**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.

* **code**

1. class Person{
2. public $name;
3. public $age;
4. function \_\_construct($name, $age){
5. $this->name = $name;
6. $this->age = $age;
7. }
8. function getUserDetails(){
9. return "Hi, My Name is ".$this->name." and I'm ". $this->age ." old <br>";
10. }
11. }
12. *//To create php object we have to use a new operator.*
13. $obj = new Person("Ajay", 23);
14. echo $obj->getUserDetails();
15. *//Output:*
16. 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

* **code**

1. *//Create an object of MyClass*
2. $obj = new MyClass();
3. OR
4. $obj = new MyClass;

**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.

* **code**

1. *//Old (PHP4)*
2. class myClass{
3. function myClass(){
4. *// Define logic in constructor*
5. }
6. }

* **code**

1. *//New (PHP5+)*
2. class myClass{
3. function \_\_construct(){
4. *// Define logic in constructor*
5. }
6. }

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.

* **code**
* source

1. class myClass{
2. function \_\_construct(){
3. *// Define logic in constructor*
4. }
5. function \_\_destruct(){
6. *// this is destructor*
7. }
8. }

**Types of constructors:**

1. Default constructor

A constructor without any parameters is called a default constructor.

1. Parameterized constructor

A constructor with at least one parameter is called a parameterized constructor.

* + **code**
  + 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

1. Copy Constructor
   * **code**
   * 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
2. Static Constructor
3. 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.

* + **code**
  + source
  + 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.

can be accessible everywhere in child or sub child class.

1. **protected:** A protected method or variable can only be called in that class & it's subclass.

- cannot access the direct any property or method which is protected like following

$child2->child1ProtectedFun1(); // Not allowed

$chid2->child1ProtectedProp1; // Not allowed

-> you can call it from another public function

$child2->child2PublicFun1();

public function child2PublicFun1()

{

$this->child1ProtectedFun1(); // allowed

$this->child1ProtectedProp1; // allowed

}

-> cannot be overwritten any protected method or property; (check following)

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

- cannot call from another public function too. It can be called in a public function from the main class. check the follwing

<?php

class Bar

{

public function test() {

$this->testPrivate();

$this->testPublic();

$this->testProtected();

}

public function testPublic() {

echo "Bar::testPublic\n";

}

private function testPrivate() {

echo "Bar::testPrivate\n";

}

protected function testProtected() {

echo "Foo::testProtected\n";

}

}

class Foo extends Bar

{

public function testPublic() {

echo "Foo::testPublic\n";

}

protected function testProtected() {

echo "Foo::testProtected\n";

}

private function testPrivate() {

echo "Foo::testPrivate\n";

}

}

$myFoo = new Foo();

$myFoo->test();

// Bar::testPrivate

// Foo::testPublic

?>

**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.

* **code**

1. class Person {
2. private $name;
3. public function setName($name) {
4. $this->name = $name;
5. }
6. public function getName($name) {
7. return $this->name;
8. }
9. }
10. $personObj = new Person();
11. $personObj->setName('Full Stack Tutorials');
12. $personObj->getName();

**Advantages of Encapsulation:**

**Data Hiding and Abstraction:** Unnecessary details, internal representation and implementation are hidden from the end-users for protection of data structure and the Class. Data access is prohibited from members of other classes by creating private methods. It protects the internal state of any object by keeping member variables private and preventing any inconsistent state. It is the enclosing of data and related operations into that object.

Note: Encapsulation is used to hide internal views from the client.

**Data security:** Encapsulation helps in making data very robust and secured, as the data and member functions are wrapped together to form an object. All the tasks are done inside without any external worry and it also makes life very easy.

**Reduces complexity:** Encapsulation helps in reducing development complexity of the software by hiding the details of implementation and exposing the methods or operations.

**Reusability:** There are instances, you don’t have to re-write same functionality that you inherited from the parent class.

Reliability: You can make the class read-only or write-only by writing SET or GET methods.

**Easier testing of code:** Encapsulated PHP code is easy to test as the functions used for testing child class ensures the testing of parent class functions also.

**Increased flexibility:** Class variables can be accessed by GET or SET methods increasing flexibility. It is easy to maintain as the internal implementation can be changed without changing the code.

**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.

* **code**
* source

1. abstract class TV {
2. private $isOn = false;
3. abstract function getBrand();
4. public function turnOnTV() {
5. $this->isOn = true;
6. }
7. public function turnOffTV() {
8. $this->isOn = false;
9. }
10. }
11. class Panasonic extends TV {
12. public function getBrand(){
13. return "Panasonic";
14. }
15. }
16. class Sony extends TV {
17. public function getBrand(){
18. return "Sony";
19. }
20. }

**Example 2**

<?php

abstract class AbstractClass

{

// CANNOT SET ANY METHOD AS PRIVATE

abstract protected function getValue();

abstract protected function prefixValue($prefix);

abstract public function pubFun1();

// Common method

public function printOut() {

print $this->getValue() . "\n";

}

}

class ConcreteClass1 extends AbstractClass

{

public function getValue() {

return "ConcreteClass1";

