Freie Universität Berlin

Lutz Prechelt

Softwaretechnik, SoSe21

Übung 09

TutorIn: Samuel Domiks
Tutorium 02
Materialien: Latex, Skript

Jonny Lam & Thore Brehmer

14. Juni 2021

---

1 Begriffe

(a) *Was versteht man im Kontext der Softwareentwicklung unter dem Begriff „Information Hiding“ ? Erläutern Sie das Prinzip und erklären Sie, in welchem Zusammenhang es zum „Need to Know“-Prinzip steht.*
Unter „Information Hiding”versteht man, das verstecken/verbergen von Daten und Informationen vor anderen Zugriffen bzw. vom Zugriff von außen.
Beim Need-to-know Prinzip verbirgt man die Informationen um Risiken in Vorhinein zu reduzieren. Die Informationen sind nur für die, die es brauchen zugänglich.[1]

(b) *Warum ist das Prinzip des Information Hiding sinnvoll? Erläutern Sie dieses an einem Beispiel.*
Angenommen wir entwickeln ein Programm um das Gehalt der Mitarbeiter zu berechnen.
Dann will ich nicht, dass meine Mitarbeiter freien Zugang zu der Methode haben, die den Gehalt berechnet.
Zum einen, weil die sich beschweren können warum der andere mehr bekommt oder auch, dass Konkurrenten (andere Arbeitgeber) mehr bieten können. Also Information Hiding ist sinnvoll um wichtige Informationen vor Personen zu schützen, denen das nichts angeht bzw. von dem man nicht will, dass sie die Information bekommen.

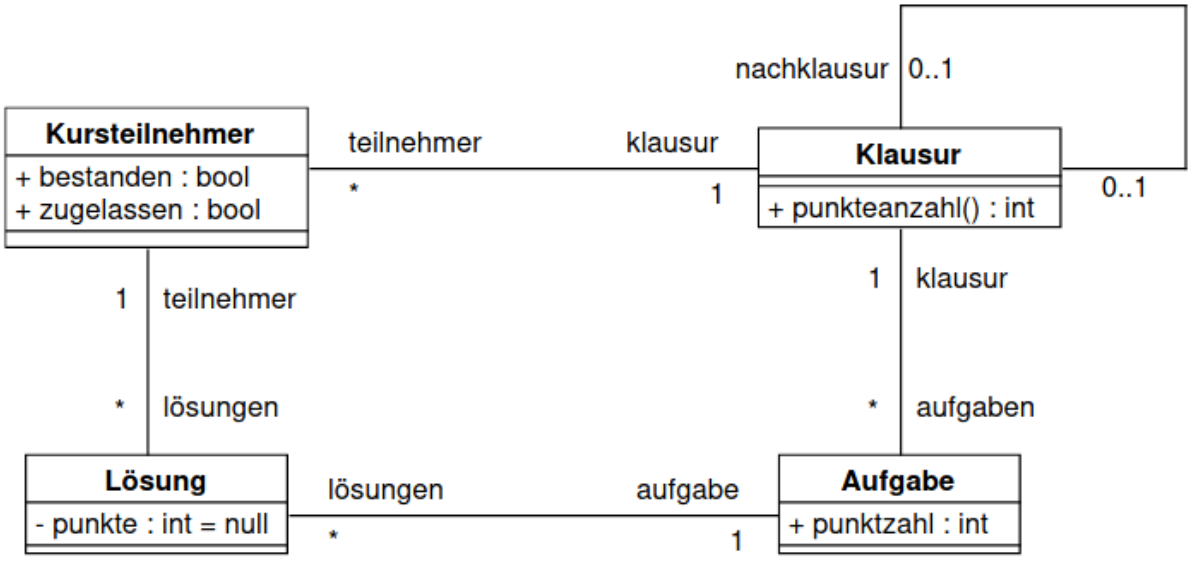
(c) *Was versteht man unter dem „Design by Contract“-Prinzip? In welchem Kontext wirdes eingesetzt und warum?*
Durch Contracts (Verträge) werden Abmachungen zwischen einzelne Programmmodule (Aufrufender und Aufgerufener) gemacht für die Verwendung von Schnittstellen. Diese Verträge soll die Kommunikation zwischen Programmodule sicherstellen. [2]

(d) *Erläutern Sie, in welchem Zusammenhang die BegriffeOCL, Invariante, Design by Con-tract, precondition, constraints, postcondition stehen.*
Mit **OCL** können wir bei der Modellierung notwendige Randbedingungen formell festlegen. Diese Randbedingungen nennen wir auch **Constraints**, die nach dem „**Design by Contract**”Prinzip gehen. Von den Constraints gibt es mehrere Arten, einmal Invarianten, die sind zu jedem Zeitpunkt wahr. Dann gibt es noch **precondition** und **postcondition**, pre Condition muss vor dem Aufruf wahr sein und post Condition nach dem Aufruf. [3]

