SCC110: Software Development Term 2. Java Programming.

Lancaster University

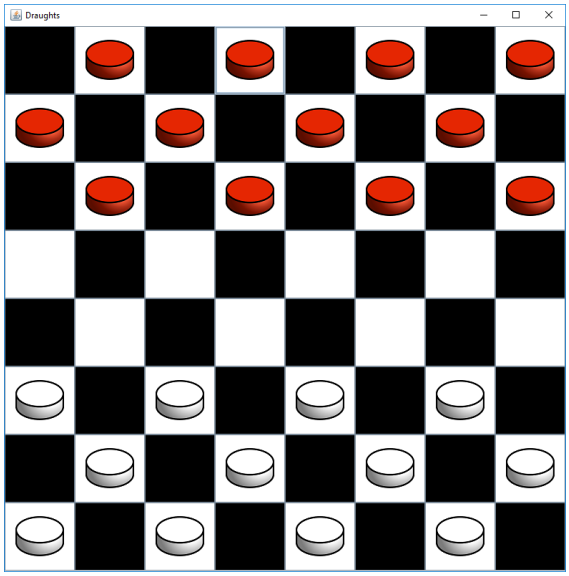
Assessed Exercise: Modelling Draughts
Moodle Submission Deadline: 16:00 Friday Week 20
Assessment Mode: IN LAB ASSESSMENT IN YOUR OWN WEEK 21 LAB SESSION

Aims

This **assessed exercise** is designed to test your understanding of all the OO programming concepts we've seen in the Lectures and Labs, and their application in Java.

Your assignment is to create an interactive model of a game of draughts (also known as checkers). The assignment is separated into incremental tasks. The more tasks you complete, the more marks you will receive.

An example of the assignment is shown below for reference:



Lancaster University

Task 1: Creating the Board…

- As professional software developers, you know you should be using a version control system for your project… So start by creating a private github repository for your work. 😊
- So you can focussing on programming rather than graphics editing, we've provided a simple set of images for you to use in this exercise. 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 a 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 given 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.

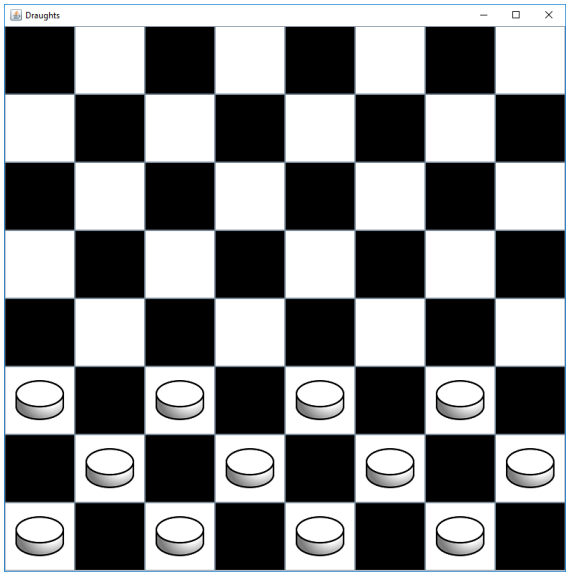
For example:

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

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

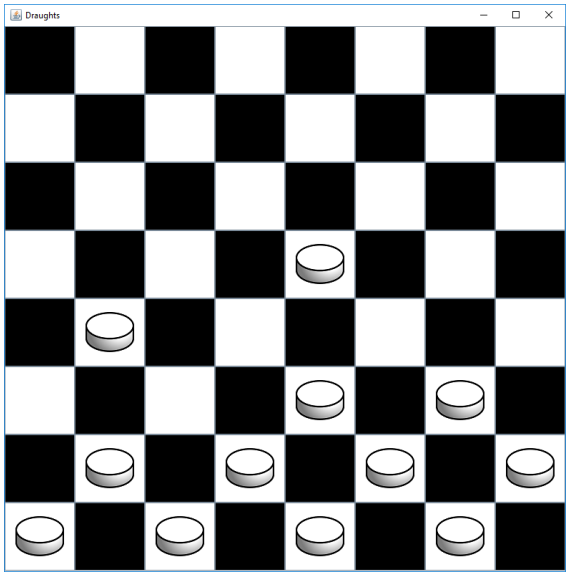
Task 2: Adding Pieces…

- Add an instance variable to your Square class so that each Square knows what piece, if any, is contained on that square. e.g. valid values could be NONE, WHITE, RED, etc. etc.
- Update the constructor in Square so that it takes an additional parameter – the type of piece (if any) that is to be placed on the square. Store this 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.



