Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Information Retrieval

Seminararbeit FS 2014

Student: Micha Schönenberger

Dozent: Dr. Ruxandra Domenig

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Versionierung

Version	Datum	Beschreibung
V0.1	18.03.2014	Ersterstellung Dokument
V0.2	11.05.2014	Einarbeitung in Lucene, Installation Software, Lesen LiA
V0.3	13.05.2014	Buch LiA lesen / Coden
V0.4	14.05.2014	Doku schreiben, Query Optionen studieren
V0.5	15.05.2014	Doku LATEX, Query Optionen studieren
V0.6	17.05.2014	eingesetzte Hardware, Kapitel Vorbereitung für Vergleich
V0.7	18.05.2014	Kapitel Vorbereitung für Vergleich, Code Refactoring
V0.8	20.05.2014	Kapitel Vorbereitung für Vergleich, PDF und Office Indexing Probleme

Tabelle 1.1.: Versionierung Dokumentation

2. Aufwände

Datum	Zeit	Beschreibung
18.03.2014	1.25h	Ersterstellung Dokument -Download und Installation Lucene -Erste Versuche mit Lucene
26.03.2014	3.75h	Suche nach Thema für Eingabe EBS -Online Suche Funktionalitäten Lucene
11.05.2014	4.25h	Installation von Netbeans -Update Eclipse -Lucene in Action: Einlesen Kapitel 1 -1. Versuch Indexer und Searcher (gemäss Beispielen LiA)
12.05.2014	1.75h	siehe 11.05.2014
13.05.2014	7h	-Indexierung erweitern auf alle Subdirectories des Root-Folders -Einschränken von File-Extensions: momentan nur Textdateien (.txt, .h, .c, .java)
13.05.2014	2.75h	-Dokumentation Kapitel 3.1.3 bis 3.1.4 -Quellenverzeichnis erstellen -Kapitel 4.1 eingesetzte Software -Kapitel 5.3 Syntax für Suche mit Lucene / Luke -Einarbeiten in Query mit Lucene/Luke
15.05.2014	4.25h	-Dokumentation von Word in LATEX umschreiben -Kapitel 5.3 Syntax für Suche mit Lucene / Luke
17.05.2014	2.25h	-Kapitel 5 mit teils Unterkapiteln -Kapitel 4.2 eingesetzte Hardware
18.05.2014	5h	-Kapitel 5 mit teils Unterkapiteln -Code Refactoring and cleaning

Tabelle 2.1.: Aufwände Seminararbeit

3. Einleitung

3.1. Das Projekt

3.1.1. Ausgangslage

In den knapp 4 Jahren an der ZHAW haben sich verschiedenste Dateien angehäuft. Die Thematiken überschneiden die Module und sind somit nicht immer über eine übersichtliche Ordnerstruktur auffindbar.
Die vielen Typen von Dokumenten (Office, pdf, txt, java...) vorhanden sind, wird durch die Betriebssystem integrierte Suche nicht immer das erwartete Resultat geliefert. Alle diese Dokumente sollen über Lucene mit einem Index versehen und für eine effiziente Suche optimiert werden. So können unter anderem auch PDF inhaltlich indexiert werden, was bei der Search-Engine des Betriebssystems nicht funktioniert.

3.1.2. Ziel der Arbeit

Das Ziel der Arbeit soll ein direkter Vergleich zwischen der Suchresultate von Lucene mit denjenigen des Betriebssystems stattfinden.
Aufgrund der subjektiven Sicht der suchenden Person soll schlussendlich entschieden werden, welche Search-Engine die besseren Suchresultate liefert. Es sollen mindestens zwei reale Begriffe gesucht werden und anhand dieser ein Fazit gezogen werden (eventuell mit Versbesserungsvorschlägen für die Optimierung von Lucene).
Für die Indexierung von PDF, welche nicht durch die Search-Engine des Betriebssystems bei der Indexierung eingeschlossen werden soll Lucene Abhilfe schaffen.

