Solution
Documentation

Java Version (JDK): 10.0.1

Apache Lucene™ Version: 7.5.0

Source Code is divided into following three files:

LuceneMain.java:

Deals with parsing of command line arguments like path of the index, document path, query and ranking model as provided by the user. After parsing of the arguments one standard analyzer has been created to set the configuration of index-writer. If ranking model is as provided as Vector Space (VS) then index-writer-configuration similarity is set to classic similarity, and if model is OKAPI (OK) then index-writer-configuration similarity is set to BM25 similarity. If neither Vector Space (VS) nor OKAPI (OK) has been provided then a message with "Please enter valid model which is OK or VS" will be printed and program will exit and will not be executed further. If valid ranking model has been provided, then a new index writer will be created with set configuration and index directory path. Thereafter, program will call the static function indexdocs() present in IndexFiles.java for which parameters will be newly created writer and document path. After indexing has been completed program will proceed with search. For search we already have our query, ranking model and index directory so program will invoke static function perform_search() present in SearchFiles.java with parameters index directory path, ranking model and query.

IndexFiles.java:

When static function indexdocs() is called from main function with parameters indexwriter object and path of document folder, at first program checks if document folder exists or not, if not it will display error message as "Please enter valid document entry" and exit as it doesn't have data folders. If valid path has been given then it uses walkfiletree() to traverse through the directory and checks if files are with extension txt or html or can't be read. If extension is txt then it will call a function index_txt() and would pass each txt file and writer to this function. Inside indexDoc _txt(), it creates a new Document object along with multiple fields for path and filename which needs to be shown during search to the user. Regarding contents or body of the document an analyzer has been created along with streams of tokens from the contents and stopwords have been removed using Default English Stop Words as provided by Lucene. PorterStemFilter() from "org.apache.lucene.analysis.en.PorterStemFilter" has been applied for stemming. After applying filter, the analyzer gets closed and adds the streams in field for indexing which won't be shown to the user after search. The program then checks if index writer configuration mode is open; if yes then it will add the document object holding multiple fields in our index writer. There might be the case if old copy of index has been present, so in that case it will update the document replacing the old one if path is the same. If file is html then indexDoc_html will be called which will have the same parameters as that of indexDoc_txt. In indexDoc_html only difference is at first *JSOUP (Version 1.11.3)* is used to parse html files and get title and body of the file for indexing and the rest things would be same as mentioned in indexDoc_txt. Function getFileExtension() has been defined at end which has been used to get the extension of file. After completing indexing of the file, control will be returned to LuceneMain.java

Team **JoKeR**:
Jalaj Vora [221510]
Karthik Kaddajji [221367]
Rabi Kumar K C [221488]

Solution
Documentation

SearchFiles.java:

From SearchFiles.java, Static function perform_search() is invoked from main and parameters like: query, ranking model, indexpath is passed. Inside this function, fields of index needed for search have been declared and have been matched the document/s with the query as indexing is done on field basis. The fields have declared to be title and contents. One more variable for maximum hit display has been created. An index-reader object will be created to read the index directory provided. Then searcher object is created by using index-reader object. Here, Similarity has been initialized according to parameters provided. If ranking model doesn't match with "VS" or "OK" then program will exit. A standard analyzer has been created and passed to multifield query parser along with fields where it needs to search and match. It stems the query as well and then it creates a Query object by using multifield query parser to parse the query. Later, this value is sent to dosearch() function where it gets the top documents if available, along with scores and total number of hits of documents according to query given. Further if matched (documents being less than 10) then it takes minimum-match (minimum between 10 or the no. Of documents) documents to display the result. A Loop has been used for displaying results. Additionally, file extension has been checked so as to show whether title is html and filename with txt (Extension). After displaying results control is returned to Main() function.

How to run the Program:

Step1: Keep jar file in your desired location.

Step2: Make sure that you have created index directory and document directory which consist of sample files for indexing in desired location.

Step3: Open command line and change directory to the directory where jar file has been present.

Step4: Enter the command as

java –jar [filename].jar "documentPath" "indexPath" "rankingmodel" query

Step5: Results will be displayed on the console.

Team **JoKeR**:
Jalaj Vora [221510]
Karthik Kaddajji [221367]
Rabi Kumar K C [221488]