PHP + OOP

Costantino Pistagna
pistagna@dmi.unict.it

How is an Object Created from a Class?

- The process of creating an object from the class 'blueprint' is called instantiation.
    - Essentially, you instantiate an instance of the class and give that instance a name by which you will refer to it when accessing members and calling methods.
- You can create as many object instances of a class as you desire.
- Objects are instantiated using the new keyword.

```
$accountObject = new bankAccount();
```

What is sub-classing?

- It is possible to build classes that are derived from other classes
  - extending the functionality of the parent class to make it specific to a particular requirement
- For example you might have a vehicle class which contains the attributes common to all vehicles
  - and a subclass called car which inherits all the generic vehicle attributes
  - but adds some its own car specific methods and properties.

Defining a PHP Class

- Before an object can be instantiated we first need to define the class 'blueprint' for the object.
- Classes are defined using the class keyword followed by braces which will be used to enclose the body of the class

```
<?php
class bankAccount {
}
?>
```

- We have now defined a class.
- The next step is add some functionality to the class.

PHP Class Constructors and Destructors

- The next step in creating a class is to define what should happen when an object is first instantiated using the class
    - and also when that object is later destroyed.
- These actions are defined in the constructor and destructor methods of the class.
- The constructor and destructor are really just functions that get called when the object is created and destroyed
    - are defined in the class body using the function keyword.
    - This needs to be prefixed with the public qualifier.
    - This means the method is accessible to code outside of the object.
- The default names for the constructor and destructor are __construct and __destruct respectively.

PHP Class Constructors and Destructors

- We can now extend our bankAccount class to include a constructor and destructor:

```
<?php
class bankAccount {

   public function __construct($accountNumber, $accountName) {
         echo “Object was just instantiated.”;
          echo “Number = $accountNumber, Name = $accountName <br>”;
   }

   public function __destruct() {
         echo “Object was just destroyed <br>”;
   }

}
?>
```

Creating Members in a PHP Class

- Class members are essentially variables and methods embedded into the class.
  - Members can be public or private and static or variable.
- public members can be accessed from outside the object.
- private members can only be accessed by methods contained in the class.
  - This is the key to what is called data encapsulation.
- Object-oriented programming convention dictates that data should be encapsulated in the class and accessed and set only through the methods of the class
  - typically called getters and setters

Creating Members in a PHP Class

- Members declared as static are immutable
  - once defined they cannot be changed (much like constants)
- Members and functions are prefixed with public, private and static when declared in the class.
  - The default is public non-static.

Creating Members in a PHP Class

- We can now **extend** out **bankAccount** class to add member variables to hold the account name and number passed into the constructor.
- True to the concept of data encapsulation we will be creating methods to access these values later
  - so will mark them as **private**.
- We will also add to our constructor to assign the passed arguments to our new members.
  - When doing so we need to use the **$this** variable to tell PHP we are setting variables in the current class

Creating Members in a PHP Class

```
<?php
class bankAccount {

   private $accountNumber;
   private $accountname;

   public function __construct($acctNumber, $acctName) {
        $this->accountNumber = $acctNumber;
        $this->accountname = $acctName;
   }

   public function __destruct() {
         echo 'Object was just destroyed <br>';
   }

}
        $myObj = new bankAccount('123456', 'Giuseppe Rossi');
?>
```

Defining and Calling Methods

- We define our own methods in much the same way as we declared the constructor and destructor
  - with exception that we get to choose the names

```
...
public function setAccountNumber($acctNumber)
   {
       $this->accountNumber = $acctNumber;
   }

   public function setAccountName($acctName)
   {
       $this->accountName = $acctName;
   }

   public function getAccountName()
   {
       return $this->accountName;
   }

   public function getAccountNumber()
   {
       return $this->accountNumber;
   }

...
```

Defining and Calling Methods

- Now that we have defined our getter and setter methods to get and set the account values we can call the methods.
  - This is done by specifying the name of the object on which the methods are being called.
  - This is followed by '**->**', and then the name of the method we are calling

```
...
$myObj = new bankAccount('123456', 'Giuseppe Verdi');
$myObj->setAccountNumber('654321');
$accountNumber = $myObj->getAccountNumber();
echo "New Account Number is $accountNumber";
...
```

Subclassing in PHP

- Once a class has been defined it is possible to create a new class **derived** from it
  - that **extends** the functionality of the original class
- The **parent** class is called the **superclass**
  - the **child** the **subclass**
- The whole process is referred to as **subclassing**.
- A subclass of a class can be defined using the **extends** keyword when declaring the subclass

```
<?php
class savingsAccount extends bankAccount {

        private $interestRate = 5;

}
```

Subclassing in PHP

```
<?php
class savingsAccount extends bankAccount {

        private $interestRate = 5;

}
```

- The important point to note here is that savingsAccount inherits all the members and methods of bankAccount
  - and adds a new member (the interest rate)

Subclassing in PHP

- We can extend the class further by adding a new method to return the interest rate:

```
class savingsAccount extends bankAccount {

   private $interestRate = 5;

   public function getInterestRate()
   {
       return $this->interestRate;
   }

}
```

PHP Object Serialization

- **Serialization** is the ability to take a snapshot of the current state of an object and then save that object to a file
  - or even transmit it over a network to another process where it will be re-activated
- All objects have built-in method called __**sleep** that is called **before** serialization.
  - If you need your object to perform any housekeeping before being serialized you will need to override this method.
- An object is serialized using the **serialize()** function
  - and unserialized, using the **unserialize()** function.

PHP Object Serialization

- As an example we can serialize our bankAccount object:

```
$myObj = new bankAccount('246810', 'Mario Rossi');

$serialized = serialize ($myObj);

echo 'Object is serialized<br>';

$newObj = unserialize ($serialized);

echo 'Object is unserialized<br>';

print_r ($newObj);
```

- Once we have the serialized data in our $serialized object
  - we can do anything we want with it
  - write it to a file or send it through a network socket to another process