OOPS Interview Questions and Answers for Experienced In PHP

Author: Full Stack Tutoorials

PHP OOPs Interview Questions: OOPs Concepts in PHP | OOPS Interview Questions and Answers for Experienced In PHP | Object Oriented Programming in PHP

Q:- What is Object Oriented Programming?
Object oriented programming is a programming technique to design your application. Application can be of any type like it can be web based application, windows based application etc.

> In object oriented programming, everything revolves around the objects and class.

Q:- What is a class?
A class is a template for an object, a user-defined datatype that contains variables, properties, and methods.

Class represents all properties and behaviors of object.

class Person{ public $name; public $age; function __construct($name, $age){ $this->name = $name; $this->age = $age; } function getUserDetails(){ return "Hi, My Name is ".$this->name." and I'm ". $this->age ." old

"; } } //To create php object we have to use a new operator. $obj = new Person("Ajay", 23); echo $obj->getUserDetails(); //Output: Hi, My Name is Ajay and I'm 23 old

Q:- What is an object?

Objects are created from Classes, is an instance of a class that is created dynamically.

Object in programming is similar to real word object. Every programming object has some properties and behaviors.

You can create object of class with the help of **new** keyword

//Create an object of MyClass $obj = new MyClass(); OR $obj = new MyClass;

You may also like - Node Js Interview Questions Answers

Q:- What is the relation between Classes and Objects?

They look very much same but are not same.

1. A class is a definition, while an object is an instance of the class.
2. A class is a blueprint while objects are actual objects existing in the real world.

Suppose we have a class Person which has attributes and methods like name, age, height, weight, color etc.

Class Person is just a prototype, now we can create real-time objects of class Person.

#Example: Ajay is real time object of class **Person**, which have name=Ajay, age=23, height=170cm, weight=60kg and color=black etc.

Class

1. A way to bind data and associated functions together.
2. Class have many objects.
3. Class is a template for creating objects.
4. It is logical existence.
5. Memory space is not allocated, when it is created.
6. Definition (Declaration) is created once.
7. Class is declared using "class" keyword.

Object

1. Basic runtime entity in object oriented environment.
2. Object belongs to only class.
3. Object are a implementation of class.
4. It is physical existence.
5. Memory space is allocated when it is created.
6. It is created many times as you required.
7. Object is the instance or variable of class.

Q:- What is Constructor and Destructor?
Constructor:
Constructor is a special type of function which will be called automatically whenever there is any object created from a class.

```
//Old (PHP4) class myClass{ function myClass(){ // Define logic in constructor } }
//New (PHP5+) class myClass{ function __construct(){ // Define logic in constructor } }
```

> Note- Old style constructors are DEPRECATED in PHP 7.0, and will be removed in a future version. You should always use __construct() in new code.

Destructor:
Destructor is a special type of function which will be called automatically whenever any object is deleted or goes out of scope.

```
class myClass{ function __construct(){ // Define logic in constructor } function __destruct(){ // this is destructor } }
```

Types of constructors:

1. Default constructor
   A constructor without any parameters is called a default constructor.

2. Parameterized constructor
   A constructor with at least one parameter is called a parametrized constructor.

   ```
   class Person{ public $name; public $address; public function __construct($name){ // parameterized constructor with name argument
   ```

$this->name = $name; } } $perObj1 = new Person("Full Stack Tutorials"); // parameterized constructor with name argument echo $perObj1->name; //Output: Full Stack Tutorials

3. Copy Constructor
   class Person{ public $name; public $address; public function __construct($name){ // parameterized constructor with name argument $this->name = $name; } public function __clone(){ } } $perObj1 = new Person("Full Stack Tutorials"); // parameterized constructor with name argument $perObj2 = clone $perObj1; // copy constructor initialize $perObj2 with the values of $perObj1 echo $perObj2->name; //Output: Full Stack Tutorials
4. Static Constructor
5. Private Constructor
   Purpose of Private Constructor: It ensures that there can be only one instance of a Class and provides a global access point to that instance and this is common with The Singleton Pattern.

   class myClass { private static $instance; private function __construct() { } public static function get_instance() { if (!self::$instance) self::$instance = new myClass(); return self::$instance; } }

Q:- What is Member Variable and Member function?
Member Variable − These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

Member function − These are the function defined inside a class and are used to access object data.

Q:- What is different types of Visibility? OR What are access modifiers?
Each method and property has its visibility. There are three types of visibility in PHP.

Types of visibility:

1. **public:** Public method or variable can be accessible from anywhere, Means a public method or variable of a class can be called outside of the class or in a subclass.
2. **protected:** A protected method or variable can only be called in that class & it's subclass.

3. **private:** A private method or variable of a class can only be called inside that class only in which it is declared.

