OOP LAB 6

1. Create a class MxNTableThread by extending Thread class. The thread calls a non- static printTable method of another class to display multiplication table of a number supplied as parameter. Create another class TablesDemo which will instantiate two objects of the MxNTableThread class to print multiplication table of 5 and 7. Observe intermixed output from the 2 threads. Also, observe output by applying synchronization concept.

Solution:

```
class P{
	void PrintTable(int x){
		for(int i=1;i<11;i++)
		System.out.println(x+" * "+i+" = "+i*x);
	}
}

class MxNTableThread extends Thread{
int count;
P table=new P();
MxNTableThread(String name,int x){
super(name);
count=x;
start();
}
public void run(){
table.PrintTable(count);
}
}
public class TableDemo{
public static void main(String[] args) {
MxNTableThread m5=new MxNTableThread("Table_of_5",5);
MxNTableThread m7=new MxNTableThread("Table_of_7",7);
}
}
```

```
$ javac TableDemo.java
$ java TableDemo
7 * 1 = 7
5 * 1 = 5
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
5 * 2 = 10
7 * 6 = 42
5 * 3 = 15
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
5 * 4 = 20
7 * 10 = 70
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
$
```

After applying synchronization concept, the code and corresponding output will be :

```
class P{
	void PrintTable(int x){
		for(int i=1;i<11;i++)
		System.out.println(x+" * "+i+" = "+i*x);
	}
}

class MxNTableThread extends Thread{
int count;
P table=new P();
MxNTableThread(String name,int x){
super(name);
count=x;
start();
```

```
}
synchronized public void run(){
table.PrintTable(count);
}
}
public class TableDemo{
public static void main(String[] args) {
MxNTableThread m5=new MxNTableThread("Table_of_5",5);
MxNTableThread m7=new MxNTableThread("Table_of_7",7);
}
}
```

```
$ javac TableDemo.java
$ java TableDemo
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
$
```

2. Write and execute a java program to create and initialize a matrix of integers. Create n threads(by implementing Runnable interface) where n is equal to the number of rows in the matrix. Each of these threads should compute a distinct row sum. The main thread computes the complete sum by looking into the partial sums given by the threads. Use join method to ensure that the main thread terminates last.

Solution:

```
import java.util.Scanner;
class rowsum implements Runnable{
        Thread thrd;
        int arr[],sum=0;
```

```
        rowsum(String name,int arr[]){
                thrd =new Thread(this,name);
                thrd.start();
                this.arr=arr;
        }
        public void run(){
                for(int i=0;i<arr.length;i++)
                        sum+=arr[i];
        }
}
public class l6e2{
        public static void main(String args[]){
        System.out.println("Enter the dimensions of the matrix");
        Scanner sc=new Scanner(System.in);
        int x=sc.nextInt();
        int y=sc.nextInt();
        int total=0;
        rowsum r[]=new rowsum[x];
        int arr[][]=new int[x][y];
        for(int i=0;i<x;i++){
                System.out.println("Enter the row "+(i+1)+" elements");
                for(int j=0;j<y;j++)
                        arr[i][j]=sc.nextInt();
        }
        for(int i=0;i<x;i++)
                r[i]=new rowsum("Thread "+(i+1),arr[i]);
        for(int i=0;i<x;i++)
                try{
                r[i].thrd.join();
        }
        catch(InterruptedException exc){
                System.out.println("Thread interrupted");
        }
        for(int i=0;i<x;i++)
                total+=r[i].sum;
        System.out.println("The total sum is "+total);
}
}
```

```
$ java l6e2
Enter the dimensions of the matrix
3 4
Enter the row 1 elements
1 2 3 4
Enter the row 2 elements
4 5 6 7
Enter the row 3 elements
2 3 4 5
The total sum is 46
```

3. Write and execute a java program to implement a producer - consumer problem using Inter-thread communication.

Solution:

```
class Q{
    int n;
    boolean valueset=false;
    synchronized int get(){
        while(!valueset){
            try{
                wait();
            }catch(InterruptedException exc){
                System.out.println("Thread interrupted");
            }
        }
        System.out.println("Got: "+n);
        valueset=false;
        notify();
        return n;
    }
    synchronized void put(int n){
        while(valueset){
            try{
                wait();
            }catch(InterruptedException exc){
                System.out.println("Thread interrupted");
            }
        }
        this.n=n;
        valueset=true;
        notify();
    }
}

class Producer implements Runnable{
    Q q;
```

```
    Producer(Q q){
        this.q=q;
        new Thread(this,"Producer").start();
    }
    public void run(){
        int i=0;
        while(true){
            q.put(i++);
        }
    }
}
class Consumer implements Runnable{
    Q q;
    Consumer(Q q){
        this.q=q;
        new Thread(this,"Consumer").start();
    }
    public void run(){
        while(true)
        q.get();
    }
}
public class Main
{
        public static void main(String[] args) {
                Q q=new Q();
                new Producer(q);
                new Consumer(q);
                System.out.println("Press ctrl+C to stop");
        }
}
```

```
Got: 2081
Got: 2082
Got: 2083
Got: 2084
Got: 2085
Got: 2086
Got: 2087
Got: 2088
Got: 2089
Got: 2090
Got: 2091
Got: 2092
Got: 2093
Got: 2094
Got: 2095
Got: 2096
Got: 2097
Got: 2098
Got: 2099
Got: 2100
Got: 2101
Got: 2102
Got: 2103
Got: 2104
Got: 2105
Got: 2106
Got: 2107
Got: 2108
Got: 2109
Got: 2110
Got: 2111
Got: 2112
Got: 2113
Got: 2114
Got: 2115
```