Birla Institute of Technology & Science – Pilani
First Semester 2011-12

Date: 5th November, 2011 (Saturday)

ONLINE TEST (OPEN BOOK)

Course Name	:	**Object Oriented Programming & Design**
Course No	:	**CS/IS C313**
MM	:	**100**
Weight age	:	**25%**
Time	:	**3:00 P.M – 6:00 P.M [180 Minutes]**

[Expected Time: 180 Minutes (60 Minutes for Understanding + 120 Minutes for Coding)]

Important Instructions:

1. Check whether java is installed on your terminal or not. If not then change the terminal. Possible look out locations for java compiler are ***(i) c:\Program Files\Java\jdk1.6.0\bin* OR *(ii) d:\jdk1.6.0\bin***
2. Write the whole java code in a single source (.java) file. Use your complete id number for naming the source file. Use only capital letters for naming the file. E.g. 2001A1PS134P.java, 2007A1B7345P.java etc.
3. **The examination is of 100 marks. Out of 100 marks, 80 marks are for coding and 20 marks are for running and execution of the code**. These total 100 marks will be scaled down to 50 later on.
4. Save your work regularly and periodically. All machines are on backup but still you are advised to save your work. You are responsible for the safekeeping of your code. No query regarding loss of code by any means will be considered later on.
5. Three Editors have been installed on your terminals namely notepad, notepad++ & TextPad. Choice of editor(s) is yours and you can use any one of these three editors for coding purpose.
6. **IT IS ADVISED TO FIRST READ AND UNDERSTAND THE WHOLE PROBLEM CAREFULLY AND THEN START WRITING THE CODE**.
7. **You can code using any one among the notepad, notepad++, Textpad editors.**

Problem Description

The problem is about the simulation of the **Fastest Finger First Part** of the Famous **Kaun Banega Crorpati (KBC)** show running on the **Sony Television** these days. The frame shown in **Figure 1** displays the main window used for simulation. **[For clear view check video] The text which is shown in red color are the actual name of the variables used for those fields.**

