LUCENE - TERMQUERY

Introduction

TermQuery is the most commonly used query object and is the foundation of many complex queries that lucene can make use of. TermQuery is normally used to retrieve documents based on the key which is case sensitive.

Class declaration

Following is the declaration for **org.apache.lucene.search.TermQuery** class:

```
public class TermQuery
   extends Query
```

Class constructors

S.N.	Constructor & Description
1	**TermQuery***Termt* Constructs a query for the term t.

Class methods

S.N.	Method & Description
1	**void addDocument***Documentdoc* Adds a document to this index.
2	**Weight createWeight***Searchersearcher* Expert: Constructs an appropriate Weight implementation for this query.
3	**boolean equals***Objecto* Returns true iff o is equal to this.
4	**void extractTerms***Set < Term > terms* Expert: adds all terms occurring in this query to the terms set.
5	**Term getTerm** Returns the term of this query.
6	

	int hashCode Returns a hash code value for this object.
7	**String toString*Stringfield*** Prints a user-readable version of this query.

Methods inherited

This class inherits methods from the following classes:

- org.apache.lucene.search.Query
- java.lang.Object

Usage

```
private void searchUsingTermQuery(
   String searchQuery)throws IOException, ParseException{
   searcher = new Searcher(indexDir);
   long startTime = System.currentTimeMillis();
   //create a term to search file name
   Term term = new Term(LuceneConstants.FILE_NAME, searchQuery);
   //create the term query object
   Query query = new TermQuery(term);
   //do the search
   TopDocs hits = searcher.search(query);
   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();
}
```

Example Application

Let us create a test Lucene application to test search using TermQuery.

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* 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;
}
```

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 TopDocs search(Query query) throws IOException, ParseException{
      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.

```java
package com.tutorialspoint.lucene;

import java.io.IOException;

import org.apache.lucene.document.Document;
import org.apache.lucene.index.Term;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TermQuery;
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.searchUsingTermQuery("record4.txt");
      } catch (IOException e) {
         e.printStackTrace();
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }

   private void searchUsingTermQuery(
      String searchQuery)throws IOException, ParseException{
      searcher = new Searcher(indexDir);
      long startTime = System.currentTimeMillis();
      //create a term to search file name
      Term term = new Term(LuceneConstants.FILE_NAME, searchQuery);
      //create the term query object
      Query query = new TermQuery(term);
      //do the search
      TopDocs hits = searcher.search(query);
      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 :13 ms
File: E:\Lucene\Data\record4.txt
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js