Fernuniversität Hagen, Informatik

Ein objektorientierter Werkzeugkasten für die juristische Falllösung

Einführung für Anwender

Till Menke
Kosselstr. 4 – 69115 Heidelberg – Tel. 06221/3538228 – studium@tillmenke.de

Stand: 24.09.2023
http://www.tillmenke.de/studium/informatik/masterarbeit

Inhaltsverzeichnis

1 Einleitung

Dieses Handbuch dient dazu, den im Rahmen der gleichnamigen Masterarbeit entwickelten Werkzeugkasten neuen Anwendern nahezubringen.

Hierzu werden zunächst in den ABSCHNITTEN 2 UND 3 die Grundlagen für absolute Einsteiger in der gebotenen Ausführlichkeit erklärt. In den ABSCHNITTEN 4 UND 5 werden dann in gestraffter Form grundlegende Funktionen erklärt, die erforderlich sind, sobald eine teilautomatische Fallprüfung implementiert werden soll. In den ABSCHNITTEN 6 BIS 8 finden sich schließlich kurze Hinweise, die erfahreneren Anwendern Anregungen für die Umsetzung anspruchsvollerer Projekte liefern sollen.

2 Vorbereitung der Entwicklungsumgebung

Um den Werkzeugkasten verwenden zu können, muss zunächst die Entwicklungsumgebung entsprechend konfiguriert werden. Diese Anleitung basiert auf der Entwicklungsumgebung eclipse; erfahrene Benutzer können jedoch auch jede andere Java-Entwicklungsumgebung entsprechend verwenden.

Die .jar-Datei mit dem Werkzeugkasten muss über Rechtsklick auf den Projekttitel → Properties → Java Build Path → Libraries → Classpath → Add External JARs eingelesen werden.

3 Umsetzung eines einfachen Prüfungsschemas

Der Werkzeugkasten ist dafür optimiert, juristische Prüfungsschemata einfach in eine Programmstruktur überführen zu können.

In diesem Abschnitt soll dargestellt werden, wie das folgende einfache, abstrakte Prüfungsschema in die Programmstruktur übersetzt werden kann:

A. Prüfungspunkt A
    1. Unterprüfungspunkt 1
    2. Unterprüfungspunkt 2
    3. Unterprüfungspunkt 3
B. Prüfungspunkt B

Hierbei soll in dieser Anleitung von der niedrigsten zur höchsten Ebene des Prüfungsschemas vorgegangen werden. Dies ist darin begründet, dass die Prüfung des Prüfungspunktes A erfordert, dass die Unterprüfungspunkte 1-3 bereits umgesetzt sind.

3.1 Definition eines einfachen Prüfungspunktes

In unserem Beispiel werden die Unterprüfungspunkte 1-3 nicht weiter verschachtelt, müssen also selbst ein Prüfungsergebnis erzeugen. Hierzu muss eine Klasse – die wir hier „`Unterprüfungspunkt1`“ nennen wollen – angelegt werden, welche die Schnittstelle „Tatbestand“ implementiert.

In Eclipse kann die Klasse über Rechtsklick auf „src“ → New → Class angelegt werden. Dabei muss darauf geachtet werden, dass unter „Interfaces“ die Schnittstelle „Tatbestand“ ausgewählt wurde.

