**Content**

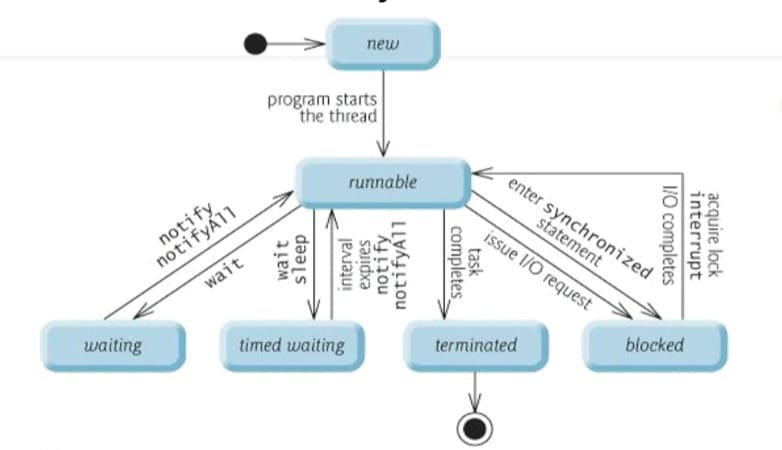
**USCSP301-USCS303:Operating System(OS)Practical-05**

**Practical-05: Thread**

**Practial Date: 14th Aug 2021**

**Practical Aim:Threads(Multi-Threading)**

* **Threads State:Lifecycle of Threads.......................................1**
* **Summation..........................................................................2**
* **Primes.................................................................................3**
* **Fibonacci.............................................................................4**
* **Content:**
* Creating and executing Threads using Runnable interface.
* **Process:**
* One can create and run multiple threads.
* Us of various methods of Thread class like start,run,wait,sleep,join,etc.
* **Prior Knowledge:**
* Multi-threaded programming concepts.
* Thread States:Life Cycle of a Thread
* A java thread can be in any of the following thread states during it’s life cycle i.e.
  + New,
  + Runnable,
  + Blocked,
  + Waiting,
  + Times Waiting or Terminated.
* These are also called life cycle events of a thread states.



# 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.

# 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.

# Timed Waiting State:

* A runnable thread can enter the timed waiting state for a specified interval of time. It transition 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.

# Blocked State:

* A runnable thread transition 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.

# 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).
* 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.

**#Source Code:**

FileName: P5\_Q1\_SummationTest\_SS.java

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 14TH Aug 2021

//PRAC-05: THREADS

class P5\_Q1\_Summation\_SS implements Runnable

{

int upperLimit,sum;

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

{

this.upperLimit=upperLimit;

}

public void run()

{

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

sum+=i;

}

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

public class P5\_Q1\_SummationTest\_SS

{

public static void main(String args[])

{

if(args.length <=0)

System.out.println("Usage: P5\_Q1\_SummationTest\_SS <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\_SS s=new P5\_Q1\_Summation\_SS(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();

}

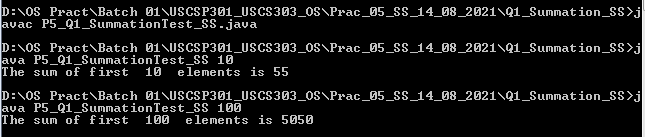
}//inner else ends

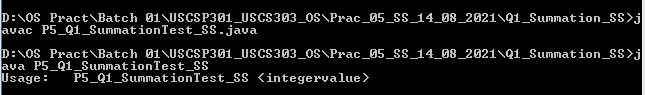
}//outer else ends

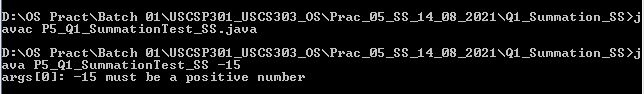
}//main ends

}//end of the class class P5\_Q1\_SummationTest\_SS

**Output:**







* Question-02:

Write a multithread Java program that output 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 end then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user.

**#Source Code 1:**

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 14TH Aug 2021

//PRAC-05: THREADS

import java.io.\*;

import java.util.\*;

public class P5\_Q2\_Primes\_SS{

public static void main(String args[]){

try{

P5\_Q2\_PrimeThread\_SS 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\_PrimeThread\_SS(limit, new FileOutputStream(fName));

else

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

pt.run();

}catch(Exception e){

e.printStackTrace();

}

}//main ends

}//class ends

**#Source code 2:**

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 14TH Aug 2021

//PRAC-05: THREADS

import java.io.\*;

class P5\_Q2\_PrimeThread\_SS extends Thread{

private PrintStream pOut= null;

private int limit= 0;

//default constructor.does nothing

public P5\_Q2\_PrimeThread\_SS(){

}

//construct to set the number below which to generate primes

public P5\_Q2\_PrimeThread\_SS(int I){

limit= I;

try {

pOut= System.out;

} catch(Exception e){

e.printStackTrace();

}

}

//consrtuct that sets both the number, as above, and specifies an output stream

public P5\_Q2\_PrimeThread\_SS(int I, OutputStream outS){

limit= I;

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

public void run(){

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++){

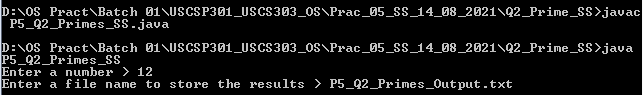
if (numbers[i])

pOut.println(i);

}//for ends

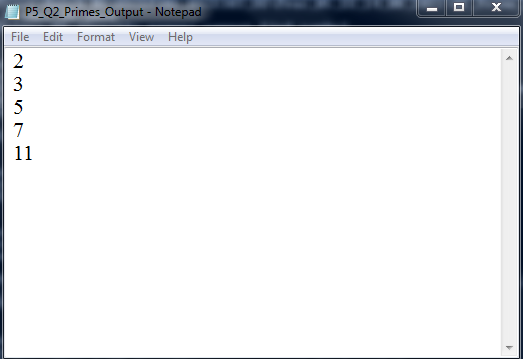
}//run ends

}//class ends

**Input:**  


Output:

Capture 2.PNG



* 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= fibn-1 + fibn-2. Write a multithreaded program that generates the Fibonacci sequence using either the Java.

**#Source code:**

//NAME: SHRADDHA SAWANT

//BATCH: B1

//PRN: 2020016400773862

//DATE: 14TH Aug 2021

//PRAC-05: THREADS

import java.util.ArrayList;

import java.util.Scanner;

public class P5\_Q3\_Fibo\_SS

{

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

fibTh.start();

try{

fibTh.join();

}catch(InterruptedException ex){

ex.printStackTrace();

}

int fseries[]= fibTh.arr;

System.out.println("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\_SS extends Thread

{

private int a, i;

Thread t;

int arr[];

public P5\_Q3\_FiboThread\_SS(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:

