OOAD with Java Self Learning – 2:

Java Multithreading

Name: SAHANA RAO

SRN: PES1UG20CS588

Section: J

<u>Introduction to Multithreading:</u>

Multithreading is a programming concept in which the application can create a small unit of tasks to execute in parallel. If the programming language supports creating multiple threads and passes them to the operating system to run in parallel, it’s called multithreading.

In java, multithreading, just like its definition, allows the creation and management of multiple threads of execution within a single program, thereby providing the ability to perform multiple tasks simultaneously.

Java's multithreading capabilities are built into the language itself and allow for the creation of user and daemon threads. Multithreading is widely used for applications that require concurrency, such as web servers, gaming, and scientific simulations.

<u>Description of Threads/synchronized threads</u>

<u>Threads:</u> In multithreading in Java, a thread is a separate path of execution within a program i.e., it allows us to create a lightweight process that executes some tasks. Java has great support for multithreaded applications.

Threads can be used to perform multiple tasks simultaneously, allowing for improved performance and responsiveness in applications.

<u>Synchronized threads</u>: They are used to ensure that multiple threads access shared resources in a thread-safe manner. When multiple threads try to access the same shared resource concurrently, there is a risk of data inconsistency or corruption. To prevent this, synchronized threads use mutual exclusion, which means that only one thread can access a shared resource at a time.

If a method/code block is synchronized then only one method/code block can access it at a time respectively.

How to access them in Java?

Java supports multithreading through the *Thread class*. Java runtime will take care of creating machine-level instructions and work with the OS to execute them in parallel.

The *synchronized* keyword can be used to synchronize methods or code blocks.

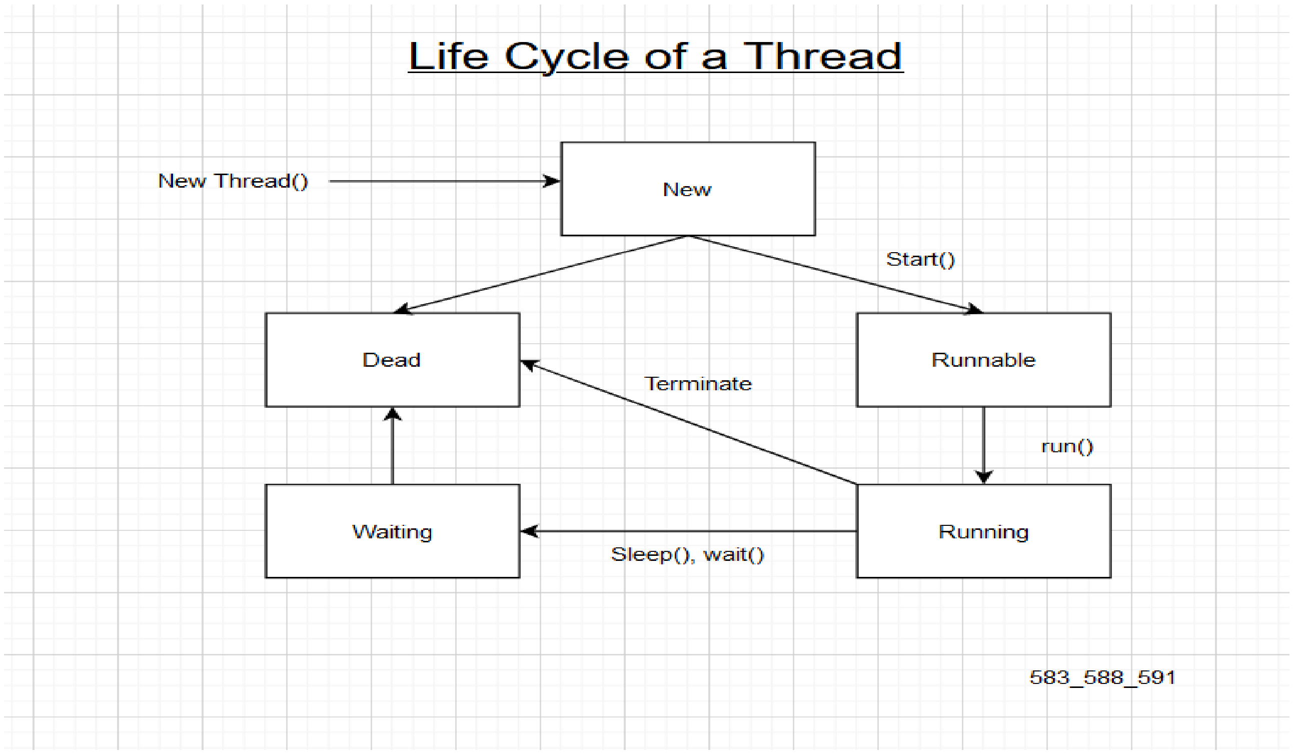
We can create multiple threads in our program and start them.

Types of threads:

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.