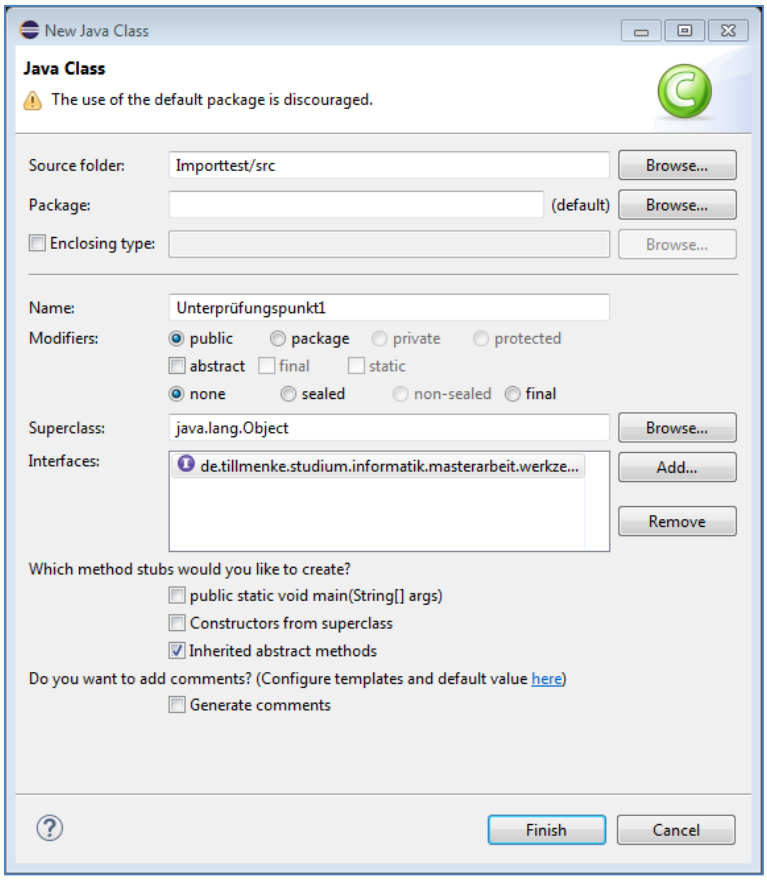

Abbildung 1: Dialog zum Anlegen der Klasse „Unterprüfungspunkt1“

Wir erhalten folgende Struktur:

```
3  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.EingabeFehltException;
4  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.Rechtsfolge;
5  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.Sachverhalt;
6  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.Sprachstil;
7  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.Tatbestand;
8
9  public class Unterprüfungspunkt1 implements Tatbestand {
10
11   @Override
12   public String getDefinition() throws EingabeFehltException {
13     // TODO Auto-generated method stub
14     return null;
15   }
16
17   @Override
18   public String getErgebnissatz() throws EingabeFehltException {
19     // TODO Auto-generated method stub
20     return null;
21   }
22
23   @Override
24   public String getObersatz() throws EingabeFehltException {
25     // TODO Auto-generated method stub
26     return null;
27   }
28
29   @Override
30   public Rechtsfolge getRechtsfolge() throws EingabeFehltException {
31     // TODO Auto-generated method stub
32     return null;
33   }
34
35   @Override
36   public String getSubsumption(Sprachstil arg0) throws EingabeFehltException {
37     // TODO Auto-generated method stub
38     return null;
39   }
40
41   @Override
42   public void setSachverhalt(Sachverhalt arg0) {
43     // TODO Auto-generated method stub
44
45   }
46
47 }
48
```

Abbildung 2: automatisch generierte Struktur für „Unterprüfungspunkt1“ in Eclipse

Die Methoden „`getObersatz`“, „`getDefinition`“, „`getSubsumtion`“ und „`getErgebnissatz`“ müssen jeweils eine Zeichenkette zurückgeben, welche den textlichen Inhalt der entsprechenden Ebene des juristischen Gutachtens enthält. Wie diese Zeichenkette ermittelt wird, ist Ihnen überlassen. Im einfachsten Fall kann die Zeichenkette einfach fest eingegeben werden, was sich praktisch vor allem für Obersatz und Definition anbieten wird.

Eine Neuerung im Vergleich zum klassischen Gutachten stellt hingegen die Methode „`getRechtsfolge`“ dar. Diese gibt ein Objekt zurück, welches die Rechtsfolge programmtechnisch abbildet. Geht es also bei der Prüfung etwa nur um (+) oder (-), so wird ein Wahrheitswert der Aufzählungsklasse „`RechtsfolgeWahrheitswert`“ zurückgegeben. Daneben existieren auch andere Klassen, etwa für die Höhe einer Geldleistung. Eine vollständige Auflistung der vorgefertigten Rechtsfolgeklassen bietet ABSCHNITT 6.1. Daneben können auch eigene Rechtsfolgen definiert werden, wie an der genannten Stelle dargestellt wird. An dieser Stelle wollen wir uns aber zunächst auf den Wahrheitswert als Rechtsfolge beschränken.

Die Methode „`setSachverhalt`“ ist für die Arbeit mit dem Sachverhalt bestimmt. Da wir hier noch nicht mit dem Sachverhalt arbeiten, kann sie hier noch leer bleiben.

Mit diesen Kenntnissen können wir nun das Programmgerüst mit Inhalt füllen. Wichtig ist, dass kein „`null`“-Wert zurückgegeben wird. Dies kann zum Absturz des Programmes führen. Soll eine Ebene leer bleiben, muss eine leere Zeichenkette („`""`“) zurückgegeben werden.

3.1.1 vollständig programmatische Prüfung

Nehmen wir an, Unterprüfungspunkt 1 ist erfüllt, wenn am Tag der Prüfung Montag ist. Dies dürfte zwar praktisch wenig Sinn ergeben, eignet sich aber hier gut als Beispiel, weil es eine vollständig programmtechnische Prüfung ohne Rückgriff auf eine Sachverhaltseingabe erlaubt.

Wir müssen also als Obersatz einprogrammieren: „Dafür müsste heute Montag sein.“. Als Definition verwenden wir „Dies ist der Fall, wenn der Kalender für heute einen Montag ausweist.“. Als Ergebnissatz muss abhängig von dem Prüfungsergebnis „Heute ist also Montag.“ oder „Heute ist also nicht Montag.“ zurückgegeben werden. Als Subsumtion soll der Satz „Heute ist der {Datum}, also ein {Name des Wochentags}.“ verwendet werden. Als Rechtsfolgeobjekt soll ein Wahrheitswert zurückgegeben werden, der angibt, ob am Tag der Prüfung Montag ist. Um diese Prüfung nicht doppelt implementieren zu müssen, soll das Prüfungsergebnis auf die Methode „`getRechtsfolge`“ zurückgreifen – dies dürfte sich in den meisten Fällen anbieten und ist auch der Weg, wie der Satz in den folgenden Abschnitten ermittelt werden muss.

