UNIT: 3 Object Oriented Concepts in PHP

3.1. OOP concepts: Class, Object, Properties, Methods, Encapsulation, Access modifiers

Objects :

- ✓ An object is an instance of a class.
- ✓ Object is one type of class variable.
- ✓ An object is represented as its properties (attributes) and the operation performed on it.
- ✓ We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.
- ✓ Example: suppose Student is a class which contains two objects s1 and s2 with properties marks and percentage.

Classes :

- ✓ Class is a collection of objects of similar type.
- ✓ Object with same properties and operations form a group known as class.
- ✓ Class contains data member and data function.
- ✓ In a class, variables are called properties (data member) and data functions are called methods.
- ✓ Example: Suppose class name is car, then we can create three individual objects with the name of: Mercedes, Bmw, and Audi.

Properties:

- ✓ Properties are variables that are defined within a class. These variables are then used by the methods, objects of the class.
- ✓ Properties can accept values like strings, integers, and booleans (true/false values), like any other variable.
- ✓ In PHP, a property is qualified by one of the access specifier keywords, **public**, **private** or **protected**.
- ✓ Name of property could be any valid label in PHP.
- ✓ Value of property can be different for each instance of class.
- ✓ Property is available to object with the help of -> operator.
- ✓ Example:

```
<?php
class Student
{
  // Properties
  public $name;
}
```

Methods:

- ✓ The classes most often contain functions. A function inside a class is called a **method**.
- ✓ These functions can then be called from an object.
- ✓ Functions can be public, private or protected. By default is public.
- ✓ Example:

```
class Student {
  // Properties
  public $name;

  // Methods
  function set_name($name)
```

```
  {
    $this->name = $name;
  }
}
```

Encapsulation:

- ✓ The wrapping up of data and methods into a single unit (called class) is known as encapsulation.
- ✓ Encapsulation is a protection mechanism for the data members and methods present inside the class.
- ✓ In the encapsulation technique, we are restricting the data members from access to outside world end-user.
- ✓ In PHP, encapsulation utilized to make the code more secure and robust.
- ✓ Using encapsulation, we are hiding the real implementation of data from the user and also does not allow anyone to manipulate data members except by calling the desired operation.

Access modifiers:

- ✓ Properties and methods can have access modifiers which control where they can be accessed.
- ✓ There are three access modifiers:
  - public - the property or method can be accessed from everywhere. This is default
  - protected - the property or method can be accessed within the class and by classes derived from that class
  - private - the property or method can ONLY be accessed within the class

3.2. Creating Classes, Objects

Class:

- ✓ A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods go inside the braces:
  Syntax:

```
<?php
class Classname {
  // code goes here...
}
?>
```

- ✓ we declare a class named Student consisting of one property ($name) and two methods set_name() and get_name() for setting and getting the $name property:

```
<?php
class Student {
  // Properties
  public $name;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Object:

- ✓ Objects of a class are created using the new keyword.
- ✓ We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

<?php
class Add {
protected $a;
protected $b;
 	
function set_no($a,$b) {
$this->a = $a;
$this->b = $b;
}
function sum() {
return $this->a + $this->b;
}
}
 	
$no = new Add();
$no->set_no(10,20);
 	
echo $no->sum();
echo "
";
 	
?>
	30

3.3. Constructors and Destructors

Constructors:

- ✓ constructor is a method defined inside a class is called automatically at the time of creation of object.
- ✓ Purpose of a constructor method is to initialize the object.
- ✓ A constructor is a public method which is named as __construct

Syntax:

```
function __construct()
{
// initialize the object and its properties by assigning values
}
```

Constructor types:

Default Constructor: It has no parameters, but the values to the default constructor can be passed dynamically.

- ✓ **Parameterized Constructor**: It takes the parameters, and also you can pass different values to the data members.
- ✓ **Copy Constructor**: It accepts the address of the other objects as a parameter.

Default Constructor	Output
<?php class rectangle{ public $height; public $width;	Height=0 Width=0

