**CS3219 KWIC**

|  |  |
| --- | --- |
| STUDENT NUMBER 1: | STUDENT NAME 1: Muhammad Nur Kamal Bin Mohammed Ariff |
| STUDENT NUMBER 2: | STUDENT NAME 2: Amrut Prabhu |

**Introduction**

This report is about the Key-Word-In-Context (KWIC) implementation we had to build in CS3219’s tutorial 2, question 6. We were presented with 4 choices of architectural solutions (Garlan, David, & Shaw, 1994). These solutions were: “Shared Data”, “Abstract Data Type”, “Implicit Invocation” and “Pipes and Filters”. Due to the time constraint and our limited knowledge on multi-threading, we chose to implement the “Shared Data” and “Abstract Data Type” (ADT) architectural solutions.

**Usage**

Set-Up:

1. Extract the kwic.java into an empty folder called kwic
2. Open the src folder inside the kwic folder
3. Run your command prompt from this folder

Running the program:

1. In your command prompt, type: java com/company/Main <Implementation Option> <Path of input text file> <words to ignore>
   * <Implementation Option> is required, and can be 1 or 2 (Implementation doesn’t affect input or output)
     1. 1 is for Shared Data Implementation
     2. 2 is for ADT Implementation
   * <Path of input text file> is required
   * <words to ignore> is optional
   * Example:
     1. java com/company/Main 2 ../sampleInput/2.txt is the of and as a after

Output:

Day after Tomorrow the

Fast and Furious

Furious Fast and

Man of Steel

Steel Man of

Tomorrow the Day after

**Shared Data Architectural Solution**

SharedDataMain

input

circularShift

alphabetize

output

Output medium

Input medium

Line

CircularShift

Subprogram Call

Direct Memory

Access

System I/O

Figure 1: Architectural Diagram of Shared Data solution

In the Shared Data Architectural Solution, data is read and stored as a global variable in the SharedStorageMain class. This class is responsible for reading input, circular shifting, alphabetizing and showing output. The helper classes Line and CircularShift, helps abstract the data to be easier to work with. Line represents a line of input and holds a List of Strings, where each String is a word. CircularShift represents a circular shifted line. It composes of a reference to the Line it represents and an integer Index representing the starting word of the circular shifted line. This minimizes space consumption since each circular shifted line is only express as a reference and an integer instead of a whole new String.

The solution is a “quick and dirty” approach that can be useful for time constrained situations, for example, in a coding challenge. However, due to low modularity, it is not a recommended approach if it is meant to be used in the long term.

**ADT Architectural Solution**

Input

MasterControl

Sorter

Shifter

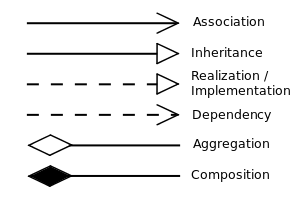
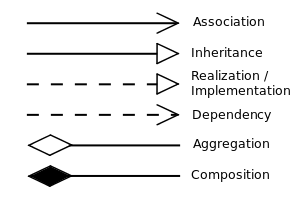
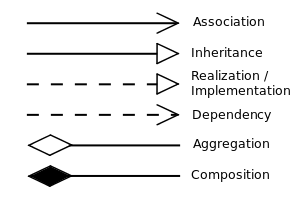
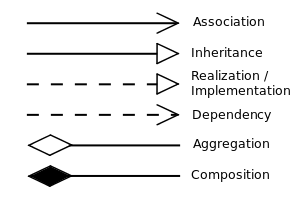
Output

KwicAlphabetizer

KwicCircularShifter

KwicInput

KwicOutput



Input medium

Output medium

Line

CircularShift

Subprogram Call

System I/O

Figure 2: Architectural Diagram of ADT solution

ADT Architectural Solution uses the same logic to process input to output. However, this time, there are 4 main Interfaces separating the 4 steps to implement KWIC and 4 corresponding Classes that implement each interface. As seen by the Figure 2, each of these classes (except Input) composes of the class representing the previous step. For example, KwicAlphabetizer (which sorts lines) holds a reference to a Shifter, since before sorting, we need to circular shift lines. This design decision allows the data to be passed throughout the 4 steps without having a global data structure.

This more modular approach allows us to easily swap any of the 4 Main classes with a class that implements the corresponding Interface, if the requirements change.