Contents

[USCSP301\_USCS303: OPERATING SYSTEM (OS) Practical 05 2](#_Toc79853142)

[Practical 05: THREADS 2](#_Toc79853143)

[(A)Practical Date : 14 August 2021 2](#_Toc79853144)

[(B)Practical Aim : Threads(Multi-Threading) 2](#_Toc79853145)

[(C) Threads States: Life Cycle of a Thread 2](#_Toc79853146)

[(D) Summation 3](#_Toc79853147)

[(E) Primes 6](#_Toc79853148)

[(F) Fibonacci 11](#_Toc79853149)

# USCSP301\_USCS303: OPERATING SYSTEM (OS) Practical 05

## Practical 05: THREADS

### (A)Practical Date : 14 August 2021

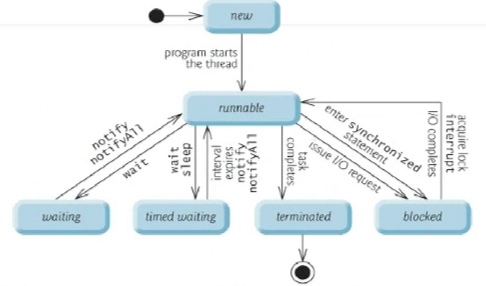
### (B)Practical Aim : Threads(Multi-Threading)

### (C) Threads States: Life Cycle of a Thread

A java thread can be in any of following thread states during it’s life cycle i.e.

1. New
2. Runnable
3. WAITING
4. Timed Waiting or Terminated

These are also called life cycle events of a thread in java.



1. **New and Runnable States:**
2. A new thread begins its life cycle in the new state.
3. It remains in this state until the program starts the thread, which places it in the runnable state;
4. A thread in the runnable state is considered to be executing the task.
5. **Waiting State:**
6. Sometimes a runnable thread transitions to the waiting state while it waits fir another thread to perform a task.
7. A waiting thread transitions back to the runnable satte only when another thread notifies it to continue executing.

**(iii)Timed Waiting State:**

1. 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 events it’s waiting for occurs.
2. Timed waiting and waiting threads cannot use a processor, even if one is available.
3. A runnable thread can transition to be 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.
4. 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.

**(iv)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.

**(v)Terminated 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).

### (D) Summation

**QUESTION 1:**

Write a multireaded 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.

//Name: Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 14 August, 2021.

//Practical 5: Threads

class P5\_Q1\_Summation\_RS implements Runnable

{

int upperLimit,sum;

public P5\_Q1\_Summation\_RS(int upperLimit)

{

this.upperLimit = upperLimit;

}

public void run()

{

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

sum +=1;

}

}//end of class P5\_Q1\_Summation\_RS

public class P5\_Q1\_SummationTest\_RS

{

public static void main(String args[])

{

if(args.length <= 0)

System.out.printIn("Usage: P5\_Q1\_SummationTest\_RS)

<integervalue>");

else

{

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

if (upp <= 0)

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

else

{

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

Thread t = new Thread(s);

t.start();

try {

t.join();

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

}

catch(Exception e) {

e.printStackTrace();

}

}

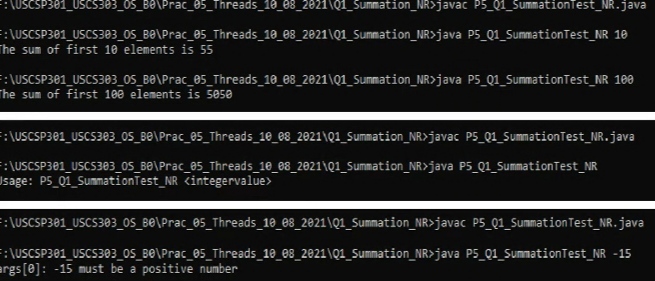
} // inner else ends

}// outer else ends

} // main ends

} // end of class class P5\_Q1\_SummationTest\_RS

**OUTPUT:**



### (E) Primes

**QUESTION 2:**

Write a multireaded java program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a seperate thread that outputs all the prime numbers less than or equal to the number entered by the user.

//Name: Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 14 August, 2021.

//Practical 5: Threads

import java.io.\*;

import.java.util.\*;

public class P5\_Q2\_Primes\_RS {

public static void main(String args[]) {

try {

P5\_Q2\_PrimesThread\_RS 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 results > ");

String fName = scan.next();

if (fName.length() > 0)

pt = new P5\_Q2\_PrimesThread\_RS(limit, new FileOutputStream(fName));

else

pt = new P5\_Q2\_PrimesThread\_RS(limit);

pt.run();

} catch (Exception e) {

e.printStackTrace();

}

} // main ends

} // class ends

//Name: Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 14 August 2021.

//Practical 5: Threads

import java.io.\*;

class P5\_Q2\_PrimeThread\_RS extends Thread {

private PrintStream pOut = null;

private int limit = 0;

// default constructor. does nothing

public P5\_Q2\_PrimeThread\_RS() {

}

// 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\_RS(int l) {

limit = l;

try {

p.Out = System.out;

} catch(Exception e) {

e.printStackTrace();

}

}

// contructor that set both the number, as above, and specifies an output stream

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

public P5\_Q2\_PrimeThread\_RS(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 prime 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

} // if ends

} // outer for ends

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

if(numbers[i])

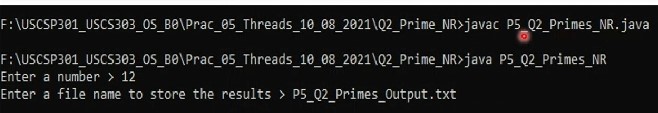
p.Out.printIn(i);

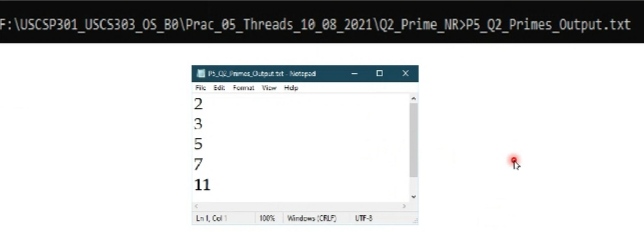
} // for ends

} // run ends

} // class ends

**INPUT:**

****

**OUTPUT:** ****

### (F) Fibonacci

QUESTION 3:

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

//Name: Ritika Sahu

//Batch : B1

//PRN:2020016400783543

//Date: 14 August 2021.

//Practical 5: Threads

import java.util.ArrayList;

import java.util.Scanner;

public class P5\_Q3\_Fibo\_RS

{

public static void main(String args[]) {

Scanner scan = new Scanner(System.in);

ArrayList al = new ArrayList();

int a;

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

a = scan.nextInt();

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

fibTh.start();

try {

fibTh.join();

} catch(InterruptedException ex) {

ex.printStackTrace();

}

int fseries[] = fibTh.arr;

System.out.printIn("First " + a + " fibonacc numbers are:");

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

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

}

} // main ends

} // class ends

class P5\_Q3\_FiboThread\_RS extends Thread

{

private int a,i;

Thread t;

int arr[];

public P5\_Q3\_FiboThread\_RS(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];

}

} // run ends

} // class ends

**OUTPUT:**

