Assignment 3: KWIC Extensions

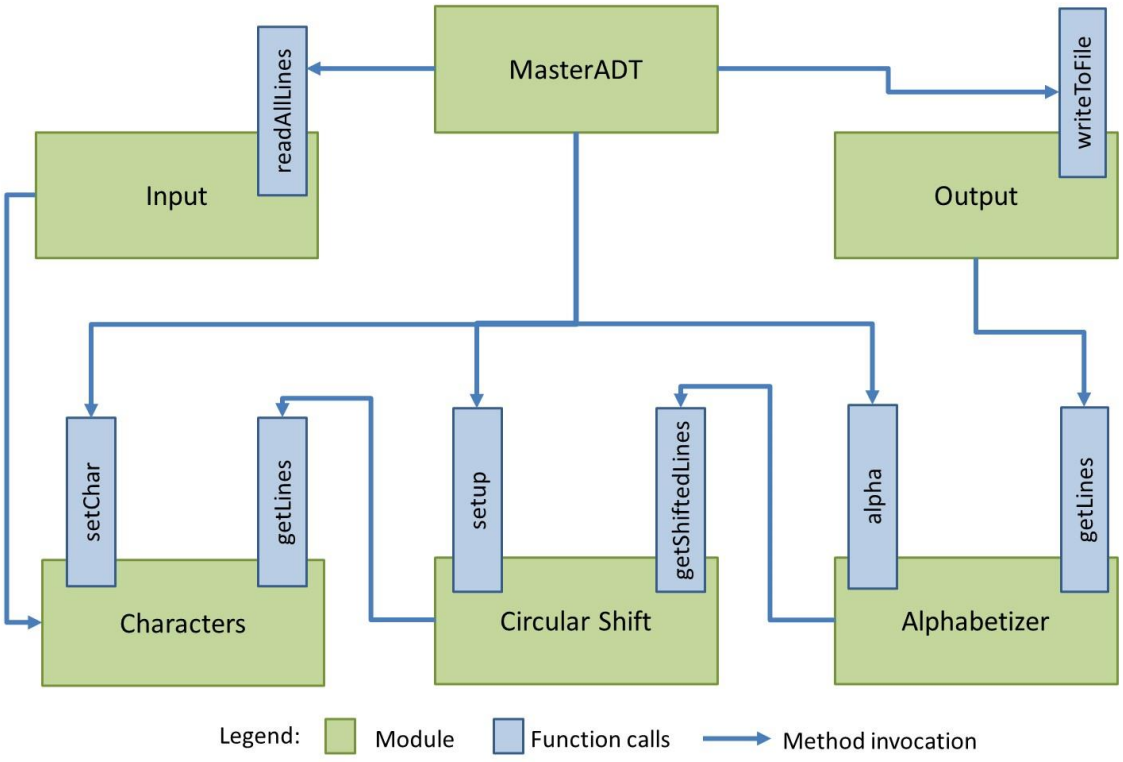
Code Repository URL:	https://github.com/TIC3001/tic3001-ay2122s2-15		
Team number:	15		
Name:	Wang Wei	Name:	Chan Khai Wen
Student Number:	A0211395J	Student Number:	A0211493L

1. Introduction

In this assignment, we integrated a search facility with previous implementation. The search facility is able to search through different files that contain the keywords, the output will display which files are containing the keyword and KWIC is being applied to each file. User is able to search different keywords throughout the period until the program is terminated by the user's command. We split the job by Wang Wei handles the majority of coding while Khai Wen assists on unit testing and refactors existing codes for easier extendibility and better user interface. We both spent time on report and presentation too.

