Advance PHP

OOPS-MODULE:4

Submitted To: Mr. Nirav Patel

Submitted By: Neha Lad

1. **What Is Object Oriented Programming?**

Ans: OOP is a way of designing and organizing code using objects and classes. Objects represent real-world things or abstract concepts, and classes define their properties and behaviour. OOP helps create modular, reusable, and easier-to-maintain code.

2. **What Are the Properties of Object-Oriented Systems?**

Ans:

1. **Abstraction**: Objects abstract real-world entities or concepts by encapsulating data and methods to manipulate that data. Abstraction hides the internal complexities of an object and only reveals the necessary details.
2. **Encapsulation**: Encapsulation involves bundling data (attributes) and methods (functions) that operate on the data into a single unit, i.e., an object. It restricts access to the inner workings of an object, providing a clear interface for interacting with it while hiding implementation details.
3. **Inheritance**: Inheritance allows objects to inherit attributes and methods from parent classes, promoting code reuse and establishing hierarchical relationships among classes. Subclasses inherit behaviour from their superclasses, enabling the creation of specialized classes that extend or modify the functionality of existing classes.
4. **Polymorphism**: Polymorphism enables objects of different classes to be treated as objects of a common superclass, allowing methods to operate on objects of various types. This concept allows for flexibility and extensibility in object-oriented systems, as different objects can respond to the same message or method call in different ways.
5. **Modularity**: Object-oriented systems promote modularity by dividing complex systems into smaller, manageable units (objects). Each object encapsulates a specific functionality, promoting code organization, maintenance, and reuse.
6. **Message Passing**: In object-oriented systems, communication between objects occurs through message passing. Objects interact by sending messages to each other and requesting actions or information. This communication mechanism enables objects to collaborate and perform tasks in a distributed and decoupled manner.
7. **Identity**: Each object in an object-oriented system has a unique identity, distinguishing it from other objects. This identity allows objects to be

referenced and manipulated independently, even if they share similar attributes and behaviours.

8. What Is the Difference Between Class and Interface?

Ans: In object-oriented programming, classes and interfaces are both fundamental concepts, but they serve different purposes and have distinct characteristics:

1. **Class**:
   - A class is a blueprint or template for creating objects.
   - It defines the attributes (data) and behaviours (methods) that objects of the class will possess.
   - Objects are instances of classes, meaning they are specific realizations of the class blueprint.
   - Classes can include constructors to initialize objects, member variables to store data, and member methods to perform actions.
   - Classes support inheritance, allowing for the creation of hierarchical relationships where subclasses can inherit attributes and methods from superclasses.
2. **Interface**:
   - An interface is a contract specifying a set of methods that a class implementing the interface must implement.
   - It defines a set of abstract methods without providing implementations. Instead, the implementing class provides the concrete implementations.
   - Interfaces can also include constant variables, but these are implicitly static and final.
   - Multiple interfaces can be implemented by a single class, enabling a form of multiple inheritance in languages that support it.
   - Interfaces are used to define a common set of behaviours that classes can adhere to, promoting code reusability, flexibility, and polymorphism.
   - Unlike classes, interfaces cannot contain constructors or instance variables. They only define method signatures.

4. What Is Overloading?

Ans: Overloading is a feature in some programming languages that allows a single function or method to have multiple definitions, each with a different number or type of parameters. In other words, it allows a function or method to behave differently depending on the number or type of arguments passed to it.

Here's a simple example in PHP using default parameter values:

```
function add($a = 0, $b = 0) {
    return $a + $b;
}

echo add(1, 2); // Outputs: 3
echo add(4); // Outputs: 4
```

In this example, the `add` function has default parameter values. You can call it with two arguments, and it will calculate their sum. If you call it with only one argument, it will use the default value for the missing argument.

5. What Is T_PAAMAYIM_NEKUDOTAYIM (Scope Resolution Operator (::) with Example.

Ans: '`T_PAAMAYIM_NEKUDOTAYIM` 'is a PHP error token that represents the Scope Resolution Operator`::`. The Scope Resolution Operator is used to access properties or methods of a class from outside the class or to access static properties or methods of a class.

Here's an example in PHP:

```
class MyClass {
    public static $myStaticProperty = 'Hello, world!';

    public static function myStaticMethod() {
        echo 'This is a static method.';
    }

    public $myProperty = 'This is a property.';

    public function myMethod() {
        echo 'This is a non-static method.';
    }
}
```

echo MyClass::$myStaticProperty; // Outputs: Hello, world!

MyClass::myStaticMethod(); // Outputs: This is a static method.

$myObject = new MyClass();

echo $myObject->myProperty; // Outputs: This is a property.

$myObject->myMethod(); // Outputs: This is a non-static method.

