Softwarekonstruktion – Übung 1

Organisatorisches

Präsenzübungen

- Kurzvorstellung wesentlicher Fallstricke aus den Hausübungen (ca. 5-10 min)
- Bearbeitung der Aufgaben in Gruppen (ca. 60 min)
- Gemeinsames Besprechen der Aufgaben (ca. 20-25 min)
- Die Tutoren stehen für Fragen bei der Bearbeitung zur Verfügung, es ist aber **keine** Vorrechenübung

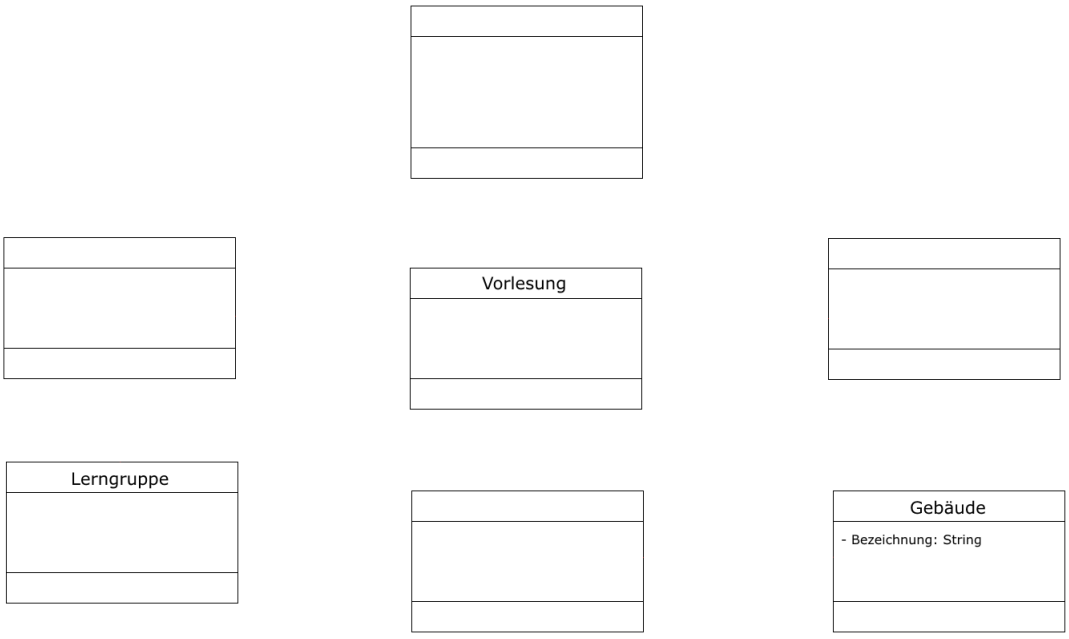
- Übungszettel:
  - Es wird insgesamt 6 Hausübungen mit insgesamt 100 Punkten geben.
  - Bei den letzten beiden Hausübungen handelt es sich voraussichtlich um Online-Übungen.
  - Die Abgabe erfolgt in Gruppen von 2-4 Teilnehmern.
  - Deadlines der Abgaben sowie Informationen zur Online-Übung sind der Webseite der Veranstaltung zu entnehmen.
- Mögliche Abgabe:
  - in den Übungen
  - Einwurf in die Briefkästen Nr. 49-51 (je nach Gruppe) in der Otto-Hahn-Str. 12
- Keine Abgabe:
  - per Mail oder Hauspost
  - persönlich am Lehrstuhl oder Sekretariat
- Rückgabe:
  - in der entsprechenden Übung
  - nicht abgeholte Übungszettel liegen im Sekretariat zur Abholung bereit
- Zum Erbringen der Übungsleistung:
  - Insgesamt sind mind. **50% der Gesamtpunkte** zu erreichen, dabei aber mind. **30% in der 1. Hälfte der Übungszettel** und **30% in der 2. Hälfte der Übungszettel** (d.h. mind. 50 Punkte in Summe, davon mind. 15 Punkte aus den Zetteln 1-3 und mind. 15 Punkte aus den Zetteln 4-6)
- **Abgabe von Duplikaten führt zu 0 Punkten für alle beteiligten Gruppen**

1 Object Constraint Language (OCL)

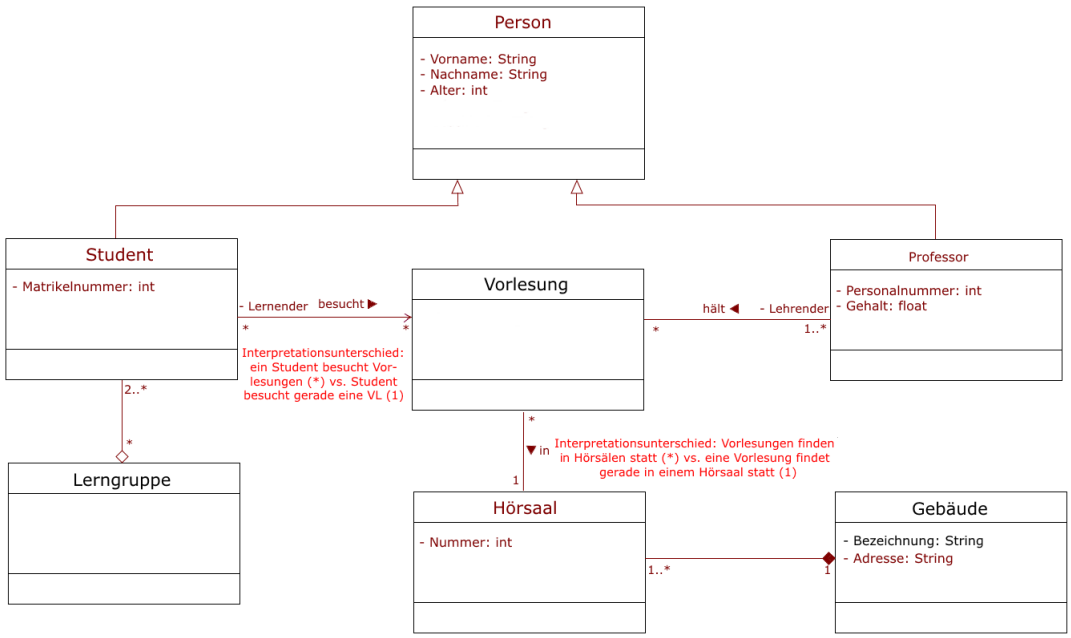
1.1 UML-Klassendiagramm

Ergänzen Sie das unten stehende UML-Klassendiagramm gemäß nachfolgender Beschreibung. Vervollständigen Sie dabei die Klassen *Person*, *Student*, *Professor*, *Vorlesung*, *Lerngruppe*, *Hörsaal* und *Gebäude* und verbinden diese entsprechend.

Verwenden Sie dabei Aggregationen, Kompositionen, Multiplizitäten und Leserichtungen. Nutzen Sie auch das Konzept der Vererbung. Geben Sie zudem zu allen Attributen passende Datentypen an.



1. Personen besitzen einen Vornamen, einen Nachnamen und ein Alter.
2. Studenten sind Personen, die außerdem eine Matrikelnummer besitzen.
3. Professoren sind ebenfalls Personen, allerdings besitzen sie aufgrund ihrer Anstellung eine Personalnummer und beziehen Gehalt, eine Matrikelnummer besitzen sie natürlich nicht.
4. Studenten besuchen Vorlesungen, Professoren halten Vorlesungen. Professoren sind dabei Lehrende, Studenten die Lernenden. Studenten können einsehen, welche Vorlesungen sie besucht haben, Vorlesungen hingegen haben keinen Zugriff auf die Listen ihrer Besucher.
5. Vorlesungen finden in Hörsälen statt. Jeder Hörsaal besitzt eine Nummer.
6. Ein Hörsaal gehört zu einem Gebäude.
7. Studenten können Lerngruppen bilden.



1.2 OCL Begriffe

Erläutern Sie stichpunktartig nachfolgende Begriffe im Zusammenhang mit der OCL. Beschreiben Sie dabei die Bedeutung der Begriffe und geben Sie entsprechende OCL-Schlüsselwörter an.

1. Klasseninvariante
2. Vorbedingung bzw. Nachbedingung

In der Vorlesung sowie der Übung wird OCL in der Version 2.4 behandelt. Die Spezifikation ist bei Bedarf unter `http://www.omg.org/spec/OCL/2.4/PDF/` zu finden.

Klasseninvariante: Eine Bedingung/ Einschränkung, die immer von allen Instanzen einer Klasse erfüllt werden muss. OCL-Schlüsselwort **inv**

Vorbedingung: Eine Bedingung, die erfüllt sein muss, bevor eine Operation ausgeführt werden kann. OCL-Schlüsselwort **pre**

Nachbedingung: Eine Bedingung, die nach dem Ausführen einer Operation erfüllt sein muss. OCL-Schlüsselwort **post**

1.3 Vordefinierte Operationen

Die OCL ist eine sehr mächtige Sprache, daher wird in der Veranstaltung eine dedizierte Teilmenge der in OCL vordefinierten Operationen betrachtet. Machen Sie sich mit den nachfolgenden Operationen vertraut, indem Sie stichpunktartig beschreiben, was die jew. Operationen berechnen.

Wenn ein Klassendiagramm gegeben ist, so wenden Sie die entsprechende Operation als gültige Invariante auf alle Klassen an!

- **logische Operatoren**
  - and, or, xor — Konjunktion (und), Disjunktion (oder) sowie Kontravalenz
  - not, implies — `not: logische Negation, implies: logische Implikation`
  - drücken Sie implies nur mit Hilfe der anderen logischen Operatoren aus: `a implies b ≡ not a or b. Achtung bei Wahrheitstabelle: nur a=true, b=false liefert (a implies b) = false`

- **Aufzählungstypen (enumeration types)**
  - *variable* = *enumeration name* :: *enumeration value* Enumerations sind Datentypen der UML, welche die Definition einer Reihe von Literalen als mögliche Werte erlauben. Für den Zugriff auf die Werte ist die :: Syntax zu verwenden.

- **Operationen auf Objekten**
  1. **boolesche Operatoren** (wenn *self* mit Objekt o vergleichbar ist)
     - =(o:T): Boolean — gibt true zurück, wenn *self* gleich dem Objekt o ist
     - <>(o:T): Boolean — gibt true zurück, wenn *self* ungleich dem Objekt o ist
     - <(o:T): Boolean — gibt true zurück, wenn *self* kleiner als o ist
     - <=(o:T): Boolean — gibt true zurück, wenn *self* kleiner gleich o ist
     - >(o:T): Booelan — gibt true zurück, wenn *self* größer als o ist
     - >=(o:T): Boolean — gibt true zurück, wenn *self* größer gleich o ist

- **Operationen auf Collections**

  1. **einfache Operationen**
     - → size(): Integer gibt die Anzahl aller Elemente der Menge zurück
     - → sum(): Real addiert alle Elemente der Menge
     - → isEmpty(): Boolean gibt true zurück, wenn Menge leer ist
     - → notEmpty(): Boolean gibt true zurück, wenn *Collection* mindestens ein Element enthält
  2. **Elementbezogene Operationen**
     - → includes(o:T): Boolean

       gibt true zurück, wenn o in Collection enthalten ist
     - → excludes(o:T): Boolean gibt true zurück, wenn o nicht in der *Collection* enthalten ist
     - → count(o:T): Integer

       zählt, wie oft o in der Menge enthalten ist
     - → isUnique(expr:OclExpression): Boolean

       gibt true zurück, wenn alle Objekte nach Anwendung der *expr* einzigartig in der Menge enthalten sind
  3. **Iterator-Ausdrücke**
     - → select(expr:OclExpression): Collection(T)

