

Dies ist keine „Musterlösung“, sondern eine gute von vielen möglichen Lösungen. Kommentare, die nicht Teil der Lösung sind, sind kursiv gesetzt.

Aufgabe 6-1:

Die Beispiele stammen aus dem eRezept-Transport von Aufgabe 3-2.3

a.

Bereichsklassen sind Klassen, die Konzepte des Anwendungsbereichs repräsentieren und die Sprache der Anforderungsanalyse.

Lösungsklassen sind Klassen, welche die Umsetzung auf technische Ebene repräsentieren. Hierbei kann es sein, dass einzelne Bereichsklassen nicht direkt 1:1 umgesetzt werden oder gar ganz aufgelöst werden und in andere integriert werden. Es gibt auch viele Lösungsklassen, die nie als Bereichsklassen aufgetaucht sind, weil sie nur für die technische Umsetzung gebraucht werden (von StringBuffer, zu Datenbankverbindungspools kann man sich hier viel vorstellen).

Beispiel:

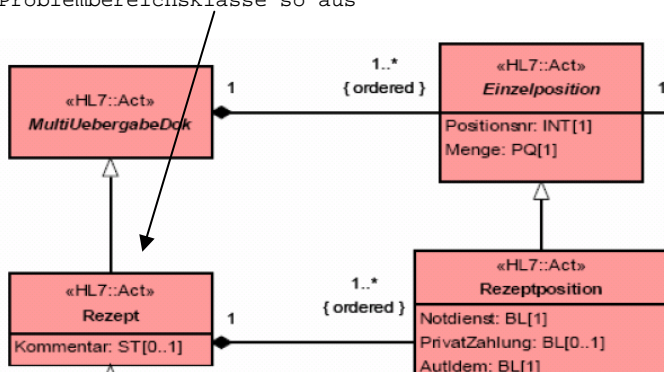
- eR-Ticket als Bereichsklasse zur Verwaltung von Patientenrechten (Info, Sek.eR und L.eR) wird als Lösungsklasse durch eine HashMap von Schlüssel-Wert-Paaren (Info) und zwei Bytearrays zur Speicherung des Sessionschlüssels und der Logging-Informationen umgesetzt.
- Das eRezept hat in unserer Lösung von UB03 eine Methode enc(). Es wäre schön ziemlich faszinierend, wenn diese Aufgabe in der Lösungswelt nicht viel eher von einem Kryptographiemodul übernommen würde und deshalb gar nicht als Methode im eRezept auftaucht.

Häufig ist diese technische/programmiertechnische Umsetzung während der Analyse auch noch nicht klar. Man denke nur an die eGK selbst, welche als Lösungsklassen durch eine SIM-Karte in Hardware umgesetzt wird.

b.

Geschäftsobjekte (dieser Begriff hat sich statt Geschäftsklassen im Deutschen durchgesetzt) beschreiben Daten-beinhaltende Klassen, meist Abbildungen realer Entitäten aus der Anwendungswelt. Beispiel: eRezept.

Achtung: Ein Geschäftsobjekt ist nicht unbedingt nur eine Problembereichsklasse. Auch in der Lösungswelt gibt es technische Umsetzungen von Geschäftsobjekten. Beim eRezept z.B. sieht die Problembereichsklasse so aus



Als Lösungsklasse vielleicht aber eher so:

```
public class Rezept {
    int version;
    String kommentar;
    List positionen;
}
```

version ist eine Variable, deren Wert nur in der Lösung interessant ist und die Liste hätten wir uns vielleicht auch eher in zwei unterschiedliche aufgeteilt vorgestellt (eine für Einzelpositionen und eine für Rezeptpositionen).

