HW2

CSCI: Object-Oriented Programming

Due Date: 8 October 2022 by 11:59 pm

1 OOP Basics

This HW will help reinforce the concepts of decomposition and generalization, writing classes from scratch and implementing interfaces.

2 Grading

The grade of this assignment is distributed as follows:

- compiles with no error (55%)
- design (use of decomposition and polymorphism) (30%)
- style, such as commenting your code properly, indentation (15%)

3 Getting Started

For this assignment, you will be implementing your own Tic Tac Toe game. Your game will have two **TicTacToePlayer**s playing against each other by placing the symbols ("X" and "O") on a **TicTacToeBoard**. You will use a **Controller** class to switch the turns between players, or check for a winner or display update messages about the game status.

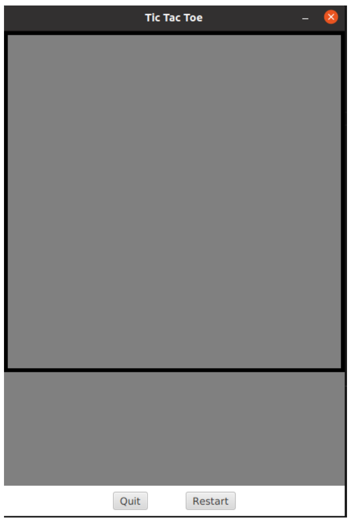
The table below summarizes what update message should be displayed on each game status.

You need to define the body for these relevant methods. You will use the methods provided in Javadocs to implement the relevant methods.

4 Incremental Coding

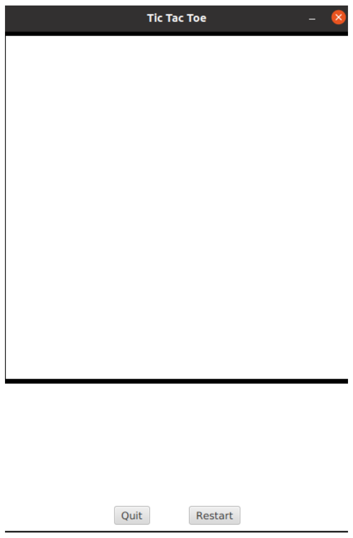
Step 1: First try to import your hw2.jar files (downloaded from the dsingh/cs.sfasu.edu) and Javafx libary, and run your program and do the first git commit. Currently, just a gray window will appear with Quit and Restart button. You should also be able to access the Javadocs file on the dsingh/cs.sfasu.edu

Game Status	Relevant Method	Display Message	Note
When a player selects a square so their turn ends	**selectSquare**	"Player X finished their turn"	The player's "name" (here X) must match that player's designated letter
When a player selects a square that's been taken	**invalidSquareChosen**	"Square chosen has already been taken. Try again."	In addition, the selected square should flash come color.
When a player wins	**playerWins**	"Player X wins"	The player's "name" (here X) must match that player's designated letter.
When the game ends in a tie	**noWinner**	"it's a tie"	
When the game starts or restarts	**restartGame**	"Select a square to start playing"	In addition on game retsart, the board icons and highlights should be cleared. Be sure to call **playAgain** on restart.

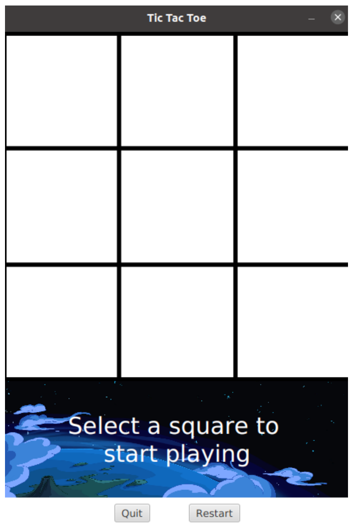


Step 2: Write a class, *myTicTacToe*, that implements the **cs331TicTacToeGame** interface, and make the game show up in the App frame. Hint: Make sure you call the addGame method to the **cs331TicTacToeFrame** object in the App class.

- When you implement an interface, then you must provide for those methods. For now you can keep these methods with empty method body.
- After successful completion of this step, you will see two separate, white portions of the screen for the **Controller** (at the bottom) and **cs331TicTacToeBoard** (at the top), with Quit and Restart buttons displayed as before, as shown in the figure below.



