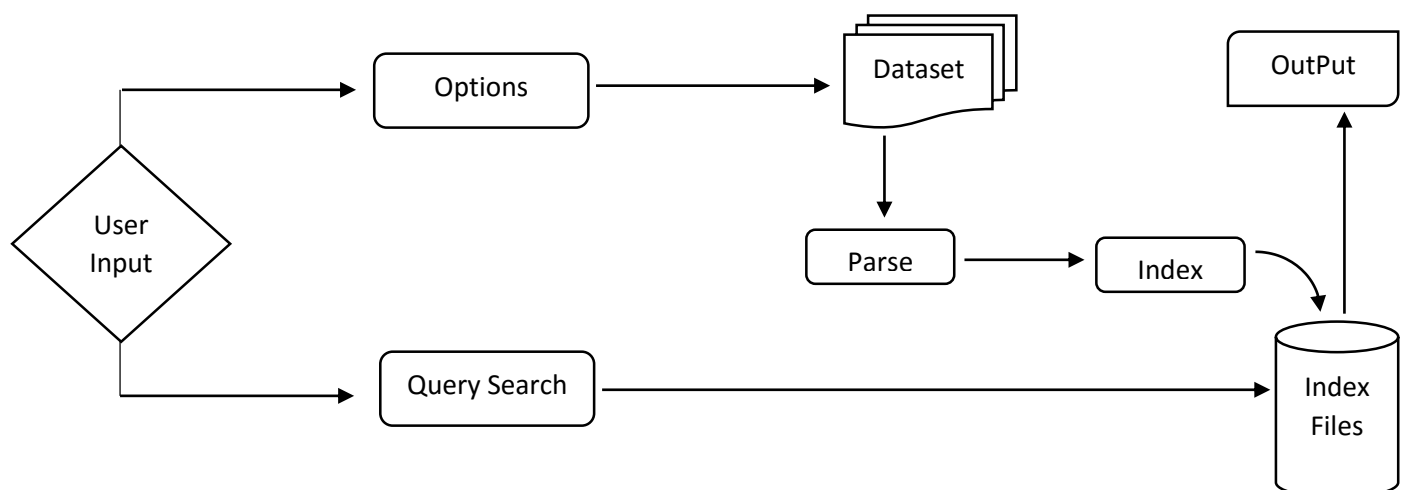


Project outline:

- The main purpose of this project is to write a **Wikipedia Search Engine** program, which is able to take an input from a user (any kind of string representation) and search for matching articles.
- Another goal for this project is to take text wiki dataset files, and parse them into meaningful XML entities. Also a part of the program is to Index that XML and make it searchable by the program.

How it works?

- **Phase 1:** The user choses to enter a query string to search, or to enter and customize some parameters for the program.
- **Phase 2a (Options):** The user can redo dataset pre-processing, re-indexing, or change the number of top retrieved documents.
- **Phase 2b (Query):** The user gets the results for the entered query string.
- **Phase 3a (redo pre-processing):** Read provided text files, and create the main XML file with concatenated fields into tags.
- **Phase 3b (redo indexing):** Use the XML file to read article at a time, and index it.
- **Phase 3c (Top Documents Retrieved):** Just entering a number would adjust that parameter for the search engine
- **Phase 4:** Search Results.



Data Used (Dataset)

Downloaded from

Total of 3,201,391 Article which are 1.35 GB.

Full set are 3,201,391 article

(Consists of: wiki_labels_all.txt / wiki_links_all.txt / wiki_abstracts_all.txt)

Partial set are 30,000 article

(Consists of: wiki_labels_part.txt / wiki_links_part.txt / wiki_abstracts_part.txt)

Constructed as follows:

Labels:

```
<http://dbpedia.org/resource/%21%21> <http://www.w3.org/2000/01/rdf-schema#label> "!!!"@en .
```

Links:

```
%21%21 http://xmlns.com/foaf/0.1/page http://en.wikipedia.org/wiki/%21%21 r
```

Abstracts:

```
<http://dbpedia.org/resource/%21%21%21> <http://www.w3.org/2000/01/rdf-schema#comment> "Desc"@en .
```

Each line contains a single article information.

