Chapter 4 : Introduction to Object Oriented Programming

Prepared by Deepali Sonawane, MIT-WPU, Pune

What is OOP?

- OOP stands for **Object Oriented Programming**.
- 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**.
- **Advantages** of Object-Oriented Programming:
  1. Focus is on data rather than procedures or functions.
  2. OOP is faster and easier to execute
  3. OOP provides a clear structure for the programs
  4. OOP makes the code easier to maintain, modify and debug
  5. OOP makes it possible to create full reusable applications with less code and shorter development time

Object Oriented Concepts

- **Class:** This is a user-defined data type, which consists of data members and member functions.
- **Object:** It is an instance of the class.
- **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 take different number of arguments and can do different task.
- **Data Abstraction:** Data Abstraction refers to the act of representing essential features without including the background details or explanations. Any representation of data in which the implementation details are abstracted.
- **Encapsulation:** The binding of data and functions into single unit is called as encapsulation.

Class

- A class is defined by using the **class** keyword.
- Class keyword is followed by the name of the class and a pair of curly brackets {}.
- All class properties and methods, we write inside the brackets.
- Syntax:

```
class class_name
{
        variable declaration;
        function definition;
}
```

- Example

```
class Book
{
    var $bname;
    function setBname()
    {
            $this->bname="Introduction to PHP";
    }
}
```

Object

- Object is an **instance** of the class.
- 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 is created using the **new** keyword.
- Syntax:

```
class class_name
{
        variable declaration;
        function definition;
}
$object_name = new class_name();
```

- Example

```
class Book
{
   public $bname;
   function setBname()
   {
            $bname="Introduction to PHP";
    }
}
$B1 = new Book();
```

Access modifiers

- **Public:** By **default** properties and methods of a class are **public**. They are accessible by any function inside the class as well as out of the class i.e. the property or method can be accessed from **everywhere**.
- **Private:** The private members are accessible **only within the class** i.e. by the member functions of their same class. Private data members are not accessible out of the class.
- **Protected:** The protected members are accessible **within the class** and also from the member functions of **derived class.**

Use of $this

- $this is mainly used to **refer properties** of a class.
- It is the way to reference an **instance of a class** from within itself, the same as many other object oriented languages.

Example:

```
class employee
{
   var $empno;
   var $ename;
  function set()
   {
       $this->empno = 101;
       $this->ename = “Suhas Shinde”;
   }
}
```

Constructor

- A constructor allows us to **initialize an object's properties** upon creation of the object.
- A constructor is a **special member function** for automatic initialization of an object.
- The constructor function starts with **two underscores** (__).
- If you create a **__construct()** function, PHP will **automatically call** this function when you create an object from a class.
- Example:

```
class employee
{
   var $empno;
   var $ename;
  function __construct()
  {
      $this->empno = "101";
      $this->ename = "Sonali Shinde";
  }
}
$e = new employee();
```

Destructor

- A destructor is called when the **object is destructed** or **the script is stopped** or **exited.**
- The destructor function starts with **two underscores** (__).
- If you write a **__destruct()** function, PHP will **automatically call** this function at the end of the script.
- Example:

```
class employee
{
  var $empno;
  var $ename;

  function __construct()
  {
      $this->empno = "101";
      $this->ename = "Sonali Shinde";
  }

function __destruct()
{
    echo "Emplyee number=", $this->empno;
    echo "Emplyee Name=", $this->ename;
}
}
$e = new employee();
```

Static Properties

- Static properties can be called **directly** - **without creating an instance** of a class.
- Static properties are declared with the **static keyword**.
- To access a static property use the **class name, double colon (::),** and the property name.
- Syntax : class_name :: property_name;
- **Characteristics of Static Properties** :
  - It is initialized to **zero** when the **first object** of its class is created.
  - Only **one copy** of that member is created for the **entire class** and is **shared** by all the objects of that class, **no matter how many objects** are created.
  - Static variables are normally used to **maintain** values **common** to the entire class.