}

public function prefixValue($prefix) {

return "{$prefix}ConcreteClass1";

}

// FATAL ERROR - PUBLIC METHOD CAN NOT BE PUBLIC. ONLY WEAKER ACCESS SPECIFIER CAN BE SET

protected function pubFun1($prefix) {

return "Protected";

}

}

$class1 = new ConcreteClass1;

$class1->printOut();

echo $class1->prefixValue('FOO\_') ."\n";

?>

**Example 3**

<?php  
abstract class AbstractClass  
{  
    // Our abstract method only needs to define the required arguments  
    abstract protected function prefixName($name);  
  
}  
  
class ConcreteClass extends AbstractClass  
{  
  
    // Our child class may define optional arguments not in the parent's signature  
    public function prefixName($name, $separator = ".") {  
        if ($name == "Pacman") {  
            $prefix = "Mr";  
        } elseif ($name == "Pacwoman") {  
            $prefix = "Mrs";  
        } else {  
            $prefix = "";  
        }  
        return "{$prefix}{$separator} {$name}";  
    }  
}  
  
$class = new ConcreteClass;  
echo $class->prefixName("Pacman"), "\n";  
echo $class->prefixName("Pacwoman"), "\n";  
?>

**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**

<https://www.php.net/manual/en/language.oop5.abstract.php>

**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.

**Polymorphism using Interfaces & Abstract Class**

**Step 1**

class poly\_base\_Article {

    public $title;

    public $author;

    public $date;

    public $category;

    public function  \_\_construct($title, $author, $date, $category = 0) {

        $this->title = $title;

        $this->author = $author;

        $this->date = $date;

        $this->category = $category;

    }

  public function write($type) {

        $ret = '';

        switch($type) {

            case 'XML':

                $ret = '<article>';

                $ret .= '<title>' . $obj->title . '</title>';

                $ret .= '<author>' . $obj->author . '</author>';

                $ret .= '<date>' . $obj->date . '</date>';

                $ret .= '<category>' . $obj->category . '</category>';

                $ret .= '</article>';

                break;

            case 'JSON':

                $array = array('article' => $obj);

                $ret = json\_encode($array);

                break;

        }

        return $ret;

    }

}

**Step 2: Define Your Interface**

interface poly\_writer\_Writer {

    public function write(poly\_base\_Article $obj);

}

## Step 3: Create Your Implementation

**XML Writter code**

class poly\_writer\_XMLWriter implements poly\_writer\_Writer {

    public function write(poly\_base\_Article $obj) {

        $ret = '<article>';

        $ret .= '<title>' . $obj->title . '</title>';

        $ret .= '<author>' . $obj->author . '</author>';

        $ret .= '<date>' . $obj->date . '</date>';

        $ret .= '<category>' . $obj->category . '</category>';

        $ret .= '</article>';

        return $ret;

    }

}

**JSON writer code**

class poly\_writer\_JSONWriter implements poly\_writer\_Writer {

    public function write(poly\_base\_Article $obj) {

        $array = array('article' => $obj);

        return json\_encode($array);

    }

}

**Original class**

class poly\_base\_Article {

public $title;

    public $author;

    public $date;

    public $category;

    public function  \_\_construct($title, $author, $date, $category = 0) {

        $this->title = $title;

        $this->author = $author;

        $this->date = $date;

        $this->category = $category;

    }

    public function write(poly\_writer\_Writer $writer) {

        return $writer->write($this);

    }

}

**Factory**

class poly\_base\_Factory {

    public static function getWriter() {

        // grab request variable

        $format = $\_REQUEST['format'];

        // construct our class name and check its existence

        $class = 'poly\_writer\_' . $format . 'Writer';

        if(class\_exists($class)) {

            // return a new Writer object

            return new $class();

        }

        // otherwise we fail

        throw new Exception('Unsupported format');

    }

}

## Step 5: Put It All Together

$article = new poly\_base\_Article('Polymorphism', 'Steve', time(), 0);

try {

    $writer = poly\_base\_Factory::getWriter();

}

catch (Exception $e) {

    $writer = new poly\_writer\_XMLWriter();

}

echo $article->write($writer);

**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
3. **Overloading** is defining functions/methods that have same signatures with different parameters in the same class.
4. **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.

* **code**

1. *//Example: Method Overloading*
2. Since PHP doesn't support overloading.
3. In PHP, you can only overload methods using the magic method \_\_call()

[**Method overloading by Magic Methods in PHP**](https://www.fullstacktutorials.com/magic-methods-php-27.html)

<?php

class B

{

function \_\_call($name,$arg){

if(count($arg)==1)

{

echo "Call Fun1()";

}

if(count($arg)==2)

{

echo "Call Fun2()";

}

if(count($arg)==3)

{

echo "Call Fun3()";

}

}

}

$b = new B();

$b->x(1);

$b->x(1,2);

$b->x(1,2,3);

* **Method overridding**

1. *//Example: Method Overriding*
2. class A {
3. function testFunc() {
4. return "I am inside class A";
5. }
6. }
7. class B extends A {
8. function testFunc() {
9. return "I am inside class B";
10. }
11. }
12. $objA = new A;
13. $objB = new B;
14. echo($objA->testFunc()); *//"I am inside class A"*
15. echo($objB->testFunc()); *//"I am inside class B"*
16. *//Output:*
17. I am inside class A
18. I am inside class B

**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 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 flights service company.
2. bookFlights() - to book flights from the available 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.

