Core Java Interview Questions

> Author: Aditya Kansana

> Lab: Special Force

💡

SET - 1

1. **What are the main features of Java?**

   Follow-up:

   - Why is Java platform-independent?
   - Can you explain how Java achieves memory management and what makes it robust in this regard?
2. **What are the different class loaders in Java?**
3. **What is bytecode in Java, and how is it generated?**

   Follow-up:

   - How does the JVM interpret bytecode?
4. **What is the Just-In-Time (JIT) compiler, and how does it optimize performance?**

   Follow-up:

   - How does JIT compilation differ from interpretation?
5. **How does Java handle memory management for loaded classes?**

   Follow-up:

   - What happens to classes that are no longer referenced?
6. **What is the difference between stack memory and heap memory in Java?**

   Follow-up:

- Which data types are stored in stack and heap memory?

7. **How does the stack memory work during method calls?**

   Follow-up:

   - What happens to local variables when a method exits?

8. **What are the implications of memory allocation in the heap for object creation?**

   Follow-up:

   - How does garbage collection affect heap memory?

9. **How does the JVM manage memory for objects in the heap?**

   Follow-up:

   - What is the role of the garbage collector in this process?

10. **What are stack overflow and heap overflow errors?**

    Follow-up:

    - How can you prevent these errors in your applications?

11. **Explain the concept of OOP in Java.**

    Follow-up:

    - How does Java implement polymorphism?
    - Can you differentiate between method overloading and method overriding?

12. **What is the difference between `==` and `equals()` in Java?**

    Follow-up:

    - Can you give a scenario where using `==` can cause a logical error?
    - How does the `equals()` method work when comparing two user-defined objects?

13. **What is the significance of the `final` keyword in Java?**

    Follow-up:

    - What is the difference between declaring a variable, method, and class as `final`?

- Can a final class be inherited? If not, why might we need `final` classes in a design?

14. **Explain the difference between `ArrayList` and `LinkedList`.**

    Follow-up:

    - In which scenario would you prefer one over the other?
    - Can you explain the time complexity for adding an element in both?

15. **What is the Java memory model?**

    Follow-up:

    - Can you explain what happens during garbage collection in Java?
    - What is the difference between stack memory and heap memory in Java?

16. **Explain how the `try-catch-finally` block works in exception handling.**

    Follow-up:

    - What happens if an exception occurs in the `finally` block?
    - Can you write a code snippet where the `finally` block does not get executed?

17. **What are checked and unchecked exceptions in Java?**

    Follow-up:

    - Why is `RuntimeException` unchecked while `IOException` is checked?
    - Can you catch both checked and unchecked exceptions in the same block?

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

    Follow-up:

    - Can an abstract class implement an interface? If yes, how?
    - What are the advantages of using interfaces in Java?

19. **What is the significance of the `volatile` keyword in Java?**

    Follow-up:

    - Can you explain how it relates to the Java Memory Model?

- How does `volatile` differ from `synchronized`?

20. **Explain the `static` keyword in Java.**

    Follow-up:

    - Can you call a non-static method from a static method? Why or why not?
    - Can you give an example of when you would use a static block?

21. **What is the use of the `transient` keyword in Java?**

    Follow-up:

    - Why do you think `transient` is not inherited by subclasses?
    - What happens if a `transient` variable is part of an object being serialized?

22. **What are wrapper classes in Java?**

    Follow-up:

    - How does autoboxing/unboxing work in Java?
    - Can you explain any performance overhead involved with autoboxing?

23. **What is the significance of the `this` keyword in Java?**

    Follow-up:

    - Can you assign a value to `this` inside a method? Why or why not?
    - What happens when you try to pass `this` as an argument to a static method?

24. **Explain multithreading in Java.**

    Follow-up:

    - Can you describe the difference between `Thread` and `Runnable`?
    - How would you implement thread safety in Java?

25. **What is the difference between `String`, `StringBuilder`, and `StringBuffer`?**

    Follow-up:

    - Which one would you use for creating mutable strings in a multithreaded environment?
    - How does immutability in `String` affect its performance?

26. **What is the purpose of the `synchronized` keyword in Java?**

    Follow-up:

    - Can you synchronize a static method? If yes, what happens internally?
    - What are the drawbacks of using `synchronized` excessively?

