**INTERVIEW QUESTIONS (OOPJ)**

**17/9/24**

**In programming, a constructor is a special type of method or function that is used to initialize objects of a class.**

**1. What is inheritance in Java, and why is it useful ? What are the access specifiers that allow inheritance in Java?**

==🡺Inheritance in Java: It's a way for one class to inherit properties and methods from another, making code reusable and easier to maintain. The access specifiers are public, protected, and default.

**2. Can a class be inherited by multiple classes in Java? If not, how can you achieve multiple inheritance?**

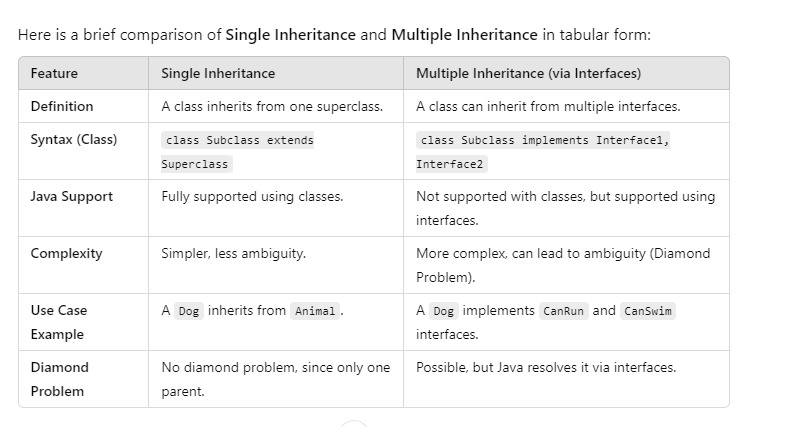
Multiple inheritance: Java doesn't allow multiple class inheritance to avoid ambiguity. You can achieve it using interfaces.

**3. What is the difference between single, multilevel, and hierarchical inheritance?**

Single: One class inherits from another.

Multilevel: A class inherits from another, which in turn is inherited by another.

Hierarchical: One class is inherited by multiple classes.



**4. Can a constructor be inherited in Java? Why or why not?**

Constructor inheritance: Constructors aren’t inherited because each class needs to handle its initialization, but a subclass can call a superclass constructor using super().

**5. What is the difference between String, StringBuilder, and StringBuffer in Java?**

String: Immutable, thread-safe.

StringBuilder: Mutable, not thread-safe, faster.

StringBuffer: Mutable, thread-safe, slower.

**6. Why are strings immutable in Java?**

Strings are immutable: This ensures security, caching, and performance optimization.

**7. How does the intern() method work with strings?**

intern() method: It stores strings in a common pool, reusing instances to save memory.

**8. when What is the difference between == and .equals() comparing strings in Java?**

== vs .equals(): == compares references, while .equals() compares the actual content of strings.

* **==**: Compares **memory references** (whether two objects are the same).
* **.equals()**: Compares the **content** (whether the values inside the objects are the same).

For strings, use **.equals()** to check for content equality.

**9. Can you explain the difference between a literal string and a new string object?**

Literal vs new string: Literals are stored in the string pool, reused if they already exist. new creates a separate object in memory.

**10. What is widening typecasting, and when is it done automatically in Java?**

Widening typecasting: Automatically converting a smaller type to a larger type (e.g., int to double).

**11. What is narrowing typecasting, and why does it require explicit casting?**

Narrowing typecasting: Manually converting a larger type to a smaller type (e.g., double to int), which requires explicit casting due to possible data loss.

**12. What happens when you try to cast incompatible data types in Java?**

Incompatible casting: Throws a ClassCastException at runtime if the types are incompatible.

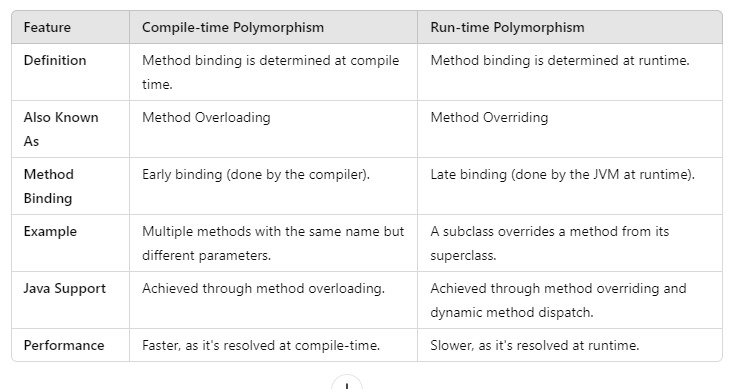
**13. Is it possible to convert a double to a byte using narrowing casting? If yes, how does Java handle it?**

Double to byte narrowing: Java allows it, but the value is truncated or wrapped if it exceeds the byte range.

**14. What is the difference between compile-time and runtime polymorphism in Java?**

Compile-time polymorphism: Achieved with method overloading, resolved at compile time.

Runtime polymorphism: Achieved with method overriding, resolved at runtime.



**15. Can you explain the significance of the instance of operator with respect to polymorphism?**

Instance of in polymorphism: It checks whether an object is an instance of a specific class, useful for determining object types during runtime when using polymorphism.

**20. Can you explain the significance of the instanceof operator with respect to polymorphism?**

The `instanceof` is a operator

It checks if an object belongs to a certain class or subclass.

In polymorphism, it helps you figure out the actual type of the object when you’re working with a parent reference.

This is handy when you need to be sure about what type of object you're dealing with before applying any operations.

**21. What is abstraction in Java, and how is it implemented?**

Process of essential things.

Outer behaviour

Major pillar of oops

Purpose-helps to achieve simplicity.

Create a instance and call the method.

