Assignment 5

Due date

- May 2nd 2020, 11:59PM EST.

Github link

The git links are the following:

- https://classroom.github.com/a/KRlYlQa7

Submit your code as per the provided instructions.

Assignment Goal

- Implement the visitor pattern to run different analytics on the given input. This is an individual assignment.

Programming Language

You are required to program using Java.

Compilation Method

- You are required to use ANT for compilation of code written in Java.
- Your code should compile and run on *remote machines* or the *debian-pods* in the Computer Science lab in the Engineering Building.

Policy on sharing of code

- EVERY line of code that you submit in this assignment should be written by you or be part of the code template provided for this assignment. Do NOT show your code to any other student. Our code-comparison software can very easily detect similarities.

Project Description

- The input file contains sentences with characters in the set **[a-zA-Z\.]**.
- A wrapper class ***MyArrayList*** stores the sentences read in from the input file, encapsulated in ***MyElement***, in an internal ArrayList.
- MyElement represents an *Element* (from the Visitor pattern) to be visited.

- MyArrayList is just a wrapper class that stores an internal ArrayList of elements. The following methods are exposed by MyArrayList.
    - *getIterator()* - Returns an iterator to the internal ArrayList.
    - *accept(...)* - Accepts a visitor and delegates the *accept(...)* to each of the elements.
- From above, it should be clear that MyArrayList is also an element.
- Once the sentences are all stored, the following analytics need to be run. *Note that for analytics, the period character '.' and space character are not to be considered as part of a word.*.
    - **KMostFrequentWords** - For each sentence store the top K most frequent words, sorted in non-increasing order of frequency to Results. The value of K will be provided via commandline. Look into PriorityQueues and Comparators. The format to store the result is **[most freq word, next most freq word, ...]**. If two words have the same frequency, then either word can be chosen. Word comparison should be case insensitive.
    - **SpellChecker** - In a sentence, check whether there is a spelling mistake in any of the constituent words. Note that for the purpose of this assignment, spell checking is simplified and performed on words that have > 2 characters.
        - A spelling mistake is said to occur if a single character of a word can be replaced to get an "acceptable word". For example, if 'cheek', 'although' and 'cheer' are acceptable words, then the word 'cheep' can be changed to 'cheek' or 'cheer' by replacing a single character.
        - The acceptable words are provided in a file, the name of which is provided via commandline.
        - For words that can be changed to acceptable words, store the word and the possible acceptable words in Results.
        - The format to store the result is **word::[acceptable word 1, acceptable word 2, ...]**. So, for the above example, the output would be cheep::[cheek, cheer].
- After running analytics, persist the results to the corresponding output files, the names of which will be provided via commandline.

Input

The following command-line arguments are to be accepted.

- *-Dinput* - Name of the input file containing sentences.
- *-DacceptableWordsFile* - Name of the file containing acceptable words.
- *-Dk* - Max size of the list containing the most frequent words. Must be > 0.
- *-DtopKOutputFile* - Name of the output file to which the top K words are written for each sentence analyzed. So, if there are 5 sentences in the input file, there should be 5 lines in this file.
- *-DspellCheckOutputFile* - Name of the output file to which the possible spelling fixes are written in the previously mentioned format for each word analyzed.

Sample run command:

```
ant -buildfile src/build.xml run \
-Dinput="input.txt" \
-DacceptableWordsFile="acceptable_words.txt" \
-Dk=2 \
-DtopKOutputFile="topk_output.txt" \
-DspellCheckOutputFile="spellcheck_output.txt"
```

Sample input and acceptable words:

input.txt	acceptable_words.txt
Mary was walkink down the aisly. She found the blue berries that she was searching for on the way.	walking aisle was

Sample output:
Assuming K = 2, below is the sample output for the sample input given above.

topk_output.txt	spellcheck_output.txt
[Mary, was] [She, the]	walkink::[walking] aisly::[aisle] way::[was]

Code Organization

- Follow a similar directory structure as that of the previous assignments for the driver code and util package. However, for the other classes and/or interfaces, make sure that they are in the appropriate package.
- Use the below driver code. Replace the TODOs. DO NOT ALTER ANYTHING ELSE IN THE DRIVER

```
// TODO add import statements.

public class Driver {
  private static void runAnalysis(FileProcessor fileProcessor,
Visitor... visitors) {
          Element myArrayList = new MyArrayList.Builder()
                  .withFileProcessor(fileProcessor)
                  .build();

          for (Visitor visitor : visitors) {
                  myArrayList.accept(visitor);
          }
  }

  private static void persistResults(Results... analysisResults) {
          for (Results results : analysisResults) {
                  results.writeToFile();
          }
```

```
    }

    public static void main(String[] args) {
            // TODO command-line args validation.
            // TODO command-line parsing and initialization of following variables.
            //      1. inputFilename.
            //      2. acceptableWordsFilename.
            //      3. k.
            //      4. topKOutputFilename.
            //      5. spellCheckOutputFilename.

            FileProcessor fileProcessor = new FileProcessor(inputFilename);

            Results topKFreqWordsResults = new TopKFreqWordsResults(topKOutputFilename);
            Visitor topKMostFreqAnalyzer = new TopKMostFreqAnalyzer(k, topKFreqWordsResults);

            Results spellCheckResults = new SpellCheckResults(spellCheckOutputFilename);
            Visitor spellCheckAnalyzer = new SpellCheckAnalyzer(acceptableWordsFilename, spellCheckResults);

            runAnalysis(fileProcessor, topKMostFreqAnalyzer, spellCheckAnalyzer);

            persistResults(topKFreqWordsResults, spellCheckResults);
    }
}
```

Submission

- Same as before

General Requirements

- Start early and avoid panic during the last couple of days.
- Class participation points will be given to students who submit 5 interesting input files along with corresponding output files on piazza. Note that the input/output should we well formatted and valid.
- Submit a README.md file (placed at the top level).
- Apply all design principles (wherever applicable).
- Separate out code appropriately into methods, one for each purpose.
- You should document your code. The comments should not exceed 72 coloums in width. Use javadoc style comments if you are coding in Java.
- Do not use "import XYZ.*" in your code. Instead, import each required type individually.

- All objects, in Java, that may be needed for debugging purposes should have the "toString()" method defined. By default, just place a toString() in every class.
- Every class that has data members, should have corresponding accessors and mutators (unless the data member(s) is/are for use just within the method.).
- If a class does not implement an interface, you should have a good justification for it. For example, it is ok to have an abstract base class and a derived class, wherein both do not implement interfaces. Note that the Driver code is written by end-users, and so the Results class must implement the interface, or else the source code for Results will have to be exposed to the end-user.
- Include javadoc style documentation for at least 3 classes. It is acceptable for this assignment to just have one parameter of a method described for each method's documentation.

General Design Requirements

- Same as other assignments.

Late Submissions

- The policy for late submissions is that you will lose 10% of the grade for each day that your submission is delayed.

Grading Guidelines

Grading guidelines have been posted here.