Shreyas Sawant
D7A - 55

AIM: To demonstrate multithreading in java

THEORY:

- Multithreading

Multithreading is a conceptual programming paradigm where a program (process) is divided into two or more subprograms (processes), which can be implemented at the same time in parallel. For eg, one subprogram can display an animation on the screen while another may build the next animation to be displayed. This is something similar to dividing a task into subtasks and assigning them to different people for execution independently and simultaneously.

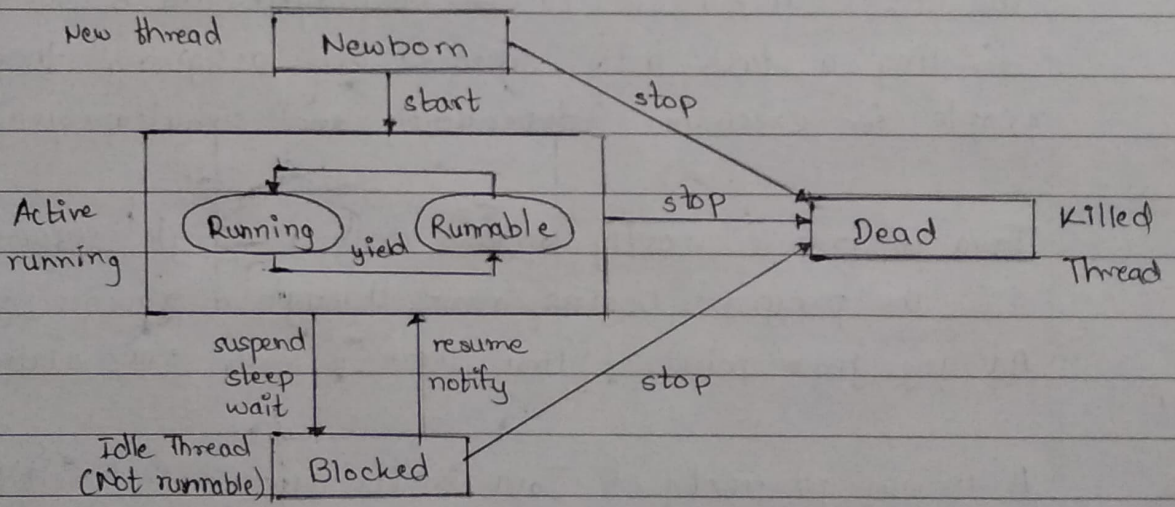
Java programme mostly contain only a single sequential flow of control i.e. the program begins, runs through a sequence of execution and ends. At any given point of time, there is only one statement under execution.

A unique property of Java is its support for multithreading. Each flow of control may be thought as a seperate tiny program known as thread that runs in parallel to others. However 'threads running in parallel' does not really mean that they actually run at the same time. Since all threads are running on a single processor, the flow is shared between the threads. The Java interpreter handles the switching of control between the threads in such a way that it appears they are running concurrently.

- **Life Cycle of a Thread**

During the lifetime of a thread, there are many states it can enter. They include:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state



State transition diagram of a thread.

1 **Newborn State:**

When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running.

2 **Runnable State:**

The runnable state means that the thread is ready for execution and is waiting for availability of the processor. That is, the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots

for execution in round robin fashion, i.e., first come, first serve. The thread that relinquishes control joins the queue at the end and again waits for its turn. The process of assigning time to threads is known as time-slicing.

3) Running State

Running means that the processor has given its time to the thread for its execution. The thread runs until its relinquishes control on its own or it is preempted by a higher priority thread.

4) Blocked State

A thread is said to be blocked when its is prevented from entering into runnable state and subsequently the running state. This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements. It is fully qualified to run again.

5) Dead State

Every thread has a life cycle. A running thread ends its life when it has completed executing its run() method. and thus it is a natural death. However we can kill it by sending the stop message at any state thus causing premature death.