**- Abstract Classes:** You define a blueprint with some methods that may not have complete code

**- Interfaces**: They are like contracts where you only define method names, and the class that implements the interface has to write the actual code.

**22. What is the difference between an abstract class and an interface in Java?**

- Abstract Class:

- Can have regular methods (with code) and abstract methods (without code).

- A class can only extend one abstract class.

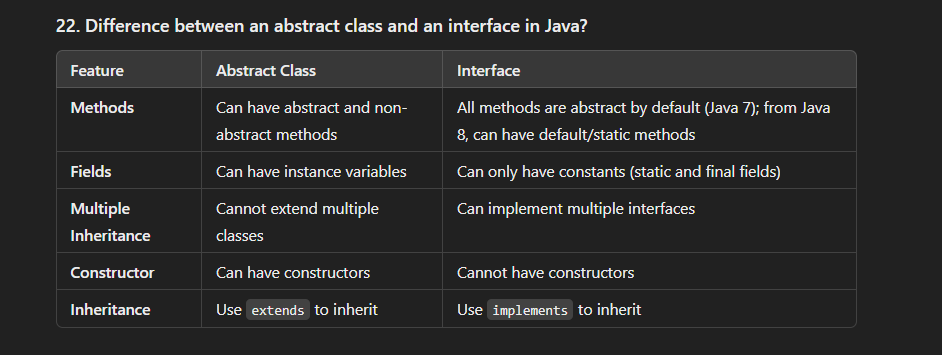
- Can have instance variables and constructors.

- Interface:

- Only has method declarations (until Java 8+ added default methods).

- A class can implement multiple interfaces.

- No constructors or instance variables.



* **(ASK) 23. Can you declare an abstract class without any abstract methods? Why would you do that?**

Yes, an abstract class can have no abstract methods. You might do this if you want to prevent the class from being instantiated but still allow other classes to inherit from it.

**24. Why can’t an abstract class be instantiated in Java?**

An abstract class is like an incomplete blueprint; it doesn’t have all the details filled in (i.e., abstract methods). Since it’s not fully defined, Java prevents you from creating an object from it.

(abstract class is fully incomplete or partial complete. This class is abstract class . If contains abstract methods that cant be overriding the abstract class).

**25. How does abstraction improve software design in OOP?**

Abstraction makes code simpler and cleaner by focusing only on the important parts. It also makes your program easier to update and manage since changes to complex internal details won’t affect other parts of your code.

**26. What happens if a class implements an interface but does not provide an implementation for all of its methods?**

If a class implements an interface but doesn’t provide code for all the methods, the class must be declared as abstract. Otherwise, it will result in a compilation error.

**27. What is exception handling in Java, and why is it important?**

**Exception**: An unexpected event that occurs during the execution of a program (e.g., division by zero, file not found).

**Types of Exceptions**:

* **Checked Exceptions**: Handled at compile time (e.g., IOException **closable**-ability to closed resources using close method).
* **Unchecked Exceptions**: Occur at runtime (e.g., NullPointerException, ArithmeticException).

**Key Keywords**:

* **try**: Wraps the code that might throw an exception.
* **catch**: Catches and handles exceptions.
* **finally**: Executes(JVM) code regardless of whether an exception occurs (usually for cleanup tasks).
* **throw**: Manually throws an exception.
* **throws**: Declares that a method throw an exception.

**Importance**:

* **Improves readability**
* **Enhances debugging** with detailed error messages (stack traces) when an exception is thrown.

**28. What is the difference between checked and unchecked exceptions in Java?**

- Checked Exceptions: These are checked at compile-time (e.g., `IOException`). You must handle them using try-catch or declare them in the method signature.

- Unchecked Exceptions: These occur at runtime (e.g., `NullPointerException`), and handling them is optional.

**29. What is the try-catch-finally block in Java, and how does it work?**

- **try**: You write the code that might throw an exception here.

- **catch:** If an exception happens, the catch block deals with it.

- **finally**: This block always runs, whether an exception happened or not. It’s usually used for cleaning up resources.

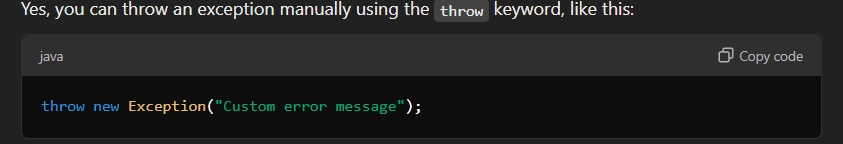
**30. What is the purpose of the finally block? Can a finally block be skipped?**

The `finally` block is used to close resources or clean up after code execution, ensuring it runs no matter what. However, it can be skipped if the program exits unexpectedly (like calling `System.exit()`).

**31. Can you throw an exception manually in Java? If so, how?**

Yes, you can throw an exception manually using the `throw` keyword, like this:

throw new Exception("Custom error message");



**32. What is the difference between throw and throws in Java?**

- throw: Used to actually throw an exception in your code.

- throws: Declares that a method throw certain exceptions.

**33. What happens when an exception is thrown but not caught(** **to reach for and try to hold (something)}  in Java?**

If an exception is thrown but not caught, the program will crash, and Java will display an error message (stack trace) explaining what went wrong.

**34. Can you have multiple catch blocks for a single try block? How does Java decide which one to execute?**

Yes, you can have multiple catch blocks for different types of exceptions. Java will execute the first catch block that matches the type of exception thrown.

**35. What is the purpose of the Throwable class in Java?is class, allowing Java to treat them uniformly in exception handling mechanisms.**

Throwable is the parent class for all errors and exceptions in Java. Both Exception and Error inherit from this class, allowing Java to treat them uniformly in exception handling mechanisms.