Laboratory Sheet 8

This Lab Sheet contains material based on Lectures 1 – 15 (up to *12 November 2014*), and contains the submission information for Laboratory 8 (week 9, *17 – 21 November 2014*).

The deadline for submission of the lab exercise is 48 hours after the end of your scheduled laboratory session in week 9 *(17 –21 November 2014).*

You may submit work that is incorrect or incomplete. In order to stretch the stronger members of the class, some of the laboratory exercises are quite challenging, and you should not be discouraged if you cannot complete all of them.

Aims and objectives

- to gain experience of file I/O in Java
- to practice String and character manipulation
- to reinforce object-oriented concepts

Set up

When you download Laboratory8.zip from moodle, please unzip this file. You will obtain a folder Laboratory8, containing file words.txt and a folder named words. In Eclipse, create a new Java project called lab8. In this project, create a new package called words. Now drag the three Java source code files from the words folder into the words package in Eclipse. Drag the words.txt file into the project (but not the package).

You will need to add a new Java class called SimpleWordAnalyzer, which implements the WordAnalyzer interface.

Submission material

Submission exercise 8

You are given a text file words.txt that contains one word per line. For each letter of the alphabet, your code needs to find one of the longest words that begins with that letter, and a word that has the most occurrences of that letter. e.g. for the letter b, the longest word in words.txt is bbbbbb, for letter a, the word that contains the most a characters is aardvark.

Your constructor for SimpleWordAnalyzer should take a `String` specifying the filename. The method `longestWordStartingWith` will open the file using the BufferedReader class and read in each line, convert it to lower case letters, and look for the longest word beginning with the specified character. The method `wordWithMostOccurrencesOf` will open the file using the BufferedReader class and read in each line, looking for the word that has the most occurrences of the specified character. If there is no matching word, then the methods will return an empty String, i.e. "".

You can test SimpleWordAnalyzer using the TestWordAnalyzer JUnit tests. Remember to add the JUnit library to your project, as in previous weeks

The Driver class has a `main` method should take a single `String` argument that specifies the name of an input text file, containing one word per line. Please check http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Ftasks%2Ftasks-java-local-configuration.htm for how to specify arguments to `main`. You might want to use the supplied words.txt file – or download a more interesting file such as the Linux words file at http://www.cs.duke.edu/~ola/ap/linuxwords . This is the output I get from the Driver class with the Linux words file:

```
a antidisestablishmentarianism adiabatically
b beautifications babbage
c characteristically acceptance
d deterministically added
e electroencephalography dereference
f franklinizations aftereffect
g gastrointestinal aggregating
h heterogeneousness chattahoochee
i incomprehensibility indivisibility
j jonathanizations abidjan
k knickerbockers kankakee
l lexicographically allegorically
m mediterraneanizations mohammedanism
n nondeterministically inconveniencing
o occidentalizations oconomowoc
p phenomenologically apprenticeship
q quantifications albuquerque
r representationally extracurricular
s straightforwardness possessiveness
t telecommunications attentionality
u uncontrollability tumultuous
v vaticanizations aviv
w wholeheartedly awkward
x xeroxing exxon
y yoknapatawpha synonymously
z zoologically bedazzle
```

Extension part 1

At the moment, the SimpleWordAnalyzer will re-process the input text file each time it is looking for a word. You should write a subclass of SimpleWordAnalyser, named CachingWordAnalyzer, that *remembers* the longestWord and the wordWithMostOccurrences for each letter, after the appropriate method has been called once – so it doesn't need to re-process the file to recalculate the same answer again. (This optimization is known as *memoization*). You can implement this optimization with a `String` array for each function, indexed by the single character input to that function.

Extension part 2 (No Extra Credit)

The methods `longestWordStartingWith` and `wordWithMostOccurrencesOf` have very similar structures. Effectively, the only difference between them is the expression that calculates whether a word is appropriate (i.e. the longest word seen so far beginning with `char` c, or the word with the most occurrences of c so far). You can express this calculation as a lambda function, that takes a `String` and a `char`, and returns an `int` value. e.g.

```
    (String s, char c) -> ((s.charAt(0) == c)?s.length():0)
```

and then express the two methods `longestWordStartingWith` and `wordWithMostOccurrencesOf` as function-parameterized versions of the same generalized method.

This extension is not worth any credit, so only complete this part for interest (or entertainment). The use of lambda functions requires Java 8 support, which you may need to enable explicitly in your Eclipse environment for this project. If you want to submit this work, please use a separate Java class called LambdaWordAnalyzer.

General Hints for the Lab Assignment

Use a `FileReader` wrapped in a `BufferedReader` to read the input text file line-by-line into memory.

Use appropriate `String` library methods to access the first character in each word, and to calculate the length of each word.

Tutors are looking for:

- `try`-with-resources statement, or `try/finally` to close input streams.
- `catch` blocks for specific `Exception` subclasses.
- `IllegalArgumentException` thrown when non-lower-case letter character parameters are supplied to the two word-finding methods
- sensible use of Java standard library methods.
- elegant code with appropriate control-flow constructs.

Submission

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 8 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code and drag *only* the Java source files SimpleWordAnalyzer.java and CachingWordAnalyzer (if you implemented this) and LambdaWordAnalyzer (if you implemented this) into the drag-n-drop area on the moodle submission page. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the files are uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your file and return feedback to you via moodle.

Outline Mark Scheme

A: Code compiles, passes all tests. Main method gives correct output. Correct implementations of both Simple and Caching versions of WordAnalyzer. Sensible design decisions in terms of control flow structures, exception handling. Good use of comments and annotations.

B: Same as A, only no implementation of CachingWordAnalyzer.

C: Code compiles, passes most tests. Main method gives sensible (if not entirely correct) output.

D: Code may not compile, may not pass tests. However a sensible attempt has been made.

E: Poor attempt, little code submitted.

F: Very poor attempt. Minimal code submitted.