Zugangsklassen sind die Klassen, mit denen Akteure in der Regel Zugriff auf das System bekommen. Das sind nicht nur Klassen aus der Benutzungsoberfläche, sondern auch API-Klassen oder allg. Schnittstellenklassen, die z.B. den Zugriff auf die interessanten Geschäftsobjekte ermöglichen. Beispiel: Fällt leider schwer hier, da keine konkreten Klassen angegeben werden, aber die eGK selbst hat ein Stellvertreterobjekt zum Schreiben auf die Karte; dies ist eine reine Schnittstellenklasse,

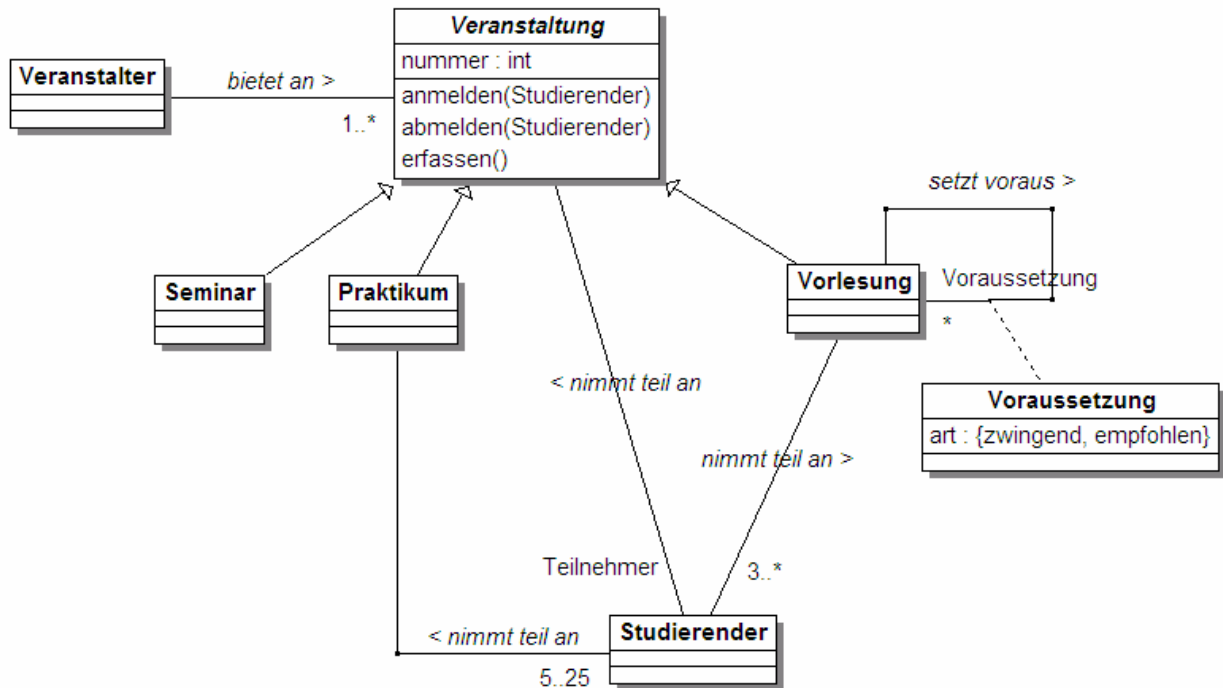
also eine Zugangsklasse.

Steuerklassen sind vor allem durch ihren dynamischen Anteil wichtig: Sie sorgen für den zentralen Ablauf einer Operation. In einem Sequenzdiagramm gehen z.B. die meisten Pfeile von ihnen aus. Sie sind meist im System verborgen.

Beispiel: PVS oder AVS.

Aufgabe 6-2:

1.a. Hinweise: Veranstaltung ist abstrakt, deshalb der kursiv gesetzte Name. „nimmt teil an“ wird in den Unterklassen überschrieben. Die Assoziationsklasse Voraussetzung gehört zu einer reflexiven Assoziation von Vorlesung. Teilnehmer ist die Rolle von StudierendeR in einer Veranstaltung.



Achtung: Eine empfohlene Voraussetzung kann man nicht als Unterklasse von Vorlesung definieren. Denn es mag ja Vorlesungen geben, welche sowohl empfohlene Voraussetzung, als auch Zwingende Voraussetzungen für unterschiedliche andere Vorlesungen sind. Lieber wie in obiger Lösung mittels einer Assoziation zwischen Vorlesungen, die diese Voraussetzungsbeziehung darstellt. Alternativ zu der Assoziationsklasse kann man übrigens natürlich auch zwei unterschiedliche Assoziationen verwenden: Eine für empfohlene und eine für zwingende.

1.b.

Viele **Substantive** sind hier eindeutig die Klassen, man muss allerdings Singular und Plural auseinander halten. *Klassennamen sind nahezu immer im Singular zu halten!* Es gibt Substantive, die nicht zu Klassen innerhalb der Analyse werden, nämlich die Phasen und Zeitangaben, wozu auch „Prüfung“ gezählt wurde (Wir haben diese als Ereignis-Substantive interpretiert). Diese Phasen und Zeitangaben werden sicherlich in der Implementierung wieder auftauchen, bei der Analyse würde man aber ein Zustandsdiagramm beilegen, was diese Aspekte deutlich besser einfangen kann (Aufgabenteil 2 ;-). Teilnehmer ist eine Rolle von Studierenden in einer Veranstaltung. Die „Reihe von Veranstaltungen“ ist keine Bereichsklasse, sondern weist darauf hin, dass das zu bauende System auch mit diesem Anwendungsklassendiagramm verbunden werden muss (z.B. kennt normalerweise das System alle Veranstaltungen eines Semesters). Genauso wie die Pluralangabe („Teilnehmer“) drückt sich dies in der Multiplizität der Assoziationen aus. Das „ist“ in b) wurde zur Vererbung, das „sein“ in j) aber nicht.

Im Tutorium haben wir letzteres Problem auch schon mal diskutiert. Einfach nur weil da ein „Sein“ steht, sollte man noch nicht eine Vererbungsbeziehung einbauen. Diese bringt nämlich viele Einschränkungen mit sich und ist nicht nützlich, wenn Objekte ihren Typ später mal ändern müssen. Wer sich einmal ordentlich mit der Frage auseinandersetzen möchte, dem sei von Martin Fowler „Dealing with Roles“ martinfowler.com/apsupp/roles.pdf empfohlen.

Attribute: Nur „Nummer“ und der von Abbott kaum zu entdeckende „Typ“ in Voraussetzung.

Operationen: Die Operationen sind gemäß Abbott klar erkennbar. Methoden von Vorlesung wurden in die Oberklasse geschoben.

Assoziationen: „Voraussetzung“ ist eine Assoziation, hier als Assoziationsklasse. Abbott hilft hier wenig. Keine der „hat“ wurde zur tatsächlich Aggregation: In e ist es die Umkehrung von „nimmt teil an“, in f die Beziehung zu den Zuständen eines Objekts (etwas was Abbott gar nicht thematisiert).

Multiplizitäten/Constraints/Rollen: S.o. bei Substantiven.

