UE20CS352 – OOAD with Java Lab Assignment – 8

Multi-Threading in Java

Name:- Tushar J

SRN:- PES1UG20CS472

Section:- H

Rollno:- 10

Problem Description:

Write a Java program that simulates a race between multiple runners. Each runner should be represented as a separate thread, and the program should output the current distance each runner has covered after each second. The distance each runner covers in each second should be determined randomly. The program should stop once one of the runners reaches a distance of 1000 meters. The program should then print the top 3 runners of the race. Requirements:

1. The program should read input as to how many runners are running the race.

2. The program should create a separate thread for each runner (optionally add names for each thread to represent the runner).

3. Each thread should print the total distance run so far by the runner after each second. The distance each runner covers in each second should be determined randomly (approx. between 5 – 10 m).

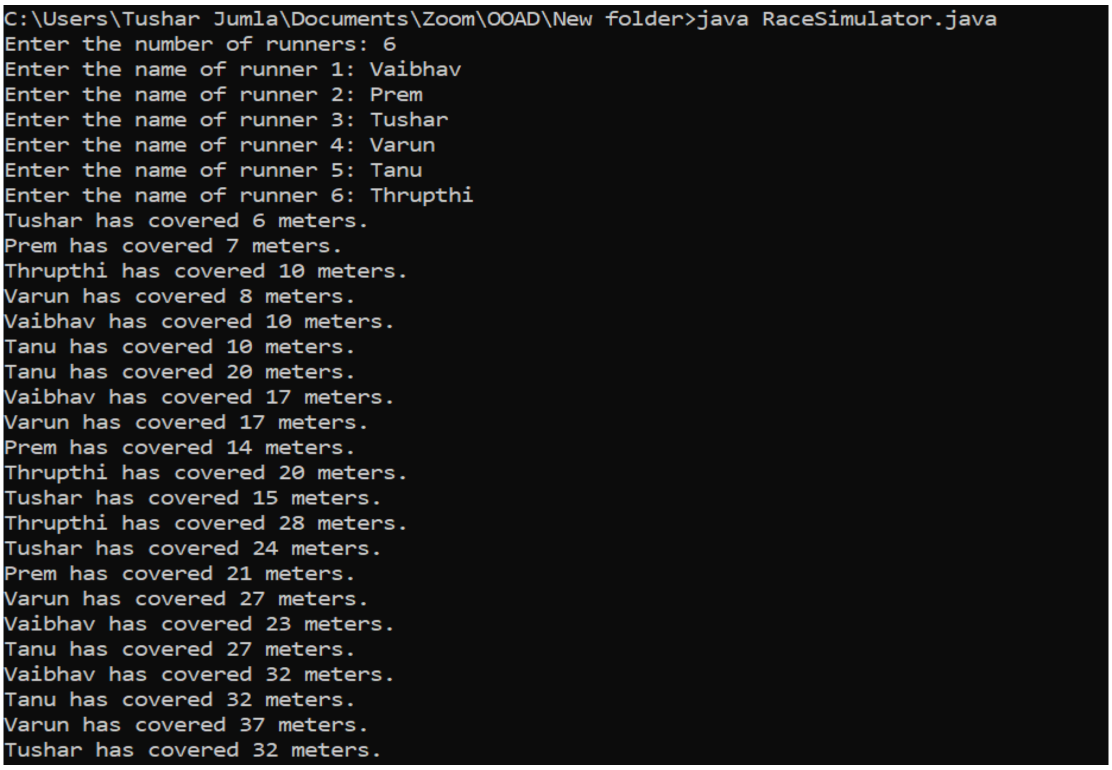
4. The program should stop once one of the runners reaches a distance of 1000 meters. The program should then print the top 3 runners of the race.

