OPPS

1. What Is Object Oriented Programming?

* Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or [objects](https://www.techtarget.com/searchapparchitecture/definition/object), rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

1. What Are Properties Of Object Oriented Systems?

* The properties of Object-Oriented Systems (often referred to as Object-Oriented Programming, OOP) are key concepts that define how these systems function. These properties are:
* Encapsulation: - This refers to bundling data (attributes) and methods (functions) that operate on the data into a single unit or class. Encapsulation helps protect the data by allowing controlled access, typically through public methods, while keeping other details private.

* Abstractio: - Abstraction allows for simplifying complex systems by representing only the necessary features without showing the inner workings. It helps reduce complexity by focusing on what an object does, not how it does it.
* Inheritance: - Inheritance allows a new class to inherit properties and behavior (methods) from an existing class. This promotes code reusability and establishes a relationship between classes, typically expressed as a parent-child hierarchy.
* Polymorphism: - Polymorphism allows objects of different classes to be treated as objects of a common superclass. It provides flexibility, as a single method can behave differently based on the object that invokes it. It can be achieved through method overriding or overloading.
* Modularity: - In OOP, systems are divided into separate modules, where each module represents a distinct functionality of the system. This makes the system more manageable, flexible, and scalable.
* Dynamic Binding: - This refers to the ability of OOP to resolve method calls at runtime rather than at compile time. This enhances flexibility and supports polymorphism, allowing for more dynamic program behavior.

1. What Is Difference Between Class And Interface?

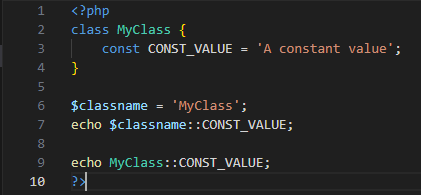
|  |  |
| --- | --- |
| Class | Interface |
| The keyword used to create a class is “class” | The keyword used to create an interface is “interface” |
| A class can be instantiated i.e., objects of a class can be created. | An Interface cannot be instantiated i.e. objects cannot be created. |
| Classes do not support multiple inheritance. | The interface supports multiple inheritance. |
| It can be inherited from another class. | It cannot inherit a class. |
| It can be inherited by another class using the keyword ‘extends’. | It can be inherited by a class by using the keyword ‘implements’ and it can be inherited by an interface using the keyword ‘extends’. |
| It can contain constructors. | It cannot contain constructors. |
| It cannot contain abstract methods. | It contains abstract methods only. |
| Variables and methods in a class can be declared using any access specifier(public, private, default, protected). | All variables and methods in an interface are declared as public. |
| Variables in a class can be static, final, or neither. | All variables are static and final. |

1. What Is Overloading?

* **Overloading** is a feature in object-oriented programming that allows a class to have multiple methods with the same name but different parameters. This is known as **method overloading**. It enables the same method to behave differently depending on the arguments passed to it.

1. What Is T\_PAAMAYIM\_NEKUDOTAYIM (Scope Resolution Operator (::) with Example

* In PHP, **T\_PAAMAYIM\_NEKUDOTAYIM** is the internal name for the **Scope Resolution Operator (::)**, often referred to as the double colon.
* The scope resolution operator is used to access static methods, properties, and constants from a class without instantiating it. It is also used to reference parent class methods or properties from a child class, or access a class's constant.
* Example



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

|  |  |
| --- | --- |
| Abstract classes | Interfaces |
| Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java 8, it can have default and static methods also. |
| Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |
| An abstract class can be extended using keyword "extends". | An interface can be implemented using keyword "implements". |

1. Define Constructor and Destructor?

* **Constructors** are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.
* Constructors are of three types:

1. Default Constructor

2. Parametrized Constructor

3. Copy Constructor

* **Destructor** is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope

1. How to Load Classes in PHP?

* PHP load classes are used for declaring its object etc. in object oriented applications. PHP parser loads it automatically, if it is registered with spl\_autoload\_register() function. PHP parser gets the least chance to load class/interface before emitting an error.

**Ex.**

spl\_autoload\_register(function ($class\_name) {

nclude $class\_name . '.php';

});

1. How to Call Parent Constructor?

* when a class inherits from a parent class, you can call the parent class’s constructor using the parent::\_\_construct() method within the child class's constructor. This allows you to initialize properties or perform actions defined in the parent class when the child class is instantiated.

1. Are Parent Constructor Called Implicitly When Create An ObjectOf Class?

* No, the **parent constructor** is **not called implicitly** when creating an object of a child class in PHP. If the child class has its own constructor, the parent constructor must be called **explicitly** using parent::\_\_construct() within the child class's constructor.

1. What Happen, If Constructor Is Defined As Private Or Protected?

* **Private Constructor:** The class can only be instantiated from within itself, often used for Singleton or Factory patterns. Child classes cannot access it.
* **Protected Constructor:** The class can be instantiated within itself and by its child classes. It prevents external instantiation but allows subclassing.

12.What are PHP Magic Methods/Functions? List them

* magic methods are special methods that are automatically called in certain situations. All magic methods start with a double underscore (\_\_). These methods allow developers to hook into certain behavior of PHP objects, such as object initialization, destruction, method calls, property access, and more.
* **\_\_construct()** - Called when an object is created.
* **\_\_destruct()** - Called when an object is destroyed.
* **\_\_call($method, $arguments)** - Invoked when calling inaccessible methods in an object context.
* **\_\_callStatic($method, $arguments)** - Invoked when calling inaccessible methods in a static context.
* **\_\_get($property)** - Invoked when reading inaccessible or non-existing properties.
* **\_\_set($property, $value)** - Invoked when writing to inaccessible or non-existing properties.
* **\_\_isset($property)** - Invoked when checking if inaccessible or non-existing properties are set.
* **\_\_unset($property)** - Invoked when calling unset() on inaccessible or non-existing properties.
* **\_\_sleep()** - Invoked when an object is serialized.
* **\_\_wakeup()** - Invoked when an object is unserialized.
* **\_\_toString()** - Called when an object is converted to a string.
* **\_\_invoke()** - Called when an object is used as a function.
* **\_\_clone()** - Invoked when an object is cloned.
* **\_\_debugInfo()** - Controls what is shown when var\_dump() is called on an object.

16. Use of The $this keyword

* The $this keyword indicates that we use the class's own methods and properties, and allows us to have access to them within the class's scope.