Thread -

- A **thread** is the **smallest unit of execution** in a program.
- A Java program can contain **multiple threads running concurrently**, which is called **multithreading**.
- Threads **share the same memory (heap)** but have **individual stacks, program counters, and local variables**.

Characteristics of Threads:

1. **Lightweight**: Less resource usage compared to full processes.
2. **Independent execution**: Each thread executes its own task.
3. **Shared memory**: Threads can communicate easily via shared variables but may require synchronization.
4. **Concurrent execution**: Threads can run in parallel or be interleaved on a single CPU.

Thread Functions

Method	Description
start()	Starts a new thread; JVM calls the run() method.
run()	Contains the thread's code; called by JVM when thread starts.
currentThread()	Returns a reference to the **currently executing thread**.
setName(String name)	Sets the name of the thread.
getName()	Returns the name of the thread.
getId()	Returns the **unique thread ID** assigned by JVM.

```
class MyThread extends Thread{
  @Override
  public void run() {

    for (int i = 0; i < 4; i++) {
      System.out.println(Thread.currentThread().getName() + " is running " + i);
```

```
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
  }
}
public class threadClass {
  public static void main(String[] args) {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();

    t1.start();
    t2.start();
  }
}
```

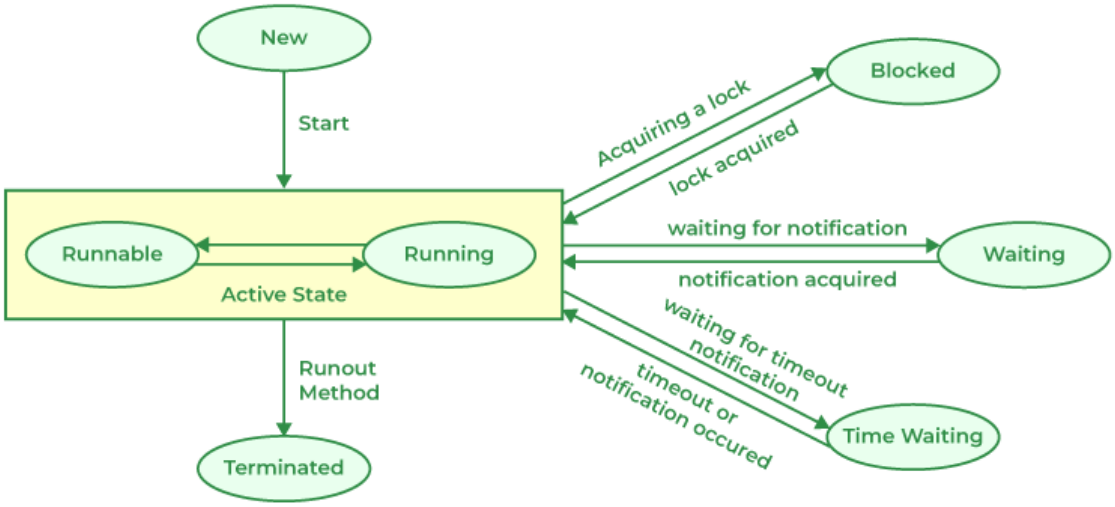
Thread Model - The **thread model** explains how Java **threads are created, executed, managed, and terminated**.

Thread States: A thread can transition through various states during its lifecycle, including:

- **New:** The thread has been created but not yet started.
- **Runnable:** The thread is ready to run and waiting for the CPU to allocate time for execution.
- **Running:** The thread is currently executing. JVM picks the thread and run() is executing.
- **Blocked/Waiting/Timed Waiting:** The thread is temporarily inactive, waiting for a resource, a notification, or a specific time period to elapse.
- **Terminated:** The thread has completed its execution.

Thread Lifecycle Functions

Method	Description
sleep(long millis)	Pauses the thread for specified milliseconds (static method).
yield()	Causes the currently executing thread to **temporarily pause**, allowing other threads to execute.
join()	Waits for a thread to **finish execution** before continuing the calling thread.
join(long millis)	Waits at most the specified time for a thread to finish.
isAlive()	Checks if the thread is still running.
interrupt()	Interrupts a thread that is in sleeping, waiting, or blocked state.
isInterrupted()	Tests whether the thread has been interrupted.



Thread Scheduling - VM uses **thread scheduling** to decide which thread runs first.

- Scheduling depends on **OS thread scheduler** (preemptive or time-sliced).

Priority levels:

- Thread.MIN_PRIORITY = 9
- Thread.NORM_PRIORITY = 5 (default)
- Thread.MAX_PRIORITY = 0

High priority does **not guarantee** earlier execution but **increases chances**.

Runnable Interface

- **Runnable** is a **functional interface** in java.lang.
- It **represents a task to be executed by a thread**.
- Contains **single abstract method** void run()
- It Defines **task logic separately** from thread control.
- It Allows a class to **implement other interfaces** while defining a thread task.
- Multiple threads can **share the same Runnable object**, enabling task sharing.
- Makes code **more flexible, reusable, and modular**.