USCS303 – OS: Practical – 05: Threads

Practical Aim: Threads (Multi-threading)

[Practical Date: 14th Aug 2021 1](#_Toc79852822)

[Threads State: Life Cycle of Threads 2](#_Threads_State:_Life)

[Summation 3](#_Toc79852824)

[Question 01: 3](#_Toc79852825)

[Code 3](#_Toc79852826)

[Output 4](#_Toc79852827)

[Primes 4](#_Toc79852828)

[Question 02: 4](#_Toc79852829)

[Code 4](#_Toc79852830)

[Output: 8](#_Toc79852831)

[Fibonacci 8](#_Toc79852832)

[Question 03: 8](#_Toc79852833)

[Code: 9](#_Toc79852834)

[Output: 10](#_Toc79852835)

# Threads State: Life Cycle of Threads

1. **New and Runnable States:**

A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread, which places it in the runnable state. A thread in the runnable state is considered to be executing its task.

1. **Waiting State:**

Sometimes a runnable thread transitions to the waiting state while it waits for another thread to perform a task. A waiting thread transitions back to the runnable state only when another thread notifies it to continue executing.

1. **Timed Waiting State:**

A runnable thread can enter the timed waiting state for a specified interval of time. It transitions back to the runnable state when that time interval expires or when the event it’s waiting for occurs. Timed waiting and waiting threads cannot use a processor, even if one is available. A runnable thread can transition to the timed waiting state if it provides an optional wait interval when it’s waiting for another thread to perform a task. Such a thread returns to the runnable state when it’s notified by another thread or when the timed interval expires – whichever comes first. Another way to place a thread in the timed waiting state is to put a runnable thread to sleep. A sleeping thread remains in the timed waiting state for a designated period of time (called a sleep interval), after which it returns to the runnable state.

1. **Blocked State:**

A runnable thread transitions to the blocked state when it attempts to perform a task that cannot be completed immediately and it must temporarily wait until that task completes.

1. **Terminate State:**

A runnable thread enters the terminated state (sometimes called the dead state) when it successfully completes its task or otherwise terminates (perhaps due to an error).

# Summation

## Question 01:

Write a multithreaded java program that determines the summation of a non-negative integer. The Summation class implements the Runnable interface. Thread creation is performed by creating an object instance of the Thread class and passing the constructor a Runnable object.

Code**:**

class P5\_Q1\_Summation\_ST implements Runnable {

int upperLimit, sum;

public P5\_Q1\_Summation\_ST(int upperLimit) {

this.upperLimit=upperLimit;

}

public void run() {

for (int i=1; i<=upperLimit; i++)

sum += i;

}

}

public class P5\_Q1\_SummationTest\_ST {

public static void main(String[] args) {

if (args.length <= 0)

System.out.println("Usage: P5\_Q1\_Summation\_ST <integervalue>");

else {

int upp = Integer.parseInt(args[0]);

if (upp <= 0)

System.out.println("args[0]: " + args[0] + "must be a positive number");

else {

P5\_Q1\_Summation\_ST s = new P5\_Q1\_Summation\_ST(upp);

Thread t = new Thread(s);

t.start();

try {

t.join();

System.out.println("The sum of first " + upp + "elements is " + (s.sum));

}

catch (Exception e) {

e.printStackTrace();

}

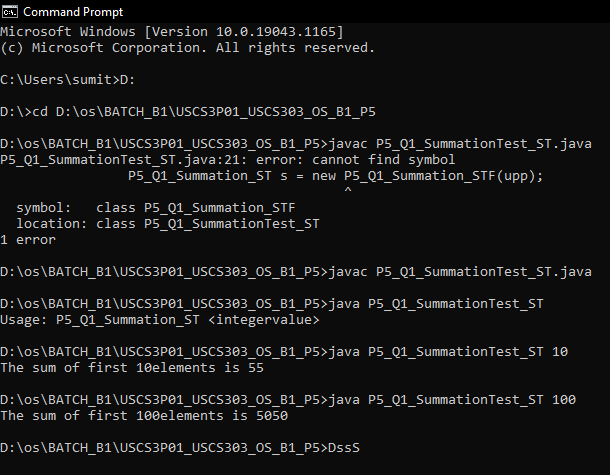
} // inner else ends

} // outer else ends

} // main ends

}

Output**:**



# Primes

## Question 02:

Write a multithreaded Java program that outputs prime numbers. This program should work as follows: The user will run the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.

