Source code of IR_P02

1.StartInformationRetrieval.java:

```
	package com.irpt2.main;


import java.io.File;
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;
import com.irpt2.manager.SearchManager;
import com.irpt2.util.Utils;
import com.irpt2.constants.Constants;
import com.irpt2.manager.IndexingManager;
import com.irpt2.crawler.WebCrawler;




public class StartInformationRetrieval {
	static Utils util = new Utils();



	 /* Takes the input parameters and passes them onto the other methods for
execution
	  * @param Seed url
	  * @param crawl depth
  * @param index folder path
  * @param query*/
public static void main(String[] args) throws Exception {

        //make sure all required parameters were entered
        if (args.length < 4) {
                System.out.println(Constants.HELP_MESSAGE);
        } else {

                //start setting data received from command line
        String urlBeforenormalization= args[0].trim();
        String url= util.normalizeUrl(urlBeforenormalization);
        int crawldepth = Integer.parseInt(args[1]);
        String indexPath = args[2].trim();
        File theinDir = new File(indexPath);
          // if the directory does not exist, create it
        if (!theinDir.exists()) {
           try{
              theinDir.mkdir();
           }
           catch(SecurityException se){
              //handle it
           }
        } else {
                for(File file: theinDir.listFiles())
                   if (!file.isDirectory())
                      file.delete();
        }
        String rankingModel = Constants.RANKING_MODEL_VS;
        String query = null;
        String filePath="C:\\Users\\Public\\Documents\\docsfromweb";
        File theDir = new File(filePath);
          // if the directory does not exist, create it
        if (!theDir.exists()) {
           try{
              theDir.mkdir();
           }
           catch(SecurityException se){
              //handle it
           }

        } else {
                for(File file: theDir.listFiles())
                   if (!file.isDirectory())
                      file.delete();
        }
                StringBuilder stringBuilder = new StringBuilder();
                for (int i = 3; i < args.length; i++) {
                          stringBuilder.append(args[i] + " "); //if there is more than one
        word in the query, append all of them to make them a single string , eg: "hi, how are you"
        should be considered as a string instead of 4 diff strings
                }
                query = stringBuilder.toString().trim();
                //end setting data received from command line
                          System.out.println(Constants.WELCOME_MESSAGE);//exeute
        program if valid inputs were recieved
                          crawlFirst(url,crawldepth,filePath,indexPath);
                          executeTask(filePath, indexPath, rankingModel, query);
        }
```

```
        }

        /* Takes the url and depth and crawls the url according to the depth mentioned,
while simultaneously collecting their links, adding them to
         * pages.txt, creates a file that has the url's body content
         * @param url
         * @param depth
         * @param filepath
         * @param indexpath
         * */

    private static Set<String> pagesVisited = new HashSet<String>();
    static Map<Integer,LinkedList<String>> urllinksmap= new HashMap<>();
    public static void crawlFirst(String url,int depth, String filepath, String indexpath) throws
IOException  {
        int i=0;
        int depthcount=1;
        int seintheloop=0;
      int loopcount=0;
        urllinksmap.put(depthcount, new LinkedList<String>());
        if(depth==0) {
                WebCrawler infocrawler = new WebCrawler();
          infocrawler.crawl(url);
           pagesVisited.add(url);
            String content = infocrawler.TextInTheWebpage();
             if (content == null) {
                System.exit(0);
                }
         String title= infocrawler.Title();
            if(title==null) {System.exit(0);}
```

```
                util.addtopagestxt(indexpath, url, 0);
                util.FileCreator(url,content,title,filepath,i);
        }
          else{
                  while (depthcount<=depth)
        {
          String currentUrl;
          WebCrawler infocrawler = new WebCrawler();
          if(urllinksmap.get(depthcount).isEmpty())
          {
            if(seintheloop==1) //this is to prevent the crash in case of the seed url being the
error
            { System.exit(0);
            }
            currentUrl = url;
            pagesVisited.add(url);
            seintheloop++;
          }
          else
          {
            currentUrl = nextUrl(urllinksmap.get(depthcount));
          }
          infocrawler.crawl(currentUrl); // Lots of stuff happening here. Look at the crawl
method in webcrawler
         if(currentUrl==url) {util.addtopagestxt(indexpath, currentUrl, 0);}
         else {util.addtopagestxt(indexpath, currentUrl, depthcount);}


         if(urllinksmap.get(depthcount).size()==0) { if(loopcount<1) {
           urllinksmap.get(depthcount).addAll(infocrawler.getLinks());}
         else  {urllinksmap.get(depthcount+1).addAll(infocrawler.getLinks());} //REMEMBER
THIS
           loopcount++;
           } //depthcount++; no chance
          else {
           if(!urllinksmap.containsKey(depthcount+1)) //check if that doesn't exist
           {urllinksmap.put(depthcount+1, new LinkedList<String>());}
          urllinksmap.get(depthcount+1).addAll(infocrawler.getLinks());
         }
         if(urllinksmap.get(depthcount).size()==0) {if(loopcount>1){depthcount++;}}
          String content = infocrawler.TextInTheWebpage();
          if (content == null) {
           continue;
          }
          String title=infocrawler.Title();
          if (title == null) {
           continue;
          }
          util.FileCreator(currentUrl,content,title,filepath,i);
          i++;
        }
    }
        System.out.println("\n**Done** Visited " + pagesVisited.size() + " web page(s)");
    }
    /* Creates objects of indexing and search  managers and calls their functions for indexing
and searching the query respectively
     * @param filepath
     * @param indexpath
     * @param rankingmodel
     * @param query*/
    private static void executeTask(String filePath, String indexPath, String rankingModel,
String query) throws Exception {


                        System.out.println(Constants.INDEXING_STARTED_MSG);
                        util.insertNewLine();
                        IndexingManager indexingManager = new IndexingManager();
                        indexingManager.startIndexing(filePath,indexPath,rankingModel);
                        System.out.println(Constants.INDEXING_COMPLETED_MSG);
                        util.insertNewLine();
                        Path path = Paths.get(indexPath);
                        //IndexingManager.getParsedDocs(path);
                        System.out.println(Constants.SEARCH_QUERY_MSG+query);
                        SearchManager searchManager = new SearchManager();
                        searchManager.initiateSearch(path,query,rankingModel);
                        util.insertNewLine();
                        System.out.println(Constants.EXITING);


            }
    /* returns the nexturl taht has to be crawled after checking whether it has already been
crawled
     * @param LINKEDLIST x*/
              private static String nextUrl(LinkedList<String> x)
               {
                   String nextUrl;
                   do
                   {
                       if(x.isEmpty()) {return "";}
                      nextUrl=util.normalizeUrl(x.remove(0));
                   } while(pagesVisited.contains(nextUrl));
                   pagesVisited.add(nextUrl);
```

```
                return nextUrl;
            }
}
```

2.WebCrawler.java:

```
package com.irpt2.crawler;

import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

import org.jsoup.Connection;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class WebCrawler {

   private List<String> links = new LinkedList<String>(); // Just a list of URLs
   private Document htmlDocument; // This is our web page

   /*Start crawling the url and collect all the links
    * @param url
     */
   private static final String USER_AGENT =
          "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko)
Chrome/13.0.782.112 Safari/535.1";
   public void crawl(String url)
   {
      try
      {
          if (url == "") // This is to avoid the error
              return;
          Connection connection = Jsoup.connect(url).header("Accept-Encoding", "gzip,
deflate").userAgent(USER_AGENT).maxBodySize(0)
              .timeout(600000);
          Document htmlDocument = connection.get();
          this.htmlDocument = htmlDocument;
          if(connection.response().statusCode() == 200) // 200 is the HTTP OK status code
indicating that everything is great.
