KWIC Implemented with Main/Subroutine Architectural Style (Assignment 1)

Table of Contents

1.1 Introduction

1.1.1 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.

1.1.2 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

1.1.3 Assignment 1

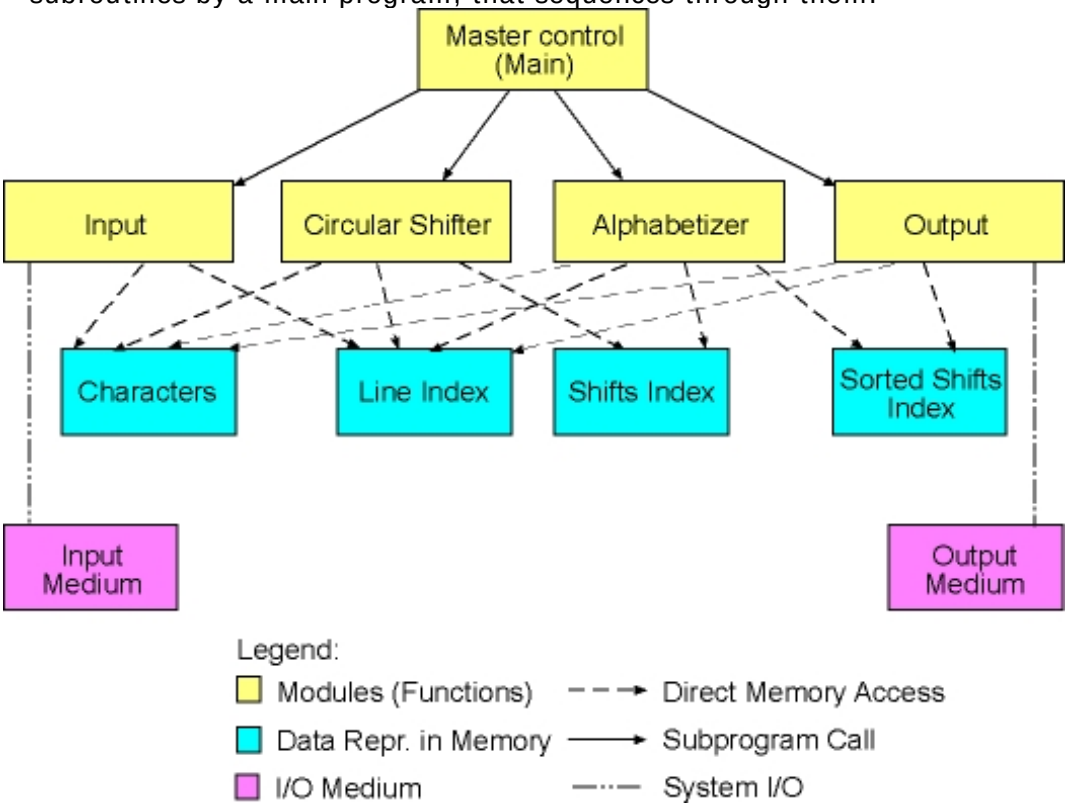
To accomplish the Assignment 1, you need first to get acquainted with the **existing KWIC system**. After that you need to extend the functionality of the existing system to meet the **following requirements**. Finally, you need to answer to these three **questions**.

1.2 The Existing System

1.2.1 Architecture

The solution with main/subroutine architectural style with shared data decomposes the system according to the four basic functiones performed:

input, shift, alphabetize and output. These functiones are coordinated as subroutines by a main program, that sequences through them.



1.2.2 Properties of the Existing KWIC System

The existing KWIC system is characterized through:

- **Data Representation**
- **Processing Algorithm**

1.2.3 Source Code and Documentation

Here is the **source code** in Java for the existing KWIC system, and the automatically generated javadoc **documentation** for the system.

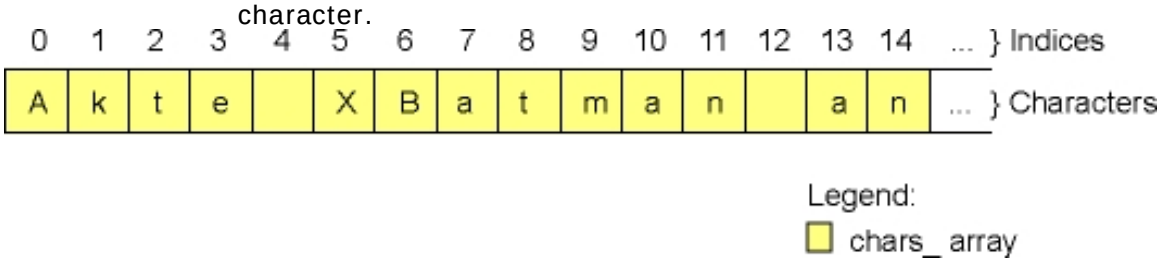
1.2.4 Test Data

Here is a **sample input** file for the KWIC system. Please, save the file on your local drive. Specify the name of the file at the command line as an argument to the KWIC program.

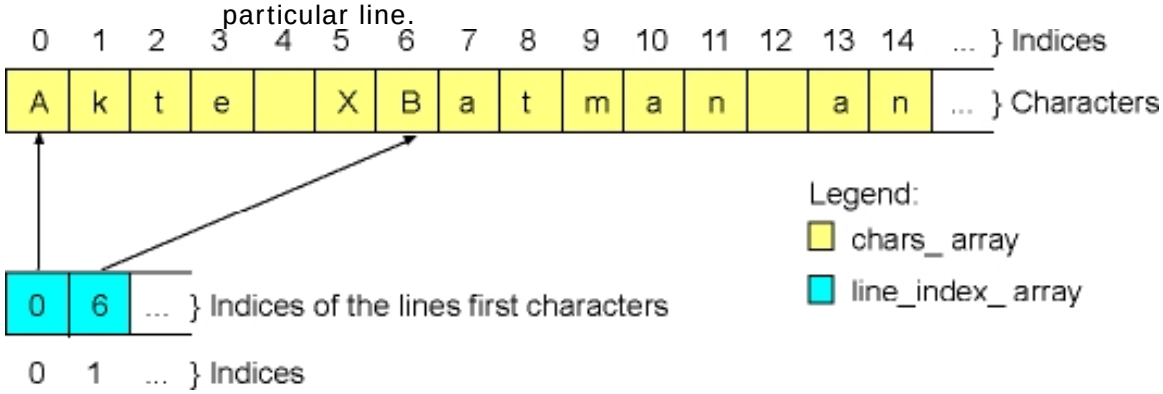
1.3 Data Representation

The current KWIC system represents its data in the following format:

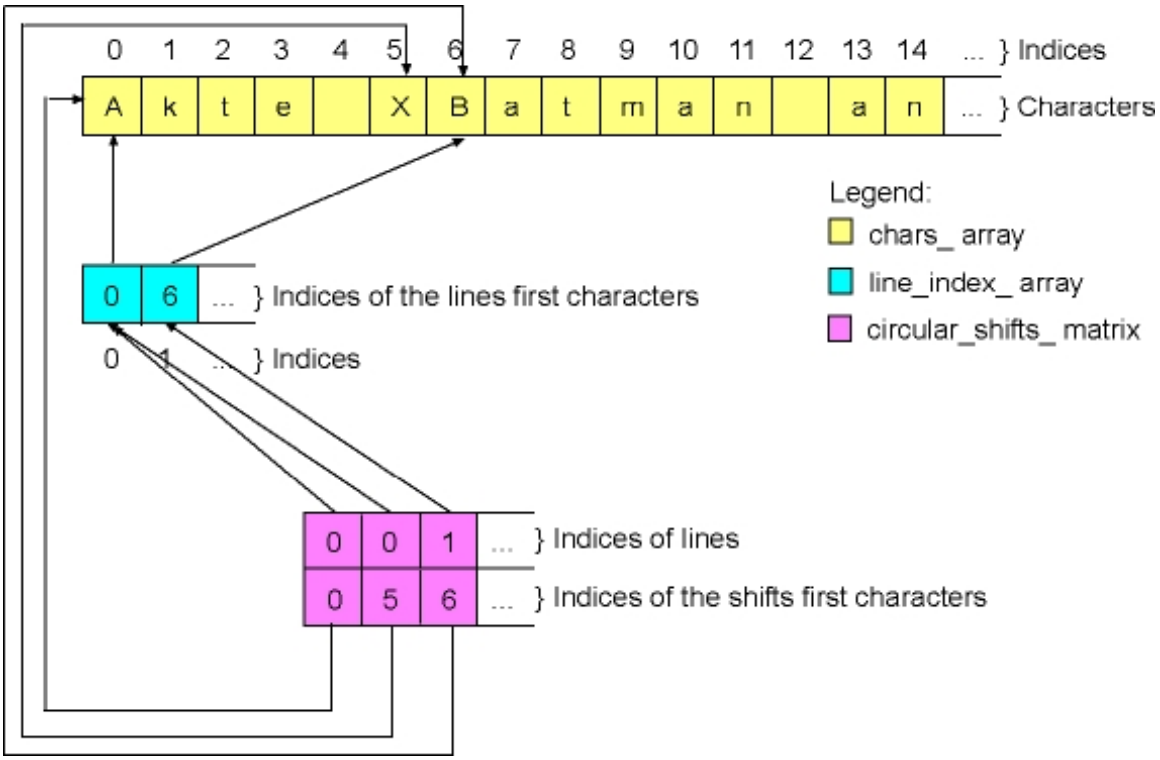
- Characters from the input file are stored in a character array named `chars_`. Words are delimited by a single space character.



