INST0004: Programming 2 [Optional Checkers Game]

Dr Daniel Onah

This is an example of using classes written by others (as mentioned in Problem set 3)

By now you should have the hang of creating instances of classes and calling methods on those instances. So let's look at a slightly more realistic examples in this modelling draughts game. I've written a few classes for you that lets you write simple, animated 2D computer games. :-D

On Moodle you will find a file called **CheckersBoard.zip**. **Download** it, and **extract** it into wherever you store your Java programs for the game. You'll find two classes in the CheckersBoard.zip file: The classes, … and … You'll will find these tasks are well documented for you to follow the guides.

- The game is designed to test your understanding of the Object Oriented Programming concepts we've seen in the Lecture and online-labs, and their applications in Java.

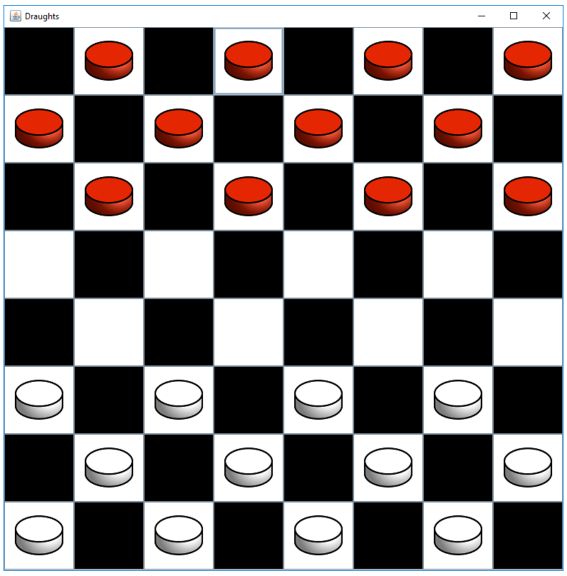
I'm making the draughts game class your responsibility. Feel free to add new instance variables and methods to the class.

Don't remove any of the methods and instance variables already defined in these classes though .. else the software may not be able to operate.

Aims

This game project and tasks is to create an interactive model of a game of draughts (also known as checkers). The group project is separated into incremental tasks that you could work together weekly or as you please. The more tasks you complete, the more marks you will receive.

An example of the end product/design is shown below for you reference:



Task 1: Creating the Board …☺

Time to write and update the program that uses the classes you have been given for this Checkers Game.

- As a programmer or aspiring programmer. You know you should be using version control system for your project … So start by creating a private github repository for your work. ☺
- So that you can focus on programming rather than graphics editing, we've provided a simple set of images for you to use in this program. Download these resources from Moodle and extract them into the directory where you will be developing your code.
- Create a class called **Board** to represent your Graphical User Interface (GUI).
- Create a class called Square to represent a clickable square on your GUI. Think carefully about which Swing class might help you here.
- Add further instance variables to your **Square** class so it can hold information about its location on board. Write accessor (get) methods for your instance variables.
- Write a **constructor** for your **Square** class that creates an empty square, and records its location based on parameters to the constructor.
- Write a **constructor** for your **Board** class that uses Swing to show a window, creates 64 instances of your Square and uses this to create an **8x8** grid.

HINT:

Images in Swing applications are represented by a class called **ImageIcon**. The ImageIcon class contains a constructor that lets you create an instance of an ImageIcon from a give filename.

The **JButton** class is able to create buttons showing images as well as text … In particular, the JButton class has a **constructor** that allows a JButton to be created from an ImageIcon, and **accessor** and **mutator** methods (getIcon and setIcon) for the image currently associated with a JButton.

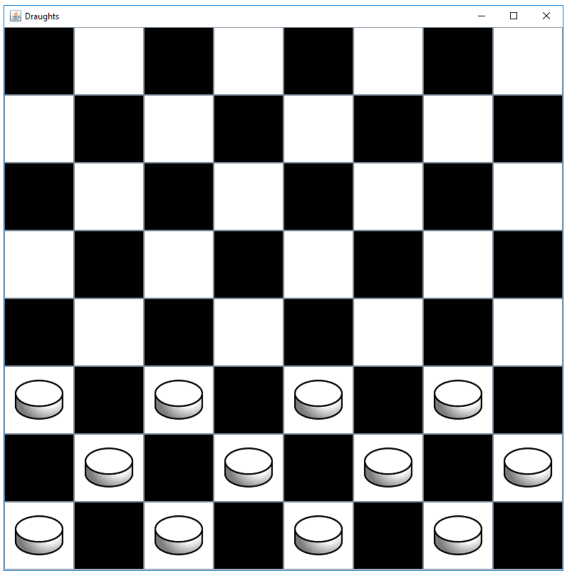
For example:

```
ImageIcon i = new ImageIcon("empty.png");
JButton b = new JButton(i);
```

```
ImageIcon i = new ImageIcon("red.png");
b.setIcon(i);
```