27. **What is the Java Collections Framework?**

    Follow-up:

    - Can you name a few key interfaces in the Collections Framework?
    - How does the `HashMap` ensure unique keys in a collection?

28. **What is garbage collection in Java?**

    Follow-up:

    - What are the different types of garbage collectors in Java?
    - Can you force garbage collection in Java?

29. **What are lambda expressions in Java?**

    Follow-up:

    - What problem do lambda expressions solve?
    - Can you demonstrate the use of a lambda expression in a simple example?

30. **What is the use of `Optional` in Java 8?**

    Follow-up:

    - Can you show how `Optional` can help avoid `NullPointerException`?
    - What is the difference between `Optional.of()` and `Optional.ofNullable()`?

31. **How does a `HashMap` work internally?**

    Follow-up:

    - Can you explain what happens when there is a collision in a `HashMap`?
    - What is the significance of the load factor in a `HashMap`?

32. **What is a thread pool, and why is it used?**

    Follow-up:

- Can you describe the difference between `newCachedThreadPool()` and `newFixedThreadPool()` in Java?
- How do you handle thread lifecycle in a thread pool?

33. **What are generics in Java?**

    Follow-up:

    - Why do generics exist, and how do they improve code quality?
    - Can you use primitive data types in generics?

34. **Explain the concept of class loading in Java.**

    Follow-up:

    - What are the different types of class loaders in Java?
    - Can you explain what happens when two classes with the same name are loaded?

35. **What are the differences between `Comparator` and `Comparable` in Java?**

    Follow-up:

    - Can you explain a scenario where you would use `Comparator` over `Comparable`?
    - What happens if two objects being compared are equal in terms of sorting?

36. **What is the purpose of `super` in Java?**

    Follow-up:

    - Can you call a constructor of the superclass using `super`? Show an example.
    - How does the `super` keyword behave in the case of method overriding?

37. **Explain the Singleton design pattern in Java.**

    Follow-up:

    - How would you implement a thread-safe Singleton?
    - Can you explain why a Singleton might cause issues in distributed systems?

38. **What are method references in Java 8?**

    Follow-up:

- Can you explain how method references differ from lambda expressions?
- What are the different types of method references in Java?

39. **What is the `default` keyword in Java interfaces?**

    Follow-up:

    - Can you override a default method in a class that implements the interface?
    - Why was the `default` method introduced in Java 8?

40. **What is a `ConcurrentHashMap`, and how is it different from a `HashMap`?**

    Follow-up:

    - Can you explain how thread safety is achieved in a `ConcurrentHashMap`?
    - How does the performance of `ConcurrentHashMap` compare with `HashMap` in a multi-threaded environment?

41. **What is the difference between the `wait()` and `sleep()` methods in Java?**

    Follow-up:

    - Why must `wait()` be called in a synchronized block?
    - Can you interrupt a thread that is sleeping? What happens?

42. **Explain how `notify()` and `notifyAll()` work in Java.**

    Follow-up:

    - What is the difference between `notify()` and `notifyAll()`? When would you use one over the other?
    - What happens if you call `notify()` without holding the monitor?

43. **What is reflection in Java?**

    Follow-up:

    - Can you give an example where reflection is used in frameworks like Spring or Hibernate?
    - What are the drawbacks of using reflection?

44. **Explain the difference between `Serializable` and `Externalizable`.**

    Follow-up:

- When would you choose to implement `Externalizable` instead of `Serializable`?
- Can you explain how to control the serialization process with `Externalizable`?

45. **What is the purpose of the `enum` keyword in Java?**

    Follow-up:

    - Can you extend an `enum` in Java? Why or why not?
    - How would you add methods or fields to an `enum`?

46. **What is a deadlock in Java?**

    Follow-up:

    - Can you explain how deadlocks happen with an example?
    - How would you prevent a deadlock scenario in a multi-threaded environment?

47. **Explain the concept of immutability in Java with an example.**

    Follow-up:

    - Why is `String` immutable in Java, and how does that affect performance?
    - How would you create an immutable class in Java? What are the key characteristics?

