
COMPLETE DOCUMENT: THREADS IN JAVA

Introduction to Threads

◆ What is a Thread?

A **Thread** is a lightweight subprocess.
It is the smallest unit of execution inside a process.

When you run a Java program:

```
java MyProgram
```

JVM creates:

- 1 Process
- 1 Main Thread (automatically)

◆ Process vs Thread

Process	Thread
Independent program	Part of process
Own memory	Shared memory
Heavyweight	Lightweight
Slower creation	Faster creation

Why Multithreading?

Without threads:

- One task at a time
- Application freezes

With threads:

- Multiple tasks simultaneously
- Better performance
- Better CPU utilization
- Responsive UI

Example:

- Downloading file
 - Playing music
 - Updating UI
-

Thread Creation Methods

Method 1: 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();  
    }  
}
```

Method 2: Implementing Runnable

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

Which is Better?

Runnable is better because:

- Supports multiple inheritance
 - Separation of task and thread
-

4 start() vs run()

start()	run()
---------	-------

Creates new thread	Normal method call
--------------------	--------------------

JVM calls run()	You call directly
-----------------	-------------------

New stack created	No new stack
-------------------	--------------

5 Thread Life Cycle

States:

NEW

RUNNABLE

RUNNING

BLOCKED

WAITING

TIMED_WAITING

TERMINATED

◆ NEW

Thread created but not started.

◆ RUNNABLE

After start().

◆ RUNNING

Scheduler gives CPU.

◆ BLOCKED

Waiting for lock.

◆ WAITING

Waiting indefinitely (join(), wait()).

◆ TIMED_WAITING

Waiting for time (sleep(), join(time)).

◆ TERMINATED

run() finished.

6 Important Thread Methods

start()

Starts thread.

sleep(ms)

Pauses current thread.

join()

Waits for another thread to finish.

yield()

Hints scheduler to switch.

interrupt()

Interrupts thread.

setPriority()

Sets priority (1–10).

setName()

Sets thread name.

7 sleep() Example

```
class SleepExample extends Thread {  
    public void run() {  
        try {  
            Thread.sleep(1000);  
            System.out.println("After 1 second");  
        } catch (InterruptedException e) {}  
    }  
}
```

Moves thread to:

TIMED_WAITING

8 join() Example

Main waits for child:

```
t.join();
```

Main goes to WAITING state.

9 yield() Example

Thread.yield();

Moves from RUNNING → RUNNABLE

Not guaranteed to switch.

interrupt() Example

t.interrupt();

If sleeping → throws InterruptedException

1 Synchronization

Problem: Race Condition

Two threads modifying shared data:

count++

Internally:

1. Read
2. Modify
3. Write

Two threads can corrupt data.

Solution: synchronized

Method level:

```
synchronized void increment() {}
```

Block level:

```
synchronized(this) {}
```

What Happens Internally?

- Object lock (monitor) acquired
 - Only one thread allowed
 - Others go to BLOCKED state
-

2 Inter-Thread Communication

Used when threads depend on each other.

Methods:

- `wait()`
- `notify()`
- `notifyAll()`

Must be inside synchronized block.

`wait()`

Thread releases lock and goes to WAITING.

`notify()`

Wakes one waiting thread.

`notifyAll()`

Wakes all waiting threads.

1 3 Producer-Consumer Example (Basic Idea)

Producer produces data.

Consumer consumes data.

They coordinate using wait/notify.

1 4 Deadlock

When two threads hold locks and wait for each other.

Example:

Thread 1 → lock A → waiting for B

Thread 2 → lock B → waiting for A

Program freezes.

How to Avoid?

- Lock ordering
 - Avoid nested locks
 - Use `tryLock()`
-

1 5 Daemon Thread

Background thread.

Example:

- Garbage Collector

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

If only daemon threads remain → JVM exits.

1 6 Thread Scheduler

Managed by OS.

Two types:

- Preemptive scheduling
- Time slicing

Priority is just a hint.

1 7 Thread States Transition Diagram (Logical)

NEW → RUNNABLE → RUNNING



BLOCKED



WAITING



TERMINATED

1 8 sleep() vs wait()

sleep()

wait()

In Thread class

In Object class

Does not release lock

Releases lock

Time-based

Condition-based

1 9 join() vs sleep()

join()

sleep()

Wait for thread

Wait for time

join()

sleep()

Depends on thread Depends on time

2 0 yield() vs sleep()

yield()

sleep()

Suggests switch Forces pause

No time

Time-based

2 1 Common Interview Questions

- ✓ Difference between process and thread
 - ✓ start vs run
 - ✓ sleep vs wait
 - ✓ wait vs notify
 - ✓ What is deadlock
 - ✓ What is race condition
 - ✓ What is daemon thread
 - ✓ What happens if run() throws exception
 - ✓ Can we restart thread? (NO)
-

2 2 Can Thread Be Restarted?

NO.

Once thread reaches TERMINATED → cannot restart.

Calling start() again throws:

IllegalThreadStateException

2 3 Best Practices

- ✓ Always use Runnable over Thread
- ✓ Keep critical section small
- ✓ Avoid nested locks
- ✓ Use Executor framework in real projects
- ✓ Do not use stop(), suspend(), resume()

2 4 Modern Threading (Advanced Hint)

In real applications:

- ExecutorService
- Callable
- Future
- Thread pools

Better than creating threads manually.