**Object Oriented Analysis and Design using Java**

**Self-Learning: Multi-Threading in Java**

**Lab Week 8**

|  |  |  |  |
| --- | --- | --- | --- |
| Vanshika Goel | PES1UG20CS484 | Section H | Roll No 40 |

**Multithreading in Java** is the process of running multiple threads at the same time.

Java Multithreading is commonly utilised in games, animation, and other applications.

Java provides excellent support for multithreaded programmes. The Thread class in Java provides multithreading capabilities. We can use Java Thread to create a lightweight process that performs some tasks. In our programme, we can create and start multiple threads. The Java runtime will create machine-level instructions and work with the operating system to execute them in parallel.

In an application, there are two types of threads: user threads and daemon threads. When we start an application, the primary user thread is generated. We have the ability to create multiple user threads as well as daemon threads. JVM terminates the programme once all user threads have been executed.

Threads can be generated via two mechanisms:

1. Adding to the Thread class

2. Putting the Runnable Interface into Action

The Thread Class vs. the Runnable Interface

1. Because Java does not support multiple inheritance, if we extend the Thread class, our class cannot extend any other class. Yet, by implementing the Runnable interface, we can still extend other base classes.

2. We may accomplish basic thread functionality by extending the Thread class, which has methods like yield(), interrupt(), and so on that are not available via the Runnable interface.

3. Using runnable returns an object that can be shared by several threads.

**Advantages of Java Multithreading**

1) It does not stall the user because threads are autonomous and can do many actions concurrently.

2) You can do multiple processes concurrently, which saves time.

3) Because threads are autonomous, an exception in one thread has no effect on other threads.

**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**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

public class Race {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Number of Runners in this Race: ");

int n = sc.nextInt();

List<Runner> runners = new ArrayList<>();

for (int i = 0; i < n; i++) {

runners.add(new Runner("Runner " + (i+1)));

}

for (Runner runner : runners) {

runner.start();

}

try {

for (Runner runner : runners) {

runner.join();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("---------------------");

Collections.sort(runners);

System.out.println("Top 3 Runners:");

for (int i = 0; i < 3; i++) {

System.out.println((i+1)+". "+runners.get(i));

}

}

}

class Runner extends Thread implements Comparable<Runner> {

private String name;

private int distance;

private Random rand;

public Runner(String name) {

this.name = name;

this.distance = 0;

this.rand = new Random();

}

public void run() {

while (distance <= 1000) {

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

int distanceCovered = rand.nextInt(6) + 5;

distance += distanceCovered;

System.out.println(name + " ran " + distance + " m.");

}

}

public int compareTo(Runner other) {

return Integer.compare(other.distance, this.distance);

}

public String toString() {

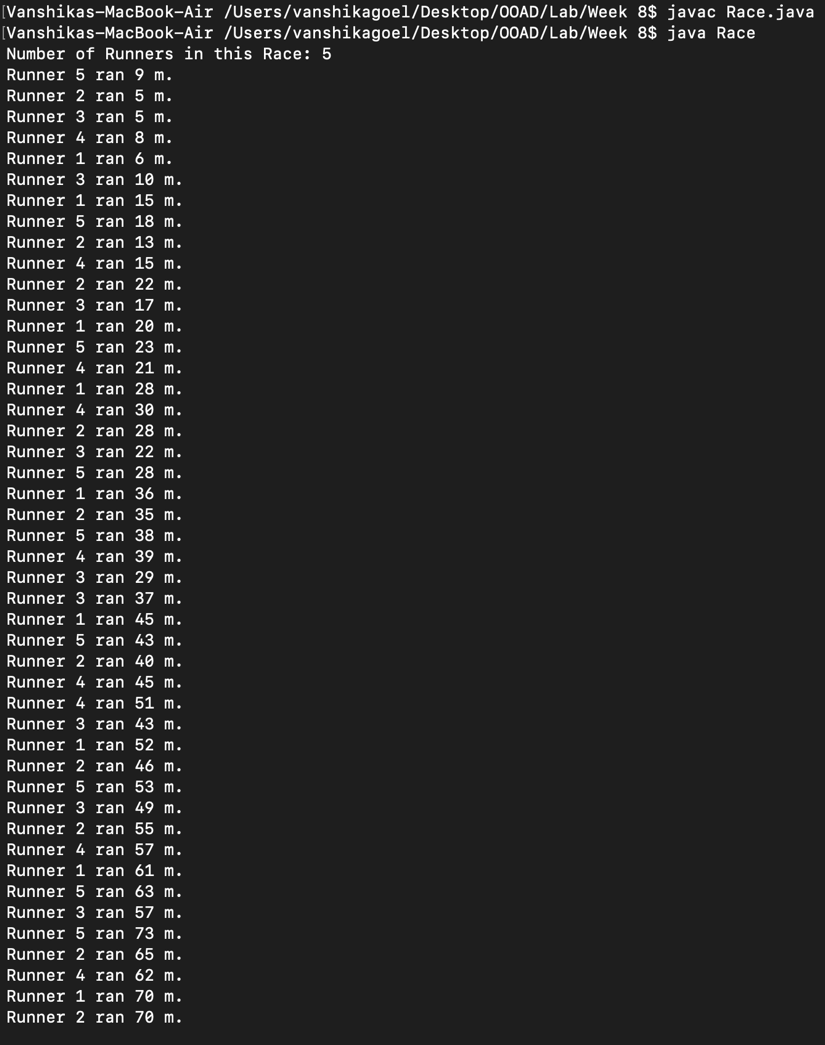
return name + " ran distance: " + distance;

