UE20CS352-OOADJ lab-8

NAME	SRN	CLASS & SECTION
Vijay J	PES2UG20CS815	6 - J

Java Multithreading

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.

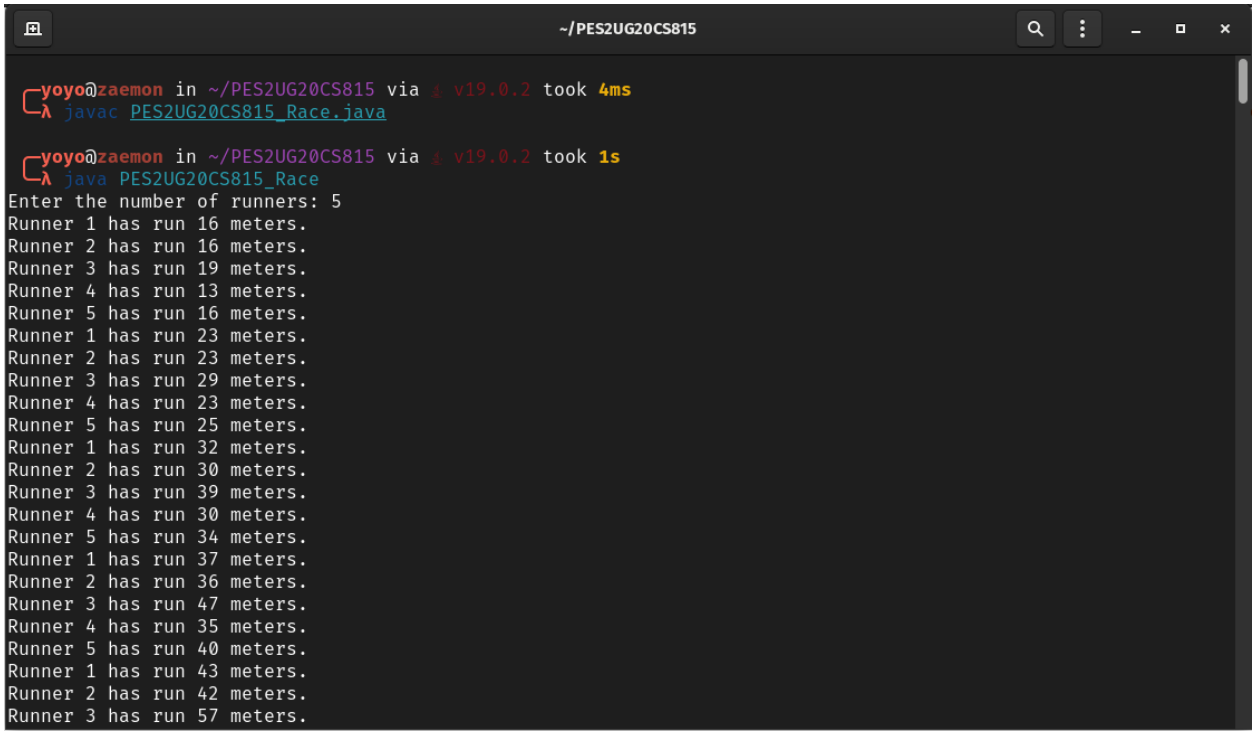
Race.java

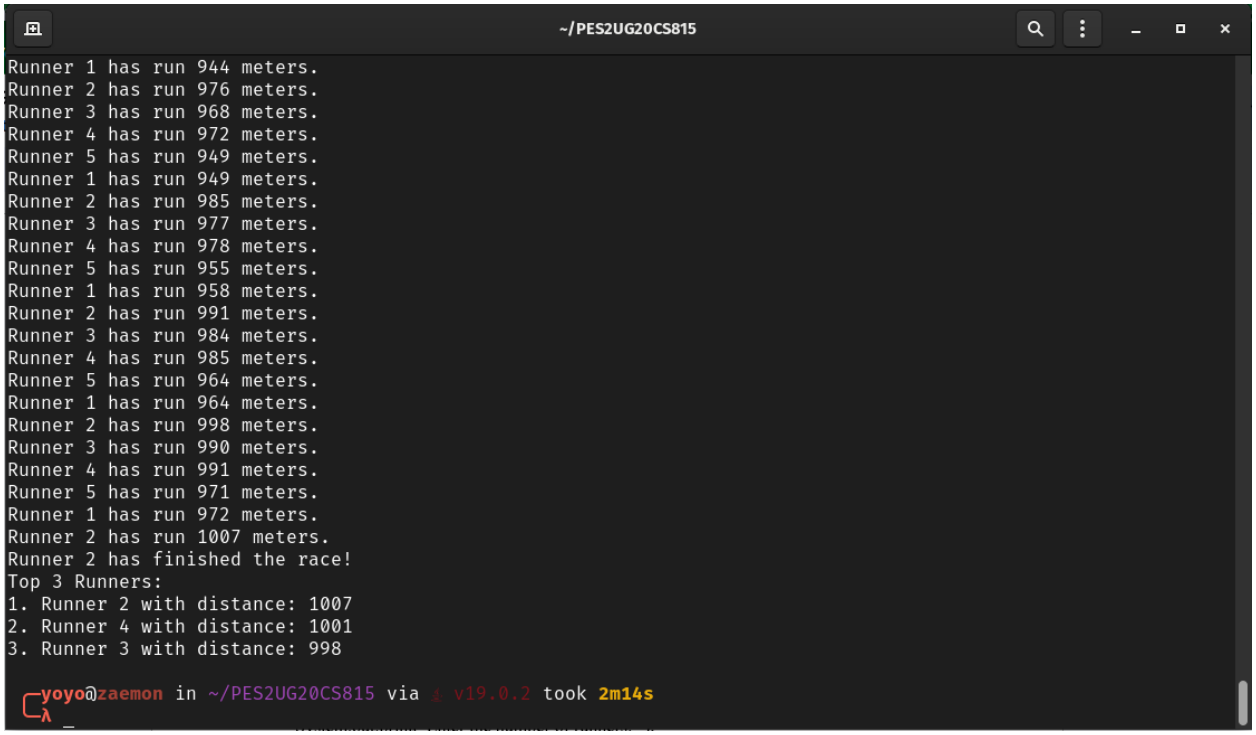
```
import java.util.*;

public class PES2UG20CS815_Race {
   public static void main(String[] args) throws InterruptedException {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the number of runners: ");
      int numRunners = scanner.nextInt();

