2014

Harim Nasser 51

Tarek Nawara 35

Sherif Hamdy 31

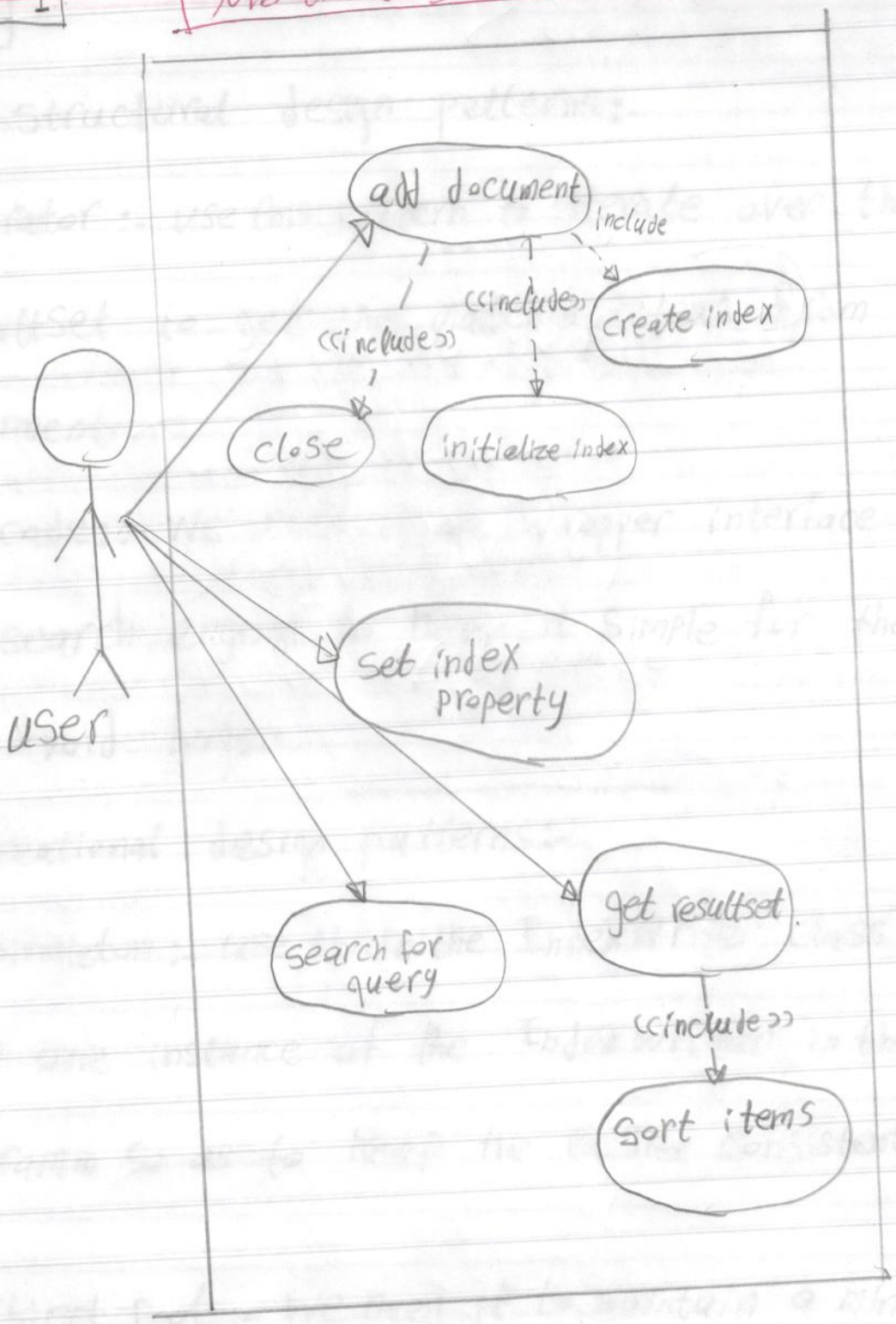
Moataz Ahmed 66

A

We have different types of users here such as: developer, admin & client

Mohamed Saad

Q1



usecase diagram

Q2

2 structural design patterns:-

① iterator :- use this pattern to iterate over the ResultSet to get the matching items from the document.

② facade:. We need it as wrapper interface for the search engine to keep it simple for the user to avoid bugs.

---

3 creational design patterns:-

① Singleton :- use it in the IndexWriter class to get only one instance of the IndexWriter in the whole programm so as to keep the B+ Tree consistent

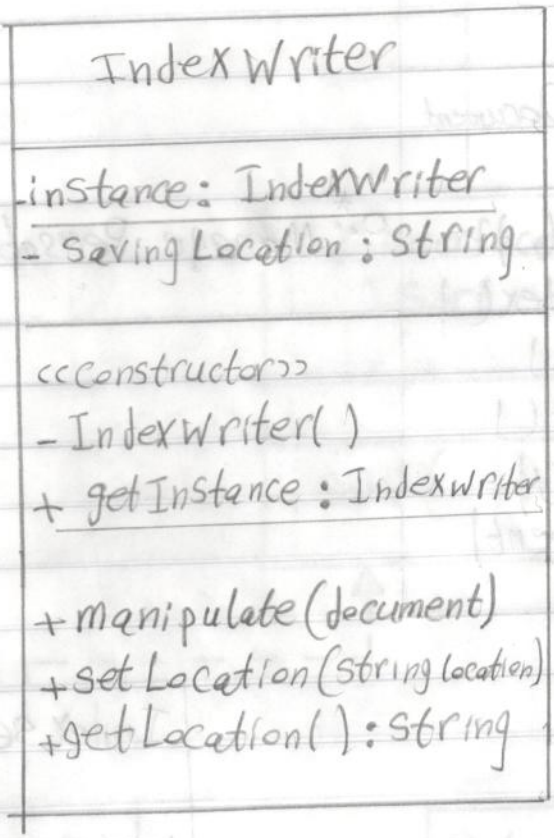
② Object pool :- We need it to maintain a minimum and a maximum number of instances of the IndexSearcher as it considered a costly object to create and it can be considered as a reusable object

③ Builder design pattern :- build the B_+ tree because it parses different text document.

