Design Specification

Content:

1. Pattern

a. Pattern has been used in stage 2:

Template method
Strategy pattern
Observer pattern

b. Pattern might be used in stage 2:

Proxy pattern
Factory pattern
Adapter pattern

Section a:

Template method:

The template method is used in the package “decode”, “simpleIndex” and “simpleSearch”. Because I find my code in these three packages has amount of similar part, I use template method to avoid rewriting the same part. For example, in package “decode” TxtDecoder`s file stream is different from “WordDecoder”, but the other processes are same, so I write a father class provide the similar code, and abstract the different operations.

Strategy pattern:

This pattern is already used in the stage 1, but the requirements of stage 2 do not change the file format, so it keeps what it used to be.

Observer pattern:

Observer pattern is used between class “SerialWordSearch”, “SingleWordSearch” and “WordIndexLoader”. It works when the main Index is opened, the subject pass the index content to the two observers, observers save the information they need and make a sign when they collect enough information. Subject will stop when all the observers` information is enough.

I choose this pattern for word search because the main index file is big and will cost a lot time if I open it many times for one query. This pattern can made the IO operate only once for one query, that`s a useful pattern to help different objects to gather the information.

Section b:

This section uses “maybe”, because I consider I use the theory of some pattern but not correctly use the patterns` interface.

Proxy pattern:

This pattern` s theory is separated some slow operations from a object and when these operations are really called then active these operations. With this pattern I can define an object of an entity and not active all the functions of the entity.

So I use this pattern theory in the query. I make each query`s element like a vector. The vector collects all the same type information in one query. When the query sentence`s first scanning is finished the vector will open the relative index file, get all the information it needs.

Factory pattern:

The factory pattern is a creation-style pattern. I use it theory inside the class “Search”. Because I make the query elements into vectors, use another pattern and examine the type is not a good method. And because they are vectors they must create once for one query and must keep their position in the global field.

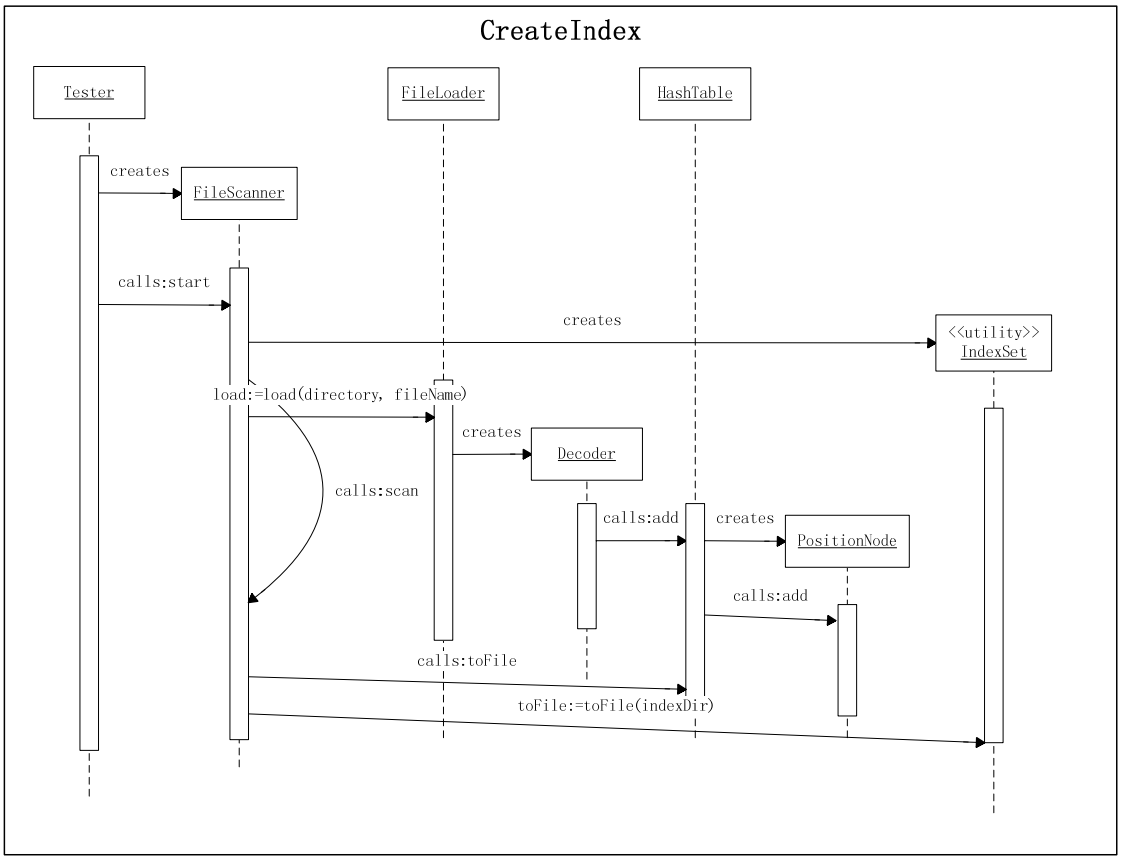
I use factory pattern`s theory inside the search. It will assign an element to its vector correctly and create the vector the first time an element appears. And in the “getContents()” method the inside factory will assign the correct vector to give out its sub query`s result.

