CSE 271 Project 4 – Graphical User Interfaces and Exceptions
Spring 2021
Assigned: 3/29/2021
Due: 4/13/2021

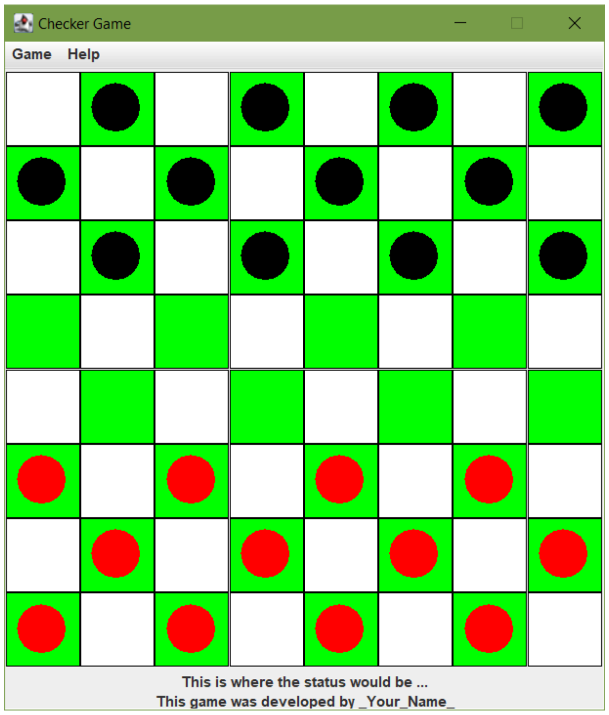
Introduction:

In this project, we will be designing a Graphical User Interface (GUI) of a checkerboard. Though, this checkerboard is **not** functional, we are simply drawing one using the knowledge acquired from the labs and lectures. Though, there is an option for **bonus credit** if you do make it a functional checkboard game that can be played by two human players. This bonus credit is worth 50 points (one half of a project) that will give you an opportunity to improve your projects category grade if you wish to do it. Below are two links if you are unaware of how to play checkers or what it looks like. Please read the wiki and watch the video before starting this project.

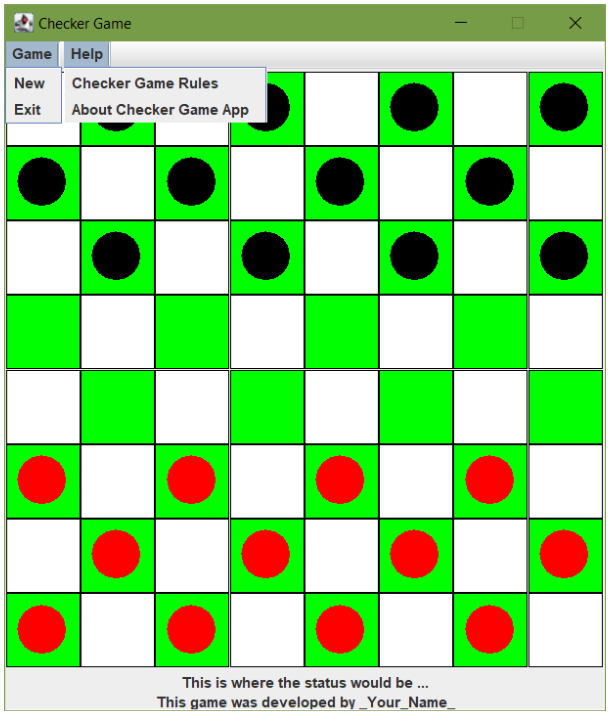
- https://www.wikihow.com/Play-Checkers
- https://www.youtube.com/watch?v=ScKIdStgAfU

CheckerBoard:

Develop a GUI using standard GUI elements that will draw a checkerboard and place multiple checker pieces in their corresponding starting positions as shown below. The board is made up of 64 alternating green and white squares which appear in 8 rows of 8. There are 32 green squares and 32 white squares. Each player places their pieces on the 12 green squares in the first three rows closest to that player. Each of these three rows should have a total of 4 checkers to equal 12 checker pieces. Remember that checkers may only move in diagonal directions on the green squares. Since the board has 8 rows, 6 of the rows will be taken up by checker pieces and the remaining 2 middle rows will be left open with no pieces at the start of the game.



