Platform Computing – Project 1

Introduction

This project is meant to give you experience with the Java Collections Framework's HashMap and to use it how it is very commonly used – to count occurrences of an event.

Part 1 – Programming (80 points)

Textual Analysis of books

You are given a large file of text (e.g., books from Project Gutenberg). You need to read in the file and perform the following text analyses for the book:

1. **Letter Frequency:** Consider all letters 'a' – 'z'. For the given book, count the number of occurrences of each letter. Print out the top-10 most frequent letters along with the frequency in the book. (Random Aside: Frequency analysis is commonly used in breaking classical cipher encryption schemes. https://en.wikipedia.org/wiki/Frequency_analysis))
2. **Word Frequency:** Consider all words. For the given book, count the number of occurrences of each word. Print out the top-10 most frequent words along with the frequency in the book.
3. **Word Frequency with Stop List:** A "stop list" is provided (stop-list.txt). This contains the 573 most common words in the English language. This includes words like "the" and "and", which usually don't add any semantic meaning to the text. Consider all words in the book *that are not included in the stop list*. For the given book, count the number of occurrences of each word. Print out the top-10 most frequent words along with the frequency in the book *excluding words in the stop list*. (Random Aside: Word frequency with a stop list can give you a surprisingly good summary of the book.)
4. **Wild Card:** Come up with an interesting question. List the question and find the answer to it.

9 books from Project Gutenberg are provided (alice-in-wonderland.txt, christmas-carol.txt, huck-finn.txt, les-mis.txt, metamorphosis.txt, my-man-jeeves.txt, pride-prejudice.txt, tale-of-two-cities.txt, tom-sawyer.txt). Run your text analysis on these books. Feel free to download other books if you like. **Note: students have reported problems reading tom-sawyer.txt. If you have that problem, move on. We will not test your code on tom-sawyer.txt.**

Reading in the text file and separating the words requires the use of Regular Expressions, which can be tricky. If you are not familiar with regular expressions, you can use the code I provide you below to get the words for the word frequency part. The code assumes that you've read in each line of text from the file and stored each line as a String in an ArrayList called fileLines. If you do that, you can use the code below to get each word from the String:

```
for(String line: fileLines) {
    scan = new Scanner(line);
    while(scan.hasNext()) {

    //this will read the line and separate out each
word

    scan.useDelimiter("[^a-zA-Z']");
    String word = scan.next();
    word=word.toLowerCase();
    //replace all leading apostrophes
    word = word.replaceAll("^'+", "");
    //replace all trailing apostrophes
    word = word.replaceAll("'+$", "");
    /* now you have a word to put in your map*/
    //Note: Make sure to check for empty String
    //don't put an empty string in the map
}
```

Your solution should use a HashMap to count the occurrences of letters and words. Other than that, you are free to design this program however you'd like. However, the program must start with a class named Project1.java. Project1.java will be the only class with a main method and will be the starting point of your project. You can use as many support classes as you'd like to design your solution.

We will test your program with alice-in-wonderland.txt first and then we may test your solution against other files. Your submission should read in alice-in-wonderland.txt and run using that file. That means you'll have to hard code "alice-in-wonderland.txt" into your code - most likely in the main method of Project1.java.

This is a good opportunity to practice designing a modular piece of software. Your solution should include at least 1 class other than Project1.java.

Make sure to use Javadoc style comments for each method in your code.

Testing Your Code

Word counts will be posted on Piazza for you to check your results. Depending on your implementation, your results may be slightly off. The results don't have to be exact, but they need to be in the ball park.

Part 1 – Reflection (20 points)

Describe how you designed your program. What classes did you use? Did each class have a separate job or purpose? If you could refactor your code now that you've completed the project, what improvements would you make?

Part 2 – Grading Criteria

5% for compilation – If your code compiles, you get full credit. If not, you get a 0.
65% for functionality – Does the code work as required? Does it crash while running? Are there bugs? …
10% for design – Is your code well designed? Do you have at least 2 classes – Project1.java and one other? Does it handle errors well? Are exceptions handled well? …For example, if the file is not found is the exception handled and a nice message printed to the user displayed or is the stack trace shown.

Submission Instructions

Submit Project1.java and any other java files you used in your solution. Do not zip your files. **The file, alice-in-wonderland.txt, should be hard coded into your program as the file that it reads.** We will test your program with alice-in-wonderland.txt. Do not ask the user for a file name to read, your program should read the file alice-in-wonderland.txt. Do not put your code in a package. If your project requires something other than putting all files in a java project in Eclipse to run, then please submit a readme file as well, describing how to run your program. Submit your reflection as a pdf file.

Late Policy

Lab assignments and projects are assessed a 3 point per-day late penalty. You do not need to request an extension, just take it. However, because the late penalty is so lenient it will be enforced.