ASSIGNMENT 1 DESCRIPTION

Assignment 1

Due date

- 11.59 PM EST, June 10th

Git url

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

Submit your code as per the provided instructions.

Assignment Goal

A simple Java program.

Team Work

- No team work is allowed. Work individually. You cannot discuss the assignment with ANYONE other than the instructor and TA.

Programming Language

You are required to program using Java.

Compiling and Running Commands

- Compilation: Your code should compile on remote.cs.binghamton.edu with the following command: ant -buildfile wordPlay/src/build.xml all
-

- Running the code: Your code should run on remote.cs.binghamton.edu with the following command: ant -buildfile wordPlay/src/build.xml run -Dinput="input.txt" -Doutput="output.txt" -Dmetrics="metrics.txt"

Policy on sharing of code

EVERY line of code that you submit in this assignment should be written by you. Do NOT show your code to any other student. Do not copy any code from any online source. Code for File I/O or String operations, if found online, should be clearly cited, and you cannot use more than 5 lines of such online code.

Code downloaded in its entirety from an online repository of code (GitHub, BitBucket, etc.) and submitted as student's own work, even if citied, is considered plagiarism.

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.

Post to the piazza if you have any questions about the requirements. DO NOT post your code to the piazza asking for help with debugging.

Project Description

Assignment Goal: Develop a program, using Java, to process an input file containing sentences and also to calculate certain metrics.

- Prior to working on the assignment, please read through the grading instructions to understand the minimum requirement of the assignment.
- An input file contains sentences, one per line. Each sentence contains words delimited by <space> character. Each sentence terminates with a period.
- Each sentence is made up of alphanumeric words (characters in the set [a-zA-Z0-9]).
- The program should process the input file word by word.
- The program should do the following.
    - Rotate each word in a sentence to the right by x places where x is the index of the word in the sentence.
      Note: Indexing starts from 1. So first word is rotated by 1 place, second by 2 places and so on.
      Note: Only the characters of a word should be rotated. The order of words in the sentence should remain as is.
      Note: The rotation should be case senstitive. An upper case character in the input should remain in upper case in the output and lower case character should remain in lower case.
      Note: Period characters remain unchanged.
      For example, consider the sentence "Welcome to the course.". As it is mentioned that indices start from 1, the index of "Welcome" is 1, "to" is 2, "the" is 3 and "course" is 4. We therefore need to rotate "Welcome" by 1 position, "to" by 2 positions, "the" by 3 positons and "course" by 4 positions to the right.
      After performing rotation, the sentence would now read "eWelcom to the urseco.". This rotated sentence is to be written to the output file.
    - Calculate the following metrics and write them to the metrics file (one metric per line).
        - AVG_NUM_WORDS_PER_SENTENCE - Average number of words per sentence. Round to 2 decimal places. Format: **AVG_NUM_WORDS_PER_SENTENCE = <value>**
        - AVG_WORD_LENGTH - Average length (number of characters) of a word in the input file. Round to 2 decimal places. Format: **AVG_WORD_LENGTH = <value>**

The following rules MUST be followed.

1. FileProcessor code has been given to you. This should not be altered. You should use the FileProcessor for reading in the input file word by word. Read the documentation to understand how the FileProcessor works.
2. The input file should be processed one word at a time.
3. The program should not read in all the input and store it in a data structure.
4. You should implement your own function for rotating a word.

5. You should implement your own function for calculating each of the metrics.

INPUT

Your program should accept three files from the commandline - input file, output file and metrics file. These file names/paths will be provided using the following command-line options.

- ***-Dinput***: Input file path.
- ***-Doutput***: Path to the output file to which the sentences with the sorted words are written.
- ***-Dmetrics***: Path to the metrics file to which the metrics are written (one per line) in their respective formats.

EXAMPLES

input

Welcome to design patterns summer 2020.
Start working on this assignment quickly.

output

eWelcom to igndes ernspatt ummers 2020.
tStar ngworki no this nmentassig uicklyq.

metrics

AVG_NUM_WORDS_PER_SENTENCE - 6.0
AVG_WORD_LENGTH - 5.67

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.
- You should have the following directory structure (replace username with your github username). The word rotation code should be in the WordRotation class of the wordPlay.handler package. The metrics calculation code should be in the MetricsCalculator class of the wordPlay.metrics package.
- ./csx42-summer-2020-assign1-username
- ./csx42-summer-2020-assign1-username/wordPlay
- ./csx42-summer-2020-assign1-username/wordPlay/src
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/handler
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/handler/WordRotator.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/metrics
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/metrics/MetricsCalculator.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/util
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/util/FileDisplayInterface.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/util/FileProcessor.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/util/Results.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/util/StdoutDisplayInterface.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/driver
- ./csx42-summer-2020-assign1-username/wordPlay/src/wordPlay/driver/Driver.java
- ./csx42-summer-2020-assign1-username/wordPlay/src/build.xml
- /README.md
- [Other Java files you may need]
-

-

Code Organization

- Your directory structure should be EXACTLY as given in the code template. You are free to add additional packages.
    - Use the command on linux/unix to create an archive: *tar -cvzf csx42-summer-2020-assign1-username.tar.gz csx42-summer-2020-assign1-username/*.
    -
    - Use the command on linux/unix to extract the file: *tar -zxvf csx42-summer2020-assign1-username.tar.gz*.

Submission

- Make sure all class files, object files (.o files), executables, and backup files are deleted before creating a zip or tarball. To create a tarball, you need to "tar" and then "gzip" your top level

directory. Create a tarball of the directory csx42-summer-2020-assign1-username. We should be able to compile and execute your code using the commands listed above.

- Instructions to create a tarball
    - Make sure you are one level above the directory csx42-summer-2020-assign1-username.
    - tar -cvzf csx42-summer-2020-assign1-username.tar csx42-summer-2020-assign1-username/
    - gzip csx42-summer-2020-assign1-username.tar
- Upload your assignment to Blackboard, assignment-1.

General Requirements

- Start early and avoid panic during the last couple of days.
- Separate out code appropriately into methods, one for each purpose.
- 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.).

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.

ASSIGNMENT 2

Assignment 2

Due date

- 11.59 PM EST, June 24th

Git url

- https://classroom.github.com/a/jTL4tq8Y

Submit your code as per the provided instructions.

Assignment Goal

Implement the State Pattern to capture the project requirements.

Team Work

- No team work is allowed. Work individually. You cannot discuss the assignment with ANYONE other than the instructor and TA.

Programming Language

You are required to program using Java.

Compiling and Running Commands

- Compilation: Your code should compile on remote.cs.binghamton.edu with the following command: ant -buildfile src/build.xml all
- Running the code: Your code should run on remote.cs.binghamton.edu with the following command: ant -buildfile src/build.xml run -Dinput="<path/to/inputfile>" -Doutput="<path/to/outputfile>"

Policy on sharing of code

EVERY line of code that you submit in this assignment should be written by you. Do NOT show your code to any other student. Do not copy any code from any online source. Code for File I/O or String operations, if found online, should be clearly cited, and you cannot use more than 5 lines of such online code.

Code downloaded in its entirety from an online repository of code (GitHub, BitBucket, etc.) and submitted as student's own work, even if citied, is considered plagiarism.

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.

Post to the piazza if you have any questions about the requirements. Sending an email will only get a response requesting you to post on piazza. DO NOT post your code to piazza asking for help with debugging.

Project Description

Design and Implement a program for Youtube to categorize a single channel based on popularity.

- Prior to working on the assignment, please read through the grading instructions to understand the minimum requirement of the assignment.
- A Youtube channel has a popularity score, defined as *The average popularity score of all videos contained in the channel*. Based on its popularity score, a channel can be in one of the following states.
    - UNPOPULAR - This is the starting state of a channel. For a channel to be in this state, its popularity score should in the range [0, 1000].
    - MILDLY_POPULAR - For a channel to be in this state, its popularity score should be in the range (1000, 10000].
    - HIGHLY_POPULAR - For a channel to be in this state, its popularity score should be in the range (10000, 100000].

- ULTRA_POPULAR - For a channel to be in this state, its popularity score should be in the range (100000, INT_MAX].

