Programming Assignment: Presidents

Back

In this assignment, you will gain some additional experience working with text files, both reading and writing them.

Educational Objectives

- Demonstrate proficiency at reading and writing textual files.
- Demonstrate proficiency at handling record information in tab-delimited files.

Instructions

For this homework assignment, create a Java application named: **cpts132.files.PresidentList**. This is the application class. Any helper classes you create should be package private classes. You will submit the application in a runnable jar file.

Data File

Download the data file for this assignment, presidents.txt.
(Please note: This file uses the LF (line feed) as the line terminator. Classic MS-DOS and Windows used CR-LF (carriage return - line feed) as the line terminator. Modern Windows tools will display this file correctly. Older Windows tools, like Notepad, will not. The file you download is correct. To verify this, open the file using a newer editor, like Notepad++.)

Each line in this file has four pieces of information, separated by tabs.

> Last name \t First name \t Date took office \t Date left office \n

The basic task for this lab is to read through the data file, and create an output file which will contain sentences with the information read from the data file. For example, the first line of the file is:

```
Lincoln\tAbraham\t3/4/1861\t4/15/1865\n
```

The four pieces of information are:

- Lincoln
- Abraham
- 3/4/1861
- 4/15/1865

The output sentence for this data is:

```
Abraham Lincoln was president from 3/4/1861 to 4/15/1865.
```

It should be pretty obvious how the four pieces of data are put together into the output sentence. If you have questions, ask in the forum.

Please note:

- You will probably find it easier if you read the data from the file line by line. Use java.io.BufferedReader to read in a line at a time.
- The String.split method can prove very helpful here.

Output File(s)

To get practice writing files, the output from this assignment will be written to files. Each level of the assignment (minimal, standard, and challenge) has its own output order. So, each different order will be written to a separate file.

The different output files vary only in the order in which the output sentences are generated. Take advantage of this fact by creating a method that will take the input text (line from the data file) and generate the output sentence. There are two options that come to mind:

- A function that takes a single string parameter, that is, the input data, and returns the sentence generated using that data. The caller will have to take that return value and write it to the output file.
- A function that takes two parameters: the input data, and the stream object for output. The function can generate the sentence and immediately write it to the file object. Notice, this suggests that the second parameter is the stream object, rather than the filename. So, after you open the output file for writing, you get the stream object, probably wrapped in a filter object to make it easier to use. Pass this stream object to the function. That way the output file only needs to be opened once.

[hidden hint]: Notice that these suggestions both treat the data for a president as a single object, the line from the input file, rather than as four separate pieces of data. The method will need to handle the separation of the input line into the four pieces of data.

Minimal Level

For the Minimal Level, you only need to create a single file. The output file is **same_order.txt**. The sentences in the output file shall be in the order that the data appears in the data file, presidents.txt. [fodder: Can you figure out what that order is?]

Here is some pseudo-code for creating same_order.txt:

```
open the input file
open the output file
for each line in the input file
  call your function to write out the sentence
close the input file
close the output file
```

This pseudo-code will work for either version of the helper function. You just have to keep in mind how much work is going on in your helper function. (Actually, if you're only doing the minimal version, there is no refactoring advantage to creating the helper function. But, the helper method is part of the specification, so it should be included.)

In naming the files for input and output, do not specify a folder. The files will then be created in the same folder as the application is run. You will incur a penalty if I have edit your files while grading.

Sample output

Here is what the beginning of the output file **same_order.txt** should look like:

```
Abraham Lincoln was president from 3/4/1861 to 4/15/1865.
Andrew Jackson was president from 3/4/1829 to 3/4/1837.
Andrew Johnson was president from 4/15/1865 to 3/4/1869.
```

Please note: The name of eighth POTUS is Martin Van Buren.
[questions to ponder: POTUS? Why should is this fact of importance to the assignment?]

Standard Level

For the Standard Level, you need to create two files. The two output files are: **same_order.txt** and **last_name.txt**. The first, same_order.txt, is the same as the one created for the minimal version, with the sentences in the same order as the data in the presidents.txt file. In the second file, **last_name.txt**, the sentences shall be ordered by name, Last first. In those cases where the last names are the same, the Johnsons, the Adamses, and the Bushes, sort by first name. If you're really concerned, the two discontiguous terms by Grover Cleveland can be ordered by date.

To accomplish this, read the whole file into the program once, generate same_order.txt, rearrange the data within your program, and generate last_name.txt.
(*Hint*: java.util.Arrays.sort)

Sample output

Here is what the beginning of the output file **last_name.txt** should look like:

```
John Adams was president from 3/4/1797 to 3/4/1801.
John Quincy Adams was president from 3/4/1825 to 3/4/1829.
Chester A Arthur was president from 9/19/1881 to 3/4/1885.
```

Challenge Level

For the Challenge Level, you need to create three files. The first two are those described for the Standard Level. The additional file is **inauguration.txt**. The sentence shall be in the order set by date that the president took office. This will give the "familiar" ordering: Washington, Adams, Jefferson, Madison, Monroe, etc.

Please note, both William Henry Harrison and James A Garfield served less than one year as president. So, you should include the month that the president took office as you order the data, not just the year. As with the standard version, the data shall be ordered within your code, rather than trying to rearrange the input data file.

Sample output

Here is what the beginning of the output file **inauguration.txt** should look like:

```
George Washington was president from 4/30/1789 to 3/4/1797.
John Adams was president from 3/4/1797 to 3/4/1801.
Thomas Jefferson was president from 3/4/1801 to 3/4/1809.
```

Written Report

In addition to the Java application, the assignment includes a brief written report that answers the following questions. Notice that the questions are slightly different because there is no user input for this program.

- How did you approach (get started with) this assignment?

- What works and what doesn't work?
- What did you learn from this assignment?
- What would you do differently next time?

Submit the written report in the same submission as the source codc.

Submission

For this assignment, submit:

- One (1) runnable Java archive file (.jar)
  Include both source code and bytecode files in appropriate folder structure
- One (1) ASCII, plain text report file (.txt)

There is no need to upload the output files for this lab, since your application will generate them.

Of course, include the standard comments in the source code file.

Grading rubric

Functionality: 8 points
5/ Minimal Level, (compiles, runs, generates same_order.txt)
8/ Standard Level, (Minimal Level, plus: generates last_name.txt)
10/ Challenge Level, (Standard Level, plus: generates inauguration.txt)

Style: 3 points
- conforms to homework guidelines

Documentation: 4 points
- JavaDoc comments
- Other comments
- Report

Back