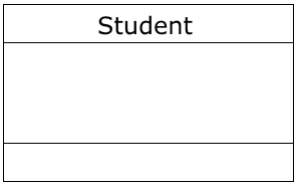
       gibt alle Elemente der C. zurück, welche die *expr* erfüllen
     - → collect(expr:OclExpression): Collection(T)

       gibt die Collection zurück, nachdem die *expr* auf alle Elemente angewandt wurde
     - → reject(expr:OclExpression): Collection(T) liefert eine Collection, die alle Elemente von *self* enthält, außer diejenigen, welche die *expr* erfüllen
     - → forAll(expr:OclExpression):Boolean

       gibt true zurück, wenn alle Elemente der Menge die *expr* erfüllen

- **Operationen auf Klassen**
  - classifier.allInstances(): Set(T)

```
gibt ein Set zurück, welches alle Instanzen des classifiers enthält
```



Es soll nicht mehr als 10.000 Studenten geben:

context Student

```
inv: self.allInstances() -> size() <= 10000
```

1.4 Erste Anwendung von OCL

Betrachten Sie das UML-Klassendiagramm über Veranstaltungshallen, wie beispielsweise die Dortmunder Westfalenhalle, in nachfolgender Abbildung 1.

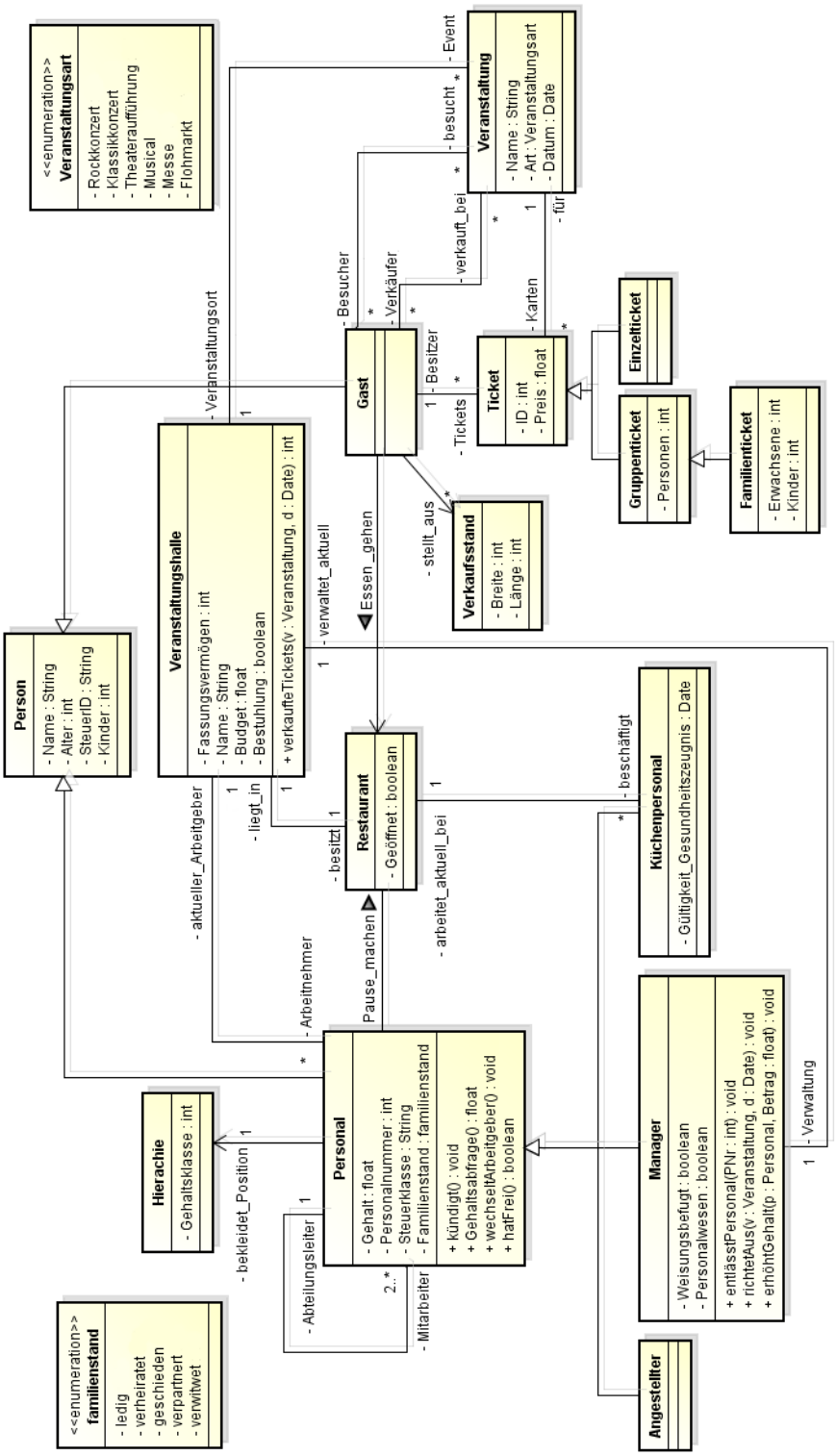


Abbildung 1: UML-Klassendiagramm der Veranstaltungshallen

Das UML-Klassendiagramm in Abbildung 1 erlaubt leider noch die Konstruktion ungewollter Konstellationen. Daher muss die OCL verwendet werden, um notwendige Randbedingungen durch die Spezifikation von Einschränkungen (constraints) zu formulieren. Drücken Sie folgende Constraints in Form von gültigen OCL-Ausdrücken aus

