

Name : Shreyash Shete

Batch : May 2022

Advanced Java Assignments

Day #1: Multithreading

Objective: At the end of the assignments, participants will be able to write multi threaded programs using synchronization.

Estimated time: 3 Hours

1. Write a Java program to demonstrate that as a high-priority thread executes, it will delay the execution of all lower-priority threads.

Class DemoThread :

```
public class DemoThread extends Thread {  
  
    private String str;  
  
    public DemoThread(String str) {  
        this.str = str;  
    }  
  
    public void run() {  
  
        for(int i=1;i<=5;i++) {  
            System.out.println("Hello" + str + "\t" + i);  
  
            try {  
                Thread.sleep(1000);  
            }  
            catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

Class DemoThreadMain :

```
public class DemoThreadMain {

    public static void main(String[] args) {

        DemoThread d1 = new DemoThread("Java");
        DemoThread d2 = new DemoThread("Oracle");

        d1.setName("Java");
        d2.setName("Oracle");

        System.out.println(d1);
        System.out.println(d2);

        System.out.println("Thread name is : " + d1.getName());
        System.out.println("Thread name is : " + d2.getName());

        System.out.println(Thread.MIN_PRIORITY); //default 1
        d1.setPriority(Thread.NORM_PRIORITY); //default 5
        d2.setPriority(Thread.MAX_PRIORITY); // default 10

        d1.start();
        d2.start();

        for(int i=1;i<=5;i++) {
            System.out.println("Main" + i);
            try {
                d1.join();
                d2.join();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. Write a Java program that demonstrates a high-priority thread using sleep to give lowerpriority threads a chance to run.

Class ThreadExample :-

```
// Importing the required classes
```

```
public class ThreadExample extends Thread
{
    public void run()
    {
        System.out.println("Inside the run() method");    //Priority
    }

    public static void main(String[] args)
    {
        ThreadExample th1 = new ThreadExample();
        ThreadExample th2 = new ThreadExample();
        ThreadExample th3 = new ThreadExample();

        System.out.println("Priority of the thread th1 is : " +
th1.getPriority());
        System.out.println("Priority of the thread th2 is : " +
th2.getPriority());
        System.out.println("Priority of the thread th2 is : " +
th2.getPriority());

        th1.setPriority(6);
        th2.setPriority(3);
        th3.setPriority(9);
        th3.run();

        System.out.println("Priority of the thread th1 is : " +
th1.getPriority());
        System.out.println("Priority of the thread th2 is : " +
th2.getPriority());
        System.out.println("Priority of the thread th3 is : " +
th3.getPriority());
    }
}
```

```
        System.out.println("Currently Executing The Thread : " +  
Thread.currentThread().getName());  
        System.out.println("Priority of the main thread is : " +  
Thread.currentThread().getPriority());  
  
        Thread.currentThread().setPriority(10);  
  
        System.out.println("Priority of the main thread is : " +  
Thread.currentThread().getPriority());  
        System.out.println("This is main thread---");  
    }  
}
```

3. Implement three classes: Storage, Counter and Printer.

The Storage class should store an integer.

The Counter class should create a thread and starts counting from 0 (0,1,2,3...) and stores each value in the Storage class.

The Printer Class should create a thread that keeps reading the value in the Storage class and printing it.

Write a program that creates an instance of the Storage class and set up a Counter and Printer object to operate on it.

Identify that, whether synchronization is required or not in this assignment. If yes, implement it.