- Methods used in Multithreading

1) start()

It is used to start the execution of the thread.

2) run()

It is used to do an action for a thread

3) static void sleep()
It sleeps a thread for the specified amount of time.

4) resume()
It is used to resume the suspended thread

5) suspend()
It is used to suspend the thread

6) stop()
It is used to stop the thread

7) wait()
It is used to wait the thread until some event occurs.

8) notify()
It is used to give the notification for only one thread which is waiting.

9) static void yield()
It causes the currently executing thread object to pause and allow other threads to execute temporarily

10) static Thread currentThread()
It returns a referenee to currently executing thread object.

11) setPriority.
It changes the priority of thread

12) int getPriority()
It returns the priority of the thread

13) long getId()
It returns the id of the thread

CONCLUSION:

Errors encountered:

1. Incorrect syntax:

```
while (k!==20)
{ ... }
```

Solution Correct syntax:

```
while (k!=20)
{ ... }
```

2. Incorrect syntax for defining anonymous class

```
Thread t2 = new Thread() {
    .....
}
```

Solution Correct syntax:

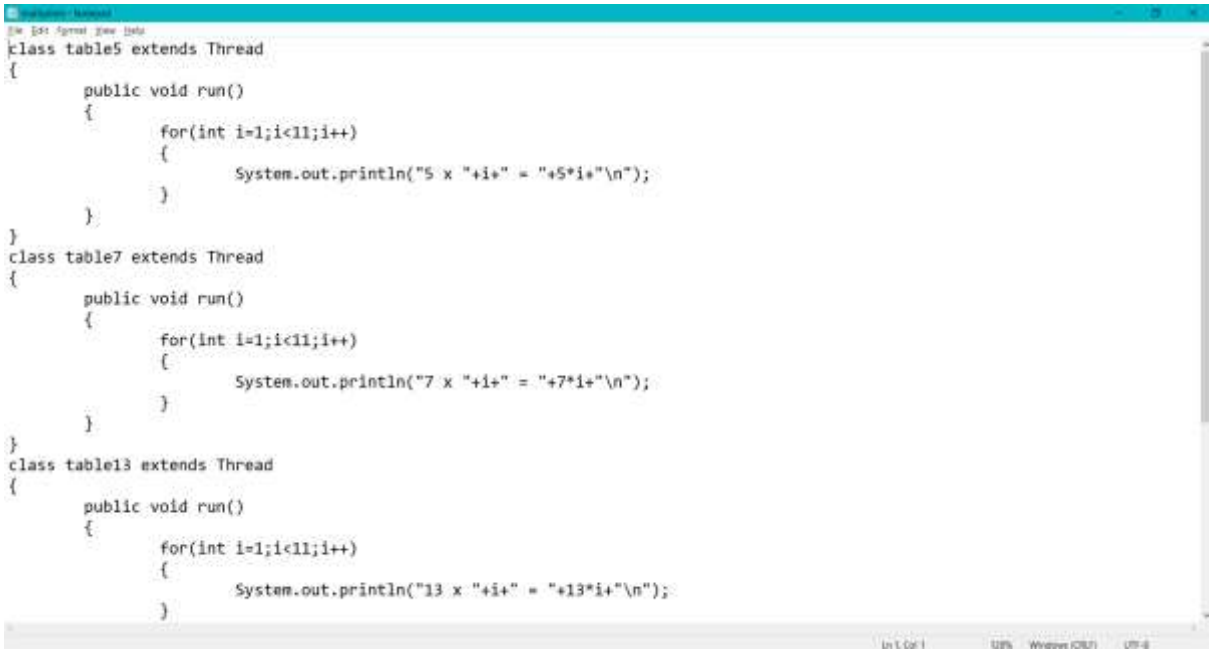
```
Thread t2= new Thread() {
    ....
};
```