NOTE: By default, in PHP, a class member is public unless declared private or protected.

Q:- What is Encapsulation?

Wrapping up member variables and methods together into a single unit (i.e. Class) is called Encapsulation.

1. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them.
2. Visibility is the mechanism for encapsulation.

class Person { private $name; public function setName($name) { $this->name = $name; } public function getName($name) { return $this->name; } } $personObj = new Person(); $personObj->setName('Full Stack Tutorials'); $personObj->getName();

You may also like - Core PHP Interview Questions Answers

Q:- What is Abstraction?

Abstraction is a concept in which implementation details are hidden.

Abstract Class:

Abstract class are class which contains atleast one or more abstract method.

Abstract Method:

Abstract method is a method which is declared, but not defined.

1. PHP 5 introduces abstract classes and methods.
2. Classes defined as abstract may not be instantiated
3. Classes that contains at least one abstract method must also be abstract.

4. Methods defined as abstract simply declare the method's signature - they cannot define the implementation. Abstract methods cannot be defined as private.
5. Classes which are inheriting it's parent class must provides implementations for the abstract methods.

abstract class TV { private $isOn = false; abstract function getBrand(); public function turnOnTV() { $this->isOn = true; } public function turnOffTV() { $this->isOn = false; } } class Panasonic extends TV { public function getBrand(){ return "Panasonic"; } } class Sony extends TV { public function getBrand(){ return "Sony"; } }

Q:- Can Class Properties be Abstract?

No. there is no need for a class property to be abstract. Class properties and constants are not implemented, instead, they are declared, which in a way already makes them abstract. They follow the same rules and are treated the same way when declared in an abstract class as they would any other class.

Note: All the remaining classes which are not abstract are called **Concrete Classes**

Q:- What is the need of abstract class?

Suppose we were modeling the behavior of animals, by creating a class hierarchy that started with a base class called Animal.

Animals are capable of doing different things like flying, digging and walking, but there are some common operations as well like eating and sleeping.

Some common operations are performed by all animals, but in a different way as well.

When an operation is performed in a different way, it is a good candidate for an abstract method (forcing subclasses to provide a custom implementation).

Q:- Explain about polymorphism?

It is simply "One thing, can use in different forms". Technically, it is the ability to redefine methods for derived classes.

#Example: One Class (Car) can extend two classes (Audi & BMW).

> Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.

You may also like - MYSQL Interview Questions Answers

Q:- Types of Polymorphism?
Polymorphism could be static and dynamic both. Overloading is static polymorphism while, overriding is dynamic polymorphism.

1. Compile time polymorphism (Static) - Method Overloading
2. Runtime time polymorphism (Dynamic) - Method Overriding

1. **Overloading** is defining functions/methods that have same signatures with different parameters in the same class.
2. **Overriding** is redefining parent class functions/methods in child class with same signature. So, basically the purpose of overriding is to change the behavior of your parent class method.

The overloading methods are invoked when interacting with properties or methods that have not been declared or are not visible in the current scope. The rest of this section will use the terms "inaccessible properties" and "inaccessible methods" to refer to this combination of declaration and visibility.

//Example: Method Overloading Since PHP doesn't support overloading. In PHP, you can only overload methods using the magic method __call()
Magic Methods in PHP

//Example: Method Overriding class A { function testFunc() { return "I am inside class A"; } } class B extends A { function testFunc() { return "I am inside class B"; } } $objA = new A; $objB = new B; echo($objA->testFunc()); //"I am inside class A" echo($objB->testFunc()); //"I am inside class B" //Output: I am inside class A I am inside class B
Polymorphism using Interfaces & Abstract Class

An interface is an agreement or a contract. When a class implements an interface, It means it contains same public methods of an interface with their implementation.

In order to implement the polymorphism principle, we can choose between abstract classes and interfaces.

In the example given below, the interface with the name of Shape commits all the classes that implement it to define an abstract method with the name of calcArea().

interface Shape { public function calcArea(); }
In accordance, the **Circle** class implements the interface **Shape** by putting into the calcArea() method the formula that calculates the area of circles.

class Circle implements Shape { private $radius; public function __construct($radius){ $this->radius = $radius; } // calcArea calculates the area of circles public function calcArea(){ return $this->radius * $this->radius * pi(); } }
The **Rectangle** class also implements the interface **Shape** but defines the method calcArea() with a calculation formula that is suitable for rectangles:

class Rectangle implements Shape { private $width; private $height; public function __construct($width, $height){ $this->width = $width; $this->height = $height; } // calcArea calculates the area of rectangles public function calcArea(){ return $this->width * $this->height; } }
Now, we can create objects from the concrete classes:

$cirObj = new Circle(3); $rectObj = new Rectangle(2,4);
We can be sure that all of the objects calculate the area with the method that has the name of calcArea(), whether it is a rectangle object or a circle object (or any other shape), as long as they implement the Shape interface.