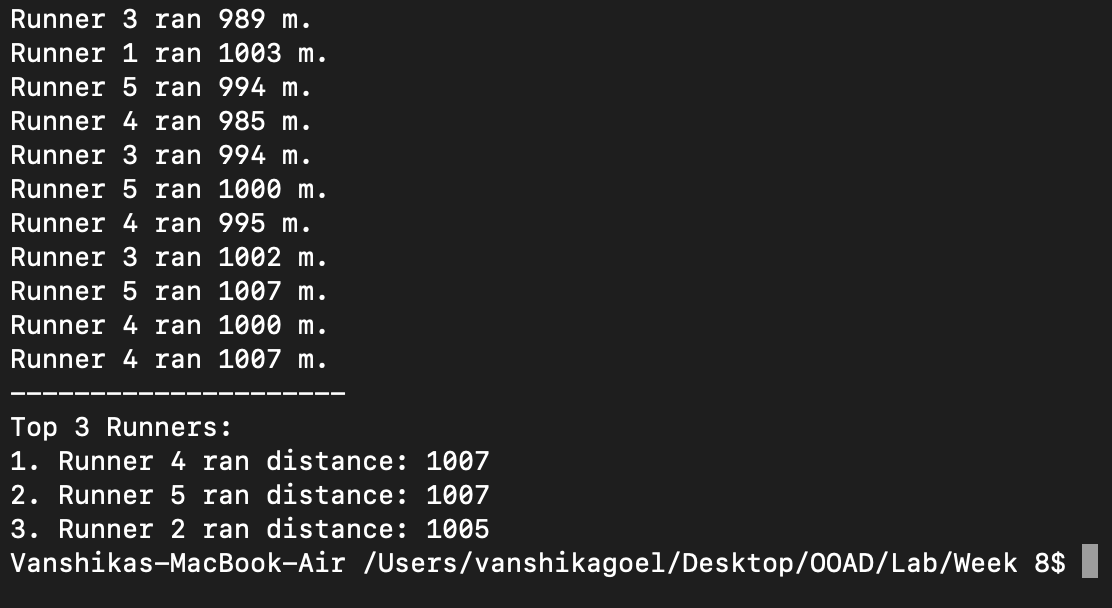
}

}

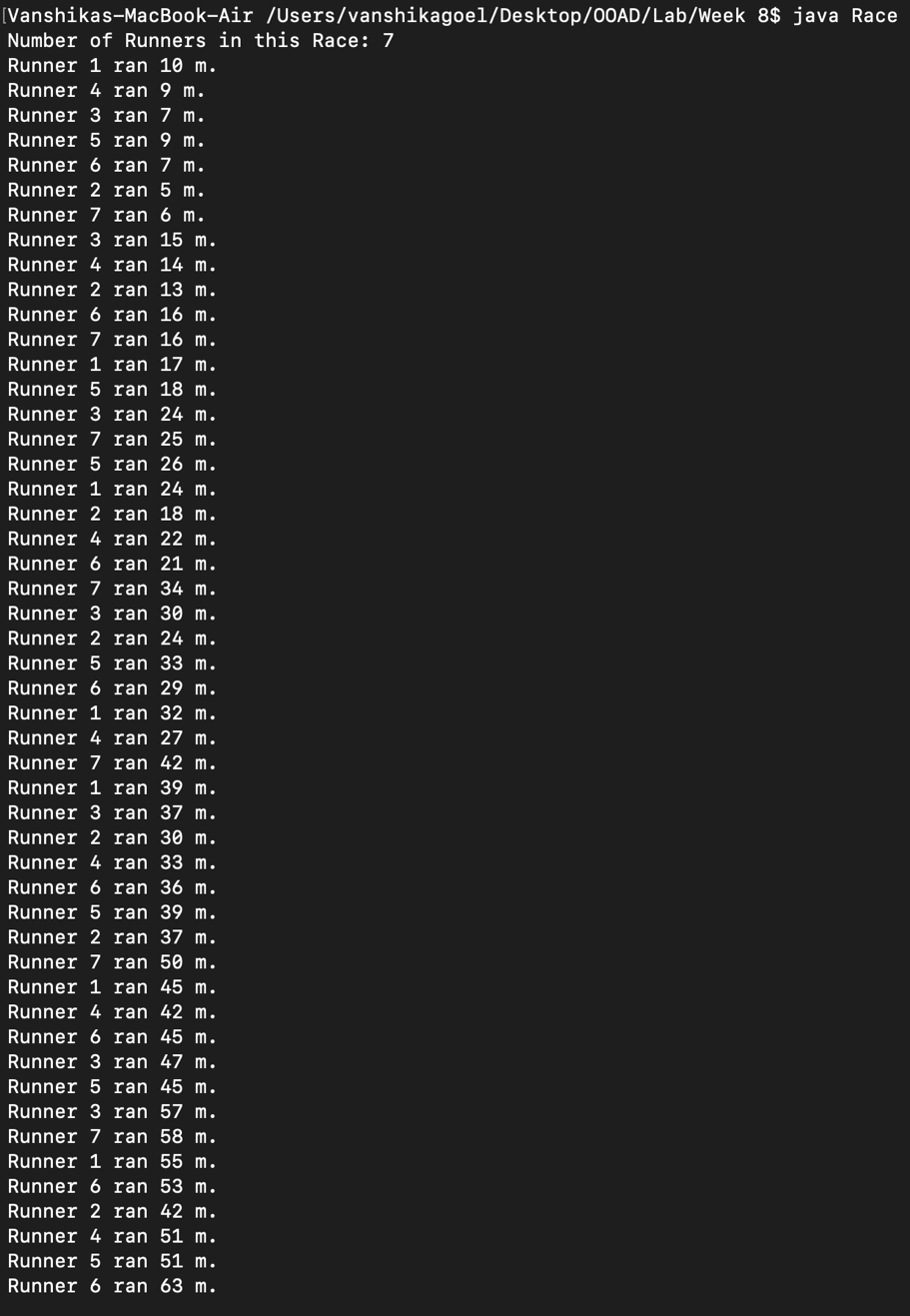