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

    Follow-up:

    - Can you throw multiple exceptions from a method? How would you handle them?
    - What happens if a method declares a checked exception but doesn't actually throw it?

49. **What are functional interfaces in Java?**

    Follow-up:

    - Can you name some built-in functional interfaces provided by Java?
    - How do functional interfaces enable the use of lambda expressions?

50. **What is the difference between `String` and `char[]` for storing passwords in Java?**

    Follow-up:

    - Why is it recommended to use `char[]` for password storage rather than `String`?
    - How would you securely clear a `char[]` after use?

SET - 2

1. **What is the difference between shallow cloning and deep cloning in Java?**

   Follow-up:

   - Can you provide an example of shallow and deep cloning with the `Cloneable` interface?
   - How does the `clone()` method affect object references in both cases?
2. **What are the key differences between a `HashSet` and a `TreeSet`?**

   Follow-up:

   - Can you explain the time complexity for basic operations in `HashSet` and `TreeSet`?
   - In what scenario would you use `TreeSet` over `HashSet`?
3. **What is method hiding in Java?**

   Follow-up:

   - How does method hiding differ from method overriding?
   - Can you give an example where method hiding causes unexpected behavior?
4. **What are the differences between `try-with-resources` and a traditional `try-catch-finally` block?**

Follow-up:

- How does `try-with-resources` improve resource management in Java?
- Can you use `try-with-resources` with custom classes? How would you implement the `AutoCloseable` interface?

5. **What is the difference between `Callable` and `Runnable` in Java?**

   Follow-up:

   - When would you use `Callable` instead of `Runnable`?
   - Can you explain how `Future` works with `Callable` to return results asynchronously?

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

   Follow-up:

   - Can you give examples of both checked and unchecked exceptions?
   - Why do we have checked exceptions in Java, but not in languages like Python?

7. **Explain the use of `try`, `catch`, and `finally` blocks in Java.**

   Follow-up:

   - What happens if an exception is thrown in the `finally` block?
   - Can the `finally` block be omitted? What are the consequences?

8. **What is the purpose of the `throws` keyword in method declarations?**

   Follow-up:

   - Can a method throw both checked and unchecked exceptions?
   - How does the `throws` keyword affect the method's signature and its usage by other methods?

9. **How would you create a custom exception in Java?**

   Follow-up:

   - Should custom exceptions extend `Exception` or `RuntimeException`? Why?
   - Can you include extra fields or methods in a custom exception?

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

Follow-up:

- Can you throw multiple exceptions from a single method?
- What happens when you throw an exception without declaring it in the method signature?

11. **What is the purpose of the `try-with-resources` statement?**

    Follow-up:

    - How does it ensure proper resource management?
    - Can you use custom classes in a `try-with-resources` block? How?

12. **What are some best practices for exception handling in Java?**

    Follow-up:

    - Why is it generally a bad idea to catch `Exception` or `Throwable`?
    - How would you log exceptions effectively in a real-world application?

13. **Explain the `finally` block. Can it be skipped in some cases?**

    Follow-up:

    - What happens if you return a value inside a `try` or `catch` block? Does the `finally` block still execute?
    - Can an exception in the `finally` block suppress an exception thrown in the `try` block?

14. **What is the difference between re-throwing an exception and throwing a new exception?**

    Follow-up:

    - How would you re-throw an exception without losing the original stack trace?
    - In what situations would you want to wrap and throw a new exception instead of re-throwing the original one?

15. **What is the difference between `final`, `finally`, and `finalize()` in Java?**

    Follow-up:

    - Can you give an example of how `finally` behaves when an exception occurs in both the `try` and `catch` blocks?

- Why is `finalize()` considered deprecated in modern Java development? What should be used instead?

16. **What is a constructor in Java?**

    Follow-up:

    - Why do we need constructors?
    - Can a constructor be marked as `final`, `static`, or `abstract`? Why or why not?

17. **What is the difference between a default constructor and a parameterized constructor?**

    Follow-up:

    - What happens if you don't define any constructor in a class?
    - Can you overload constructors in Java? How?

18. **Can a constructor call another constructor of the same class?**

    Follow-up:

    - How would you use the `this()` keyword to achieve constructor chaining?
    - What are the rules regarding calling constructors using `this()`?