- *Note: () represents an open interval (not including end points) and [] represents a closed interval (including endpoints). For example, $X \in (10, 20]$ iff $X \in R$ AND $10 < X \leq 20$.*
  Note: The popularity score of a video should not be negative. Therefore, popularity score of a video should be set to 0 if it is negative.
  Question to ask yourself at this point - Is Channel a state or a context?
  Enumerate the names of the states.

- A channel can hold multiple videos. The popularity score of a video is defined by the following three metrics.
  - Total number of views (≥ 0).
  - Total number of likes (≥ 0).
  - Total number of dislikes (≥ 0).

- Using the above information, the popularity score of a video at any point is given by the formula ***#Views + 2 * (#Likes - #Dislikes)*** where # signifies total count so far. The data for number of views, likes and dislikes are provided in the input file.
  Question to ask yourself at this point - How and where are videos stored?

- The current state of the channel determines whether advertisement requests are approved or rejected based on how long they are. The following are the rules for the same.
  - When state=UNPOPULAR, advertisements of length in range (1,10] are approved and the rest are rejected.
  - When state=MILDLY_POPULAR, advertisements of length in range (1, 20] are approved and the rest are rejected.

    - When state=HIGHLY_POPULAR, advertisements of length in range (1, 30] are approved and the rest are rejected.
    - When state=ULTRA_POPULAR, advertisements of length in range (1, 40] are approved and the rest are rejected.

- The requests to add advertisements of a certain length are also provided via the input file.
  Question to ask yourself at this point - Where are the advertisement requests processed?

Input Processing and Control flow

6. The input file is processed one line at a time.
7. Each line can correspond to one of the following.
    - Adding a video to the channel. Input format: **ADD_VIDEO::<video name>**.
    - Removing a video from the channel. Input format: **REMOVE_VIDEO::<video name>**.
    - Views, Likes and Dislikes. Input format: **METRICS__<video name>::[VIEWS=<delta in #views>,LIKES=<delta in #likes>,DISLIKES=<delta in #dislikes>]**.
      Note: There are no spaces before or after the comma character.
      Note: Views, Likes and Dislikes MUST be integers.
    - Advertisement requests. Input format: **AD_REQUEST__<video name>::LEN=<length>**.
      Note: Advertisement length MUST be an integer.
8. ADD_VIDEO - If the video already exists or the format of the input is invalid, throw the appropriate exception reporting a meaningful error message and terminate. If the input is valid, then add the video to the collection of videos in the channel. This new video is assigned an initial popularity score of 0. This operation is

performed by the current state.
The processing of this line should result in the string **<current state name>__VIDEO_ADDED::<video name>** being written to Results.

9. REMOVE_VIDEO - If the video does not exist or the format of the input is invalid, throw an appropriate exception reporting a meaningful error message and terminate. If the input is valid, then remove this video from the collection of videos in the channel. The channel's popularity score will need to be updated after the removal. This operation is performed by the current state.
   The processing of this line should result in the string **<current state name>__VIDEO_REMOVED::<video name>** being written to Results.

10. METRICS - If the video does not exist or the format of the input is invalid, throw an appropriate exception reporting a meaningful error message and terminate. If the input is valid, then update the metrics of the corresponding video and recalculate the popularity score of the channel. This operation is performed by the current state. The metric data is provided as delta changes.

    Consider the input line METRICS__testvideo::[VIEWS=10,LIKES=10,DISLIKES=-20]. This means that the total views increased by 10, total likes increased by 10 and the total dislikes decreased by 20, for the video named 'testvideo'.

    The processing of this line should result in the string **<current state name>__POPULARITY_SCORE_UPDATE::<new popularity**

score of channel> being written to Results.
Note: unlike likes and dislikes, the delta in the number of views cannot be negative.
Note: If there is a decrease in the number of likes or dislikes, it cannot be more than the total number of likes or dislikes received thus far for the video.

11. AD_REQUEST - If the video does not exist or the length is negative, throw an appropriate exception reporting a meaningful error message and terminate. If the input is valid, report if the request is approved or rejected based on whether the length of the advertisement falls within the current state's acceptable ad length range (mentioned above).
The processing of this should result in the string **<current state name>__AD_REQUEST::<APPROVED / REJECTED>** being written to Results.

