

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

Aufgabe 9-1:

1. Siehe Folie 10 im Chapter 9 "Object Design: Specifying Interfaces". Der einzige Unterschied ist die Sichtbarkeit bei protected (was die Standardsichtbarkeit ist), welche bei Java sowohl Klassen als auch Paketsichtbarkeit ist. Pakete im eindeutig definierten Java-Sinne gibt es in UML nicht.
2. Invarianten kann man z.B. einsetzen für:
 - * genauere Typangabe, z.B. „i >= 0“ statt nur „int i“
 - * Angabe der invarianten Beziehung zwischen zwei Attributen einer Klasse, z.B. count <= max
 - * Angaben der Beziehungen auch über Klassengrenzen hinaus*Siehe auch Vorlesungsfolie 22 im Chapter 9 "Object Design: Specifying Interfaces"*
3. Assoziationsklassen, Vererbung und genauer spezifizierte Assoziationen bieten ER-Modelle nicht. Natürlich ist auch die Syntax verschieden, insbesondere die andere Schreibweise der Kardinalitäten/Multiplizitäten.

Aufgabe 9-2:

Machen Sie sich bitte klar, wozu OCL überhaupt benötigt wird: „Nacktes“ UML ist nicht mächtig genug für genauere Spezifikationen, die sonst im Code verloren gehen würden, wenn sie überhaupt implementiert werden. Der Schritt zwischen Modell und Code wird verkleinert, vielleicht sogar teilweise automatisierbar. Zudem geben OCL-Konstrukte Testkriterien an.

1.a. Lösung ist die Aufgabenbearbeitung eines Studenten im Rahmen einer abgegebenen Klausur.

1.b. punktanzahl() berechnet die Summe der punktzahl der Aufgaben dieser Klausur. Es ist also die maximal erreichbare Punktzahl dieser Klausur.

1.c. Nein, denn jeder Student kann nur eine Klausur schreiben. Das widerspricht der Studienordnung als Bereichsanforderung. Das Modell wurde absichtlich so „simpel“, damit die OCL-Ausdrücke machbar bleiben. Sauber wäre es, mehrere Klausuren möglich zu machen, wobei bestanden und zugelassen dann Werte der Assoziation von Student zur Klausur wären.

2.a. Aufgaben haben mindestens einen Punkt.

2.b. Die Anzahl der Lösungen eines Studenten ist gleich der Anzahl der Aufgaben der Klausur.

2.c. Alle Studenten, die die Klausur bestanden haben, haben bei mindestens einer Lösung der Klausur mindestens 1 Punkt. Oder: Man kann nicht mit 0 Punkten bestehen. *Man beachte den Term* 1.aufgabe.klausur = self, *der notwendig ist, damit der Bezug der Lösungen zur Klausur erhalten bleibt.*

3.a. context Klausur inv: aufgaben->exists (a | a.punktzahl = 1)

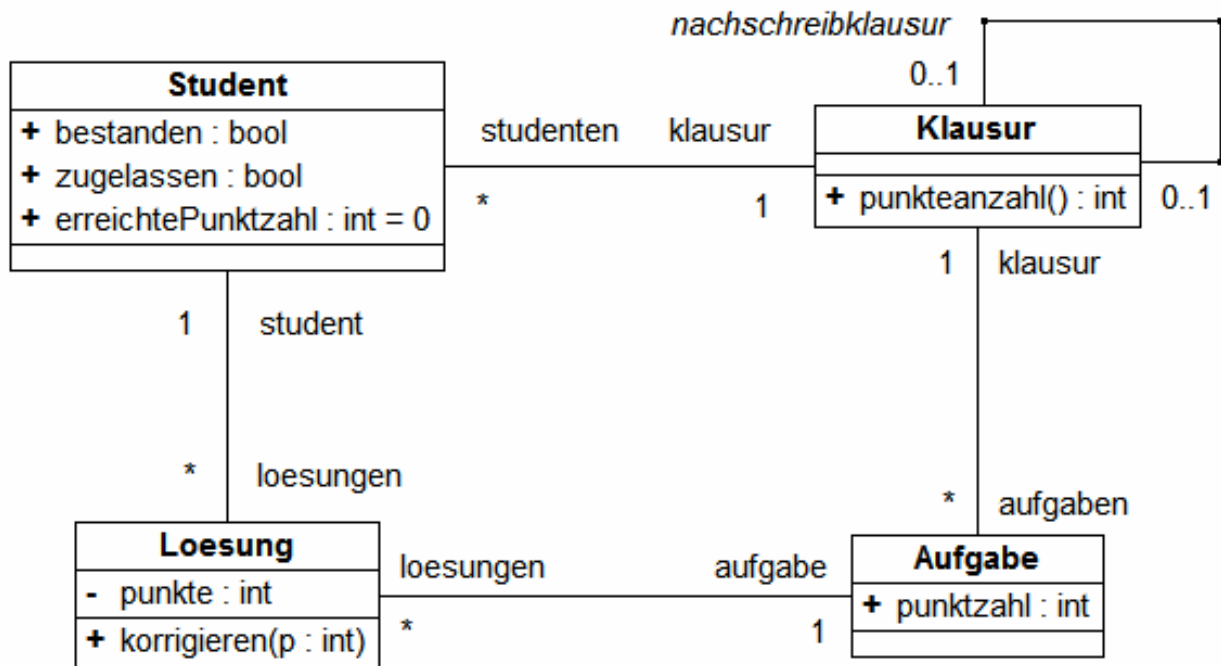
3.b. context Klausur inv: Klausur.allInstances->exists(k | k.nachschreibklausur->notEmpty() and
k.nachschreibklausur = self) implies self.nachschreibklausur->isEmpty()

