Baby Names

Haverford CS 106 - Introduction to Data Structures

Lab 3 (1.5 weeks)

1 Overview

In this assignment, we'll be exploring linked lists and more complex custom-designed classes. Your task will be to design a linked list that manages annual statistics about baby names in the United States, allow it to take specific command line inputs, and print out statistics based on the input.

Note: this dataset presents a very gendered way of viewing baby names. We are using this dataset not because we endorse this binary, but because it is a good dataset for understanding the material. It is also useful to see how data has historically been collected and to think about how that could change in the future.

2 The Input

2.1 Input File Format

We'll be taking input from files containing lines in the following format:

```
rank,male-name,male-number,female-name,female-number
```

where the comma-separated fields have the following meanings:

`rank`	the ranking of the names in this file
`male-name`	a male name of this rank
`male-number`	number of males with this name
`female-name`	a female name of this rank
`female-number`	number of females with this name

This is the format of database files obtained from the U.S. Social Security Administration of the top 1000 registered baby names. Each line begins with a rank,

followed by the male name at that rank, followed by the number of males with that name, etc. Here is an example showing data from the year 2002:

```
1,Jacob,30568,Emily,24463
2,Michael,28246,Madison,21773
3,Joshua,25986,Hannah,18819
4,Matthew,25151,Emma,16538
5,Ethan,22108,Alexis,15636
...
996,Ean,157,Johana,221
997,Jovanni,157,Juana,221
998,Alton,156,Juanita,221
999,Gerard,156,Katerina,221
1000,Keandre,156,Amiya,220
```

As you can see from the above, in 2002, there were 30,568 male babies named Jacob and 24,463 babies named Emily, making them the most popular names used in that year. Similarly, going down the list, we see that there were 220 newborn females named Amiya, making it the 1000th most popular female baby name.

The entire data set contains a file for each year from 1990 to 2017, named `names1990.csv`, ... , `names2017.csv` respectively.

2.2 Input Command

One of the end goals of this lab is to print out the statistics of a given (input) name for given (input) files. This will be further specified in Section 5.7 but for now just understand that the program will eventually give output for specific names and files that are given.

3 Resources

This section contains some information that will be helpful throughout. Consult the Java String documentation for more details `https://docs.oracle.com/javase/7/docs/api/java/lang/String.html`.

1. Use `.equals()` when comparing `String`s. For this lab, you do not need to overwrite the `.equals()` method for lists or nodes, you can use `String` to compare two names directly.

2. Consult the documentation to learn about the `.compareTo` method for `String`s.

3. Consult the documentation to learn about the `.subString` method for `String`s.

4. In Java, dividing two integers results in an integer (rounded down). To achieve a double or fractional result, first *cast* one of the integers to a double.

5. For all classes you create, add a a `.toString` method to help with debugging. It may also be helpful to read in only a few lines of one file to start, to make sure you're sorting the names correctly.

6. To round your results to 6 decimal places, you can use string formatting (similar to Python). For example

```
double fraction = 0.456622723;
System.out.println(String.format("%.6g%n", fraction));
```

4 End goal

4.1 Components of output

You will be building two linked lists to store the baby names found in the given files, one for the male names and one for the female names. The linked lists should be kept in alphabetically sorted order by name.

Specifically, the program needs to be able to look up a name and report the following statistics:

1. Position in linked list - an integer indicating the position of the name in your linked list so that we can verify your list is sorted lexicographically. The position of the first name in the linked list should be `0`.

2. For each year
   (a) rank - the rank of the name that year
   (b) number - the number of babies given that name that year
   (c) percentage - the percentage of babies given that name that year (for that gender)

3. Total
   (a) rank - the rank of the name among all years (for that gender)
   (b) number - the number of babies given that name among all years
   (c) percentage - the percentage of babies given that name among all years (for that gender)

4.2 Output format

Considering each of the items listed in Section 4.1, the output format should like like the following:

```
Position of Name in the Linked List:  integer
(One empty line)
Year
Name:  Rank, Number, Percent
(One empty line)
Year
Name:  Rank, Number, Percent
(One empty line)
...
(One empty line)
Total
Name:  Rank, Number, Percent
```

where each italicized item should be replaced with what they are representing. There should be exactly one space between the colon and the rank and between the comma and the other attributes of the year.

For example, if a given input wants all 28 files to be read in and wants the statistics for the name "Mary" (female), the following should be printed:

```
Position of Mary in the Linked List: 1423

1990
Mary: 35, 8666, 0.005432

1991
Mary: 38, 8760, 0.005596

...

2017
Mary: 126, 2381, 0.001877

Total
Mary: 51, 142630, 0.003630
```

4.3 VerifyFormat.java

The correctness of your program will be graded using the autograder which requires the format to be as outlined in Section 4.2. Among the starter files, there is a file called `VerifyFormat_Lab3.java` which you can run to check your output format. It checks for the format but not the correctness, and you should avoid making changes to the file to guarantee a correct checking tool.

5 Specific tasks

5.1 Foreword

The two linked lists you are building should be from scratch. You are not allowed to use Java's built-in `LinkedList`, although you may use Java's `ArrayList` for tasks besides storing the lists of names. You should implement a *doubly* linked list with *sentinels* for this lab. (You are welcome to then implement a singly linked list with the same functionality, but only if you first understand how to do everything with a doubly linked list.)

Although generic linked lists have many advantages, it is acceptable for this lab for your code to be simplified with non-generic linked lists that are locked to a `Node` class with the data type `String`, which represents one name. In other words, you should have a `Node` class, but it does not need to be generic and the type of data inside each `Node` can be of type `String`. Note: there are two other options if you feel very comfortable with nodes, classes, and generics:

- You can instead have two classes: one ~~generic~~ `Node` class and one `Name` class. Then your linked list class will be made up of `Node` objects, each with data type `Name`. This is probably the most theoretically sound way of doing the lab, but it is *not required*.
- You can instead have a single `Name` class that acts as both node and name (this is similar to the recommended way, but with the class name changed to `Name`. If this makes more sense to you, this way is fine too, but it is not recommended since it might cause confusion between `Node` objects and the `String` representing the name.

