Dokumentation
Impressumscrawler

Marius Messer
11. Juli 2023

Inhaltsverzeichnis

1 Einführung

Dies ist Dokumentation zum Impressumscrawler, der im Rahmen der Bachelorarbeit "Entwicklung und Evaluation eines Impressumscrawlers - ein Webcrawler spezialisiert auf die automatische Impressumsextraktion von Webseiten unter Berücksichtigung der deutschen Rechtslage" entwickelt wurde. Er bietet Funktionen zur Extraktion von Feldern eines Impressums und kann auch in einem einfachen Rahmen konfiguriert und in seinen Funktionen erweitert beziehungsweise verbessert werden. Zusätzlich bietet es sich jedoch an, den Impressumscrawler programmiertechnisch zu erweitern, um bestehende Regeln zu erweitern oder das Programm für spezielle Nutzungsszenarien anzupassen. Die Dokumentaion des Crawlers unterteilt sich somit in sechs grundlegende Abschnitte:

- Nutzung des Crawlers
- Konfiguration des Crawlers
- Beschreibung des Crawlers
- Erweitern und Modifizieren bestehender Regeln
- Implementation der verschiedenen Extraktions Komponenten
- Dokumentation der verschiedenen Feld-Objekte

Innerhalb des Abschnittes "Nutzung des Crawlers" wird die Interaktion mit dem Impressumscrawler und wie seine Funktionalitäten genutzt werden können erläutert. Das bedeutet: Wie werden Inputs an den Crawler übergeben und welcher Output ist zu erwarten.

Unter dem Punkt "Konfiguration des Crawlers" wird genauer beschrieben, unter welchen Umständen es sinnvoll ist, die Konfiguration des Crawlers zu ändern oder zu erweitern.

Beim Abschnitt "Beschreibung des Crawlers " werden die Architektur und die Packages des Crawlers genauer erläutert.

Der Abschnitt "Erweitern und Modifzieren bestehender Regeln" erklärt anhand eines Beispiels, wie vorzugehen ist, wenn Abläufe im Code zu Fehlern führen und eine Ergänzung durchgeführt werden muss.

Der Punkt "Implementation der verschiedenen Extraktions Komponenten " beschreibt die aktuell bestehenden Implementationen der verschiedenen Regeln, welche zum extrahieren der Felder programmiert wurden.

Innerhalb des letzten Punktes "Dokumentation der verschiedenen Feld-Objekte" befindet sich die Dokumentation für die einzelnen Variabeln und Funktionen der Klassen, des "datatypes-Packages ", welche die Daten der Felder eines Impressums beinhalten.

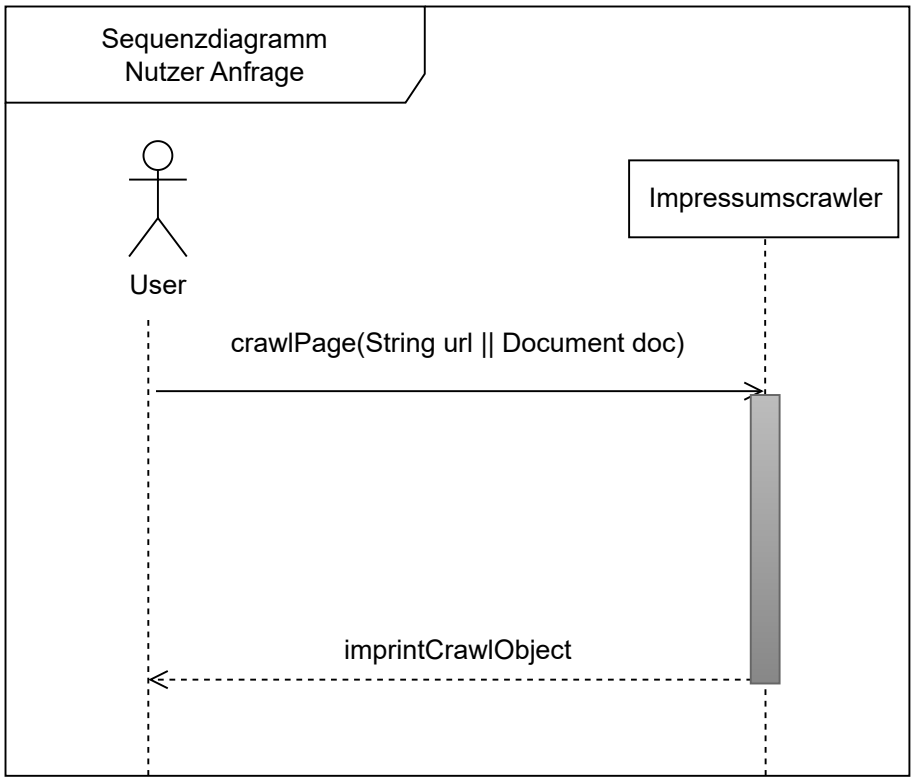

Abbildung 1: Sequenzdiagram der Nutzerinteraktion

2 Nutzung

Um eine Homepage nach einem Impressum zu crawlen, ist es lediglich erforderlich, ein "Impressumscrawler" Objekt zu erstellen. Dieses Objekt bietet die überladene "crawlPage()" Methode an. Einmal kann man diese Funktion mit einem String als Parameter aufrufen, die Url sollte dabei der Form

www.<Domain>.<Ländercode>

entsprechen. Und es gibt die Möglichkeit, dem Impressumscrawler direkt ein Jsoup Document zu übergeben, womit die Suchfunktion des Crawlers übersprungen und das übergebene Dokument als ein Impressum betrachtet wird (siehe Abbildung 1). Der Output der "crawlPage()" Methode ist in beiden Fällen ein "ImprintCrawlObject", wobei bei letzterer Art der Anfrage, das "ImprintSearchObject" auf "Null" gesetzt ist.

Das "ImprintCrawlObject" beinhaltet dabei im Wesentlichen zwei Objekte, einmal das "ImprintSearchObject" und einmal das "ImprintObject" (siehe Abbildung 2). Ersteres beinhaltet Informationen über die Suche und Letzteres

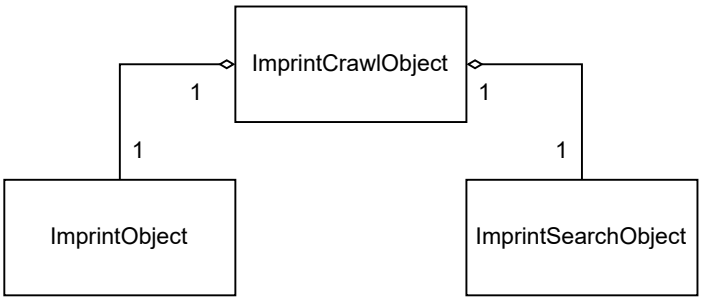

Abbildung 2: Komposition eines ImprintCrawlObjects

beinhaltet Informationen über das Impressum und die Impressumsfelder. Eine genauere Beschreibung der Objekte kann der technischen Dokumentation entnommen werden. Sollte lediglich eine leicht lesbare Zusammenfassung gewünscht sein, kann dies, für die Suche, über die "report()" Funktion des "ImprintSearchObject" Objektes und, für die Impressumsinhalte, die gleichnamige Funktion des "ImprintObject" Objektes angefordert werden.

3 Konfiguration des Impressumscrawlers

In der Extraktion von Feldern greift der Impressumscrawler häufig auf Indikatoren zurück, die einen Hinweis darauf geben, dass sich an der aktuell untersuchten Stelle des Dokumentes, mit hoher Wahrscheinlichkeit, eine gesuchte Information befindet. Da im Rahmen der Entwicklung nicht davon auszugehen ist, dass alle möglichen Indikatoren für ein bestimmtes Feld gefunden werden können, besteht die Möglichkeit, die zugrundeliegenden Daten zu erweitern. Um einen entsprechenden Indikator hinzuzufügen, muss innerhalb des Ordners "/src/config" die entsprechende Datei mit den Indikatoren erweitert werden. Es besteht gleichzeitig die Möglichkeit, unerwünschte Indikatoren zu entfernen. Für die folgenden Indikatoren gibt es die Möglichkeit der Konfiguration:

- Berufsordnung (berufsordnung_indicators)
- Datenschutz (datenschutz_indicators)
- Redaktionelle Rolle (editor_indicators)
- Firma (firm_indicators)
- Redaktioneller Inhalt (redaktions_indicators)
- Straße (street_indicators)
- Verkammerte Berufe (verkammerte_berufe_indicators)
- Vertreter (vertreter_indicators)
- Nummern (numeric_number_indicators)

Jede Zeile entspricht dabei einem Indikator. Sollte es Probleme mit den Konfigurationen geben, fällt der Impressumscrawler auf Standardwerte zurück. Diese Standardwerte können ebenfalls der technischen Dokumentation entnommen werden.

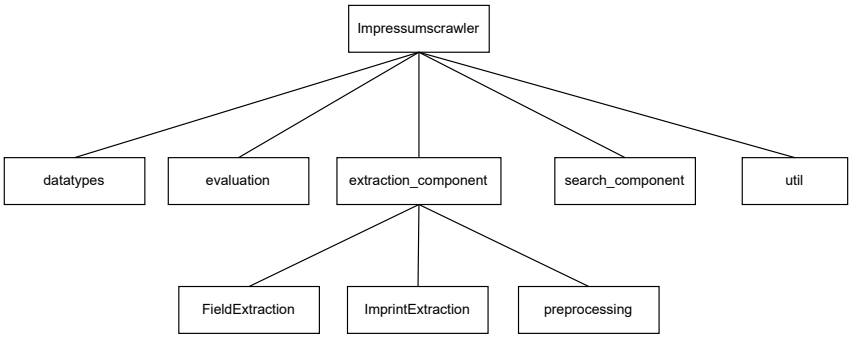

Abbildung 3: Package Struktur des ’impressumscrawler’ Packages

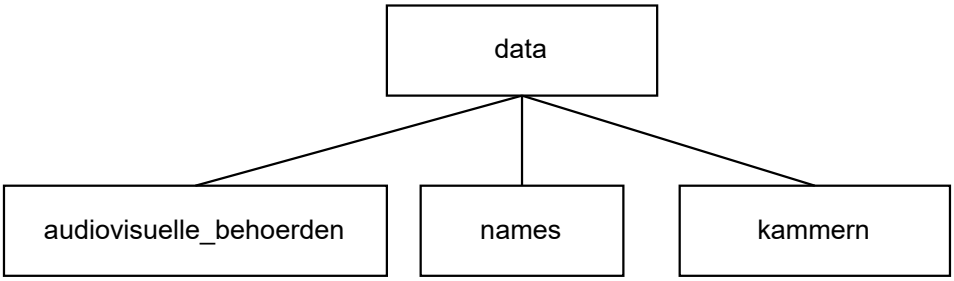

Abbildung 4: PackageStruktur des ’data’ Packages

4 Beschreibung des Crawlers

4.1 Programmstruktur

Der Impressumscrawler besteht in oberster Ebene aus zwei Packages. Einmal dem "impressumscrawler" Package und einmal dem "data" Package. Ersteres beinhaltet dabei die konkreten Funktionalitäten des Crawlers und Datentypen für die einzelnen Felder, sowie die ansonsten eng mit dem Crawler verbundenen Funktionen. Das "data" Package umfasst Web-Scraper, welche Daten zu bestimmten domänenspezifischen Informationen sammeln, abspeichern und über Klassen im Singleton Pattern den Impressumscrawler Komponenten zur Verfügung stellen. Das "Impressumscrawler" Package hat als Unterpackages ein Package für die Datentypen, eins für die Evaluation, die Extraktionskomponente, die Suchkomponente und ein Package für Utility-Funktionen (siehe Abbildung 3). Das "data" Package hat Unterpackages für die audiovisuellen Behörden, die Kammern und Vornamen (siehe Abbildung 4).

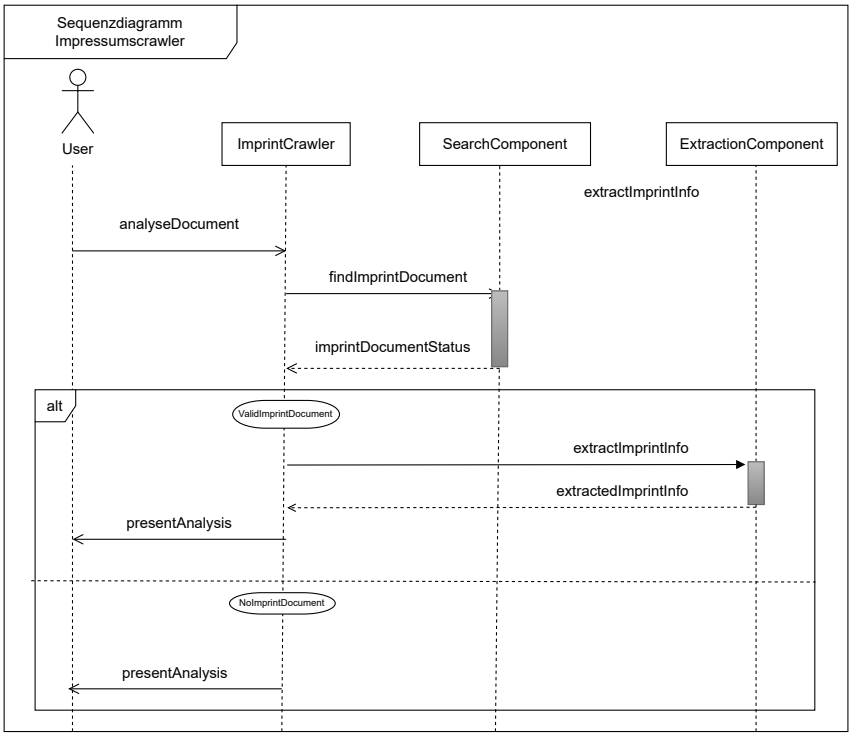

Abbildung 5: Interaktion der beiden Unterkomponenten und dem Impressumscrawler bei einer Anfrage

4.2 Impressumscrawler Package

Innerhalb des "impressumscrawler" Packages ist die "Impressumscrawler" Klasse enthalten. Diese Klasse bildet das Herzstück des Crawlers und ist die Klasse, mit der ein Nutzer interagiert, sollte er eine Crawl-Anfrage stellen. Die "Impressumscrawler" Klasse verarbeitet die Anfrage, leitet sie weiter und gibt die Ergebnisse gesammelt zurück (siehe Abbildung 5).

4.3 datatypes Package

Innerhalb eines Impressums können mehrere voneinander unterschiedliche und unabhängige Informationen auftreten. Um all diese Informationen sinnvoll und strukturiert zu erfassen, wurden 12 verschiedene, auf die Felder eines Impressums spezialisierte, Datentypen erstellt. Jeder Datentyp kann dabei indirekt in eine Beziehung zu anderen Datentypen gesetzt werden, da Informationen über die Elemente, in denen die feldspezifischen Informationen gefunden wurden, mit abgespeichert werden. So ist es möglich, in einer späteren Analyse die entsprechenden Felder semantisch zu verknüpfen.

die feldspezifischen Datentypen sind dabei wie folgt:

- AdressObject
- AufsichtsObject
- BerufsordnungsObject
- ChamberObject
- ContactObject
- EditorInfoObject
- NumericContactObject
- RechtsformObject
- RedaktionsObject
- RegisterObject
- Ust_IdentificationNumber
- VertretungsberechtigterObject

Des Weiteren gibt es drei Datentypen spezfifisch für das Zusammenführen der Ergebnisse und die Suche:

- ImprintCrawlObject
- ImprintObject
- ImprintSearchObject

Das "ImprintObject" wird von dem "ImprintExtractor" erstellt und enthält gesammelt neben den Daten für die einzelnen Felder ebenfalls Ergebnisse einer Analyse. Die Analyse versucht zu ermitteln, welche gefundene Adresse die Hauptadresse und welche die jeweilig verantwortlichen Vertreter sind und ob zu manchen Feldern Informationen gefunden wurden oder nicht.

4.4 extraction_component Package

Die Extraktionskomponente unterteilt sich in mehrere Subkomponenten. Die Extraktion der Impressumsinformationen ist in zwei wesentliche Teile unterteilt. Der Extraktion der Felder und Extraktion des tatsächlichen Impressums. Innerhalb der Feld-Extraktion wird davon ausgegangen, dass die Felder unabhängig voneinander Eigenschaften besitzen, mit denen sie gesondert extrahiert werden können. Bei der tatsächlichen Impressums-Extraktion werden diese aufgefundenen Daten dann miteinander in Verbindung gebracht, und es werden Annahmen über die wesentlichen und wichtigen Informationen des Impressums gemacht und speziell extrahiert. Zusätzlich gibt es eine Preprocess Komponente, welche dazu da ist, ein gegebenes Dokument von nicht notwendigen bzw. nicht strukturrelevanten Informationen zu bereinigen. Nach aktuellem Stand sind dies <stronger> und Tags. Eine weitere Funktion des Preprocessings, die sinnvoll sein könnte, wäre das eliminieren von verschachtelten Elementen, die keinen Inhalt führen.

4.4.1 FieldExtraction Package

In dem Package ist die FieldExtractor Klasse enthalten. Sie hält als Variable ein Jsoup.Document, das über die Funktion s̈etCurrent_document(Document current_document)g̈esetzt werden kann. Die Klasse bietet Funktionen für die feldspezifische Extraktion von Informationen von dem aktuell gesetzten Dokumentes und geht davon aus, dass es sich um ein Impressum handelt. Die Funktionen, die für ein Feld des Impressums zuständig sind, beginnen mit einem ’find´,andere Funktionen dienen diesen Extraktionsfunktionen bei dem Erarbeiten der Aufgabe.

- findKontaktformular()
- find_email_contact()
- find_numeric_contact()
- findEditorInfo()
- findRedaktionellerAngabe()
- findBerufsordnung()
- findVerkammerteBerufe()
- findRegister_Registernummer()
- findZustaendigeAufsichtsbehoerde()

- findUst()
- findVertretungsberechtigten()
- findRechtsform()
- findPLZandCity()

Die Namen dieser Funktionen beschreiben an und für sich ihren Zweck. Eine zusätzliche Information bezüglich der "findPLZandCity()" Funktion, in ihr wird ebenfalls ein potenzieller Adressat gesucht. Eine Ausnahme bezüglich der Namenskonvention stellt die "findAllNames()" Funktion dar. Diese Funktion hat als Aufgabe, alle Namen innerhalb eines Dokumentes zu extrahieren.

Eine weitere Namenskonvention innerhalb der Funktionen ist das Wort "check". Es gibt an, dass eine bestimmte Sache überprüft werden soll und, falls vorhanden, einen entsprechenden Wahrheitswert zurückgibt. Eine Ausnahme gibt es auch hier. Die c̈heck_return_matches()F̈unktion, welche die gematchten Indikatoren als eine Liste von Strings zurückgibt.

Funktionen, welche keinerlei Präfix haben, sind welche, die innerhalb einer Feld-Extraktion aufgerufen werden. Meistens, wenn der Code sich in mehrere Pfade aufteilt oder der Prozess logisch abgekapselt werden kann.

1. extractNames(Element e) - wird angewendet, wenn ein Vertreter Indikator gefunden wird und Namen in dem gleichen, dem Element danach, oder darüber gefunden wird
2. noStreetNoName(Element e, AdressObject ao) - findet Anwendung, falls bei einer Adressfindung bisher nur eine Postleitzahl gefunden wurde, schaut in den vorherigen Elementen nach Straße und Adressaten.
3. foundStreetNoName(Element e, AdrtessObject ao - ebenfalls bei einer Adressfindung, sucht die Adresse, Postleitzahl und Straße wurden bereits gefunden.
4. previous_sibling_non_br(Element element) - rekursive Funktion, um zu schauen, ob es ein vorheriges Geschwister Element gibt, das Text enthält. Ein
 Tag ist ebenfalls ein Element, jedoch wird bei einem Versuch auf seinen Text zuzugreifen, eine NullPointerException geworfen, weshalb es sich anbietet es zu überspringen.

4.4.2 ImprintExtraction Package

Innerhalb dieses Packages befindet sich die "ImprintExtractor" Klasse. Sie hat die Aufgabe, die gewonnenen Informationen aus der Feld-Extraktion in

eine Ordnung zu bringen und in einem ImprintObject hinzuzufügen, welches die gesamten Informationen über das Impressum einer Webseite beinhaltet.

Die Klasse hält ebenfalls ein Jsoup.Document als Variable, das dem aktuell zu untersuchenden Dokument entspricht. Es kann über die Funktion "setDocument(Document doc)" gesetzt werden und setzt es gleichzeitig für das eigene "FieldExtractor" Objekt.

Für die Analyse des Dokumentes wird die Funktion "AnalyzeDoc" aufgerufen. In dieser werden in einem ersten Schritt alle Informationen der Felder gespeichert. Im Anschluss wird eine Analyse durchgeführt, welche wesentlichen Informationen zusammengehören.

Diese Klasse kann ebenso die Verantwortung für das Generieren von Informationen über Felder eines Impressums haben. Zum aktuellen Stand wird das Feld der Rechtsform über den Adressaten einer E-Mail überprüft.

Innerhalb des "ImprintExtractors" gibt es ebenfalls zwei weitere Funktionen. Die Funktion "adressed_is_person(AdressObject ao)" überprüft, ob innerhalb des Adressaten ein Name zu finden ist. Als auch die "check_return_matches()" Funktion, welche die gematchten Indikatoren als eine Liste von Strings zurück gibt.

