Homework 1: KWIC - KWAC - KWOC

Code Repository URL: https://github.com/jasontbk/KWIC-KWAC-KWOC-.git

Name	Lin Weizhi	Jason Teo Boon Kuang
Matriculation Number	A0087084X	A0084895J

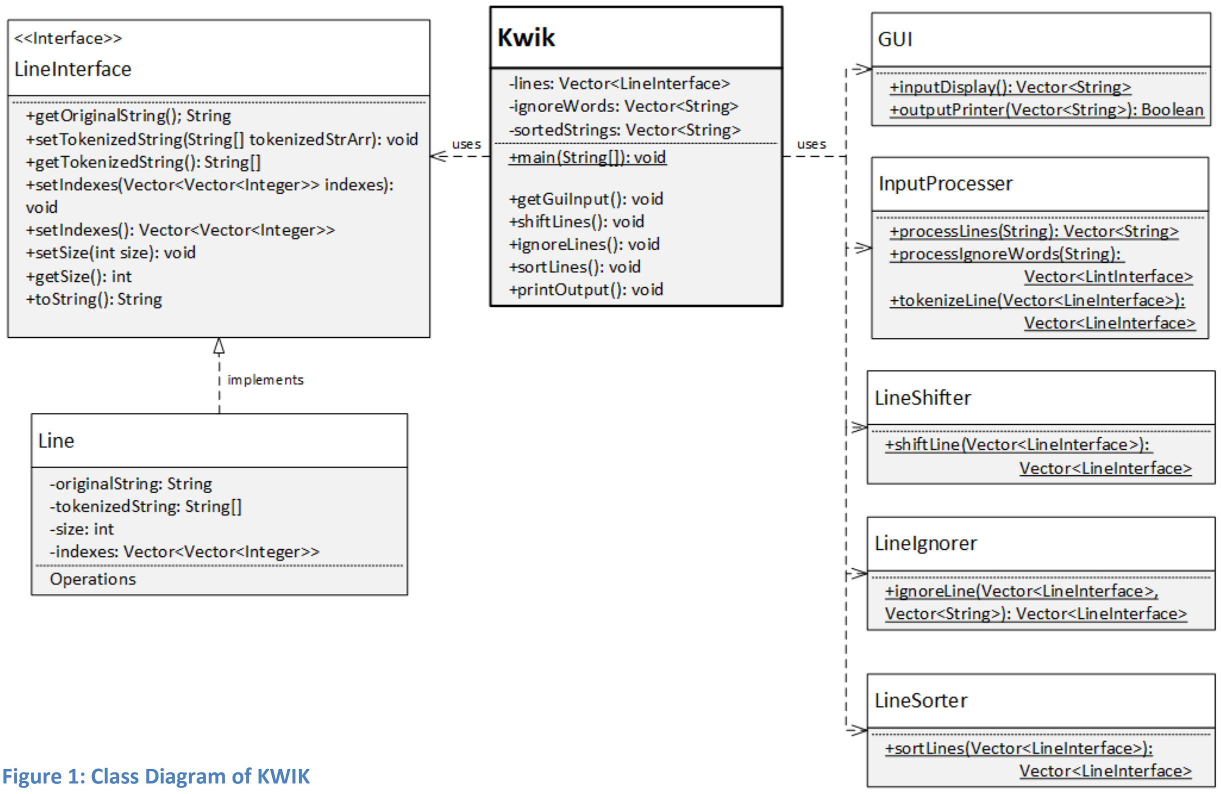
Introduction

KWIC is a Key Word In Context index system. It takes in a list of lines, such as movie titles, and a list of "words to ignore", generating a Key Word In Context index of the titles. Users can interact with KWIK by providing a text file containing required information or by manually inputting it.

Design

KWIC comprises of 8 modules that are designed to process different stages. It follows a batch processing design as each stage will process through all input before proceeding to the next stage.

The class diagram of the program is as follows in Figure 1 below.



<<Interface>>
LineInterface
+getOriginalString(); String
+setTokenizedString(String[] tokenizedStrArr): void
+getTokenizedString(): String[]
+setIndexes(Vector<Vector<Integer>> indexes): void
+setIndexes(): Vector<Vector<Integer>>
+setSize(int size): void
+getSize(): int
+toString(): String
uses
Kwik
-lines: Vector<LineInterface>
-ignoreWords: Vector<String>
-sortedStrings: Vector<String>
+main(String[]): void
+getGuiInput(): void
+shiftLines(): void
+ignoreLines(): void
+sortLines(): void
+printOutput(): void
uses
GUI
+inputDisplay(): Vector<String>
+outputPrinter(Vector<String>): Boolean
InputProcesser
+processLines(String): Vector<String>
+processIgnoreWords(String): Vector<LintInterface>
+tokenizeLine(Vector<LineInterface>): Vector<LineInterface>
LineShifter
+shiftLine(Vector<LineInterface>): Vector<LineInterface>
LineIgnorer
+ignoreLine(Vector<LineInterface>, Vector<String>): Vector<LineInterface>
LineSorter
+sortLines(Vector<LineInterface>): Vector<LineInterface>
implements
Line
-originalString: String
-tokenizedString: String[]
-size: int
-indexes: Vector<Vector<Integer>>
Operations