Now, we can use the calcArea() methods to calculate the area of the shapes:

echo $cirObj->calcArea(); //Output: 28.274 echo $rectObj->calcArea(); //Output: 8
Note: Polymorphism enhances the flexibility and reusability of the application

Q:- What is the key difference between concrete class and abstract class?
Concrete classes are those classes which has to declare body of abstract methods which extends or implements from abstract class or interface

OR
Abstract classes usually have partial or no implementation. On the other hand, Concrete classes always have full implementation of its behavior. Unlike Concrete classes, Abstract classes cannot be instantiated.

Q:- What is the difference between Abstract class and Interface?

Abstract class	**Interface**
In abstract class a method must be declared as abstract. Abstract methods doesn't have any implementation.	In interface all the methods by default are abstract.

Abstract class	Interface
Abstract class can also contain member variables and concrete functions/methods.	Interfaces cannot contain any member variables and concrete functions/methods except constants.
An Abstract methods can be declare with access modifiers like public, protected etc. Concrete Class which is extending the abstract class must be defined with the same or visibility.	All methods declared in an interface must be public.
A class can Inherits only one Abstract class and Multiple inheritance is not possible for Abstract class.	A class can implements many interfaces and Multiple interface inheritance is possible.
Only complete member of abstract class can be static.	Memebrs of Interface can not be static.

Q:- When to use abstract class and interface in PHP? Explain with real world Example?
Abstract class:

> Abstract Class is used when you something you know (Concrete Methods) and something which you don't know (Abstract Methods).

Interface:

> Interface is used when you don't know anything about implementation. (All methods are abstract)

Now, Let's understand it with real world example:
Consider we are developing a flights booking service.

1. Displaying flights available from vendors like "Air India", "IndiGO" and "British Airways" etc
2. Place an order for seat to respective vendor.

Remember, In this application, we don't own any flights. we are just a service provider mediator.

So, basically we need two methods:

1. checkFlightsAvailability() - to check available flights from all filghts service comapny.
2. bookFlights() - to book flights from the availbale flights service company.

So finally, Interface is useful because we are not aware of the actual implementation of all above methods required, we just only know the contract methods that vendor(implementer) should provide.

interface flightServices { public function checkFlightsAvailability(); public function bookFlights(); } class AirIndia implements flightServices { public function checkFlightsAvailability(){ //Get List of available flights } public function bookFlights(){ //Book available flights } } class IndiGO implements flightServices { public function checkFlightsAvailability(){ //Get List of available flights } public function bookFlights(){ //Book available flights } } class BritishAirways implements flightServices { public function checkFlightsAvailability(){ //Get List of available flights } public function bookFlights(){ //Book available flights } }

Q:- What is final keyword?

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with the final. If the class itself is being defined final then it cannot be extended.

Example #1 Final methods example

class BaseClass { public function test() { echo "BaseClass::test() called\n"; } final public function moreTesting() { echo "BaseClass::moreTesting() called\n"; } } class ChildClass extends BaseClass { public function moreTesting() { echo "ChildClass::moreTesting() called\n"; } } // Results in Fatal error: Cannot override final method BaseClass::moreTesting()

Example #2 Final class example

final class BaseClass { public function test() { echo "BaseClass::test() called\n"; } // Here it doesn't matter if you specify the function as final or not final public function moreTesting() { echo "BaseClass::moreTesting() called\n"; } } class ChildClass extends BaseClass { } // Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)

Note: Properties cannot be declared final, only classes and methods may be declared as final.

Q:- What is STATIC keyword and what is it's use in PHP ?

Declaring class member variables or methods as static makes them accessible without needing an instantiation of the class. A member declared as static cannot be accessed with an instantiated class object (though a static method can).

1. Static Methods and Properties in PHP will be treated as public if no visibility is defined.
2. static can also be used to define static variables and for late static bindings.

Note:

- In PHP 5, calling non-static methods statically generates an E_STRICT level warning.
- In PHP 7, calling non-static methods statically is deprecated, and will generate an E_DEPRECATED warning. Support for calling non-static methods statically may be removed in the future.

#Example : Calling non-static method statically
error_reporting(E_ALL); // Incase if you have hidden error & warning message.
class testClass { public function sayHello() { echo "hello"; } } echo testClass::sayHello(); //Output: Deprecated: Non-static method testClass::sayHello() should not be called statically in C:\xampp\htdocs\test_projects\index.php on line 7 hello
#Static Example

class testClass{ public static function testMethod(){ echo "This is static method."; } } echo testClass::testMethod(); //Output: This is static method.

