UE20CS352 – OOAD with Java
Lab Assignment – 8
Multi-Threading in Java

SUBMITTED BY: MAHIKA GUPTA
SRN: PES1UG20CS243

Multithreading in Java

Multithreading is a feature used by Java programs to maximize CPU utilization by running more than one process at once. Each of these processes being run concurrently, which are parts of the program, are called threads. A thread is a lightweight sub-process, which is the smallest unit of a process.
Multithreading is used over multiprocessing as multithreading uses a shared memory system, where separate memory is not allotted for each process. This saves memory and reduces the overhead occurring in context-switching.
The OS divides processing time amongst different applications, and also divides it amongst different threads in one application.

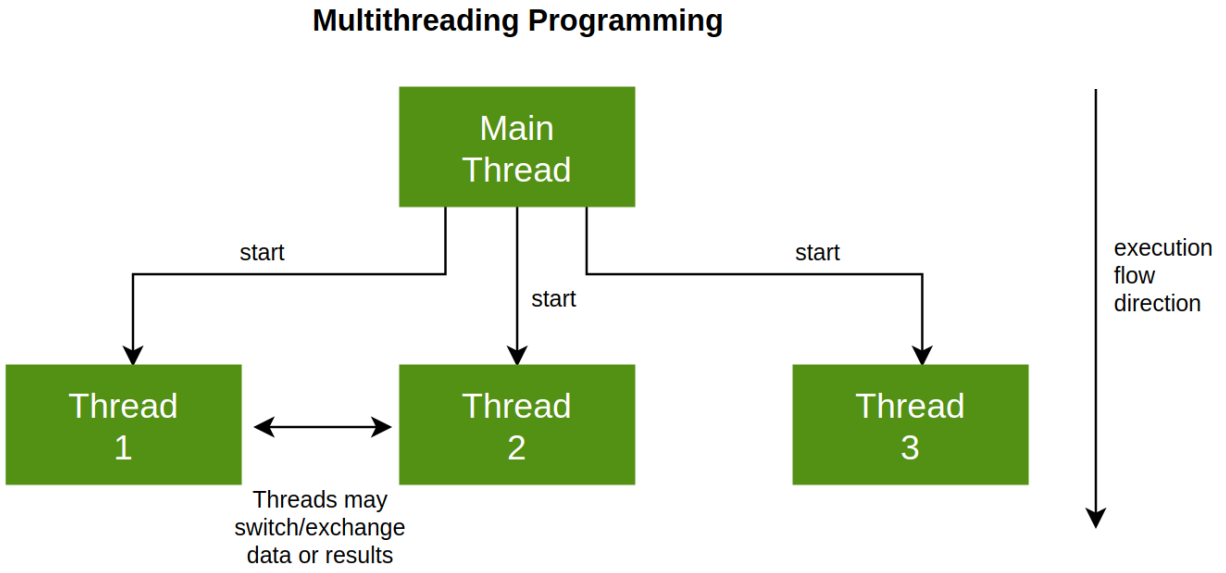
In Java, multithreading can be done using the following methods:

1. Extending the Thread Class
   A class extending the java.lang.thread class is created, which includes the run() and start() methods. These methods are inherited from the Thread class and overridden in the new class. The start() method executes the process, and returns two threads: the current thread which returns from the call to the start() method, and the second thread which executes the run() method. The Thread class also includes other methods like yield(), interrupt(), etc.
2. Implementing the Runnable Interface
   A new class is created which implements the java.lang.Runnable Interface. The new class overrides the run() method of the Runnable interface and the start() method is called to execute the run() method.

Using the Runnable interface, we can reuse the thread object for multiple classes. Furthermore, the Thread class can be extended as there is no issue of multiple inheritance.

Advantages of Multi-threading:

1. The user isn't blocked by multithreading as processes execute independently and multiple operations can be performed at once.
2. Time-saving
3. Optimal utilization of CPU
4. If exception occurs in one thread, other threads are not affected.



https://www.baeldung.com/cs/async-vs-multi-threading

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.*;

public class Run {
   public static void main(String args[]) throws InterruptedException {
       Scanner in = new Scanner(System.in);
       System.out.println("Enter the number of runners:");
       int n = in.nextInt();
       List<Runner> runners = new ArrayList<>();
       for (int i = 1; i <= n; i++) {
           runners.add(new Runner("Runner " + i));
       }

       for (Runner runner : runners) {
           runner.start();
       }
```

```
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println();
            Collections.sort(runners);
            for (int i = 0; i < runners.size(); i++) {
                Runner runner = runners.get(i);
                System.out.println(runner.getName() + " has run " +
runner.getDistance() + " m");
                if (runner.getDistance() >= 1000) {
                    System.out.println(runner.getName() + " has finished the race!\n");

                    for (int k = 0; k < runners.size(); k++) {
                        runner = runners.get(k);
                        if(runner.getDistance()<1000){
                            System.out.println(runner.getName() + " has run " +
runner.getDistance() + " m");
                        }