**Screenshots of Output:**

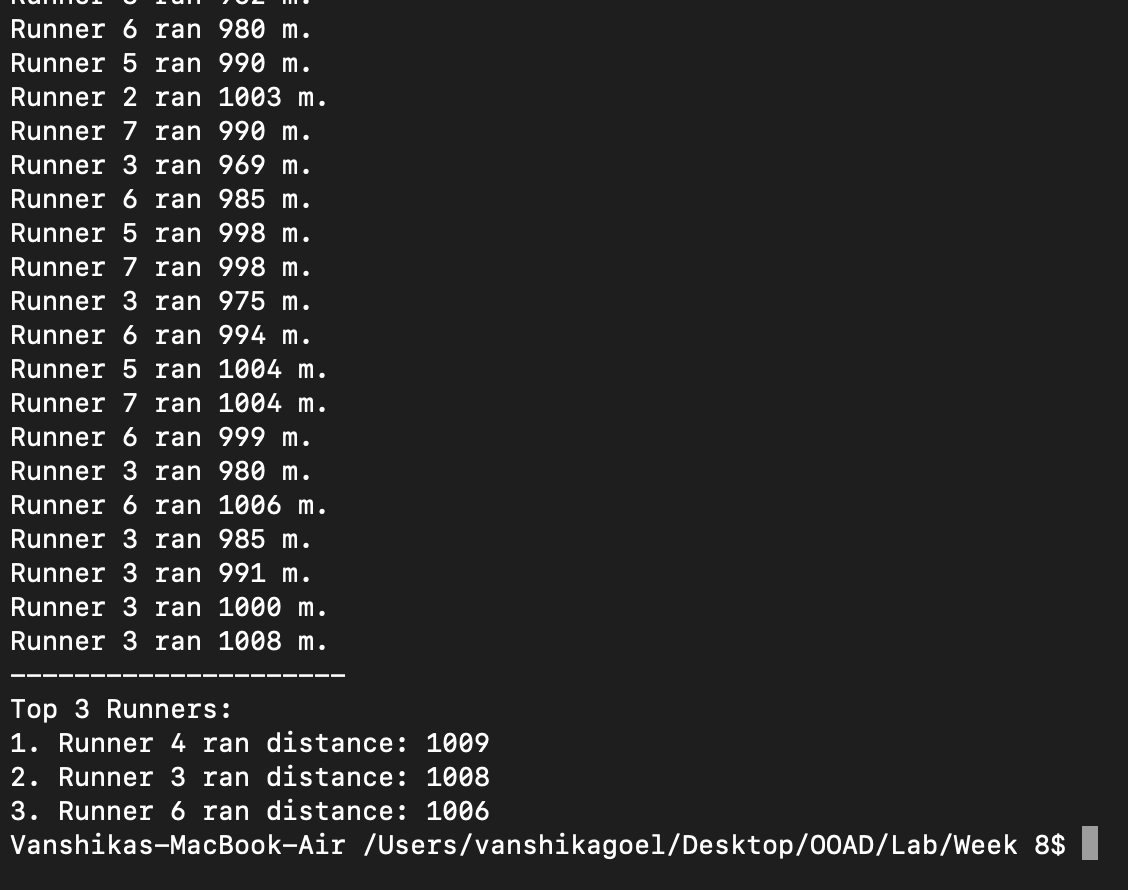
**For 5 runners in the