Life cycle of threads

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread:



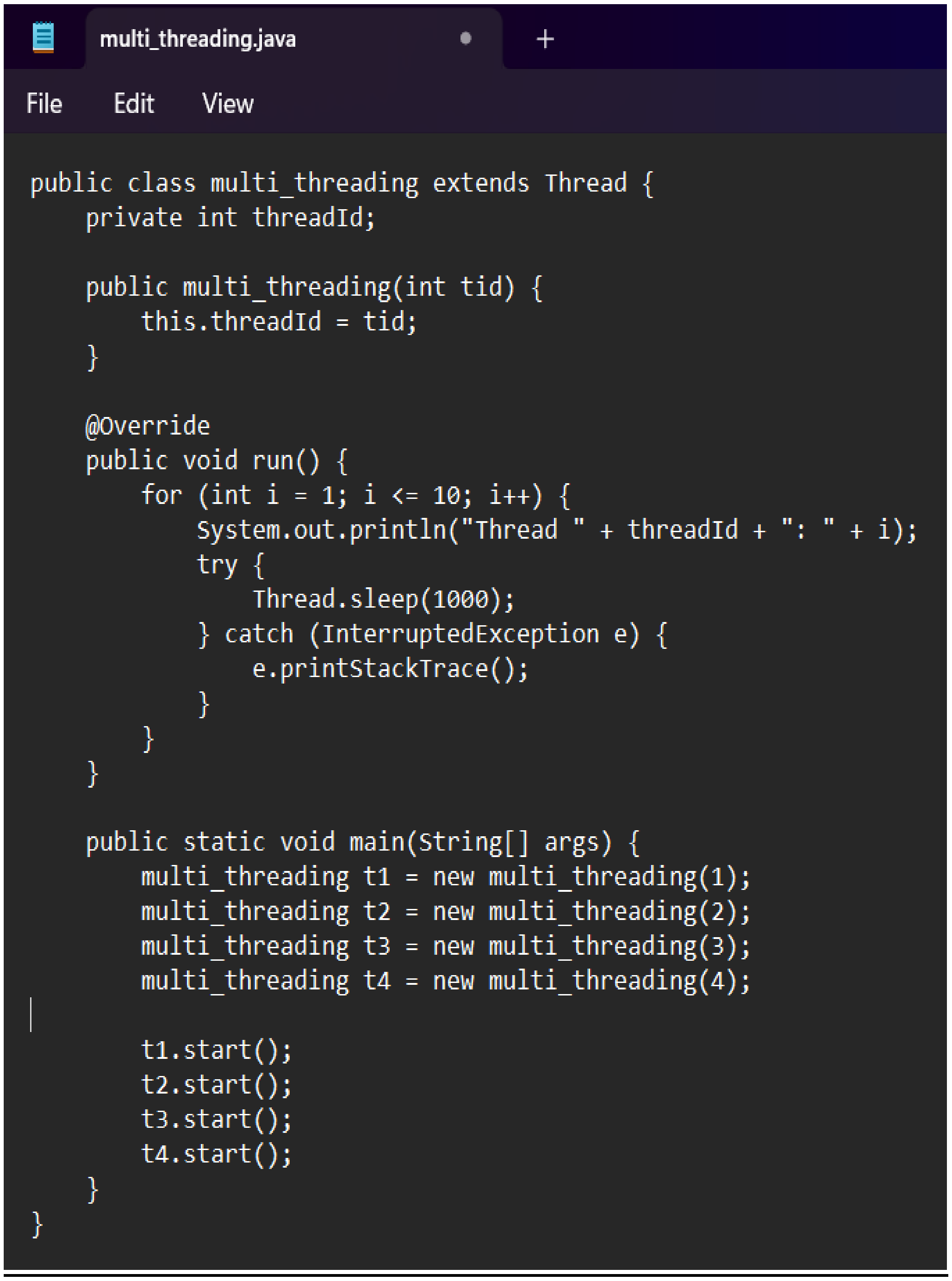
Different thread states

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting/Blocked** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.
- **Running:** In the running state, a thread is currently being executed by the CPU.

Example codes to understand multithreading

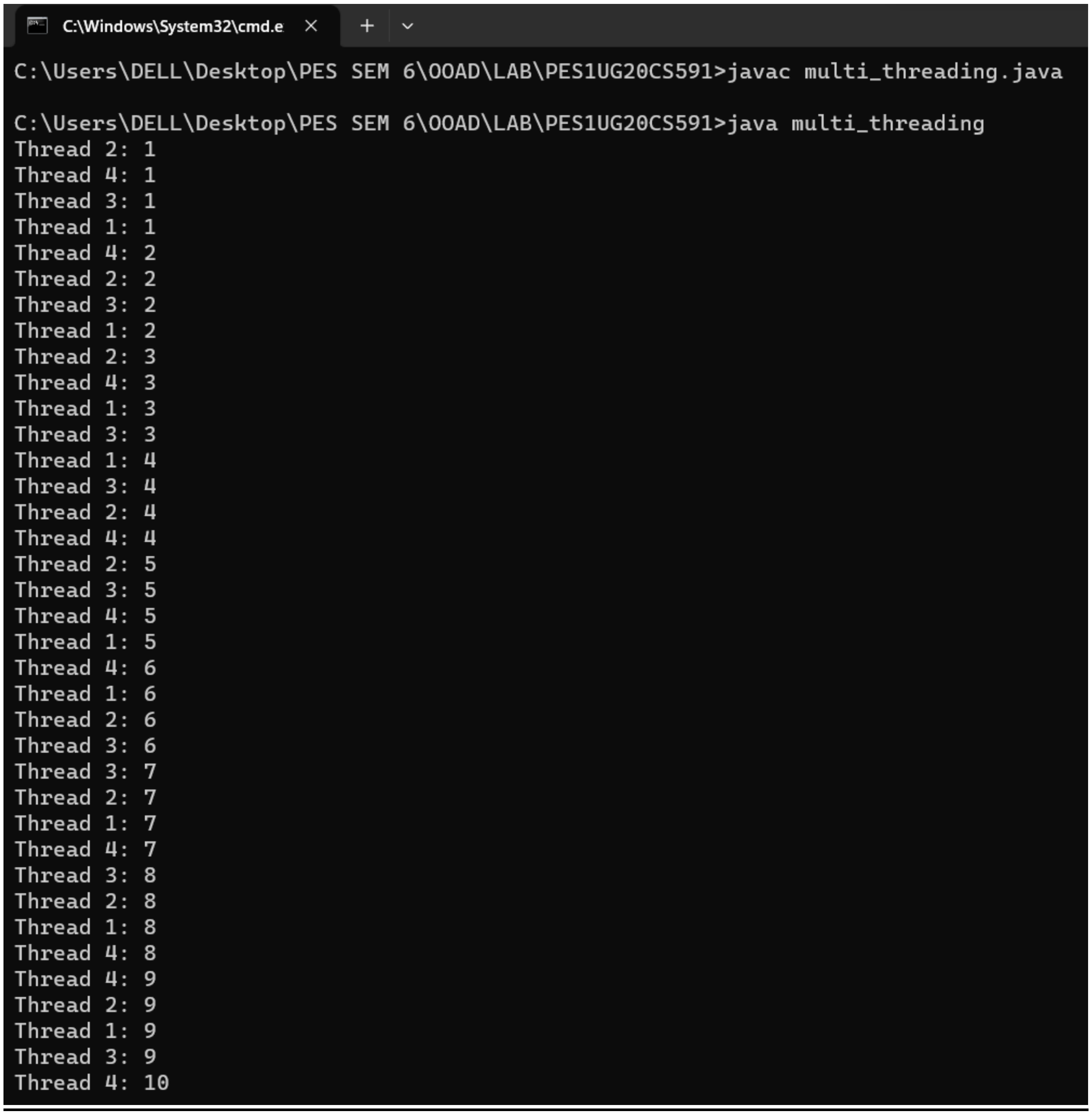
Code creates four threads that run concurrently and print a message to the console every second. Each thread has a unique ID that is passed to its constructor.