Class Storage :

```
public class Storage {  
  
    int i;  
    boolean printed = true;  
    public void setValue(int i){  
        this.i = i;  
    }  
    public int getValue(){  
        return this.i;  
    }  
    public boolean isPrinted() {  
        return printed;  
    }  
    public void setPrinted(boolean p) {  
        printed = p;  
    }  
}
```

Class Counter :

```
public class Counter implements Runnable {

    Storage st;
    public Counter(Storage store){
        st = store;
    }

    public void run() { // Run Method
        synchronized(st) {
            for(int i = 0 ; i < 10; i++){
                while(!st.isPrinted()) { //loop to take care
of spontaneous wake-ups
                    try {
                        st.wait();
                    } catch(Exception e) { }
                }
                st.setValue(i);
                st.setPrinted(false);
                st.notify();
            }
        }
    }
}
```

Class Printer :

```
public class Printer implements Runnable {

    Storage st;
    public Printer(Storage store){
        st = store;
    }

    public void run() {          // Run method
        synchronized(st) {
            for(int i = 0; i < 10; i++) {
                while(st.isPrinted()) {           //loop to take
care of spontaneous wake-ups
                    try {
                        st.wait();
                    } catch(Exception e) { }
                }
            }

            System.out.println(Thread.currentThread().getName() + " " +
st.getValue());
            st.setPrinted(true);
            st.notify();
        }
    }
}
```

MainClass :

```
public class MainClass {

    public static void main(String[] args) {
        Storage st = new Storage();
        Counter c = new Counter(st);
        Printer p = new Printer(st);
        new Thread(c, "Counter").start(); //start the counter
        new Thread(p, "Printer").start(); //start the printer
    }
}
```

4. Write a multithreaded program that will accept 4 strings from the command line and search in a particular file for a given string and display the status of each search on the screen.

Note that, all threads are operating on the same file.

Class FileScanner :

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.lang.Runnable;

public class FileScanner implements Runnable {

    private String filename;
    private String[] targetStrings;
    private String monikerName;

    public FileScanner(String filename, String[] targetStrings,
String monikerName) { // Constructor
        super();
        this.filename = filename;
        this.monikerName = monikerName;
        this.targetStrings = new String[4];

        for (int iLoop=0; iLoop<4; iLoop++)
        {
            this.targetStrings[iLoop] = new
String(targetStrings[iLoop]);
        }
    }

    public void run() {
```



```

        try
        {
            String content = new
String(Files.readAllBytes(Paths.get(filename)));
            System.out.println(content);

            int strlen = content.length();
            for (int iLoop=0; iLoop<4; iLoop++)
            {
                int iIndexPos= -1;
                while (iIndexPos<strlen)
                {
                    int iTemp =
content.indexOf(targetStrings[iLoop],iIndexPos+1);

                    if (iTemp>=0)
                    {
                        System.out.println(this.monikerName +
" : target string " + targetStrings[iLoop] + " at index " + iTemp);
                        iIndexPos = iTemp;
                    }
                    else
                    {
                        break;
                    }
                }
            }
        }
        catch (IOException ex) { System.out.println("IO Exception
"); }
    }
}

```

```

static class FileThread {

    public static void main(String[] args) {

        if (args.length!=6)
        {
            System.out.println(" java FileThread filename1
filename1 targetStr1 targetStr2 targetStr3 targetStr4");
        }
        else

```

```
{
    String targetStrings[] = new String[4];

    targetStrings[0] = new String(args[2]);
    targetStrings[1] = new String(args[3]);
    targetStrings[2] = new String(args[4]);
    targetStrings[3] = new String(args[5]);

    FileScanner fileScanner1 = new
FileScanner(args[0],targetStrings,"scanner1");
    FileScanner fileScanner2 = new
FileScanner(args[1],targetStrings,"scanner2");
    fileScanner1.run();
    fileScanner2.run();
}
}
}
```

5. Write a Java application that will accept two filenames (text files) as command line arguments and use two threads to read contents from the two text files. Each of the threads should sleep for a random time after displaying filename with each line.

Class ThreadReader ;

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ThreadReader implements Runnable
{
    private String filename;

    public ThreadReader(String filename)
    {
        this.filename=filename;
    }

    public void run()
    {
        Scanner filereader=null;
        try
        {
            filereader=new Scanner(new File(filename)); //Open
file stream Scanner object to read filename
            while(filereader.hasNextLine())
            {

                System.out.println("File Name: "+filename);

                Thread.sleep((long) (Math.random()*1000));
//Call Sleep method to sleep for a random time
//for every line to display
            }
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
    }
}
```

```

        System.out.println(filereader.nextLine());
//display the text of file
    }

    filereader.close(); //close the file filereader
object
    }
    catch (FileNotFoundException e)
    {
        System.out.println(e.getMessage());
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
}

