Web Development Server-Side

Exercises - Object Oriented (OO) PHP

Computer Work	***Practical Computer Exercises***

NOTE: For most exercise your workflow should be as follows:

- Create a folder for each exercise
- In the exercise folder
  - have a folder named **'public'** containing an **index.php** page,
  - and another folder '**src'** containing your classes
- Name each class's file with a Capital first letter and ending **.php**

1.1 Practical Exercise

Create a new class **Person**, in file **Person.php** in folder **src**/. This class should have a single public property **$name**.

> *NOTE: We are using a **public** property, just this once, to make it really quick and easy to learn how to declare a class in PHP – I'm sure you know that you should have private properties and public 'getter' methods when following good coding practice … which we'll do in the exercise after this one ….*

In folde `public` write a file **index.php** which does the following:

- The Person class declaration file is read in and processed (required)
- Two objects of class Person are created, **$person1** and **$person2**
- The name of **$person1** is set to 'matt', and **$person2** to 'joelle'
- You print out the name of **$person1** and **$person2** on different lines

HINT (in: /src/**Person.php**):

```
class Person
{
     public $name;
}
```

HINT (in: /**index.php**):

```
print 'Name of person1 is: ' . $person1->name;
print PHP_EOL;
```

1.2 Practical Exercise

Copy the folder for the previous exercise, and make the following changes:

- Make property $name **private**
- Add a **public** function **setName()** that takes as input one argument (the string new name)
- Add a **public** function **getName()** that returns the name of the object
- Update **index.php** to make use of these public functions

HINT (in: /src/**Person.php**):

```
class Person
{
	private $name;

	public function setName($name)
	{
		$this->name = $name;
	}
```

1.3 Practical Exercise

Copy the folder for the previous exercise, and make the following changes:

- Add a public constructor function that takes one argument (the string new name) and assigns the given value to variable **$name**
- Update **index.php** to make use of this public constructor, rather than using setName(…)

HINT (in: /src/**Person.php**):

```
public function __construct(
```

1.4 Practical Exercise

Copy the folder for the previous exercise, and make the following changes:

- Namespace your class into the namespace: **Itb**
- Set **composer.json** PSR-4 settings for autoload
- Make Composer regenerate its autoloader
- In **index.php** require the ../**vendor/autoload.php** script
- In **index.php** add a 'use' statement so that class Person refers to namespaced **Itb\Person**

HINT:

```
{
  "autoload": {
    "psr-4": {
      "Itb\\": "src"
    }
  }
}
```

HINT:

```
composer dump-autoload
```

or

```
composer dumpautoload
```

FROM THIS POINT ONWARDS PLEASE WRITE EVERYTHING TO WORK WITH PSR-4 autoloading and Composer.

Stick with namespace `Itb`

1.5 Practical Exercise

In the next few exercises we'll create some classes to allow us to work with students, modules and results

Create the following class in folder **src**/:

- Class **Student.php**
  - fields
    - id
    - firstName
    - String surname
    - String phoneNumber
  - Constructor takes as input:
    - Integer id
    - String firstName
    - String surname
    - String phoneNumber
  - Write a getter function for ID
  - Write getters and setter functions for the other fields

Write a file **index.php**, which does the following:

- Two objects of class Student are created, $student1 and $student2
  - Student1: id=1, name=matt smith, phone number = 007
  - Student2: id=2, name=joelle murphy, phone number = 321
- Display the contents of each object using **var_dump() or print_r()**

```
student 1
Itb\Student Object
(
    [id:Itb\Student:private] => 1
    [firstName: Itb \Student:private] => matt
    [surname: Itb \Student:private] => smith
    [phoneNumber: Itb \Student:private] => 007
)
--------------
student 2
Itb\Student Object
(
    [id: Itb \Student:private] => 2
    [firstName: Itb \Student:private] => joelle
    [surname: Itb \Student:private] => murphy
    [phoneNumber: Itb \Student:private] => 321
)
```

HINT:

```
print 'student 1';
print PHP_EOL;
var_dump($student1);
print PHP_EOL . '-------------' . PHP_EOL;
```

1.6 Practical Exercise

Create the following class in folder **src/**:

- Class **Module.php**
  - fields
    - id
    - title
    - moduleCode
  - Constructor takes as input:
    - Integer id
    - String title
    - String moduleCode
  - Write a getter function for ID
  - Write getters and setter functions for the other fields

Write a file **index.php**, which does the following:

- Three objects of class **Module** are created, $module1, $module2 and $module3
  - Module1: id=1, title=web-deb-1, module code = COMPH6030
  - Module2: id=2, title=web-deb-2, module code = COMPH6033
  - Module3: id=3, title=web-deb-3, module code = COMPH6034

- Then print the name and display the contents of each object with **var_dump()**

```
Itb \Module Object
(
    [id: Itb \Module:private] => 1
    [title: Itb \Module:private] => COMP6030
    [moduleCode: Itb \Module:private] => COMP6030
)
Itb \Module Object
(
    [id: Itb \Module:private] => 2
    [title: Itb \Module:private] => COMP6033
    [moduleCode: Itb \Module:private] => COMP6033
)
Itb \Module Object
(
    [id: Itb \Module:private] => 3
    [title: Itb \Module:private] => COMP6034
    [moduleCode: Itb \Module:private] => COMP6034
)
```

HINT:

```
var_dump($module1);
```

1.7 Practical Exercise

Let's create a ModuleRepository class, that will create and manage an array of module objects.