In this example, we define a class `MyClass` with some static and non-static properties and methods. We use the Scope Resolution Operator `::` to access the static property `$myStaticProperty` and the static method `myStaticMethod()` from outside the class. We create an instance of the class `$myObject` and access the non-static property `$myProperty` and the non-static method `myMethod()` using the object operator `->`.

The term `T_PAAMAYIM_NEKUDOTAYIM` is a Hebrew phrase that means "double colon" and is used in PHP to represent the Scope Resolution Operator `::`. It is a reserved word in PHP and cannot be used as a variable or function name.

6. What are the differences between abstract classes and interfaces?

Ans:

➢ CLASS	➢ INTERFACE
➢ The 'class' keyword is used to create a class.	➢ The 'interface' keyword is used to create an interface.
➢ An object of a class can be created.	➢ An object of an interface cannot be created.
➢ Class doesn't support multiple inheritance.	➢ Interface supports multiple inheritance.
➢ A class can inherit another class.	➢ An Interface cannot inherit a class.

➢ A class can be inherited by another class using the keyword ‘extends’.	➢ An Interface can be inherited by a class using the keyword ‘implements’ and it can be inherited by another interface using the keyword ‘extends’.
➢ A class can contain constructors.	➢ An Interface cannot contain constructors.
➢ It cannot contain abstract methods.	➢ It consists of abstract methods only.
➢ Variables and methods can be declared using any specifiers like public, protected, default, private.	➢ Variables and methods are declared as public only.

7. Define Constructor and Destructor.

Ans: Constructors are special member functions for initial settings of newly created object instances from a class, which is the key part of the object-oriented concept in **PHP5**.

Constructors are the very basic building blocks that define the future object and its nature. You can say that the Constructors are the blueprints for object creation providing values for member functions and member variables.

Once the object is initialized, the constructor is automatically called. Destructors are for destroying objects and are automatically called at the end of execution.

Syntax:

- *__construct():*

```
function __construct()
    {
    // initialize the object and its properties by assigning
```

```
    //values
    }
```

- *__destruct():*

```
function __destruct()
    {
    // destroying the object or cleaning up resources here
    }
```

8. How to Load Classes in PHP?

Ans:

In PHP, you can load classes using the **include** or **require** functions to include the PHP file containing the class definition. Here's a basic example:

Create the PHP file containing the class definition: Let's say you have a class named **MyClass** defined in a file named **MyClass.php**:

```
php


Copy code


<?php class MyClass { public function __construct() { echo "MyClass instance

created!"; } public function someMethod() { echo "Some method called!"; } } ?>
```

Include the file in your PHP script: Now, in another PHP file where you want to use the **MyClass**, you include the **MyClass.php** file:

```
php

Copy code

<?php

// Include the file require 'MyClass.php';
```

```
// Create an instance of MyClass $myObject = new MyClass();

// Call a method of MyClass $myObject->someMethod(); ?>
```

The **require** function will include the file **MyClass.php** in the current script. If the file is not found or fails to load, **require** will cause a fatal error and halt script execution. If you want to include the file but allow the script to continue if it's not found, you can use **include** instead of **require**.

Autoloading: As your application grows and you have more classes, manually including each class file becomes cumbersome. PHP provides autoloading functionality to automatically include the class files when needed. You can define your autoloader function or use a pre-built solution like Composer's autoloader.

Here's a basic example of a custom autoloader function:

```
php

Copy code

<?php spl_autoload_register(function ($className) {

 // Assuming class files are named ClassName.php $file = __DIR__ .

'/path/to/classes/' . $className . '.php'; if (file_exists($file)) { require $file; } });

 // Now, when you create an instance of a class, PHP will attempt to autoload it if

it hasn't been included yet. $obj = new MyClass(); ?>
```

9. How to Call the Parent Constructor?

Ans:

In PHP, to call a parent class constructor from within a subclass constructor, you can use the **parent::__construct()** method. Here's how you can do it:

Let's say you have a parent class **ParentClass** with its constructor, and you want to call this constructor from a subclass **SubClass**:

```
<?php

class ParentClass {

    public function __construct() {
```

```
        echo "ParentClass constructor called";
    }
}

class SubClass extends ParentClass {
    public function __construct() {
        // Call parent class constructor
        parent::__construct();

        // Other subclass constructor code goes here
        echo "SubClass constructor called";
    }
}

// Instantiate the subclass
$obj = new SubClass();
?>
```

In this example:

- The **ParentClass** has a constructor that echoes "ParentClass constructor called".
- The **SubClass** extends **ParentClass** and has its constructor.
- In the constructor of **SubClass**, **parent::__construct()** is called to invoke the constructor of the parent class.
- After that, any additional initialization specific to the subclass can be performed.