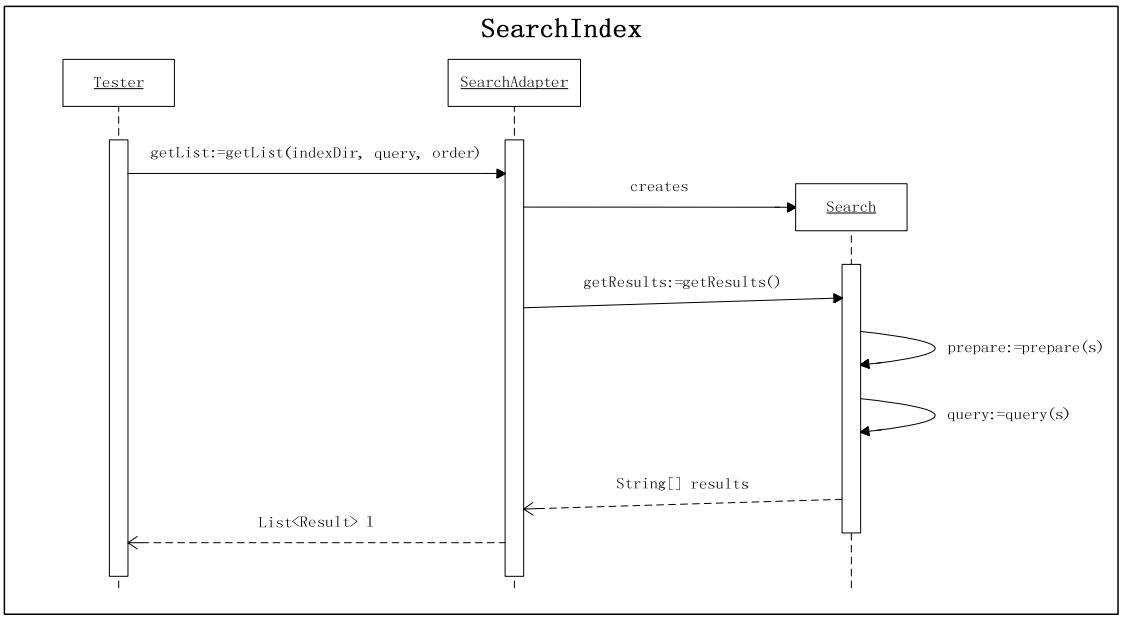
Adapter pattern:

The adapter pattern makes an old interface to a new one. I used it between “Search” and “Tester” .But “Search” is not implemented on an interface, the adapter class “Search Adapter” is not implemented on interface.

