Homework Assignment 1

SE 311: Software Architecture II

In this assignment, you will design and implement a **Key Word in Context (KWIC)** program that accepts a .txt file with a set of sentences as input and supports the following functions:

Requirements:

1. KWIC Processing:

Given a file with a set of lines, output all circular shifted lines in alphabetic order. Each line is followed by the original line number. For example, if the input is:

Sense and Sensibility
Architecture Software
Crouching Tiger Hidden Dragon

Then the output should be:

Index	CircularShifted Lines	Original Line Index
1	and Sensibility Sense	1
2	Architecture Software	2
3	Crouching Tiger Hidden Dragon	3
4	Dragon Crouching Tiger Hidden	3
5	Hidden Dragon Crouching Tiger	3
6	Sense and Sensibility	1
7	Sensibility Sense and	1
8	Software Architecture	2
9	Tiger Hidden Dragon Crouching	3

2. Keyword Search:

The user enters a keyword, and the system will return all the sentences containing the keyword. In the returned lines, the keyword will be highlighted. The system will also return the total number of lines containing the keyword. For example, if the input is "*Tiger*," the output should look like this:

1 sentence is found:
Crouching **Tiger** Hidden Dragon

If the keyword is not contained in any sentences, the system will return:

``*[keyword] not found.*”

3. 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. Following is an example:

and, 1
Architecture, 2
Crouching, 3
Dragon, 3
Hidden, 3
Sense, 1
Sensibility, 1
Software, 2
Tiger, 3

The user should be able to assess these functions using console commands.

Possible Future Changes to consider:

1. The **input format** may change: in this homework, your program should accept a plain .txt file, but in the future, it may need to accept .csv files or other file formats.

2. The **index generation policy** may change: you can index all the keywords in this homework. In the future, your program should allow the user to filter out words that do not need to be indexed, such as “*a, an, the, and, ...*”

3. The **alphabetizing policy** may change: in this homework, you can choose any sorting policy, but in the future, your program will need to allow the user to select from multiple sorting policies, such as ascending, descending, case sensitive, or case insensitive.

4. The **output method** may change: in this homework, you can choose to output these results to the console, but in the future, you may need to output them to a file or a webpage. The output format may also change.

Submission Guidelines:

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

Item 1: Data Design:

Briefly answer the following questions:

- What data structure did you use to store the lines read from the input file?
- What data structure did you use to store circular shifted lines or their indexes?
- What data structure did you use to store sorted lines or their indexes?

Item 2: Module Design: UML Class diagram

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

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

Item 3: Logic and Concept/Naming Design: Source code in Java

Turn in an VSCode project that we can open, view the source code, and run the program. You should also include a readme.txt to indicate how to execute the program.

Basic Naming Conventions:

- Name entities with concatenated words as in *displayTime*
- Begin class names with capitals
- Name variables beginning with lowercase letters
- Name constants with capital as in I_AM_A_CONSTANT
- Use *get …, set…* and *is…* for accessor methods as in *getTime(), setName(), isBox()*, where the latter returns a Boolean value
- Name functions starting with a verb as in calculateWeight()
- Consider a convention for parameters. One convention is to use the prefix *a*, as in *sum(int anInteger1, int anInteger2)*
- Files, classes, methods should be named to reflect domain concepts or functions.