OOP with PHP

Objectives

- Object Oriented Concepts
- Class & Object in OOP
- Constructors
- OOP features

Introduction to OOP

- Object-oriented programming is a style of coding that allows developers to group similar tasks into **classes**.
- **Class**
  - This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object**
  - An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Member Variable**
  - These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

Introduction to OOP

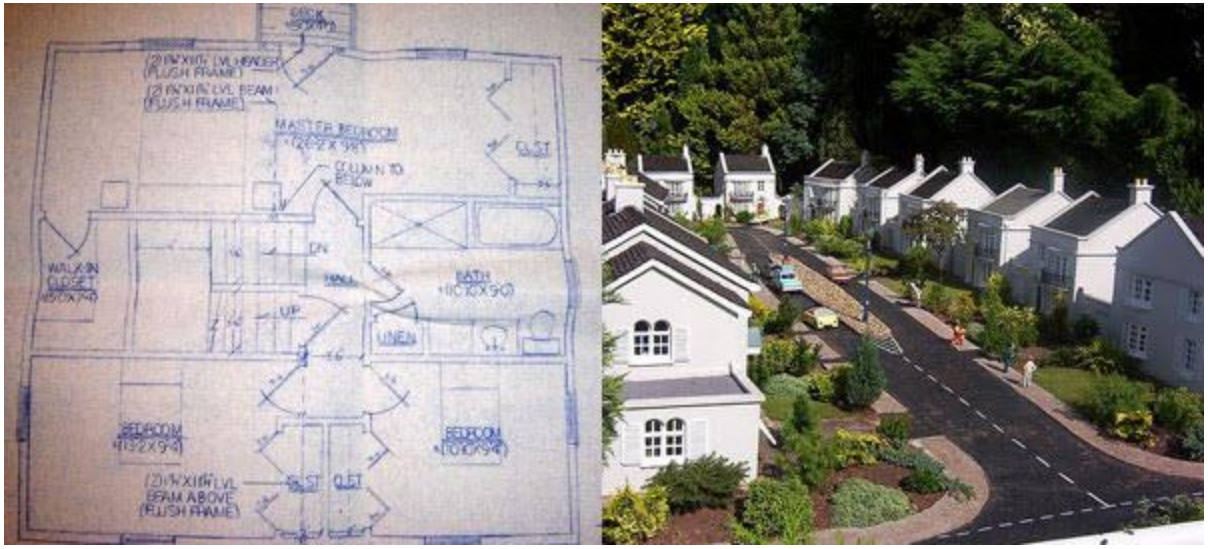
- **Member function**
  - These are the function defined inside a class and are used to access object data.
- **Parent class**
  - A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class**
  - A class that inherits from another class. This is also called a subclass or derived class.
- **Constructor**
  - Refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructor**
  - Refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

OOP Features

- **Inheritance**
  - When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Polymorphism**
  - This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.
- **Overloading**
  - a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction**
  - Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation**
  - refers to a concept where we encapsulate all the data and member functions together to form an object.

Class & Object

- A class, for example, is like **a blueprint for a house**. It defines the shape of the house on paper, with relationships between the different parts of the house clearly defined and planned out, even though the house doesn't exist.
- An object, then, is like **the actual house** built according to that blueprint. The data stored in the object is like the wood, wires, and concrete that compose the house: without being assembled according to the blueprint, it's just a pile of stuff. However, when it all comes together, it becomes an organized, useful house.



Defining class

- The special form **class**, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.
- The variable **$this** is a special variable and it refers to the same object ie. itself.

```php
<?php
   class Books {
      /* Member variables */
      var $price;
      var $title;

      /* Member functions */
      function setPrice($par){
         $this->price = $par;
      }

      function getPrice(){
         echo $this->price ."<br/>";
      }

      function setTitle($par){
         $this->title = $par;
      }

      function getTitle(){
         echo $this->title ." <br/>";
      }
   }
?>
```

Creating Objects in PHP

- Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.
- Example:

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

- Here we have created three objects and these objects are independent of each other and they will have their existence separately.