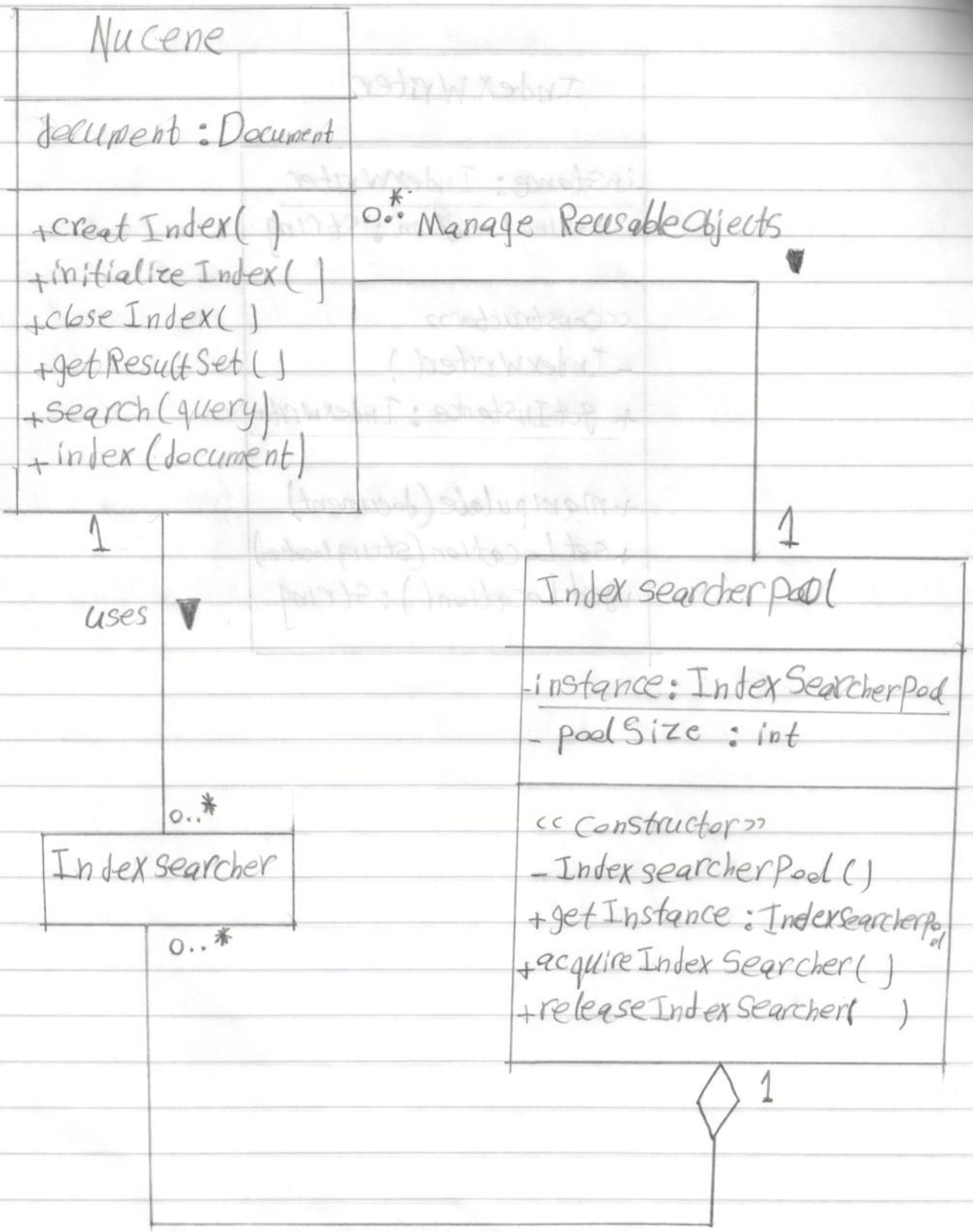
1- Behavioral Design Pattern :-

strategy desing pattern need it to implement the two different algorithms for the Sorting techniques to Sort the ResultSet items as the user can choose between the sorting algorithms at run time

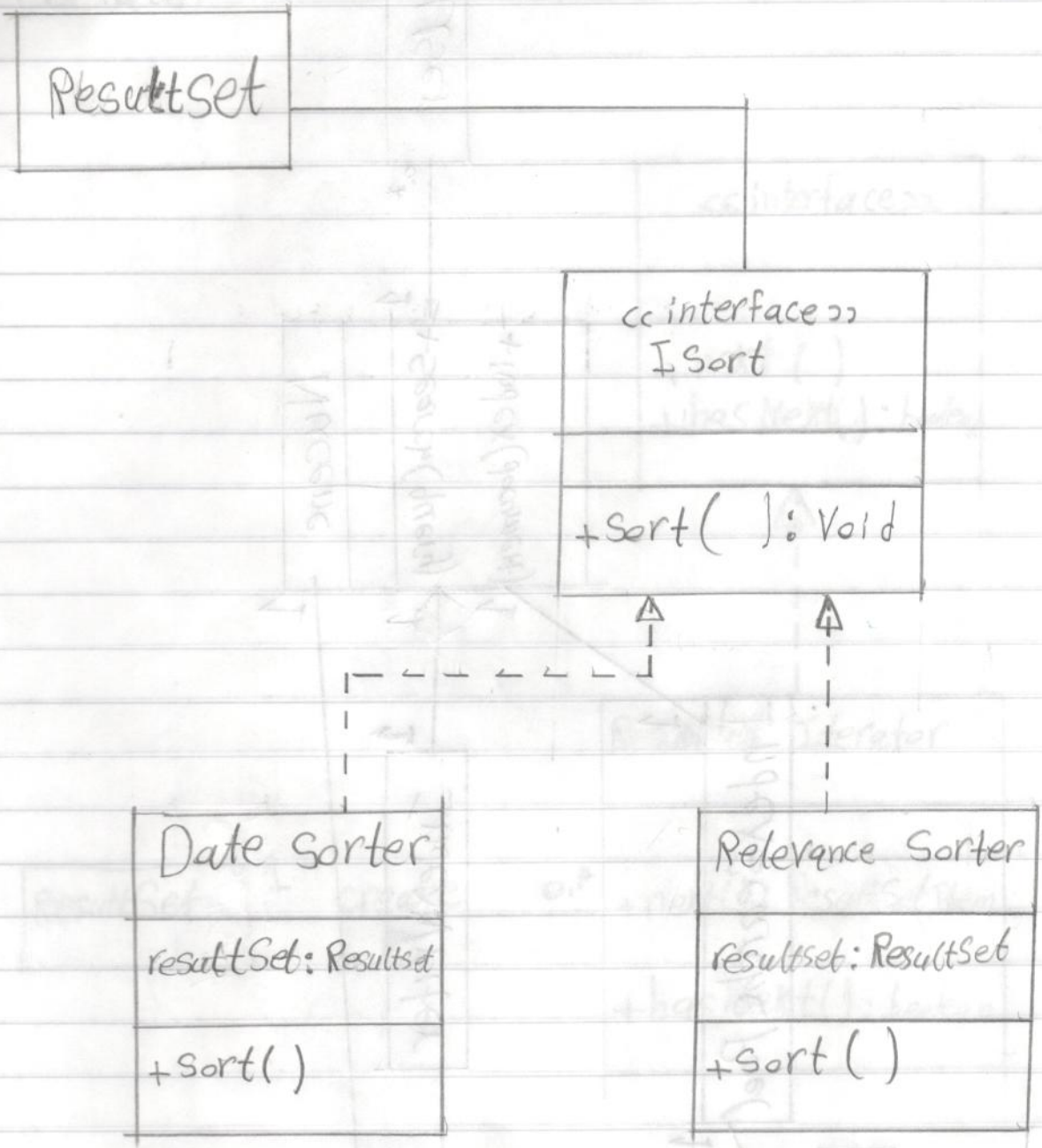
Singleton :-

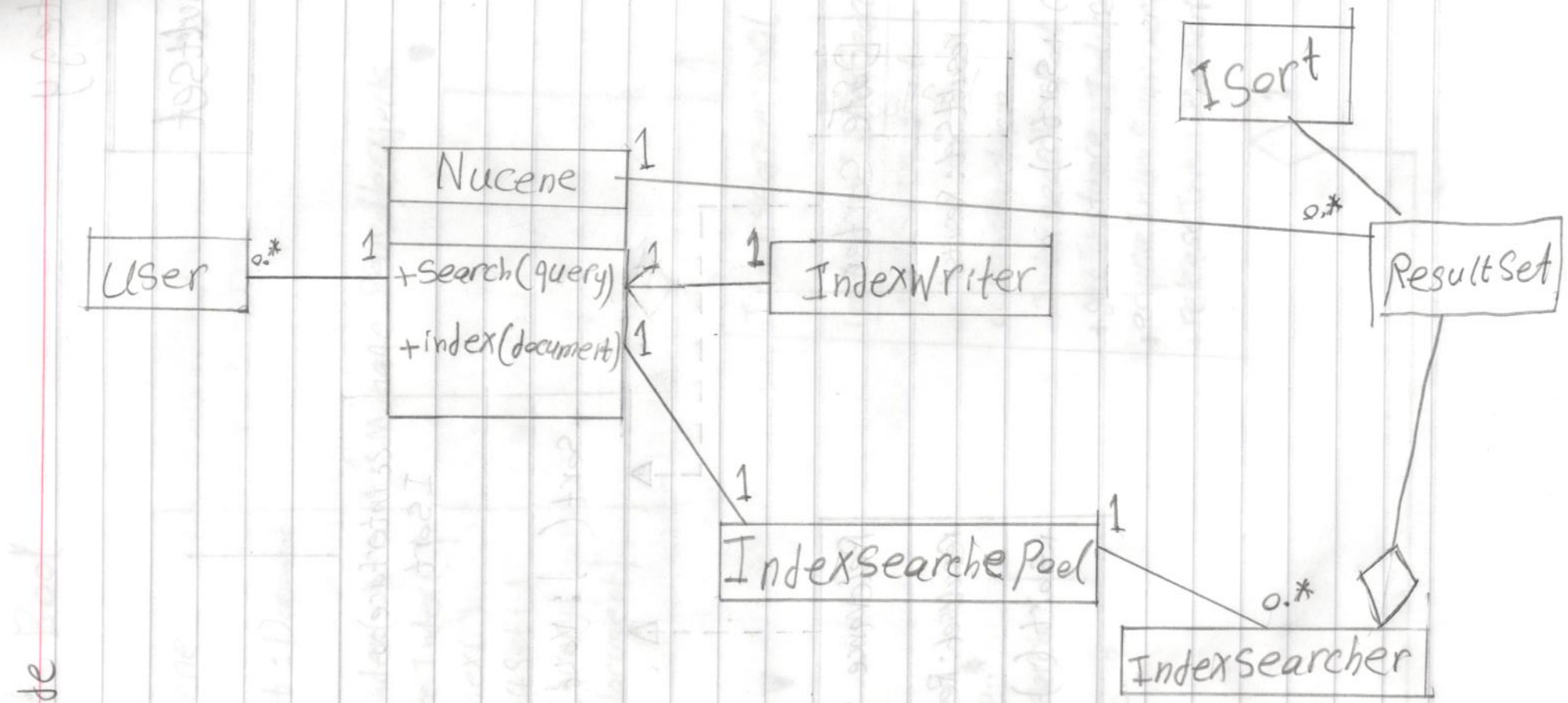

IndexWriter
-instance: IndexWriter
- SavingLocation : String
<<constructor>>
- IndexWriter( )
+ getInstance : Indexwriter
+ manipulate (document)
+ setLocation (String location)
+ getLocation( ) : String

Object Book


Nucene
document : Document
+creat Index( )
+initialize Index( )
+close Index( )
+get Result Set ( )
+search (query)
+index (document)
0..* Manage Reusable Objects
1
uses
1
Index searcher pool
-instance: Index Searcher Pool
-pool Size : int
<< constructor >>
- Index searcher Pool ( )
+get Instance : Index Searcher Pool
+acquire Index Searcher( )
+release Index Searcher( )
1
0..*
Index searcher
0..*