4.5 preprocessing Package

Dieses Package beinhaltet die "Preprocessor" Klasse, innerhalb dieser Klasse gibt es eine Funktion. Diese Funktion hat den Namen "preprocess_document(Document doc)". Das Dokument, das entgegen genommen wird, wird in einen Text umgewandelt und alle Tags, die in dem "non_structural_tags_array" Array stehen, werden aus dem Dokument entfernt. Im Anschluss wird das Dokument wieder in ein Jsoup.Document umgewandelt.

Das Entfernen der Tags hat den Vorteil, dass semantisch zusammengehörige Informationen nicht durch Tags getrennt werden. Beispielsweise können Tags genutzt werden, um Meta-Informationen zu verschiedenen Textblöcken hinzuzugeben. Diese stören jedoch die Struktur und verhindern eine entsprechende Analyse.

4.6 search_component Package

Innerhalb dieses Packages findet die Suche nach einem Impressums-Dokument innerhalb einer Webseite statt. Die Klasse, die dafür zuständig ist, ist die "ImprintFinder" Klasse. Diese bietet zwei Möglichkeiten, das Impressum einer Homepage zu extrahieren. Einmal durch das Übergeben einer Url und einmal durch das Übergeben eines Jsoup.Documents. Im ersteren Fall fragt die Funktion selbstständig eine Url an und erstellt ein Jsoup.Document für die

weitere Weiterverarbeitung. Im zweiten Fall wird das übergebene Dokument als Homepage betrachtet und darauf die Suche durchgeführt.

Abschließend wird von der Suchkomponente ein "ImprintSearchObject" erstellt, was mit den entsprechenden Informationen bezüglich der Suche angereichert wird. Auch falls keine Impressumsinformationen aufgefunden werden können, wird diese Information entsprechend hinterlegt.

4.7 util Package

Das Package beinhaltet die "Util " Klasse. Dies Klasse bietet verschiedene, nicht Funktional relevante, statische Methoden, welche von den verschiedenen Komponenten genutzt werden.

- *getConfigStringArrayFromFile(String fileName)* - liest die Konfigurationsdateien aus und erstellt einen entsprechenden String-Array
- *getArrayListFromFile(File file* - ähnlich wie "getConfig" Funktion. Nur das Liste wieder gegeben wird
- *getArrayListFromFile(String path* - statt einer Datei muss nur ein Pfad zu einer Datei angegeben werden
- *writeImprintObjectToPath(String path, ImprintObject imprintObject* - nutzt die "report" Funktion des "ImprintObject" Objekts und schreibt den Bericht in den, in "path", angegebenen Pfad
- *wroteStringToPath(String path, String content* - schreibt einene übergebenen String an den angegebenen Pfad

5 Erweitern und Modifizieren bestehender Regeln

Der "Vertrag", der zwischen den verschiedenen Funktionen steht, sind die entsprechenden Rückgabewerte der Funktionen und somit die entsprechenden feldbezogenen Objekte des "datatypes" Packages. Sollte beispielsweise ein umgesetzter Algorithmus fehlerhaft sein, so ergibt es sich, über die Information des Feldes zu schauen, an welcher Stelle ein solcher Fehler aufgetreten ist. Am besten wird diese Situation einmal an einem Beispiel näher erläutert. Innerhalb einer Analyse (der Evaluation) ist bezüglich der Adressen aufgefallen, dass Adressen neben der Struktur wie im folgenden Codebeispiel:

```
<tag>
    <tag>Adressat</tag>
```

```
    <tag>STRASSE</tag>
    <tag>PLZ + Ort</tag>
</tag>
```

auch in folgender Form vorkommen können:

```
<tag>
    <tag>ADRESSAT</tag>
    <tag>GESCHÄFTSFÜHRER INDIKATOR: NAME</tag>
    <tag>STRASSE</tag>
    <tag>PLZ + Ort</tag>
</tag>
```

Was als Folge hat, dass der eigentliche Adressat nicht hinzugefügt wird, sondern der Geschäftsführer Indikator mit dem Namen. Um diese Eventualität in der Analyse mit in Betracht zu ziehen, sucht man innerhalb der Codebase nun nach dem entsprechenden Setter für das korrespondierende Feld des "AdressObjects" Objektes. Es kann festgestellt werden, dass der Wert, der zugewiesen wird, innerhalb des Strings "adressed" steht, welcher wiederum dem Wert von "adress_matches.get(2).getValue()" entspricht. Betrachten wir die "adress_matches" Datenstruktur, erkennen wir, dass es sich um eine Struktur handelt mit drei änderbaren Paaren:

```
ArrayList<MutablePair<String, String>> adress_matches =
                                            new ArrayList<>();
adress_matches
   .add(new MutablePair<String, String>("PLZ + Ort:", null));
adress_matches
   .add(new MutablePair<String, String>("Straße:", null));
adress_matches
   .add(new MutablePair<String, String>("Adressat:", null));
Iterator<MutablePair<String, String>> adress_matches_it =
   adress_matches.iterator();
```

Zusätzlich wird ein Iterator für dieses Objekt erstellt. Insgesamt ergeben sich drei Stellen, an denen Informationen für eine Adresse zu finden sind. Der für diesen Regel-Zusatz relevante Teil ergibt sich aus dem Fall, dass die Adresse in einem vorherigen Element enthalten ist. Somit müssen die Funktionen "noStreetNoName" und "foundStreetNoName" berücksichtigt werden. Innerhalb dieser sind die entsprechenden Stellen gekennzeichnet, wo die Straße und wo die Adresse gesetzt wird. Um die Regel zu realisieren, soll das Element, das den Indikator "Geschäftsführerënthält, übersprungen werden. Somit wird an die entsprechend Stelle innerhalb des Codes eine entsprechende Überprüfung eingefügt. Folgender Teil wird:

```
String previous_sibling_as_string = previous_sibling.toString();
```

entsprechend ergänzt:

```
String previous_sibling_as_string =
    previous_sibling.toString();
boolean element_contains_geschaeftsfuehrer_indikator =
    check(new String[]{"geschäftsführer"},
            previous_sibling_as_string);
if(element_contains_geschaeftsfuehrer_indikator) {
    previous_sibling_as_string = previous_sibling
        .previousElementSibling()
        .toString();
}
```

Und die gleiche Ergänzung findet noch einmal in der jeweils anderen Funktion statt. Somit wurde der Algorithmus entsprechend erweitert. In einem nächsten Schritt sollte natürlich noch überprüft werden, dass die Änderungen auch zu korrekten Ergebnissen führen. In einem solchen Fall ließe es sich auch überlegen, ob innerhalb der "AdressObject" Klasse, ein weiteres Feld für einen Geschäftsführer

Wird eine ganz neue Regel für die Extraktion eines Feldes erstellt , so sollte sie nach Möglichkeit in einer Funktion implementiert werden, jedoch den gleichen Rückgabewert, also entweder ein AdresObject Objekt oder eine Liste von Adressobject Objekten. Welche der beiden Funktionen, oder ob beide Funktionen in der Impressumsanalyse schließlich berücksichtigt kann nach aktuellem stand im "ImprintExtractor" festgelegt werden.

Sollte es gewünscht sein, eine kleinere Analyse des Dokumentes durchzuführen, so besteht ebenfalls die Möglichkeit, einfach über das "Impressums-Crawler" Objekt gezielt auf die entsprechenden Subkomponenten zuzugreifen und bestimmte Felder einzeln zu extrahieren.

6 Implementation der verschiedenen Extraktions Komponenten

6.1 Adressfindung

6.1.1 Zu implementierende Regel

finde PLZ + Ort $\rightarrow$ Segmente vorher bilden Straße und Adressat

6.1.2 Implementierung

Die Implementierung finder innerhalb der "findPLZandCity()" Funktion der "FieldExtractor" Klasse statt. Zuerst wurde ein regulärer Ausdruck, zum Auffinden einer Postleitzahl und dem dazugehörigen Ort, erstellt. Im Anschluss wird über eine Liste mit allen Elementen des Dokumentes iteriert und alle Matches werden einem HashSet hinzugefügt, sodass keine Einträge im nächsten Schritt doppelt berücksichtigt werden.

```
HashSet<String> found_numbers = new HashSet<String>();
for (Element e : elements) {
    Matcher matcher =
        plz_location_pattern
            .matcher(e.toString());

    if(matcher.find()) found_numbers.add(matcher.group());
}
```

Nachdem alle in dem Dokument vorhandenen PLZ + Ort Stellen aufgefunden wurden, wird über das HashSet iteriert und alle Elemente, welche die entsprechenden PLZ + Ort Kombinationen enthalten, in einer Liste aus Paaren gespeichert.

```
for (String match : found_numbers) {
    plz_element_pairs
        .add(
            new Pair<String, Elements>(
                match,
                current_document
                    .getElementsContainingOwnText(match)
            )
        );
}
```

Der linke Eintrag des Paares ist dabei die PLZ + Ort Kombination und der rechte Eintrag, eine Liste von Elementen, welche die PLZ + Ort Kombination enthalten. Über die Liste an Paaren wird nun erneut iteriert und innerhalb des Loops wird über die Liste an Elementen iteriert. Dabei wird das Element in Segmente unterteilt, mittels eines entsprechenden Patterns.

```
Pattern tag_matcher = Pattern.compile("[,>][^,<>]+[,<]");
```

Nun können mehrere Fälle eintreten: Der Adressat und die Straße sind im selben Element enthalten, oder die Straße ist im selben Element, der Adressat

jedoch nicht, oder der Adressat und die Straße sind nicht im selben Element enthalten. Bevor nach den entsprechenden Feldern gesucht wird, wird eine Liste aus MutablePairs erstellt, welche mit den Informationen der PLZ + Ort, der Straße und dem Adressaten befüllt werden soll. Dafür wird zusätzlich ein Iterator erstellt, damit flexibel, je nach Eintreten der Fälle, diese Liste befüllt werden kann. Zusätzlich wird ein "AdressObject" Objekt, welches über den Prozess der Adresssuche mit den gefundenen Informationen angereichert wird, sodass für eine Weiterverarbeitung alle nötigen Informationen zur Verfügung stehen.

Ein schematischer Ablauf des Algorithmus' ist in Abbildung 6 zu sehen.

6.2 Aufsichtsbehörde

6.2.1 Zu implementierende Regel

Es gibt einen Indikator für Aufsichtsbehörde $\rightarrow$
nächste gefundene Adresse im Document ist Adresse der Aufsichtsbehörde

6.2.2 Implementierung

Die Implementierung findet innerhalb der Funktion "findZustaendigeAufsichtsbehoerde()" der "FieldExtractor" Klasse statt und später innerhalb der "analyzeDoc()" der "ImprintExtractor" Klasse. Innerhalb der "findZustaendigeAufsichtsbehoerde()" Funktion werden zu Beginn alle Elemente des Dokumentes, welche das Wort "Aufsichtsbehörde" enthalten, gesammelt. Im Anschluss wird ein "Aufsichtsobject" Objekt erstellt. In diesem wird ein Wahrheitswert gespeichert, ob Elemente mit dem Inhalt gefunden wurden, und die Liste an Elementen wird übergeben.

