**Assignment: (Core Java)**

**13 : Multithreading**

**Que.1 Introduction to Threads**

**Ans.1** A Thread in Java is the smallest unit of execution in a program.

* By default, every Java program runs with a main thread (the one that starts when you call public static void main).
* Threads allow multitasking (running multiple parts of a program at the same time).

Example: A video player can play video + play audio + download subtitles simultaneously using threads.

**Que.2 Creating Threads by Extending Thread Class or Implementing Runnable Interface**

**Ans.2 1.**Create a class that **extends Thread**.

2.Override the run() method (code that thread will execute).

3.Create an object of your class.

4.Call start() (not run() directly).

Example:

class MyThread extends Thread {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println("Thread running: " + i);

try { Thread.sleep(500); } catch (InterruptedException e) {}

}

}

}

public class ThreadDemo {

public static void main(String[] args) {

MyThread t1 = new MyThread(); // create thread object

t1.start(); // start thread, internally calls run()

}

}

**Que.3 Thread Life Cycle**

**Ans.3 1.New (Created) :**A thread is in the new state when you create it using new Thread() but haven’t started it yet with start().

2.**Runnable :** After calling start(), the thread enters the Runnable state. It is ready to run but waiting for CPU scheduling.

**3.Running:** When the CPU scheduler picks the thread, it moves from Runnable → Running.

* At any time, only one thread per core can be in the running state.
* Code inside run() executes here.

**4.Waiting / Timed Waiting / Blocked:**

* **Waiting:** A thread is waiting indefinitely for another thread to notify it (wait()).
* **Timed Waiting:** A thread waits for a specific time (sleep(ms), join(ms), wait(ms)).
* **Blocked:** A thread is waiting to acquire a lock

**5.Terminated (Dead)**

* When the run() method finishes, the thread enters the **Terminated state**.
* Once dead, a thread **cannot be restarted**.

**Que.4 Synchronization and Inter-thread Communication**

**Ans.4** Synchronization and inter-thread communication are crucial concepts in Java multithreading, ensuring data consistency and coordinated execution among multiple threads.

Synchronization in Java controls access to shared resources by multiple threads, preventing data corruption and race conditions. It ensures that only one thread can access a critical section at a time.

Inter-thread communication allows synchronized threads to communicate and coordinate their activities. Java provides wait(), notify(), and notifyAll() methods for this purpose. These methods must be called within a synchronized block or method.