Chapter 1: Introduction to Object oriented Programming in PHP

Introduction

Like C++ and Java, PHP also supports object oriented programming (OOP) features. OOP opens the door to cleaner designs, easier maintenance and greater code reuse. It include the fundamental connection between data and the code that works on that data. The class and object are the fundamental construct behind object-oriented programming.

1.1 Class and 1.3 Encapsulation:

PHP also allows you to group related functions together using a *class.*

- Class is a programmer-defined data type, which includes variables and functions. Variables within a class are called *properties*; functions are called *methods*.
- Class is a collection of objects. Object has properties and behaviour.

Class names are case-insensitive and must conform to the rules for PHP identifiers.

We define our own class by starting with the keyword **'class'** followed by the name you want to give your new class. Inside a class you can define variables and functions as shown below:

Syntax:

```
class classname [ extends baseclass ]
{
[ var $property [ = value ]; ... ]
[ function functionname (args) {
// code
}...
]
}
```

Declaring Properties and Methods:

Property of a class can be defined as as private, public or protected. Property declaration is optional.
It's a good programming style to declare it, but you can add new properties at any time.

A method is a function defined inside a class as shown in above syntax. In PHP, most methods act only on data of a class in which the method resides. Within a method, the $this variable contains a reference to the object on which the method was called.

Example:

```
<?php
class Employee{
        private $emp_code, $emp_name, $emp_designation;   // property declaration
       public function accept($c,$n,$d)                          //method definition
       {
        $this->emp_code=$c;   //$this is used to access properties of current invoking object
        $this->emp_name=$n;
        $this->emp_designation=$d;
       }
       function display()        //default visibility is public
       {
                echo "<br>Emp Code: ".$this->emp_code;
                echo "<br>Emp Name: ".$this->emp_name;
                echo "<br>Emp Designation: ".$this->emp_designation;
       }
}
```
If you call $emp->accept() then inside accept method, $this hold the same value as $emp. Methods use the $this variable to access the properties of the current object. As you can see, the accept() and display() methods use $this to access and set emp_code, emp_name and emp_designation properties of the current object.

- Using access modifiers, you can change the visibility of methods. Methods that are accessible outside the class using object should be declared as *public*; methods on an instance that can only be called by methods within the same class should be declared *private*. Finally, methods declared as *protected* can only be called from within the object's class methods and the class methods of classes inheriting from the class.
- Defining the visibility of class methods is optional; if a visibility is not specified, a method is *public*

1.2 Object:
An individual instance of a class is called as Object. You define a class once and then make many objects that belong to it. Objects are also known as instance.
To create an object of a given class, use the new keyword:
Syntax: $*object* = new *Class*;
Example: $emp=new Employee();

Accessing Properties and Methods:
The class methods and properties can directly be accessed through the object instance using the -> notation.
Syntax: *$object->propertyname //if visibility of property is public*
$object->methodname([*arg, ...]*)
Example: $emp->accept(1,"Atharv", "Manager");
$emp->display();

Static property and Static Method:
PHP allows you to define static properties , which are variables of class and can be accessed by referencing the property with the class name.

Also to access methods in terms of a class rather than an object, use **static** keyword. Any method declared as static is accessible without the creation of an object. Static methods are associated with the class, not with an instance of the class. They are permitted to access other static methods and static variables only.

Example:

```
<?php

class example_static {
        static $count;          //static property
        public static function updateCount() {    //static method
          return self::$count++;
      }
}
  example_static::$count = 1;

for($i = 0; $i < 5; ++$i) {
   echo 'The value is: '.example_static::updateCount() . "<BR>";
}

?>
```

```
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
```

Inside a class, you can refer to the static property using the ***self*** keyword

1.4 Declaring Constructor and Destructor:

Constructor in PHP is special type of function of a class which is automatically executed as any object of that class is created or instantiated. Usually it starts with two underscore characters.

Syntax:

```
        function __construct([args..])
        { //definition
        }
```

You may provide a list of arguments following the class name when instantiating an object. These arguments are passed to a constructor, a special function that initializes the properties of the class.

When an object is destroyed, such as when the last reference to an object is removed or the end of the script is reached, its destructor is called.

