Practicals assessment

1) **Give solution to the producer-consumer problem using shared memory**

```
package com.mycompany.mavenproject4;


import java .io.*;
import java .util.*;


public class Mavenproject4
{
public static void main(String []args)
{
CubbyHole c=new CubbyHole();
Producer p1=new Producer(c,1);
Consumer c1=new Consumer(c,1);
p1.start();
c1.start();
}
}
class CubbyHole
{
private int  Contents;
private boolean available=false;
```

```
public synchronized int get()
{
while(available==false)
{
try
{
wait();
}
catch(Exception e)
{
}
}
available=false;
notifyAll();
return Contents;
}
public synchronized void put(int value)
{
while(available==true)
{
try
{
wait();
}
catch(Exception e1)
{
}
```

```
}
Contents=value;
available=true;
notifyAll();
}
}
class Consumer extends Thread
{
private CubbyHole Cubbyhole;
private int number;
public Consumer(CubbyHole c,int number)
{
Cubbyhole=c;
this.number=number;
}
public void run()
{
int value=0;
for(int i=0;i<10;i++)
{
value=Cubbyhole.get();
System.out.println("Consumer#"+this.number+"got"+value);
}
}
}
class Producer extends Thread
{
```

```
private CubbyHole Cubbyhole;
private int number;
public Producer(CubbyHole c, int number)
{
Cubbyhole=c;
this.number=number;
}
public void run()
{
for(int i=0;i<10;i++)
{
Cubbyhole.put(i);
System.out.println("Producer"+this.number+"put"+i);
{
try
{
sleep((int)(Math.random()*100));
}
catch(Exception e)
{
} }}}}
```

OUTPUT:-

```
Consumer#1got0
Producer1put0
Consumer#1got1
Producer1put1
```

Consumer#1got2

Producer1put2

Consumer#1got3

Producer1put3

Consumer#1got4

Producer1put4

Producer1put5

Consumer#1got5

Producer1put6

Consumer#1got6

Producer1put7

Consumer#1got7

Producer1put8

Consumer#1got8

Producer1put9

Consumer#1got9

2) **write a java program of multithreaded that determines the summation of a non-negative integer.**

```
package com.mycompany.mavenproject4;


import java .io.*;
import java .util.*;


public class Mavenproject4{
   // Function to calculate the sum of numbers from 0 to n using multithreading
   public static long calculateSum(int n, int numThreads) throws
InterruptedException {
       // Array to hold the sum computed by each thread
       long[] partialSums = new long[numThreads];
       // Array to hold the thread objects
       SummationThread[] threads = new SummationThread[numThreads];


       // Calculate the portion size for each thread
       int portionSize = (n + numThreads - 1) / numThreads; // Ceiling division of
n by numThreads


       // Create and start threads
       for (int i = 0; i < numThreads; i++) {
          int start = i * portionSize;
          int end = Math.min(start + portionSize, n);
          threads[i] = new SummationThread(start, end, partialSums, i);
          threads[i].start();
       }
```

```
    // Wait for all threads to complete
    for (int i = 0; i < numThreads; i++) {
        threads[i].join();
    }

    // Calculate total sum from partial sums
    long totalSum = 0;
    for (long partialSum : partialSums) {
        totalSum += partialSum;
    }

    return totalSum;
}

// Thread class to compute sum for a portion of numbers
static class SummationThread extends Thread {
    private final int start;
    private final int end;
    private final long[] partialSums;
    private final int index;

    public SummationThread(int start, int end, long[] partialSums, int index) {
        this.start = start;
        this.end = end;
        this.partialSums = partialSums;
        this.index = index;
    }
```

```
        @Override
        public void run() {
            long sum = 0;
            for (int i = start; i < end; i++) {
                sum += i;
            }
            partialSums[index] = sum;
        }
    }

    public static void main(String[] args) {
        int n = 10; // Example: Calculate sum from 0 to 1000
        int numThreads = 4; // Example: Use 4 threads

        try {
            long result = calculateSum(n, numThreads);
            System.out.println("Sum from 0 to " + n + " is: " + result);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:-

```
Sum from 0 to 10 is: 45
```

3) write a java program for multi-thread that generate the fibonacci sequence.

```
package com.mycompany.mavenproject4;


import java.util.Scanner;


public class Mavenproject4 extends Thread {
       int n;
    int[] series;
 public Mavenproject4(int n) {
        this.n = n;
        series = new int[n];
    }
public void run() {
        series[0] = 0;
        series[1] = 1;
        for (int i = 2; i < n; i++) {
            series[i] = series[i-1] + series[i-2];
        }
    } public void printSeries() {
        for (int i = 0; i < n; i++) {
            System.out.print(series[i] + " ");
        }
    }public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements in the series: ");
        int n = sc.nextInt();
```

```
        Mavenproject4 ft = new Mavenproject4(n);
        ft.start();
        try {
            ft.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        ft.printSeries();
    }
}
```

OUTPUT:-

```
Enter the number of elements in the series: 20
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```