12. If any of the lines in the input file are invalid, then the whole input is to be considered invalid.
13. Upon processing of a ADD_VIDEO, REMOVE_VIDEO or METRICS input if the new popularity score of the channel falls in a range acceptable by a state other than the current state then a state change occurs.
14. Once all the lines of the input file have been processed cast the results instance to the appropriate interface and call the necessary method to persist the results to the output file.

INPUT

Your program should accept two files from the commandline - input file and output file. These file names/paths will be provided using the following command-line options.

- ***-Dinput***: Path to the Input file.
- ***-Doutput***: Path to the output file.

EXAMPLES

input

ADD_VIDEO::video1
ADD_VIDEO::video2
METRICS__video1::[VIEWS=1000,LIKES=20,DISLIKES=20]
AD_REQUEST__video1::LEN=8
METRICS__video2::[VIEWS=2000,LIKES=400,DISLIKES=20]
METRICS__video1::[VIEWS=20000,LIKES=1000,DISLIKES=-10]
AD_REQUEST__video2::LEN=39
METRICS__video2::[VIEWS=50,LIKES=-50,DISLIKES=0]
REMOVE_VIDEO::video2
ADD_VIDEO::video3
METRICS__video3::[VIEWS=2000,LIKES=100,DISLIKES=20]
METRICS__video1::[VIEWS=0,LIKES=-1000,DISLIKES=500]
ADD_VIDEO::video4
METRICS__video4::[VIEWS=100,LIKES=5,DISLIKES=0]
REMOVE_VIDEO::video1
AD_REQUEST__video3::LEN=15
REMOVE_VIDEO::video3
REMOVE_VIDEO::video4

output

UNPOPULAR__VIDEO_ADDED::video1
UNPOPULAR__VIDEO_ADDED::video2
UNPOPULAR__POPULARITY_SCORE_UPDATE::500
UNPOPULAR__AD_REQUEST::APPROVED
UNPOPULAR__POPULARITY_SCORE_UPDATE::1880
MILDLY_POPULAR__POPULARITY_SCORE_UPDATE::12890
HIGHLY_POPULAR__AD_REQUEST::REJECTED
HIGHLY_POPULAR__POPULARITY_SCORE_UPDATE::12865
HIGHLY_POPULAR__VIDEO_REMOVED::video2
HIGHLY_POPULAR__VIDEO_ADDED::video3
HIGHLY_POPULAR__POPULARITY_SCORE_UPDATE::12590
HIGHLY_POPULAR__POPULARITY_SCORE_UPDATE::11090
HIGHLY_POPULAR__VIDEO_ADDED::video4
MILDLY_POPULAR__POPULARITY_SCORE_UPDATE::7430
MILDLY_POPULAR__VIDEO_REMOVED::video1
MILDLY_POPULAR__AD_REQUEST::APPROVED
MILDLY_POPULAR__VIDEO_REMOVED::video3
UNPOPULAR__VIDEO_REMOVED::video4

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.
- You should have the following directory structure (replace username with your github username).
- ./csx42-summer-2020-assign2-username
- ./csx42-summer-2020-assign2-username/README.md
- ./csx42-summer-2020-assign2-username/.gitignore
- ./csx42-summer-2020-assign2-username/channelpopularity
- ./csx42-summer-2020-assign2-username/channelpopularity/src
- ./csx42-summer-2020-assign2-username/channelpopularity/src/build.xml
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/operation
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/operation/Operation.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/context

- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/context/ContextI.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/StateName.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/StateI.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/AbstractState.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/factory
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/factory/SimpleStateFactoryI.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/state/factory/SimpleStateFactory.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/FileProcessor.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/Results.java

- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/FileDisplayInterface.java
- ./csx42-summer-2020-assign2-username/channelpopularity/src/channelpopularity/util/StdoutDisplayInterface.java
- [Other Java files and packages you may need]

Code Organization

- Your directory structure should be EXACTLY as given in the code template. You are free to add additional packages and source files.

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-assign2-username. We should be able to compile and execute your code using the commands listed above.
- Instructions to create a tarball
    - Make sure you are one level above the directory csx42-summer-2020-assign2-username.
    - tar -cvzf csx42-summer-2020-assign2-username.tar.gz csx42-summer-2020-assign2-username/
- Upload your assignment to Blackboard, assignment-2.