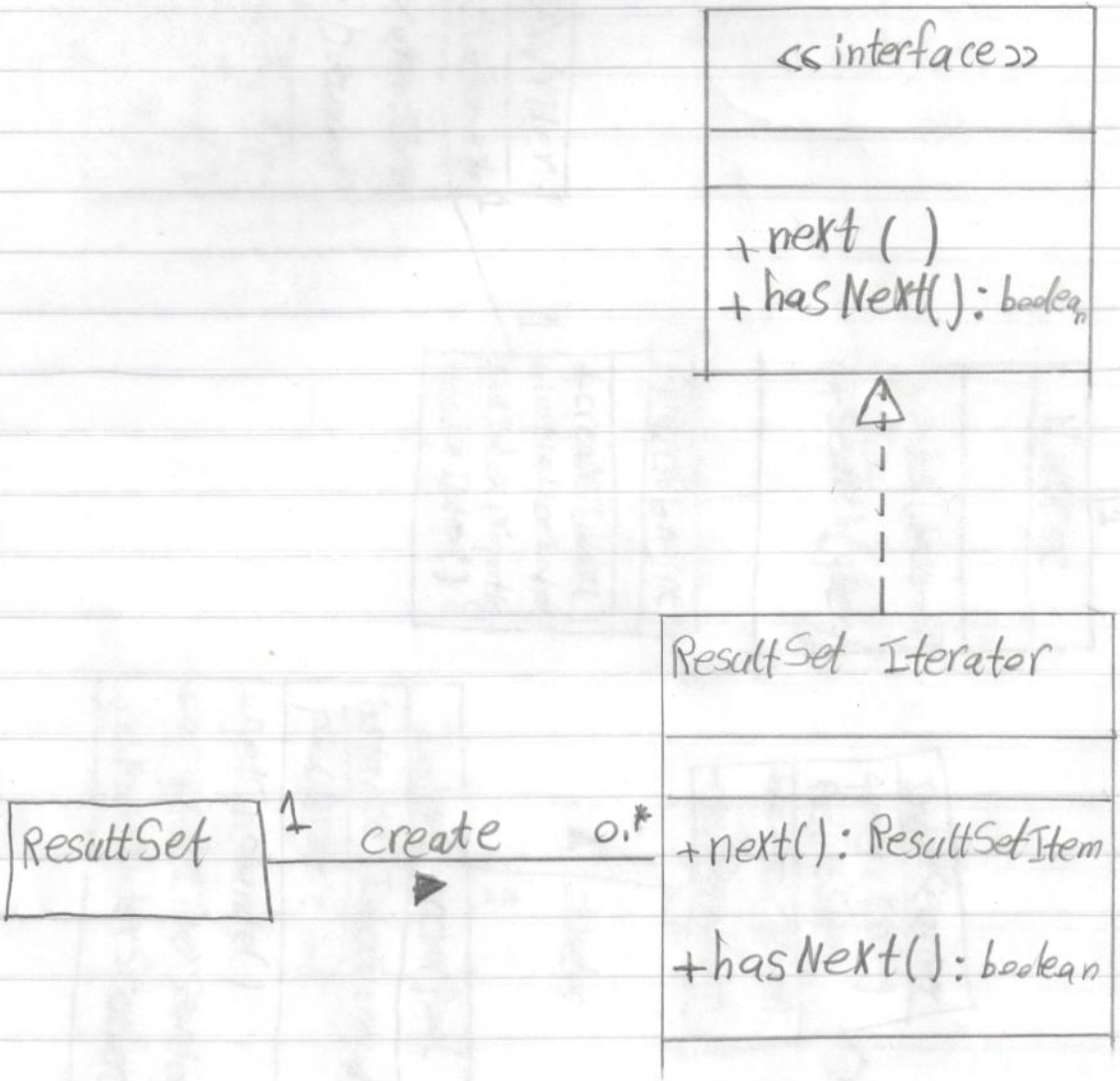
strategy

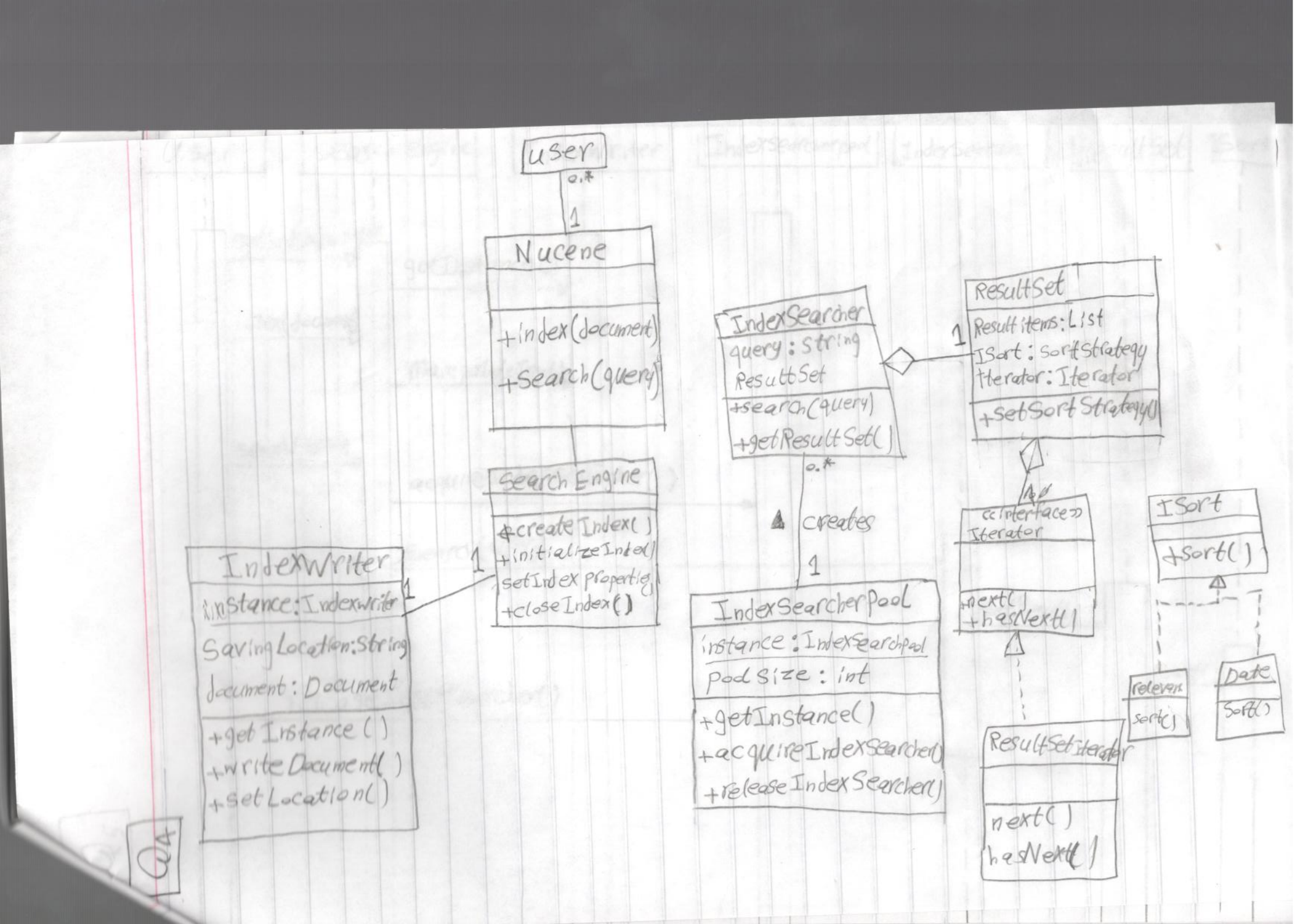

ResultSet
<<interface>>
ISort
+Sort( ): Void
Date Sorter
resultSet: ResultSet
+Sort( )
Relevance Sorter
resultSet: ResultSet
+Sort( )