Mit diesen Angaben erhalten wir folgenden Quelltext:

```java
import java.text.DateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;

import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.*;

public class Unterprüfungspunkt1 implements Tatbestand {

	private Calendar heute = new GregorianCalendar();

	@Override
	public String getDefinition() throws EingabeFehltException {
		return "Dies ist der Fall, wenn der Kalender für heute einen Montag ausweist.";
	}

	@Override
	public String getErgebnissatz() throws EingabeFehltException {
		if(getRechtsfolge()==RechtsfolgeWahrheitswert.Wahr) {
			return "Heute ist also Montag.";
		}
		else {
			return "Heute ist also nicht Montag.";
		}
	}

	@Override
	public String getObersatz() throws EingabeFehltException {
		return "Dafür müsste heute Montag sein.";
	}

	@Override
	public Rechtsfolge getRechtsfolge() throws EingabeFehltException {
		if (heute.get(Calendar.DAY_OF_WEEK) == Calendar.MONDAY) {
			return RechtsfolgeWahrheitswert.Wahr;
		} else {
			return RechtsfolgeWahrheitswert.Falsch;
		}
	}

	@Override
	public String getSubsumption(Sprachstil arg0) throws EingabeFehltException {
		DateFormat datum = DateFormat.getDateInstance(DateFormat.MEDIUM);
		String wochentag = "unbekannter Wochentag";
		switch (heute.get(Calendar.DAY_OF_WEEK)) {
			case Calendar.MONDAY:
				wochentag = "Montag";
				break;
			case Calendar.TUESDAY:
				wochentag = "Dienstag";
				break;
			case Calendar.WEDNESDAY:
				wochentag = "Mittwoch";
				break;
			case Calendar.THURSDAY:
				wochentag = "Donnerstag";
				break;
			case Calendar.FRIDAY:
				wochentag = "Freitag";
				break;
			case Calendar.SATURDAY:
				wochentag = "Samstag";
				break;
			case Calendar.SUNDAY:
				wochentag = "Sonntag";
				break;
		}
		return "Heute ist der " + datum.format(heute.getTime()).toString() + ", also ein "+wochentag+".";
	}

	@Override
	public void setSachverhalt(Sachverhalt arg0) {
	}
}
```

Abbildung 3: Quelltext der Klasse „Unterprüfungspunkt1“

Damit liegt eine vollständige Implementierung unseres Unterprüfungspunkts 1 vor.

3.1.2 manuelle Prüfung

Für Unterprüfungspunkt 2 wollen wir die einfachstmögliche Implementierung einer Prüfung umsetzen: Der Nutzer soll auf Basis eines Obersatzes und einer Definition selbst um die Subsumtion gebeten werden.

Hierzu muss eine Klasse – die wir hier „`Unterprüfungspunkt2`“ nennen wollen – angelegt werden, welche von der Klasse „`TatbestandManuell`“ implementiert. Es ist auch möglich, dem Nutzer bei der Abfrage bereits einen Subsumtions- und Ergebnisvorschlag zu unterbreiten. Hierzu muss statt von „`TatbestandManuell`“ von „`TatbestandManuellVorlage`“ geerbt werden.

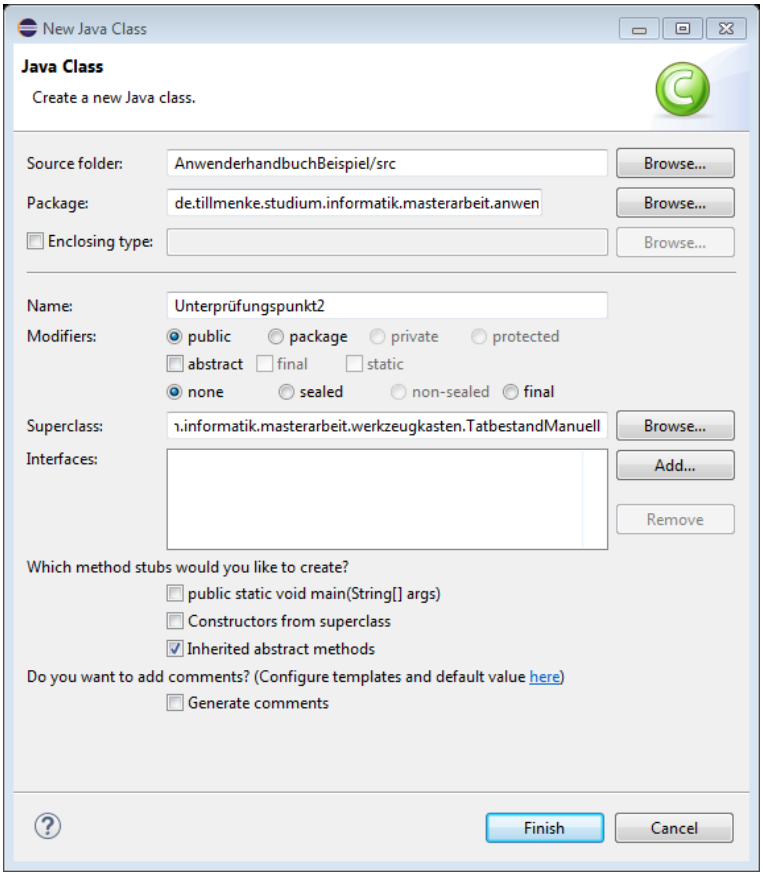

Abbildung 4: Dialog zum Anlegen der Klasse „`Unterprüfungspunkt2`“

Da hier mit der Subsumption ein Teil durch das Rahmenwerk unterstützt auf den Anwender verlagert wird, müssen weniger Methoden implementiert werden. Die Methoden „`getObersatz`“ und „`getDefinition`“ funktionieren wie oben. Die Methode „`getErgebnissatz`“ funktioniert im Grundsatz auch wie oben, es muss aber darauf geachtet werden, dass das Ergebnis durch den Nutzer bestimmt wird. Dieses kann durch die vordefinierte Methode „`getRechtsfolge`“ abgerufen werden. Um die Form des Ergebnisses der Nutzerabfrage zu bestimmen, wird in der Methode „`getErgebnisClass`“ eine Referenz auf die Klasse des Rechtsfolgeobjektes übergeben, also z. B. „`return RechtsfolgeWahrheitswert.class;`“

