KWIC Implemented with Pipe and Filter Architectural Style (Assignment 4)

1. Introduction

1.1. Definition of the Problem

The Key Word In Context (KWIC) problem is defined as follows:

The KWIC index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be circularly shifted by repeatedly removing the first word and appending it at the end

of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

1.2. Example

Consider the following set of lines:

- Star Wars
- The Empire Strikes Back
- The Return of the Jedi

The KWIC index system produces the following output (comparison is case-sensitive, that is capital letters are greater than small letters):

- Back The Empire Strikes
- Empire Strikes Back The
- Jedi The Return of the
- Return of the Jedi The
- Star Wars
- Strikes Back The Empire
- The Empire Strikes Back
- The Return of the Jedi
- Wars Star
- of the Jedi The Return
- the Jedi The Return of

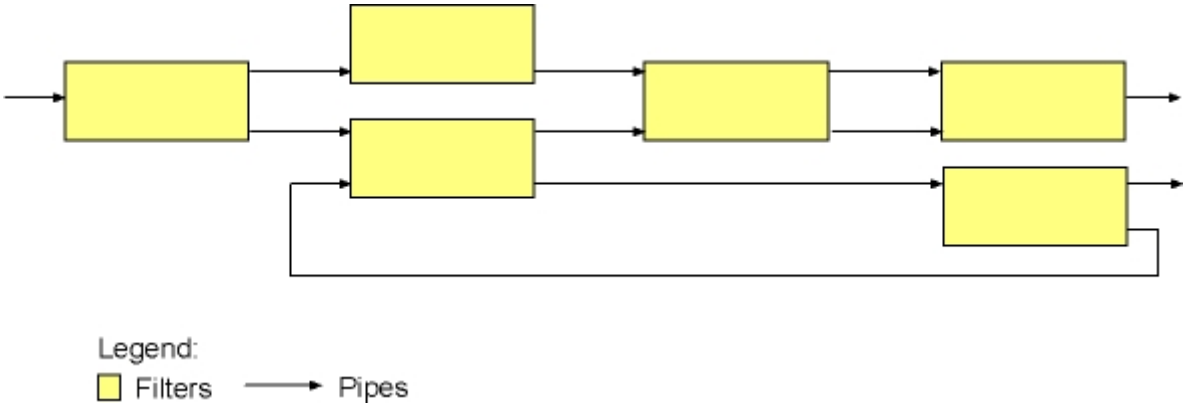
1.3. Assignment 4

To accomplish the Assignment 4, you need first to get acquainted with the **existing KWIC system**. After that you need to extend the functionality of the existing system to meet the **following requirements**. Finally, you need to answer to these three **questions**.

2. Pipe and Filter Systems in General

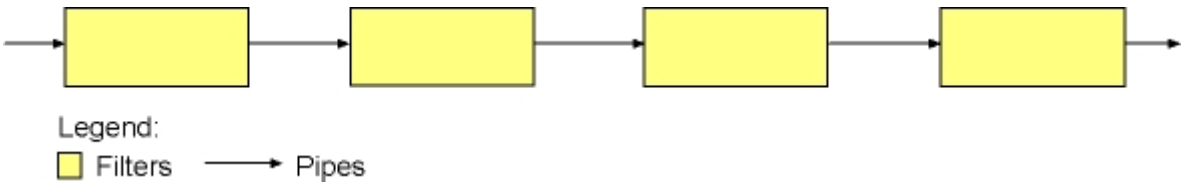
In a pipe and filter style each component has a set of input streams and a set of output streams. A component reads streams of data on its input streams, processes

the data and writes the resulting data on its output streams. Hence components are termed *filters*. The connectors of this style merge the streams together, i.e., they transmit outputs of one filter to inputs of another filter. Hence the connectors are termed *pipes*.



Among the important invariants of the style is the condition that filters must be independent entities: in particular, they should not share state with other filters. Another important invariant is that filters do not know the identity of their upstream and downstream filters. Thus, filters may not identify the components on the ends of their pipes.

Common specializations of this style include *pipelines* (see figure below), which restrict the topologies to linear sequences of filters; bounded pipes, which restrict the amount of data that can reside on a pipe; and typed pipes, which require that the data passed between two filters have a well-defined type.

