Unit 5: OOP

OOP introduction

OOP stands for Object-Oriented Programming.

The word **object-oriented** is the combination of two words i.e. **object** and **oriented**. The dictionary meaning of the object is an article or entity that exists in the real world. The meaning of oriented is interested in a particular kind of thing or entity.

The **object-oriented programming** is basically a computer programming design philosophy or methodology that organizes/ models software design around data, or objects rather than functions and logic.

An object is referred to as a data field that has unique attributes and behavior. Everything in OOP is grouped as self-sustainable objects.

It is the most popular programming model among developers. It is well suited for programs that are large, complex, and actively updated or maintained.

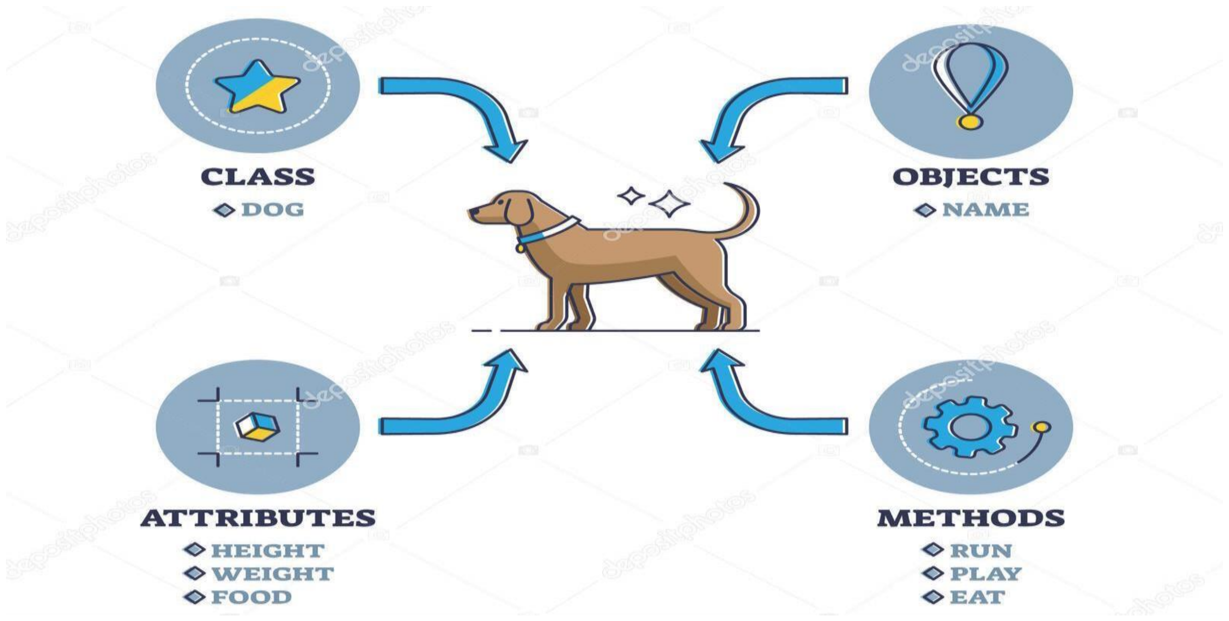
Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug

OOP makes it possible to create full reusable applications with less code and shorter development time

The four main principles of object-oriented programming
(abstraction, inheritance, encapsulation, and polymorphism).



<u>Class</u>

In object-oriented programming , a class is a template definition of the method s and variable s in a particular kind of object . Thus, an object is a specific instance of a class; it contains real values instead of variables. The class is one of the defining ideas of object-oriented programming.

A class is a way of organizing information about a type of data so a programmer can reuse elements when making multiple instances of that data type—for example, if a programmer wanted to make three instances of Car , maybe a BMW, a Ferrari, and a Ford instance.

A class is a template for objects, and an object is an instance of class**.**

OOP Case

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Define a 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 Fruit {
  // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Note: In a class, variables are called properties and functions are called methods!

Define Objects

Classes are nothing without objects! 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.

Objects of a class are created using the `new` keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

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

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

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

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

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

Property:-

Property is sometimes referred to as **attribute or field**. 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. That's why it is sometimes referred as instance variable.