LAB 10: MULTITHREADING IN JAVA

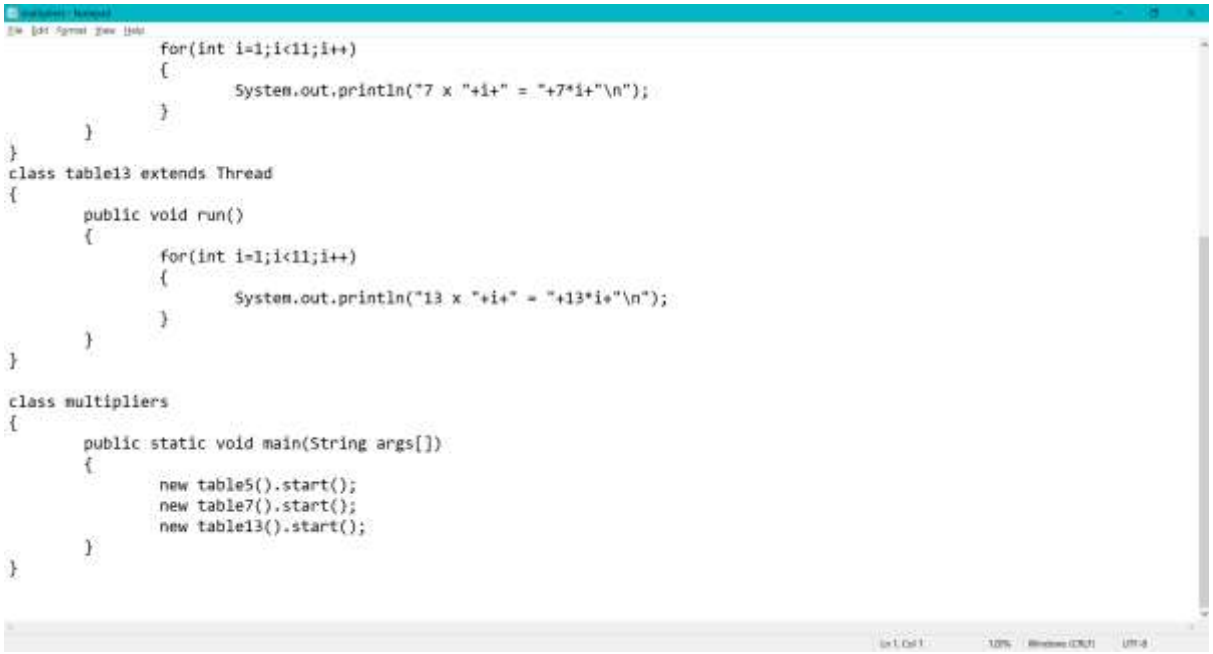
Name: Shreyas Sawant Div: D7A Roll No.: 55

Q.1 Write a program to print the table of 5,7,13 using multithreading (use thread class)

CODE:

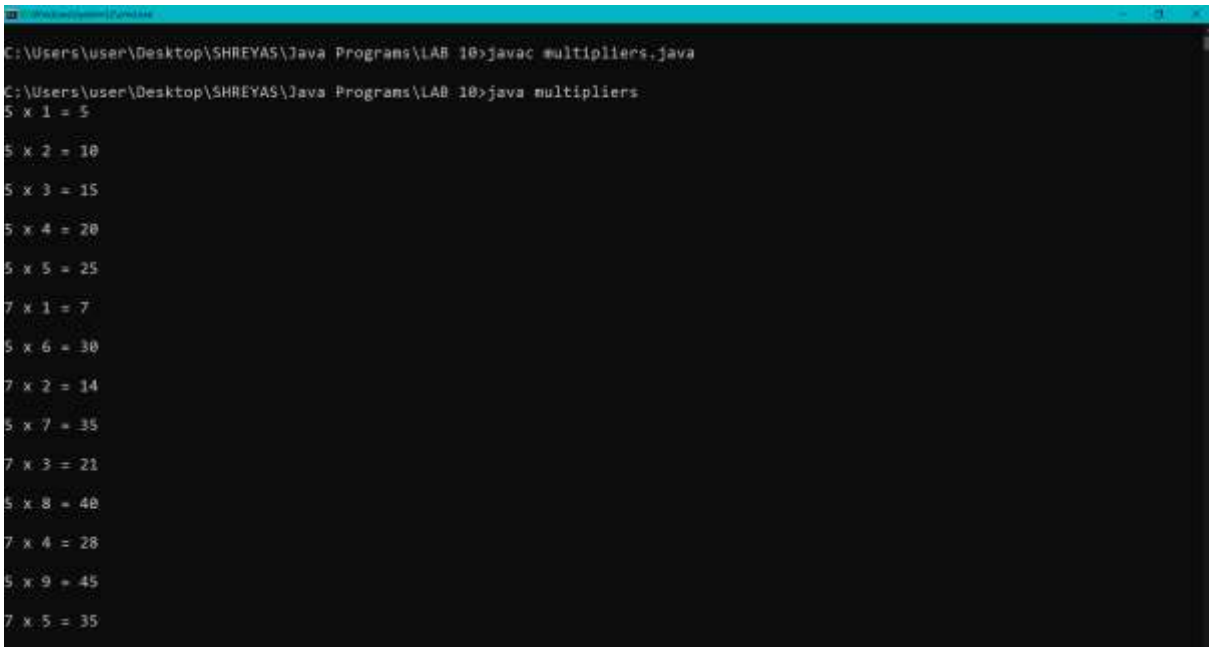


```
class table5 extends Thread
{
	public void run()
	{
		for(int i=1;i<11;i++)
		{
			System.out.println("5 x "+i+" = "+5*i+"\n");
		}
	}
}
class table7 extends Thread
{
	public void run()
	{
		for(int i=1;i<11;i++)
		{
			System.out.println("7 x "+i+" = "+7*i+"\n");
		}
	}
}
class table13 extends Thread
{
	public void run()
	{
		for(int i=1;i<11;i++)
		{
			System.out.println("13 x "+i+" = "+13*i+"\n");
		}
```



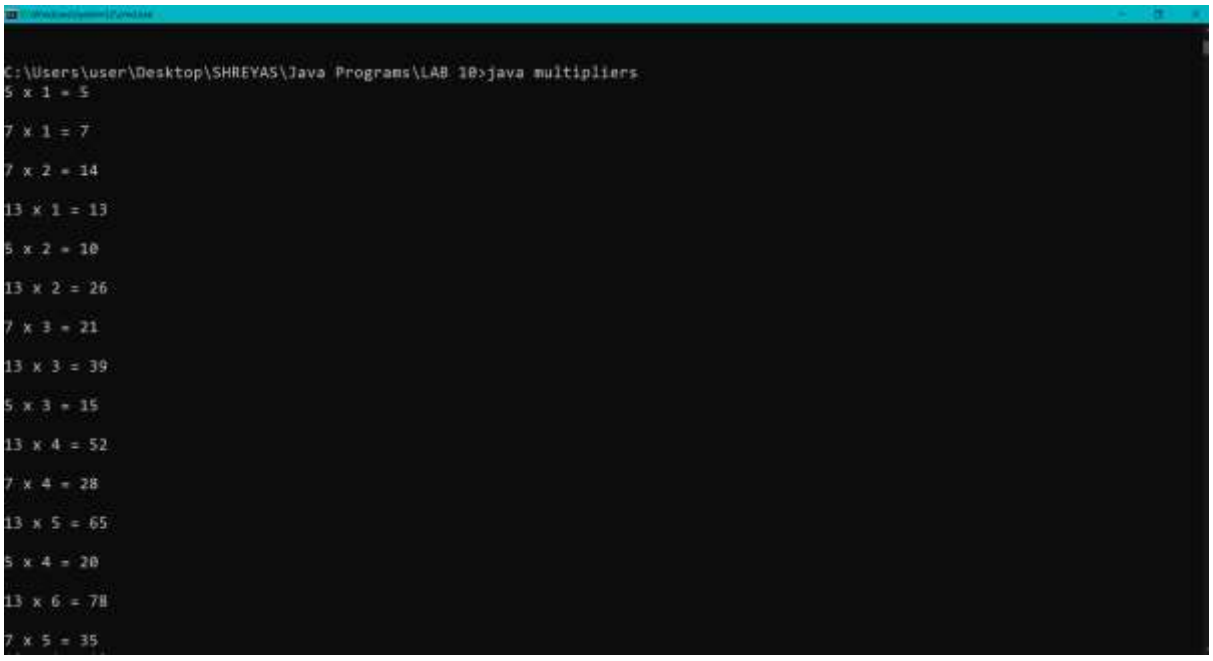
```
		for(int i=1;i<11;i++)
		{
			System.out.println("7 x "+i+" = "+7*i+"\n");
		}
	}
}
class table13 extends Thread
{
	public void run()
	{
		for(int i=1;i<11;i++)
		{
			System.out.println("13 x "+i+" = "+13*i+"\n");
		}
	}
}

class multipliers
{
	public static void main(String args[])
	{
		new table5().start();
		new table7().start();
		new table13().start();
	}
}
```

OUTPUT:


C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>javac multipliers.java
C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>java multipliers
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
7 x 1 = 7
5 x 6 = 30
7 x 2 = 14
5 x 7 = 35
7 x 3 = 21
5 x 8 = 40
7 x 4 = 28
5 x 9 = 45
7 x 5 = 35


7 x 5 = 35
13 x 1 = 13
5 x 10 = 50
13 x 2 = 26
7 x 6 = 42
13 x 3 = 39
7 x 7 = 49
13 x 4 = 52
7 x 8 = 56
13 x 5 = 65
7 x 9 = 63
13 x 6 = 78
7 x 10 = 70
13 x 7 = 91
13 x 8 = 104
13 x 9 = 117


5 x 10 = 50
13 x 2 = 26
7 x 6 = 42
13 x 3 = 39
7 x 7 = 49
13 x 4 = 52
7 x 8 = 56
13 x 5 = 65
7 x 9 = 63
13 x 6 = 78
7 x 10 = 70
13 x 7 = 91
13 x 8 = 104
13 x 9 = 117
13 x 10 = 130


C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>java multipliers
5 x 1 = 5
7 x 1 = 7
7 x 2 = 14
13 x 1 = 13
5 x 2 = 10
13 x 2 = 26
7 x 3 = 21
13 x 3 = 39
5 x 3 = 15
13 x 4 = 52
7 x 4 = 28
13 x 5 = 65
5 x 4 = 20
13 x 6 = 78
7 x 5 = 35


13 x 7 = 91
5 x 5 = 25
13 x 8 = 104
7 x 6 = 42
13 x 9 = 117
5 x 6 = 30
13 x 10 = 130
7 x 7 = 49
5 x 7 = 35
7 x 8 = 56
5 x 8 = 40
7 x 9 = 63
5 x 9 = 45
7 x 10 = 70
5 x 10 = 50
C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>