race**



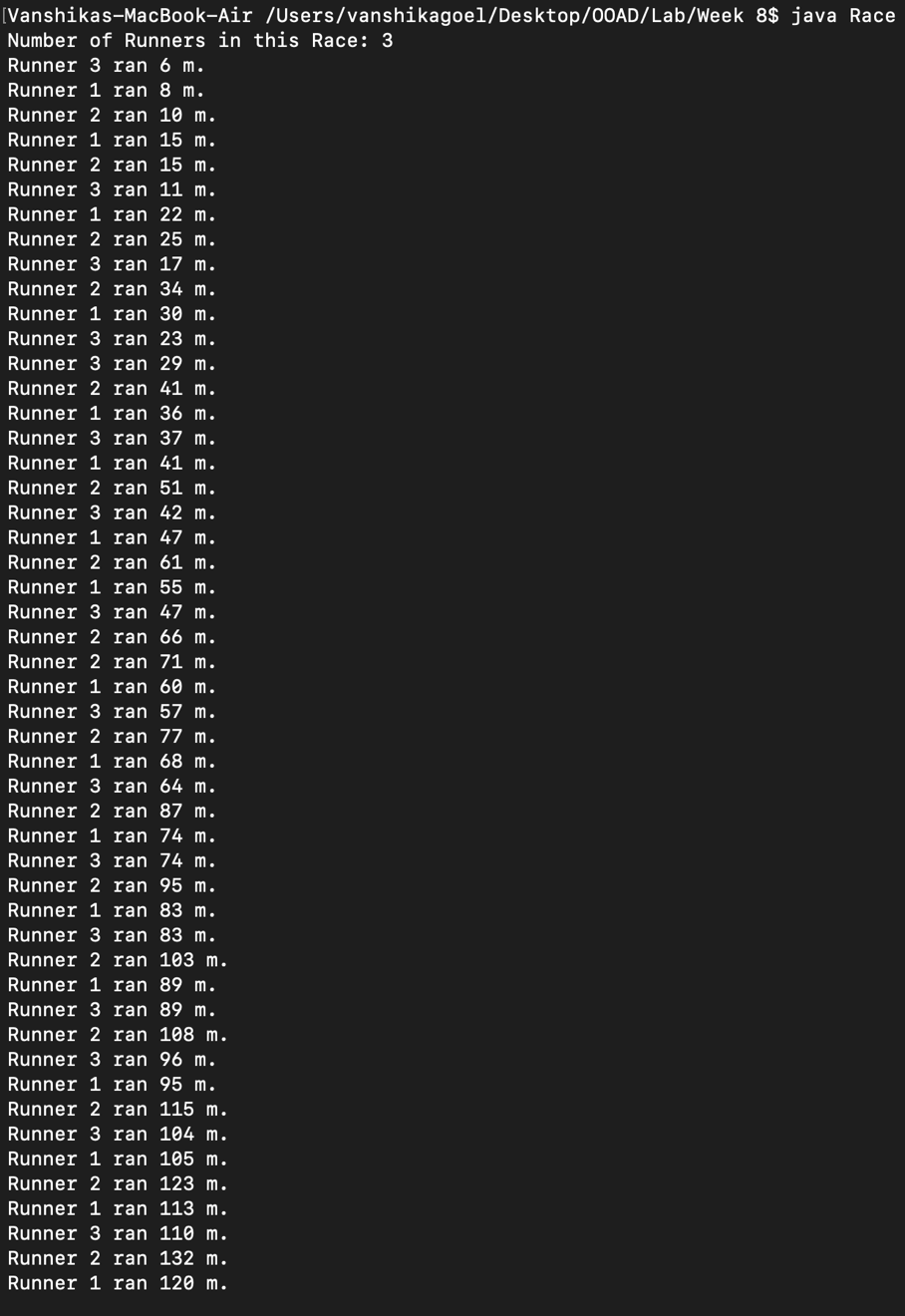


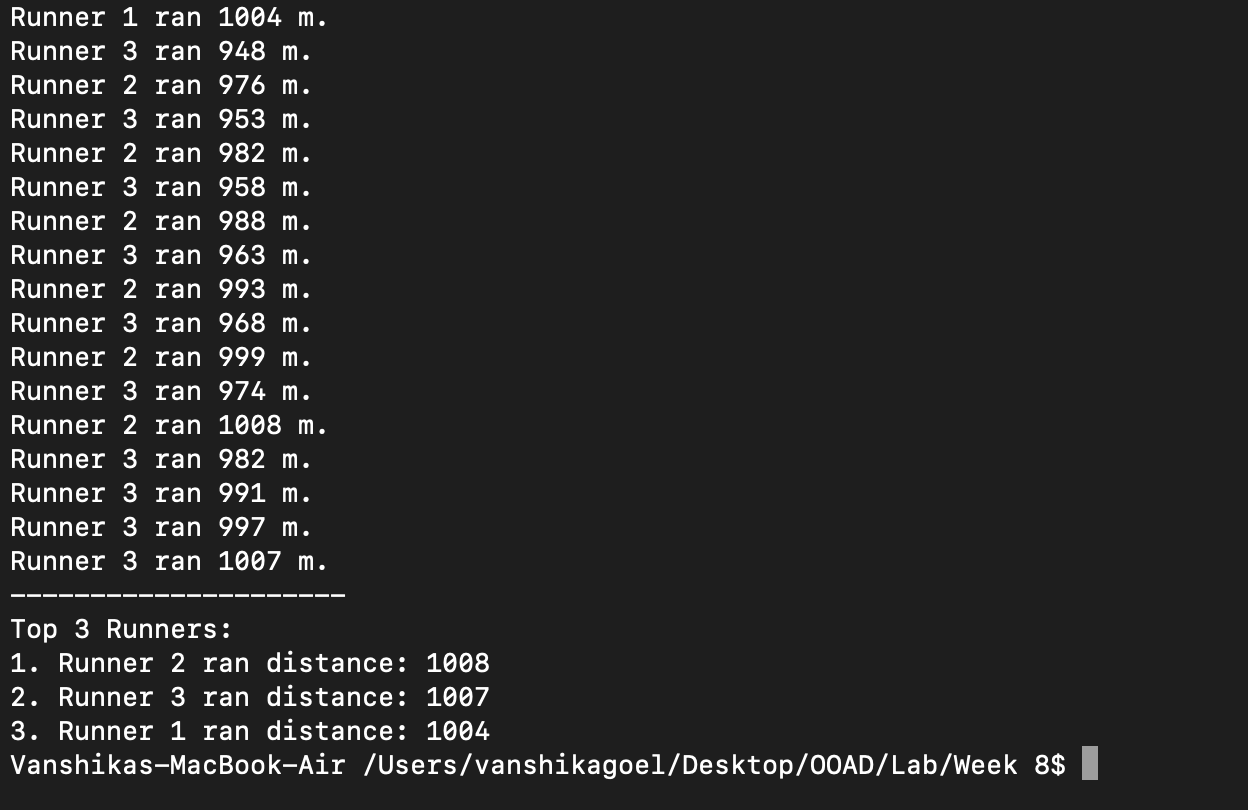
**For 7 Runners in the race:**

****

****

**For 3 Runners in the race:**

****

****