CSSE2002/7023 – Programming in the Large
Semester 1, 2010
Assignment 3

Goal: The goal of this assignment is to gain practical experience with implementing GUIs and with the MVC architecture.

Due date: The assignment is due at **5pm on Wednesday 26 May**. Late assignments will lose 20% of the total mark immediately, and a further 20% of the total mark for each day late.

Problem description: In this assignment you will develop a GUI for the spell checker implemented in Assignments 1 and 2. The GUI will have the following components.

- A text field (labelled "Dictionary:") and a text field (labelled "Document:") for entering the name of the dictionary file and document to be checked, respectively.
- A button (labelled "Load") for loading the dictionary and document files in the text fields. Note that this button loads both files and so both text fields must have a valid file name in them for the load to succeed.
- An unlabelled text area for displaying the next error in a loaded document. Before a document is loaded, the text area should be blank. After a document is loaded, if there are errors then the first error should be displayed (in black text) as the line from the document in which the error occurs followed on a new line by "Error: " and the error. For example:

  ```
  The cath sat on the mat.
  Error:  cath
  ```

  If there are no errors then the text area should blank.

  The user is not allowed to type into the text area.
- An unlabelled text area for displaying messages to the user, including error messages. Initially, this text area is blank.

  The user is not allowed to type into the text area.
- A text field (labelled "Correction:") for manually entering a correction for a displayed error. This correction need not be a word that is in the dictionary. The user is not allowed to type in this text field when no error is displayed in the error text area.
- A combo box for selecting a correction. The combo box should list all the corrections from the dictionary (where corrections are as defined in previous assignments). On selection, the selected correction should automatically appear in the "Correction" text field.
- A button (labelled "Correct") for correcting the error with the correction in the text field. After pressing the button the error text area should display two lines (in black text) corresponding to the next error (as above) or, when there are no further errors, should be blank. The message text area should be unchanged if there is another error or, when there are no further errors, display the single line "Spell check complete." (in blue text). When either text area is updated, the next error or the "Spell check complete." message is not appended to the text already in the text area, but replaces it.

Note that if a user corrects an error using a word that is not in the dictionary, or decides to ignore an error (using the "Ignore" button described below) then it is no longer regarded as an error. Hence "Spell check complete." should be displayed after the user has responded to each error in the original document.

Pressing the "Correct" button when no error is displayed in the text area has no effect.

- A button (labelled "Ignore") for not correcting the displayed error. The displayed error should be regarded as corrected, and the text areas should be updated as for the "Correct" button. Pressing this button when no error is displayed in the text area has no effect.
- A text field (labelled "Save as:") for entering the name of a file to which to save the loaded document. The user is not allowed to type in this text field when a document has not been loaded.

  A button (labelled "Save") for saving the loaded document to the file in the text field. After the file is saved, the text area should display (in blue text) the single line

  Document saved.

  Pressing this button when no document has been loaded has no effect.

The GUI should catch any exceptions thrown by the spell checker and display an appropriate error in red text in the text area.

Practical considerations: The GUI will be implemented using an MVC architecture. Starter files for the required classes are on the course website and must be imported to a package called assignment3.

The classes SpellCheck (the class with the main method) and LoadPanel are complete. You should not modify or submit these classes.

The remaining classes SpellCheckerView (the view component of the MVC architecture), SpellCheckerController (the controller component of the MVC architecture), and OutputPanel, CorrectionPanel and SavePanel (panels which, together with LoadPanel, provide structure to the view component) include only instance variables. You must add the required constructors and methods.

You may also add further instance variables, but must use the provided variables without modification of their names or types. (If you fail to do this, our automatic test driver won't work and you will lose marks.) You may also add further import clauses, but do not change those that are already there. There is no need to add a toString method or an invariant to any class in this assignment (although you won't be penalised for doing so).

You 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 Javadoc. The use of Javadoc tags such as @param are optional in this assignment.
- Your view and controller components must adhere to their expected roles in the MVC architecture.

The appearance of your GUI in terms of layout, colors, fonts, etc, is not important and won't be taken into consideration when marking.

Submission: Submit your classes SpellCheckerView, SpellCheckerController, OutputPanel, CorrectionPanel and SavePanel 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 the GUI (8 marks)

- All of our tests pass — 8 marks
- At least 3/4 of our tests pass — 6 marks
- At least 1/2 of our tests pass — 4 marks
- At least 1/4 of our tests pass — 2 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**.