```

Class MainThread :

```

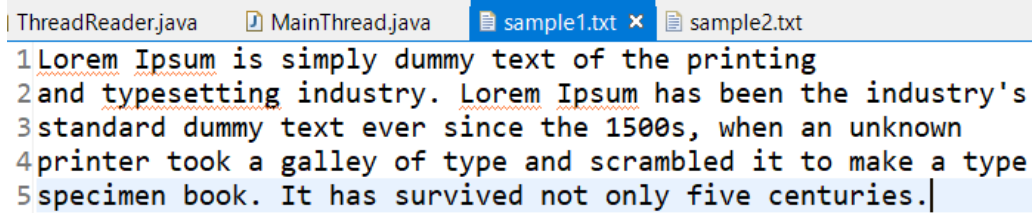
public class MainThread
{
    public static void main(String[] args)
    {
        String file1="sample1.txt"; //Set two file names as string
variables
        String file2="sample2.txt";

        ThreadReader t1 = new ThreadReader(file1); //Create an
object of the ThreadReader class with file1
        ThreadReader t2= new ThreadReader(file2); //Create an
object of the ThreadReader class with file2

        t1.run(); //call run method to start the thread t1
        t2.run(); //call run method to start the thread t2
    }
}

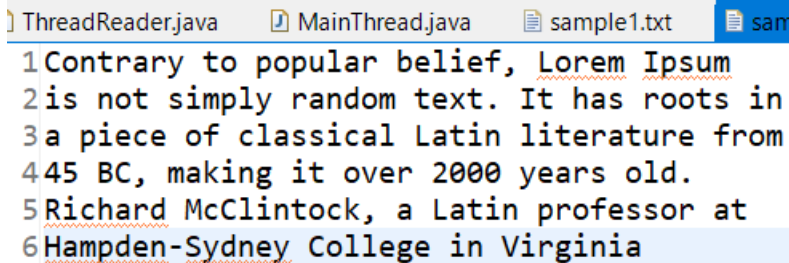
```

Sample1.txt :

A screenshot of a code editor window. The title bar shows four tabs: 'ThreadReader.java', 'MainThread.java', 'sample1.txt' (which is the active tab), and 'sample2.txt'. The editor contains five lines of text, each preceded by a line number from 1 to 5. The text is a paragraph about Lorem Ipsum. The entire paragraph is highlighted with a light blue background.

```
1 Lorem Ipsum is simply dummy text of the printing
2 and typesetting industry. Lorem Ipsum has been the industry's
3 standard dummy text ever since the 1500s, when an unknown
4 printer took a galley of type and scrambled it to make a type
5 specimen book. It has survived not only five centuries.
```

Sample2.txt :

A screenshot of a code editor window. The title bar shows four tabs: 'ThreadReader.java', 'MainThread.java', 'sample1.txt', and 'sample2.txt' (which is the active tab). The editor contains six lines of text, each preceded by a line number from 1 to 6. The text is a paragraph about the origins of Lorem Ipsum. The entire paragraph is highlighted with a light blue background.

```
1 Contrary to popular belief, Lorem Ipsum
2 is not simply random text. It has roots in
3 a piece of classical Latin literature from
4 45 BC, making it over 2000 years old.
5 Richard McClintock, a Latin professor at
6 Hampden-Sydney College in Virginia
```

6. Write a java application that will create and start two threads.

One thread will read a text file (Number.txt) containing five positive integers one on each line.

The second thread should calculate factorial of the number read by the first thread and print the message on the screen as “Factorial of x is y” ,here x is number & y is factorial of the number

The two threads should work in synchronization. Handle all necessary exceptions.