When you create an instance of **SubClass**, both the parent class constructor and the subclass constructor will be executed in order, resulting in the output:

10. Are Parent Constructors Called Implicitly When Create An Object Of Class?

Ans: No, parent constructors are not called automatically when creating an object of a class in PHP. You need to call the parent constructor explicitly from the child class constructor using the **parent::__construct()** method. Here's an example:

Parent class:

```
class ParentClass {
    public function __construct() {
        echo "Parent constructor called.<br>";
```

```
    }
}
```

Child class:

```
class ChildClass extends ParentClass {
    public function __construct() {
        parent::__construct();
        echo "Child constructor called.<br>";
    }
}

$child = new ChildClass();
```

Output:

1. Parent constructor called.
2. Child constructor called.

when we create a new object of the **ChildClass**, the **ChildClass** constructor is called implicitly. However, the **ParentClass** constructor is not called automatically. We need to call it explicitly using **parent::__construct()** in the **ChildClass** constructor.

11. What Happens, If Constructor Is Defined as Private Or Protected?

Ans: If a constructor is defined as private or protected in PHP, it means that the constructor cannot be called from outside the class. This is useful when you want to prevent the creation of objects from outside the class, or when you want to enforce a specific way of creating objects.

Here's an example of a class with a private constructor:

```
class MyClass {
    private function __construct() {
        // constructor code here
    }

    public static function createInstance() {
        return new MyClass();
    }
}
```

```
// This will cause an error because the constructor is private
$obj = new MyClass();

// This will work because we're calling the createInstance method
$obj = MyClass::createInstance();
```

Here's an example of a class with a protected constructor:

```
class ParentClass {
    protected function __construct() {
        // constructor code here
    }
}

class ChildClass extends ParentClass {
    public function __construct() {
        parent::__construct();
        // constructor code here
    }
}

// This will cause an error because the constructor is protected
$obj = new ParentClass();

// This will work because we're creating an object of the child class
$obj = new ChildClass();
```

12. **What are PHP Magic Methods/Functions? List them Write a program for Static Keyword in PHP.**

    Ans: PHP magic methods are predefined methods in PHP classes that are triggered in response to certain events. These methods have special names that start with double underscores (__) and provide functionality for various operations like object instantiation, property access, method invocation, etc.

Here's a list of PHP magic methods:

1. `__construct()`: Automatically called when an object is created.
2. `__destruct()`: Automatically called when an object is destroyed.
3. `__call($name, $arguments)`: Invoked when inaccessible methods are called on an object.
4. `__callStatic($name, $arguments)`: Invoked when inaccessible static methods are called on a class.
5. `__get($name)`: Invoked when inaccessible properties are accessed.
6. `__set($name, $value)`: Invoked when inaccessible properties are set.
7. `__isset($name)`: Invoked when `isset()` or `empty()` is called on an inaccessible property.
8. `__unset($name)`: Invoked when `unset()` is called on an inaccessible property.
9. `__toString()`: Invoked when an object is treated as a string.
10. `__invoke($arguments)`: Invoked when an object is treated as a function.
11. `__set_state($properties)`: Invoked for classes exported by `var_export()`.
12. `__clone()`: Invoked when an object is cloned.
13. `__debugInfo()`: Invoked when `var_dump()` is called on an object.

14. Create multiple Traits and use them in to a single class.

Ans:

```
<?php
trait SayHello {
    public function sayHello() {
        echo "Hello, I'm saying hello!\n";
    }
}

trait SayGoodbye {
    public function sayGoodbye() {
        echo "Goodbye, I'm saying goodbye!\n";
    }
}
?>
```

Now, let's create a class that uses both traits:

```
<?php
class MyClass {
    use SayHello, SayGoodbye;
}

$obj = new MyClass();
$obj->sayHello();
```

```
$obj->sayGoodbye();
?>
```

Output:

1. **Hello, I'm saying hello!**
2. **Goodbye, I'm saying goodbye!**

15. **Write PHP Script of Object Iteration.**

Ans:

```
<?php class MyClass
{
   public $var1 = 'value 1';    public $var2 = 'value
2';    public $var3 = 'value 3';

   protected $protected = 'protected var';
   private   $private   = 'private var';

   function iterateVisible() {       echo
"MyClass::iterateVisible:\n";       foreach ($this as $key =>
$value) {
         print "$key => $value\n";
     }
   }
}

$class = new MyClass();

foreach($class as $key => $value) {
   print "$key => $value\n";
}
echo "\n";


$class->iterateVisible();

?>
```

Output:-

var1 => value 1
var2 => value 2
var3 => value 3

MyClass::iterateVisible:

var1 => value 1
var2 => value 2
var3 => value 3
protected => protected var
private => private var

16. Use of The $this keyword.
Ans:

$this is a reserved keyword in PHP that refers to the calling object. It is usually the object to which the method belongs, but possibly another object if the method is called statically from the context of a secondary object.
This keyword is only applicable to internal methods.

```
<?php
class simple{

        public $k = 9;

            public function display()
    {
                return $this->k;
        }
}

$obj = new simple();
echo $obj->display();

?>
```

Output:-

9