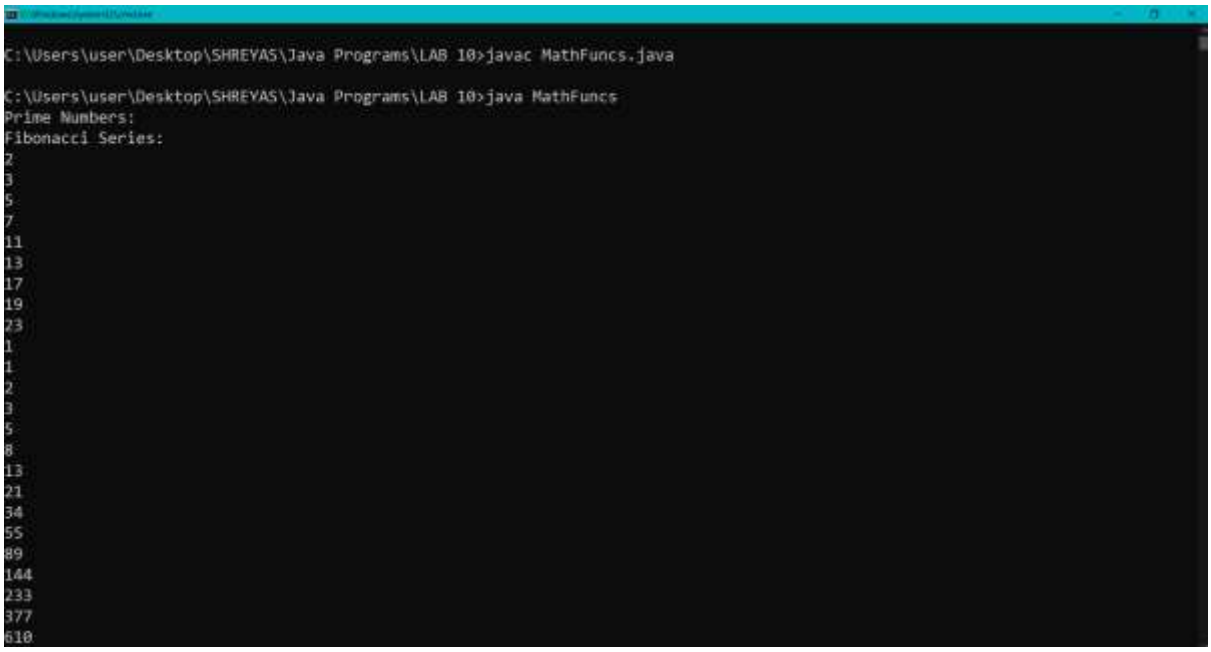
Q.2 Write a program to print first 20 prime numbers and 15 fibonacci terms using runnable interface.