3.1.3 Verwendung von ChatGPT

Unterprüfungspunkt 3 soll anstelle des Benutzers von ChatGPT bearbeitet werden. Dies funktioniert programmiertechnisch wie die manuelle Prüfung, es muss lediglich von „`TatbestandNeuronalesNetz`“ geerbt werden. In dieser Implementierung ist bereits vorgegeben,

dass die Antwort vom Typ „`RechtsfolgeWahrheitswert`“ ist. Statt der Typangabe muss mit der Methode „`getAPIFrage`“ angegeben werden, welche Anfrage an ChatGPT übermittelt werden soll.

Da ChatGPT dem Autor nicht hinreichend einsatztauglich erscheint, wird das Ergebnis in der derzeitigen Implementierung noch dem Nutzer zur Prüfung vorgelegt (wie bei „`TatbestandManuellVorlage`“). Es ist Ihnen freilich unbenommen, eigene Zugänge zu neuronalen Netzen (oder anderen bereits bestehenden Lösungsansätzen zur Verarbeitung der Falllösung) auszuprogrammieren, indem die Schnittstelle „`Tatbestand`“ entsprechend implementiert wird.

3.2 Definition einer Prüfungsfolge

In unserem Beispiel wird der Prüfungspunkt A dadurch geprüft, dass die Unterprüfungspunkte 1-3 jeweils geprüft werden. Diese Funktionalität ist in den Subklassen der Klasse „`TatbestandPrüfungsfolge`“, namentlich „`TatbestandPrüfungsfolgeUnd`“ und „`TatbestandPrüfungsfolgeOder`“ implementiert. Diese unterscheiden sich danach, ob für das Vorliegen des Tatbestands alle („und“) oder nur eines der Merkmale („oder“) vorliegen müssen.

Um diese zu verwenden, muss eine neue Klasse – die wir hier „`PrüfungspunktA`“ nennen wollen – angelegt werden, welche von der Klasse „`TatbestandPrüfungsfolgeUnd`“ erbt. Lassen wir eclipse hier wieder die notwendigen Methoden anlegen, erhalten wir folgende Struktur:

```
3  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.EingabeFehltException;
4  import de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.Rechtsfolge;
5
6  public class PrüfungspunktA extends de.tillmenke.studium.informatik.masterarbeit.werkzeugkasten.TatbestandPrüfungsfolgeUnd {
7
8    @Override
9    public String getDefinition() throws EingabeFehltException {
10     // TODO Auto-generated method stub
11     return null;
12   }
13
14   @Override
15   public String getObersatz() throws EingabeFehltException {
16     // TODO Auto-generated method stub
17     return null;
18   }
19
20   @Override
21   protected String getErgebnissatzNegativ() throws EingabeFehltException {
22     // TODO Auto-generated method stub
23     return null;
24   }
25
26   @Override
27   protected String getErgebnissatzPositiv() throws EingabeFehltException {
28     // TODO Auto-generated method stub
29     return null;
30   }
31
32   @Override
33   protected Rechtsfolge getRechtsfolgeNegativ() {
34     // TODO Auto-generated method stub
35     return null;
36   }
37
38   @Override
39   protected Rechtsfolge getRechtsfolgePositiv() {
40     // TODO Auto-generated method stub
41     return null;
42   }
43
44   @Override
45   protected void prüfungsfolgeFestlegen() {
46     // TODO Auto-generated method stub
47
48   }
49
50 }
51
```

Abbildung 5: automatisch generierte Struktur für „PrüfungspunktA“ in Eclipse

Die Methoden „getDefinition“ und „getObersatz“ funktionieren wie mittlerweile bereits bekannt. Da die logische Operation der Prüfungsfolge („und“ oder „oder“) nur das Ergebnis „positiv“ oder „negativ“ kennt, wurden die von oben bekannten Methoden „`getErgebnissatz`“ und „`getRechtsfolge`“ durch eine Methode für jedes mögliche Ergebnis „ersetzt“. Diese erlauben der Programmstruktur, das zurückzugebende Ergebnis anhand des Ergebnisses der Prüfungsfolge auszu-

wählen. Da naheliegt, als Rechtsfolge „RechtsfolgeWahrheitswert.Wahr“ und „RechtsfolgeWahrheitswert.Falsch“ zu verwenden, existieren mit „TatbestandPrüfungsfolgeUndWahrheitswert“ und „TatbestandPrüfungsfolgeOderWahrheitswert“ vorgefertigte Klassen, welche die Werte für „getRechtsfolge“ vorbelegen, sodass diese nicht immer neu ausprogrammiert werden müssen. Sofern keine besondere Funktion verwendet werden soll, empfiehlt es sich, diese zu verwenden.