General Requirements

- Upload a picture of a hand-drawn state diagram showing the states and the state changes to piazza and tag it with the "assignment2-state-diagram" folder.
- 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.
- 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.
- Design Guidelines and Considerations
    - The context should use the simple factory design pattern to fetch the required state. Use the following sub points to get further clarity on this.
        i. The constructor of the context accepts an instance of SimpleStateFactory.

ii. The create(...) method of SimpleStateFactory accepts an enum representing the state to be instantiated.
iii. The setState(...) method of the context accepts an enum representing the state to change to. This is provided to the SimpleStateFactory instance to fetch an instance of the respective state.
iv. Code snippet shown below should help.// Assume imports.public ChannelContext implements ContextI { private StateI curState; private Map<StateName, StateI> availableStates; public ChannelContext(SimpleStateFactoryI stateFactoryIn, List<StateName> stateNames) {// initialize states using factory instance and the provided state names.// initialize current state. } // Called by the States based on their logic of what the machine state should change to. public void setCurrentState(StateName nextState) {if (availableStates.containsKey(nextState)) { // for safety.curState = availableStates.get(nextState);} }}

- As the logic for calculating the popularity score for videos, and in turn of the channel, is the same regardless of the current state, this should be put in a method in an abstract class that implements the StateI interface. Note that this would result in the states extending the abstract class instead of directly implementing the StateI interface.
- Enumerating the state names and operations follows good design practices.
- The name of a state is always fixed. Therefore, the state implementations should have their name as a constant

member (Question: What access specifiers and modifiers would you use?).
    - Can popularity score be performed in constant time? This would increase the processing throughput.
    - The state should not directly write to the output file. Instead, they should all share access to the same Results instance and store the output strings in it. Once all processing is complete, the driver code casts the results instance to the specific interface and calls the associated method to persist the data either to the output file.
    - Always program to an interface. This applies to java standard library as well. For example, when instantiating an ArrayList, write *List<? extends Object> l = new ArrayList<? extends Object>();*.
- The following rules MUST be followed.
    b. FileProcessor code has been given to you. This should NOT be altered. Use the FileProcessor for reading in the input file line by line. Read the documentation to understand how the FileProcessor works.
    c. The input file should be processed one line at a time.
    d. The program should not read in all the input and store it in a data structure.
    e. Use enums to represent operations to be performed.
    f. Use enums to represent states.
    g. State change, if any, has to happen after performing respective operation and not before performing the operation.
    h. The output format has to be exactly the same as that shown. Failure to generate output adhering to the given format will result in loss of points.

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** .
- If you are using slack days, please mention it in the README file.

Grading Guidelines

Grading guidelines have been posted here.

ASSIGNMENT 3

Assignment 3

Due date

- 11.59 PM EST, July 10th

Git url

- https://classroom.github.com/a/QNRu9iUa

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 need to work alone on this assignment.

- You cannot discuss with anyone the design patterns(s) to be used for this assignment.

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 you. 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

Replica System for Student Records

- Each Replica System needs to be represented as a tree, wherein each node of the tree corresponds to a Student Record.
- Each Tree needs to be given a unique ID (0, 1, 2, for example). Use a final int in each Tree to save this id. Write a utility class with a

static method to return this unique id to be returned whenever it is called.

- Create a class called StudentRecord to serve as nodes for the tree. A student record consists of the following:
    - a bNumber (4 digit positive integer),
    - a String for "firstName",
    - a String for "lastName",
    - a double value for "gpa",
    - a String for "major",
    - and a set of Strings for skills. The max number of skills that will be listed in the input file will be 10.
- Write code for a tree that has the features to insert and search Student Records. You need to write code for the tree by yourself. However, you are allowed to get help from a book or online source (for example the source code for BST, AVL, 2-3, 2-3-4, etc.) If you do so, cite the URL both in your source code and also README. It will be considered CHEATING if you do not cite the source of any code on tree that you use/adapt.
- The three trees (replica_0, replica_1, and replica_2) should be separate instances of the same Tree. Note that you should appropriately use Camel case (as per Java naming convention)) instead of the names I have listed here.
- Each Node of the tree should implement both the SubjectI and the ObserverI interface.
- As you need to modify the code to implement the Observer pattern, you can't use the in-built Observer pattern in Java.
- Populate the tree using the data from an input file provided with the -Dinput commandline option. A line in the input file has the following format:<B_NUMBER>:<FIRST_NAME>,<LAST_NAME>,<GPA>,<MA