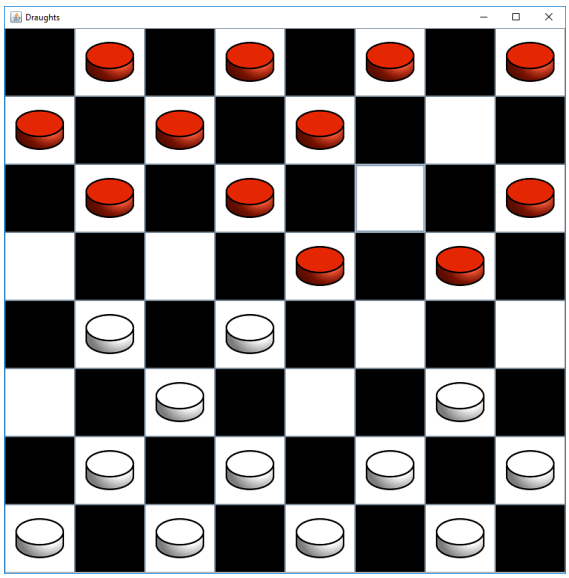
Task 3: Moving Pieces…

- 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.



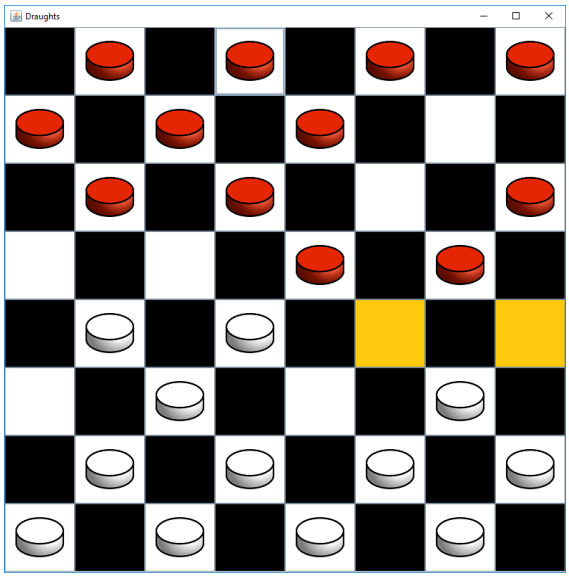
Task 4: Other Pieces…

- Update your program so that opposition (RED) pieces are also displayed in their correct positions.
- Develop you 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 modularly, it should take very little additional code to achieve this.



Task 5: Highlighting Moves…

- Write a method called **canMoveTo** in your Square class. This method should take another Square as a parameter and return a boolean. Write 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 (note the selected.png file will assist with this).
- Update your Board class so that your program only allows the user to make legal moves.

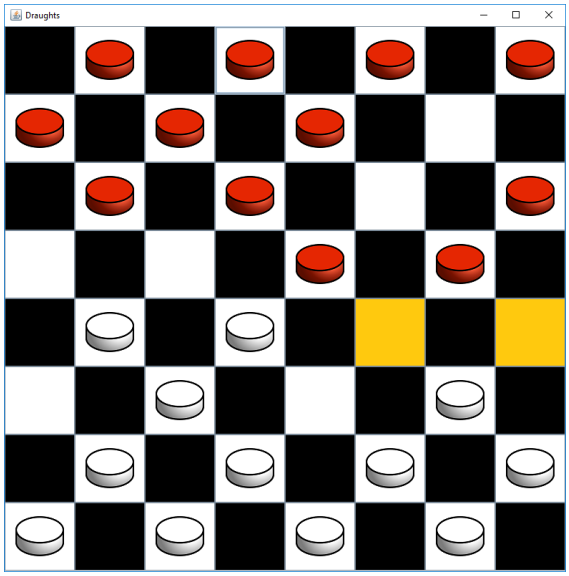


If you are unsure about the valid moves of draughts, see this link for a good summary:
https://en.wikipedia.org/wiki/English_draughts

Extra Task: Playing the Game…

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

- Players must take it in 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.



