Baby Names

Haverford CS 106 - Introduction to Data Structures

Lab 3 (two weeks)

In this assignment, we'll be exploring linked lists and more complex custom-designed classes.

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 Secutiry 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
6,Andrew,22017,Ashley,15342
7,Joseph,21891,Abigail,15297
8,Christopher,21681,Sarah,14758
9,Nicholas,21389,Samantha,14662
```

```
10,Daniel,21315,Olivia,14630
...
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 Specific Tasks

You will be building two linked lists to store the baby names found in all 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. Linked list rank - just an integer indicating the position of the name in your linked list so that we can verify your list is sorted
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)

For example, for the name "Mary" (female), the following statistics should be printed for all 28 files:

```
1424

1990
Mary: 35, 8666, 0.005432

1991
Mary: 38, 8760, 0.005596

...

2017
Mary: 126, 2381, 0.001877

Total
Mary: 51, 142630, 0.003630
```

You should design a `Name` class that stores all the relevant stats for a particular name. The two linked lists you are building should be from scratch and you are not allowed to use Java's built-in `LinkedList`. Although generic linked lists have many advantages, your code will be simplified with non-generic linked lists that are locked to the `Name` class. This is acceptable.

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.

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.

Suggested steps:

1. Read the files into two lists of unique names in sorted order. (Your `Name` class needs to have only a `String`) If you are having trouble debugging the sorted order, I suggest 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 the data from the files.

2. Expand your `Name` class to provide storage for yearly number and rank. Modify your file-reading code to create a new `Name` object if it's not already in the list, or update it with the given yearly stats if it is.

3. Compute all the necessary totals to enable yearly percentage reporting and storing them in reasonable data structures.

4. Compute additional totals to enable total number and total percentage reporting

5. Design an algorithm to compute total rank

6. Enable single name lookup on a single file

7. Enable single name lookup on a multiple (or all) files

8. Enable multiple name lookup on multiple files

3 Look-up via Command-line Arguments

Your program should take command-line arguments to input zero or more file names to process.

Add flags `-m name` and `-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
```

will print out the rank, number and percentages (as explained in Section **??**) 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 preceeding `-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. Make sure you error-check your arguments thoroughly, i.e. illegal/badly-formated options, non-existent options. Remember the order of flags should not matter. Your program should behave rationally no matter how unreasonable the input or the value of flags. Check out Unix utility programs to see examples of flag and error handling.

Remember that the unix wildcard `*` will let you specify multiple file names that fit a certain pattern easily, for example:

```
java Main -f Dianna names200*
```

will be expanded to:

```
java Main -f Dianna names2000.csv names2001.csv names2002.csv
names2003.csv names2004.csv names2005.csv names2006.csv names2007.csv
names2008.csv names2009.csv
```

by the shell for you. Using this feature to test might reduce tedious typing.

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 names200*`.

4 Write-up

Please include a write-up 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` 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?