Task 2: Adding Pieces … ☺

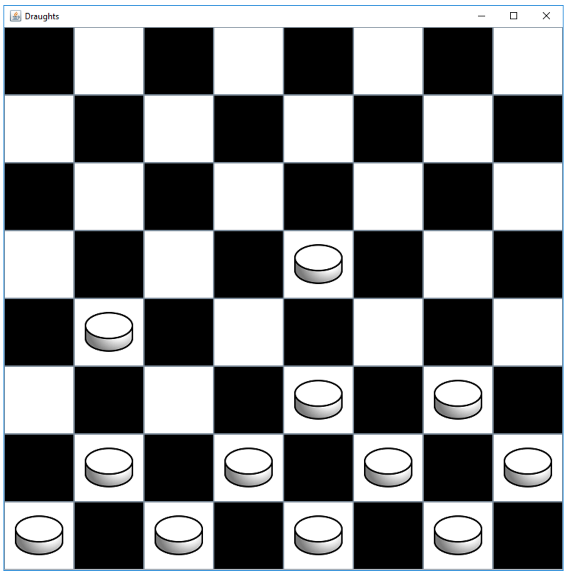
- Add an instance variable to Square class so that each Square knows what piece, if any is contained on that square. Example, valid values could be NONE, WHITE, RED, etc. etc. etc.
- Update the constructor in Square so that it takes an additional parameter – say … the type of piece (if any) that is to be placed on the square. Store its information in the instance variable you just created , and write code so that the appropriate image is shown on the GUI.
- Update your board class to create a Board that has one player's set of pieces in the standard position, as shown below.



Task 3: Moving Pieces …☺

- Develop your Board class so that it can detect when any of the Squares are clicked with a mouse.
- Write a **moveTo** method for your Square class. This should take another Square as a parameter. When invoked, this method should move the piece on that square to the square provided as a parameter.
- Update your Board class so that when a user clicks on a square containing a piece, a subsequent click to an empty square will move that piece to the square indicated. You should use your **moveTo** method to achieve this.

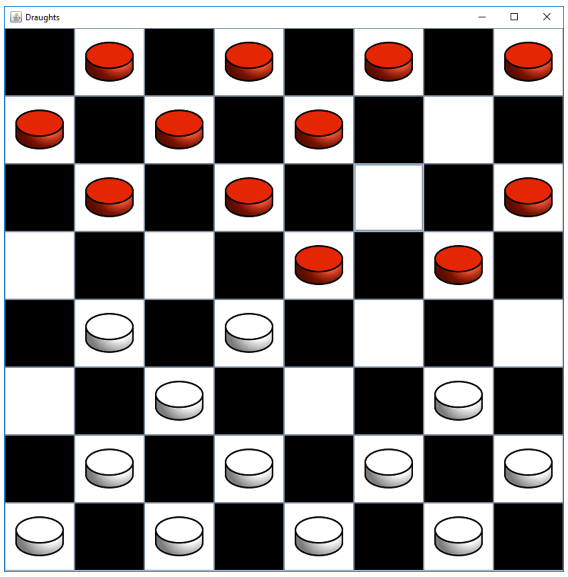
  See example below:



Task 4: Other Pieces …

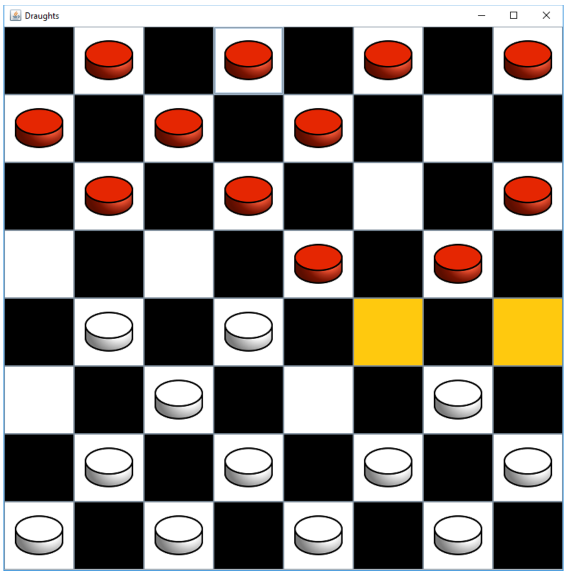
- Update your program so that opposition (RED) pieces are also displayed in their correct positions.
- Develop your code such that they too can be moved according to the rules of draughts. Remember that these pieces will need to move in the opposite direction to the WHITE pieces. Avoid repeating code as much as you can – if you write your code modularity, it should take very little additional code to achieve this.