Code:

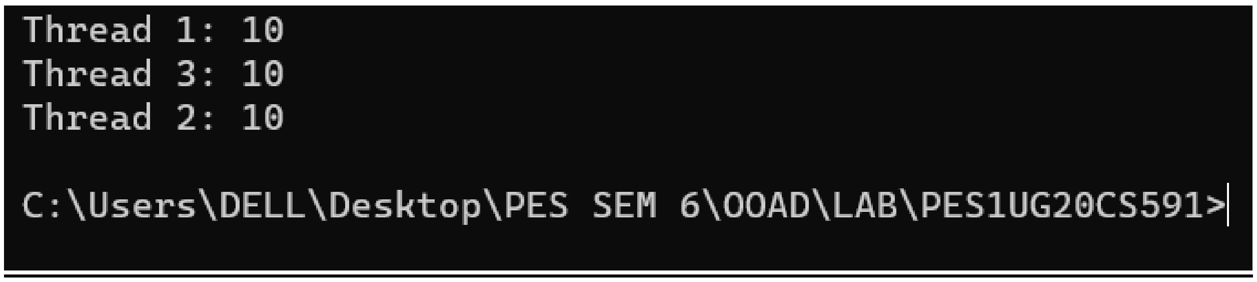


```java
public class multi_threading extends Thread {
    private int threadId;

    public multi_threading(int tid) {
        this.threadId = tid;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Thread " + threadId + ": " + i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        multi_threading t1 = new multi_threading(1);
        multi_threading t2 = new multi_threading(2);
        multi_threading t3 = new multi_threading(3);
        multi_threading t4 = new multi_threading(4);

        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

Result:



```
C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>javac multi_threading.java

C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>java multi_threading
Thread 2: 1
Thread 4: 1
Thread 3: 1
Thread 1: 1
Thread 4: 2
Thread 2: 2
Thread 3: 2
Thread 1: 2
Thread 2: 3
Thread 4: 3
Thread 1: 3
Thread 3: 3
Thread 1: 4
Thread 3: 4
Thread 2: 4
Thread 4: 4
Thread 2: 5
Thread 3: 5
Thread 4: 5
Thread 1: 5
Thread 4: 6
Thread 1: 6
Thread 2: 6
Thread 3: 6
Thread 3: 7
Thread 2: 7
Thread 1: 7
Thread 4: 7
Thread 3: 8
Thread 2: 8
Thread 1: 8
Thread 4: 8
Thread 4: 9
Thread 2: 9
Thread 1: 9
Thread 3: 9
Thread 4: 10
```



