Lab 5.   OOP in PHP

Prepared: TrangNTT
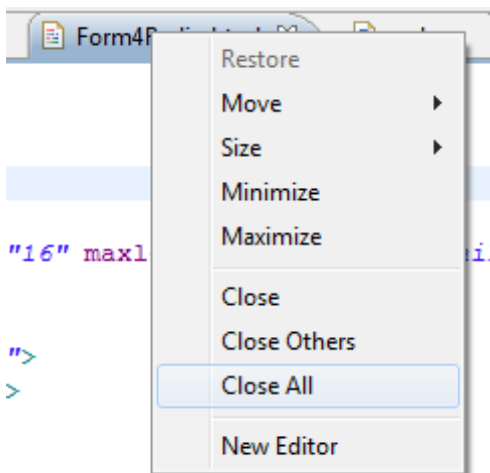
5.1. Open workspace and Create Folder

Step 1. Open Zend Studio 7.0

- Select Start → All Programs → Zend Studio – 7.0.0 → Zend Studio – 7.0.0

- Choose OK to confirm the workspace.
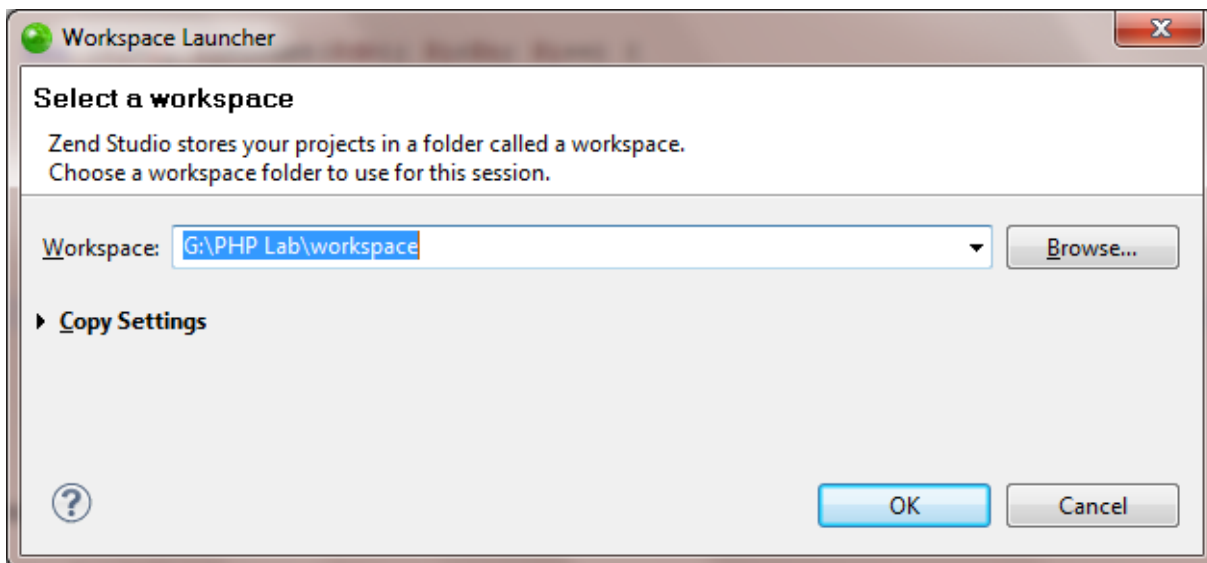
- Close all the opening file in the **PHP Editor** view by right click to a file and choose **Close All**.

Step 2. Right click on LabProject, choose New Folder. Enter Lab5 as folder name.

5.2. Constructor & Destructor

Step 1. Con_des.php

```php
1  <?php
2      class BaseClass {
3      protected $name = "BaseClass";
4          function __construct(){
5              print("In " . $this->name . " constructor<br>");
6          }
7          function __destruct(){
8              print("Destroying " . $this->name . "<br>");
9          }
10     }
11     class SubClass extends BaseClass {
12         function __construct(){
13             $this->name = "SubClass";
14             parent::__construct();
15         }
16         function __destruct(){
17             parent::__destruct();
18         }
19     }
20 ?>
```

Step 2. Add the following code to the end of con_des.php file:

$obj1 = new SubClass();

Run and give comments.

Step 3. Add the following code to the end of con_des.php file:

$obj2 = new BaseClass();

Run and give comments.

5.3. Counter – Static member

Step 1.   Counter.php

```php
1  <?php
2      class Counter  {
3          private static $count = 0;
4          const VERSION = 2.0;
5
6          function __construct()
7          {
8              self::$count++;
9          }
10
11         function __destruct()
12         {
13             self::$count--;
14         }
15
16         static function getCount()
17         {
18             return self::$count;
19         }
20     }
21
22     $c1 = new Counter;
23     print($c1->getCount() . "<br>\n");
24
25     $c2 = new Counter();
26     print(Counter::getCount() . "<br>\n");
27
28     $c2 = NULL;
29
30     print($c1->getCount() . "<br>\n");
31     print("Version used: " . Counter::VERSION . "<br>\n");
32 ?>
```

Step 2. Run

1
2
1
Version used: 2

5.4. Object Cloning

Step 1. ObjectCloning.php

```php
<?php
    class ObjectTracker {
        private static $nextSerial = 0;
        private $id, $name;

        function __construct($name) {
            $this->name = $name;
            $this->id = ++self::$nextSerial;
        }

        function __clone() {
            $this->name = "Clone of $this->name";
            $this->id = ++self::$nextSerial;
        }

        function getId() {
            return($this->id);
        }

        function getName() {
            return($this->name);
        }
        function setName($name) {
            $this->name = $name;
        }
    }

    $ot = new ObjectTracker("Zeev's Object");
    $ot2 = clone $ot; $ot2->setName("Another object");

    //1 Zeev's Object
    print($ot->getId() . " " . $ot->getName() . "<br>");

    //2 Clone of Zeev's Object
    print($ot2->getId() . " " . $ot2->getName() . "<br>");
?>
```

Step 2. Run

```
1 Zeev's Object
2 Another object
```

Step 3. Modify ObjectCloning.php

Do not use clone but using = operator to assign ot to ot2. Observe the result and give comments.

5.5. Overloading

Step 1. AttributeOverloading.php

```php
1  <?php
2  class PropertyTest {
3      /**  Location for overloaded data.  */
4      private $data = array();
5
6      /**  Overloading not used on declared properties.  */
7      public $declared = 1;
8
9      /**  Overloading only used on this when accessed outside the class.  */
10     private $hidden = 2;
11
12     public function __set($name, $value) {
13         echo "Setting '$name' to '$value'\n";
14         $this->data[$name] = $value;
15     }
16
17     public function __get($name) {
18         echo "Getting '$name'\n";
19         if (array_key_exists($name, $this->data)) {
20             return $this->data[$name];
21         }
22
23         $trace = debug_backtrace();
24         trigger_error(
25             'Undefined property via __get(): ' . $name .
26             ' in ' . $trace[0]['file'] .
27             ' on line ' . $trace[0]['line'],
28             E_USER_NOTICE);
29         return null;
30     }
31
32     /**  As of PHP 5.1.0  */
33     public function __isset($name) {
34         echo "Is '$name' set?\n";
35         return isset($this->data[$name]);
36     }
37
38     /**  As of PHP 5.1.0  */
39     public function __unset($name) {
40         echo "Unsetting '$name'\n";
41         unset($this->data[$name]);
42     }
43
44     /**  Not a magic method, just here for example.  */
45     public function getHidden() {
46         return $this->hidden;
47     }
48 }
49
50
51 echo "<pre>\n";
52
53 $obj = new PropertyTest;
54
55 $obj->a = 1;
56 echo $obj->a . "\n\n";
57
58 var_dump(isset($obj->a));
59 unset($obj->a);
60 var_dump(isset($obj->a));
61 echo "\n";
```

```
62
63  echo $obj->declared . "\n\n";
64
65  echo "Let's experiment with the private property named 'hidden':\n";
66  echo "Privates are visible inside the class, so __get() not used...\n";
67  echo $obj->getHidden() . "\n";
68  echo "Privates not visible outside of class, so __get() is used...\n";
69  echo $obj->hidden . "\n";
70  ?>
```

Step 2. Run

```
Setting 'a' to '1'
Getting 'a'
1

Is 'a' set?
bool(true)
Unsetting 'a'
Is 'a' set?
bool(false)

1

Let's experiment with the private property named 'hidden':
Privates are visible inside the class, so __get() not used...
2
Privates not visible outside of class, so __get() is used...
Getting 'hidden'
```

Step 3. MethodOverloading.php

```
1
2  <?php
3  class MethodTest {
4      public function __call($name, $arguments) {
5          // Note: value of $name is case sensitive.
6          echo "Calling object method '$name' "
7              . implode(', ', $arguments). "<br>";
8      }
9
10     /**  As of PHP 5.3.0  */
11     public static function __callStatic($name, $arguments) {
12          // Note: value of $name is case sensitive.
13          echo "Calling static method '$name' "
14              . implode(', ', $arguments). "<br>";
15     }
16 }
17
18 $obj = new MethodTest;
19 $obj->runTest('in object context');
20
21 MethodTest::runTest('in static context');   // As of PHP 5.3.0
22 ?>
```