Calling Member Functions

- After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.
- Example:

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

Constructor

- Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.
- PHP provides a special function called **__construct()** to define a constructor. You can pass as many as arguments you like into the constructor function.
- Example:

```
function __construct( $par1, $par2 ) {
   $this->title = $par1;
   $this->price = $par2;
}
```

Constructor

- Example:

```
$physics = new Books( "Physics for High School", 10 );
$maths = new Books ( "Advanced Chemistry", 15 );
$chemistry = new Books ("Algebra", 7 );

/* Get those set values */
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();

$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
```

- Output:

```
Physics for High School
Advanced Chemistry
Algebra
10
15
7
```

Destructor

- Like a constructor function you can define a destructor function using function **__destruct()**.
- You can release all the resources within a destructor.

Demo

- Teacher demo code about defining class and creating object in PHP

Inheritance

- PHP class definitions can optionally inherit from a parent class definition by using the extends clause.
- Syntax:

```
class Child extends Parent {
    <definition body>
}
```

- The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:
  - Automatically has all the member variable declarations of the parent class.
  - Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

Example

```
class Novel extends Books {
   var $publisher;

   function setPublisher($par){
      $this->publisher = $par;
   }

   function getPublisher(){
      echo $this->publisher. "<br />";
   }
}
```

Public Members

- Unless you specify otherwise, properties and methods of a class are public. Public member may be accessed in three possible situations:
  - From outside the class in which it is declared
  - From within the class in which it is declared
  - From within another class that implements the class in which it is declared
- If you wish to limit the accessibility of the members of a class then you define class members as **private** or **protected**.

Private members

- By designating a member private, you limit its accessibility to the class in which it is declared.
- The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.
- A class member can be made private by using **private** keyword in front of the member.
- Example:

```
class MyClass {
   private $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }

   private function myPrivateFunction() {
      return("I'm  not visible outside!");
   }
}
```

Protected members

- A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class.
- Protected members are not available outside of those two kinds of classes.
- A class member can be made protected by using **protected** keyword in front of the member.
- Example:

```
class MyClass {
   protected $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }

   protected function myPrivateFunction() {
      return("I'm  visible in child class!");
   }
}
```

Interfaces

- Interfaces are defined to provide a common function names to the implementers.
- Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.
- Example:

```
interface Mail {
   public function sendMail();
}
```

- Implement interface:

```
class Report implements Mail {
   // sendMail() Definition goes here
}
```

Abstract Classes

- An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**.
- When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.
- Note that function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.
- Example:

```
abstract class MyAbstractClass {
    abstract function myAbstractFunction() {
    }
}
```

Final Keyword

- Final keyword prevents child classes from overriding a method by prefixing the definition with final.
- If the class itself is being defined final then it cannot be extended.
- Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```php
<?php

   class BaseClass {
      public function test() {
         echo "BaseClass::test() called<br>";
      }

      final public function moreTesting() {
         echo "BaseClass::moreTesting() called<br>";
      }
   }

   class ChildClass extends BaseClass {
      public function moreTesting() {
         echo "ChildClass::moreTesting() called<br>";
      }
   }
?>
```

Static Keyword

- Declaring class members or methods as static makes them accessible without needing an instantiation of the class.
- A member declared as static can not be accessed with an instantiated class object (though a static method can).
- Example:

```php
<?php
   class Foo {
      public static $my_static = 'foo';

      public function staticValue() {
         return self::$my_static;
      }
   }

   print Foo::$my_static . "\n";
   $foo = new Foo();

   print $foo->staticValue() . "\n";
?>
```

Demo

- Teacher demo code about OOP features in PHP

Summary

- Object Oriented Concepts
- Class & Object in OOP
- Constructors
- OOP features