This is where the status would be ...
This game was developed by _Your_Name_



This is where the status would be ...
This game was developed by _Your_Name_

The pieces are specified by an 8x8 char array called *boardStatus*. The status of any given square is either **r** (red), **b** (black), or **e** (empty) as shown in the array below.

```
private char[][] boardStatus = new char[][] {
    {‘e’, ‘b’, ‘e’, ‘b’, ‘e’, ‘b’, ‘e’, ‘b’},
    {‘b’, ‘e’, ‘b’, ‘e’, ‘b’, ‘e’, ‘b’, ‘e’},
    {‘e’, ‘b’, ‘e’, ‘b’, ‘e’, ‘b’, ‘e’, ‘b’},
    {‘e’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’},
    {‘e’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’, ‘e’},
    {‘r’, ‘e’, ‘r’, ‘e’, ‘r’, ‘e’, ‘r’, ‘e’},
    {‘e’, ‘r’, ‘e’, ‘r’, ‘e’, ‘r’, ‘e’, ‘r’},
    {‘r’, ‘e’, ‘r’, ‘e’, ‘r’, ‘e’, ‘r’, ‘e’}
}
```

The above design of the GUI involves three classes which are **CheckerPiece**, **CheckerBoard**, and **CheckerGame**. The specifications for these classes are discussed below. You also need to include a class called **IllegalCheckerboardArgumentException** for which the specification is given later in this document.

1. **CheckerPiece –** This class represents a square in the checkerboard that extends the JComponent class. It draws a square if it is empty (e) or a square with black (b) or red (r) circle depending on if a checker piece is in the square. It overrides the paintComponent(Graphics g) method to draw. You can use fillRect(), fillOval(), and drawRect() methods to draw squares and checkers. Write the following instance properties and public interface:
    a. **Instance Properties:**
        i. **char status –** The status stored as a char for a square on the checkerboard. The valid values are ‘r’, ‘b’, and ‘e’, for red, black, and empty, respectively.
        ii. **int row –** The row index of the square in the checkerboard. For example, row 0 is the top of the checkerboard and row 7 is the bottom of the checkerboard.
        iii. **int column –** The column index of the square in the checkerboard. For example, column 0 is the left side of the checkerboard and column 7 is the right side of the checkerboard.
        iv. You can declare as many instance properties and constants as you deem necessary. The solution used constants for height and width of the squares and checkers as well as x and y coordinates for the squars and checkers. For the figure above, the length of the squares is 60 pixels and the length of a side of checkers is 40 pixels.
    b. **Public Interface:**
        i. **public CheckerPiece(int row, int column, char status) –** The constructor that sets the row, column, and status of a checker piece. It throws an **IllegalCheckerboardArgumentException** if the value of the **row**, **column**, or **status** is invalid. For example, **row** and **column** must be between 0 and 7 and **status** can be either ‘r’, ‘b’, or ‘e’. Also, throw an **IllegalCheckerboardArgumentException** if there is an attempt to put a red or black checker on in invalid square, i.e., checkers cannot be placed on whit squares.
        ii. **public void paintComponent(Graphics g) –** The ***overridden*** paintComponent() method from the JComponent class. Draw checkerboard squares with or without the checker piece based on the value of **status**.
        iii. You can add as many helper methods or inner classes as you deem necessary to make the game functional.

2. **CheckerBoard –** This class represents a checkerboard that contains 64 **CheckerPiece** objects (squares with or without checkers). It extends JPanel and adds 64 CheckerPiece objects to the panel, i.e., the **CheckerBoard** class. You can use the GridLayout(8, 8) to organize the **CheckerPiece** objects on the panel. Write the following instance properties and public interface:
    a. **Instance Properties:**
        i. **char[][] boardStatus –** A two-dimensional character array that is a size 8x8 which stores the status values for each of the squares.
        ii. You may declare as many instance variables and constants as you deem necessary. The solution uses two constants, **ROW** and **COLUMN** for the **boardStatus** array.
    b. **Public Interface:**
        i. **public CheckerBoard(char[][] boardStatus) –** The constructor that sets the **boardStatus** array using the received parameter. It also uses this array to instantiate 64 **CheckerPiece** objects based on the row, column, and status values. It adds all of these objects to the panel. As mentioned above, you can use the GridLayout(8, 8) to organize the **CheckerPiece** objects on the panel.
        ii. **public void setBoardStatus(chare[][] boardStatus) –** A setter method that sets the **boardStatus** array using the received parameter. For making the game functional, you can call the repaint() method after setting the status to redraw the board based on the new **boardStatus**.
        iii. **public void setCheckerPiece(int row, int column, char status) –** A setter method that sets the status in **boardStatus** for a specific square on the board based on the row and column. For making the game functional, you can call the repaint() method after setting the status to redraw the board based on the new **boardStatus**.
        iv. You can add as many help methods or inner classes as you deem necessary to make the game functional.
3. **CheckerGame –** This ***driver*** class represents the checker game window, i.e., a JFrame. The checker game window has a checkerboard along with the status bar and menu bar. The status bar is a JPanel object with two JLabels that show the number of red and black checkers remaining during the game as well as the name of the programmer (your name) who developed the game. Specifically, **CheckerGame** extends JFrame to create a window and adds a **CheckerBoard** object (a child of JPanel) and a status JPanel. It also includes a menu bar to the window as shown in the GUI example above. Write the following instance properties and public interface:
    a. **Instance Properties:** You may declare as many instance properties and constants as you deem necessary. The minimum used for the non-functional solution is a **char[][] boardStatus** with values as shown above which is passed to the **CheckerBoard** object.
    b. **Public Interface:**
        i. **public CheckerGame() –** The constructor that sets the layout using the **CheckerBoard** object (a JPanel), the status bar (a JPanel object) and the menu bar. You need to pass a two-dimensional char array with boardStatus (as shown above) when you create a **CheckerBoard** object. You can use the BoarderLayout to set the layout of the JFrame to organize the GUI elements. Also, set the title, size, default closer operation, etc. for the JFrame window. The solution utilizes a 505x585 size for the window. Finally, add the JMenu and JMenuItems as shown in the figure above.
        ii. **public static void main(String[] args) –** The main method which creates an object of the **CheckerGame** class and makes the window visible.
4. **IMPORTANT –** You may add as many helper classes and methods as you deem necessary to complete this project. The classes and methods specified above are enough for the non-functional GUI of the checkerboard game.

