CS1003 Week 5 Practical: Text Processing

Due Thursday, week 5 (28 February 2019) – weighting 20%

This practical is worth 20% of the overall coursework mark. It is due on Thursday 28th February 2019 (week 5) at 21:00. As for every practical, you should arrive in the lab having prepared in advance, by studying this specification and reviewing relevant course material.

In this practical, you are going to write a small program that generates *"lost consonants"* versions of given sentences. From its Wikipedia page: "Lost Consonants is a comic collage series created by Graham Rawle, appearing in Britain's Guardian newspaper from 1990 to 2005. The text and image word play series illustrates a sentence from which one vital letter has been removed, altering its meaning."

As a simple example, consider the phrase "barking dog". There are several consonants in this sentence (b, r, k, n, g, d, g), so there are several alternative lost consonant style phrases that can be constructed. Your program will systematically produce all alternatives by dropping one consonant at a time. After dropping a consonant your program will check against a dictionary, and discard any alternatives that contain a word not listed in the dictionary.

The phrase "barking dog" contains 7 consonants, hence there are initially 7 candidates.

- arking dog	X
- baking dog	✓
- baring dog	✓
- barkig dog	X
- barkin dog	X
- barking og	✓
- barking do	✓

Some of these candidate phrases contain words that are not valid words (marked with X above). Your program is supposed to only print alternatives that exist in a dictionary (marked with ✓ above).

For this practical, you may develop your code in any IDE or editor of your choice, however, you must ensure you're your source code is in a folder named **W05Practical/src** and your main method is in a file called **LostConsonants.java** .

Basic requirements

- Accept two arguments from the command line:
  - First argument is the path of the file containing the dictionary. The dictionary file will be plain text, containing one word per line.
  - Second argument is a word, phrase or sentence, given as a single source String. The lost consonant alternatives will be generated from this String.
  - An appropriate message to be printed when the command line arguments are invalid. (See stacscheck test cases for expected messages)
- Read the dictionary (list of words) from the provided file, found here:
  https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W05/Tests/data/words
- For each consonant in the source String, produce a candidate String that is completely identical to the source String except losing the consonant. Only print the sentence to standard output if losing the consonant results in a word that is found in the dictionary.

Extensions

- Do not edit your main submission. In a separate file, write a second version of your program that lists lost vowel alternatives.
- Again, in a separate file, write a version of your program that adds consonants instead of removing them. Are you able to recover the original phrases using this program? i.e. how hard would it be to go from "barking dog" to "baking dog" using your basic program, and back to "barking dog" again using this program? There are many more ways of adding a consonant to a phrase than there are ways of removing them, so do not run this program on very long phrases unless you are happy to wait some time for your program to finish! If you attempt this extension, please briefly explain your approach and compare it with the basic program in your report. (See the Deliverables section below)

Implementation

You are going to read text from a file to construct a dictionary. The dictionary file contains one word per line, provided at the following location:

https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W05

When running your program on files in different directories remember to use the absolute or relative path to the file. For instance, if you are running your program from `W05Practical/src` but your test files are in `W05Practical`, you could use the following command, specifying the relative path to the dictionary file:

```
java LostConsonants ../words "test phrase"
```

Note: you must include the quote marks around the file name if there are spaces in the file name/path.

To see examples and example outputs, examine the tests provided (see Automated Checking below). Make use of the stacscheck tests when developing your solution, they are provided to help you!

As with any programming practical you will probably need to look at the Java API, found at
http://docs.oracle.com/javase/8/docs/api/

Specifically for this practical you may find the documentation of the *String*, *StringBuilder*, and *List* classes particularly useful in showing you how to find out whether a word is contained in a list, how to remove a character from a string using StringBuilder.deleteCharAt, etc. You may want to explore the use of a *Set* class as well for representing the dictionary.

Automated Checking

You are provided with some basic unit test to check your code provides the required functionality. The tests can be found at `/cs/studres/CS1003/Practicals/W05/Tests.`
In order to run the automated checker on your program, open a terminal window connected to the Linux lab clients/servers and **change directory** to your **W05Practical** directory and execute the following command:

```
stacscheck /cs/studres/CS1003/Practicals/W05/Tests
```

If the automated checker doesn't run, or the build fails, or all tests fail, you may have mis-typed a command or not have followed the instructions above.

You should open the provided test files and make sure you understand what the tests are doing and try to come up with some interesting tests of your own. You can run your own via the command line, or via the automated checker. You can create new tests in a local sub-directory in your assignment directory and run `stacscheck` with your own directory as the argument e.g.

```
stacscheck ~/W05Practical/MyTests
```

You should also look at the documentation for the automated checker at:

https://studres.cs.st-andrews.ac.uk/Library/stacscheck/

Deliverables

Hand in via MMS to the Week 5 Practical slot, as single .zip file containing your entire W05Practical directory which should contain:

- All your Java source files as specified above
- A brief report containing the usual sections for *Overview, Design & Implementation, Testing, Evaluation* (against requirements), *Conclusions* (your achievements, difficulties and solutions, and what you would have done given more time). You can use any software you like to write your report, but your submitted version must be in PDF format.

Marking

A very good attempt at satisfying the basic requirement together with the first suggested enhancement above can achieve a mark of 14 - 16. This means you should produce very good, re-usable code demonstrating a very good decomposition of the problem into sensible methods with sensible parameters. To achieve a 17 or above, your code must in addition make a very good attempt at the second suggested enhancement or other additional enhancements that you come up with. See the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:

https://www.st-andrews.ac.uk/students/rules/academicpractice/