Step 4. Run

Calling object method 'runTest' in object context
Calling static method 'runTest' in static context

5.6. Abstract methods and abstract classes

Step 1. Shape.php (New Class)

```php
1  <?php
2      //abstract root class
3      abstract class Shape
4      {
5          abstract function getArea();
6      }
7  ?>
```

Step 2. Polygon.php (New Class)

```php
1  <?php
2      include "Shape.php";
3      //abstract child class
4      abstract class Polygon extends Shape
5      {
6          abstract function getNumberOfSides();
7      }
```

Step 3. Triangle.php (New Class)

```php
3      require "Polygon.php";
4      //concrete class
5      class Triangle extends Polygon
6      {
7          public $base;
8          public $height;
9
10         public function getArea()
11         {
12             return(($this->base * $this->height)/2);
13         }
14
15         public function getNumberOfSides()
16         {
17             return(3);
18         }
19     }
```

Step 4. Rectangle.php (New Class)

```php
3      require "Polygon.php";
4      //concrete class
5      class Rectangle extends Polygon
6      {
7          public $width;
8          public $height;
9
10         public function getArea()
11         {
12             return($this->width * $this->height);
13         }
14
15         public function getNumberOfSides()
16         {
17             return(4);
18         }
19     }
```

Step 5. Circle.php (New Class)

```php
3      require "Shape.php";
4      //concrete class
5      class Circle extends Shape
6      {
7          public $radius;
8
9          public function getArea()
10         {
11             return(pi() * $this->radius * $this->radius);
12         }
13     }
```

Step 6. Color.php

```php
3      //concrete root class
4      class Color
5      {
6          public $name;
7      }
```

Step 7. Test_Shape.php

```php
1 <?php
2
3      $myCollection = array();
4
5      //make a rectangle
6      $r = new Rectangle;
7      $r->width = 5;
8      $r->height = 7;
9      $myCollection[] = $r;
10     unset($r);
11
12     //make a triangle
13     $t = new Triangle;
14     $t->base = 4;
15     $t->height = 5;
16     $myCollection[] = $t;
17     unset($t);
18
19     //make a circle
20     $c = new Circle;
21     $c->radius = 3;
22     $myCollection[] = $c;
23     unset($c);
24
25     //make a color
26     $c = new Color;
27     $c->name = "blue";
28     $myCollection[] = $c;
29     unset($c);
30
```

```
31        foreach ($myCollection as $s)
32        {
33            if ($s instanceof Shape)
34            {
35                print ("Area: " . $s->getArea() .
36                    "<br>\n");
37            }
38
39            if ($s instanceof Polygon)
40            {
41                print ("Sides: " .
42                    $s->getNumberOfSides() .
43                    "<br>\n");
44            }
45
46            if ($s instanceof Color)
47            {
48                print ("Color: $s->name<br>\n");
49            }
50
51            print ("<br>\n");
52        }
53
54 ?>
```

Step 8. Add autoloading class scripts at the beginning of Test_Shape.php so that when runnning Test_Shape.php; it gives the following result:

Area: 35
Sides: 4

Area: 10
Sides: 3

Area: 28.274333882308

Color: blue

Step 9. Try to use namespace for each class.

5.7. ClassIntrospection.php

```
1 <?php
2 function display_classes (  ) {
3   $classes = get_declared_classes(  );
4   foreach ($classes as $class) {
5     echo "Showing information about $class<br />";
6
7     echo "$class methods:<br />";
8     $methods = get_class_methods($class);
9     if (!count($methods)) {
10      echo "<i>None</i><br />";
11    }
12    else {
13      foreach ($methods as $method) {
14        echo "<b>$method</b>(  )<br />";
15      }
16    }
17
```

```
18      echo "$class properties:<br />";
19      $properties = get_class_vars($class);
20      if(!count($properties)) {
21        echo "<i>None</i><br />";
22      }
23      else {
24        foreach(array_keys($properties) as $property) {
25          echo "<b>\$$property</b><br />";
26        }
27      }
28
29      echo "<br />";
30    }
31 }
32
33 display_classes();
34
35 ?>
```

Showing information about stdClass
stdClass methods:
None
stdClass properties:
None

Showing information about Exception
Exception methods:
__construct( )
getMessage( )
getCode( )
getFile( )
getLine( )
getTrace( )
getPrevious( )
getTraceAsString( )
__toString( )
Exception properties:
None

Showing information about ErrorException
ErrorException methods:
__construct( )
getSeverity( )
getMessage( )
getCode( )
getFile( )
getLine( )
getTrace( )
getPrevious( )
getTraceAsString( )