2. UML and Specification

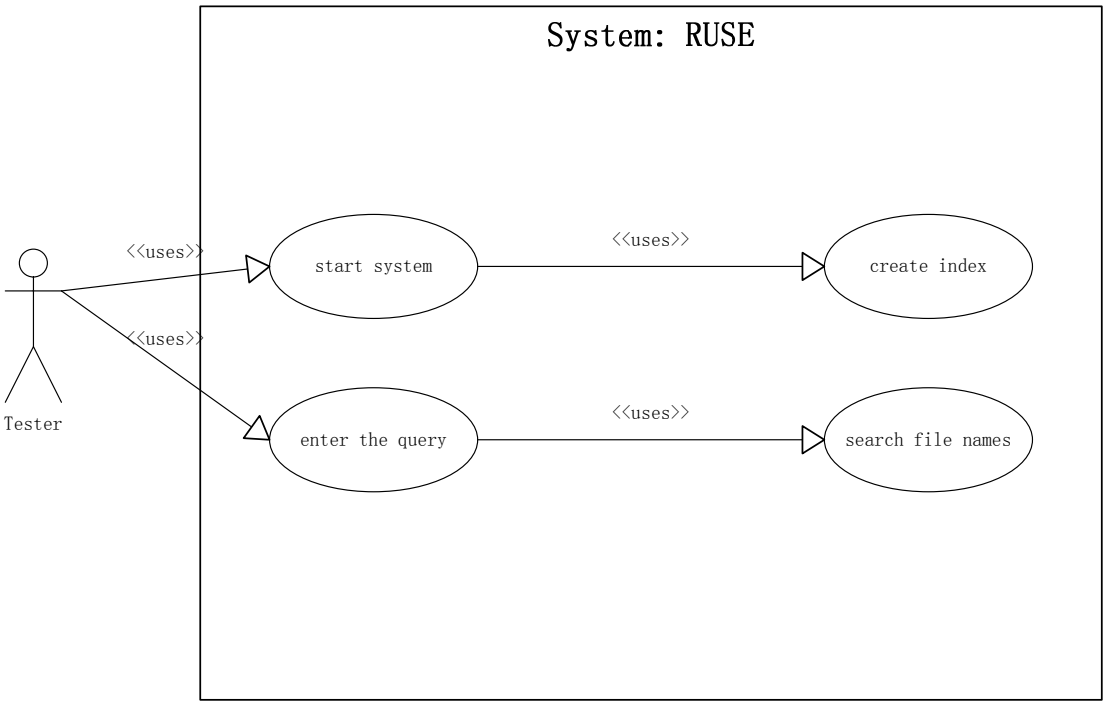
Sequence Graph



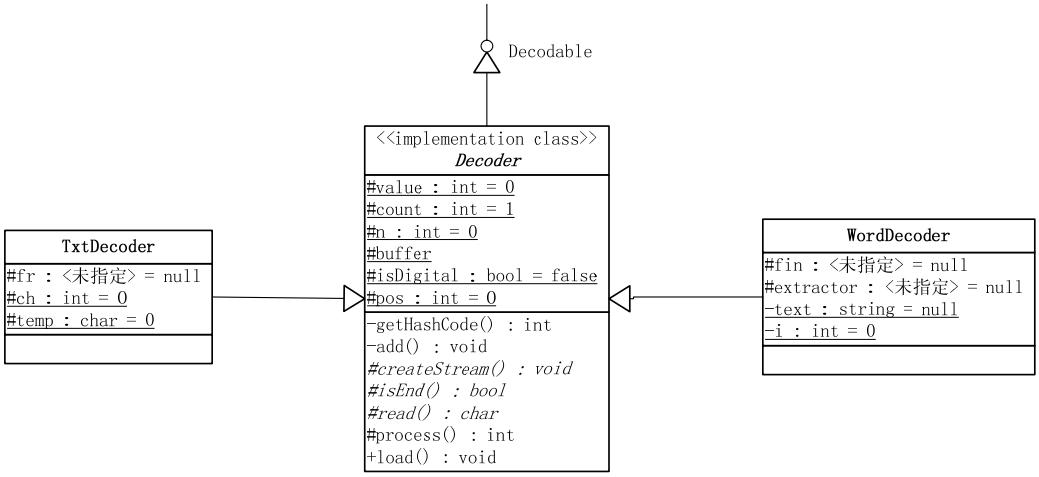


Use Case Graph

The use case graph is same to the old one

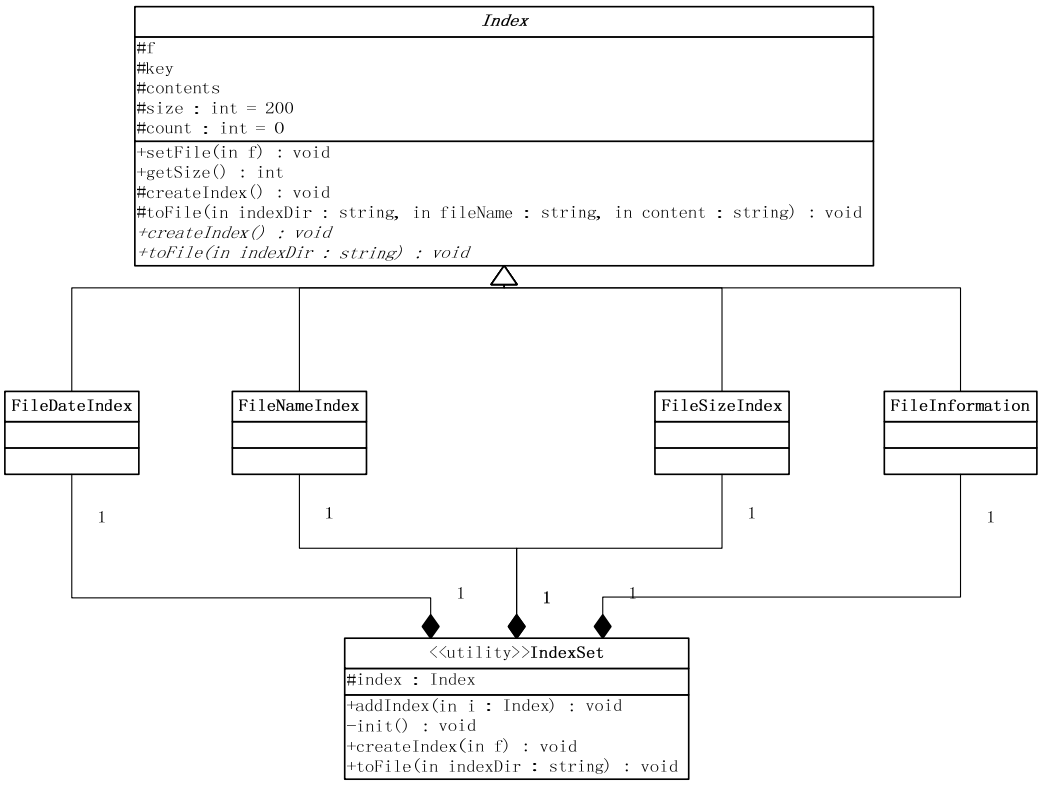


Package “decode”



The “fin”`s type is FileInputStream, “fr”`s type is FileReader, extractor`s type uses the POI library. Class “Decoder” is an abstract class and implements