JOR>,[<SKILL>,[<SKILL>, ...]]Example is shown below.1234:John,Doe,3.9,ComputerScience,Skill3,Skill1,Skill2,Skill5,Skill4,2345:John,Doe,3.9,Chemistry,Skill0,Skill7,Skill8,Skill5,Skill3,Skill4,Skill3,Skill91124:Jane,Doe,3.9,Chemistry,Skill9,Skill8,Skill7,Skill6,Skill2,Skill3,Skill4...

- The input lines may have bNumbers that are repeated. If a node already exists, change the data member values in the Node with the new values from the latest input line. Note that by using a set for the skills, only unique Strings can be added.
- Your TreeHelper should be such that when you create a node (replica_Node_0), you should clone it twice to get two Nodes (let's say replica_Node_1 and replica_Node_2). Setup these three nodes to be replicas of each other. The TreeHelper is a helper that helps build the tree recrusively.
    - So, replica_Node_1 and replica_Node_2 should be observers of replica_Node_0. As replica_Node_0 is the subject in this case, you should store the references of replica_Node_1 and replica_Node_2 in a data structure for listeners (array list of references, for example).
        - replica_Node_0 should be inserted in replica_tree_0.
        - replica_Node_1 should be inserted in replica_tree_1.
        - replica_Node_2 should be inserted in replica_tree_2.
    - replica_Node_0 and replica_Node_1 should be observers of replica_Node_2. Continue with steps corresponding to the ones listed in the 1st bullet.
    - replica_Node_0 and replica_Node_2 should be observers of replica_Node_1. Continue with steps corresponding to the ones listed in the 1st bullet.
- You do NOT need to write code to delete nodes, as the input and modification files should not require deletion of any node.

- The Replica trees should NOT be setup a observers or subjects. The replicas should be setup at only the Node level.
- Process one line at a time from the modification file provided using the -Dmodify commandline option. The file has the following format (first entry corresponds to the replica number, the 2nd to the bNumber, and 3rd to the value that needs to be searched and replaced/modified).<REPLICA_ID>,<B_NUMBER>,<ORIG_VALUE>:<NEW_VALUE>An example is shown below.0,1234,John:John71,2345,Chemistry:MathematicalSciences 2,1234,Skill1:Skill9...
- The modification file will not have any request to change the GPA.
- If the request for modified value is empty (for example, "Mathematical Sciences:"), print a meaningful message to error file and proceed to the next line.
- Search for the node with the bNumber in the line from the modification file, and then modify the corresponding attribute value in that Node. If that attribute value does not exist to modify, then ignore and move to the next line. Once the change to the specified replica_Node_X is done, then the change should be sent to both the observer nodes. Note that the updates should be sent before the next line is processed.
- The update(...) call should do the following:
    - determine if the call on the observer is for INSERT or MODIFY.
    - search for the appropriate node using the bNumber.
    - If it is an INSERT based update call, then use the latest values. Note that for skills, insert should only add to the existing set, and not delete any existing skill.

    - If it is a MODIFY based update call, then every occurrence of the origValue should be replaced with the provided modifiedValue.
- From the command line accept the following args:
    - Input file path with the *-Dinput* commandline option.
    - Modification path file with the *-Dmodify* commandline option.
    - Output file path for tree 1 with the *-Dout1* commandline option.
    - Output file path for tree 2 with the *-Dout2* commandline option.
    - Output file path for tree 3 with the *-Dout3* commandline option.
    - Error file path with the *-Derror* commandline option.
    - Debug value with the *-Ddebug* commandline option.
- Do not hardcode the file names in your code. Use the file names provided via command line.
- Add a method to your tree, printNodes(Results r, ...), that stores in Results the list of skills for each student, sorted by the bNumber in ascending order.
- Your driver code should do the following:
    - Read command line arguments.
    - Build the three trees, based on the input file and the observer pattern. It is recommended that you use a TreeHelper to build each tree recursively. The TreeHelper can hold references to the roots of the 3 trees that are being built, and have helper methods to insert, printNodes(...), etc. It is also ok to build your trees in other ways as long as the design will work if the number of replicas is increased/decreased.