                    }

                    System.out.println("\nThe top 3 runners are:");
                    for (int j = 0; j < 3; j++) {
                        System.out.println((j + 1) + ". " + runners.get(j).getName());
                    }
                    return;
                }
                runner.addDistance(new Random().nextInt(100) + 5);
            }
        }

    }
}

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

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

    public int getDistance() {
        return distance;
    }

    public void addDistance(int distance) {
        this.distance += distance;
    }

    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public int compareTo(Runner other) {
        return Integer.compare(other.getDistance(), this.getDistance());
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Output 1:

```
(base) mahika@the-book OOADJ LAB % cd "/Users/mahika/OOADJ LAB/" && javac Run.java && java Run
Enter the number of runners:
3

Thread-0 has run 0 m
Thread-1 has run 0 m
Thread-2 has run 0 m

Thread-0 has run 102 m
Thread-2 has run 90 m
Thread-1 has run 12 m

Thread-0 has run 118 m
Thread-2 has run 108 m
Thread-1 has run 27 m
```

```
Thread-0 has run 674 m
Thread-1 has run 447 m
Thread-2 has run 436 m

Thread-0 has run 683 m
Thread-2 has run 482 m
Thread-1 has run 457 m

Thread-0 has run 695 m
Thread-2 has run 507 m
Thread-1 has run 486 m

Thread-0 has run 763 m
Thread-1 has run 586 m
Thread-2 has run 525 m
```

```
Thread-0 has run 935 m
Thread-1 has run 778 m
Thread-2 has run 744 m

Thread-0 has run 967 m
Thread-1 has run 853 m
Thread-2 has run 818 m

Thread-0 has run 1037 m
Thread-0 has finished the race!

Thread-1 has run 858 m
Thread-2 has run 839 m

The top 3 runners are:
1. Thread-0
2. Thread-1
3. Thread-2
```