ODER:

context Klausur inv: nachschreibklausur->notEmpty()
implies nachschriebklausur.nachschreibklausur->isEmpty()

3.c. context Student inv:
self.zugelassen implies klausur.aufgaben->forall (a |
self.loesungen->exists(l | 1.aufgabe = a))

4.a.



Die Methode `korrigieren()` wurde zu `Lösung` ergänzt und zu `Student` die `erreichtePunktzahl` hinzugefügt. Achtung: Für Lösung 4.c. muss `erreichtePunktzahl` die `=0` Initialisierung haben.

4.b. context `Lösung::korrigieren(p: int)` pre:
`student.zugelassen and`
`p <= aufgabe.punktzahl and`
`p >= 0`

4.c. context `Lösung::korrigieren(p: int)` post:
`punkte = p and`
`student.erreichtePunktzahl = student.erreichtePunktzahl@pre - punkte@pre + p and`
`student.erreichtePunktzahl <= student.klausur.punkteanzahl()`

Die letzte Bedingung kann man weglassen, wenn `punkteanzahl()` korrekt definiert ist. `-@pre.punkte` wurde hinzugefügt, weil die Methode `korrigieren()` mehrmals auf eine `Lösung` aufgerufen werden könnte (was durchaus realistisch ist, siehe z.B. Korrekturen nach einer Klausureinsicht).

Man beachte, dass hier nicht das `result`-Konstrukt benötigt wurde, weil `korrigieren()` keinen Rückgabewert liefert. Dieses Konstrukt ist dennoch wichtiger Bestandteil von OCL!

Aufgabe 9-3:

Für die Lösungsmöglichkeit Attribut:

```
context Troll inv:
  keulen->size >= 2 implies istToll
```

Für die Lösungsmöglichkeit Methode:

```
context Troll::istToll() post:
  keulen->size >= 2 implies result
```

(So lesen: "Wenn die Anzahl der Keulen größer gleich 2 ist, dann ist das Ergebnis von `istToll()` wahr")

Aufgabe 9-4:

1. Es sind zwar beides Unterklassen von `Throwable`, aber `Exceptions` sind für erwartete, wenn auch evtl. ungünstige Sondersituationen, die explizit auftreten können, während `Errors` nicht erwartet werden, z.B. ein Fehler in der Virtual Machine. Man kann `Errors` zwar auch in `catch`-Klauseln fangen, aber in der Regel sind `Errors` so definiert, dass eh „alles zu spät“ ist. Da `Assertions` nur unerwarteterweise nicht zutreffen, nimmt man also `Errors`.

2. Für `Exceptions` spräche, dass die Software in einem Versagensfall ordentlich reagieren kann. Wenn aber dieser Versagensfall nicht eintreffen wird (wegen ausreichenden Testens) oder die Software gar nicht sinnvoll reagieren kann (einfach ein „interner Fehler“ hilft niemandem), ist ein `Error` angesagt.

3. Sollte man nicht tun mit Assertions, es sei denn, die Parameterwerte sind wirklich unerwartet, z.B. wenn man (in alle Zukunft) alle aufrufenden Stellen kennt und diese auf jeden Fall für korrekte Parameter sorgen. Man würde Exceptions nehmen – so machen es auch viele API-Methoden.
4. Java erlaubt es uns auch neben den schlimmen Errors (bei denen keine Hoffnung auf Rettung mehr besteht) auch Exceptions als unerwartet zu klassifizieren, indem wir sie als RuntimeExceptions markieren. Solche Exceptions müssen von uns nicht deklariert und gefangen werden. Die Entscheidung zu dieser dritten Kategorie war wohl eine sehr pragmatische: Wenn man alle möglichen Sonderfälle wie NullPointerException, IllegalArgumentException etc. deklarieren und wieder bearbeiten müsste, dann wäre die Programmierung furchtbar umständlich. Als Programmierer, der selbst Exceptions an einer Methode anbringt, sollte man nämlich immer überlegen: Will ich den Aufrufer wirklich zwingen einen Catch-Block um seinen Aufruf machen zu müssen, oder kann der Aufrufer nicht vorher prüfen, ob der Aufruf scheitern würde (z.B. indem er auf Null prüft).