Die eigentliche Konfiguration der Klasse erfolgt in der Methode „prüfungsfolgeFestlegen“. Dort muss für jeden Unterprüfungspunkt durch Aufruf der Methode „addPrüfungspunkt“ ein Objektverweis eingefügt werden. Diese Methode erfordert als erstes Argument einen Verweis auf das Tatbestandsobjekt. Als zweites Argument kann vorgegeben werden, bei welcher Rechtsfolge der Prüfungspunkt als erfüllt angesehen wird (Standardwert: „RechtsfolgeWahrheitswert.Wahr“). Als drittes Argument kann vorgegeben werden, ob der Prüfungspunkt nur geprüft werden soll, wenn die vorherigen Prüfungspunkte bereits erfüllt sind (Standardwert: nein). Für unsere Unterprüfungspunkte 1-3 muss also – wenn alle Argumente vorgegeben werden sollen – festgelegt werden:

```
addPrüfungspunkt(new Unterprüfungspunkt1(),RechtsfolgeWahrheitswert.Wahr,false);
addPrüfungspunkt(new Unterprüfungspunkt2(),RechtsfolgeWahrheitswert.Wahr,false);
addPrüfungspunkt(new Unterprüfungspunkt3(),RechtsfolgeWahrheitswert.Wahr,false);
```

Abbildung 6: Quelltext der Methode „prüfungsfolgeFestlegen“ der Klasse „PrüfungspunktA“

Analog kann eine Klasse „Prüfungsfolge“ implementiert werden, welche die Prüfungspunkte A und B enthält.

> Eine Funktion für fortgeschrittene Anwender bietet die Methode „setRechtsfolgenVergleicher()“ der Klasse „Prüfungsfolge“: Über diese kann ein Objekt mit der Schnittstelle „RechtsfolgenVergleicher“ übergeben werden. Dessen Methode wird dann anstelle des integrierten Vergleichs der Rechtsfolgen aufgerufen. So ist ein erfolgreicher Vergleich auch für Rechtsfolgen möglich, deren „equals“-Methode (auf welcher sonst der Vergleich beruht) aus den unterschiedlichsten Gründen nicht überschrieben werden kann. Ein Anwendungsbeispiel hierfür sind Aufzählungen, bei denen verschiedene Werte für die Erfüllung des Tatbestands ausreichen.

3.3 testweiser Aufruf

Diese Prüfungsfolge kann nun in Funktion gesetzt werden. Dafür muss die Klasse instanziiert werden und anschließend die Methoden „getUrteil“ und „getGutachten“ der obersten Ebene, hier also „Prüfungsfolge“ aufgerufen werden. Alternativ können Teile des Gutachtens mit den oben definierten Methoden wie „getObersatz“ aufgerufen werden.

Der Abruf für die oben definierte Prüfungsfolge wird jedoch zu einem Fehler führen, da die Subsumption ja noch nicht festgelegt wurde. Dieser Fehler muss abgefangen werden. Im Folgenden werden daher zwei mögliche Herangehensweise hierfür aufgezeigt.

3.3.1 Bibliotheksklasse

Zunächst existiert die Bibliotheksklasse „Falllösungsassistent“, welche ein AWT-Panel bereitstellt, welches die erforderlichen Abfragen übernimmt, soweit nur die im Werkzeugkasten vorgesehenen Rückfragen verwendet werden, und nach Abschluss die Falllösung grafisch aufbereitet darstellt. Ihr Konstruktor erfordert als ersten Parameter ein Objekt der Tatbestandsklasse und als zweiten Parameter die Angabe, ob nur die jeweils nächste oder alle vorhersehbaren Rückfragen auf einmal angezeigt werden sollen.

Um das Panel in ein Fenster, welches ggf. auch Scrollbalken enthält, zu übernehmen, kann die Bibliotheksklasse „`FalllösungsgrafikFenster`“ verwendet werden, welcher im Konstruktor schlicht das Objekt der Klasse „`Falllösungsassistent`“ übergeben werden muss.

Insgesamt ist der Aufruf daher denkbar einfach:

```
	public static void main(String[] args) {
		Tatbestand meintest = new Prüfungsfolge();
		Falllösungsassistent panel = new Falllösungsassistent(meintest,true);
		new FalllösungsgrafikFenster("Prüfungsfolge Demo",panel);
	}
```

Abbildung 7: Quelltext einer einfachen grafischen Anwendung unter Verwendung der Bibliothek zum Aufruf der Struktur

3.3.2 einfache Konsolenanwendung

Alternativ ist es möglich, die Fehlerbehandlung selbst zu übernehmen. Für den bisherigen Stand des Programms reicht hier die „`SubsumptionFehltException`“. Eine einfache Konsolenanwendung, welche dies übernimmt, könnte wie folgt aussehen:

```
	public static void main(String[] args) {
		Tatbestand meintest = new Prüfungsfolge();
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		boolean done = false;
		while (!done) {
			try {
				System.out.println(meintest.getGutachten());
				done=true;
			} catch (SubsumptionFehltException e) {
				System.out.println("Subsumption fehlt");
				System.out.println("================");
				System.out.println("Obersatz: "+e.getObersatz());
				System.out.println("Definition: "+e.getDefinition());
				System.out.println("Subsumption eingeben:");
				String s="";
				try {
					s = br.readLine();
				} catch (IOException e1) {
					System.err.println("keine Eingabe");
				}
				System.out.println("Ergebnis eingeben (true/false):");
				Boolean b = false;
				try {
					b = Boolean.parseBoolean(br.readLine());
				} catch (IOException e1) {
					System.err.println("keine Eingabe");
				}
				e.setResult(s, b ? RechtsfolgeWahrheitswert.Wahr :
	RechtsfolgeWahrheitswert.Falsch);
				System.out.println("----------------");
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
```

