Homework Assignment 2

SE 311: Software Architecture II

In this assignment, you will extend the KWIC system in homework assignment 1 to make it a KWIC product family with the following features:

Requirements:

1. Support multiple types of input and output: For all the three functions in HA1, *kwic-processing*, *keyword-search,* and *index-generation*, the user can have the following options:

- The input can be either a .txt file in which each sentence takes one line or a .csv file in which each sentence is separated by a period, ".";
- The output can be exported to a .txt file or the console.

2. Support multiple alphabetizing policies: For the *kwic-processing* and *index-generation* functions, the user can choose to sort ascendingly or descendingly.

3. Support trivial word filtering: for the *index-generation* function, the user can choose to filter out a set of pre-defined trivial words.

The user should be able to make these configurations using a `config.properties` file as attached in BBLearn.

Submission Guidelines:

You need to turn in the following items from BB Learn:

Item 1: UML Class diagram

Turn in a ***1-page PDF*** of your UML exported from draw.io or startUML.

Do not handwrite your diagram or reverse engineer it from the source code.

If you applied design patterns, please label the name of the patterns and the pattern role a class takes.

Item 2: UML Sequence Diagram

Turn in a ***1-page PDF*** of your sequence diagram, exported from draw.io or startUML, demonstrating which and how objects work together to accomplish the following scenario:

Index-generation: *Given the list of sentences, the system will output an index with all the keywords in alphabetic order. Each keyword should be followed by index(es) of the original line(s) containing the keyword. If a keyword is contained in multiple lines, all the line numbers should be listed.*

Item 3: Source code in Java, VSCode project

Turn in an VSCode project that we can open, view the source code, and run the program.

- The source code must include a `config.properties` file so that our TAs can configure and execute your programs as shown in a script (as shown in the Appendix). This means that your program must accept arguments as indicated in the scripts.
- A sample `config.properties` file is copied in the Appendix below. Your config.properties must have all the keys shown in the sample config.properties file. The values for the keys must follow the comments.
- You can also download sample code from BBLearn to study how to use the `config.properties` file and object loader.
- In this homework, you only need to apply strategy patterns for input and output. If you have more strategy objects other than input and output and need to configure more objects, you can add them to the configuration file. The `config.properties` file must contain the keys as shown in the Appendix, but it can have more key-value pairs if needed.
- For the *keyword-search* function, use brackets to highlight the searched keyword in the console output, such as:

    Crouching [Tiger] Hidden Dragon

Item 3: DSMs for both HW1 and HW2

Analyze your HW1 and HW2 using DV8. For each homework, submit a `.dv8-dsm` file and a `.dv8-clsx` file. The `.dv8-clsx` file should cluster the DSM to reflect the structure in which you want your design to be presented.

- If you have refactored HW1 to create HW2—for example, by applying patterns or removing unnecessary classes—please document these changes in a Readme.txt file.
- Please note that we will use these DSM to assess your design structure. For example, Your source code DSM could have different dependency numbers, but the structure should have the following features:
    - Each concrete input/output object should be independent, meaning they can be removed or revised without influencing other parts of the system, as reflected in the DSM.

- The control class should only depend on abstract input, output, sorting, and shifting interfaces/classes, line storage class, and configuration reader classes, but not concrete input/output strategy classes.
- "There should be no unnecessary classes—for example, an Input class that merely aggregates an Input interface is redundant.

Appendix

1. Sample `config.properties` file

```
#Input/Output Configurations
#You can change the values but not the keys
InputFileName=SampleInputFile.txt
OutputFileName=SampleOutputFile.txt


#Locate input/output objects
#You can change the values but not the keys
TxtInputObj=drexel.se311.input.TxtInput
CSVInputObj=drexel.se311.input.CSVInput
TxtOutputObj=drexel.se311.input.TxtOutput
ConsoleOutputObj=drexel.se311.input.ConsoleOutput


#The value of Input can be either "TxtInputObj" or "CSVInputObj"
Input=TxtInputObj

#The value of Output can be either "TxtOutputObj" or
"ConsoleOutputObj"
Output=ConsoleOutputObj

#Sorting Configurations
#The value of Order can be either "Ascending" or "Descending"
Order=Ascending

#Index Configs
#The value of WordFiltering can be either "Yes" or "No"
WordFiltering=Yes

#The value of TrivialWords can be customized
TrivialWords=”a,the,and,an,this,that”
```

2. Testing scripts

```
java -jar hw2.jar kwic-processing config.properties
java -jar hw2.jar keyword-search [keyword] config.properties
java -jar hw2.jar index-generation config.properties
```

3. Frequently Asked Questions (FAQ)

Q1: How do I generate a `.jar` file from a Java project?

A: The simplest way is to run the following command:

```
jar cf MyApp.jar -C out
```

For more detailed instructions, you can ask ChatGPT:
"How do I create a `.jar` file from my Java project?"

Q2: Does "trivial word filtering" apply to all three functions?

A: As discussed in class, if `WordFiltering=Yes`:

1) **Index-generation:** Predefined trivial words should be ignored.
2) **KWIC-processing:** You can decide whether to generate circularly shifted lines for predefined trivial words—either approach is acceptable.
3) **Keyword-search:** This option does **not** apply. That is, if the original sentences contain trivial words and a user searches for them, the sentences containing those words will still be returned.

Q3: Should my program allow users to enter input and/or output file names interactively through the CLI?

A: No. As discussed in class, the TAs should only need to run the provided three scripts without having to determine how to pass additional parameters to your program.