Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester:(Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 10
Course Title: Object-Oriented Programming Lab
Course Code: CSE 202 Section: 223 D9

Lab Experiment Name: Multi-threaded Programming in Java

Student Details

Name		ID
1.	Promod Chandra Das	231002005

Lab Date :
Submission Date :
Course Teacher's Name : Noyan Ali

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

➢ TITLE OF THE LAB REPORT EXPERIMENT:-

Multi-threaded Programming in Java

➢ OBJECTIVES/AIM :

- After this lecture, the students will be able to:
- Understand and implement java thread
- Apply thread in different use cases

o Problem analysis:

The lab is designed to introduce the students to get the basic idea on the java thread programming. In this lab
the basics of java thread, their implementation, and different applications of thread will be taught. To start,
we will solve the following problem:
Take two integers M and N as input from the user. Generate all the prime numbers from 1 to M. Divide the input range among N number of threads and generate the numbers in parallel. Show the outputs including
which thread generated which number.Terms

➢ Theory:-

- **Multitasking**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU.
Multitasking can be achieved by two ways:

- **Process-based Multitasking(Multiprocessing)**

1. Each process have its own address in memory i.e. each process allocates separate memory area.
2. Process is heavyweight.
3. Cost of communication between the processes is high.
4. Switching from one process to another require some time for saving and loading registers, memory maps,
updating lists etc.

- **Thread-based Multitasking (Multithreading)**

1. Threads share the same address space.
2. Thread is lightweight.
3. Cost of communication between the thread is low.

- **Lifecycle of a Thread**

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution. Threads
are independent, if there occurs exception in one thread, it doesn't affect other threads. But threads of a process
share a common memory area.
The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
1. New – When we create an instance of Thread class, a thread is in a new state.
2. Runnable – The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
3. Running– The thread is in running state if the thread scheduler has selected it.
4. Blocked/Non-Runnable – This is the state when the thread is still alive, but is currently not eligible to run.
5. Terminated – A thread can be terminated, which halts its execution immediately at any given time.
Once a thread is terminated, it cannot be resumed. A thread is in terminated or dead state when its run() method exits.

- **Creating Thread**

There are two ways to create a thread:
1. By extending Thread class
2. By implementing Runnable interface

- **Creating thread by extending Thread class**

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface. Commonly used constructors of Thread class:
1. Thread()
2. Thread(String name)
3. Thread(Runnable r)
4. Thread(Runnable r, String name)

- **Creating thread by implementing Runnable interface**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a
thread. Runnable interface have only one method named run().
public void run(): is used to perform action for a thread.

- **Sleeping Thread**

Thread.sleep causes the current thread to suspend execution for a specified period. As you know that, at a time, only one thread is executed.If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on. This is an efficient means of making processor time available to the other threads of
an application or other applications that might be running on a computer system.
The Thread class provides two methods for sleeping a thread:
1. public static void sleep(long miliseconds)throws InterruptedException
2. public static void sleep(long miliseconds, int nanos)throws InterruptedException

➢ Algorithm

Problem Statement: Take two integers M and N as input from the user. Generate all the prime numbers from 1 to M. Divide the input range among N number of threads and generate the numbers in parallel. Show the outputs including which thread generated which number.

Algorithm 1: Using Threads to generate prime numbers

Input: int M, int N

1 Step 1: Take two integers M and N as input.
2 Step 2: Create N threads and divide the input range 1 to M into N threads
3 Step 3: Each thread will generate and print the prime numbers in its defined range.
4 Step 4: Exit

➢ Lab Exercise (Submit as a report

- Search for a number N inside a randomly generated array of size 200 using 5 threads.

✓ Answer:01:

```
import java.util.Random;

class SearchTask implements Runnable {
    private int[] array;
    private int start;
    private int end;
    private int numberToFind;
    private boolean found;

    public SearchTask(int[] array, int start, int end, int numberToFind) {
        this.array = array;
        this.start = start;
        this.end = end;
        this.numberToFind = numberToFind;
        this.found = false;
    }

    @Override
    public void run() {
        for (int i = start; i < end; i++) {
            if (array[i] == numberToFind) {
                found = true;
                System.out.println("Number " + numberToFind + " found at index " + i);
                break;
            }
        }
    }

    public boolean isFound() {
        return found;
```

```
    }
}

public class MultiThreadSearch {
    public static void main(String[] args) {
        int arraySize = 200;
        int[] array = new int[arraySize];
        Random random = new Random();

        // Generate random array
        for (int i = 0; i < arraySize; i++) {
            array[i] = random.nextInt(1000);  // Random numbers between 0 and 999
        }

        int numberToFind = random.nextInt(1000);  // Random number to find
        System.out.println("Number to find: " + numberToFind);

        int numThreads = 5;
        int segmentSize = arraySize / numThreads;
        Thread[] threads = new Thread[numThreads];
        SearchTask[] tasks = new SearchTask[numThreads];

        // Create and start threads
        for (int i = 0; i < numThreads; i++) {
            int start = i * segmentSize;
            int end = (i == numThreads - 1) ? arraySize : start + segmentSize;
            tasks[i] = new SearchTask(array, start, end, numberToFind);
            threads[i] = new Thread(tasks[i]);
            threads[i].start();
        }

        // Wait for all threads to finish
        for (int i = 0; i < numThreads; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // Check if any thread found the number
        boolean found = false;
        for (SearchTask task : tasks) {
            if (task.isFound()) {
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("Number " + numberToFind + " not found in the array.");
        }
    }
}
```

➢ Discussion & Conclusion:

Multi-threaded programming in Java allows concurrent execution, improving performance by utilizing multiple cores. Threads can be created by extending Thread or implementing Runnable. Proper synchronization is crucial to avoid race conditions, using synchronized methods or blocks, and volatile keywords. The java.util.concurrent package simplifies thread management with ExecutorService and thread pools. Advanced concurrency controls include Locks and Conditions. Efficient multi-threading enhances resource utilization and speeds up complex tasks.