**Assignment 1: KWIC-KWAC-KWOC**

|  |
| --- |
| Code Repository URL: https://github.com/HyungJon/KWIC |

|  |  |  |
| --- | --- | --- |
| Name | Kim Hyung Jon |  |
| Matric number | A0115864B |  |

1. **Introduction**

This software is a simple program that generates a set of circularly shifted strings from input titles and words that are not keywords. It aims to meet the three non-functional requirements: user-friendliness, speed and extendibility.

Some more non-functional requirements set by our group were that the software should immediately display the results after the process is completed so that the user does not have to find and open it, and that

**2. Design**

The program runs by executing the UI class, a component common for both solutions. Upon execution, UI displays a simple GUI containing 4 main parts: two text fields for specifying input and output file paths, and two buttons for selecting the solution. The UI component was implemented by Hyung Jon.

Both methods require that the input file contains the data in the following format: the first line contains all words to ignore, and each of the following lines contains one title. An example is:

is the of and as a after

The Day after Tomorrow

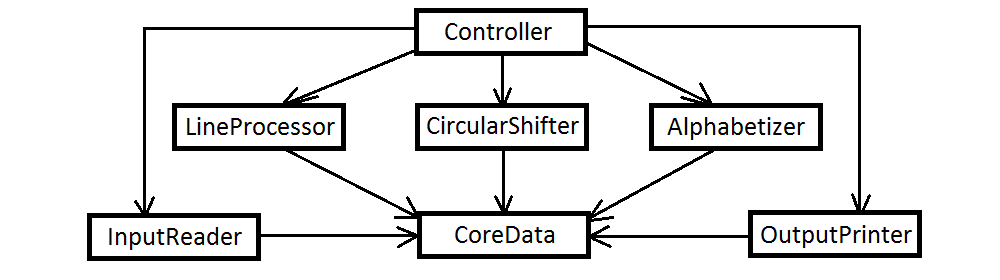
Fast and Furious

Man of Steel

By clicking either button, the user can choose one of the following solutions to generate circular shifts. Solution 1 was implemented by Jerrold, while Solution 2 was implemented by Hyung Jon. Some methods that can be used in common for both approaches were implemented by Hyung Jon as well.

**2.1 Solution 1: Shared Data**

Solution 1 by Jerrold



**Controller**

**CoreData**

**InputReader**

**LineProcessor**

**CircularShifter**

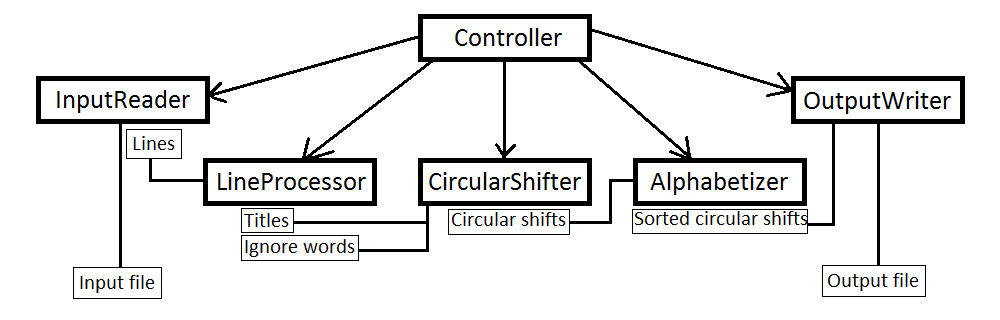
**Alphabetizer**

**OutputWriter**

**2.1.1 Benefits**

**2.1.2 Drawbacks**

**2.2 Solution 2: Abstract Data Types**

****

Solution 2 implemented by Hyung Jon is a simplified version of the Solution 2: Abstract Data approach in Reading 2. It consists of the following components:

**Controller**

Master control in charge of invoking other components in the correct order with correct parameters.

**InputReader**

Reads the input file and stores the data line by line, without knowledge regarding what each line is.

**LineProcessor**

Takes all lines stored in InputReader and converts them to a form usable by CircularShifter by constructing two arraylists of strings: for titles and words to ignore.

Rather than storing all words in all lines as 2D arraylist, LineProcessor contains the method getTitle() that converts a title at given index to an arraylist of words, invoked by CircularShifter only when constructing circular shifts of that title.

**CircularShifter**

Constructs and stores all circular shifts, not starting with words to ignore and with first word in upper case, from each title obtained from LineProcessor.

**Alphabetizer**

Sorts and stores the circular shifts in the alphabetical order.

**OutputWriter**

Writes all sorted circular shifts to the output file specified by user and opens the output file.

**2.2.1 Benefits**

**Comprehensibility**

The software is highly modularized, each component in charge of a specific purpose. Therefore each component requires only the knowledge relevant to that purpose. For example, LineProcessor is the only module that knows where to find words to ignore in the input. Therefore it is much easier to understand the behavior of individual modules compared to the Shared Data approach where the system is only comprehensible as a whole.

**Modifiability**

Because the system is modularized, modifying the system requires modifying only the relevant components. For example, changing the circular shifts generation to splitting the title into two substrings at the index of the keyword and swapping the two will affect almost all components in Shared Data solution. Conversely, in Abstract Data solution, only LineProcessor and CircularShifter need to be modified.

**Extendibility**

Data is stored separately for each stage of the process, and each stage offers separate methods to access the data produced at that stage. This makes it easier to add new components or functions that change the data in the middle, such as a function to edit the list of words to ignore.

**2.2.2 Drawbacks**

**Extendibility**

The modularization makes it difficult make major modifications involving more than one module, such as extending the software by adding another module to the process. This means to add a component X between CircularShifter and Alphabetizer, both components must be modified so that CircularShifter converts the data into a form usable by X and Alphabetizer accepts data processed by X.

**Memory space**

Unlike the core data approach where all components work on a shared repository, this approach requires each component to store the output data separately, thus requiring more memory space.