          {
          System.out.println("Received web page at " + url);
          }
          if(!connection.response().contentType().contains("text/html"))
          {
              System.out.println("**Failure** Retrieved something other than HTML");
              Connection connection1 =
Jsoup.connect("http://www.daenotes.com/").userAgent(USER_AGENT);
          this.htmlDocument=connection1.get();//this is to handle non-html documents
          }
          Elements linksOnPage = htmlDocument.select("a[href]");
          System.out.println("Found (" + linksOnPage.size() + ") links");
          for(Element link : linksOnPage)
          {
              this.links.add(link.absUrl("href"));
          }
      }
      catch(IOException ioe)
      {
          // We were not successful in our HTTP request
          System.out.println("Error in out HTTP request " + ioe);
      }
      catch (Exception e) {
          System.out.println("Error " + e);
      }
  }
  /* Extracts the body text from the Html Document*/
  public String TextInTheWebpage()
  {
      if(this.htmlDocument == null)
      {
          System.out.println("ERROR! Call crawl() before performing analysis on the
document");
      return null;}
      String bodyText = this.htmlDocument.body().text();
      return bodyText.toLowerCase();
  }

  /* Extracts the title from the Html Document*/
  public String Title()
  {

           if(this.htmlDocument == null)
       {
          System.out.println("ERROR! Call crawl() before performing analysis on the
document");
       }
       String Title = this.htmlDocument.title();
       return Title;
  }
  /* Return the links found on the url */
  public List<String> getLinks()
  {
      return this.links;
  }

}
```

3.IndexingManager.java:

```
package com.irpt2.manager;


import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.nio.file.FileVisitResult;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.attribute.BasicFileAttributes;
import java.nio.file.attribute.UserDefinedFileAttributeView;


import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.en.EnglishAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.StringField;
import org.apache.lucene.document.TextField;
import org.apache.lucene.index.DirectoryReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;
import org.apache.lucene.index.Term;
import org.apache.lucene.index.IndexWriterConfig.OpenMode;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;

import com.irpt2.constants.Constants;
import com.irpt2.util.Utils;

/**
 * Handles indexing of files
 *
 */
public class IndexingManager {

	/**
	 * Start indexing of documents in the provided document path
	 *
	 * @param folderPath
	 * @param indexPath
	 * @param rankingModel
	 */
	public void startIndexing(String folderPath, String indexPath, String rankingModel) {
		final Path docPath = Paths.get(folderPath);

		Path indexFilePath = Paths.get(indexPath);

		if (!Files.isReadable(docPath)) {
			System.out.println(Constants.UNABLE_TO_READ_FILE +
docPath.toAbsolutePath());
			System.exit(0);
		}

		try {
			System.out.println(Constants.INDEXING_MSG + indexFilePath);

			Directory directory;

			/**
			 * EnglishAnalyzer implements PorterStemmer Algorithm using
			 * PorterStemFilter
			 */
			Analyzer analyzer = new EnglishAnalyzer();

			IndexWriterConfig indexWriterConfig = new
IndexWriterConfig(analyzer);
			/*
			 * Creating index in directory
			 */
			if (Files.notExists(indexFilePath)) {
				// Create a new index in the directory, removing any
previously
```

```
                // indexed documents:
                indexWriterConfig.setOpenMode(OpenMode.CREATE);
                directory = FSDirectory.open(indexFilePath);
            } else {
                // Add new documents to an existing index:

indexWriterConfig.setOpenMode(OpenMode.CREATE_OR_APPEND);
                directory = FSDirectory.open(indexFilePath);
            }
            Utils util = new Utils();
            indexWriterConfig.setSimilarity(util.getSimilarity(rankingModel));
            IndexWriter indexWriter = new IndexWriter(directory,
indexWriterConfig);
            indexFiles(indexWriter, docPath);
            indexWriter.close();
            IndexReader indexReader =
DirectoryReader.open(FSDirectory.open(indexFilePath));
            System.out.println("Total indexed file: " + indexReader.numDocs());
            indexReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    /**
     * Indexes the given file using the given writer, or if a directory is
     * given, recurses over files and directories found under the given
```

```
	 * directory.
	 *
	 *
	 * @param writer
	 *          Creates/updates indexes during the indexing process.
	 * @param path
	 *          The path of files to be indexed.