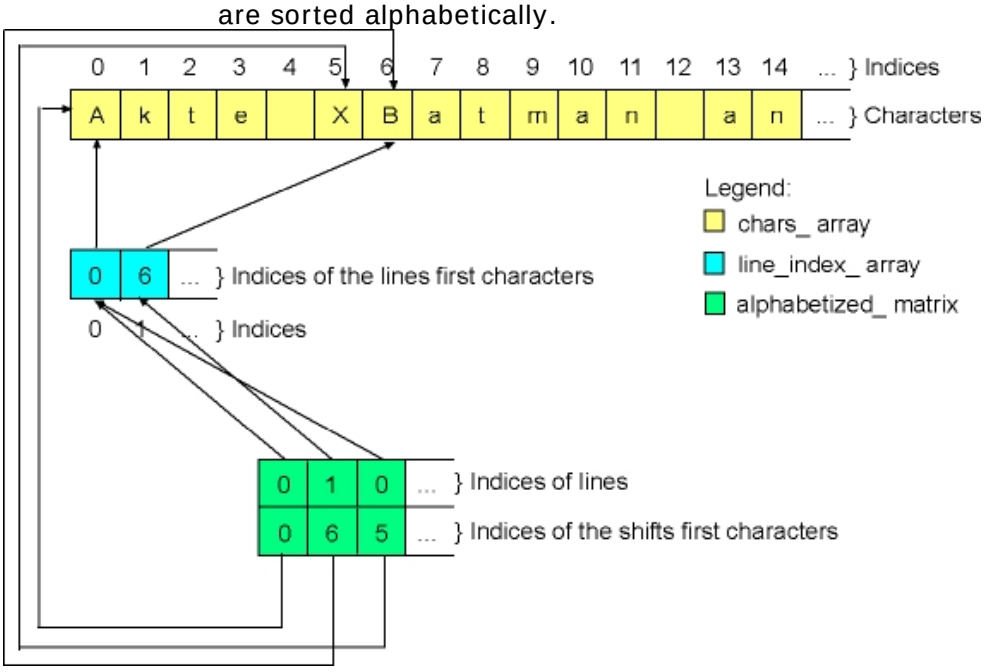
- `line_index_` is an integer array, which keeps indices of characters from the character array. Each index kept in the line index array is the index of the first character of a particular line.



- Two dimensional integer array (matrix) named `circular_shifts_` represents the set of all circular shifts of all original lines. A particular circular shift is represented as a pair of indices. The first value in such a pair is an index of the original line from the line index array. The second value in the pair is an index of a character from the character array. The character pointed by this index is the first character of a particular circular shift.



- Two dimensional integer array (matrix) named `alphabetized_` represents the sorted lines. This array has the same format as the circular shifts array. The difference is that in the alphabetized array lines come in a different order, i.e., they are sorted alphabetically.



1.4 Processing Algorithm

The main function controls the execution of the KWIC program:

- The `input` function is called to read and parse the lines from an input file. This function reads and processes all lines at once and represents it by means of the character (`chars_`) and line index (`line_index_`) array, as described before.
- The `main` function calls the `circularShift` function, which produces circular shifts of each particular line and stores it in the circular shifts array (`circular_shifts_`), as discussed before.
- After the circular shifts have been produced, the `main` function calls the `alphabetize` function, which sorts circular shifts alphabetically. The result is stored in the alphabetized array (`alphabetized_`), as described before.
- Finally, the `main` function calls the `output` function, which prints the sorted lines.

1.5 Your Assignment

1.5.1 Programming Assignment

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

- **Modification of data representation**
- **Modification of functionality**

1.5.2 Understanding Check

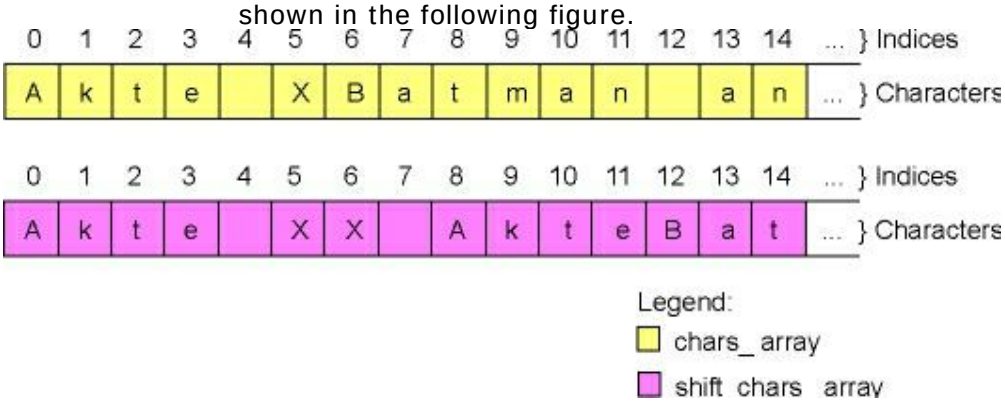
Finally, you need to answer to these three **questions**.

1.6 Modification of the Existing Data Representation

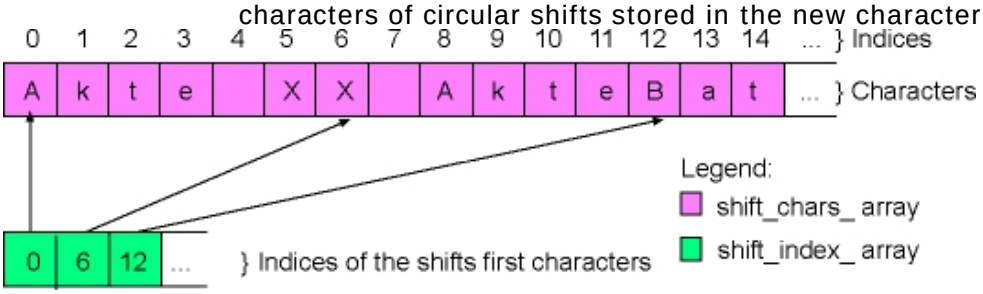
You need to modify the way how circular shifts and sorted circular shifts are represented by the KWIC system. In the new version, circular shifts should be now represented in the same way as the original lines, that is:

- All characters from all circular shifts should be stored in a new character array. For instance, consider the following original line: "Akte X". This line has two circular shifts (the first

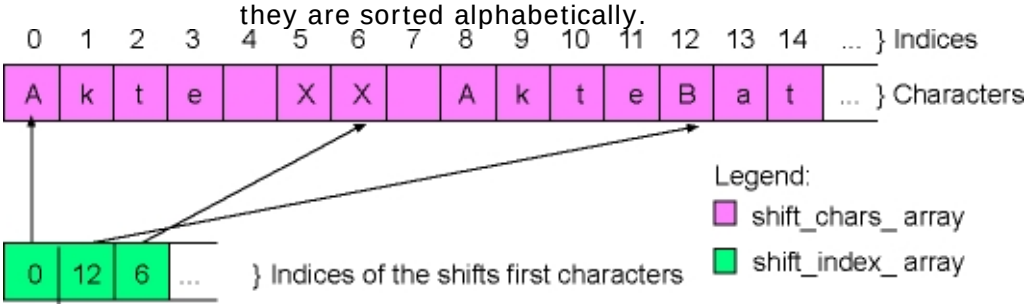
circular shift is equal to the original line): "Akte X", "X Akte". All characters from both circular shifts should be stored completely in the new character array one after another, as shown in the following figure.



- Shift index is an integer array, which keeps indices of the first characters of circular shifts stored in the new character array.



- The sorted shifts are represented in the same way and with the exactly same arrays as the circular shifts. Of course, the indices in the shift index array are arranged differently, i.e., they are sorted alphabetically.



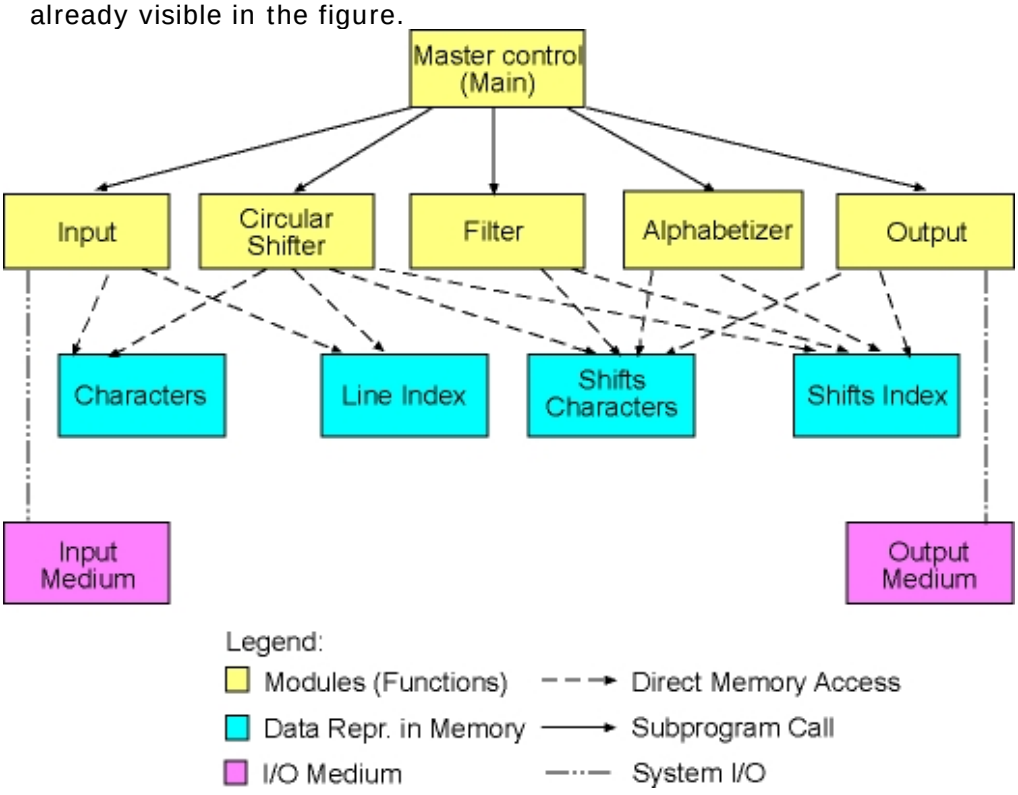
1.7 Modification of Functionality of the Existing System

You need to provide the existing KWIC system with additional functionality. The new KWIC system should be able to filter out some of the circular shifts that are produced. All circular shifts that start with a number, i.e., all shifts where the first character is a character between

'0' and '9' should be removed from the shifts array before the sorting starts.