interface “Decodable” and provide template methods “process”, “load” , “add” and “getHashCode” for the derived class.

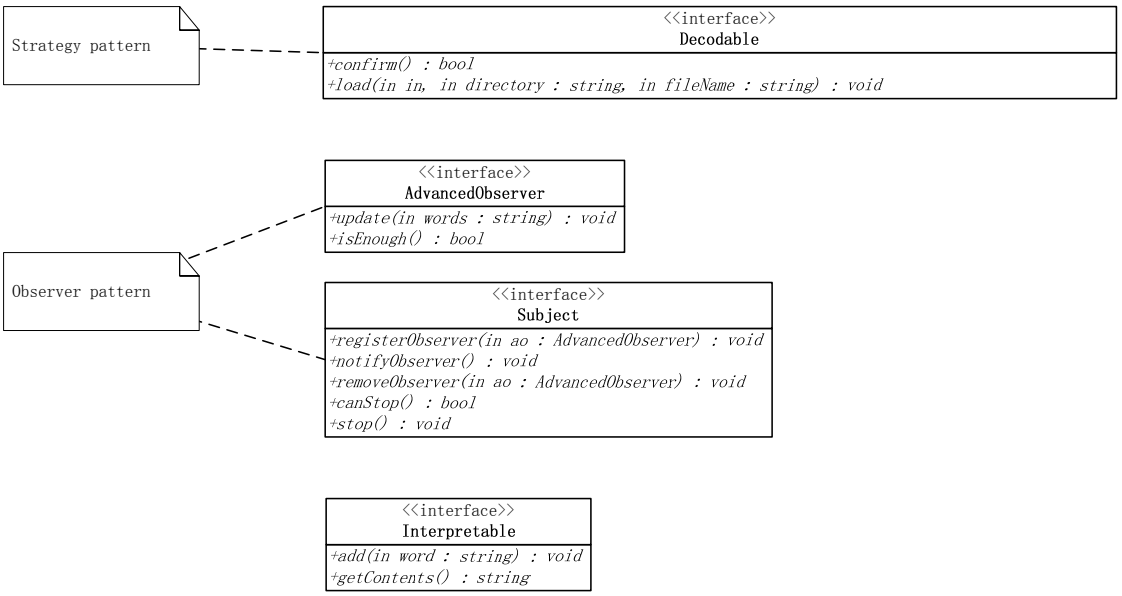
Package “simpleIndex”



Index is an abstract class provides template method “createIndex”, “toFile” for derived class.
The “f”`s type is File. “key”`s type is Comparable and “key” is an array. “contents” is also an array but the type is String.

