1. What are the main principles of Object-Oriented Programming?

The main principles of the OOP are the

Encapsulation

Abstraction

Inheritance

Polymorphism

2. What are inheritance and polymorphism in OOP?

Inheritance :

Inheritance is a well-established programming principle, and PHP makes use of this principle in its object model. This principle will affect the way many classes and objects relate to one another.

For example, when extending a class, the subclass inherits all of the public and protected methods, properties and constants from the parent class. Unless a class overrides those methods, they will retain their original functionality.

This is useful for defining and abstracting functionality, and permits the implementation of additional functionality in similar objects without the need to reimplement all of the shared functionality.

Polymorphism :

In PHP, polymorphism is achieved through method overriding and method overloading. Method overriding allows a child class to provide its own implementation of a method that is already defined in its parent class. This means that objects of different classes can respond differently to the same method call, based on their specific implementations. Method

overloading, on the other hand, allows multiple methods with the same name but different parameter lists to coexist in a class. This enables you to perfor...

By utilizing inheritance and interfaces, we achieve polymorphism in PHP, where

different objects can share a common interface but have their own specific behaviors.

One of the key benefits of polymorphism is the ability to write generic code that can accommodate future additions of new classes without requiring modifications to the existing code. This enhances the extensibility and maintainability of your PHP applications.

3. What is encapsulation in OOP, and why is it important?

Encapsulation in OOP is that we keep some properties and methods private inside the class and so they are not accessible from outside the class

There are some reasons why we need Encapsulation and data privacy

First is to prevent code accidentally manipulate our data inside the class

Second is that when we expose a small interface for example a small API consisting only of a few public methods then we can change all the other internal methods with more confidence because we know that external code doesn't rely on these private methods and therefor our code will not break when we work on private methods .

encapsulation promotes code maintenance and reusability.

4. What is abstraction and how is it different from encapsulation ?

Abstraction:

Abstraction is the concept of hiding the implementation details of a feature while exposing only the essential information. It focuses on what an object does rather than how it achieves it.

Simplifies code by reducing complexity.

Allows focus on the functionality, not the implementation.

Encapsulation:

Encapsulation is the practice of restricting access to certain parts of an object by making

properties and methods private or protected. It exposes a controlled interface using public methods.

5. What is polymorphism and how does php support it ?

In programming the polymorphism is the provision of a single interface to entities of different types and it is essential in oop programming

Php support this with to concepts interfaces and abstraction

6. What are namespaces in php and why are they important in oop ?

Namespaces are a way of encapsulating items it helps to make the code readable because it groups the related classes, interfaces , constants together

7. What are magic methods in php ? Provide examples of commonly used magic methods like __construct(), __destruct(), __get(), __set(), and __call() ?

Overloading is different in php with other oop languages

The overloading methods are invoked when interacting with properties or methods that have not been declared or are not visible in the current scope.

__set() is run when writing data to in

accessible (protected or private) or non-existing properties.

__get() is utilized for reading data from inaccessible (protected or private) or non-existing properties.

__construct : A constructor allows you to initialize an object's properties upon creation of the object.

PHP will automatically call this function when you create an object from a class

__destruct :A destructor is called when the object is destructed or the script is stopped or

exited.

If you create a __destruct() function, PHP will automatically call this function at the end of the script.

The __call() method is invoked automatically when a nonexisting method or an inaccessible method is called.

Use the __call() method to wrap existing API into a class.

8. When should you use an abstract class instead of an interface?

Actually there are some differences between abstract classes and interfaces and deciding whether to use an abstract class or an interface depends on your application's design requirements and the behavior you want to enforce

Use an Abstract Class When:

1. You want to provide shared behavior or state:

If you want to define methods with actual functionality that multiple child classes can inherit and optionally override, an abstract class is a better choice.

2. You need to define properties:

Abstract classes allow the declaration of properties (with or without values), while interfaces do not support properties.

3. You want to use non-public methods or properties:

Abstract classes can have protected or private properties and methods, enabling encapsulation. Interfaces only allow public methods.

4. You need a base class for shared logic:

When multiple child classes share common functionality or logic that should be implemented once, use an abstract class.

5. You need to enforce a class hierarchy:

Abstract classes inherently create a strong "is-a" relationship and enforce a specific hierarchy, making them suitable for designs where such a relationship is essential.

Use an Interface When:

1. You want to define a contract for unrelated classes:

Interfaces are ideal when unrelated classes need to implement the same methods without sharing logic or state.

2. You want multiple inheritance-like behavior:

PHP supports multiple interfaces but only single inheritance. Use interfaces to allow a class to adopt behaviors from multiple sources.

3. You don't need shared implementation:

Interfaces cannot have implemented methods or properties, so they are ideal for defining only the structure without imposing specific logic.

When to Choose an Abstract Class Over an Interface:

1. Shared State or Behavior:

o Use an abstract class if your classes share properties, methods, or behavior that you can define in one place.

2. Hierarchy or Base Class Design:

o If you want to define a base class that child classes must extend, with both abstract and concrete methods.

3. Encapsulation:

o When you need to define protected or private members.

4. Single Responsibility:

o If the shared functionality is significant and requires managing state, abstract classes help

maintain single responsibility and clarity.

In general, abstract classes are better for hierarchical, structured designs with shared behavior, while interfaces excel at defining a contract for loosely related or unrelated classes.