OOP Interview questions and answers

What is an object?

An object is an **instance** of a class

What is an order of access modifiers from the most visible to the least visible?

public-protected-package private-private

What are key principles in OOP?

Encapsulation
Inheritance
Polymorphism
Abstraction

What is an Encapsulation?

Bundling the data (attributes) and methods (actions) into a single unit, the object. This keeps the internal workings hidden and only exposes what's necessary.

1. **Data Hiding**: The internal details of an object (like its attributes) are hidden from the outside world. You control access to this data by providing getter and setter methods.
2. **Control of Data Access**: You define which data should be accessible from outside the class and which should be hidden using access modifiers like private, public, or protected

What is Inheritance?

Inheritance is a core principle of Object-Oriented Programming (OOP) that allows a class (known as the **child** or **subclass**) to inherit fields (attributes) and methods (behavior) from another class (called the **parent** or **superclass**). This means that the subclass can reuse the code and functionality of the superclass without rewriting it.

Can you inherit a few classes?

No,a **subclass** can inherit from only one **superclass** in Java, but a class can implement few interfaces

What are the drawbacks of inheritance?

1. **Tight Coupling**

Issue: Inheritance creates a strong relationship between the child and parent classes, meaning that changes in the parent class can directly affect the child classes.

Consequence: This can lead to fragile code, where modifying the parent class might unintentionally break functionality in multiple subclasses, increasing the likelihood of bugs.

2. **Inheritance Overuse (Deep Inheritance Hierarchy)**

Issue: A deep inheritance hierarchy (many levels of parent-child relationships) can make the code difficult to understand and maintain.

Consequence: As the hierarchy grows, it becomes harder to trace how functionality is inherited, overridden, or extended, leading to confusion and making the system more error-prone.

3. **Performance Overhead**

Issue: Inheritance can introduce additional layers of abstraction, which sometimes result in performance overhead during method lookups or object instantiations.

Consequence: This overhead can impact the performance of the application, especially in deep inheritance hierarchies.

4. **Difficulty in Testing**

Issue: Unit testing classes that are deeply tied through inheritance can be difficult because of the inherited dependencies from parent classes.

Consequence: This tight coupling complicates testing, as changes in the parent class can affect tests for the child class, making it harder to isolate and test specific functionalities.

5. **Overcomplicating Design**

Issue: Sometimes developers use inheritance when it isn't necessary or when composition would be a better fit.

Consequence: This can lead to overcomplicated designs where inheritance is forced, making the code harder to maintain and extend.

What is the use of the super keyword?

The super keyword is used to call the constructor or methods of the parent class from the subclass.

What is Polymorphism?

Polymorphism is a core concept in Object-Oriented Programming (OOP) that allows objects of different classes to be treated as objects of a common base class.

Types of Polymorphism

Compile-time Polymorphism (Method Overloading)

Run-time Polymorphism (Method Overriding)

Method Overloading allows multiple methods in the same class to have the same name but different parameter lists (i.e., different types or numbers of parameters). The correct method to call is determined at compile time based on the method signature.

Method Overriding occurs when a subclass provides its own implementation of a method that is already defined in its superclass. This allows the subclass to modify or extend the behavior of the inherited method. The correct method is determined at **runtime**, based on the object's actual type.

Can you return a covariant type while method overriding?

You can return a covariant type in your subclass, for example: parent method returns List, method of a subclass can return ArrayList or LinkedList

Can you inherit from a class marked as final?

No, you cannot inherit from final classes

What is the final keyword used for?

Classes: When a class is declared as final, it **cannot be subclassed** (i.e., you can't inherit from it).

Methods: When a method is declared as final, it **cannot be overridden** by subclasses.

Variables: When a variable is declared as final, it becomes a **constant** (i.e., it can only be assigned once and cannot be changed later).

What is an abstraction?

Abstraction is one of the core principles of Object-Oriented Programming (OOP). It refers to the concept of **hiding the complex details** of how something works and exposing only the necessary and relevant information to the user. In other words, abstraction simplifies the interaction with complex systems by providing a simplified view of the essential features and behaviors, while hiding the underlying implementation details.

Key idea of abstraction?

Focus on "What" not "How": With abstraction, the focus is on "what" an object does rather than "how" it does it. The user only needs to know what functionalities are available (through methods or attributes), without needing to understand the internal workings.

Hide Complexity: The goal of abstraction is to reduce complexity by hiding irrelevant details and exposing only the functionality that is necessary for the user to interact with an object or system.

Can Abstract class be instantiated?

An **abstract class** is a class that cannot be instantiated on its own and is meant to be inherited by other classes. It can contain both **abstract methods** (methods without implementation) and **concrete methods** (methods with implementation).

Can an abstract class have a constructor?

Although you cannot instantiate an abstract class, it can have constructors that are called when its subclass is instantiated.

Can an interface contain fields?

Interfaces cannot have instance variables; they can only have constants (static final fields).

Can you mark methods in interfaces as protected?

No, interface methods are public and abstract by default

What are differences between abstract classes and interfaces?

Feature	Abstract Class	Interface
Instantiation	Cannot be instantiated directly.	Cannot be instantiated directly.
Methods	Can have both abstract and concrete (implemented) methods.	Can have abstract methods and default methods
Fields (Variables)	Can have instance variables (non-static fields).	Cannot have instance variables, only constants (static final).
Constructors	Can have constructors.	Cannot have constructors.

Multiple Inheritance	A class can extend only one abstract class (single inheritance).	A class can implement multiple interfaces, allowing multiple inheritance-like behavior.
Access Modifiers	Abstract methods can have different access modifiers (e.g., protected, public).	Methods in an interface are public and abstract by default.
When to Use	Use when classes share common functionality or behavior with some default implementation.	Use when you want to specify common behavior that can be implemented by multiple, unrelated classes.
Inheritance Relationship	Defines an "is-a" relationship between classes (e.g., Dog is an Animal).	Defines a "can-do" relationship (e.g., a Car can be Driveable and Repairable).

What is an anonymous class?

An **anonymous class** in Java is a local class without a name. It is declared and instantiated in a single expression, for implementing an interface or extending a class. Anonymous classes are often used when you need to create a one-off object with a slight customization of behavior, especially for short-lived tasks.

How many times static blocks are executed?

A **static block** (also known as a static initialization block) is used to initialize static variables and is executed **once** when the class is loaded into memory, **before** any object is created.

What is an order of execution of blocks and constructors?

Static Block (only once, when the class is loaded).

Dynamic Block (every time an object is created, before the constructor).

Constructor (after the dynamic block, for initializing the object).

Can you overload constructors?

Constructors can be **overloaded** (having multiple constructors with different parameter lists).