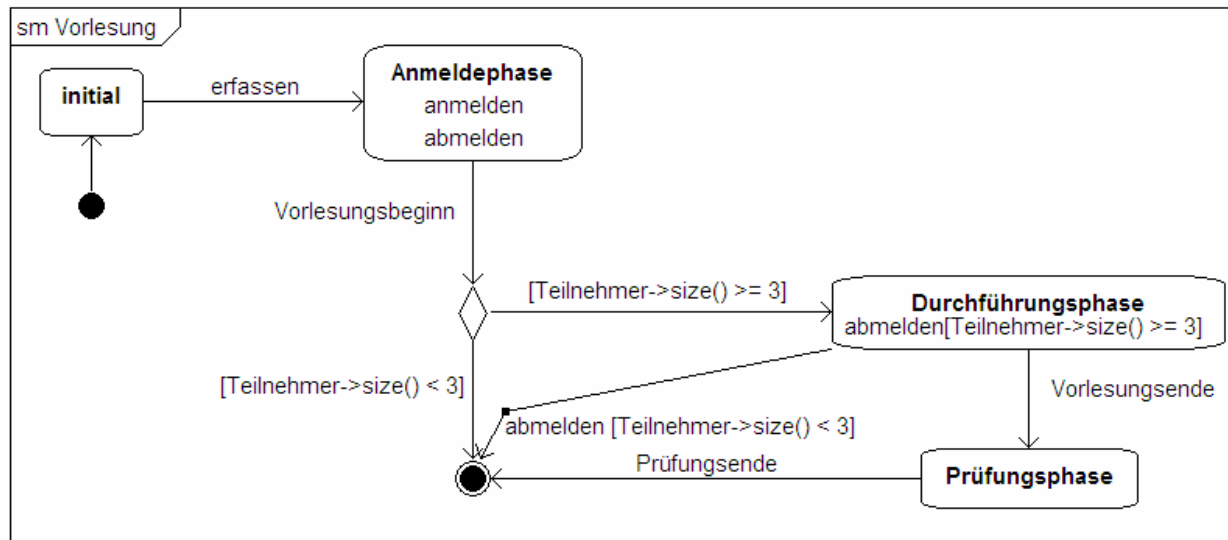
Fazit: Es hilft nur oberflächlich und meist gibt es genauso viele Ausnahmen und Uneindeutigkeiten wie klare Fälle. Aber es bewahrt vor dem Leeren-Blatt-Syndrom.

1.c. Grundlage ist der Lösungshinweis zu 5-2: erfassen() entspricht „Veranstaltung eingeben“ und anmelden() entspricht „Vorlesungstermin auswählen“. Man müsste hier eine Vereinheitlichung der Terminologie beginnen und die Diagramme anpassen.

2. X-beginn und X-ende sind Zeitereignisse, erfassen, anmelden und abmelden Operationen des Vorlesungsobjektes. Man merkt hierbei auch, dass Zustandsdiagramme keine guten Möglichkeiten bieten Ereignisse als ungültig zu deklarieren. Z.B. sollte es nicht möglich sein, sich in der Durchführungsphase noch anzumelden, was sich aber mit dem bisherigen Analyseklassendiagramm nicht modellieren lässt. Dementsprechend bin ich nicht ganz so zufrieden mit der untenstehenden Lösung.

Die folgende Lösung führt auch noch ein neues syntaktisches Element ein: Das unausgefüllte Karo (Diamond auf Englisch) entspricht einer Auswahl in Abhängigkeit von Guards. Die wegführenden Transitionen (bei uns die beiden Prüfungen auf die Anzahl der Teilnehmer) dürfen wie Initialzustände keine Trigger haben, da sie ja durch die Trigger an den eingehenden Transitionen (bei uns die Transition mit Vorlesungsbeginn als Ereignis) bereits ausgelöst wurden.

Die Notation `->size()` ist eine Operation aus OCL und liefert zu einer Assoziation die Anzahl der Objekte, die gerade assoziiert sind.



Achtung: Abmelden und Anmelden sind zwar Ereignisse, verändern, aber den internen Zustand der Veranstaltung (nicht schön), d.h. wenn der Guard geprüft wird nach dem Ereignis abmelden, dann ist die Teilnehmer-Zahl schon reduziert.

Häufig falsch gemacht: Verwendung von Ereignissen, Aktionen, Attributen die in der Vorlesung gar nicht vorkommen.

Aufgabe 6-3:

Das Vorgehen ist zunächst sehr Objekt- und Klassenorientiert. Damit werden zum einen Datenrepräsentationen stark repräsentiert, zum anderen Interaktionen zwischen den Daten beinhaltenden Objekten. Dies führt meist zu einer datenflussorientierten, Objektnetz-ähnlichen oder ablagebasierten Architektur (siehe Architekturstile später). Dies sind typischerweise Informationssysteme. Weniger typisch ist dies für Realzeitsysteme (mit hohem Kontrollflussanteil) oder eingebettete Systeme (wo es viele externe Vorgaben gibt).