**SE4433/CMSC5433 Software Architecture and Design**

**KWIC Software Architecture for a Web-based Search Engine**

**Assignment 1 by Jacob Theobald and Sharon Too**

**Requirement Specification**

**Functional:** The KWIC\* (Key Word In Context) index system shall accept an ordered set of lines, where each line is an ordered set of words, and each word is an ordered set of characters. Any line should be *circularly shifted* by repeatedly removing the first word and appending it at the end of the line.

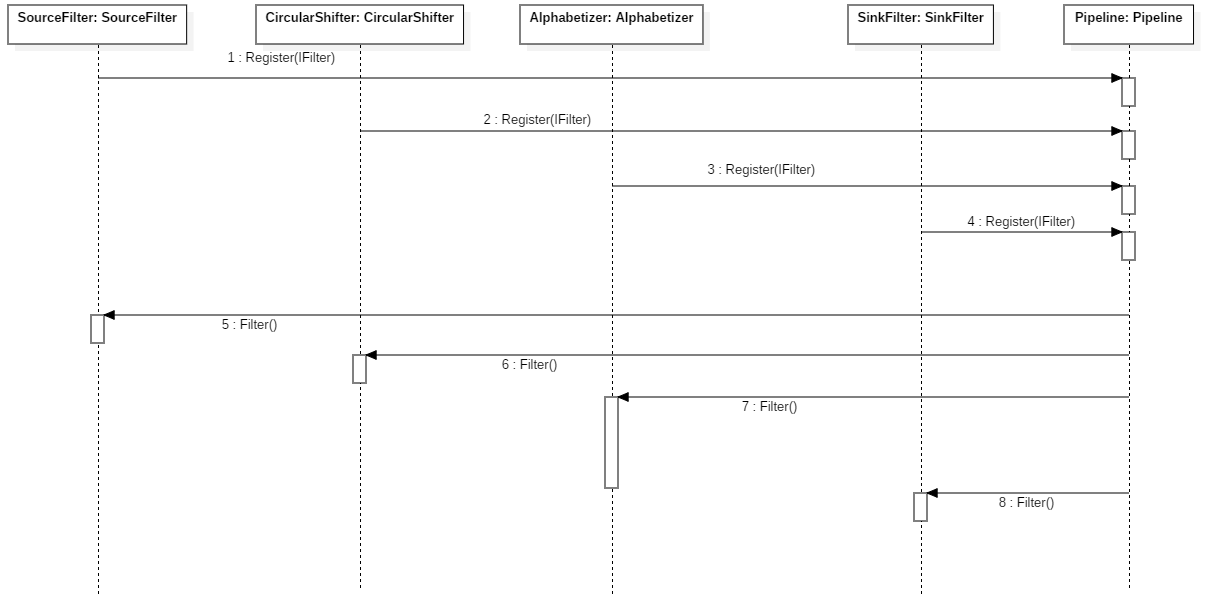
The KWIC\* index system shall output a list of all circular shifts of all lines in ascending alphabetical order, where “a<A<b<B<…<y<Y<z<Z”. The input of the system will be characters only, including uppercase and lowercase.

The system shall be accessible by web browser. A user should be able to type a list of zero or more lines into a text box, submit the list, and receive back the KWIC indexed results in another text box.

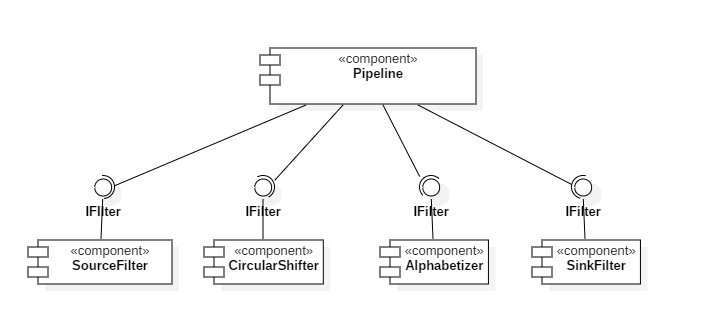
**Non-functional:** The KWIC\* system shall be easily understandable, portable, scalable, and reusable with good performance. The KWIC\* system must also be user-friendly, responsive, and adaptive.

