**INTERVIEW QUESTIONS (OOPJ)**

**17/9/24**

**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?**

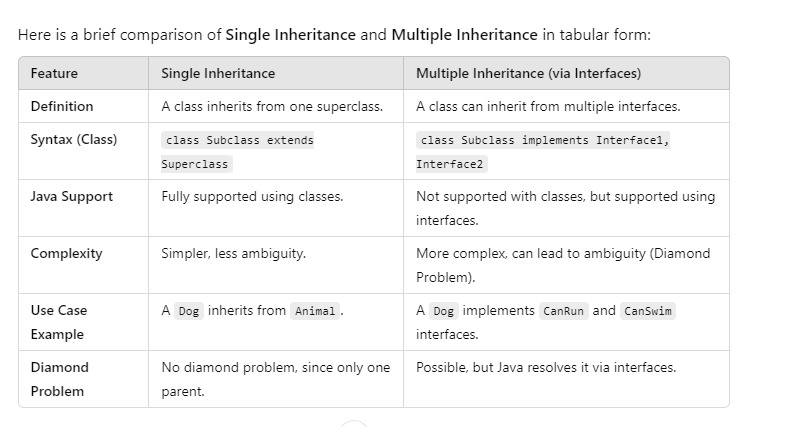
Multiple inheritance: Java doesn't allow multiple class inheritance to avoid ambiguity. You can achieve it using interfaces.

**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.

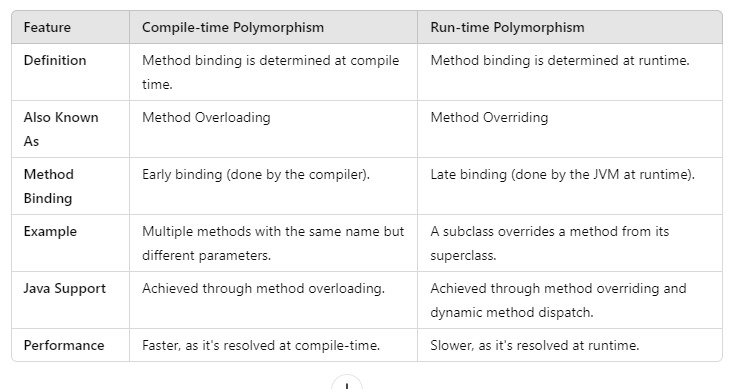
**13. Is it possible to convert a double to a byte using narrowing casting? If yes, how does Java handle it?**

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. Significance of the `instanceof` operator with respect to polymorphism:**

The `instanceof` operator is used to test whether an object is an instance of a specific class or interface. In polymorphism, where a reference variable of a parent class can point to an object of its subclass, the `instanceof` operator becomes significant to determine the true type of the object at runtime. This helps to safely cast the object and avoid `ClassCastException`.

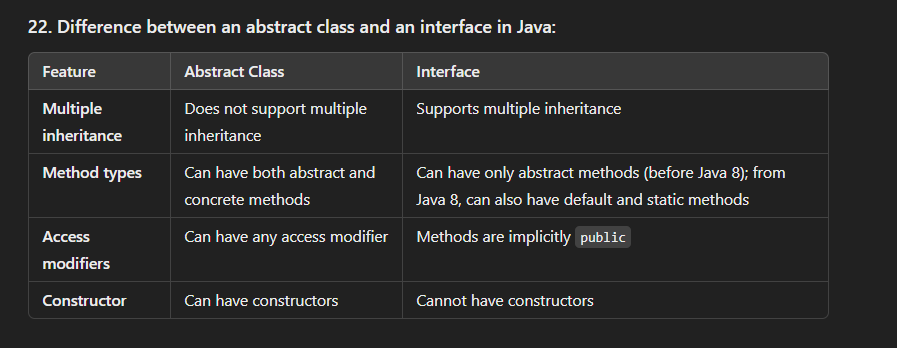
**21. What is abstraction in Java, and how is it implemented?**

Abstraction is the concept of hiding the internal details and showing only the essential features of an object. It is implemented in Java using:

- \*\*Abstract classes:\*\* Classes that cannot be instantiated directly and may contain abstract methods (methods without a body) and concrete methods.

- \*\*Interfaces:\*\* A blueprint of a class that can contain abstract methods (since Java 8, interfaces can also have default and static methods).

**22. Difference between an abstract class and an interface in Java:**



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

Yes, an abstract class can be declared without any abstract methods. This is done when you want to prevent the instantiation of the class but still allow other classes to inherit from it and provide their implementations or extend its functionality.

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

An abstract class is incomplete by design because it may contain abstract methods that do not have implementations. Instantiating it directly would not make sense as it cannot provide concrete behavior. It is meant to be extended by subclasses that provide the implementations for the abstract methods/

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

Abstraction improves software design by:

- Simplifying complex systems by hiding unnecessary details.

- Allowing developers to focus on the high-level design without being bogged down by implementation specifics.

- Promoting a clear separation of concerns, which leads to cleaner, more maintainable, and flexible 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 does not provide implementations for all its methods, it must be declared abstract. If the class is not abstract, the compiler will throw an error because the class is considered incomplete.

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

Exception handling in Java is the mechanism to handle runtime errors and ensure the normal flow of the program. It is important because it:

- Provides a way to gracefully handle errors and recover from them.

- Prevents the program from crashing unexpectedly.

- Helps in debugging and maintaining code by using structured error-handling.

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

**- Checked exceptions:** Exceptions that are checked at compile-time. The code must either handle these exceptions using `try-catch` or declare them using `throws`. Example: `IOException`.

**- Unchecked exceptions**: Exceptions that occur at runtime and are not checked during compilation. These are typically issues like programming bugs (e.g., `NullPointerException`, `ArrayIndexOutOfBoundsException`).

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

The `try-catch-finally` block is used for handling exceptions:

- The code that may throw an exception is placed inside the `try` block.

- If an exception occurs, it is caught by the corresponding `catch` block(s).

- The `finally` block (optional) contains code that will always execute, regardless of whether an exception occurred or not.

---

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

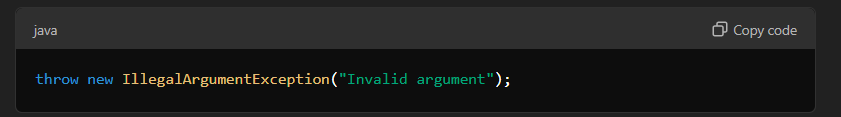
The `finally` block is used to execute important cleanup code (e.g., closing resources) that must run no matter what happens. The `finally` block is not skipped unless:

- The JVM exits due to `System.exit()`.

- A fatal error occurs that causes the JVM to crash.

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

Yes, an exception can be thrown manually using the `throw` keyword. Example:



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

- **`throw`:** Used to explicitly throw an exception.

**- `throws`:** Used in a method signature to declare the exceptions that a method might throw.

**33. What happens when an exception is thrown but not caught in Java?**

If an exception is thrown and not caught, it propagates up the call stack. If no higher-level method catches it, the program terminates, and the JVM prints the exception's stack trace.

---

**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 a single `try` block. Java will execute the first `catch` block that matches the type of exception thrown. If none match, the exception will propagate further.

**35. What is the purpose of the `Throwable` class in Java?**

The `Throwable` class is the superclass for all errors and exceptions in Java. It allows the handling of both:

**- Exceptions**: Recoverable conditions (subclass of `Throwable`).

**- Errors**: Severe conditions that are typically not recoverable (e.g., `OutOfMemoryError`).