```
  function __construct(){
    $this->height=0;
    $this->width=0;
  }
  function show(){
    echo "Height=$this->height Width=$this-
>width";
  }
}
$obj=new rectangle();
$obj->show();
?>
```

Destructors

- ✓ Destructors are called when an object destructs. Usually, it is when the script ends.
- ✓ A destructor is a public method which is named as __ destruct

Syntax:

```
function __destruct()
{
// destroying the object or clean up resources here
}
```

Example:

```
<?php
class MyClass {
   public function __construct() {
      echo "Object created"."<br>";
   }

   public function __destruct() {
      echo "Object destroyed";
   }
}

$obj = new MyClass();
unset($obj);
?>
```

Output:

```
Object created
Object destroyed
```

3.4. Inheritance

- ✓ **It is the process the by which object of one class derived (acquired) the properties of object of another class.**
- ✓ In inheritance, the old class is called as the base class and the new class is called as the derived class or subclass.
- ✓ Inheritance provides you with many benefits that make PHP programming a lot more convenient.
- ✓ One such benefit is code reusability.
- ✓ Reusability allows you to generate clean code and the replication of code gets reduced to almost zero.
- ✓ Reusing existing codes serves various advantages.

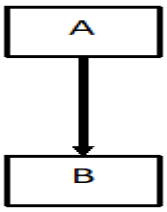
- ✓ It saves time, cost, effort, and increases a program's reliability.
- ✓ PHP supports three types of inheritance based on their functionality.

  - ✓ Single Inheritance: There is only one base class and one sub/derived class in a single inheritance and the subclass is directly inherited from the base class.
  - ✓ Hierarchical Inheritance: the hierarchical inheritance adopts a tree-like structure, where multiple derived classes are inherited from the base class.
  - ✓ Multilevel Inheritance: Multilevel occurs at different levels. one base class is inherited by a derived class, then that derived class is inherited by other derived classes, and so on.

Syntax for Inheriting a Class: class ChildClass extends ParentClass

where, ChildClass is a derived class (also called extended class and subclass) that extends ParentClass (also called superclass and base class).

Single inheritance: A derived class with only one base Class is called Single Inheritance.



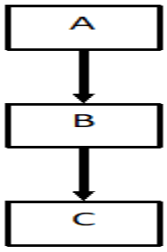
- ✓ Here, A is a base class and B is a derived class. Class B will acquire all the members declared in Class A.

Example : 1

```
<?php

class ParentClass
{
  function a()
  {
    echo "A member function of the base class.<br>";
  }
}


class ChildClass extends ParentClass
{
  function b()
  {
    echo "A member function of the child class.<br>";
  }
}

$Obj = new ChildClass();
$Obj->a();
```

```
$Obj->b();
?>
```
Output:
member function of the base class.
A member function of the child class.

Multilevel Inheritance: A chain process of deriving a class from another 'derived class' is called multilevel inheritance.



- ✓ Here, Class A is a Base Class, class C is a Derived Class and class B is a intermediate Base Class.
- ✓ Class C inherits from class B and class B inherits from class A.
- ✓ So, class C will acquire all the members of class B as well as class A.

Example 1:

```
<?php

class ParentClass
{
  function a()
  {
    echo "A member function of the base class.<br>";
  }
}


class ChildClass1 extends ParentClass
{
  function b()
  {
    echo "A member function of the child class1.<br>";
  }
}
class ChildClass2 extends ChildClass1
{
  function c()
  {
    echo "A member function of the child class2.<br>";
  }
}
$Obj = new ChildClass2();
$Obj->a();
$Obj->b();
$Obj->c();
```

```
?>

Output:

A member function of the base class.
A member function of the child class1.
A member function of the child class2.
```

3.5. Polymorphism: Overloading, Overriding

- ✓ **Overloading** means to use the same thing for different purpose.
- ✓ It contains the same function name and that function performs different tasks according to the number of arguments.
- ✓ For example, find the area of certain shapes where the radius is given then it should return the area of a circle if height and width are given then it should give the area of rectangle and others.

Function Overloading