Innerhalb der "analyzeDoc()" Funktion des "ImprintExtractors", wird zu jedem Element geschaut, welche Adresse die nächste ist.

```
for (AdressObject ao : adressObjects) {
    int adressObject_index =
        elements
            .indexOf(ao.getAdressed_element());
    int distance = adressObject_index - aufsichts_index;
    if(distance < closest_distance && distance >= 0) {
        closest_distance = distance;
        closest_adressObject = ao;
```

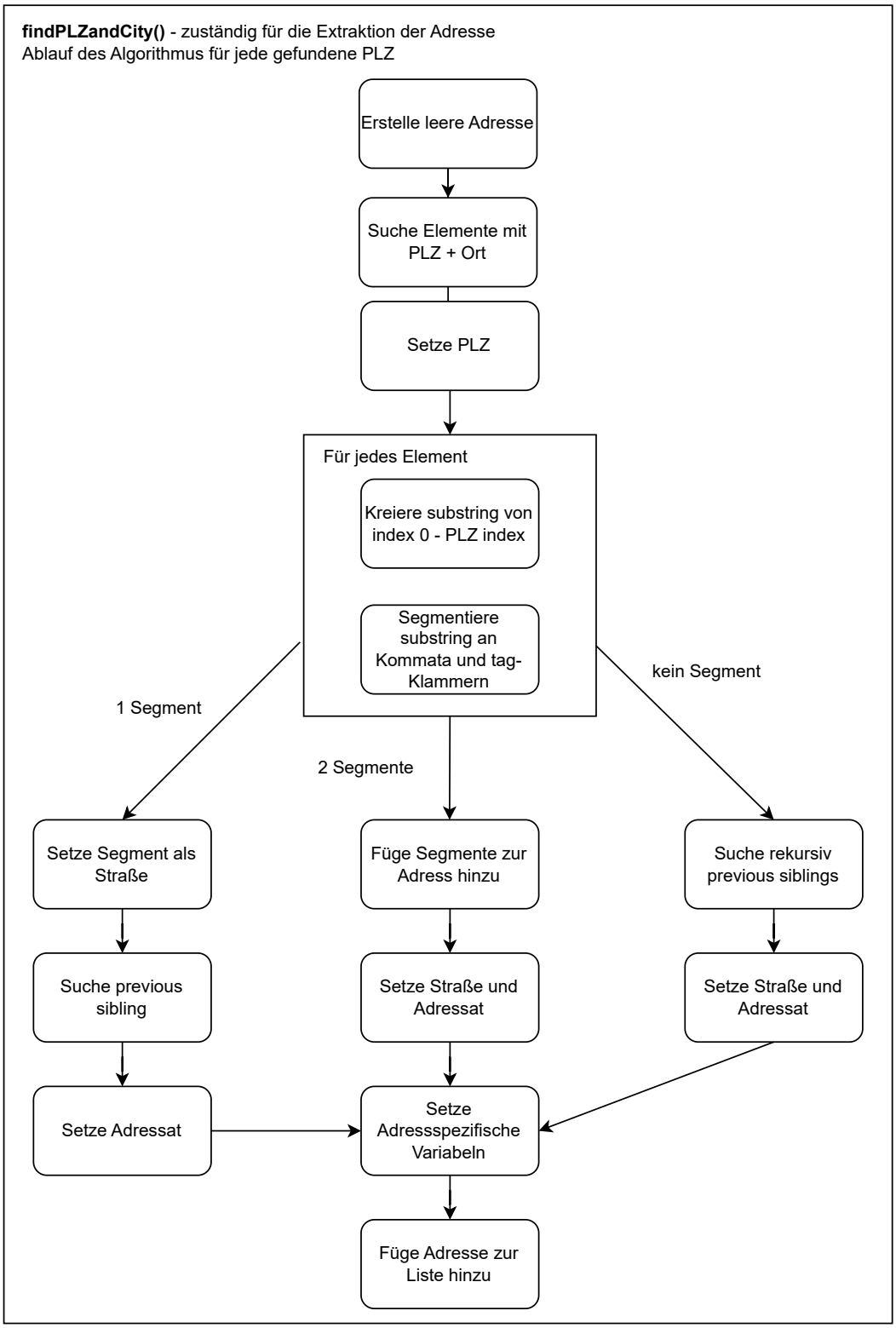

Abbildung 6: Ablauf der Adressfindung

```
    }
}
```

Sollte eine nachstehende, nahe Adresse existieren, wird diese als Aufsichtsbehörde vermerkt und an das "ImprintObject" Objekt übergeben.

6.3 Berufsordnung

6.3.1 Zu implementierende Regeln

Ein Element hat Ordnungs Indikatoren und keine Datenschutzindikatoren → Element enthält relevante Berufsordnung

Ein Element hat Regelungs-Indikatoren und keine Datenschutz-Indikatoren → Element enthält relevante berufsrechtliche Nennung

6.3.2 Implementierung

Die Funktion "findBerufsordnung()" der "FieldExtractor" Klasse ist für die Extraktion von Berufsordnungen zuständig. Hierzu wird über alle Elemente eines Dokumentes iteriert und je nach vorliegenden Indikatoren, wird das Element einer entsprechenden Liste hinzugefügt:

```
    for (Element e : elements) {
        String e_ownText = e.ownText();
        if (check(regelung_nennung_indicator, e_ownText) &&
            !check(datenschutz_indicator, e_ownText))
        {
            elements_with_nennung.add(e);
        }
        if(check(regelung_indicator, e_ownText) &&
            !check(datenschutz_indicator, e_ownText))
        {
            elements_with_regelung.add(e);
            regelungen.add(e_ownText);
        }
    }
```

die Indikatoren des "regelung_nennung_indicator" Arrays zeigen an, ob eine berufsrechtliche Nennung vorliegt. Die Indikatoren des "regelung_indicator" Arrays zeigen an, wenn eine Ordnung oder ähnliches angegeben wird. Über die Bedingung, dass keine Datenschutz Indikatoren in dem Element enthalten

sein dürfen, wird dafür gesorgt, dass typische, nicht Impressumsrelevante Template Texte fälschlicherweise für berufsrelevante Ordnungen und Satzungen gehalten werden.

6.4 Einzelne Personen und Verantwortlichkeit

6.4.1 Zu implementierende Regel

Indikator&ein naher Name $\rightarrow$
redaktionell verantwortliche Person wurde gefunden

6.4.2 Implementierung

Die Implementierung findert innerhalb der "findEditorInfo()" Funktion der "FieldExtractor" Klasse statt.

Zuerst werden die Indikatoren initialisiert.

```
String[] rollen =
    {"Redakteur", "Leiter", "Webredaktion",
    "Redaktionelle Mitarbeit", "Redaktionsassistenz",
    "Verantwortliche Redakteurin", ...};
```

Und alle Namen und das Element, in dem sie enthalten sind, werden extrahiert.

```
    ArrayList<MutablePair<String, Element>> names_in_doc_list =
                                                findAllNames();
```

Über die Indikatoren wird nun iteriert, und es wird geschaut, ob sie innerhalb des Dokumentes enthalten sind.

```
    Elements with_role_indicator =
        elements.select("*:containsOwn("+s+")");
```

Wenn Elemente mit Indikatoren gefunden werden können, wird über diese ebenfalls iteriert, und es wird aus der Liste an Namen geschaut, welcher Name dem Indikator Element am nächsten ist. Dabei wird zusätzlich eine Grenze gesetzt, dass ein entsprechender Name nicht zu weit entfernt sein darf, um zum Indikator zu gehören.

```
if(e_to_name_distance < distance) {
    distance = e_to_name_distance;
    if(distance < x) {
        closest_element = mp.getRight();
    }
}
```

Sollte ein "closest_element" ungleich "null" existieren, dass eine bestimmte Länge nicht überschreitet, wird ein "EditorInfoObject" erstellt und einer Liste von "editorInfoObjects" hinzugefügt, welche abschließend zurückgegeben wird.

6.5 Elektronische Kontaktaufnahme und unmittelbare Kommunikation

6.5.1 Zu implementierende Regeln

Durchsuche mit E-Mail Pattern → Match ist eine E-Mail

Für die Telefon oder Faxnummmer ergibt sich folgende Regel:

Finde Indikator → schaue ob nachstehend Nummer → Nummer gefunden

Für das Auffinden eines Formulares ergibt sich folgende Regel:

Finde Indikator für Kontaktformular → Extrahiere Link von Element

6.5.2 Implementierung

Die Implementierung für das Kontaktformular ergibt sich dabei wie folgt: Über einen geeigneten css-Selektor werden Link-Elemente, welche einen entsprechenden Indikator beinhalten, selektiert.

```
Elements kontaktformular_links
    = elements
        .select("a[href]:containsOwn(kontaktformular)");
```

Im Anschluss wird über diese Elemente iteriert und das "href"Attribut wird extrahiert, es wird ein "contactObject" erstellt, die entsprechenden Variabeln werden gesetzt, und es wird anschließend einer Liste übergeben, welche abschließend zurückgegeben wird.

```
for (Element e : kontaktformular_links) {
    ContactObject contactObject = new ContactObject();
    String kontakt_link = e.attr("href");
    contactObject.setElement(e);
    contactObject.setElements(elements);
    contactObject.setContact_info(kontakt_link);
    contactObjects.add(contactObject);
    kontakt_list.add(kontakt_link);
}
return kontakt_list;
```

Für die E-Mail wurde ein entsprechender regulärer Ausdruck erstellt.

```
Pattern email_pattern =
    Pattern
        .compile(
            "[a-z0-9+._-]+@[a-z0-9._-]+\\.[a-z0-9_-]+"
            , Pattern.CASE_INSENSITIVE);
```

Dieser wurde auf das Dokument als String angewendet. Gefundene Matches werden einem HashSet hinzugefügt, welches anschließend in eine ArrayList umgewandelt und zurückgegeben wird.

```
for(String contact_indicator : contact_patterns) {
    Elements elements_containing_indicators =
        current_document
            .getElementsContainingOwnText(contact_indicator);
    contact_indicator_element_list
        .add(
            new Pair<String, Elements>(
                contact_indicator, elements_containing_indicators));
}
```

Für die Telefonnummern: Zuerst wird eine ArrayList an Paaren erstellt. Jedes Paar beinhaltet dabei einen Indikator und eine Liste von Elementen, welche diesen Indikator enthalten.

```
ArrayList<Pair<String, Elements>> contact_indicator_element_list =
    new ArrayList<>();

for(String contact_indicator : contact_patterns) {
    Elements elements_containing_indicators =
        current_document
```

```
                .getElementsContainingOwnText(contact_indicator);
        contact_indicator_element_list
            .add(
                new Pair<String, Elements>(
                    contact_indicator, elements_containing_indicators));
    }
```

In einem nächsten Schritt wird über diese Liste an Paaren iteriert. Dabei wird bei jeder Iteration über die dem Indikator zugehörigen Elemente iteriert. Dafür wird das Element in einen String umgewandelt und der String an der entsprechenden Stelle, wo der Indikator ist, abgeschnitten, so wird aus dem String mit dem Indikator "Tel.":

```
"Hier ist ein Text. Tel.: 000000000"
```

Folgender String:

```
": 000000000"
```

Im Anschluss wird geschaut, ob es möglich ist, mit Hilfe des Patterns für Telefonnummern ein Match zu finden. Ist dies der Fall, wird ein "NumericContactObject" Objekt erstellt, die entsprechenden Informationen werden hinterlegt und abschließend einer Liste von "NumericContactObject" Objekten hinzugefügt, welche nach Beendigung der Schleife zurückgegeben wird.