Formulieren Sie in dem Kontext, den die Aufgabenstellung impliziert. Verwenden Sie dabei ausschließlich die Operationen aus Aufgabe 1.3.

1. Das Alter eines Menschen ist immer positiv.
```
context Person
inv: self.Alter > 0
```

2. Eine Veranstaltung können maximal so viele Gäste besuchen, wie der entsprechende Veranstaltungsort fassen kann.
```
context Veranstaltung
inv: self.Besucher -> size() <= self.Veranstaltungsort.Fassungsvermögen
oder
inv: self.Veranstaltungsort.Fassungsvermögen >= self.Besucher -> size()
```

3. Die Veranstaltungshalle *Westfalenhalle* muss es für jede Veranstaltung mindestens 60 Karten geben.
```
context Veranstaltungshalle
inv: self.Name= 'Westfalenhalle' implies self.Event ->
forAll(v:Veranstaltung|v.Karten-> size() >= 60)
```

4. Aus wirtschaftlichen Gründen muss die *Westfalenhalle* sparen. Daher dürfen die Gehälter aller Arbeitnehmer der Westfalenhalle 60% des Budgets der Halle nicht übersteigen.
```
context Veranstaltungshalle
inv: self.Name= 'Westfalenhalle' implies self.Arbeitnehmer ->
collect(p:Personal|p.Gehalt) -> sum() <= self.Budget * 0.6
```

5. Bei einem Flohmarkt darf es in der Veranstaltungshalle keine Bestuhlung geben.
```
context Veranstaltung
inv: self.Art= Veranstaltungsart::Flohmarkt implies
self.Veranstaltungsort.Bestuhlung = false
oder
inv: self.Art= Veranstaltungsart::Flohmarkt implies
self.Veranstaltungsort.Bestuhlung <> true
```