Inside any instance method, property can be accessed by calling object's context available as a pesudo-variable **$this**. If a property is declared as public, it is available to object with the help of **->** operator. If a property is defined with **static** keyword, its value is shared among all objects of the class and is accessed using scope resolution operator **(::)** and name of class.

property declaration and access

This example shows how a property is defined and accessed

Example

```
<?php
class myclass{
   private $fname="Kiran";
   public $mname="Pratap";
   static $lname="Singh";
   function dispdata(){
      echo "$this->fname
";
      echo "$this->mname
";
      echo myclass::$lname;
   }
}
$obj=new myclass();
$obj->dispdata();
?>
```

Output

The output of above code is as follows −

Kiran

Pratap

Singh

Outside class, instance properties declared as public are available to object, but private properties are not accessible.

In previous versions of PHP, **var** keyword was available for property declaration. Though it has now been deprecated, it is still available for backward compatibility and is treated as public declaration of property.

PHP 7.4 introduces type declaration of property variables

Example

```
<?php
class myclass{
   private string $name;
   private int $age;
   function setdata(string $x, int $y){
      $this->name=$x;
      $this->age=$y;
   }
}
$obj=new myclass("Kiran",20);
?>
```

Visibility ¶

The visibility of a property, a method or (as of PHP 7.1.0) a constant can be defined by prefixing the declaration with the keywords `public`, `protected` or `private`.

Class members declared public can be accessed everywhere. Members declared protected can be accessed only within the class itself and by inheriting and parent classes.

Members declared as private may only be accessed by the class that defines the member.

Property Visibility ¶

Class properties may be defined as public, private, or protected. Properties declared without any explicit visibility keyword are defined as public.

Example #1 Property declaration

```
<?php
/**
 * Define MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj-
>printHello(); // Shows Public, Protected and Private


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // We can redeclare the public and protected proper
ties, but not private
    public $public = 'Public2';
    protected $protected = 'Protected2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Works
echo $obj2->protected; // Fatal Error
echo $obj2->private; // Undefined
$obj2-
>printHello(); // Shows Public2, Protected2, Undefined

?>
```

Method Visibility ¶

Class methods may be defined as public, private, or protected. Methods declared without any explicit visibility keyword are defined as public.

Example #2 Method Declaration

```
<?php
/**
 * Define MyClass
 */
class MyClass
{
    // Declare a public constructor
    public function __construct() { }

    // Declare a public method
    public function MyPublic() { }

    // Declare a protected method
    protected function MyProtected() { }

    // Declare a private method
    private function MyPrivate() { }

    // This is public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Works
$myclass->MyProtected(); // Fatal Error
$myclass->MyPrivate(); // Fatal Error
$myclass->Foo(); // Public, Protected and Private work



/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // This is public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Fatal Error
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Works
$myclass2-
>Foo2(); // Public and Protected work, not Private

class Bar
{
    public function test() {
        $this->testPrivate();
        $this->testPublic();
    }

    public function testPublic() {
        echo "Bar::testPublic\n";
    }

    private function testPrivate() {
        echo "Bar::testPrivate\n";
    }
}

class Foo extends Bar
{
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
        echo "Foo::testPrivate\n";
    }
}

$myFoo = new Foo();
$myFoo->test(); // Bar::testPrivate
                // Foo::testPublic
?>
```

Constructors and Destructors

Introduction

In object oriented programming terminology, 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. In PHP, a method of special name **__construct** acts as a constructor.

Syntax

__construct ([mixed $args = "" [, $...]]) : void

Constructor example

This example shows that constructor is automatically executed when object is declared

Example

```
<?php
class myclass{
  function __construct(){
      echo "object initialized";
   }
}
$obj=new myclass();
?>
```

Output
This will produce following result. –

```
object initialized
```

Constructor with arguments

Class properties are initialized by constructor with arguments

Example

```
<?php
class rectangle{
   var $height;
   var $width;
   function __construct($arg1, $arg2){
      $this->height=$arg1;
      $this->width=$arg2;
   }
   function show(){
      echo "Height=$this->height Width=$this->width";
   }
}
$obj=new rectangle(10,20);
$obj->show();
?>
```

Output
This will produce following result. −

```
Height=10 Width=20
```

Constructor in inheritance

If the parent class has constructor defined in it, it can be called within child class's constructor by **parent::_construct**. However, if child class doesn't define a constructor, it inherits the same from is base class.

