**OOPS CONCEPTS IN PHP**

PHP is a server-side scripting language, mainly used for [web development](https://www.valuebound.com/services/drupal-web-development-services) but also used as a general-purpose programming language. Object-Oriented Programming (PHP OOP),  is a type of programming language principle added to php5, that helps in building complex, reusable web applications.

The  PHP Object-Oriented Programming concepts are:

* Class
* Objects
* Inheritance
* Interface
* Abstraction
* Magic Methods

**Class  & Object:**

* Class is a programmer-defined data type, which includes local methods and local variables.
* Class is a collection of objects. Object has properties and behaviour.
* First we have to define a php class, where classname should be same as filename.

**Example for simple class:**

|  |  |
| --- | --- |
|  | <?php |
|  | class Books{ |
|  | public function name(){ |
|  | echo “Drupal book”; |
|  | } |
|  | public function price(){ |
|  | echo “900 Rs/-”; |
|  | } |
|  | } |
|  | To create php object we have to use a  new operator. Here php object is the object of the Books Class. |
|  | $obj = new Books(); |
|  | $obj->name(); |
|  | $obj->price(); |
|  | ?> |

**Output:  
Drupalbook  
900 Rs/-**

In the basics of object-oriented, let see how to define a class and create an object:

**Creating Objects in PHP**  
When class is created, we can create any number of objects in that class. The object is created with the help of the new keyword.

**Calling Member Function**  
When the object is created we can access the variables and method function of the class with the help of operator ‘->, accessing the method is done to get the information of that method. Also, look into [how we can access object properties via variables](https://www.valuebound.com/resources/blog/til-access-object-properties-variable)

|  |  |
| --- | --- |
|  | <?php |
|  | class Mobile { |
|  | /\* Member variables \*/ |
|  | var $price; |
|  | var $title; |
|  | /\* Member functions \*/ |
|  | function setPrice($par){ |
|  | $this->price = $par; |
|  | } |
|  | function getPrice(){ |
|  | echo $this->price ."<br/>"; |
|  | } |
|  | function setName($par){ |
|  | $this->title = $par; |
|  | } |
|  | function getName(){ |
|  | echo $this->title ." <br/>"; |
|  | } |
|  | } |
|  | $Samsung = new Mobile(); |
|  | $Xiaomi = new Mobile(); |
|  | $Iphone = new Mobile(); |
|  | $Samsung->setName( "SamsungS8 ); |
|  | $Iphone->setName( "Iphone7s" ); |
|  | $Xiaomi->setName( "MI4" ); |
|  | $Samsung->setPrice( 90000 ); |
|  | $Iphone->setPrice( 65000 ); |
|  | $Xiaomi->setPrice( 15000 ); |
|  | Now you call another member functions to get the values set by in above example |
|  | $Samsung->getName(); |
|  | $Iphone->getName(); |
|  | $Xiaomi->getName(); |
|  | $Samsung->getPrice(); |
|  | $Iphone->getPrice(); |
|  | $Xiaomi->getPrice(); |
|  | ?> |

**Output for the above code**

**Samsung S8  
Iphone S7  
MI4  
90000  
65000  
15000**

**Inheritance**

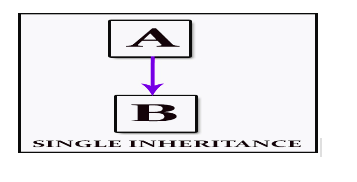
When the properties and the methods of the parent class are accessed by the child class, we call the concept has inheritance. The child class can inherit the parent method and give own method implementation, this property is called overridden method. When the same method of the parent class is inherited we call as inherited method. Now let us see types of inheritance supported in Object Oriented Programming and corresponding Php inheritance examples.

**Types Of Inheritance**

1. Single Level Inheritance
2. Multilevel Inheritance

**Single Level Inheritance:**  In Single Level Inheritance the Parent class methods will be extended by the child class. All the methods can be inherited.

**Single Level Inheritance**

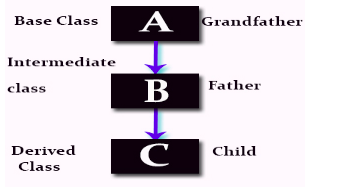


**Example for Single Level Inheritance**

|  |  |
| --- | --- |
|  | <?php |
|  | class A { |
|  | public function printItem($string) { |
|  | echo ' Hi : ' . $string; |
|  | } |
|  | public function printPHP() { |
|  | echo 'I am from valuebound' . PHP\_EOL; |
|  | } |
|  | } |
|  | class B extends A { |
|  | public function printItem($string) { |
|  | echo 'Hi: ' . $string . PHP\_EOL; |
|  | } |
|  | public function printPHP() { |
|  | echo "I am from ABC"; |
|  | } |
|  | } |
|  | $a = new A(); |
|  | $b = new B(); |
|  | $a->printItem('Raju'); |
|  | $a->printPHP(); |
|  | $b->printItem('savan'); |
|  | $b->printPHP(); |
|  | ?> |

**Output  
Hi : Pavan  
I am from valuebound  
Hi: savan  
I am from ABC**

**MultiLevel Inheritance :**  In MultiLevel Inheritance, the parent class method will be inherited by child class and again subclass will inherit the child class method.



|  |  |
| --- | --- |
|  | <?php |
|  | class A { |
|  | public function myage() { |
|  | return ' age is 80'; |
|  | } |
|  | } |
|  | class B extends A { |
|  | public function mysonage() { |
|  | return ' age is 50'; |
|  | } |
|  | } |
|  | class C extends B { |
|  | public function mygrandsonage() { |
|  | return 'age is 20'; |
|  | } |
|  | public function myHistory() { |
|  | echo "Class A " .$this->myage(); |
|  | echo "Class B ".$this-> mysonage(); |
|  | echo "Class C " . $this->mygrandsonage(); |
|  | } |
|  | } |
|  | $obj = new C(); |
|  | $obj->myHistory(); |
|  | ?> |

