Lösung: Semestrale Softwaretechnik I – WS02/03

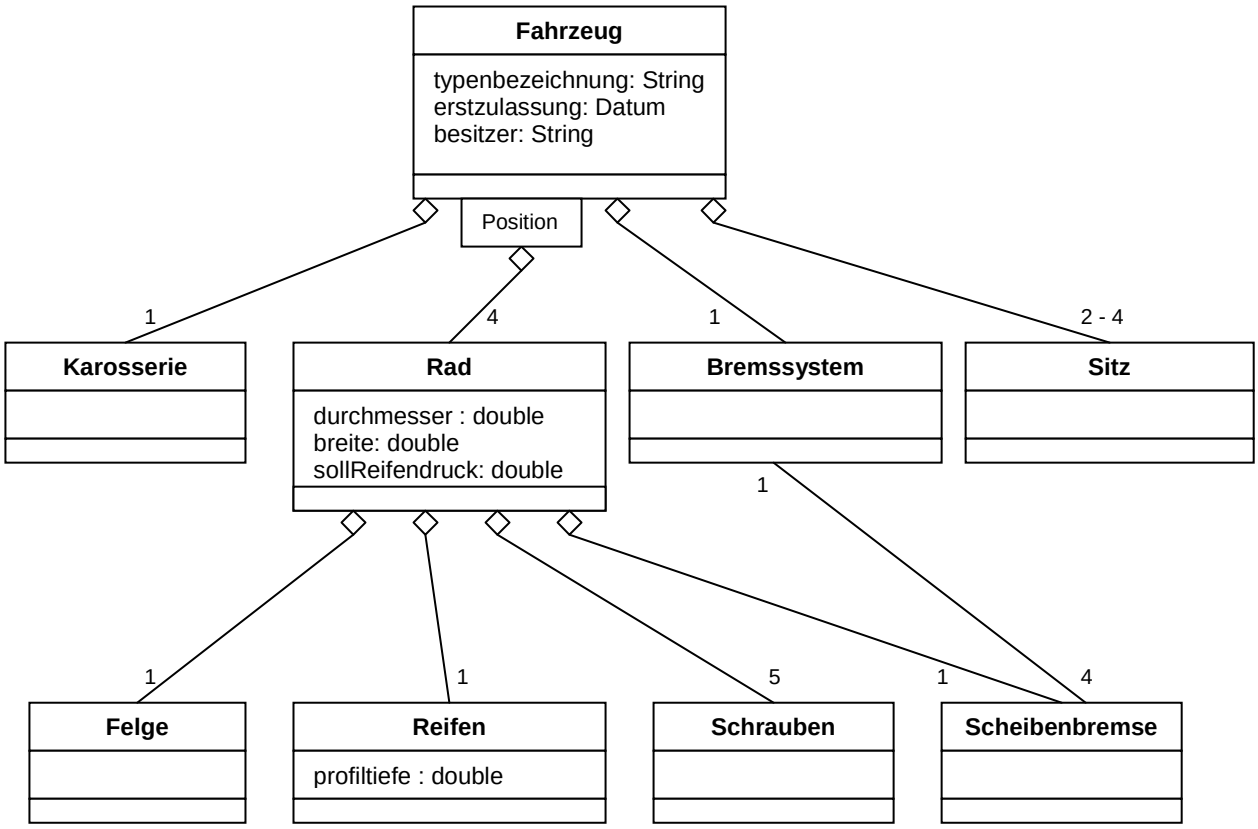
1.)Klassendiagramm

a) Klassenaufbau

Zur Umsetzung der Beziehung zwischen Auto und Rad gibt es mehrere Möglichkeiten. In diesem Beispiel wird eine qualifizierte Assoziation verwendet. Ebenso können statt der Aggregationen Kompositionen verwendet werden.