6.6 Impressumsanalyse

6.6.1 Zu implementierende Regeln

Rechtsform in Adressat $\rightarrow$ Nächster Vertreter gehört zur Adresse

Name als Adressat $\rightarrow$ Ist Hauptadresse

Nur eine valide Adresse $\rightarrow$ valide Adresse ist Hauptadresse

6.6.2 Implementierung

Die Verantwortung für die Analyse übernimmt dabei die Klasse des "ImprintExtractors". Im oFlgenden wird nun einmal das Vorgehen erläutert, auf welcher Grundlage sich der Extractor sich innerhalb der "analyzeDoc" Funktion für welche Zuordnung entscheidet.

Der genaue Ablauf, wie dabei entschieden wird, wie welche Objekte als relevant und zusammengehörig erkannt werden, wird einmal im Folgenden

genauer erläutert. In einem ersten Schritt werden alle Informationen bezüglich eines Impressumsdokumentes gesammelt. Dann wird für jede Adresse einzeln geschaut, ob der Adressat einer Firma entspricht. Im Anschluss daran wird überprüft, ob innerhalb eines Adressobjektes ein Name genannt wird. Sollte eine Adresse mit einer Firma gefunden worden sein, so wird über die gefundenen Vertretungsberechtigten iteriert und das "AdressObject", was diesem am nächsten ist, gefunden. Jedes so gefundene Paar wird in einer Liste gespeichert. Über diese Liste wird nun iteriert und sollten innerhalb des "VertretungsberechtigterObject" Objektes Namen gelistet sein, so wird die zugeordnete Adresse innerhalb des "ImprintObject" Objekts als Hauptadresse gesetzt.

Wird keine Firma und auch kein Name innerhalb der Adressen gefunden werden, so wird überprüft, wie viele der gefundenen Adressen überhaupt valide sind. Sollte nur eine Adresse innerhalb des Impressums genannt werden, beziehungsweise sollte nur eine valide Adresse vorliegen, so wird diese einzige Adresse als Hauptadresse gesetzt. Sollten mehrere Adressen vorhanden sein, so wird davon ausgegangen, dass es mehrere gleichwertige Adressen gibt, was ebenfalls entsprechend in dem "Imprintobject" Objekt hinterlegt wird. Sollten dennoch Vertreter ausgemacht werden, wird bei mehreren gefundenen Vertretern hinterlegt, dass mehrere Vertreter identifiziert wurden, jedoch kein Hauptvertreter gesetzt, liegt ein Vertreter vor, wird dieser als Vertreter gesetzt.

Die weiteren Schritte, die für die Analyse getätigt werden, beziehen sich auf die Überprüfung des Vorhandenseins von Informationen. Es wird überprüft, ob Mails vorhanden sind, ob ein Kontaktformular gefunden wurde und Werte innerhalb des "ImprintObject" Objekts entsprechend gesetzt wurden.

Falls ein Aufsichtsbehörden Indikator gefunden wurde, wird geschaut, ob eine entsprechende Adresse in unmittelbarer Nähe aufgefunden werden kann, da es die Richtlinie gibt, dass eine Aufsichtsbehörde mitsamt ihrer Adresse genannt werden soll.

Bei den Kammern werden ebenfalls Indikatoren gesetzt, einmal, ob Kammern gefunden wurden und einmal, ob Berufsbezeichnungen gefunden wurden. Sollte dies jeweils der Fall sein, werden diese auch direkt in dem "ImprintObject" hinzugefügt.

Für die Umsatzsteuer-Identifikaitonsnummern wird ebenfalls ein Indikator gesetzt, sowie die erst genannte als Haupt-Identifikationsnummer innerhalb des "Impressumsobjects" Objektes gesetzt.

Das gleiche geschieht für die Redaktionsinformationen und die Editorinformationen sowie die berufsrechtlichen Regelungen.

Die Analyse betreffend wird zuletzt die Information über das Amtsgericht und die Registernummer, sofern vorhanden, im ImprintObject hinterlegt.

Abschließend werden ebenfalls die ArrayListen und Objekte der Feldanalyse in dem Impressum hinterlegt, sodass die Informationen auch weiterhin leicht verfügbar sind.

6.7 Impressumssuche

6.7.1 Zu implementierende Regeln

Reiter Impressum → Dokument anfordern → Dokument zurückgeben

Reiter Kontakt → Dokument anfordern → Enthält "Impressum" → Dokument zurückgeben

Indikator für Kontakt oder Impressum auf Homepage →

6.7.2 Implementierung

Die Funktion "search_imprint(String homepage_url) der ImprintFinder Klasse ist für die Suche nach dem Impressums-Dokument verantwortlich.

Im ersten Schritt wird dafür ein "ImprintSearchObject" erstellt und das Dokument über die Url abgerufen.

Sollte eine Homepage aufgefunden werden können, werden die drei Regeln nacheinander überprüft. Zuerst wird geschaut, ob Indikatoren für ein Impressum innerhalb von Link-Elementen gefunden werden können.

```
for (Element link : homepage.select("a[href]")) {
    String current_link = link.absUrl("href");
    if (current_link.contains("impressum") ||
        current_link.contains("imprint"))
    {
        (...)
        imprint = request(current_link);
        imprint_found = true;
        this.imprintSearchObject.setImprint_doc(imprint);
    }
}
```

Sollte der Wahrheitswert "imprint_found" nicht erfüllt sein, so wird ein ähnlicher Vorgang für den Indikator "Kontakt" durchgeführt, mit dem Zusatz, dass bei einem gefundenen Element überprüft wird, ob ein Impressums-Indikator vorliegt.

```
for (Element link : homepage.select("a[href]")) {
    String current_link = link.absUrl("href");
    if (current_link.contains("kontakt") ||
        current_link.contains("contact"))
    {
        (...)
        imprint = request(current_link);
        for (Element e : imprint.getAllElements()) {
            if (e.ownText().contains("impressum") ||
                e.ownText().contains("imprint"))
            {
                (...)
                imprint_found = true;
            }
        }
        this.imprintSearchObject.setImprint_doc(imprint);
    }
}
```

Sollte in beiden Fällen die Suche erfolglos bleiben, wird überprüft, ob entsprechende Informationen auf der Homepage zu finden sind.

```
for (Element element : homepage.getAllElements()) {
    if (element.ownText().contains("kontakt") ||
        element.ownText().contains("contact") ||
        element.ownText().contains("Impressum") ||
        element.ownText().contains("imprint"))
    {
        this.imprintSearchObject.setImprint_doc(homepage);
        (...)
    }
}
```

Falls kein valides Impressum gefunden wird, wird dies ebenfalls im "ImprintSearchObject" hinterlegt. Abschließend wird das "ImprintSearchObject" zurückgegeben.

6.8 Rechtsform

6.8.1 Zu implementierende Regeln

Adressat einer Adresse → suche indikatoren → finde Rechtsform

&
Durchsuche Text nach Rechtsform Indikatoren → finde Rechtsform

6.8.2 Implementierung

Die Funktion "findRechtsform()" der "FieldExtractor" Klasse ist für die Bearbeitung dieses Unterproblems zuständig und implementiert die zweite Regel. Zu Beginn wird ein Array an Strings initialisiert, welcher Indikatoren für die verschiedene Rechtsformen enthält.

```
String[] rechtsform_indicators =
    {"Körperschaft des öffentlichen Rechts",
    "Gesellschaft bürgerlichen Rechts",
    "Unternehmensgruppe", " ag",
    " aktiengesellschaft", " aör",
    " co.kg", " commerce", " e.," eigenbetrieb",
    ...};
```

Im nächsten Schritt wird über über "current_document.getAllElements()" ein Elements Objekt erstellt. Im Anschluss wird über diese Liste an Elementen iteriert und für jedes Element überprüft, ob innerhalb des eigenen Textes vom Element Indikatoren aufgefunden werden können. Die gefundenen Indikatoren werden dabei als ArrayList<String> zurückgegeben.

```
Elements elements = current_document.getAllElements();
for(Element e : elements) {
        ArrayList<String> found_rechtsform_indicators
        = check_return_matches(rechtsform_indicators, e.ownText());
```

Sollten nun innerhalb eines Elementes solche Indikatoren gefunden worden sein, so wird ein RechtsformObject Objekt erstellt und die Elemente des Dokumentes, das Element, in dem die Indikatoren gefunden wurden, und die Liste mit den Indikatoren an dieses übergeben und abschließend einer Liste von Rechtsformobjekten hinzugefügt, welche ganz zum Schluss zurückgegeben wird.

```
if(found_rechtsform_indicators.size() > 0 ) {
    RechtsformObject ro = new RechtsformObject();
    ro.setFound_rechtsform_indicators(found_rechtsform_indicators);
    ro.setElements(elements);
    ro.setElement(e);
    rechtsformObjects.add(ro);
}
```

Die erste Regel wird von der gleichnamigen Funktion der "ImprintExtractor" Klasse implementiert und nimmt sich die Ergebnisse der Adressextraktion zur Hilfe. Dafür wird der vermeintliche Adressat der Hauptadresse betrachtet, und es wird überprüft, ob Indikatoren für eine Rechtsform vorliegen und wenn ja, werden diese zurückgegeben.

6.9 Redaktionell gestaltete Angebote

6.9.1 Zu implementierende Regel

Redaktioneller Indikator → Impressum hat redaktionelle Angaben

6.9.2 Implementierung

Die Implementierung erfolgt hierfür innerhalb der Funktion "findRedaktionellerAngabe" innerhalb der "FieldExtractor" Klasse.

In diesem Fall wird über die Indikatoren iteriert und über einen CSS-Selektor werden Elemente, welche einen Indikator enthalten, extrahiert.

