BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
SECOND SEMESTER 2017-2018
LAB - 10
OBJECT ORIENTED PROGRAMMING (CS F213)

DATE: 19th Nov 2018 **TIME: 120 Min**

Problem Description: Read the following specification which explains **HEADS & TAILS** which is a multi-player game.

A. Description of how the game is played: In this game there are two primary entities involved (a) Monitor and (b) Player. There is just one Monitor and there can be multiple Players. Each Player tosses a coin only once and communicates the result (Head = 1, Tail = 0) to the Monitor. The Monitor keeps the count of number of heads and number of tails. Every time the Monitor receives the result from a player it inspects the result and increment number of heads or tails by one. Having received the results from all the players the Monitor publishes the final results. writes the final results (Number of heads = X, Number of tails = Y) in an output file named out.txt.

B. Description of Concurrent Tasks: Both the Monitor and Players run as separate threads. Given below is the description of both the threads.

Monitor carries out the following tasks

a) The Monitor thread continues to execute until all the players don't communicate the results of their coin toss.
b) The Monitor thread must wait for a Player thread (the one which is scheduled currently) to toss a coin.
c) The Monitor thread updates count for number of heads and number of tails.

Player carries out the following activities

a) The Player thread tosses a coin and communicate the result to the waiting Monitor.
b) Other Player threads must wait for the Monitor to process the result obtained from previous Player thread (if any).

WE WANT A STRICT SCHEDULE: Player Monitor Player Monitor Player Monitor ... until number of players

Partial code is given to you. Please write code only for boxed questions.

1. class Data: This class represents shared data object. The Player and Monitor threads are going to use the object of this class for the purpose of communication. The meaning of instance variable ***lock: Object*** is self-explanatory.

```
public class Data {
      // store the result of coin toss
      private int result;
      // set if it is player's chance
      private boolean pChance;
      // set if it is monitor's chance
      private boolean mChance;
      // number of players
      private int nop;
      private Object lock;
      public Data(){

          Q.1 Initialize the first four state variables by reading data from a
          file named "input.txt". See Fig.1 for file format.

      }
      // GETTER AND SETTER METHODS ARE ALREADY PROVIDED
}
```

Fig.1 (input.txt)

```
-1    // initial value of result
true  // player chance flag
false // monitor chance flag
10    // number of Players
```

2. **class Monitor**: This class encapsulates the logic of Monitor thread. Provide implementation for the boxed questions i.e. Q.2 (a), Q.2. (b), Q.2. (c), Q.2. (d) and Q.2. (e).

```
public class Monitor implements Runnable {

      private Data d;          // shared object
      private int tails = 0;  // count number of tails
      private int heads = 0;  // count number of heads
      public Monitor(Data d){ this.d = d; } // constructor

      public void run() {

            Q.2 (a) How long should this thread run?

            Q.2 (b) How to access the shared object?

            Q.2 (c) What is the reason for the monitor to wait?

            System.out.println("monitor is going to read value ...");
            if(d.getResult() == 0) tails++;     else heads++;

            Q.2 (d) The monitor should do some important work after reading
            the result of last player and before attempting to read the
            result of current player in the next iteration

            Q.2 (e) Write the results i.e., number of heads and number of
            tails to a file named "out.txt"

      } // end of run method
}// end of class definition
```

3. **class Player**: This class encapsulates the logic of Player thread. Provide implementation for the boxed questions i.e. Q.3 (a), Q.3. (b) and Q.3. (c).

```
public class Player implements Runnable{
      private Data d;   // shared object
      Random rand = new Random();   // random number generator
      public Player(Data d) { this.d = d; } // constructor

      public void run() {
            Q.3 (a) How to access the shared object?

            Q.3 (b) What is the reason for the monitor to wait?

                  d.setResult(rand.nextInt(2)); //tossing coin and writing result

            Q.3 (c) The player should do some important work after writing the
            result of coin toss to the shared object

      } //end of run method
}
```

4. **class Lab10**: This class is the driver class.

```
public class Lab10 {
      public static void main(String[] args){
            Data d = new Data();
            Thread[] players;
            Thread monitor;

            Q.4 (a) Create and start player threads

            Q.4 (b) Create and start monitor thread

      }
}                              ********** BEST OF LUCK **********
```