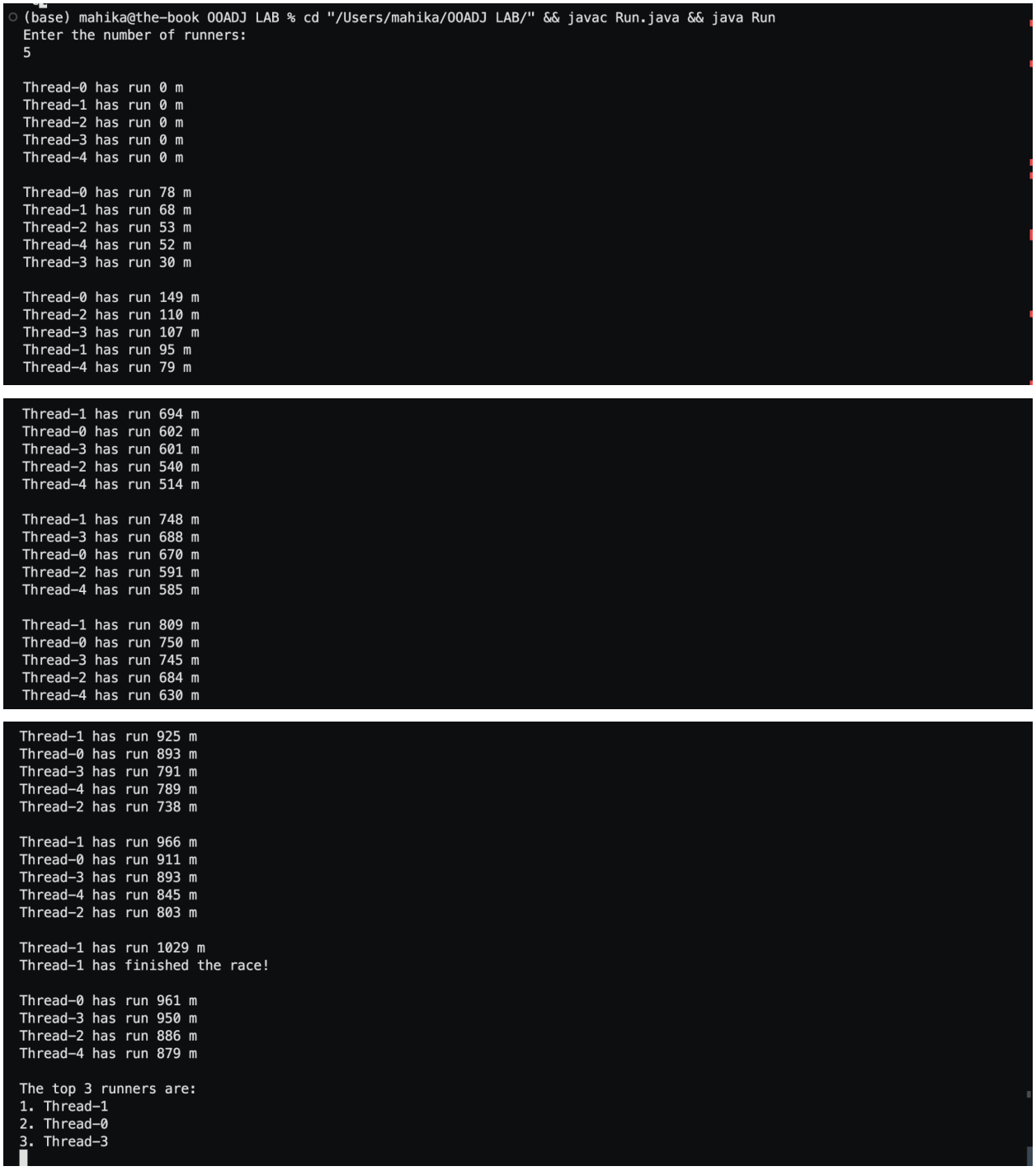
Output 2:

```
(base) mahika@the-book OOADJ LAB % cd "/Users/mahika/OOADJ LAB/" && javac Run.java && java Run
Enter the number of runners:
4

Thread-0 has run 0 m
Thread-1 has run 0 m
Thread-2 has run 0 m
Thread-3 has run 0 m

Thread-1 has run 91 m
Thread-2 has run 69 m
Thread-3 has run 52 m
Thread-0 has run 17 m

Thread-2 has run 157 m
Thread-1 has run 114 m
Thread-3 has run 71 m
Thread-0 has run 22 m
```

```
Thread-2 has run 396 m
Thread-1 has run 352 m
Thread-0 has run 275 m
Thread-3 has run 162 m

Thread-1 has run 422 m
Thread-2 has run 418 m
Thread-0 has run 306 m
Thread-3 has run 229 m

Thread-1 has run 510 m
Thread-2 has run 490 m
Thread-0 has run 316 m
Thread-3 has run 285 m
```

```
Thread-1 has run 866 m
Thread-2 has run 691 m
Thread-0 has run 565 m
Thread-3 has run 445 m

Thread-1 has run 954 m
Thread-2 has run 748 m
Thread-0 has run 608 m
Thread-3 has run 474 m

Thread-1 has run 1058 m
Thread-1 has finished the race!

Thread-2 has run 793 m
Thread-0 has run 620 m
Thread-3 has run 519 m

The top 3 runners are:
1. Thread-1
2. Thread-2
3. Thread-0
```

Output 3:



```
(base) mahika@the-book OOADJ LAB % cd "/Users/mahika/OOADJ LAB/" && javac Run.java && java Run
Enter the number of runners:
5

Thread-0 has run 0 m
Thread-1 has run 0 m
Thread-2 has run 0 m
Thread-3 has run 0 m
Thread-4 has run 0 m

Thread-0 has run 78 m
Thread-1 has run 68 m
Thread-2 has run 53 m
Thread-4 has run 52 m
Thread-3 has run 30 m

Thread-0 has run 149 m
Thread-2 has run 110 m
Thread-3 has run 107 m
Thread-1 has run 95 m
Thread-4 has run 79 m

Thread-1 has run 694 m
Thread-0 has run 602 m
Thread-3 has run 601 m
Thread-2 has run 540 m
Thread-4 has run 514 m

Thread-1 has run 748 m
Thread-3 has run 688 m
Thread-0 has run 670 m
Thread-2 has run 591 m
Thread-4 has run 585 m

Thread-1 has run 809 m
Thread-0 has run 750 m
Thread-3 has run 745 m
Thread-2 has run 684 m
Thread-4 has run 630 m

Thread-1 has run 925 m
Thread-0 has run 893 m
Thread-3 has run 791 m
Thread-4 has run 789 m
Thread-2 has run 738 m

Thread-1 has run 966 m
Thread-0 has run 911 m
Thread-3 has run 893 m
Thread-4 has run 845 m
Thread-2 has run 803 m

Thread-1 has run 1029 m
Thread-1 has finished the race!

Thread-0 has run 961 m
Thread-3 has run 950 m
Thread-2 has run 886 m
Thread-4 has run 879 m

The top 3 runners are:
1. Thread-1
2. Thread-0
3. Thread-3
```