Victoria University of Wellington

School of Engineering and Computer Science
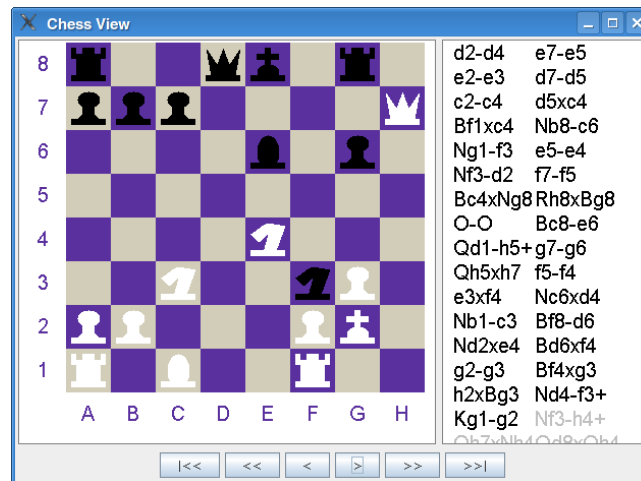
SWEN221: Software Development

Assignment 3 (worth 8%)

Due: Monday 25th May @ Mid-night

1 Introduction

The `ChessView` system is a simple Java application for viewing chess games written in *long algebraic chess notation*. The following shows a screenshot from `ChessView`:

ChessView has only one window, which allows the user to move forward and backward through a chess game. The moves of the game, written in long algebraic chess notation, are given in the rightmost pane of the window; the current state of the chess board is shown in the leftmost pane.

Long Algebraic Notation Long algebraic chess notation is a way for writing down the moves taken during a chess game. The following illustrates the start of a game in this notation:

White	Black
e2-e4	e7-e5
d2-d4	e5xd4
Nb1-c3	Qd8-f6

The first move by White, `e2-e4`, indicates the pawn at position `e2` will advance to position `e4`. When indicating pawn movement, no piece specifier is given. However, when indicating the movement of other pieces a specifier is always given. The specifiers are: `N`=kNight; `B`=Bishop; `R`=Rook; `Q`=Queen; `K`=King. So, for example, White's last move above, `Nb1-c3`, indicates his Knight moves from `b1` to
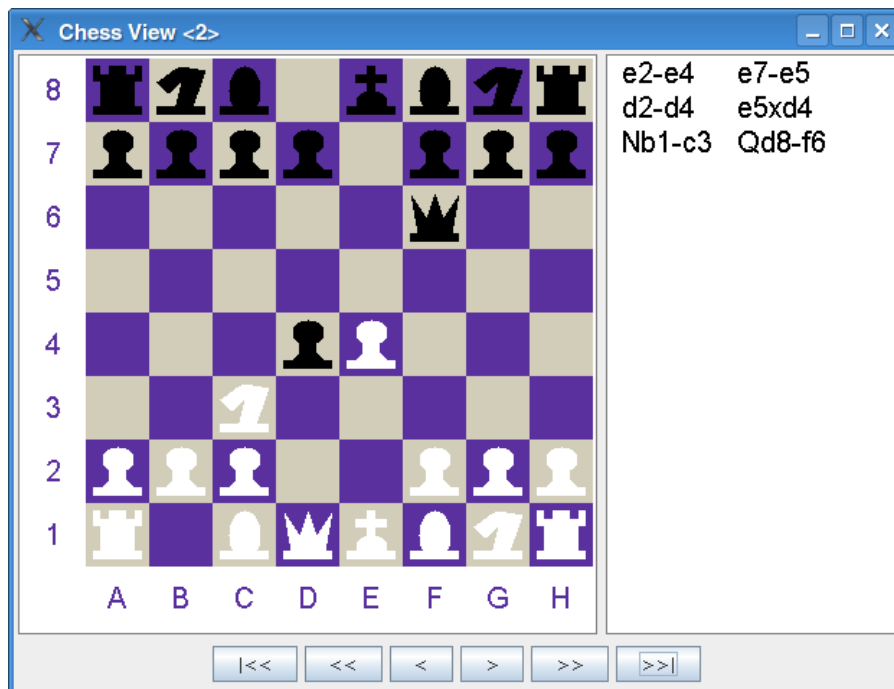
Figure 1: Illustrating the state of a game after 3 rounds.

c3. Finally, a move where one piece takes another is indicated using a x, as in Black's move e5xd4 — the (black) pawn at position e5 takes the (white) pawn at d4. Figure 1 shows the state of the board after the moves in the above game have been made.

