listado 2.3.1. gettersSetters.php

2.4. Métodos mágicos _get y _set

- Otros de los llamados "***métodos mágicos***" son los __get y __set, que curiosamente NO sirven para hacer getters y setters.

__set() se ejecuta al escribir datos sobre propiedades inaccesibles.

__get() se utiliza para consultar datos a partir de propiedades inaccesibles

```
1  <?php
2  class Gato{
3  protected $nombre;
4  private $colorPelo;
5  private $corbata = "SI";
6
7  function __construct($nombre="", $pelo="negro"){
8  $this->nombre = $nombre;
9  $this->colorPelo = $pelo;
10 }
11
12 function __destruct(){
13 print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14 }
15
16 function __set($name, $valor){
17 print "La propiedad ".$name." se actualizo a ".$valor."<br>";
18 if($name=="corbata"){
19 if($valor!="SI") $valor = "NO";
20 }
21 $this->$name = $valor;
22 }
23
24 function __get($name){
25 return $this->$name;
26 }
27
28 function setCorbata($c="SI"){
29 if($c!="SI"){
30 $corbata = "NO";
31 }
32 $this->corbata = $c;
33 }
34
35 function getCorbata(){
36 return $this->corbata;
37 }
38
39 function maullar(){
40 return "miau, miau";
41 }
```

```
42
43 function tieneCorbata(){
44 return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$this->
45 }
46
47 function saludo(){
48 $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
49 $cadena .= $this->colorPelo;
50 return $cadena;
51 }
52 }
53
54 class GatoVolador extends Gato{
55 function nombreGatoVolador(){
56 return $this->nombre;
57 }
58 }
59
60 $cucho = new Gato("Cucho", "rosa");
61 $benito = new GatoVolador("Benito","azul");
62
63 print $cucho->saludo()."<br>";
64 print $benito->saludo()."<br>";
65
66 print "El nombre del gato volador es: ".$benito->nombreGatoVolador()."<br>";
67
68 //$cucho->setCorbata("NO");
69
70 $cucho->corbata = "NO";
71
72 print $cucho->tieneCorbata();
73 print $benito->tieneCorbata();
74
75 ?>
```

Listado 2.4.1. getset.php

2.5. Métodos y propiedades estáticas

- Un modificador de acceso de mucha utilidad es "***static***", que nos permitirá utilizar métodos y propiedades sin necesidad de crear instancias de la clase.
- El ***Operador de Resolución de Ámbito*** o el ***doble dos-puntos*** (::), es un operador que permite acceder a elementos estáticos, constantes, y sobrescribir propiedades o métodos de una clase.

```
1  <?php
2  class Gato{
3    public static $claveSecreta = "12345";
4    protected $nombre;
5    private $colorPelo;
6    private $corbata = "SI";
7
8    function __construct($nombre="", $pelo="negro"){
9      $this->nombre = $nombre;
10     $this->colorPelo = $pelo;
11   }
12
13   function __destruct(){
14     print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
15   }
16
17   function __set($name, $valor){
18     print "La propiedad ".$name." se actualizo a ".$valor."<br>";
19     if($name=="corbata"){
20       if($valor!="SI") $valor = "NO";
21     }
22     $this->$name = $valor;
23   }
24
25   function __get($name){
26     return $this->$name;
27   }
28
29   public static function mensajeSecreto(){
30     return "Haz el bien, sin mirar a quien";
31   }
32
33   function setCorbata($c="SI"){
34     if($c!="SI"){
35       $corbata = "NO";
36     }
37     $this->corbata = $c;
38   }
39
40   function getCorbata(){
41     return $this->corbata;
```

```
42   }
43
44   function maullar(){
45     return "miau, miau";
46   }
47
48   function tieneCorbata(){
49     return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
50   }
51
52   function saludo(){
53     $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
54     $cadena .= $this->colorPelo;
55     return $cadena;
56   }
57 }
58
59 class GatoVolador extends Gato{
60   function nombreGatoVolador(){
61     return $this->nombre;
62   }
63 }
64 //Paamayim Nekudotayim
65 print "La clave secreta es: ".Gato::$claveSecreta."<br>";
66 print "La frase secreta es: ".Gato::mensajeSecreto()."<br>";
67
68 ?>
```

Listado 2.5.1. static.php

2.6. Sobreescribir un método en una clase

- Cuando creamos una nueva clase a partir de otra, podemos hacer tres cosas: añadir nuevas propiedades y métodos, eliminarlos o modificarlos. Aquí veremos cómo ***sobreescribir*** o modificar métodos con ***overriding***.

```
1  <?php
2  class Gato{
3    protected $nombre;
4    private $colorPelo;
5    private $corbata = "SI";
6
7    function __construct($nombre="", $pelo="negro"){
8      $this->nombre = $nombre;
9      $this->colorPelo = $pelo;
10   }
11
12   function __destruct(){
13     //print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
14   }
15
16   function __set($name, $valor){
17     print "La propiedad ".$name." se actualizo a ".$valor."<br>";
18     if($name=="corbata"){
19       if($valor!="SI") $valor = "NO";
20     }
21     $this->$name = $valor;
22   }
23
24   function __get($name){
25     return $this->$name;
26   }
27
28   function setCorbata($c="SI"){
29     if($c!="SI"){
30       $corbata = "NO";
31     }
32     $this->corbata = $c;
33   }
34
35   function getCorbata(){
36     return $this->corbata;
37   }
38
39   function maullar(){
40     return "miau, miau";
41   }
42
43   function tieneCorbata(){
```

```
44      return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
45    }
46
47    function saludo(){
48      $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
49      $cadena .= $this->colorPelo;
50      return $cadena;
51    }
52  }
53
54  class GatoVolador extends Gato{
55    function nombreGatoVolador(){
56      return $this->nombre;
57    }
58    function maullar(){
59      return "miau, miau, miau y miau";
60    }
61  }
62
63  $cucho = new Gato("Cucho", "rosa");
64  $benito = new GatoVolador("Benito","azul");
65
66  print $cucho->nombre." maulla asi: ".$cucho->maullar()."<br>";
67  print $benito->nombre." maulla asi: ".$benito->maullar()."<br>";
68
69
70  ?>
```

