**Lab 22**

**Exercise 1**

In this exercise you are going to write an improved version of the prime numbers exercise from last week.

The program you will write will count all the primes in a given range of integers, and divide this work up among several threads. Each thread will be assigned part of the range of integers and will count the primes in its assigned range. At the end of its computation, the thread has to add its count to the overall total number of primes found. The variable that represents the total is shared by all the threads.

1. Create a new class called CountPrimesThread which implements the Runnable interface. A Thread belonging to this class will count the number of primes in a specified range of integers. The range is from min to max, inclusive, where min and max are given as parameters to the constructor. After counting, the thread outputs a message about the number of primes that it has found, and it adds its count to the overall total by calling the addToTotal(int) method.

public class CountPrimesThread implements Runnable {

}

1. Inside this class define
   1. 3 private integer member variables, min and max to represent the range of integers and count to hold the count of primes generated by the thread.
   2. 1 private static integer variable called total to hold the total number of primes found

private int count;

private int min, max;

private static int total=0;

1. Write the constructor to initialize the min and max using the parameter list. Initialise the count to 0.

public CountPrimesThread(int min, int max) {

this.min = min;

this.max = max;

count=0;

}

1. Write a private static method called addToTotal which updates the total variable defined earlier. This method also needs to be synchronized so that it can be safely used by different threads.

synchronized private static void addToTotal(int x) {

total = total + x;

System.out.println(total + " primes found so far.");

}

1. Write a private static method isPrime() that takes an integer as a parameter and returns a boolean indicating whether the number is a prime or not.

private static boolean isPrime(int x) {

int top = (int) Math.sqrt(x);

for (int i = 2; i <= top; i++) {

if (x % i == 0) {

return false;

}

}

return true;

}

1. Write a private static method countPrimes() which takes two integers as parameters representing the range i.e. min and max. This method then calls the isPrime() method to determine if each number is a prime or not. A count of all the primes is stored in the local variable count.

private static int countPrimes(int min, int max) {

int count = 0;

for (int i = min; i <= max; i++) {

if (isPrime(i)) {

count++;

}

}

return count;

}

1. Override the run() method to call the method countPrimes() and store the return value in the member variable count. Print out the number of primes found by this thread. The addToTotal() method is then called to update the overall static count variable.

public void run() {

count = countPrimes(min, max);

System.out.println("There are " + count + " primes between " + min + " and " +

max);

addToTotal(count);

}

Create a separate class called TestPrime

1. Write the code to ask the user to enter the number of threads they wish to use between 1 and 5. Provide validation on this range.

Scanner in = new Scanner(System.in);

int numberOfThreads = 0;

while (numberOfThreads < 1 || numberOfThreads > 5) {

System.out.print("How many threads do you want to use (from 1 to 5) ? ");

numberOfThreads = in.nextInt();

if (numberOfThreads < 1 || numberOfThreads > 5) {

System.out.println("Please enter 1, 2, 3, 4, or 5 !");

}

}

1. Write code to ask the user to enter the starting and ending range for the numbers. Declare these two variables at the start of the main method.

System.out.println("Enter the starting range: ");

start=in.nextInt();

String s = in.nextLine();

System.out.println("Enter the ending range: ");

end=in.nextInt();

1. Calculate the size of the range part for each thread. Print out a message showing the range and the number of threads being used. Calculate the start time for the process using System.currentTimeMillis().

int rangePart = (end-start) / numberOfThreads;

System.out.println("\nCounting primes between " + (start + 1) + " and "

+ (end) + " using " + numberOfThreads + " threads...\n");

long startTime = System.currentTimeMillis();

1. Create an array of threads and use a for loop to create runnable objects of type CountPrimesThread. Create a new thread for each runnable object and store it in the array.

Thread[] worker = new Thread[numberOfThreads];

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

CountPrimesThread cpt = new CountPrimesThread(start + i \* rangePart + 1, start + (i + 1) \* rangePart);

worker[i] = new Thread(cpt);

}

1. Use a for loop to go through the array and start each thread running.

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

worker[i].start();

}

1. Use a for loop to determine if any of the threads are still alive. If so then wait for these threads to finish by calling on the join method.

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

while (worker[i].isAlive()) {

try {

worker[i].join();

} catch (InterruptedException e) {

}

}

}

1. Calculate the elapsed time of the process and display it on the screen.

long elapsedTime = System.currentTimeMillis() - startTime;

System.out.println("\nTotal elapsed time: " + (elapsedTime / 1000.0) + " seconds.\n");

**Sample Output**

How many threads do you want to use (from 1 to 5) ? 3

Enter the starting range:

3000000

Enter the ending range:

6000000

Counting primes between 3000001 and 6000000 using 3 threads...

There are 66330 primes between 3000001 and 4000000

66330 primes found so far.

There are 65367 primes between 4000001 and 5000000

131697 primes found so far.

There are 64336 primes between 5000001 and 6000000

196033 primes found so far.

Total elapsed time: 1.518 seconds.