Figure 1: Class Diagram of KWIK

The master control, which is the KWIK program itself, will call its different methods stage by stage. Each KWIK method will then call the methods of the respective classes to process the data. Once done processing, each of these methods, which made up 8 modules in total, will then return the processed data to the master control. The master control thus has full access to the data at any time and the different modules will be invoked to process the data from input to output in sequential order. This is illustrated in Figure 2 below.

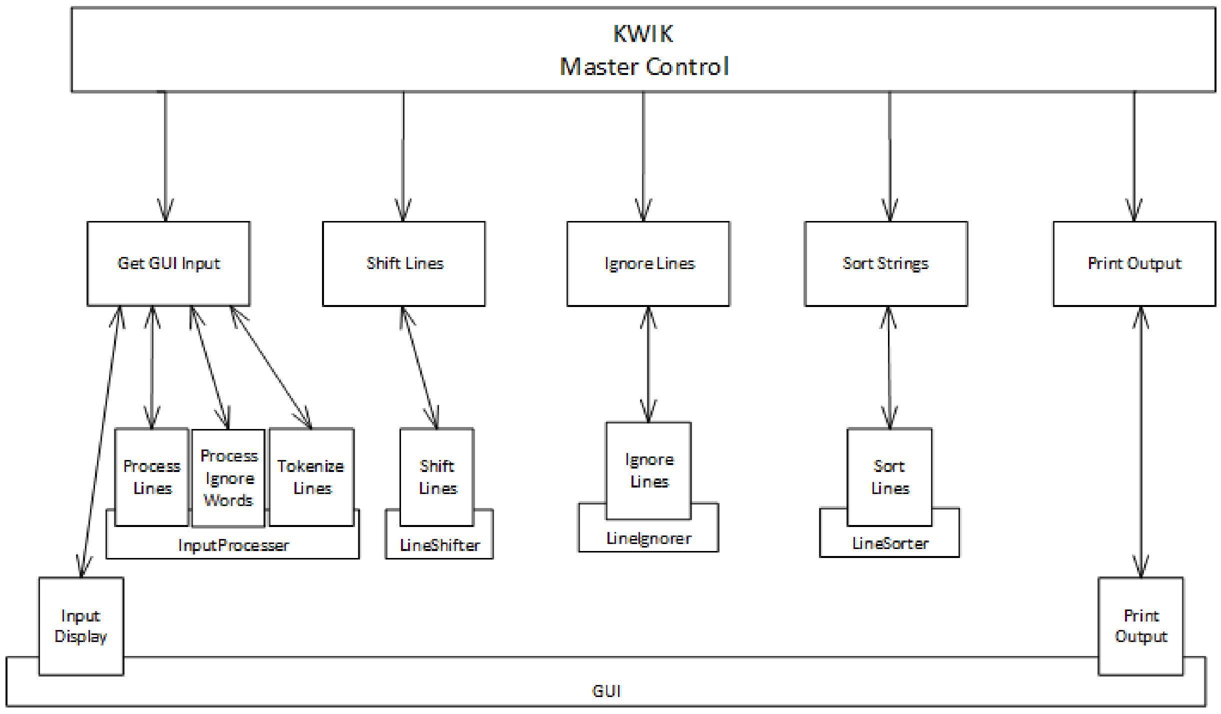


Figure 2: Method calls of KWIK

Limitation & Benefits

KWIC comprises of multiple modules which process the data and return it to the main control module. This gives KWIC several benefits.

Firstly, it supports the reuse of modules. This is because each of the modules is capable of performing in isolation provided that it is given input in the form it expects, according to its signature.

Next, KWIC is highly extensible. Since the data is always returned back to the main control module, intuitive flow of processing is maintained as the same data is processed by different modules at different stage of the process. This is similar to a pipeline solution, meaning that KWIC can easily be extended to fulfill new requirements by inserting new modules to process the data in the main control module.

Data representation can also be changed easily without affecting the rest of the program since each module is designed to accept an interface rather than a concrete class. Future implementation of the interfaces by different classes can thus be added without adverse effects to the other parts of the program.

Finally, the different module being loosely coupled means that each module is able to be tested individually and independently. Testing of KWIC can hence be done more easily, leading to shorter debugging time.

However, KWIC's design is not without limitation. KWIC is inefficient in terms of its use of space. In order to better support extension, the data representation of a single line will need to contain at least the original line and the tokenized line, effectively doubling the required space. This in turn translate into longer processing time as modules which are required to process through all the data will need longer time for more data.