6. OOP en PHP

1º DAW – Programación

David Soler Talens

Conceptos básicos (I)

- **Clase**

  Es un modelo que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad.

- **Objeto/instancia**

  Es una entidad provista de métodos o mensajes a los cuales responde (comportamiento); atributos con valores concretos (estado); y propiedades (identidad).

- **Método**

  Es el algoritmo asociado a un objeto que indica la capacidad de lo que éste puede hacer.

- **Propiedades/atributos**

  Son variables que contienen datos asociados a un objeto.

Conceptos básicos (II)

- **Abstracción**

  Aislación de un elemento de su contexto. Define las características esenciales de un objeto.

- **Encapsulamiento**

  Reúne al mismo nivel de abstracción, a todos los elementos que puedan considerarse pertenecientes a una misma entidad.

- **Modularidad**

  Característica que permite dividir una aplicación en varias partes más pequeñas (denominadas módulos),independientes unas de otras.

Conceptos básicos (III)

- **Ocultación (aislamiento)**

  Los objetos están aislados del exterior, protegiendo a sus propiedades para no ser modificadas por aquellos que no tengan derecho a acceder a las mismas.

- **Polimorfismo**

  Es la capacidad que da a diferentes objetos, la posibilidad de contar con métodos, propiedades y atributos de igual nombre, sin que los de un objeto interfieran con el de otro.

- **Herencia**

  Es la relación existente entre dos o más clases, donde una es la principal (madre) y otras son secundarias y dependen (heredan) de ellas (clases "hijas"), donde a la vez, los objetos heredan las características de los objetos de los cuales heredan.

Definición de clase PHP (I)

```php
<?php
class MyClass {
    // define una propiedad
    public $myProperty = 'valor de la propiedad';

    // define un método
    public function display() {
        echo $this->myProperty;
    }
}

$var = new MyClass();
$var->display();
?>
```

Definición de clase PHP (II)

- **palabra clave 'new'**

  Crea una instancia de una clase ▸ *new MyClass ();*

- **pseudo-variable $this**

  Hace eferencia al objeto actual, solo se usa dentro de la clase.

- **Operador de objeto ->**

  Se usa cuando se llama a un método o se accede a una propiedad en una instancia de objeto. También se usa con *$this*.

Herencia (I)

```php
<?php
class MyParentClass {
    protected $var = 'Daw2';

    public function display() {
        echo $this->var . ' desde la clase padre <br>';
    }
}
class MySubclass extends MyParentClass {
    public function getVar() {
        return $this->var.' desde la clase hija';
    }
}

$a = new MySubClass();
$a->display();
echo $a->getVar();
?>
```

Herencia (II)

- **La herencia es el proceso de extender una clase existente (clase padre) a una nueva clase (subclase) usando la palabra clave *'extends'*.**

- **Una subclase hereda todas las propiedades y métodos de su super-clase (principal) excepto las privadas.**

- **La herencia se usa para la reutilización de códigos y en el polimorfismo.**

- **PHP no permite herencia múltiple (como máximo una superclase) pero si herencia multinivel.**

- **Una clase declarada con la palabra clave *'final'* no se puede extender.**

Visibilidad (I)

```php
<?php
class MyClass {
  public $var1 = 'propiedad pública';
  protected $var2 = 'propiedad protegida';
  private $var3 = 'propiedad privada';

  function printHello() {
    echo $this->var1 . '<br>';
    echo $this->var2 . '<br>';
    echo $this->var3 . '<br>';
  }
}

$obj = new MyClass();
echo $obj->var1 . '<br>'; // muestra la propiedad pública
$obj->printHello(); // muestra todas las propiedades

echo $obj->var2; // Error Fatal
echo $obj->var3; // Error Fatal
```

Visibilidad (II)

- **determina cómo se puede acceder a las propiedades/métodos de un objeto:**
  - público: se puede acceder desde cualquier lugar.
  - protegido: solo puede acceder la clase y las subclases
  - privado: solo se puede acceder a la clase.
- **Una propiedad debe definirse con una de las palabras clave de visibilidad anteriores.**
- **Un método definido sin ninguno de ellos tendrá visibilidad pública por defecto.**

Constantes de clase(I)

```php
<?php
class MyClass {
    const PI = 3.14159;

    function showPI() {
        echo self::PI . "<br>";
    }
}

echo MyClass::PI . "<br>";
$class = new MyClass();
$class->showPI();
echo $class::PI . "<br>";
?>
```

Constantes (II)

- **La visibilidad predeterminada de las constantes de clase es pública.**
- **Las constantes se asignan una vez por clase, y no para cada instancia de clase.**
- **Operador de resolución de alcance (::)**

  En lugar de usar **->**, los dos puntos dobles permiten el acceso a propiedades estáticas y constantes. Este operador también se utiliza para acceder a características de superclase.

