CSSE2002 / CSSE7023 **Semester 1, 2010**

Assignment 2

Goal: The goal of this assignment is to gain practical experience with procedural abstraction and file I/O.

Due date: The assignment is due at **5pm on Friday 5 May**. 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, you will continue implementing the spell checker started in Assignment 1. Specifically, you will implement two more classes: Dictionary, used to store a set of correctly spelled words, and SpellChecker, the class which will interface with the spell checker's GUI.

As well as being used to check if a word is spelled correctly, the Dictionary can be used to suggest corrections for a misspelled word. Suggested corrections will be based on four simple spelling errors:

1. Insertion errors correspond to a single character being inserted into a word, e.g., "missspelled" instead of "misspelled".
2. Deletion errors correspond to a single character being deleted from a word, e.g., "scond" instead of "second".
3. Substitution errors correspond to a single character in a word being substituted by another, e.g., "scond' instead of "scone".
4. Transposition errors correspond to two adjacent characters being swapped in a word, e.g., "teh" instead of "the".

We will only consider corrections that correspond to exactly one of these errors, and not combinations of them.

SpellChecker will allow us to load the words in any text file into a Dictionary. The words that are loaded correspond to the strings within the file that are separated by space characters (spaces or tabs) and new line characters with any leading or trailing non-letter characters (punctuation and digits) removed. For example, a text file with the line:

"It's a little hit-and-miss, I'm afraid!"

will result in the following six words being loaded into the Dictionary.

It's
a
little
hit-and-miss
I'm
afraid

SpellChecker will also allow us to load a text file into a Document. Spelling errors are detected by extracting words from the text file in the same fashion as above, and comparing them with the words in the Dictionary.

Practical considerations: Starter files for the classes are on the course website and must be imported to a package called assignment2. These files include specifications of the

constructors and methods you need to complete. You need also to provide instance variables and import clauses as required.

As in Assignment 1, 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. (toString may return any reasonable string representation for the classes in this assignment.)

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

- You are expected to use private methods to stop any method doing too much.
- Private methods must be commented using preconditions and postconditions.
- 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.)
- All non-trivial for and while loops should include a loop invariant. (It is expected that loops in the methods for generating corrections, and for extracting words from text files will require loop invariants.)

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

To test your classes, you will need to build test drivers. You are encouraged to use JUnit to simplify this task. Your test drivers will not be assessed and should not be submitted.

Submission: Submit your classes Dictionary and SpellChecker 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 marked.

Evaluation: Your assignment will be given a mark out of 15 according to the following marking criteria.

Testing of Dictionary (4 marks)

- All of our tests pass — 4 marks
- At least 3/4 of our tests pass — 3 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

Testing of SpellChecker (4 marks)

- All of our tests pass — 4 marks

- At least 3/4 of our tests pass — 3 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 (7 marks)

- Code that is clearly written and commented, and satisfies the specifications — 7 marks
- Minor problems, e.g., lack of commenting or private methods — 4-6 marks
- Major problems, e.g., code that does not satisfy the specification, or is too complex, or is too difficult to read or understand — 1-3 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 an assignment deadline, see the lecturer **as soon as possible**.