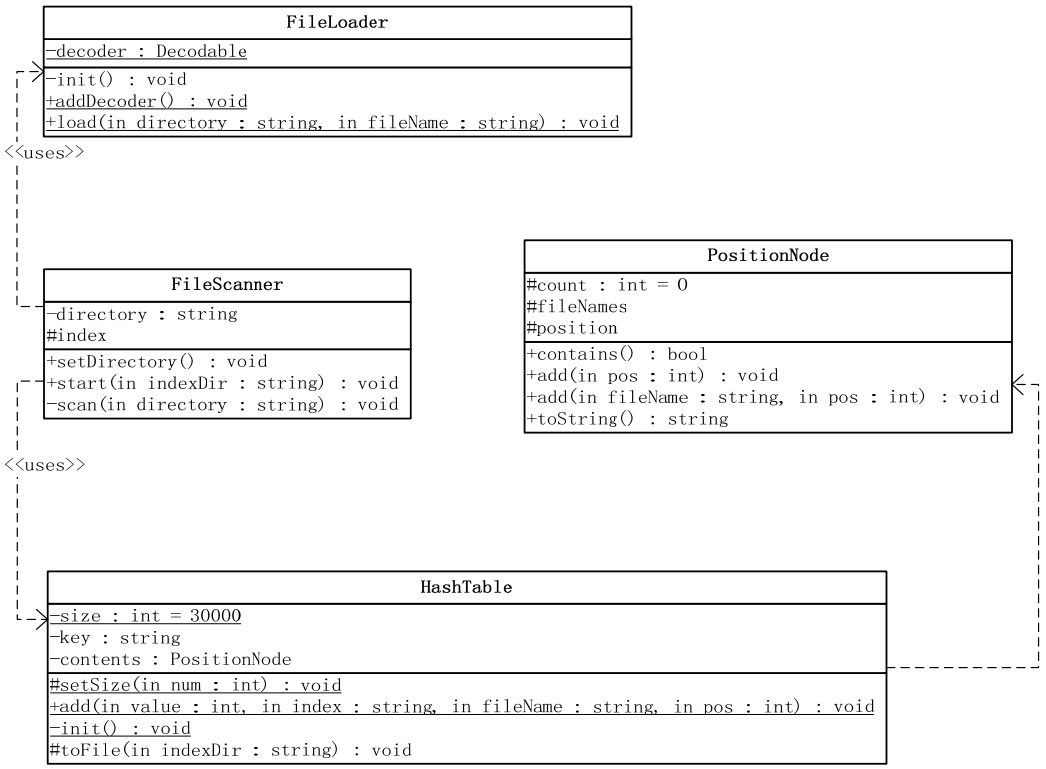
IndexSet is a vector of Index the public method will call the method of object it keeps.

Package “pattern”



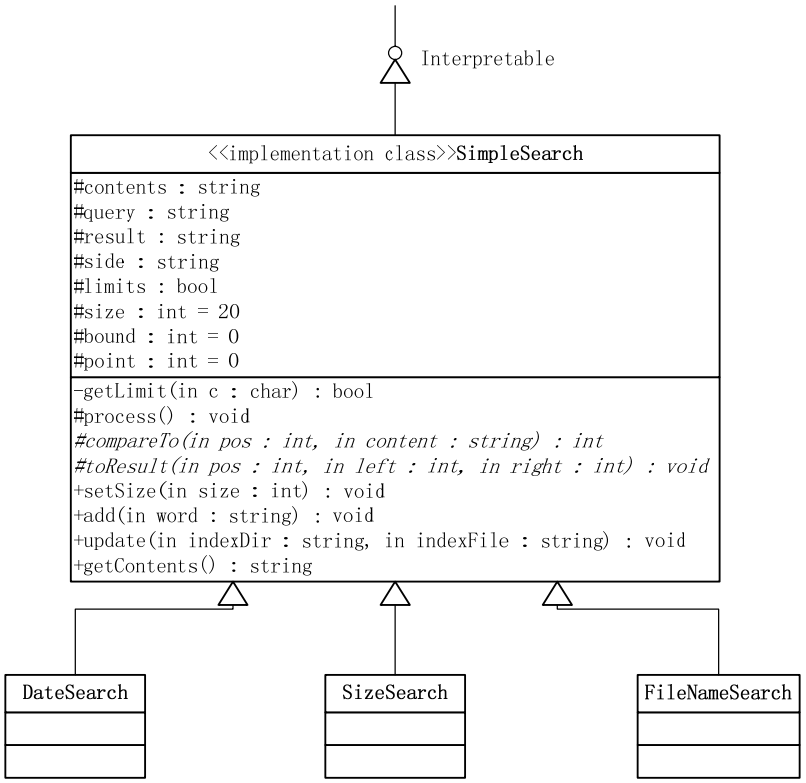
The interface “Decodable” support strategy pattern in package “decode”.
The interface “AdvancedObserver” and “Subject” will support observer pattern in package “searchWord”.
The interface “Interpretable” will let implementation class to be a vector keeps a series of elements of same type.