```
Thread 1: 10
Thread 3: 10
Thread 2: 10

C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>
```

Problem statement

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

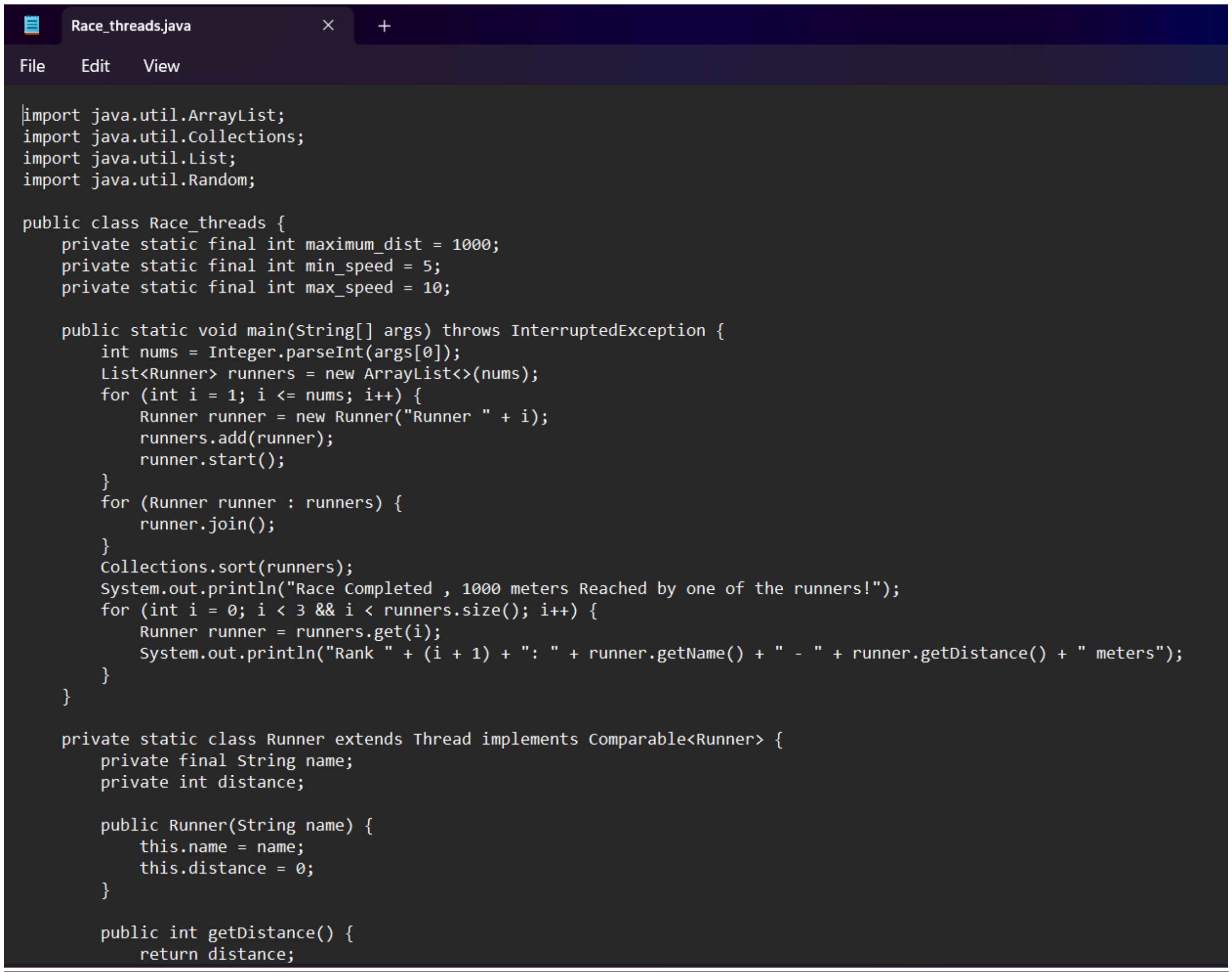
randomly (approx. between 5 – 10 m).

4. The program should stop once one of the runners reaches a distance of 1000 meters.

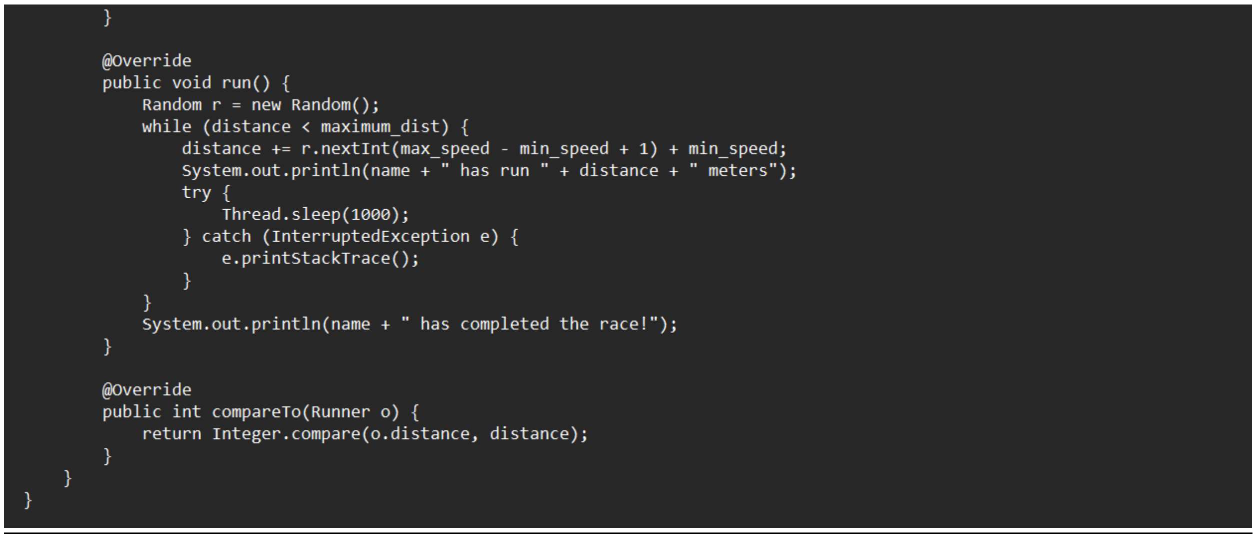
The program should then print the top 3 runners of the race.

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

Solution Code



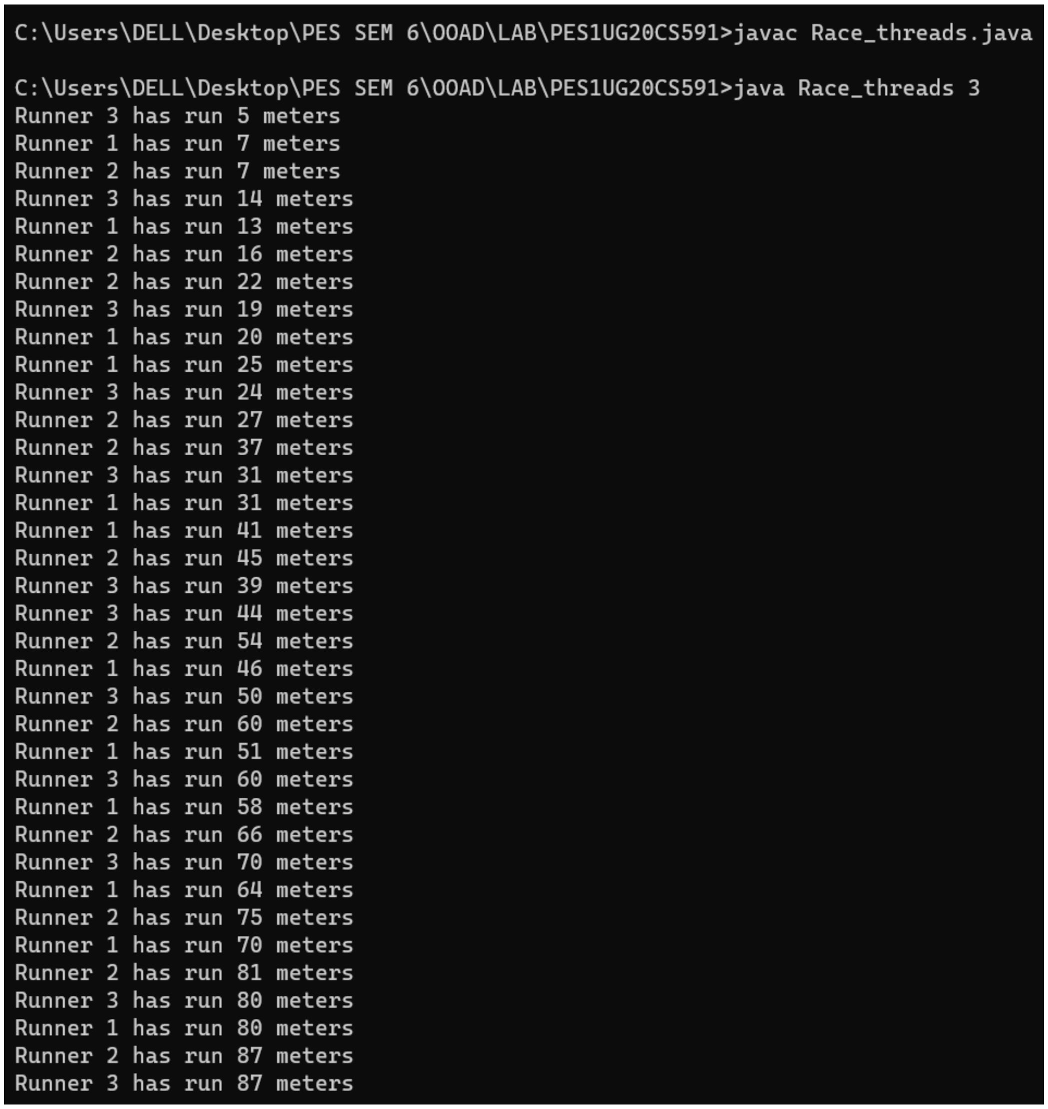
```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class Race_threads {
    private static final int maximum_dist = 1000;
    private static final int min_speed = 5;
    private static final int max_speed = 10;

