Assignment 1

Goal: The goal of this assignment is to gain practical experience with data abstraction and using the Java API.

Due date: The assignment is due at **5pm on Wednesday 14 April**. Late assignments will lose 20% of the total mark immediately, and a further 20% of the total mark for each day late.

Problem: In this assignment, and the following two assignments in this course, you will develop the component classes of a simple spell checker. The spell checker will allow the user to find and correct misspelled words in text files.

In this assignment, you will create classes for the main data abstractions required for the spell checker:

Document – stores the contents of a text file as a sequence of numbered lines along with any spelling errors in the line and, for each error, a list of possible corrections. The String representation of this class (as returned by the toString method) should be of the form:

```
0  This is a three line document.
Errors: []
1  This si the scond line.
Errors: [si(5-6): [so, is], scond(12-16): [second, scone]]
2  This is teh last line.
Errors: [teh(8-10): [ten, tea, tee, the]]
```

where the number range in brackets denotes the start and end position of the error in the line, e.g., si starts at position 5 and ends at position 6 above (the first position is position 0). The misspelled words are listed in the order they occur in the line.

Line – represents a single line of a document including its spelling errors and their possible corrections. The String representation of this class should be of the form:

```
This si the scond line.
Errors: [si(5-6): [so, is], scond(12-16): [second, scone]]
```

where again the number range in brackets denotes the start and end position of the error in the line, e.g., si starts at position 5 and ends at position 6 above, and the misspelled words are listed in the order they occur in the line.

Error – stores the details of an error including its position on the line in which it occurs, and a list of possible corrections. The String representation of this class should be of the form:

```
teh(8-10): [ten, tea, tee, the]
```

where the number range in brackets denotes the start and end position of the error in a line, i.e., teh starts at position 8 and ends at position 10 above.

Practical considerations: Starter files for the classes are on the course website and must be imported to a package called assignment1. These files include Javadoc specifications of the constructors and methods you need to complete. You need also to provide instance variables and import clauses as required.

You must implement these classes as if other programmers were, at the same time, implementing the classes that instantiate them and call their methods. Hence:

- You must not change the specifications provided, or method names, parameters and return types.
- You may override methods inherited from class Object, e.g., toString, but may not otherwise add any new methods except private methods.

You also must implement these classes as if other programmers are going to be maintaining them. Hence, to improve readability:

- You should use private methods to stop any method doing too much.
- Private methods must be commented using Javadoc.
- Allowable values of instance variables must be specified using a class invariant when appropriate. (If you have no class invariant in a class then the checkInv method should simply return true.)

Two exception classes required by the classes you implement are also provided on the course website[1]. You do not need to modify or submit these.

To test your completed classes, test drivers are also provided on the course website. As given, these files are not meant to adequately test your classes. You should edit them to create your own test cases to ensure your classes have been tested thoroughly before you submit. These files should not be submitted.

Note that since Document references Line, and Line references Error, you need implementations of Line and Error in order to test Document. Similarly, you need an implementation of Error in order to test Line. You can either use your own implementations of Line and Error if you have completed these, or use *test stubs*, i.e., simple classes which are used in place of the classes which are referenced. For example, a stub for Error might implement getStart and getEnd as always returning 0, getWord as returning the empty String, etc. Such stubs allow you to compile the class you wish to test, and test its basic functionality.

[1] In the exception classses, you will notice a line @SuppressWarnings("serial"). This is a Java annotation to suppress a warning which Eclipse would otherwise give regarding the serialisation of objects of the class. We will not be covering annotations, or serialisation in this course. If you are interested in what these terms mean, you can find explanations on the Internet. Otherwise, simply ignore the line in these files, and in future assignment files in which it appears.

Submission: Submit your classes Document, Line and Error electronically using the website:

http://submit.itee.uq.edu.au/

You can submit your assignment multiple times before the assignment deadline but only the last submission will be saved by the system and marked.

Evaluation: Your assignment will be given a mark out of 10 according to the following marking criteria.

Testing (5 marks)

- All of our tests pass — 5 marks
- At least 3/4 of our tests pass — 4 marks
- At least 1/2 of our tests pass — 2 marks
- At least 1/4 of our tests pass — 1 mark
- Work with little or no academic merit — 0 marks

Code quality (5 marks)

- Code that is clearly written and commented, and satisfies the specifications and rules of data abstraction — 5 marks
- Minor problems, e.g., lack of commenting or private methods — 3-4 marks
- Major problems, e.g., code that does not satisfy the specification or rules of data abstraction, or is too complex, or is too difficult to read or understand — 1-2 marks
- Work with little or no academic merit — 0 marks

Note you will lose marks for code quality

a) for failing to comment variable and constant declarations (except for the index variable of a for-loop),

b) failing to comment tricky sections of code,

c) inconsistent indentation, and

d) lines which are excessively long (lines over 80 characters long are not supported by most printers).

School Policy on Student Misconduct: You are required to read and understand the School Statement on Misconduct, available on the School's website at:

http://www.itee.uq.edu.au/about_ITEE/policies/student-misconduct.html

This is an individual assignment. If you are found guilty of misconduct (plagiarism or collusion) then penalties will be applied.

If you are under pressure to meet the assignment deadline, see the lecturer **as soon as possible**.