    - Apply updates according to modification file.
    - Create a new Results instance and call printNodes(...) on each of the three trees to store the bNumber and list of skills to store in Results. So, each of the three trees will use a different instance of Results.
    - Call a method in Results, via the method in FileDisplayInterface, to write the data stored in Results to three output files, for the three trees. You can run a "diff" on the three output files to make sure your Observer pattern worked correctly.
- When you read in an input file, you should insert the skill into the main tree and in that tree update the node corresponding to the bNumber. If a node with the bNumber exists, you should update that node in the main tree, notifyAll(...) to the two corresponding listener nodes. If no such node with that bNumber exists in the main tree, then do the following:
    - create a new Node (lets call it new node)
    - clone this new node twice, and setup all the listeners and subjects.
    - insert the nodes in their respective trees, recursive insert from the root.
- Note that notifyAll(...) could be used for processing modification file and also input file (for example, when you need to add a skill to an existing bNumber from the input file). So, add enum as a parameter to the notifyAll(...) call in the SubjectI, also to update(...) in ObserverI, to indicate whether the notifyAll(...) call, which calls update(...) on the observer nodes, is for INSERT or MODIFY.

- Calls made to notifyAll(...) should be propagated to the observers. However, calls to update(...) on a node should not be propagated any further to prevent a cycle (infinite loop).
- In your README.md, briefly describe how the observer pattern has been implemented, and the URLs from where you borrowed ideas/code for the tree implementation.
- For class participation, post interesting, but valid, input and modification files to piazza.
- Create and use your own MyLogger scheme for debugging. Here is an example of how you should use MyLogger.

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.
- You should have the following directory structure (replace username with your github username).
- ./csx42-summer-2020-assign3-username
- ./csx42-summer-2020-assign3-username/README.md
- ./csx42-summer-2020-assign3-username/.gitignore
- ./csx42-summer-2020-assign3-username/studentskills/src/build.xml
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/driver
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/driver/Driver.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/FileProcessor.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/Results.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/FileDisplayInterface.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/StdoutDisplayInterface.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/util/MyLogger.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree

- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/TreeHelper.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/StudentRecord.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/SubjectI.java
- ./csx42-summer-2020-assign3-username/studentskills/src/studentskills/mytree/ObserverI.java
- [Other Java files and packages you may need]

Code Organization

- Your directory structure should be EXACTLY as given in the code template. You are free to add additional packages and source files.

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-assign3-username. We should be able to compile and execute your code using the commands listed above.
- Instructions to create a tarball
    - Make sure you are one level above the directory csx42-summer-2020-assign3-username.
    - tar -cvzf csx42-summer-2020-assign3-username.tar.gz csx42-summer-2020-assign3-username/
    - gzip csx42-summer-2020-assign3-username.tar

- Upload your assignment to Blackboard, assignment-3.

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.
- 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.

ASSIGNMENT 4

Assignment 4

Due date

- 11.59 PM EST, July 22th

Git url

- https://classroom.github.com/a/5ctzf7o1

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 in a team of 2 students for this project.~~ 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

Program with visitors to determine features in two input files that have integers.

- The input for the program consists of two separate input files, both with a positive 2 digit integer in each line. Assume that the input may contain duplicates in the individual files.
- Create an interface MyArrayI that defines an API for an Abstract Data Type (ADT) representing an array. In addition, MyArrayI is an *Element*.
- Define an ADT, MyArray, that implements MyArrayI and stores an internal array of integers. The internal array should be created with an initial capacity of 10 and increased by 50% in capacity each time an integer has to be added beyond the current capacity.
- Each MyArray object will store the integers from a single input file.

