Assignment 3: Noughts and Crosses (OXO) [25%]

This assignment has a weighting of 25% (it **WILL** contribute to your unit mark)

Overview

The purpose of this assignment is to build a "Noughts and Crosses" (OXO) game.
The core Data Model and a Graphical User Interface (GUI) are provided for you.
All you have to do is to write the "Controller" that handles all of the game logic.
As part of this assignment, you will also get the opportunity to practice using exceptions.

JavaFX

This exercise uses JavaFX for the graphical elements of the user interface.
If you have Oracle Java 8 installed, you can build the JavaFX application without external libraries.
If you have OpenJDK installed, you will need to install JavaFX separately and then tell java where to find it:

```
javac -classpath <path_to_jfxrt.jar>:. *.java
java -classpath <path_to_jfxrt.jar>:. OXOView
```

The JavaFX library is installed on the lab machines in a number of locations
We'd recommend pointing java towards `/opt/arduino-ide/1.8/java/lib/ext/jfxrt.jar`

Some of the more recent versions of Java also require you to use the 'module-path' and 'add-modules' flags:

```
javac --module-path <path_to_javafx> --add-modules javafx.controls *.java
java --module-path <path_to_javafx> --add-modules javafx.controls OXOView
```

Model-View-Controller

You will be given a template/skeleton project to help get you started.
This template conforms to the Model-View-Controller (MVC) pattern.

You should **NOT** alter the *OXOModel* or *OXOView* classes, but must only change *OXOController* .
(populating existing methods and include additional ones that you might need).
You should however not change the signatures of any existing methods in the class
(if you change the names or parameters, you may break the marking scripts !)

OXOModel

The *OXOModel* class is the core data structure for the game.
Use the public methods provided by this class to manipulate its internal state:

- The list of players currently playing the game (2 in a standard game !)
- The player whose current turn it is
- The winner of the game (when the game ends)
- Whether or not the game has been drawn (all cells are filled, but no one won !)
- The rows and columns of the game grid
- The "owner" (player who has claimed) each cell in the game grid
- The number of cells in a row required to win the game (3 in a standard game)

When manipulated through its public methods, the state of *OXOModel* will be rendered by the *OXOView* .
(Note: You will NOT need to interface directly to the *OXOView* class from your *OXOController* !)
.

Gameplay

Players will take it in turns to enter a two-character position into the *OXOView* GUI window.
This position consists of the row letter and the column number of the cell they wish to claim.
For example, if a user wished to claim the centre cell, they would type: `b2` This will be automatically passed to the *OXOController* (via the *handleIncomingCommand* method).

On receiving a cell position, your code should:

- Check to make sure the specified cell is a valid one (it actually exists and is still available)
- Notify the system that an invalid cell was requested (if an **invalid** cell was requested)
- Claim the specified cell on behalf of the current player (if a **valid** cell was requested)
- Check if the game has been won by the new move (reporting the winner to the model if so)
- Check if the game is a draw (reporting the draw to the model if so)
- Switch to the next player, ready for the next move (the board view will show who's turn it is)

Exceptions

Three *Exception* classes are provided to enable your code to notify the system of invalid moves:

- Invalid Cell Identifier: The row and/or column characters are not valid (e.g. punctuation marks)
- Cell Does Not Exist: The identifiers are valid characters, but they too big (out of range)
- Cell Already Taken: The specified cell exists, but it has already been claimed by a player

Your *OXOController* should *instantiate* and *throw* instances of these exceptions as appropriate.
These will be detected by the OXOView and appropriate action taken (user informed of the problem).

Win Detection

Your controller should check for wins in all directions (Horizontal, Vertical and Diagonal).
Horizontal, Vertical are relatively easy (Diagonals are a bit more tricky !)

Extra marks will be awarded for flexibility and versatility of your *OXOController*...
You might like to allow grids of any size (not just 3x3).
(you are allowed to alter just one line of the *OXOView* class in order to test this !).
For bigger grids, you can set the number of cells in a row required to win (the *Win Threshold*).

Also, how about more that just two players ?!?

Testing your code

No test file is provided for this assignment...
You are becoming experienced programmers now and should be writing your own.
You should write a suite of tests to help you develop your application.
You might like to use the test classes from previous assignments as a template.

Submission

This assignment **IS** assessed: go to "Assessment, submission and feedback" to submit it.
It is **essential** that you ensure your code compiles and runs before you submit it
(otherwise we will not be able to run it to mark it !)
Just Submit your *OXOController.java* file (nothing else should be needed)
Scripts will be used to automatically test your code to make sure it operates correctly.
Don't change the name of the class or the signatures of any of the methods that already exist
(or we won't be able to test your code !).

Remember: Code quality metrics will be used to assess the "quality" of your code.
This will have an impact on your final mark...
So be sure to adhere to the structure and style guidelines outlined in the lectures.

Plagiarism

You are encouraged to discuss assignments and possible solutions with other students.
HOWEVER it is **essential** that you only submit your own work.
This may feel like a grey area, however if you adhere to the following advice, you should be fine:

- Never exchange code with other students (via IM/email, USB stick, GIT, printouts or photos !)
- Although pair programming is encouraged in some circumstances, on this unit you must type your own work !
- It's OK to seek help from online sources (e.g. Stack Overflow) but don't just cut-and-paste chunks of code...
- If you don't understand what a line of code actually does, you shouldn't be submitting it !
- Don't submit anything you couldn't re-implement under exam conditions (with a good textbook !)

An automated checker will be used to flag any incidences of possible plagiarism.
If the markers feel that intentional plagiarism has actually taken place, marks may be deducted.
In serious or extensive cases, the incident may be reported to the faculty plagiarism panel.
This may result in a mark of zero for the assignment, or perhaps even the entire unit
(if it is a repeat offence).
Don't panic - if you stick to the above list of advice, you should remain safe !