Listado 2.6.1. overriding.php

2.7. El operador de resolución de alcance (self, parent)

- El operador ***doble dos puntos*** o ***scope resolution*** nos será de mucha utilidad a lo largo del desarrollo de la programación orientada a objetos en PHP.
- self: sustituye a $this cuando llamamos propiedades o métodos estáticos.
- parent: la utilizamos cuando queremos llamar desde la clase hija, propiedades o métodos de la clase padre.

```
1  <?php
2  class Gato{
3  const EDAD = 18;
4  static $claveSecreta = "12345";
5  protected $nombre;
6  private $colorPelo;
7  private $corbata = "SI";
8
9  function __construct($nombre="", $pelo="negro"){
10 $this->nombre = $nombre;
11 $this->colorPelo = $pelo;
12 }
13
14 function __destruct(){
15 //print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
16 }
17
18 function __set($name, $valor){
19 print "La propiedad ".$name." se actualizo a ".$valor."<br>";
20 if($name=="corbata"){
21 if($valor!="SI") $valor = "NO";
22 }
23 $this->$name = $valor;
24 }
25
26 function __get($name){
27 return $this->$name;
28 }
29
30 function setCorbata($c="SI"){
31 if($c!="SI"){
32 $corbata = "NO";
33 }
34 $this->corbata = $c;
35 }
36
37 function getCorbata(){
38 return $this->corbata;
39 }
40
41 function maullar(){
```

```
42 //despedida(); error
43 //$this->despedida(); ok
44 //self::despedida(); ok
45 return "miau, miau ".self::$claveSecreta;
46 }
47
48 static function despedida(){
49 print "'Adios, mundo cruel'"."<br>";
50 }
51
52 function tieneCorbata(){
53 return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$this->
54 }
55
56 function saludo(){
57 $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
58 $cadena .= $this->colorPelo;
59 return $cadena;
60 }
61 }
62
63 class GatoVolador extends Gato{
64 function nombreGatoVolador(){
65 return $this->nombre;
66 }
67 function maullar(){
68 print parent::maullar()."<br>";
69 return "miau, miau, miau y miau";
70 }
71 }
72
73 $cucho = new Gato("Cucho", "rosa");
74 $benito = new GatoVolador("Benito","azul");
75
76 print $cucho->nombre." maulla asi: ".$cucho->maullar()."<br>";
77 print $benito->nombre." maulla asi: ".$benito->maullar()."<br>";
78
79 print Gato::EDAD;
80 ?>
```

Listado 2.7.1. selfParent.php

2.8. Clonar objetos en PHP

- Con la palabra reservada "***clone***" verdaderamente duplicamos el objeto.

```
1  <?php
2  class Gato{
3    const EDAD = 18;
4    static $claveSecreta = "12345";
5    protected $nombre;
6    private $colorPelo;
7    private $corbata = "SI";
8
9    function __construct($nombre="", $pelo="negro"){
10     $this->nombre = $nombre;
11     $this->colorPelo = $pelo;
12   }
13
14   function __destruct(){
15     //print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
16   }
17
18   function setCorbata($c="SI"){
19     if($c!="SI"){
20       $corbata = "NO";
21     }
22     $this->corbata = $c;
23   }
24
25   function setNombre($n="gato"){
26     $this->nombre = $n;
27   }
28
29   function getCorbata(){
30     return $this->corbata;
31   }
32
33   function maullar(){
34     //despedida(); error
35     //$this->despedida(); ok
36     //self::despedida(); ok
37     return "miau, miau ".self::$claveSecreta;
38   }
39
40   static function despedida(){
41     print "'Adios, mundo cruel'"."<br>";
42   }
43
44   function tieneCorbata(){
```

```
45     return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
46   }
47
48   function saludo(){
49     $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
50     $cadena .= $this->colorPelo;
51     return $cadena;
52   }
53 }
54
55 class GatoVolador extends Gato{
56   function nombreGatoVolador(){
57     return $this->nombre;
58   }
59   function maullar(){
60     print parent::maullar()."<br>";
61     return "miau, miau, miau y miau";
62   }
63 }
64
65 $cucho = new Gato("Cucho", "rosa");
66 $benito = new GatoVolador("Benito","azul");
67
68 //Copiar por valor
69 $a = 10;
70 $b = $a;
71
72 //Copiado por referencia
73 $panza = clone $cucho;
74 $panza->setCorbata("SI");
75 $cucho->setCorbata("NO");
76 print $panza->tieneCorbata();
77 print $cucho->tieneCorbata();
78 $panza->setNombre("Panza");
79 print $panza->tieneCorbata();
80 print $cucho->tieneCorbata();
81 ?>
```

Listado 2.8.1. clone.php

Introducción a la programación orientada a objeto con PHP

Maximiza la potencia de PHP.

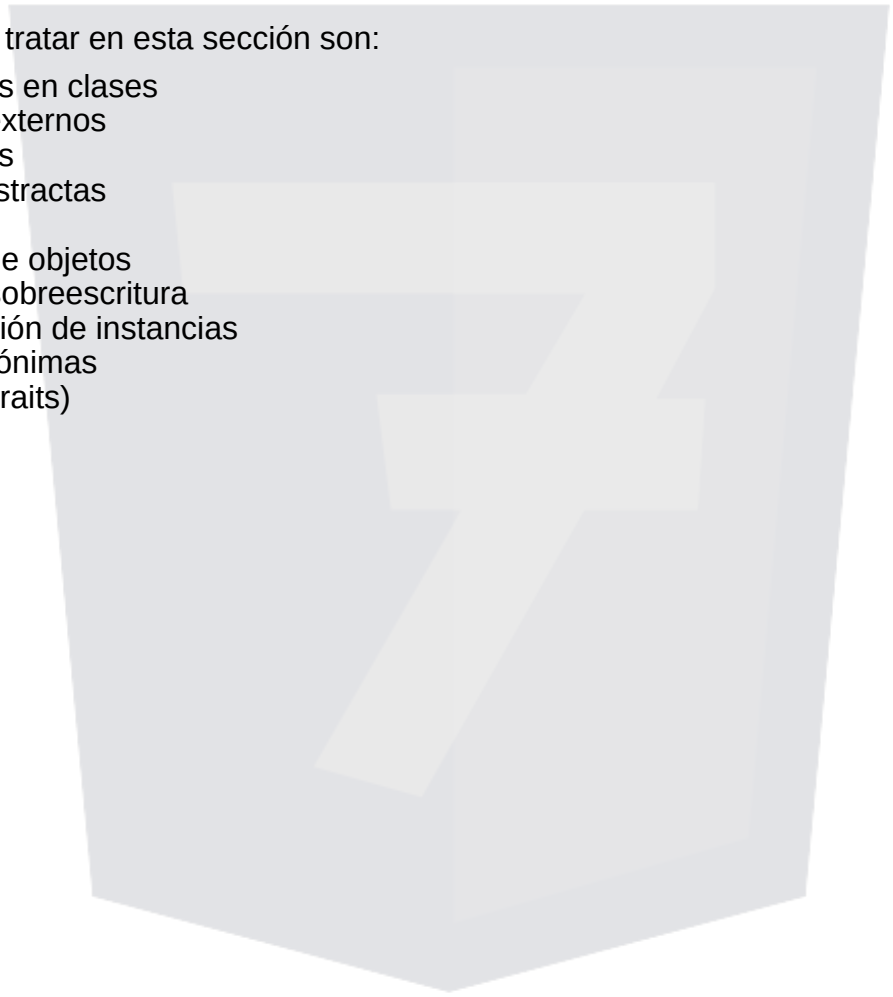
Capítulo 3: Otros temas de la programación orientada a objetos con PHP

Objetivo: El alumno creará interfaces, clases abstractas, clases anónimas, rasgos o traits para modelar nuestras aplicaciones con clases.