- Design a visitor, PopulateMyArrayVisitor, that reads from a file and populates MyArray one integer at a time. The visitor should use an instance of the FileProcessor to read from the input file one line at a time (one integer at a time). Check for the boundary condition that the input file does not exist or is empty. Throw an exception and exit if a string from the file cannot be converted to an integer. Other than that, assume the input file is well formed and no other exceptions need to be handled.
- The PopulateMyArrayVisitor can take the name of the input file in its constructor or have a setX(..) method for it. You need to apply this visitor once for each of the two input files to get two instances of MyArray that are populated with integers.
- Create an interface MyArrayListI that defines an API for a ADT representing an arraylist. In addition, MyArrayListI is an *Element*.
- Define an ADT, MyArrayList, that implements MyArrayListI and maintains an internal array of the MyArray objects.
- Design a visitor, CommonIntsVisitor, that determines the integers that are common in the two ADTs stored in MyArrayList, and stores those integers (without duplicates) in an appropriate data structure in Results.
- Design a visitor, MissingIntsVisitor that determines the 2 digit integers (between 00 and 99) that are missing in MyArray and stores them in an appropriate data structure in Results.
- The output files should contain a single integer per line.
- Use a singleton Logger and design your own debugging scheme.
- The driver should accept the input file names, output file names, and debug value, via the command line.
    i. *-Dinput1* - First input file containing 2 digits integers per line.
    j. *-Dinput2* - Second input file containing 2 digits integers per line.

k. *-Dcommonintsout* - Output file to store results of applying CommonIntsVisitor.
l. *-Dmissingintsout* - Output file to store results of applying MissingIntsVisitor.
    - Recommended for this assignment - upload to the same output file. Include headings to indicate which MyArray or input file the output corresponds to.
    - Also acceptable - upload to missing integers for each MyArray object to a different output file.

m. *-Ddebug* - Debug value.

- The driver should do the following:
    - Create required instances of Results.
    - Create instance of FileProcessor.
    - Create instances of the visitors.
    - Create two instances of MyArray.
    - Use the PopulateMyArrayVisitor to populate two instances of MyArray.
    - Apply CommonIntsVisitor to determine common ints in ADTs stored in MyArrayList.
    - Apply MissingIntsVisitor separately to each of the MyArray instances.
    - Call appropriate methods in Result instances to print the output of each of the visitors.
- Helpers:
    n. There is only a single visitor interface.
    o. Unlike the traditional viistor pattern where there is just a single visit(...) method in the Visitor interface, you will need to overload the method here for each of the two visitors.
    p. When calling methods on the ADTs/Elements make sure to cast to the appropriate interface and then call the method.

For example, ADT methods should be called by casting to the interface that defines the API for the ADT.

q. No business logic should be written in the driver code.

r. ADT design

- Empty constructor.
- Explicit value constructor.
- Getters and Setters.
- Override toString.
- Empty finalize method.
- Override Clone.

s. Driver code

- Driver code should not include business logic.
- Simple and concise.

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-assign4-username
- ./csx42-summer-2020-assign4-username/README.md
- ./csx42-summer-2020-assign4-username/.gitignore
- ./csx42-summer-2020-assign4-username/arrayvisitors
- ./csx42-summer-2020-assign4-username/arrayvisitors/src
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/build.xml
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/driver
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/driver/Driver.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/adt
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/adt/MyArrayI.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/adt/MyArrayListI.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/adt/MyArray.java

- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/adt/MyArrayList.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/visitors
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/visitors/Element.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/visitors/Visitor.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/visitors/PopulateMyArrayVisitor.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/visitors/CommonIntsVisitor.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/visitors/MissingIntsVisitor.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/util
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/util/FileProcessor.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/util/Results.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/util/FileDisplayInterface.java
- ./csx42-summer-2020-assign4-username/arrayvisitors/src/arrayvisitors/util/StdoutDisplayInterface.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-assign4-username. We should be able to compile and execute your code using the commands listed in the README.md file.
- Instructions to create a tarball
    - Make sure you are one level above the directory csx42-summer-2020-assign4-username.
    - tar -cvzf csx42-summer-2020-assign4-username.tar.gz csx42-summer-2020-assign4-username/
    - gzip csx42-summer-2020-assign4-username.tar
- Upload your assignment to Blackboard, assignment-4.

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.
- 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.

ASSIGNMENT 5

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();((File DisplayInterface) 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;}@Overridepublic 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.
    t. MostFrequentWordsDecorator
    u. KeywordDecorator
    v. SpellCheckDecorator
    w. 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.