Day 4 - Interview Questions

1. What is Polymorphism?

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

Types of Java polymorphism
In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

2. Explain the difference between compile-time (static) polymorphism and runtime (dynamic) polymorphism.

Compile-time polymorphism, also known as method overloading, occurs when multiple methods in the same class have the same name but different parameters. The appropriate method to be called is determined by the compiler based on the method's signature.

Runtime polymorphism, also known as method overriding, occurs when a subclass provides a specific implementation of a method declared in its superclass. The method to be executed is determined at runtime based on the object's actual type.

3. How is polymorphism achieved in Java?

Polymorphism is achieved through method overriding and method overloading. Method overriding allows subclasses to provide their own implementation of methods from their superclass, while method overloading allows multiple methods with the same name but different parameter lists to coexist in the same class or subclass.

4. What is method overloading?

Method overloading is a feature in Java that allows a class to have multiple methods with the same name but different parameter lists. These methods are differentiated by the number or types of parameters they accept. Overloading enhances code readability and provides convenience by allowing multiple variations of a method to be used.

5. Can method overloading be based solely on the return type of the method?

No, method overloading cannot be based solely on the return type of the method. Method overloading is determined by the number and types of parameters in the method's parameter list. Two methods with the same name and parameter types but different return types would result in a compilation error.

6. Explain the role of autoboxing and varargs in method overloading.

Autoboxing allows automatic conversion between primitive data types and their corresponding wrapper classes. When it comes to method overloading, autoboxing allows you to define methods that accept parameters of primitive types and their wrapper classes interchangeably.

Varargs (variable-length argument lists) allow a method to accept a variable number of arguments of the same type. When used in method overloading, varargs can provide a more flexible way to define methods that accept a variable number of parameters.

7. What is method overriding?

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to implement the interface methods.

Rules for Method overriding

- The method must have the same name as in the parent class.
- The method must have the same signature as in the parent class.
- Two classes must have an IS-A relationship between them.

8. Difference between method Overloading and Overriding.

Method Overloading	Method Overriding
1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its superclass.
2) Method overloading occurs within the class.	Method overriding occurs in two classes that have IS-A relationship between them.
3) In this case, the parameters must be different.	In this case, the parameters must be the same.

9. What is the difference between compile-time polymorphism and runtime polymorphism?

There are the following differences between compile-time polymorphism and runtime polymorphism.

Sn	Compile-time polymorphism	Runtime polymorphism
1	In compile-time polymorphism, call to a method is resolved at compile-time.	In runtime polymorphism, call to an overridden method is resolved at runtime.
2	It is also known as static binding, early binding, or overloading.	It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch.
3	Overloading is a way to achieve compile-time polymorphism in which we can define multiple methods or constructors with different signatures.	Overriding is a way to achieve runtime polymorphism in which we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class.
4	It provides fast execution because the type of an object is determined at compile-time.	It provides slower execution as compared to compile-time because the type of an object is determined at run-time.
5	Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time.	Run-time polymorphism provides more flexibility because all the things are resolved at runtime.

10. What is the abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It displays just the essential things to the user and hides the internal information, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction enables you to focus on what the object does instead of how it does it. Abstraction lets you focus on what the object does instead of how it does it.

In Java, there are two ways to achieve the abstraction.

- Abstract Class
- Interface

11. What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

12. What is the interface?

An interface is a blueprint for a class that has static constants and abstract methods. It can be used to achieve full abstraction and multiple inheritance. It is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, no method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. However, we need to implement it to define its methods. Since Java 8, we can have the default, static, and private methods in an interface.

13. What are the differences between abstract class and interface?

Abstract class	Interface
An abstract class can have a method body (non-abstract methods).	The interface has only abstract methods.
An abstract class can have instance variables.	An interface cannot have instance variables.
An abstract class can have the constructor.	The interface cannot have the constructor.
An abstract class can have static methods.	The interface cannot have static methods.
You can extend one abstract class.	You can implement multiple interfaces.
The abstract class can provide the implementation of the interface.	The Interface can't provide the implementation of the abstract class.

The abstract keyword is used to declare an abstract class.	The interface keyword is used to declare an 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 class can be implemented using keyword implements
A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

14. Can you use abstract and final both with a method?

No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

15. Can the Interface be final?

No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

16. What is a marker interface?

A Marker interface can be defined as the interface which has no data member and member functions. For example, Serializable, Cloneable are marker interfaces. The marker interface can be declared as follows.

17. How is an interface different from a class?

An interface is different from a class in several ways:

1. An interface cannot be instantiated directly; it cannot have a constructor.
2. An interface can have only abstract methods (methods without a body) and default methods.

3. A class can implement multiple interfaces, but it can extend only one class (single inheritance).
4. An interface is used to achieve multiple inheritance in Java.

18. What is the purpose of using interfaces in Java?

Interfaces serve as a blueprint for classes that want to provide specific behavior while ensuring a consistent method signature. They enable achieving abstraction, allowing classes to provide different implementations for the same methods. Interfaces also play a key role in achieving loose coupling and creating reusable and maintainable code.

19. Can an interface extend another interface?

Yes, an interface can extend another interface using the extends keyword. This allows the sub-interface to inherit the abstract methods and constants of the parent interface while adding more abstract methods or default methods if needed.

20. What is a default method in an interface?

A default method is a method defined in an interface with a default implementation. It allows adding new methods to interfaces without breaking existing implementations. Classes that implement the interface can choose to override the default method or use the provided implementation.

21. How does Java 8's introduction of default methods in interfaces affect existing code?

Java 8's introduction of default methods allows interfaces to evolve without breaking backward compatibility. Existing implementations of the interface will continue to work without any modifications. However, classes implementing the interface can choose to override the default method if they need a different implementation.

22. Can an interface have static methods?

Yes, starting from Java 8, interfaces can have static methods. These methods are defined using the static keyword and can be invoked using the interface name itself, without creating an instance of the implementing class.

23. How do interfaces help in achieving multiple inheritance in Java?

Java classes can implement multiple interfaces, which allows them to implement abstract methods and constants from each of those interfaces. This is a way to achieve

multiple inheritance of type and behavior in Java, without the ambiguity issues often associated with multiple inheritance of classes.

24. Can an interface extend a class?

No, in Java, an interface cannot extend a class. Interfaces and classes are distinct concepts, and while a class can implement an interface, it cannot extend it.