Introducción:

Los temas a tratar en esta sección son:

- Constantes en clases
- Archivos externos
- Autocargas
- Clases abstractas
- Interfaces
- Iteración de objetos
- Evitar la sobreescritura
- Comparación de instancias
- Clases anónimas
- Rasgos (Traits)



3.1. Constantes en clases

- Las constantes se diferencian de las variables en que no es necesario el símbolo de pesos o dolar para definirlas ni para utilizarlas.
- Por omisión, las variables de clase son públicas.
- El valor de una constante no puede ser una variable, función o expresión.
- El valor de la constante es la misma para todas las instancias, no se pueden modificar.

```
1  <?php
2  class MiClase
3  {
4      const CONSTANTE = 'valor constante';
5
6      function mostrarConstante() {
7          echo  self::CONSTANTE . "\n";
8      }
9  }
10
11 echo MiClase::CONSTANTE . "\n";
12
13 $nombreclase = "MiClase";
14 echo $nombreclase::CONSTANTE . "\n"; // A partir de PHP 5.3.0
15
16 $clase = new MiClase();
17 $clase->mostrarConstante();
18
19 echo $clase::CONSTANTE."\n"; // A partir de PHP 5.3.0
20 ?>
```

Listado 3.1.1. constantes.php

```
1  <?php
2  class Gato{
3    const EDAD = 18;
4    //5.3.0 Nowdow heredoc
5    const PESO = <<<'EOT'
6    20
7  EOT;
8    //5.6.0
9    const UNO = 1;
10   const DOS = self::UNO + self::UNO;
11   const TRES = self::UNO + self::DOS;
12   const SUMA = "Las vidas de los gatos son ".self::TRES * 3;
13
14   function edadGatuna(){
15     return self::EDAD;
16   }
17 }
18
```

```
19 $benito = new Gato();
20
21 print "La edad maxima de los gatos es ".Gato::EDAD."<br>";
22 print "La edad maxima de los gatos es ".$benito->edadGatuna()."<br>";
23
24 //5.3.0
25 $clase = "Gato";
26 print "El peso maximo de los gatos es ".$clase::PESO."<br>";
27 print $clase::SUMA;
28
29 ?>
```

Listado 3.1.2. constantesClase.php

```
1  <?php
2  class foo {
3      // A partir de PHP 5.3.0
4      const BAR = <<<'EOT'
5  bar
6  EOT;
7      // A partir de PHP 5.3.0
8      const BAZ = <<<EOT
9  baz
10 EOT;
11 }
12 ?>
```

Listado 3.1.3. constantesHeredoc.php

3.2. Archivos externos

- La práctica más común, es almacenar en un archivo externo a cada una de las clases.
- Las podemos llamar con los comandos include, include_once, require, require_once.

```
1  <?php
2  require_once "clases/Gato.php";
3  require_once "clases/GatoVolador.php";
4
5  $cucho = new Gato("Cucho", "rosa");
6  $benito = new GatoVolador("Benito","azul");
7
8  //Copiar por valor
9  $a = 10;
10 $b = $a;
11
12 //Copiado por referencia
13 $panza = clone $cucho;
14 $panza->setCorbata("SI");
15 $cucho->setCorbata("NO");
16 print $panza->tieneCorbata();
17 print $cucho->tieneCorbata();
18 $panza->setNombre("Panza");
19 print $panza->tieneCorbata();
20 print $cucho->tieneCorbata();
21 ?>
```

Listado 3.2.1. includeRequire.php

3.3. Autocargas

- A partir de la versión 5.0.0 de PHP podemos hacer autocargas para evitar las sentencias de llamada include y require.
- Con la función spl_autoload_register() podemos autocargar las clases que llamemos en nuestro programa.
- La función spl_autoload() se considera obsoleta.

```
spl_autoload_register(function ($nombre_clase) {
    include $nombre_clase . '.php';
});
```

```
1  <?php
2  class Gato{
3    const EDAD = 18;
4    static $claveSecreta = "12345";
5    protected $nombre;
6    private $colorPelo;
7    private $corbata = "SI";
8
9    function __construct($nombre="", $pelo="negro"){
10     $this->nombre = $nombre;
11     $this->colorPelo = $pelo;
12   }
13
14   function __destruct(){
15     //print $this->nombre." dice: 'Adios, mundo cruel"."<br>";
16   }
17
18   function __set($name, $valor){
19     print "La propiedad ".$name." se actualizo a ".$valor."<br>";
20     if($name=="corbata"){
21       if($valor!="SI") $valor = "NO";
22     }
23     $this->$name = $valor;
24   }
25
26   function __get($name){
27     return $this->$name;
28   }
29
30   function setCorbata($c="SI"){
31     if($c!="SI"){
32       $corbata = "NO";
33     }
34     $this->corbata = $c;
35   }
36
37   function setNombre($n="gato"){
38     $this->nombre = $n;
```

```
39   }
40
41   function getCorbata(){
42     return $this->corbata;
43   }
44
45   function maullar(){
46     //despedida(); error
47     //$this->despedida(); ok
48     //self::despedida(); ok
49     return "miau, miau ".self::$claveSecreta;
50   }
51
52   static function despedida(){
53     print "'Adios, mundo cruel'"."<br>";
54   }
55
56   function tieneCorbata(){
57     return $this->nombre." ".$this->corbata." tiene corbata y su color de pelo es ".$th
58   }
59
60   function saludo(){
61     $cadena = "Hola, me llamo ".$this->nombre." y mi color de pelo es ";
62     $cadena .= $this->colorPelo;
63     return $cadena;
64   }
65 }
66 ?>
```

Listado 3.3.2. Gato.php

```
1  <?php
2  class GatoVolador extends Gato{
3    function nombreGatoVolador(){
4      return $this->nombre;
5    }
6    function maullar(){
7      print parent::maullar()."<br>";
8      return "miau, miau, miau y miau";
9    }
10 }
11 ?>
```

Listado 3.3.3. GatoVolador.php

3.4. Clases abstractas

- Las ***clases abstractas*** no pueden ser instanciadas.
- Pueden declarar la existencia de los métodos, pero no su implementación.
- Puede contener métodos ***no-abstractos***, pero debe de contener al menos un método abstracto.
- Los ***métodos abstractos*** deben ser definidos en las clases heredadas, pero deben tener la misma visibilidad.
- La implementación debe tener la misma "***firma***": la declaración de tipos y el número de argumentos.
- Las ***clases abstractas*** están implementadas desde la versión 5.4.0.

```
1  <?php
2  abstract class Mamifero{
3    //Metodo Abstracto
4    abstract public function saludo();
5
6    //Método no-abstracto
7    public function maullar(){
8      return "miau, miau";
9    }
10 }
11 class Gato extends Mamifero{
12   public function saludo(){
13     return "Hola Mundo";
14   }
15 }
16
17 $demostenes = new Gato();
18
19 print "saludo ".$demostenes->saludo()."<br>";
20 print "Maullar ".$demostenes->maullar()."<br>";
21 ?>
```