Hint: Add a new module, i.e., a new function to the existing sytem. The name of the new module can be `filter`, and it should be responsible for iterating through the produced shifts and removing all the shifts that starts with a number. The `main` function should call the `filter` function right after the `circularShift` function and just before the call of the `alphabetizing` function.

Thus, the architecture of the new system looks as shown in the following picture. Note, that the modification in data representation is already visible in the figure.



1.8 Questions

Please, answer the following questions:

1. Which modules from the original KWIC system did you need to modify to implement the new data representation of circular shifts? What conclusion can you draw here? How does the KWIC system implemented by means of main/subroutine

architecture with shared data withstand design changes in data representation?

2. Which modules from the original KWIC system did you need to modify to add filtering function to the system? Was it difficult to add a new processing component (`filter` function) accessing the shared data? Can we conclude here that the KWIC system with main/subroutine architecture with shared data is robust to desing changes in system's function (assuming that new functions access the shared data)?

3. Would it be difficult to reuse modules from the existing KWIC system in other systems? For example, suppose that in another software system we need to sort a number of lines. Suppose also that in this system lines are represented differently than in the existing KWIC system. Could we reuse the existing `alphabetizing` function as it is, or do we need to make some modification to the exisiting code to be able to reuse it?

KWIC Implemented with Object-Oriented Architectural Style (Assignment 2)

Table of Contents

1.1 Introduction

1.1.1 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.

1.1.2 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

1.1.3 Assignment 2

To accomplish the Assignment 2, 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**.

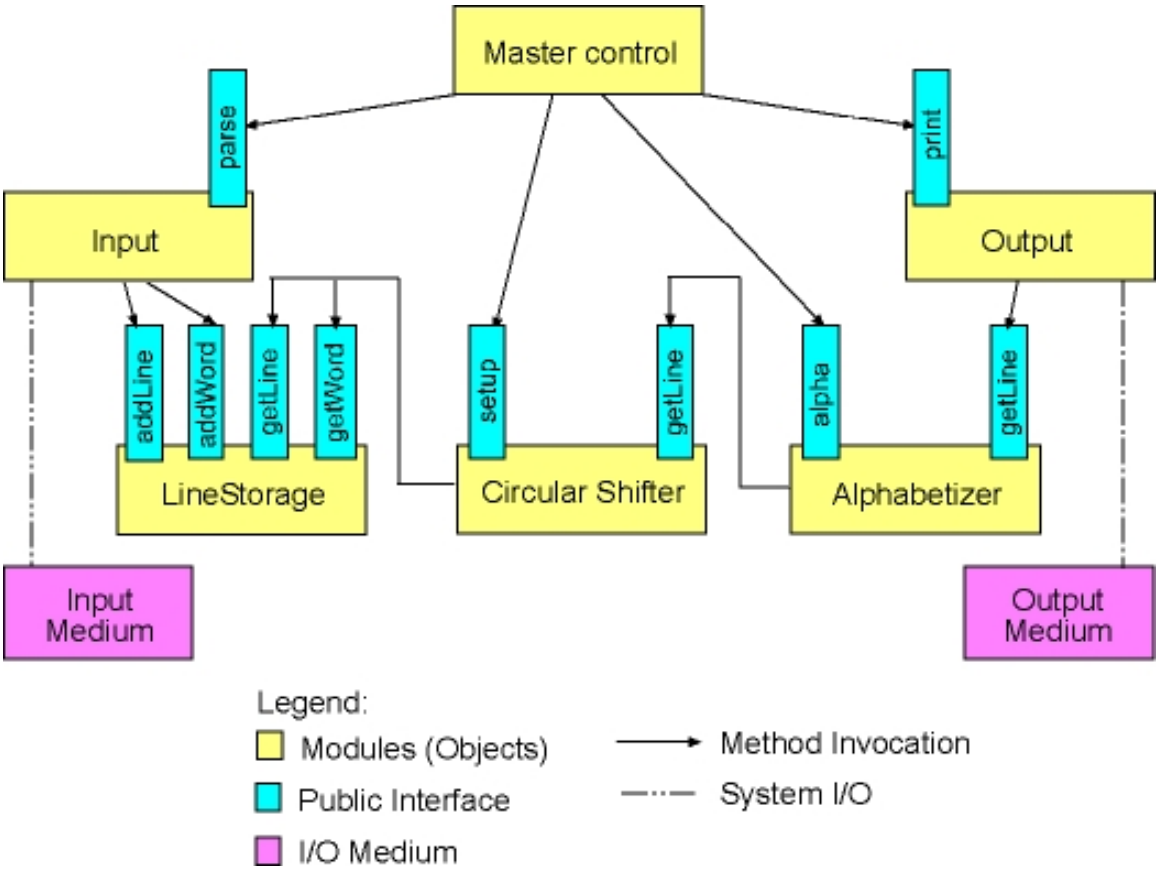
1.2 The Existing System

1.2.1 Architecture

The solution with object-oriented architectural style decomposes the system into six modules:

- LineStorage module, responsible for holding all characters from all words and lines.
- Input module, responsible for reading the data from a file and storing it in LineStorage module.
- CircularShifter module, responsible for producing circular shifts of lines stored in LineStorage module.
- Alphabetizer module, responsible for sorting circular shifts alphabetically.
- Output module, responsible for prinitng the sorted shifts.
- Master control module, responsible for controling the sequence of method invokations in other modules.

Contrary to the main/subroutine solution, in the object-oriented solution the data is not shared between the computational components. Instead, each module provides an interface that permits other components to access data only by invoking methods in that interface. For example, LineStorage module provides the public interface that allows other modules to set character in a particular word in a particular line, read a specific character, read, set or delete a particular word in a specific line, read whole line at once, etc.



1.2.2 *Properties of the Existing KWIC System*

Since the existing KWIC system is an object-oriented system, it can be easily described by:

- **Class Diagram**, depicting the result of static analysis and design of the system.
- **Sequence Diagram**, depicting the dynamics of the system.

1.2.3 *Source Code and Documentation*

Here is the **source code** in Java for the existing KWIC system, and the automatically generated javadoc **documentation** for the system.

1.2.4 *Test Data*

Here is a **sample input** file for the KWIC system. Please, save the file on your local drive. Specify the name of the file at the command line as an argumnet to the KWIC program.

1.3 Static Class Diagram

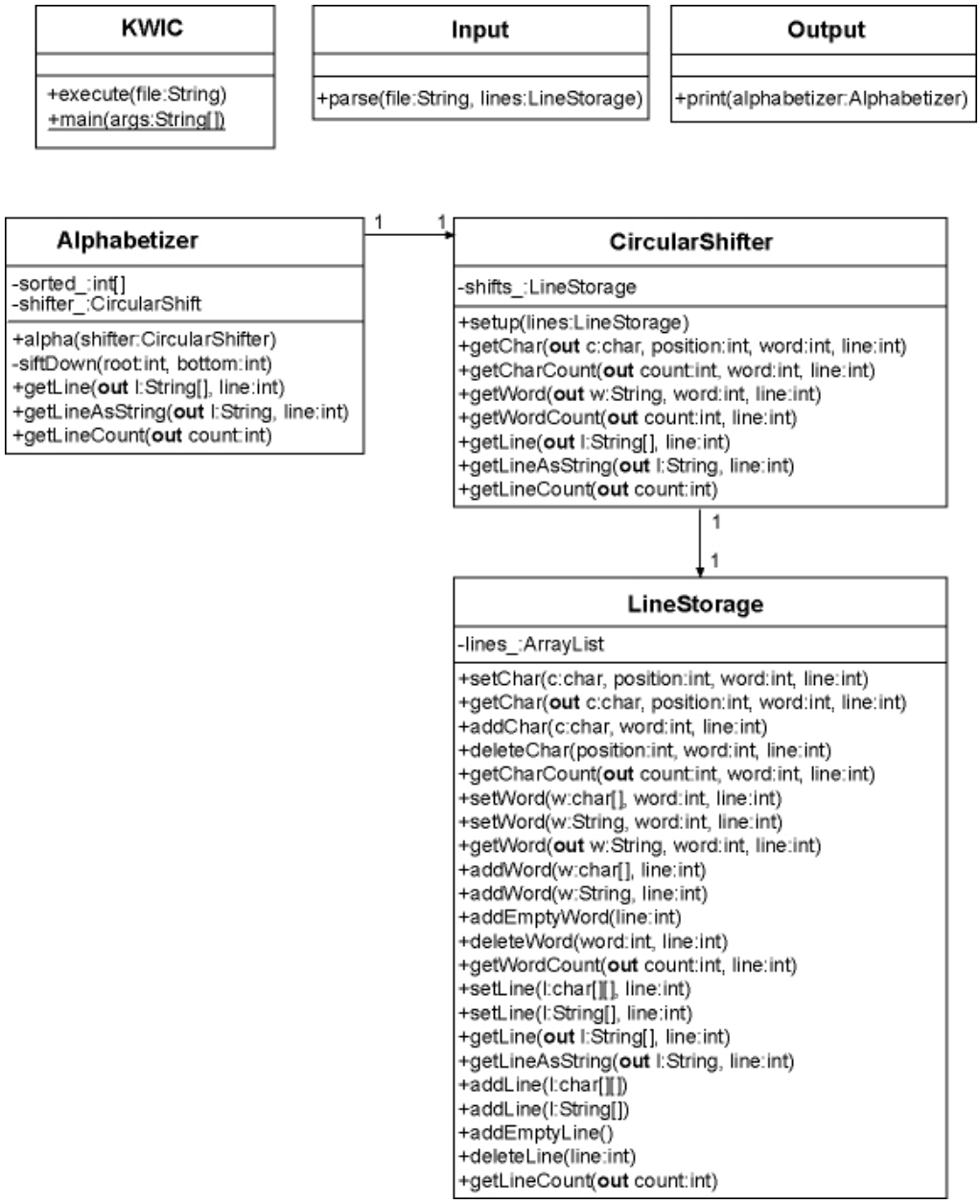
1.3.1 What is a Static Class Diagram?