	 * @param numofpv
	 * @throws IOException
	 */
	private void indexFiles(final IndexWriter indexWriter, Path path) throws IOException
{

		if (Files.isDirectory(path)) {
			Files.walkFileTree(path, new SimpleFileVisitor<Path>() {
				@Override
				public FileVisitResult visitFile(Path filePath, BasicFileAttributes
fileAttr) throws IOException {
					try {
						if (Files.isReadable(filePath)) {
							indexFile(indexWriter, filePath);
						} else {

System.out.println(Constants.UNABLE_TO_READ_FILE + filePath.toAbsolutePath());
						}
					} catch (IOException e) {
						e.printStackTrace();
					} catch (Exception e) {
						e.printStackTrace();
					}
```

```
                                        return FileVisitResult.CONTINUE;
                                }
                        });
                } else {
                        if (Files.isReadable(path)) {
                                indexFile(indexWriter, path);
                        } else {
                                System.out.println(Constants.UNABLE_TO_READ_FILE +
path.toAbsolutePath());
                        }
                }
        }

        /** Index each document
         * @param i */
        private void indexFile(IndexWriter indexWriter, Path filePath) throws IOException {
                try (InputStream stream = Files.newInputStream(filePath)) {
                        // document represents a virtual document with Fields
                        Document document = new Document();

                        // Field represents the key value pair relationship where a key is
                        // used to identify the value to be indexed.

                        // Create fields in the document.
                        Field filePathData = new StringField(Constants.FIELD_PATH,
filePath.toString(), Field.Store.YES);
                        Field content = new TextField(Constants.FIELD_CONTENT,new
BufferedReader(new InputStreamReader(stream, StandardCharsets.UTF_8)));

                         final UserDefinedFileAttributeView view =
Files.getFileAttributeView(filePath, UserDefinedFileAttributeView.class);
```

```
                              Utils util= new Utils();
                       Field url = new StringField("url",util.getUserDefinedAttribute(view, "url"),
Field.Store.YES);
                       Field title = new StringField("title",util.getUserDefinedAttribute(view,
"title"), Field.Store.YES);
                       document.add(title);
                       document.add(url);//}
                            // Adding created fields to the document.
                            document.add(filePathData);
                            document.add(content);


                            if (indexWriter.getConfig().getOpenMode() == OpenMode.CREATE) {
                                      // New index, adding new document:
                                      System.out.println(Constants.ADD_INDEX + filePath);
                                      indexWriter.addDocument(document);
                            } else if (indexWriter.getConfig().getOpenMode() ==
OpenMode.CREATE_OR_APPEND) {
                                      System.out.println(Constants.UPDATE_INDEX + filePath);
                                      indexWriter.updateDocument(new
Term(Constants.FIELD_PATH, filePath.toString()), document);// Delete
                            }

                   } catch (Exception e) {
                            e.printStackTrace();
                   }
         }
}
```

<u>4.SearchManager.java:</u>

```
package com.irpt2.manager;
```

```
import java.io.IOException;
import java.nio.file.Path;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.en.EnglishAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.DirectoryReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.queryparser.classic.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.FSDirectory;
import com.irpt2.constants.Constants;
import com.irpt2.util.Utils;


/**
 * Handles searching of entered query.
 *
 */
public class SearchManager {
	static Utils util = new Utils();

	public void initiateSearch(Path indexPath, String searchString, String rankingModel)
throws Exception {

		String splChrs = Constants.SPL_CHARS;//search for special chars, and throw
error if found one
```

```
		boolean found = searchString.matches("[" + splChrs + "]+");
		if (found) {
			System.out.println(Constants.IMPROPER_SEARCH_QUERY);
			System.exit(0);
		}

		IndexReader indexReader =
DirectoryReader.open(FSDirectory.open(indexPath));

		IndexSearcher searcher = new IndexSearcher(indexReader);
		searcher.setSimilarity(util.getSimilarity(rankingModel));

		// English Analyzer used for both Indexing and Searching as it uses
		// Porter Stemmer
		Analyzer analyzer = new EnglishAnalyzer();
		QueryParser parser = new QueryParser(Constants.FIELD_CONTENT, analyzer);
		Query query = parser.parse(searchString);
		search(searcher, query);
		indexReader.close();
	}


	/**
	 * Makes a search using indexSearcher by passing the query to the searcher
	 * @param indexSearcher
	 * @param query
	 * @throws IOException
	 */
	private void search(IndexSearcher indexSearcher, Query query) throws IOException {
```

```
                    TopDocs topDocs =
indexSearcher.search(query,Constants.MAX_SEARCH_RESULTS); //TopDocs points to the
top N search results which matches the search criteria.

