Dalia Faria

CST338-30_SU19: Software Design

(M8) GUI with Multithreading (Tic Tac Toe Game)

Tic Tac Toe Game

This is a solo final project

Understand the Problem

This assignment combines the applications of Java's *Swing* components and *multithreading*. This project proposes a simple grid layout with a timer aspect. The timer does not interact with the gameplay. The player does not play against a computer, but the game does alternate player characters (X and O) to simulate a two-player turn based set-up. There are two major classes that contain a few nested classes:

- **TicTacToeGame**: A class that contains the *main which sets up the empty JFrame*, instantiates *instances of the game and timer*, and holds the *Timer* class.
- **TicTacToeView**: A class which that *contains the game's constructor* that *creates the major layout on the JFrame*, and the *logic for the gameplay and mouse adapter*.

It is crucial to include the proper java import modules such as the following:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.border.LineBorder;
```

Phase 1: The TicTacToeGame Class

Member Data

Inside the public class **TicTacToeGame**, define a public static **TicTacToeView** instance called **tttGame** above of the inner main class. We will define the instance later.

Inner Classes

- **public static void main(String[] args)** – In this class, we declare the instance we created earlier, **tttGame** to a **new** instance of **TicTacToeGame().** We also want to use **setTitle** to set the title of game instance. We will use this class to set up an empty JFrame by using **setSize** by 600 x 600. Use these standard JFrame set up methods to properly create the JFrame:

```
tttGame.setLocationRelativeTo(null);
tttGame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
tttGame.setVisible(true);
```

- **private static class Timer extends Thread** – This class controls the Timer Thread. Include these variables which are used in the following methods:

```
static int PAUSE = 1000;
private int seconds = 0;
```

Create the three following methods:
public void incrementSecondsonTimer() – Simply increment the seconds.
public void doNothing(int milliseconds) – Include a **try catch** exception for the **Thread**, that will **sleep** for a given argument called **milliseconds**. In the **catch**, try printing a message as such:

```
catch (InterruptedException e)
{
   System.out.println("Unexpected interrupt");
   System.exit(0);
}
```

public int getSeconds() – This should be an accessor method that returns **seconds**.

Now, we must override the **public void run()**. Here, we want to first *check that the game is not over* **while** we perform the following methods: IncrementSecondsonTimer(), doNothing(PAUSE), and setTimerDisplay(getSeconds()). To do this, we can create a Boolean called **getGameStatus()** in the **TicTacToeView** class.

Phase 2: The TicTacToeView Class (extends JFrame)

Member Data

Include these variables which are used in the following methods:

```
private boolean gameOver = false;
private char playerChar = 'X';
private Cell[][] cells = new Cell[3][3];
private JPanel mainPanel, pnlTimerDisplay;
private JLabel timerDisplay = new JLabel("0:00", SwingConstants.CENTER);
private JLabel playerTurn = new JLabel("X's turn to play!");
```

Methods

- **TicTacToe()** – No argument constructor which acts as a set up for the layout onto the empty JFrame created earlier in the main() in TicTacToeGame. First, use **setLayout(new BorderLayout())** to set up a new layout. Now, create the panels, **mainPanel** and **pnlTimerDisplay**. You will want to set the mainPanel to a **new GridLayout(3, 3, 0, 0)**, and the pnlTimerDisplay to a **new GridLayout(1,1)**. Next,

add cells onto the **mainPanel** (*we will create the cells array in the inner Cell class*) with a double nested **for loop** that loops through two-dimensions (rows and columns) to make a grid of 3 by 3 cells. Lastly, follow the methods below to set border lines, add labels onto the appropriate panels and add panels onto the JFrame.

```
//Sets border lines
mainPanel.setBorder(new LineBorder(Color.blue, 2));
playerTurn.setBorder(new LineBorder(Color.red, 3));

//Adds the panels and label onto JFrame
add(pnlTimerDisplay, BorderLayout.WEST);
add(mainPanel, BorderLayout.CENTER);
add(playerTurn, BorderLayout.SOUTH);

//Adds timer JLabel onto timer panel
pnlTimerDisplay.add(timerDisplay);
```

- **public void setTimerDisplay(int seconds)** – This method is to check if the timer display needs a leading zero. If you have a better or more simple approach, feel free to use it. Here is an example of the method declaration:

```
//Check to see if the seconds need a leading zero
if( seconds%60 >= 0 && seconds%60 < 10)
{
   this.timerDisplay.setText(Integer.toString(seconds/60) + ":0" +
           Integer.toString(seconds%60));
}
else
{
   this.timerDisplay.setText(Integer.toString(seconds/60) + ":" +
           Integer.toString(seconds%60));
}
```

- **public Boolean getGameStatus()** – This should be an accessor method that returns the status of **gameOver**.
- **public boolean isFull()** – This method checks whether the grid is full or not. Do this by using a **for loop** that checks each cell for an empty character. If it equals and empty character, return false; otherwise, return true.
- **public Boolean isWinning(char token)** – This method checks each group of cells for a winning pattern by looping through the cells of the grid. There are a few ways to do this. You may find the easiest approach is to use **for loops** to run through the rows and columns. You can create separate **if conditional statements** to check for the diagonal win conditions.

Inner Classes

- **class Cell extends JPanel** - Includes a **private char token** variable set to an empty character. Create the following constructor and methods:

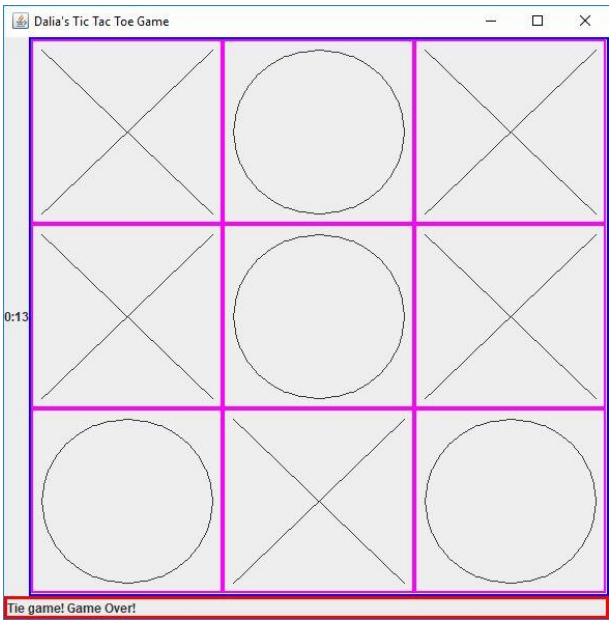
public Cell() - No argument constructor that creates a border line for the cells and adds a mouse listener. To do this, use **setBorder(new LineBorder())** then **addMouseListener(new MyMouseListener()).**
public char getToken() – This is an accessor method that returns **token**.
public void setToken(char playerToken) – This is a mutator method that sets the **token** to **playerToken** and then calls **repaint().**
protected void paintComponent(Graphics playerToken) – This is an overridden method that takes in the **playerToken** as a **Graphics** parameter to draw the image for **X** and **O**. You must include **super.paintComponent(playerToken)** in this overridden method. You are welcome to change the drawing methods, however, here is an effective example:

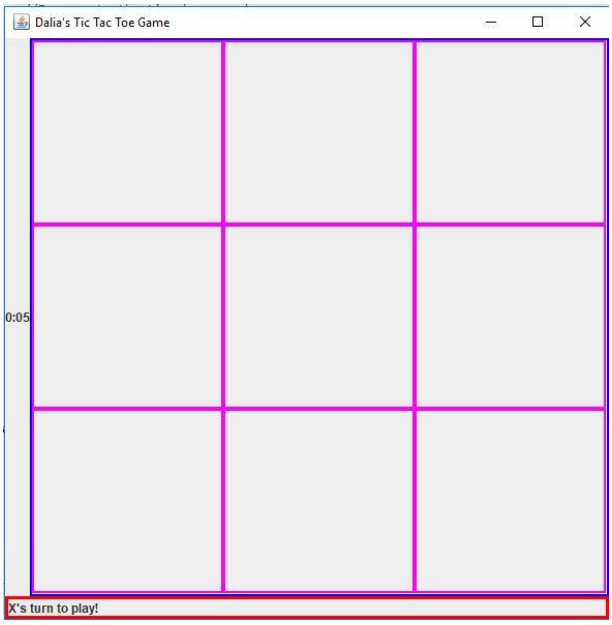
```
super.paintComponent(playerToken);

//Creates the drawn graphic for X and O
if(token == 'X')
{
   playerToken.drawLine(10, 10, getWidth()-10, getHeight()-10);
   playerToken.drawLine(getWidth()-10, 10, 10, getHeight()-10);
}
else if(token == 'O')
{
   playerToken.drawOval(10, 10, getWidth()-20, getHeight()-20);
}
```

- **private class MyMouseListener extends MouseAdapter** - This class implements the logic for the mouse action listener. We will consider what the game will allow the player to do when clicking on a cell based *on if the game is over, if the cell is empty, if a player won, and if the grid is full*. This class also alternates the player character from X to O. First, create an **if conditional** that **checks if the game is over** and do not do anything if it is. Create another **if conditional** that **checks if the token is an empty character**, if true then **setToken(playerChar)**. We will also check if **playerChar isWinning()**. If true, then we want to **setText** of **playerTurn** to which ever character (**playerChar**) won, and set **gameOver** to true. Otherwise, we want to check if the grid **isFull()**, then set the **playerTurn** text to a tie game, set the **playerChar** to an empty character, and **gameOver** to true. If all else does not apply, we want to alternate the **playerChar** from **X** to **O**, and have the text to **playerTurn** display whose turn it is to play next.

Example Output of Tic Tac Toe Game with Timer





Test Run Requirements:

Be sure to test that the timer stops appropriately when the game is over. Also, check that both the X player and O player can win on separate turns. Ensure that the tie-game works properly. Verify that all text messages show properly on the bottom title.