MiniSearch

Aufgabenstellung zur Hausarbeit
Softwaretechnologie: Java
im Fach Informationsverarbeitung
Universität zu Köln
Sommersemester 2016

Mihail Atanassov

Inhaltsverzeichnis

1 Allgemeine Informationen

Die Hausarbeit besteht aus 4 Anforderungen:

1) Implementation des Programms gemäß den Anforderungen dieser Beschreibung (Kapitel 4.1), Abgabe auf CD oder per Email.
2) Interne Dokumentation aller Klassen, Attribute und Methoden mit JavaDoc - Evtl. zusätzlich notwendige Dokumentation innerhalb von Methodenkörpern (bspw. komplexere Algorithmen) ist sinnvoll (siehe auch Kapitel 4.2)
3) Externe Dokumentation – ausgedruckt, der Form einer wissenschaftlichen Arbeit entsprechend (siehe www.spinfo.phil-fak.uni-koeln.de/spinfo-wissarbeiten). Inhalte der externen Dokumentation werden in Kapitel 4.3 beschrieben.
4) Bei Aushändigung der Arbeit müssen Sie in der Lage sein, den Programmablauf Ihrer Implementation nachzuvollziehen und uns darzulegen, damit sichergestellt werden kann, dass Sie selbst programmiert haben.

Kennzeichnen Sie CD **und** Dokumentation mit Ihrem **Namen**, Ihrer **Matrikelnummer** sowie Ihrer **email-Adresse** (wichtig! siehe Kapitel).

Das Programm kann sich an den Vorschlägen in diesem Dokument orientieren (siehe Kapitel 5), Sie können jedoch auch eine völlig unabhängige, eigene Lösung programmieren. In diesem Fall sollten sie jedoch bei der Dokumentation besonders sorgfältig sein und Abweichungen zu den Vorschlägen begründen, zudem müssen die Anforderungen, die an das Programm gestellt werden, erfüllt werden.

Gruppenarbeit ist möglich, allerdings mit maximal 2 Teilnehmern und lediglich bei der Programmierung, nicht bei der internen und externen Dokumentation. Sie sollten mit dieser Arbeit beweisen, dass Sie der Sprache Java mächtig sind und in der Lage sind, Anforderungen wie die hier gestellten zu erfüllen. Ähneln sich Implementierungen zu sehr, muss die mündliche Darlegung (s.o. Anforderung 4) umfangreicher ausfallen.

Abgabe: Die Arbeit muss bis spätestens **Donnerstag, 13.09.2016, 12 Uhr** in der Sprachlichen Informationsverarbeitung abgegeben werden. **Achtung!!!** Um sichergehen zu können, dass Sie nicht vor verschlossenen Türen stehen, sollten Sie zur Abgabe nur zu den Geschäftszeiten kommen! Daher auch die Zeitangabe beim spätesten Abgabetermin. Eine Abgabe per Email ist auch möglich – bitte schicken Sie in diesem Fall das aus Eclipse exportierte Projekt als ZIP-Datei, sowie die Dokumentation als PDF-Datei per Email an matanass@uni-koeln.de. Um sicherzugehen, dass die Arbeit angekomen ist, lassen Sie sich den Eingang bitte bestätigen.

Das Programm muss in einem lauffähigen Zustand vorliegen, d.h. es muss **a)** ausführbar sein und **b)** die mitgelieferten JUnit-Tests bestehen, andernfalls gilt die Arbeit als nicht bestanden. Sollten einige oder weniger relevante Anforderungen nicht erfüllt sein, so werden Sie (per Email! Bitte auf Deckblatt der Dokumentation angeben) aufgefordert, die entsprechenden Punkte nachzubessern.

2 Programmbeschreibung

2.1 Allgemeines zu Suchmaschinen

Ihre Aufgabe ist es, eine Suchmaschine für ein lokales Dateisystem zu entwickeln und zu implementieren. Vergleichen Sie dazu den Anfang des Eintrags in der Wikipedia zu

Suchmaschinen[1]:

> Eine **Suchmaschine** ist ein Programm zur Recherche von Dokumenten, die in einem Computer oder einem Computernetzwerk wie z. B. dem World Wide Web gespeichert sind. Internet-Suchmaschinen haben ihren Ursprung in Information-Retrieval-Systemen. Sie erstellen einen Schlüsselwort-Index für die Dokumentbasis, um Suchanfragen über Schlüsselwörter mit einer nach Relevanz geordneten Trefferliste zu beantworten. Nach Eingabe eines Suchbegriffs liefert eine Suchmaschine eine Liste von Verweisen auf möglicherweise relevante Dokumente, meistens dargestellt mit Titel und einem kurzen Auszug des jeweiligen Dokuments. Dabei können verschiedene Suchverfahren Anwendung finden.
>
> Die wesentlichen Bestandteile bzw. Aufgabenbereiche einer Suchmaschine sind:
>
> - Erstellung und Pflege eines Indexes (Datenstruktur mit Informationen über Dokumente),
> - Verarbeiten von Suchanfragen (Finden und Ordnen von Ergebnissen) sowie
> - Aufbereitung der Ergebnisse in einer möglichst sinnvollen Form.
>
> In der Regel erfolgt die *Datenbeschaffung* automatisch, im WWW durch Webcrawler, auf einem einzelnen Computer durch regelmäßiges Einlesen aller Dateien in vom Benutzer spezifizierten Verzeichnissen im lokalen Dateisystem.

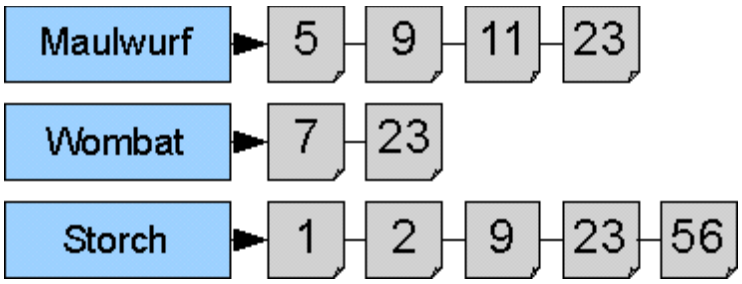
Mit der Aufgabenstellung bekommen Sie eine graphische Benutzeroberfläche (Graphical User Interface - GUI) ausgeliefert, welche die Kommunikation mit dem Suchmaschinenbenutzer steuert und die die Suchergebnisaufbereitung übernimmt (vgl. Kapitel 3). Ihr Programm muss mit dieser mitgelieferten GUI kommunizieren, selbstverständlich können Sie diese auch erweitern, wenn Sie noch weitere Funktionalitäten realisieren wollen.

2.2 Invertierter Index

Eine Suchmaschine lässt sich auf viele verschiedene Arten realisieren, z.B. könnte man die zu durchsuchenden Dokumente bei jeder Suche auf einen String einlesen und die gesuchten Terme per `indexOf(Term)` suchen lassen, was eine sehr ineffiziente Variante wäre. Moderne Suchmaschinen sind in der Mehrzahl indexbasiert, was bedeutet, dass sie zunächst eine Datenstruktur erzeugen, die für Suchanfragen konsultiert wird. Der momentane Industriestandard für eine solche Datenstruktur ist der **invertierte Index**.

Ein invertierter Index enthält das gesamte Vokabular der indexierten Dokumente (also alle in ihnen vorkommenden verschiedenen Wörter, auch *Dictionary* genannt). Jedes Wort ist dabei an eine Liste (Tipp: Hier eignet sich ein Set!) geknüpft, in der alle Dokumente aufgelistet sind, in denen das Wort vorkommt. *Invertiert* wird der Index deshalb genannt, da nicht die Dokumente die in ihnen enthaltenen Wörter listen, sondern die Wörter die Dokumente, in denen sie vorkommen. Eine solche Dokumentenliste bezeichnet man auch als *Postings List*. Folgende Abbildung zeigt einen invertierten Index: Links finden sich die Wörter, rechts in einer Liste die Dokumente, in denen die entsprechenden Wörter auftreten:

1 http://de.wikipedia.org/wiki/Suchmaschine



Um den Index nicht größer als notwendig werden zu lassen, wird gemeinhin darauf verzichtet, hochfrequente, aber semantisch arme Wörter wie Artikel und Präpositionen in den Index aufzunehmen. Sie sind nicht dazu gezwungen, in Ihrer Suchmaschine genauso zu verfahren, weil v.a. die Phrasensuche etwas komplizierter wird. Sie können es aber gerne so realisieren, quasi als fakultative Zusatzaufgabe.