- **Usando 'self'**

  En lugar de usar '*$this'*, la palabra clave *'self'* se usa para acceder a las constantes dentro de la clase. Generalmente, para todos los accesos a nivel de clase se debe usar *'self'* y para todos los accesos de instancia de objeto, *'$this*' debe usarse dentro de la clase

Propiedades y métodos estáticos(I)

```php
<?php
class MyClass {
    static $var = 'propiedad estática';

    static function aMethod() {
        return self::$var;
    }
}

echo MyClass::$var . '<br>';
echo MyClass::aMethod();
?>
```

Propiedades y métodos estáticos(II)

- **La palabra clave ‘estatic’**

  convierte propiedades o métodos en características de nivel de clase y no necesitamos una instancia de clase para acceder a estas.

- **Para acceder a propiedades/métodos estáticos**
  - Con (::)
  - para acceder a ellos dentro de la clase utilizamos la palabra clave *‘self’.*
- **$this NO está disponible dentro de un método estático.**
- **Por defecto, las características estáticas tienen accesibilidad pública.**

Constructores y destructores (I)

```php
<?php
class MyClass {
    private $prop;

    function __construct($var) {
        echo 'Se ha creado la clase ' . __CLASS__ . '<br>';
        $this->prop = $var;
    }

    public function displayProp() {
        echo $this->prop .'<br>';
    }

    function __destruct() {
        echo 'Se ha destruido ' . __CLASS__;
    }
}

$a = new MyClass('Daw2');
$a->displayProp();
?>
```

Constructores y destructores (II)

- **__construct ()**
  - es una función especial (mágica) que se llama automáticamente cuando se crea una instancia de objeto con la palabra clave '*new*'.
  - Un constructor puede tener cualquier número de parámetros definidos por el usuario. Los constructores se usan para inicializar el objeto
  - El constructor de una super-clase puede ser llamado con *parent::__construct();*
- **__destruct ()**
  - se llama automáticamente cuando el recolector de basura elimina el objeto de la memoria.

Sobre-escritura de métodos(I)

```php
<?php
class A {
    function aMethod() {
        return "aMethod from A";
    }
}

class B extends A {
    function aMethod() {
            return "aMethod from B, ".
            parent::aMethod();
    }
}

$a = new A;
echo($a->aMethod());
echo('<br>');
$b = new B;
echo($b->aMethod());
?>
```

Sobre-escritura de métodos(II)

- **Es el proceso en el que una subclase redefine un método de clase principal para cambiar su comportamiento.**
  - La declaración debe ser exactamente la misma.
  - En caso de que queramos acceder a las funciones de nivel primario desde una subclase, utilizaremos *'parent::'*
- **Palabra clave 'final'**
  - Una subclase no puede sobre-escribir un método declarado con la palabra clave *'final'* en la super-clase.

Clases y métodos abstractos (I)

```php
<?php
abstract class A{
    abstract protected function aMethod();

    public function doSomething(){
        $this->aMethod();
    }
}
class B extends A{
    protected function aMethod(){
        echo 'aMethod ha sido llamado';
    }
}

$b = new B();
$b->doSomething();
?>
```

Clases y métodos abstractos (II)

- **Una clase abstracta no puede ser instanciada.**
  - Proporcionan implementación abstracta de clase que debe ser ampliada para proporcionar un comportamiento específico.
  - Una definición de clase abstracta comienza con una palabra clave *'abstract'.*
-
- **Los métodos abstractos son aquellos que se declaran inicialmente en una clase abstracta**
  - No se especifica el algoritmo que implementarán
  - No incluyen código
- **Una subclase debe sobre-escribir los métodos abstractos.**

Interfaces (I)

```php
<?php
interface Task {
  public function runTask();
  public function anotherMethod();
}

abstract class TaskImpl implements Task {
  public function runTask() {
    echo $this->anotherMethod();
  }
}

class TaskImpl2 extends TaskImpl {
  public function anotherMethod() {
    return "Hola Daw2";
  }
}

$task = new TaskImpl2();
$task->runTask();
?>
```

Interfaces(II)

- **La interfaz es otra forma de definir el tipo abstracto.**
- **Las interfaces definen métodos sin implementación.**
  - No tienen que usar palabras clave ‘abstract’.
  - Para definir una interfaz, usamos la palabra clave *‘interface’* en lugar de *‘class’*.
  - Todos los métodos en una interfaz deben ser públicos.
- **Clases que implementan Interfaces**
  - Una clase puede implementar una o más interfaces separadas por comas
  - Se usa palabra clave *‘implements’*.
  - La clase debe implementar todos los métodos de interfaz o, de lo contrario, debe declararse abstracta.