PITECHNOLOGIES GROUP

# WEB DEVELOPMENT USING OPEN SOURCE TECHNOLOGIES

Lab Guide No. 9

## Table of Contents

# Introduction

## What is OOP?

- Object-oriented programming is a style of coding that allows developers to group similar tasks into classes.
- This helps keep code following the tenet “don’t repeat yourself” (DRY) and easy-to-maintain.

## What does OOP aim to achieve?

- Allow compartmentalized refactoring of code.
- Promote code re-use.
- Promote extensibility, flexibility and adaptability.
- Better for team development.
- Many patterns are designed for OOP.
- Some patterns lead to much more efficient code.

## What are the features of OOP?

### Encapsulation

- Encapsulation is about grouping of functionality (operations) and related data (attributes) together into a coherent data structure (classes).
- Classes represent complex data types and the operations that act on them. An object is a particular instance of a class.

### Access Control Modifiers

- **Public**: This property or method can be used from anywhere in the script
- **Private**: This property or method can be used only by the class or object it is part of; it cannot be accessed elsewhere
- **Protected**: This property or method can be used only by code in the class it is part of, or by descendants of that class
- **Final**: This property, method, or class cannot be overridden in subclasses
- **Abstract**: This method or class cannot be used directly you have to subclass this

### Inheritance

- Inheritance allows a class to specialize (or extend) another class and inherit all its methods, properties and behaviors.
- **This promotes**
    - Extensibility
    - Reusability
    - Code Consolidation
    - Abstraction
    - Responsibility

## Polymorphism

- Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.
- A real world analogy for polymorphism is a button. Everyone knows how to use a button: you simply apply pressure to it. What a button "does," however, depends on what it is connected to and the context in which it is used — but the result does not affect how it is used.
- In the programming world, polymorphism is used to make applications more modular and extensible. Instead of messy conditional statements describing different courses of action, you create interchangeable objects that you select based on your needs. That is the basic goal of polymorphism.

**Example: Main Class**

```
class Humans{
        public function __construct($name) {
                /*...*/
        }
        public function eat() { /*...*/}
        public function sleep() { /*...*/}
        public function snore() { /*...*/}
        public function wakeup() { /*...*/}
}
```

**Example: Simple Inheritance**

```
class Humans{
        public function __construct($name) { /*...*/}
        public function eat() { /*...*/ }
        public function sleep() { /*...*/ }
        public function snore() { /*...*/ }
        public function wakeup() { /*...*/ }
}
class Women extends Humans{
        public function giveBirth() { /*...*/ }
}
```

**Example: Inheritance & Polymorphism**

```
class Humans{
        public function __construct($name) { /*...*/}
        public function eat() { /*...*/ }
        public function sleep() { /*...*/ }
        public function snore() { /*...*/}
        public function wakeup() { /*...*/ }
}
class Women extends Humans{
        public function giveBirth() { /*...*/ }
}
class Men extends Humans{
        public function snore() { /*...*/}
}
```

**Example: Abstraction**

```
abstract class Humans{
        public function __construct($name) { /*...*/}
        abstract public function gender();
        public function eat() { /*...*/ }
        public function sleep() { /*...*/ }
        public function wakeup() { /*...*/ }
}
class Women extends Humans{
        public function gender() { return 'female'; }
        public function giveBirth() { /*...*/ }
}
class Men extends Humans{
        public function gender() { return 'male'; }
        public function snore() { /*...*/ }
}
```

## Structuring Classes

- The syntax to create a class is pretty straightforward: declare a class using the class keyword, followed by the name of the class and a set of curly braces ({}):

```
class MyClass{
        // Class properties and methods go here
}
```

- After creating the class, a new class can be instantiated and stored in a variable using the new keyword:

```
$obj = new MyClass;
```

## Defining Class Properties

- To add data to a class, properties, or class-specific variables, are used. These work exactly like regular variables, except they're bound to the object and therefore can only be accessed using the object.

```
<?php
class MyClass
{
   public $prop1 = "I'm a class property!";
}
$obj = new MyClass;
var_dump($obj);
?>
```

- The keyword public determines the visibility of the property
- Next, the property is named using standard variable syntax, and a value is assigned (though class properties do not need an initial value).
- To read this property and output it to the browser, reference the object from which to read and the property to be read:

```
echo $obj->prop1;
```

