Übungen zur Vorlesung

Softwaretechnologie

-Wintersemester 2009/2010-

Dr. Günter Kniesel, Pascal Bihler

Übungsblatt 10 - Lösungshilfe

Aufgabe 1. *CRC-Karten* (7 Punkte)

Es geht um die Modellierung der folgenden Bibliotheks-Anwendung:

> *Die Unibibliothek verwaltet Bücher, Zeitschriften und Seminararbeiten. Diese Gegenstände sollen rechnergestützt gefunden und ausgeliehen werden.*
> *Zur Identifikation eines Benutzers bei Ausleihe und Rückgabe dient der Informatik-Login. Jeder Benutzer kann höchstens sieben Medien entleihen. Die Ausleihfrist beträgt bei Büchern 4 Wochen, bei Zeitschriften 2 Wochen und bei Seminararbeiten 1 Woche. Die Überziehungsgebühr beträgt bei Büchern 1 €, bei Zeitschriften 3 €, bei Seminararbeiten 5 € pro Tag der Überschreitung.*
> *Benutzer dürfen nur dann weitere Medien ausleihen, wenn sie noch keine 42€ Überziehungsgebühren angesammelt haben und auch bei keinem entliehenen Medium die Frist überziehen.*

a) Erstellen Sie ein minimales Objektmodell für die Bibliotheksanwendung:

   a. Eine textuelle Aufzählung von Klassen mit aussagekräftigen Namen.
   b. Je ein kurzer prägnanter Text (je < 50 Wörter) der die Aufgabe der Klasse beschreibt.

- **Bibliothek**
  Verwaltet Medien und ermöglicht Benutzern deren Suche, Ausleihe, und Rückgabe.
- **Benutzer**
  Ein durch einen Informatik-Login identifizierter Bibliotheksnutzer. Kann Medien suchen, entleihen und zurückgeben. Erhält evtl. Bescheide über Überziehungsgebühren.
- **Medium**
  Gegenstand der gesucht, ausgeliehen und zurückgegeben werden kann. Bei Rückgabe nach Überschreitung einer gewissen Ausleihfrist ist eine Überziehungsgebühr fällig, die sich nach der Art des Mediums richtet.
- **Buch**
  Medium mit Ausleihfrist = 4 Wochen und Überziehungsgebühr = 1 Euro pro Tag
- **Zeitschrift**
  Medium mit Ausleihfrist = 2 Wochen und Überziehungsgebühr = 3 Euro pro Tag
- **Seminararbeit**
  Medium mit Ausleihfrist = 1 Woche und Überziehungsgebühr = 5 Euro pro Tag

b) Unter Einsatz von CRC-Cards[1] soll das Objektmodell um Verantwortlichkeiten und Kollaborationen erweitert werden, und zwar so, dass die Klassen anschließend alles beinhalten, was für folgende Szenarien erforderlich ist:

   a. Suchen
   b. Erfolgreiches Ausleihen
   c. Verweigertes Ausleihen
   d. Rückgabe ohne Überziehung
   e. Rückgabe mit Überziehung

Wenden Sie die in der Vorlesung eingeführten Techniken an. Es kann durchaus sein, dass Sie auf neue sinnvolle Klassen stoßen. Beschreiben Sie diese gegebenenfalls auch durch je eine CRC-Karte.

Klasse: Bibliothek Verwaltet Medien und ermöglicht Benutzern deren Suche, Ausleihe, und Rückgabe.	
Verantwortlichkeiten (Responsibilities)	**Zusammenarbeit (Collaboration)**
Suche Medium anhand bestimmter Merkmale	Medium
Ausleihe eines Mediums für einen Benutzer bis zu einem bestimmten Datum	Medium, Benutzer
Rücknahme eines Mediums	Medium, Benutzer
Überschreitungsgebühren für alle zurückgegebenen Medien berechnen	Medium, Benutzer
Überschreitungsgebühren-Bescheid erstellen	Medium, Benutzer

Klasse: Benutzer Ein durch eine Personenkennzahl identifizierter Bibliotheksnutzer. Kann die Bibliothek nutzen um Medien zu suchen, entleihen und zurückzugeben. Erhält evtl. Bescheide über Überziehungsgebühren.	
Verantwortlichkeiten (Responsibilities)	**Zusammenarbeit (Collaboration)**
Überprüfen ob dieser Benutzer Medien ausleihen darf	---