    public static void main(String[] args) throws InterruptedException {
        int nums = Integer.parseInt(args[0]);
        List<Runner> runners = new ArrayList<>(nums);
        for (int i = 1; i <= nums; i++) {
            Runner runner = new Runner("Runner " + i);
            runners.add(runner);
            runner.start();
        }
        for (Runner runner : runners) {
            runner.join();
        }
        Collections.sort(runners);
        System.out.println("Race Completed , 1000 meters Reached by one of the runners!");
        for (int i = 0; i < 3 && i < runners.size(); i++) {
            Runner runner = runners.get(i);
            System.out.println("Rank " + (i + 1) + ": " + runner.getName() + " - " + runner.getDistance() + " meters");
        }
    }

    private static class Runner extends Thread implements Comparable<Runner> {
        private final String name;
        private int distance;

        public Runner(String name) {
            this.name = name;
            this.distance = 0;
        }

        public int getDistance() {
            return distance;
```



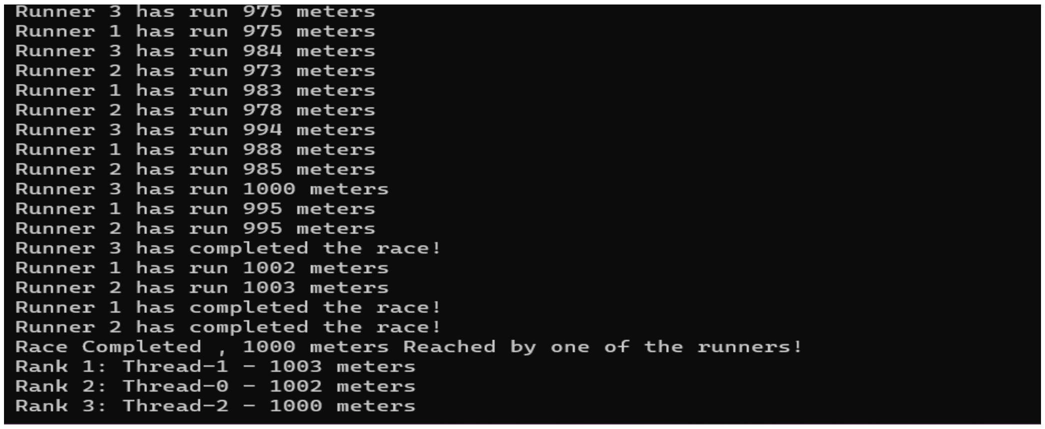
```java
        }