                    ScoreDoc[] scoreDocs = topDocs.scoreDocs;

                    System.out.println(Constants.ALL_RESULTS+ topDocs.totalHits);

                    if(topDocs.totalHits > Constants.MAX_SEARCH_RESULTS)//show the top most
n relevant results required

                    System.out.println(Constants.MAX_SEARCH_RESULTS
+Constants.MOST_REL_MSG);

                    Document doc = new Document();

                    int i = 1;


                    //print required information of the results

                    for (ScoreDoc scoreDoc : scoreDocs) {

                              doc = indexSearcher.doc(scoreDoc.doc);

                              String path = doc.get(Constants.FIELD_PATH);

                              if (path != null) {

                                        System.out.println(Constants.URL_RANK_MSG + i);


System.out.println(Constants.URL_TITLE_MSG+doc.get("title"));

                                        System.out.println(Constants.URL_MSG+doc.get("url"));

                                        System.out.println(Constants.URL_REL_SCORE_MSG +
scoreDoc.score);

                              } else {

                                        System.out.println(Constants.NO_PATH+i);

                              }

                              util.insertNewLine();

                              i++;

                    }

          }

}
```

5.Utils.java:

```
package com.irpt2.util;

import java.io.*;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.attribute.UserDefinedFileAttributeView;

import org.apache.lucene.search.similarities.BM25Similarity;
import org.apache.lucene.search.similarities.ClassicSimilarity;
import org.apache.lucene.search.similarities.Similarity;

import com.irpt2.constants.Constants;


/**
 * This class is used to create utility methods for this application
 *
 */
public class Utils {


	/**Checks whether the input ranking model is a valid one.
	 * @param rankingModel
	 * @return
	 */
```

```
        public boolean validRankingModel(String rankingModel){
                if(rankingModel.equalsIgnoreCase(Constants.RANKING_MODEL_VS) ||
rankingModel.equalsIgnoreCase(Constants.RANKING_MODEL_OK))
                        return true;
                else
                        return false;
        }
         /* Create a file that contains the body text of the url and then adds two user defined
attributes to the document namely url ands title
          * @param url
          * @param content
          * @param title
          * @param filepath
          * @param filenum*/
        public void FileCreator(String url,String content,String title,String filepath,int filenum
) throws IOException
         {
                 String seperatedpath = "C:" + File.separator + "Users" + File.separator +
"Public"+File.separator+"Documents"+File.separator+"docsfromweb"+File.separator+String.
valueOf(filenum)+"tfurl.txt";
                 try {
                 File file= new File(seperatedpath);
                 if(!file.exists()) {
                        file.createNewFile();}
                 PrintWriter pw=new PrintWriter(file);
                 pw.println(content);
                 pw.close();


                 }
                catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();}
                    final Path docPath = Paths.get(seperatedpath);
                    final UserDefinedFileAttributeView view =
Files.getFileAttributeView(docPath, UserDefinedFileAttributeView.class);
                    view.write("url", StandardCharsets.UTF_8.encode(url));
                    view.write("title", StandardCharsets.UTF_8.encode(title));
                }



        /**Print an empty line
         *
         */
        public void insertNewLine(){
                System.out.println("\n");
        }



        /**Return the similarity to be set for indexing and searching as per the user input
         * @param rankingModel
         * @return Similarity
         */
        public Similarity getSimilarity(String rankingModel){
                Similarity classicSimilarity = new ClassicSimilarity();
                Similarity bm25Similarity = new BM25Similarity();
                if(rankingModel.equalsIgnoreCase(Constants.RANKING_MODEL_VS)){
                        return classicSimilarity;
                }
                else if(rankingModel.equalsIgnoreCase(Constants.RANKING_MODEL_OK)){
                        return bm25Similarity;
                }
                else{
                        return classicSimilarity;
                }


        }
        /* Converts the String's domain part into lowercase, checks for a trailing slash at the
