CS5001 Object-Oriented Modelling, Design and Programming
Practical 1 – Word Counter

School of Computer Science
University of St Andrews

Due Friday week 4, weighting 15%
MMS is the definitive source for deadlines and weightings.

Now that you are familiar with your programming environment from having completed the introductory exercise, you are going to write a small program in your first practical that reads in a simple text file and searches for some words. It should count the number of times this word appears in the file, and report the counts to the terminal.

For this practical, you may develop your code in any IDE or editor of your choice. However, you must ensure that your source code is in a folder named `CS5001-p1-wordcounter/src` and your main method is in a file called `WordCounter.java` (see Practical 0 for instructions).

Requirements

Your program should have the following features.

- Accept two arguments from the command line.
  - The first argument is the name (and path) of the file containing the text
  - The second argument is the word that is being searched for.
  - If either argument is missing, you should output the following message:

```
Usage: java WordCounter <filename> <searchTerm>
```

  - If the specified file does not exist in the given location, you should output:

```
File not found:
```

  followed by a space and the name of the file.

- Search through the file for occurrences of the specified word.
  - The word should only be counted when it appears as a whole word. For example, when searching for `hope`, the word `orthopedic` should not count as a match.
  - To keep things simple, a word is defined as a continuous sequence of characters from the ranges A–Z, a–z, 0–9, and the underscore (`_`) character. All other characters should be ignored. Hence, `they're` counts as two separate words `they` and `re`.

- Print the results to the terminal.
    - The message printed should have the form "The word '<word>' appears <number> time(s)."
    - For example:

```
The word 'animal' appears 17 times.
```

```
The word 'lugubrious' appears 1 time.
```

- Support searching for multiple words at once.
    - All command-line arguments from the second argument onwards are treated as words to search for.
    - If multiple words are specified, instead of printing the message specified above, the results are shown as a table in the following form:

```
|---------|-------|
| WORD    | COUNT |
|---------|-------|
| round   |    17 |
| ability |     0 |
| enemy   |     1 |
|---------|-------|
| TOTAL   |    18 |
|---------|-------|
```

    - The columns in this view should adapt to the lengths of the words and numbers shown: they should be wide enough to display all words and numbers with at least one space on either side, as shown.
    - Words and column-headers should be left-aligned, while numbers should be right-aligned.

Hints

So far, we have only briefly talked about reading text in from a file, so you might have to do a little research to find out how to do this reading. You might find the `Scanner` object very useful when reading through the text. You can create a `Scanner` that reads through a text file using the following code:

```
File textFile = new File(filename);
Scanner textScanner = new Scanner(textFile);
```

Then you can use the `Scanner` object's methods to get the strings you need from the file. You can find information on this class at its page in the Java API, and you might pay particular attention to the `hasNext`, `next`, and `useDelimiter` methods.

You may find it useful to look up other features in the Java API, including Strings, different subclasses of Exception, and regular expressions.

We have provided you with two pieces of example text which you can use to test your program. The files are available at:

https://studres.cs.st-andrews.ac.uk/CS5001/Coursework/p1-wordcounter/Resources/

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 CS5001-p1-wordcounter/src but your text files are in CS5001-p1-wordcounter, you could use the following command, specifying the relative path to `pride_and_prejudice.txt`, to search through the file for occurrences of the word "Elizabeth":

```
java WordCounter "../pride_and_prejudice.txt" Elizabeth
```

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

The expected output of a few examples is shown in Appendix A – Sample Runs. You can find further examples of output by examining the tests provided (see Automated Checking below).

For this practical you may find the documentation of arrays and strings particularly useful in showing you how to find out the length of an array and of a string, how to extract substrings from a string, how to find the occurrence of certain characters in a string, split a string, remove leading and trailing white space, etc.

In future we'll be thinking a lot about object-oriented style, but for this practical it's okay to do everything in a single class with static methods, if you wish. If you want to break it down into multiple classes, think hard about what natural object types are applicable to this problem.

Automated Checking

You are provided with some basic unit tests to check your code provides the required functionality. The tests can be found at `/cs/studres/CS5001/Coursework/p1-wordcounter/Tests`.