Done

5.8. ObjectInstrospection.php

```php
1  <?php
2  // return an array of callable methods (include inherited methods)
3  function get_methods($object) {
4    $methods = get_class_methods(get_class($object));
5
6    if(get_parent_class($object)) {
7      $parent_methods = get_class_methods(get_parent_class($object));
8      $methods = array_diff($methods, $parent_methods);
9    }
10
11   return $methods;
12 }
13
14 // return an array of inherited methods
15 function get_inherited_methods($object) {
16   $methods = get_class_methods(get_class($object));
17
18   if(get_parent_class($object)) {
19     $parent_methods = get_class_methods(get_parent_class($object));
20     $methods = array_intersect($methods, $parent_methods);
21   }
22
23   return $methods;
24 }
25
26 // return an array of superclasses
27 function get_lineage($object) {
28   if(get_parent_class($object)) {
29     $parent = get_parent_class($object);
30     $parent_object = new $parent;
31
32     $lineage = get_lineage($parent_object);
33     $lineage[] = get_class($object);
34   }
35   else {
36     $lineage = array(get_class($object));
37   }
38
39   return $lineage;
40 }
41
42 // return an array of subclasses
43 function get_child_classes($object) {
44   $classes = get_declared_classes(  );
45
46   $children = array(  );
47   foreach($classes as $class) {
48     if (substr($class, 0, 2) == '_ _') {
49         continue;
50     }
51     if(get_parent_class($class) == get_class($object)) {
52       $children[] = $class;
53     }
54   }
55
56   return $children;
57 }
```

```php
58
59  // display information on an object
60  function print_object_info($object) {
61    $class = get_class($object);
62    echo '<h2>Class</h2>';
63    echo "<p>$class</p>";
64
65    echo '<h2>Inheritance</h2>';
66
67    echo '<h3>Parents</h3>';
68    $lineage = get_lineage($object);
69    array_pop($lineage);
70    echo count($lineage) ? ('<p>' . join(' -&gt; ', $lineage) . '</p>')
71                          : '<i>None</i>';
72
73    echo '<h3>Children</h3>';
74    $children = get_child_classes($object);
75    echo '<p>' . (count($children) ? join(', ', $children)
76                                   : '<i>None</i>') . '</p>';
77
78    echo '<h2>Methods</h2>';
79    $methods = get_class_methods($class);
80    $object_methods = get_methods($object);
81    if(!count($methods)) {
82      echo "<i>None</i><br />";
83    }
84    else {
85      echo '<p>Inherited methods are in <i>italics</i>.</p>';
86      foreach($methods as $method) {
87      echo in_array($method, $object_methods) ? "<b>$method</b>(  );<br />"
88                                              : "<i>$method</i>(  );<br />";
89      }
90    }
91
92    echo '<h2>Properties</h2>';
93    $properties = get_class_vars($class);
94    if(!count($properties)) {
95      echo "<i>None</i><br />";
96    }
97    else {
98      foreach(array_keys($properties) as $property) {
99        echo "<b>\$$property</b> = " . $object->$property . '<br />';
100     }
101   }
102
103   echo '<br />';
104 }
105
106     class A {
107       var $foo = 'foo';
108       var $bar = 'bar';
109       var $baz = 17.0;
110
111       function first_function(  ) { }
112       function second_function(  ) { }
113     };
114
115     class B extends A {
116       var $quux = false;
117
118       function third_function(  ) { }
119     };
120
121     class C extends B {
122     };
```

```
124        $a = new A;
125        $a->foo = 'sylvie';
126        $a->bar = 23;
127
128        $b = new B;
129        $b->foo = 'bruno';
130        $b->quux = true;
131
132        $c = new C;
133
134        print_object_info($a);
135        print_object_info($b);
136        print_object_info($c);
137  ?>
```

Result:

Class

A

Inheritance

Parents

None

Children

B

Methods

Inherited methods are in *italics*.

first_function( );
second_function( );

Properties

$foo = sylvie
$bar = 23
$baz = 17

Class

5.9. Exercise

Represented as a UML diagram, the Page class is shown in the following figure:

```
┌─────────────────────────────────────┐
│                Page                  │
├─────────────────────────────────────┤
│ -page : string                       │
│ -title : string                      │
│ -year : int                          │
│ -copyright : string                  │
├─────────────────────────────────────┤
│ -addHeader() : void                  │
│ +addContent(in content : string) : void │
│ -addFooter() : void                  │
│ +get() : string                      │
└─────────────────────────────────────┘
```

Write classes and web pages to generate several different pages and save/display them in a web browser.