Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Assignment 1

Due: Monday 1st April @ 23:59

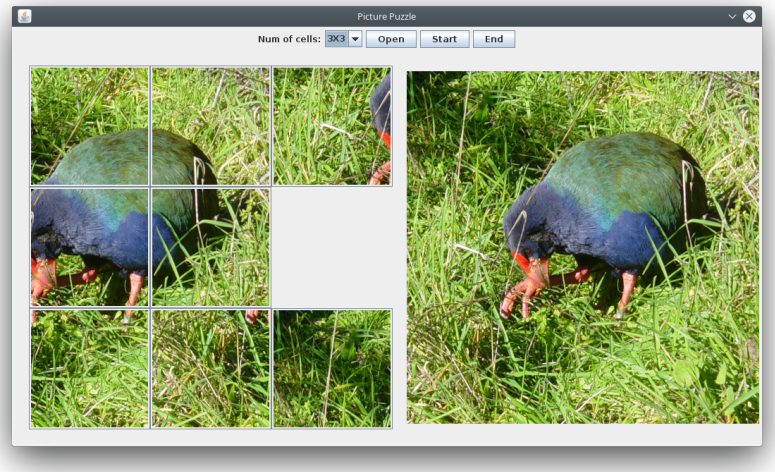
In this assignment you will work with a simple program which implements a "Sliding Picture Puzzle". This is a well-known style of game where an image is divided up into squares which are scrambled and need to be manipulated into the correct order:

`https://en.wikipedia.org/wiki/Sliding_puzzle`

The program in this assignment has a relatively simple class hierarchy. Nevertheless, understanding the flow of control through the program will still be challenging at times and will test your debugging skills.

Picture Puzzle

The picture puzzle game provides a simple GUI where you can load an image, choose the difficulty of the game (e.g. `3x3` vs `4x4`) and then play the game. The puzzle is shown in the left panel, with the solution shown on the right:



The game is relatively straightforward to play, though there are some variations. In particular, we can *rotate* squares by right-clicking on them. This makes the game slightly harder to play, and also makes the program code more interesting (though also more complex).

Getting started

To get started, download the `picturepuzzle.jar` file from the lecture schedule on the course website. As usual, you can run the program from the command-line as follows:

```
java -jar picturepuzzle.jar
```

A simple GUI should appear on your screen, and you should be able to play the game. *Remember, however, that at this stage the game contains a number of bugs and missing features.* For example, pieces may not move in the direction you are expecting!

When you import the `picturepuzzle.jar` file, you should find the following Java packages:

- The `swen221/picturepuzzle/gui/` package contains the graphical user interface and the main method. **You do not need to understand the inner workings of this in order to complete the assignment. NOTE:** you do not need to modify any code in this package.
- The `swen221/picturepuzzle/model/` package contains the class `Game` encoding the logic of a game, and the class `Board` representing the current state of the board.
- The `swen221/picturepuzzle/moves/` package contains a class for the two different kinds of move that can be made in the game (move and rotation). These contain code related to structuring a move, and ensuring it is valid.
- The `swen221/picturepuzzle/tests/` packages contains many jUnit tests to check your implementation of the game. **NOTE:** To make the automatic marking possible, you can not modify the files already present in this folder, but you may add your tests in a separate file (e.g. `MyTests.java`).

Part 1 — Small Boards (20%)

The first objective is to make sure the game correctly implements simple *moves* on small (i.e. 2×2) boards (for now, ignoring rotations). There is a simple *defect* in the Location class which you must identify and fix. Tests for this part is provided in `swen221/picturepuzzle/tests/Part1Tests.java`.

Part 2 — Big Boards (30%)

The second objective is to make sure the game correctly implements simple *moves* on larger (e.g. 3×3) boards (for now, still ignoring rotations). There are several defects spread across the Board and Move classes which you must identify and fix. Tests are provided in `swen221/picturepuzzle/tests/Part2Tests.java`.

Part 3 — Rotation Moves (30%)

The third objective is to implement the *rotation* move in the game. This is a tricky little algorithm to get right, though the test cases should help. You will need to implement the method Rotation.apply(). Tests for this part are provided in `swen221/picturepuzzle/tests/Part3Tests.java`.

Part 4 — Game Over (20%)

The final objective is to implement the test to determine when the game is over. This requires changes to the Board class. Tests are provided in `swen221/picturepuzzle/tests/Part4Tests.java`.

Submission

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

```
swen221/picturepuzzle/model/Location.java
swen221/picturepuzzle/model/Board.java
swen221/picturepuzzle/model/Cell.java
swen221/picturepuzzle/model/Game.java
swen221/picturepuzzle/model/Operation.java
swen221/picturepuzzle/moves/Move.java
swen221/picturepuzzle/moves/Rotation.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 of Part 1 (30%)** — does submission adhere to specification given for Part 1.
- **Correctness of Part 2 (20%)** — does submission adhere to specification given for Part 2.
- **Correctness of Part 3 (20%)** — does submission adhere to specification given for Part 3.
- **Correctness of Part 4 (20%)** — does submission adhere to specification given for Part 4.
- **Style (10%)** — 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:

- **Division of Concepts into Classes**. This refers to how *coherent* your classes are. That is, whether a given class is responsible for single specific task (coherent), or for many unrelated tasks (incoherent). In particular, big classes with lots of functionality should be avoided.
- **Division of Work into Methods**. This refers to how well a given task is split across methods. That is, whether a given task is broken down into many small methods (good) or implemented as one large method (bad). The approach of dividing a task into multiple small methods is commonly referred to as *divide-and-conquer*.
- **Use of Naming**. This refers to the choice of names for the classes, fields, methods and variables in your program. Firstly, naming should be consistent and follow the recommended Java Coding Standards (see `http://g.oswego.edu/dl/html/javaCodingStd.html`). Secondly, names of items should be descriptive and reflect their purpose in the program.
- **JavaDoc Comments**. This refers to the use of JavaDoc comments on classes, fields and methods. We certainly expect all `public` and `protected` items to be properly documented. For example, when documenting a method, an appropriate description should be given, as well as for its parameters and return value. Good style also dictates that `private` items are documented as well.
- **Other Comments**. This refers to the use of commenting within a given method. Generally speaking, comments should be used to explain what is happening, rather than simply repeating what is evident from the source code.
- **Overall Consistency**. This refers to the consistent use of indentation and other conventions. Generally speaking, code must be properly indented and make consistent use of conventions for e.g. curly braces.

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