Übungen zur Vorlesung

# Softwaretechnologie

-Wintersemester 2010/2011-

Dr. Günter Kniesel

# Übungsblatt 11 - Lösungshilfe

## **Aufgabe 1.** *Design by Contract* (11 Punkte)

a) Erläutern Sie die drei Kernkonzepte von ,,Design by Contract“.

- *Vorbedingungen: Aussagen über Zustand bei Methodenaufruf.*
- *Nachbedingungen: Aussagen über Zustand wenn die Vorbedingungen gelten und die Methode normal beendet wird (ohne „exceptions“).*
- *Invarianten:*
  - ⇨ *Eigenschaften, die für den „von außen beobachtbaren Zustand“ eines Objektes immer gelten.*
  - ⇨ *Invarianten dürfen während der Methodenausführung verletzt werden, sofern der inkonsistente Zwischenzustand nicht von außen sichtbar ist (zB durch einen parallel auf dem gleichen Objekt arbeitenden anderen Thread). Inkonsistente Zwischenzustände sind oft notwendig, um den beabsichtigten Endzustand zu erreichen (vergleiche „Consistency“-Eigenschaft bei Datenbanktransaktionen).*

b) Wie können diese Konzepte in Java durch die Verwendung von assert()-Anweisungen umgesetzt werden (siehe http://java.sun.com/developer/technicalArticles/JavaLP/assertions/)? Beschreiben Sie zu jedem kurz die allgemeine Vorgehensweise. Hier ein Auszug aus der Dokumentation von Assertions:

> *An assertion has a Boolean expression that, if evaluated as false, indicates a bug in the code. This mechanism provides a way to detect when a program starts falling into an inconsistent state. Assertions are excellent for documenting assumptions and invariants about a class. Here is a simple example of assertion:*
>
> ```
> BankAccount acct = null;
>
> // ... Get a BankAccount object ...
>
> // Check to ensure we have one
> assert acct != null;
> ```
>
> *This asserts that acct is not null. If acct is null, an AssertionError is thrown. Any line that executes after the assert statement can safely assume that acct is not null. Using assertions helps developers write code that is more correct, more readable, and easier to maintain. Thus, assertions improve the odds that the behavior of a class matches the expectations of its clients.*

- *Vorbedingungen: assert der Vorbedingung sofort am Methodenanfang. Somit wird sichergestellt, dass der Rest der Methode sich darauf verlassen kann, dass die Vorbedingung gilt.*
- *Nachbedingungen: assert der Nachbedingung* nach dem Aufruf einer Operation. *Somit wird sichergestellt, dass die dynamisch gebundene Methode die Nachbedingung der Operation einhält und der Rest des Codes nach der Aufrufstelle sich auf die Gültigkeit der assertion verlassen kann.*
  - ⇨ *Hinweis: Alternativ könnte man versuchen, assertions der Nachbedingung* in der aufgerufenen Methode vor jedem return statement einzubauen*. Somit würde sichergestellt, dass die Methode nur dann normal beendet wird, wenn, ihre Nachbedingung gilt. Da die aufrufende Stelle jedoch nicht weiß, welche Methode von ihr aufgerufen wird (wegen des dynmischen Bindens), und ob die aufgerufene Methode die Nachbedingung tatsächlich sicherstellt, würde dieses Vorgehen es dem Aufrufer nicht ersparen nach der Aufrufstelle noch einmal zu prüfen. Somit macht diese Variante wenig Sinn.*
- *Invarianten: Realisierung wie ein Paar von genau gleich lautenden Vorbedingungen und Nachbedingungen.*

c) Wie würde man mit einem *AssertionError* umgehen, der im Fehlerfall ausgelöst wird? Würde man ihn abfangen und wenn dann wo (direkt nach der Assertion oder an der Stelle die die Methode aufruft, in der sich die Assertion befindet)? Begründen Sie ihre Meinung.