Listado 3.4.1. clasesAbstractas.php

3.5. Interfaces

- Las **interfaces** nos permiten crear código con el cual especificar qué métodos deben ser implementados por una clase (prototipos), sin definir su implementación.
- Las ***interfaces*** se definen con la palabra reservada *interface*.
- Los métodos declarados en una interfaz deben ser **públicos**.
- Para implementar una interface desde una clase se utiliza la palabra reservada *implements*.
- Una clase puede implementar más de una interface si se desea.

```
1  <?php
2  interface iMamifero{
3    public function andar();
4    public function decir();
5  }
6
7  class Gato implements iMamifero{
8    public function andar(){
9      return "camina";
10   }
11   public function decir(){
12     return "miau, miau";
13   }
14 }
15
16 class Delfin implements iMamifero{
17   public function andar(){
18     return "nada";
19   }
20   public function decir(){
21     return "iu. iu";
22   }
23 }
24
25 class Murcielago implements iMamifero{
26   public function andar(){
27     return "volar";
28   }
29   public function decir(){
30     return "iiii, iiii";
31   }
32 }
33
34 $gato = new Gato();
35 $delfin = new Delfin();
36 $murcielago = new Murcielago();
37
38 print "El gato ".$gato->andar()." y dice ".$gato->decir()."<br>";
39 print "El delfin ".$delfin->andar()." y dice ".$delfin->decir()."<br>";
40 print "El murcielago ".$murcielago->andar()." y dice ".$murcielago->decir()."<br>";
```

```
41  ?>
```

Listado 3.5.1. interfaces.php

3.6. Iteración de objetos

- Desde la versión 5 podemos iterar las propiedades visibles o públicas de una clase.

```
1  <?php
2  class Gato{
3    public $peso = "publico";
4    private $genero = "private";
5    protected $edad = "protegido";
6
7    function iteracion(){
8      print "Iteracion dentro de la clase"."<br>";
9      foreach ($this as $clave => $valor) {
10       print $clave." -> ".$valor."<br>";
11     }
12   }
13 }
14
15 $gato = new Gato();
16
17 $gato->iteracion();
18
19 print "<br>"."Iteracion fuera de la clase"."<br>";
20     foreach ($gato as $clave => $valor) {
21       print $clave." -> ".$valor."<br>";
22     }
23 ?>
```

Listado 3.6.1. iteracion.php

3.7. Evitar la sobreescritura

- Una clase también puede ser final y no se permitiría que se extendiera o heredara a otra clase.
- Por medio de la palabra reservada final evitamos que la clase derivada sobreescriba un método.

```
1  <?php
2  final class Gato{
3    public function ronronear(){
4      return "ronronear";
5    }
6    final function maullar(){
7      return "miau, miau";
8    }
9  }
10
11 class GatoVolador extends Gato{
12   /*public function maullar(){
13     return "miauuuuu, miauuuuu";
14   }*/
15
16   public function ronronear(){
17     return "ron, ron, ron";
18   }
19 }
20
21 $gatoVolador = new GatoVolador();
22
23 print "Maullar: ".$gatoVolador->maullar()."<br>";
24 print "Ronronear: ".$gatoVolador->ronronear()."<br>";
25 ?>
```

Listado 3.7.1. final.php

3.8. Comparación de instancias

- Dos instancias de una clase son iguales si tienen los mismos atributos y valores (los valores se comparan con el operador de igualdad ==).
- Cuando se utiliza el operador identidad (===), las variables de un objeto son idénticas sí y sólo sí hacen referencia a la misma instancia de la misma clase.

```
1  <?php
2  class Gato{
3  public $bandera;
4  }
5  class Perro{
6  public $bandera;
7  }
8  $gato1 = new Gato();
9  $gato2 = new Gato();
10 $gato3 = $gato1;
11 $perro1 = new Perro();
12
13 print "Comparamos elementos de la misma clase."."<br>";
14 print "gato1 == gato2 =>";
15 print ($gato1==$gato2)?"Verdadero":"Falso";
16 print "<br>";
17
18 print "gato1 != gato2 =>";
19 print ($gato1!=$gato2)?"Verdadero":"Falso";
20 print "<br>";
21
22 print "gato1 === gato2 =>";
23 print ($gato1===$gato2)?"Verdadero":"Falso";
24 print "<br>";
25
26 print "gato1 !== gato2 =>";
27 print ($gato1!==$gato2)?"Verdadero":"Falso";
28 print "<br>";
29
30 print "<br>";
31 print "Comparamos elementos de la misma clase a la misma referencia."."<br>";
32 print "gato1 == gato3 =>";
33 print ($gato1==$gato3)?"Verdadero":"Falso";
34 print "<br>";
35
36 print "gato1 != gato3 =>";
37 print ($gato1!=$gato3)?"Verdadero":"Falso";
38 print "<br>";
39
40 print "gato1 === gato3 =>";
41 print ($gato1===$gato3)?"Verdadero":"Falso";
```

```
42 print "<br>";
43
44 print "gato1 !== gato3 =>";
45 print ($gato1!==$gato3)?"Verdadero":"Falso";
46 print "<br>";
47
48 print "<br>";
49 print "Comparamos elementos de la diferente clase"."<br>";
50 print "gato1 == perro1 =>";
51 print ($gato1==$perro1)?"Verdadero":"Falso";
52 print "<br>";
53
54 print "gato1 != perro1 =>";
55 print ($gato1!=$perro1)?"Verdadero":"Falso";
56 print "<br>";
57
58 print "gato1 === perro1 =>";
59 print ($gato1===$perro1)?"Verdadero":"Falso";
60 print "<br>";
61
62 print "gato1 !== perro1 =>";
63 print ($gato1!==$perro1)?"Verdadero":"Falso";
64 print "<br>";
65 ?>
```

Listado 3.8.1. comparacion.php

3.9. Clases anónimas

- Las clases anónimas son útiles para definir objetos sencillos y “desechables”.
- Las clases anónimas se definen sólo en la versión 7 y superiores.

```
1  <?php
2  $gato = new Class("Don Gato"){
3  private $nombre;
4
5  function __construct($nombre){
6  $this->nombre = $nombre;
7  }
8  public function getNombre(){
9  return "Mi nombre es ".$this->nombre."<br>";
10 }
11 };
12 print $gato->getNombre();
13 ?>
```

Listado 3.9.1. clasesAnonimas.php

3.10. Rasgos (Traits)

- Por medio de los rasgos o traits podemos reducir las limitaciones de la herencia simple (sólo se puede heredar una clase).
- Podemos reutilizar código (conjuntos de métodos) sobre clases independientes o a jerarquía de clase diferentes.
- Un rasgo o trait es similar a las clases, pero solo agrupa métodos específicos

```
1  <?php
2  interface Animal{
3  function nacer();
4  function crecer();
5  function reproducir();
6  function morir();
7  }
8
9  abstract class Vertebrado implements Animal{
10 private $huesos;
11
12 public function getHuesos(){ return $this->huesos; }
13 public function nacer(){};
14 public function crecer(){};
15 public function reproducir(){};
16 public function morir(){};
17 }
18
19 abstract class Invertebrado implements Animal{
20 use Oviparo;
21 private $hemocianina;
22
23 public function getHemocianina(){ return $this->hemocianina; }
24 public function nacer(){};
25 public function crecer(){};
26 public function reproducir(){};
27 public function morir(){};
28 }
29
30 trait Oviparo{
31 private $huevos;
32 public function getHuevos(){ return $this->huevos; }
33 }
34
35 class Molusco extends Invertebrado{
36 private $radula;
37 public function getRadula(){ return $this->radula; }
38 }
39 class Reptil extends Vertebrado{
40 use Oviparo;
41 private $escamas;
```

```
42 public function getEscamas(){ return $this->escamas; }
43 }
44
45 class Tortuga extends Reptil{}
46 class Pulpo extends Molusco{}
47 ?>
```

Listado 3.10.1. traits.php

Introducción a la programación orientada a objeto con PHP

Maximiza la potencia de PHP.