        @Override
        public void run() {
            Random r = new Random();
            while (distance < maximum_dist) {
                distance += r.nextInt(max_speed - min_speed + 1) + min_speed;
                System.out.println(name + " has run " + distance + " meters");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println(name + " has completed the race!");
        }

        @Override
        public int compareTo(Runner o) {
            return Integer.compare(o.distance, distance);
        }
    }
}
```

Snapshots of outputs

Number of runners: 3

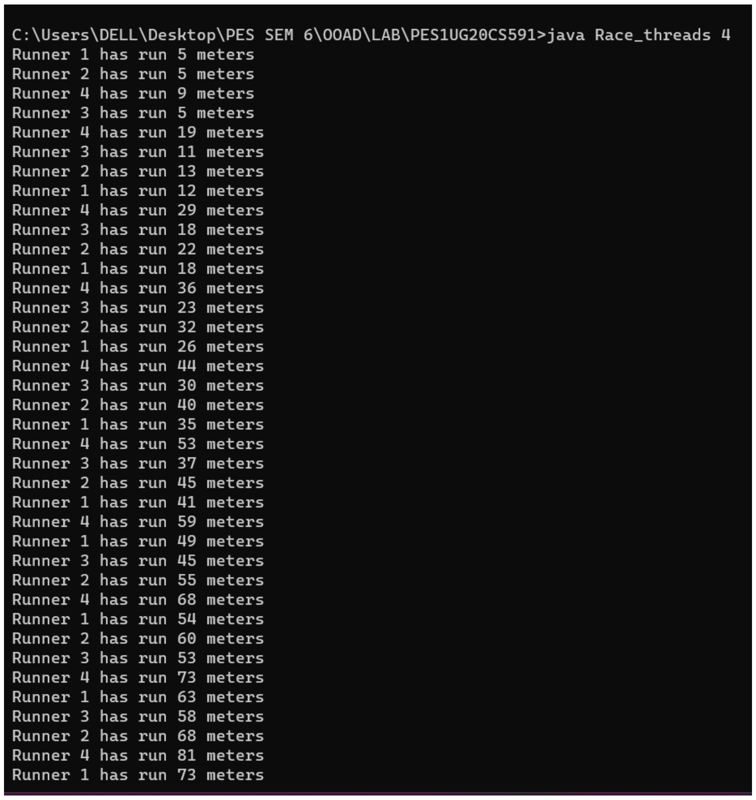


```
C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>javac Race_threads.java

C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>java Race_threads 3
Runner 3 has run 5 meters
Runner 1 has run 7 meters
Runner 2 has run 7 meters
Runner 3 has run 14 meters
Runner 1 has run 13 meters
Runner 2 has run 16 meters
Runner 2 has run 22 meters
Runner 3 has run 19 meters
Runner 1 has run 20 meters
Runner 1 has run 25 meters
Runner 3 has run 24 meters
Runner 2 has run 27 meters
Runner 2 has run 37 meters
Runner 3 has run 31 meters
Runner 1 has run 31 meters
Runner 1 has run 41 meters
Runner 2 has run 45 meters
Runner 3 has run 39 meters
Runner 3 has run 44 meters
Runner 2 has run 54 meters
Runner 1 has run 46 meters
Runner 3 has run 50 meters
Runner 2 has run 60 meters
Runner 1 has run 51 meters
Runner 3 has run 60 meters
Runner 1 has run 58 meters
Runner 2 has run 66 meters
Runner 3 has run 70 meters
Runner 1 has run 64 meters
Runner 2 has run 75 meters
Runner 1 has run 70 meters
Runner 2 has run 81 meters
Runner 3 has run 80 meters
Runner 1 has run 80 meters
Runner 2 has run 87 meters
Runner 3 has run 87 meters
```

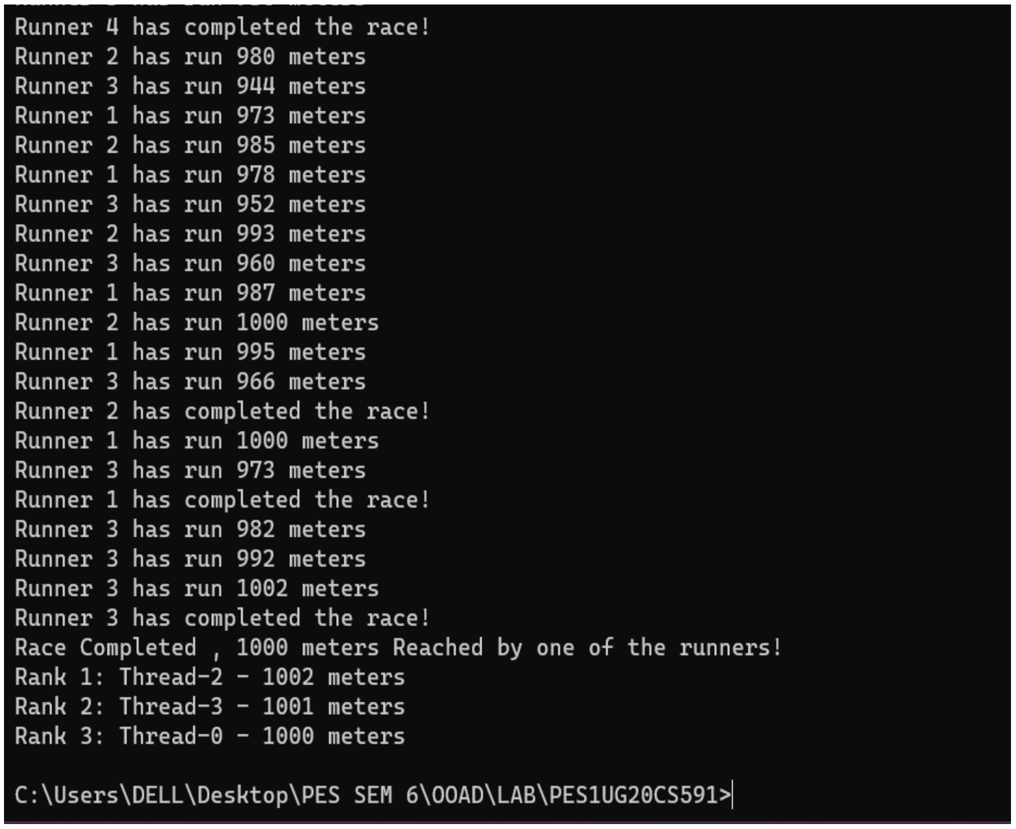