- *In allen Fällen wird der AssertionError weder in der Methode die das assert enthält, noch an der aufrufenden Stelle abgefangen.*
- *Begründung, Teil 1: Sie soll nur sicherstellen, dass Fehler frühzeitig auffallen und im Fehlerfall anhand des „Stack Trace" der Exception und der spezifischen Fehlermeldung die Fehlerquelle schnell lokalisierbar ist.*
- *Begründung, Teil 2: Das Abfangen des AssertionError an allen Stellen wo sie auftreten könnten würde den Code sehr aufblähen, schwer lesbar machen und außerdem das Programm verlangsamen*

d) Implementieren Sie für den Entwurf am Ende des Übungsblatts die Methoden `Veranstaltung.addStudent(Student s)`, `Veranstaltung.removeStudent(Student s)` und `Veranstaltungsangebot.eintragen(Veranstaltung v)`. Garantieren Sie dabei durch die Verwendung geeigneter `assert`-Ausdrücke, dass:

a. ein Student nicht mehrfach als Teilnehmer einer Veranstaltung eingetragen sein kann,
b. ein Student sich nur von einer Veranstaltung abmelden kann, für die er eingetragen ist,
c. der Titel einer Veranstaltung im System zur Raumreservierung eindeutig ist (d.h. es darf keine zwei Veranstaltungen mit demselben Titel geben).

```
public class Veranstaltung {
   Collection<Student> hoerer;

   public void addStudent (Student s) {
      // check that this student is not registered
      assert !hoerer.contains(s) :
    "Student " + s + " already registered for this course";
      hoerer.add(s);
```

```
      }

   public void removeStudent (Student s) {
      // check that this student is registered
      assert hoerer.contains(s) :
     "Student " + s + " not registered for this course";
      hoerer.remove(s);
      }
}


public class Veranstaltungsangebot {
   Collection<Veranstaltung> umfasst;

   public void eintragen (Veranstaltung v) {
      // check that no already registered course has same title
      assert !containsName(v) :
      "Course title" + v.getTitle() + " already registered.";
      umfasst.add(v);
      }

   private bool containsName (Veranstaltung v) {
     String title = v.getTitle();

       for (Veranstaltung veranst : umfasst) {
          if (title.equals(veranst.getTitle()) )
             return true;
       }
       return false;
      }
}
```

e) Wie würden Sie Vorbedingungen in Java ohne Assertions realisieren? Nennen Sie zwei unterschiedliche Herangehensweisen. Beschreiben Sie die Unterschiede mit Beispiel-Code unter Abwandlung eines der Codestücke aus der vorherigen Teilaufgabe.

*Das geht entweder unmittelbar mit einer unchecked Exception:*

```
  class MyAssertionException extends RuntimeException {}

  public class Veranstaltung {
    public void addStudent (Student s) {
            // check that this student is not registered
            if hoerer.contains(s) throw new MyAssertionException (
             "Student " + s + " already registered for this
                  course");
            hoerer.add(s);
      }
    }
```

*Oder alternativ mit einer zusätzlichen eigenen Assertion-Klasse:*

```
  class MyAssertionError extends Error {}
  class MyAssertion {
            public static void myAssert(bool condition, String message) {
                if !condition throw new MyAssertionError(message);
          }
    }

  public class Veranstaltung {
    public void addStudent (Student s) {
            // check that this student is not registered
```

```
            MyAssertion.myAssert(!hoerer.contains(s), “Student " ...);
            hoerer.add(s);
        }
    }
```

f) In Teilaufgabe e) haben Sie festgestellt, dass es sehr einfach ist DBC in Java zu realisieren, selbst ohne spezielle Sprachunterstützung. Warum glauben Sie, haben sich die Entwickler von Java trotzdem dazu entschlossen, die Sprache um Assertions zu erweitern?

- Lesbarkeit / Verständlichkeit (Im Gegensatz zu Exceptions klare Bedeutung)
- *Aus Effizienzüberlegungen, damit der Compiler* assertions von *normalem Programmcode unterscheiden und wenn eine* entsprechende Option *gesetzt ist, die Ausführung von Assertions unterbinden kann.*
- *Hintergrund: Sie sahen assertions als eine Technik, die nur während Entwicklung und Test eingesetzt werden sollte (http://java.sun.com/developer/technicalArticles/JavaLP/assertions/):*
  - *Note that assertions can be compiled out. In languages such as C/C++, this means using the preprocessor. … . This causes the C/C++ preprocessor to ignore all assertions, instead of deleting them manually.*
  - *In other words, this is a requirement for performance reasons.* You should write assertions into software in a form that can be optionally compiled. *Thus, assertions should be executed with the code only when you are debugging your program -- that is, when assertions will really help flush out errors. You can think of assertions as a uniform mechanism that replaces the use of ad hoc conditional tests.*

## **Aufgabe 2.** *Split Objects* (8 Punkte)

Zur weiteren Flexibilisierung des Studiums hat die Kultusministerkonferenz eine beitragsorientierte Reform vorgeschlagen. Als ersten Schritt zur Verwaltung der Reform-Studierenden existiert ein Eclipse-Projekt *StudienPlan* (in ihrem SVN-Repository), welches die Basis-Studiengänge modelliert.
Die Klassen enthalten Operationen zur Abfrage der Kostenbeiträge und der Beschreibung eines Studiengangs.

a) Es soll nun möglich sein, zu jedem der obigen Basis-Studiengängen folgende Studien-Optionen zusätzlich zu belegen:

   a. Tutorial Option mit Beitragszuschlag 700.00 €
   b. Crisis Hotline Option mit Beitragszuschlag 200.00 €
   c. Advertising Medium Option mit Beitragszuschuss von 75.00 €
   d. Gratuity Option[1] mit Beitragszuschlag 16000.00 €

   Fügen Sie Klassen für die Studien-Optionen dem existierenden Modell hinzu. Die Klassen sollen Operationen erhalten, die die Kostenbeiträge und eine sinnvolle Beschreibung lie-

[1] Soll diskret behandelt werden und daher nicht in der Beschreibung auftauchen.

fern. Erweitern Sie das Modell (mit Hilfe eines Entwurfsmusters) so, das folgende Anforderungen erfüllt sind:

a. Jeder Studiengang soll mit beliebig vielen der obigen Optionen kombinierbar sein.
b. Optionen können auch mehrfach (beitragspflichtig) belegt werden.
c. Ein Basis-Studiengang und eine Option bilden selbst wieder einen Studiengang.
d. Die Berechnung der Kostenbeiträge wird abhängig von dem gewählten Basisstudiengang und der Optionen durchgeführt. Gleiches gilt für die Beschreibung.
e. Das spätere Hinzufügen von Basis-Studiengängen und Studien-Optionen soll möglich sein, ohne Anpassungen an bereits existierenden Code vornehmen zu müssen.
f. Es ist nicht notwendig auf bestimmte Komponenten des Studiengangs zugreifen zu können, das Endergebnis (Gesamtbeschreibung des kombinierten Studiengangs und Gesamtbeiträge) ist ausreichend. z. B.:
`Bachelor-Studium, Tutorial Option, Advertising Medium Option – 1125.00€`

### Anwendung des Decorator Entwurfsmusters

```
public abstract class StudienOption extends Studiengang {

	Studiengang decorated;

	public StudienOption(Studiengang decorated) {
		this.decorated = decorated;
	}
}
public class TutorialOption extends StudienOption {
	public TutorialOption(Studiengang decorated) {
		super(decorated);
	}
	@Override
	public String getDescription() {
		return decorated.getDescription()+ ", Tutorial Option";
	}
	@Override
	public double cost() {
		return decorated.cost() + 700.00;
	}
}
public class CrisisHotlineOption extends StudienOption {
	public CrisisHotlineOption(Studiengang decorated) {
		super(decorated);
	}
	@Override
	public String getDescription() {
		return decorated.getDescription()+ ", Crisis Hotline Option";
	}
	@Override
	public double cost() {
		return decorated.cost() + 200.00;
	}
}
public class AdvertisingMediumOption extends StudienOption {
	public AdvertisingMediumOption(Studiengang decorated) {
		super(decorated);
```