Syntax:

```
        function __destruct()
       { //definition
```

```
          }
```
Example:

```
<?php
class Employee{
private $emp_code;
private $emp_name;
private $emp_designation;
function __construct($c,$n,$d)     //constructor definition
        {
                $this->emp_code=$c;
                $this->emp_name=$n;
                $this->emp_designation=$d;
                echo "<br> Object is created";
         }
         function display()
        {
                echo "<br>Emp Code: ".$this->emp_code;
                echo "<br>Emp Name: ".$this->emp_name;
                echo "<br>Emp Designation: ".$this->emp_designation;
         }
        function __destruct()   //destructor definition
        {
         echo "<br>Object is destroyed";
         }
}
$c=1;
$n="Atharv ";
$d="Manager";
$emp=new Employee($c,$n,$d);  //constructor is called automatically when object is created
$emp->display();
?>
```
Output:

```
Object is created
Emp Code: 1
Emp Name: Atharv
Emp Designation: Manager
Object is destroyed
```
In above PHP script, the statement $emp=new Employee($c,$n,$d) invokes the constructor function automatically when object is created and at the end of the script destructor is called automatically.

1.5 Inheritance:

To inherit the properties and methods from another class, use the extends keyword in the class definition, followed by the name of the base class:

```
class Employee {
private $emp_code, $emp_name, $emp_designation;
};
class EmpoyeeSalary extends Employee{
private $basic_pay, $earning, $deduction;
}
```

The EmpoyeeSalary class contains the $basic_pay, $earning and $deduction, as well as the $emp_code, $emp_name and $emp_designation properties inherited from the Employee class.

If a derived class has a property or method with the same name as one in its parent class, the Property or method in the derived class *overrides*, the property or method in the parent class. Referencing the property returns the value of the property on the child, while referencing the method calls the method on the child.

To access an overridden method, on an object's parent class, use the parent::*method*() notation:

Example: parent::display();

Similarly in the derived class, parent class constructor can be invoked as

```
        parent::__construct();
```

Consider above class Employee definition. Now let's define derived class EmpoyeeSalary.

Example:

```
<?php
class EmployeeSalary extends Employee{
        private $basic_pay;
        private $earnings;
        private $deduction;
        function EmployeeSalary($c,$n,$d,$b,$e,$dd)
        {
                parent::__construct($c,$n,$d);
                $this->basic_pay=$b;
                $this->earnings=$e;
                $this->deduction=$dd;
        }
        function CalculateSalary()
        {
        echo "<br> Net Salary(Rs.) :".($this->basic_pay+ $this->earnings-$this->deduction);
        }

}
$c=1;
$n="Atharv ";
$d="Manager";
$b=10000;
$e=500;
$dd=750;
$emp=new EmployeeSalary($c,$n,$d,$b,$e,$dd);
$emp->display();
$emp->CalculateSalary();
?>
```

Output:

```
Object is created
Emp Code: 1
Emp Name: Atharv
Emp Designation: Manager
Net Salary(Rs.) :9750
Object is destroyed
```

Abstract Class and Abstract Method:

Abstraction is a way of hiding information. In abstraction, there should be at least one method that must be declared but not defined. The class that inherit this abstract class need to define that method.

Characteristics:

- There must be an abstract keyword that must be written before the class for it to be an abstract class.
- This class cannot be instantiated. Only the class that implements the methods of an abstract class can be instantiated. There can be more than one methods that can be left undefined.

Example:

```
<?php
abstract class A
{
abstract function one();
public function two()
{
echo "Non-abstract method";
}
}
```

```
class B extends A
{
public function one()
{
echo "Abstract Function one defined by subclass<br/>";
}
}
$obj = new B();
$obj->one();
$obj->two();
?>
```
Output:

```
Abstract Function one declared in A defined by subclass B
Non-abstract method
```

1.6 Interface:

An Interface allows the users to create programs, specifying the public methods that a class must implement. The interface contains no data variables.

Characteristics:

- Interface is similar to a class except that it cannot contain code.
- An interface can define method names and arguments, but not the contents of the methods.
- Any classes implementing an interface must implement all methods defined by the interface.
- A class can implement multiple interfaces.
- An interface is declared using the "interface" keyword.
- Interfaces can't contain Non-abstract methods.

Example:

```
<?php
  interface myInterface {
      public  function method_one();
      public  function method_two();
}
class myClass implements myInterface{
     public  function method_one() {
     echo "<BR> Method one is implemented";

  }
  public  function method_two(){
   echo "<BR> Method two is implemented";
  }
}
$obj=new myClass ();
```

```
$obj->method_one();
$obj->method_two();
?>
```
Output:
```
Method one is implemented
Method two is implemented
```

1.7 Introspection

It is the ability of a program to examine an object’s characteristics, such as its name, parent class (if any), properties, and methods. With introspection, you can write code that operates on any class or object. You don’t need to know which methods or properties are defined when you write your code; instead, you can discover that information at runtime, which makes it possible for you to write generic debuggers.