In order to run the automated checker on your program before saving it to an archive, log into one of the School computers running Linux, then **change directory** to your `CS5001-p1-wordcounter` directory and execute the following command:

```
stacscheck /cs/studres/CS5001/Coursework/p1-wordcounter/Tests
```

Similarly to the instructions in Practical 0, 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 even create new tests in a local sub-directory in your assignment directory and run stacscheck with your own directory as the argument, for example:

```
stacscheck /cs/home/<yourusername>/CS5001-p1-wordcounter/MyTests
```

When you are ready to submit, make sure that you also run the checker on the archive you are preparing to submit, by calling, for example:

```
stacscheck --archive CS5001-p1.zip /cs/studres/CS5001/Coursework/p1-wordcounter/Tests
```

You can see the documentation for the automated checker at:

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

Deliverables – Software (and README)

If you have attempted any enhancements beyond those mentioned in the handout, you should include a short `README` file in your assignment folder (`CS5001-p1-wordcounter`). You should describe what you have done, including any instructions for compiling, running and using your program. Hand in a `.zip` archive of your entire assignment folder, which includes your src directory, the `README` file (if applicable), and any local test sub-directories, via MMS.

Marking

Grades will be awarded according to the General Mark Descriptors in the Feedback section of the School Student Handbook. The following table shows some example descriptions for projects that would fall into each band.

Grade	Examples
1–3	Little or no working code, and little or no understanding shown.
4–6	Acceptable code for part of the problem, but with serious issues such as not compiling or running. Some understanding of the issues shown.
7	A running program that reads text from a file and searches for a single word, but with serious problems such as incorrect text processing, incorrectly formatted output, and crashes. Poor code quality.
8–10	Reads from a file and searches for a single word, achieving the correct answer most of the time. Possibly includes bugs or misformatted output. Code quality is legible but with major room for improvement, perhaps all being done in a single method.
11–13	Single-word search complete, and some progress on multiple-word search, but with problems such as poor error-handling or invalid output formatting. Code quality is reasonable, with correct indentation and some method decomposition.
14–16	Almost all required functionality implemented correctly, with only minor bugs or missing features. Code is clear and well-structured, with good method decomposition, Java code conventions followed, and some use of Javadoc.
17–18	All required functionality implemented correctly, with appropriate error-handling and robustness. High-quality code that is easy to understand, decomposed into short methods with very little code duplication, with high-quality comments and Javadoc.
19–20	Outstanding implementations of all features, and possibly some extra features, with exceptional clarity of design, no code duplication, concise but full documentation, and an appropriate object-oriented structure.

Submissions will be marked according to the School's General Mark Descriptors, which can be found in the student handbook at

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/

We will run automated plagiarism checks on all submissions, so please don't be tempted to share your code with another student or use code you find online.

Appendix A – Sample Runs

Two sample files are provided on StudRes, containing the full text of two famous works of literature: *Pride and Prejudice* by Jane Austen, and *A Tale of Two Cities* by Charles Dickens. You can use these files to test your work. Some sample desired outputs are shown below, and you can see more by looking at the test files.

The examples below assume that you are working in the `src` directory, and that the text files are located in the parent directory. The `$` symbol represents the prompt in the terminal, but your terminal prompt might have some more information in it.

```
$ java WordCounter ../pride-and-prejudice.txt Elizabeth
The word 'Elizabeth' appears 634 times.
$ java WordCounter ../pride-and-prejudice.txt the
The word 'the' appears 4060 times.
$ java WordCounter ../pride-and-prejudice.txt Jane Elizabeth Mary Kitty Lydia
|-----------|-------|
| WORD      | COUNT |
|-----------|-------|
| Jane      |   292 |
| Elizabeth |   634 |
| Mary      |    39 |
| Kitty     |    71 |
| Lydia     |   170 |
|-----------|-------|
| TOTAL     |  1206 |
|-----------|-------|
$ java WordCounter ../a-tale-of-two-cities.txt times
```

```
The word 'times' appears 51 times.
$ java WordCounter ../a-tale-of-two-cities.txt London Paris
|--------|-------|
| WORD   | COUNT |
|--------|-------|
| London |    28 |
| Paris  |    63 |
|--------|-------|
| TOTAL  |    91 |
|--------|-------|
$ java WordCounter ../a-tale-of-two-cities.txt pizza
The word 'pizza' appears 0 times.
$ java WordCounter ../a-tale-of-two-cities.txt
Usage: java WordCounter <filename> <searchTerms>
$ java WordCounter ../the-house-at-pooh-corner.txt Piglet
File not found: ../the-house-at-pooh-corner.txt
```