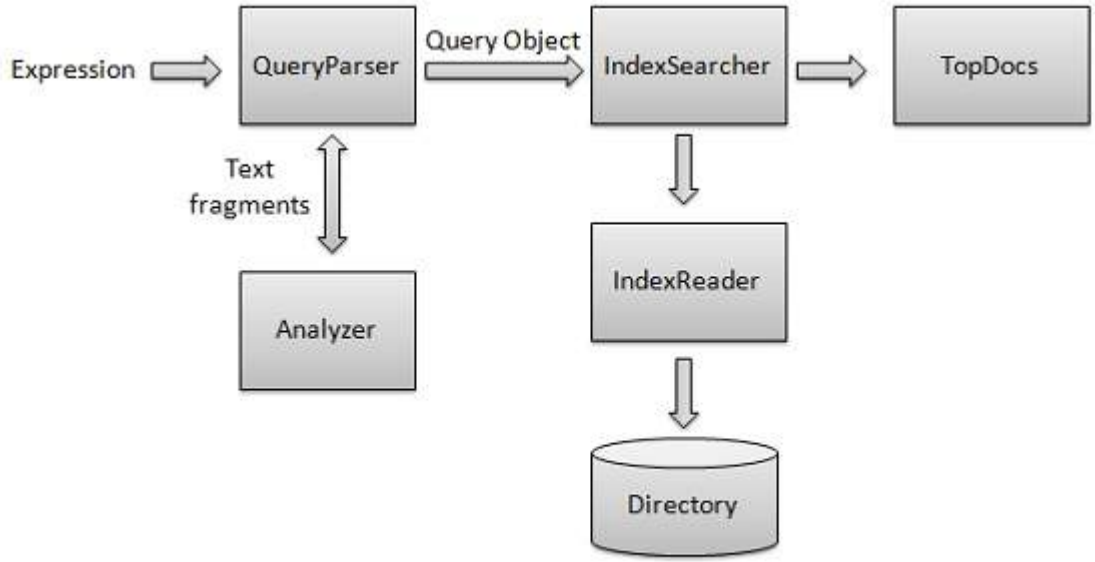
LUCENE - SEARCH OPERATION

Searching process is one of the core functionality provided by Lucene. Following diagram illustrates the searching process and use of classes. IndexSearcher is the most important and core component of the searching process.



Searching Process

We first create *Directory*s containing *indexes* and then pass it to *IndexSearcher* which opens the *Directory* using *IndexReader*. Then we create a *Query* with a *Term* and make a search using *IndexSearcher* by passing the *Query* to the searcher. *IndexSearcher* returns a *TopDocs* object which contains the search details along with document ID*s* of the *Document* which is the result of the search operation.

Now we'll show you a step by step process to get a kick start in understanding of indexing process using a basic example.

Create a QueryParser

- QueryParser class parses the user entered input into lucene understandable format query.
- Create object of QueryParser.
-
- Initialize the QueryParser object created with a standard analyzer having version information and index name on which this query is to run.

```
QueryParser queryParser;

public Searcher(String indexDirectoryPath) throws IOException{

   queryParser = new QueryParser(Version.LUCENE_36,
      LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
}
```

Create a IndexSearcher

- IndexSearcher class acts as a core component which searcher indexes created during indexing process.
- Create object of IndexSearcher.

-
- Create a lucene directory which should point to location where indexes are to be stored.
-
- Initialize the IndexSearcher object created with the index directory

```
IndexSearcher indexSearcher;

public Searcher(String indexDirectoryPath) throws IOException{
   Directory indexDirectory =
      FSDirectory.open(new File(indexDirectoryPath));
   indexSearcher = new IndexSearcher(indexDirectory);
}
```

Make search

- To start search, create a Query object by parsing search expression through QueryParser.
- Make search by calling IndexSearcher.search method.
-

```
Query query;

public TopDocs search( String searchQuery) throws IOException, ParseException{
   query = queryParser.parse(searchQuery);
   return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
}
```

Get the document

```
public Document getDocument(ScoreDoc scoreDoc)
   throws CorruptIndexException, IOException{
   return indexSearcher.doc(scoreDoc.doc);
}
```

Close IndexSearcher

```
public void close() throws IOException{
   indexSearcher.close();
}
```

Example Application

Let us create a test Lucene application to test searching process.

Step	Description
1	Create a project with a name *LuceneFirstApplication* under a package *com.tutorialspoint.lucene* as explained in the *Lucene - First Application* chapter. You can also use the project created in *Lucene - First Application* chapter as such for this chapter to understand searching process.
2	Create *LuceneConstants.java*,*TextFileFilter.java* and *Searcher.java* as explained in the *Lucene - First Application* chapter. Keep rest of the files unchanged.
3	Create *LuceneTester.java* as mentioned below.
4	Clean and Build the application to make sure business logic is working as per the requirements.

LuceneConstants.java

This class is used to provide various constants to be used across the sample application.

```
package com.tutorialspoint.lucene;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10;
}
```

TextFileFilter.java

This class is used as a .txt file filter.

```
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

Searcher.java

This class is used to read the indexes made on raw data and searches data using lucene library.

```
package com.tutorialspoint.lucene;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Searcher {

   IndexSearcher indexSearcher;
   QueryParser queryParser;
   Query query;

   public Searcher(String indexDirectoryPath) throws IOException{
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));
      indexSearcher = new IndexSearcher(indexDirectory);
      queryParser = new QueryParser(Version.LUCENE_36,
         LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
   }
```

```
   public TopDocs search( String searchQuery)
      throws IOException, ParseException{
      query = queryParser.parse(searchQuery);
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
   }

   public Document getDocument(ScoreDoc scoreDoc)
      throws CorruptIndexException, IOException{
     return indexSearcher.doc(scoreDoc.doc);
   }

   public void close() throws IOException{
      indexSearcher.close();
   }
}
```

LuceneTester.java

This class is used to test the searching capability of lucene library.

```
package com.tutorialspoint.lucene;

import java.io.IOException;

import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;

public class LuceneTester {

   String indexDir = "E:\\Lucene\\Index";
   String dataDir = "E:\\Lucene\\Data";
   Searcher searcher;

   public static void main(String[] args) {
      LuceneTester tester;
      try {
         tester = new LuceneTester();
         tester.search("Mohan");
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }

   private void search(String searchQuery) throws IOException, ParseException{
      searcher = new Searcher(indexDir);
      long startTime = System.currentTimeMillis();
      TopDocs hits = searcher.search(searchQuery);
      long endTime = System.currentTimeMillis();

      System.out.println(hits.totalHits +
         " documents found. Time :" + (endTime - startTime) +" ms");
      for(ScoreDoc scoreDoc : hits.scoreDocs) {
         Document doc = searcher.getDocument(scoreDoc);
         System.out.println("File: "+ doc.get(LuceneConstants.FILE_PATH));
      }
      searcher.close();
   }
}
```

Data & Index directory creation

I've used 10 text files named from record1.txt to record10.txt containing simply names and other details of the students and put them in the directory **E:\Lucene\Data.** Test Data. An index directory path should be created as **E:\Lucene\Index**. After running the indexing program during

chapter *Lucene - Indexing Process*, you can see the list of index files created in that folder.

Running the Program:

Once you are done with creating source, creating the raw data, data directory, index directory and indexes, you are ready for this step which is compiling and running your program. To do this, Keep LuceneTester.Java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **LuceneTester** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console:

```
1 documents found. Time :29 ms
File: F:\Lucene\Data\record4.txt
```