3.1.3. Aufgabenstellung

1. Definierung der Suchkriterien über alle zu indexierenden Dateien
2. Erstellen einer einfachen Demo-Applikation in Java
3. Indexierung aller Schuldateien
4. Implementierung für die Indexierung von PDF. Wenn dies nicht direkt möglich sein sollte in Lucene, werden PDF in Textdateien umgewandelt und dann indexiert
5. Überprüfung der Suchresultate und Optimierung von Lucene
6. Reale Suche von mindestens zwei Begriffen in Lucene und direkter Verlgeich mit den Suchresultaten des Betriebssystemes
7. Fazit der Implementierung von Lucene. Sind die Suchresultate besser als diejenigen des Betriebssystemes? Begründung

3.1.4. Erwartetes Resultat

1. PDF sollen in den Suchresultaten mitberücksichtigt werden
2. Die Eingabe der Suchbegriffe soll intuitiv und einfach gehalten werden
3. Die Qualität der Suchresultate von Lucene soll diejenige vom Betriebssystem übertreffen. Falls dies nicht der Fall sein sollte, soll eine kurze Analyse aufzeigen, wieso das Betriebssystem bessere Resultate hervorbringen kann

3.1.5. Geplante Termine

Datum	Beschreibung
14.03.2014	Kick-Off Meeting
11.06.2014	Abgabe der schriftlichen Arbeit (1 Woche vor Präsentation)
18.06.2014	Präsentation der Arbeit

Tabelle 3.1.: geplante Termine

4. Die Search-Engine

4.1. eingesetzte Software

Um mit Lucene programmieren zu können, wurden folgende Software eingesetzt:

- Mac OS X 10.9 (Mavericks)
- Eclipse SDK (Indigo), Version 3.7.2
  http://www.eclipse.org/downloads/packages/release/indigo/sr2
- Java(TM) SE Runtime Environment (build 1.8.0_05-b13)
  http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Lucene Version 3.0.2 (für Indexierung)
  http://www.apache.org/dyn/closer.cgi/lucene/java/3.0.2
- Apache PDFBox Version 1.8.5 (Open Source Library für Indexierung von PDF)
  http://pdfbox.apache.org/downloads.html
- Luke - Lucene Index Toolbox - v 3.5.0 (2011-12-28)
  stellt Indexierte Dateien von Lucene dar und erlaubt die Suche nach Stichwörtern.
  https://code.google.com/p/luke/downloads/detail?name=lukeall-3.5.0.jar&
- Tika
  http://www.apache.org/dyn/closer.cgi/tika/tika-app-1.5.jar

4.2. eingesetzte Hardware

- Apple Mac Book Air 2011 (MacBookAir4,2)
  - OS: MAC OS X 10.9.2
  - RAM: 4GB 1333 MHz DDR3
  - CPU: 1.8 GHz (2677M) Dual-Core i7 mit 4 MB on-chip L3 cache
  - Harddisk: SSD 256GB - APPLE SSD SM256C Media
- Asus P7H55E
  - OS: Windows 8.1 Professional
  - RAM: 8GB DDR3
  - CPU: Intel Core i5 661 @ 3.33GHz
  - Harddisk: Western Digital black Edition - 1TB (WD 1002FAEX-00Y9A0)

5. Vorbereitung für Vergleich

Bevor ein Vergleich zwischen den Verschiedenen Search-Engines gemacht werden kann, sollte ein Überblick über die Search-Engines erstellt werden.

5.1. Windows Index Suche

Standardmässig werden bei Windows nur einzelne Ordner indexiert. Gleichzeitig indexiert Windows nicht die kompletten Datei Inhalte, sondern lediglich die Einstellungen.
Um die Inhalte auch zu indexieren, muss dies in den Einstellungen geändert werden (siehe Abbildung 5.1 rechte Seite).

Um nun ein Netzlaufwerk indexieren zu können (was hier auch gewünscht ist), muss ein Add-in von Microsoft installiert werden. Dieses nennt sich «Windows Desktop Search: Add-in for Files on Microsoft Networks»und ist auf der Microsoft Homepage verfügbar[1].

