Java Multithreading – Interview Question Bank

BASIC LEVEL (Concepts & Simple Coding)

1. MCQs

Q1: Which method starts a new thread in Java?
a) run()
b) start() ✅
c) execute()
d) init()

Q2: Which interface can be implemented to create a thread?
a) Runnable ✅
b) Callable
c) Comparator
d) Serializable

Q3: Which method is used to pause the current thread for a specified time?
a) wait()
b) sleep() ✅
c) yield()
d) join()

Q4: Which thread state is reached when a thread is waiting to be scheduled?
a) New
b) Runnable ✅
c) Running
d) Terminated

---

2. Scenario-Based Questions

Q5: How do you create a thread using Runnable interface?
Answer:

```
class MyRunnable implements Runnable {
    public void run() { System.out.println("Thread running"); }
```

```
}
public class Test {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```

Q6: Difference between Thread.run() and Thread.start().

- run(): Called like a normal method, executes in the same thread.
- start(): Creates a new thread and calls run() internally.

3. Coding Questions

Q7: Create a thread by extending Thread class.

```
class MyThread extends Thread {
    public void run() { System.out.println("Thread running"); }
}
public class Test {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

Q8: Make two threads print numbers 1-5 simultaneously.

```
class MyThread extends Thread {
    String name;
```

```
    MyThread(String name) { this.name = name; }
    public void run() {
        for(int i=1;i<=5;i++) System.out.println(name + ": " + i);
    }
}
public class Test {
    public static void main(String[] args) {
        new MyThread("T1").start();
        new MyThread("T2").start();
    }
}
```

INTERMEDIATE LEVEL (Synchronization, Join, Sleep, Yield)

1. MCQs

Q9: Which keyword is used for synchronization in Java?
a) wait
b) notify
c) synchronized ✅
d) volatile

Q10: Which method waits for another thread to complete?
a) wait()
b) join() ✅
c) sleep()
d) notify()

Q11: volatile keyword guarantees:
a) Mutual exclusion
b) Visibility of changes across threads ✅
c) Atomicity
d) Thread priority

---

2. Scenario-Based Questions

Q12: What is a race condition? How to prevent it?
Answer: Race condition occurs when multiple threads access shared data concurrently without proper synchronization. Prevent using **synchronized blocks/methods** or **locks**.

Q13: Difference between sleep() and wait().

Feature	**sleep()**	**wait()**
Belongs to	Thread class	Object class
Releases lock?	No	Yes
Used in	Thread scheduling	Inter-thread communication

3. Coding Questions

Q14: Synchronize a method to prevent race condition.

```
class Counter {
   int count = 0;
   synchronized void increment() { count++; }
}
public class Test {
   public static void main(String[] args) throws Exception {
      Counter c = new Counter();
      Runnable r = () -> {
         for(int i=0;i<1000;i++) c.increment();
      };
      Thread t1 = new Thread(r), t2 = new Thread(r);
      t1.start(); t2.start();
      t1.join(); t2.join();
```

```
        System.out.println(c.count); // 2000
    }
}
```

Q15: Demonstrate yield() usage.

```
class MyThread extends Thread {
    public void run() {
        for(int i=0;i<5;i++){
            System.out.println(getName()+" running");
            Thread.yield();
        }
    }
}
public class Test {
    public static void main(String[] args){
        new MyThread().start();
        new MyThread().start();
    }
}
```

ADVANCED LEVEL (Executor, Callable, Locks, Deadlock, Thread Pools)

1. MCQs

Q16: Which interface allows threads to return a result?
a) Runnable
b) Callable ✅
c) Executor
d) FutureTask

Q17: Which class is used to implement thread-safe collections?
a) ArrayList
b) Vector ✅
c) HashMap
d) LinkedList

Q18: Which of these can cause deadlock?
a) Nested synchronized blocks ✅
b) Sleep()
c) Yield()
d) Thread.join()

2. Scenario-Based Questions

Q19: How to avoid deadlocks?

- Use **lock ordering** consistently.
- Use tryLock() with timeout (ReentrantLock).
- Minimize the scope of synchronized blocks.

Q20: Difference between ExecutorService and manual thread creation.

- ExecutorService manages a **thread pool** and lifecycle.
- Manual thread creation is **less efficient** for large number of threads.

---

3. Coding Questions

Q21: Callable and Future example.

```
import java.util.concurrent.*;
class MyCallable implements Callable<Integer> {
   public Integer call() { return 123; }
}
public class Test {
   public static void main(String[] args) throws Exception {
```

```
        ExecutorService exec = Executors.newFixedThreadPool(1);
        Future<Integer> future = exec.submit(new MyCallable());
        System.out.println("Result: " + future.get());
        exec.shutdown();
    }
}
```

Q22: Deadlock example (simplified).

```
class A { synchronized void foo(B b){ b.last(); } synchronized void last(){} }
class B { synchronized void bar(A a){ a.last(); } synchronized void last(){} }
```

- Thread1: a.foo(b)
- Thread2: b.bar(a) → deadlock occurs.

Q23: Using ReentrantLock.

```
import java.util.concurrent.locks.*;
Lock lock = new ReentrantLock();
lock.lock();
try { // critical section }
finally { lock.unlock(); }
```

Q24: Thread-safe producer-consumer using BlockingQueue.

```
import java.util.concurrent.*;
BlockingQueue<Integer> q = new ArrayBlockingQueue<>(5);
```