Capítulo 4: Espacios de nombres o namespace

Objetivo: El alumno manejará las sesiones por medio de una clase bajo el paradigma de la programación orientada a objetos

Introducción:

En esta capítulo estudiaremos los siguientes puntos de los espacios de nombres o namespace:

1. Introducción a los espacio de nombres o namespace.
2. Definir espacios de nombres
3. Definir sub-espacios de nombres
4. Definir varios espacios de nombres
5. Llamado de los espacios de nombres
6. Espacios de nombres y características dinámicas del lenguaje
7. La palabra reservada namespace y la constante __NAMESPACE__
8. Uso de los espacios de nombres: el comando “use”
9. Espacio global

4.1. Introducción a los espacio de nombres o namespace.

- Los Espacios de nombres o namespaces son una forma de “encapsular” elementos de tal manera de distinguirlos de otros elementos, que podrían llamarse igual.
- Una analogía podría ser “apellidos”: los archivos se llaman “juan.txt” pero uno es “perez” y el otro “lopez”.
- Una mejor analogía es el sistemas de archivos de un disco duro.
- Los espacios de nombres nos permiten agrupar clases, interfaces, funciones y constantes relacionadas por lo general en una librería.
- En PHP se utilizan los espacios de nombres para la creación de librerías, evitando el conflicto de nombre de funciones o variables, y para simplificar nombres de elementos muy largos, que tratan de evitar el primer problema en el desarrollo de código reutilizable.

Nota: Los nombres de los espacios de nombres PHP y php, y los nombres compuestos a partir de estos (como PHP\Clases) están reservados para el uso interno del lenguaje y no deben utilizarse en el código del espacio del usuario.

4.2. Definir espacios de nombres

Versiones: PHP 5 >= 5.3.0, PHP 7

- Aunque cualquier código de PHP válido puede estar contenido dentro de un espacio de nombres, solamente se ven afectados por espacios de nombres los siguientes tipos de código: clases (incluyendo abstractas), interfaces, funciones y constantes.
- Los *espacios de nombres* se pueden definir en diferentes archivos.
- Los espacios de nombres se definen con la palabra reservada *namespace* que debe de definirse antes de cualquier sentencia (incluyendo espacios en blanco), excepto de las sentencia "*declare*".

```
1 <?php
2 namespace Perro;
3
4 const NOMBRE = "Lazzy";
5 class MiPerro { /*...*/}
6 function pasearAlPerro() { /*******/ }
7
8 ?>
```

Listado 4.2.1. crearEspacioNombres.php

4.3. Definir sub-espacios de nombres

Versiones: PHP 5 >= 5.3.0, PHP 7

- Al igual que los archivos y sus carpetas, podemos tener subespacios de nombres. Un espacio de nombres se puede definir por sus subespacios.

```
1 <?php
2 namespace Perro;
3
4 const NOMBRE = "Lazzy";
5 class MiPerro { /*...*/}
6 function pasearAlPerro() { /*******/ }
7
8 ?>
```

Listado 4.3.1. crearEspacioNombres.php

```
1 <?php
2 namespace Perro;
3
4 const NOMBRE = "Lazzy";
5 class MiPerro { /*...*/}
6 function pasearAlPerro() { /*******/ }
7
8 ?>
```

Listado 4.3.2. NamespaceSubespacio.php

4.4. Definir varios espacios de nombres

Versiones: PHP 5 >= 5.3.0, PHP 7

- Si se usa la segunda sintaxis, sólo se permiten las sentencias “declare” fuera de las llaves.
- Es más recomendada la segunda a la primera. Nunca hay que combinar ambas sintaxis.
- Podemos definir varios espacios de nombre dentro de un mismo archivo bajo dos posibilidades: combinación simple y sintaxis de llaves.

```
1  <?php
2  namespace Perro{
3  const NOMBRE = "Lazzy";
4  class Comer{/***********/}
5  function pasear(){/***********/}
6  }
7
8  namespace Gato{
9  const NOMBRE = "Garfield";
10 class Comer{/***********/}
11 function pasear(){/***********/}
12 }
13 ?>
```

Listado 4.4.1. namespaceVarios.php

```
1  <?php
2  namespace Perro;
3  const NOMBRE = "Lazzy";
4  class Comer{/***********/}
5  function pasear(){/***********/}
6
7  namespace Gato;
8  const NOMBRE = "Garfield";
9  class Comer{/***********/}
10 function pasear(){/***********/}
11 ?>
```

Listado 4.4.2. namespaceVariosSinLlaves.php

4.5. Llamado de los espacios de nombres

Versiones: PHP 5 >= 5.3.0, PHP 7

- Llamar al archivo con una ruta absoluta.
- Llamar al archivo con una ruta relativa.
- Llamar al archivo sin ninguna ruta.

El manejo de los espacios de nombres es muy similar al uso de un sistema de archivos de un disco duro.

- Nombre no cualificado
- Nombre cualificado
- Nombre completamente cualificado
- Global (no pertenece a ningún espacio de nombre)

```
1  <?php
2  namespace Animal\Perro;
3  include "namespaceLlamarPerro.php";
4
5  const NOMBRE = "Perro";
6  function comer(){
7  print "Estoy comiendo, perro<br>";
8  }
9  class Pasear{
10 static function paseo(){
11 print "Estoy paseando, perro<br>";
12 }
13 }
14
15 //Nombre no cualificado
16 comer();
17 Pasear::paseo();
18 print NOMBRE."<br>";
19 print "<hr>";
20 //Nombre cualificado
21 salchicha\comer();
22 salchicha\Pasear::paseo();
23 print salchicha\NOMBRE."<br>";
24 print "<hr>";
25 //Nombre completamente cualificado
26 \Animal\Perro\salchicha\comer();
27 \Animal\Perro\salchicha\Pasear::paseo();
28 print \Animal\Perro\salchicha\NOMBRE."<br>";
29 ?>
```

Listado 4.5.1. namespaceLlamarAnimal.php

```
1  <?php
2  namespace Animal\Perro\salchicha;
3
```

```
4  const NOMBRE = "Canelita";
5  function comer(){
6  print "Estoy comiendo, Canelita<br>";
7  }
8  class Pasear{
9  static function paseo(){
10 print "Estoy paseando, Canelita<br>";
11 }
12 }
13 ?>
```