      Runner[] runners = new Runner[numRunners];
      for (int i = 0; i < numRunners; i++) {
         runners[i] = new Runner("Runner " + (i + 1));
         runners[i].start();
      }

      while (true) {
         Thread.sleep(1000);
         for (Runner runner : runners) {
            System.out.println(runner.getName() + " has run " + runner.getDistance() + "
meters.");
            if (runner.getDistance() >= 1000) {
               System.out.println(runner.getName() + " has finished the race!");
               Arrays.sort(runners, Comparator.comparingInt(Runner::getDistance).reversed());
               System.out.println("Top 3 Runners:");
```

```
                for (int i = 0; i < 3 && i < numRunners; i++) {
                    System.out.println(
                            (i + 1) + ". " + runners[i].getName() + " with distance: " +
runners[i].getDistance());
                }
                System.exit(0);
            }
        }
    }
}

class Runner extends Thread {
    private int distance;

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

    public void run() {
        Random random = new Random();
        while (true) {
            distance += random.nextInt(6) + 5;
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

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

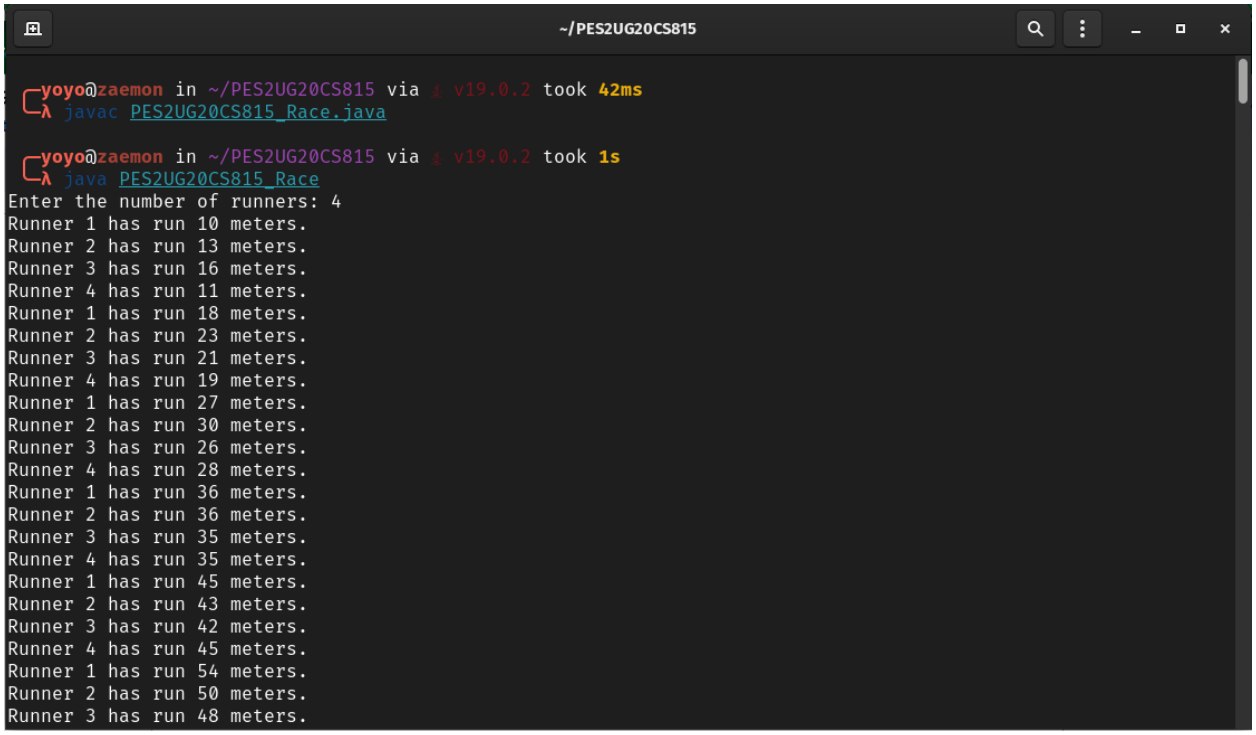
OUTPUT:

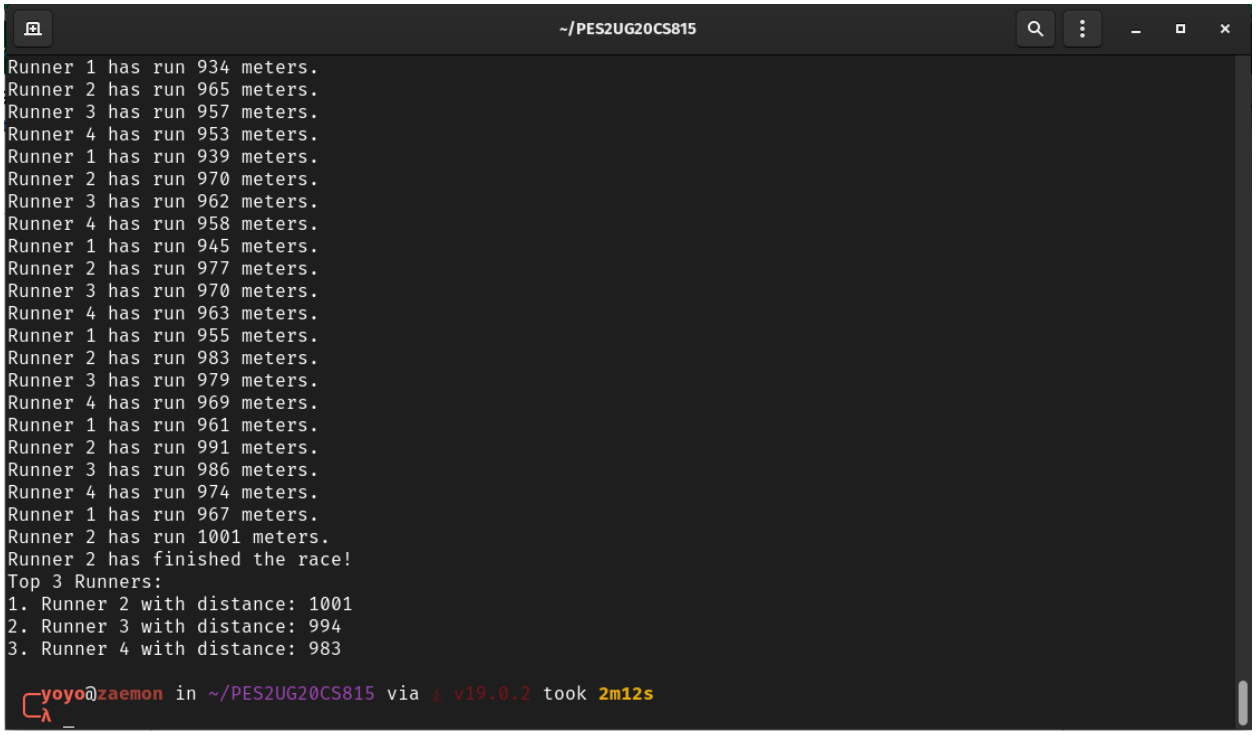
