Übungen zur Vorlesung

Softwaretechnologie

-Wintersemester 2009/2010-

Dr. Günter Kniesel, Pascal Bihler

Übungsblatt 11

Zu bearbeiten bis: 17.01.2010

Bitte fangen Sie **frühzeitig** mit der Bearbeitung an, damit wir Ihnen bei Bedarf helfen können. Checken Sie die Lösungen zu den Aufgaben in Ihr SVN-Repository ein, Diagramme als VP-Dateien, Texte als Textdatei. Fragen zu Übungsaufgaben/Vorlesung können Sie auf der Mailingliste swt-tutoren@lists.iai.uni-bonn.de, bzw. swt-vorlesung@lists.iai.uni-bonn.de stellen.

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

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

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.*

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.

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).

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.

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?

Aufgabe 2. *Stukturierung von OO-Modellen* (4 Punkte)

In der Vorlesung wurden gute und schlechte Designentwürfe gegenübergestellt. Überlegen Sie sich für die folgenden Prinzipien zur Strukturierung von OO-Modellen je ein Negativ-Beispiel und zeigen Sie, wie es mit einem „korrekten“ Design aussehen würde:

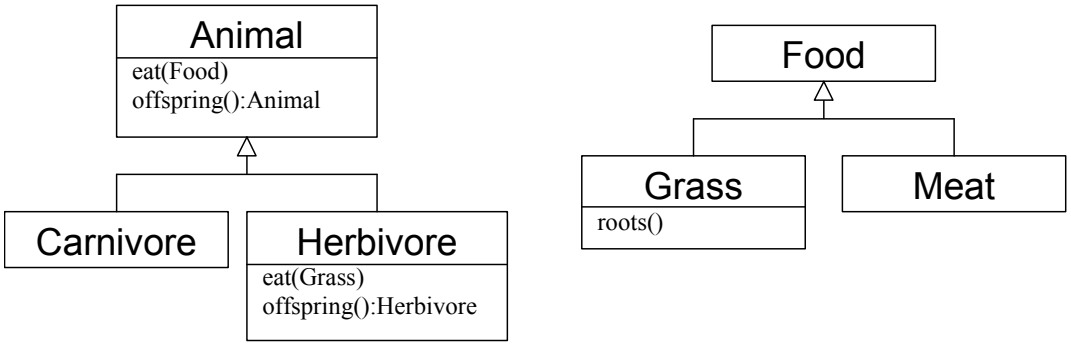
- Abhängigkeiten
- Fehlender Ersetzbarkeit

Beschreiben Sie die Strukturen in UML 2.0 und begründen Sie ihre Wahl des Beispiels und den Vorteil Ihrer verbesserten Lösung.

Tipp: Man findet einige Beispiele falscher Entwürfe im JDK, insbesondere zum Thema „fehlende Ersetzbarkeit“. Sie können sich aber auch gern eigene Beispiele ausdenken.

Aufgabe 3. *Strukturierung von OO Modellen* (7 Punkte)

Gegeben seien folgende Klassen, die die Intuition ausdrücken sollen, dass Pflanzenfresser nur Pflanzen essen und auch als Nachkommen nur Pflanzenfresser zeugen:



a) Beschreiben Sie, welche Probleme auftreten könnten, wenn die Methode `eat(Grass)` aus der Klasse `Herbivore` die Methode `eat(Food)` aus `Animal` *überschreiben* würde. Gehen Sie dabei von einem Aufruf der `eat`-Methode auf einer Variablen vom Typ der Oberklasse aus.

b) Nehmen Sie nun an, dass die Methode `eat(Grass)` aus `Herbivore` die Methode `eat(Food)` aus `Animal` *überlädt*. Tritt das vorherige Problem immer noch auf? Warum (nicht)?

c) Sehen Sie bei der Methode `offspring()`, die einen Nachkommen eines Tieres liefert, ein ähnliches Problem? Gibt es einen Unterschied zu `eat(…)`?

d) Wie verhält sich Java im Hinblick auf obiges Beispiel (`eat(…)` und `offspring()`)?