Note 1:

Program comes **pre-loaded** and ready to run (Datasets / FULL XML file / FULL index).

Note 2:

Indexing and Searching techniques are done with **java Lucene API**.

Note 3:

For now the program only runs from **command prompt** or any **IDE console**.

Phase 1 – User input *wikisearchengine.Main*

What is it?

The program allows the user to choose what route to be executed. (Figure 1).

User can:

- Manually insert the query string needed.
- Press **“op”** (non-case-sensitive) to be redirected to adjust the options.
- Exit the program by pressing **“Q”** (non-case-sensitive).

Figure 1

```
*****
Welcome To Wikipedia offline *Mini* Search Engine V1.0
*****
For the system options, press OP anytime
To exit, press Q anytim
Enter a search query:
```

How it works?

- By running **“wikisearchengine.main”** the user will be prompted with the options shown in Figure 1.
- Can enter any string right away, to search with it as a query.
- Press “OP” or “op” to get redirected to adjust the parameters (later).
- Can exit anytime by pressing “Q” or “q”.
- Functions do as expected by dispatching the input properly...
 - Either to enter a query string, and the corresponding function is invoked to evaluate the input and search for a match.
 - Or redirect the user to another screen, with the options available cases.
 - Or exit the program.

Code snippet:

Figure 2

```
101 // dispatch user input
102 do {
103     switch (op) {
104         case 0:
105             options = false;
106             break;
107         case 1:
108             preprocessData(true);
109             reIndexAll();
110             options = false;
111             break;
112         case 2:
113             preprocessData(false);
114             reIndexPart();
115             options = false;
116             break;
117         case 3:
118             reIndexAll();
119             options = false;
120             break;
121         case 4:
122             reIndexPart();
123             options = false;
124             break;
125         default:
126             topDocNum(op);
127             options = false;
128             break;
129     }
130 } while (options);
131 }
```

Figure 3

```
190
191 /**
192  * Searches a user query against the current dataset index.
193  *
194  * @param q
195  *         Query string from the user
196  */
197 private static void searchQuery(String q) {
198     try {
199         System.out.println("Performing search");
200
201         // SEARCH
202         long startTime = System.nanoTime();
203         SearchEngine se = new SearchEngine();
204         TopDocs td = se.performSearch(q, ret);
205         long estimatedTime = System.nanoTime() - startTime;
206         double seconds = (double) estimatedTime / 1000000000.0;
207
208         System.out.println("Results Found (Elapsed Time: " + seconds + ")");
209         ScoreDoc[] hits = td.scoreDocs;
210
211         prettyPrint(hits, se);
212     } catch (Exception e) {
213         System.out.println("Oops! :( something went wrong!");
214         e.printStackTrace();
215     }
216 }
217 }
```

Phase 2a – Options *wikisearchengine.Main*

What is it?

The program allows the user to make some adjustments on the system. (Figure 4).

User can:

Figure 4

- a) Re-create the main XML file from scratch. (full / partial)
- b) Re-create the indexes from scratch. (full / partial)
- c) Change the default number for the top documents retrieved.
- d) Go back to main screen.

```
*****
Customizable Options
*****
0. Go Back
1.Redo data preprocessing & Indexing (Full version)
2.Redo data preprocessing & Indexing (Partial version)
3.Reindex wikipedia Articles (Full version)
4.Reindex wikipedia Articles (Partial version)
Or, Set max number of retrieved documents (> 4 - Default 10)
Enter a number:
```

How it works?

- By running **"wikisearchengine.main"** and choosing to get redirected to *Options*, the user will be prompted with the options shown in Figure 4.
- Can redo **preprocessing** steps from scratch (full or partial).
- Can redo **indexing** steps from scratch (full or partial).
- Can change the value for number of top retrieved documents.
- Can go back to the previous screen
- Functions do as expected by dispatching the input properly...
 - Either to invoke *preProcessData()*, with the correct Boolean value for full or partial.
 - invoke *reIndexAll()*.
 - invoke *reIndexPart()*.
 - invoke *topDocNum()*.
 - Or return to previous screen.