There are a number of other moves which can be made during a game of chess, including: *putting a king in check* (e.g Qe1-h4+)), *castling* (e.g. O-O), *pawn promotion* (e.g. b7-b8=N) and *en passant* (e.g. b4xa3ep). If you are not familiar with the rules of Chess, the following links provide an excellent starting point:

> http://en.wikipedia.org/wiki/Chess
> http://en.wikipedia.org/wiki/Chess_notation
> http://en.wikipedia.org/wiki/Algebraic_chess_notation

You can also find many other good resources on the Internet regarding the game of chess.

NOTE: *you do not need to be any good at playing chess in order to complete this assignment.*

Running Chess View

To get started, download the `chessview-buggy.jar` file from the lecture schedule on the course website. You will find several example games are provided as well. As usual, you can run the program from the command-line as follows:

```
java -jar chessview-buggy.jar
```

You should see the ChessView window open up and a chess board being displayed. You should also be able to navigate forward and backward through a game.
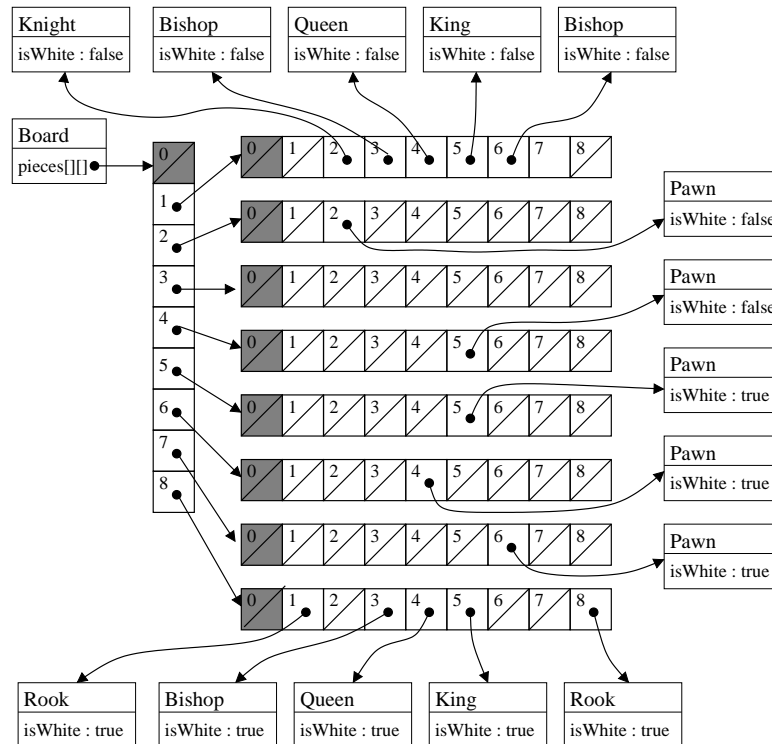
Figure 2: Illustrating an example `Board` object, which contains a 2-dimensional array of pieces. An empty square is indicated by a box with a slash, and this corresponds to the array cell holding `null`. The first element of the row array, and the first element of each column array are unused (marked in gray). This allows the row numbers to line up with those of the normal chess board. Column `a` of the chess board corresponds to index 1 in the column array, and so on for columns `b`, `c`, ..., etc.

Understanding Chess View

When you expand or import the chessview jar file, you should find the following java packages:

```
swen221/chessview/
swen221/chessview/pieces/
swen221/chessview/moves/
swen221/tests/
```
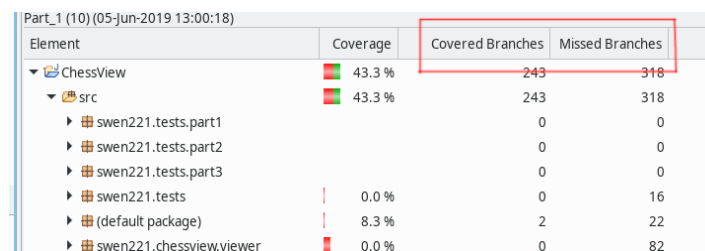
Some notes on these packages:

- The `swen221/chessview/` package contains the high-level game classes, including those for representing the chess board and the game itself.

