Assignment 5

Due date

- 11.59 PM EST, Aug ~~4th~~ 5th

Git url

- https://classroom.github.com/a/PTlhx8ud

Submit your code as per the provided instructions.

Assignment Goal

Apply the design principles you have learned so far to develop software for the given problem.

Team Work

- You are required to work ALONE on this project

Programming Language

You are required to use Java.

Compiling and Running Commands

- Compilation: Your code should compile on remote.cs.binghamton.edu
- You are required to use ANT as the build tool for this assignment.

Policy on sharing of code

EVERY line of code that you submit in this assignment should be written by members of your team. Do NOT show your code to any other group. Our code-comparison software can very easily detect similarities.

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

Code snippets, for File I/O, if used from an online source should be cited by mentioning it in the README.md and also in the documentation of every source file in which that code appears.

Project Description

Text Decorators

- Design a program that accepts the following inputs.
  - -Dinput - Input file containing the text to process. The valid characters in the input file are [a-zA-Z0-9\.,\s] where
    - a-z - Lowercase alphabets.
    - A-Z - Uppercase alphabets.
    - \. - Period character.
    - , - Comma.
    - \s - Any whitespace character. Matches [\r\n\t\f\v].
  - -Dmisspelled - Misspelled words file containing words, one per line, that are misspelled.
  - -Dkeywords - Keywords file containing keywords, one per line.
  - -Doutput - Output file to which the processed input is persisted.
  - -Ddebug - Debug value.
- Design a class InputDetails that has datastructure(s) to store, retrieve and update sentences.
- Words in the input file are delimited by one or more spaces.
- Design the following decorators.
  - Design a SentenceDecorator. Each sentence is separated by a "." (period) character. The SentenceDecorator prefixes the sentence with **BEGIN_SENTENCE__** and suffixes the sentence with **__END_SENTENCE**. Note that the period character is not considered a part of the sentence.
  - Design a SpellCheckDecorator. This decorator checks whether any of the words is a misspelled word. A word is misspelled if it is present in the file provided with the commandline option -Dmisspelled. Misspelled words are prefixed with **SPELLCHECK_** and suffixed with **_SPELLCHECK**. *Note: The algorithm should be case insensitive.*
  - Design a MostFrequentDecorator. This decorator adds the prefix **MOST_FREQUENT_** and suffix **_MOST_FREQUENT** to all the occurrences of the most frequently occurring word in the entire input file. *Note: The algorithm should be case*

insensitive. Note: Think of what data structure can be used here to facilitate efficient search of the most frequently occurring word..

- Design a KeywordDecorator. This decorator checks whether any of the words is a keyword. A word is a keyword if it is present in the file provided with the commandline option -Dkeywords. Keywords are prefixed with **KEYWORD_** and suffixed with **_KEYWORD**. *Note: The algorithm should be case insensitive.*

Note: All decorators update the contents in-place. Hint: In SpellCheckDecorator and KeywordDecorator You can use String#indexOf(...) to search for words to make the program more robust.

- Assume that the input file is well formatted.
- Use loggers and debug values as used in the previous assignments. Each decorator will have its own debug value, that when set will result in that decorator writing the transformations made to a log.txt file. When writing the transformations to log.txt, prefix and suffix the log entry with the name of the decorator. Note that this is in addition to updating InputDetails in-place.
- Decorators are all derived types of AbstractTextDecorator (the name of the type should give enough information regarding its blueprint). AbstractTextDecorator declares a method called *processInputDetails()* that has no return type and no arguments.
- Decorators are instantiated by passing the decorator to wrap around and the InputDetails reference (the one to process) to the constructor.
- After processing input and applying all the decorators, the updated text in InputDetails should be persisted to the output file, the name of which will be provided via the commandline option -Doutput.
- The driver code should create the decorators, wrapping them as shown in the video. See below driver code snippet to get an understanding of how the decorators are instantiated.

```
InputDetails inputD = new InputDetails(inputFilename, outputFilename);
AbstractTextDecorator sentenceDecorator = new SentenceDecorator(null, inputD);
AbstractTextDecorator spellCheckDecorator = new SpellCheckDecorator(sentenceDecorator, inputD);
AbstractTextDecorator keywordDecorator = new KeywordDecorator(spellCheckDecorator, inputD);
AbstractTextDecorator mostFreqWordDecorator = new MostFrequentWordDecorator(keywordDecorator, inputD);

mostFrequentWordDecorator.processInputDetails();

((FileDisplayInterface) inputD).writeToFile();

------------------------------

public abstract class AbstractTextDecorator {
        public abstract void processInputDetails();
}

public class SentenceDecorator extends AbstractTextDecorator {
        private AbstractTextDecorator atd;
        private InputDetails id;

        public SentenceDecorator(AbstractTextDecorator atdIn, InputDetails idIn) {
                atd = atdIn;
                id = idIn;
        }

        @Override
        public void processInputDetails() {
                // Decorate input details.

                // Forward to the next decorator, if any.
                if (null != atd) {
                        atd.processInputDetails();
                }
        }
}
```