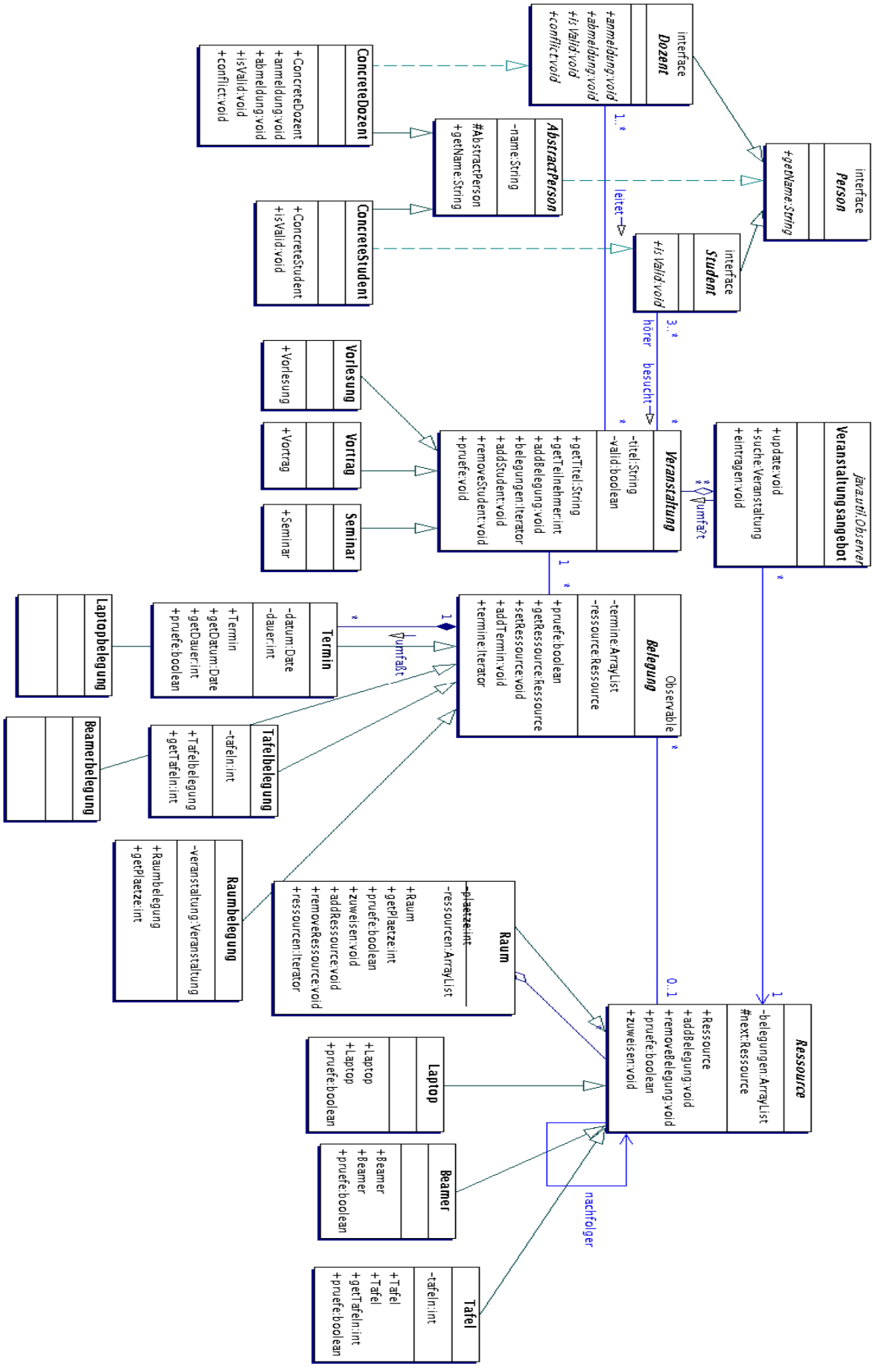
```
        }
        @Override
        public String getDescription() {
          return decorated.getDescription()+ ", Advertising Medium Option";
        }
        @Override
        public double cost() {
              return decorated.cost() - 75.00;
        }
}
public class GratuityOption extends StudienOption {
        public GratuityOption(Studiengang decorated) {
              super(decorated);
        }
        @Override
        public String getDescription() {
              return decorated.getDescription();
        }
        @Override
        public double cost() {
              return decorated.cost() + 16000.00;
        }
}
```