Package “fileManager”



“FileLoader” provide two static methods to maintain the “Decodable” list and execute reading files.
“HashTable” is a static structure providing a hash table.
“PositionNode” is a improve method to replace the “contains” method of “String”, and it will really raise the speed up.
“FileScanner” is a driver to start the index creating part.

Package “simpleSearch”



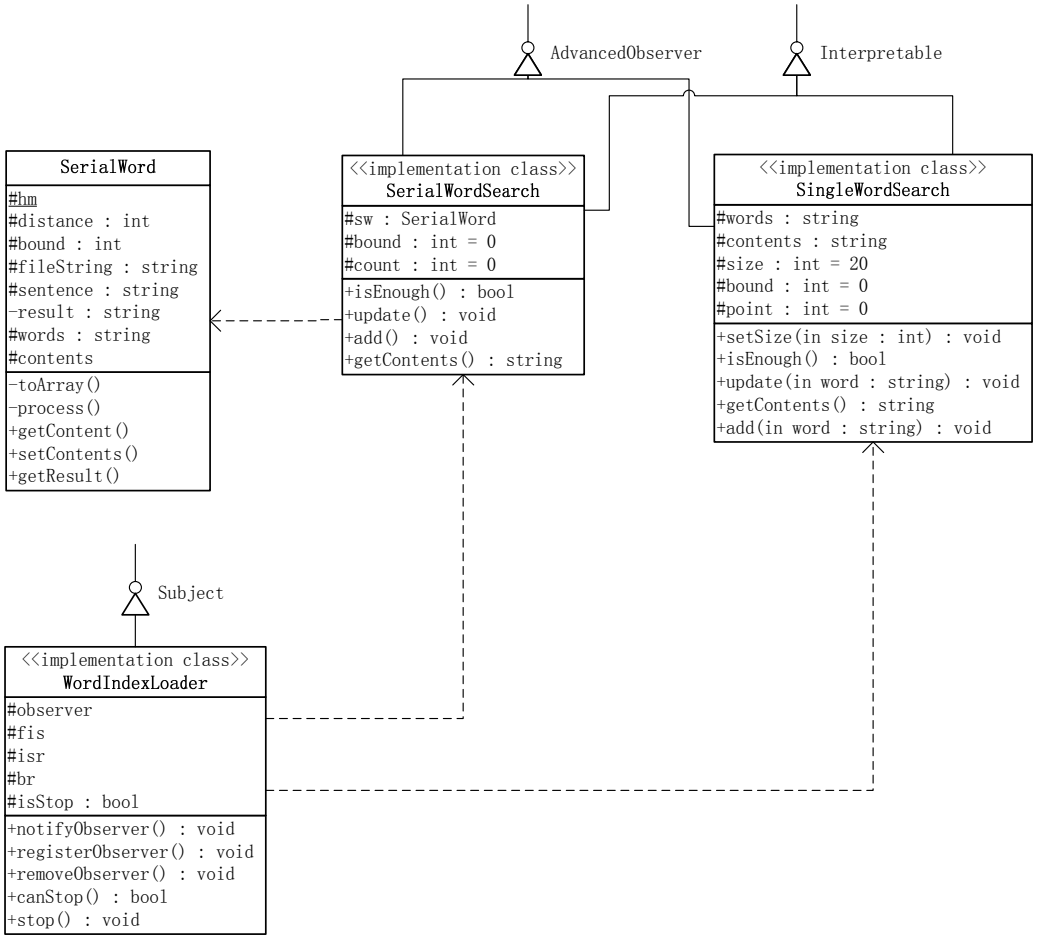
The “SimpleSearch” class implements the interface “Interpretable” and provide template method “getLimit”, ”process”, “add” , “update”, “getContents” for the derived class.

