Introducción a la programación orientada a objeto con PHP

La programación orientada a objetos te permitirá hacer proyectos con PHP de gran envergadura.

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

Indice

Capítulo 2: Fundamentos de la programación orientada a objetos ..... p. 1
2.1. Principios de herencia en PHP ..... p. 2
2.2. Modificadores de acceso ..... p. 4
2.3. Crear setters y getters ..... p. 6
2.4. Métodos mágicos _get y _set ..... p. 8
2.5. Métodos y propiedades estáticas ..... p. 10
2.6. Sobreescribir un método en una clase ..... p. 12
2.7. El operador de resolución de alcance (self, parent) ..... p. 14
2.8. Clonar objetos en PHP ..... p. 16
Indice ..... p. 18