Abbildung 8: Quelltext einer einfachen Konsolenaufwendung zum Aufruf der Struktur

4 Arbeit mit dem Sachverhalt

Wesentlich für programmgestützte Falllösungen ist der Zugriff auf einen Sachverhalt. Der Werkzeugkasten bietet hierfür die Möglichkeit, einen Sachverhalt in die Tatbestandsstruktur zu injizieren.

Der dafür verwendete Sachverhalt basiert auf einer typsicheren, schlüsselbasierten Datenbank. Diese kann Subtypen des Typs „`Sachverhaltselement`“ speichern. Die Klasse „`Sachverhaltsdetail`“ implementiert dessen Schnittstelle und kann als parametrisierte Klasse beliebige Objekte des Parametertyps speichern. Dies dürfte in den meisten Fällen ausreichen, sodass keine eigene Implementierung erarbeitet werden muss. Allerdings muss eine Klasse angelegt werden, da die Typprüfung keine Parameter überprüfen kann.

4.1 Sachverhalt anlegen

Um den Sachverhalt anzulegen muss entweder eine eigene Implementierung der Schnittstelle „`Sachverhalt`“ verwendet werden oder die auf der in Java implementierten HashMap basierende Implementierung „`SachverhaltHashMap`“ verwendet werden. Diese muss instanziiert und mittels der Methode „`setSachverhalt`“ an das Tatbestandsobjekt übergeben werden. Dies muss spätestens vor Aufruf der Fallprüfung geschehen.

```
Sachverhalt testsachverhalt = new SachverhaltHashMap();
testsachverhalt.addElement("Beschuldigter", new Personendaten("T"));
meintatbestand.setSachverhalt(testsachverhalt);
```

Abbildung 9: Quelltext zur Injektion eines Sachverhalts in ein Tatbestandsobjekt

> Sollte die „`setSachverhalt`“-Methode einer Klasse des Typs „`TatbestandPrüfungsfolge`“ überschrieben werden, muss darauf geachtet werden, dass die Prüfungsfolgenelemente ebenfalls den Sachverhalt injiziert bekommen. Die einfachste Möglichkeit ist das Aufrufen der Originalmethode mittels „`super.setSachverhalt`“.

4.2 Sachverhalt speichern

Der so übergebene Sachverhalt muss – wenn er aufgerufen werden soll – in der individuellen Klasse gespeichert werden, indem der Parameter in einer Instanzvariable gespeichert wird. Die im Werkzeugkasten enthaltenen Prüfungsfolgeimplementierungen leiten den Sachverhalt an enthaltende Prüfungsfolgenelemente weiter.

> Hier gilt das oben gesagte: Soweit eine eigene Verschachtelung implementiert wird, muss darauf geachtet werden, den Sachverhalt notwendigenfalls durchzureichen.

4.3 Sachverhalt abfragen

Der Sachverhalt kann über seine Methode „`getElement`“ abgefragt werden. Hierzu muss als erster Parameter der gewählte Schlüssel und als zweiter Parameter eine Referenz auf den Typ des gewünschten Elements übergeben werden. Der Rückgabewert wird schon auf Compilerebene entsprechend festgelegt. Als dritter Parameter kann ein Objekt der Schnittstelle „`Invalidator`“ übergeben werden. Die Datenbank ruft dessen Methode „`invalidate`“ auf, wenn sich die hinterlegten Daten geändert haben. Das Tatbestandsobjekt sollte dann prüfen, ob die in ihm gespeicherten Werte noch korrekt sind. Speichert das Tatbestandsobjekt keinen Zustand, ist eine Invalidierung nicht erforderlich. In diesem Fall kann „`null`“ übergeben werden. Ansonsten empfiehlt sich eine Implementierung in einer anonymen Klasse, um einfacher auf die Instanzvariablen zugreifen zu können.

```
täter = sachverhalt.getElement("Beschuldigter", Personendaten.class, null);
```

Abbildung 10: Quelltext einer Variablendeklaration durch Abfrage des Sachverhalts

5 Rückfragen an den Benutzer stellen

Rückfragen an den Benutzer werden im Werkzeugkasten dargestellt, indem eine Programmflussausnahme des Typs „`EingabeFehltException`“ ausgelöst wird. Dies ist semantisch so zu deuten, dass das Programm nicht erfolgreich abgeschlossen werden kann, da ja eine notwendige Eingabe fehlt.

Im Werkzeugkasten aktiv Verwendung finden die „`MeinungstreitentscheidFehltException`“ und die „`SubsumptionFehltException`“ sowie die „`SachverhaltsdetailFehltException`“. Daneben unterstützen die AWT-Bibliotheksklassen auch die „`StrategiewahlFehltException`“. Um eine Klassifikation der Ausnahme zu erleichtern, teilen sich die beiden erstgenannten sowie die letztgenannte den Typ „`WertungFehltException`“.