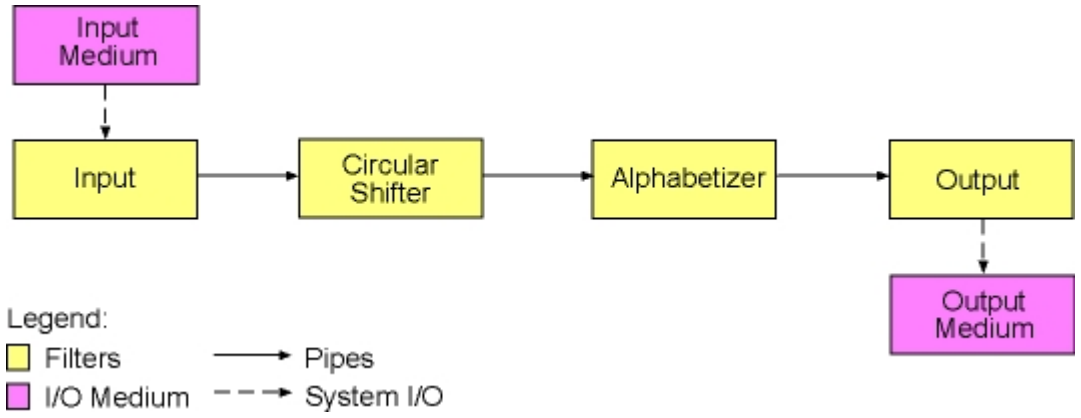


The best-known examples of pipe and filter architectures are programs written in the UNIX shell. UNIX supports this style by providing a notation for connecting components (represented as UNIX processes) and by providing run-time mechanisms for implementing pipes.

3. The Existing System

Pipe and Filter KWIC system is composed of the following components:

- Input filter, which reads the content of a KWIC input file, parses it, and writes the parsed lines to its output stream.
- CircularShifter filter, which has its input stream connected with a pipe to the output stream of Input filter. Thus, the lines produced by Input filter serve as input for CircularShifter filter. CircularShifter processes the input lines producing circular shifts of those lines. The produced shifts are written to the CircularShifter's output stream.
- Alphabetizer filter, which has its input stream connected with a pipe to the output stream of CircularShifter filter. In that way circular shifts produced by CircularShifter serve as input for Alphabetizer. Alphabetizer sorts these shifts alphabetically and writes them to its output stream.
- Output filter, whose input stream is connected with a pipe to the output stream of Alphabetizer. Thus, Output reads sorted shifts from its input stream and writes them to the standard output.
- Master Control, which manages filter and pipe mechanism, by creating a pipeline of the above described filters.



3.1. Properties of the Existing System

The existing KWIC system may be described by:

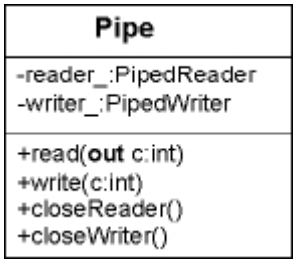
- **Pipes and filters mechanism**
- **Current pipeline**

3.2. Pipes and Filters Mechanism

3.2.1. Pipes in KWIC System

Pipes in the current KWIC system are represented as instances of Pipe class. An instance of Pipe class is a composition of two streams: an input and an output stream. Pipe class connects these two streams in the following way. The data that are written to the input stream are transmitted to the output stream. In this way these data become available for reading from the output stream.

In the current implementation Pipe class encapsulates the input and output streams as its private instance variables and provide just a simple public interface for writing and reading data to a pipe object. Thus, clients simply write some data to a pipe object by calling its write method. This data become then available for other clients, which call read method of the pipe object to retrieve the data.
Here is the class diagram of Pipe class:



There are few important properties of pipe objects:

- Pipe objects limit the amount of data they can hold. However, whenever clients read the data from a pipe, this data is considered "consumed" and the space that was occupied by the retrieved data is made free again.
- Pipe objects are synchronized. This has two major consequences. Firstly, there can be only one thread working with a pipe object at a specific moment, i.e, there can be only one thread currently writing or reading the data from a pipe object. Secondly, threads reading from an empty pipe or writing to a full pipe are blocked as long as there are not some data to read from the pipe, or there is no free space in the pipe to write the data. Thus, a pipe object may be seen as a typical shared synchronized data structure for producer/consumer multi-thread scenarios. Thus, a pipe object is always shared between a producer and a consumer thread, where the producer

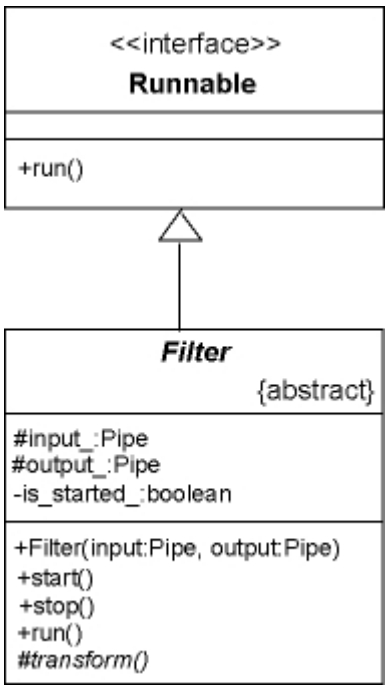
thread writes some data to the pipe, and the consumer thread "consumes" that data from the pipe.