```
<?php
class Area
{

function __call($name, $arg)
{

if($name == 'area')
{
switch(count($arg))
{

case 0 : return 0 ;
case 1 : return 3.14 * $arg[0] ;
case 2 : return $arg[0] * $arg[1];
}

}

}
}
$c = new Area();
echo "Area of circle:".$c->area(5)."</br>";
$rect = new Area();
echo "Area of rectangle:".$rect->area(5,10);

?>


Area of circle:15.7
Area of rectangle:50
```

function overriding: The two methods with the same name and same parameter is called overriding.

- ✓ Both parent and child classes should have same function name with and number of arguments.
- ✓ It is used to replace parent method in child class. The purpose of overriding is to change the behavior of parent class method.

```
<?php
  class Base
 {
     function display()
  {
        echo "Base class function :display <br>";
   }

function demo()
{
        echo "Base class function:demo<br>";
   }
 }
   class Derived extends Base
 {
     function demo()
  {
        echo "Derived class function:demo<br>";
     }
   }
   $ob = new Base;
   $ob->demo();
   $ob->display();
   $ob1 = new Derived;
   $ob1->demo();
   $ob1->display();
?>

Output:
Base class function: demo
Base class function :display
Derived class function: demo
Base class function :display
```

3.6. Interface

- ✓ Interfaces allow you to specify what methods a class should implement.
- ✓ Interfaces are declared with the **interface** keyword.
- ✓ All methods declared in an interface must be public.
- ✓ To implement an interface, a class must use the **implements** keyword.
- ✓ A class that implements an interface must implement all of the interface's methods.

```
<?php
interface MyInterface
{
```

```
    public function method1();
    public function method2();

}

class MyClass implements MyInterface
{

    public function method1()
{
        echo "Method1 Called<br>" ;
    }

    public function method2()
{
        echo "Method2 Called<br>";
    }
}

$obj = new MyClass;
$obj->method1();
$obj->method2();

?>
```

Output:

```
Method1 Called
Method2 Called
```

3.7. Abstract Class

- ✓ An abstract class is a class that contains at least one abstract method.
- ✓ An abstract method is a method that is declared, but not implemented in the code.
- ✓ An abstract class or method is defined with the abstract keyword.
- ✓ When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child class.

```
<?php

abstract class Base
{
    abstract function display();
}
class Derived extends Base
{
    function display()
{
        echo "Derived class";
    }
```

```
}


$b1 = new Derived;
$b1->display();
?>

Output:

Derived class
```

3.8. Final keyword

- ✓ The final keyword is used only for methods and classes.
- ✓ if we declare class method as a Final then that method cannot be override by the child class.
- ✓ Same as method if we declare class as a Final then that class cannot be extended any more.
- ✓ A final class can contain final as well as non final methods. But there is no use of final methods in class when class is itself declared as final because inheritance is not possible.

Example (method as final)

```
<?php

   class base
   {
      final public function display()
      {
         echo "Base class..";
      }
   }
   class derived extends base
   {
      public function display()
      {
         echo "derived class";
      }
   }
   $obj = new derived();
   $obj->display();

?>
```

Output:

3.9. Cloning Objects

- ✓ The clone keyword is used to create a copy of an object.
- ✓ When an object is cloned, PHP will perform a shallow copy of all of the object's properties.

Syntax: $copy_object_name = clone $object_to_be_copied;

✓ The **clone** keyword creates a shallow copy. Change in value of property doesn't reflect in cloned object.

✓ cloned object have different values than original object but original and referenced object created by using '=' operator have same value.

Example using clone keyword

```
<?php

class Student {
   public $name;
   public $sem;
   public $spi;
}

$obj = new Student ();

$copy = clone $obj;

$obj->name = "abc";
$obj->sem = "3";
$obj->spi = "9.2";


$copy->name = "xyz";
$copy->sem = "3";
$copy->spi = "8.7";

echo $obj->name .$obj-> sem .$obj->spi."<br>" ;

echo $copy->name. $copy-> sem. $copy->spi."<br>";

?>
```

Output:

```
abc 3 9.2
xyz 3 8.7
```