Sollten solche Elemente gefunden werden, so wird ein "RedaktionsObject" erstellt und der Indikator mit den entsprechenden Elementen hinzugefügt und in einer ArrayList von "RedaktionsObject" Objekten hinzugefügt, welche abschließend zurückgegeben wird.

```
String[] redaktions_indikatoren =
    {"Redaktion", "§ 10 Absatz 3 MDStV",
    "inhaltlich verantwortlich",
    ...};
    ...
ArrayList<RedaktionsObject> redaktionsObjects = new ArrayList<>();
for (String s : redaktions_indikatoren) {
    Elements elements_with_indicator =
        elements
            .select("*:containsOwn("+s+")");
    RedaktionsObject ro = new RedaktionsObject();
    ro.setElements(elements);
    ro.setIndikator(s);
    ro.setElements_with_indikator(elements_with_indicator);
    if(elements_with_indicator.size() > 0) {
        redaktionsObjects.add(ro);
    }
```

```
    }
    return redaktionsObjects;
```

6.10 Registerinformation

6.10.1 Zu implementierende Regel

Amtsgericht Indikator Match & RegisternummerMatch →
wir haben beide Informationen

Registernummerpattern match →
vor der Registernummer wird ein Amtsgericht genannt

6.10.2 Implementierung

Die Implementierung erfolgt komplett unabhängig von anderen Funktionen oder Teilen des Programms innerhalb der "findRegister_Registernummer()" Funktion. Zu Beginn werden drei Patterns kompiliert, einmal um das Register zu finden, einmal um das Amtsgericht oder Registergericht zu finden und einmal um eine Registernummer zu matchen.

```
//Beispielmatch: Handelsregister
"[a-z]*register[:]?[\\s]+"
//Beispielmatch: amtsgericht duisburg
"[\\s>]*[a-z]*gericht\\s[\\w]*[\\s]?"
//Beispielmatch: HRB 928
"\\s[hra\\sbvgnp]{2,4}\\s?[\\d]{3,5}"
```

Im Anschluss wird über alle Elemente des Dokumentes iteriert und die Matches auf den eigenen Text des Elementes angewandt. Für jedes Match wird ein Tripel erstellt, welches die Gruppe enthält, wann das Match endet und den Index des Elementes, in dem das Match gefunden wurde. Nun wird überprüft, ob für ein Registernummer-Match ebenfalls ein Amtsgericht-Match gefunden wurde. Ist dies der Fall, wird ein Registerobject erstellt, das Element, die Elemente und die beiden Matches gesetzt und einer ArrayList aus Registerobjekten hinzugefügt. Werden keine zwei Matches gefunden, so wird davon ausgegangen, dass das Gericht direkt vor der Nummer genannt wird.

Bei der Erstellung der Amtsgericht-Registernummer Kombination wird nun das Wort vor der Registernummer mit dem Wort 'Amtsgericht' konkateniert und als Registergericht in dem Registerobjekt hinterlegt,

```
ro.setRegister_court(
    "Amtsgericht " + element_own_text_as_list
```

```
                        .get(element_own_text_as_list.size()-1)
);
```

welches abschließend der RegisterObjekt ArrayList hinzugefügt wird, welche nach Beenden der Iteration über alle Elemente zurückgegeben wird.

6.11 Umsatzsteuer-Identifikationsnummer

6.11.1 Zu implementierende Regel

Matche Text eines Elementes mit Pattern → Match ist Ust.

6.11.2 Implementierung

Die Umsetzung erfolgt innerhalb der "findUst()" Funktion der "FieldExtractor" Klasse. Zu Beginn wird dabei der reguläre Ausdruck kompiliert und verschiedene Ust-Indikatoren initialisiert.

```
Pattern pattern_ust =
    Pattern
        .compile(
            "[\\s>][a-z][a-z](\\s?\\d){9}",
            Pattern.CASE_INSENSITIVE
        );
```

Die Anwesenheit oder Abwesenheit von Indikatoren hat dabei keinen Einfluss auf die Suche der Ust-Nummer, sondern wird als zusätzliche Information bei einem Ust_IdentificatioNumber Objekt mit hinzugefügt, sofern vorhanden. Die Funktion iteriert über alle Elemente eines Dokumentes. Innerhalb dieser Iteration wird das Element als String gespeichert mitsamt seiner Tags, und es wird nach Matches gesucht. Sollte ein Match gefunden werden, welches nicht bereits gefunden wurde, so werden ungewollte Symbole entfernt

```
ust_number = ust_number.replace(">", "");
```

und ein "Ust_IdentificationNumber" Objekt erstellt, dass einer Liste hinzugefügt wird, welche, nachdem über alle Elemente des Dokumentes iteriert wurde, zurückgegeben wird.

6.12 Verkammerte Berufe

6.12.1 Zu implementierende Regeln

Übergreifende Regel:

Segment ist im HashSet enthalten → wir haben unsere Kammer gefunden

Regeln für Segmente:

Kammer Indikator → füge x Wörter hitner Match hinzu →
Kammer gefunden | Kammer nicht gefunden

Kammer Indikator → füge x Wörter vor Match hinzu →
Kammer gefunden | Kammer nicht gefunden

Kammer Indikator →
füge x Wörter vorne und y Wörter hinter Match hinzu →
Kammer gefunden | Kammer nicht gefunden

6.12.2 Implementierung

Um diese Regeln zu realisieren, wurde innerhalb der Liste von Kammern nach den jeweiligen möglichen maximalen Fällen gesucht, um sinnvolle Grenzen zu setzen. Für die erste Regel ergibt sich für den Wert x=10, da die längste Kammer mit einem Kammer Indikator am Anfang aus elf Wörtern besteht, nämlich der "Kammer für Psychologische Psychotherapeuten und Kinder- und Jugendlichenpsychotherapeuten im Land Berlin", die Grenzen für Regel zwei und drei wurden analog kreiert.

Die Implementierung der Regeln findet innerhalb der "findVerkammerte-Berufe()()" Funktion der "FieldExtractor" Klasse statt.

Um diese Regeln zu realisieren, werden alle Elemente mit einem Match für eine Kammer in einer ArrayList gespeichert.

```
Matcher kammer_matcher = kammer_pattern.matcher(e.ownText());
while (kammer_matcher.find()) {
    String match = kammer_matcher.group().trim();
    matches.add(match);
}
```

Im Anschluss wird innerhalb des eigenen Textes des Elementes jedes Komma oder Doppelpunkt entfernt, anhand seiner Leerzeichen gesplittet und als Liste gespeichert. In einem nächsten Schritt werden die Indizes der Indikator Matches herausgefiltert und in einer weiteren Liste gespeichert.

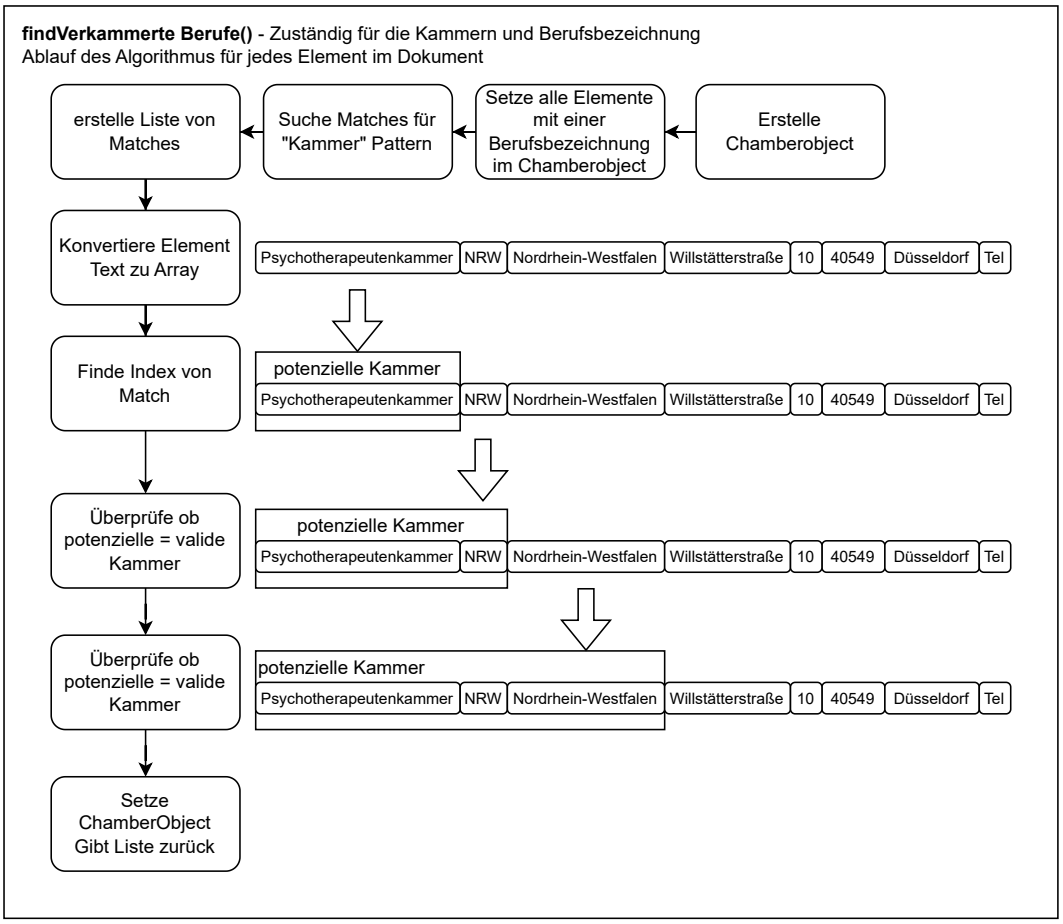

Abbildung 7: Beispielhafter Ablauf der Suche nach einer Kammer

Über die Liste der Indizes wird nun iteriert. Es wird der entsprechende Kammer-Indikator in einem String hinterlegt, und dann werden die einzelnen Fälle überprüft, indem das Wort entweder nach vorne, nach hinten oder in beide Richtungen erweitert wird. Jedes Mal, sobald ein Wort hinzugefügt wird, wird überprüft, ob es einer validen Kammer entspricht.

Sollte dies der Fall sein, wird es in einem "ChamberObject" Objekt hinterlegt. Nach Ablauf der Funktion wurde das gesamte Dokument auf vorkommende Kammern untersucht, es wird eine Liste von "ChamberObject" Objekten zurückgegeben, welche für die Weiterverarbeitung genutzt werden kann. Ein beispielhafter Ablauf ist in Abbildung 7 zu sehen.