```
Runner 3 has run 975 meters
Runner 1 has run 975 meters
Runner 3 has run 984 meters
Runner 2 has run 973 meters
Runner 1 has run 983 meters
Runner 2 has run 978 meters
Runner 3 has run 994 meters
Runner 1 has run 988 meters
Runner 2 has run 985 meters
Runner 3 has run 1000 meters
Runner 1 has run 995 meters
Runner 2 has run 995 meters
Runner 3 has completed the race!
Runner 1 has run 1002 meters
Runner 2 has run 1003 meters
Runner 1 has completed the race!
Runner 2 has completed the race!
Race Completed , 1000 meters Reached by one of the runners!
Rank 1: Thread-1 - 1003 meters
Rank 2: Thread-0 - 1002 meters
Rank 3: Thread-2 - 1000 meters
```

Number of runners: 4

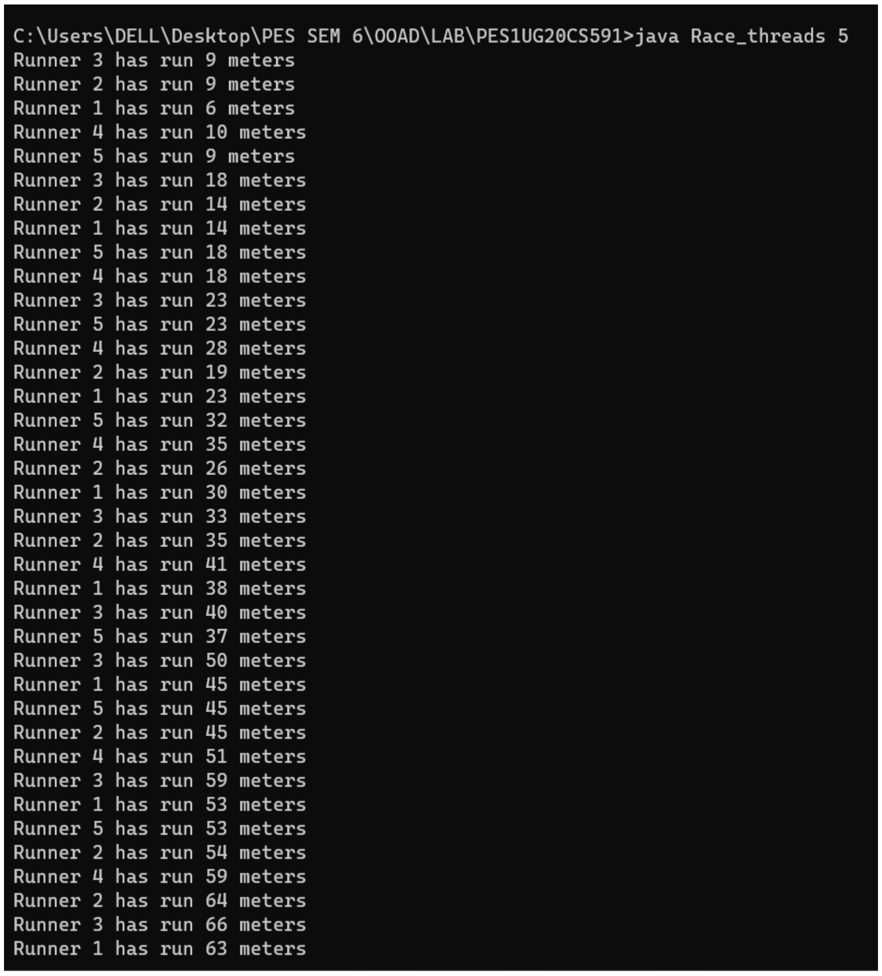