Da dies leider nach einem Neustart nicht funktionierte, wurde eine weitere Möglichkeit gesucht.
So wurde gemäss dem Microsoft Technet-Artikel[2] folgdene Punkte erledigt:

- Erstellen einer neuen Windows Library «ZHAW Docs» erstellt.
- Erstellen eines neuen Ordners C:\ZHAW Docs
- Hinzufügen des neu erstellten Ordners zur neuen Library
- Ordner C:\ZHAW Docs löschen
- Über cmd (als Administrator) folgenden Befehl ausführen:
  mklink /d “C:\ZHAW Docs“ “\\<IP Synology>\<Share-Path>“
  Mit diesem Befehl wurde ein symbolischer Link des Netzwerkpfades auf den lokalen Pfad gesetzt.

[1] http://www.microsoft.com/en-us/download/details.aspx?id=3383
[2] http://social.technet.microsoft.com/Forums/windows/en-US/afb904c1-1c61-4aae-b6b1-5cf525b9f8de/how-do-i-get-windows-7-to-index-a-network-mapped-drive?forum=w7itpronetworking

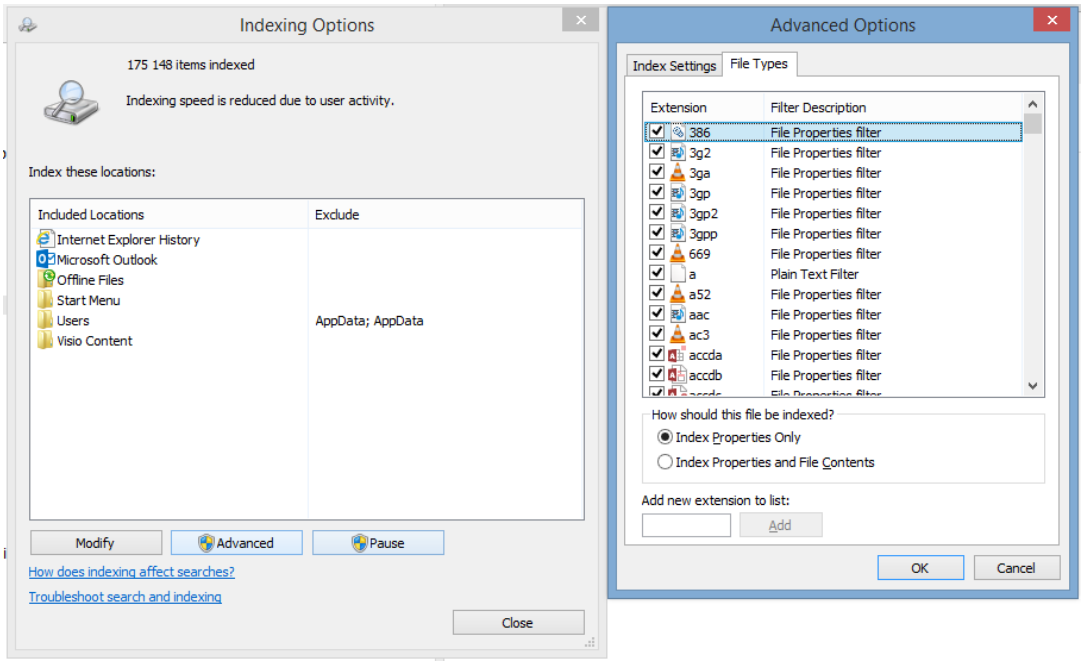

Abbildung 5.1.: Windows Indexing - Default Einstellungen
Quelle: eigener Screenshot

5.2. Mac OS X Spotlight

Beim MAC OS X Spotlight gibt es weniger Einstellungen, die man selber vornehmen kann. So kann zum Beispiel nicht gewählt werden, ob das ganze File oder nur die Einstellungen oder Metadaten zum Index hinzugefügt werden.