19. **Can a constructor throw an exception in Java?**

    Follow-up:

    - How would you handle an exception thrown by a constructor?
    - Is it a good practice to throw exceptions from constructors? Why or why not?

20. **What is a static constructor (or static block) in Java?**

    Follow-up:

    - When would you use a static initialization block?
    - Can a static block access non-static variables? Why?

21. **What is the difference between a constructor and a method in Java?**

    Follow-up:

    - Can a constructor return a value?

- What happens if you mistakenly specify a return type for a constructor?

22. **What is the purpose of the `static` keyword in Java?**

    Follow-up:

    - Can static methods access instance variables? Why or why not?
    - How would you call a static method of another class?

23. **Explain the `main` method in Java. Why is it `public`, `static`, and `void`?**

    Follow-up:

    - Can you overload the `main` method in Java?
    - What happens if you try to run a Java program without the `main` method?

24. **What is the difference between a static method and an instance method?**

    Follow-up:

    - Can you override a static method in Java? Why or why not?
    - What is method hiding, and how does it relate to static methods?

25. **Explain memory management in Java. How does Java manage memory allocation and deallocation?**

    Follow-up:

    - What is the difference between the stack and the heap in Java?
    - What role does the garbage collector play in memory management?

26. **What is the garbage collector in Java? How does it work?**

    Follow-up:

    - Can you manually trigger garbage collection in Java? How?
    - What are some of the key garbage collection algorithms used in modern JVMs (e.g., G1, CMS)?

27. **What are some ways to prevent an object from being garbage collected in Java?**

    Follow-up:

    - What is a strong reference, and how does it affect garbage collection?
    - Can `finalize()` prevent an object from being collected? How?

28. **Explain the role of the `finalize()` method in Java.**

    Follow-up:

    - Why is `finalize()` considered deprecated in newer Java versions?
    - What are some alternatives to `finalize()` for cleanup operations?

29. **How is the `Object` class the superclass of all Java classes?**

    Follow-up:

    - What is the significance of the `Object` class in the inheritance hierarchy?
    - Can you create a class in Java that doesn't inherit from `Object`?

30. **What are some of the commonly used methods of the `Object` class?**

    Follow-up:

    - Can you explain how the `equals()` and `hashCode()` methods work together?
    - How would you override the `toString()` method in a custom class?

31. **What are the four main principles of Object-Oriented Programming (OOP) in Java?**

    Follow-up:

    - Can you give a brief explanation of each principle (Encapsulation, Inheritance, Polymorphism, Abstraction)?

32. **Explain encapsulation with an example. How does Java support encapsulation?**

    Follow-up:

    - What is the difference between encapsulation and data hiding?
    - How does access control (`private`, `protected`, `public`) help in achieving encapsulation?

33. **What is inheritance in Java? How does it promote code reusability?**

    Follow-up:

    - Can a class in Java inherit multiple classes? Why or why not?
    - How does the `super` keyword work in the context of inheritance?

34. **Explain polymorphism in Java with an example.**

Follow-up:

- What is the difference between compile-time (static) and runtime (dynamic) polymorphism?
- How does method overriding and method overloading relate to polymorphism?

35. **What is abstraction in Java, and how can it be implemented?**

    Follow-up:

    - What is the difference between an abstract class and an interface in Java?
    - When would you choose an abstract class over an interface and vice versa?

36. **What is method overriding in Java?**

    Follow-up:

    - How does method overriding support runtime polymorphism?
    - What are the rules and restrictions of method overriding in Java?

37. **What is method overloading in Java, and how is it different from method overriding?**

    Follow-up:

    - Can you overload a method by changing only the return type?
    - What happens if you overload the `main` method in Java?

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

    Follow-up:

    - Can an interface have default methods in Java? How is this feature useful?
    - Can a class implement multiple interfaces in Java? What about extending multiple abstract classes?

39. **Explain the concept of constructors and inheritance. Can a subclass call the constructor of its superclass?**

    Follow-up:

- How does the `super()` keyword facilitate calling a superclass constructor?
- What happens if you don't explicitly call the superclass constructor in a subclass?

40. **What is the relationship between objects and classes in Java?**

    Follow-up:

    - Can you explain how objects are created and destroyed in Java?
    - What is the role of constructors in creating objects, and how does garbage collection handle object destruction?