  Example outcome below:



Task 5: Highlighting Moves …

- Write a method **canMoveTo** in your Square class. This method should take another Square as a parameter and return a boolean. Write your code in this method so that it returns true if the piece on that square can legally move to the square provided as a parameter. It should return false otherwise.
- Update **your Board** class so that when a square containing a piece is clicked, the empty squares where that piece can legally move to are highlighted in orange (not the selected.png file will assist with this).
- Update your **Board class** so that your program only allows the user to make legal moves.



If you are unsure about the valid moves of draughts/checkers, see this link for a good summary:
https://en.wikipedia.org/wiki/English_draughts

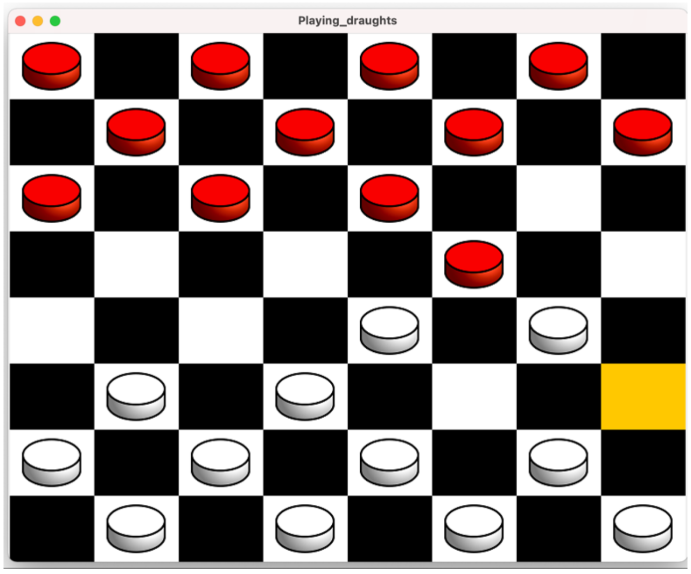
Extra Task: Playing the Game …☺

Now you have your building blocks, develop your own program so that:

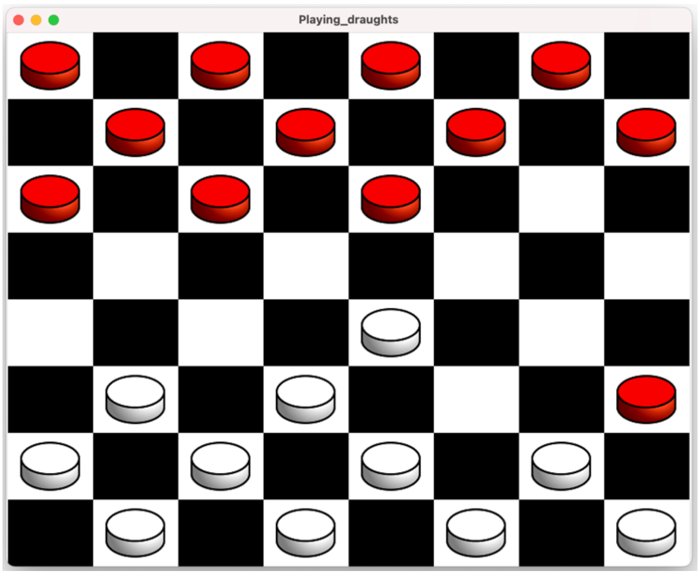
- Players must take their turn to move their pieces. White always plays first. Ensure that players are prevented from braking this rule.
- Extend your program to allow pieces to be taken … remember of course, then a player's piece can jump over more than one opponent's piece at a time.

Example moves:

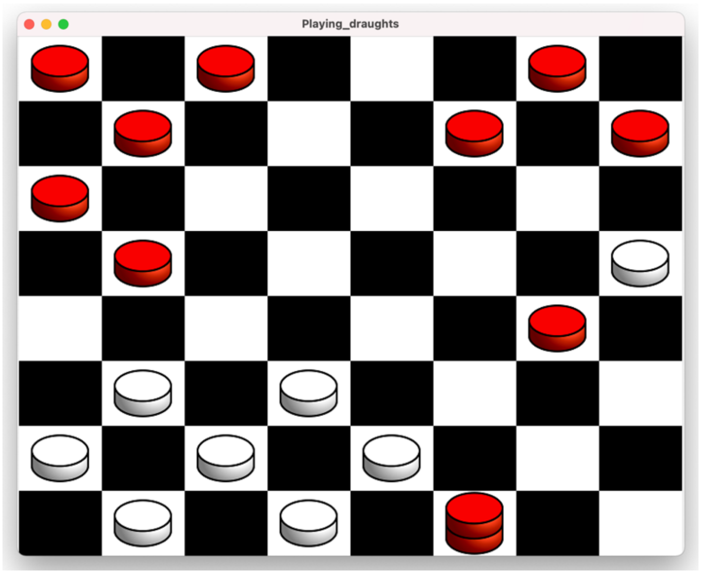
First step: Move



Second step: Move



Final step might look like the outcome below. This shows the crowing on a RED piece.


Playing_draughts