Klasse: Medium Gegenstand der gesucht, ausgeliehen und zurückgegeben werden kann. Bei Rückgabe nach Überschreitung einer gewissen Ausleihfrist ist eine Überziehungsgebühr fällig, die sich nach der Art des Mediums richtet.	
Verantwortlichkeiten (Responsibilities)	**Zusammenarbeit (Collaboration)**
Max. Ausleihdauer bestimmen für dieses Medium	---
Überschreitungsgebühr pro Tag bestimmen	---
für dieses Medium Ausleihbarkeit prüfen	---

[1] Abgabe in Tabellenform als PDF-Dokument

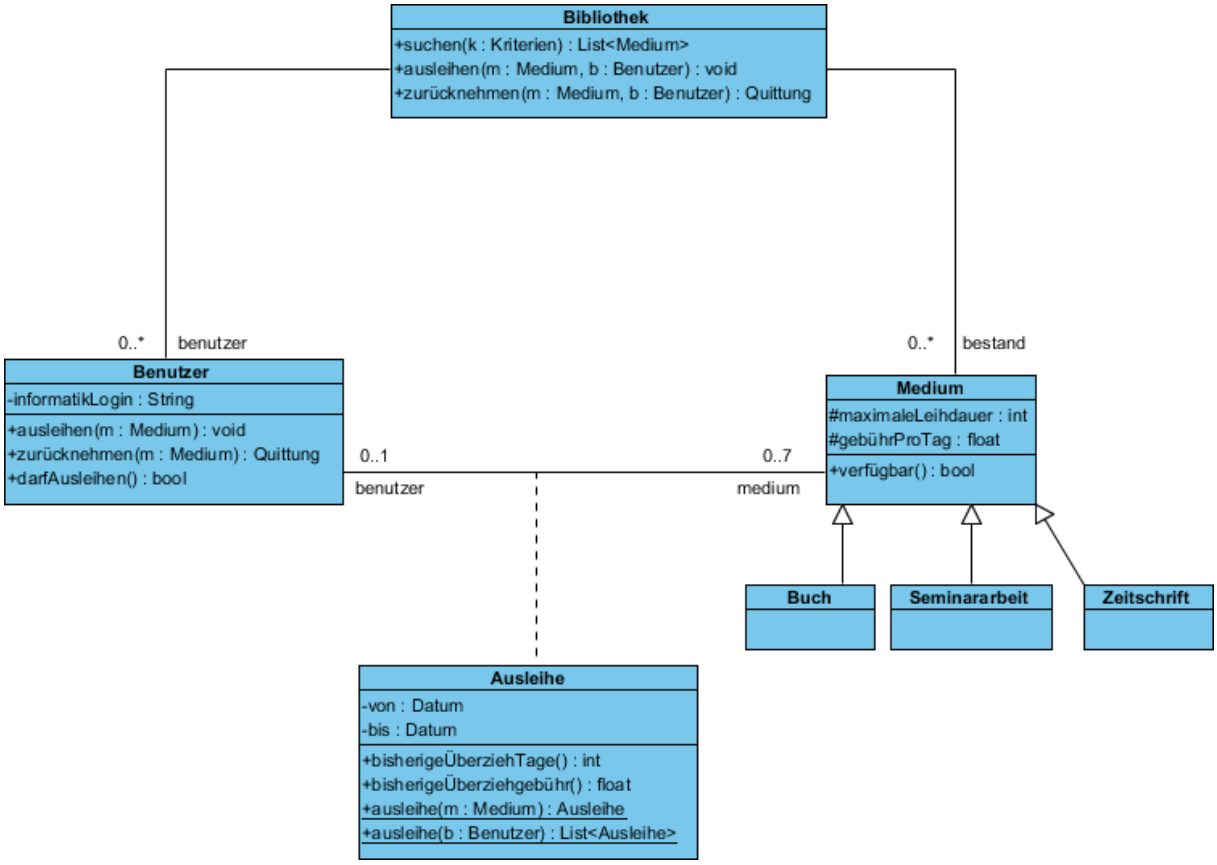
Überschreitungsdauer bestimmen für dieses Medium	---
Aktuelle Überschreitungsgebühr bestimmen für dieses Medium	---

Die CRC-Karten für die Klassen „Buch“, „Zeitschrift“ und „Seminararbeit“ sehen hinsichtlich Aufgaben und Zusammenarbeit genauso aus wie die von „Medium“.

Aufgabe 2. *Design by Contract* (Punkte)

Verfeinern Sie ihren Entwurf zu Aufgabe 1 wie folgt:

a) Erstellen Sie ein Klassendiagramm, das die Ergebnisse der CRC-Analyse widerspiegelt, aber komplett typisiert ist (Ergebnis- und Parametertypen zu jeder Operation). Achten Sie darauf, dass manche Beziehungen evtl. eigene Attribute und Operationen besitzen.



b) Beschreiben Sie für drei Operationen mit unterschiedlichen Verantwortlichkeiten, jeweils ihre Vor- und Nachbedingungen, die sich aus Aufgabe 1 ableiten lassen. Versuchen sie diese so genau wie möglich zu formulieren, unter Verwendung der Object Constraint Language (OCL).

Beispielsweise:

context Ausleihe::bisherigeÜberziehGebühr()

a. **pre**: true
b. **post**: *result* = *self*.medium.gebührProTag * *self*.bisherigeÜberziehTage()

context Bibliothek::ausleihen(Medium m, Benutzer b)

a. **pre**: m.verfügbar() && b.darfAusleihen()

b. **post:** not(m.verfügbar())
&& Ausleihe.ausleihe(m).benutzer = b
&& Ausleihe.ausleihe(m).medium = m

context Bibliothek::zurücknehmen(Medium m)

a. **pre**: not(m.verfügbar())

b. **post**: m. verfügbar ())
&& Ausleihe.ausleihe(m) = null
&& Ausleihe.ausleihe(Ausleihe.ausleihe(m)*@pre*.benutzer).size()
= Ausleihe.ausleihe(Ausleihe.ausleihe(m)*@pre*.benutzer)*@pre*.size()-1

Aufgabe 3. *Jahreszeitbedingte Anwendung von Entwurfsmustern* (Punkte)

Ein reichlich geschmückter Weihnachtsbaum besteht bekannterweise ausschließlich aus *Baumbestandteilen*, d.h. aus *Zweigen*[2] (an denen weitere Zweige wachsen können) und am Ende eines Zweiges evtl. *Baumschmuck*. Baumschmuck können *Engel*, *Kugeln*, *Sterne,* oder *Kerzen* sein.

a) Modellieren Sie den Weihnachtsbaum in Java unter Zuhilfenahme eines geeigneten Entwurfsmusters. Markieren Sie dabei die Rollen des Entwurfsmusters in dem zur entsprechenden Klasse gehörenden JavaDoc-Kommentar.

```
/**
 * Rolle im Composite Pattern: Component
 */
public interface Baumelement {
	void add(Baumelement e);
	void remove(Baumelement e);
	List<Baumelement> getChildren();
	String toString();
}

/**
 * Rolle im Composite-Pattern: Composite
 */
public class Zweig implements Baumelement {

	List<Baumelement> children = new ArrayList<Baumelement>();

	@Override
	public void add(Baumelement e) {
		children.add(e);
	}

	@Override
	public void remove(Baumelement e) {
		children.remove(e);
	}

	@Override
```

[2] Der Einfachheit halber sehen wir Nadeln als integralen Bestandteil der Zweige an und modellieren sie hier nicht. Des Weiteren handelt es sich beim Stamm des Baumes auch um einen (überdimensionierten) Zweig.

```
	public List<Baumelement> getChildren() {
		return children;
	}

	@Override
	public String toString() {
		StringBuffer sb = new StringBuffer("Zweig[");
		for(Baumelement e: children) {
			sb.append(e.toString());
		}
		sb.append("] ");
		return sb.toString();
	}

}

/**
 * Rolle im Composite-Pattern: Leaf
 */
public abstract class Baumschmuck implements Baumelement {

	@Override
	public void add(Baumelement e) {
		throw new NoSuchMethodError();
	}

	@Override
	public void remove(Baumelement e) {
		throw new NoSuchMethodError();
	}

	@Override
	public List<Baumelement> getChildren() {
		throw new NoSuchMethodError();
	}
}

public class Engel extends Baumschmuck {
	@Override
	public String toString() {
		return "Engel ";
	}
}

public class Kugel extends Baumschmuck {
	@Override
	public String toString() {
		return "Kugel ";
	}
}

public class Stern extends Baumschmuck {
	@Override
	public String toString() {
		return "Stern ";
	}
}

public class Kerze extends Baumschmuck {
	@Override
	public String toString() {
		return "Kerze ";
	}
```

```
}
```

b) Entwickeln Sie ein Programm, das Ausgehend vom Stamm einen zufällig generierten Weihnachtsbaum mit verschiedenem Baumschmuck erzeugt. Achten Sie dabei darauf, dass in unserem Kulturkreis die maximale Zweigrekursionstiefe 3 beträgt und jeder Zweig maximal 5 Kindelemente haben kann. Geben Sie eine Beschreibung ihres Baumes auf der Konsole aus, indem Sie die *toString()*-Methoden der Baumelemente geeignet implementieren und auf dem Stamm aufrufen.

```
public class WeihnachtsbaumSimulator {

	static Random rand = new Random();

	public static Baumelement generiereZweig(int ebene) {
		Zweig z = new Zweig();
		int childrenCount = rand.nextInt(5);
		for (int i = 0; i < childrenCount; i++) {
			Baumelement e = null;
			if (ebene < 3) {
				e = generiereZweig(ebene+1);
			} else {
				switch (rand.nextInt(4)) {
				case 3:	e = new Kerze(); break;
				case 2:	e = new Stern(); break;
				case 1:	e = new Kugel(); break;
				default:	e = new Engel(); break;
				}
			}
			z.add(e);
		}
		return z;
	}

	public static void main(String[] args) {
		Baumelement baum = generiereZweig(0);
		System.out.println(baum);
	}
}
```

c) Zur Baumpflege werden im Rheinland traditionell Heinzelmännchen eingesetzt, kleine Spezialisten, die den Baum entlang klettern und eine spezifische Aktion auf den konkreten Baumelementen ausführen können. Modellieren Sie in Ihrem Projekt unter Nutzung eines geeigneten Entwurfsmusters ein abstraktes *Heinzelmännchen*. Ändern Sie, wenn nötig auch bestehende Klassen, so dass später beliebige konkrete Heinzelmännchen die Baumpflege übernehmen können. Markieren Sie die Rollen der am Entwurfsmuster beteiligten Klassen wie in (a) im Quelltext und beachten Sie, dass die Ausprägung einer konkreten Heinzelmännchen-Aktion je nach Baumelement später unterschiedlich sein kann!

```
/**
 * Rolle im Visitor-Pattern: Visitor
 */
public abstract class Heinzelmännchen {

	public abstract void visitEngel(Engel e);
	public abstract void visitKugel(Kugel k);
	public abstract void visitStern(Stern s);
	public abstract void visitKerze(Kerze k);
```

```
	public void visitZweig(Zweig z) {
		for (Baumelement child: z.getChildren()) {
			child.accept(this);
		}
	}
}

/**
 * Rolle im Composite Pattern: Component
 * Rolle im Visitor-Pattern: Element
 */
public interface Baumelement {
	…
	public void accept(Heinzelmännchen h);
}

public class Zweig implements Baumelement {
	…
	public void accept(Heinzelmännchen h) {
		h.visitZweig(this);
	}
}

public class Engel extends Baumschmuck {
	…
	public void accept(Heinzelmännchen h) {
		h.visitEngel(this);
	}
}

public class Kugel extends Baumschmuck {
	…
	public void accept(Heinzelmännchen h) {
		h.visitKugel(this);
	}
}

public class Stern extends Baumschmuck {
	…
	public void accept(Heinzelmännchen h) {
		h.visitStern(this);
	}
}

public class Kerze extends Baumschmuck {
	…
	public void accept(Heinzelmännchen h) {
		h.visitKerze(this);
	}
}
```

d) Implementieren Sie zwei konkrete Heinzelmännchen:

- *Kugel-Anmal-Heinzelmännchen*: Setzt die Farbe von Kugeln auf „rot“.
- *Kerzen-Anzünde-Heinzelmännchen*: Setzt den Status jeder Kerze auf „brennend“.

Ergänzen Sie evtl. zusätzlich benötigte Eigenschaften in den konkreten Baumelementen.

```
public class Kugel extends Baumschmuck {
	public String farbe;
…
```

```
}

public class KugelAnmalHeinzelmännchen extends Heinzelmännchen {

	@Override
	public void visitEngel(Engel a) {}

	@Override
	public void visitKugel(Kugel k) {
		k.farbe = "rot";
	}

	@Override
	public void visitStern(Stern s) {}

	@Override
	public void visitKerze(Kerze k) {}
}


public class Kerze extends Baumschmuck {
	public boolean brennend;
…
}

public class KerzenAnzündeHeinzelmännchen extends Heinzelmännchen {

	@Override
	public void visitEngel(Engel a) {}

	@Override
	public void visitKugel(Kugel k) {}

	@Override
	public void visitStern(Stern s) {}

	@Override
	public void visitKerze(Kerze k) {
		k.brennend = true;
	}
}
```