Multithreading

---

1. Introduction to Multithreading

Definition: Multithreading is the capability of a CPU, or a single core in a multi-core processor, to execute multiple threads concurrently.

- Thread: A thread is a lightweight sub-process; it's the smallest unit of processing.
- Multitasking: Executing multiple tasks at the same time.
  - Process-based: Each process has its own memory.
  - Thread-based: Multiple threads within the same process share memory.
- Why Multithreading:
  - Efficient CPU utilization
  - Simultaneous tasks like file download + media playback
  - Better performance in I/O operations

🧠 Why is it Used?

Multithreading is used to:

✅ Improve CPU utilization
✅ Perform tasks in parallel (e.g., downloading files while updating UI)
✅ Enhance performance in multicore systems
✅ Prevent UI freezing in GUI applications
✅ Handle concurrent users/requests in server applications

🔧 How Java Supports Multithreading:

Java provides rich support for multithreading via:

1. `Thread` **class**
2. `Runnable` **interface**
3. **Executors / ThreadPool**
4. **Synchronization and Inter-thread communication**

2. Creating Threads in Java

Java provides two main ways to create threads:

a) Extending `Thread` class

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running");
    }
}
public class Demo {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start(); // starts the thread and calls run()
    }

}
```

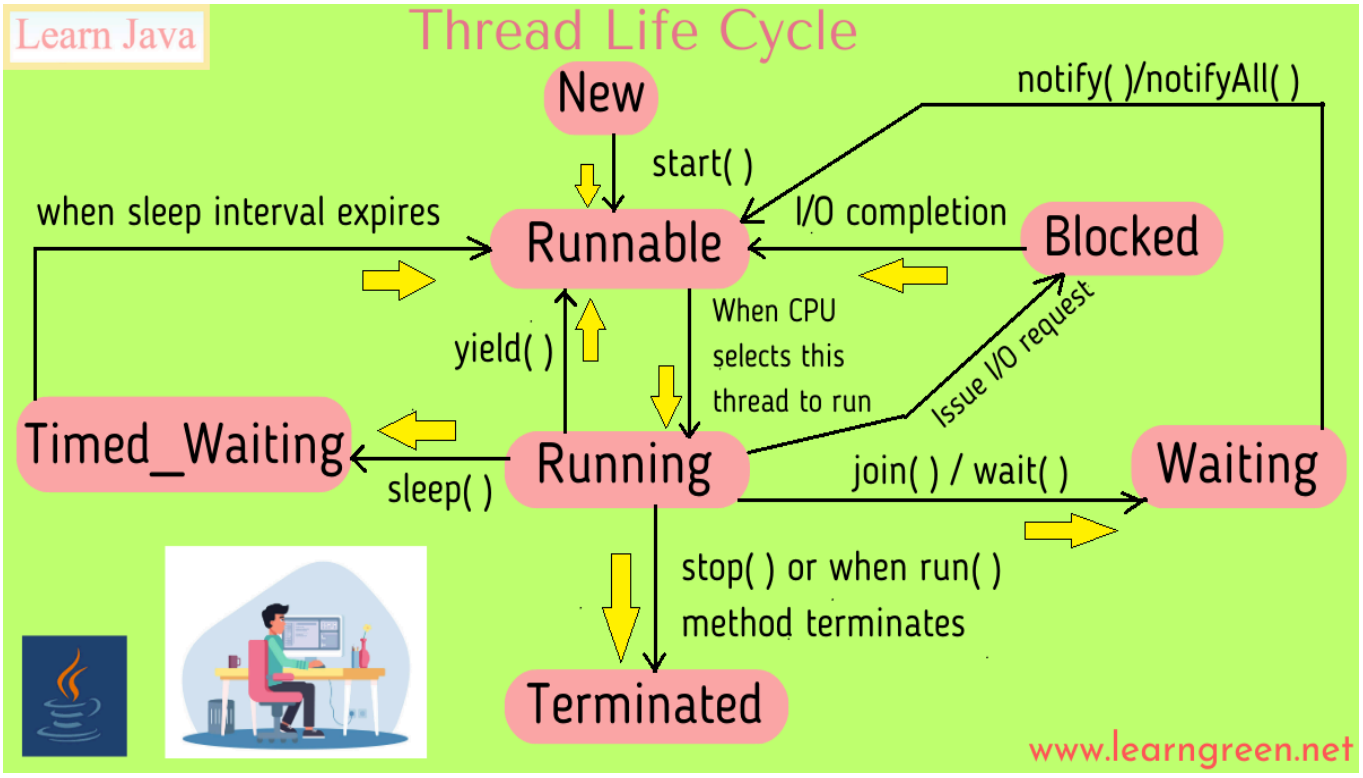
b) Implementing `Runnable` interface

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

3. Thread Lifecycle (States)

Java threads go through the following states:

1. **New**
2. **Runnable**
3. **Running**
4. **Blocked/Waiting**
5. **Terminated**



Use methods like `start()`, `sleep()`, `join()`, `wait()`, `notify()`, `stop()` (deprecated) to control the states.

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

    public static void main(String[] args) throws InterruptedException
{
        LifecycleDemo t = new LifecycleDemo();
        System.out.println("State before start: " + t.getState());
        t.start();
        System.out.println("State after start: " + t.getState());
        t.join();
        System.out.println("State after finish: " + t.getState());
    }
}
```

4. Thread Methods

Commonly used methods:

Method	Description
`start()`	Starts the thread
`run()`	Entry point of thread logic
`sleep(ms)`	Suspends thread temporarily
`join()`	Waits for a thread to finish execution
`isAlive()`	Checks if thread is still alive
`setPriority()`	Sets thread priority (1–10)
`getPriority()`	Gets thread priority
`yield()`	Pauses current thread for others

```
class ThreadMethods extends Thread {
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(Thread.currentThread().getName() + " " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {}
        }
    }

    public static void main(String[] args) {
        ThreadMethods t1 = new ThreadMethods();
        ThreadMethods t2 = new ThreadMethods();
        t1.setName("Thread-1");
        t2.setName("Thread-2");
        t1.start();
        t2.start();
    }
}
```

5. Thread Priorities

- Java uses integer values between 1 and 10.
  - Thread.MIN_PRIORITY = 1
  - Thread.NORM_PRIORITY = 5
  - Thread.MAX_PRIORITY = 10
- Schedulers may consider priority, but not guaranteed.

```
public class PriorityExample extends Thread {
  public void run() {
    System.out.println(getName() + " Priority: " + getPriority());
  }

  public static void main(String[] args) {
    PriorityExample t1 = new PriorityExample();
    PriorityExample t2 = new PriorityExample();
    t1.setPriority(Thread.MIN_PRIORITY);
    t2.setPriority(Thread.MAX_PRIORITY);
    t1.start();
    t2.start();
  }
}
```

6. Daemon Threads

- Daemon threads are background service threads like garbage collection.
- Use `setDaemon(true)` **before** `start()`.

```
class DaemonExample extends Thread {
    public void run() {
        if (Thread.currentThread().isDaemon()) {
            System.out.println("Daemon thread running");
        } else {
            System.out.println("User thread running");
        }
    }

    public static void main(String[] args) {
        DaemonExample d = new DaemonExample();
        d.setDaemon(true);
        d.start();

        DaemonExample u = new DaemonExample();
        u.start();
    }
}
```