Custom Exception Creation:

In Java, you are capable of creating your own custom Exceptions instead of using the ones provided. Fortunately, they are just a form of class which you have plenty of experience working with now. Write a custom Java Exception as follows:

- **IllegalCheckerboardArgumentException –** A custom Java Exception which extends **Exception**. Define a default constructor and a constructor that receives a String message.
    - **public IllegalCheckerboardArgumentException()**
    - **public IllegalCheckerboardArgumentException(String message)**

Example of a custom Exception class:

```
class MyException extends Exception {
        public MyException() {
                super();
        }

        public MyException(String message) {
                super(message);
        }
}
```

Example of using a custom Exception:

```
if(<some condition>) {
        throw new MyException(“some message”);
}
```

Making the Checkerboard Functional (Bonus Credit):

You will receive bonus credit it you complete the following items:

1. **(30 points)** Make the game work following the rules posted in the link in the introduction. The rules are summarized as follows:
    a. It should be a two-player game where the player with black checkers will play first, then the player with the red checkers. These turns will continue until the game concludes.
    b. A player can move on space diagonally to an empty space or jump over the opponent’s checker piece.
    c. You need to update the board with every valid move the players make. To move a checker piece, the player has to click on that piece and then click the empty square where they want to move that piece.
    d. Do not move the checker piece is the player makes an invalid move.
2. **(10 points)** You need to make the menus from the menu bar functional.
    a. **New**: Starts a new game and resets the checkerboard to the starting configuration.
    b. **Exit**: Exit the game and program.
    c. **Check Game Rules**: Display a JOptionPane message box that has the link to the game rules (the wiki link from the introduction).
    d. **About Checker Game App**: Display a JOptionPane message box that has your name, email, and the university name.

3. **(10 points)** You need to update the status label with the current count of black and red checkers as the game progresses. Also, declare the winner (result of the game) once the game finishes.

Bonus Credit Hint:

You can use MouseListener and MouseMotionListener to listen to the coordinate of the of the button clicked and determine which checker piece has been selected. Also, use variables to store and determine player turns and invalid moves.

JavaDoc Style Comments:

You are required to make JavaDoc comments for all classes and methods including parameters and return descriptions.

Important Note:

Please make sure the file names are correct and the code is well commented!

Submission Instructions:

Submit your .java files (**CheckerBoard.java, CheckerGame.java, CheckerPiece.java, and IllegalCheckerboardArgumentException.java**) zipped to the corresponding Project 4 assignment folder on Canvas.

Rubric:

Task	Grade
CheckerPiece	
CheckerPiece extends JComponent	2
Instance properties correctly defined	2
Constructor correctly implemented with exception	3
paintComponent() method overridden and correctly implemented	20
CheckerBoard	
CheckerBoard extends Jpanel	2
Instance properties correctly defined	2
Constructor correctly implemented	20
setBoardStatus() correctly implemented	4
setCheckerPiece() correctly implemented	3
CheckerGame	
CheckerGame extends JFrame	2
Passes a char[][] to initialize the CheckerBoard correctly	5
Updates the frame title, size, and default close operation correctly	5
Constructor adds CheckerBoard object, status panel, and menu bar	10
GUI look similar to the one in the manual	5
The main method creates a CheckerGame object and sets the visibility	3
IllegalCheckerboardArgumentException	
Correctly extends Exception	2
Both constructors correctly implemented	10
JavaDoc	
JavaDoc present for all classes and methods (can only lose points)	-10
Total	**100**
Bonus Credit	
Functional game following the rules	30
Menus are functional	10
Status panel shows the game status	10
Total	**50**