

Threads in Java explanation



give me the deep and detailed explaination of Threads in java so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explaination

Below is **the deepest, most detailed, and easiest-to-understand explanation of *Threads in Java*** you will ever need.

I will explain **what threads are, why they exist, how they work internally, how Java handles them, lifecycle, scheduling, context switching, synchronization, problems, examples, and best practices** — everything step-by-step so no doubts remain.

✓ 1. What Is a Thread in Java (Simple + Deep Meaning)?

Simple Definition

A **thread** is a lightweight, independent path of execution inside a program.

Deep Definition

A Java program runs inside a **JVM process**.

Inside the JVM, multiple *threads* can run **simultaneously**, each executing a different part of the code **independently but sharing the same memory**.

🔥 Important Insight

- A **process** = independent program with its own memory.
- A **thread** = a sub-task inside a process which **shares memory** with other threads.

✓ Example

A browser process has:

- One thread for UI
- One for network
- One for video rendering
- One for input handling

Similarly, your Java application can have:

- One thread performing database operations

- One thread processing API requests
 - One thread for background calculations
-

✓ 2. Why Do We Need Threads?

Threads help solve real-world problems like:

1. Parallelism

Perform multiple tasks at the same time.

Example:

Processing 10,000 users → handle multiple users using different threads.

2. Responsiveness

UI should not freeze.

Example:

Downloading a file in a separate thread keeps UI free.

3. Better CPU Utilization

Modern CPUs have 4, 8, 16+ cores → threads let you use them.

4. Asynchronous Programming

Threads help run slow tasks like:

- I/O operations
- File reading
- Database queries

without blocking the entire program.

✓ 3. Thread vs Process (Deep Understanding)

Feature	Process	Thread
Memory	Separate for each process	Shared among all threads
Communication	Slow, expensive (IPC)	Fast (shared memory)
Creation Cost	Heavy	Lightweight
Crash	One process does not crash another	One thread can crash the whole process

✓ Important Note

Because threads share memory → fast

But sharing memory → causes **race conditions**, so synchronization is needed.

✓ 4. How JVM Creates Threads Internally

When JVM starts:

- The first thread is **main thread**
- JVM also creates system threads:
 - Garbage Collector thread
 - Finalizer thread
 - Reference Handler thread
 - Signal Dispatcher thread

All user-created threads run **beside** these JVM threads.

✓ 5. How to Create Threads in Java

You can create threads in **two ways**.

Method 1 — Extending Thread class

```
java

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

public class Main {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start(); // starts a new thread
    }
}
```

! Important

Calling **run()** does NOT start a new thread.

Calling **start()** creates a new thread and then calls **run()** internally.

Method 2 — Implementing Runnable interface (Recommended)

```
java

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

```

    }
}

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

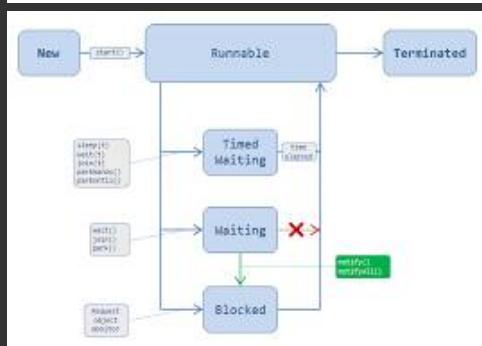
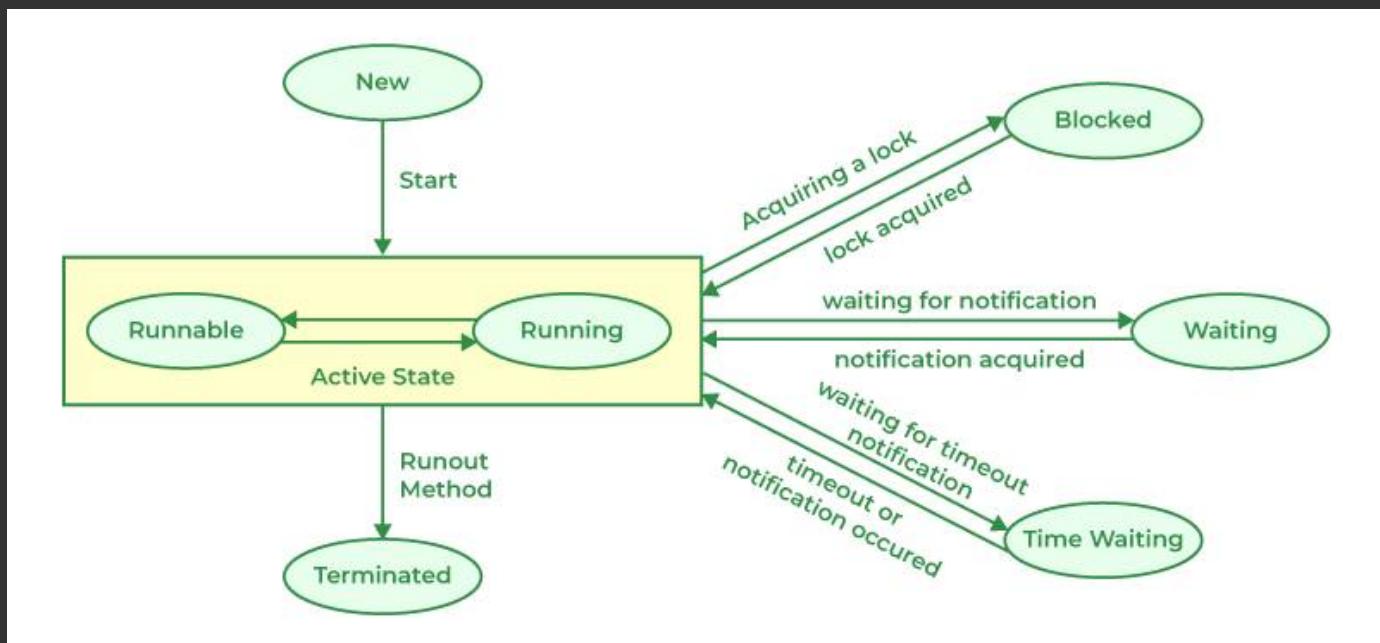
```

Why Runnable is better?

- Java does not support multiple inheritance
- Runnable separates *task* from *thread*
- More flexible for thread pools

✓ 6. Thread Lifecycle (VERY IMPORTANT)

A thread goes through **5 main states**:



1. NEW

Thread object is created but not started.

2. RUNNABLE

Thread is ready to run or running.

3. BLOCKED

Thread is waiting for a lock (synchronization).

4. WAITING / TIMED_WAITING

Thread waits:

- `wait()`
- `sleep()`
- `join()`
- `park()`

5. TERMINATED

Thread finished execution OR stopped due to error.

✓ 7. Thread Scheduling (How the CPU Decides Which Thread Runs)

Two factors decide thread execution:

✓ 1. Preemptive Scheduling

OS can pause one thread and run another.

✓ 2. Time-Slicing

Each thread gets a short time slice.

Thread priority

Threads have priority from **1 to 10**.