41. **What is the difference between `this` and `super` keywords in Java?**

    Follow-up:

    - Can you provide examples where you would use `this` and `super` in method calls or constructors?
    - Can `this()` and `super()` be used together in the same constructor? Why or why not?

42. **What is the difference between composition and inheritance in Java?**

    Follow-up:

    - Can you give an example where you would prefer composition over inheritance?

43. **What is the significance of the `instanceof` operator in Java?**

    Follow-up:

    - How does `instanceof` support polymorphism?
    - Can you use `instanceof` with interfaces? How?

44. **Explain multiple inheritance in Java. Why is it not supported with classes, but allowed with interfaces?**

    Follow-up:

    - What problem does multiple inheritance cause, and how does Java solve it with interfaces?
    - How do default methods in interfaces complicate multiple inheritance in Java?

45. **What are getter and setter methods? How do they relate to encapsulation?**

    Follow-up:

    - Why would you use getter and setter methods instead of directly accessing class variables?
    - Can you make a setter method `private`? Why or why not?

46. **What is an object in Java, and how is it related to a class?**

    Follow-up:

    - How is object creation handled in Java, and what is the role of the `new` keyword?
    - What is object identity in Java, and how is it different from object equality?

47. **Explain the concept of "upcasting" and "downcasting" in Java.**

    Follow-up:

    - What is the role of polymorphism in upcasting and downcasting?
    - Can downcasting throw an exception? If so, which one?

48. **What is the `toString()` method in Java? Why would you override it?**

    Follow-up:

    - What is the default behavior of `toString()` in the `Object` class?
    - How does overriding `toString()` improve debugging?

49. **What is an inner class in Java, and what are the different types of inner classes?**

    Follow-up:

    - How does an inner class have access to the members of its outer class?
    - Can a static nested class access instance members of its outer class? Why or why not?

50. **What is the difference between single inheritance and multilevel inheritance in Java?**

Follow-up:

- Can you provide examples of both?

- How does multilevel inheritance affect the constructor calling chain?

SET - 3

1. **How does Java handle access control in inheritance (e.g., `private`, `protected`, `public`)?**

   Follow-up:

   - What is the significance of the `protected` keyword in the context of inheritance?
   - How does package-level access affect inheritance?

2. **Can you inherit `private` fields and methods from a superclass? If not, how can a subclass access them?**

   Follow-up:

   - Can you access private members through getter and setter methods in inheritance?
   - What happens when a subclass tries to access a private member of its parent class?

3. **Explain the concept of "is-a" relationship in inheritance with an example.**

   Follow-up:

   - How does the "is-a" relationship differ from the "has-a" relationship (composition)?
   - Can an "is-a" relationship break down if the subclass adds behaviors unrelated to the parent class?

4. **What is the role of polymorphism in collections (like List, Set) in Java?**

   Follow-up:

   - How can you use polymorphism when storing different object types in a collection?

- What happens if you store subclass objects in a collection of the superclass type?

5. **Can an abstract class have a constructor in Java? Why or why not?**

   Follow-up:

   - How would the constructor of an abstract class be used?
   - Can you instantiate an abstract class using its constructor?

6. **What is the difference between an abstract method and a concrete method in Java?**

   Follow-up:

   - How does an abstract method force subclass implementation?
   - Can a class be marked abstract without having any abstract methods?

7. **Can you provide an example where an interface would be more appropriate than an abstract class in Java?**

   Follow-up:

   - Why can't you instantiate an interface, and how is this related to abstraction?
   - What are the benefits of having multiple interfaces versus a single abstract class?

8. **What are the limitations of abstraction in Java?**

   Follow-up:

   - Can abstraction hide all implementation details?
   - How does abstraction affect performance in large-scale systems?

9. **How does abstraction differ between abstract classes and interfaces in terms of method implementations?**

   Follow-up:

   - Can interfaces have method implementations? How has this evolved in Java 8 and later?
   - How does the ability to define default methods in interfaces affect abstraction?

10. **How does Java's `private` access modifier help in achieving encapsulation?**

    Follow-up:

    - Can you still access private variables directly if they are inherited by a subclass?
    - How would you handle scenarios where a `private` field needs to be accessed by external classes?

