The University of Melbourne
School of Computing and Information Systems

COMP90041
Programming and Software Development
Second Semester, 2019

Fifth Assessed Exercise (lab5)

Submission due Friday, 11 October 2019, 5:00PM

These exercises are to be assessed, and so **must be done by you alone**. Sophisticated similarity checking software will be used to look for students whose submissions are similar to one another.

1. Write an abstract Java class `ChessPiece` that represents a chess piece, and create concrete (*i.e.*, non-abstract) subclasses `Rook`, Bishop, and `Knight`. These classes should all have constructors that take two arguments, `row` and `column`, which should be integers between 1 and 8, and should all have no-argument accessor methods `getRow` and `getColumn` that return the row and column number, respectively.

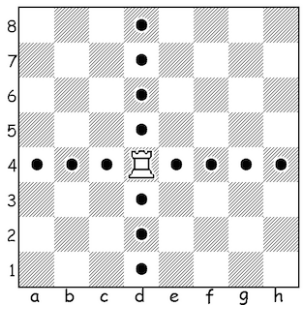
   All three concrete classes should also provide a method

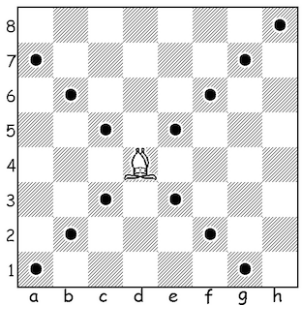
   ```
   public boolean validMove(int toRow, int toColumn)
   ```

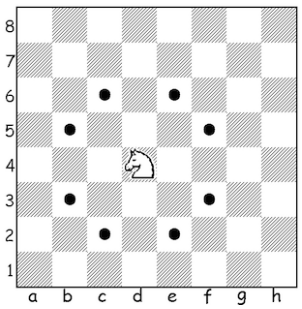
   that returns true if and only if it is valid to move that piece to the specified square. It is invalid to move a piece to the square it is already on, or one that is off the board. Do not consider any other pieces that may be on the chess board in deciding what squares a piece may move to.

   The `toString` method for all three concrete classes should return a string "*piece* `at (`*row*`,`*column*`)`", where *piece* is either `Rook`, `Bishop` or `Knight`, and *row* and *column* comprise the location of the piece. For example, `toString` for a rook at row 3 column 4 would return "`Rook at (3,4)`".

   Recall that in chess, rooks move either vertically or horizontally as far as the edge of the board. Bishops move diagonally (either northeast, northwest, southeast, or southwest) as far as the edge of the board. Knights may move one square vertically and two squares horizontally, or one square horizontally and two squares vertically. The rook, bishop, and knight moves, respectively, may be visualised as:

   A key requirement is that `ChessPiece` should be a Java type that supports the four methods above. For example, it must be possible to do this:

   ```
   ChessPiece piece = new Knight(6,2);
   System.out.println(piece);
   ```

   to print out "`Knight at (6,2)`". Note that the class of the `piece` variable is `ChessPiece`. Declaring it to be of type `Knight` is not good enough.

   Hint: Put as much as possible of the common logic for the three concrete classes in the abstract base class, so the concrete classes contain only what is unique about them.

Submission and Verification

You must submit your project from any one of the student unix servers. Make sure the version of your program source files you wish to submit is on these machines (your files are shared between all of them, so any one will do), then `cd` to the directory holding your source code and issue the command:

```
submit COMP90041 lab5 ChessPiece.java Rook.java Bishop.java Knight.java
```

Important: you must wait a minute or two (or more if the servers are busy) after submitting, and then issue the command

```
verify COMP90041 lab5 | less
```

This will show you the test results and the marks from your submission, as well as the file(s) you submitted.

If your output is different from the expected (correct) output, when you verify your submission you will see the differences between your output and what was expected. This will be shown as some number of lines beginning with a minus sign (`-`) indicating the expected output and some number of lines beginning with a plus sign (`+`) presenting your actual output. There may also be some lines beginning with a single space showing lines you produced that were as expected. Carefully compare the expected and actual lines, and you should be able to find the error in your output. The actual and expected outputs will be aligned, making it easier to find the differences. Some differences are hard to spot visually, however, such as the difference between a capital `O` and a zero (`0`) or the difference between a small `l` and a capital `I` and a one (`1`). This depends on the font you are using. If the only difference between actual and expected output are in whitespace or capitalisation, you will receive partial credit; this is shown in your verification feedback.

Also note that the differences shown only reflect program *output*, not input. Therefore, if your program outputs a prompt, waits for input, and then outputs something else, the differences shown will not include the input, or even the newline the user types to end the input. In that case, the prompt would be shown immediately followed by your program's next output (which may be another prompt), on the same line. This is as expected.

If your program compiles on your computer but the verification output reports that your program does not compile on the server, you may have failed to submit all your files, or you may have named them incorrectly. It is also possible that your program contains a `package` declaration. This would appear near the top of your .java file. If you have such a declaration, your program will probably not compile, so you should delete any such declaration before submitting.

If the verification results show any problems, you may correct them and submit again, as often as you like; only your final submission will be assessed. If your submission involves multiple files, you must submit *all* the files every time you submit.

If you wish to (re-)submit after the project deadline, you may do so by adding "`.late`" to the end of the project name (*i.e.,* `lab5.late`) in the `submit` and `verify` commands. But note that a penalty, described below, will apply to late submissions, so you should weigh the points you will lose for a late submission against the points you expect to gain by revising your program and submitting again. **It is your responsibility to verify your submission.**

Late Penalties

Late submissions will incur a penalty of 1% of the possible value of that submission per hour late, including evening and weekend hours. This means that a perfect project that is a little more than 2 days late will lose half the marks. These lab exercises are frequent and of low point value, and your lowest lab mark will be dropped. Except in unusual circumstances, I will not grant extensions for lab submissions.

Academic Honesty

This lab submission is part of your final assessment, so cheating is not acceptable. Any form of material exchange between students, whether written, electronic or any other medium, is considered cheating, and so is the soliciting of help from electronic newsgroups. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties.