DESARROLLO DE SOFTWARE DE BACKEND I

ROMERO CARRASCO SAMANTHA BELEM

Aarón Hernández García

Actividad. POO

29/06/2024

Programación Orientada a Objetos en PHP

La Programación Orientada a Objetos (POO) en PHP es un paradigma que permite estructurar el código en objetos, los cuales encapsulan tanto datos como métodos para manipular esos datos. Este enfoque mejora la modularidad, reutilización y mantenimiento del código. A continuación, se detallan los conceptos principales de POO en PHP junto con referencias bibliográficas.

3.1 Los Conceptos OO

3.1.1 Variables: Locales, Globales, de Objeto y de Clase

- **Variables Locales**: Son aquellas definidas dentro de una función o método y solo pueden ser utilizadas dentro de ese contexto. Se destruyen una vez que la función o método finaliza su ejecución.
- **Variables Globales**: Se definen fuera de cualquier función o método y pueden ser accedidas desde cualquier parte del código utilizando la palabra clave global o el array superglobal $GLOBALS.
- **Variables de Objeto**: Se refieren a los atributos de una instancia específica de una clase. Estas variables son prefijadas con $this-> dentro de la clase para hacer referencia a ellas.
- **Variables de Clase**: Son atributos estáticos definidos con la palabra clave static dentro de una clase. Estas variables pueden ser accedidas sin necesidad de crear una instancia de la clase utilizando self::.

```
php
Copiar código
class Ejemplo {
   public $variableDeObjeto;
   public static $variableDeClase;

   public function __construct($valor) {
      $this->variableDeObjeto = $valor;
   }

   public static function metodoEstatico() {
      return self::$variableDeClase;
   }
}

$ejemplo = new Ejemplo('Hola');
echo $ejemplo->variableDeObjeto;
echo Ejemplo::$variableDeClase;
```

3.1.2 Herencia

La herencia permite que una clase (clase hija) herede propiedades y métodos de otra clase (clase padre). En PHP, se utiliza la palabra clave extends para indicar que una clase hereda de otra.

```
php
Copiar código
class ClasePadre {
   public $atributo;

   public function metodo() {
      echo "Método de la clase padre";
   }
}

class ClaseHija extends ClasePadre {
   public function metodoHija() {
      echo "Método de la clase hija";
   }
}

$obj = new ClaseHija();
$obj->metodo(); // Llama al método de la clase padre
$obj->metodoHija();
```

3.1.3 Sobrecargando Métodos

La sobrecarga de métodos no es soportada directamente en PHP como en otros lenguajes. Sin embargo, se puede simular utilizando argumentos opcionales o funciones mágicas como __call.

```
php
Copiar código
class Sobrecarga {
   public function __call($name, $arguments) {
      if ($name == 'metodo') {
         if (count($arguments) == 1) {
            return $this->metodo1($arguments[0]);
         } elseif (count($arguments) == 2) {
            return $this->metodo2($arguments[0], $arguments[1]);
         }
      }
   }

   private function metodo1($arg1) {
      return "Un argumento: $arg1";
   }

   private function metodo2($arg1, $arg2) {
      return "Dos argumentos: $arg1, $arg2";
   }
}
```

```
$obj = new Sobrecarga();
echo $obj->metodo('uno');
echo $obj->metodo('uno', 'dos');
```

3.1.4 Encapsulación

La encapsulación es el principio de ocultar los detalles internos de una clase y exponer solo lo necesario a través de métodos públicos. En PHP, se puede controlar la visibilidad de propiedades y métodos utilizando public, protected, y private.

```
php
Copiar código
class Encapsulacion {
   private $atributoPrivado;
   protected $atributoProtegido;
   public $atributoPublico;

   public function setPrivado($valor) {
      $this->atributoPrivado = $valor;
   }

   public function getPrivado() {
      return $this->atributoPrivado;
   }
}

$obj = new Encapsulacion();
$obj->setPrivado('valor');
echo $obj->getPrivado();
```

3.1.5 Polimorfismo

El polimorfismo permite que diferentes clases puedan ser tratadas a través de una misma interfaz. Esto se logra mediante la implementación de métodos comunes en diferentes clases o el uso de interfaces y clases abstractas.

```
php
Copiar código
interface Forma {
   public function area();
}

class Cuadrado implements Forma {
   private $lado;

   public function __construct($lado) {
```

```
        $this->lado = $lado;
    }

    public function area() {
        return $this->lado * $this->lado;
    }
}

class Circulo implements Forma {
    private $radio;

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

    public function area() {
        return pi() * $this->radio * $this->radio;
    }
}

$formas = [new Cuadrado(4), new Circulo(3)];

foreach ($formas as $forma) {
    echo $forma->area() . "\n";
}
```

Referencias Bibliográficas

- Lerdorf, R., & Tatroe, K. (2006). *Programming PHP* (3rd ed.). O'Reilly Media.
- Ullman, L. (2011). *PHP for the Web: Visual QuickStart Guide* (4th ed.). Peachpit Press.
- PHP Manual. (n.d.). OOP Basics. Recuperado de https://www.php.net/manual/en/language.oop5.basic.php
- PHP Manual. (n.d.). Visibility. Recuperado de https://www.php.net/manual/en/language.oop5.visibility.php
- PHP Manual. (n.d.). Inheritance. Recuperado de https://www.php.net/manual/en/language.oop5.inheritance.php
- PHP Manual. (n.d.). Interfaces. Recuperado de https://www.php.net/manual/en/language.oop5.interfaces.php
- Toptal. (n.d.). Understanding Polymorphism in PHP. Recuperado de https://www.toptal.com/php/understanding-polymorphism-in-php
- SitePoint. (n.d.). PHP OOP: Encapsulation. Recuperado de https://www.sitepoint.com/php-oop-encapsulation/