11. **How does encapsulation relate to immutability in Java?**

    Follow-up:

    - How can encapsulation help create immutable objects?
    - What is the advantage of having immutable objects in multithreaded environments?

12. **Is multiple inheritance possible in Java with classes? Why or why not?**

    Follow-up:

    - How does Java resolve the "Diamond Problem" with multiple inheritance in interfaces?
    - Can you achieve multiple inheritance using interfaces? If so, how does it differ from class inheritance?

13. **Can a class in Java implement multiple interfaces? What are the implications of doing so?**

    Follow-up:

    - How would you handle the situation if two interfaces have methods with the same signature but different behavior?
    - What happens when two default methods from different interfaces conflict?

14. **Why can’t we create an object of an abstract class in Java?**

    Follow-up:

    - If you can't instantiate an abstract class, what is the purpose of its constructor?
    - Can you have a reference variable of an abstract class? How is this useful?

15. **Why is Java's `Object` class considered the superclass of all classes? What are the implications of this?**

    Follow-up:

    - Can a class exist in Java without extending `Object`? Why or why not?
    - How do methods like `toString()`, `equals()`, and `hashCode()` benefit from `Object` being the root class?

16. **What is method hiding in Java, and how does it relate to static methods and inheritance?**

    Follow-up:

    - Can a static method be overridden? Why or why not?
    - What is the difference between method overriding and method hiding in Java?

17. **Explain the architecture of the Java Collections Framework. What are the key interfaces and classes?**

    Follow-up:

    - How does the `Collection` interface relate to the `Map` interface, if at all?
    - Can you name some concrete classes that implement the `Collection` interface?

18. **What is the difference between the `Collection` and `Collections` classes in Java?**

    Follow-up:

    - Why does the `Collections` class contain static utility methods?
    - Give an example of a commonly used method from the `Collections` class.

19. **What is the root interface of the Java Collections Framework, and what are its key subinterfaces?**

    Follow-up:

    - What are the main subinterfaces of the `Collection` interface?
    - How does the `Map` interface fit into the overall hierarchy?

20. **What is the difference between a `List`, `Set`, and `Map` in the Collections Framework?**

Follow-up:

- Which one allows duplicate elements, and which enforces uniqueness?
- Can you give examples of implementations of each interface?

21. **How does the Collections Framework support generic types, and why is this important?**
22. **What are the main differences between an array and a collection in Java?**

    Follow-up:

    - How does the size of an array differ from the size of a collection?
    - Can you store primitive types directly in a collection like you can in an array?

23. **What is the difference between `ArrayList` and `LinkedList` in Java?**

    Follow-up:

    - In which scenarios would you prefer using `LinkedList` over `ArrayList`?
    - How do both handle insertion and deletion operations?

24. **What is a `Set` in Java, and how does it differ from a `List`?**

    Follow-up:

    - Can a `Set` contain duplicate elements? Explain why or why not.
    - What are some examples of `Set` implementations in Java?

25. **How does a `HashSet` work internally in Java?**

    Follow-up:

    - How does a `HashSet` handle duplicate elements?
    - What is the role of `hashCode()` and `equals()` in a `HashSet`?

26. **What is the difference between a `HashMap`, `TreeMap`, and `LinkedHashMap`?**

    Follow-up:

    - Which of these maps maintains insertion order?
    - In which case would you use a `TreeMap` instead of a `HashMap`?

27. **What is a `Queue` in Java, and how does it differ from a `List`?**

    Follow-up:

- What is the difference between a `Queue` and a `Deque`?
- What are common use cases for a `Queue`?

28. **How does a `PriorityQueue` work in Java?**

    Follow-up:

    - What criteria does a `PriorityQueue` use to determine the order of elements?
    - Can you customize the ordering in a `PriorityQueue`? How?

29. **What is an `Iterator` in Java, and how do you use it with collections?**

    Follow-up:

    - What is the difference between `Iterator` and `ListIterator`?
    - Can you modify a collection while iterating over it using an `Iterator`?

30. **What is the difference between `Enumeration` and `Iterator` in Java?**

    Follow-up:

    - Which one is considered legacy, and why?
    - How does a `ListIterator` provide more flexibility than an `Iterator`?

