
✅ Multithreading in Java – Quick Smart Notes

🧠 What is Multithreading?

Multithreading is the ability of a **CPU or a single core** to execute **multiple threads** concurrently, enabling multitasking in a Java program.

A **thread** is the smallest unit of execution within a process.

✅ Java Packages for Multithreading

Package	Purpose
java.lang	Contains Thread, Runnable, Exception, Object (with wait(), notify())
java.util.concurrent	Advanced thread control: Executor, Semaphore, CountDownLatch, Future, Callable

---

✅ Thread Class Hierarchy

```
java.lang.Object
  └── java.lang.Thread (implements Runnable)
```

java.lang.Runnable → Functional Interface (only one method: run())

✅ Ways to Create a Thread

1. Extending Thread Class

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread running...");
   }
}
```

2. Implementing Runnable Interface

```
class MyRunnable implements Runnable {
```

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

3. Using ExecutorService (from java.util.concurrent)

```
ExecutorService executor = Executors.newFixedThreadPool(3);
executor.execute(new MyRunnable());
```

✅ **Thread Lifecycle**

State	Method/Trigger
New	Thread created
Runnable	.start() called
Running	Scheduler picks it
Blocked/Waiting	.wait() or blocked on I/O
Terminated	Completed or exception occurred

✅ **Important Thread Methods**

Method	Purpose
start()	Starts thread (calls run() internally)
run()	Code that runs in thread
sleep(ms)	Pause execution temporarily
join()	Wait for thread to die
yield()	Suggests thread scheduler to switch
isAlive()	Check if thread is alive
interrupt()	Interrupts sleeping/waiting thread

Method	Purpose
setPriority(int)	Set priority from 1 (min) to 10 (max)

✅ Daemon Thread

A **daemon thread** is a background thread that dies when all user threads finish.

Example:

Thread t = new Thread();

t.setDaemon(true);

- GC (Garbage Collector) runs as a daemon thread.
- Must be set **before** start() is called.

✅ Thread Priorities

Priority Level	Value
MIN_PRIORITY	1
NORM_PRIORITY	5
MAX_PRIORITY	10

✅ Key Interfaces & Classes

Class / Interface	Extends / Implements	Role
Thread	extends Object, implements Runnable	Represents a thread
Runnable	Functional Interface	Represents task for thread
Callable<V>	Functional Interface (concurrent)	Returns value + throws exception
ExecutorService	Interface	Manages and executes tasks
Future<V>	Interface	Result of async computation

Class / Interface	Extends / Implements	Role
ThreadGroup	Class	Manage group of threads

✅ **Tricky One-Liner Concepts**

🧠 **1. start() vs run()**

- start() creates a new thread.
- run() just executes the method in the current thread.

🧠 **2. A thread can't be restarted once it is dead.**

🧠 **3. Runnable is preferred over extending Thread (for multiple inheritance).**

🧠 **4. Thread.sleep() is a static method; it pauses the **current thread**, not necessarily the one it's called on.

🧠 **5. Calling wait() requires synchronized block.**

🧠 **6. You can call start() only once on a thread object. Re-calling gives IllegalThreadStateException.**

🧠 **7. Daemon threads die when main threads die — they don't stop JVM from exiting.**

🧠 **8. yield() is just a hint — scheduler may ignore it.**

🧠 **9. ExecutorService is better for managing large number of threads.**

🧠 **10. Callable is like Runnable but returns a result and can throw checked exceptions.**

✅ **Sample Code with Explanation**

```
public class MyThread extends Thread {
  public void run() {
    System.out.println("Running thread...");
  }
```

```
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start(); // creates new thread
    }
}
```

OR

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

    public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnable());
        t1.start();
    }
}
```

✅ **Summary Table**

Concept	Meaning/Use
Thread	Class to create threads
Runnable	Interface with run() method
ExecutorService	Thread pool manager
sleep(ms)	Pause thread temporarily
join()	Wait for another thread to complete
synchronized	Lock access to prevent race conditions
daemon	Background threads (GC, loggers)