Runner-5



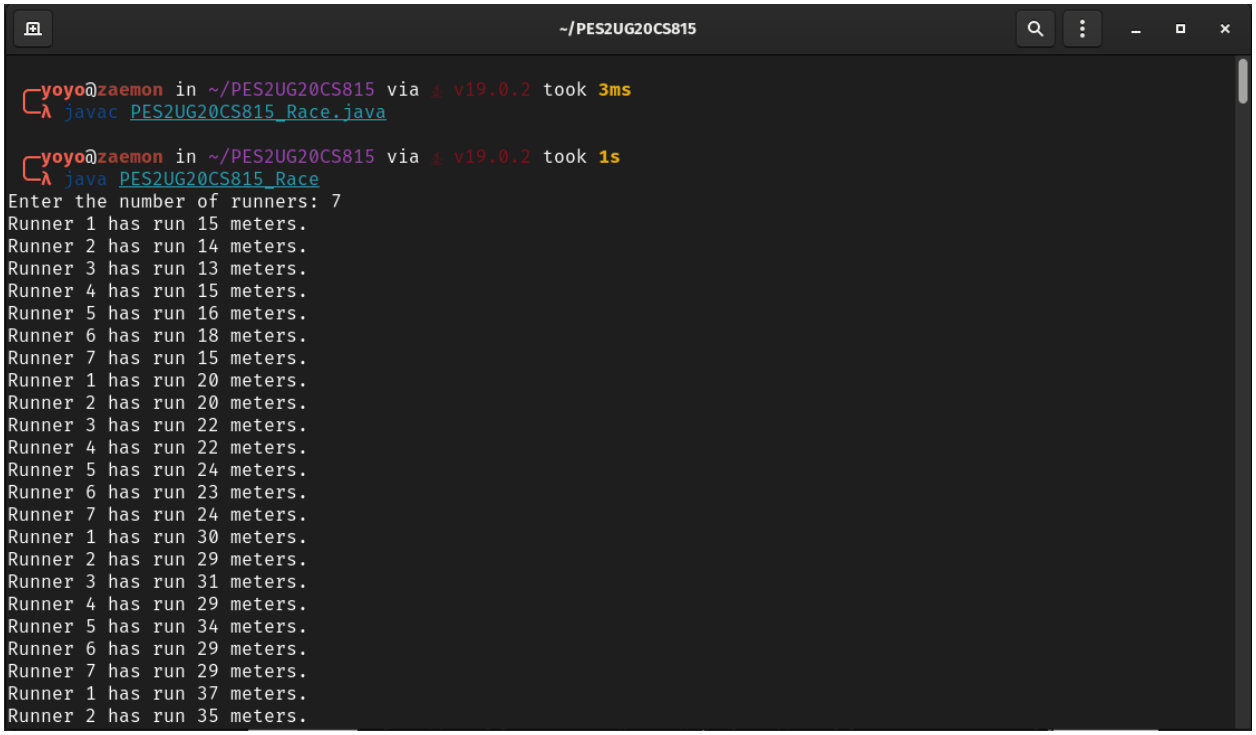


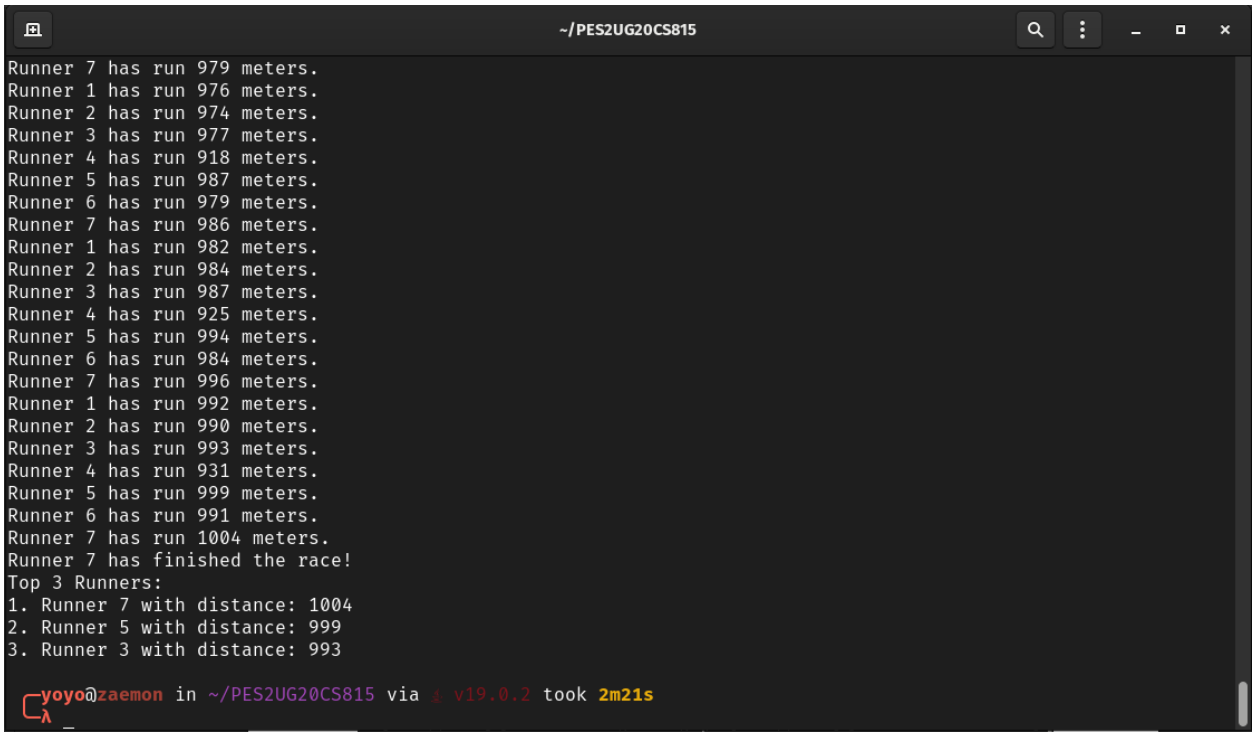
Runner-4


~/PES2UG20CS815
yoyo@zaemon in ~/PES2UG20CS815 via v19.0.2 took 42ms
λ javac PES2UG20CS815_Race.java
yoyo@zaemon in ~/PES2UG20CS815 via v19.0.2 took 1s
λ java PES2UG20CS815_Race
Enter the number of runners: 4
Runner 1 has run 10 meters.
Runner 2 has run 13 meters.
Runner 3 has run 16 meters.
Runner 4 has run 11 meters.
Runner 1 has run 18 meters.
Runner 2 has run 23 meters.