Static Properties Example

- Example:

```
<?php
class summation
{
        public static $s=0;
}
echo “Static Variable is ”, summation::$s;
?>
```

```
class StaticExample
{
        public static $cnt;
        var $s;
        function get($n)
        {
                $this->s = $n;
                self::$cnt=self::$cnt+1;
        }
        function display()
        {
                echo "<br>Non Static value=",$this->s;
                echo "<br>Static value=",self::$cnt;
        }
}
$s1 = new StaticExample();
$s1->get(4); $s1->display();
$s2 = new StaticExample();
$s2->get(41); $s1->display();
```

Static Methods

- Static methods can be called **directly** - without **creating an instance of a class**.
- Static methods are declared with the **static** keyword.
- To access a static method use the **class name**, double colon (::), and the method name.
- Syntax : class_name :: method_name();
- A static methods can have access to **only other static members** declared in the same class.
- Example:

```
class StaticExample
{
        static function display()
        { echo "Example of static method!!!";}
}
StaticExample::display() ;
```

Static Methods Example

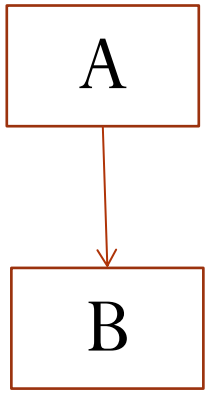
```
class StaticExample
{
        public static $cnt;
        var $s;
        function get($n)
        {
                $this->s = $n;
                self::$cnt=self::$cnt+1;
        }
        static function displayS()
        { echo "<br>Static value=",self::$cnt; }
        function display()
        { echo "<br>Non Static value=",$this->s; }
}
$s1 = new StaticExample();
$s1->get(4); StaticExample ::displayS();    $s1->display();
$s2 =new StaticExample();
$s2->get(41);   $s2::displayS();    $s2->display();
```

Inheritance

- The mechanism of creating new classes from an existing class is called a **inheritance.**
- The old class or existing class is known as **base class or parent class or superclass**
- The newly created class is called as **derived class or child class or subclass**. To create derived class **extends** keyword is used.
- In some OOP languages, the base and derived classes are called as super and subclasses respectively.

Base class/Super class/ Parent class

Derived class/Sub class / Child class




```
class A
{
        //definition of class A
}
class B extends A
{
        //definition of class B
}
```

Advantages and Disadvantages of Inheritance

Advantages of Inheritance

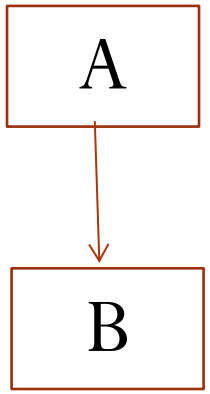
1. **Reusability** - Facility to use public methods of base class without rewriting the same.
2. **Extensibility** - Extending the base class logic as per business logic of the derived class.
3. **Data hiding** - Base class can decide to keep some data private so that it cannot be altered by the derived class
4. **Overriding** -With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Disadvantages of Inheritance

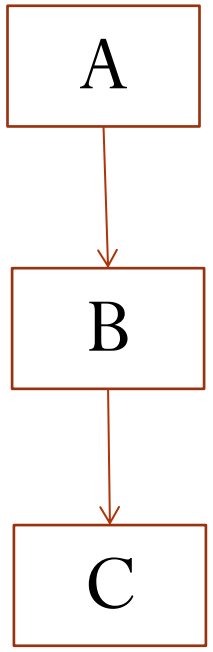
1. Inherited functions work **slower** than normal function as there is indirection.
2. **Improper** use of inheritance may lead to **wrong solutions**.
3. Often, data members in the base class are left **unused** which may lead to **memory wastage**.
4. Inheritance increases the **coupling** between base class and derived class. A change in base class will **affect** all the child classes.