Listado 4.5.1. namespaceLlamarPerro.php

4.6. Espacios de nombres y características dinámicas del lenguaje

Versiones: PHP 5 >= 5.3.0, PHP 7

- La implementación de PHP de los espacios de nombres está influenciada por su naturaleza dinámica como lenguaje de programación.

```
1  <?php
2  namespace miNamespace;
3
4  class miClase{
5  function __construct(){
6  print "Hola desde la funcion constructora<br>";
7  }
8  }
9  function miFuncion(){
10 print "Hola desde la función<br>";
11 }
12 const miConstante = "Hola<br>";
13
14 //dinamica
15 $a = "\miNamespace\miClase";
16 $clase = new $a;
17 $b = "\miNamespace\miFuncion";
18 $b();
19 print constant("\miNamespace\miConstante");
20 ?>
```

Listado 4.6.1. namespaceDinamico.php

4.7. La palabra reservada namespace y la constante __NAMESPACE__

Versiones: PHP 5 >= 5.3.0, PHP 7

- El valor de __NAMESPACE__ es una cadena que contiene el nombre del espacio de nombres actual. En código global, que no es de espacio de nombres, contiene una cadena vacía.
- PHP admite dos formas de acceder de manera *abstracta* a elementos dentro del espacio de nombres actual: la constante __NAMESPACE__, y la palabra reservada namespace.

```
1  <?php
2  namespace Perro;
3  //print "***".namespace."***";
4  class Ladrar{
5  function __construct()
6  {
7  print "guau, guau<br>";
8  }
9  }
10
11 $perro = new namespace\Ladrar();
12 ?>
```

Listado 4.7.1. __NAMESPACE__.php

4.8. Uso de los espacios de nombres: el comando use

Versiones: PHP 5 >= 5.3.0, PHP 7

- El comando “use” nos permite referirnos a un nombre completamente *cualificado externo* con un alias. Esto es similar a los sistemas de ficheros basados en un enlace simbólico a un archivo o directorio.
- PHP 5.6+ también permite apodar o importar nombres de funciones y constantes.
- Se admiten tres tipos de alias o importación.

Versiones: PHP7

- Las clases, funciones y constantes que se importen desde el mismo namespace ahora pueden ser agrupadas en una única sentencia use.

```
1  <?php
2  require "use.php";
3
4  use Animales\Mamiferos\{ Perro as MiPerro, Gato };
5  use function Animales\Mamiferos\{ladrar as ladrido, maullar};
6  use const Animales\Mamiferos\{PERRO as DOG, GATO as CAT };
7
8  print "<h2>Clases del espacio de nombres</h2>";
9  $perro = new MiPerro;
10 $gato = new Gato;
11
12 print "<h2>Funciones del espacio de nombres</h2>";
13 ladrido();
14 maullar();
15
16 print "<h2>Constantes del espacio de nombres</h2>";
17 print DOG."<br>";
18 print CAT."<br>";
19 ?>
```

Listado 4.8.1. index.php

```
1  <?php
2  namespace Animales\Mamiferos;
3
4  class Perro{
5  function __construct(){
6  print "Hola, soy un perro<br>";
7  }
8  }
9  class Gato{
10 function __construct(){
11 print "Hola, soy un gato<br>";
12 }
13 }
14 function ladrar(){ print "Guau, guau<br>"; }
15 function maullar(){ print "Miau, miau<br>"; }
16
```

```
17 const PERRO = "Lazzy";
18 const GATO = "Gardfield";
19 ?>
```

Listado 4.8.2. use.php

4.9. Espacio global

Versiones: PHP 5 >= 5.3.0, PHP 7

- Por medio de \ se especificará que el nombre es requerido desde el *espacio global* incluso en el contexto del espacio de nombres.
- Si no hay una definición de espacio de nombres o *namespace*, todas las definiciones de clases y funciones son colocadas en el *espacio global*.

```
1  <?php
2  namespace miNameSpace\miSubNameSpace;
3
4  function fopen($archivo){
5  $f = \fopen($archivo);
6  return $f;
7  }
8
9  class Exception extends \Exception{}
10
11 $e = new Expection("Hola"); //la clase del namespace
12 $e2 = new \Expection("Hola, otra vez"); //nos referimos a la clase global
13
14 const E_ERROR = 40;
15
16 print E_ERROR;
17 print \E_ERROR;
18 ?>
```

Listado 4.9.1. namespaceGlobal.php

Introducción a la programación orientada a objeto con PHP

Maximiza la potencia de PHP.

Capítulo 5: Realizar una sencilla aplicación con programación orientada a objetos

Objetivo: Creará una sencilla aplicación para intercambio de estampitas bajo el paradigma de la programación orientada a objetos

Introducción:

5.1. Listado de la aplicación de control de álbumes y estampitas

En esta sección encontrarás los listados de la aplicación final.

```
1 <?php
2 require_once("clases/Sesion.php");
3 $sesion = new Sesion();
4 if($sesion->estadologin()){
5 header("location:inicio.php");
6 } else {
7 header("location:login.php");
8 }
9 ?>
```

Listado 5.1.1. index.php

```
1  <?php
2  require_once("clases/Sesion.php");
3  require_once("clases/MySQLdb.php");
4  require_once("clases/Usuario.php");
5  $sesion = new Sesion();
6  if(isset($_POST["correo"])){
7  $usuario = $_POST["correo"];
8  if(Usuario::buscaUsuario($usuario)){
9  $sesion->inicioLogin($usuario);
10 header("location:inicio.php");
11 exit;
12 }
13 }
14 ?>
15 <!DOCTYPE html>
16 <html>
17 <head>
18 <meta charset="utf-8">
19 <title>Login</title>
20 </head>
21 <body>
22 <form action="login.php" method="post">
23 <label for="correo">Correo:</label>
24 <input type="email" name="correo" id="correo"/><br>
25 <input type="submit" value="Entrar"/>
26 </form>
27 </body>
28 </html>
```

Listado 5.1.2. login.php

```
1 <?php
2 require "clases/Sesion.php";;
```

```
3 $sesion = new Sesion();
4 $sesion->finLogin();
5 header("location:index.php");
6 ?>
```

