PA2 Job Skills

Due: Wednesday 2/3 by 11:30PM
Submission: Submit only PA2Main.java to Gradescope. Do not submit the entire project.

Overview

The goal of this assignment is to learn how to use the dictionary abstract data type in Java, sorting on lists, and to continue practice reading parameters in from the command line (Eclipse has an interface for providing command-line options to a Java program, see Run As $->$ Run Configurations $->$ Arguments).

In Java, a HashMap implements a dictionary. By the end of the assignment you will have practiced using the following operations on a HashMap:

- insertion
- modifying the value for a particular key
- iterating over all of the key, value pairs
- the fact that HashMaps do NOT store their key, value pairs in order and how to handle it

Assignment

For this project we will be using job skills data that you could download from the following website: https://www.kaggle.com/niyamatalmass/google-job-skills. **However you do not have to download the file**, because we provided it and other variants of it with the same format in the PublicTestCases/subdirectory. In fact we had to do a lot of cleaning of the files, so see the "clean" versions and note that the test cases only involve those clean versions.

See ScrubCSVFile.java if you are interested. **There is no need to change or even look at this file, but we left it there in case you wanted to see.**

Like in PA1, for PA2 you will be taking an input file name from the command line.

In addition to the input file, the second command-line argument will be a command to run on the data from your input file. The commands consist of CATCOUNT and LOCATIONS. If an unknown command is provided, the program should print the message:

```
Invalid Command
```

Each command will be most easily implemented with a HashMap. Therefore, your implementation will read in the csv file and be using one or more HashMaps.

CATCOUNT - This command reads an input file and for each job "Category" prints out the category and the number of jobs in that category.

LOCATIONS - For a given category, list the locations that have a position in that category open. Each location will be followed by a number indicating how many positions in that category are open in a particular location.

Input and Output

Example input (see Kaggle for more in-depth description). The cutoff word below is "Qualifications".

```
Company,Title,Category,Location,Responsibilities,Minimum Qualifications,Preferred Qualific
Google,TitleA,CategoryX,Tel Aviv,Everything and the rest, BS, MS
Google,TitleB,CategoryX,Tel Aviv,Everything and the rest, BS, MS
Google,TitleB,CategoryY,Houston,Everything and the rest, BS, MS
Google,TitleC,CategoryX,Jonesboro,Everything and the rest, BS, MS
```

Output for command line arguments 'CATCOUNT'

```
Number of positions for each category
-------------------------------------
CategoryX, 3
CategoryY, 1
```

Output for command line arguments 'LOCATIONS CategoryX'

```
LOCATIONS for category: CategoryX
-------------------------------------
Jonesboro, 1
Tel Aviv, 2
```

Public Test Cases

The .out files in the PublicTestCases/ have the command line arguments encoded in their name. For example, for the file

```
pa2-example-LOCATIONS-CategoryZ.out
```

The command line arguments should be:

```
example.csv LOCATIONS CategoryZ
```

Put the command line arguments into Eclipse by doing the following:

- right click on PA2Main.java
- select Run As $--> $ Run Configurations
- Select the arguments tab
- Put the command line arguments in there and run the program

Hints

The class notes posted on piazza have examples of reading from the command line. The input files are CSV files for this assignment, you can assume that splitting on commas is ok.

Since HashMaps are unordered, special steps must be taken for them to be outputted in an ordered format. As shown below, take the set of keys and input it into an ArrayList of type String. Then use a Collections sort to sort the keys in place.

```
List<String> sortedKeys = new ArrayList<String>(mymap.keySet());
Collections.sort(sortedKeys);
```

Grading Criteria

Just like with PA1, we provided some public testcases (more than last time), but we will be testing your code on lots of private test cases. So be sure to make your own testcases to thoroughly test your code. Remember it is okay to share these testcases. It is not okay to share code.

Grades for PA1 are due Wednesday 2/3 by 5:00PM. If you have not received your grade by then, it means your UGTA is slacking (or a myriad of other issues)! Post privately to Christian and me and we will look into it. This also means that at a minimum you will have 6.5 hours to take feedback into account, not very long. So we wanted to mention a few things we will be looking for in PA2.

- Should use only static methods.
- Each static method should be short and concise. A method should only perform one task. If you have a method that is performing multiple tasks, break it into smaller methods. We think a good rule of thumb for method length is 30 lines.
- Make things as simple as possible. This means avoiding nested loops, nested conditionals, and too many levels of user-defined methods calling other user-defined method (chaining) whenever possible.
- You should be able to read, understand, and explain your own code weeks after you wrote it.

- The file header should include instructions on how someone would use your program and what an input file would look like.
- Use meaningful variable names.
- Remember that indentation and spacing are great tools to make your code more readable. Eclipse should automatically indent the proper levels for you. As long as you stick with those defaults you should be fine.

Write your own code. We will be using a tool that finds overly similar code. Do not look at other students' code. Do not let other students look at your code or talk in detail about how to solve this programming project.

Iterative Development

In class we talked briefly about iterative development. This was the most important thing I learned from my first computer science classes. This project is a great place to start. Break this large, daunting project into as small of pieces as you can. Then work on completing **and testing** each small piece one at a time. Some benefits include:

- Easier debugging - if you encounter a bug, you know exactly where it came from! It must be that small piece that you just finished and tested.
- It changes the large, complex PA into many smaller problems (almost like a drill). It is a lot easier to code a small 20-line method than an entire PA.
- Decomposition - The small pieces of the program you created do the work of decomposing the code for you!
- You will make me happy. (This could be seen as a negative).

Shoot for a goal of being able to run your code every 10-15 minutes and have something that compiles and runs. So each small piece should take anywhere from 5-15 minutes to complete. If it takes longer, it is too big of a piece!