Static class diagram is the result of static analysis and design of the KWIC problem and proposed object-oriented architectural solution of the problem. Such diagram includes the following information:

- It describes how modules from the original architectural solution maps onto classes from an object-oriented programming language (Java in our case).
- It depicts the public interface of each identified class. (Since we have an object-oriented system here, each object of a class from the system encapsulates its data. That means that all information contained in an object of a particular class is hidden from the outside world, i.e. other objects in the system can not access it directly. To be still able to manipulate the data of an object, its class provides a public interface, i.e., a number of public methods, which are called by other objects.)
- It depicts instance variables (attributes) of classes, i.e., data representation from each class is shown in the diagram.
- Finally, the diagram shows how classes relate to each other. Class relations are shown by means of so-called associations.

1.3.2 Static Class Diagram of the Existing System

Here is the UML (Unified Modeling Language) class diagram of the current system.



Here is a short description of the diagram:

- Each module from the architectural design maps exactly onto one class. Thus, there exist `KWIC` class (Master Control module), `Input` class, `Output` class, `LineStorage` class, `CicrularShifter` class and `Alphabetizer` class. Each class is represented as a rectangle box, having three horizontally arranged sub-boxes. The name of each class is written in the top sub-box.

- Public interface of each class is written into the bottom sub-box, as a list of method declarations. The plus (+) sign in front of a method name means that the method is defined as public and may be invoked by objects of other classes. The argument list is written within parenthesis. Each argument is represented with the name, its type and the direction in which the argument is passed. If there is no direction modifier that means that this method takes this argument, and if there is `out` modifier that means that the method returns this argument as a return value to the calling object. For example, `+getChar(out c:char, position:int, word:int, line:int)` method from the `LineStorage` class takes as arguments three integere values: position of the character, index of the word, and index of the line, and returns the value of that character.
- Instance variables (attributes) of a class are shown in the middle sub-box. The minus (-) sign in front of an instance variable means that the variable is defined as private and may not be manipulated by other objects directly. Instead, the public interface of the class should be applied to do so. That is completely in accordance with object-oriented principles of data encapsulation and information hiding, discussed above. After the minus sign, the name and the type of a particular instance variable are shown. For example, the `LineStorage` class has one private instance variable: `lines_`. This instance variable is of type `ArrayList`, and holds a list of lines.
- Finally, we have a number of associations between some of the classes. An association means that associated classes have as instance variables the objects of the associated class. For example, the `Alphabetizer` class has an association with the `CircularShift` class. That means that an object of the `Alphabetizer` class has as an instance variable an object of the `CircularShift` class. This variable is also depicted in the middle box of the `Alphabetizer` class. We

can spot there `shifter_` variable, which is of type `CircularShifter`. The numbering at each association end means that exactly one object of the `Alphabetiter` class holds exactly one object of the `CircularShifter` class.

1.4 Dynamic Sequence Diagram

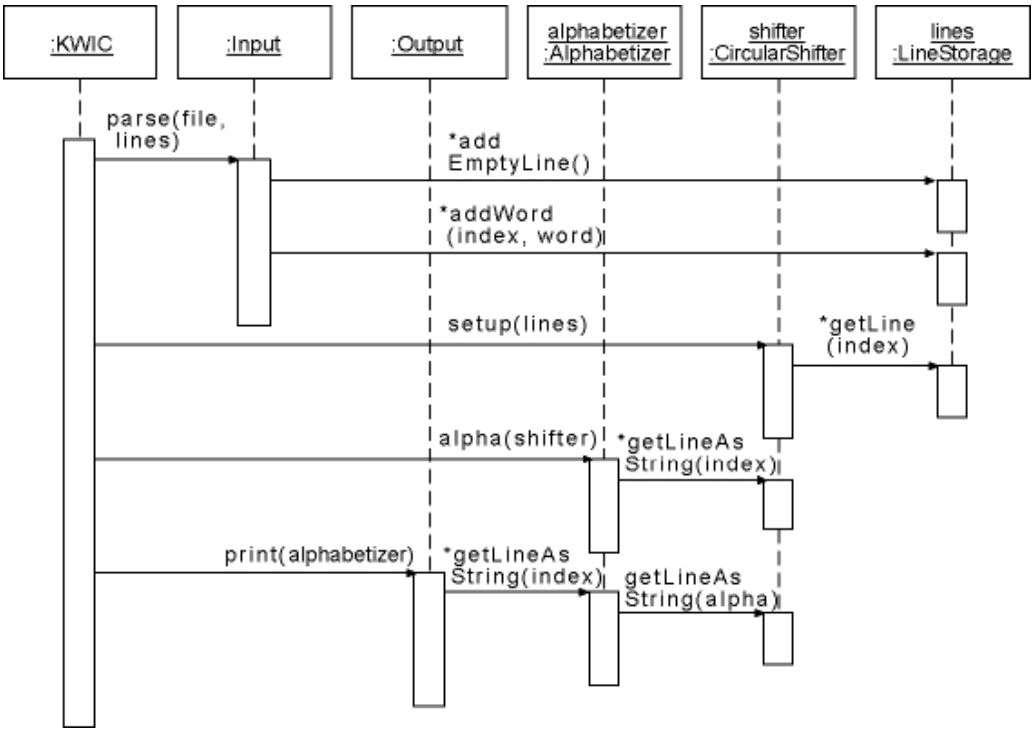
1.4.1 What is a Dynamic Sequence Diagram?

Dynamic (interaction) diagrams describe how group of objects collaborate to execute a certain task in a running program. In other words these diagrams show graphically the processing algorithm and its dynamic behaviour in a running system. Sequence diagram is a special type of dynamic diagrams that describes the time sequence of method invokations on the run-time in the following way:

- Objects are shown as boxes at the top of the diagram.
- Time is represented with the so-called object's lifeline - vertical line beneath the object's box. The time increases downward along the vertical object's lifeline.
- Method invokation is simbolized through a horizontal line connecting two vertical object's lifelines. The name of the method is written above such horizontal line. Obviously, horizontal lines which are further downwardds on an object's lifeline are invoked later in time than methods represented as horizontal lines nearer to the top of the object's lifeline.
- Conditions for method invokation and iterative method invokations may be simbolized with special signs. For example, iterative method invokation is depicted with the * sign.

1.4.2 Dynamic Sequence Diagram of the Existing System

Here is the UML (Unified Modeling Language) sequence diagram of the current system.



Here is a short description of the diagram:

1. An object of the `KWIC` class controls the whole execution of the program. Firstly, the `KWIC` object invokes the `parse` method of an object of the `Input` class. This method takes as arguments the name of the file containing lines, and an object of the `LineStorage` class to store the data.
2. The `parse` method of the `Input` object reads the data from the file. For each line in that file it creates a new line in the `LineStorage` object by invoking the `addEmptyLine` method on that object. After a new line has been created the `Input` object parses the line and adds all words (by means of `addWord`), one after another into the new line. The iterative property of this process is simbolized in the diagram with * signs in the front of the method names. After all words and lines has been processed, the `parse` method returns.
3. The `KWIC` object invokes the `setup` method on an object of the `CircularShifter` class. This method takes as an argument the `LineStorage` object holding the lines and produces circular shifts of all lines at once. During the

execution of the `setup` method, the `CircularShifter` object iterates through all lines in the `LineStorage` object, simbolized in the diagram with * sign in the front of the `getLine` method of the `LineStorage` object.

4. After circular shifts have been created, the `KWIC` object invokes the `alpha` method of an object of the `Alphabetizer` class. This method takes as an argument the `CircularShifter` object and sorts circular shifts alphabetically. During this process the `Alphabetizer` object invokes `getLineAsString` method of the `CircularShifter` class a number of times. Again, this is simbolized in the diagram with * sign in the front of the `getLineAsString` method of the `CircularShifter` object.
5. Finally, the `KWIC` object invokes the `print` method on an object of the `Output` class. This method takes as an argument the `Alphabetizer` object from the previous step. It iterates (* sign) then through all lines from the `Alphabetizer` object by invoking `getLineAsString` on that object. Notice that this method just forwards the request to the `CircularShifter` object to retrieve the data. Of course, the forwarded request does not include the original index, but the sorted one instead. When the `print` method returns, the execution of the program is finished.

1.5 Your Assignment

1.5.1 Programming Assignment

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

- **Modification of data representation**
- **Modification of processing algorithm**

1.5.2 Understanding Check

Finally, you need to answer to these three **questions**.