end of the url and removes it, checks if the url is an anchor(i.e., # or?), then it removes the
partr of the url after the anchor
         * and then returns the url
         * @param url*/
  public String normalizeUrl(String url) {
         String lcUrl=url;
         String dot= ".";
         int count=0;
                for(int index=0;index>=0 && index < lcUrl.length()&& count<4;index++)
                {
                        index = lcUrl.indexOf(dot, index + 1);
                  if(index>0) {
                          String s= lcUrl.substring(0, index);
                          lcUrl=lcUrl.replaceAll(s, s.toLowerCase());
                          count++;}
                }


        if(lcUrl.endsWith("/")){lcUrl=lcUrl.substring(0, lcUrl.length()-1);}
         String hash="#";
         String questionmark="?";
         if(lcUrl.indexOf(questionmark)>0) {lcUrl= lcUrl.substring(0,
lcUrl.indexOf(questionmark));}
         if(lcUrl.indexOf(hash)>0)
         { lcUrl= lcUrl.substring(0, lcUrl.indexOf(hash));  }
         return lcUrl;}
/* adds the url along with its crawldepth to pages.txt in the index folder
 * @param indexpath
 * @param url
 * @param depth*/
public void addtopagestxt(String path,String url, int depth)
{
            String x= path+"/pages.txt";
            String con=url+","+String.valueOf(depth);
                  try {
                  File file= new File(x);
                  if(!file.exists()) {
                        file.createNewFile();}
                  String filecontent = new String(Files.readAllBytes(Paths.get(x)));
                  String content=filecontent+con;
                 // if(content is there in file)
                  PrintWriter pw=new PrintWriter(file);
                 pw.println(content);
                  pw.close();
                  }
                 catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();}
}

/* Gets the Userdefinedattritube from the document
 * @param view
 * @param attributeName
 * */
```

```
  public String getUserDefinedAttribute(UserDefinedFileAttributeView view, String
attributeName) throws IOException {
            if (view.list().contains(attributeName)) {
              ByteBuffer buffer = ByteBuffer.allocateDirect(view.size(attributeName));
              view.read(attributeName, buffer);
              buffer.flip();
              return StandardCharsets.UTF_8.decode(buffer).toString();
            } else {
              return "";
            }
  }
}
```

6.Constants.java:

```
package com.irpt2.constants;

/**
 * All Static Data Captured Here
 *
 */
public class Constants {

        //RANKING MODEL CONSTANTS
        public static final String RANKING_MODEL_VS = "VS";
        public static final String RANKING_MODEL_OK = "OK";

        //INDEXING CONSTANTS
        public static final String INDEX = "Index";
        public static final String INDEXING_STARTED_MSG = "Indexing your files...";
        public static final String INDEXING_MSG = "Indexing started for:\t";
        public static final String INDEXING_COMPLETED_MSG = "Indexing Completed.";
	public static final String ADD_INDEX = "Adding index file:\t";
	public static final String UPDATE_INDEX = "Updating index file:\t";


	public static final String FIELD_PATH = "FilePath";
	public static final String FIELD_LAST_UPDATED = "LastModified";
	public static final String FIELD_CONTENT = "Content";


	//SEARCHING CONSTANTS
	public static final String SEARCH_QUERY_MSG = "Searching for query:\t";
	public static final int MAX_SEARCH_RESULTS = 10;
	public static final String ALL_RESULTS = "Overall matching documents:\t";
	public static final String  MOST_REL_MSG = "\tmost relevant documents:\t";
	public static final String DATE_FORMAT = "MM/dd/yyyy HH:mm:ss";


	public static final String URL_RANK_MSG = "Url Rank:\t";
	public static final String URL_MSG = "Url is:\t";
	public static final String URL_TITLE_MSG = "Title is:\t";
	public static final String URL_REL_SCORE_MSG = "Url Relevance score:\t";



	//WARNING MESSAGES
	public static final String HELP_MESSAGE="You must enter all valid arguments.Please
try again.";
	public static final String PATH_NON_EXISTENT="The entered file path is not valid";
	public static final String UNABLE_TO_READ_FILE="Unable to read file from the
path:\t";
	public static final String IMPROPER_SEARCH_QUERY="Improper Search Query.";
	public static final String NO_PATH = "No Path field available at search result:\t";


	//PROGRAM START AND END MESSAGES
	public static final String WELCOME_MESSAGE="====\tWelcome To Information
Retrieval System\t====";
	public static final String EXITING="====\tEXECUTION COMPLETED.\t====";


	//SPECIAL CHARS
	public static final String SPL_CHARS = "-/@#$%^&_+=()!{};.*,<>?':|";
}
```