Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Assignment 7

Due: Sunday 7th June @ Mid-night

1 Introduction

You are tasked with testing a simple implementation of the well-known *Monopoly* board game. If you have not heard of the game Monopoly before, don't worry — a simple introduction is provided below.

NOTE: you do not need to improve or fix the Monopoly game in any way, as this is working correctly.

1.1 Monopoly Board

A Monopoly board is divided into two main areas: the border region; and, the inner region. There are 40 locations on the board, each of which is either: a property or a special area. Properties have a name, a price, a mortgage value, a rental value and can be bought or sold; the special areas are: "Go", "Free Parking", "Jail", "Goto Jail", "Chance" and "Community Chest".

A property is either: a *street*, a *railway* or a *utility*.

Streets have a colour and are organised into colour groups, consisting of two or three streets. Each street can have up to five houses or one hotel built on them; houses may only be built on a street when all the streets in its colour group are owned by the same player; a hotel may be built on a street which has five houses and, when built, all houses on that street are removed. The cost of building a house or hotel on a particular street depends upon the street in question. The rental value of a street depends upon its base rate and the number of houses/hotels on that street. The calculation is:

$$\text{Street rent} = \text{base rate} + 25 * (\text{number of houses}) + 200 * (\text{number of hotels})$$

The cost of constructing houses or hotels depends upon the street in question. For those on the south side (e.g. Euston), the cost is $50 per house, and $150 per hotel; for those on the west side (e.g. Bow Street), the cost is $100 per house, and $200 per hotel; for those on the north (e.g. The Strand) it's $150 per house, and $250 per hotel; finally, those on the east side (e.g. Mayfair) it's $200 per house, and $300 per hotel.

There are four railways in the board: "Marylebone station", "Fenchurch Street station", "Liverpool Street station" and "King's Cross station". The rental cost for a station depends upon how many stations are owned by the player. The calculation is:

$$\text{Station rent} = \text{base rate} * \text{number of stations owned}$$

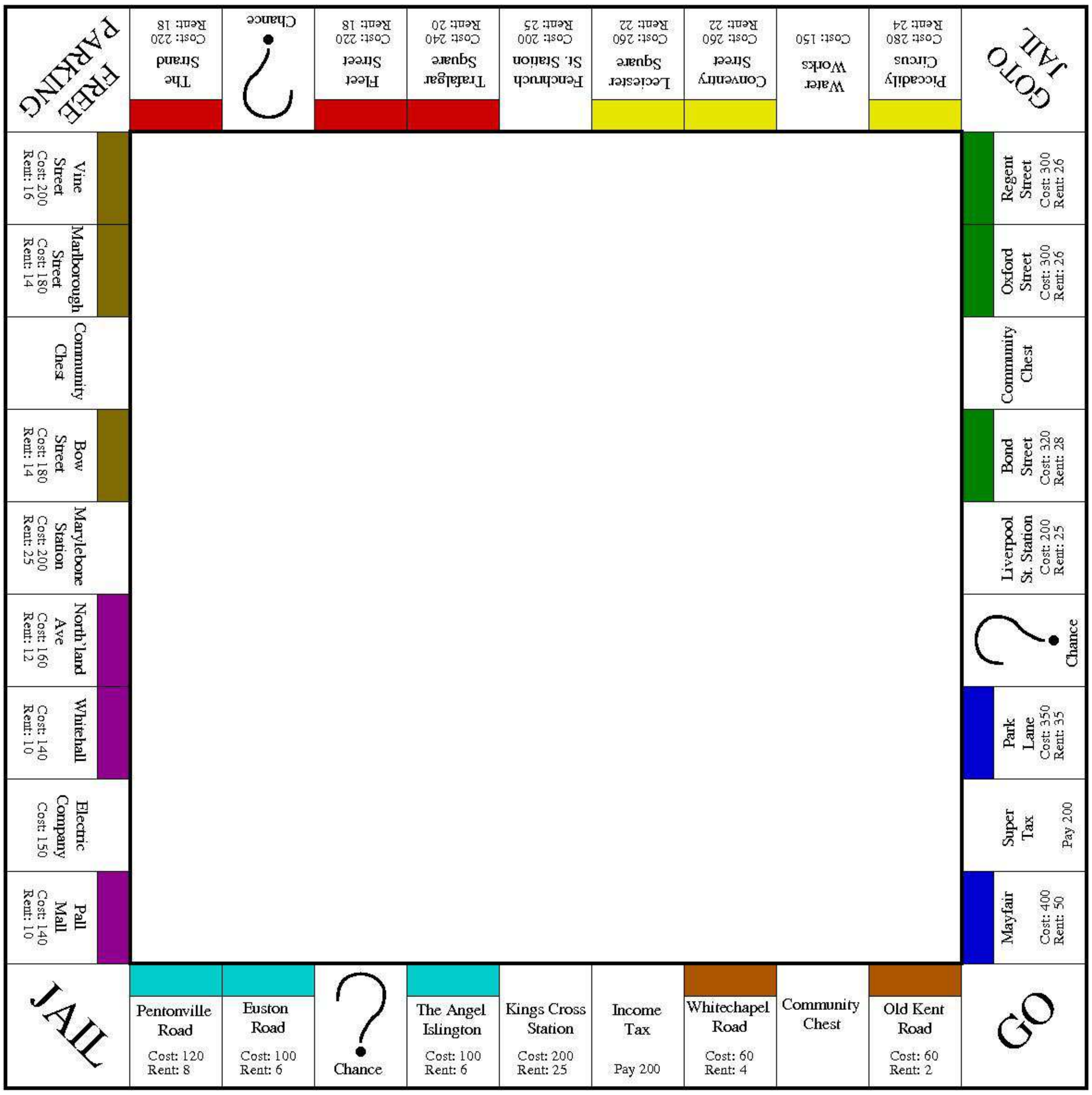

GO
Old Kent Road Cost: 60 Rent: 2
Community Chest
Whitechapel Road Cost: 60 Rent: 4
Income Tax Pay 200
Kings Cross Station Cost: 200 Rent: 25
The Angel Islington Cost: 100 Rent: 6
Chance
Euston Road Cost: 100 Rent: 6
Pentonville Road Cost: 120 Rent: 8
JAIL
Pall Mall Cost: 140 Rent: 10
Electric Company Cost: 150
Whitehall Cost: 140 Rent: 10
North'land Ave Cost: 160 Rent: 12
Marylebone Station Cost: 200 Rent: 25
Bow Street Cost: 180 Rent: 14
Community Chest
Marlborough Street Cost: 180 Rent: 14
Vine Street Cost: 200 Rent: 16
FREE PARKING
The Strand Cost: 220 Rent: 18
Chance
Fleet Street Cost: 220 Rent: 18
Trafalgar Square Cost: 240 Rent: 20
Fenchurch St. Station Cost: 200 Rent: 25
Leciester Square Cost: 260 Rent: 22
Conventry Street Cost: 260 Rent: 22
Water Works Cost: 150
Piccadily Circus Cost: 280 Rent: 24
GOTO JAIL
Regent Street Cost: 300 Rent: 26
Oxford Street Cost: 300 Rent: 26
Community Chest
Bond Street Cost: 320 Rent: 28
Liverpool St. Station Cost: 200 Rent: 25
Chance
Park Lane Cost: 350 Rent: 35
Super Tax Pay 200
Mayfair Cost: 400 Rent: 50

Figure 1: The Monopoly Board

There are two utilities in the board: the “Water Works” and the “Electric Company”. Again, the rental cost for these depends upon how many are owned by the player:

$$\text{Utility rent} = \text{value of last dice roll} * 4 * \text{number of utilities owned}$$

Properties can be mortgaged through the bank at any time; when this happens, the bank loans the player half the value of the property. However, rent cannot be collected on mortgaged properties. To unmortgage a property, the player must pay back the loan, plus 10% interest. A property may only be mortgaged once. Real estate may be sold to another player for an agreed price at any time.

An illustration of the Monopoly board is given in Figure 1. This includes the price and (base rate) rental cost of each piece of real estate and identifies all the special zones. Tip: you might find it useful to print this!

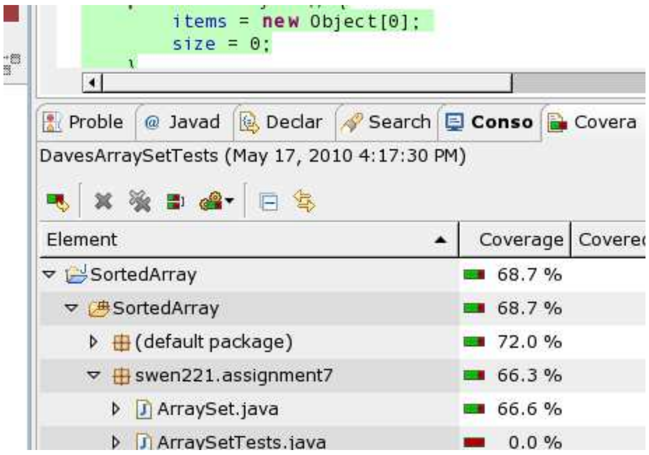
1.2 Playing the Game

Each player in the game has a token which is placed onto the location the player is currently visiting. As the game proceeds, the player’s token will move from location to location. There are seven tokens: the ”Hat”, the ”Car”, the ”Dog”, the ”Wheelbarrow”, the ”Thimble”, the ”Boot” and the ”Iron”.

Each player maintains a list of properties they own, and the amount of cash they have left. When it’s their turn, the player roles the dice and moves the number of locations equal to their sum in the clockwise direction. If the person lands on a property owned by another, he/she must pay its rent to the owner (note, this only applies if the owner is not in jail). If the player lands on an unowned property, they have the option to purchase it for the stated price.
NOTE: for fun, you can play the game using the simple `TextClient` interface provided.

2 What to Do

Currently, there is one test provided for the monopoly game in the file `MonopolyTests`. You must add tests to this file for the classes in package `swen221.monopoly` , such that you achieve as close to 100% statement coverage as possible using EMMA. The following illustrates EMMA being used in Eclipse:



```
        items = new Object[0];
        size = 0;
    }
```

DavesArraySetTests (May 17, 2010 4:17:30 PM)

Element	Coverage	Covere
SortedArray	68.7 %	
SortedArray	68.7 %	
(default package)	72.0 %	
swen221.assignment7	66.3 %	
ArraySet.java	66.6 %	
ArraySetTests.java	0.0 %	

In this example, we see that the coverage obtained for the class `ArraySet` is 66.6%.

HINT: The closer you get to 100%, the better mark you will be awarded. You should find it relatively easy to get a score above 80%, but getting very close to 100% will be harder. Note, however, that the scale is *non-linear* (i.e. 80% coverage does not mean a mark of 80%). For example to score an A+, you will need to obtain *at least* 99% coverage. Similarly, to score a C grade, you will need to obtain *at least* 80% coverage.

3 Submission

Your program code should be submitted electronically via the *online submission system*, linked from the course homepage. 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**. See the export-to-jar tutorial linked from the course homepage for more on how to do this. *Note, the jar file does not need to be executable.*

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 tests should be provided in the file `MonopolyTests`.** An example test has been provided to illustrate. *This is to ensure the automatic marking script can determine the coverage obtained from your tests.*

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 you getting zero marks for the assignment.

4 Assessment

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

- **Correctness** — does the program (in this case, the tests that you write) adhere to the given specification?

- **Style** — does the 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.
- **Additional Tests**. This refers to the presence of additional tests being written. Generally speaking, additional tests should be written to account for limitations in those tests provided.
- **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.