	…überschreibt…	…überlädt…	
a) Herbivore.eat(Grass)…			…Animal.eat(Food)
b) Herbivore.offspring()…			…Animal.offspring()

Aufgabe 4. *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 €

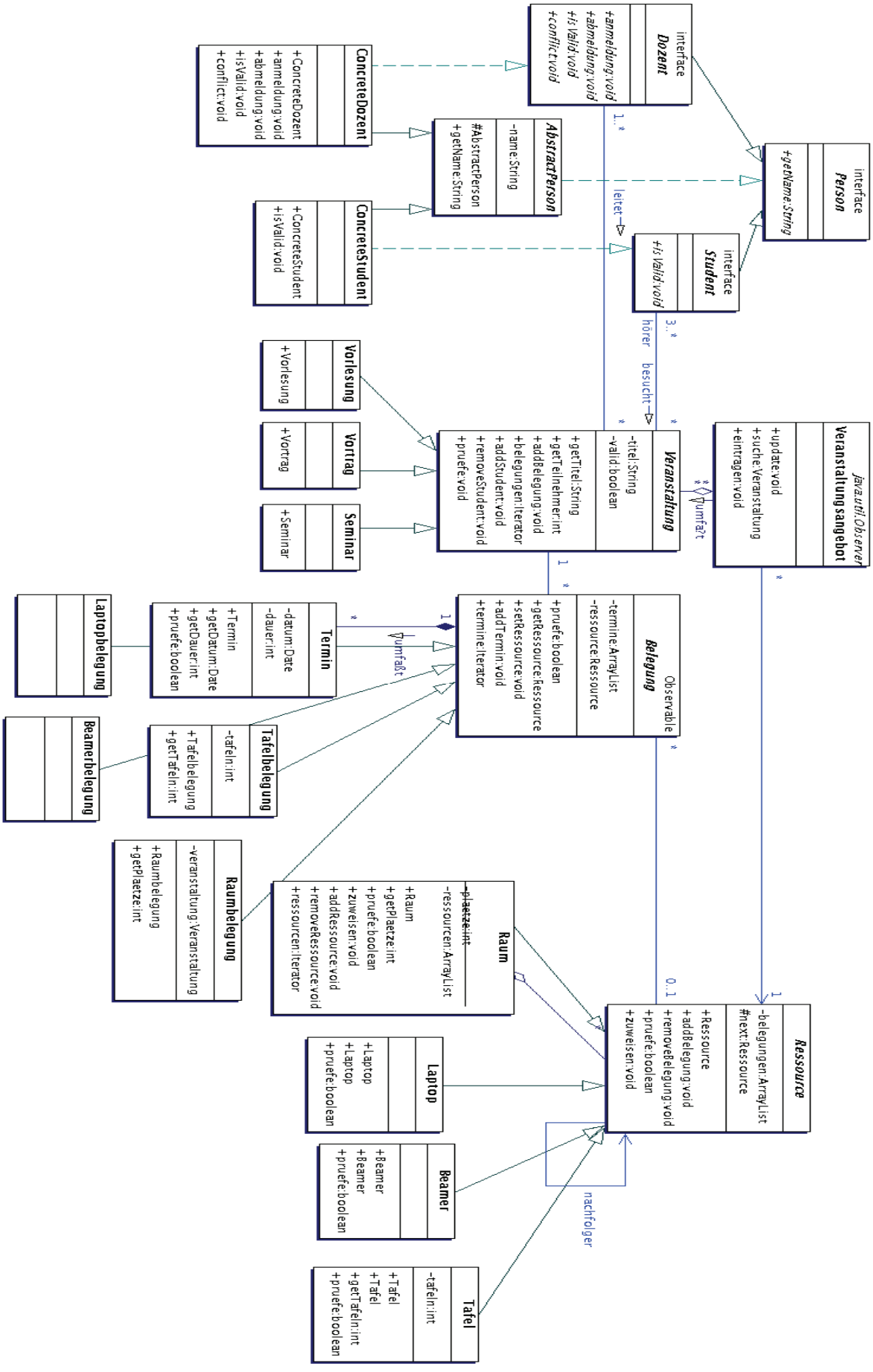
   Fügen Sie Klassen für die Studien-Optionen dem existierenden Modell hinzu. Die Klassen sollen Operationen erhalten, die die Kostenbeiträge und eine sinnvolle Beschreibung liefern. Erweitern Sie das Modell (mit Hilfe eines Entwurfsmusters) so, das folgende Anforderungen erfüllt sind:

   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 bereits existierenden Code anpassen zu müssen.
   f. Es ist <u>nicht</u> 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€`

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.

c) Gehen Sie nun davon aus, dass jeder Basis-Studiengang mit nur einer <u>einzigen</u> 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.

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


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
java.util.Observer
Veranstaltungsangebot
+update:void
+suche:Veranstaltung
+eintragen:void
umfaßt
Observable
Belegung
-termine:ArrayList
-ressource:Ressource
+pruefe:boolean
+getRessource:Ressource
+setRessource:void
+addTermin:void
+termine:Iterator
Termin
-datum:Date
-dauer:int
+Termin
+getDatum:Date
+getDauer:int
+pruefe:boolean
Laptopbelegung
Beamerbelegung
Tafelbelegung
-tafeln:int
+Tafelbelegung
+getTafeln:int
Raumbelegung
-veranstaltung:Veranstaltung
+Raumbelegung
+getPlaetze:int
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