Code snippet:

Figure 5

```
101 // dispatch user input
102 do {
103     switch (op) {
104         case 0:
105             options = false;
106             break;
107         case 1:
108             preProcessData(true);
109             reIndexAll();
110             options = false;
111             break;
112         case 2:
113             preProcessData(false);
114             reIndexPart();
115             options = false;
116             break;
117         case 3:
118             reIndexAll();
119             options = false;
120             break;
121         case 4:
122             reIndexPart();
123             options = false;
124             break;
125         default:
126             topDocNum(op);
127             options = false;
128             break;
129     }
130 } while (options);
```

Phase 2b – Query *wikisearchengine.Main / wikisearchengine.SearchEngine*

What is it?

Taking the input from the user, processing it, and returning the matching documents as a search results (Figure 6).

How it works:

- When the function *searchQuery()* is invoked with the query string,
- It creates a new instance from *SearchEngine* class.
- pass it the query and the number of top documents retrieved.
- get back a list of top documents matching the query, along with its contents.
- pass those documents to *prettyPrint()* function, to be returned to the user as search results.
- present the user with the same initial first screen.

Figure 6

```
OR! Enter a search query:
1
Performing search
Results Found (Elapsed Time: 0.009611536)

-----
| Search Result Top 10 found |
-----

| Article 32      ( 2.2704797) |
-----

| $1
| -> No Description Available!
| http://en.wikipedia.org/wiki/%241

*****
Welcome To Wikipedia offline *Mini* Search Engine V1.0
*****
```

Code snippet:

Figure 7

```
190
191 /**
192  * Searches a user query against the current dataset index.
193  *
194  * @param q
195  *       Query string from the user
196  */
197 private static void searchQuery(String q) {
198     try {
199         System.out.println("Performing search");
200
201         // SEARCH
202         long startTime = System.nanoTime();
203         SearchEngine se = new SearchEngine();
204         TopDocs td = se.performSearch(q, ret);
205         long estimatedTime = System.nanoTime() - startTime;
206         double seconds = (double) estimatedTime / 1000000000.0;
207
208         System.out.println("Results Found (Elapsed Time: " + seconds + ")");
209         ScoreDoc[] hits = td.scoreDocs;
210
211         prettyPrint(hits, se);
212     } catch (Exception e) {
213         System.out.println("Oops! :( something went wrong!");
214         e.printStackTrace();
215     }
216 }
217
```

Phase 3a – Redo pre-processing [wikidataset.TextToXmlAll](#) / [wikidataset.TextToXmlPart](#)

What is it?

A text to xml converter class, that reads from each of the three files one line at a time, compare them.

Whenever it finds a match, it starts an <Article> tag and put in the appropriate information.

How it works?

- Opens 3 *buffered readers*, one for each text file (labels / links / abstracts).
- Opens 1 *stream result*, to open one xml file to write the articles with the information associated with it.
- **The case is that every label (title) has a link, but not every instance has a description.**
- Every text line is compared against *java regex*, to be cleaned and info to be extracted properly.
- Finally after all text processing is done, the xml file gets populated with information.

XML structure:

Code snippet:

Figure 8

```
<Wiki>

    <Article>

        <ID> ..... </ID>

        <Title> ..... </Title>

        <Link>.....</Link>

        <Description> .....</Description>

    </Article>

    <Article> ..... </Article>

    <Article> ..... </Article>

</wiki>
```

Figure 9

```
27  /**
28   * Main method to run the Converter and get the work done, and calculate the elapsed time
29   *
30   * @param args
31   *         argument parameters
32   */
33  public static void main(String args[]) {
34      long startTime = System.nanoTime();
35      new TextToXmlAll().buildUp();
36      long estimatedTime = System.nanoTime() - startTime;
37      double seconds = (double) estimatedTime / 1000000000.0;
38
39      System.out.println("Elapsed Time : " + seconds);
40      System.out.println("*****");
41      System.out.println("** Data Preprocessing DONE! **");
42      System.out.println("*****");
43  }
```

Phase 3b – Redo Indexing [wikiarticle.Article](#) / [wikiarticle.ArticleIndexer](#) / [wikiarticle.ArticleParser](#)

What is it?