Example

```
<?php
class a{
   function __construct(){
      echo "this is a constructor of base class
";
   }
}
class b extends a{
   function __construct(){
      parent::__construct();
      echo "this a constructor class b
";
   }
}
class c extends a {
   //
}
$a=new a();
$b=new b();
$c=new c();
?>
```

Output
This will produce following result. −

```
this is a constructor of base class
this is a constructor of base class
this a constructor class b
this is a constructor of base class
```

Destructor

Destructor is a method automatically as soon as garbage collector fins that a particular object has no more references. In PHP, destructor method is named as **__destruct**. During shutdown sequence too, objects will be destroyed. Destructor method doesn't take any arguments, neither does it return any data type

Example

```
<?php
class myclass{
function __construct(){
echo "object is initialized
";
   }
   function __destruct(){
      echo "object is destroyed
";
   }
}
$obj=new myclass();
?>
```

Output

This will show following result

```
object is initialized
object is destroyed
```

PHP - What is Inheritance?

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the `extends` keyword.

Inheritance

It is a concept of accessing the features of one class from another class. If we inherit the class features into another class, we can access both class properties. We can extends the features of a class by using 'extends' keyword.

- It supports the concept of **hierarchical classification**.
- Inheritance has three types, **single, multiple and multilevel Inheritance**.
- **PHP** supports only **single inheritance**, where only one class can be **derived from single parent class**.
- We can simulate multiple inheritance by using **interfaces**.

Example 1

```
<?php
class a
   {
      function fun1()
      {
         echo "javatpoint";
      }
   }
   class b extends a
   {
      function fun2()
      {
         echo "SSSIT";
```

```
        }
    }
    $obj= new b();
    $obj->fun1();
?>
```

Scope Resolution Operator (::)

The Scope Resolution Operator, the double colon, is **a token that allows access to static, constant, and overridden properties or methods of a class**. When referencing these items from outside the class definition, use the name of the class.

PHP, the double colon **::** is defined as **Scope Resolution Operator**. It used when when we want to access constants, properties and methods defined at class level. When referring to these items outside class definition, name of class is used along with scope resolution operator.

This operator is also called **Paamayim Nekudotayim**, which in Hebrew means double colon.

Syntax

```
<?php
class A{
   const PI=3.142;
```

```
  static $x=10;
}
echo A::PI;
echo A::$x;
$var='A';
echo $var::PI;
echo $var::$x;
?>
```

Inside class

To access class level items inside any method, keyword - **self** is used

```
<?php
class A{
  const PI=3.142;
  static $x=10;
  static function show(){
    echo self::PI . self::$x;
  }
}
A::show();
?>
```

In child class

If a parent class method overridden by a child class and you need to call the corresponding parent method, it must be prefixed with **parent** keyword and scope resolution operator

Example

```
<?php
class testclass{
   public function sayhello(){
      echo "Hello World
";
   }
}
class myclass extends testclass{
   public function sayhello(){
      parent::sayhello();
      echo "Hello PHP";
   }
}
$obj=new myclass();
$obj->sayhello();
?>
```

Output

This will produce following output −

```
Hello World
Hello PHP
```

Auto loading classes

Autoloading

The easiest way to make our `src` directory known to Composer as "the place for everything" is to add this "autoload" section to `composer.json.` `autoload.php`

```
"autoload":  {"psr-4": { "": "src/" } }
```

This tells Composer to use the `src` directory to find all classes.

The details of that standard are not that important right now.

Auto loading *is PHP's way to automatically find classes and their corresponding files without having to* `require` *all of them manually. You can read more about it here. Auto loading is also one of the reasons why PHP code is usually organized in classes because PHP only supports auto loading for classes, not simple functions for example.*

The only thing we still need to do is to add Composer's auto loading to our application:

```
<?php
```

```
require "vendor/autoload.php";
```

This should be right on top of your main application file to make sure, all classes can be found right from the beginning.

It loads a PHP file that Composer created automatically (`vendor/autoload.php`). This file loads Composer's auto loading logic into our project.

After that all third party namespaces from our dependencies as well as our own are available in our code.