31. **How does a `HashMap` handle collisions?**

    Follow-up:

    - What is the default load factor in a `HashMap`, and how does it affect performance?

32. **What is the difference between `ArrayDeque` and `LinkedList` when used as a `Queue`?**

    Follow-up:

    - In which scenarios would you prefer `ArrayDeque` over `LinkedList` for queue operations?
    - What are the time complexities of `ArrayDeque` operations?

33. **What are the key differences between `HashMap` and `Hashtable`?**

    Follow-up:

    - Which one is synchronized, and what are the performance implications of this?

- Is there any scenario in modern Java where using `Hashtable` is preferable?

34. **How does `TreeMap` sort its keys?**

    Follow-up:

    - What happens if you try to insert a null key into a `TreeMap`?
    - How can you customize the sorting of keys in a `TreeMap`?

35. **What is the difference between `poll()`, `remove()`, and `peek()` in a `Queue`?**

    Follow-up:

    - What does `poll()` return when the queue is empty, and how does that differ from `remove()`?
    - How can you use `peek()` to avoid removing elements from a queue?

36. **How does Java Collections provide thread-safety?**

    Follow-up:

    - What is the difference between `synchronized` collections (like `Vector`, `Hashtable`) and concurrent collections (like `ConcurrentHashMap`, `CopyOnWriteArrayList`)?

37. **Explain the `fail-fast` and `fail-safe` behavior of iterators in the Java Collections Framework.**

    Follow-up:

    - Which collections provide fail-fast iterators?
    - How do fail-safe iterators work internally?

38. **Explain the internal working of a `HashMap` in Java.**

    Follow-up:

    - How does a `HashMap` store key-value pairs?
    - What happens when two keys have the same `hashCode()`? How are collisions handled in Java 8+?

39. **What is `hashCode()` and `equals()` and how are they used in `HashMap`?**

    Follow-up:

    - Why is it important to override both `hashCode()` and `equals()` when using objects as keys in a `HashMap`?

- What happens if the `hashCode()` of two keys is the same but `equals()` returns false?

40. **What is the load factor in a `HashMap` and how does resizing work?**

    Follow-up:

    - What happens when the number of entries in a `HashMap` exceeds the product of its load factor and current capacity?
    - How does resizing affect the performance of `HashMap`?

41. **What are the main differences between `HashMap`, `LinkedHashMap`, and `TreeMap` in terms of performance and internal structure?**

    Follow-up:

    - How does `LinkedHashMap` maintain the insertion order or access order?
    - How does `TreeMap` sort its elements, and what are the performance implications compared to `HashMap`?

42. **Can a `HashMap` have null keys and values? How does Java handle this internally?**

    Follow-up:

    - What is the specific case when a `HashMap` inserts or retrieves a null key?
    - How is the `hashCode()` calculated for null keys?

43. **What is the difference between a thread and a process in Java?**

    Follow-up:

    - How does the memory management differ between threads and processes?
    - Can a Java program contain multiple processes, and how are threads managed within a process?

44. **Provide a real-life example where multithreading is used.**

    Follow-up:

    - How does multithreading improve the performance of your example?
    - How would thread synchronization play a role in this scenario?

45. **How does thread communication work in Java?**

    Follow-up:

- Can you explain how the `wait()` and `notify()` methods are used for thread communication?
- What happens if `notify()` is called when no thread is in the waiting state?

46. **What are the key differences between `Thread` and `Runnable` in Java?**

    Follow-up:

    - When would you prefer implementing `Runnable` over extending the `Thread` class?
    - Can you create multiple threads using a single `Runnable` object?

47. **Explain the concept of thread pooling and why it is important.**

    Follow-up:

    - How does thread pooling help in reducing the overhead of creating new threads?
    - Can you describe a scenario where thread pooling would be beneficial?

48. **What is a deadlock, and how can you detect and avoid it in a real-world Java application?**

    Follow-up:

    - Can you explain how a deadlock might occur in a banking transaction scenario?
    - What are some common strategies to avoid deadlock in multithreaded applications?