- The `swen221/chessview/pieces/` packages contains a class for each of the different chess pieces. These contain code related to the movement of the pieces.

- The `swen221/chessview/moves/` packages contains a class for each of the different kind of move that can be made in the game. These contain code related to structuring a move, and ensuring it is valid.

- The `Main` class provides a simple Graphical User Interface for the ChessView system. **NOTE:** you do not need to modify or fix any code in this package.

The class `swen221/chessview/Board` is one of the main classes in the ChessView system. This is responsible for representing the state of the chess board, including the position of all of the pieces. This uses a 2-dimensional array to represent the chess board, where each cell in that array represents a location on the board. The board also provides methods for determining whether a particular diagonal, horizontal or vertical move will be unobstructed. Figure 2 provides a visualisation of a simple `Board` object.

What to Do

Currently, there are a number of tests provided for the chess game in the file `ChessViewTests`. You must add tests to this file with the aim of achieving close to 100% **branch coverage** using Emma and, in the process, identify and fix problems in the code. The following illustrates Emma being used in Eclipse:

In this example, the **branch coverage** obtained for the class `ChessView` is $243/561 = 43.3\%$.

HINT: Make sure to check branch coverage by showing the columns *Covered Branches* and *Missed Branches*

HINT: The purpose of using Emma is to help you find problems in the code base. These problems should be relatively clear (based on the rules of Chess) once you have found them, but have been deliberately obfuscated to make it hard to find them just by looking at the code.

HINT: There are *at least ten* distinct problems that have been planted in the code base. See if you can find them all!

HINT: Your submitted code will be marked against a hidden set of test cases which achieves 100% **branch coverage**.

HINT: The viewer supplied is to help you see the games being played. It is not intended to be tested.

Submission

Your source files should be submitted electronically via the *online submission system*, linked from the course homepage. The minimal set of required files is:

```
swen221/chessview/Board.java
swen221/chessview/ChessGame.java
swen221/chessview/Position.java
swen221/chessview/Round.java
swen221/chessview/moves/Castling.java
swen221/chessview/moves/Check.java
swen221/chessview/moves/EnPassant.java
swen221/chessview/moves/Move.java
swen221/chessview/moves/MultiPieceMove.java
swen221/chessview/moves/NonCheck.java
swen221/chessview/moves/PawnPromotion.java
swen221/chessview/moves/SinglePieceMove.java
swen221/chessview/moves/SinglePieceTake.java
swen221/chessview/pieces/Bishop.java
swen221/chessview/pieces/King.java
swen221/chessview/pieces/Knight.java
swen221/chessview/pieces/Pawn.java
swen221/chessview/pieces/Piece.java
swen221/chessview/pieces/PieceImpl.java
swen221/chessview/pieces/Queen.java
swen221/chessview/pieces/Rook.java
swen221/tests/ChessViewTests.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. *Note, the jar file does not need to be executable.* See the following Eclipse tutorials for more on this:

   http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code.*

3. **All testing mechanism supplied with the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. However, this does not prohibit you from adding new tests. *This is to ensure the automatic marking script can test your code.*

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information.*

Note: Failure to meet these requirements could result in your submission being reject by the submission system and/or zero marks being awarded.

Assessment

This assignment will be marked as a letter grade (A+ ... E), based primarily on the following criteria:

- **Correctness (50%)** — does submission adhere to given specification.

- **Style (50%)** — does the submitted code follow the style guide and have appropriate comments (inc. Javadoc)

As indicated above, part of the assessment for the coding assignments in SWEN221 involves a qualitative mark for style, given by a tutor. Whilst this is worth only a small percentage of your final grade, it is worth considering that good programmers have good style.

The qualitative marks for style are given for the following points:

- **Test Coverage**. This refers to the final amount of coverage obtained, with full marks being awarded for 100% **branch coverage**.

- **Use of Tests**. This refers to the number of test cases written for the assignment. Given the complexity of the problem space, we are expecting 100 (or more) individual test cases.

- **Comments**. This refers to the use of comments in the written test cases which describe the test's purpose. Generally speaking, comments should be used to explain what is happening, rather than simply repeating what is evident from the source code.

Finally, in addition to a mark, you should expect some written feedback highlighting the good and bad points of your solution.