Semestrale

Softwaretechnik: Software-Engineering

Dr. Bernhard Rumpe
WS 2002/2003
06. Februar 2003

Nachname:	
Vorname:	
Matrikelnummer:	
Hauptfach:	
Informatik als:	[ ] Diplom [ ]Bachelor [ ]Master [ ]Nebenfach [ ]Aufbau [ ]Sonstiges:...
Semester:	
Geburtsdatum:	

Veröffentlichung von Prüfungsergebnissen im Internet

Die Ergebnisse von Klausuren / Prüfungen werden durch Aushang in Schaukästen mittels Listen mit Matrikelnummer und Punktezahl / Note bekannt gegeben. Viele Studierende wünschen überdies eine Veröffentlichung dieser Listen im Internet. Durch die Vielzahl von Veröffentlichungen im Internet kann nicht sichergestellt werden, dass nicht doch eine Zuordnung von der Matrikelnummer zur Person in Einzelfällen möglich wird. Daher dürfen die einzelnen Ergebnisse nur nach vorheriger Zustimmung der Betroffenen im Internet veröffentlicht werden.

Einwilligungserklärung

Hiermit stimme einer Veröffentlichung meines Klausur- / Prüfungsergebnisses in der Semestrale Softwaretechnik vom 6.2.2003 unter Verwendung meiner Matrikelnummer im Internet zu.

Bitte ankreuzen: [ ] ja [ ] nein

Name:_____

Matrikelnummer:_____

Datum, Unterschrift:_____

Aufgabe 1: Klassendiagramm (6,5 Punkte)

In dieser Aufgabe ist ein Klassendiagramm mit Attributen zu entwickeln. Lesen Sie die ganze Aufgabe, bevor Sie mit dem Zeichnen des Diagramms anfangen.

a) Zeichnen Sie ein Klassendiagramm eines Fahrzeugs, das folgender Beschreibung entspricht: „Ein Fahrzeug besteht aus einer Karosserie, einem Bremssystem, zwei bis vier Sitzen und vier Rädern an den unterschiedlichen Positionen (vorneLinks, vorneRechts, hintenLinks, hintenRechts). Jedes Rad besteht aus einer Felge, einem Reifen, einer Scheibenbremse und fünf Schrauben."

b) Tragen Sie folgende Aussagen in das Diagramm ein (wählen Sie ggf. geeignete Bezeichnungen für Attribute oder Assoziationen):

   (1) Räder haben Durchmesser, Breite und Soll-Reifendruck.
   (2) Das Bremssystem ist mit allen Scheibenbremsen verbunden.
   (3) Fahrzeuge haben eine Typenbezeichnung, ein Datum der Erstzulassung und einen Besitzer.
   (4) Reifen haben eine Profiltiefe.

c) Eine Vorgabe fordert, dass bei jedem Fahrzeug die beiden Reifen einer Seite jeweils denselben Soll-Reifendruck besitzen. Bei den Reifen vorne und hinten soll außerdem die Profiltiefe jeweils gleich sein. Formulieren Sie diese Eigenschaft in Ihrem Modell als Invariante in der OCL.

Aufgabe 2: Verhaltensspezifikation (7 Punkte)

In dieser Aufgabe wird mit der UML ein Bankautomat spezifiziert.

a) Beschreiben Sie jeweils mit jeweils einem Sequenzdiagramm einen exemplarischen Ablauf der Aktivitäten aus der Sicht des Kunden, die ein Kunde bei den folgenden Szenarien an dem Bankautomaten ausführt:

   (1) Der Kunde hebt Geld ab.
   (2) Der Kunde bricht vor der Auswahl der abzuhebenden Summe ab.

   Nutzen Sie dabei den Kunden, Display mit Tastatur, Kartenleser und Geldspender als interagierende Objekte.

b) Der Bankautomat hat einen internen Controller. Zeichnen Sie ein StateChart, das das Verhalten dieses Controllers spezifiziert.
   Auslöser für Transitionen sind dabei die Ereignisse (/Aktivitäten), wie sie in Teil a) der Aufgabe identifiziert worden sind.
   Die Aktionen des Automaten sind ebenfalls an den Transitionen zu notieren, aber Zustände müssen nicht mit Namen versehen werden.

   Beachten Sie dabei folgende Randbedingungen:

   (1) Der Benutzer kann den Vorgang an sinnvollen Stellen abbrechen.
   (2) Der Benutzer kann bei falscher Eingabe der Geheimzahl diese Eingabe nur einmal wiederholen. Nach zwei Fehlversuchen wird die Karte eingezogen.

Aufgabe 3: Testdefinition / JUnit (8 Punkte)

Gegeben sei eine Methode *containsCharacters* mit folgender Signatur:

```
class StringAnalyzer {
     public static String containsCharacters (String input) {
       … // Der Code der Methode ist unbekannt.
     }
}
```

Die Methode bekommt als Eingabeparameter *input* einen beliebigen String.

Die Methode untersucht, welche ASCII-Buchstaben aus [m-sM-S] (also den Buchstaben von „m“ bis „s“ in Groß- und Kleinschreibung) in dem String *input* vorkommen, und gibt diese in einem sortierten String zurück. Sollte der gleiche Buchstabe mehrmals im Eingabestring vorkommen, so wird nur ein Repräsentant im Rückgabestring aufgeführt. Der Rückgabestring soll alphabetisch sortiert sein (aber Großbuchstaben jeweils vor dem entsprechenden Kleinbuchstaben). Alle anderen Zeichen werden ignoriert.

(Hinweis: Die Testdefinition im Folgenden erlaubt einigen Freiraum, der nicht dazu genutzt werden sollte, sich in zu vielen Tests zu verlieren.)

a) Überlegen Sie, welche Äquivalenzklassen von Tests Sie sinnvollerweise testen sollten.

b) Geben Sie nun für jede Äquivalenzklasse wenigstens einen Eingabestring und das Soll-Ergebnis an.

c) Welche Randfälle sind sinnvollerweise ebenfalls zu testen?

d) Schreiben Sie eine Java-Klasse *StringAnalyzerTest*, mit der Sie unter Zuhilfenahme von JUnit den Code der oben gegebenen Methode *containsCharacters* testen. Kleine syntaktische Fehler sind erlaubt.

Aufgabe 4: Testfallüberdeckung (6,5 Punkte)

Gegeben sei folgende Funktion „sortiere“, die mittels Bubblesort ein Feld von Variablen des Typs „int“ sortiert.

```
public int[] sortiere (int bestand[]) {
 //Anweisung-Nr.
  boolean change = true;                        // 1
  if (bestand.length > 1) {                     // 2
    while (change) {                            // 3
      change = false;                           // 4
      for (int i = bestand.length - 1;          // 5
                   i > 0;                       // 6
                   i--) {                       // 7
        int i1 = bestand[i];                    // 8
        int i2 = bestand[i – 1];                // 9
        if (i1 < i2) {                          // 10
          bestand[i] = i2;                      // 11
          bestand[i – 1] = i1;                  // 12
          change = true;                        // 13
        }
      }
    }
  }
  return bestand;                               // 14
}
```

a) Entwerfen Sie einen Kontrollflussgraphen für die Methode “sortiere”.

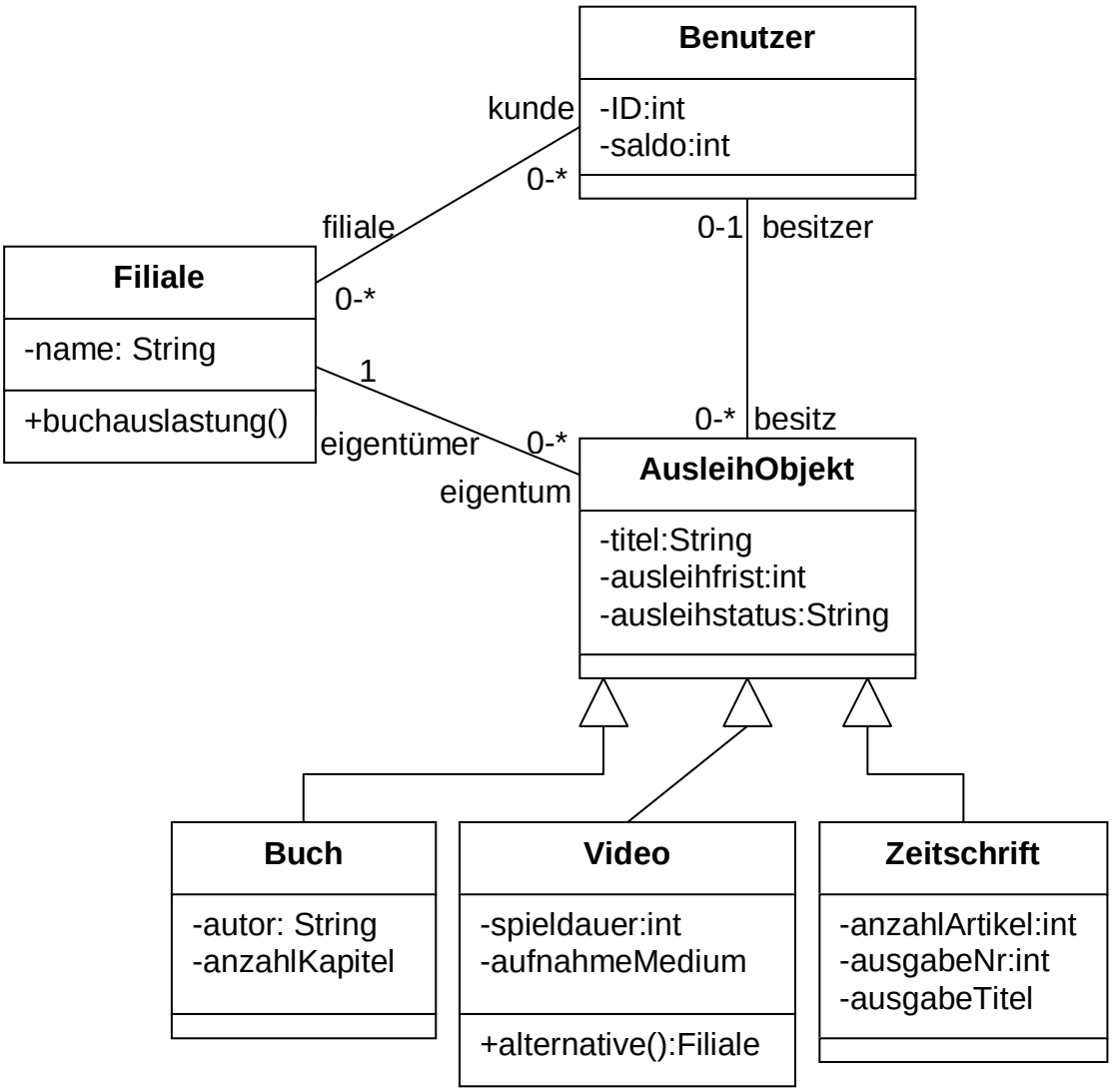
b) Geben Sie ein Feld mit Eingabewerten an, das nötig ist, um eine Anweisungsüberdeckung (C_0-Test) zu erreichen. Schreiben Sie die Reihenfolge der Anweisungen auf, in der sie getestet werden.

c) Erreichen Sie damit auch eine Zweigüberdeckung? Begründen Sie kurz Ihre Antwort. Falls nicht, dann formulieren Sie ein weiteres Feld, um auch die übrigen Zweige zu überdecken. Notieren Sie zum neuen Testfall wieder den Pfad (als Anweisungsfolge), der damit getestet wird.

d) Wie viele Fälle sind in der Pfadabdeckung notwendig? Kurze Begründung.

e) Formulieren Sie ein minimales Programm, für das mindestens zwei Eingabewerte notwendig sind, um eine Anweisungsüberdeckung zu erreichen.

Aufgabe 5: Invarianten / Vor-/Nachbedingungen (6 Punkte)



Gegeben sei obiges Klassendiagramm.

a) Drücken Sie mit der OCL folgende Bedingungen aus:
   (1) Eine Filiale kann maximal 1000 Ausleihobjekte als Eigentum haben.
   (2) Ein Benutzer kann Ausleihobjekte von maximal 3 verschiedenen Filialen gleichzeitig ausleihen.
   (3) In einer Filiale können nur die dort registrierten Kunden ausleihen.
   (4) Ein Benutzer kann nicht mehr als ein Ausleihobjekt mit dem gleichen Titel gleichzeitig ausgeliehen haben.

b) Formulieren Sie ein Vor-/Nachbedingungspaar für folgende Methoden:
   (1) Die Methode *buchauslastung*() berechnet für eine Filiale, wie groß der Anteil ausgeliehener Bücher ist.
   (2) Die Methode *alternative*() bestimmt für ein ausgeliehenes Video eine alternative Filiale, die dasselbe Video vorrätig hat.

Aufgabe 6: Persistente Objekte (6 Punkte)

Beschreiben Sie eine Tabellenstruktur für ein relationales Datenbanksystem, in der die Daten der Objekte des Klassendiagramms aus Aufgabe 5 gespeichert werden können. Zeichnen Sie die Tabellenköpfe und geben Sie zur Veranschaulichung für jede Tabelle mindestens einen fiktiven Datensatz an.

Lösung: Semestrale Softwaretechnik I – WS02/03

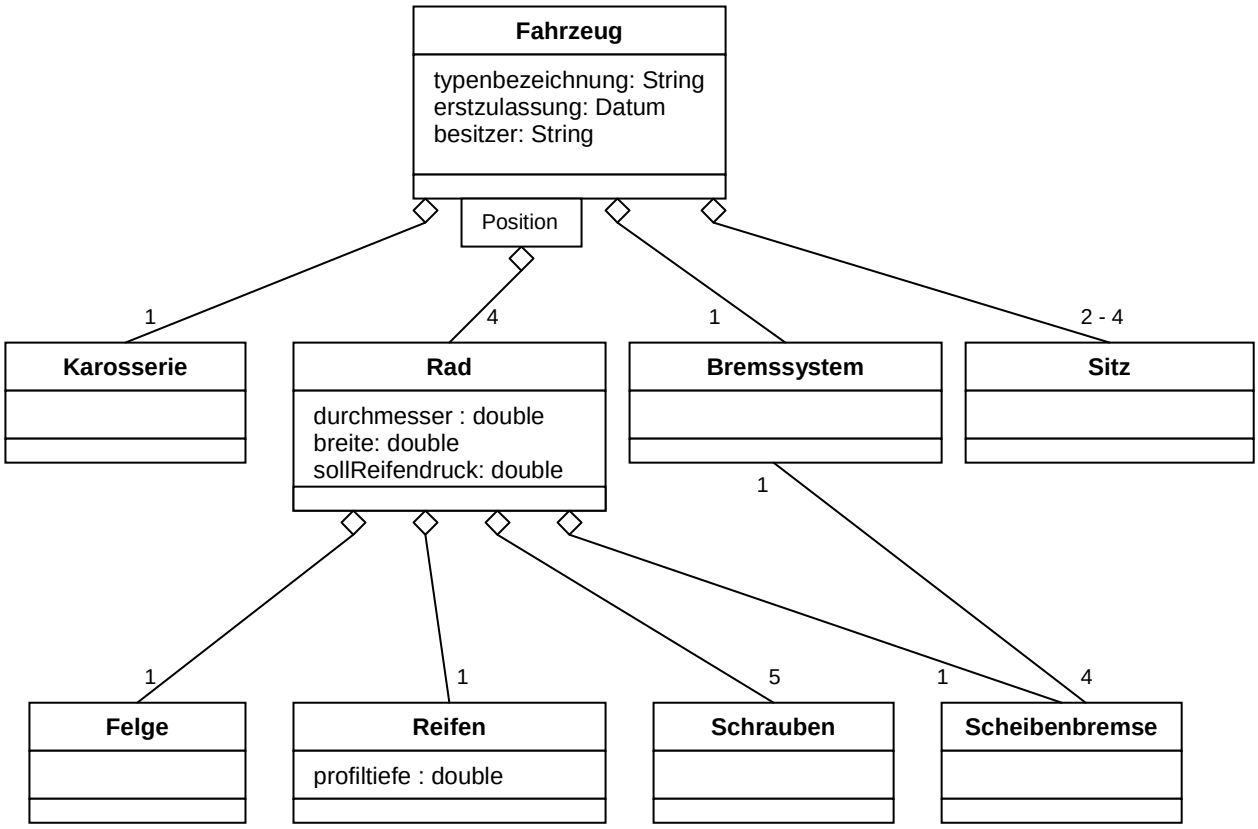
1.)Klassendiagramm

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

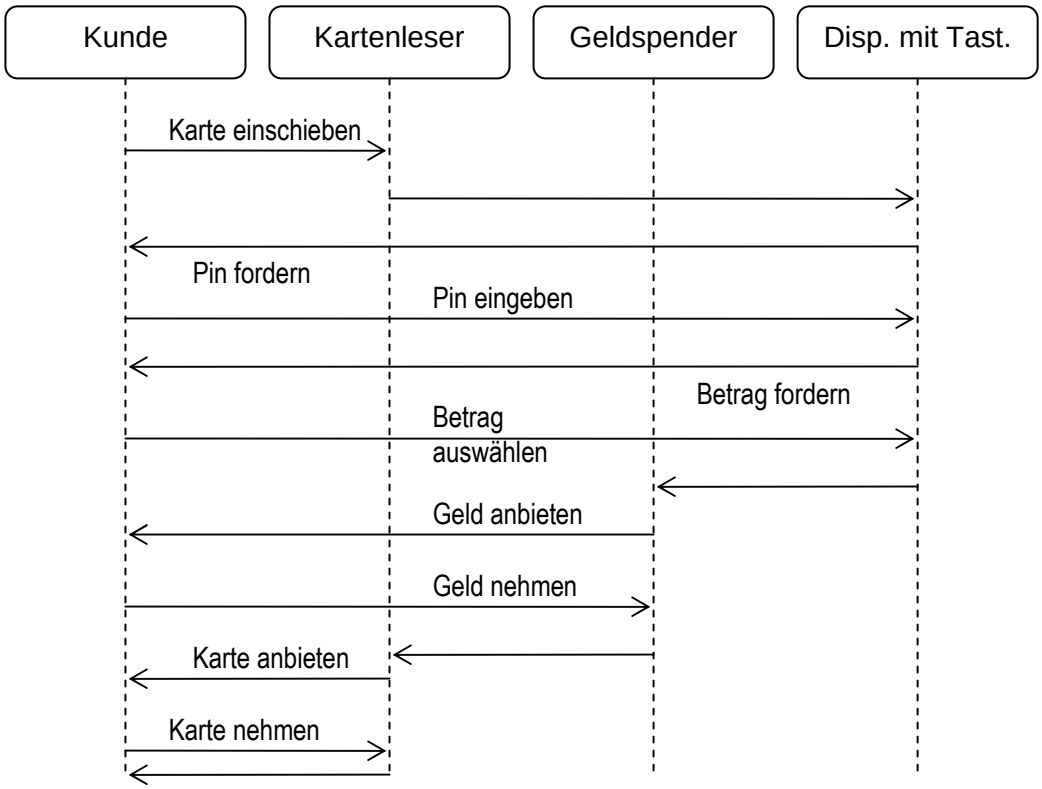
context Fahrzeug f inv:
f.rad[vorneLinks].reifen.profiltiefe == f.rad[vorneRechts].reifen.profiltiefe

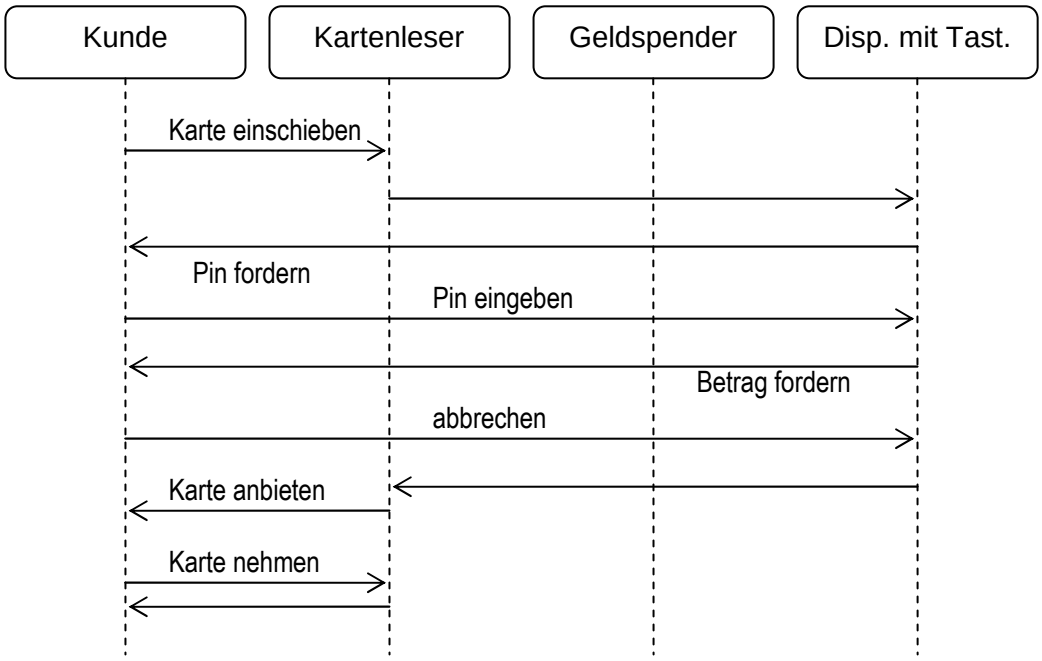
context Fahrzeug f inv:
f.rad[hintenLinks].reifen.profiltiefe == f.rad[hintenRechts].reifen.profiltiefe

2.)Verhaltensspezifikation

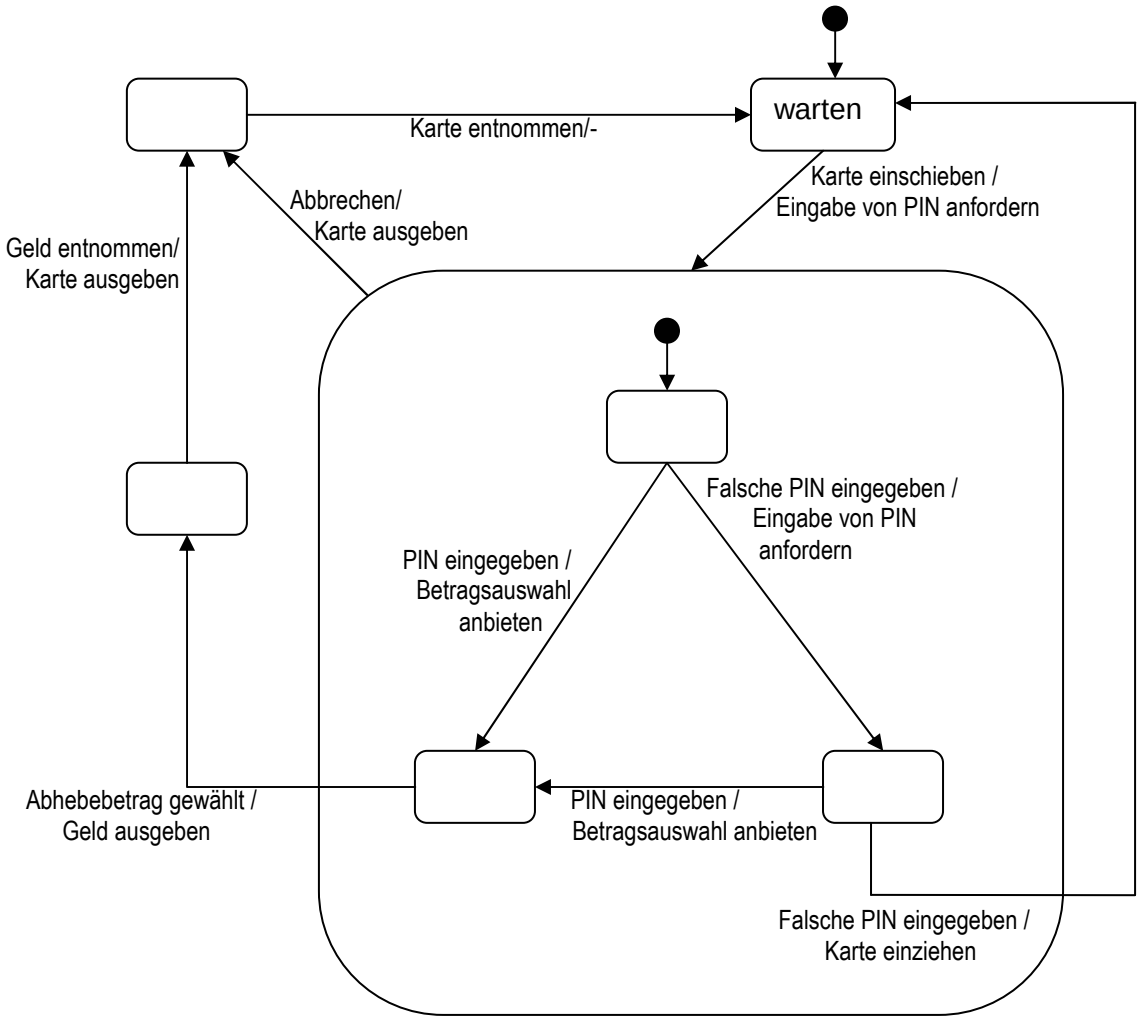
a) Sequenzdiagramme

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“

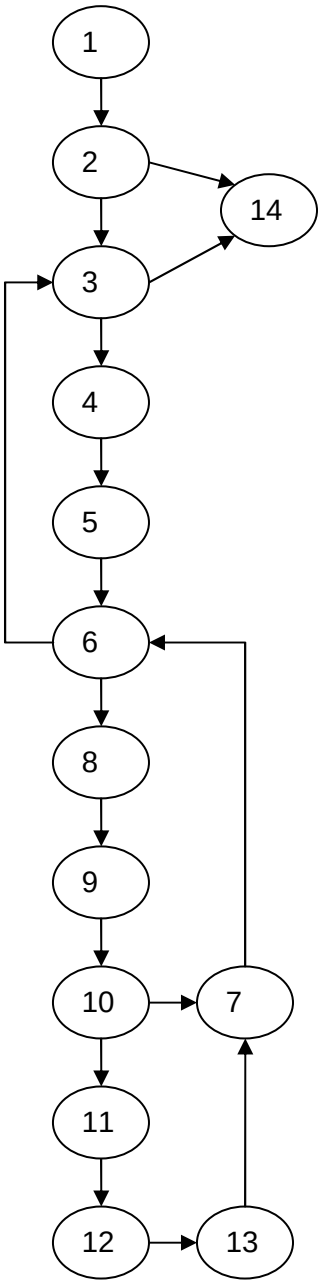
d) TestImplementierung

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