Lancaster University

Assessment

This work will be assessed through a practical demonstration and code inspection of your work.

Be prepared to show your program in your Week 21 practical session, and to answer questions about it posed by your markers.

To gain marks from this assignment:

- **You MUST submit your code to Moodle by the advertised deadline.**
- **You MUST demonstrate your work IN YOUR OWN LAB SESSION.**

FAILURE TO ADHERE TO THE ABOVE MARKS WILL RESULT IN A MARK OF ZERO FOR THIS EXERCISE. THE UNIVERISTY'S TYPICAL LATE SUBMISSION PENALTY DOES NOT APPLY TO THIS EXERCISE.

Marking Scheme

Your work will be marked based on the following five categories. Your final grade will be determined based on a weighted mean of these grades according to the weighting shown in the table below.

Functionality	50%
Code Structure and Elegance	20%
Code Style and commenting	10%
Use of Version Control	10%
Ability to Answer Questions on Code	10%

In all cases a grade descriptor (A, B, C, D, F) will be used to mark your work in each category. The following sections describe what is expected to attain each of these grades in each category.

Markers can also recommend the award of a distinction (+) category overall if they feel a piece of work exhibits clearly demonstrable good programming practice.

Functionality:
A: Fully working program meeting all requirements of Tasks 1, 2, 3, 4 and 5.
B: Working program meeting all requirements of Tasks 1, 2 and 3 and 4.
C: Working program meeting all requirements of Tasks 1, 2 and 3.
D: Working program meeting all requirements of Task 1.
F: No working program demonstrated, or program does not meet requirements of Task 1.

Code Structure and Elegance:
A: Well written, clearly structured code showing student's own examples of good OO practice.
B: Well written, clearly structured code.
C: Clearly identifiable but **occasional** weakness, such as repetitive code that could be removed through use of a loop, poor use of public/private, unnecessary / unused code, inappropriate naming and scoping of variables.
D: Clearly identifiable **systematic** weakness, such as multiple examples of repetitive code that could be removed through use of a loop, systematically poor use of public/private, large sections of unnecessary / unused code, consistently inappropriately named and scoped variables.
F: All the above.

Code Style and Commenting
A: Consistently well indented, well named and well scoped variables with Javadoc commenting.
B: One code block showing poor naming, scope or indentation or occasionally vague and/or inaccurate comments.
C: Two or Three code blocks showing poor naming, scope or indentation, code is partially commented **or** systematically vague and/or inaccurate comments.
D: Four or Five code blocks showing poor naming, scope or indentation or comments.
F: Five or more code blocks showing poor naming, scope or indentation or comments.

Version Control:
A: Project is versioned using a private git repository with a strong and timely commit history
B: Project is versioned using a private git repository with a strong or timely commit history
C: Limited use of git, commits are infrequent or weak commit messages
D: Limited use of git, commits are infrequent and weak commit messages
F: No use of version control

Ability to Answer Questions on Code:
A: All questions answered in detail.
B: Student failed to explain one technical question about their code.
C: Student failed to explain two technical questions about their code.
D: Student failed to answer questions, but could provide a basic overview of their code.
F: No evidence that the student understands the code being demonstrated.

Star Categories:
By showing a demonstrable example of additional work and/or good programming practice, your marker may choose to award a star (*) grade to your work. If you feel your work shows additional merit beyond the specification, it is your responsibility to make your marker aware of this during your in lab marking session. Examples of a star award would be for a functionally complete draughts simulator (including crowning, movement of king pieces and detecting the end of game).