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.