**1. Core Multithreading Concepts**

1. **What is multithreading in Java?**
   * Follow-up: How is it different from multiprocessing?
2. **What are the differences between a process and a thread?**
3. **What is the lifecycle of a thread?**
   * Key states: New, Runnable, Running, Blocked/Waiting, Terminated.
4. **How do you create a thread in Java?**
   * Extending Thread vs. implementing Runnable.
   * Follow-up: Which is better and why?
5. **What are Daemon threads?**
   * How do you create them, and what are their use cases?

**2. Synchronization**

1. **Why is synchronization needed in multithreading?**
   * Follow-up: Explain race conditions and critical sections.
2. **What are the different ways to achieve synchronization in Java?**
   * synchronized blocks or methods.
   * Lock interface (e.g., ReentrantLock).
3. **What is the difference between synchronized and Lock?**
4. **What are deadlocks?**
   * Follow-up: How can you avoid deadlocks in Java?
5. **What are volatile and transient keywords?**
   * How does volatile prevent thread safety issues?

**3. Thread Communication**

1. **How do threads communicate with each other?**
   * wait(), notify(), notifyAll().
   * Follow-up: What are their common pitfalls?
2. **What happens if wait() is called without holding a lock?**
3. **What is the difference between sleep() and wait()?**
4. **How does join() work in Java?**

**4. Thread Safety**

1. **What is thread safety?**
   * Examples of thread-safe classes (e.g., StringBuffer, ConcurrentHashMap).
2. **What are immutable objects, and how do they help in thread safety?**
3. **What are some ways to make a class thread-safe?**
   * Immutable classes, synchronized methods, Atomic classes.
4. **What is the ThreadLocal class?**
   * When should you use it?

**5. Advanced Concepts**

1. **What is a thread pool?**
   * Follow-up: Why is it better to use thread pools instead of creating new threads?
2. **What is the Executor framework in Java?**
   * Types of thread pools (e.g., FixedThreadPool, CachedThreadPool).
3. **What is the difference between Callable and Runnable?**
   * When would you use one over the other?
4. **What is ForkJoinPool in Java?**
   * Follow-up: How is it different from traditional thread pools?
5. **What is the difference between Future and CompletableFuture?**
6. **What are CountDownLatch and CyclicBarrier, and how do they differ?**
7. **Explain the role of Semaphore in concurrency control.**

**6. Common Problems/Scenarios**

1. **What happens if two threads call the same synchronized method on the same object?**
   * Follow-up: What if they call different synchronized methods?
2. **How would you implement a producer-consumer problem in Java?**
   * Use of wait() and notify().
   * Alternative: Blocking queues like ArrayBlockingQueue.
3. **How would you prevent a thread from running indefinitely (infinite loop)?**
4. **How would you implement a thread-safe singleton in Java?**
5. **How can you implement a thread-safe counter?**

**7. Debugging and Practical Scenarios**

1. **How would you identify and fix a deadlock in your code?**
2. **How would you test a multithreaded application for thread safety?**
   * Tools or approaches (e.g., stress testing).
3. **What are some performance considerations when working with multithreading?**
4. **What is context switching?**
   * How does it impact performance?

**Behavioral/Scenario-Based Questions**

1. **Have you worked with multithreading in your projects?**
   * Be prepared to explain specific use cases or challenges you solved.
2. **Describe a time when you encountered a bug related to multithreading. How did you resolve it?**
3. **How do you decide when to use multithreading in an application?**

**Preparation Tips**

* Be ready to write small code snippets (e.g., thread creation, synchronized blocks).
* Practice implementing producer-consumer, thread-safe singleton, and thread pools.
* Understand real-world applications of multithreading (e.g., parallel processing, background tasks).

If you'd like detailed answers or code examples for any of these, let me know!