Abbildung 5.2 zeigt die Einstellungsmöglichkeiten. Wird hier Nummer 7 (=PDF) angewählt, indexiert MAC OS X automatisch den ganzen Inhalt des PDF, vorausgesetzt der Text im PDF ist lesbar.

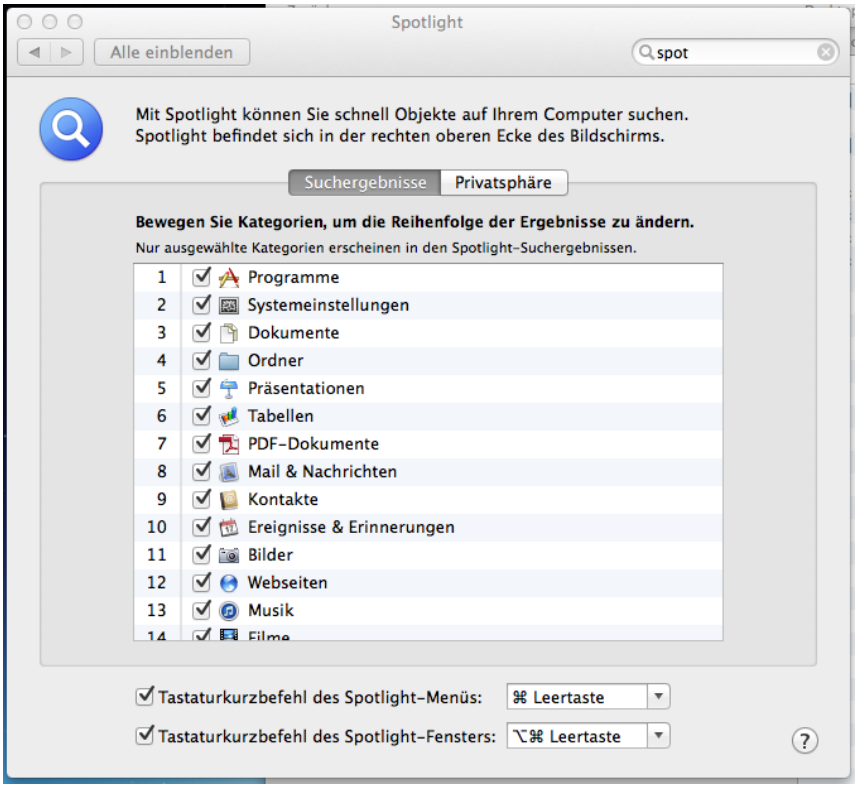

Abbildung 5.2.: Mac OS X Spotlight - Default Einstellungen
Quelle: eigener Screenshot

Beim MAC OS ist war es nicht notwendig, ein Netzlaufwerk zu indexieren, da die Daten bereits lokal auf der Festplatte vorhanden waren.