Runner 3 has run 21 meters.
Runner 4 has run 19 meters.
Runner 1 has run 27 meters.
Runner 2 has run 30 meters.
Runner 3 has run 26 meters.
Runner 4 has run 28 meters.
Runner 1 has run 36 meters.
Runner 2 has run 36 meters.
Runner 3 has run 35 meters.
Runner 4 has run 35 meters.
Runner 1 has run 45 meters.
Runner 2 has run 43 meters.
Runner 3 has run 42 meters.
Runner 4 has run 45 meters.
Runner 1 has run 54 meters.
Runner 2 has run 50 meters.
Runner 3 has run 48 meters.


~/PES2UG20CS815
Runner 1 has run 934 meters.
Runner 2 has run 965 meters.
Runner 3 has run 957 meters.
Runner 4 has run 953 meters.
Runner 1 has run 939 meters.
Runner 2 has run 970 meters.
Runner 3 has run 962 meters.
Runner 4 has run 958 meters.
Runner 1 has run 945 meters.
Runner 2 has run 977 meters.
Runner 3 has run 970 meters.
Runner 4 has run 963 meters.
Runner 1 has run 955 meters.
Runner 2 has run 983 meters.
Runner 3 has run 979 meters.
Runner 4 has run 969 meters.
Runner 1 has run 961 meters.
Runner 2 has run 991 meters.
Runner 3 has run 986 meters.
Runner 4 has run 974 meters.
Runner 1 has run 967 meters.
Runner 2 has run 1001 meters.
Runner 2 has finished the race!
Top 3 Runners:
1. Runner 2 with distance: 1001
2. Runner 3 with distance: 994
3. Runner 4 with distance: 983
yoyo@zaemon in ~/PES2UG20CS815 via v19.0.2 took 2m12s
λ _

Runner-7


~/PES2UG20CS815
yoyo@zaemon in ~/PES2UG20CS815 via v19.0.2 took 3ms
λ javac PES2UG20CS815_Race.java
yoyo@zaemon in ~/PES2UG20CS815 via v19.0.2 took 1s
λ java PES2UG20CS815_Race
Enter the number of runners: 7
Runner 1 has run 15 meters.
Runner 2 has run 14 meters.
Runner 3 has run 13 meters.
Runner 4 has run 15 meters.
Runner 5 has run 16 meters.
Runner 6 has run 18 meters.
Runner 7 has run 15 meters.
Runner 1 has run 20 meters.
Runner 2 has run 20 meters.
Runner 3 has run 22 meters.
Runner 4 has run 22 meters.
Runner 5 has run 24 meters.
Runner 6 has run 23 meters.
Runner 7 has run 24 meters.
Runner 1 has run 30 meters.
Runner 2 has run 29 meters.
Runner 3 has run 31 meters.
Runner 4 has run 29 meters.
Runner 5 has run 34 meters.
Runner 6 has run 29 meters.
Runner 7 has run 29 meters.
Runner 1 has run 37 meters.
Runner 2 has run 35 meters.


~/PES2UG20CS815
Runner 7 has run 979 meters.
Runner 1 has run 976 meters.
Runner 2 has run 974 meters.
Runner 3 has run 977 meters.
Runner 4 has run 918 meters.
Runner 5 has run 987 meters.
Runner 6 has run 979 meters.
Runner 7 has run 986 meters.
Runner 1 has run 982 meters.
Runner 2 has run 984 meters.
Runner 3 has run 987 meters.
Runner 4 has run 925 meters.
Runner 5 has run 994 meters.
Runner 6 has run 984 meters.
Runner 7 has run 996 meters.
Runner 1 has run 992 meters.
Runner 2 has run 990 meters.
Runner 3 has run 993 meters.
Runner 4 has run 931 meters.
Runner 5 has run 999 meters.
Runner 6 has run 991 meters.
Runner 7 has run 1004 meters.
Runner 7 has finished the race!
