**Basic Java Multithreading Questions**

1. **What is a thread in Java?**  
   A thread is a lightweight sub-process, the smallest unit of processing. Java allows concurrent execution of two or more parts of a program using threads.
2. **How do you create a thread in Java?**
   * By **extending Thread** class and overriding run() method.
   * By **implementing Runnable** interface and passing it to a Thread object.

java

CopyEdit

public class MyRunnable implements Runnable {

public void run() {

System.out.println("Thread running...");

}

}

new Thread(new MyRunnable()).start();

1. **Difference between start() and run() method?**
   * start(): Creates a new thread and executes run() in it.
   * run(): Executes in the current thread like a normal method.

**🔹 Intermediate Questions**

1. **What is the difference between Runnable and Callable?**
   * Runnable does not return a result or throw checked exceptions.
   * Callable (Java 5+) returns a result and can throw checked exceptions.
   * Used with ExecutorService.
2. **How does synchronization work in Java?**
   * Used to avoid thread interference and memory consistency errors.
   * Synchronized methods or blocks allow only one thread to access the resource.

java

CopyEdit

synchronized void update() {

// only one thread can enter here at a time

}

1. **What is a deadlock? How do you avoid it?**
   * A situation where two or more threads are blocked forever, waiting for each other.  
     **Avoid by:**
   * Lock ordering
   * Timeout
   * Using tryLock()

**🔹 Advanced Questions for 12+ Yrs**

1. **What are thread-safe classes in Java?**  
   Examples: Vector, Hashtable, StringBuffer. These use synchronized methods to ensure thread safety.
2. **What is ThreadLocal and when should it be used?**
   * Provides thread-local variables.
   * Each thread accessing such a variable has its own independent copy.

java

CopyEdit

ThreadLocal<Integer> tl = ThreadLocal.withInitial(() -> 1);

1. **How does volatile differ from synchronized?**
   * volatile ensures visibility but **not atomicity**.
   * synchronized ensures both **atomicity and visibility**.
2. **Explain the Java Memory Model (JMM).**

* Describes how threads interact through memory and what behaviors are allowed in multithreaded programs.
* Ensures **happens-before** relationship for correct visibility.

1. **How does ExecutorService work?**

* A high-level replacement for managing threads manually.

java

CopyEdit

ExecutorService executor = Executors.newFixedThreadPool(5);

executor.submit(() -> { /\* task \*/ });

executor.shutdown();

**🔹 Real-World Case/Design Questions**

1. **Design a thread pool or explain how to handle high-load concurrent requests.**

* Use ThreadPoolExecutor with proper core/max pool sizes.
* Use bounded BlockingQueue.
* Handle rejected tasks with RejectedExecutionHandler.

1. **How do you debug a deadlock in production?**

* Use jstack to dump thread info.
* Analyze waiting and locked states.
* Tools: VisualVM, JConsole.

1. **What’s the difference between wait(), notify(), and notifyAll()?**

* wait(): Puts the thread in waiting state until notified.
* notify(): Wakes up one waiting thread.
* notifyAll(): Wakes up all waiting threads.

Must be used inside a synchronized block or method.

1. **What are atomic classes in Java?**

* From java.util.concurrent.atomic package, e.g., AtomicInteger.
* Provide lock-free thread-safe operations.

**🔹 Code Snippet: Producer-Consumer Using BlockingQueue**

java

CopyEdit

BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(10);

Runnable producer = () -> {

try {

while (true) {

queue.put(new Random().nextInt());

}

} catch (InterruptedException e) { }

};

Runnable consumer = () -> {

try {

while (true) {

System.out.println(queue.take());

}

} catch (InterruptedException e) { }

};

new Thread(producer).start();

new Thread(consumer).start();

**🔥 Advanced Java Multithreading Interview Q&A (12+ Yrs)**

**✅ 1. What are the key challenges in multithreaded application design?**

**Answer:**

* **Race Conditions**
* **Deadlocks**
* **Livelocks**
* **Thread starvation**
* **Context switching overhead**
* **Memory consistency (visibility issues)**
* **Testing and reproducibility**

**Cross-Question:**

How would you detect thread starvation in production?  
→ Use monitoring tools (e.g., Java Flight Recorder, JMC), metrics for long waits, thread dumps.

**✅ 2. Explain the Java Memory Model and its impact on multithreading.**

**Answer:**

* Defines how threads interact through memory.
* Key concept: **Happens-before relationship**.
* Ensures visibility and ordering of variables between threads.
* volatile, synchronized, final, Thread.start/join establish happens-before relationships.

**Cross-Question:**

What’s the difference between visibility and atomicity?  
→ Visibility = what one thread sees of another’s updates; Atomicity = indivisible operations.

