**Thread and multithreading**

By: Shashi Prasad

We can create a thread in java **3 main ways:**

By extending the Thread class

By implementing the Runnable interface.

By implementing the Callable interface (with ExecutorService).

**What us Runnable Interface?**

The **Runnable interface** in Java is one of the simplest and most important interfaces used for creating threads or defining tasks that can be executed by a thread. It’s a **functional interface**, meaning it has only **one abstract method** → run(). It is present in the java.lang package.

It represents a **task** that can be executed by a Thread or an executor service.

How to use :

class MyRunnable **implements Runnable** {

public void **run()** {

System.out.println("Thread running via Runnable interface");

}

public static void main(String[] args) {

Thread **t1** = new Thread(**new MyRunnable()**);

Thread **t2** = new Thread(new MyRunnable());

**t1**.start(); // starts a new thread that calls run()

**t2**.start(); // starts a new thread that calls run()

}

}

Using Lambda Expression (Java 8+)

public class RunnableLambda {

public static void main(String[] args) {

**Runnable task = () -> System.out.println("Running with lambda!");**

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

T1.start();

T2.start();

}

}

Runnable the foundation of multithreading in Java.

**What is callable?**

The Callable interface represents a task that can run in a separate thread, return a result, and throw exceptions — used with concurrency utilities like ExecutorService and Future. The **Callable** interface in Java is part of the **java.util.concurrent** package You **cannot** directly pass a Callable to a Thread. Instead, use an **ExecutorService** to run it.

**What is Thread class?**

Thread is part of the **java.lang** package. It implements **Runnable** interface. Java 21, Thread supports both **platform threads** (the “classic” threads tied to OS-level threads) and **virtual threads** (lightweight threads introduced in recent Java versions). Here A virtual thread is also an instance of Thread, but it is not permanently tied to a specific OS thread.

**Key Responsibilities of Thread class.**

Start a thread,

Define what to execute by Override run() method.

Control execution by using sleep(), join(), yield(), and interrupt()

Thread identity by using getName(), getId(), setName()

Thread priority by using getPriority(), setPriority()

Daemon threads by using setDaemon(true) makes background threads that stop when main thread ends

Thread state Tracks states via Thread.State (NEW, RUNNABLE, BLOCKED, etc.)

Synchronization Works with synchronized blocks to ensure thread safety

**When to use thread class and when to use runnable interface?**

The choice between Thread class and Runnable interface depends on your design needs

1 - If want to stop inheritance on class then Must extend Thread class, so you can’t extend any other class other wise use Runnable because it allow to extend other classes as well.

2 – Reusability, when we extend thread class and implements task as run method then Task logic and thread control are mixed together. Which may create reusability issue for the class. BUT when we create thread using Runnable interface Task logic and thread control are separatedso it maintains class reusable**.**

3- Less flexible (tight coupling) when we extend thread class. But with Runnable it is More flexible (loose coupling)

4 Use Thread class only when: **You need to override Thread methods** - If you want to customize methods like interrupt(), join(), or other Thread-specific behaviour

5- **Use Runnable in most cases** because its provide Inheritance flexibility, **Better design** - Separates the task from the thread mechanism, **Thread pool compatibility** - Works seamlessly with ExecutorService and thread pools, Same Runnable can be passed to multiple threads, Doesn't carry unnecessary Thread class overhead.

*Go Default to Runnable interface unless you have a specific reason to extend Thread.*

END