3 Informationen zur Vorlage

3.1 Projektstruktur

Die Vorlage enthält folgende Verzeichnisse:

- **src** enthält den Quellcode der Vorlage. Fügen Sie hier eigene Klassen in geeigneten Unterpackages hinzu.
- **doc** Javadoc zur Vorlage. Das beiliegende Ant-Skript javadoc.xml speichert javadoc immer in diesem Verzeichnis ab, auch die Kommentare zu Ihren Klassen.
- **lib** Die Bibliotheken, die von der Vorlage verwendet werden.
- **texts** Die Texte, die von Ihrem Programm verarbeitet werden sollen.
- **.settings** Verschiedene Einstellungen von Eclipse.

sowie die Dateien:

- **javadoc.xml** Ant-Skript, dass JavaDoc zu den Klassen im src-Verzeichnis generiert.
- **.classpath** Vorgefertigte Abhängigkeiten zu den Bibliotheken in lib/.
- **Aufgabenstellung.pdf** Diese Datei.

3.2 Schnittstelle zwischen Vorlage und Ihrem Programm

Die Schnittstelle, über die Ihr Programm und die graphische Benutzeroberfläche kommunizieren, besteht aus den zwei Java-Interfaces `IResult` und `ISearchEngine`. Die genauen Anforderungen der Interfaces sind durch die deklarierten Methoden und javadoc-Kommentare spezifiziert.

Beim Start des Programms wird die GUI mit dem ISearchEngine-Interface verknüpft, anschließend wird die start-Methode von `ISearchEngine` aufgerufen. Alle weiteren Aufrufe von `ISearchEngine`-Methoden sind abhängig von den Benutzeraktionen (oder den jeweiligen JUnit-Tests), bis das Programm schließlich beendet wird und die stop-Methode von `ISearchEngine` aufgerufen wird. Damit gefundene Dokumente sowie deren Inhalte in der GUI dargestellt werden können, muss die search-Methode von

`ISearchEngine` eine Liste von `IResult`-Implementationen zurückgeben.

Die meisten Methoden von `ISearchEngine` und `IResult` sind derart deklariert, dass eine Exception vom Typ `SearchEngineException` geworfen werden kann. So können Meldungen über Fehler an die GUI zurückgegeben werden. Tritt beispielsweise eine FileNotFound-Exception in der Methode `IResult.getContent()` auf, können Sie eine `SearchEngineException` werfen und den Grund für diese Exception (z.B. „Der indizierte Text wurde nicht mehr gefunden.“) als Konstruktorparameter angeben.

Damit Ihre Implementation von ISearchEngine im Hauptprogramm und in den JUnit-Tests genutzt wird, müssen Sie in der Klasse Main die Methode getSearchEngine() so anpassen, dass eine Instanz Ihrer Klasse zurückgegeben wird. Da diese Methode auch vom JUnit-Test aufgerufen wird, müssen Sie für die Tests keine weitere Änderung durchführen.

3.3 Das Singleton-Design-Pattern, SearchEngineFrame, Preferences

Damit in Ihrem Code an jeder beliebigen Stelle auf die Methoden, die die Vorlage zur Verfügung stellt, zugegriffen werden kann, sind die Klassen `SearchEngineFrame` und `Preferences`, von denen jeweils nur ein Objekt pro virtueller Maschine existieren darf, nach dem Singleton-Entwurfsmuster programmiert. Benötigen Sie bspw. Zugriff auf die Klasse SearchEngineFrame, so ist dies durch die Zeile

```
SearchEngineFrame frame = SearchEngineFrame.getInstance();
```

realisierbar.

Die Klasse SearchEngineFrame bietet folgende Methoden, die Sie in Ihrem Code benötigen, wenn Sie den Fortschrittsbalken verwenden wollen (was für lang laufende Methoden, wie die Indexierung oder die Serialisierung/Deserialisierung des Indexes sehr empfohlen ist):

- `setMaximumWork()`
- `setWorkDone()`
- `stopProgressBar()`
- `setInfiniteProgress()`

Die Klasse Preferences bietet die Möglichkeit, mit Hilfe von get-Methoden auf Einstellungen wie die Liste der zu indexierenden Verzeichnisse zuzugreifen. Beachten Sie die javadoc-Kommentare zu diesen Methoden.

3.4 Korpora

Mit der Vorlage erhalten Sie zwei Ordner mit Texten, mit denen Sie die Vorlage und auch Ihr Programm testen können. Der Ordner "texts" enthälten ein relativ großes Korpus verschiedener längerer Texte, um die Suchmaschine einem (zugegebenermaßen immer noch relativ beschränktem) Real-World-Szenario auszusetzen. Der Ordner "test_texts" enthält eine Sammlung von sehr kurzen Texten, die aber bestimmte Fälle von potenziellen Problemen testen können. Diese Texte werden auch für die JUnit-Tests genutzt.

4 Spezifikation der Anforderungen

In diesem Kapitel finden Sie nähere Informationen zu den Anforderungen an Ihre Hausarbeit.

4.1 Implementierung

Ihr Programm muss alle mitgelieferten JUnit-Tests (s.u.) aus der Klasse MiniSearchTest bestehen, d.h. die Ausführung muss in einem grünen Balken resultieren. Mit einem grünen Balken sind Sie schon auf der relativ sicheren Seite. Es gelten aber noch folgende Einschränkungen:

- Ihr Programm muss über die definierten Schnittstellen mit der mitgelieferten GUI interagieren. Ihnen steht dabei frei, die GUI weiter auszubauen. Damit soll nur verhindert werden, dass Sie irgendwelche fertige Fremdsoftware abgeben.
- Ihr Programm darf bis auf die mitgelieferten externen Bibliotheken keine weiteren nutzen. Wenn Sie der Meinung sind, dass Ihr Programm durch die Nutzung bestimmter Bibliotheken gewinnen könnte (und Sie sich nicht nur die Arbeit erleichtern wollen), könnte nach Rücksprache eine Ausnahmegenehmigung erteilt werden. Wenden Sie sich dann bitte frühzeitig an uns.
- Sie müssen einen invertierten Index aufbauen und für ihre Suchanfragen nutzen. Suchmaschinen, die lediglich Operationen über Strings (wie indexOf(Suchterm)) vornehmen, werden nicht akzeptiert.

Die JUnit-Tests, die Ihr Programm bestehen muss, liegen im Package `spinfo.minisearch.tests`. Die Klasse MiniSearchTests lässt sich per *run as --> JUnitTest* ausführen. Damit die Tests bestanden werden, muss ihre Suchmaschine mindestens folgende Anforderungen erfüllen:

4.1.1 Verwaltung des Index

- **Korrekte Indexierung**: Bei der Indexierung von Textdateien muss auf einige Details geachtet werden:
  - *Bereitstellen der Dateien*: Wenn ein Ordner indexiert werden soll, so müssen alle Textdateien des Ordners selbst und aller seiner Unterordner gefunden werden.
  - *Lesen der Daten*: Umlaute müssen korrekt dargestellt werden (Einlesen der mitgelieferten Textdateien als UTF-8)
  - *Tokenisierung*: Die Vorlage nutzt ein einfaches Wortmodell: Wörter sind Abfolgen von Buchstaben (Überprüfbar mit der Methode `Character.isLetter(char)`). Es steht Ihnen natürlich frei, auch solche Dinge wie Abkürzungen, Mailadressen o.ä. als Wörter zu behandeln.
  - *Indexierung*: Im Index finden sich standardmäßig alle Wörter in Kleinschreibung. Dies sollte in Ihrem Index auch der Fall sein (empfehlenswert hier: Methode `toLowerCase()` der Klasse `String`).
- **Index erweitern**: Wenn man der Suchmaschine einen Ordner zum Einfügen übergibt, so werden alle enthaltenen Textdateien indexiert. Bereits vorhandene Textdateien sollen nicht mehrfach indexiert werden.
- **Index verkleinern**: Übergibt man einen Ordner, der gelöscht werden soll, so sollen

alle enthaltenen Textdateien aus dem Index gelöscht werden.

- **Index aktualisieren**: Die indexierten Dateien können auch von außerhalb der Suchmaschine manipuliert werden. Folgende Fälle müssen bei der Aktualisierung nachgeprüft werden:
  - Sind alle indexierten Dateien noch vorhanden? Wenn nicht, müssen die betreffenden Dokumente aus dem Index gelöscht werden.
  - Sind neue Textdateien in den indexierten Ordnern hinzugekommen? Ist das der Fall, so müssen diese in den Index aufgenommen werden.
  - Haben sich Textdateien verändert, d.h. entspricht das Datum der letzten Änderung (zugreifbar über die Methode `lastModified()` der Klasse `File`) dem der indizierten Datei? Wenn nicht, muss die alte Datei aus dem Index entfernt und die neue aufgenommen werden.
- **Index serialisieren/deserialisieren**: Um zu vermeiden, dass der Index bei jedem Programmstart neu erstellt werden muss, soll er bei Beendung des Programms in einer Datei serialisiert werden. Entsprechend muss bei Start des Programms überprüft werden, ob evtl. ein serialisierter Index besteht, der eingelesen werden muss.d

4.1.2 Suchfunktionen

- **Und-Verknüpfung**: Werden mehrere Wörter gesucht, sollen nur die Dokumente geliefert werden, die alle Wörter enthalten.
- **Phrasensuche**: Werden mehrere Wörter mit Anführungszeichen gruppiert, so sollen nur die Dokumente geliefert werden, die diese Wörter in genau dieser Abfolge enthalten.
- **Fundstellenangabe**: Damit die gesuchten Terme in der GUI farbig hinterlegt werden können, müssen die Fundstellen innerhalb der Dokumente angegeben werden.

Es empfiehlt sich auch, eigene Tests zu generieren, um die Funktionalität Ihres Programms besser überprüfen zu können und Fehlfunktionen, die sich evtl. eingeschlichen haben, schnell zu entdecken.

4.2 JavaDoc: Vorlage und Anforderung

Die Klassen der Vorlage sind vollständig mit JavaDoc kommentiert. Öffnen Sie die Datei „index.html“ im Verzeichnis „doc“ (ziehen Sie sie in einen Browser oder wählen Sie den Punkt „Open With->System Editor“ im Kontextmenü), um zur Übersicht über die Klassen im Projekt zu gelangen.

Sie müssen die von Ihnen angelegten Klassen ebenfalls dokumentieren. Evtl. ist es jedoch nicht notwendig, wirklich jede einzelne Methode zu kommentieren – beachten Sie jedoch folgende Anforderungen:

- jede Methode und jede Variable, die nicht als „private“ markiert ist, **muss** dokumentiert werden, inkl. der Parameter (`@param`), Rückgabewerte (`@returns`) oder Exceptions (`@throws`).
- Methoden mit geringerer Sichtbarkeit **müssen** dokumentiert werden, wenn sich die Funktionalität nicht ohne weiteres erschließt, oder wenn bestimmte Anforderungen an die Parameter gestellt werden (X darf nicht null sein, String darf keine Leerzeichen

enthalten o.ä.). Dies bedeutet umgekehrt, dass Sie umso weniger kommentieren müssen, je mehr Methoden und Variablen Sie „private“ deklarieren, und je besser Sie die Namen der Methoden/Variablen wählen. „String s“ muss in jedem Fall dokumentiert werden, String „currentWord“ nicht unbedingt.

Im Projektverzeichnis befindet sich eine Datei namens „javadoc.xml“. Dies ist eine „Bauanleitung“ für das Java-Werkzeug „ant“, das die JavaDocs des Projekts automatisch neu erzeugen kann. Wählen Sie die Datei im Package Explorer aus, und rufen Sie im Kontextmenu den Punkt „Run As->Ant Build“ aus. In der Console von Eclipse sollte eine Ausgabe wie diese erscheinen:

```
Buildfile: D:\workspace_pp\MiniSearch2\javadoc.xml
javadoc:
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source files for package spinfo.minisearch.gui...
  [javadoc] Loading source files for package spinfo.minisearch.interfaces...
  [javadoc] Constructing Javadoc information...
  [javadoc] Standard Doclet version 1.5.0_11
  [javadoc] Building tree for all the packages and classes...
  [javadoc] D:\workspace_pp\MiniSearch2\src\spinfo\minisearch\interfaces\
            ISearchEngine.java:29: warning - @return tag has no arguments.
  [javadoc] Building index for all the packages and classes...
  [javadoc] Building index for all classes...
  [javadoc] Generating D:\workspace_pp\MiniSearch2\doc\stylesheet.css...
  [javadoc] 1 warning
BUILD SUCCESSFUL
Total time: 10 seconds
```

Warnungen bzgl. JUnit können Sie ignorieren. Warnungen, welche die von Ihnen ausprogrammierten Klassen betreffen, weisen Sie darauf hin, dass Sie im javadoc-Kommentar obligatorische Elemente vergessen oder leer gelassen haben. Diese Warnungen sollten Sie beherzigen und die Gründe dafür beseitigen.

Nach Ausführung des Build-Files sollte die Dokumentation im Verzeichnis „doc“ aktualisiert, neue JavaDoc-Kommentare sollten nun ebenso wie neue, von Ihnen erzeugte und kommentierte Klassen, auch in den html-Seiten erscheinen. Ist dies nicht der Fall, könnte das an folgenden Ursachen liegen:

- Die von Ihnen programmierten Klassen liegen nicht im src-Ordner des MiniSearch-Projektes. Fügen Sie sie entweder dort ein oder verändern sie die Pfadangabe in der javadoc.xml.
- Sie haben noch nie javadoc benutzt, dann kennt ihr eclipse das javadoc-Kommando nicht. Rechtsklicken Sie die javadoc.xml-Datei, wählen Sie den javadoc-Wizzard und geben Sie im obersten Textfenster den Pfad zum javadoc-Kommando (im JDK-bin-Ordner) an (wobei es eigentlich zu spät ist, erst jetzt zum erstenmal mit javadoc zu arbeiten).

4.3 Externe Dokumentation

Die externe Dokumentation soll dazu dienen, kurz die Funktionsweise Ihres Programms und Ihre Lösungsstrategie zu erläutern und das Zusammenspiel der von Ihnen

implementierten Klassen zu erklären, sowie Probleme, Alternativen und Verbesserungsmöglichkeiten zu diskutieren. UML-Klassen- oder Sequenzdiagramme sind dabei hilfreich, jedoch nicht erforderlich. Der Umfang der Dokumentation soll in mindestens 8 Seiten betragen.

Für die Formatierung der Dokumentation gilt die Anleitung unter http://www.spinfo.uni-koeln.de/studieninfos/spinfo_stilneu.pdf verbindlich. Dokumentationen, die sich nicht an diese Vorgabe halten, werden **nicht akzeptiert**!

5 Tipps zur Vorgehensweise

Die nachfolgenden Erläuterungen sind lediglich Hinweise, wie das komplexe Problem Suchmaschine angegangen werden kann. Sie können diese Überlegungen einfach ignorieren und ein eigenes Design entwerfen, allerdings müssen Sie die oben (Kapitel 4) spezifizierten Anforderungen (Bestehen der JUnit-Tests, Nutzung eines Invertierten Indexes, Bedienung der von der GUI geforderten Schnittstellen) sämtlich erfüllen.

Auch wenn es vielleicht etwas trivial klingt: Überlegen Sie, wie Sie ein Problem lösen können, vielleicht auch ohne Computer, sondern mit Zettel und Stift. Versuchen Sie, das Problem zunächst grob zu umschreiben und versuchen Sie dann, einzelne Teilprobleme darin ausfindig zu machen. Dann können Sie diese Strategie rekursiv auf die Teilprobleme anwenden, bis Sie schließlich feststellen, dass Sie nur eine relativ große Menge von einfachen Problemen lösen müssen.

Es ist kein guter Stil, alle Funktionalität in einer Klasse zu implementieren, auch wenn das Interface ISearchEngine dies suggerieren mag. Klassen, die mehr als 300 Zeilen Code enthalten, sollten vermieden werden – stattdessen bietet es sich an, Aufgaben aufzuteilen und an Hilfsobjekte - auch Komponenten genannt - zu delegieren. Ähnliches gilt auch für Methoden – hier gilt als Richtwert 40 Zeilen, die nicht überschritten werden sollten. Versuchen Sie also auch hier, eine Funktionalität in Teilfunktionen zu zerlegen. Natürlich sind dies lediglich Richtwerte, Sie sollten jedoch versuchen, sich an diesen zu orientieren, Abweichungen zu vermeiden und zu begründen.

Nutzen Sie außerdem die Möglichkeit, eigene Pakete anzulegen und so Ihr Programm weiter zu strukturieren. Beispielsweise bieten sich Pakete an, welche die Komponentenklassen Ihrer Suchmaschine von den Klassen für die Datenstrukturen trennen.

5.1 Komponenten einer Suchmaschine

Eine Suchmaschine (SearchEngine) besteht im Grunde aus zwei Hauptkomponenten: Dem Indexerzeuger (Indexer) und der Suchanfragenbearbeitung (QueryEngine). Die Indexerzeugung selbst besteht aus dem Einlesen von Daten aus Textdateien (Crawler, Reader), dem Tokenisieren (Tokenizer) und der Speicherung der eingelesenen Daten (Indexer). Diese Überlegungen führten in der Vorlage zu folgenden Komponenten:

Komponente	*Konsumiert*	*Liefert*
Crawler	Ordner	Alle enthaltenen Textdateien
Reader	Textdatei	Text als String
Tokenizer	Text als String	Wörter mit Positionsangabe
Indexer	Wörter mit Positionsangabe	Gesamtindex, Dokumentenindex
QueryEngine	Suchanfrage als String	Texte mit Markierungen

Die Komponenten *Crawler* und *Reader* sind relativ trivial zu implementieren, hierfür können Sie gerne auch Lösungen zu im Kurs gestellten Hausaufgaben nutzen (so Sie sie denn angefertigt haben sollten).

Beim *Tokenizer* könnte man auf die Idee kommen, die methode `split()` der Klasse String zu nutzen. Allerdings liefert diese nicht die Positionen der einzelnen Tokens, die jedoch benötigt werden, um gefundene Suchbegriffe in der GUI darzustellen. Sie müssen sich also eine andere Tokenisierungsmethode überlegen.

Auch die Suchanfragen für die *QueryEngine* müssen tokenisiert werden, dafür bietet sich an, denselben Tokenizer wie der Indexer zu benutzen, da der Index wie auch Suchanfrage durch den gleichen Algorithmus bearbeitet werden sollten, um zu vermeiden, dass gleiche Wörter unterschiedlich behandelt und dadurch nicht gefunden werden.

Die QueryEngine soll zu jeder Suchanfrage eine Liste von *IResults* zurückgeben, deren einzelne Elemente (also die IResults) jeweils ein Dokument repräsentieren. Wie am Interface IResult zu sehen ist, liefert ein Resultat neben der Pfadangabe zum Ergebnistext dem vollen Text (um ihn in der GUI anzeigen zu können) und eine Liste von in einer HashMap gebündelten Integern, welche die Start- und Endposition der gesuchten Wörter enthält (da diese in der GUI farblich unterlegt werden).

5.2 Verwaltete Daten einer Suchmaschine

Der in Kapitel 2.2 vorgestellte invertierte Index enthält nur die Information, welche Wörter in welchen Texten enthalten sind, nicht aber wie oft und wo im Text sie enthalten sind. Diese Daten werden aber von unserer Suchmaschine benötigt, da sie die Ergebnisse nach Häufigkeit sortieren und die gesuchten Wörter in den Ergebnistexten hervorheben soll. Zur Bündelung dieser Informationen bietet sich eine eigene Klasse an (hier DocumentData genannt):

Datenstruktur	*Enthält*
Index	Alle vorhandenen Wörter mit Postingslist (Dokumentenfundstellen)
DocumentData	Informationen über einzelne indizierte Dokumente: Pfad zur Datei, ID, Zeitpunkt der letzten Änderung, Wörter mit Positionen...

5.3 Noch Fragen?

Bitte melden Sie sich per Mail an matanass@uni-koeln.de. Wenn Sie lediglich auf gut Glück in der Sprachlichen Informationsverarbeitung vorbeikommen, kann es sein, dass Sie vor verschlossenen Türen stehen!