Examining Classes

- To determine whether a class exists, use the class_exists() function, which takes in a string and returns a Boolean value.
  ```
  $yes_no = class_exists(classname);
  ```

- Alternately, you can use the get_declared_classes() function, which returns an arrayof defined classes and checks if the class name is in the returned array:
  ```
  $classes = get_declared_classes( );
  ```

- You can get the methods and properties that exist in a class (including those that are inherited from superclasses) using the get_class_methods() and get_class_vars( ) functions. These functions take a class name and return an array:
  ```
  $methods = get_class_methods(classname);
  $properties = get_class_vars(classname);
  ```

NOTE: get_class_vars() is that it returns only properties that have default values; there’s no way to discover uninitiailized properties.

- Use get_parent_class() to find a class’s parent class:
  ```
  $superclass = get_parent_class(classname);
  ```

Examining an Object

- To get the class to which an object belongs, first make sure it is an object using the is_object() function, then get the class with the get_class( ) function:
  ```
  $yes_no = is_object(var);
  $classname = get_class(object);
  ```

- Before calling a method on an object, you can ensure that it exists using the method_exists() function:
  $yes_no = method_exists(object, method);
  Calling an undefined method triggers a runtime exception.

- get_object_vars( ) returns an array of properties set in an object.
  $array = get_object_vars(object);

  Example:

```
<?php
class A {
  private $x=10, $y;
  function A($a)
  {
       $this->y=$a;
      echo "<BR> Constructor A is called";
  }
  function displayA()
  {
      echo "<BR> x=$this->x y=$this->y";
  }
}

class B extends A {
  private $p,$q;
  function B($l,$m,$n)
  {
       parent::A($l);
       $this->p=$m;
      $this->q=$n;
      echo "<BR> Constructor B is called";
  }
  function displayB()
  {
      echo "<BR> p=$this->p q=$this->q";
  }
}
$obj = new B(20,30,40);
$obj->displayA();
$obj->displayB();

echo "<BR> Class A exists or not ";
echo class_exists('A');
echo "<BR> Class C exists or not ";
echo class_exists('C');

echo "<BR> Get existing all classes";
$allclasses= get_declared_classes();
//print_r($allclasses);


echo "<BR> Get methods of class A";
print_r(get_class_methods('A'));

echo "<BR> Get methods of class B";
print_r(get_class_methods('B'));

echo "<BR> Get variables of class A";
var_dump(get_class_vars('A'));

echo "<BR> Get variables of class B(no property with default value)";
print_r(get_class_vars('B'));

echo "<BR> Parent of class B";
echo get_parent_class($obj);  //from object of class B
echo get_parent_class('B');   // from class name

echo "<BR> Parent of class A";  //no parent so no output
echo get_parent_class('A');

echo "<BR>obj is obejct or not";
echo is_object($obj);

echo "<BR>obj1 is obejct or not";  //obj1 is not an object
echo is_object($obj1);

echo "<BR>Get obj's class name";
echo get_class($obj);


echo "<BR>Method displayA() exists or not ";
echo method_exists($obj, 'displayA');

echo "<BR>Method displayC() exists or not ";
echo method_exists($obj, 'displayC');

echo "<BR> Get obj's variables ";
var_dump(get_object_vars($obj1));
```

```
echo "<BR> Get variables of class B(no property with default value)";
print_r(get_object_vars($obj));

?>
```

Output:

```
Constructor A is called
Constructor B is called
x=10 y=20
p=30 q=40
Class A exists or not 1
Class C exists or not
Get existing all classes
Get methods of class AArray ( [0] => A [1] => displayA )
Get methods of class BArray ( [0] => B [1] => displayB [2] => A [3] => displayA )
Get variables of class Aarray(0) { }
Get variables of class B(no property with default value)Array ( )
Parent of class BAA
Parent of class A
obj is obejct or not1
obj1 is obejct or not
Notice: Undefined variable: obj1 in D:\xampp\htdocs\introspection.php on line 68

Get obj's class nameB
Method displayA() exists or not 1
Method displayC() exists or not
Get obj's variables
Notice: Undefined variable: obj1 in D:\xampp\htdocs\introspection.php on line 81

Warning: get_object_vars() expects parameter 1 to be object, null given in
D:\xampp\htdocs\introspection.php on line 81
NULL
Get variables of class B(no property with default value)Array ( )
```