5. Execute the code at least 3 times with different number of runners.

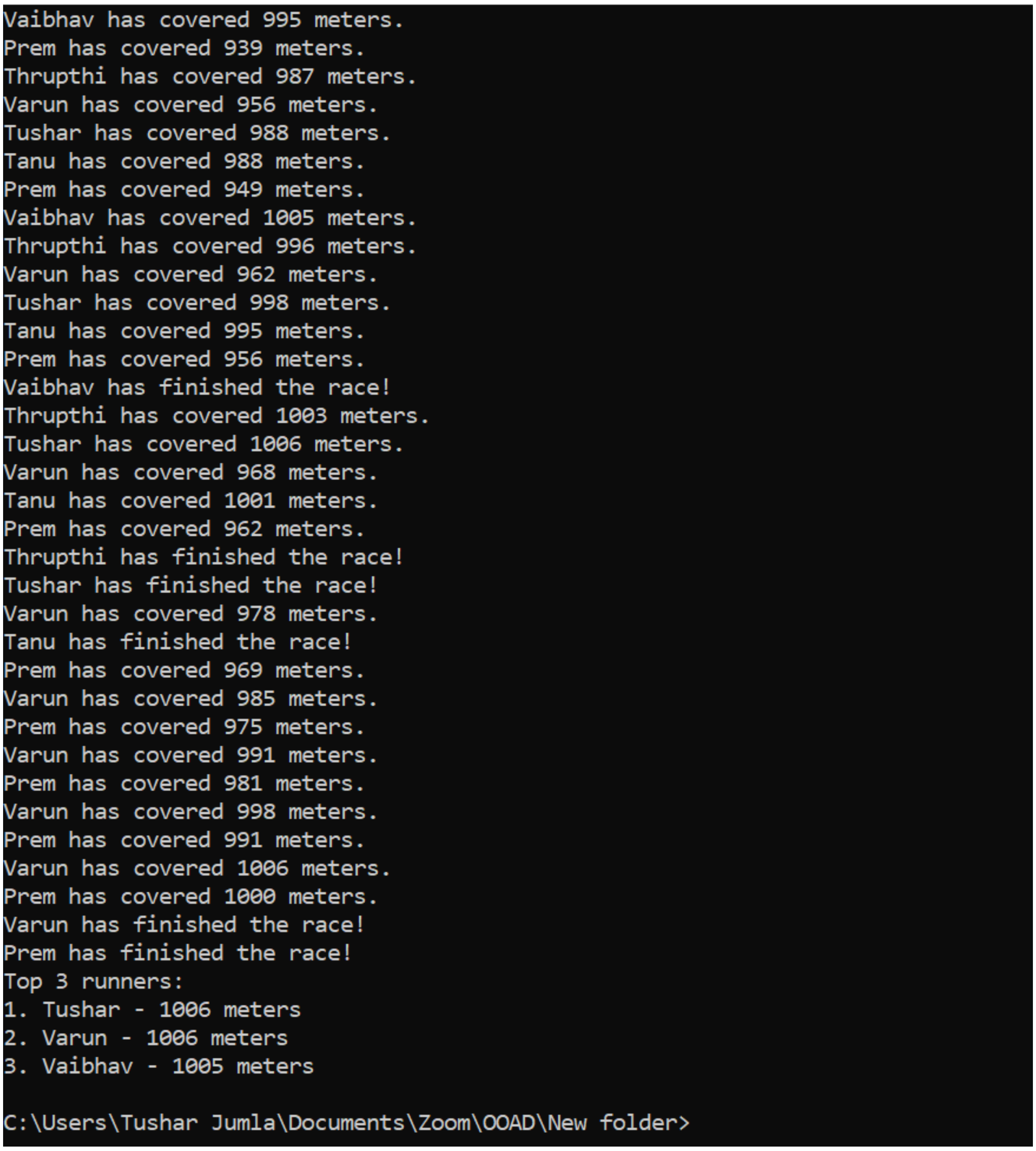
Code:-

```java
import java.util.Random;
import java.util.Scanner;

public class RaceSimulator implements Runnable {

    private static final int TARGET_DISTANCE = 1000;
    private static final int MIN_DISTANCE = 5;
    private static final int MAX_DISTANCE = 10;

    private final String name;
    private int distanceCovered;

    public RaceSimulator(String name) {
        this.name = name;
        this.distanceCovered = 0;
    }

    public void run() {
        Random random = new Random();

        while (distanceCovered < TARGET_DISTANCE) {
            int distanceThisSecond = MIN_DISTANCE +
random.nextInt(MAX_DISTANCE - MIN_DISTANCE + 1);
            distanceCovered += distanceThisSecond;
            System.out.println(name + " has covered " + distanceCovered + "
meters.");

            try {
                Thread.sleep(1000); // pause for 1 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println(name + " has finished the race!");
    }

    public int getDistanceCovered() {
        return distanceCovered;
    }

    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of runners: ");
        int numRunners = sc.nextInt();
        sc.nextLine(); // consume newline character

        String[] runnerNames = new String[numRunners];
        for (int i = 0; i < numRunners; i++) {
            System.out.print("Enter the name of runner " + (i + 1) + ": ");
            runnerNames[i] = sc.nextLine();
        }

        RaceSimulator[] runners = new RaceSimulator[numRunners];
        Thread[] threads = new Thread[numRunners];

        for (int i = 0; i < numRunners; i++) {
            runners[i] = new RaceSimulator(runnerNames[i]);
            threads[i] = new Thread(runners[i]);
            threads[i].start();
        }
        for (Thread thread : threads) {
            thread.join(); // wait for all threads to finish
        }

        // sort runners by distance covered
        for (int i = 0; i < numRunners - 1; i++) {
            for (int j = i + 1; j < numRunners; j++) {
                if (runners[i].getDistanceCovered() <
runners[j].getDistanceCovered()) {
                    RaceSimulator temp = runners[i];
                    runners[i] = runners[j];
                    runners[j] = temp;
                }
            }
        }

        System.out.println("Top 3 runners:");
        for (int i = 0; i < 3 && i < numRunners; i++) {
            System.out.println((i + 1) + ". " + runners[i].name + " - " +
runners[i].distanceCovered + " meters");
        }
    }
}
```