Calculating total rank is non-trivial. Think through your data structure and algorithm needs before you start. You should write the program so that it makes best use of available storage. Resist the urge to store redundant information in many different places.

The following sections are suggested steps that you could take to accomplish this lab.

5.2 Designing the Node class

Before creating a linked list, you should establish the `Node` class. Since the linked list will be composed of baby names, its nodes will have data of type `String`. You will eventually need to modify this `Node` class to store all the relevant statistics for a particular name, but for now your `Node` constructor needs to take in only a `String` for the `name` - don't worry about other statistics until Section 5.4.

5.3 Designing the linked list object

Create an appropriately-named class that represents the linked list object of `Node`s with data of type `String`. Allow the insertion of nodes into this list, keeping in mind that the linked list should be sorted lexicographically (alphabetically). Keep in mind that multiple files may contain the same name. Note that the `name` field of your `Node` class should be `final` and may not be changed. In other words, you must rearrange `Node`s, not the strings contained within the `Node`s.

After you have written your method(s) for insertion, try in your `Main` class reading the file(s) into two lists of unique names in sorted order. If you are having trouble debugging the sorted order, try creating a smaller input file (by keeping only the first 10 or 20 names) and using that instead. Recall that you can use the `opencsv` library to read in data from csv files - you may want to look into the difference between `CSVReader` and `CSVReaderHeaderAware`.

You may also want to make the method that returns the position of a name in the linked list and test it out now as well.

5.4 Storing the yearly rank and number

Since multiple different years may have the same name (each with its own statistics), taking care of storing the statistics will require an auxiliary data structure. Consider what you need and decide where and how to store the information carefully. Then, appropriately expand the `Node` class to allow the storage of yearly number and rank.

You may also need to modify the insertion method and the file-reading code to create a new `Node` object if it's not already in the list or to update it with the given yearly stats if it is.

5.5 Storing the totals and Calculating percentages

Computing the yearly percentages, as well as total number and total percentage, require additional auxiliary data structures besides the linked lists. Consider what you need and decide where and how to store the information carefully.

After implementing the reasonable data structures needed, enable computing of necessary totals. Then, enable reporting of yearly percentages, Total number, and Total percentage for a given name.

Ensure that you enable the Total rank computing for a given name as well. Think carefully about how to do this, and design and implement an algorithm. *Note that the totals at the very end of the printout are not optional, but worth a relatively low fraction of the total points.*

5.6 Reporting the baby name information

Make a method that returns a String formatted as mentioned in Section 4.2. It should take in the name of the baby as input, and report all years that were read in and the Total statistics. This method should only be called after all files are read in.

Remember that Strings can be concatenated with the + sign between two strings, and "`\n`" will allow you to add a line break.

Do not forget to print this String in your Main class.

5.7 Look-up via command-line arguments

5.7.1 Preamble

You may want to tackle this task in pieces - single name lookup on a single file, enable single name look up on multiple (or all) files, and then enable multiple name lookup on multiple files.

5.7.2 Input format revisited

As mentioned earlier in Section 2.2, your program should take command-line arguments to input zero or more file names to process. The name(s) will be preceded by the appropriate flag: `-m name` or `-f name` which indicate a male name or a female name to look up, respectively.

For example:

```
java Main -f Dianna names1990.csv names2000.csv
```

should print out the rank, number and percentages (as explained in Section 4.2) of the female name Dianna used in 1990, 2000 as well as the combined statistics of these two years. More than one name may be searched, each with the appropriate preceding `-f` or `-m`.

You many assume that the list of filenames is always last, i.e. the first non-flag argument you encounter is assumed to be the beginning of the list of file names.

5.7.3 Implementing name lookup via command line arguments

Implement checking the flags for `-m name` and `-f name` and report the statistics of the name from appropriate linked list.

Make sure you error-check your arguments thoroughly, i.e. illegal/badly-formatted options, non-existent options. Your program should behave rationally no matter how unreasonable the input or the value of flags. It is acceptable for this lab to print out a generic error message if the flags do not conform to the description here.

5.7.4 Testing out the linked list with command line arguments

Recall that if you are sending your main method arguments from within Eclipse, you should go to Run Configurations, then Arguments, and enter only the input from the command line above that comes after `java Main`. For example, you might enter just the arguments `-f Dianna names1990.csv names2000.csv`.

6 Writeup

Please include a writeup in your README that explains your class design and algorithms. In particular, address the following questions:

1. Which instance variables do you have in your `Name` (correction: `Node`) class?
2. How do you organize the storage of the yearly statistics per name versus the totals?
3. Where are the overall totals stored and where are the yearly totals stored?
4. How do you keep the linked lists in alphabetically sorted order?
5. How is total rank computed?

Finally, make sure to complete the Lab Questionnaire in the README. Please also answer the following question in this section: was the `VerifyFormat_Lab3.java` useful for you during the development of your code?
Note: you may copy/paste the following to run the test above:

```
-f Mary names1990.csv names1991.csv names1992.csv names1993.csv
names1994.csv names1995.csv names1996.csv names1997.csv names1998.csv
names1999.csv names2000.csv names2001.csv names2002.csv names2003.csv
names2004.csv names2005.csv names2006.csv names2007.csv names2008.csv
names2009.csv names2010.csv names2011.csv names2012.csv names2013.csv
names2014.csv names2015.csv names2016.csv names2017.csv
```