Assignment 4

Due date

- Due by 11.59 PM EST on November 17.

Submit your code as per the provided instructions. A signup sheet will be provided to you during class to setup an appointment with the TA to provide a demo of your project.

Updates

- Github Link - https://classroom.github.com/a/hzfVrV0f

Assignment Goal

Apply the design principles and the visitor pattern you have learned so far to develop and test code for the given problem.

- You are required to work alone on this project

Programming Language

You are required to program this project in Java.

Compilation Method

- You are required to use ANT for the following:
  - Compiling the code
  - running the code
  - Generating a tarball for submission
- Your code should compile and run on remote machines systems 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 students. Our code-comparison software can very easily detect similarities.

- Post to piazza if you have any questions about the requirements. Do NOT post your code to the listserv asking for help with debugging. However, it is okay to post design/concept questions on programming in Java/C/C++.

Project Description

Search based Customer Support for Trouble Shooting

The dSeaGate Portable Drive vendor wants to provide an online trouble shooter for customers. Your task is to design the engine for this purpose. The requirements are as follows.

- dSeaGate has a lot of product information. Each piece of information can be assumed to be a line in the input file.
- The product information is stored in two data structures.
    1. ArrayList of strings. Each string in the ArrayList represents a piece of information (line in the input file). The ArrayList should be composed by a wrapper class, MyArrayList, that is an *Element*.
    2. Tree of words. The tree class should be named *MyTree* and is an *Element*. Unlike the ArrayList, the nodes in the tree store only a single word. The outline of the node class is shown below.

```
                        public class Node {
                                private String word;

                                // Line numbers of the lines in the
input file in which the word was present.
                                // Feel free to use a collection other
than List.
                                private List<Integer>
lineNumbersFoundIn;

                                // rest of the code.
                        }
```

- Users can enter text (one string or many strings) to describe their problem. Only keywords or key-phrases are entered. For this assignment, accept keywords/key-phrases from a file named *userInput.txt*. Each entry in this file should be in a different line.
- dSeaGate does not want a sophisticated search algorithm. Instead, we are going to setup visitors for the following.
    1. *Exact Match*: In the ArrayList, find and write to Results all sentences that have an exact match for the entered keyword/key-phrase. So, for a key-phrase "problem detecting", a sentence that is an exact match could be "If Debian has a problem detecting the drive, it could be that the portable drive is not receiving enough power.". If none of the sentences

in the ArrayList were an exact match for the keyword/key-phrase, then the visitor should report "No exact match".

2. *Naive Stemming Match*: In the tree, find and write to Results the following information for all words of which the first keyword in the input keyword/key-phrase forms a part. So, for a key-phrase "detect error", the words "detecting", "detector" etc, would be matches. If none of the words in the Tree formed a naive stemming match, then the visitor should report "No naive stemming match".
    - Word Count - The number of such words.
    - Line numbers - The line numbers of lines in the input file that contained the matched words.
3. *Semantic Match*: Use an input file named *synonyms.txt* that contains synonyms in the format **<word>=<synonym>**. To determine a semantic match, find the synonym of the last word in a keyword/key-phrase and employ "Exact match". In the ArrayList, find and write to Results all sentences that are semantic matches to the user input keyword/key-phrase. For example, if the user enters "compatible CPU", and *synonyms.txt* contains a line "CPU=processor", a sentence that semantically matches could be "All products from AMD have a compatible processor as far as dSeaGate is concerned.". If none of the sentences in the ArrayList formed a semantic match, then the visitor should report "No semantic match". synonyms are bi-directional.

- The sentences that make up all of dSeaGate's information should be read from a file named *technicalInfo.txt*.
- The user input keywords/key-phrases should be read from a file named *userInput.txt*.
- Synonyms should be read from a file named *synonyms.txt*.
- Searches should be case insensitive.
- Data from the file will be read by a single thread. So, do NOT design for multi-threaded application.
- Use Debug in the following manner.
    1. Here is a sample MyLogger file
    2. 0: No output should be printed. Only error messages should be printed (for example, message from a catch caluse before exiting).
    3. 1: Only the search results should be printed
    4. 2: Design on your own and mention in the README what is printed at this debug granularity
    5. 3: Design on your own and mention in the README what is printed at this debug granularity
    6. 4: Design on your own and mention in the README what is printed at this debug granularity

Input and Output

- The program should take 4 arguments.
    - technicalInfo.txt
    - userInput.txt
    - synonyms.txt
    - output.txt
- The following are example input/output files. Some of the formatting (like underlines) in these examples are not mandatory. If you feel like formatting the output differently, then please include the format in your README file.
    - technicalInfo.txt
    - userInput.txt
    - synonyms.txt
    - output.txt

General Design Requirements

- Same as assignment 3

Code Organization

- Your directory structure should be the following.

```
-firstName_lastName_assign4
  ---troubleShootSearch
        ----- README.md
        ----- src
                 ----- build.xml
                 ---troubleshootsearch
                  ----------driver
                  ----------util
                  ----------other packages that you need
```

Submission

- Same as previous 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. **There is NO difference in penalty for assignments that are submitted 1 second late or 23 hours late** .

Grading Guidelines

Grading guidelines have been posted here.