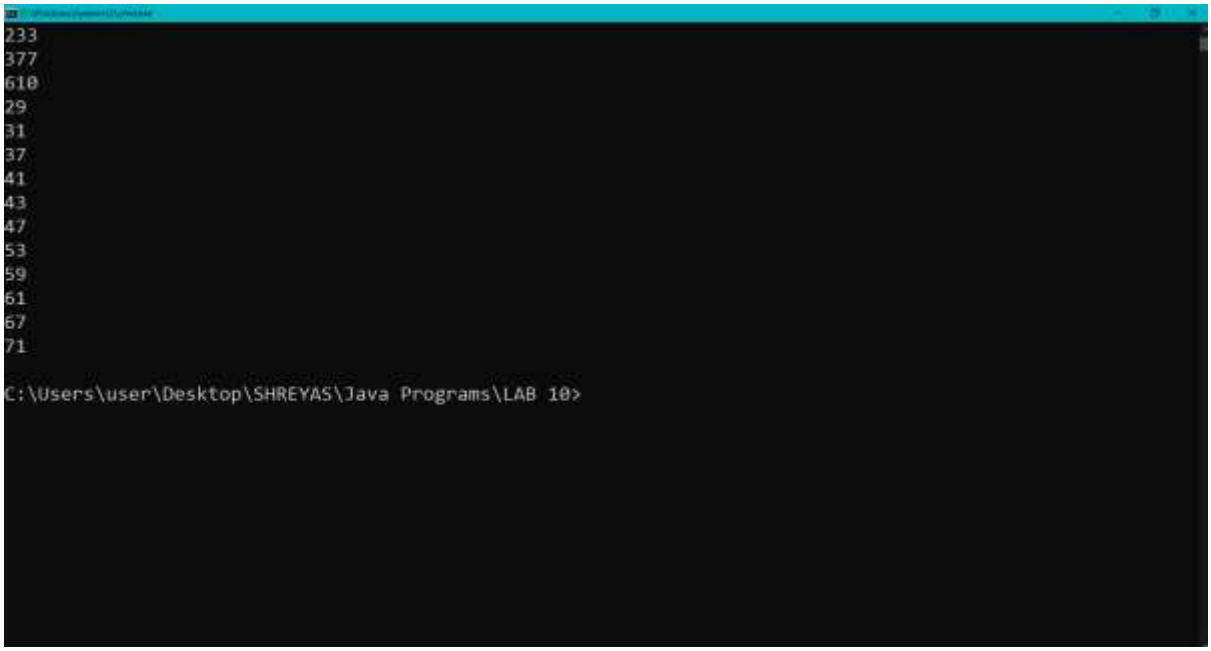
CODE:

```
class Prime implements Runnable
{
	public void run()
	{	System.out.println("Prime Numbers: ");
		int k=0;int i=2;
		while(k!=20)
		{
			if(prime(i))
			{ System.out.println(i);
				k++;
			}
			i++;
		}
	}

	boolean prime(int a)
	{	int k=0;
		for(int i=1;i<a;i++)
		{	if(a%i==0)
				k++;
		}
		if(k==1)
			return true;
		else
			return false;
	}
}
class Fibs implements Runnable
```

```
class Fibs implements Runnable
{
	public void run()
	{
		System.out.println("Fibonacci Series: ");
		int a=0,b=1,c,k=0;
		System.out.println(a+b);
		while(k!=14)
		{
			c=a+b;
			a=b;
			b=c;
			System.out.println(c);
			k++;
		}
	}
}
class MathFuncs
{
	public static void main(String args[])
	{
		Prime runnable=new Prime();
		Fibs runnable1=new Fibs();
		new Thread(new Prime()).start();
		new Thread(new Fibs()).start();
	}
}
```

OUTPUT:


C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>javac MathFuncs.java
C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>java MathFuncs
Prime Numbers:
Fibonacci Series:
2
3
5
7
11
13
17
19
23
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610


233
377
610
29
31
37
41
43
47
53
59
61
67
71
C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>

Q.3. Write a program to demonstrate concept of synchronisation

CODE:

Non-synchronised method

```
class Table
{
        void printTable(int n)
        {       System.out.println("Table of "+n+": ");
                for(int i=1;i<=5;i++)
                        System.out.println(n*i);
        }
}
public class NonSynchro
{
        public static void main(String args[])
        {
                Table obj = new Table();
                Thread t1=new Thread(){
                public void run(){
                        obj.printTable(5);
                }
                };
                Thread t2=new Thread(){
                public void run(){
                        obj.printTable(13);
                }
                };
                t1.start();
                t2.start();
        }
}
```

OUTPUT:

```
C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>java NonSynchro
Table of 5:
Table of 13:
13
26
39
52
65
5
10
15
20
25

C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>java NonSynchro
Table of 13:
Table of 5:
5
10
15
20
25
13
26
39
52
65

C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>
```

CODE:

Synchronised method

```
class Table
{
        synchronized void printTable(int n)
        {       System.out.println("\nTable of "+n+": ");
                for(int i=1;i<=10;i++)
                        System.out.println(n*i);
        }
}

public class Synchro
{
        public static void main(String args[])
        {
                Table obj = new Table();
                Thread t1=new Thread(){
                public void run(){
                        obj.printTable(5);
                }
                };
                Thread t2=new Thread(){
                public void run(){
                        obj.printTable(13);
                }
                };
                t1.start();
                t2.start();
        }
}
```

OUTPUT:

```
C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>javac Synchro.java

C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>java Synchro

Table of 5:
5
10
15
20
25
30
35
40
45
50

Table of 13:
13
26
39
52
65
78
91
104
117
130

C:\Users\user\Desktop\SHREYAS\Java Programs\LAB 10>
```