CO320 Assignment 3 (Version 2019.12.06)

Introduction

This assignment is designed to allow you to practice working with further collection classes and writing tests for code you have developed. You will hand the assignment in via the CO320 Moodle page.

Date set: 2nd December 2019
Deadline: 23:55 13th January 2020
Weighting: 15% of the module's coursework mark.

Please try to read through the assignment as soon as possible so that you know what you need to understand to complete it, and can think about how best to organise your time.

Getting started

Download the starting project from:

https://www.cs.kent.ac.uk/~djb/co320/LOCAL-ONLY/assign3.zip

For the implementation, you will need to make extensive use of collection classes, such as those covered in Chapters 4 and 6 of the course text book. You will also likely need to know how to use *wrapper classes*, a description of which can be found in Chapter 6 on pages 227-229.

For the JUnit test class, you can find material in Chapter 9 of the course text book.

Plagiarism and Duplication of Material

The work you submit must be your own. We will run checks on all submitted work in an effort to identify possible plagiarism, and take disciplinary action against anyone found to have committed plagiarism.

Some guidelines on avoiding plagiarism:

- One of the most common reasons for programming plagiarism is leaving work until the last minute. Avoid this by making sure that you know what you have to do (that is not necessarily the same as how to do it) as soon as an assessment is set. Then decide what you will need to do in order to complete the assignment. This will typically involve doing some background reading and programming practice. If in doubt about what is required, ask a member of the course team.

- Another common reason is working too closely with one or more other students on the course. *Do not* program together with someone else, by which I mean do not work together at a single PC, or side by side, typing in more or less the same code. By all means *discuss* parts of an assignment, but do not thereby end up submitting the same code.

- It is not acceptable to submit code that differs only in the comments and variable names, for instance. It is very easy for us to detect when this has been done and we will check for it.
- **Never** let someone else have a copy of your code, no matter how desperate they are. Always advise someone in this position to seek help from their class supervisor or lecturer. Otherwise they will never properly learn for themselves.
- It is not acceptable to post assignments on sites such as RentACoder and we treat such actions as evidence of attempted plagiarism, regardless of whether or not work is payed for.

Further advice on plagiarism and collaboration is available from https://www.cs.kent.ac.uk/teaching/student/assessment/plagiarism.local

You are reminded of the rules about plagiarism that can be found in the Stage I Handbook. These rules apply to programming assignments. We reserve the right to apply checks to programs submitted for assignment in order to guard against plagiarism and to use programs submitted to test and refine our plagiarism detection methods both during the course and in the future.

The Task

The task is to complete the method bodies of the WordAnalyser class, which has been provided in the starting project at the URL above. Use that project as a starting point and don't re-key in the given code.

Once completed, the WordAnalyser class will be used as part of an application to gather information about word use in documents. The purpose of an instance of the class is to keep a count of how many times each word in a document is used. It must also keep track of which words follow each other. Some examples are given below, after the descriptions of the methods.

In addition to completing the implementation of WordAnalyser, you must develop a JUnit test class that thoroughly tests WordAnalyser in an attempt to demonstrate that the requirements have been met. Marks will be awarded both for the implementation of WordAnalyser and the thoroughness of the test class. You must submit both WordAnalyser and the test class in a single project.

Your implementation of WordAnalyser will also be tested for correctness by a separate test class developed by the marker.

The assessment overall is about code quality. While that is partly evaluated by correctness demonstrated through testing, the implementation should also be of good quality. So, you will be assessed on the appropriateness of your solution as well as its correctness. Nevertheless, correctness is, in general, to be valued over efficiency. So don't be satisfied with a highly efficient but slightly incorrect solution.

The Word Analyser class

The class provided in the starting project consists of a constructor and four methods. There is no useful implementation in any of those elements and you must add further code to complete them. You must not modify any of the names, parameters or return types of those elements, although you may add further non-public methods if you wish and you have complete freedom over the fields you define, although all instance fields should be kept private.

The class must be implemented to provide the following functionality:

- addWord: The parameter represents a case-sensitive word that has been found in a document. The count of occurrences for that word must be increased by 1.

  You may assume that all strings passed to this method will consist only of one or more alphabetic characters and there is no need to check that this is the case. In other words, all parameters will be valid words.

- getCount: The parameter represents a case-sensitive word that has been found in a document. The record of the number of times that word occurs in the document must be returned.

  In other words, the method must return the number of preceding times that the addWord method has been called with the exact same parameter.

- followedBy: The two parameters represent two case-sensitive words that might have been found in a document. The method must return true if the first parameter has occurred at least once in the document and was immediately followed at least once by the second word. Otherwise it must return false.

  In other words, the method must return true if at some point, a call was made to addWord with the first parameter, and the next call to addWord was made with the second parameter. Otherwise it must return false. See below for an example.

- getCaseInsensitiveCount: The parameter represents a case-sensitive word that has been found in a document. This method must return the total number of times that the word occurs in the document, *irrespective of the case of those occurrences.* It is important to appreciate the difference between the functionality of this method and the functionality of getCount, which returns a case-sensitive count. See below for an example.

Example data

Consider the following text found in a document that is being analysed using an instance of WordAnalyser:

> The first day of the third month was always difficult for Joy because there was little to look forward to in the month of March.

If a call to addWord has been made for each word, in order, then the following would be the results of a sample of method calls:

- A call to getCount("The") would return 1.
- A call to getCount("the") would return 2.
- A call to getCaseInsensitiveCount("The") would return 3.
- A call to follows("The", "month") would return false.
- A call to follows("the", "month") would return true.

The test class

You need to develop a JUnit test class for WordAnalyser. Each method of WordAnalyser should be tested with multiple items of data designed to identify the most likely errors. Aim to test just one potential error per test method because if a test method identifies a failure, it will be easier to work out what has failed, and thereby track down the problem.

David Barnes, 2019.12.01
Updated 2019.12.06 to make it clear that appropriateness of solution will be assessed as well as its correctness.