Top 50 Java Interview Questions for Experienced Dev.

1. **Explain OOP principles and their implementation in Java.**

   [Hint]: Core principles include Encapsulation, Abstraction, Inheritance, and Polymorphism.

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

   [Hint]: String is immutable; StringBuilder and StringBuffer are mutable, with StringBuffer being thread-safe.

3. **Explain the hashCode() and equals() methods.**

   [Hint]: Used to compare objects and ensure consistency in collections like HashMap.

4. **How does final, finally, and finalize() differ?**

   [Hint]: final defines constants; finally for cleanup; finalize() for object garbage collection.

5. **What are Java annotations? How are custom annotations created?**

   [Hint]: Provide metadata for code; custom annotations use @interface.

6. **Explain Java memory management and garbage collection.**

   [Hint]: Automatic memory management with heap/stack and garbage collection for unused objects.

7. **What are immutable objects, and how do you create them in Java?**

   [Hint]: Objects that cannot be modified after creation; achieved with final fields and no setters.

8. **Explain the working of synchronized in Java.**

   [Hint]: Ensures thread safety by allowing only one thread to access a method/block at a time.

9. **How does volatile differ from transient in Java?**

   [Hint]: volatile ensures visibility in multi-threading; transient prevents serialization of fields.

10. **What is the ExecutorService framework?**

    [Hint]: Manages thread pools and task execution using methods like submit() and shutdown().

11. **What is the difference between ArrayList and LinkedList?**

    [Hint]: ArrayList offers fast random access; LinkedList is efficient for insertions/deletions.

12. **Explain HashMap internals and how collisions are handled.**

    [Hint]: Stores key-value pairs and handles collisions using chaining or open addressing.

13. **What is the difference between HashSet and TreeSet?**

    [Hint]: HashSet is unordered; TreeSet keeps elements in sorted order.

14. **Explain the fail-fast and fail-safe iterators.**

    [Hint]: Fail-fast throws exceptions on structure change; fail-safe works on a copy.

15. **What is the role of Comparable and Comparator interfaces?**

    [Hint]: Comparable compares objects within a class; Comparator compares objects across classes.

16. **Explain Concurrent Collections in Java.**

    [Hint]: Thread-safe collections like ConcurrentHashMap allow safe multi-threaded access.

17. **How does ConcurrentHashMap work internally?**

    [Hint]: Uses segment locking to allow concurrent reads/writes without blocking.

18. What are weak references in Java, and when are they used?

[Hint]: References that allow objects to be garbage-collected when no strong references exist.

19. What is the difference between Queue and Deque?

[Hint]: Queue follows FIFO; Deque supports both FIFO and LIFO.

20. How does TreeMap maintain order?

[Hint]: Maintains sorted order using natural ordering or a custom comparator.

21. What are thread states in Java?

[Hint]: Java threads transition through states like NEW, RUNNABLE, BLOCKED, WAITING, and TERMINATED.

22. Explain ThreadLocal and its use cases.

[Hint]: Provides thread-specific variables, ensuring isolation between threads.

23. What is the difference between wait(), notify(), and notifyAll()?

[Hint]: Used for thread synchronization and communication in multi-threaded environments.

24. How does ReentrantLock work?

[Hint]: A lock that allows the same thread to acquire it multiple times without deadlocking.

25. What are semaphores and their use cases?

[Hint]: Used to limit access to resources and control concurrency.

26. What are daemon threads?

[Hint]: Background threads that terminate when the JVM shuts down.

27. How does the Fork/Join framework work?

[Hint]: Breaks tasks into smaller sub-tasks and processes them in parallel to improve performance.

28. What is the difference between Callable and Runnable?

[Hint]: Runnable doesn't return a result; Callable can return a result or throw exceptions.

29. Explain deadlock, livelock, and starvation.

[Hint]: Deadlock: circular waiting; Livelock: active waiting; Starvation: no CPU time for threads.

30. How can deadlocks be avoided in Java?

[Hint]: Use resource ordering and timeouts to prevent deadlocks in multi-threaded programs.

31. Explain streams in Java.

[Hint]: A functional way to process sequences of elements in collections or arrays.

32. What is the difference between map() and flatMap()?

[Hint]: map() transforms each element; flatMap() flattens nested structures into a single stream.

33. How do Optional and ifPresent work?

[Hint]: Optional helps avoid NullPointerException; ifPresent() executes code if value is present.

34. What is the use of default methods in interfaces?

[Hint]: Allows interfaces to have method implementations while maintaining backward compatibility.

35. Explain method references and constructor references.

[Hint]: Short syntax for referring to methods and constructors in lambda expressions.

36. What is the purpose of the Collector class in Java?

[Hint]: Provides utility methods for reducing streams like toList(), joining(), etc.

37. Explain functional interfaces and provide examples.

[Hint]: Interfaces with a single abstract method, commonly used with lambda expressions.

38. How do you create custom collectors?

[Hint]: Implement custom collection logic using the Collector interface.

39. What are the new time and date APIs introduced in Java 8?

[Hint]: Improved date and time handling with classes like LocalDate, LocalTime, and ZonedDateTime.

40. How do you implement parallel streams?

[Hint]: Enables parallel processing of data in streams using parallel() for improved performance.

41. How does the Factory pattern differ from the Builder pattern?

[Hint]: Factory creates objects without exposing creation logic; Builder constructs complex objects step by step.

42. Explain Dependency Injection and how Spring implements it.

[Hint]: Spring uses DI to manage object dependencies automatically through inversion of control.

43. What are the SOLID principles in Java?

[Hint]: Five design principles for creating maintainable and flexible object-oriented systems.

44. How does the Observer pattern work?

[Hint]: One object notifies dependent objects about state changes, commonly used in event handling.

45. When would you use the Prototype pattern?

[Hint]: Creates new objects by cloning existing ones, useful when creating similar objects.

46. Explain the Template Method pattern.

[Hint]: Defines a method skeleton in a superclass and allows subclasses to implement specific steps.

47. What is the Strategy design pattern?

[Hint]: Defines a family of algorithms and makes them interchangeable at runtime.

48. How does the Proxy pattern work in Java?

[Hint]: Controls access to an object by acting as a surrogate, used for lazy initialization or access control.

49. What is the difference between a decorator and an adapter?

[Hint]: Decorator adds functionality dynamically; Adapter converts one interface to another.

50. Explain the Singleton design pattern.

[Hint]: Ensures only one instance of a class is created, with global access to that instance.