* **code**
* source

1. interface flightServices {
2. public function checkFlightsAvailability();
3. public function bookFlights();
4. }
5. class AirIndia implements flightServices {
6. public function checkFlightsAvailability(){
7. *//Get List of available flights*
8. }
10. public function bookFlights(){
11. *//Book available flights*
12. }
13. }
14. class IndiGO implements flightServices {
15. public function checkFlightsAvailability(){
16. *//Get List of available flights*
17. }
19. public function bookFlights(){
20. *//Book available flights*
21. }
22. }
23. class BritishAirways implements flightServices {
24. public function checkFlightsAvailability(){
25. *//Get List of available flights*
26. }
28. public function bookFlights(){
29. *//Book available flights*
30. }
31. }

**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**

* **code**
* source

1. class BaseClass {
2. public function test() {
3. echo "BaseClass::test() called\n";
4. }
5. final public function moreTesting() {
6. echo "BaseClass::moreTesting() called\n";
7. }
8. }
9. class ChildClass extends BaseClass {
10. public function moreTesting() {
11. echo "ChildClass::moreTesting() called\n";
12. }
13. }
14. *// Results in Fatal error: Cannot override final method BaseClass::moreTesting()*

**Example #2 Final class example**

* **code**
* source

1. final class BaseClass {
2. public function test() {
3. echo "BaseClass::test() called\n";
4. }
5. *// Here it doesn't matter if you specify the function as final or not*
6. final public function moreTesting() {
7. echo "BaseClass::moreTesting() called\n";
8. }
9. }
10. class ChildClass extends BaseClass {
11. }
12. *// 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**

* **code**
* source

1. error\_reporting(E\_ALL); *// Incase if you have hidden error & warning message.*
2. class testClass {
3. public function sayHello() {
4. echo "hello";
5. }
6. }
7. echo testClass::sayHello();
8. *//Output:*
9. Deprecated: Non-static method testClass::sayHello() should not be called statically in C:\xampp\htdocs\test\_projects\index.php on line 7
10. hello

**#Static Example**

* **code**
* source

1. class testClass{
2. public static function testMethod(){
3. echo "This is static method.";
4. }
5. }
6. echo testClass::testMethod();
7. *//Output:*
8. This is static method.

**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**

* **code**
* source

1. trait Hello {
2. public function sayHello() {
3. echo 'Hello ';
4. }
5. }
6. trait World {
7. public function sayWorld() {
8. echo 'World';
9. }
10. }
11. class MyHelloWorld {
12. use Hello, World;
14. public function sayExclamationMark() {
15. echo '!';
16. }
17. }
18. $o = new MyHelloWorld();
19. $o->sayHello();
20. $o->sayWorld();
21. $o->sayExclamationMark();
22. *//Output:*
23. 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.

* **code**
* source

1. trait A {
2. public function smallTalk() {
3. echo 'a';
4. }
5. public function bigTalk() {
6. echo 'A';
7. }
8. }
9. trait B {
10. public function smallTalk() {
11. echo 'b';
12. }
13. public function bigTalk() {
14. echo 'B';
15. }
16. }
17. class Talker {
18. use A, B {
19. B::smallTalk insteadof A;
20. A::bigTalk insteadof B;
21. }
22. }
23. $obj = new Talker();
24. $obj->smallTalk();
25. $obj->bigTalk();
26. *//Output:*
27. 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.

* **code**
* source

1. function myFunction(array $argument\_name)
2. {
3. *//block of code*
4. }

It is also know as Type Declaration.

**//declare(strict\_types=1);**

**function mySuperFunction(): string{**

**return 1234;**

**}**

**echo mySuperFunction();**

**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.

* **code**
* source

1. namespace Google;
2. Class Search {
3. public function query() {
4. return 'Searching Google';
5. }
6. }
7. namespace Bing;
8. Class Search {
9. public function query() {
10. return 'Searching Bing';
11. }
12. }

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**

* **code**
* source

1. class countClassObjects{
2. public static $count = 0;
4. public function \_\_construct(){
5. self::$count++;
6. }
7. public function xyz(){
8. *//code*
9. }
10. }
11. $obj1 = new countClassObjects();
12. $obj2 = new countClassObjects();
13. $obj3 = new countClassObjects();
14. $obj4 = new countClassObjects();
15. $obj5 = new countClassObjects();
16. echo "The number of objects in the countClassObjects class is " . countClassObjects::$count;
17. *//Output*
18. *//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

<https://www.fullstacktutorials.com/interviews/oops-interview-questions-and-answers-for-experienced-in-php-5.html>