5.3. Syntax für Suche mit Lucene / Luke

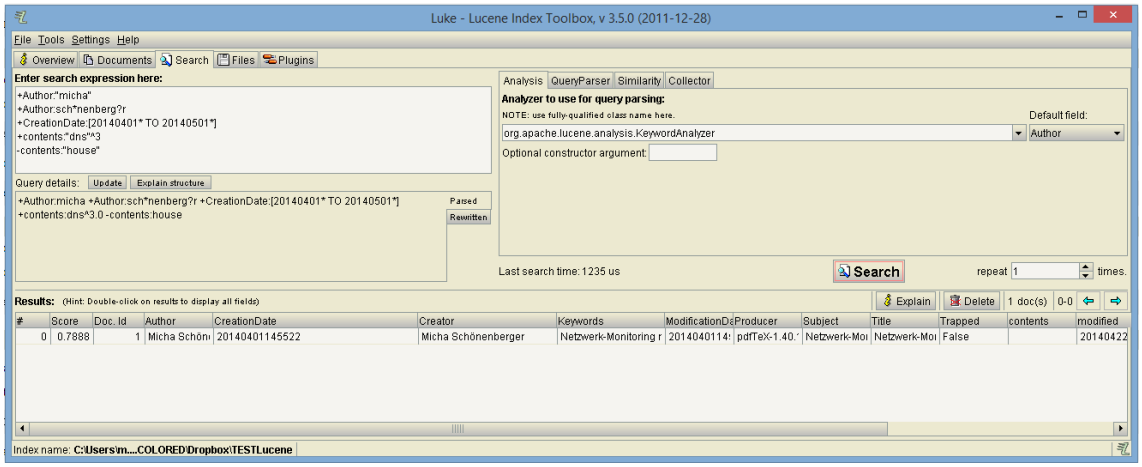

Abbildung 5.3.: Such-Maske von Luke
Quelle: eigener Screenshot

Anbei einige Beispiele, wie man den Syntax optimal einsetzen kann für sie Suche mit Luke/Lucene:

5.3.1. Felder

contents:prtg	sucht das Wort “prtg“ im Feld contents
Author:micha monitor	sucht das Wort “micha“ im Feld Author, das Wort “netzwerk“ im default Feld
contents:?mit prtg?	sucht die Wörter “mit prtg“ als ganzes im Feld contents

5.3.2. Wildcard Suche

pr?g	Das ? ersetzt einen einzelnen Charakter. So wird hier z.B. nach ?prtg?, aber auch nach ?prag? gesucht
bildschirm*	* steht für beliebig viele Zeichen (0 bis x) . Suche nach z.B. “bildschirmeingabe“ oder “bildschirme“
ver*en	* steht auch hier für beliebige Zeichen (0 bis x) Suche nach z.B. “verwerfen“ “verwalten“ oder “veruntreuen“

Grundsätzlich gilt: * und ? dürfen nicht am Anfang der suche sein. (bei Luke gibt es eine Einstellung, mit der man * am Anfang erlauben kann)

5.3.3. Fuzzy Suche

Die Fuzzy Suche basiert auf dem Levenshtein Distanz Algorithmus[3], auf welchen hier nicht eingegangen wird.
Um eine Fuzzy Suche durchzuführen, wird mit der Tilde ( ) gearbeitet.

roam	Sucht ähnliche Wörter zu roam
roam 0.8	Gibt die gesuchte Ähnlichkeit an [0-1]. Je grösser die Zahl, desto ähnlicher.

5.3.4. Proximity Suche (Nachbarschaftliche)

Lucene erlaubt es, zwei Wörter in einer bestimmten Distanz zu suchen.

“prtg monitor“~10	Sucht die Wörter “prtg“ und “monitor“ mit einem maximalen Abstand von 10 Wörtern

[3] siehe http://www.levenshtein.de/

5.3.5. Range Suche (Bereichssuche)

Lucene erlaubt es, zum Beispiel Daten in einem bestimmten Bereich zu suchen.

modified:[20140501 TO 20140515]	Sucht eine Datei mit einem Änderungsdatum zwischen dem 01.05.2014 und 15.05.2014
Title:{Anton TO Max}	Sucht eine Datei mit dem Titel zwischen Anton und Max (alphabetisch)

5.3.6. Boosting Suche (verstärktes Suchen)

Lucene erlaubt es, gewissen Suchbegriffen eine stärkere Gewichtung zu verleihen. Durch diesen Boost wird das Ergebnis die Relevanz der gefundenen Dateien verändert.

prtg^ 4 monitoring	Sucht eine Datei mit «prtg» und «monitoring» wobei «prtg» 4-fach gewertet wird.

5.3.7. Boolsches Suchen

Ganz wichtig beim Suchen sind die verschiedenen Verknüpfungen der eingegebenen Suchbegriffe. Erlaubt sind AND, +, NOT, -

prtg OR monitor	Sucht entweder nach «prtg» oder nach «monitor»
prtg AND monitor	Sucht nach «prtg» und nach «monitor». Beide Wörter müssen im selben Dokument enthalten sein.
prtg NOT monitor	Sucht ein Dokument, in welchem «prtg» vorkommt, «monitor» aber nicht.
NOT prtg	NOT darf nicht mit nur einem Wort verwendet werden. Das Resultat wird hier leer sein.