**✅ 3. How does ReentrantLock compare to synchronized?**

| **Feature** | **synchronized** | **ReentrantLock** |
| --- | --- | --- |
| Reentrant | Yes | Yes |
| Timeout | No | Yes (tryLock) |
| Interruptible lock | No | Yes (lockInterruptibly) |
| Fairness | No | Yes (optional) |
| Condition support | No | Yes |

**Code Example:**

java

CopyEdit

Lock lock = new ReentrantLock(true); // fair lock

lock.lock();

try {

// critical section

} finally {

lock.unlock();

}

**Cross-Question:**

When would you prefer ReentrantLock over synchronized?  
→ When fairness, timed lock acquisition, or interruptible waits are required.

**✅ 4. What are thread-safe design patterns?**

**Answer:**

* **Immutable objects**
* **Producer-Consumer using BlockingQueue**
* **Thread Pool (using Executors)**
* **Read-Write Lock pattern**
* **Double-Checked Locking with volatile**

**✅ 5. What is a ThreadPoolExecutor and how would you configure it for high-throughput systems?**

**Answer:**

* Core class in Java concurrency for handling thread pools.

java

CopyEdit

ThreadPoolExecutor executor = new ThreadPoolExecutor(

10, 50, 60, TimeUnit.SECONDS,

new LinkedBlockingQueue<>(100),

new ThreadPoolExecutor.CallerRunsPolicy());

* Tune the following:
  + corePoolSize, maximumPoolSize
  + Queue type (ArrayBlockingQueue vs LinkedBlockingQueue)
  + Rejection policy
  + Custom thread factory (for naming, priority)

**Cross-Question:**

What happens when the queue is full and all threads are busy?  
→ The RejectionHandler decides the fate: reject, run in caller thread, discard, etc.

**✅ 6. Explain the Fork/Join Framework. When would you use it?**

**Answer:**

* Introduced in Java 7 for **divide-and-conquer** tasks.
* Best suited for **recursive parallelism**, like sorting, searching, or matrix multiplication.

java

CopyEdit

ForkJoinPool pool = new ForkJoinPool();

pool.invoke(new MyRecursiveTask());

**Cross-Question:**

How does ForkJoinPool differ from ExecutorService?  
→ ForkJoinPool uses work-stealing, optimized for CPU-bound tasks.

**✅ 7. Difference between wait()/notify() and Condition interface?**

| **Feature** | **wait/notify** | **Condition** |
| --- | --- | --- |
| Locking object | Intrinsic lock | Explicit lock |
| Fairness | No | Yes |
| Multiple conditions | No (one monitor queue) | Yes (many conditions per lock) |
| Interruptible? | No (legacy behavior) | Yes |

**✅ 8. How do you implement a custom blocking queue without using java.util.concurrent?**

**Answer:**

java

CopyEdit

class CustomBlockingQueue {

private Queue<Integer> queue = new LinkedList<>();

private int capacity = 10;

public synchronized void enqueue(int value) throws InterruptedException {

while (queue.size() == capacity) wait();

queue.add(value);

notifyAll();

}

public synchronized int dequeue() throws InterruptedException {

while (queue.isEmpty()) wait();

int val = queue.poll();

notifyAll();

return val;

}

}

**Cross-Question:**

How would you add fairness here?  
→ Use separate condition variables or track thread arrival order.

**✅ 9. What are some best practices you follow in production multithreading systems?**

* Avoid using low-level synchronization unless necessary.
* Use high-level concurrency APIs: ExecutorService, ConcurrentHashMap, BlockingQueue.
* Profile & benchmark under real load.
* Never block on I/O in shared thread pools.
* Monitor thread states regularly (ThreadMXBean, jstack).

**✅ 10. Have you handled deadlocks or performance bottlenecks in production? How?**

**Answer:**  
Yes, by:

* Taking thread dumps via jstack
* Analyzing lock graphs (e.g., using VisualVM or JMC)
* Redesigning critical sections
* Minimizing lock scope
* Using lock ordering to avoid circular waits
* Replacing blocking locks with lock-free structures (e.g., Atomic\* classes)

**Cross-Question:**

Can you simulate a deadlock in Java?  
→ Yes, using two threads locking two resources in opposite order.

**✅ 11. Explain CompletableFuture with an example.**

**Answer:**  
Used for **asynchronous programming** with callbacks, chaining, and combining results.

java

CopyEdit

CompletableFuture.supplyAsync(() -> compute())

.thenApply(data -> transform(data))

.thenAccept(System.out::println);

**✅ 12. What are alternatives to Java threads in reactive systems?**

* Use **Project Reactor**, **RxJava**, **Vert.x**, or **Akka**
* They follow **event-driven** and **non-blocking** models.
* Based on **reactive streams** and **backpressure** principles.