1.6 Modification of the Existing Data Representation

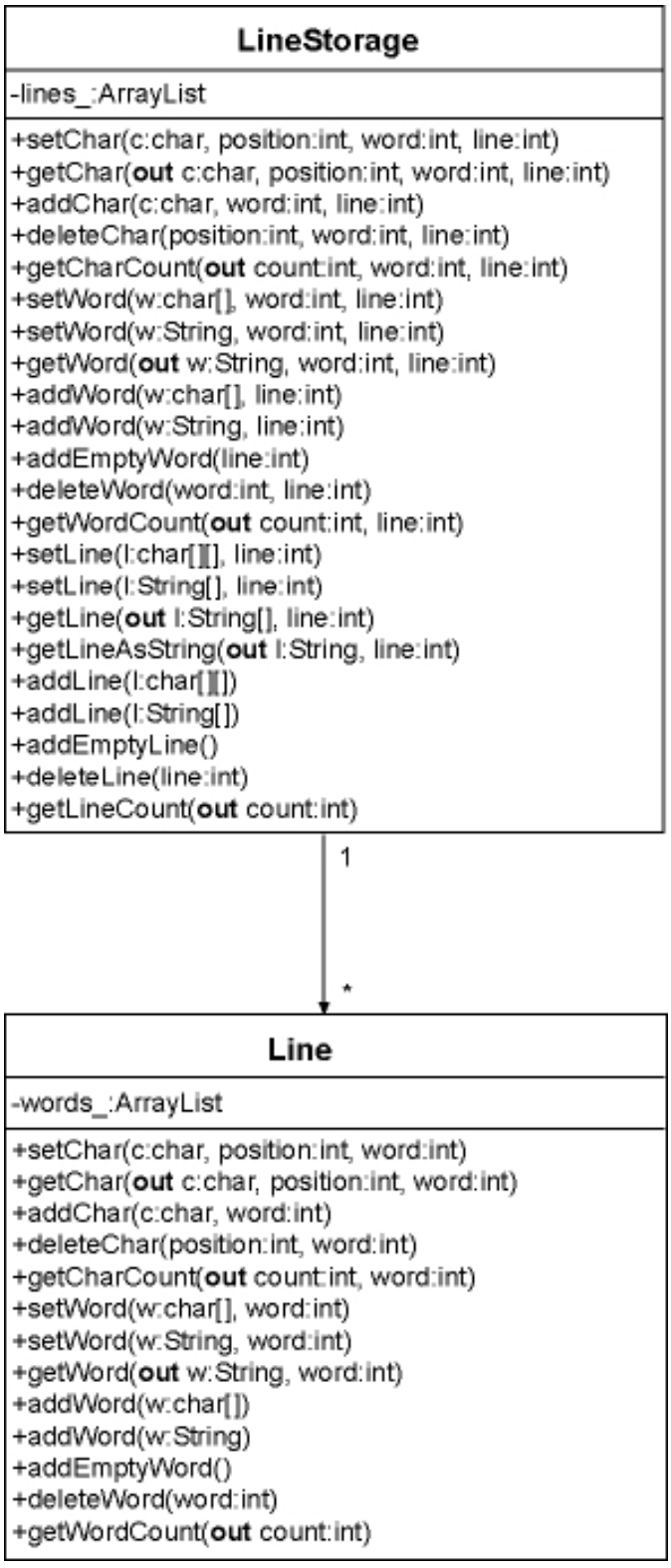
In the current version of the `LineStorage` class we have the following data structure for storing lines:

- An instance variable with the name `lines_` of type `ArrayList` holds all lines. The `ArrayList` class implements all standard dynamic list operations, i.e., we can add elements at the end of the list, we can insert elements in an arbitrary position in the list, we can delete elements, etc. An element stored in an instance of the `ArrayList` class can be any Java object. This can be done because instances of the `ArrayList` class hold instances of the `Object` class, which is the root of the single rooted Java class hierarchy, i.e., any Java class is a subclass of the `Object` class.
- In the current implementation `lines_` variable holds objects of the `ArrayList` class. That is, each line is again represented through an object of `ArrayList` class. Inside this list we store words of that particular line. Words are stored as instances of the `String` class.

You need to modify the data structure for storing lines in `LineStorage` objects. The new data representation should look as follows:

- A new class should be defined, providing the storage for a single line and a public interface to manipulate the content of a line. The name of the new class should be `Line`.
- The `Line` class stores a line into an object of the `ArrayList` class. The name of this instance variable should be `words_`. Words should be stored as instances of the `String` class in the `words_` object.
- The `LineStorage` class keeps lines in the same `lines_` object. However, this list is now filled with instances of the `Line` class.

The following class diagram depicts the modifications you need to implement. Note, that just those two classes that are relevant for the modification are shown. All other classes remain the same.

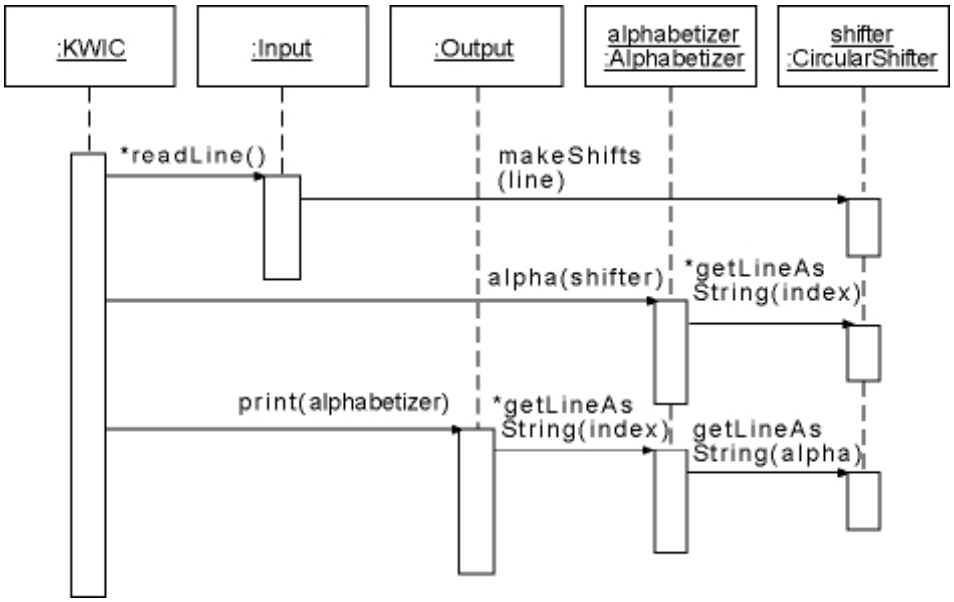

LineStorage
-lines_:ArrayList
+setChar(c:char, position:int, word:int, line:int)
+getChar(out c:char, position:int, word:int, line:int)
+addChar(c:char, word:int, line:int)
+deleteChar(position:int, word:int, line:int)
+getCharCount(out count:int, word:int, line:int)
+setWord(w:char[], word:int, line:int)
+setWord(w:String, word:int, line:int)
+getWord(out w:String, word:int, line:int)
+addWord(w:char[], line:int)
+addWord(w:String, line:int)
+addEmptyWord(line:int)
+deleteWord(word:int, line:int)
+getWordCount(out count:int, line:int)
+setLine(l:char[][], line:int)
+setLine(l:String[], line:int)
+getLine(out l:String[], line:int)
+getLineAsString(out l:String, line:int)
+addLine(l:char[][])
+addLine(l:String[])
+addEmptyLine()
+deleteLine(line:int)
+getLineCount(out count:int)
1
*
Line
-words_:ArrayList
+setChar(c:char, position:int, word:int)
+getChar(out c:char, position:int, word:int)
+addChar(c:char, word:int)
+deleteChar(position:int, word:int)
+getCharCount(out count:int, word:int)
+setWord(w:char[], word:int)
+setWord(w:String, word:int)
+getWord(out w:String, word:int)
+addWord(w:char[])
+addWord(w:String)
+addEmptyWord()
+deleteWord(word:int)
+getWordCount(out count:int)

1.7 Modification of the Existing Processing Algorithm

First, you need to implement an interactive version of the KWIC index system. That means the system won't read lines from a file but lines are inserted interactively by means of a simple command line user interface. Here is a transcript of a sample session:

- Add, Print, Quit: a
- > Star Wars
- Add, Print, Quit: a
- > The Empire Strikes Back
- Add, Print, Quit: a
- > The Return of the Jedi
- Add, Print, Quit: p
- 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

Further, you need to modify how line shifting is performed. In the current version all lines are read at once, and line shifting is done on all lines at once after they are read. In the new version you need to perform line shifting on each line as it is read from the command line. The following sequence diagram depicts the new situation.



Here is a short description of the diagram. The `KWIC` object controls the command line interface. After a command for adding a line has been issued, the `KWIC` object invokes `readLine` method of the `Input` object. The `Input` reads the line and passes it to the `CircularShifter` object by invoking its `makeShifts` method. After shifts have been made, `makeShifts` and `readLine` methods return and the `KWIC` object is again in charge, waiting for a new command from the user.

1.8 Questions

Please, answer the following questions:

1. Which modules from the original KWIC system did you need to modify to implement the new data representation of lines within the `LineStorage` class? What conclusion can you draw here? How does the KWIC system implemented by means of object-oriented architecture withstand design changes in data representation within a particular module? Is this true for any other object-oriented system?
2. Which modules from the original KWIC system did you need to modify to implement a new processing algorithm in the system? Can we conclude here that the KWIC system with object-oriented architecture is robust to desing changes in processing algorithm of the system?

3. Would it be difficult to reuse modules from the existing KWIC system in other systems? For example, suppose that in another software system we need to sort a number of lines. Suppose that the sorting algorithm expects lines to be represented as arrays of characters. Can you describe a possible object-oriented solution to reuse the `LineStorage` class without need to modify the source code of the class.

KWIC Implemented with Event Based Architectural Style (Assignment 3)

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 3

To accomplish the Assignment 3, 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. Event Based Systems in General

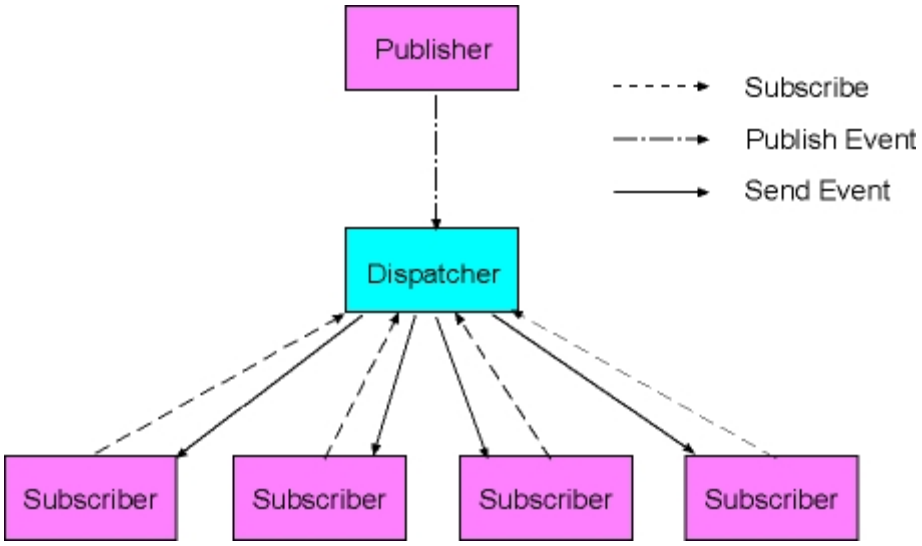
The idea behind event based systems is that instead of invoking a procedure directly, a component can announce or broadcast one or more events. Other components in the system can register an interest in an event by associating a procedure with it. When the event is announced, the system itself invokes all of the procedures that have been registered for the event. Thus, an event announcement "implicitly" causes

the invocation of procedures in other modules. For example, let us consider an Integrated Development Environment for Java. Such IDE consists of tools such as editors for source code, variable monitors, a debugger, etc. Usually, such systems utilize event based architecture. Thus, editors and variable monitors register for the debugger's breakpoint events. When a debugger stops at a breakpoint, it announces an event that allows the system to automatically invoke procedures of those registered tools. These procedures might scroll an editor to the appropriate source line or redisplay the value of monitored variables. In this scheme, the debugger simply announces an event, but does not know what other tools or actions (if any) are concerned with that event, or what they will do when the event is announced.