Code**:**

**Filename:** P5\_Q2\_Primes\_ST.java

import java.io.\*;

import java.util.\*;

public class P5\_Q2\_Primes\_ST {

public static void main(String[] args) {

try {

P5\_Q2\_PrimeThread\_ST pt = null;

System.out.print("Enter a number > ");

Scanner scan = new Scanner(System.in);

int limit = scan.nextInt();

System.out.print("Enter a file name to store the result > ");

String fName = scan.next();

if (fName.length() > 0)

pt = new P5\_Q2\_PrimeThread\_ST (limit, new FileOutputStream(fName));

else

pt = new P5\_Q2\_PrimeThread\_ST(limit);

pt.run();

} catch(Exception e) {

e.printStackTrace();

}

**}**

**}**

**Filename:** P5\_Q2\_PrimeThread\_ST.java

import java.io.\*;

class P5\_Q2\_PrimeThread\_ST extends Thread {

private PrintStream pOut = null;

private int limit = 0;

// default constructor.does nothing

public P5\_Q2\_PrimeThread\_ST() {

}

// constructor to set the number below which to generate primes no

// output stream is specified, so it outputs to the System.out

public P5\_Q2\_PrimeThread\_ST(int l) {

limit = l;

try {

pOut = System.out;

} catch (Exception e) {

e.printStackTrace();

}

}

// constructor that sets both the number, as above, and specifies an

// output stream if the specified stream is null, uses System.out

public P5\_Q2\_PrimeThread\_ST(int l, OutputStream outS) {

limit = l;

try {

if (outS != null){

pOut = new PrintStream(outS);

} else {

pOut = System.out;

}

} catch(Exception e) {

e.printStackTrace();

}

}

// method that performs the work of the thread in this case the

// generation of prime numbers

public void run() {

// compute primes via the seive

boolean numbers[] = new boolean[limit+1];

numbers[0] = false;

numbers[1] = false;

for (int i=2; i<numbers.length; i++){

numbers[i] = true;

}

for (int i=2; i<numbers.length; i++){

if (numbers[i]) {

for (int j=(2\*i); j<numbers.length; j+= i){

numbers[j] = false;

} // inner for ends

}

}

for (int i=0; i<numbers.length; i++){

if(numbers[i])

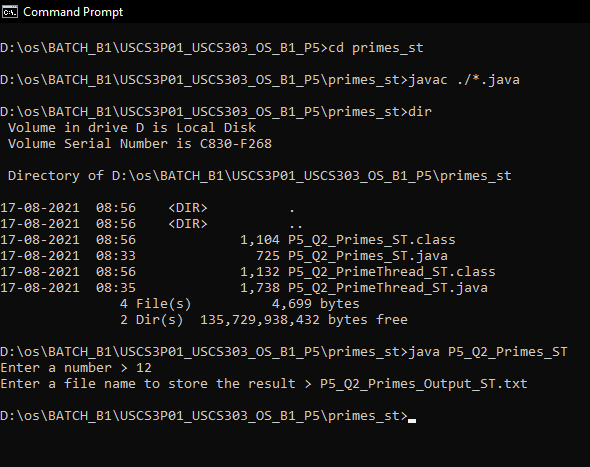
pOut.println(i);

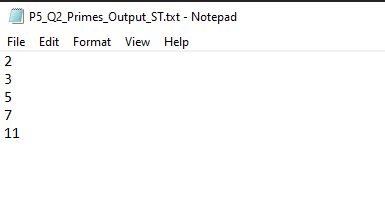
} // for ends

}

}

## Output:





# Fibonacci

## Question 03:

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8 … Formally, it can be expressed as: fib0 = 0, fib1 = 1, fibn = fib n-1 + fib n-2. Write a multithreaded program that generates the Fibonacci sequence using either the Java.

## Code:

import java.util.ArrayList;

import java.util.Scanner;

public class P5\_Q3\_Fibo\_ST {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

ArrayList al = new ArrayList();

System.out.print("Enter the number: ");

int a = scan.nextInt();

P5\_Q3\_FiboThread\_ST fibTh = new P5\_Q3\_FiboThread\_ST(a);

fibTh.start();

try {

fibTh.join();

} catch(InterruptedException ex) {

ex.printStackTrace();

}

int fseries[] = fibTh.arr;

System.out.println("First " + a + " fibonacci numbers are: ");

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

System.out.print(fseries[i] + " ");

}

} // main ends

}

class P5\_Q3\_FiboThread\_ST extends Thread {

private int a, i;

Thread t;

int arr[];

public P5\_Q3\_FiboThread\_ST(int a) {

this.a = a;

arr = new int[a];

}

public void run() {

arr[0] = 0;

arr[1] = 1;

for (i=2; i<a; i++){

arr[i] = arr[i-1] + arr[i-2];

}

} }

## Output:

