St Francis Institute of Technology, Mumbai-400 103

Class: SE-ITA/ITB Semester: III; A.Y. 2020-2021
Subject: Java Labs

Title-7: Java Program to implement Multithreading and Thread Synchronization.

1. **Aim:**
   i. Create two threads such that one thread will print even number and another will print odd number in an ordered fashion.(Use Thread Class)
   ii. Write java program to print Table of Five, Seven and Thirteen using Multithreading(Use Runnable Interface)

2. **Prerequisite:** Knowledge of Multithreading in Java.

3. **Requirements:** Personal Computer (PC), Windows Operating System, Net beans 8.0.

4. **Pre-Experiment Exercise:**
   Theory:

   a. **Multithreading:**
   Java's multithreading support is centered around the java.lang.Thread class. The Thread class provides the capability to create objects of class Thread, each with its own separate flow of control. The Thread class encapsulates the data and methods associated with separate threads of execution and allows multithreading to be integrated within the object-oriented framework. Java provides two approaches to creating threads. In the first approach, you create a subclass of class Thread and override the run() method to provide an entry point into the thread's execution. When you create an instance of your Thread subclass, you invoke its start() method to cause the thread to execute as an independent sequence of instructions. The start() method is inherited from the Thread class. It initializes the Thread object using your operating system's multithreading capabilities and invokes the run() method.

   Java's other approach to creating threads does not limit the location of your Thread objects within the class hierarchy. In this approach, your class implements the java.lang.Runnable interface. The Runnable interface consists of a single method, the run() method, which must be overridden by your class. The run() method provides an entry point into your thread's execution. In order to run an object of your class as an independent thread, you pass it as an argument to a constructor of class Thread.

   b. **Thread Synchronization**

   There are times when you might want to coordinate access to shared resources. For example, in a database system, you might not want one thread to be updating a database record while another thread is trying to read it. Java enables you to coordinate the actions of multiple threads using *synchronized methods* and *synchronized statements*.

An object for which access is to be coordinated is accessed through the use of synchronized methods. These methods are declared with the synchronized keyword. Only one synchronized method can be invoked for an object at a given point in time. This keeps synchronized methods in multiple threads from conflicting with each other.

All classes and objects are associated with a unique *monitor*. The monitor is used to control the way in which synchronized methods are allowed to access the class or object. When a synchronized method is invoked for a given object, it is said to *acquire* the monitor for that object. No other synchronized method may be invoked for that object until the monitor is released. A monitor is automatically released when the method completes its execution and returns. A monitor may also be released when a synchronized method executes certain methods, such as wait(). The thread associated with the currently executing synchronized method becomes not runnable until the wait condition is satisfied and no other method has acquired the object's monitor.

5. Laboratory Exercise

A. Procedure

i. Open Net beans for Java.
ii. Open File and Create New Java Project.
iii. Inside the Java Project rename give name to your Java Class.
iv. Click on Finish.
v. Type the Java Code in the opened class.
vi. Save the code by pressing Ctrl+S.
vii. Run the code by pressing Shift+F6.

B. Program code with comments:

Write and execute your program code to achieve the given aim and attach it **with your own comments with neat indentation**.

6. Post-Experiments Exercise

A. Extended Theory:

1. Explain deadlock situation and how can it be eliminated.
2. Explain with diagram Java Thread Model and Thread Lifecycle Model.

B. Results/Observations/Program output:

Present the program input/output results and comment on the same.

C. Questions/Programs:

1. Write java program to implement the concept of Thread Synchronization.

D. Conclusion:

1. Write what was performed in the experiment/program.
2. What is the significance of experiment/program?
3. Mention few applications of what was studied.

E. References

1. Balguruswamy, “Programming with java A primer”, Fifth edition, Tata McGraw Hill Publication.
2. Let Us Java-Yashwant Kanetkar.
3. Learn to Master JAVA, from Star EDU solutions , by ScriptDemics.
4. Java 8 Programming-Black Book,by-Dreamtech Publications.
5. www.programmingsimplified.com
6. www.javatpoint.com

Program 1:

```
//Create two threads such that one thread will print even number and another
//will print odd number in an ordered fashion.(Use Thread Class)

class even extends Thread
{
    public void run(){
      for(int i=0;i<=50;i=i+2)
    {
    System.out.println("Thread 1:"+i);
    }
    }

}
class odd extends Thread
{
    public void run()
    {
      for(int i=1;i<50;i=i+2)
    {
    System.out.println("Thread 2:"+i);
    }

    }
}
public class Exp7_1 {
    public static void main(String[] args){
    even e=new even();
    odd o=new odd();
    e.start();
    o.start();

    }

}
```

Output:

```
D:\College\JAVA\Experiments\Exp7>java Exp7_1
Thread 1:0
Thread 2:1
Thread 1:2
Thread 2:3
Thread 1:4
Thread 2:5
Thread 1:6
Thread 1:8
Thread 1:10
Thread 2:7
Thread 2:9

D:\College\JAVA\Experiments\Exp7>
```

Program 2:

```
//Write java program to print Table of Five, Seven and Thirteen using
//Multithreading(Use Runnable Interface)

class table implements Runnable{
    int n;
    table(int n){
        this.n=n;
    }
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.printf("%s: %d * %d = %d\n",
Thread.currentThread().getName(),n, i, i * n);        }
    }
}
public class Exp7_2 {
    public static void main(String[] args) {
        System.out.println("Printing table of 5,7,13");
```

```
        table mul1 = new table(5);
        Thread t1= new Thread(mul1);
        table mul2 = new table(7);
        Thread t2= new Thread(mul2);
        table mul3 = new table(13);
        Thread t3= new Thread(mul3);
        t1.start();
        t2.start();
        t3.start();

    }
}
```

Output:

```
D:\College\JAVA\Experiments\Exp7>javac Exp7_2.java

D:\College\JAVA\Experiments\Exp7>java Exp7_2
Printing table of 5,7,13
Thread-0: 5 * 1 = 5
Thread-0: 5 * 2 = 10
Thread-0: 5 * 3 = 15
Thread-1: 7 * 1 = 7
Thread-1: 7 * 2 = 14
Thread-2: 13 * 1 = 13
Thread-2: 13 * 2 = 26
Thread-2: 13 * 3 = 39
Thread-1: 7 * 3 = 21
Thread-0: 5 * 4 = 20
Thread-0: 5 * 5 = 25
Thread-0: 5 * 6 = 30
Thread-0: 5 * 7 = 35
Thread-1: 7 * 4 = 28
Thread-1: 7 * 5 = 35
Thread-2: 13 * 4 = 52
Thread-2: 13 * 5 = 65
Thread-1: 7 * 6 = 42
Thread-0: 5 * 8 = 40
Thread-1: 7 * 7 = 49
Thread-2: 13 * 6 = 78
Thread-1: 7 * 8 = 56
Thread-0: 5 * 9 = 45
Thread-0: 5 * 10 = 50
Thread-1: 7 * 9 = 63
Thread-1: 7 * 10 = 70
Thread-2: 13 * 7 = 91
Thread-2: 13 * 8 = 104
Thread-2: 13 * 9 = 117
Thread-2: 13 * 10 = 130

D:\College\JAVA\Experiments\Exp7>
```

Questions:

Question 1:

```
//Write java program to implement the concept of Thread Synchronization.
class List{

    synchronized public void display(int[] arr){ //synchronized method
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]);
            try {
                Thread .sleep( 100 );
            }
            catch ( Exception e){ System .out.println(e);}
        }
    }
}
//Class myThread 2 for printing array 0,1,2,3,4,5
class myThread1 extends Thread{

    List l;
    private int[] myNum = {0,1,2,3,4,5};
    myThread1(List l){
        this.l = l;
    }
    public void  run(){
        l.display(myNum);
    }
}
//Class myThread 2 for printing array 10,11,12,13,14,15
class myThread2 extends Thread{

    List l;
    private int[] myNum = {10,11,12,13,14,15};
    myThread2(List l){
        this.l = l;
    }
    public void  run(){
        l.display(myNum);
    }
```

```java
}

public class Questions{

    public static void main(String[] args) {

        List l = new List();//only one object

        myThread1 t1 = new myThread1(l);
        myThread2 t2 = new myThread2(l);

        t1.start();
        t2.start();
    }
}
```

Output:

```
D:\College\JAVA\Experiments\Exp7>javac Questions.java

D:\College\JAVA\Experiments\Exp7>java Questions
0
1
2
3
4
5
10
11
12
13
14
15

D:\College\JAVA\Experiments\Exp7>
```

Yash Mahajan SE IT B 04

6) Post Experiment Exercise :-

A] Extended Theory

(1) Explain deadlock situation & how can it be eliminated.

Deadlock is a situation where a set of processes are blocked because each process is holding a resource & waiting for another resource acquired by some other process.
For eg, In OS when there are two or more resources that hold some resource & wait for resources held by other.
We can prevent deadlock by eliminating the condition.
a) Eliminate mutual exclusion.
It is not possible to dis-satisfy the mutual exclusive
a) Avoid nested locks :- You must avoid giving locks to multiple threads, this is the main reason for deadlock condition.
b) Avoid unnecessary locks - The locks should be given to the important threads. Giving locks to unnecessary threads - that cause the deadlock condition.
c) Using thread join - Deadlock usually happens when one thread is waiting for the other to finish. In this case, we can use Thread.join with a maximum time that a thread will take.

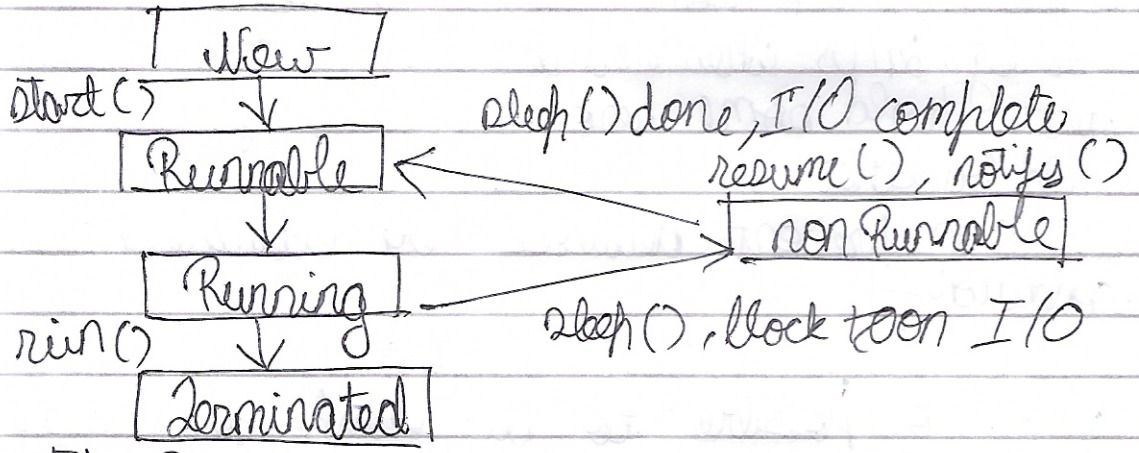
Page No :
Date :

Yash Mahajan SE IT B 04

2) Explain with diagram Java thread model and thread life cycle model.

The Java language & it's runtime system was designed keeping in mind about multi-threading. Java provides synchronous thread environment this helps to increase the utilization of CPU.

Life cycle of a thread :-



(i) New :- A new thread begins in its cycles in new state. It remains there until program starts the thread.

(ii) Runnable :- After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing tasks.

Yash Mahajan SE IT B 04

(iii) Waiting/Blocked :- Sometimes a thread transitions to waiting state while the thread waits for another thread to perform a task.

(iv) Terminated :- A runnable thread enters the terminated state when it completes its task or otherwise terminates.

D] Conclusion :-

In this experiment we have implemented programs related to Multithreading and Thread Synchronization.

Multithreading can be used to increase parallelism to make most of the available CPU resources & to improve application responsiveness. Java also provides thread synchronization to control access to shared resources and maintain consistency of data.

A web server will utilize multiple threads to simultaneously process requests for data at the same time. Multithreading also leads to minimizing & more efficient use of computing resources.