6.13 Vertretungsberechtigte

6.13.1 Zu implementierende Regel

suche nach indikatoren $\rightarrow$
bei gefundenem Indikator suche nach Namen $\rightarrow$ Verantwortliche gefunden

6.13.2 Implementierung

Die Regeln wurden in der "findVertretungsberechtigten()" Funktion der "Field-Extractor" Klasse realisiert Für die Extraktion und Auffindung dieses Feldes wurde somit in einem ersten Schritt eine Liste an Indikatoren erstellt, die Hinweise auf solche Rollen gibt.

```
String[] vertreter =
    {"inhaber", "inhaberin", "inh.",
    "geschäftsführender", "geschäftsführung"
    ...};
```

Zusätzlich wurde eine andere Liste erstellt, die nach Wortlauten sucht, welche implizieren, dass Vertreter:innen aufgelistet werden. Stellen, an denen solche Indikatoren gefunden wurden, werden dabei im weiteren Verlauf berücksichtigt. Um nun zu filtern, welche Worte die korrespondierenden Namen sind, wurde ein, von der Extraktionskomponente unabhängiger, Webscraper geschrieben.

```
Names names = Names.getInstance();
```

Es gibt auf Wikipedia einen Artikel, welcher auf 26 verschiedene andere Artikel verlinkt, welche jeweils alle bekannten Vornamen zu einem Buchstaben des Alphabets auflisten. Der Webscraper extrahiert die Links dieser 26 Seiten und in einem weiteren Schritt extrahiert er alle Namen, welche auf dieser aufzufinden sind, und schreibt sie in eine Datei. Eine weitere Klasse "Names", welche mit dem Singleton Pattern realisiert wurde, liest diese Datei aus, erstellt ein Hashset und liefert eine Wrapper Funktion für die Funktion "ontains(String s)" des HashSets. Innerhalb der Namens Datei sind 10.000 Vornamen enthalten. Wurde innerhalb der "findVertretungsberechtigten()" Funktion nun ein Element mit Indikatoren gefunden,

```
if(check(vertreter, e_as_string)) { ...
```

so wird überprüft, ob innerhalb des Elementes mittels der "extractNames(Element e)" Funktion Namen aufgefunden werden können. Sollten im Element keine

Namen gefunden werden, wird das gleiche mit dem Parent Element durchgeführt, wobei dabei der String bis zur Stelle des Indikators gestutzt wird, um fälschlich unabhängigen Matches vorzubeugen. Innerhalb der "extractNames(Element e)" Funktion wird für die Extraktion der Namen ein Element von Sonderzeichen befreit und in seine Tokens zerlegt.

```
String[] e_splitted =
    e
        .toString()
        .replaceAll("[,.:<>]", " ")
        .split("\\s");
```

Über diese Array wird dann iteriert, und die Indezes von gefundenen Namen werden in einem Array gespeichert.

```
for(int i = 0; i< e_splitted.length; i++) {
    if(names.containsName(e_splitted[i])) {
        indezes.add(i);
        }
    }
```

Im Folgenden eine Veranschaulichung:

```
        {"Max", "Mustermann", "Max", "Mustermann"}
```

Der Index "0 " und Index "2 " würden dem Array hinzugefügt werden.

In einem weiteren Loop werden nun die Nachnamen extrahiert. Dafür wird davon ausgegangen, dass zwischen zwei Namen der Nachname steht und der Nachname des letzten Namens direkt hinter dem Namen steht.

Wurde dadurch ein Name gefunden, so werden die Informationen in einem "VertretungsberechtigterObject" Objekt hinerlegt und einer Liste von solchen hinzugefügt, welche zum Abschluss der Funktion zurückgegeben wird, wie in Abbildung 8 zu sehen.

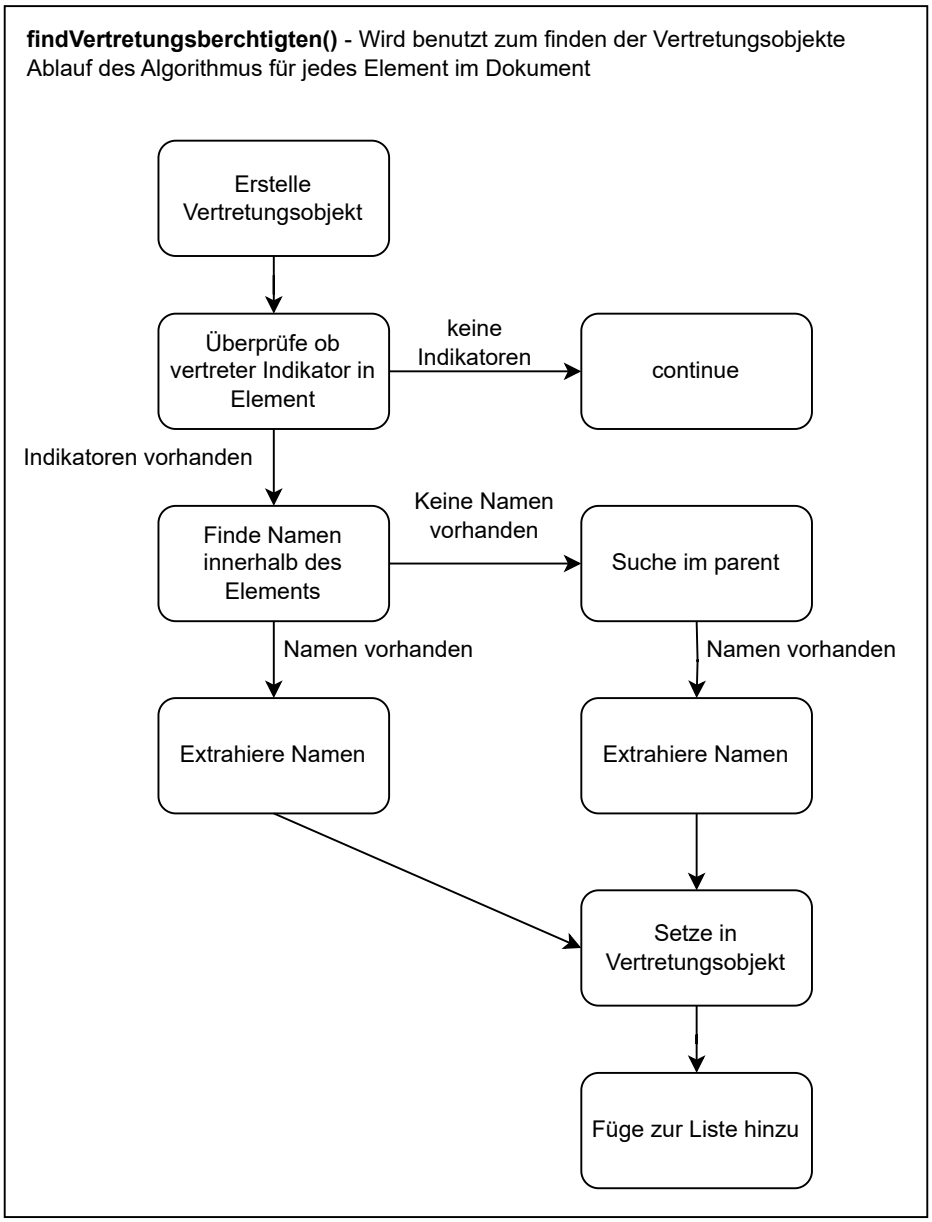

Abbildung 8: Ablauf des Algorithmus für die Extraktion von Vertretern

7 Technische Dokumentation der Feldobjekte

7.1 AdressObject

Ein AdressObject beinhaltet alle Informationen, die bezüglich einer Adressextraktion gefunden wurden. Eine Liste der Felder und Erklärungen ist der folgenden Tabelle zu entnehmen:

AdressObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Element *plz_element*	das Element, in dem die Postleitzahl enthalten ist
Element *street_element*	das Element, in dem die Straße enthalten ist
Element *adressed_element*	das Element, in dem die Postleitzahl enthalten ist
String *adressed*	Textinhalt des Adressatensegments
String *street*	Textinhalt des Straßen Segments
boolean *street_indication*	enthält Wahrheitswert ‘true‘, wenn der für ‘street‘ gesetzte Text einen Indikator für eine Straße enthält

boolean *firm_indication*	enthält Wahrheitswert ‘true‘, wenn der für ‘adressed‘ gesetzte Text einen Indikator für eine Firma enthält
boolean *is_valid_address*	enthält Wahrheitswert ‘true‘, wenn PLZ + Ort, Straße und adressed nicht ‘null‘ entsprechen
boolean *is_postal_address*	enthält Wahrheitswert ‘true‘, wenn innerhalb der Adresse ein Indikator, der auf ein Postfach hinweist, enthalten ist
String *street_indicator*	enthält den Straßen-Indikator
String *firm_indicator*	enthält den Firmen-Indikator
int *plz_element_index*	Index des Elementes, in dem die Postleitzahl enthalten ist
int *steet_element_index*	Index des Elementes, in dem die Straße enthalten ist
int *name_element_index*	Index des Elementes, in dem der Adressat enthalten ist
function *report_indizes()*	gibt Output über die vorhandenen Indizes der inhaltlichen Felder

function *report_adress_and_indezes()*	gibt Output über die vorhandenen Indizes der inhaltlichen Felder, sowie die inhaltlichen Felder
function *report_adress_and_indezes_indicators()*	gibt Output über die vorhandenen Indizes der inhaltlichen Felder, sowie der Wahrheitswerte über die Indikatoren, sowie die Indikatoren
function *report_adress*	gibt Output über die Adresse
String *report()*	gibt einen formatierten String der Adresse zurück

Tabelle 1: Variablen und Funktionen der AdressObject Klasse

7.2 Aufsichtsobject

Ein Aufsichtsobject beinhaltet Informationen über eine Aufsichtsbehörde. Innerhalb des FieldExtractors werden Elemente, die den Indikator Àufsichts-behördeènthalten, in dem Objekt hinterlegt. Innerhalb des ImprintExtractors wird die zugehörige, nahestehende Adresse dem Objekt hinzugefügt.

Aufsichtsobject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Elements *elements_with_structural_indicator*	Liste der Elemente, die das Wort ‘Aufsichtsbehörde‘ im eigenen Text besitzen
boolean *hat_aufsichts_indicator*	Wahrheitswert wird auf ṫrue ġesetzt, wenn strukturelle Elemente innerhalb des Dokumentes existieren
String *Aufsichtsbehörde*	Textform der Adresse der Aufsichtsbehörde, welche dem Indikator zugeordnet werden konnte

Tabelle 2: Variablen und Funktionen der AufsichtsObject Klasse

7.3 BerufsordnungsObject

Ein BerufsordnungsObject enthält Elemente, in denen Elemente mit Indikatoren auf Regelungen enthalten sind, und Elemente, in denen Regelungen genannt werden, und eine Liste, welche die entsprechenden Regelungen in Textform enthält.

BerufsordnungsObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Elements *elements_with_nennung*	Liste der Elemente, welche einen Indikator für eine berufsrechtliche Regelung enthalten
Elements *elements_with_regelung*	Liste der Elemente, welche eine berufsrechtliche Regelung enthalten
ArrayList<String> *elements_with_regelung*	Liste von Strings. Strings entsprechen dabei denen Textinhalt der Elemente, welche eine Regelung enthalten

Tabelle 3: Variablen und Funktionen der Berufsordnungs-Object Klasse

7.4 ChamberObject

Ein ChamberObject enthält Informationen, ob valide Kammern gefunden wurden, welche Indikatoren für die Kammern ausschalggebend waren und die Kammern, die in einem Element enthalten sind.

ChamberObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Element *element*	Element, in dem die Kammer oder Kammerindikatoren enthalten sind
boolean *valid_chamber_found*	Wahrheitswert entspricht wahr, wenn das ChamberObject eine valide Kammer besitzt
ArrayList<String> *chamber_indicatorss*	enthält die gefundenen Kammer Indikatoren
ArrayList<String> *valid_chambers*	enthält die validen Kammern, die innerhalb des Objektes gefunden wurden

Tabelle 4: Variablen und Funktionen der ChamberObject Klasse

7.5 ContactObject

Ein ContactObject enthält Information über eine Kontaktinformation

ContactObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Element *element*	Element, in dem die Kontaktinformation enthalten sind
String *contact_info*	enthält die gefundene Kontaktinformation

Tabelle 5: Variablen und Funktionen der ContactObject Klasse

7.6 EditorInfoObject

Ein EditorInfoObject enthält Informationen über eine inhaltlich verantwortliche Person. Enthält sowohl die Rolle als auch den Namen

EditorInfoObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Element *element*	Element in dem die Kontaktinformation enthalten sind
String *role*	enthält den String, welcher der Rolle entspricht
String *name*	enthält den String, welcher dem Namen entspricht

Tabelle 6: Variablen und Funktionen der EditorInfoObject Klasse

7.7 ImprintCrawlObject

beinhaltet Informationen bezüglich der Suche und Extraktion der Impressumsinformation.

ImprintCrawlObject	
Typ *Name*	*Beschreibung*
ImprintSearchObject *imprintSearchObject*	enthält Informationen über die Suche des Impressums
ImprintObject *imprintObject*	enthält die Information über die einzelnen Felder des Impressums und zusätzlich eine Bewertung, welche Adresse die Hauptadresse ist und welches Vertretungsobjekt dazugehört

Tabelle 7: Aufbau des ImprintCrawlObjects.

ImprintObject	
Typ *Name*	*Beschreibung*
void *report()*	gibt einen natürlichsprachlichen Bericht über den Inhalt eines Impressums
String *adress*	beinhaltet String der Hauptadresse
boolean *has_vertreter*	sst true, wenn ein Vertreter vorhanden ist
String *vertreter*	beinhaltet Name des Vertreters
String *rechtsform*	beinhaltet Rechtsform der Firma
String[] *elektronische_kontaktaufnahme*	beinhaltet Mails
String *aufsichtsbehoerde*	beinhaltet die gefundene Aufsichtbehörde
String *register_info*	beinhaltet die Registerinformationen
String[] *chambers*	beinhaltet die Kammer
String[] *berufsbezeichnung*	beinhaltet die Berufsbezeichnungen
String *berufsrechtliche_regelung*	beinhaltet die berufsrechtlichen Regelungen
String *ust_number*	beinhaltet die Umsatzsteueriden-tifikationsnummern
boolean *has_editorial_indicator*	beinhaltet Wahrheitswert über das Vorhandensein von redaktionellen Inhalten
String *editors*	beinhaltet die verschiedenen Redakteure

boolean *adress_contains_firm*	beinhaltet Information, ob die Hauptadesse einen Firmen Indikator enthält
boolean *has_ust_numbers*	enthält Information, ob innerhalb des Dokumentes eine Ust-Nummer enthalten ist
boolean *has_chamber_indicator*	enthält Information, ob das Dokument einen Indikator für eine Kammer besitzt
boolean *has_chambered_job_indicator*	besitzt Informationen, ob ein verkammerter Beruf genannt wird
boolean *has_register_info_indicator*	beinhaltet Informationen darüber, ob Informationen über Register gemacht werden.
boolean *contains_aufsichtsbehoerde_indicator*	enthält Informationen, ob eine Aufsichtsbehörde vorliegt oder nicht
boolean *has_email_address*	enthält Informationen, ob eine Mail Adresse vorliegt
boolean *has_contact_form*	enthält Informationen, ob ein Kontaktformular vorliegt
boolean *has_elektronische_kontaktaufnahme*	enthält Informationen, ob Möglichkeiten der elektronischen Kontaktaufnahme vorliegen
boolean *address_contains_name*	enthält Informationen darüber, ob innerhalb der Adresse ein Name genannt wird
boolean *multiple_addresses_no_main_address*	enthält Information, ob mehrere gleichwertige Informationen vorhanden sind

ArrayList<> *adressObjects*	Liste der verschiedenen AdressObjekte, welche im Rahmen der Adress-Extraktion gefunden wurden
ArrayList<> *vertretungsberechtigterObjects*	Liste der verschiedenen VertretungsberechtigterObjekte, welche im Rahmen der Vertretungsberechgtiten Extraktion gefunden wurden
ArrayList<> *mails*	eine Liste von Strings, jeder String steht für eine gefundene Mail
ArrayList<> *contactforms*	eine Liste von Strings, jeder String steht für den Link zu einem gefundenen Kontaktformular
ArrayList<> *numericContactObjects*	Liste der verschiedenen NumericContactObject Objekte, die innerhalb des Dokumentes gefunden wurden
AufsichtsObject *aufsichtsObject*	das Aufsichtsobjekt, das die verschiedenen Elemente mit Indikatoren für Aufsichtsbehörden enthält
ArrayList<> *redaktionsObjects*	Liste der verschiedenen RedaktionsObject Objekte, die innerhalb des Dokumentes gefunden wurden

ArrayList<> *editorInfoObjects*	Liste der verschiedenen EditorInfoObject Objekte, die innerhalb des Dokumentes gefunden wurden
ArrayList<> *ust_identificationNumbers*	Liste der verschiedenen Ust_IdentificationNumber Objekte, die innerhalb des Dokumentes gefunden wurden
ArrayList<> *chamberObjects*	Liste der verschiedenen ChamberObject Objekte, die innerhalb des Dokumentes gefunden wurden
BerufsordnungsObject *berufsordnungsObject*	das BerufsordnungsObject, das die verschiedenen Elemente mit Berufsordnungen enthält
ArrayList<> *registerObjects*	Liste der verschiedenen RegisterObject Objekte, die innerhalb des Dokumentes gefunden wurden
ArrayList<> *rechtsformObjects*	Liste der verschiedenen RechtsformObject Objekte, die innerhalb des Dokumentes gefunden wurden

Tabelle 8: Datenstruktur, in denen die vom fieldExtractor gefundenen Felder enthalten sind. Beziehungsweise Variablen und Funktionen der ImprintObject Klasse

7.8 ImprintSearchObject

ImprintSearchObject	
Typ *Name*	*Beschreibung*
String *homepage_url*	die url, auf der die Suche gestartet ist
String *imprint_link*	der Link, hinter dem das Impressum vermutet wird
Document *imprint_doc*	das Jsoup.Document des vermuteten Impressums
ImprintLocation *imprintLocation*	Enum Wert, gibt an, an welchem Ort das Impressum vermutet wird

Tabelle 9: Variablen und Funktionen der ImprintSearch-Object Klasse

7.9 NumericContactObject

Ein NumericContactObject enthält Information über eine numerische Kontaktinformation

NumericContactObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Element *element*	Element, in dem die Kontaktinformation enthalten sind
String *contact_info*	enthält die gefundene Kontaktinformation
String *contact_type*	enthält den Typ der Kontaktinformation, meist Telefon oder Fax.

Tabelle 10: Variablen und Funktionen der NumericContactObject Klasse

7.10 RechtsformObject

Ein RechtsformObject enthält Informationen über alle vorkommenden Rechtsformindikatoren eines Elementes

RechtsformObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes.
Element *element*	Element, in dem die Kontaktinformation enthalten sind
ArrayList<String> *found_rechtsform_indicators*	enthält die gefundenen Rechtsformen des Dokumentes

Tabelle 11: Variablen und Funktionen der RechtsformObject Klasse

7.11 RedaktionsObject

Ein RedaktionsObject enthält Informationen über Elemente mit einem Indikator.

RedaktionsObject	
Typ *Name*	Beschreibung
Elements *elements*	Liste der Elemente eines Dokumentes
Element *element*	Element, in dem die Kontaktinformation enthalten sind
String *indikator*	gefundener Indikator
Elements *elements_with_indikaor*	enthält Elemente eines Dokumentes, in dem die Indikatoren enthalten sind

Tabelle 12: Variablen und Funktionen der RedaktionsObject Klasse

7.12 RegisterObject

Ein RegisterObject enthält Informationen über eine Registernummer und ein Amtsgericht..

RegisterObject	
Typ *Name*	Beschreibung
Element *register_number_element*	Element, in dem die Registernummer Information enthalten sind
String *register_number*	String der gefundenen Registernummer
int *register_number_element_index*	enthält den Index des Registernummer-Elements
Element *register_court_element*	Element, in dem die Registergericht Informationen enthalten sind
String *register_court*	String des gefundenen Registergerichts
int *register_court_element_index*	enthält den Index des Registergerichts-Elements

Tabelle 13: Variablen und Funktionen der RegisterObject Klasse

7.13 Ust_IdentificationNumber

Ein Ust_IdentificationNumber Objekt enthält Informationen bezüglich einer Umsatzsteueridentifikationsnummer, welche innerhalb eines Dokumentes erscheint.

Ust_IdentificationNumber	
Typ *Name*	Beschreibung
Element *number_element*	Element in dem die Ust-Nummer Information enthalten ist.
boolean *indicator_in_same_element*	Element, in dem die Ust-Nummer Information enthalten ist
int *element_index*	enthält Index des Elementes.
String *ust_number*	enthält Ust-Nummer als String

Tabelle 14: Variablen und Funktionen der Ust*IdentificationNumberKlasse*

7.14 VertretungsberechtigterObject

Ein VertretungsberechtigterObject Objekt enthält Informationen bezüglich einer Vertretungsberechtigten Person. Es kann nur den Indikator enthalten, oder dazugehörige Personen.

VertretungsberechtigterObject	
Typ *Name*	Beschreibung
Elements *elements*	enthält Elemente des Dokuments
Element *role_element*	das Element, in dem die Rolle des Vertretungsberechtigten enthalten ist
int *role_element_index*	enthält Index des Elementes, in dem die Rolle enthalten ist
Element *name_element*	das Element, in dem der/die Namen enthalten sind
int *name_element_index*	enthält Index des Elementes, in dem der/die Namen enthalten sind
ArrayList<String> *vertretungs_indikatoren*	Liste von Rollen Indikatoren
ArrayList<String> *found_names*	Liste von gefundenen zugehörigen Namen

Tabelle 15: Variablen und Funktionen der VertretungsberechtigterObject Klasse