2. Architectural Design

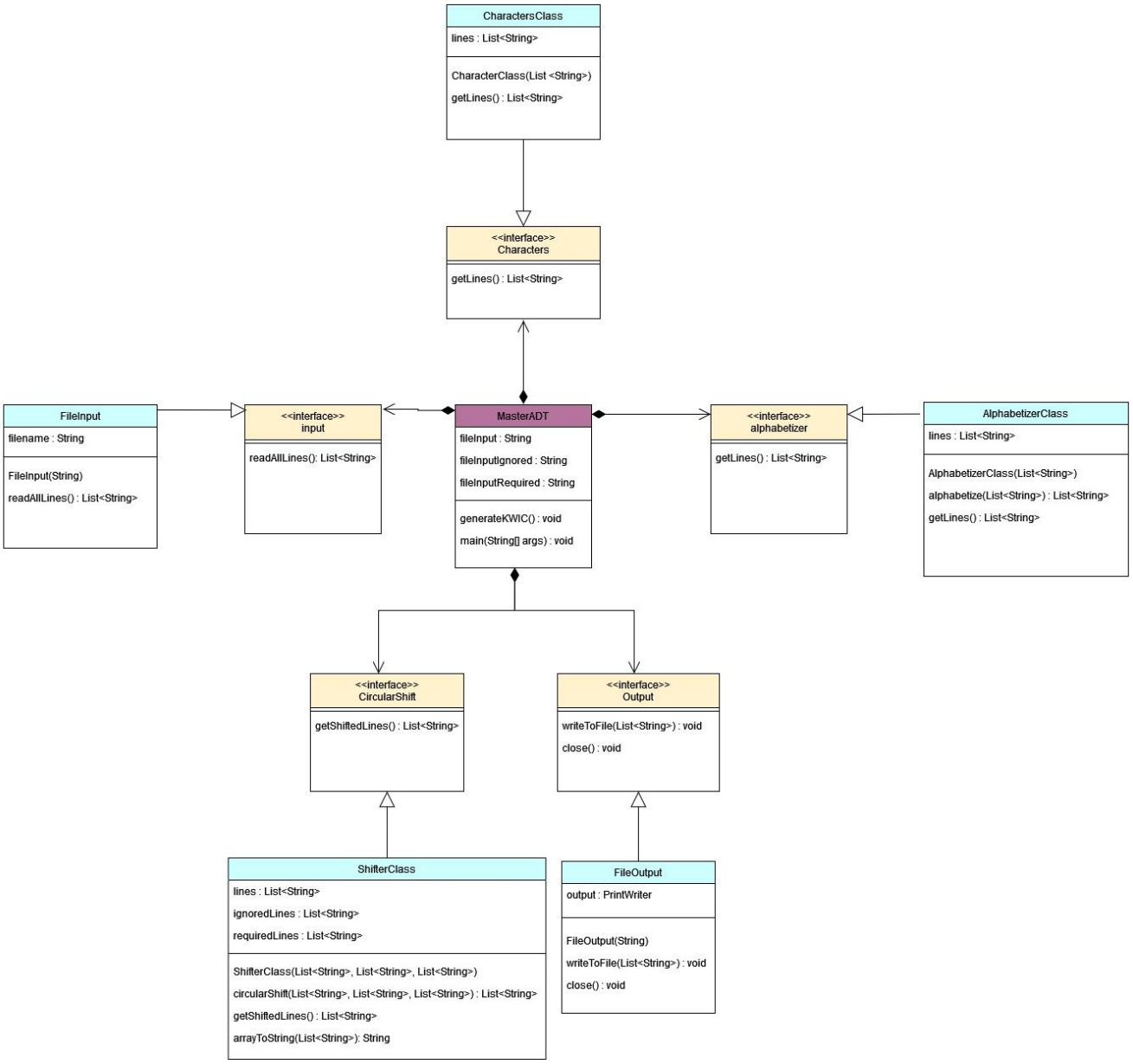
Component Diagram:



The architecture we choose to implement is Abstract Data Type (ADT), this architecture consists of 5 main modules which are Input, Characters, Circular Shift, Alphabetizer and Output. Each module is controlled by MasterADT and they interact with each other through function calls. They do not know how data is being stored/represented in other modules. The algorithm of new requirements can be

changed easily inside each module without affecting other modules. As our assignments require extendibility and reuse, the modules in this architecture are able to implement the interfaces and allow developers to create new classes and extend the functionality. Besides, the ADT also follows a few basic design principles (read more in section 3), hence we choose to implement this architecture.