[**view raw**](https://gist.github.com/pavanBS/c04a91c76ae910367b264bee14f7b5c4/raw/32c6cad17617308fcd1efac381bbc48cce04fdc9/A.php)[**A.php**](https://gist.github.com/pavanBS/c04a91c76ae910367b264bee14f7b5c4#file-a-php)hosted with ❤ by [**GitHub**](https://github.com/)

**Output**

**Class A is 80  
Class B is 50   
Class C 20**

**INTERFACES:**

* An interface is a description of the actions that an object can do.
* Interface is written in the same way as the class the declaration with interface keyword.

**Rules of Interfaces:**

* All methods declared in an interface must be public; this is the nature of an interface.
* All methods in the interface must be implemented within a class; failure to do so will result in a fatal error.
* The class implementing the interface must use the exact same method signatures as are defined in the interface
* Interfaces can be extended like classes using the extends operator.

**Example for the interface class**

|  |  |
| --- | --- |
|  | <?php |
|  | interface A { |
|  | public function setProperty($x); |
|  | public function description(); |
|  | } |
|  | class Mangoes implements A { |
|  | public function setProperty($x) { |
|  | $this->x = $x; |
|  | } |
|  | public function description() { |
|  | echo 'Describing' . $this->x . tree; |
|  | } |
|  | } |
|  | $Mango = new Mangoes(); |
|  | $Mango->setProperty(mango); |
|  | $Mango->description(); |
|  | ?> |

**Output:  
Describing Mango tree**

2) Interface can be extended with another interface using extends keyword

|  |  |
| --- | --- |
|  | <?php |
|  | interface A { |
|  | public function Compute(); |
|  | } |
|  | interface B extends A { |
|  | public function Divide(); |
|  | } |
|  | class C implements B { |
|  | public function Divide() { |
|  | $var=10; |
|  | $var1=2; |
|  | $var3=$var/$var1; |
|  | echo “division of 10/2 is” . $var3; |
|  | } |
|  | public function Compute() { |
|  | $a=2; |
|  | $b=3; |
|  | $c=$a\*$b; |
|  | echo “multiplication of 2\*3 is” . $c; |
|  | } |
|  | } |
|  | $obj = new C(); |
|  | $obj->Divide(); |
|  | $obj->Compute(); |
|  | ?> |

**Output:  
division of 10/2 is 5  
multiplication of 2\*3 is 6**

**Note on Interfaces:-**

* We cannot create objects to interface, but the class implementing the interface can have objects
* We cannot define a variable in an interface.
* If we extend interface all the methods of the interface must be implemented in the child class.

**Abstract Classes:**

* An abstract class is a class that contains at least one abstract method. The abstract method is function declaration without anybody and it has the only name of the method and its parameters.
* There can be any number of methods in the class and we have to declare the class as abstract only when there is an abstract method

**Example for Abstract class**

|  |  |
| --- | --- |
|  | <?php |
|  | abstract class Cars { |
|  | public abstract function getCompanyName(); |
|  | public abstract function getPrice(); |
|  | } |
|  | class Baleno extends Cars { |
|  | public function getCompanyName() { |
|  | return "Maruthi Suzuki" . '<br/>'; |
|  | } |
|  | public function getPrice() { |
|  | return 720000 . '<br/>'; |
|  | } |
|  | } |
|  | class Santro extends Cars { |
|  | public function getCompanyName() { |
|  | return "Hyundai" . '<br/>'; |
|  | } |
|  | public function getPrice() { |
|  | return 300000 . '<br/>'; |
|  | } |
|  | } |
|  | $car = new Baleno(); |
|  | $car1 = new Santro(); |
|  | echo $car->getCompanyName(); |
|  | echo $car->getPrice(); |
|  | echo $car1->getCompanyName(); |
|  | echo $car1->getPrice(); |
|  | ?> |

[**view raw**](https://gist.github.com/pavanBS/9ed1543c46684f2d579b7a4a83cff06c/raw/80319b87fac0eb0cb77fed08faf9dcba2f141122/cars.php)[**cars.php**](https://gist.github.com/pavanBS/9ed1543c46684f2d579b7a4a83cff06c#file-cars-php)hosted with ❤ by [**GitHub**](https://github.com/)

**Output for the above code is:**  
**Maruthi Suzuki  
720000  
Hyundai  
300000**

**Notes on Abstract classes:**

* Objects cannot be created for the abstract classes.
* If a class has only one method as abstract, then that class must be an abstract class.
* The child class which extends an abstract class must define all the methods of the abstract class.
* If the abstract method is defined as protected in the parent class, the function implementation must be defined as either protected or public, but not private.
* The signatures of the methods must match, optional parameter given in the child class will not be accepted and error will be shown.
* Abstract classes that declare all their methods as abstract are not interfaces with different names. One can implement multiple interfaces, but not extend multiple classes (or abstract classes).

Now Let us see the difference between abstract class and interface.

|  |  |
| --- | --- |
| Abstract class | Interface |
| It can have constants, members, method stubs (methods without a body), methods | It can only have constants and methods stubs. |
| Methods and members can have public or protected  visibility | Methods of interface should only be public not any other visibility |
| The concept of multiple inheritances not supported. | An interface can extend or a class can implement multiple other interfaces. |
| Child class must implement all the abstract method of parent class when extend keyword is used. | No need of implementing methods from parent interface when interface  is extending another interface |