BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI SECOND SEMESTER 2019-2020
LAB - 09
OBJECT ORIENTED PROGRAMMING (CS F213)

DATE: 29/03/2020
TIME: 120 Min

IMPORTANT INSTRUCTIONS

1. Download **Online_Batch1.pdf** and **Online_Batch1.java** files.

Problem Description: Read the following specification which explains **TAMBOLA** which is a multi player game.

A. **Description of how the game is played:** In this game there are two primary entities involved (a) Dealer and (b) Player. There is just one Dealer and there can be multiple Players (however we restrict it to **only two players**). Dealer repeatedly makes an announcement of a number in the range 1 to 30 (the numbers are randomly selected and once the number is announced it is not announced again) until all the numbers are not announced or the game is not complete i.e., one or more players have won the game. Both players have a ticket on which there are 6 random numbers in the range 1 to 30. Once the dealer makes an announcement of a number all the players start searching for that number in their tickets. If the number is found the player marks this number. Either one or both the players who finds all the numbers in their ticket is/are declared as a winner.

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

Dealer carries out the following tasks

a) The dealer thread continues to execute until either of the player threads reports success i.e., won the game.

b) The dealer thread repeatedly announces a number in the range 1 to 30 (basically the announced number is an input to the dealer thread via the game GUI).
   (i) The dealer thread waits for the user to press a number button on the game GUI which is then used as an announced number.
   (ii) The dealer thread continuous to execute until all the numbers are not announced or one or more player is not detected as a winner i.e., if one or more players have found all the numbers on his or her ticket.

c) Before announcing the next number the dealer checks if one or more players have found all the numbers on his or her ticket and then declares the winner and sets the game status as complete.

d) Before announcing the next number the dealer allows all the players to search for currently announced number on their tickets and wait for all of them to finish searching for the currently announced number.

<u>Player carries out the following activities</u>

a) Both player threads continues to execute until the game is not complete.
b) Both player threads waits for the dealer thread to announce a number.
c) Once the dealer thread announces the number both the player threads search for the currently announced number in their tickets, if the number is found then the players mark this number as found in their respective tickets (see more on this in the Player class description below).
d) Both the player threads check if they have won the game i.e., found all the numbers on their tickets, then they set their corresponding success flag.

C. <u>Description of several classes:</u>

a) **<u>GameData:</u>** the instance of this class (named **gameData** in our solution) is a shared memory which is used by the dealer and the player threads as a means of communication and synchronization. Both the dealer and the player threads read and write the **gameData**. The **description of several fields** of GameData class are given below:

(i) **+ announcedNumber : int (initialized to zero)**
This variable is initialized by the dealer thread every time the button is pressed on the game GUI.

(ii) **+gameCompleteFlag : boolean (initialized to false)**
The dealer thread sets this variable to true if it detects that either or both the player threads have set the playerSuccessFlag.

(iii) **+ noAnnouncedFlag : boolean (initialized to false)**
The dealer thread sets this variable to true whenever a number is pressed on the game GUI. The dealer thread resets this flag to false before making an announcement of the next number.

(iv) **+ playerSuccessFlag : boolean[ ] (initialized to false)**
playerSuccessFlag[0] corresponds to player thread with a id = 0 (which is player1), and playerSuccessFlag[1] corresponds to the player thread with a id = 1(which is player2). The two player thread sets their respective playerSuccessFlag to true if they find all the numbers announced by the dealer in their ticket which also indicates that the player has won the game.

(v) **+ playerChanceFlag : boolean[ ] (initialized to false)**
playerChanceFlag[0] corresponds to the player with a id = 0 (which is player1) and playerChanceFlag[1] corresponds to the player with a id = 1

(which is player2). The two player thread sets their respective playerChanceFlag to true after they finish searching for the currently announced number. This is done so as to deny the player threads to enter in their critical section for the same number again. And if the player threads gets a chance again before the new number is announced by the dealer they should get blocked. Before making an announcement of the new number the dealer thread sets this flag for both the players to false again.

(vi) **+ lock1 : Object**
This variable is used by the dealer and the player threads to access the shared instance of GameData in mutually exclusive manner.

(vii) **+ lock2 : Object**
The game GUI runs as a separate thread other than the dealer and the player threads. This variable is used to facilitate interaction between the dealer and the game GUI threads i.e., the dealer thread should wait for the user to press a button on the game GUI and the game GUI thread notifies the waiting dealer thread when the user presses a button.

b) **<u>Dealer:</u>** this class implements the Runnable interface. The fields and the methods of this class are explained below:

(i) **<u>Description of the fields:</u>**

- gameData : GameData
This reference is shared by the dealer and both the player threads. It is used as a means of communication between the dealer and the player threads. The fields of GameData and their specifications are already explained above.

- numberAnnounced : int (initialized to zero)
Until the user doesn't press a button of the game GUI the dealer thread keeps waiting. When the user presses a button on the game GUI this variable is initialized with the value of the button pressed. Further the dealer thread initializes the variable **announceNumber : int** of the **gameData** for the player threads to read.

(ii) **<u>Description of the methods:</u>**

<u>+ Dealer(gameData : GameData)</u> - constructor (**code is already provided and should not be modified**)

<u>+ run()</u> - this method is important as it is executed by the dealer thread and you have to write the code for this method. [See the comments written in the run() method of the Dealer class and write your solution]

<u>+ setAnnouncedNumber(i : int)</u> - setter method called when the button is pressed on the game GUI (**code is already provided and should not be modified**).

c) **Player:** this class implements the Runnable interface. The fields and the methods of this class are explained below:

(i) **Description of the fields:**

- id : int (0 for player1 and 1 for player2)

- gameData : GameData

This reference is shared by the dealer and both the player threads. It is used as a means of communication between the dealer and the player threads. The fields of GameData and their specifications are already explained above.

- playerTicketPanel : JPanel

It is a GUI component which is initialized by the player in its constructor (**code is already provided and should not be modified**). Both players have separate copies of playerTicketPanel.

- btnOnTicket : JButton[ ]

Each player has six buttons which are randomly initialized and are added to the playerTicketPanel. This is done by the Player constructor (**code is already provided and should not be modified**).

- totalNumbersFound : int

This variable is incremented by the player threads whenever a player finds a number announced by the dealer in his ticket.

- MAXNO : int

This variable is declared to be static and final and indicates the maximum numbers on the player's ticket.

- ticket : int[ ]

Each player has a separate copy of this array which stores the numbers on the player ticket. It is initialized in the player constructor (**code is already provided and should not be modified**).

(ii) **Description of the methods:**

+ Player(gameData : GameData, id : int) - constructor (**code is already provided and should not be modified**)

- randInt(min : int, max : int) : int - This method generates random integer number in the range min - to - max. It is used by the Player constructor (**code is already provided and should not be modified**).

+ run() - this method is important as it is executed by the player threads and you have to write the code for this method. [See the comments written in the run() method of the Player class and write your solution]

+ getPlayerTicketPanel() : JPanel - this method is called from the game GUI to display the player ticket. It returns the initialized playerTicketPanel to the game GUI (**code is already provided and should not be modified**).

D. You have to write the code for the following:

(a) First implement the **run()** method of the **Dealer class** as per the specification and the comments provided in the **Dealer.java** file.

(b) Second implement the **run()** method of the **Player class** as per the specification and the comments provided in the **Player.java** file.