Invoking the *Indexing class*, which invokes *parsing class* to read the xml file in a stream, and parse each tag and index each independent article as a document in the index. Resulting in one index holding all the documents to search for.

How it works (Using Lucene API)?

- Create an *index reader*, and an index directory.
- pass that index reader to *Parser class*.
- *Parser class*, opens the xml file and reads it in a stream.
- For each <Article> tag, write its information into the index writer as a separate document.
- After finishing the whole xml file, close the index, release the lock, and close the index writer.

Code snippet:

Figure 10

```
21  /**
22   * Start the whole process of parsing the XML file to be indexed.
23   *
24   * @param _indexer
25   *         ArticleIndexer class instance, that have an open Index Writer.
26   * @throws XMLStreamException
27   * @throws IOException
28   */
29  public void run(ArticleIndexerAll _indexer) throws XMLStreamException, IOException {
30      indexer = _indexer;
31      XMLStreamReader read = this.openReader();
32      this.parse(read);
33      this.close(read);
34  }
```

Figure 11

```
86  /**
87   * Rebuild Index by reindexing all the current data set.
88   *
89   * @throws IOException
90   * @throws XMLStreamException
91   */
92  public void rebuildIndexes() throws IOException, XMLStreamException {
93      // get index writer
94      getIndexWriter();
95
96      // run parser
97      ArticleParserAll parser = new ArticleParserAll();
98      parser.run(this);
99
100     // close index writer
101     closeIndexWriter();
102 }
```

Phase 3c – Top Documents Retrieved [wikisearchengine.Main](#)

What is it?

This is just about changing one value in the main class (default 10).

How it works?

- User enters any number larger than 4.
- A method *topDocNum()* is invoked with the entered value.
- The function changes the maximum top number of document retrieved in the search engine (**only for the current search engine instance**).
- Return to the initial first screen.

Code snippet:

Figure 12

```
232  /**
233   * Changes the default number of the documents retrieved (10).
234   *
235   * @param i
236   *         the new number
237   */
238  private static void topDocNum(int i) {
239      ret = i;
240  }
```


Phase 4 – Search Results [wikisearchengine.Main](#)

Results Found (Elapsed Time: 0.279498327)

| Search Result Top 10 found |

| Article 5 (1.9222708) |

| !!!_(album)
| -> "!!! is the eponymous debut album by !!! . It was released in 2001."
| http://en.wikipedia.org/wiki/%21%21%21_%28album%29

| Article 14 (1.6018922) |

| !Hero_(album)
| -> "!Hero an album featuring the songs from the rock opera, !Hero."
| http://en.wikipedia.org/wiki/%21Hero_%28album%29

| Article 27 (1.6018922) |

| !_ (album)
| -> "! is an album by The Dismemberment Plan. It was released on October 2, 1995 on DeSoto Records."
| http://en.wikipedia.org/wiki/%21_%28album%29

| Article 6 (0.7928962) |

| !!Destroy-Oh-Boy!!
| -> "!!Destroy-Oh-Boy!! is the debut album by the American garage punk group New Bomb Turks. It was released in 1993 by Crypt Records."
| <http://en.wikipedia.org/wiki/%21%21Destroy-Oh-Boy%21%21>

*** Some Stats (i7 Processor – 8GB RAM) ***

Process	Numbers	Unit
Xml creation for 3,201,391 line of text	281.15964686	Seconds
Number of complete Articles found (on the above)	2,959,360	Seconds
Indexing 2,959,360 articles in one Lucene index	254.757844039	Seconds
Searching for album	0.056861375	Seconds

Future Improvements:

- ✚ Storing the data and the index in a database for easier access.
- ✚ Implementing the indexing part myself, for more control over it.
- ✚ Building GUI for easier interaction.