```
C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>java Race_threads 4
Runner 1 has run 5 meters
Runner 2 has run 5 meters
Runner 4 has run 9 meters
Runner 3 has run 5 meters
Runner 4 has run 19 meters
Runner 3 has run 11 meters
Runner 2 has run 13 meters
Runner 1 has run 12 meters
Runner 4 has run 29 meters
Runner 3 has run 18 meters
Runner 2 has run 22 meters
Runner 1 has run 18 meters
Runner 4 has run 36 meters
Runner 3 has run 23 meters
Runner 2 has run 32 meters
Runner 1 has run 26 meters
Runner 4 has run 44 meters
Runner 3 has run 30 meters
Runner 2 has run 40 meters
Runner 1 has run 35 meters
Runner 4 has run 53 meters
Runner 3 has run 37 meters
Runner 2 has run 45 meters
Runner 1 has run 41 meters
Runner 4 has run 59 meters
Runner 1 has run 49 meters
Runner 3 has run 45 meters
Runner 2 has run 55 meters
Runner 4 has run 68 meters
Runner 1 has run 54 meters
Runner 2 has run 60 meters
Runner 3 has run 53 meters
Runner 4 has run 73 meters
Runner 1 has run 63 meters
Runner 3 has run 58 meters
Runner 2 has run 68 meters
Runner 4 has run 81 meters
Runner 1 has run 73 meters
```

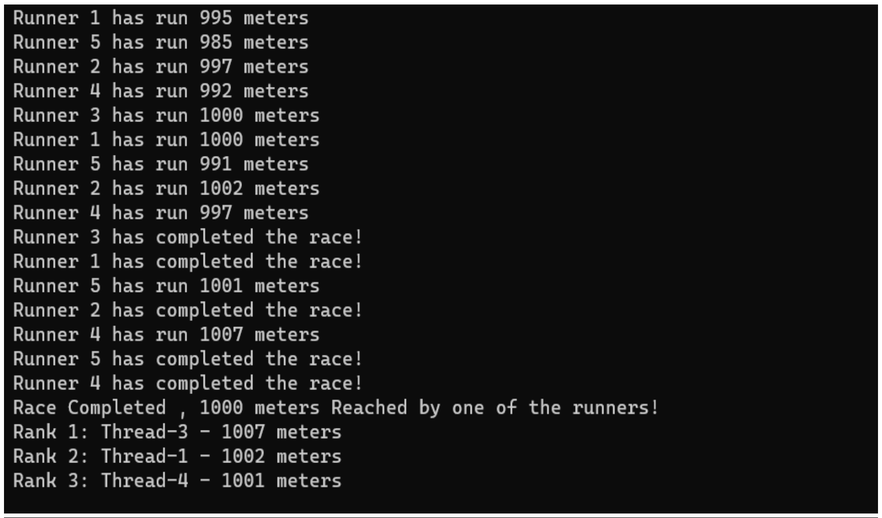


```
Runner 4 has completed the race!
Runner 2 has run 980 meters
Runner 3 has run 944 meters
Runner 1 has run 973 meters
Runner 2 has run 985 meters
Runner 1 has run 978 meters
Runner 3 has run 952 meters
Runner 2 has run 993 meters
Runner 3 has run 960 meters
Runner 1 has run 987 meters
Runner 2 has run 1000 meters
Runner 1 has run 995 meters
Runner 3 has run 966 meters
Runner 2 has completed the race!
Runner 1 has run 1000 meters
Runner 3 has run 973 meters
Runner 1 has completed the race!
Runner 3 has run 982 meters
Runner 3 has run 992 meters
Runner 3 has run 1002 meters
Runner 3 has completed the race!
Race Completed , 1000 meters Reached by one of the runners!
Rank 1: Thread-2 - 1002 meters
Rank 2: Thread-3 - 1001 meters
Rank 3: Thread-0 - 1000 meters

C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>
```

Number of runners: 5



```
C:\Users\DELL\Desktop\PES SEM 6\OOAD\LAB\PES1UG20CS591>java Race_threads 5
Runner 3 has run 9 meters
Runner 2 has run 9 meters
Runner 1 has run 6 meters
Runner 4 has run 10 meters
Runner 5 has run 9 meters
Runner 3 has run 18 meters
Runner 2 has run 14 meters
Runner 1 has run 14 meters
Runner 5 has run 18 meters
Runner 4 has run 18 meters
Runner 3 has run 23 meters
Runner 5 has run 23 meters
Runner 4 has run 28 meters
Runner 2 has run 19 meters
Runner 1 has run 23 meters
Runner 5 has run 32 meters
Runner 4 has run 35 meters
Runner 2 has run 26 meters
Runner 1 has run 30 meters
Runner 3 has run 33 meters
Runner 2 has run 35 meters
Runner 4 has run 41 meters
Runner 1 has run 38 meters
Runner 3 has run 40 meters
Runner 5 has run 37 meters
Runner 3 has run 50 meters
Runner 1 has run 45 meters
Runner 5 has run 45 meters
Runner 2 has run 45 meters
Runner 4 has run 51 meters
Runner 3 has run 59 meters
Runner 1 has run 53 meters
Runner 5 has run 53 meters
Runner 2 has run 54 meters
Runner 4 has run 59 meters
Runner 2 has run 64 meters
Runner 3 has run 66 meters
Runner 1 has run 63 meters
```



```
Runner 1 has run 995 meters
Runner 5 has run 985 meters
Runner 2 has run 997 meters
Runner 4 has run 992 meters
Runner 3 has run 1000 meters
Runner 1 has run 1000 meters
Runner 5 has run 991 meters
Runner 2 has run 1002 meters
Runner 4 has run 997 meters
Runner 3 has completed the race!
Runner 1 has completed the race!
Runner 5 has run 1001 meters
Runner 2 has completed the race!
Runner 4 has run 1007 meters
Runner 5 has completed the race!
Runner 4 has completed the race!
Race Completed , 1000 meters Reached by one of the runners!
Rank 1: Thread-3 - 1007 meters
Rank 2: Thread-1 - 1002 meters
Rank 3: Thread-4 - 1001 meters
```

--END--