49. **How do you handle uncaught exceptions in a multithreaded environment?**

    Follow-up:

    - What is the purpose of the `UncaughtExceptionHandler` interface in Java?
    - How would you implement custom error handling in a multithreaded application?

50. **What is a generic class in Java, and why do we use it?**

    Follow-up:

    - Can a generic class have multiple type parameters?

51. **Explain how to define a generic method in Java.**

Follow-up:

- What is the difference between a generic method and a generic class?

52. **What are wildcards in Java generics, and when would you use `? extends T` vs `? super T`?**

Follow-up:

- Can you explain covariance and contravariance with examples?

53. **What are the two main types of streams in Java, and what do they represent?**

Follow-up:

- What are the differences between byte streams and character streams?

54. **How do you read and write data using `FileInputStream` and `FileOutputStream`?**

Follow-up:

- How do these streams handle binary data?

55. **What are the key differences between `BufferedReader` and `FileReader`?**

Follow-up:

- Why is `BufferedReader` considered more efficient for reading text files?

56. **Explain the working of `InputStream` and `OutputStream` in Java.**

Follow-up:

- How does `BufferedInputStream` improve performance over `InputStream`?

57. **How would you close a file resource properly in Java?**

Follow-up:

- How does the try-with-resources statement improve this process?

58. **How can you read a text file in Java using `BufferedReader`?**

Follow-up:

- What is the benefit of using `BufferedReader` over `FileReader`?

59. **How can you write to a file using `BufferedWriter`?**

Follow-up:

- How would you handle exceptions that occur during file writing?

30. **What is the difference between `PrintWriter` and `BufferedWriter`?**

Follow-up:

- When would you prefer one over the other?

31. **How do you append data to an existing file in Java?**

Follow-up:

- What parameter would you use to ensure that the data is appended rather than overwritten?

32. **What are the common exceptions encountered during file handling?**

Follow-up:

- How do you handle `FileNotFoundException`?

33. **What is serialization in Java, and why do we use it?**

Follow-up:

- What role does the `Serializable` interface play in this process?

34. **How does deserialization work in Java?**

Follow-up:

- What happens if a class does not implement `Serializable` but you try to serialize it?

35. **What is a lambda expression in Java, and how does it simplify coding?**

Follow-up:

- How do lambda expressions support functional programming in Java?

36. **How do you define a lambda expression for a `Comparator` in Java?**

Follow-up:

- Can lambda expressions have multiple parameters?

37. **What are the limitations of lambda expressions in Java?**

Follow-up:

- Can a lambda expression capture variables from its enclosing scope?

38. **How does type inference work with lambda expressions in Java?**

Follow-up:

- Can you explicitly specify the type of parameters in a lambda expression?

69. **What are functional interfaces, and how are they related to lambda expressions?**

Follow-up:

- Can you define your own functional interface in Java?

70. **What is a stream in Java 8, and how does it differ from collections?**

Follow-up:

- How would you use a stream to filter a list of objects?

71. **What are the common terminal operations in streams, such as `reduce()`?**

Follow-up:

- Can you explain the difference between intermediate and terminal operations in streams?

72. **How does the `map()` function work in Java Streams?**

Follow-up:

- Can you use `map()` to convert one type of stream to another?

73. **Explain how `reduce()` is used in the Streams API.**

Follow-up:

- How can `reduce()` be used to find the sum of a list of numbers?

74. **What is the purpose of default methods in Java 8 interfaces?**

Follow-up:

- Can you override default methods in implementing classes?

75. **What is the difference between default and static methods in interfaces?**

Follow-up:

- Can static methods in interfaces be inherited?

76. **Why were default methods introduced in Java 8?**

Follow-up:

- How do default methods help with backward compatibility?

77. **How do static methods in interfaces differ from regular static methods in classes?**

Follow-up:

- Can you call a static method in an interface using an instance?

78. **Can an interface have both default and abstract methods?**

Follow-up:

- How would a class that implements such an interface handle both?

79. **What is the `Optional` class in Java 8, and why is it used?**

Follow-up:

- How does `Optional` help in avoiding `NullPointerException`?

80. **How do you create an empty `Optional` in Java?**

Follow-up:

- What is the difference between `Optional.of()` and `Optional.empty()`?

"We are here to WIN"