Listado 5.1.3. salida.php

```
1   <?php
2   require "clases/Sesion.php";
3   require "clases/MySQLdb.php";
4   require "clases/Albumes.php";
5   require "clases/Usuario.php";
6   require "clases/Estampa.php";
7   $sesion = new Sesion();
8   if($sesion->estadologin()==false){
9   header("location:login.php");
10  }
11  $albumes = new Albumes();
12  $estampas = new Estampa();
13  $usuarios = new Usuario();
14  //0 id, 1 nombre, 2 numestampas del album
15  $album = $albumes->leeAlbum();
16  $idUsuario = $usuarios->getIdUsuario($sesion->getUsuario());
17  //
18  if (isset($_POST["estampa"])) {
19  $estampa = isset($_POST["estampa"])?$_POST["estampa"]:"";
20  $estado = isset($_POST["estado"])?$_POST["estado"]:"";
21  $usuario = isset($_POST["usuario"])?$_POST["usuario"]:"";
22  $idAlbum = isset($_POST["album"])?$_POST["album"]:"";
23  if(!$estampas->actualizarEstampa($idAlbum, $usuario, $estampa, $estado)){
24  print "Error al actualizar la estampa"."<br>";
25  }
26  }
27  //
28  $estampasUsuario = $estampas->getEstampasUsuario($idUsuario,$album[0]);
29  if ($estampasUsuario==0) {
30  $albumes->creaAlbum($idUsuario, $album[0], $album[2]);
31  $estampasUsuario = $estampas->getEstampasUsuario($idUsuario,$album[0]);
32  }
33  $estampasUsuario_array = $estampas->getAlbumUsuario($idUsuario,$album[0]);
34  $estadisticas_array = $estampas->getEstadisticas($idUsuario,$album[0]);
35  function cadenaEstado($indice, $numEstampas,$idUsuario,$idAlbum){
36  global $estampasUsuario_array;
37  $edos = array("No la tengo", "Ya la tengo", "Repetida");
38  if($indice<$numEstampas){
39  print "<td>";
40  print armaEstampa($estampasUsuario_array[$indice]["numero"],$idAlbum,$idUsuario);
41  print "</td>";
42  if($estampasUsuario_array[$indice]["estado"]>=0 &&
43     $estampasUsuario_array[$indice]["estado"]<count($edos)){
44  print "<td class='";
45  if($estampasUsuario_array[$indice]["estado"]==0) print "nolatengo";
```

```
46  if($estampasUsuario_array[$indice]["estado"]==1) print "yalatengo";
47  if($estampasUsuario_array[$indice]["estado"]==2) print "repetida";
48  print "'>".$edos[$estampasUsuario_array[$indice]["estado"]]."</td>";
49  } else {
50  print "<td class='repetida'>".$estampasUsuario_array[$indice]["estado"]."</td>";
51  }
52  } else {
53  print "<td> </td>";
54  print "<td> </td>";
55  }
56  }
57  function armaEstampa($estampa, $album, $usuario){
58  $cadena = "<a href='inicio.php?e=".$estampa."&u=".$usuario."&a=".$album."'>";
59  $cadena .= $estampa;
60  $cadena .= "</a>";
61  return $cadena;
62  }
63  $busca = false;
64  if(isset($_GET["a"]) && isset($_GET["u"]) && isset($_GET["e"])){
65  $busca = true;
66  $a = $_GET["a"];
67  $u = $_GET["u"];
68  $e = $_GET["e"];
69  $quien = $estampas->busca($e, $u, $a);
70  }
71  ?>
72  <!DOCTYPE html>
73  <html>
74  <head>
75  <title>Indice</title>
76  <link rel="stylesheet" href="css/main.css">
77  </head>
78  <body>
79  <h1>Bienvendi@ <?php print $sesion->getUsuario(); ?></h1>
80  <h2>El album abierto es <b><i>'<?php print $album[1]; ?>'</i></b> con <?php print $alb
81  <form action="inicio.php" method="post">
82  <table border="1">
83  <tr >
84  <td><label for="estampa">Número de estampa:</label></td>
85  <td><input type="text" name="estampa"></td>
86  </tr>
87  <tr>
88  <td class="izq"><label for="estado" >Estado:</label></td>
89  <td class="izq">
90  <select name="estado" id="estado">
91  <option value="0">No la tengo</option>
92  <option value="1">Ya la tengo</option>
93  <option value="2">Repetida</option>
94  </select>
95  </td>
96  </tr>
97  <tr>
```

```
98  <td> 
99  <input type="hidden" name="usuario" value="<?php print $idUsuario; ?>"/>
100 <input type="hidden" name="album" value="<?php print $album[0]; ?>"/>
101 </td>
102 <td class="izq"><input type="submit" value="Enviar"/></td>
103 </tr>
104 </table>
105 </form>
106 <br>
107 <?php
108 print "<table border='1' width='100%'>";
109 print "<tr>";
110 print "<th>Estampa</th>";
111 print "<th>Estado</th>";
112 print "<th>Estampa</th>";
113 print "<th>Estado</th>";
114 print "<th>Estampa</th>";
115 print "<th>Estado</th>";
116 print "<th>Estampa</th>";
117 print "<th>Estado</th>";
118 print "<th>Estampa</th>";
119 print "<th>Estado</th>";
120 print "</tr>";
121 for ($i=0; $i < count($estampasUsuario_array); $i+=5) {
122 print "<tr>";
123 cadenaEstado($i,$estampasUsuario,$idUsuario, $album[0]);
124 cadenaEstado($i+1,$estampasUsuario,$idUsuario, $album[0]);
125 cadenaEstado($i+2,$estampasUsuario,$idUsuario, $album[0]);
126 cadenaEstado($i+3,$estampasUsuario,$idUsuario, $album[0]);
127 cadenaEstado($i+4,$estampasUsuario,$idUsuario, $album[0]);
128
129 print "</tr>";
130 }
131 print "</table>";
132 print "<br>";
133 if($busca){
134 print "<fieldset>";
135 if(count($quien)){
136 print "La estampa ".$e." la tiene:"."<br>";
137 for ($i=0; $i < count($quien); $i++) {
138 print $quien[$i]["correo"]."</br>";
139 }
140 } else {
141 print "Lo sentimos, nadie tiene repetida la estampa ".$e."<br>";
142 }
143 print "</fieldset>";
144 }
145 print "<br>";
146 print "<table border='1'>";
147 print "<tr>";
148 print "<th>Estado</th>";
149 print "<th>Número</th>";
```

```
150 print "<th>%</th>";
151 print "</tr>";
152 print "<tr class='nolatengo'>";
153 print "<td>No la tengo</td>";
154 print "<td>".$estadisticas_array[0]["COUNT(*)"]."</td>";
155 if(isset($estadisticas_array[0]["COUNT(*)"])){
156 print "<td>".number_format($estadisticas_array[0]["COUNT(*)"]/$estampasUsuario*100,2).
157 } else {
158 print "<td>0</td>";
159 }
160 print "</tr>";
161 print "<tr class='yalatengo'>";
162 print "<td>Ya la tengo</td>";
163 print "<td>".$estadisticas_array[1]["COUNT(*)"]."</td>";
164 if(isset($estadisticas_array[1]["COUNT(*)"])){
165 print "<td>".number_format($estadisticas_array[1]["COUNT(*)"]/$estampasUsuario*100,2).
166 } else {
167 print "<td>0</td>";
168 }
169 print "</tr>";
170 print "<tr class='repetida'>";
171 print "<td>Repetidas</td>";
172 print "<td>".$estadisticas_array[2]["COUNT(*)"]."</td>";
173 if(isset($estadisticas_array[2]["COUNT(*)"])){
174 print "<td>".number_format($estadisticas_array[2]["COUNT(*)"]/$estampasUsuario*100,2).
175 } else {
176 print "<td>0</td>";
177 }
178 print "</tr>";
179 print "</table>";
180 ?>
181 <br>
182 <a href="salida.php">Salir de la sesión</a>
183 </body>
184 </html>
```

