## Assignment 1: KWIC-KWAC-KWOC

Code Repository URL: https://github.com/java-wei/KWIC

|  |  |  |
| --- | --- | --- |
| Name | Cheng Yingjie | Zhang Chenwei |
| Matriculation Number | A0105709H | A0105753J |

## Introduction

KWIC (Key Word In Context) index system provides a search mechanism for information in a long list of lines, such as movie titles. For each input title, it shall be “circularly shifted” exhaustively by removing the first word and appending it at the end of the line to create a set of circularly shifted lines unless the first word belong to the words to ignore. The system will output on the screen a listing of the circularly shifted lines for all input titles in ascending alphabetical order.

## Design

**Architecture 1 – Implicit Invocation**

Output medium

Input Processor

Output Displayer

Alphabetizer

Lines

Circular shifter

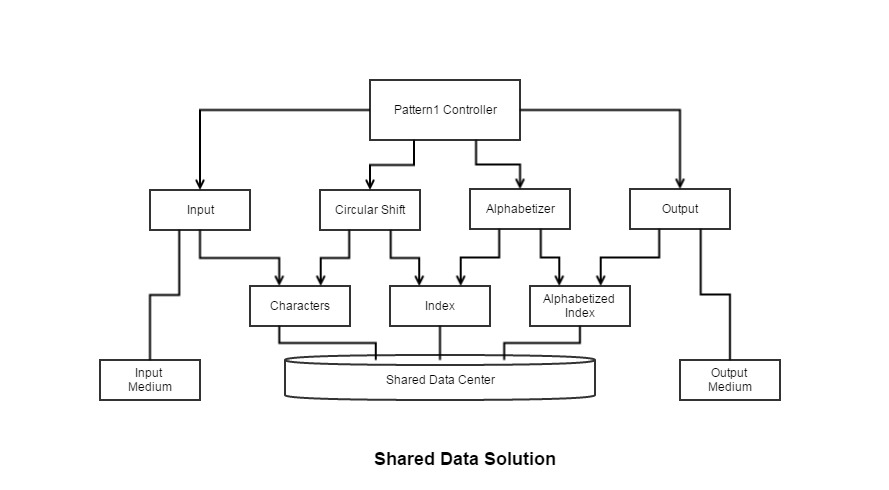
Titles

Input medium

Controller

This design follows the Implicit Invocation architecture design. There is a master controller that controls the process of input processor and output displayer which directly interact with the input and output medium. When there is a change to the titles, it will implicitly invoke the circular shift module and alphabetize module to produce sorted shifted lines.

**Architecture 2 - Shared Data Solution**



This design follows shared data solution architecture design. Data is communicated between the components through shared storage. Controller is used to control the work flow: Process Input – Circular Shift – Alphabetizer – Display output. Title list, word-to-ignore list and index were stored in data center. Circular shifted index will be stored in result first, and then alphabetizer will sort and update the index producing the output.

## Benefits of Design

## Architecture 1 – Implicit Invocation

There are several advantages in this design. This design is easily extendable. Data are abstracted and modules are encapsulated. They can be changed without affecting others. And additional modules can be attached to the system by registering them to be invoked on data-changing events. Hence, this design easily supports functional enhancement. Moreover, module reuse is also supported since the implicitly invoked modules only rely on the existence of certain externally triggered events.

## Architecture 2 – Shared Data Solution

Using this solution data can be represented efficiently, since computations can share the same storage. It has a certain intuitive appeal as well, since distinct computational aspects are isolated in different modules. And it is relatively easy to implement with high coherence. Each module can use the data freely and can be called freely.

## Limits of Design

## Architecture 1 - Implicit Invocation

This design requires a relatively large of storage space since all the invocations are data-driven. And it becomes difficult to control the order of processing of the implicitly invoked modules. Moreover, the natural flow of processing the tasks is not very intuitive.

## Architecture 2 - Shared Data Solution

This design has high coupling for the controller and the shared data. A change in data storage format will affect almost all of the modules. Besides that, changes in the overall processing algorithm and enhancements to system function are not easily accommodated. Finally, this decom-position is not particularly supportive of reuse.