“DataSearch” is a vector of “modTime”

“SizeSearch” is a vector of “fileSize”

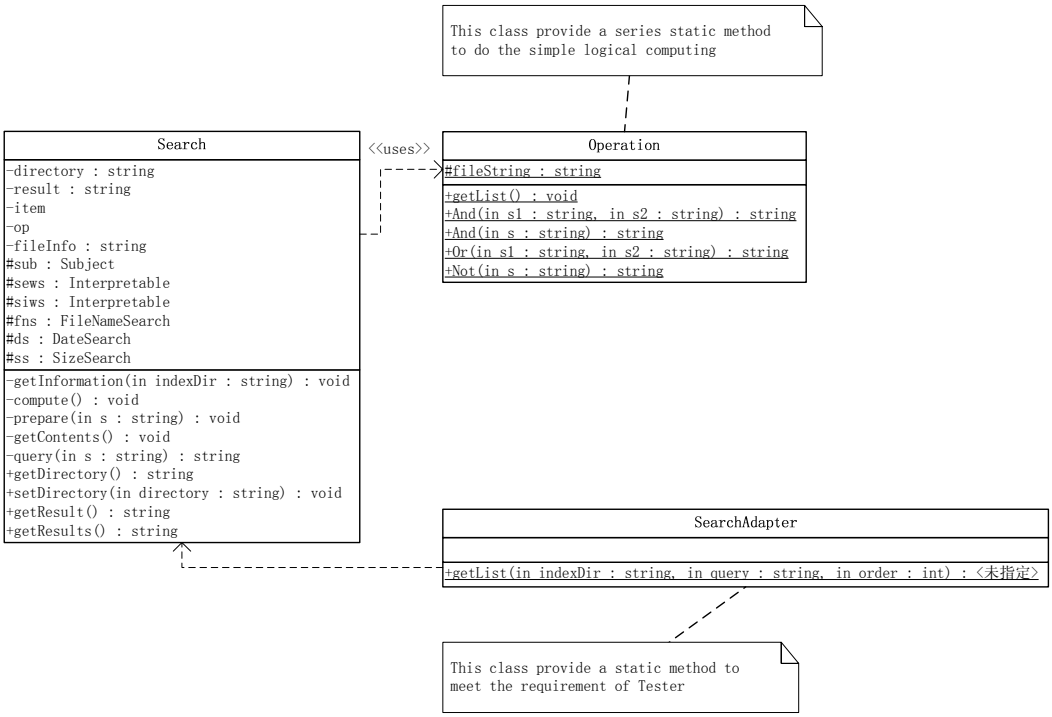
“FileNameSearch” is a vector of “fileName”

Package “searchWord”



The “SerialWordSearch” implements interface “AdvancedObserver” and “Interpretable” so it is a vector of “SerialWord”.
The “SingleWordSearch” implements interface “AdvancedObserver” and “Interpretable” and it is vector of “fileContents”.
The “SerialWord” provides proximity query, uses queue arithmetic.
The “WordIndexLoader” implements interface “Subject” and will pass the contents of main index to the two observers.

Package “indexSearch”



Class “Operation” provides a series of operations about logical computing.
“SearchAdapter” uses theory of adapter pattern to meet the new Tester.
Class “Search” is a combination of all kinds of element types.

3. Changing and Details

Changing:

The class “Search” is changed into a combination of all kinds of elements class, its old responsibility now gives to “SingleWordSearch” class.

The class “Analyzer” was renamed to “Operation”

The HashTable uses a new class “PositionNode” to save the main index.

The TxtDecoder and WordDecoder now have a parent class “Decoder”.

The Index structure now changes to “index” + “:” + {“fileN” + “ ” + “pos1 +… +posN”}

Implement Details:

Proximity query:
The proximity query is implemented by queue. For example, if want to search “A B”
First step: let temp := A AND B
Second step: put every file name in temp into a String queue
Third step: start a loop do these things:

1. Remove peek of queue
2. If new position meet condition, add new position to get a new String and add to the back of the queue
3. If no positions to be added, loop terminate.

Forth step: get elements left in the queue.

Create Index:

Use a new class “PositionNode” to keep index`s contents, it would save a lot of searching time.