Extending Object Oriented Programming with PHP

Terminology

the single most important part

Terms

- Encapsulation
- Abstraction
  - Inheritance
- Polymorphism
  - Interface
  - Abstract Class
- Scope
  - Access Modifiers
- Namespaces

Encapsulation

Actions and data are bundled together in a way that restricts their access to the rest of the programming.

Example:

- function - actions with variables
- object - methods and properties

Abstraction

“An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of object and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.”
— G. Booch

This is the class architecture itself.

Inheritance: passes knowledge down

- Subclass, parent and a child relationship, allows for reusability, extensibility.
- Additional code to an existing class without modifying it. Uses keyword “extends”
- NUTSHELL: create a new class based on an existing class with more data, create new objects based on this class

Example: Child Adding and Overriding

```
class Developer extends User
{
  public $skills = array();  // additional property
  public function getSkillsString() {  // additional method
    return implode(", ", $this->skills);
  }
  public function getSalutation(): string {  // overriding method
    return $this->name . ", " . __CLASS__";
  }
}
```

Example: Using the Object Instance

```
$developer = new Developer("rasmus lerdorf", "mr");
echo $developer->getSalutation();
echo "<br />";
$developer->skills = array("JavasScript", "HTML", "CSS");
$developer->skills[] = "PHP";
echo $developer->getSkillsString();
```

Rasmus Lerdorf, Developer

JavasScript, HTML, CSS, PHP

Example: Extending a Parent Method

```
class Developer extends User
{ ...
  public $level = [
     " Unicorn ",
     " Rock Star ",
     " 10X "
  ];
  public function getSalutation() : string {  // extending method
     return parent::getSalutation() . ", "
     . array_rand(array_flip($this->level)) . __CLASS__;
  }
}
```

Example: Using the Object Instance

```
$developer = new Developer("rasmus lerdorf", "mr");
echo $developer->getSalutation();
```

When the script is run, it will return:

Rasmus Lerdorf, Unicorn Developer

Challenge

Extend the User class for another type of user, such as our Developer example

Polymorphism

Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface

Interface

- Interface, specifies which methods a class must implement.
- All methods in interface must be public.
- Multiple interfaces can be implemented by using comma separation
- Interface may contain a CONSTANT, but may not be overridden by implementing class

Example: UserInterface

```
interface UserInterface {
  public function getName();
  public function setName($name);
  public function getTitle();
  public function setTitle($title);
}
class User implements UserInterface { … }
```

Example: UserInterface with Type Definitions

```
interface UserInterface {
  public function getName(): string;
  public function setName(string $name);
  public function getTitle(): string;
  public function setTitle(?string $title);
}
class User implements UserInterface { … }
```

Abstract Class

- A mix between an interface and a class.
- Can define functionality as well as interface (in the form of abstract methods).
- Classes extending an abstract class must implement all of the abstract methods defined in the abstract class.

Example: Abstract Class

```
abstract class User implements UserInterface { //class
  ...
  abstract public function getSkillsString(): string;
}
class Developer extends User
{
  public function getSkillsString() {  // additional required from parent
    return implode(", ",$this->skills);
  }
}
```

Second Interface

```
interface WeightInterface {
  public function getWeight(): float;
}
class User implements UserInterface, WeightInterface
{...
  public $weight;
  public function setWeight(float $amount) {
    $this->weight = $amount;
  }
  public function getWeight(): float {
    return $this->weight;
  }
}
```

Challenges

Add an Interface

Change to User class to an abstract class.

Define an abstract method

Scope

Restricting access

Personal Scope

- Acquaintance (Social)
- Friends
- Professional
- Family
- Romantic

Access Modifiers

Controls who can access what. Restricting access to some of the object's components (properties and methods), preventing unauthorized access.

- Visibility
  - Public - everyone
  - Protected - inherited classes
  - Private - class itself, not children
- Keywords Final and Static
  - self
  - parent
  - static

Abstract Scope

```
abstract class AbstractScope {
  const CLASS_CONSTANT = 'abstract constant';
  public $myvar = 101;
}
```

Example: ClassScopeA extends AbstractScope

```
class ClassScopeA extends AbstractScope {
  const CLASS_CONSTANT = 'class constant';
  public $myvar = 100;
  public static $staticvar = 200;
  private $privatevar = 300;
  public function displayProperties() {
    echo 'parent constant: ' . parent::CLASS_CONSTANT .'<br />';
    echo 'self constant: ' . self::CLASS_CONSTANT . '<br />';
    echo 'static constant: ' . static::CLASS_CONSTANT .'<br />';
    echo '$this->privatevar: ' . '(' . gettype($this->privatevar) . ') '
      . $this->privatevar . '<br />';
  }
}
```

Example: ClassScopeB extends ClassScopeA

```
class ClassScopeB extends ClassScopeA {
  const CLASS_CONSTANT = 'classB constant';
  public $myvar = 1000;
  public static $staticvar = 2000;
  private $privatevar = 3000;
  final public function displayProperties()
  {
     parent::displayProperties();
  }
}
```