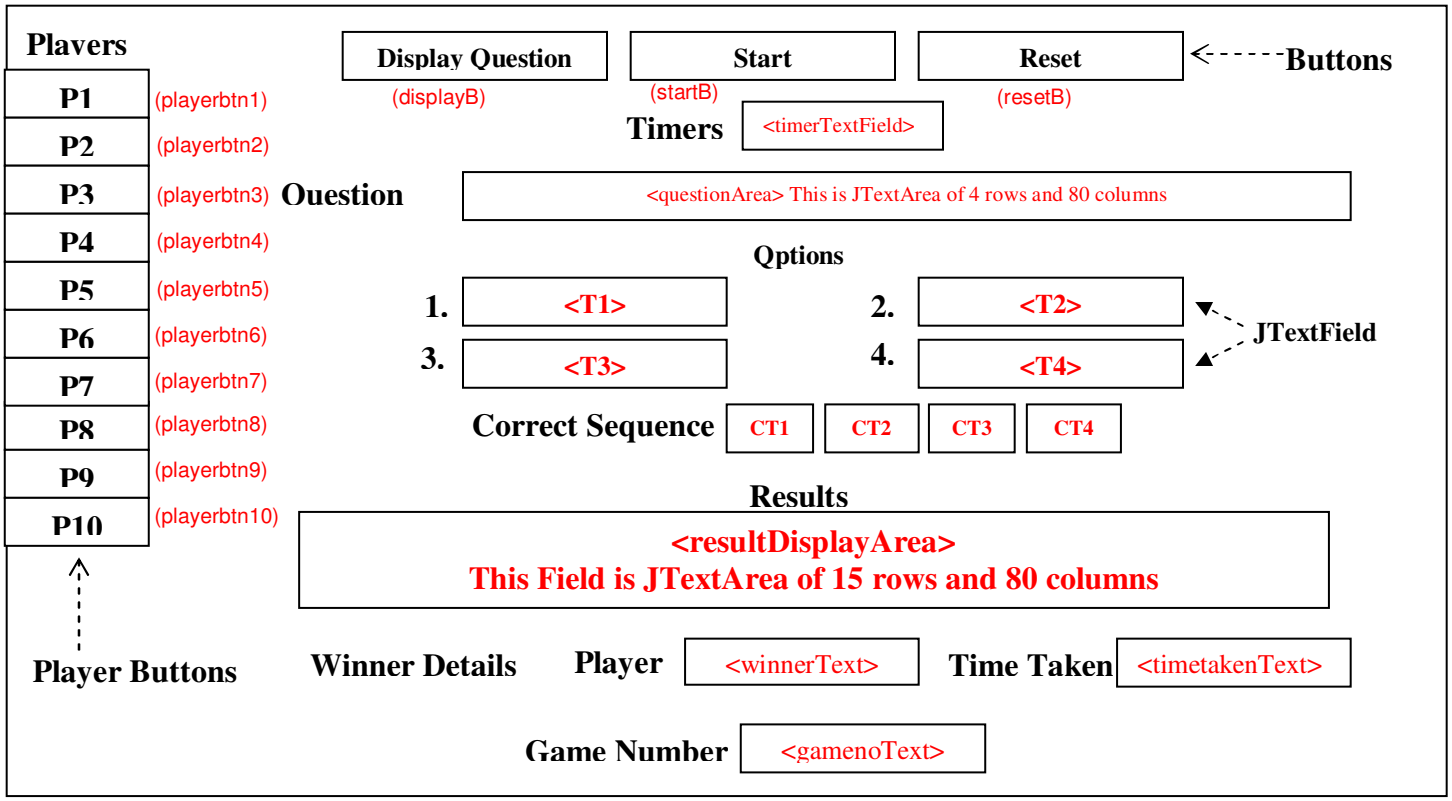

Figure 1

There are ten players which can participate in the game and game is to simulated for maximum ten number of times. The game starts when question (being asked during the game) is being displayed in question display area and the game number has been indicated in *gamenoText* field by pressing **Display Question** button. After that **Start** button is pressed to start the game. When Start button is pressed the color of the player buttons change to GRAY to indicate activation of the players, the timer is started and all the four options associated with the question has been displayed. Each player has been implemented as a thread and during the game each player has to arrange the options in a correct sequence within 10 seconds. When any player answered within timer expiry then the color of the button associated with that player changes to BLUE (indicating answered) and if player answered after timer expiry then the color of the button changes to CYAN (indicating not answered). After the timer expiry (i) Correct sequence of the question is displayed, (ii) details (player-id, answered sequence and time taken to answer) of the players who has answered has been displayed in result display area, (iii) the color of the player button changes to GREEN/RED if answer sequence matches/not-matches with the correct sequence of the question and (iv) the details of the player who has won the game will be displayed. At the start of the game only Display Question button is enabled where as Start and Reset buttons are disabled. When Display Question button is pressed then after doing the activities mentioned above it will be disabled and after that Start button become active/enabled. When an iteration of the game is over (after timer expiry, displaying of the results of the game) only then Reset button is enabled. When Reset button is pressed the details of the previous game has been erased completely and after that Reset button is disabled where as Display Question button is enabled and the new iteration of the game will start. When Display Question button is pressed for the eleventh time the game will be exited.

There are seven classes (**Question, Answer, Player, TimerThread, MasterController, AnswerBoard and Driver**) defined to simulate this game for maximum ten numbers of times. The description of each class has been descried below

1. Question :

This class encapsulates the details for a particular question (such as text of question, its option sequence and its correct sequence) which is to be displayed at the start of the game when Display Question button is pressed). The whole implementation of this class is given to you which is as follows.

```
class Question
{
        private  String          question;              // Actual Question
        private  String[]        optionSequence;        // Option Sequence
        private  int             correctSequence[];     // Correct Sequence
        Question(String question,String[]      optionSequence,      int[]     sequence)
        {
                this.question          =      question;
                this.optionSequence    =      optionSequence;
                this.correctSequence   =      sequence;
        }
        // Accessor Methods
        public   String          getQuestion()          {      return   question;          }
        public   int[]           getCorrectSequence()   {      return   correctSequence;   }
        public   String[] getOptionSequence()           {      return  optionSequence;     }
}// End of class Question
```

2. Answer :

This class encapsulates the details for a particular answer (such as answer sequence, player id and time taken to answer)given by a particular player during the game. The whole implementation of this class is also given to you which is as follows.

```
class Answer    implements      Comparable<Answer>
```

```
{
private int     answerSequence[]    =    new    int[4];  //    Answer Sequence
private int     playerid;                                //    Player ID
private long    timetaken;                               //    Time Taken
//      Accessor Methods
public  int[]   getAnswerSequence()  {    return  answerSequence;      }
public  int     getPlayerid()        {    return  playerid;            }
public  long    getTimetaken()       {    return  timetaken;           }
//      Mutator Methods
public  void    setAnswerSequence(int[] answerSequence)    {this.answerSequence=answerSequence;}
public  void    setPlayerid(int         id)                {this.playerid    =d;              }
public  void    setTimetaken(long       timeTaken)         {this.timetaken  =timeTaken;    }

//      CompareTo Method compares by timetaken values
public  int     compareTo(Answer other)
{
                if (this.timetaken      >       other.timetaken)        return  1;
                if (this.timetaken      <       other.timetaken)        return  -1;
                return 0;

}
//      toString()
public  String  toString()
{
        String s1 = "Player Id: "+playerid+"\t";
        String s2 = "Answered Sequence : [ "+ answerSequence[0]+"  "+answerSequence[1]+"
                        "+answerSequence[2]+"  "+answerSequence[3]+" ] "+"\t";
        String s3 = "Time Taken: "+timetaken;
        return s1+s2+s3;
}
}// End of class Answer
```