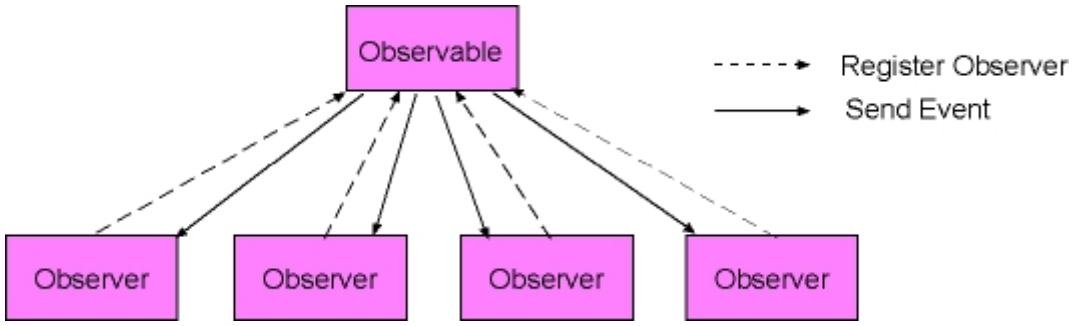
2.1. Strategy of Events Handling

When an event is announced in an event based system, the system itself invokes automatically all of the procedures that have been registered for the event. That means that the system must implement a certain strategy how events are handled in the system, i.e., how events are being dispatched to registered components in the system. There exist a number of different strategies of dispatching events. Generally, we may divide these strategies into two groups:

- Systems with a separate dispatcher module, which is responsible for receiving all incoming events and dispatching them to other modules in the system. The implementation of the dispatcher decides how events are sent to other modules. For example, the dispatcher may broadcast events to all modules in the system, and then modules themselves need to decide what they want to do with events. Another example is that the dispatcher manages lists of modules interested in particular events. In that case the dispatcher sends an event just to those modules that registered for that event. This strategy is usually called Publish/Subscribe strategy and may be seen on the following picture:



- Systems without a central dispatcher module, where each module allows other modules to declare interest in events that they are sending. Thus, whenever a module sends an event it sends the event itself to those modules that registered an interest in such event with that module. This strategy is usually called Observable/Observer and may be depicted as follows:

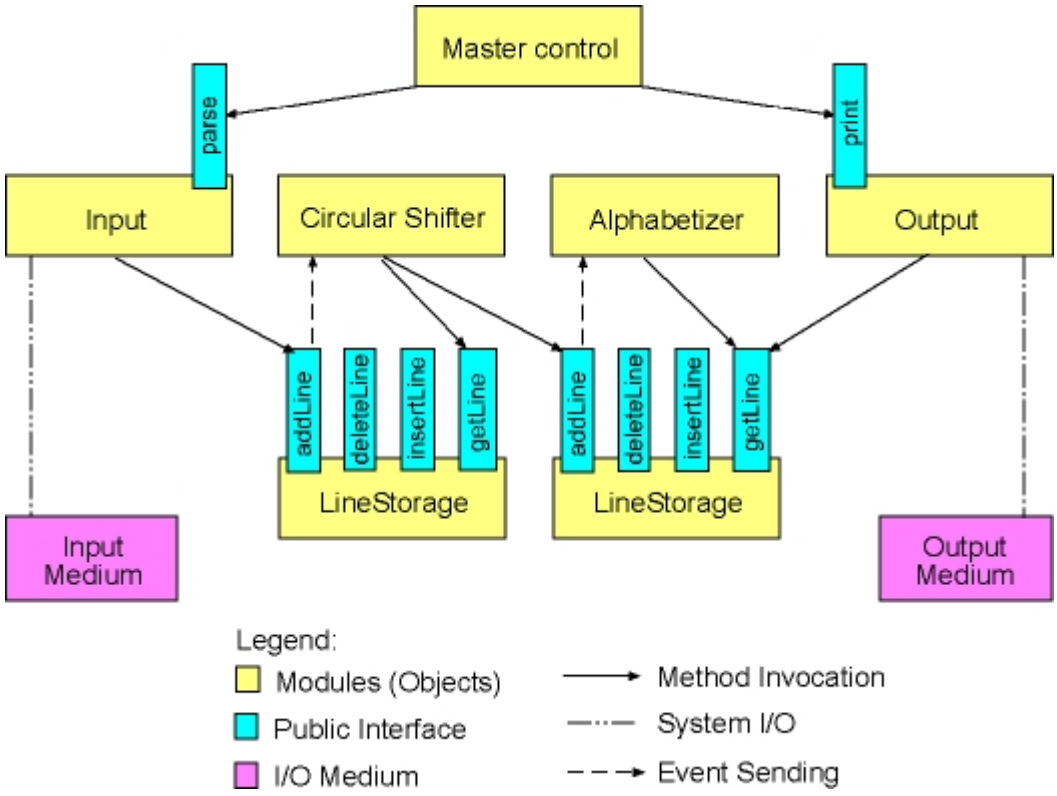


3. The Existing System

Event based KWIC system is composed of the following components:

- Two LineStorage modules. The first LineStorage module is responsible for holding all original lines, and the second LineStorage module is responsible for holding all circular shifts.

- Input module, responsible for reading the data from a file and storing it in the first LineStorage module.
- CircularShifter module, responsible for producing circular shifts and storing them in the second LineStorage module.
- Alphabetizer module, responsible for sorting circular shifts alphabetically.
- Output module, responsible for printing the sorted shifts.
- Master control module, responsible for overall control of the system.



3.1. Properties of the Existing System

The existing KWIC system may be described by:

- **Run-Time Event Interactions**
- **Observable / Observers Dispatching Strategy**

3.2. Event Based KWIC System in Run-Time

The existing KWIC system works as follows. The act of adding a new line to the first LineStorage causes an event to be sent to the CircularShifter module. This allows the CircularShifter module to produce all circular shifts of the line and to store them into the second line storage. This in turn causes another event to be sent to the Alphabetizer module, so that this module may sort the newly added circular shifts.

Thus, we have the following event based interactions in the system:

- The CircularShifter registers its interest in the first LineStorage module. The first LineStorage module sends an event whenever a new line has been added to it. The CircularShifter module receives the event sent by the first LineStorage module. As a response to receiving the event the CircularShifter module produces all circular shifts of the newly added line and store those shifts in the second LineStorage module.
- The Alphabetizer module registers its interest in the second LineStorage module. The second LineStorage module sends an event whenever a new circular shifts has been added to it. The Alphabetizer module receives the event sent by the second LineStorage module. As a response to receiving the event the Alphabetizer module sorts circular shifts.

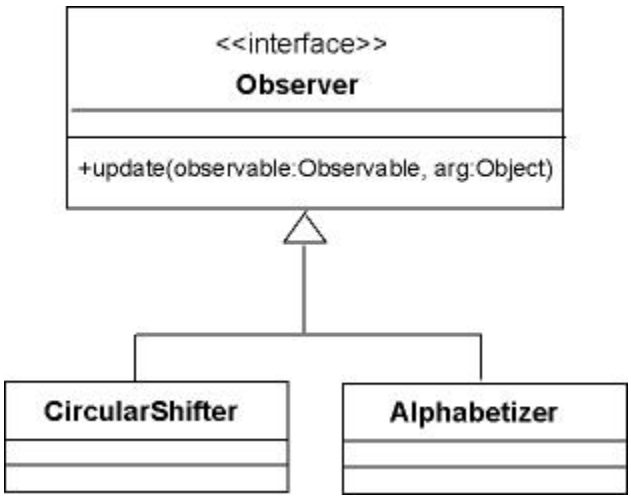
3.3. Observable & Observers

The current KWIC system implements Observable/Observers pattern for handling events in the system. Thus, the two LineStorage modules (the first one applied to store the original lines, and the second one applied to store the circular shifts) are implemented as Observable modules. On the other hand the CircularShifter and the Alphabetizer module are implemented as Observers. Thus, the CircularShifter is an Observer of the first LineStorage module, whereas the Alphabetizer is an Observer of the second LineStorage module.