- The decorators have to be processed in the following order.
  1. MostFrequentWordsDecorator
  2. KeywordDecorator
  3. SpellCheckDecorator
  4. SentenceDecorator

Input/Output

An example input, along with the corresponding output is given below. Input:

> The Electors shall meet in their respective States and vote by Ballot for two Persons of whom one at least shall not be an Inhabitant of the same State with themselvs. And they shall make a List of all the Persons voted for and of the Number of Votes for each 3 which List they shall sign and sertify and transmit sealed to the Seat of the Government of the United States 3 directed to the President of the Senate .

Keywords:

```
government
persons
states
```

Misspelled Words:

```
themselvs
sertify
```

Output:

```
BEGIN_SENTENCE__MOST_FREQUENT_The_MOST_FREQUENT Electors shall meet in their respective KEYWORD_States_KEYWORD and vote by Ballot for two KEYWORD_Persons_KEYWORD of whom one at least shall not be an Inhabitant of MOST_FREQUENT_the_MOST_FREQUENT same State with SPELLCHECK_themselvs_SPELLCHECK__END_SENTENCE.BEGIN_SENTENCE__ And they shall make a List of all MOST_FREQUENT_the_MOST_FREQUENT KEYWORD_Persons_KEYWORD voted for and of MOST_FREQUENT_the_MOST_FREQUENT Number of Votes for each 3 which List they shall sign and SPELLCHECK_sertify_SPELLCHECK and transmit sealed to MOST_FREQUENT_the_MOST_FREQUENT Seat of MOST_FREQUENT_the_MOST_FREQUENT KEYWORD_Government_KEYWORD of MOST_FREQUENT_the_MOST_FREQUENT United KEYWORD_States_KEYWORD 3 directed to MOST_FREQUENT_the_MOST_FREQUENT President of MOST_FREQUENT_the_MOST_FREQUENT Senate __END_SENTENCE.
```

NOTES ON GRADING

- Class participation points will be given to students who post good questions on piazza seeking clarification on the assignment or finding ambiguities, and also to those who respond and participate to help make the discussion interesting and informative.
- Class participation points will be given to students who also post sample interesting input examples with corresponding output and metrics.

Sample Input Files sent by students in this course

Please check piazza.

Compiling and Running Java code

- Your submission must include a readme in markdown format with the name **README.md**.
- Your README.md file should have the following information:
  - instructions on how to compile the code
  - instructions on how to run the code
  - justification for the choice of data structures (in terms of time and/or space complexity).
  - citations for external material utilized.

Code Organization

- You should have the following directory structure (replace username with github username).
- ./csx42-summer-2020-assign5-username
- ./csx42-summer-2020-assign5-username/README.md
- ./csx42-summer-2020-assign5-username/.gitignore
- ./csx42-summer-2020-assign5-username/textdecorators
- ./csx42-summer-2020-assign5-username/textdecorators/src
- ./csx42-summer-2020-assign5-username/textdecorators/src/build.xml
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/driver
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/driver/Driver.java
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/util
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/util/FileProcessor.java
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/util/InputDetails.java
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/AbstractTextDecorator.java
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/SpellCheckDecorator.java

- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/KeywordDecorator.java
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/MostFrequentWordDecorator.java
- ./csx42-summer-2020-assign5-username/textdecorators/src/textdecorators/SentenceDecorator.java
- [Other classes and packages that you may want to add]

Submission

- Make sure all class files, object files (.o files), executables, and backup files are deleted before creating a zip or tarball (use the gitignore file for this). To create a tarball, you need to "tar" and then "gzip" your top level directory. Create a tarball of the directory csx42-summer-2020-assign5-username. We should be able to compile and execute your code using the commands listed in the README.md file.
- You are free to also use ANT to generate the tarball for you. However, note that this is not mandatory.
- Instructions to create a tarball
  - Make sure you are one level above the directory csx42-summer-2020-assign5-username.
  - tar -cvzf csx42-summer-2020-assign5-username.tar.gz csx42-summer-2020-assign5-username/
  - gzip csx42-summer-2020-assign5-username.tar
- Upload your assignment to Blackboard, assignment-5.

General Requirements

- Start early and avoid panic during the last couple of days.
- Separate out code appropriately into methods, one for each purpose.
- Class names should be nouns. Method names should be verbs.
- The name of the method should tell us what the method is doing, nothing more or nothing less.
- Always program to an interface or to the super type (depends on whether the top most level in the hierarchy is an interface or a class).
- You should document your code. The comments should not exceed 72 columns in width. Use javadoc style comments if you are coding in Java. Include javadoc style documentation. It is acceptable for this assignment to just have the return type described for each method's documentation.
- 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.).
- Java naming conventions MUST be followed.

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.