b) Schreiben Sie ein Programm, das einen Basis-Studiengang mit drei Optionen kombiniert und geben Sie dann die Studienbeiträge und die Beschreibung des resultierenden Studiengangs auf die Konsole aus.

```
public class StudienDemo {
        public static void main(String[] args) {
              Studiengang s =  new GratuityOption(
                                    new CrisisHotlineOption(
                                      new AdvertisingMediumOption(
                                        new Bachelor()
                                      )
                                    )
                                  );
              System.out.println(
                  String.format("Ihr Studiengang: %s", s.getDescription()));
              System.out.println(
                  String.format("Ihr Studienbeitrag: %.2f €", s.cost()));
        }
}
```

c) Gehen Sie nun davon aus, dass jeder Basis-Studiengang mit nur einer einzigen Studien-Option kombiniert werden darf (ansonsten gelten die Anforderungen aus Teilaufgabe a). Würde sich dadurch ein anderes Entwurfsmuster anbieten, um die Aufgabenstellung zu lösen? Begründen Sie stichpunktartig Ihre Antwort.

Das Strategie-Entwurfsmuster ist bei Kombination mit nur einer Option sinnvoller, da man auf den Basis-Studiengang und die Option zugreifen kann, ohne Verschachtelungen traversieren zu müssen. Weiterhin ist keine Vererbungsbeziehung notwendig und durch die flachere Struktur eine geringere Komplexität gegeben.


interface
Dozent
+anmeldung:void
+abmeldung:void
+isValid:void
+conflict:void
interface
Person
+getName:String
interface
Student
+isValid:void
ConcreteDozent
+ConcreteDozent
+anmeldung:void
+abmeldung:void
+isValid:void
+conflict:void
AbstractPerson
-name:String
#AbstractPerson
+getName:String
ConcreteStudent
+ConcreteStudent
+isValid:void
leitet
hörer
besucht
Vorlesung
+Vorlesung
Vortrag
+Vortrag
Seminar
+Seminar
Veranstaltung
-titel:String
-valid:boolean
+getTitel:String
+getTeilnehmer:int
+addBelegung:void
+belegungen:Iterator
+addStudent:void
+removeStudent:void
+pruefe:void
Umfaßt
java.util.Observer
Veranstaltungsangebot
+update:void
+suche:Veranstaltung
+eintragen:void
Laptopbelegung
Termin
-datum:Date
-dauer:int
+Termin
+getDatum:Date
+getDauer:int
+pruefe:boolean
Beamerbelegung
Tafelbelegung
-tafeln:int
+Tafelbelegung
+getTafeln:int
Raumbelegung
-veranstaltung:Veranstaltung
+Raumbelegung
+getPlaetze:int
Observable
Belegung
-termine:ArrayList
-ressource:Ressource
+pruefe:boolean
+getRessource:Ressource
+seRessource:void
+addTermin:void
+termine:Iterator
Raum
-plaetze:int
-ressourcen:ArrayList
+Raum
+getPlaetze:int
+pruefe:boolean
+zuweisen:void
+addRessource:void
+removeRessource:void
+ressourcen:Iterator
Ressource
-belegungen:ArrayList
#next:Ressource
+Ressource
+addBelegung:void
+removeBelegung:void
+pruefe:boolean
+zuweisen:void
Laptop
+Laptop
+Laptop
+pruefe:boolean
Beamer
+Beamer
+Beamer
+pruefe:boolean
Tafel
-tafeln:int
+Tafel
+Tafel
+getTafeln:int
+pruefe:boolean
nachfolger