Anstelle von AND kann auch && oder + verwendet werden.
Anstelle von NOT kann auch ! oder - verwendet werden.

Quelle Kapitel 5.3: [1]

6. Vergleich der Ergebnisse

Wie in der Aufgabenstellung erwähnt, sollen die Suchresultate von Lucene mit denjenigen des Betriebssystems verglichen werden. Die Suche im Betriebssystem ist sehr intuitiv. Es wird der gewünschte Suchbegriff eingegeben und nach ein paar Sekunden (oder Minuten), je nach dem ob alle Ordner indexiert waren oder nicht, erscheint das Resultat.

Bevor jedoch ein Vergleich stattfinden kann, wird aufgezeigt, wie mit Luke eine effiziente Suche ausgeführt werden kann über die indexierten Dateien der Lucene-Applikation. Diese werden im nächsten Unterkapitel erläutert:

7. Fazit

A. Anhang

A.1. Java Code

A.1.1. Main.java

```
1  package zhaw;
2
3  import java.io.File;
4
5  import org.apache.lucene.analysis.standard.StandardAnalyzer;
6  import org.apache.lucene.index.IndexWriter;
7  import org.apache.lucene.store.Directory;
8  import org.apache.lucene.store.FSDirectory;
9  import org.apache.lucene.util.Version;
10
11 public class Main {
12    static String indexDir = "/Users/micha/Dropbox_ZHAW/Dropbox/TestLucene/";
13    static String dataDir = "/Users/micha/Dropbox_ZHAW/Dropbox/TestLucene/
          ToIndex/";
14    static myFunctions _myfunctions;
15    private static IndexWriter writer;
16    static Logger _myLogger;
17    static Indexer _indexer;
18
19    public static void main(String[] args) throws Exception {
20       _myLogger = new Logger();
21
22       long start = System.currentTimeMillis();
23       int numIndexed = 0;
24
25       /* prepare choosen directory and a IndexWriter */
26       Directory dir = FSDirectory.open(new File(indexDir));
27       writer = new IndexWriter(dir, new StandardAnalyzer(Version.LUCENE_30), true,
             IndexWriter.MaxFieldLength.UNLIMITED);
28
```

```
29      /* init my functions */
30      _myfunctions = new myFunctions(writer);
31
32      /* init the indexer */
33      _indexer = new Indexer();
34      _indexer.createSubindexer();
35
36      /* try to index the files */
37      try {
38        int _numIndexed = _indexer.index(_indexer, dataDir, new TextFilesFilter(),
              numIndexed);
39        numIndexed += _numIndexed;
40
41      }
42      /* at end, close the writer */
43      finally {
44        _indexer.closeWriter();
45      }
46
47      long end = System.currentTimeMillis();
48
49      /* Print time to used for indexing the set folder */
50      // System.out.println("Indexing " + numIndexed + " files took " + (end -
51      // start) + " milliseconds");
52    }
53
54    /* returns the path of the directory, which should be indexed */
55    public static String getIndexDir() {
56      return indexDir;
57    }
58
59    /* returns my functions */
60    public static myFunctions getMyFunctions() {
61      return _myfunctions;
62    }
63
64    /* returns writer */
65    public static IndexWriter getwriter() {
66      return writer;
67    }
68
```

```
69   /* returns Indexer */
70   public static Indexer getIndexer() {
71     return _indexer;
72   }
73 }
```