## Defining Class Methods

- Methods are class-specific functions. Individual actions that an object will be able to perform are defined within the class as methods.
- For instance, to create methods that would set and get the value of the class property $prop1.

```
class MyClass
{
   public $prop1 = "I'm a class property!";
   public function setProperty($newval)
   {
      $this->prop1 = $newval;
   }
   public function getProperty()
   {
      return $this->prop1 . "<br />";
   }
}
```

- **Note** — OOP allows objects to reference themselves using $this. When working within a method, use $this in the same way you would use the object name outside the class.
- To use these methods, call them just like regular functions, but first, reference the object they belong to. Read the property from MyClass, change its value, and read it out again by making the modifications below:

```
<?php
class MyClass
{
  public $prop1 = "I'm a class property!";
  public function setProperty($newval)
  {
    $this->prop1 = $newval;
  }
  public function getProperty()
  {
    return $this->prop1 . "<br />";
  }
}
$obj = new MyClass;
echo $obj->getProperty(); // Get the property value
$obj->setProperty("I'm a new property value!"); // Set a new one
echo $obj->getProperty(); // Read it out again to show the change
?>
```

## Using Constructors and Destructors

- When an object is instantiated, it's often desirable to set a few things right off the bat. To handle this, PHP provides the magic method __construct(), which is called automatically whenever a new object is created.
- For the purpose of illustrating the concept of constructors, add a constructor to MyClass that will output a message whenever a new instance of the class is created:

```
<?php
class MyClass
{
  public $prop1 = "I'm a class property!";
  public function __construct()
  {
    echo 'The class "', __CLASS__, '" was initiated!<br />';
  }
  public function setProperty($newval)
  {
    $this->prop1 = $newval;
  }
  public function getProperty()
  {
    return $this->prop1 . "<br />";
  }
}
// Create a new object
$obj = new MyClass;
// Get the value of $prop1
echo $obj->getProperty();
// Output a message at the end of the file
echo "End of file.<br />";
?>
```

- **Note** — __CLASS__ returns the name of the class in which it is called; this is what is known as a magic constant. There are several available magic constants, which you can read more about in the PHP manual
- To call a function when the object is destroyed, the __destruct() magic method is available. This is useful for class cleanup (closing a database connection, for instance).
- Output a message when the object is destroyed by defining the magic method __destruct() in MyClass

```
<?php
class MyClass
{
  public $prop1 = "I'm a class property!";
  public function __construct()
  {
    echo 'The class "', __CLASS__, '" was initiated!<br />';
  }
  public function __destruct()
  {
    echo 'The class "', __CLASS__, '" was destroyed.<br />';
  }
  public function setProperty($newval)
  {
    $this->prop1 = $newval;
  }
  public function getProperty()
  {
    return $this->prop1 . "<br />";
  }
}
// Create a new object
$obj = new MyClass;
// Get the value of $prop1
echo $obj->getProperty();
// Destroy the object
unset($obj);
// Output a message at the end of the file
echo "End of file.<br />";
?>
```

## Task 1

- Create a class for querying a MySQL database within the class create for functions to manage all the tasks we can do with a database (INSERT, UPDATE, DELETE, SELECT).
- **Select Function:**
    - This function is used to return fields from a table with a condition.
    - **Input:**
        - ($table) the table needed to load data from.
        - ($outFields) the output fields from the table.
        - ($condition) the condition of the output fields.
    - **Output:**
        - ($data) an array of the needed fields from the input table.
- **Add Function:**
    - This function is used to insert values in a fields of a table.
    - **Input:**
        - ($table) the table needed to insert data into.
        - ($inFields) the array contains the input fields of the table as index of this array and the input value to the table as the value of this array.
    - **Output:**
        - (true) or (false).
- **Edit Function:**
    - This function is used to edit values of fields of a table.
    - **Input**:
        - ($table) the table needed to edit its data.
        - ($editFields) the array contains the edit fields of the table as index of this array and the edited values to the table as the value of this array.
    - **Output**:
        - (true) or (false).
- **Delete Function:**
    - this function is used to delete values from a table.
    - **Input**:
        - $table) the table needed to delete values from.
        - ($condition) the condition needed for the delete.
    - **Output**:
        - (true) or (false).

## Task 2

- Re-code the task we made on the last lecture using the class we implemented on the previous task.