Top 3 Runners:
1. Runner 7 with distance: 1004
2. Runner 5 with distance: 999
3. Runner 3 with distance: 993
yoyo@zaemon in ~/PES2UG20CS815 via v19.0.2 took 2m21s
λ

Introduction to Multithreading

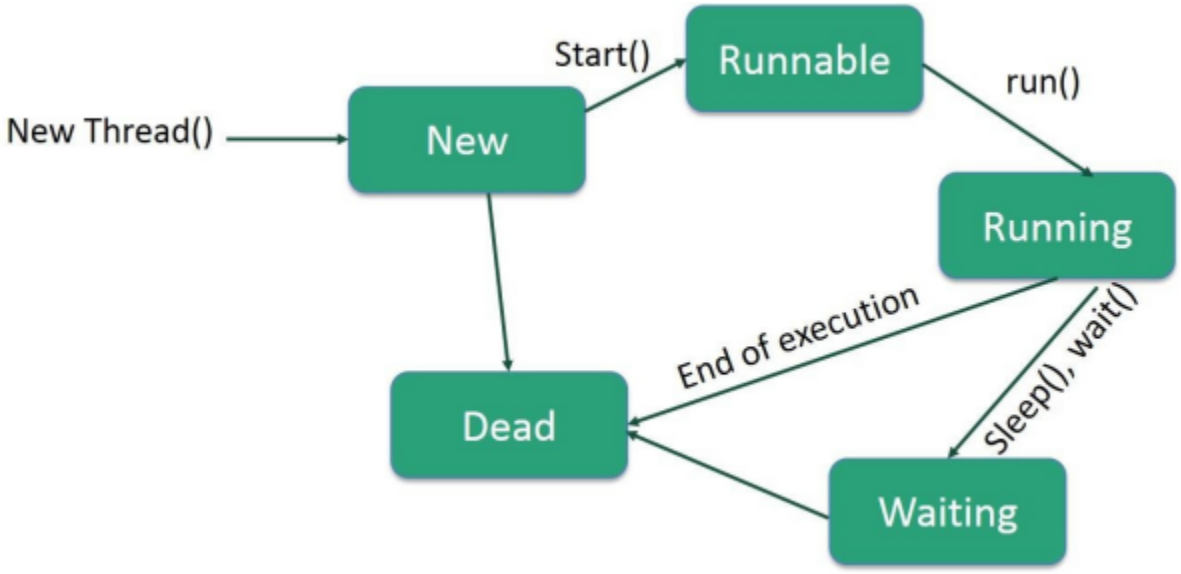
Multithreading is a programming concept that allows a program to perform multiple tasks simultaneously. A thread is a single path of execution within a program, and a program with multiple threads can perform multiple tasks concurrently. In other words, multithreading allows a program to make the most efficient use of available resources, such as the CPU, by dividing a program into smaller units of execution that can run concurrently.

Description of Threads/synchronized threads

Threading is a programming concept that involves dividing a program into smaller threads of execution that can run concurrently. Each thread represents a separate flow of execution within the program and operates independently of the other threads.

Threads share the same memory space and can communicate with each other to exchange data and synchronize their execution. This allows for more efficient and responsive programs, as multiple tasks can be performed simultaneously without the need for separate processes.

Life cycle of threads



Different thread states

New: A thread is created using the new keyword and is in the new state. In this state, the thread has been created but not yet started.

Runnable: After the start() method is called on the thread object, the thread moves to the runnable state. In this state, the thread is ready to be executed but has not been scheduled by the operating system yet.

Running: When the operating system selects the thread for execution, it enters the running state. In this state, the thread's code is executed, and it is actively running.

Blocked: If the thread is waiting for some external event, such as I/O or synchronization, it enters the blocked state. In this state, the thread's resources are temporarily released until the event occurs.

Waiting: If the thread is waiting for a specific signal from another thread, it enters the waiting state. In this state, the thread's resources are released until it receives the required signal.

Timed Waiting: If the thread is waiting for a specific signal, but with a timeout, it enters the timed waiting state. In this state, the thread's resources are released until it receives the required signal or until the timeout expires.

Terminated: When the thread has completed its work or is no longer needed, it enters the terminated state. In this state, the thread's resources are released, and it cannot be restarted.