3.3.1. Java Support for Observer

Since the standard Java library, more specifically the standard java.util package provides the Observable class and the Observer interface, which implement the described behavior, these elements were reused to implement the

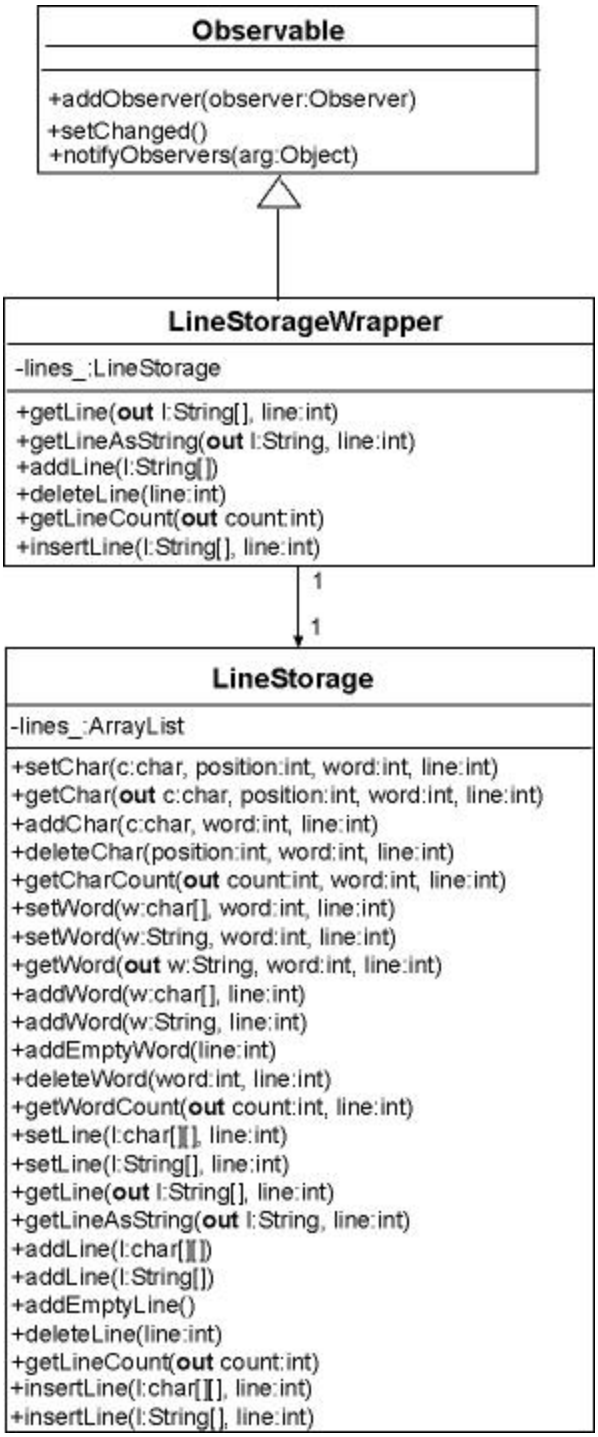
Observable/Observers mechanism in the KWIC system. The following diagram shows the CircularShifter and the Alphabetizer class.



Thus, the Observer interface from the standard java.util package has only one public method: update. This method is invoked each time when an Observable object (observed by that specific Observer object) sends an event in the system. Thus, the CircularShifter and the Alphabetizer class implement this method to take their action whenever the LineStorage module that they observe sends an event. The event itself is passed as the second argument in the update method.

3.3.2. Java Support for Observable

The second diagram shows the LineStorageWrapper class which reuses the LineStorage class from the second assignment, but extends its functionality by inheriting from the Observable class from the java.util package.



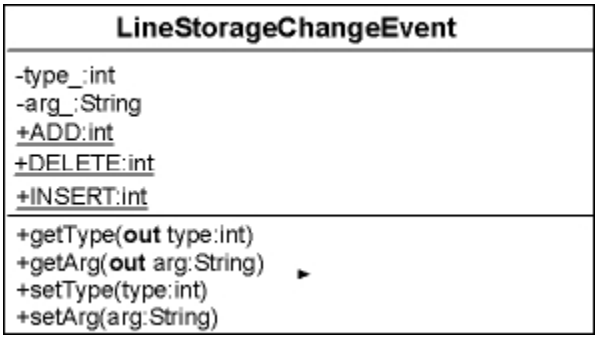
Thus, the Observable class from the standard java.util interface provides public methods that allow other objects to register themselves as Observers of objects of that class (addObserver() method). Further, the Observable class provides a method to notify Observers by sending an event in the system (notifyObservers() method). This method takes as an argument a single object of an Event class.

3.3.3. Current Implementation

In the current KWIC system we apply two LineStorageWrapper objects to store the original lines and circular shifts respectively. The added functionality from the Observable class allows us to invoke notifyObservers() method whenever a new line is added, deleted or inserted in any of these two objects. This in turn causes update method in the CircularShifter or Alphabetizer to be implicitly invoked, thus allowing us to take desired actions, i.e., to produce and add circular shifts in the case of CircularShifter object or to sort circular shifts alphabetically in the case of the Alphabetizer object.

3.3.4. Event Class

Finally, here is the diagram of the Event class from the existing KWIC system.



Note that the Event class implements a simple protocol for describing the type of the event that occurred in a LineStorageWrapper object. Thus, by setting the type of the event we may tell the Observers what happened exactly. For example, if we set type of the event to ADD, we are telling Observers that a new line has been added to a LineStorageWrapper object. On the other hand, setting the type of the event to DELETE, we are informing Observers that a line was deleted.

4. Your Assignment

4.1. Programming Assignment

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

- **Implement an interactive version of the system**
- **Extend the functionality of the system**

4.2. Understanding Check

Finally, you need to answer to these three **questions**.

5. Implementing Interactive KWIC System

You need to implement an interactive version of the KWIC index system. That means the system won't read lines from a file but lines may be manipulated interactively by means of a simple command line user interface. Here is a transcript of a sample session:

- Add, Delete, Print, Quit: a
- > Star Wars
- Add, Delete, Print, Quit: a
- > The Empire Strikes Back
- Add, Delete, Print, Quit: a
- > The Return of the Jedi
- Add, Delete, Print, Quit: p
- ----------
- 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
- ----------
- Add, Delete, Print, Quit: d
- > Star Wars
- Add, Delete, Print, Quit: p
- ----------
- Back The Empire Strikes
- Empire Strikes Back The
- Jedi The Return of the
- Return of the Jedi The
- Strikes Back The Empire
- The Empire Strikes Back
- The Return of the Jedi
- of the Jedi The Return
- the Jedi The Return of
- ----------

Thus, the command that the interactive version of KWIC system should be able to execute are:

- Add command for adding a new line with key binding 'a'
- Delete command for deleting a line with key binding 'd'
- Print command for printing shifts sorted alphabetically with key binding 'p'
- Quit command for exiting the system with key binding 'q'

5.1. Implementation Hint

The current implementation of the system and the LineStorageChangeEvent already supports sending/dispatching a delete event. Thus, when a line is deleted from the first LineStorage a LineStorageChangeEvent with the DELETE type is sent to the CircularShifter object. Now, to implement the delete functionality you need to handle the DELETE event inside the update method in CircularShifter class and to delete all circular shifts of the deleted line as well.

6. Extending the Functionality of the System

To extend the functionality of the existing KWIC system you need to implement words index. This index should keep all words that appear in the original lines and the number of their occurrences in the original lines. For example, the words index for the following original lines looks as follows:

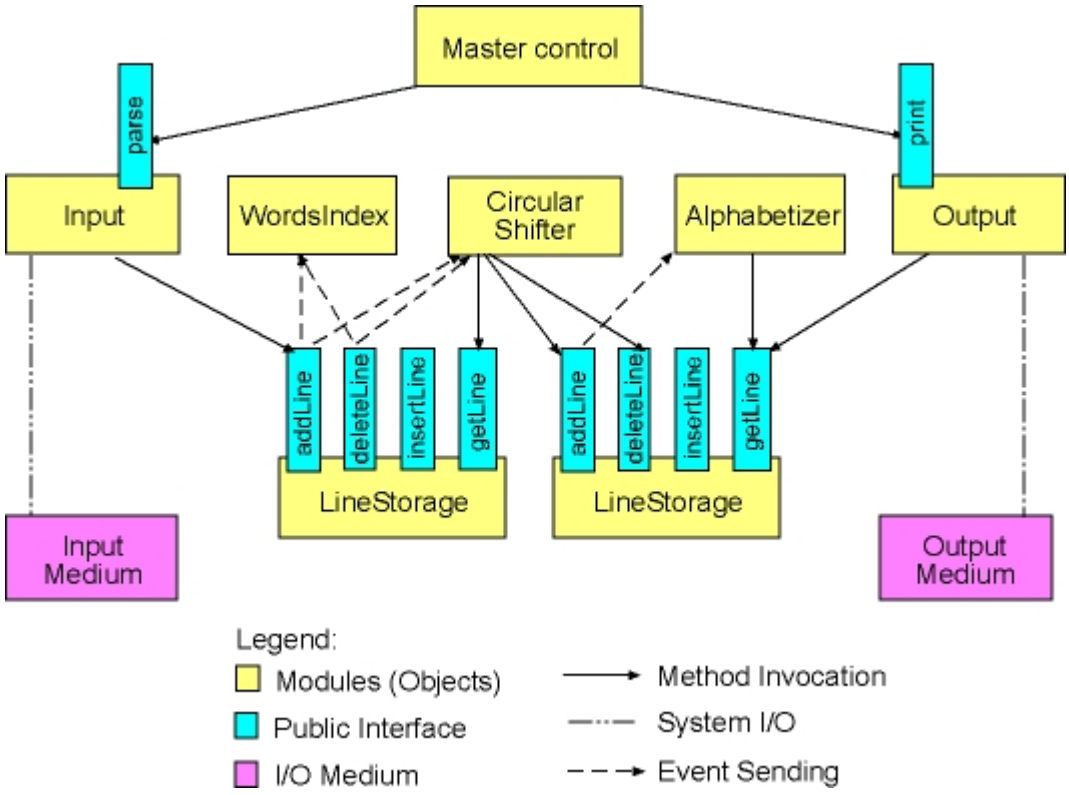
- Original Lines:
  - Star Wars
  - The Empire Strikes Back
  - The Return of the Jedi
  - The Phantom Menace
  - The Clone Wars
- Words Index:
  - Star: 1
  - Wars: 2
  - The: 4
  - Empire: 1
  - Strikes: 1
  - Back: 1
  - Return: 1
  - of: 1
  - the: 1
  - Jedi: 1
  - Phantom: 1
  - Menace: 1
  - Clone: 1

Further, you need to incorporate a new command in your command line user interface of KWIC system. This new command should allow users to print out the words index. Thus, your KWIC system need to support the following commands:

- Add command for adding a new line with key binding 'a'
- Delete command for deleting a line with key binding 'd'
- Print command for printing shifts sorted alphabetically with key binding 'p'
- Index command for printing words index with key binding 'i'
- Quit command for exiting the system with key binding 'q'

6.1. Implementation Hint

To implement the words index you may add a new module into the system, e.g. WordsIndex module. The new module should be declared as an additional Observer of the first LineStorage module. Thus, each event sent by the first LineStorage module will be sent to the WordsIndex module additionally to the CircularShifter module. In the update method of the WordsIndex you should then provide an implementation that will keep the WordsIndex up-to-date. Here is the architecture of the extended KWIC system:



7. Questions

Please, answer the following questions:

1. Which modules from the original KWIC system did you need to modify to implement the WordsIndex module? What can you conclude here? Do systems implemented with event based architecture allow for easy adding

of new modules, assuming that the new modules apply the already existing event protocol?