Siehe unten.

b) Attribute und Assoziationen



Position = {vorneRechts, vorneLinks, hintenRechts, hintenLinks}

c) OCL

context Fahrzeug f inv:
f.rad[vorneLinks].sollReifendruck == f.rad[hintenLinks].sollReifendruck

context Fahrzeug f inv:
f.rad[vorneRechts].sollReifendruck == f.rad[hintenRechts].sollReifendruck

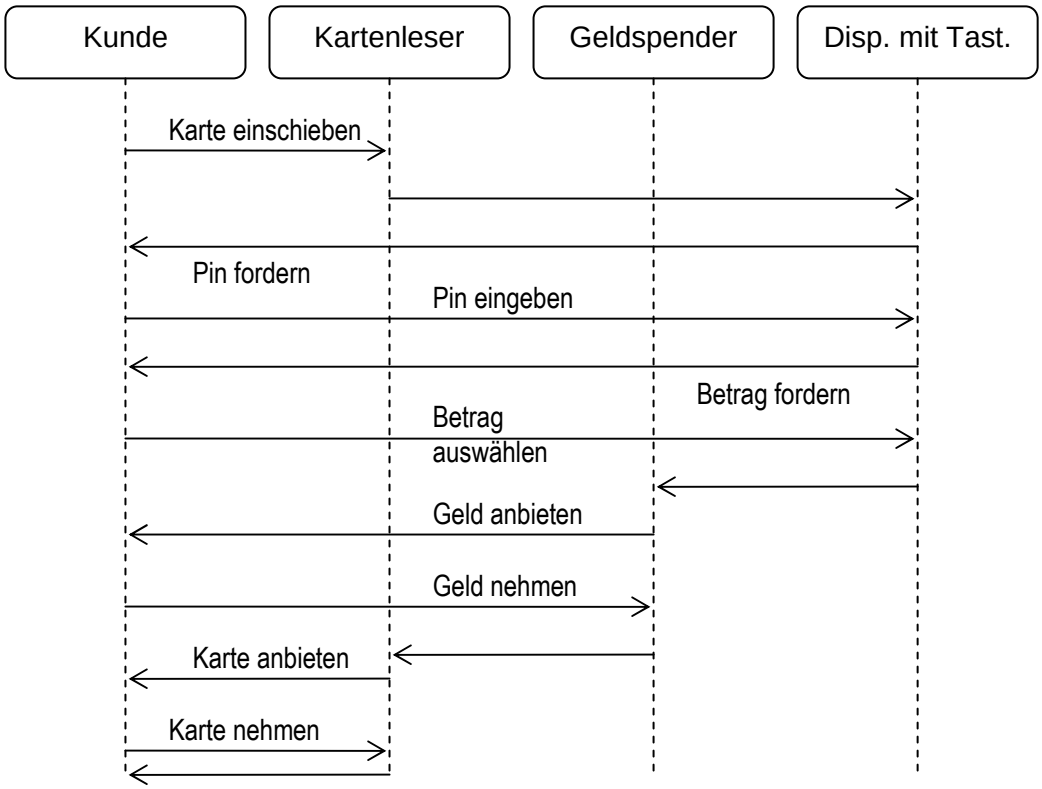
context Fahrzeug f inv:
f.rad[vorneLinks].reifen.profiltiefe == f.rad[vorneRechts].reifen.profiltiefe

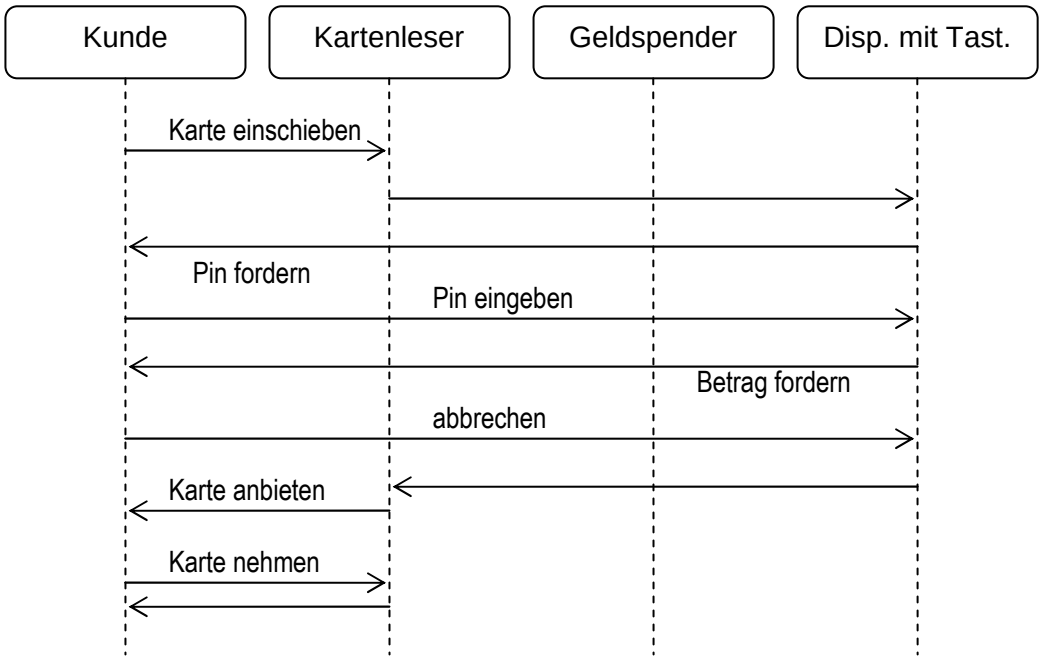
context Fahrzeug f inv:
f.rad[hintenLinks].reifen.profiltiefe == f.rad[hintenRechts].reifen.profiltiefe

2.)Verhaltensspezifikation

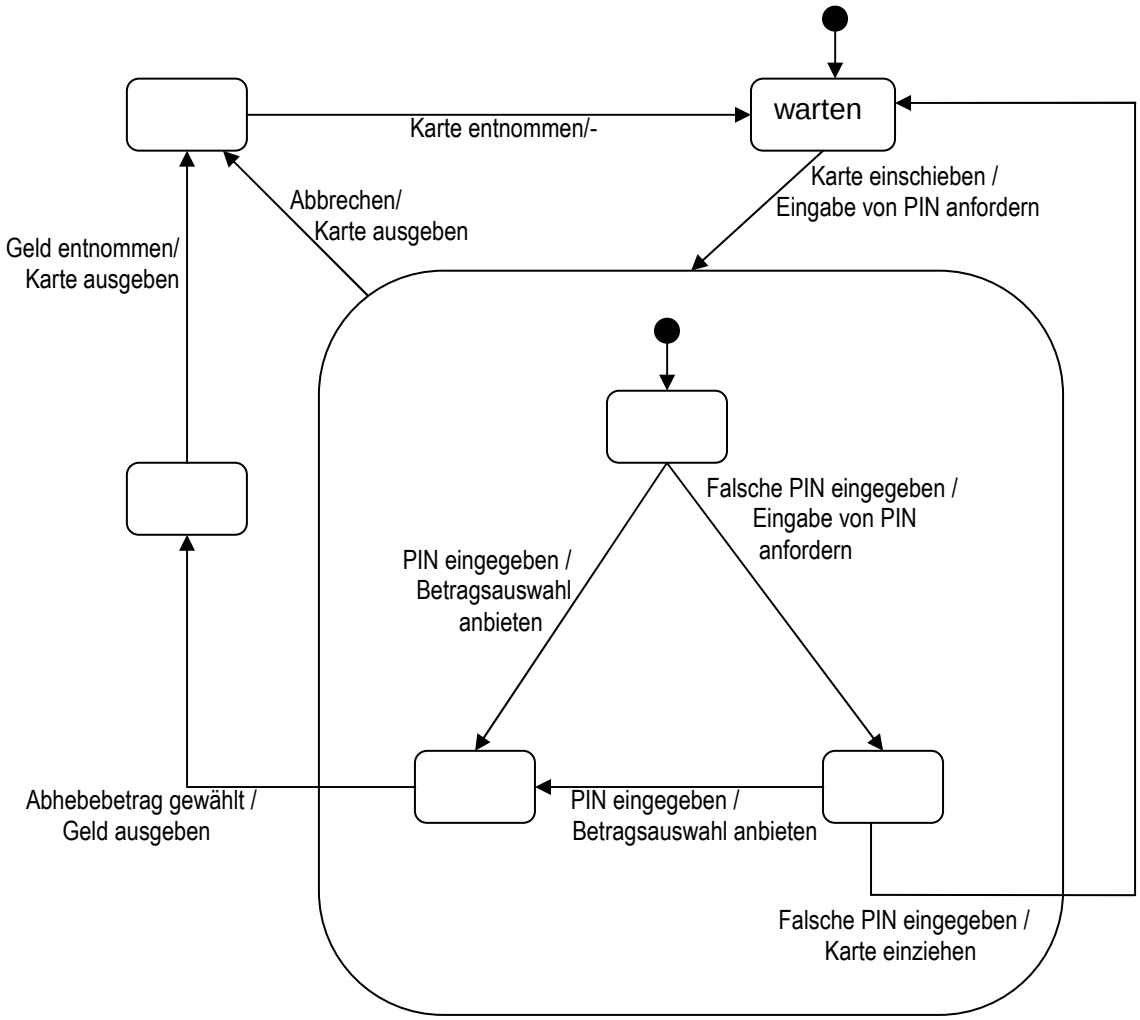
a) Sequenzdiagramme

Bei den Sequenzdiagrammen sind verschiedene Abläufe möglich. Bewertet wird hier die korrekte Darstellung eines Ablaufes, d.h. das SD muss in sich konsistent sein. Ebenso gibt es bei der Interpretation der Aktivitätsbalken verschiedene Möglichkeiten.





b) Statechart



3.) Testdefinition / JUnit

a) Äquivalenzklassen

Aufgrund der relativ geringen Komplexität der Methode, ist zu erwarten, dass sich die Anzahl der Tests etwa bei 10 bewegt. Folgende Kriterien für die Unterteilung in Äquivalenzklassen werden identifiziert:

Äquivalenzklasse	***Beschreibung***
A1	Kein Objekt
A2	Leerer String
A3	Nichtleerer String mit Kleinbuchstaben
A4	Nichtleerer String mit Großuchstaben
A5	Nichtleerer String mit anderen Zeichen
A6	Mehrere gleiche Buchstaben

Wie in der Softwaretechnik oft üblich, gibt es hier keine absolut korrekte Lösung. So kann die Äquivalenzklassenbildung durchaus verfeinert werden.

b+c) Eingabestrings, die die Äquivalenzklassen repräsentieren.

Wir wählen:

Testfall-Nummer	***Eingabestring***	***Äquivalenz-klassen***	***Kommentar***
1	Null	A1	Kein Objekt
2	„“	A2	Leerer String
3	„o“	A3	Kleiner Buchstabe
4	„O“	A4	Großer Buchstabe
5	„1:!öÜ§“	A5	String ohne Buchstaben
6	„mm“	A3,A6	Mehrere gleiche Buchstaben
7	„mM“	A3,A4	Gleicher Buchstabe (Groß/Klein)
8	„sONmM“	A3,A4	Mehrere Buchstaben (Groß/Klein)
9	„2mmr21M14“	A3,A4,A5,A6	Buchstaben und andere Zeichen gemischt
10	„tTsSrRqQpPoOnNmMlL“	A3,A4	Kompletter Ausgabeumfang und Randwerte

Dabei ist eine Grenzfallanalyse bereits enthalten, die auch folgende Fälle betrachtet:

- Prüfung der ASCII-Zeichen an den Randwerten {l,m,s,t,L,M,S,T} des Bereichs [m-sM-S] (Fall 10)
- Prüfung, ob erstes und letztes Zeichen im String korrekt bearbeitet werden (Fall 8,9)
- Werden gleiche Buchstaben nur einfach ausgegeben (Fall 6,9)
- Werden Groß- und Kleinbuchstaben derselben Sorte unterschieden (Fall 7)
- Jedes Zeichen wird einmal getestet (Fall 10)

b) Erwartete Ausgaben

ID	*Ausgabestring*
1	„“
2	„“
3	„o“
4	„O“
5	„“
6	„m“
7	„Mm“
8	„MmNOs“
9	„Mmr“
11	„MmNnOoPpQqRrSs“

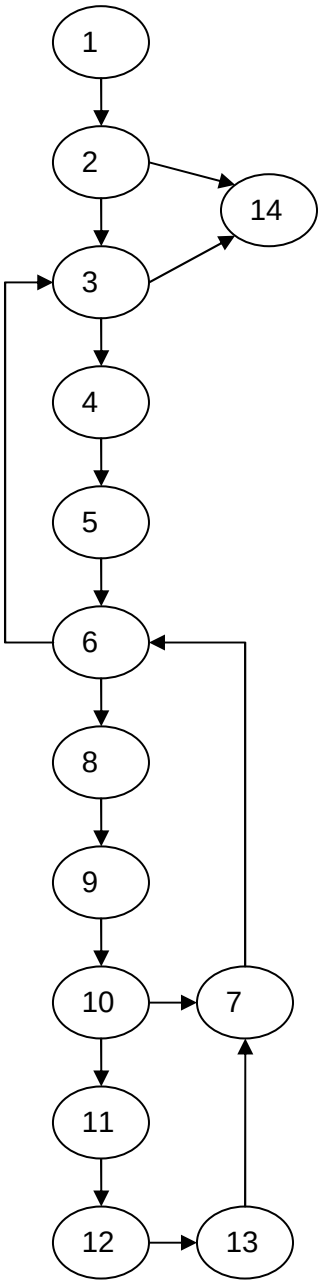
d) TestImplementierung

Es reicht aus, alle Tests in eine Methode zu schreiben. Auf- und Abbau von Testumgebungen mit „setUp“ und „tearDown“ ist nicht nötig:

```
import org.junit.*;

public class StringAnalyserTest extends TestCase {

	/**
	 *	Prüfung aller 11 identifizierten Fälle in je einer Zeile
	 */
	public void testContainsCharacters () {

		assertTrue(StringAnalyzer.containsCharacters(null).equals(“”));
		assertTrue(StringAnalyzer.containsCharacters(“”).equals(“”));
		assertTrue(StringAnalyzer.containsCharacters(“o”).equals(“o”));
		assertTrue(StringAnalyzer.containsCharacters(“O”).equals(“O”));
		assertTrue(StringAnalyzer.containsCharacters(“1:!öÜ§”).equals(“”));
		assertTrue(StringAnalyzer.containsCharacters(“mm”).equals(“m”));
		assertTrue(StringAnalyzer.containsCharacters(“mM”).equals(“Mm”));
		assertTrue(StringAnalyzer.containsCharacters(“sONmM”).equals(“MmNOs”));
		assertTrue(StringAnalyzer.containsCharacters(“2mmr21M14”).equals(“Mmr”));
	assertTrue(StringAnalyzer.containsCharacters(“tTsSrRqQpPoOnNmMlL”).equals(“MmNnOoPpQqRrSs”));
	}
}
```

4.) Testfallüberdeckung

a) Kontrollflussgraph



b) Anweisungsüberdeckung

Ein einziger Datensatz ist bereits ausreichend, um eine Anweisungsüberdeckung zu erreichen. Wir wählen:

Eingabe: {4,3}

Anweisungen werden in dieser Reihenfolge ausgeführt:

1-2-3-4-5-6-8-9-10-11-12-13-7-6-3-4-5-6-8-9-10-7-6-3-14

und überdeckt damit alle Anweisungen.

c) Zweigüberdeckung

Die Zweigüberdeckung erfordert
Eingabe: {4,3} Zweige: 1-2-3-4-5-6-8-9-10-11-12-13-7, 10-7, 7-6, 6-3-14
Eingabe: {1} 1-2-14

d) Pfadüberdeckung

Es gibt unendlich viele Pfade durch den Graphen, da Schleifen enthalten sind, die aus dem while- und dem for-Statement entstehen. Eine Pfadabdeckung ist daher nicht möglich.

e) Beispiel für zwingend zwei Testfälle für den C_0-Test

Sei ein Attribut a vom Typ int gegeben:

```
public void method (int a) {
   if (a >= 0)
      a--;
   else
      a++;
}
```

5.)Invarianten / Vor-/Nachbedingungen

Die Lösung wird in der Fassung der OCL, wie in der Vorlesung vorgestellt. Der weniger komfortable OCL-Standard wird ebenfalls akzeptiert.

a) Invarianten:

(1) context Filiale inv:
eigentum->size <= 1000

(2) context Benutzer inv:
besitz.eigentümer->size <= 3

(3) context AusleihObjekt inv:
besitzer == null ||
eigentümer.kunde->contains(besitzer)

oder:
context Filiale inv:
kunde->contains(eigentum.besitzer)

(4) context Benutzer inv:
forall o,p in besitz:
o.titel == p.titel implies o == p

oder:
context Ausleihobjekt a,b inv:
a.titel == b.titel implies a.besitzer == null || a.besitzer != b.besitzer

b) Vor-/Nachbedingungen

(5) context Filiale::buchauslastung():real
pre: exists Buch b: eigentum->contains(b)
post:result == {Buch o in kunde.besitz}->size / {Buch o in eigentum}->size

(6) context Video::alternative():Filiale
pre: besitzer != null &&
exists Video v: v.titel == self.titel && v.besitzer==null
post: exists x in result.eigentum:
x.titel == titel && x.besitzer==null

oder:
context Video::alternative():Filiale
pre: ausleihstatus != „frei“
post: exists x in result.eigentum:
x.titel == titel && x.ausleihstatus == „frei“

6.)Persistente Objekte

Wie in der Vorlesung besprochen, gibt es mehrere alternative Umsetzungsmöglichkeiten. Eine davon ist im folgenden angegeben. Sie nutzt eine gemeinsame Tabelle für alle Ausleihobjekte und drei Erweiterungstabellen für klassenspezifische Informationen.

Benutzer	*BenutzerOID*	*ID*	*saldo*
	13	2232	0

Filiale	*FilialeOID*	*Name*
	55	München Nord

AusleihObjekt	*AusleihobjektOID*	*Titel*	*Ausleihfrist*	*ausleihstatus*	FilialeOID	BenutzerOID
	1	IX	5.5.2003	entliehen	55	13
	2	Dr.No	5.5.2003	entliehen	55	13
	3	JSP	5.5.2003	entliehen	55	13

Filiale_Benutzer	FilialeOID	BenutzerOID
	2	2

Buch	*AusleihobjektOID*	*autor*	*anzahlKapitel*
	3	James	3

Video	*AusleihobjektOID*	*spieldauer*	*aufnahmeMedium*
	2	3	DVD

Zeitschrift	*AusleihobjektOID*	*anzahlArtikel*	*ausgabeNr*	*ausgabeTitel*
	1	2	2	Flora und Fauna