Wenn die Rahmenwerksklassen des Typs „`Tatbestand`“ Verwendung finden, dürfte es nicht erforderlich sein, eine der vorgefertigten Ausnahmen selbst auszulösen, da für alle typischen juristischen Prüfungsvarianten eine entsprechende Basisklasse im Werkzeugkasten enthalten ist. Sie sollten aber berücksichtigt werden, sofern nicht die AWT-Bibliotheksklasse zur Ausnahmebehandlung verwendet wird, sondern eine individuelle Schnittstelle programmiert wird.

Sollten eigene Nutzerabfragen implementiert werden, so können diese von der passenden Klasse abgeleitet werden, damit sie im Rahmenwerk verwendet werden können. Sie können jedoch über die AWT-Bibliotheksklassen nicht aufbereitet werden.

6 erweiterte Funktionen

6.1 Rechtsfolge

Die Schnittstellenbeschreibung „`Rechtsfolge`“ setzt keine Methoden voraus. Ihre Implementierung zeigt lediglich den Rahmenwerksklassen an, dass die Klasse als Rechtsfolge Verwendung finden kann. Damit die Prüfung in Prüfungsfolgen reibungslos funktioniert, sollte allerdings die allgemeine „`equals`“-Methode richtig implementiert sein, damit Übereinstimmungen richtig erkannt werden.

Für die in Java enthaltenen Standardtypen, die einer Implementierung nicht zugänglich sind, existiert die parametrisierte Klasse „`RechtsfolgeStandardtyp`“, welche ein Objekt des Parametertyps verpacken und so der Verarbeitung im Rahmen des Rahmenwerks zuführen kann. Für den Typ „`String`“ existiert zur Vereinfachung des Programmierens die Klasse „`RechtsfolgeText`“.

6.2 TatbestandMeinungsstreit

Die Klasse „`TatbestandMeinungsstreit`“ funktioniert in der Anwendung des Rahmenwerks ähnlich der Klasse „`TatbestandPrüfungsfolge`“. Anders als letztere verschachtelt sie aber nicht auf Ebene der Subsumtion, sondern auf Ebene der Definition, indem verschiedene Meinungen zur Auswahl gestellt werden. Diese Meinungen werden – analog der Methoden „`prüfungsfolgeFestlegen`“ und „`addPrüfungspunkt`“ bei der Prüfungsfolge – innerhalb der zu überschreibenden Methode „`meinungenFestlegen`“ mittels Aufruf der Rahmenwerksmethode „`addMeinung`“ festgelegt.

6.3 TatbestandStrategiewahl

Die Klasse „`TatbestandStrategiewahl`“ fragt nach dem Vorbild von DISUM ab, ob eine Prüfung vertieft erfolgen soll oder ob der Nutzer direkt das Ergebnis vorgeben möchte. Die Wahl wird in der Struktur einer üblichen Tatbestandsprüfung gespeichert. Im Rahmen der Prüfung innerhalb „`TatbestandPrüfungsfolge`“ wird diese Klasse gesondert behandelt. Sie sperrt die weitere Abar-

beitung der Prüfung, bis der Nutzer nicht explizit die genaue Prüfung gefordert hat. Die Strategiewahl kann damit also einfach in eine Prüfungsfolge eingebunden werden.

Als erwartete Rechtsfolge für ein Prüfungsfolgenelement können „StrategiewahlFehltException.Auswahlmöglichkeit. UnproblematischGegeben“ und „StrategiewahlFehltException.Auswahlmöglichkeit. UnproblematischNichtGegeben“ analog „RechtsfolgeWahrheitswert.Wahr“ und „RechtsfolgeWahrheitswert.Falsch“ verwendet werden. Ist das Ergebnis „StrategiewahlFehltException.Auswahlmöglichkeit. GenauePrüfung“, so wird das Element in der Prüfungsfolge im Folgenden ignoriert. Technischer Hintergrund hierfür ist eine spezielle Behandlung in der Klasse „Prüfungsfolgenelement“.

6.4 Falllösungsgrafik

Soll nicht die Klasse „Falllösungsassistent“ verwendet werden, ermöglicht die Klasse „Falllösungsgrafik“, den Sachverhalt grafisch als AWT-Panel aufzubereiten. Dies setzt jedoch voraus, dass das übergebene Tatbestandsobjekt keine Rückfragen mehr auslöst.

7 Vorlagen

7.1 Zivilrecht

Im Rahmen des Anwendungsbeispiels „Vertragsschluss“ ist eine vollständige Prüfungshierarchie für zivilrechtliche Ansprüche entstanden. Inhaltlich vollständig geprüft wird jedoch nur der Vertragsschluss, welcher in die Prüfung des Hauptleistungsanspruchs des Kaufvertrags integriert wird.

Die Struktur kann allerdings ohne Veränderung verwendet werden, um weitere Anspruchsgrundlagen abzudecken. Hierzu muss die Schnittstelle „Anspruchsgrundlage“ implementiert werden und eine Instanz dem Konstruktor der Klasse „Anspruch“ übergeben werden. Sollen weitere vertragliche Hauptleistungsansprüche umgesetzt werden, kann auch die Klasse „VertraglicherErfüllungsanspruch“ bzw. eine ihrer Subklassen erweitert werden. Für vertragliche Nebenansprüche dürfte es – sofern diese teilautomatisiert geprüft werden sollen – erforderlich sein, das semantische Modell des Vertragsinhalts anzupassen, was Anpassungen zumindest an der Klasse „Vertragsinhalt“ erfordert.