You may also like - MongoDB Interview Questions Answers

Q:- What is Traits in PHP?

Traits are a mechanism for code reuse in single inheritance languages such as PHP.

1. A Trait is intended to reduce some limitations of single inheritance by enabling a developer to reuse sets of methods freely in several independent classes living in different class hierarchies.
2. The semantics of the combination of Traits and classes is defined in a way which reduces complexity and avoids the typical problems

associated with multiple inheritance and Mixins.

3. A Trait is similar to a class, but only intended to group functionality in a fine-grained and consistent way.
4. It is not possible to instantiate a Trait but an addition to traditional inheritance. It is intended to reduce some limitations of single inheritance to reuse sets of methods freely in several independent classes living in different class hierarchies.
5. Multiple Traits can be inserted into a class by listing them in the use statement, separated by commas(,).
6. If two Traits insert a method with the same name, a fatal error is produced.

Note: Multiple traits can be used in a class by listing them in the use statement, separated by commas.

#Example - Multiple Traits Usage
trait Hello { public function sayHello() { echo 'Hello '; } } trait World { public function sayWorld() { echo 'World'; } } class MyHelloWorld { use Hello, World; public function sayExclamationMark() { echo '!'; } } $o = new MyHelloWorld(); $o->sayHello(); $o->sayWorld(); $o->sayExclamationMark(); //Output: Hello World!
#insteadof

> If two traits insert a method with the same name, a fatal error is produced, if the conflict is not explicitly resolved. To resolve naming conflicts between traits used in the same class, the **insteadof** operator needs to be used to choose exactly one of the conflicting methods.

Since this only allows one to exclude methods, the as operator can be used to add an alias to one of the methods. Note the as the operator does not rename the method and it does not affect any other method either.

#Example - Conflict Resolution
In this example, Talker uses the traits A and B. Since A and B have conflicting methods, it defines to use the variant of Smalltalk from trait B, and the variant of bigTalk from trait A.

The Aliased_Talker makes use of the as an operator to be able to use B's bigTalk implementation under an additional alias talk.

```
trait A { public function smallTalk() { echo 'a'; } public function bigTalk() { echo 'A'; } } trait B { public function smallTalk() { echo 'b'; } public function bigTalk() { echo 'B'; } } class Talker { use A, B { B::smallTalk insteadof A; A::bigTalk insteadof B; } } $obj = new Talker(); $obj->smallTalk(); $obj->bigTalk();
//Output: bA
```

Q:- What is Type Hinting in PHP?

> Type Hinting is used to specify the expected data type (arrays, objects, interface, etc.) for an argument in a function declaration.

It is beneficial because it results in better code organization and improved error messages.

Suppose we want to force a function to get only arguments of the type array, then we can simply put the array keyword before argument's name.

```
function myFunction(array $argument_name) { //block of code }
```

It is also know as Type Declaration.

Q:- What is Namespaces in PHP?

> Namespaces is used to avoid conflicting definitions and enables more flexibility and organization in your code base.

In PHP version 5.3 a new feature was added to the language known as namespace.

In PHP we can’t have two classes that have same name. Name should be be unique. The issue with this restriction is that if you are using any third party library which has a class named User, then you can’t create your own class with name User.

```
namespace Google; Class Search { public function query() { return 'Searching Google'; } } namespace Bing; Class Search { public function query() { return 'Searching Bing'; } }
```

You can import namespaces into anohter namespace or class using the "use" and "as" keywords.

Note: Namespace names are case-insensitive.

Q:- What is __NAMESPACE__ ?
This is a constant that refers to the current namespace. In the global namespace, this constant has no value, or an empty string.

Q:- How to count objects of a class in PHP

```
class countClassObjects{ public static $count = 0; public function __construct(){
self::$count++; } public function xyz(){ //code } } $obj1 = new
countClassObjects(); $obj2 = new countClassObjects(); $obj3 = new
countClassObjects(); $obj4 = new countClassObjects(); $obj5 = new
countClassObjects(); echo "The number of objects in the countClassObjects class
is " . countClassObjects::$count; //Output //The number of objects in the
countClassObjects class is 5
```

Q:- What are the advantages of object oriented programming?

1. **Code Resusability:** it can be acheived through inheritance and traits
2. **Modularity:** it can be acheived through breaking large code into small modules, Modularity reduces complexity.
3. **Flexibility:** it can be acheived through polymorphism
4. **Maintainability:** it is to maintian code which follow Object Oriented Programming Concepts.
5. **Security:** it can be acheived through Encapsulation
6. **Testability:** it is easy to test.

This is all about PHP OOPs Interview Questions | OOPs Concepts in PHP