Assignment 5
CSc 210 Fall 2017
Exercises Due October 2nd, 8:00 pm MST
Programs Due October 5th, 8:00 pm MST

Introduction

In class we have discussed the HashMap class and how this concept is similar to that of a dictionary from python. On this assignment, we will be using the hashmap class to solve some simple problems

NOTE: THIS ASSIGNMENT HAS TWO DUE DATES.
There are two parts (and two due dates!) to this assignment, exercises done via CodeStepByStep and programs to be turned in via github classroom. The exercises will be due Monday night while the actual programs will be due Thursday night.

You will turn in your code via github classroom. Here is the link to receive your repository: https://classroom.github.com/a/rjRMulFd

For each Java program assigned, you will turn in a .java file for that program. We should be able to run "javac *file*.java" and then "java *file*" on lectura and your program should compile and run successfully.

Your Java programs should adhere to our style guidelines we will give to you. Here is the link: https://www2.cs.arizona.edu/classes/cs210/fall17/StyleGuidelines.pdf

Specification Part I: CodeStepByStep

Complete the following exercises on CodeStepByStep by Monday, October 2nd, at 8:00 pm.

- Java->collections->maps->collectionMystery1
  - Note: This uses an "enhanced for loop"
  - This assumes the order of items entered is the order of the table in the hashmap
- Java->collections->maps->collectionMystery4
  - This uses an ArrayList, you may want to look into this to understand what it is doing.
- Java->collections->maps->hasDuplicateValue
- Java->collections->maps->leastCommon
- Java->collections->maps->maxOccurences

Specification Part II: HashMaps and Word Counts

What are the words in a document? How frequently are specific words used? What can we infer from words about the well being or opinions of the author?

Before computers, such questions required labor-intensive analysis of texts, so word analysis was done only for major religious and literary works. In the 13th century, Hugh of St. Cher employed 500 monks to assist him in analyzing the Vulgate Bible to create a "concordance," an alphabetical list of words, showing the context for each occurrence of a word. Computer assisted word analysis began shortly after World War II, but the efforts in the 1950s and 1960s were hampered by inadequate programming language support and computing facilities.

Much has changed. A relatively recent application is sentiment analysis to mine text and voice data to quantify attitudes, emotions, and well being. A 1999 study asked participants to rate their reactions to 1,030 words on a list called the ANEW list (from Affective Norms for English Words). Each word was rated on a scale of 1 to 9, where higher scores reflect happier reactions. For example, "grin" got a score of 7.40 and "gripe" got a score of 3.14.

Let the happiness index for a text be the average score for the occurrences in the text of words on the ANEW list. For example, consider the following line from a Taylor Swift song:

Honey, I rose up from the dead, I do it all the time

This line has three ANEW words: honey, score 6.73; dead, score 1.94; and time, score 5.31. The average of the scores for the occurrences of the ANEW words is :

(6.73 + 1.94 + 5.31)/3 = 4.66

Program 1: HappyScore.java

This program will calculate the happiness index for a given file and its set of words. Your program will receive a file name as a command line argument and should analyze the given file and report its happiness score to stdout. You will also be given a file in your repos called ANEW.txt which contains happiness index score to word associations.

You should read in the ANEW.txt file and somehow store information about each words happiness score. Then, read in each word from the given file(s) and determine if it has a happiness score associated with it. You should calculate the average happiness score

of every word that appears and print it to stdout. Note, you may be given multiple file names as arguments.

Example:

```
% javac HappyScore.java
% cat helloWorld
hello world
% cat test
happy sad
very mad glad rain day cake
food gloom rad sad bad man talks
% java HappyScore test helloWorld error test
Happiness Index for 'test': 4.718889
Happiness Index for 'helloWorld': 6.500000
Error: Encountered a problem when opening file 'error'
```

Important Points:

- Note the message used to report the score calculated. Your program should follow that same format.
- When your program detects errors, you should print a message to stderr and exit with a status of 1. One error you may want to catch is non-existent files. You can do this via a 'try-catch' block.
- When considering a word, its case should not matter.

You shall turn in a file called HappyScore.java for this problem.

Program 2: MorseCode.java

For this program you will be writing a morse code converter. You will be given a file that maps basic characters to their morse code encoding. Your program should store this information somehow. Then, you will read input from standard input, translating morse code into full english words, and keep a count of how many times each word has been seen.

You will have a file (found in your repo) called morse.txt. This has the character to morse code mapping in it. The input to operate on will be given to your program via stdin. It will be a sequence of .'s and -'s representing morse code. Each morse character in this input will be separated by a single space and each full word will be

separated by two spaces. Using the string .split() method may be helpful in parsing this. You will keep track of how many times each english word has been seen and print out this out when your program has finished executing.

Example:

```
% cat test
--- ... ---  --- ... ---  .--. .--.  ... ... --- ... --.
% java MorseCode < test
{PP=1, OSO=2, SSOSG=1}
```

Note the format of how the count was printed. This same output can easily be replicated by passing a HashMap to a print method.

When your program detects errors, you should print a message to stderr and exit with a status of 1.

For this program you will turn in a file called MorseCode.java.

Miscellaneous

In order to receive a grade, your repo must only contain your source java files in the root directory of your repo.

This assignment will be submitted through github classroom. Make sure all of your code you would like to submit is in your repository when the due date arrives.

Note: Your output must match what is defined here in the spec. We will give you a small selection of test cases so you can make sure you have the right format. Do not print extraneous output, you may lose a lot of points (This includes prompts when reading in input!). **These test cases can be found in your repo.**

Remember, do not cheat! Refer to the syllabus and first lecture for more information.