Quellen:
[1]: Vorlesung 13 und https://de.wikipedia.org/wiki/Datenkapselung_(Programmierung)
[2]: Vorlesung 13 und https://de.wikipedia.org/wiki/Design_by_contract
[3]: Vorlesung 13 und https://de.wikipedia.org/wiki/Object_Constraint_Language#:~:text=Die%20Object%20Constraint%20Language%20(OCL,von%20Computerprogrammen%20formal%20festlegen%20können.&text=OCL%20ist%20seit%20der%20UML,besteht%20auch%20in%20der%20Modelltransformation.

2 OCL lesen und schreiben

In einem Prüfungsverwaltungssystem sollen die Klausurergebnisse und die Punktzahlen derStudierenden bei den Lösungen der einzelnen Aufgaben verwaltet werden. Das folgende UML-Klassendiagramm modelliert einen Teil der benötigten Daten.



(a) *Drücken Sie auf Basis des Diagramms die OCL-Constraints c1 bis c3n natürlichsprachlich aus, also in einer Form, die Sie in einem Gespräch mit einer Person ohne Informatik-Hintergrund verwenden würden.*

```
context Aufgabe inv c1: punktzahl > 0

context Kursteilnehmer inv c2: lösungen->size() = klausur.aufgaben->size()

context Klausur inv c3:
    teilnehmer->forAll (t |
        t.bestanden implies t.lösungen->exists (l |
            l.punkte > 0 and l.aufgabe.klausur = self
        )
    )
```

c1: Eine Aufgabe hat eine Punktzahl größer Null.

c2: Ein Kursteilnehmer hat genau soviel Lösungen, wie eine Klausur aufgaben hat. Anders gesagt: ein Kursteilnehmer muss genau eine Lösung für jede Klausur Aufgabe haben.

c3: Für jeden Teilnehmer an der Klausur gilt: Wenn die Klausur bestanden ist, dann hat jede Klausur Aufgabe von den Teilnehmer mehr als Null Punkte. Außerdem ist wichtig, dass die Aufgaben nicht Klausur übergreifend betrachtet werden. (Bsp. Klausur1: 1. bestanden, 2. nicht bestanden. Klausur2: 1. nicht bestanden, 2. bestanden. Daraus folgt trotzdem nicht bestanden!)

(b) *Geben Sie OCL-Constraints an, die die folgenden Sachverhalte formalisieren. AchtenSie darauf, syntaktisch einwandfreie OCL-Ausdrücke zu formulieren. Das schließt auchdie Beachtung der genauen Schreibweisen von Klassen, Attributen, Operationen undAssoziationen aus dem UML-Klassendiagramm ein.Schauen Sie auch in der OCL 2.4 Spezifikation (http://www.omg.org/spec/OCL/2.4/PDF) nach, falls Ihnen Ausdrucksmittel fehlen.*

a) *In jeder Klausur gibt es mindestens eine Aufgabe mit genau einem Punkt.*

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

b) *Eine Nachklausur kann keine Nachklausur haben.*

```
#2
context Klausur inv:
   nachklausur->size() < 0 implies:
       nachklausur->forAll(n | n.nachklausur->size() = 0)
```

c) *Ist ein/e Kursteilnehmer/in zugelassen, gibt es für jede Klausuraufgabe auch eine Lösung von ihm/ihr.*

```
#3
context Kursteilnehmer inv:
   zugelassen implies:
       lösungen->size() = klausur.aufgaben->size()
```

3 OCL-Modellerweiterungen

Diese Aufgabe baut auf dem gleichen UML-Modell auf wie Aufgabe 9-2. Nun soll zusätzlich auch die Klausurkorrektur modelliert werden: Bei der Korrektur geht der/die Dozent/in alle Lösungen einzeln durch, bewertet sie jeweils und errechnet die Gesamtpunktzahl für jede/n Teilnehmer/in.