Types of Inheritance

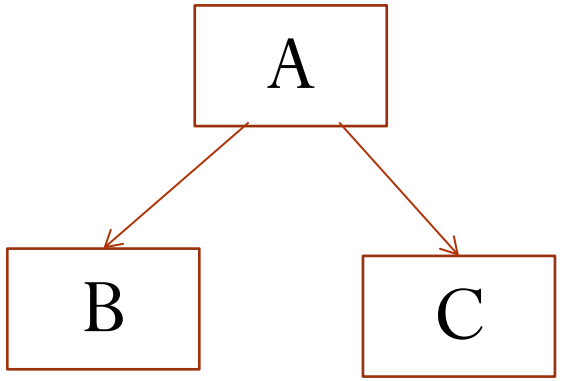
1. Single Inheritance



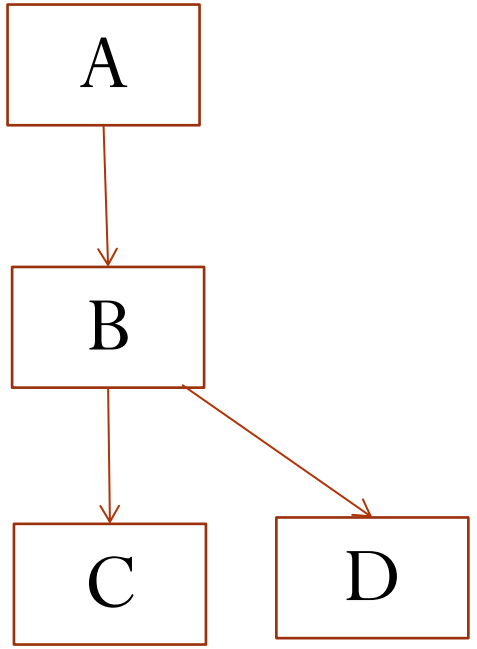
2. Multilevel Inheritance



3. Hierarchical Inheritance



4. Hybrid Inheritance



Function Overriding

- **Function overriding** is a feature that allows us to have a **same function** in **child class** which is already present in the parent class.
- A child class **inherits** the data members and member functions of parent class, but when you want to override a functionality in the child class then you can use **function overriding**.
- It is like creating a **new version** of an old function, in the child class.
- To override a function you must have the **same signature** in child class. Same signature means same **data type and sequence of parameters**.

Parent:: & self:: keyword

- **parent::** keyword used for the parent class and it is mostly used when you want to call the **parent properties or methods**. It also is used to access members and methods.
- **self ::** keyword used for the current class and basically it is used to access static members, methods.
- **Difference between $this, self::, parent::**
- **$this :** $this refers to the object. $this point to the **current object** instance does not point to any other object or class.
- **self :** self:: refer to the class itself, not point to any object. Using **self::** you can access the **static properties and static methods** of the current class.
- **parent:** parent:: is refer to the parent class. Using **parent::,** you can access the **static properties and static methods** of the parent class.
- With the use of self and parent, you allow to avoid explicitly reference the class by name.

Abstract class and method

- Abstract class is a **super class** that only defines a **generalised form** which will be shared by all its **sub classes**.
- An abstract class is a class that contains **at least one abstract method**. It can contains abstract as well as non-abstract method.
- We can not use abstract classes to **instantiate object**.
- An abstract method is a method that is declared in **abstract class** and defined in sub classes.
- An abstract class or method is defined with the **abstract** keyword.
- **Example:**

```
abstract class shape
{
   abstract public function area();
}
```

Final keyword

- The **final** keyword is used in PHP for **methods and classes**.
- The final for methods **prevent method overriding**. A method that is declared final **cannot be overridden** in a subclass.
- The final for class **prevent Inheritance**. A final class can not be sub classed (extended) by any other class.
- **Example of final class:**

```
final class ArithmeticOperation
{
   public function add()
   {
         //Code of addition
   }
}
```

- **Example of final method:**

```
class ArithmeticOperation
{
   final public function add()
   {//Code of addition
   }
}
```

Interface

- **Interfaces** allow you to specify what methods a class should **implement**.
- Interfaces make it easy to use a **variety of different classes** in the same way. When one or more classes use the same interface, it is referred to as "**polymorphism**".
- Single class can implement **one or more interfaces**.
- Interfaces are declared with the **interface** keyword.
- 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.
- **Syntax:**

```
interface interface_name
{
   public function function_name();
}
class class_name implements interface_name
{ }
```

Difference between Interface and Abstract class

Interface	Abstract class
Interfaces cannot have properties (data members).	Abstract classes can have properties (data members).
All interface methods must be **public**.	Abstract class methods may be **private, protected or public**
All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary.	Abstract keyword is used to declare abstract method in the abstract class.
Classes **implements** an interface while inheriting from another class.	Classes **extends** abstract class while inheriting from another class.
A class can implements more than one interfaces.	A class can extends only one abstract class.

instanceof operator

- instanceof operator is used to check whether an object is an **instance of the class or not**.
- This can be useful to **controlling** objects in large applications as you can make sure that a parameter is a particular **instance** of an object before using it.
- The **first (left) parameter** is the **object** to test. If this variable is not an object, instanceof always returns false.
- The **second (right) parameter** is the class to compare with. The class can be provided as the class name itself, a string variable containing the class name or an object of that class.

instanceof operator Example

```
<?php
class MyClass
{

}
$ob1 = new MyClass();
$ob2 = new MyClass();
$name = 'MyClass'; // in the cases below, $a gets boolean value true

$a = $ob1 instanceof MyClass;
$a = $ob1 instanceof $name;
$a = $ob1 instanceof $ob2;
?>
```

Thank you