New types of modifications to the input lines should be able to be added easily and reordered if necessary. The resulting KWIC indexed lines should be returned to a user with broadband internet speed in under 2 seconds.

The data should be able to enter the Pipe and Filter system as any type, and be passed through the system as a stream to allow flexibility in input and output types.



**Architecture Specification**

****

The architecture style used for the KWIC indexing is a Pipe and Filter system, where a Pipeline and Filter make up the main two components. Input is taken from a web call into a source filter, piped through a circular shifter filter and alphabetizer filter, and then collected in a sink filter to be returned to the user.

Each filter is a realization of the IFilter class, which contains the methods SetInput, SetOutput, and Filter. The Pipeline class contains the methods Register and Run. After instantiating the Pipeline class, any number and type of IFilter can be registered to it. The Run method then calls each filter to process the input data.

The connectors between filters are not a discrete class or type, but are MemoryStream objects. MemoryStreams are set between two adjacent filters by the Pipeline Register function, and each stream is written to and read from when filters pass data forward.

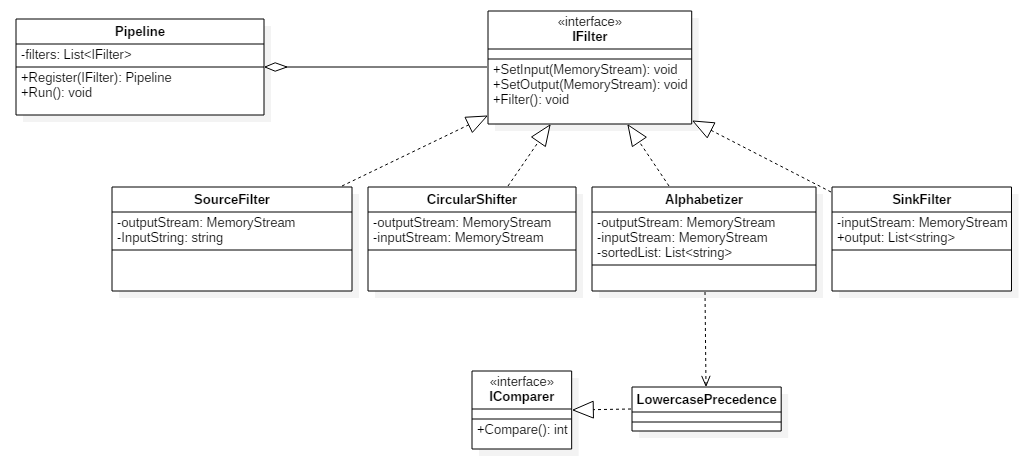
There are a few constraints imposed by this architecture. One constraint is that the pipeline is entirely linear and unidirectional, that is, once the data is processed there is no way for a user to modify that same data. Additionally, the pipe connections must be a MemoryStream type. This means that each filter will need to parse and type the data, perform its operation, then convert that type back to a stream type. The Pipeline itself will only accept objects that realize an IFilter interface. No other filter types will be accepted.

There are several advantages and disadvantages for this style of architecture. The key advantage of Pipes and Filters is its incremental nature and modularity. Because each filter is self-contained manipulation on a data set, each filter component can be easily understood just by looking at it. Additionally, because the filters are completely discrete and have no knowledge of each other, new filters can be added and removed or filters can be reordered without the need for large code changes.

However, the Pipes and Filters architecture also has several disadvantages. Because each filter is discrete, the overall system has a low fault tolerance. If one filter fails, there is no easy way for the next filter to properly handle that failure such that it won’t disrupt the entire process. Additionally, because data often needs to be parsed and unparsed in each filter, performance can degrade as a system becomes more complex. Further, the filters themselves may become increasingly difficult to write.

**Design Specification**

The following class diagram details the structure of this design.

****

**User Manual**

The system as it exists is exceptionally easy to use. The user is presented with a webpage with only three interactive pieces: an input text box, an output text box, and a process button.

To use the system, input zero or more lines of text into the input box and click “Process.” The KWIC indexed output will be printed to the output box below.

A user can modify the input at any time and process as many times as they wish.

One caveat is that this system is currently not hosted online. It is only able to be accessed through the local host when the build is running.