



INFORMATION RETRIEVAL

Evaluierung der Retrieval-Leistung einer Search
Engine am Beispiel der Suche in Dokumenten von 4
Jahren ZHAW

Seminararbeit FS 2014

Student: Micha Schönenberger

Dozent: Dr. Ruxandra Domenig

© 2014

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Verzeichnis der Listings	V
1. Versionierung	1
2. Aufwände	2
3. Einleitung	3
3.1. Das Projekt	3
3.1.1. Ausgangslage	3
3.1.2. Ziel der Arbeit	3
3.1.3. Aufgabenstellung	4
3.1.4. Erwartetes Resultat	5
3.1.5. Geplante Termine	5
4. Die Search-Engine	6
4.1. eingesetzte Software	6
4.2. eingesetzte Hardware	7
4.3. Lucene	7
4.4. Definition der zu indizierenden Feldern	9
4.5. Aufbau Java Programm	11
4.6. Der Indexer	13
4.6.1. TextFile Indexer	13
4.6.2. PDF Indexer	14
4.6.3. OfficeDoc Indexer	14
4.6.4. Suchresultate	15
5. Vorbereitung für Vergleich	16
5.1. Windows Index Suche	16
5.2. Syntax für Suche mit Lucene / Luke	18
5.2.1. Felder	18
5.2.2. Wildcard Suche	18
5.2.3. Fuzzy Suche	19

5.2.4. Proximity Suche (Nachbarschaftliche)	19
5.2.5. Range Suche (Bereichssuche)	19
5.2.6. Boosting Suche (verstärktes Suchen)	19
5.2.7. Boolesches Suchen	20
6. Vergleich der Ergebnisse	21
6.1. Vergleich Lucene mit Windows 8.1 Pro	22
6.1.1. Suche #1 mit Windows 8.1 Professional	22
6.1.2. Suche #1 mit Lucene und Luke	24
6.1.3. Fazit zur Suche #1	25
6.1.4. Kurzer Exkurs zu Analyzern	26
6.2. Vergleich 2 Lucene mit Windows 8.1 Pro	27
6.2.1. Suche #1 mit Windows 8.1 Professional	27
6.2.2. Suche #2 mit Lucene und Luke	28
6.2.3. Fazit zur Suche #2	29
7. Fazit	30
A. Anhang	i
A.1. Java Code	i
A.1.1. Main.java	i
A.1.2. Logger.java	iii
A.1.3. Indexer.java	iv
A.1.4. TextFileIndexer.java	x
A.1.5. PDFIndexer.java	xi
A.1.6. OfficeDocIndexer.java	xix
A.1.7. myFunctions.java	xxviii
A.1.8. TextFilesFilter	xxxi
Index	xxxii
Literaturverzeichnis	xxxiii

Abbildungsverzeichnis

4.1. Klassendiagramm der Java-Applikation	11
5.1. Windows Indexing - Default Einstellungen	16
5.2. Such-Maske von Luke	18
6.1. Gewohnte Windows Suche 1	22
6.2. Erweiterte Suche 1 mit Windows	23
6.3. Suche 1 mit Lucene	24
6.4. Suche 1 mit Lucene mit English-Analyzer	25
6.5. Gewohnte Windows Suche 2	27
6.6. Erweiterte Suche 2 mit Windows 1	27
6.7. Erweiterte Suche 2 mit Windows 2	28
6.8. Suche 2 mit Lucene	28
6.9. Angepasste Suche 2 mit Lucene	29

Tabellenverzeichnis

1.1. Versionierung Dokumentation	1
2.1. Aufwände Seminararbeit	2
3.1. geplante Termine	5

Verzeichnis der Listings

4.1. Auszug aus Indexer.java	12
4.2. getValidFileextension()	13
6.1. Auszug aus Index von Dokumentation SW Projekt 2. Semester	26
A.1. Main.java	i
A.2. Logger.java	iii
A.3. Indexer.java	iv
A.4. TextFileIndexer.java	x
A.5. PDFIndexer.java	xi
A.6. OfficeDocIndexer.java	xix
A.7. myFunctions.java	xxviii
A.8. TextFilesFilter	xxxi

1. Versionierung

Version	Datum	Beschreibung
V0.1	18.03.2014	Erstellung 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
V0.9	29.05.2014	Kapitel Dokumentation, Probleme Indexgrösse Kapitel 6
V1.0	30.05.2014	Kapitel Vergleich der Ergebnisse, Dokumentation überarbeiten
V1.1	07.06.2014	Dokumentation überarbeiten

Tabelle 1.1.: Versionierung Dokumentation

2. Aufwände

Datum	Zeit	Beschreibung
18.03.2014	3.0h	Kickoff-Metting
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	-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)
14.05.2014	2.75h	-Dokumentation Kapitel 3.1.3 bis 3.1.4 -Quellenverzeichnis erstellen -Kapitel 4.1 eingesetzte Software -Kapitel 5.2 Syntax für Suche mit Lucene / Luke -Einarbeiten in Query mit Lucene/Luke
15.05.2014	2.75h	-Kapitel 5.2 Syntax für Suche mit Lucene / Luke
17.05.2014	2.25h	-Kapitel 5 mit teils Unterkapiteln -Kapitel 4.2 eingesetzte Hardware
18.05.2014	3.75h	-Kapitel 5 mit teils Unterkapiteln -Code Refactoring and cleaning
19.05.2014	7.25h	-Implementierung Office-Doc Indexer
20.05.2014	5.75h	-Fehlersuche Office-Doc Indexer -Finetuning Fields indexing alle Indexer
28.05.2014	4.75h	-Code Refactoring and cleaning -Probleme mit Index-Grösse
29.05.2014	8h	-Probleme mit Index-Grösse -Kapitel 6
30.05.2014	8.75h	-Kapitel 6.2.1 -Kapitel 6 fertig stellen -Probleme mit content Indizierung von PDF beheben
07.06.2014	3.5h	-Dokumentation komplett überarbeiten für finale Version

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.

Da viele 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 indiziert werden, was bei der Search-Engine des Windows Betriebssystems nicht standardmässig 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 Verbesserungsvorschlägen 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

Dieser Teilbereich wird im Kapitel 4.4 behandelt.

2. Erstellen einer einfachen Demo-Applikation in Java

Dieser Teilbereich wird im Kapitel 4.5 behandelt.

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 indizieren.

Dieser Teilbereich wird im Kapitel 4.6.2 behandelt.

5. Überprüfung der Suchresultate und Optimierung von Lucene

Behandelt im Kapitel 4.6.4 und zusätzliche Implementierung von Office Indexer (siehe Kapitel 4.6.3)

6. Reale Suche von mindestens zwei Begriffen in Lucene und direkter Vergleich mit den Suchresultaten des Betriebssystems

Behandelt in Kapitel 6

7. Fazit der Implementierung von Lucene. Sind die Suchresultate besser als diejenigen des Betriebssystems?

Dieser Teilbereich wird im Kapitel 7 behandelt.

3.1.4. Erwartetes Resultat

1. PDF sollen in den Suchresultaten mitberücksichtigt werden

Implementiert gemäss Kapitel 4.6.2

2. Die Eingabe der Suchbegriffe soll intuitiv und einfach gehalten werden

Suchbegriffe gemäss Standards von Lucene gemäss Kapitel 5.2

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

Die Vergleiche der Suchresultate werden im Kapitel 6 behandelt.

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 Indizierung)
<http://www.apache.org/dyn/closer.cgi/lucene/java/3.0.2>
- Apache PDFBox Version 1.8.5 (Open Source Library für Indizierung von PDF)
<http://pdfbox.apache.org/downloads.html>
- Luke - Lucene Index Toolbox - v 3.5.0 (2011-12-28)
stellt indizierte 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)

4.3. Lucene

Der Apache Lucene Core ist in Java geschrieben und ist eine frei verfügbare Information Retrieval Software Bibliothek.

Der grosse Vorteil von Lucene ist, dass die Software unter der Apache License frei verfügbar ist und unter den Entwicklern eine sehr grosse Akzeptanz findet. Auf die Frage, wo Lucene eingesetzt wird, sind online viele Antworten zu finden. Anbei eine kleine Liste von bekannten Firmen, welche Software des Apache Lucene Projektes einsetzen (Lucene Core, PyLucene, Solr):

- HP benutzt Solr
- Apple benutzt Solr
- Cisco benutzt Solr als Core in der Social Networking Suche
- Instagram nutzt Solr für die Geo-Search API
- Boing benutzt Solr
- Ford benutzt Solr
- ...

Anmerkung: Solr ist Teil des Apache Lucene Projektes. Solr ist ein Enterprise Search-Server¹ Quelle: [3]

Lucene selber ist eine Volltext-Such-Bibliothek, welche in Java geschrieben ist. Dies macht es für einen Programmierer einfach, eine effiziente Volltext-Suche in ihr Programm oder eine Webapplikation zu implementieren. Das Konzept von Lucene besteht aus folgenden (nicht abschliessenden) Punkten, welche kurz erläutert werden:

- indexing and searching (indizieren und suchen)
Damit Lucene bei einer Suchanfrage schnelle und präzise Antworten liefern kann, werden vorgängig die relevanten Dokumente indiziert. Der Index wird als sogenannter invertierter Index abgespeichert.
Invertierter Index bedeutet, dass z.B. nicht gespeichert wird, dass auf der Seite X die Wörter A, B und C stehen, sondern genau umgekehrt. Es wird abgespeichert, dass das Wort A auf der Seite X zu finden ist (genau so bei B und C).
Auf diese Weise wird erreicht, dass die Suche sehr effizient ist, denn Lucene kann sofort sagen, in welchem Dokument das gesuchte Wort vorhanden ist.
- document (= Dokument)
Ein Dokument in Lucene ist die Einheit für die Indizierung und die Suche. Es ist eine Aneinanderreihung von Feldern.
Ein Dokument in Lucene ist nicht zu verwechseln mit einem Dokument auf einer Dateiablage. Es entspricht in Lucene nicht einem gewöhnlichem Dokument.
- field (= Feld)
Ein Feld ist eine benannte Aneinanderreihung von Begriffen. Jedes Feld hat einen Namen und einen Textwert. Ein Feld kann in einem Lucene Dokument gespeichert werden. Ist dies der Fall, wird es mit den Suchtreffern des Dokumentes zurückgegeben.
Wird das Feld bei der Indizierung nicht dem Dokument zugewiesen, wird später über die Suche das Dokument über dieses Feld nicht mehr gefunden.

Quelle: [1]

¹Details siehe: <http://lucene.apache.org/solr/>

4.4. Definition der zu indizierenden Feldern

Die Suchkriterien, welche im Kapitel 3.1.3 gefordert werden, müssen vor der Entwicklung der Java Applikation definiert werden.

Die grundlegende Frage stellt sich aus den Anforderungen, nach welchen Kriterien ein Dokument gesucht werden kann.

Für diese Arbeit wurden folgende Felder definiert, welche indiziert werden sollen:

- **content**

Dieses Feld beinhaltet den Inhalt eines Dokumentes. Es wird den grössten Teil der indizierten Felder ausmachen. Grundsätzlich stehen hier alle Wörter, welche in irgendeinem der Dokumente vorkommen.

Bsp: «1», «hausfriedensbruch», «private»

- **filename**

Dieses Feld beinhaltet den kompletten Dokumentennamen. filename ist nicht zwingend unique

Bsp: «Programm.txt»

- **fullpath**

Dieses Feld beinhaltet den kompletten Pfad des indizierten Dokumentes. Da es pro Pfad nur ein File geben kann, ist dieses Feld unique

Bsp: «/Users/micha/ZHAW/1.Jahr/Programmieren/Uebung1/Main.java»

- **FileExtension**

Diese Feld beinhaltet lediglich die Endung des Files. Um die Files mit den verschiedenen Endungen richtig einlesen zu können benötigt man verschiedene Indexer (Text-Indexer, PDF-Indexer, legacy Word-Indexer...)

Bsp: «txt», «java»

- **Author**

Dieses Feld beinhaltet den Namen des Authors. Es kann nur aus einem Teil der Dokumente gelesen werden, sofern diese Information verfügbar ist. So kann der Author in einem Office Dokument oder einem PDF Dokument stehen, in einem .java oder .txt Dokument fehlt dieser Eintrag.

Bsp: «micha», «schönenberger»

- **Title**

Dieses Feld beinhaltet den Titel des Dokumentes. Wie beim Feld Author kann es nur aus einem Teil der Dokumente ausgelesen werden.

Bsp: «prtg», «analyse»

- **CreationDate**

Dieses Feld beinhaltet das Datum der Erstellung des Dokuments in der Form von YYYY-MM-DD-HH-MM-SS. Wie beim Feld Author kann es nur aus einem Teil der Dokumente ausgelesen werden. Grundsätzlich wäre es möglich, über die Systemstruktur (anstelle über das File direkt) diese Information herauszulesen. Aus Zeitgründen wurde dies aber nicht implementiert bei dieser Arbeit.

Bsp: «20110622101000», «20140528130112»

- **ModificationDate**

Dieses Feld beinhaltet das Datum der letzten Änderung des Dokuments in der Form von YYYY-MM-DD-HH-MM-SS. Wie beim Feld Author kann es nur aus einem Teil der Dokumente ausgelesen werden. Grundsätzlich wäre es möglich, über die Systemstruktur (anstelle über das File direkt) diese Information herauszulesen. Aus Zeitgründen wurde dies aber nicht implementiert bei dieser Arbeit.

Bsp: «20110622101000», «20140528130112»

Es gibt noch weitere Felder wie appname, summary..., welche hier nicht speziell erwähnt werden. Sie funktionieren momentan nur bei PDF Dokumenten.

4.5. Aufbau Java Programm

Um einen kurzen Überblick über die Java-Applikation zu erhalten, soll das Klassendiagramm der Applikation ein Grundverständnis liefern.

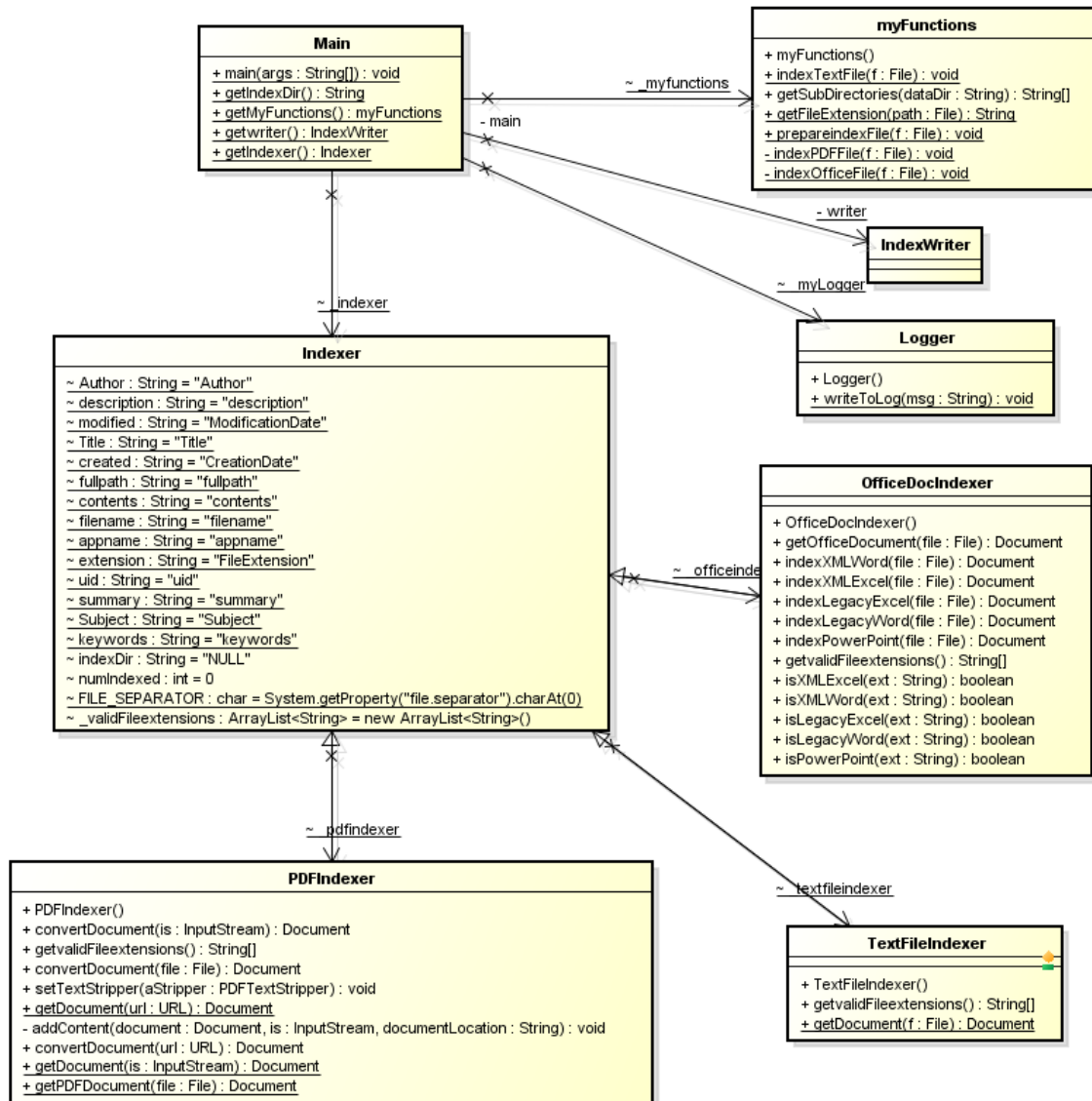


Abbildung 4.1.: Klassendiagramm der Java-Applikation

Quelle: eigener Screenshot

Die Main-Klasse startet die Applikation und initialisiert die notwendigen Komponenten wie den Ordner, welcher indiziert werden sollte oder die Initialisierung der Indexer Klasse. Danach übergibt die Main-Klasse die komplette Rechenarbeit der Indexer-Klasse.

Die Indexer-Klasse führt den Scan aller Dokumente und Ordner durch, übergibt der entsprechenden Klasse die Files und ruft für die gefundenen Ordner rekursiv die eigene Funktion wieder auf.

Folgender Code zeigt das Vorgehen des rekursiven Aufrufes:

```
1 public int index(Indexer indexer, String dataDir, FileFilter filter , int count) throws
   Exception {
2     File[] files = new File(dataDir).listFiles ();
3
4     /* iterate over all files and index it */
5     for (File f : files ) {
6         /* if it's a file and ... index it */
7         if (!f.isDirectory() && !f.isHidden() && f.exists() && f.canRead() && (filter
            == null || filter .accept(f))) {
8             Logger.writeToLog("* File Name = " + f.getCanonicalPath());
9             myFunctions.prepareindexFile(f);
10            count++;
11        }
12        /* if its a directory, do: */
13        else if (f.isDirectory() && !f.isHidden()) {
14            Logger.writeToLog("# Directory Name = " + f.getCanonicalPath());
15            String subdir = f.getAbsolutePath();
16            int tmpcount = indexer.index(indexer, subdir, new TextFilesFilter(), count);
17            count += tmpcount;
18        }
19        ...
20    }
```

Listing 4.1: Auszug aus Indexer.java

4.6. Der Indexer

Gegenüber anderen Aufgabenstellungen wie eine Indizierung von reinen Text-Dokumenten oder einer MP3-Sammlung erfordert diese Aufgabenstellung eine Implementierung mehrere Indexer. Lucene selber bietet sehr bequem eine Indizierung von reinen Text-Dokumenten.

Damit Lucene weiss, welche Indexer für welches File benutzt werden muss, ist es notwendig, jede mögliche Endung eines Dokumentes einem Indexer zuzuweisen. Dazu besitzt jeder Indexer die Methode `getValidFileextension()`. Das Listing 4.2 zeigt die Methode des `TextFileIndexer.java`

```
1  /** return all valid file extensions for this Indexer */
2  public String[] getvalidFileextensions () {
3      String[] retString = { "c", "txt", "java", "h" };
4      return retString;
5  }
```

Listing 4.2: `getValidFileextension()`

Anmerkung:

Aufgrund des Aufwandes für die Impelementierung von verschiedenen Indexer wurde bei dieser Arbeit auf eine Analyse und Implementation von Stop-Wörtern und/oder vor definierten Boosts verzichtet. Bei der Suche nach Dokumenten kann, sofern gewünscht, der Boost eines Attributes selber mitgegeben werden (siehe Kapitel 5.2.6).

4.6.1. TextFile Indexer

Der `TextFileIndexer.java` beinhaltet die Indizierung von reinen Text-Files.

- Mögliche Endungen: *.txt, *.c, *.java, *.h
- Mögliche indizierte Felder: content, filename, fullpath, FileExtension

Der komplette Code ist im Anhang A.1.4 zu finden.

4.6.2. PDF Indexer

Der PDFIndexer.java beinhaltet die Indizierung von nicht geschützten PDF Dokumenten. Dazu ist Apache PDFBox Library notwendig, wie im Kapitel 4.1 erwähnt wurde.

- Mögliche Endungen: *.pdf
- Mögliche indizierte Felder: content, filename, fullpath, FileExtension, Author, summary, Subject, Title, CreationDate, ModifiedDate

Der komplette Code ist im Anhang A.1.5 zu finden.

4.6.3. OfficeDoc Indexer

Der OfficeDocIndexer.java beinhaltet die Indizierung von Microsoft Office Dokumenten. Hierbei gilt es jedoch zu beachten, dass es mehrere Extractors gibt, welche die verschiedenen Dokumente indizieren.

- ExcelExtractor
Indiziert ältere Excel Formate: *.xls, *.xlt, *.xlm
- WordExtractor
Indiziert ältere Word Formate: *.doc, *.dot
- XSSFExcelExtractor
Indiziert neuere Excel Formate, welche auf XML basieren:
*.clsx, *.xlsm, *.xltx, *.xltm
- XWPFWordExtractor
Indiziert neuere Word Formate, welche auf XML basieren: *.docx, *.dotx, *.dotm

Die Implementierung dieser Dokumente hat einiges an Zeit benötigt. Vor allem auch deshalb, weil die Vorkenntnisse nicht vorhanden waren, dass es für Excel und Word je zwei Extractor gibt. So wurden die XML-basierten Office-Dokumente indiziert, jedoch mit einem sinnlosen, unbrauchbaren Index.

Aufgrund von fehlenden Zeitressourcen wurde auf die korrekte Implementierung von PowerPoint Dokumenten verzichtet.

- Mögliche indizierte Felder aller Office Indexer: content, filename, fullpath, FileExtension, Author, description, Title, CreationDate, ModifiedDate

Der komplette Code ist im Anhang A.1.6 zu finden.

4.6.4. Suchresultate

Die Qualität der Suchresultate kann nicht mittels eines Algorithmus oder mathematischen Formeln beurteilt werden. Er ist alleine abhängig vom Benutzer, der die Suche initialisiert hat. Er muss entscheiden, ob das Suchresultat dem entspricht, was er sich erhofft hat. Ist der Benutzer mit dem Resultat zufrieden heisst dies jedoch nicht zwingend, dass es kein besserer Resultat gegeben hätte.

Die Stichproben der Qualität der Suche mit der jetzigen Implementierung der Java-Applikation war überzeugend. Ob Lucene sich gegen die Windows Search-Engine behaupten kann, wird sich im Kapitel 6 zeigen.

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 indiziert. Gleichzeitig indiziert Windows nicht die kompletten Dateiinhalte, sondern lediglich die Einstellungen und deren Metadaten.

Um die Inhalte auch zu indizieren, muss dies in den Einstellungen geändert werden (siehe Abbildung 5.1 rechte Seite).

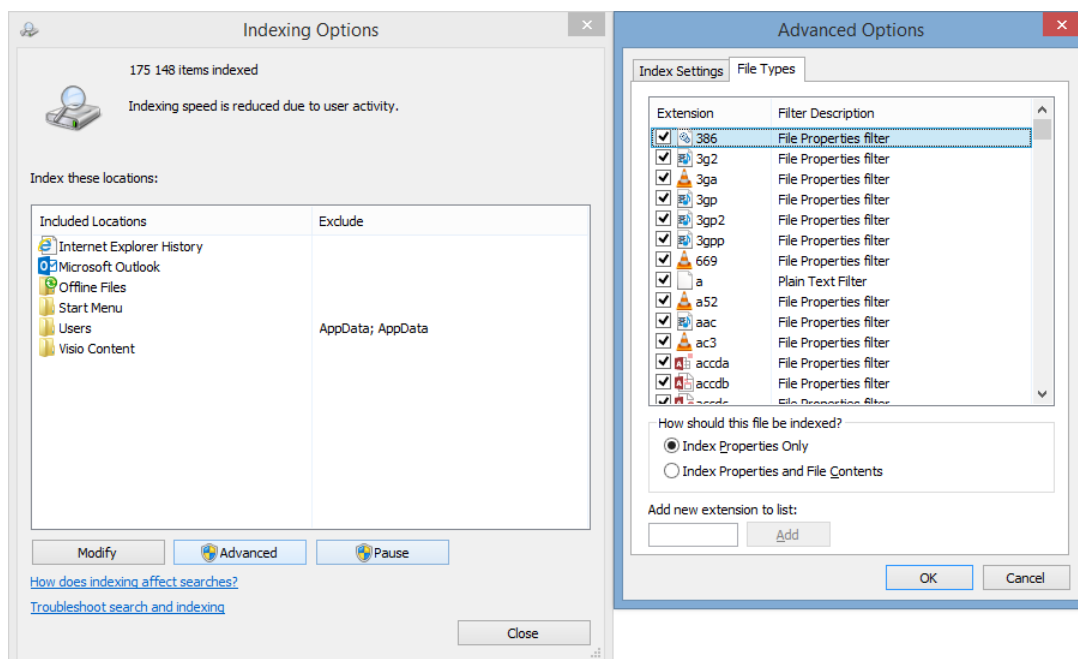


Abbildung 5.1.: Windows Indexing - Default Einstellungen
Quelle: eigener Screenshot

Um 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 verfügbar².

Da dies leider nach einem Neustart nicht funktionierte, wurde eine weitere Möglichkeit gesucht.

So wurden gemäss dem Microsoft Technet-Artikel³ folgende 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.

Da die Indizierung über den Softlink gemäss Anleitung leider nicht wie gewünscht funktionierte, wurden alle Schuldaten auf dem Windows Rechner auf die lokale Festplatte kopiert und indiziert. So kann sichergestellt werden, dass ein fairer Vergleich der Resultate stattfinden kann.

²<http://www.microsoft.com/en-us/download/details.aspx?id=3383>

³<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>

5.2. Syntax für Suche mit Lucene / Luke

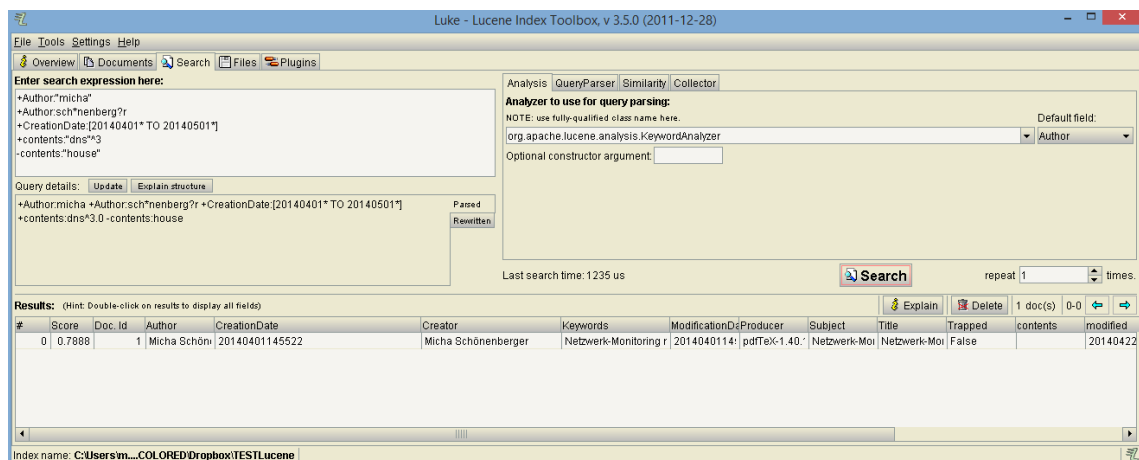


Abbildung 5.2.: Such-Maske von Luke
Quelle: eigener Screenshot

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

5.2.1. 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

5.2.2. 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 einer Suchanfrage stehen.

Anmerkung: Bei Luke gibt es jedoch eine Einstellung, mit der * am Anfang erlauben sein darf.

5.2.3. Fuzzy Suche

Die Fuzzy Suche basiert auf dem Levenshtein Distanz Algorithmus⁴, 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.

5.2.4. 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.
-------------------	---

5.2.5. 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)

5.2.6. 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.
--------------------	---

⁴siehe <http://www.levenshtein.de/>

5.2.7. Boolesches 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».
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 5.2: [2]

6. Vergleich der Ergebnisse

Wie in der Aufgabenstellung (siehe Kapitel 3.1.3) erwähnt, sollen die Suchresultate von Lucene mit denjenigen des Betriebssystems verglichen werden. Die einfache 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 indiziert wurden oder nicht, erscheint das Resultat.

Die Qualität der Suche wird nicht anhand der Geschwindigkeit ermittelt. Anhand der im Kapitel 4.2 aufgelisteten Hardware wäre es kein fairer Vergleich, eine SSD mit einer herkömmlichen HDD zu vergleichen.

Um einen Vergleich zu starten, wird ein beliebiges File ausgewählt. Danach wird anhand verschiedener Kriterien gesucht. Eine erlaube Suchanfrage ist zum Beispiel: File enthält «beispiel», eine nicht erlaube Suchanfrage: Der Filename ist «beispiel.txt».

6.1. Vergleich Lucene mit Windows 8.1 Pro

Die gewählte Datei, welche durch die Suche gefunden werden sollte ist:

«../1. Jahr/Software Projekt 1/Dokumentation SW Projekt 2. Semester V0.1.docx»

Annahme für die Suche

- Filename ist unbekannt
- Autor: entweder «micha» oder «reto»
- File Extension: *.doc oder *.docx
- Im Inhalt muss vorkommen: «Address.java».

6.1.1. Suche #1 mit Windows 8.1 Professional

Bei der klassischen Suche im Windows werden die gesuchten Attribute in die Suchmaske eingegeben:

Die meisten Windows-Benutzer würden hier 2 Suchanfragen starten. Die Erste mit «micha Address.java» wie Abbildung 6.1 zeigt, die Zweite mit «reto Address.java»

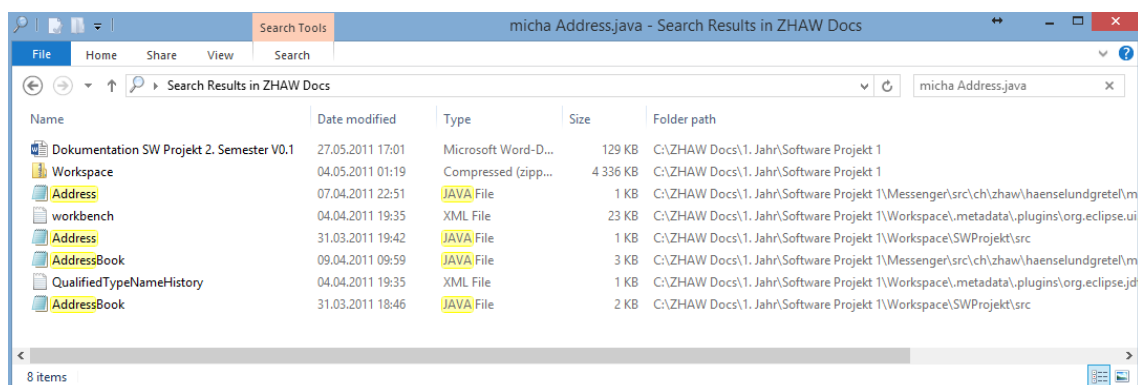


Abbildung 6.1.: Gewohnte Windows Suche 1

Quelle: eigener Screenshot

Durch das Auseinandersetzen mit der effizienten Windows-Suche wurde klar, dass Windows von Haus aus weit mehr beherrscht als nur nach Stichworten zu suchen. Erweiterte Tipps für die Suche mit Windows gibt es von Microsoft über <http://windows.microsoft.com/de-ch/windows7/advanced-tips-for-searching-in-windows>.

Werden die fortgeschrittenen Syntax bei der Suche eingehalten, sieht diese Anfrage folgendermassen aus: «authors:~="micha" OR authors:~="reto" AND Address.java». Diese Anfrage bedeutet, dass entweder «micha» oder «reto» im Feld Author vorhanden sein muss. Da «Address.java» nicht zugeordnet wurde, muss es irgendwo vorkommen.

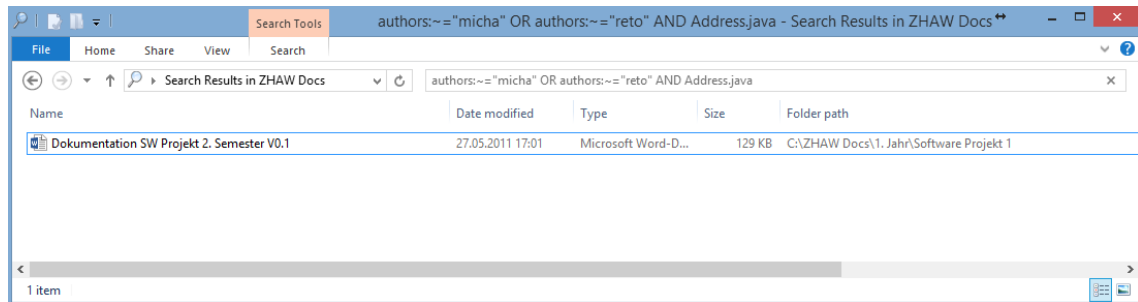


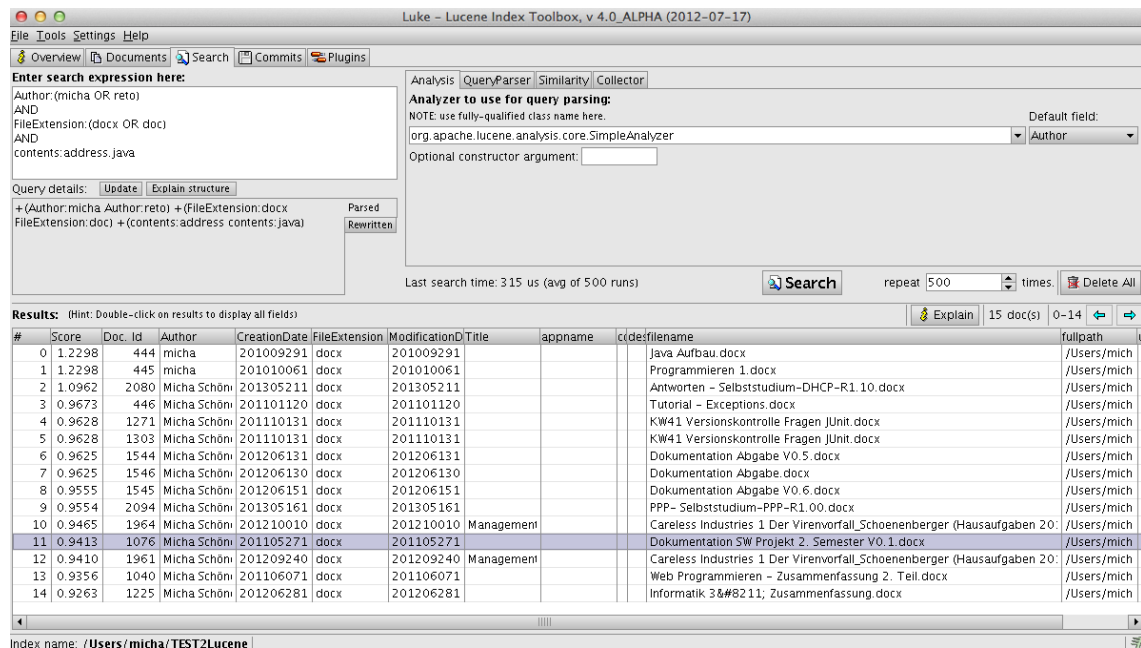
Abbildung 6.2.: Erweiterte Suche 1 mit Windows
Quelle: eigener Screenshot

Durch den Einsatz des optimierten Syntax für die Windows suche lässt das Resultat mit genau einem File dem Herausforderer Lucene bereits im Vorfeld keine grosse Chance, wie die Abbildung 6.2 zeigt.

6.1.2. Suche #1 mit Lucene und Luke

Die Suche mit Luke, welche auf die von Lucene indizierte Files zugreift, dauerte im Schnitt 300-500µs.

Wie die Abbildung 6.3 zeigt, werden hier mehrere Dateien gefunden. Die gesuchte Datei ist ebenfalls vorhanden (#11).



Search expression: Author:(micha OR reto) AND FileExtension:(docx OR doc) AND contents:address.java

Query details: Update Explain structure

Parser: Rewritten

Last search time: 315 us (avg of 500 runs)

Search repeat 500 times Delete All

Results: (Hint: Double-click on results to display all fields)

#	Score	Doc. Id	Author	CreationDate	FileExtension	ModificationD	Title	appname	c:\de\filename	fullpath
0	1.2298	444	micha	201009291	docx	201009291			Java Aufbau.docx	/Users/mich
1	1.2298	445	micha	201010061	docx	201010061			Programmieren 1.docx	/Users/mich
2	1.0962	2080	Micha Schöni	201305211	docx	201305211			Antworten - Selbststudium-DHCP-R1.10.docx	/Users/mich
3	0.9673	446	Micha Schöni	201101120	docx	201101120			Tutorial - Exceptions.docx	/Users/mich
4	0.9628	1271	Micha Schöni	201110131	docx	201110131			KW41 Versionskontrolle Fragen JUnit.docx	/Users/mich
5	0.9628	1303	Micha Schöni	201110131	docx	201110131			KW41 Versionskontrolle Fragen JUnit.docx	/Users/mich
6	0.9625	1544	Micha Schöni	201206131	docx	201206131			Dokumentation Abgabe V0.5.docx	/Users/mich
7	0.9625	1546	Micha Schöni	201206130	docx	201206130			Dokumentation Abgabe.docx	/Users/mich
8	0.9555	1545	Micha Schöni	201206151	docx	201206151			Dokumentation Abgabe V0.6.docx	/Users/mich
9	0.9554	2094	Micha Schöni	201305161	docx	201305161			PPP - Selbststudium-PPP-R1.00.docx	/Users/mich
10	0.9465	1964	Micha Schöni	201210010	docx	201210010	Management		Careless Industries 1. Der Virenvorfall_Schoenenberger (Hausaufgaben 20:	/Users/mich
11	0.9413	1076	Micha Schöni	201105271	docx	201105271			Dokumentation SW Projekt 2. Semester V0.1.docx	/Users/mich
12	0.9410	1961	Micha Schöni	201209240	docx	201209240	Management		Careless Industries 1. Der Virenvorfall_Schoenenberger (Hausaufgaben 20:	/Users/mich
13	0.9356	1040	Micha Schöni	201106071	docx	201106071			Web Programmieren - Zusammenfassung 2. Teil.docx	/Users/mich
14	0.9263	1225	Micha Schöni	201206281	docx	201206281			Informatik 3– Zusammenfassung.docx	/Users/mich

Index name: /Users/micha/TEST2Lucene

Abbildung 6.3.: Suche 1 mit Lucene
Quelle: eigener Screenshot

6.1.3. Fazit zur Suche #1

Die Windows Search Engine sowie die eigene Lucene Applikation finden beide das gewünschte Dokument. Beim Windows ist es das einzige Dokument, welches aufgelistet wird. Lucene hingegen listet das gesuchte Dokument an der 11. Stelle auf. Ist nun Lucene schlechter deswegen? Um auf diese Frage eine Antwort zu finden, müssen die anderen von Lucene ausgegebenen Dokumenten überprüft werden. Es könnte auch sein, dass Lucene alle Dokumente auflistet, Windows jedoch nur per “Zufall“ das Richtige. Was wäre, wenn das File “Java Aufbau.docx“, welches bei Lucene zuoberst steht, gesucht worden wäre?

Die beiden ersten Dokumente wurden genauer untersucht. Das Fazit hierbei ist, dass alle gesuchten Merkmale bei den Dokumenten vorhanden waren, ausser dem contents:address.java. «address.java» wurde in beiden Dokumenten nicht gefunden. Wo liegt der Fehler?

Wird die identische Suche mit Luke nochmals durchgeführt, jedoch anstelle des Simple-Analyzer der Standard, German- oder English-Analyzer eingesetzt, findet Lucene genau eine einzige Datei (siehe Abbildung 6.4). Das Resultat entspricht derjenigen Datei, die gefunden werden sollte.

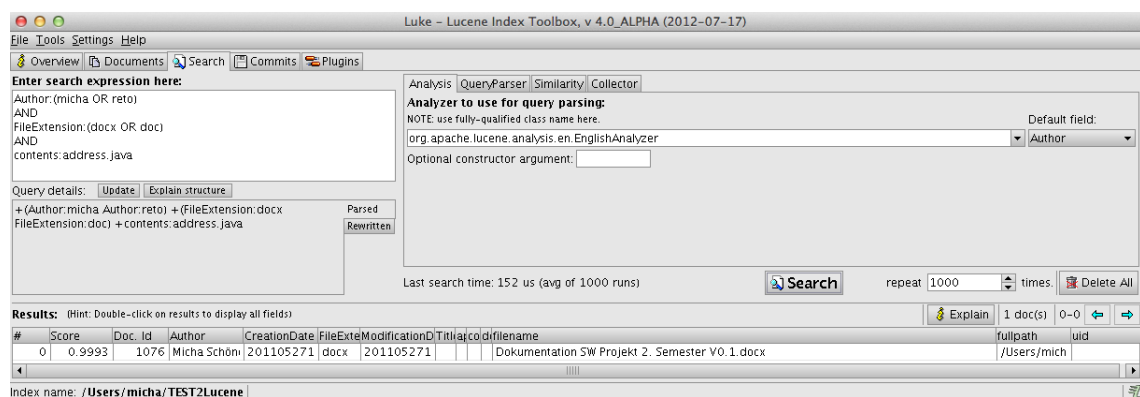


Abbildung 6.4.: Suche 1 mit Lucene mit English-Analyzer
Quelle: eigener Screenshot

6.1.4. Kurzer Exkurs zu Analyzern

Werden die verschiedenen Analyzer genauer betrachtet, wird das vermeintlich falsche Resultat logisch erklärbar.

Beim ersten Durchgang mit dem «SimpleAnalyzer» wird der indizierte Text bei Nicht-Schriftzeichen (z.B. «.», « », «!») getrennt und alle Buchstaben klein geschrieben.

Wird nun wie bei der Suche #1 jedoch «address.java» gesucht, ändert dies der Analyzer zu «address» und «java». Aus diesem Grund sind nun Dokumente auffindbar, welche «address.java» nicht enthalten. Bei genaueren Hinschauen sind jedoch die beiden Wörter einzeln vorhanden.

Beim zweiten Durchgang findet Luke genau das gesuchte Dokument. Als Beispiel wird hier der «StandardAnalyzer» betrachtet. Dieser basiert auf einer ausgefeilten Grammatik, welche unter anderem alphanumerische, chinesische, japanische oder koreanische Zeichen, E-Mail Adressen, Akronyme und vieles mehr erkennt. Zusätzlich werden alle Buchstaben wie beim SimpleAnalyzer klein geschrieben. Der «StandardAnalyzer» entfernt ebenfalls Stop-Wörter (auf die hier nicht eingegangen wird).

Anbei ein Auszug aus Luke, bei welchem der Index der gesuchten Datei rekonstruiert wurde. Hier ist ersichtlich, dass «address.java» in Zeile 2 korrekt indiziert wurde. Der SimpleAnalyzer erkennt diese beiden verknüpften Wörter im Gegensatz zum StandardAnalyzer jedoch als zwei einzelne Wörter.

```
1 ...
2 null_3 address.java null_1 used null_2 receivers null_3 ...
3 null_1 read from null_1 addressbook
4 null_3 checks null_1 validating we
5 expect correct entries validating null_1 addressbook null_1 ...
6 null_2 scope null_2 project.there null_2 setters null_1 ...
7 set null_1 values addressbook.java contains
8 ...
```

Listing 6.1: Auszug aus Index von Dokumentation SW Projekt 2. Semester

Quelle: [4], S. 127

6.2. Vergleich 2 Lucene mit Windows 8.1 Pro

Die gewählte Datei, welche durch die Suche gefunden werden sollte ist:

«../3. Jahr/Kryptologie/KK 05/05 - Slides der Woche 5.pdf»

Annahme für die Suche

- Filename ist unbekannt
- Im Inhalt muss vorkommen: «Inverse»
- File Extension: *.pdf

6.2.1. Suche #1 mit Windows 8.1 Professional

Zu Beginn soll Windows anhand der klassischen Benutzersuche die Anfrage starten.

Die Abbildung 6.5 zeigt, dass das gewünschte Dokument nicht vorhanden ist.

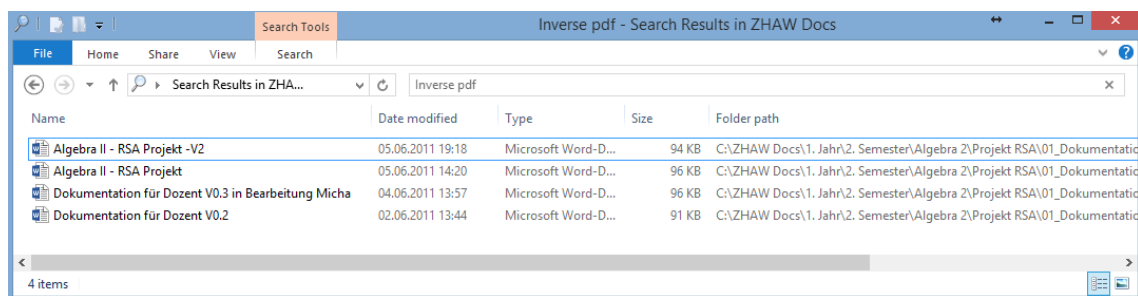


Abbildung 6.5.: Gewohnte Windows Suche 2
Quelle: eigener Screenshot

Deshalb soll nun versucht werden, mit dem erweiterten Windows Syntax das Dokument zu finden:

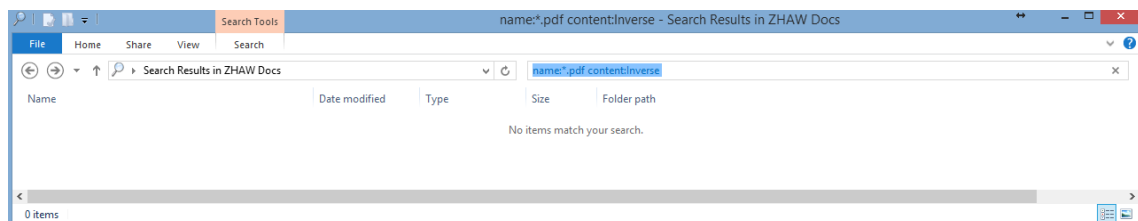


Abbildung 6.6.: Erweiterte Suche 2 mit Windows 1
Quelle: eigener Screenshot

Wie nicht anders erwartet, liefert Windows keine richtige Antwort, wie Abbildung 6.6 zeigt. Da bereits die herkömmliche Suche, welche nur nach «Invers» und «pdf» gesucht hat, kein einziges PDF gefunden hat, ist es nicht verwunderlich, dass bei der erweiterten Suche keine Treffer gab.

Die untenstehende Abbildung 6.7 zeigt jedoch, dass das Dokument existiert.

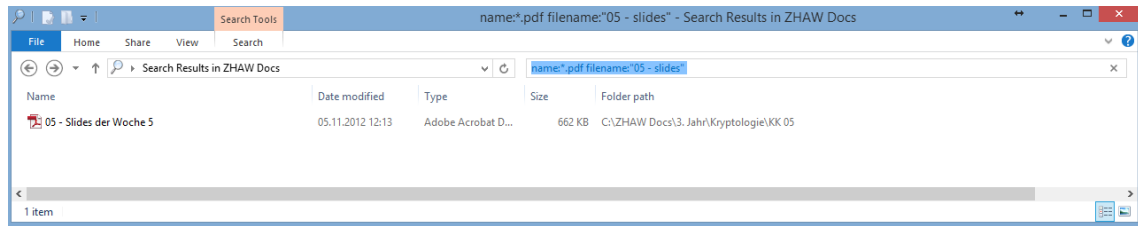


Abbildung 6.7.: Erweiterte Suche 2 mit Windows 2
Quelle: eigener Screenshot

6.2.2. Suche #2 mit Lucene und Luke

Bei der Suche mit Lucene findet Luke leider auch keinen Treffer.

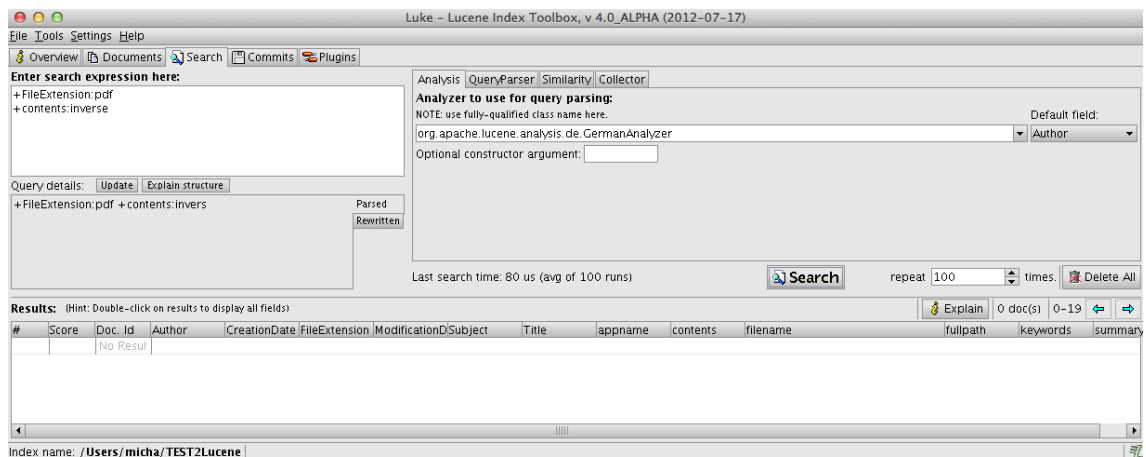


Abbildung 6.8.: Suche 2 mit Lucene
Quelle: eigener Screenshot

6.2.3. Fazit zur Suche #2

Weder Windows noch Lucene konnten das gesuchte Dokument finden. Woran könnte dies liegen?

Beim Windows wurde bereits bei der Abbildung 6.7 gezeigt, dass das Dokument grundsätzlich gefunden wird. Jedoch indiziert Windows nicht den Inhalt des PDF.

Die Frage stellt sich nun, ob die Lucene Java Applikation Fehler enthält, so dass das PDF Dokument zwar indiziert wurde, jedoch wie bei der Windows Suche nur die Metadaten, nicht aber die Inhalte indiziert wurden.

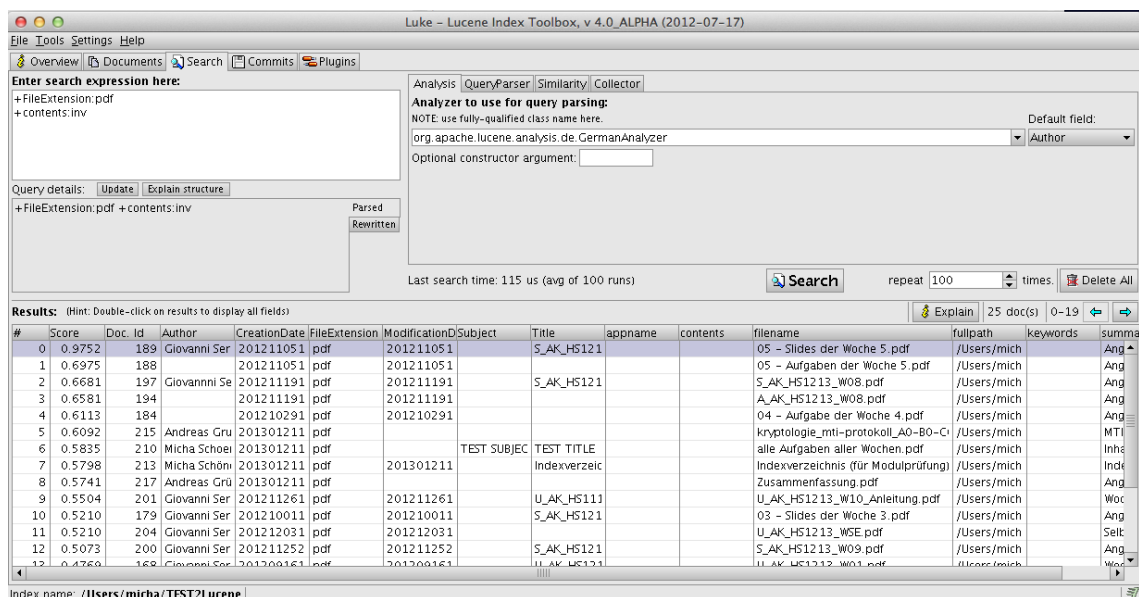


Abbildung 6.9.: Angepasste Suche 2 mit Lucene
Quelle: eigener Screenshot

Die Abbildung 6.9 zeigt eindeutig, dass Lucene den Inhalt der PDF Files indiziert hat und das gesuchte File auch mit der besten Trefferrate findet.

Beim näheren hinschauen ist ersichtlich, dass bei der Indizierung anstelle von «inverse» nur «inv» eingelesen wurde. Auf die möglichen Gründe wird hier nicht eingegangen.

Als Fazit der zweiten Suchanfrage kann man sagen, dass Lucene klar den Vorteil hat, dass PDF Inhalte indiziert werden und auch gefunden werden. Mit kleinen Anpassungen der Suchanfrage kann man das gesuchte Dokument ohne Probleme finden.

7. Fazit

Die Einarbeitung mit Lucene war nicht immer einfach. Vor allem wegen der verschiedenen Datei-Formaten. Die Nicht-Kompatibilität des Standard-Indexer von Lucene mit PDF und Office Dokumenten war recht schnell ersichtlich. Jedoch wirft Lucene keine Fehlermeldung, sondern indiziert die Dateien sinnlos, da der Inhalt falsch interpretiert wird.

Bei den Office-Dokumenten war es schwieriger. Anfangs wurde eine Indexer für Excel und ein Indexer für Word Dokumente eingesetzt. Jedoch wurden mit diesem Indexer nur ältere Office-Dokumente richtig indiziert. Für XML-basierte Office-Dokumente musste nachträglich nochmals ein eigener Indexer hinzugefügt werden.

Aufgrund dessen, dass vor Beginn der Arbeit noch nie mit Lucene gearbeitet wurde, war eine Einarbeitungszeit unabdingbar. Mittels eines Testordners mit verschiedenen Dateien und einem Subfolder wurde die Implementation der Java-Applikation getestet, was einwandfrei funktionierte. Leider war die Applikation in der Praxis beim Indizieren der Schuldateien nicht performant und es wurden hundertaussende Dateien indiziert. Das Problem wurde erst nach Stunden durch Debuggen entdeckt. Es versteckte sich in der Klasse Indexer.java in der Methode `public int indexer(...)`. Hier war ein rekursiver Aufruf falsch implementiert, was zur 1000fachen Indizierung einzelner Dateien führte.

Aufgrund der Bequemlichkeit von bereits vorhandenen Search-Engine (vor allem beim Arbeiten mit Mac OS X) würde ich persönlich kein Anlass sehen, im privaten Umfeld auf eine Lucene basierte Applikation umzusteigen, um meine Dateien indizieren zu lassen. Bei grösseren Datenmengen in Firmen oder bei Webapplikationen im geschäftlichen Umfeld jedoch wäre Lucene sicherlich eine sehr gute Möglichkeit für eine schnelle, kostengünstige und effiziente Indexierung und Suche.

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.de.GermanAnalyzer;
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/Test2Lucene/";
13     static String dataDir = "/Users/micha/Docs/Dokumente/Ausbildung, Weiterbildung
14         /Micha/ZHAW (2010–2014)/";
15     static myFunctions _myfunctions;
16     private static IndexWriter writer;
17     static Logger _myLogger;
18     static Indexer _indexer;
19
20     public static void main(String[] args) throws Exception {
21         _myLogger = new Logger();
22
23         long start = System.currentTimeMillis();
24         int numIndexed = 0;
25
26         /* prepare chosen directory and a IndexWriter */
27         Directory dir = FSDirectory.open(new File(indexDir));
28         writer = new IndexWriter(dir, new GermanAnalyzer(Version.LUCENE_30), true,
29             IndexWriter.MaxFieldLength.LIMITED);
```

```
29  /* init my functions */
30  _myfunctions = new myFunctions();
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(),
39          numIndexed);
40      numIndexed += _numIndexed;
41      System.out.println("Anzahl Files = " + numIndexed);
42  }
43  /* at end, close the writer */
44  finally {
45      _indexer.closeWriter();
46  }
47
48  long end = System.currentTimeMillis();
49  System.out.println("Indexing in " + (end - start) + " milliseconds");
50  }
51  /* returns the path of the directory, which should be indexed */
52  public static String getIndexDir() {
53      return indexDir;
54  }
55  /* returns my functions */
56  public static myFunctions getMyFunctions() {
57      return _myfunctions;
58  }
59  /* returns writer */
60  public static IndexWriter getwriter() {
61      return writer;
62  }
63  /* returns Indexer */
64  public static Indexer getIndexer() {
65      return _indexer;
66  }
```

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     /* static final Strings for Field names which can be indexed */
18     static final String Author = "Author";
19     static final String description = "description";
20     static final String modified = "ModificationDate";
21     static final String Title = "Title";
22     static final String created = "CreationDate";
23     static final String fullpath = "fullpath";
24     static final String contents = "contents";
25     static final String filename = "filename";
26     static final String appname = "appname";
27     static final String extension = "FileExtension";
28     static final String uid = "uid";
29     static final String summary = "summary";
30     static final String Subject = "Subject";
31     static final String keywords = "keywords";
32
33     String indexDir = "NULL";
34     int numIndexed = 0;
35     static PDFIndexer _pdfindexer;
36     static OfficeDocIndexer _officeindexer;
37     static TextFileIndexer _textfileindexer;
38     static final char FILE_SEPARATOR = System.getProperty("file.separator").charAt
        (0);
```



```

39 private DateTools.Resolution dateTimeResolution = DateTools.Resolution.SECOND;
40 ArrayList<String> _validFileextensions = new ArrayList<String>();
41
42 public Indexer() {
43     indexDir = Main.getIndexDir();
44 }
45
46 /**
47  * update all valid file extensions
48  */
49 public void updateValidFileExtensions() {
50     String[] tmpvalid = _pdfindexer.getvalidFileextensions();
51     for (int i = 0; i < tmpvalid.length; i++) {
52         _validFileextensions.add(tmpvalid[i]);
53     }
54     tmpvalid = _officeindexer.getvalidFileextensions();
55     for (int i = 0; i < tmpvalid.length; i++) {
56         _validFileextensions.add(tmpvalid[i]);
57     }
58     tmpvalid = _textfileindexer.getvalidFileextensions();
59     for (int i = 0; i < tmpvalid.length; i++) {
60         _validFileextensions.add(tmpvalid[i]);
61     }
62 }
63
64
65 /**
66  * create all Sub-indexer "PDF", "TEXT", "OFFICE"
67  */
68 public void createSubindexer() {
69     _officeindexer = new OfficeDocIndexer();
70     _pdfindexer = new PDFIndexer();
71     _textfileindexer = new TextFileIndexer();
72     updateValidFileExtensions();
73 }
74
75 /** returns the File Function "TEXT", "PDF", "OFFICE" */
76 public String getFileExtensionFunction(String ext) {
77
78     String[] tmpvalid = _pdfindexer.getvalidFileextensions();
79     for (int i = 0; i < tmpvalid.length; i++) {

```

```

80     if (ext.equals(tmpvalid[i]))
81         return "PDF";
82     }
83
84     tmpvalid = _officeindexer.getvalidFileextensions ();
85     for (int i = 0; i < tmpvalid.length; i++) {
86         if (ext.equals(tmpvalid[i]))
87             return "OFFICE";
88     }
89
90     tmpvalid = _textfileindexer.getvalidFileextensions ();
91     for (int i = 0; i < tmpvalid.length; i++) {
92         if (ext.equals(tmpvalid[i]))
93             return "TEXT";
94     }
95
96     return "NULL";
97
98 }
99
100 /* return true if file extension is in index, else false */
101 public boolean isValidFileExtension(String ext) {
102     for (int i = 0; i < _validFileextensions.size(); i++) {
103         if (ext.equals(_validFileextensions.get(i))) {
104             return true;
105         }
106     }
107
108     return false ;
109 }
110
111 public int index(Indexer indexer, String dataDir, FileFilter filter , int count)
112     throws Exception {
113     File[] files = new File(dataDir).listFiles ();
114
115     /* iterate over all files and index it */
116     for (File f : files) {
117         /* if it's a file and ... index it */
118         if (!f.isDirectory() && !f.isHidden() && f.exists() && f.canRead() && (filter
            == null || filter .accept(f))) {
            Logger.writeToLog("* File Name = " + f.getCanonicalPath());

```

```

119     myFunctions.prepareindexFile(f);
120     count++;
121 }
122 /* if its a directory, do: */
123 else if (f.isDirectory() && !f.isHidden()) {
124     Logger.writeToLog("# Directory Name = " + f.getCanonicalPath());
125     String subdir = f.getAbsolutePath();
126     int tmpcount = indexer.index(indexer, subdir, new TextFilesFilter(), count);
127     count += tmpcount;
128 }
129 }
130 return Main.getwriter().numDocs();
131 }
132
133 /** will close the writer of the index Files */
134 public void closeWriter() throws IOException {
135     System.out.println("Optimizing index...");
136     Main.getwriter().optimize();
137     Main.getwriter().close();
138     System.out.println("Finished with indexing .... ");
139 }
140
141 /** returns the PDF Indexer */
142 public static zhaw.PDFIndexer getPDFIndexer() {
143     return _pdfindexer;
144 }
145
146 /** returns the Office Indexer */
147 public static zhaw.OfficeDocIndexer getOfficeIndexer() {
148     return _officeindexer;
149 }
150
151 /** returns the Text Indexer */
152 public static zhaw.TextFileIndexer getTextFileIndexer() {
153     return _textfileindexer;
154 }
155
156 /**
157  * Get the Lucene data time resolution.
158  *
159  * @return current date/time resolution

```

```
160  */
161  protected DateTools.Resolution getDateTimeResolution() {
162      return dateTimeResolution;
163  }
164
165  /**
166   * Set the Lucene data time resolution.
167   *
168   * @param resolution
169   *         set new date/time resolution
170   */
171  protected void setDateTimeResolution(DateTools.Resolution resolution) {
172      dateTimeResolution = resolution;
173  }
174
175  protected void addTextField(Document document, String name, Reader value) {
176      if (value != null) {
177          document.add(new Field(name, value));
178      }
179  }
180
181  protected void addTextField(Document document, String name, String value) {
182      if (value != null) {
183          document.add(new Field(name, value, Field.Store.YES, Field.Index.ANALYZED))
184              ;
185      }
186  }
187
188  protected void addTextField(Document document, String name, Date value) {
189      if (value != null) {
190          addTextField(document, name, DateTools.dateToString(value, dateTimeResolution));
191      }
192  }
193
194  protected void addTextField(Document document, String name, Calendar value) {
195      if (value != null) {
196          addTextField(document, name, value.getTime());
197      }
198  }
```

```
199     protected static void addUnindexedField(Document document, String name, String
        value) {
200         if (value != null) {
201             document.add(new Field(name, value, Field.Store.YES, Field.Index.
                NOT_ANALYZED));
202         }
203     }
204
205     protected void addUnstoredKeywordField(Document document, String name, String
        value) {
206         if (value != null) {
207             document.add(new Field(name, value, Field.Store.NO, Field.Index.
                NOT_ANALYZED));
208         }
209     }
210
211     protected void addKeywordField(Document document, String name, String value) {
212         if (value != null) {
213             document.add(new Field(name, value, Field.Store.YES, Field.Index.
                NOT_ANALYZED));
214         }
215     }
216 }
```

Listing A.3: Indexer.java

A.1.4. TextFileIndexer.java

```
1 package zhaw;
2
3 import java.io.File;
4 import java.io.FileReader;
5 import org.apache.lucene.document.Document;
6 import org.apache.lucene.document.Field;
7
8 public class TextFileIndexer extends Indexer {
9
10     public TextFileIndexer() {
11     }
12
13     /** return all valid file extensions for this Indexer */
14     public String[] getvalidFileextensions () {
15         String[] retString = { "c", "txt", "java", "h" };
16         return retString;
17     }
18
19     static public Document getDocument(File f) throws Exception {
20         Document doc = new Document();
21         doc.add(new Field(Indexer.contents, new FileReader(f)));
22         doc.add(new Field(Indexer.filename, f.getName(), Field.Store.YES, Field.Index.
23             NOT_ANALYZED));
24         doc.add(new Field(Indexer.fullpath, f.getCanonicalPath(), Field.Store.YES, Field.
25             Index.NOT_ANALYZED));
26         String extension = myFunctions.getFileExtension(f);
27         doc.add(new Field(Indexer.extension, extension, Field.Store.YES, Field.Index.
28             NOT_ANALYZED));
29         return doc;
30     }
31 }
```

Listing A.4: TextFileIndexer.java

A.1.5. PDFIndexer.java

```
1  /*
2  * Licensed to the Apache Software Foundation (ASF) under one or more
3  * contributor license agreements. See the NOTICE file distributed with
4  * this work for additional information regarding copyright ownership.
5  * The ASF licenses this file to You under the Apache License, Version 2.0
6  * (the "License"); you may not use this file except in compliance with
7  * the License. You may obtain a copy of the License at
8  *
9  *      http://www.apache.org/licenses/LICENSE-2.0
10 *
11 * Unless required by applicable law or agreed to in writing, software
12 * distributed under the License is distributed on an "AS IS" BASIS,
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
14 * implied.
15 * See the License for the specific language governing permissions and
16 * limitations under the License.
17 */
18
19 package zhaw;
20
21 import java.io.File;
22 import java.io.FileInputStream;
23 import java.io.IOException;
24 import java.io.InputStream;
25 import java.io.Reader;
26 import java.io.StringReader;
27 import java.io.StringWriter;
28 import java.net.URL;
29 import java.util.Calendar;
30 import java.util.Date;
31
32 import org.apache.lucene.document.DateTools;
33 import org.apache.lucene.document.Document;
34 import org.apache.lucene.document.Field;
35 import org.apache.pdfbox.exceptions.CryptographyException;
36 import org.apache.pdfbox.exceptions.InvalidPasswordException;
37 import org.apache.pdfbox.pdmodel.PDDocument;
38 import org.apache.pdfbox.pdmodel.PDDocumentInformation;
39 import org.apache.pdfbox.searchengine.lucene.LucenePDFDocument;
40 import org.apache.pdfbox.util.PDFTextStripper;
```

```

39
40 /**
41  * This class is used to create a document for the lucene search engine. This
42  * should easily plug into the IndexHTML or IndexFiles that comes with the
43  * lucene project. This class will populate the following fields .
44  *
45  * @author <a href="mailto:ben@benlitchfield.com">Ben Litchfield</a>
46  * @version Revision : 1.23
47  */
48 public class PDFIndexer {
49     // given caveat of increased search times when using
50     // MICROSECOND, only use SECOND by default
51     private DateTools.Resolution dateTimeResolution = DateTools.Resolution.SECOND;
52
53     /**
54      * Constructor.
55      */
56     public PDFIndexer() {
57     }
58
59     /**
60      * Set the text stripper that will be used during extraction.
61      *
62      * @param aStripper
63      *         The new pdf text stripper.
64      */
65
66     /** return all valid file extensions for this Indexer */
67     public String[] getvalidFileextensions () {
68         String[] retString = { "pdf" };
69         return retString;
70     }
71
72     /**
73      * Get the Lucene data time resolution.
74      *
75      * @return current date/time resolution
76      */
77     public DateTools.Resolution getDateTimeResolution() {
78         return dateTimeResolution;
79     }

```



```

80
81  /**
82   * Set the Lucene data time resolution.
83   *
84   * @param resolution
85   *         set new date/time resolution
86   */
87  public void setDateTimeResolution(DateTools.Resolution resolution) {
88      dateTimeResolution = resolution;
89  }
90
91  @SuppressWarnings("unused")
92  private void addKeywordField(Document document, String name, String value) {
93      if (value != null) {
94          document.add(new Field(name, value, Field.Store.YES, Field.Index.
95                          NOT_ANALYZED));
96      }
97  }
98
99  private void addTextField(Document document, String name, Reader value) {
100      if (value != null) {
101          document.add(new Field(name, value));
102      }
103  }
104
105  private void addTextField(Document document, String name, String value) {
106      if (value != null) {
107          document.add(new Field(name, value, Field.Store.YES, Field.Index.ANALYZED))
108              ;
109      }
110  }
111
112  private void addTextField(Document document, String name, Date value) {
113      if (value != null) {
114          addTextField(document, name, DateTools.dateToString(value, dateTimeResolution
115                          ));
116      }
117  }
118
119  private void addTextField(Document document, String name, Calendar value) {
120      if (value != null) {

```

```

118     addTextField(document, name, value.getTime());
119     }
120 }
121
122 private static void addUnindexedField(Document document, String name, String
    value) {
123     if (value != null) {
124         document.add(new Field(name, value, Field.Store.YES, Field.Index.NO));
125     }
126 }
127
128 @SuppressWarnings("unused")
129 private void addUnstoredKeywordField(Document document, String name, String
    value) {
130     if (value != null) {
131         document.add(new Field(name, value, Field.Store.NO, Field.Index.
            NOT_ANALYZED));
132     }
133 }
134
135 /**
136  * Convert the PDF stream to a lucene document.
137  * @param is
138  *         The input stream.
139  * @return The input stream converted to a lucene document.
140  * @throws IOException
141  *         If there is an error converting the PDF.
142  */
143
144 /**
145  * This will take a reference to a PDF document and create a lucene
146  * document.
147  *
148  * @param file
149  *         A reference to a PDF document.
150  * @return The converted lucene document.
151  * @throws IOException
152  *         If there is an exception while converting the document.
153  */
154 @SuppressWarnings("static-access")
155 public Document convertDocument(File file) throws IOException {

```

```

156     Document document = new Document();
157
158     document.add(new Field(Indexer.filename, file.getName(), Field.Store.YES, Field.
159         Index.NOT_ANALYZED));
160     addTextField(document, Indexer.extension, Main.getMyFunctions().getFileExtension
161         (file.getCanonicalFile()));
162     addTextField(document, Indexer.fullpath, file.getAbsolutePath());
163
164     FileInputStream input = null;
165     try {
166         input = new FileInputStream(file);
167         // addContent(document, input, file.getPath());
168         addContent(document, input, "<inputstream>");
169     } finally {
170         if (input != null) {
171             input.close();
172         }
173     }
174     return document;
175
176 }
177
178 /**
179  * This will get a lucene document from a PDF file.
180  *
181  * @param is
182  *         The stream to read the PDF from.
183  * @return The lucene document.
184  * @throws IOException
185  *         If there is an error parsing or indexing the document.
186  */
187 public static Document getDocument(InputStream is) throws IOException {
188     LucenePDFDocument converter = new LucenePDFDocument();
189     return converter.convertDocument(is);
190 }
191
192 /**
193  * This will get a lucene document from a PDF file.
194  *
195  * @param file
196  *         The file to get the document for.

```

```

195  *
196  * @return The lucene document.
197  * @throws IOException
198  *         If there is an error parsing or indexing the document.
199  */
200 public static Document getDocument(File file) throws IOException {
201     LucenePDFDocument converter = new LucenePDFDocument();
202     return converter.convertDocument(file);
203 }
204
205 /**
206  * This will get a lucene document from a PDF file.
207  *
208  * @param url
209  *         The file to get the document for.
210  * @return The lucene document.
211  * @throws IOException
212  *         If there is an error parsing or indexing the document.
213  */
214 public static Document getDocument(URL url) throws IOException {
215     LucenePDFDocument converter = new LucenePDFDocument();
216     return converter.convertDocument(url);
217 }
218
219 /**
220  * This will add the contents to the lucene document.
221  *
222  * @param document
223  *         The document to add the contents to.
224  * @param is
225  *         The stream to get the contents from.
226  * @param documentLocation
227  *         The location of the document, used just for debug messages.
228  * @throws IOException
229  *         If there is an error parsing the document.
230  */
231 private void addContent(Document document, InputStream is, String
    documentLocation) throws IOException {
232     PDDocument pdfDocument = null;
233     PDFTextStripper stripper;
234     try {

```

```

235 pdfDocument = PDDocument.load(is);
236 if (pdfDocument.isEncrypted()) {
237     // Just try using the default password and move on
238     pdfDocument.decrypt("");
239 }
240
241 // create a writer where to append the text content.
242 StringWriter writer = new StringWriter();
243 stripper = new PDFTextStripper();
244 try {
245     stripper.writeText(pdfDocument, writer);
246
247 } catch (Exception e) {
248     System.out.println("Error in stripper.writeText()");
249 }
250 String contents = writer.getBuffer().toString();
251
252 StringReader reader = new StringReader(contents);
253 addTextField(document, Indexer.contents, reader);
254 PDDocumentInformation info = pdfDocument.getDocumentInformation();
255 if (info != null) {
256     addTextField(document, Indexer.Author, info.getAuthor());
257     try {
258         addTextField(document, Indexer.created, info.getCreationDate());
259     } catch (IOException io) {
260         // ignore, bad date but continue with indexing
261     }
262
263     addTextField(document, Indexer.keywords, info.getKeywords());
264     try {
265         addTextField(document, Indexer.modified, info.getModificationDate());
266     } catch (IOException io) {
267         // ignore, bad date but continue with indexing
268     }
269     addTextField(document, "Subject", info.getSubject());
270     addTextField(document, Indexer.Title, info.getTitle());
271 }
272 int summarySize = Math.min(contents.length(), 500);
273 String summary = contents.substring(0, summarySize);
274 // Add the summary as an UnIndexed field, so that it is stored and
275 // returned

```

```

276     // with hit documents for display.
277     addUnindexedField(document, Indexer.summary, summary);
278 } catch (CryptographyException e) {
279     throw new IOException("Error decrypting document(" + documentLocation + "):
        " + e);
280 } catch (InvalidPasswordException e) {
281     // they didn't supply a password and the default of "" was wrong.
282     throw new IOException("Error: The document(" + documentLocation + ") is
        encrypted and will not be indexed.");
283 } finally {
284     if (pdfDocument != null) {
285         pdfDocument.close();
286     }
287 }
288 }
289
290 /**
291  * This will test creating a document.
292  * usage: java pdfparser.searchengine.lucene.LucenePDFDocument
293  * <pdf-document>;
294  * @param args
295  *         command line arguments.
296  * @throws IOException
297  *         If there is an error.
298  */
299 public static void main(String[] args) throws IOException {
300     if (args.length != 1) {
301         String us = LucenePDFDocument.class.getName();
302         System.err.println("usage: java " + us + " <pdf-document>");
303         System.exit(1);
304     }
305     System.out.println("Document=" + getDocument(new File(args[0])));
306 }
307 }

```

Listing A.5: PDFIndexer.java

A.1.6. OfficeDocIndexer.java

```
1 package zhaw;
2
3 import java.io . File ;
4 import java.io . FileInputStream;
5 import java.io . FileNotFoundException;
6 import java.io . IOException;
7 import java.io . StringReader;
8 import java. util . Date;
9
10 import org.apache.lucene.document.Document;
11 import org.apache.lucene.document.Field;
12 import org.apache.poi.POIXMLDocument;
13 import org.apache.poi.hslf.extractor.PowerPointExtractor;
14 import org.apache.poi.hssf.extractor.ExcelExtractor;
15 import org.apache.poi.hwpf.extractor.WordExtractor;
16 import org.apache.poi.openxml4j.opc.OPCPackage;
17 import org.apache.poi.poifs.filesystem.POIFSFileSystem;
18 import org.apache.poi.xssf.extractor.XSSFExcelExtractor;
19 import org.apache.poi.xwpf.extractor.XWPFWordExtractor;
20
21 public class OfficeDocIndexer extends Indexer {
22
23     public OfficeDocIndexer() {
24     }
25
26     public Document getOfficeDocument(File file) throws IOException {
27         Document doc = null;
28         String fileExtension = "NULL";
29         Main.getMyFunctions();
30         fileExtension = myFunctions.getFileExtension(file);
31
32         if (isXMLExcel(fileExtension)) {
33             doc = indexXMLExcel(file);
34             return doc;
35         }
36
37         if (isXMLWord(fileExtension)) {
38             doc = indexXMLWord(file);
39             return doc;
```

```

40     }
41
42     if (isLegacyExcel(fileExtension)) {
43         doc = indexLegacyExcel(file);
44         return doc;
45     }
46
47     if (isLegacyWord(fileExtension)) {
48         doc = indexLegacyWord(file);
49         return doc;
50     }
51     if (isPowerPoint(fileExtension)) {
52         doc = indexPowerPoint(file);
53         return doc;
54     }
55
56     return doc;
57 }
58
59 /** indexing XML Word File */
60 public Document indexXMLWord(File file) {
61     Document doc = new Document();
62     XWPFWordExtractor wordxmlextractor = null;
63     try {
64         OPCPackage pkgDoc = POIXMLDocument.openPackage(file.toString());
65         wordxmlextractor = new XWPFWordExtractor(pkgDoc);
66
67     } catch (Exception e) {
68         System.out.println("Failed to set Word XML Parser");
69         e.printStackTrace();
70     }
71
72     String content = wordxmlextractor.getText();
73     doc.add(new Field(Indexer.contents, new StringReader(content)));
74
75     doc.add(new Field(Indexer.filename, file.getName(), Field.Store.YES, Field.Index.
        NOT_ANALYZED));
76
77     try {
78         doc.add(new Field(Indexer.fullpath, file.getCanonicalPath(), Field.Store.YES,
            Field.Index.NOT_ANALYZED));

```



```

79     } catch (Exception e) {
80         System.out.println("Failed to set fullpath of XML Excel File");
81         e.printStackTrace();
82     }
83
84     String author = wordxmlextractor.getCoreProperties().getCreator();
85     addTextField(doc, Indexer.Author, author);
86
87     String description = wordxmlextractor.getCoreProperties().getDescription();
88     addTextField(doc, Indexer.description, description);
89
90     String title = wordxmlextractor.getCoreProperties().getTitle();
91     addTextField(doc, Indexer.Title, title);
92
93     Date modified = wordxmlextractor.getCoreProperties().getModified();
94     addTextField(doc, Indexer.modified, modified);
95
96     Date created = wordxmlextractor.getCoreProperties().getModified();
97     addTextField(doc, Indexer.created, created);
98
99     String extension = myFunctions.getFileExtension(file);
100    addTextField(doc, Indexer.extension, extension);
101
102    return doc;
103
104 }
105
106 /** indexing XML Excel File */
107 public Document indexXMLExcel(File file) {
108     Document doc = new Document();
109     XSSFExcelExtractor excelXMLExtractor = null;
110     try {
111         String strfile = file.toString();
112         excelXMLExtractor = new XSSFExcelExtractor(strfile);
113
114     } catch (Exception e) {
115         System.out.println("Failed to set Excel XML Parser");
116         e.printStackTrace();
117     }
118
119     String content = excelXMLExtractor.getText();

```

```

120     doc.add(new Field(Indexer.contents, new StringReader(content)));
121
122     doc.add(new Field(Indexer.filename, file .getName(), Field.Store.YES, Field.Index.
        NOT_ANALYZED));
123
124     try {
125         doc.add(new Field(Indexer.fullpath, file .getCanonicalPath(), Field.Store.YES,
            Field.Index.NOT_ANALYZED));
126     } catch (Exception e) {
127         System.out.println("Failed to set fullpath of XML Excel File");
128         e.printStackTrace();
129     }
130
131     String author = excelXMLExtractor.getCoreProperties().getCreator();
132     addTextField(doc, Indexer.Author, author);
133
134     String description = excelXMLExtractor.getCoreProperties().getDescription();
135     addTextField(doc, Indexer.description, description );
136
137     String title = excelXMLExtractor.getCoreProperties().getTitle();
138     addTextField(doc, Indexer.Title, title );
139
140     Date modified = excelXMLExtractor.getCoreProperties().getModified();
141     addTextField(doc, Indexer.modified, modified);
142
143     Date created = excelXMLExtractor.getCoreProperties().getModified();
144     addTextField(doc, Indexer.created, created);
145
146     String extension = myFunctions.getFileExtension(file);
147     addTextField(doc, Indexer.extension, extension);
148
149     return doc;
150
151 }
152
153 /** indexing legacy Excel File */
154 public Document indexLegacyExcel(File file) {
155     Document doc = new Document();
156     POIFSFileSystem fs;
157     ExcelExtractor extractor = null;
158     try {

```

```

159     fs = new POIFSFileSystem(new FileInputStream(file));
160     extractor = new ExcelExtractor(fs);
161
162     String content = extractor.getText();
163     doc.add(new Field(Indexer.contents, new StringReader(content)));
164
165     doc.add(new Field(Indexer.fullpath, file.getCanonicalPath(), Field.Store.YES,
166         Field.Index.NOT_ANALYZED));
167
168     } catch (FileNotFoundException e) {
169         System.out.println("File " + file + " not found");
170         e.printStackTrace();
171     } catch (IOException e) {
172         System.out.println("Exception in parsing legacy excel document");
173     }
174
175     doc.add(new Field(Indexer.filename, file.getName(), Field.Store.YES, Field.Index.
176         NOT_ANALYZED));
177
178     String appName = extractor.getSummaryInformation().getApplicationName();
179     addTextField(doc, Indexer.appname, appName);
180
181     String author = extractor.getSummaryInformation().getAuthor();
182     addTextField(doc, Indexer.Author, author);
183
184     String extension = myFunctions.getFileExtension(file);
185     addTextField(doc, Indexer.extension, extension);
186
187     return doc;
188 }
189
190 /** indexing legacy Word File */
191 public Document indexLegacyWord(File file) {
192     Document doc = new Document();
193     POIFSFileSystem fs;
194     WordExtractor extractor = null;
195     try {
196         fs = new POIFSFileSystem(new FileInputStream(file));
197         extractor = new WordExtractor(fs);
198         String content = extractor.getText();
199         doc.add(new Field(Indexer.contents, new StringReader(content)));

```

```

198
199     doc.add(new Field(Indexer.fullpath, file .getCanonicalPath(), Field.Store.YES,
200         Field.Index.NOT_ANALYZED));
201
202     } catch (FileNotFoundException e) {
203         System.out.println("File " + file + " not found");
204         e.printStackTrace();
205     } catch (IOException e) {
206         System.out.println("Exception in parsing legacy word document");
207     }
208
209     doc.add(new Field(Indexer.filename, file .getName(), Field.Store.YES, Field.Index.
210         NOT_ANALYZED));
211
212     String appName = extractor.getSummaryInformation().getApplicationName();
213     addTextField(doc, Indexer.appname, appName);
214     String author = extractor.getSummaryInformation().getAuthor();
215     addTextField(doc, Indexer.Author, author);
216
217     String extension = myFunctions.getFileExtension(file);
218     addTextField(doc, Indexer.extension, extension);
219
220     return doc;
221 }
222
223 /** indexing Powerpoint File */
224 public Document indexPowerPoint(File file) {
225     Document doc = new Document();
226     POIFSFileSystem fs;
227     String content = "";
228     PowerPointExtractor extractor = null;
229     try {
230         fs = new POIFSFileSystem(new FileInputStream(file));
231         extractor = new PowerPointExtractor(fs);
232         content = extractor.getText();
233         doc.add(new Field(Indexer.contents, new StringReader(content)));
234
235         doc.add(new Field(Indexer.fullpath, file .getCanonicalPath(), Field.Store.YES,
236             Field.Index.NOT_ANALYZED));
237
238     } catch (FileNotFoundException e) {

```

```

236     System.out.println("File " + file + " not found");
237     e.printStackTrace();
238 } catch (IOException e) {
239     System.out.println("Exception in parsing legacy word document");
240 }
241
242 doc.add(new Field(Indexer.filename, file .getName(), Field.Store.YES, Field.Index.
    NOT_ANALYZED));
243
244 String appName = extractor.getSummaryInformation().getApplicationName();
245 addTextField(doc, Indexer.appname, appName);
246 String author = extractor.getSummaryInformation().getAuthor();
247 addTextField(doc, Indexer.Author, author);
248
249 String extension = myFunctions.getFileExtension(file);
250 addTextField(doc, Indexer.extension, extension);
251
252 return doc;
253 }
254
255 /**
256  * return all valid file extensions for this Indexer see
257  * http://en.wikipedia.org/wiki/List\_of\_Microsoft\_Office\_filename\_extensions
258  */
259 public String[] getvalidFileextensions () {
260     String[] retString = { "xls", "xlt", "xlm", "xlsx", "xlsm", "xltx", "xltm", "doc",
        "dot", "docx", "docm", "dotx", "dotm" };
261
262     return retString;
263 }
264
265 /**
266  * returns true if ext is a newer XML Excel Office Document, otherwise
267  * return false
268  */
269 public boolean isXMLExcel(String ext) {
270     if (ext.equals("xlsx"))
271         return true;
272     if (ext.equals("xlsm"))
273         return true;
274     if (ext.equals("xltx"))

```

```
275     return true;
276     if (ext.equals("xltm"))
277         return true;
278
279     return false ;
280 }
281
282 /**
283  * returns true if ext is a newer XML Word Office Document, otherwise
284  * return false
285  */
286 public boolean isXMLWord(String ext) {
287     if (ext.equals("docx"))
288         return true;
289     if (ext.equals("dotx"))
290         return true;
291     if (ext.equals("dotm"))
292         return true;
293
294     return false ;
295 }
296
297 /** returns true if ext is an excel extension, otherwise return false */
298 public boolean isLegacyExcel(String ext) {
299     if (ext.equals("xls"))
300         return true;
301     if (ext.equals("xlt"))
302         return true;
303     if (ext.equals("xlm"))
304         return true;
305     return false ;
306 }
307
308 /** returns true if ext is an word extension, otherwise return false */
309 public boolean isLegacyWord(String ext) {
310     if (ext.equals("doc"))
311         return true;
312     if (ext.equals("dot"))
313         return true;
314
315     return false ;
```

```
316 }
317
318 /** returns true if ext is an word extension, otherwise return false */
319 public boolean isPowerPoint(String ext) {
320     if (ext.equals("ppt"))
321         return true;
322     if (ext.equals("pot"))
323         return true;
324     if (ext.equals("pps"))
325         return true;
326     if (ext.equals("pptx"))
327         return true;
328     if (ext.equals("pptm"))
329         return true;
330     if (ext.equals("potx"))
331         return true;
332     if (ext.equals("potm"))
333         return true;
334     if (ext.equals("ppam"))
335         return true;
336     if (ext.equals("ppsx"))
337         return true;
338     if (ext.equals("ppsm"))
339         return true;
340     if (ext.equals("sldx"))
341         return true;
342     if (ext.equals("sldm"))
343         return true;
344     return false ;
345 }
346
347 }
```

Listing A.6: OfficeDocIndexer.java

A.1.7. myFunctions.java

```
1 package zhaw;
2
3 import java.io.File;
4 import java.io.FilenameFilter;
5
6 import org.apache.lucene.document.Document;
7
8 public class myFunctions {
9
10     public myFunctions() {
11     }
12
13     /* will return a String Array of all Subdirectories */
14     public String[] getSubDirectories(String dataDir) {
15         File file = new File(dataDir);
16         String[] directories = file.list(new FilenameFilter() {
17             @Override
18             public boolean accept(File current, String name) {
19                 return new File(current, name).isDirectory();
20             }
21         });
22         return directories;
23     }
24
25     /*
26      * @return: returns the Extension of the file return "NULL" if the file has
27      * no extension
28      */
29     public static String getFileExtension(File path) {
30
31         String pathstring = path.getName().toLowerCase();
32         String fileExtension;
33         try {
34             fileExtension = "NULL";
35             fileExtension = pathstring.substring(pathstring.lastIndexOf('.'), pathstring.
36                 length()).substring(1);
37         } catch (StringIndexOutOfBoundsException e) {
38             return "NULL";
39         }
40     }
41 }
```



```

39     return fileExtension ;
40
41 }
42
43 /*
44  * this function will prepare the indexer and choose, which indexer should
45  * be chosen. e.g. .txt or .c file will use textfile indexer
46  */
47 public static void prepareindexFile(File f) {
48     String fileExtension = "NULL";
49     fileExtension = getFileExtension(f);
50
51     String IndexType = Main.getIndexer().getFileExtensionFunction(fileExtension);
52     // System.out.println("Index Type = " + IndexType + "\tFilename = " +
53     // f.getName());
54     /* if File Extension is "TEXT", index it as a text file */
55     if (IndexType.equals("TEXT")) {
56         try {
57             indexTextFile(f);
58         } catch (Exception e) {
59             System.out.println("Could not index text file " + f.getName());
60             Logger.writeToLog("Could not index text file " + f.getName());
61         }
62     }
63
64     /* if File Extension is "PDF", index it as a PDF */
65     else if (IndexType.equals("PDF")) {
66         try {
67             indexPDFFile(f);
68         } catch (Exception e) {
69             System.out.println("Could not index pdf file " + f.getName());
70             Logger.writeToLog("Could not index pdf file " + f.getName());
71         }
72     }
73
74     /* if File Extension is "Office", index it as a Office Doc */
75     else if (IndexType.equals("OFFICE")) {
76         try {
77             indexOfficeFile(f);
78         } catch (Exception e) {
79             System.out.println("Could not index office document " + f.getName());

```

```
80     Logger.writeToLog("Could not index office document " + f.getName());
81     }
82 }
83
84 }
85
86 /* will index all text files extensions */
87 public static void indexTextFile(File f) throws Exception {
88     Indexer.getTextFileIndexer();
89     Document doc = TextFileIndexer.getDocument(f);
90     Main.getwriter().addDocument(doc);
91 }
92
93 /* will index all pdf files extensions */
94 private static void indexPDFFile(File f) throws Exception {
95     Document doc = Indexer.getPDFIndexer().convertDocument(f);
96     Main.getwriter().addDocument(doc);
97 }
98
99 private static void indexOfficeFile(File f) throws Exception {
100     Document doc = Indexer.getOfficeIndexer().getOfficeDocument(f);
101     Main.getwriter().addDocument(doc);
102
103 }
104
105 }
```

Listing A.7: myFunctions.java

A.1.8. TextFilesFilter

```
1 package zhaw;
2
3 import java.io . File ;
4 import java.io . FileFilter ;
5
6 /* Filter for File extensions, which should be indexed */
7 class TextFilesFilter implements FileFilter {
8
9     public boolean accept(File path) {
10         Main.getMyFunctions();
11         String fileExtension = myFunctions.getFileExtension(path);
12         boolean returnbool = Main.getIndexer().isValidFileExtension(fileExtension);
13         return returnbool;
14     }
15 }
```

Listing A.8: TextFilesFilter

Index

A

Analyzer	26
Aufgabenstellung	4
Ausgangslage	3

E

erwartetes Resultat	5
---------------------------	---

F

Fazit der Arbeit	30
Fazit Suchergebnis	25

I

Indexer	13–15
OfficeDoc Indexer ... <i>siehe</i> OfficeDoc Indexer	
PDF Indexer <i>siehe</i> PDF Indexer	
TextFile Indexer	<i>siehe</i> TextFile Indexer
indizierte Felder	9–10

J

Java	11–12
Klassendiagramm	11–12
Souce-Code	i–xxx

L

Lucene Syntax	<i>siehe</i> Syntax Lucene
---------------------	----------------------------

O

OfficeDoc Indexer	14
-------------------------	----

P

PDF Indexer	14
-------------------	----

S

Syntax Lucene	18–20
---------------------	-------

T

TextFile Indexer	13
------------------------	----

V

Vergleich mit Windows	22–29
Vorbereitung für Suche	16–20

W

Windows 8.1	<i>siehe</i> Vergleich mit Windows
-------------------	------------------------------------

Z

Ziel der Arbeit	3
-----------------------	---

Literaturverzeichnis

- [1] *Apache Lucene - Index File Formats*. http://lucene.apache.org/core/3_5_0/fileformats.html, may 2014. 4.3
- [2] *Lucene Query Parser Syntax*. http://lucene.apache.org/core/2_9_4/queryparsersyntax.html, may 2014. 5.2.7
- [3] *Who is using Lucene/Solr*. <http://searchhub.org/2012/01/21/who-uses-lucenesolr/>, may 2014. 4.3
- [4] MCCANDLESS, MICHAEL; HATCHER, ERIK; GOSPONDENETIC OTIS: *Lucene in Action - Second Edition*. Manning Publications Co, 2010. 6.1.4