Create the following class in folder **src**/:

- Class **ModuleRepository.php**
  - fields
    - private: modules (array)
  - public: a **constructor** method (no arguments)
    - Initialises **modules** to an empty array
    - Creates module1, 2 and 3 as per previous exercise, and adds them to the array
      - Indexed by each modules 'id'
  - public: method **getAll()**
    - Returns the array of module objects

HINT:

```
class ModuleRepository
{
    private $modules;

    public function __construct()
    {
        // create 3 module objects
        $id1 = 1;
        $module1 = new Module($id1, 'web dev 1', 'COMP6030');

          ...

        // add each module object to the array - indexed by
the module ID
        $this->modules[$id1] = $module1;
```

1.8 Practical Exercise

Now let's use the ModuleRepository from the previous exercise.

Let's display the size of the array, and all module details using **var_dump()** and a **foreach{}** loop:

- In file **index.php** do the following:
  - (first line is always to require the Composer autoloader…)
  - create a new ModuleRepository object
  - call **getAll()** and store this array in a variable, such as **$modules**
  - Display the number of items in array **$moduleArray**, in the form "length of array = <n>",
    and then loop through the array (use a PHP foreach) displaying some minus signs, and then the output of **var_dump()** for each module:

```
Terminal
Matthews-MacBook-Air-2:public matt$ php main.php
length of array = 3
-------------
/Users/matt/Library/Mobile Documents/com~apple~Clo
class Itb\Module#3 (3) {
  private $id =>
  int(1)
  private $title =>
  string(9) "web dev 1"
  private $moduleCode =>
  string(8) "COMP6030"
}

-------------
/Users/matt/Library/Mobile Documents/com~apple~Clo
class Itb\Module#4 (3) {
  private $id =>
  int(2)
  private $title =>
  string(9) "web dev 2"
  private $moduleCode =>
  string(8) "COMP6033"
}

-------------
/Users/matt/Library/Mobile Documents/com~apple~Clo
class Itb\Module#5 (3) {
  private $id =>
  int(8)
  private $title =>
  string(9) "web dev 3"
  private $moduleCode =>
  string(8) "COMP6034"
}
```

HINT:

```
$numModules = sizeof($modules);
print 'length of array = ' . $numModules;

foreach($modules as $module){
    print PHP_EOL . '-------------' . PHP_EOL;
    var_dump($module);
```

1.9 Practical Exercise

Rather than displaying the objects with **var_dump()** let's output nicer text by doing the following:

- In class **Module.php** do the following:
  - Write the code for a **`__toString()`** 'magic method' that outputs the details of the current object's id, title and moduleCode as follows:

```
id = 1
title = web dev 1
code = COMP6030
```

- In file **index.php** do the following:
  - Same as previous exercise, first print out a summary of the number of items in the array, then loop through using foreach(), simply using a **print** statement for each Module object. Since **print** expects a string expression, then the objects **`__toString()`** method should be called automatically, and output the objects properties as a string:

```
length of array = 3
-------------
id = 1
title = web dev 1
code = COMP6030
-------------
id = 2
title = web dev 2
code = COMP6033
-------------
id = 8
title = web dev 3
code = COMP6034
-------------
```

HINT:

```
public function __toString()
{
    $output = '';
    $output .= 'id = ' . $this->id;
    $output .= PHP_EOL;
    ...
    return $output;
}
```

HINT:

```
foreach($modules as $module){
    print $module;
}
```

1.10 Practical Exercise

Add to your ModuleRepository class, so that we can search for an object given an ID.

(As you can see, our ModuleRepository is going to play the intermediary between our web application code and our database, when we eventually get around to PDO DB connectivity…)

Add the following to class **ModuleRepository** in folder **src/**:

- public method **getOne($id)**
  - takes one argument, the ID of the object to try to retrieve from the repository
  - returns the Module object (if in the repo) or **null** (if not in the repo)

TEST your new method – in file **index.php** do the following:

- Attempt to retrieve a module from the repository with the ID of 1
- Either display it (using the Module __toString() method), or display a message stating that no such module could be found
- Repeat the code, but for an ID of 100 (which should not exist yet …)

```
/public/$ php index.php

id = 1
title = web dev 1
code = COMP6030
-------------
no module found with ID = 100
```

HINT:

You could test for whether an array contains an item mapped to a key using either **isset()** or **array_key_exists()**

e.g.

```
$isInArray = isset($this->modules[$id]);
```

HINT:

```
public function getOne($id)
{
    $isInArray = array_key_exists($id, $this->modules);

    if ($isInArray) {
       return $this->modules[$id];
```

HINT:

```
$moduleWithId1 = $moduleRepository->getOne(1);

if (null != $moduleWithId1){
    print $moduleWithId1;
} else {
    print 'no module found with ID = 1';
}
```