Class Diagram:



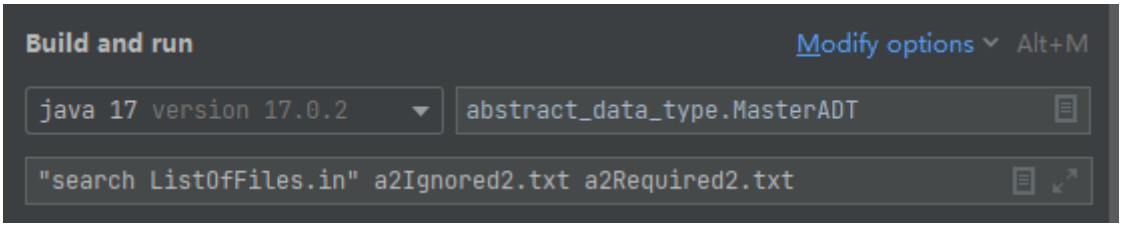
Below are the important modules in ADT:

- MasterADT acts as a main control that controls the flow of all modules.
- Input reads data lines from the input source and stores them in Characters.
- Characters acts as a container to store relevant data for use in other modules.
- CircularShift generates a list of indexes, which indicates the position of the first character of each circular shift.
- Alphabetizer makes use of the data generated from CircularShift and outputs a KWIC index.
- Output uses the list generated from Alphabetizer and write into the output source

3. Applying Design Principles and Software Patterns

We choose to implement Abstract Data type design as it has all the basic design principles such as abstraction, information hiding, Separation of Concerns (SoC), high cohesion, low coupling and Single Responsibility Principle. The operations in ADT are highly encapsulated in abstract data type/object. Each object only provides interfaces for other objects to access/communicate data (information hiding). The separation of concerns (SoC) is also implemented whereby each module addresses different types of concerns. For example, there are 5 different interfaces that handle different types of processing. The SoC also allows us to extend the functions easily in ShifterClass without affecting other modules. We just need to change the algorithms in ShifterClass to handle the extensions.

We have implemented patterns such as Abstraction occurrence, Facade, MVC. In MVC, users will use KWIC as the controller and manipulate the data by entering the Search keywords in the arguments and words to ignore or require. Once it is done the Output file will be generated for the user to view. In Facade, it provides a user interface to input what is the file required, what is the keyword to be searched. And it tells the user when the execution has been done. In Abstraction occurrence, our design reuse the same codes to manipulate multiple files with the same type of attributes.



In Assignment 1, the requirement is straightforward. And no restriction for the input and output method. We have followed the same approach in Assignment 2. Where ask users to input which file they would like to take in to perform the KWIC functions and what will be the Output file name.

In Assignment 3 we have corrected this portion where to use CLI arguments to run the files. And in Assignment 3 there is an implementation for users to key in the word they would like to search. The Output file name has been fixed with a system generated file name.

4. Search using KWIC

The functions for Assignment 2 still present in our extended KWIC. Where keyword “search” is present in the argument, the program will execute extended KWIC instead of Assignment 2. And if the argument is absent, the program will run the Assignment 2 function instead and still ask the user to enter the word they would like to be searched for due to what has been implemented in Assignment 3.

The program will break what inside ListOfFiles into multiple parts. One TXT file at a time. The first TXT file will be split the content of files by new lines. And applied KWIC to search on each of the files with the search input provided by the user.

Once the program is running, the user may input multiple keywords to search in the same file. Once current search is completed, the user is allowed to enter a new keyword to search. The search for this current file will only end when the user input “q” as the keyword.

And it will run the second TXT file for searching, users will perform the same actions as above. Once "q" has been entered, the whole program will be ended with 2 Output files with the TXT files' name to be created for the user to view the result. With the TXT files name as part of the file name, the users will be able to know which is for which.

If the file contains multiple paragraphs, it will follow how Assignment 1 handles it. And filtered out based on what the user has input for the search.

The architecture Abstract Data Type is one of the fastest solutions. The design (SoC) helped to separate the program to multiple classes and each class only handled their own tasks (Single Responsibility Principle). Where also allowed some part of the programs able to run multiple items at the same time. This will reduce the time when the user is performing search functions.

5. Testing

We are using unit testing (JUnit) and System testing as the testing method.

By designing the JUnit test cases and the test results, we are able to tell if the function we have implemented is able to run correctly based on the test results.

For the System testing, we have created several test files for the system to run and check against the correct answers we have prepared to check if these 2 are tally.