3. Player :

This class is actually used to simulate the player of the game. There are 10 instances of this class created in the game. Each player is implemented as a Thread and has been assigned a unique-id (an int value).Whenever initiated (during game) it performs the task indicated as follows:

"Whenever activated (resumed) by a **master** (an instance of **MasterController** Thread explained later) to answer a question then it will first change the color of its associated player button to GRAY (indicating activation) and then it will randomly generate a non-repeating sequence of four integers (each integer in range 1-4) such that there will be a time gap of **x***250+10(where **x** is a randomly generated integer less than 11) milliseconds between successively generated non-repeating integers in the sequence. After generating the sequence it will add/store that sequence (**as an int[]**) as an answer in a shareable data structure named **answerBoard** (an instance of class **AnswerBoard** explained later). After that it will suspend it self and wait for the next activation from **master**. It may be possible that it will generate the non-repeating sequence after the timer expiry and at that time also it will suspend it self and wait for the next activation".

The partial implementation of this class is given to you and you have to complete it.

4. TimerThread :

This class is responsible for implementing the timer feature of the game and is implemented as Thread. There is only one instance of this class active in the game. Whenever activated (resumed) by a **master** (an instance of **MasterController** Thread explained later) during the game it will display values in **timerTextField** field from 10 to 0 after 1 second of interval between each successive display of values i.e. after 10 , 9 will be displayed after a gap of 1 second and son on. After displaying all the values it will suspends itself and waits for the next activation.

The partial implementation of this class is given to you and you have to complete it.

5. MasterController:

This class acts as the master controller for all the player threads and a timer thread and has been implemented as Thread. There is only one instance of this class active in the game. Whenever initiated (resumed) during the game via pressing of the Start Button it works as follows:

"Whenever activated (resumed) via pressing of a Start button, it performs the following tasks in sequence

(i) Displays the question sequence in their respective text fields
(ii) Activates all the players and timer and then wait for the timer to expire
(iii) When timer expires (a) It locks **answerBoard** (an instance of class **AnswerBoard** explained later) (b) displays the correct sequence of the answer (c) displays the answers given by players in result display area (d) changes the player buttons to RED/GREEN depending upon whether their answer is wrong/correct (e) Displays the winner details (if any) and (f) reinitialize the timer for the next iteration of the game.

After doing all the above tasks it will suspend itself and waits for the next activation".

The partial implementation of this class is given to you and you have to complete it.

6. AnswerBoard:

This class simulates the data structure whose single instance will be shared by all the players as well as the master threads. It provides operations for locking and unlocking the answer board. Players can write their answers if and only if answer board is unlocked. It also records the initial start time of the game (when iteration of the game started). This answer board will be unlocked when Display Question button is pressed and will be locked by **master** after the timer expiry. At the start of the game it is locked. This class supplies an instance field named '**answerList**' of type ArrayList<answer>. It has been initialized to size 10 with each element value as null. Every player will add its answer at a fixed place indicated by playerId-1 only. When any player tries to add its answer in the answerList, the time taken by the corresponding player in nano seconds (use System.nanoTime() for the same) to answer that question will be updated in the answer. At the end of the game if any player's place of adding an answer has a null value then it means that player has not answered that question. At the end of the game when Restart button is pressed then all the current entries stored will be deleted and answer board will be reinitialized to default state.

The partial implementation of this class is given to you and you have to complete it.

7. Driver

This class acts as the driver class. This class holds the questionList (of type ArrayList<Question>) which has 10 questions initialized. Each question from this will be selected for each iteration of the game. This class supplies operations for displaying the main frame (already implemented). All the variables that can be used during game have been defined as private static and you have to use only those variables. Apart of those if you want to define any other variable you are free to do so. In this class you have to

(i) Modify and Complete the createThread() method.
(ii) Write the action listeners for Display Question, Start and Reset buttons.

The partial implementation of this class is given to you and you have to complete it.

What is Given?

1. **You are given an executable (with exceptions) source java file named Online.java. Download it from course web-page and store it in your id-n0.java as mentioned earlier.**
2. **A sample run of the final deliverable code is also given. You can download it from course web-page.**

Tasks You have to do?

1. Complete the implementation of the thread classes named **Player**, **TimerThread** and **MasterController**.
2. Complete the implementation of the **AnswerBoard** class.
3. Complete the implementation of **Driver** class.

Note:

1. **Use System.nanoTime() to read System time in nano seconds. This method will return time in nano seconds elapsed since 1st Jan, 1970.**
2. **append() method of JTextArea class adds at the end.**
3. **Do not modify the variables used in the file, but you are free to add any other variable of your own choice.**