Listing A.1: Main.java

A.1.2. Logger.java

```
1  package zhaw;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileWriter;
6  import java.io.IOException;
7
8  public class Logger {
9
10   File f;
11   FileWriter fstream;
12   static BufferedWriter out;
13
14   public Logger() {
15     try {
16       fstream = new FileWriter("Lucene_Logger.txt");
17       out = new BufferedWriter(fstream);
18     } catch (IOException e) {
19       System.err.println("Error: " + e.getMessage());
20     }
21
22   }
23
24   public static void writeToLog(String msg) {
25     try {
26       out.write(msg + "\n");
27       // out.close();
28     } catch (IOException e) {
29       System.err.println("Error: " + e.getMessage());
30     }
31
```

```
32    }
33
34 }
```

Listing A.2: Logger.java

A.1.3. Indexer.java

```
1  package zhaw;
2
3  import java.io.File;
4  import java.io.FileFilter;
5  import java.io.IOException;
6  import java.io.Reader;
7  import java.util.ArrayList;
8  import java.util.Calendar;
9  import java.util.Date;
10
11 import org.apache.lucene.document.DateTools;
12 import org.apache.lucene.document.Document;
13 import org.apache.lucene.document.Field;
14
15 public class Indexer {
16
17   String indexDir = "NULL";
18   int numIndexed = 0;
19   static PDFIndexer _pdfindexer;
20   static OfficeDocIndexer _officeindexer;
21   static TextFileIndexer _textfileindexer;
22   static final char FILE_SEPARATOR = System.getProperty("file.separator").charAt
         (0);
23   private DateTools.Resolution dateTimeResolution = DateTools.Resolution.SECOND;
24   ArrayList<String> _validFileextensions = new ArrayList<String>();
25
26   public Indexer() {
27     indexDir = Main.getIndexDir();
28   }
29
30   /**
31    * update all valid file extensions
32    */
```

```
33  public void updateValidFileExtensions() {
34    String[] tmpvalid = _pdfindexer.getvalidFileextensions();
35    for (int i = 0; i < tmpvalid.length; i++) {
36      _validFileextensions.add(tmpvalid[i]);
37    }
38    tmpvalid = _officeindexer.getvalidFileextensions();
39    for (int i = 0; i < tmpvalid.length; i++) {
40      _validFileextensions.add(tmpvalid[i]);
41    }
42    tmpvalid = _textfileindexer.getvalidFileextensions();
43    for (int i = 0; i < tmpvalid.length; i++) {
44      _validFileextensions.add(tmpvalid[i]);
45    }
46
47  }
48
49  /**
50   * create all Sub-indexer "PDF", "TEXT", "OFFICE"
51   */
52  public void createSubindexer() {
53    _officeindexer = new OfficeDocIndexer();
54    _pdfindexer = new PDFIndexer();
55    _textfileindexer = new TextFileIndexer();
56    updateValidFileExtensions();
57  }
58
59  /** returns the File Function "TEXT", "PDF", "OFFICE" */
60  public String getFileExtensionFunction(String ext) {
61
62    String[] tmpvalid = _pdfindexer.getvalidFileextensions();
63    for (int i = 0; i < tmpvalid.length; i++) {
64      if (ext.equals(tmpvalid[i]))
65        return "PDF";
66    }
67
68    tmpvalid = _officeindexer.getvalidFileextensions();
69    for (int i = 0; i < tmpvalid.length; i++) {
70      if (ext.equals(tmpvalid[i]))
71        return "OFFICE";
72    }
73
```

```
74      tmpvalid = _textfileindexer.getvalidFileextensions();
75      for (int i = 0; i < tmpvalid.length; i++) {
76        if (ext.equals(tmpvalid[i]))
77          return "TEXT";
78      }
79
80      return "NULL";
81
82    }
83
84    /* return true if file extension is in index, else false */
85    public boolean isValidFileExtension(String ext) {
86      for (int i = 0; i < _validFileextensions.size(); i++) {
87        if (ext.equals(_validFileextensions.get(i))) {
88          return true;
89        }
90      }
91
92      return false;
93    }
94
95    public int index(Indexer indexer, String dataDir, FileFilter filter, int count)
          throws Exception {
96      String[] subdirectories = Main.getMyFunctions().getSubDirectories(dataDir);
97
98      /* Print on console every subfolder found */
99      try {
100       for (int i = 0; i < subdirectories.length; i++) {
101         // System.out.println("Found subdirectory: " +