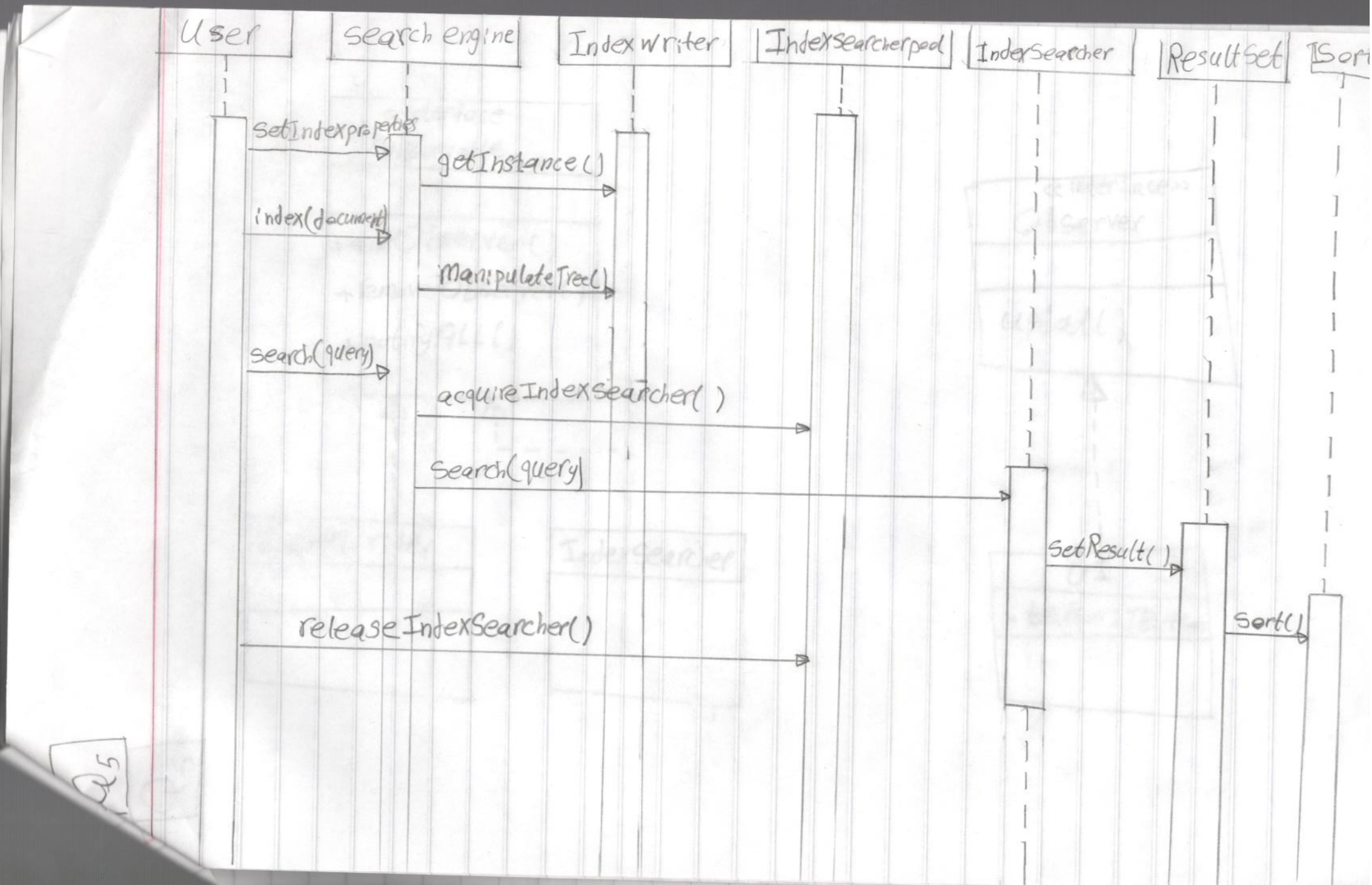

ISort
Nucene
1
0..*
User
0..*
1
+search(query)
+index(document)
1
1
1
IndexWriter
ResultSet
1
1
IndexsearchePool
1
0..*
Indexsearcher