```
java  
t.setPriority(Thread.MAX_PRIORITY);
```

But priorities are:

- **Hints**, not guaranteed
- Actual scheduling depends on OS

✓ 8. Context Switching (Internal Working)

Context switching means:

- CPU pauses one thread
- Saves its state (Program Counter, registers)
- Loads another thread's state
- Resumes it

Switching is **fast but not free** → too many threads slow performance.

✓ 9. Thread Methods (Important)

Method	Meaning
<code>start()</code>	starts new thread
<code>run()</code>	code executed by thread
<code>sleep(ms)</code>	thread pauses
<code>join()</code>	wait for another thread to finish
<code>yield()</code>	tells scheduler to give chance to other threads
<code>isAlive()</code>	checks if thread is running

✓ 10. Thread Synchronization (VERY IMPORTANT)

Because threads share memory → they can cause:

! Race Conditions

Example:

Two threads incrementing same variable → wrong result.

Solution → **synchronized keyword**

```
java
synchronized void increment() {
    count++;
}
```

This ensures **only one thread** executes the method at a time.

🔥 11. Types of Locks

✓ Intrinsic Lock (Monitor Lock)

Used by:

- synchronized methods
- synchronized blocks

✓ Reentrant Lock

More advanced, flexible lock from `java.util.concurrent.locks`

🔥 12. Thread Communication (`wait()`, `notify()`, `notifyAll()`)

These methods allow threads to communicate.

✓ `wait()`

Thread releases lock and waits.

✓ `notify()`

Wakes up **one** waiting thread.

✓ `notifyAll()`

Wakes up **all** waiting threads.

✓ 13. Daemon Threads

Daemon threads support main threads.

Examples:

- Garbage Collector thread
- Scheduler thread

```
java
t.setDaemon(true);
```

Daemon threads stop automatically when main thread ends.

🔥 14. Thread Pools (Highly Recommended)

Creating too many threads slows performance.

Java provides **Executor framework**:

```
java
ExecutorService service = Executors.newFixedThreadPool(5);
service.submit(() -> {
```

```
    System.out.println("Task executed");
});
```

Advantages:

- Faster execution
- Reuses threads
- Avoids overhead

🔥 15. Common Problems in Multithreading

1. Race condition

Multiple threads access shared data incorrectly.

2. Deadlock

Two threads wait forever for each other's lock.

3. Starvation

Thread never gets CPU time.

4. Livelock

Threads keep reacting to each other → no progress.

5. Thread Interference

Two threads modifying shared variable give incorrect output.

🔥 16. Real-World Examples Where Threads Are Used

- Web servers handle multiple clients → each request in thread pool
- Video players → audio, video decoding
- Chat applications → background message listener
- Banking systems → concurrent transactions
- Search engines → parallel indexing

✓ 17. Simple Multithreading Example

```
java

class Task1 extends Thread {
    public void run() {
        for(int i=0; i<5; i++)
```

```

        System.out.println("Task 1 running...");
    }

}

class Task2 extends Thread {
    public void run() {
        for(int i=0; i<5; i++)
            System.out.println("Task 2 running...");
    }
}

public class Main {
    public static void main(String[] args) {
        new Task1().start();
        new Task2().start();
    }
}

```

Output will be mixed:

arduino

```

Task 1 running...
Task 2 running...
Task 1 running...
Task 2 running...
...

```

Because threads run **concurrently**.

18. Summary (No Doubts Version)

Concept	Meaning
Thread	Small execution unit sharing memory
Runnable	Preferred way to create task
start()	Starts new thread
run()	Thread logic
Sleep	Pause thread
Join	Wait for another thread
Synchronization	Avoid race conditions
Wait/Notify	Thread communication
ExecutorService	Manages thread pools
Daemon thread	Background service thread

If you want...

I can also provide:

- Thread Interview Questions**
- Diagram-based explanations (with visuals)**

- ✓ Practical real-world multithreading project
- ✓ Explanation of Executors, Future, Callable, Fork/Join, CompletableFuture
- ✓ Deadlock examples with diagrams

Just tell me — I'll explain everything in the same simple, detailed way.