Listado 5.1.4. inicio.php

5.2. Clases de la aplicación final

En este capítulo encontrarás los listados finales de las clases de la aplicación.

```
1  <?php
2  class MySQLdb{
3  private $host = "localhost";
4  private $usuario = "root";
5  private $clave = "root";
6  private $db = "albumes";
7  private $puerto = "3306"; //MAMP para windows
8  private $conn;
9
10 public function __construct(){
11 $this->conn = mysqli_connect(
12 $this->host,
13 $this->usuario,
14 $this->clave,
15 $this->db,
16 $this->puerto);
17 if(mysqli_connect_error()){
18 printf("Error en la conexión de la base de datos %d",mysqli_connect_error());
19 exit;
20 }
21 }
22
23 public function query($q){
24 $data = array();
25 if($q!=""){
26 if($r = mysqli_query($this->conn, $q)){
27 $data = mysqli_fetch_row($r);
28 }
29 }
30 return $data;
31 }
32
33 public function querySelect($q){
34 $data = array();
35 if($q!=""){
36 if($r = mysqli_query($this->conn, $q)){
37 while($row = mysqli_fetch_assoc($r)){
38 array_push($data,$row);
39 }
40 }
41 }
42 return $data;
43 }
44
45 public function queryNoSelect($q){
46 //insert, delete o update
```

```
47 $r;
48 if($q!=""){
49 $r = mysqli_query($this->conn, $q);
50 }
51 return $r;
52 }
53
54 public function close(){
55 mysqli_close($this->conn);
56 //print "Se cerro exitosamente la base de datos";
57 }
58 }
59 ?>
```

Listado 5.2.1. MySQLdb.php

```
1  <?php
2  class Albumes{
3  private $id;
4  private $nombre;
5  private $estampas;
6
7  public function __construct(){}
8
9  public function numAlbumes(){
10 $data = array();
11 $db = new MySQLdb();
12 $data = $db->query("SELECT count(*) FROM albumes");
13 $db->close();
14 unset($db);
15 return $data[0];
16 }
17
18 public function leeAlbum(){
19 $db = new MySQLdb();
20 $data = $db->query("SELECT * FROM albumes");
21 $db->close();
22 unset($db);
23 return $data;
24 }
25
26 public function creaAlbum($idUsuario, $idAlbum, $numEstampas){
27 //estado
28 //0 no la tengo
29 //1 ya la tengo
30 //2 o más repetido
31 $db = new MySQLdb();
32 for ($i=1; $i <= $numEstampas ; $i++) {
33 $sql = "INSERT INTO estampas VALUES(0,";
34 $sql .= $idAlbum.", ";
35 $sql .= $idUsuario.", ";
36 $sql .= "0, ";  // estado
```

```
37 $sql .= $i.")"; //num estampa en mi album
38 $r = $db->queryNoSelect($sql);
39 }
40 $db->close();
41 unset($db);
42 }
43 }
44 ?>
```

Listado 5.2.2. Albumes.php

```
1  <?php
2  class Estampa{
3  private $id;
4  private $album;
5  private $usuario;
6  private $numero;
7  private $estado;
8
9  public function __construct(){}
10
11 public function numEstampas(){
12 $data = array();
13 $db = new MySQLdb();
14 $data = $db->query("SELECT count(*) FROM estampas");
15 $db->close();
16 unset($db);
17 return $data[0];
18 }
19
20 public function getEstampasUsuario($idUsuario, $idAlbum){
21 $data = array();
22 $db = new MySQLdb();
23 $sql = "SELECT count(*) FROM estampas WHERE album=".$idAlbum;
24     $sql .= " AND usuario=".$idUsuario;
25 $data = $db->query($sql);
26 $db->close();
27 unset($db);
28 return $data[0];
29 }
30
31 public function getAlbumUsuario($idUsuario, $idAlbum){
32 $data = array();
33 $db = new MySQLdb();
34 $sql = "SELECT * FROM estampas WHERE album=".$idAlbum." AND usuario=".$idUsuario;
35 $data = $db->querySelect($sql);
36 $db->close();
37 unset($db);
38 return $data; // arreglo
39 }
40
41 public function actualizarEstampa($album, $usuario, $estampa, $estado){
```

```
42 $db = new MySQLdb();
43 $sql = "UPDATE estampas SET ";
44 $sql .= "estado=".$estado. " WHERE album=".$album;
45 $sql .= " AND usuario=".$usuario." AND ";
46 $sql .= "numero=".$estampa;
47 $r = $db->queryNoSelect($sql);
48 $db->close();
49 unset($db);
50 return$r;
51 }
52
53 public function getEstadisticas($usuario, $album){
54 $db = new MySQLdb();
55 $sql = "SELECT estado, COUNT(*) FROM estampas";
56 $sql .= " WHERE usuario=".$usuario." AND album=".$album;
57 $sql .= " GROUP BY estado ORDER BY estado";
58 $data = $db->querySelect($sql);
59 $db->close();
60 unset($db);
61 return$data;
62 }
63
64 public function busca($estampa, $usuario, $album){
65 $db = new MySQLdb();
66 $sql = "SELECT u.correo FROM estampas as e, usuarios AS u ";
67 $sql .="WHERE e.usuario = u.id AND e.album=".$album." AND ";
68 $sql .="e.usuario!=".$usuario." AND e.numero=".$estampa." AND ";
69 $sql .="e.estado=2";
70 $data = $db->querySelect($sql);
71 $db->close();
72 unset($db);
73 return $data;
74 }
75 }
76 ?>
```

Listado 5.2.3. Estampa.php

```
1  <?php
2  class Sesion{
3  private $login = false;
4  private $usuario;
5
6  function __construct(){
7  session_start();
8  $this->verificaLogin();
9  if ($this->login) {
10 # code...
11 } else {
12 # code...
13 }
14 }
```

```
15
16 private function verificaLogin(){
17 if (isset($_SESSION["usuario"])) {
18 $this->usuario = $_SESSION["usuario"];
19 $this->login = true;
20 } else {
21 unset($this->usuario);
22 $this->login = false;
23 }
24 }
25
26 public function inicioLogin($usuario){
27 if($usuario){
28 $this->usuario = $_SESSION["usuario"] = $usuario;
29 $this->login = true;
30 }
31 }
32
33 public function finLogin(){
34 unset($_SESSION["usuario"]);
35 unset($this->usuario);
36 $this->login = false;
37 }
38
39 public function estadoLogin(){
40 return $this->login;
41 }
42
43 public function getUsuario(){
44 return $this->usuario;
45 }
46 }
47 ?>
```

Listado 5.2.5. Sesion.php

5.3. Los estilos en cascada

En esta clase encontrarás el listado de los estilos en cascada

```
1  body{
2  width:800px;
3  margin:0 auto;
4  }
5  td{
6  text-align: center;
7  }
8  th{
9  background-color: #66ffff;
10 }
11 h1,h2{
12 text-align: center;
13 }
14 .yalatengo{
15 background-color: #00ff66;
16 }
17 .nolatengo{
18 background-color: #FFFFCC;
19 }
20 .repetida{
21 background-color: #FF9966;
22 }
```

Listado 5.3.1. main.css

Indice