Facade

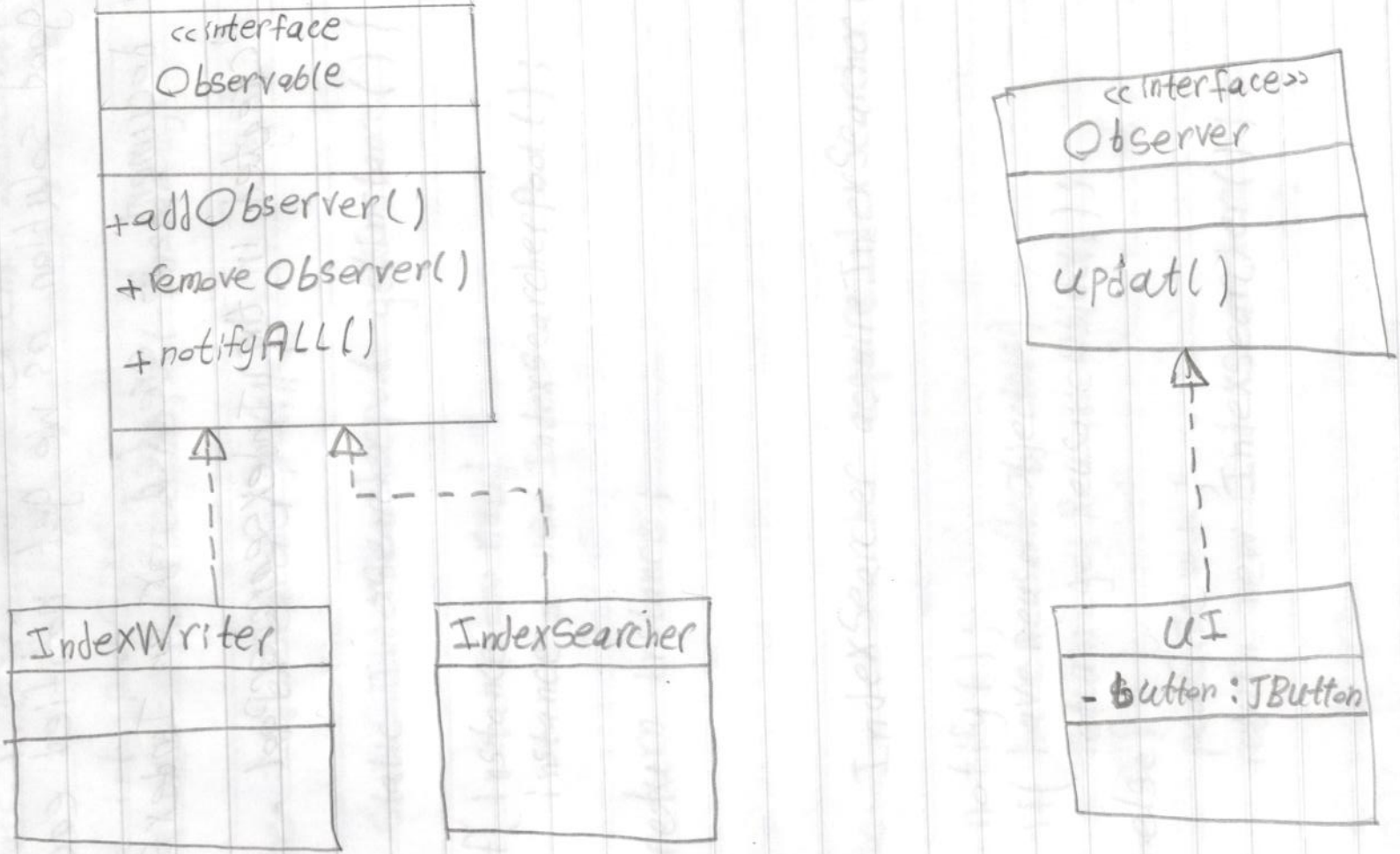
Iterator


<<interface>>
+next()
+hasNext(): boolean
ResultSet Iterator
+next(): ResultSetItem
+hasNext(): boolean
ResultSet
1
create
0..*


user
0..*
1
Nucene
+index(document)
+search(query)
SearchEngine
+createIndex()
+initializeIndex()
setIndexProperties()
+closeIndex()
1
1
IndexWriter
instance:IndexWriter
SavingLocation:String
document:Document
+getInstance()
+writeDocument()
+setLocation()
IndexSearcher
query:string
ResultSet
+search(query)
+getResultSet()
1
ResultSet
ResultItems:List
ISort:SortStrategy
Iterator:Iterator
+setSortStrategy()
0..*
creates
1
IndexSearcherPool
instance:IndexSearchPool
PodSize:int
+getInstance()
+acquireIndexSearcher()
+releaseIndexSearcher()
«Interface»
Iterator
+next()
+hasNext()
ResultSetIterator
next()
hasNext()
ISort
+Sort()
relevance
sort()
Date
Sort()


User
Search engine
IndexWriter
IndexSearcherpool
IndexSearcher
ResultSet
ISort
setIndexproperties
getInstance()
index(document)
manipulateTree()
search(query)
acquireIndexSearcher()
search(query)
setResult()
releaseIndexSearcher()
sort()

Q5


<<interface
Observable
+addObserver()
+removeObserver()
+notifyALL()
<<interface>>
Observer
update()
IndexWriter
IndexSearcher
UI
-button : JButton

Q5

Using the Observer desing pattern will be a good solution as we get notified each time a document is created or an IndexSearcher is created in the IndexsearcherPool.

Q7

```
public class IndexSearcherpool extends Observable {

    private int poolSize;
    private static IndexSearcherPool Instance;

    private IndexSearcherPool(){ }

    public static IndexSearcherpool getInstance(){

        if(instance == null)
            instance = new IndexSearcherPool();

        return instance;

    }

    public IndexSearcher acquireIndexSearcher(){

        notify();
        if(haveReusableObjects())
            return getReusableObject();
        else{
            poolSize++;
            return new IndexSearcher();

    }
```

```
public class IndexWriter extends Observable {
    // Singleton

    public void manipulateTree(Document document) {
        notify();
        //
    }
}
```

```
public class UI implements Observer {

    @Override
    public void update(observable obs, Object o){
        if (obs instanceof IndexSearcherPool) {
            if(obs.getPodSize() >= 99)
                button.setColor(Color.RED);
                return;
        } else
            if(obs.getDocumentsNumber >= Max-Val)
                button.setColor(Color.RED);
        }
                return;

    button.setColor(Color.Green);
```