Step 3: In the class that you created, instantiate a **cs331TicTacToeBoard** and a **Controller** so that the screen appears ready to start a round of tic toc toe. When the game starts, the **Controller** should display the message ”Select a square to start playing!”.



Step 4: This is an important step. Create a class, *Players* to represent the players, implementing the **cs331TicTacToePlayer** interface. You will have to think about how to design this class.

Hint:

- Imagine a tic tac toe game where two players bring their own board game to play the game. You won’t be playing the game together in that case. Both players needs to share the same board and know the status of the board at any round in order to play the next move in the game. The same logic also goes for the Controller. Therefore, you should think of using

Decomposition principle to write the Player class, and ask yourself what member variables would you need for players to be able to play this game.

- How will each player know what letter (X or O) to put down on the board? We can think of each icon as a property of that player - when instantiating a player, it needs to be told what its icon is (i,e "X" or "O"). Think about how this can be stored so that the player knows its icon. You will need to add a private member variable to denote the icon ("X" or "O") for the players.

Step 5: Create two instances of your player class and add them to the game via the **Controller**. Please note, the Controller class is written in such a way that it only accepts "X" and "O" as player icons, so be sure to use those letters when initializing the Player class.

Step 6: Implement the **selectSquare** method so that when the user clicks, the player's icon (i,e "X" or "O") appears in that square. Be sure to update the **Controller**'s message per the table above and then (after displaying the message), display another message telling it that this turn is over.

Step 7: Implement the four remaining methods in the *myTicTacToe* class to deal with various game scenarios described in the table above. You can also refer to Javadocs to read about the functionality of each methods.

5 Miscelleneous

- After finishing each step, you should commit to your local repo.
- Finally, you can push your work to the remote github repo and share your work with me as a collaborator.
- Remember you are required to write two new classes (myTicTacToe and Players) from scratch. You must implement cs331TicTacToePlayer and cs331TicTacToeGame to classes *myTicTacToe* and *Players* classes respectively.
- Please refer to Javadocs to get the list of all classes in the helper code and what they do.
- You should create a UML diagram to represent your design and submit it as pdf file along with your final submission.
- Make sure you add a README file with your name, and any other information you would like me to look at before running your program.

6 Using JavaFX color

JavaFX is a library that consists of set of packages used for creating graphics. The package **javafx.scene.paint** contains one very useful class called **Color** and defines a series of constants that represent color values. You can find a full list of colors in the **javafx.scene.paint.Color** Javadocs on the Oracle documentation website. In order to use any of these colors in a class, you should import the class **javafx.scene.paint.Color** at the top of the file and then reference a call like **Color.RED** or **Color.BLUE**.

You will find the methods in the helper code that require a **Color** as an argument. You can chose any of the JavaFX Colors as long as the color change is clear and the tic tac toe board is still visible.

7 Note on getWinningPlayer

In the **playerWins** method, you will need to know *which* player has won in order to include the winning player's symbol in the controller update message, so you can use the **getWinningPlayer** method, which the Javadocs say returns type **cs331TicTacToePlayer** i.e, any class that implements the *cs331TicTacToePlayer* interface. Using the interface as a return type guarantees that in your code, any class that implements this interface (in this case, Players) will be accepted. (This is why Polymorphism is so useful).

Remember that implementing an interface does not limit the methods that you can have in that class. You are still allowed to explicitly declare and define any methods you need that are required by the interface.

Consider this block of code you might write:

```
cs331TicTacToePlayer player = this.controller.getWinningPlayer();
player.doSomething();
```

While the declared type of this **player** variable is **cs331TicTacToePlayer**, its actual type would be the class that you wrote to represent the player. But, because **doSomething** method is not defined in the interface (but defined in the *Players* class), this code will give you compilation error (due to Narrowing). Therefore, you would need to use explicit typecasting to fix this problem. Also remember that before you use explicit typecasting, it's a good practice to use *instanceof* to make sure if the object can be type casted to the desired subtype.

Now go and find a friend, or a sibling or even your pet to play your Tic Tac Toe Game!