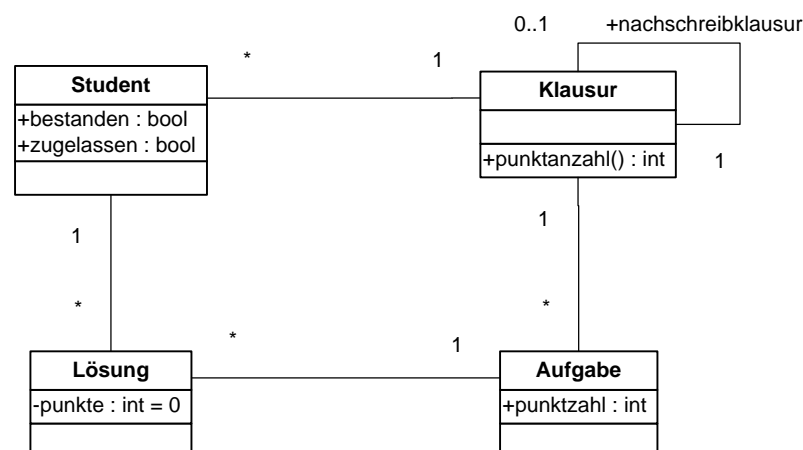


**Aufgabe 9-1: (Begriffe)**

1. Benennen Sie die Unterschiede bei der Spezifikation von Sichtbarkeiten (visibility) in UML zu denen in Java.
2. Wozu sind Invarianten geeignet und wo können sie eingesetzt werden? Nennen Sie einige überzeugende Beispiele.
3. Welche für die Objektimplementierung wichtigen Unterschiede gibt es zwischen einem ER-Modell (siehe Datenbanksysteme) und einem UML-Modell?

**Aufgabe 9-2: (OCL)**

In dem Prüfungsverwaltungssystem sollen auch die Klausurergebnisse und die Punktzahlen der Studierenden bei den Lösungen der einzelnen Aufgaben verwaltet werden. Folgendes UML-Klassendiagramm modelliert einen Teil der benötigten Daten. Die Rollennamen der Klassen in ihren Assoziationen entsprechen dabei dem kleingeschriebenen Klassennamen (evtl. im Plural), so dass sie hier nicht angegeben wurden; einzige Ausnahme ist die „nachschiebklausur“.



1. Schauen Sie sich das Klassendiagramm an und beantworten Sie folgende Fragen.
  - a. Was repräsentiert die Klasse *Lösung* Ihrer Interpretation nach?
  - b. Was berechnet *punktanzahl()* in der Klasse *Klausur*?
  - c. Ist dieses Modell geeignet für unser Prüfungsverwaltungssystem?
2. Drücken Sie auf Basis des Diagramms folgende OCL-Constraints natürlichsprachlich aus. *size* berechnet die Elementanzahl einer Menge oder Liste. *self* bezeichnet das Kontextexemplar.
  - a. `context Aufgabe inv: self.punktzahl > 0`
  - b. `context Student inv: self.loesungen->size = self.klausur.aufgaben->size`
  - c. `context Klausur inv: self.studenten->forall (s | s.bestanden = true implies s.loesungen->exists (l | l.punkte > 0 && l.aufgabe.klausur = self))`
3. Geben Sie OCL-Constraints an, die folgende Sachverhalte formalisieren. Schauen Sie auch in der OCL 2.0 Spezifikation nach, falls Ihnen Ausdrucksmittel fehlen.
  - a. In jeder Klausur gibt es mindestens eine Aufgabe mit genau einem Punkt.
  - b. Eine Nachschreibklausur hat die gleiche Punktzahl wie die ursprüngliche Klausur.
  - c. Pro zugelassenen Student gibt es für jede Aufgabe auch eine Lösung des Studenten.

4. Nun soll zusätzlich auch die Korrektur von Lösungen modelliert werden.

a. Erweitern Sie das Klassendiagramm um ein passendes Attribut und um eine Methode *korrigieren(...)* mit passender Signatur. *korrigieren(...)* wird aufgerufen, sobald der Dozent eine Lösung korrigiert hat. Es soll insbesondere damit möglich sein, dass

- das Attribut *punkte* in *Lösung* gefüllt wird. (Die Sichtbarkeit des Attributs darf aber nicht verändert werden!)
- die erreichte Punktzahl des Studenten aktualisiert wird.

Beschreiben Sie zunächst verbal, was *korrigieren()* genau leisten soll.

b. Geben Sie möglichst strenge Vorbedingungen für Ihre Methode *korrigieren()* in OCL an. Nutzen Sie die *pre*-Notation von OCL.

c. Spezifizieren Sie den Effekt (post condition) der Methode *korrigieren()* komplett in OCL. Nutzen Sie die *post*-Notation von OCL.

### Aufgabe 9-3: (Tolle Trolle in OCL)

Die Lösung zur Aufgabe 4-3 war nicht ganz befriedigend: Dass Trolle toll sind, wenn Sie mehr als eine Keule haben, ließ sich in UML-Klassendiagrammen nicht korrekt darstellen. Holen Sie dies nun nach, indem Sie OCL einsetzen.

### Aufgabe 9-4: (Assertions in Java)

Schauen Sie sich, falls Ihnen nicht ausreichend bekannt, die Konstrukte der Exceptions und der Assertions (ab Java 1.4) in Java an. Mit letzteren wurde die Möglichkeit eingeführt, Bedingungen (constraints) zur Laufzeit zu überprüfen.

1. Klären Sie den Unterschied zwischen einem *Error* und einer *Exception* in Java. Warum erzeugt eine unerfüllte Assertion einen *Error* und nicht eine *Exception*?

2. Was spricht dafür, zur Prüfung von Effekten und Invarianten *Exceptions* einzusetzen und was spricht eher für *Errors*, also zum Einsatz von *assert*?

3. Kann man das *assert*-Konstrukt benutzen, um in Methoden die übergebenen Parameterwerte auf die erfüllten Vorbedingungen zu prüfen? Oder würden Sie dafür eher das *throws*-Konstrukt der *Exceptions* nutzen? Oder gar nichts von beidem?