3.2.2. Filters in KWIC System

Filters in the current KWIC system are represented by an abstract class named Filter. An instance of Filter class is composed of:

- Input pipe object, from which the filter object reads the data
- Output pipe object, to which the filter object writes the processed data
- Thread of control. Each filter object runs in its own thread, thus making it totally independent of any other filters in the system.

  Here is the class diagram of Filter class:



The current abstract Filter class provides the basic filter functionality. Thus, clients start the execution of the filter object by invoking its start method. In turn, this method starts the control thread, which consecutively invokes the filter's transform method.

In the current implementation the transform method is an abstract method, which means that subclasses of Filter class should provide a particular implementation for that specific filter object. For example, CircularShifter filter

implements transform method in the following way. The lines retrieved from the input pipe are processed and circular shifts of those lines are made. Thereafter the produced shifts are written to the output pipe. Speaking more generally, subclasses of Filter class implement their transform methods in the following way: the data is retrieved from the input pipe, transformed (processed) and written out to the output pipe.

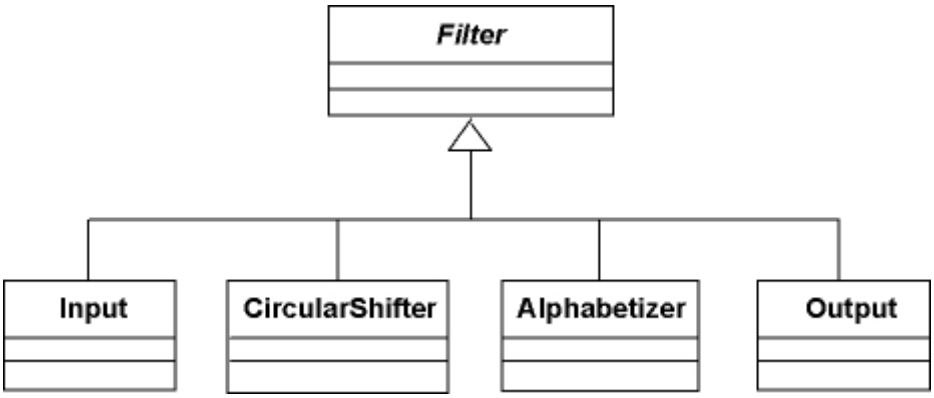
Filter objects may be combined into a typical producer/consumer scenario by simply sharing a pipe object. For example, we assign the output pipe of the first filter to the input pipe of the second filter. In that way whenever the first filter produces some data and writes it to its output pipe, this data is available for the second filter to consume it. In that way we may combine a number of filters into a sequence (pipeline) to achieve the desired functionality of the system.

3.3. Pipeline

The current KWIC system utilizes a pipeline consisting of the following four filters:

- Input filter
- CircularShifter filter
- Alphabetizer filter
- Output filter

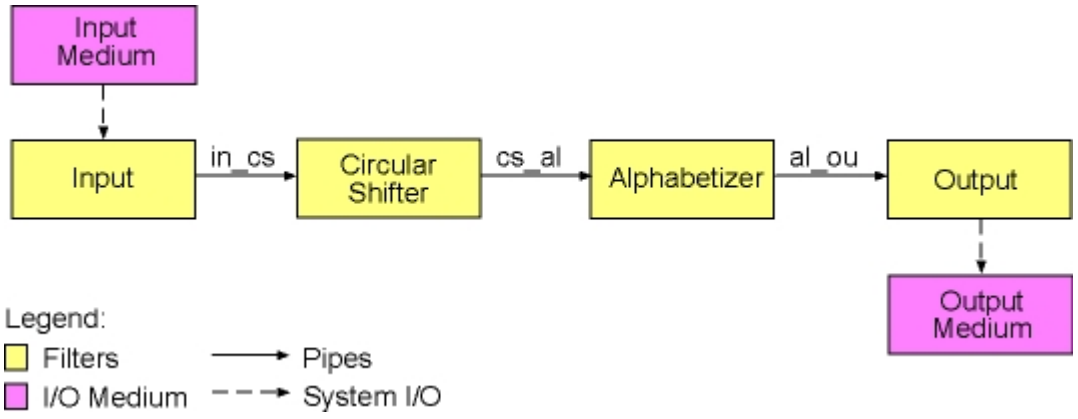
All KWIC filters are of course subclasses of abstract Filter class. These relationships are depicted on the following class diagram:



The four KWIC filters are connected by means of the following pipes:

- in_cs pipe, which is shared between Input and CircularShifter filter, i.e, this pipe is the output pipe for Input filter and the input pipe for CircularShifter filter.
- cs_al pipe, which is shared between CircularShifter and Alphabetizer filter, i.e, this pipe is the output pipe for CircularShifter filter and the input pipe for Alphabetizer filter.
- cs_ou pipe, which is shared between Alphabetizer and Output filter, i.e, this pipe is the output pipe for Alphabetizer filter and the input pipe for Output filter.

The following figure shows the current pipeline including the pipe names:



4. Your Assignment

4.1. Programming Assignment

You need to modify the existing system in the following way:

- **Implement shift filter mechanism to selectively remove shifts from the data pipeline**
- **Implement line and shift transform mechanism to alter lines and shifts**

4.2. Understanding Check

Finally, you need to provide answers to these three **questions**.

5. Implementing Shift Filter Mechanism

Shift Filter Mechanism removes shifts starting with certain words from the data flow in the KWIC system. The starting words that decide which shifts should be removed from the data flow are denoted as noise words. Noise words should be defined in a special file, which must be passed to the KWIC system as the second command line argument (the first command line argument is the name of the input KWIC file). Thus, to invoke the system from the command line we type:

java kwic.pf.KWIC kwic.txt noise.txt

The format of the noise words file is very simple:

- A single noise word is stored on a single line in the file.

Thus, the system should parse the noise words file, keep all noise words in the memory, compare the first word of each single shift with all of noise words, and finally remove all shifts that start with a noise word.

5.1. Implementation Hint

You should implement a new Filter subclass, say ShiftFilter. ShiftFilter may keep all noise words as its instance variables. In the transform method of ShiftFilter class, you may want to remove all shifts starting with a noise word from the data flow. Finally, you need to insert an instance of ShiftFilter at the proper place in the current pipeline.

6. Implementing Line and Shift Transform Mechanism

Generally, Line Transform Mechanism alters the content of a single line in the following way. The first word of the line is capitalized. i.e., all characters of the first word are converted to upper case. Here is an example:

Star Wars

After transforming it:

STAR Wars

You need to apply this mechanism for both all original lines and all circular shifts as well. Thus, first all original lines are transformed resulting in each original line starting with a capitalized word. Secondly, we make circular shifts with transformed lines. Finally, all circular shifts are transformed resulting in each circular shift starting with a capitalized word. For example, let us look on the following original line:

The Empire Strikes Back

The resultant shifts after all transformations look as follows:

THE Empire Strikes Back
BACK THE Empire Strikes
STRIKES Back THE Empire
EMPIRE Strikes Back THE

6.1. Implementation Hint

Write a new subclass of Filter class. Let us call this new class LineTransformer class. In the transform method of the new class implement the upper case conversion of the first word. To implement the above described functionality create two instances of LineTransformer class and insert them at the proper position in the current pipeline.

7. Questions

Please, answer the following questions:

1. In both modifications of the current system you needed to extend the functionality of the system. Which modules from the original KWIC system did you need to modify to implement the new functionality in both cases? Does this mean that the pipe and filter KWIC system is robust to design changes in

the functionality of the system? Is the same true for any pipe and filter system?

2. Filters in pipe and filter systems are completely independent entities. Thus, filters:
    a. Do not share state (data) with other filters.
    b. Do not know the identity of neighbor filters in the pipeline.
    c. Process their data concurrently and incrementally, i.e., they do not wait for other filters to start or finish their jobs.

    Would it be possible to implement an interactive version of the KWIC system, which will allow users to delete certain lines from the data flow but still not violate any of the above defined filter properties?
3. Normally, filters in pipe and filter systems work as follows. They read data from the input pipe, transform it and write it to the output pipe. Usually, this involves copying of data from the input to the output pipe. If the number of filters in a pipeline of a complex pipe and filter system is large then there is a lot of in-memory data copying. How this may affect the overall system performance?