(a) *Erweitern Sie das Modell nun um eine Operation korrigieren mit passender Signatur sowie ein neues Attribut; darüber hinaus darf das Klassendiagramm in keiner Weise verändert werden (also keine neuen Klassen, veränderte Sichtbarkeiten, etc.). Die Operation korrigieren wird aufgerufen, sobald der/die Dozent/in eine Lösung korrigiert hat. Es soll damit insbesondere möglich sein, dass*

- *das Attribut punkte in Lösung gefüllt wird, und*
- *die erreichte Gesamtpunktzahl des Teilnehmers aktualisiert wird.*

Beschreiben Sie zunächst verbal, was korrigieren genau leisten soll. Finden Sie geeignete Stellen (und für das neue Attribut: auch einen geeigneten Namen) für die neuen Member im Klassendiagramm.

Lösung:

- Die Methode korrigieren soll punkte in den Lösungen sowie in der Gesamtpunktzahl des Teilnehmers aktualisieren.
- Da die punkte von Lösungen privat sind, wir aber darauf zugriff brauchen, schreiben wir die Methode in die Klasse Lösung. Also: **+korrigieren(punkte: int)**

- Damit wir die Gesamtpunktzahl für den Kursteilnehmer eintragen können, erstellen wir ein public Attribut gesamtpunktanzahl im Kursteilnehmer. Also: **+gesamtpunktanzahl : int**

(b) *Gehen Sie zunächst in einer ersten Version davon aus, dass korrigieren nur einmal pro Exemplar der Klasse Lösung aufgerufen werden darf; die einmal gesetzte Punktzahl ist danach unveränderlich. Spezifizieren Sie sowohl möglichst strenge Vorbedingungen für Ihre Operation korrigieren, als auch alle Effekte(post condition)der Operation komplett in OCL.*

Wir möchten mit den OCL Bedienungen folgendes aussagen:

- **Pre:**
  - Es können nur Lösungen korrigiert werden, welche noch nicht korrigiert wurden.
  - Es dürfen nicht negative Punkte für die Lösung eingetragen werden.
  - Es dürfen nicht mehr Punkte, als die Aufgabe angibt, für die Lösung eingetragen werden.
  - Der Teilnehmer muss zugelassen sein, damit eine Lösung korrigiert wird.
- **Post:**
  - Nach der Korrektur wurden die Punkte korrekt eingetragen.
  - Nach der Korrektur wurde die Lösung wirklich korrigiert (Vielleicht unnötig?)
  - Nach der Korrektur wurden die Punkte auf die Gesamtpunktzahl des Teilnehmers addiert.

```
1  #Mehfraches korrigieren nicht erlaubt
2  context Lösung::korrigieren(p : int)
3      pre:
4          self.punkte->size() = 0 and
5          p >= 0 and
6          p <= self.aufgabe.punktzahl and
7          self.teilnehmer.zugelassen

9      post:
10         self.punkte = p
11         self.punkte->size() = 1 #Vielleicht unnötig?
12         self.teilnehmer.gesamtpunktzahl = self.teilnehmer.gesamtpunktzahl@pre + p
```

(c) *Gehen Sie nun von der realistischeren Anforderung aus, dasskorrigierenmehrfachaufgerufen werden kann, um z.B. die Punktzahl einer Lösung bei einer Klausureinsichtzu korrigieren.Spezifizieren Sie wiederum möglichst strenge Vorbedingungen und alle Effekte derOperation in OCL.*

- Wir haben lediglich die Bedienung "Es können nur Lösungen korrigiert werden, welche noch nicht korrigiert wurden." aus Zeile 4 entfernt.
- Sowie zu der Bedienung "Nach der Korrektur wurden die Punkte auf die Gesamtpunktzahl des Teilnehmers addiert." aus Zeile 12 (jetzt 11) etwas hinzugefügt.
- Denn nun muss der alte Punkte Wert für die Aufgabe noch von der Gesamtpunktzahl abgezogen werden, wenn man einen neuen Punkte Wert hinzu addiert will.

```
1  #Mehfraches korrigieren erlaubt
2  context Lösung::korrigieren(p : int)
3      pre:
4          p >= 0 and
5          p <= self.aufgabe.punktzahl
6          and self.teilnehmer.zugelassen

8      post:
9          self.punkte = p
10         self.punkte->size() = 1 #Vielleicht unnötig?
11         self.teilnehmer.gesamtpunktzahl = self.teilnehmer.gesamtpunktzahl@pre - self.
       punkte@pre + p
```

(d) *Spezifizieren Sie den Effekt der OperationKlausur.punkteanzahlunter Verwendungdes OCL-Schlüsselwortesresult.Hinweis: Der resultierende Ausdruck passt bequem auf eine Zeile. Recherchieren Siebei Bedarf nach OCL-Collections und ihren Operationen.*

```
#Klausur.punkteanzahl()
context Klausur::punkteanzahl : int
    result = self.aufgaben.punktzahl->sum()
```