Output:-



```
C:\Users\Tushar Jumla\Documents\Zoom\OOAD\New folder>java RaceSimulator.java
Enter the number of runners: 6
Enter the name of runner 1: Vaibhav
Enter the name of runner 2: Prem
Enter the name of runner 3: Tushar
Enter the name of runner 4: Varun
Enter the name of runner 5: Tanu
Enter the name of runner 6: Thrupthi
Tushar has covered 6 meters.
Prem has covered 7 meters.
Thrupthi has covered 10 meters.
Varun has covered 8 meters.
Vaibhav has covered 10 meters.
Tanu has covered 10 meters.
Tanu has covered 20 meters.
Vaibhav has covered 17 meters.
Varun has covered 17 meters.
Prem has covered 14 meters.
Thrupthi has covered 20 meters.
Tushar has covered 15 meters.
Thrupthi has covered 28 meters.
Tushar has covered 24 meters.
Prem has covered 21 meters.
Varun has covered 27 meters.
Vaibhav has covered 23 meters.
Tanu has covered 27 meters.
Vaibhav has covered 32 meters.
Tanu has covered 32 meters.
Varun has covered 37 meters.
Tushar has covered 32 meters.
```



```
Vaibhav has covered 995 meters.
Prem has covered 939 meters.
Thrupthi has covered 987 meters.
Varun has covered 956 meters.
Tushar has covered 988 meters.
Tanu has covered 988 meters.
Prem has covered 949 meters.
Vaibhav has covered 1005 meters.
Thrupthi has covered 996 meters.
Varun has covered 962 meters.
Tushar has covered 998 meters.
Tanu has covered 995 meters.
Prem has covered 956 meters.
Vaibhav has finished the race!
Thrupthi has covered 1003 meters.
Tushar has covered 1006 meters.
Varun has covered 968 meters.
Tanu has covered 1001 meters.
Prem has covered 962 meters.
Thrupthi has finished the race!
Tushar has finished the race!
Varun has covered 978 meters.
Tanu has finished the race!
Prem has covered 969 meters.
Varun has covered 985 meters.
Prem has covered 975 meters.
Varun has covered 991 meters.
Prem has covered 981 meters.
Varun has covered 998 meters.
Prem has covered 991 meters.
Varun has covered 1006 meters.
Prem has covered 1000 meters.
Varun has finished the race!
Prem has finished the race!
Top 3 runners:
1. Tushar - 1006 meters
2. Varun - 1006 meters
3. Vaibhav - 1005 meters

C:\Users\Tushar Jumla\Documents\Zoom\OOAD\New folder>
```

Multi Threading:-

Multithreading is a programming concept in which the application can create a small unit of tasks to execute in parallel. If you are working on a computer, it runs multiple applications and allocates processing power to them. A simple program runs in sequence and the code statements execute one by one. This is a single-threaded application. But, if the programming language supports creating multiple threads and passes them to the operating system to run in parallel, it’s called multithreading.

Java has great support for multithreaded applications. Java supports multithreading through Thread class. Java Thread allows us to create a lightweight process that executes some tasks. We can create multiple threads in our program and start them. Java runtime will take care of creating machine-level instructions and work with OS to execute them in parallel.

There are two types of threads in an application - **user thread** and **daemon thread**. When we start an application, the **main** is the first user thread created. We can create multiple user threads as well as daemon threads. When all the user threads are executed, JVM terminates the program.

==