Die Prüfung einer Willenserklärung bzw. eines Vertragsschlusses kann über die gleichnamigen Klassen in Prüfungsfolgen integriert werden. Die Prüfung des Vertragsschlusses setzt einen „Handlungskomplex“ voraus, welcher im Sachverhalt unter dem Schlüssel „Kommunikationsverlauf“ gespeichert ist. Die Prüfung der Willenserklärung setzt nur die Übergabe eines einzelnen Kommunikationsaktes voraus; für die Zugangsprüfung ist das Vorliegen eines Kommunikationsverlaufs jedoch zu empfehlen.

Die Subklassen der Klasse „Leistung“ bieten eine im Vergleich zu den Rechtsfolgeklassen des Werkzeugkastens deutlich verbesserte Repräsentation einer Leistungspflicht, entweder in Geld oder sonstiger Weise. Sie werden vollständig durch die Klassen des Werkzeugkastens – insbesondere die Darstellungskomponenten – unterstützt. Dies gilt auch für etwaige Ergänzungen der in ihnen enthaltenen Aufzählungstypen.

Auch die Klassen zur Integration der Software „Schusters Kostentenor“ können weiterverwendet werden.

7.2 Strafrecht

Die im Rahmen des Anwendungsbeispiels „Straftaten“ entwickelten Klassen sind deutlich schlechter verallgemeinerungsfähig als die Anspruchsprüfung, da der Aufbau wie auch die teilautomatisierten Prüfungen auch des allgemeinen Teils an verschiedenen Stellen auf den Sachverhalt der umgesetzten Beispielstraftaten abstellen. Dennoch kann das Anwendungsbeispiel einen leichten Einstieg in die Entwicklung der Prüfung weiterer Straftaten bieten. Hierfür müssen die Klassen an den entsprechenden Stellen angepasst werden. An dieser Stelle soll zum besseren Verständnis dargestellt werden, wie die Klassen aufgebaut sind. Einstiegspunkt der Prüfung ist die Klasse „`Gesamtprüfung`“, deren Konstruktor Objekte des Typs „`Straftat`“ übergeben werden. Diese muss zur Anpassung an eine spezielle Straftat überschrieben werden. Zentral ist dabei die Methode „`getTatbestandsmerkmaleInitialisieren()`“, welche ein Feld an Objekten der Schnittstelle „`Straftatbestandsmerkmal`“ liefert. Durch die Gestaltung dieses Feldes wird der Inhalt des Straftatbestandes implementiert. Die einzelnen Tatbestandsmerkmale sollten dabei den passenden Subtypen der Schnittstelle zugewiesen werden, damit die Prüfungslogik notwendigenfalls den dogmatischen Typ des Merkmals erkennt und das Merkmal entsprechend behandeln kann. Durch Überschreiben der Methoden „`getBesondereRechtfertigungsgründe()`“ und „`getBesondereEntschuldigungsgründe()`“ können entsprechende Prüfungspunkte an der richtigen Stelle eingefügt werden.

7.3 Sachverhalt

Im Rahmen der Anwendungsbeispiele sind auch Klassen entstanden, welche auch in anderen Kontexten Sachverhaltselemente darstellen können.

Das allgemeinste Element ist die Klasse „`Personendaten`“. Der Name ist selbsterklärend.

Die übrigen verwendeten Klassen sind alle der Handlungssatzstruktur zugehörig, deren Funktion und Begründung in der Masterarbeit, in deren Rahmen dieser Werkzeugkasten entwickelt wurde, beschrieben werden. Deren Zentralelement ist die Klasse „`Handlungssatz`“ welche Personendaten zu Subjekt und Objekt, ein Prädikat, welches die Art der Handlung klassifiziert, sowie verschiedene Attribute enthalten. Die Art (=Klassenzugehörigkeit) der Attribute und ihre Verwendung sowie das Vorhandensein eines Objekts wird durch die Prädikate festgelegt, welche von der Klasse „`Prädikat`“ abgeleitet werden. Mehrere Handlungssätze können in einem „`Handlungskomplex`“-Objekt zusammengefasst werden. Handlungskomplexe, Handlungssätze und Attribute können auch unmittelbar Teil des Sachverhalts sein. Diese Stufung ist wichtig, um einen genaueren Zugriff zu ermöglichen, damit ein möglichst kleiner Teil des Sachverhalts invalidiert werden muss. In jedem Fall ist zu erwarten, dass für den Handlungssatz anwendungsspezifische Prädikate und Attribute implementiert werden müssen.

8 sonstige Informationen

Tiefergreifende technische Details können der Javadoc-Dokumentation sowie der Masterarbeit entnommen werden, in deren Rahmen dieser Werkzeugkasten entwickelt wurde.

Diese Informationen sind für die übliche Anwendung nicht erforderlich. Sie sind jedoch wichtig, wenn entweder eine eigene Schnittstelle implementiert oder atypische Prüfungen vorgenommen werden müssen. Auch ermöglichen sie eine bessere Anpassung des zurückgegebenen Textes, etwa indem für Urteil und Gutachten unterschiedliche Formulierungen implementiert werden.