Example: Calling Object Scope

```
echo "STATIC CALL ClassScopeA" . '<br />';
echo 'ClassScopeA $staticvar: '  . ClassScopeA::$staticvar. '<br />';

$classA = new ClassScopeA();
echo "FROM classA Object" . '<br />';
var_dump($classA);
$classA->displayProperties();

echo "FROM classB Object" . '<br />';
$classB = new ClassScopeB();
var_dump($classB);
$classB->displayProperties();
```

```
STATIC CALL ClassScopeA
ClassScopeA $staticvar: 200

FROM classA Object
object(ClassScopeA)#2 (2) { ["myvar"]=> int(100)
["privatevar":"ClassScopeA":private]=> int(300) } parent constant:
abstract constant
self constant: class constant
static constant: class constant
$this->privatevar: (integer) 300

FROM classB Object
object(ClassScopeB)#3 (3) { ["myvar"]=> int(1000)
["privatevar":"ClassScopeB":private]=> int(3000)
["privatevar":"ClassScopeA":private]=> int(300) } parent constant:
abstract constant
self constant: class constant
static constant: classB constant
$this->privatevar: (integer) 300
```

Example: Access Modifiers for User

```
abstract class User implements UserInterface, WeightInterface { //class
  protected $name; //property
  protected $title = "Mx.";
  private $acceptedTitles = ["Mr.", "Ms.", "Mrs.", "Mx."];
  public static $encouragements =
    [ "You are beautiful!",  "You have this!", "Stop touching your face!"];
  protected $weight;

  private function getSalutation(): string {
    return "Hello " . $this->title . " " . $this->name;
  }
...
}
```

```php
class Developer extends User
{...
  public $skills = array();  // additional property
  public $title = "M.";  // more open not less
  protected $acceptedTitles = ["Mr.", "Ms.", "Mrs.", "Mx.", "M.", "Miss"];
  public $level = [
    " Unicorn ",
    " Rock Star ",
    " 10X "
  ];
  public function getSalutation() : string {  // extending method
    //return parent::getSalutation() . ", " //won't work when parent private
    return parent::__toString() . ", "
    . array_rand(array_flip($this->level))
    . __CLASS__;
  }
}
```

```
["title"]=>string(3) "Mr." // overrides same or more open
//if parent is private, you would ADD a second variable in the child
["acceptedTitles":protected]=> array(6) {
        [0]=>string(3) "Mr."
        [1]=>string(3) "Ms."
        [2]=>string(4) "Mrs."
        [3]=>string(3) "Mx."
        [4]=>string(2) "M."
        [5]=>string(4) "Miss"
}
["acceptedTitles":"User":private]=>array(4) {//
        [0]=>string(3) "Mr."
        [1]=>string(3) "Ms."
        [2]=>string(4) "Mrs."
        [3]=>string(3) "Mx."
}
```

Challenges

Lock access to your properties so they MUST use getters and setters

Throw an error because your access is too restricted.

Namespacing

Application / Package Scope

Namespaces

- Help create a new layer of code encapsulation
- Keep properties from colliding between areas of your code
- Only classes, interfaces, functions and constants are affected
- **Anything that does not have a namespace is considered in the Global namespace (namespace = "")**

Namespaces

- must be declared first (except 'declare')
- Can define multiple in the same file
- You can define that something be used in the "Global" namespace by enclosing a non-labeled namespace in {} brackets.
- Use namespaces from within other namespaces, along with aliasing

```php
namespace MidwestPHP;

class User implements \UserInterface {
  ... getName ... getTitle ... setTitle
  public function setName(string $name){
    $this->name = '';
    $words = explode(' ', $name);
    foreach ($words as $word) {
      $this->name .= strtoupper(substr($word, 0, 1));
    }
  }
  public function greet(): string {
    return "Hello " . __CLASS__ . " "
    . $this->title . " " . $this->name;
  }
}
```

Example: Autoloading Namespace

```php
spl_autoload_register(function ($class) {
  $file = 'classes/' . str_replace('\\', '/', $class) . '.php';
  if (file_exists($file)) {
    require $file;
  }
});
```

Example: Using a Namespaced Class

```
use \MidwestPHP\User; // namespace referenced first
$stowe = new User();  // used from specified namespace

$stowe = new \MidwestPHP\User();  // full namespace path
$stowe->setName("mike stowe");
echo $stowe->greet();
if ($stowe instanceof \User) {
  echo 'stowe is a User<br />';
}
if (is_a($stowe, 'UserInterface')) {
  echo 'stowe is a UserInterface<br />';
}
```

Example: Output

When the script is run, it will return:

Hello MidwestPHP\User Mx. MS
stowe is a UserInterface

Example: Global Namespaced Class

```
$user = new Developer("rasmus lerdorf", "mr");
echo $user;
echo "<br />\n";
if ($developer instanceof \User) {
  echo 'developer is a User<br />';
}
if (is_a($developer, 'UserInterface')) {
  echo 'developer is a UserInterface<br />';
}
```

Example: Output

When the script is run, it will return:

Hello Mr. Rasmus Lerdorf, 10X Developer
developer is a User
developer is a UserInterface

Questions and Answers

Thank you for your participation