102         // subdirectories[i]);
103       }
104     } catch (Exception e) {
105       // System.out.println("No subdirectories ...");
106     }
107
108     File[] files = new File(dataDir).listFiles();
109
110     /* iterate over all files and index it */
111     for (File f : files) {
112       // System.out.println("Filename = " + f.getName());
```

```
113         if (!f.isDirectory() && !f.isHidden() && f.exists() && f.canRead() && (filter
                == null || filter.accept(f))) {
114           myFunctions.prepareindexFile(f);
115         }
116         for (int i = 0; i < subdirectories.length; i++) {
117           String subdir = dataDir + subdirectories[i] + "/";
118           int tmpcount = indexer.index(indexer, subdir, new TextFilesFilter(), count);
119           count += tmpcount;
120         }
121
122       }
123       return Main.getwriter().numDocs();
124     }
125
126     /** will close the writer of the index Files */
127     public void closeWriter() throws IOException {
128       System.out.println("Optimizing index...");
129       Main.getwriter().optimize();
130       Main.getwriter().close();
131     }
132
133     /** returns the PDF Indexer */
134     public static zhaw.PDFIndexer getPDFIndexer() {
135       return _pdfindexer;
136     }
137
138     /** returns the Office Indexer */
139     public static zhaw.OfficeDocIndexer getOfficeIndexer() {
140       return _officeindexer;
141     }
142
143     /**
144      * Get the Lucene data time resolution.
145      *
146      * @return current date/time resolution
147      */
148     protected DateTools.Resolution getDateTimeResolution() {
149       return dateTimeResolution;
150     }
151
152     /**
```

```
153     * Set the Lucene data time resolution.
154     *
155     * @param resolution
156     *                set new date/time resolution
157     */
158    protected void setDateTimeResolution(DateTools.Resolution resolution) {
159      dateTimeResolution = resolution;
160    }
161
162    protected String timeToString(long time) {
163      return DateTools.timeToString(time, dateTimeResolution);
164    }
165
166    protected void addTextField(Document document, String name, Reader value) {
167      if (value != null) {
168        document.add(new Field(name, value));
169      }
170    }
171
172    protected void addTextField(Document document, String name, String value) {
173      if (value != null) {
174        document.add(new Field(name, value, Field.Store.YES, Field.Index.ANALYZED))
               ;
175      }
176    }
177
178    protected void addTextField(Document document, String name, Date value) {
179      if (value != null) {
180        addTextField(document, name, DateTools.dateToString(value, dateTimeResolution
               ));
181      }
182    }
183
184    protected void addTextField(Document document, String name, Calendar value) {
185      if (value != null) {
186        addTextField(document, name, value.getTime());
187      }
188    }
189
190    protected static void addUnindexedField(Document document, String name, String
         value) {
```

```
191     if (value != null) {
192       document.add(new Field(name, value, Field.Store.YES, Field.Index.NO));
193     }
194   }
195
196   protected void addUnstoredKeywordField(Document document, String name, String
          value) {
197     if (value != null) {
198       document.add(new Field(name, value, Field.Store.NO, Field.Index.
              NOT_ANALYZED));
199     }
200   }
201
202   protected void addKeywordField(Document document, String name, String value) {
203     if (value != null) {
204       document.add(new Field(name, value, Field.Store.YES, Field.Index.
              NOT_ANALYZED));
205     }
206   }
207 }
```

Listing A.3: Indexer.java

Literaturverzeichnis

[1] *Lucene Query Parser Syntax.* http://lucene.apache.org/core/2_9_4/queryparsersyntax.html, may 2014. 5.3.7

[2] McCandless, Michael; Hatcher, Erik; Gospondenetic Otis: *Lucene in Action - Second Edition.* Manning Publications Co, 2010.