2. Please, describe the easiest way to implement a WordsIndex that should keep all words that appear in the original lines and the number of their occurrences in the circular shifts.
3. One of the main properties of event based systems is so-called "implicit" invocation of procedures. That means some of the procedures in the system are not invoked directly, but rather an event is sent into the system, and then the system itself after processing the events invokes these procedures. Now, taking into account that the processing of events and deciding which procedures to call might be quite complex in systems with a large number of modules and a complex event protocol what can you conclude about the performance of event based system? Do you think that these systems may achieve the same level of the performance as systems with direct procedure calls?

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

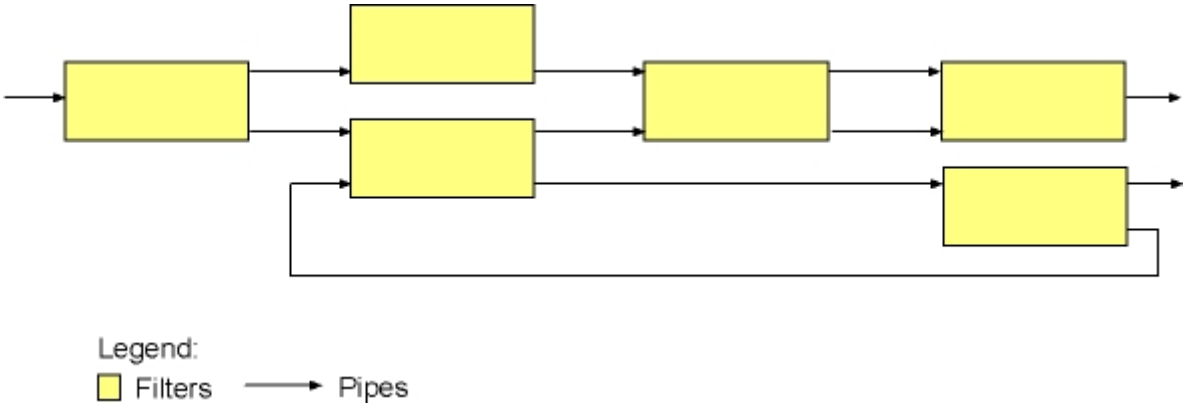
1.3. Assignment 4

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

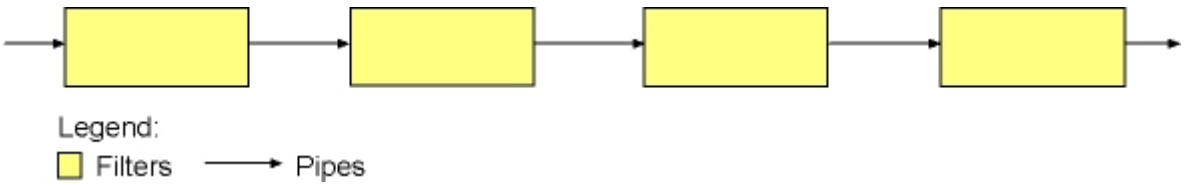
In a pipe and filter style each component has a set of input streams and a set of output streams. A component reads streams of data on its input streams, processes

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.

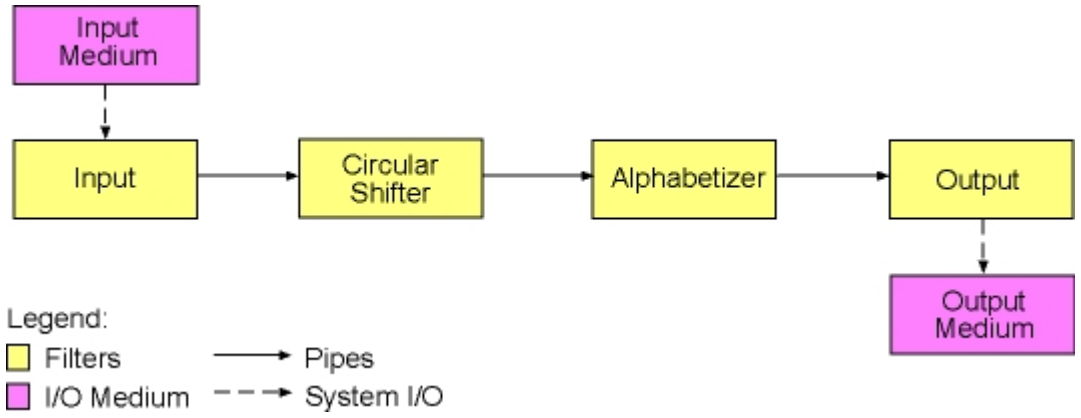


The best-known examples of pipe and filter architectures are programs written in the UNIX shell. UNIX supports this style by providing a notation for connecting components (represented as UNIX processes) and by providing run-time mechanisms for implementing pipes.

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:

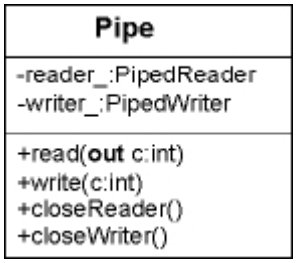
- **Pipes and filters mechanism**
- **Current pipeline**

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

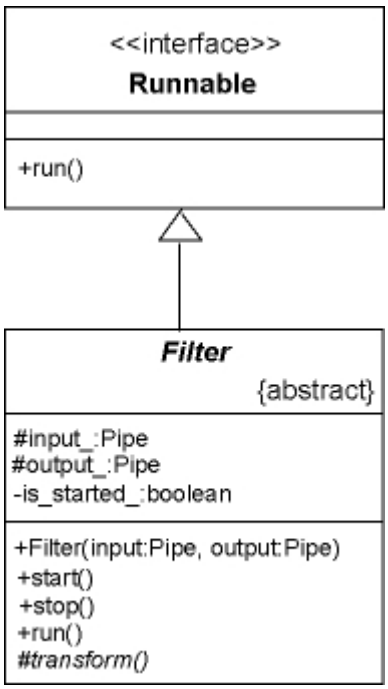
thread writes some data to the pipe, and the consumer thread "consumes" that data from the pipe.

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.

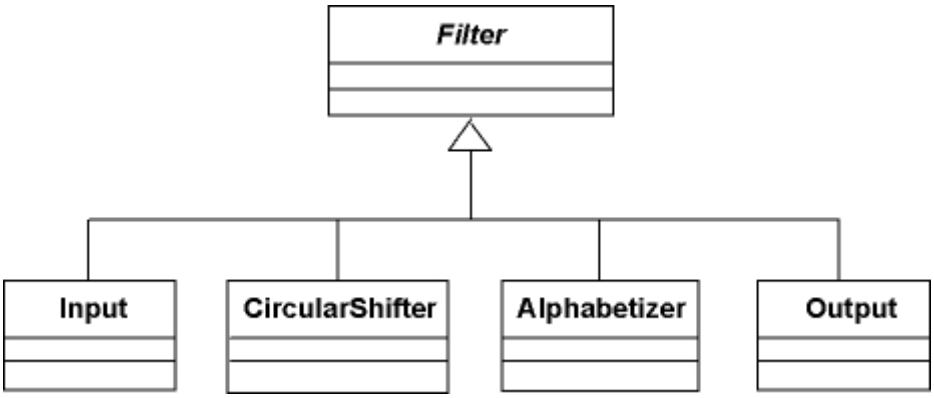
Filter objects may be combined into a typical producer/consumer scenario by simply sharing a pipe object. For example, we assign the output pipe of the first filter to the input pipe of the second filter. In that way whenever the first filter produces some data and writes it to its output pipe, this 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

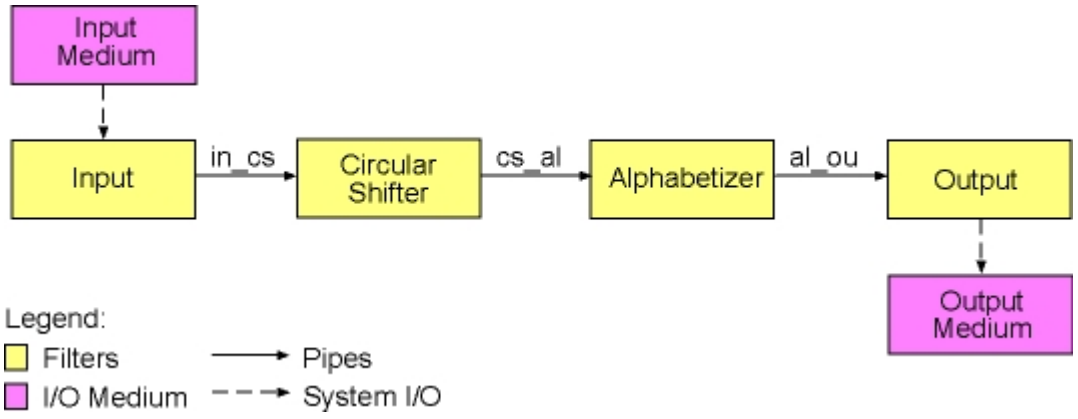
All KWIC filters are of course subclasses of abstract Filter class. These relationships are depicted on the following class diagram:



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?