Universität Koblenz-Landau
Institut für Informatik
Dr. Bernhard Beckert
www.uni-koblenz.de/~beckert

WS '02/03

Praktikum

Entwicklung objektorientierter Software mit formalen Methoden

Aufgabenblatt 2

Aufgabe 1

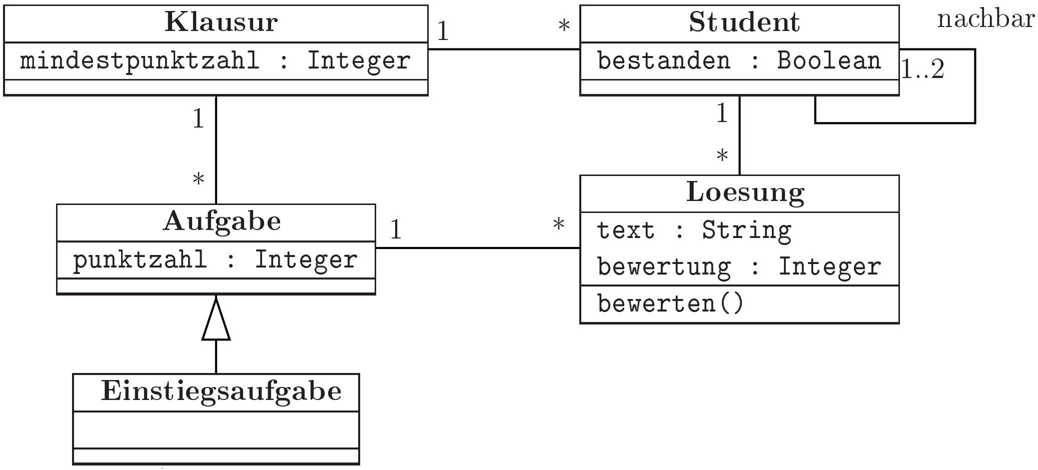
In dem Buch *A Programming Approach to Computability* von Kfoury, Moll und Arbib (Springer Verlag, 1982) findet sich auf den Seiten 19f die folgende Definition einer einfachen (künstlichen) Programmiersprache:

> We define a **while**-program to be a finite (possibly empty) sequence of *statements*, separated by semicolons, and preceded by **begin** and followed by **end**:
>
> $$\textbf{begin}\ S_1; S_2; \ldots; S_n\ \textbf{end}$$
>
> where $S_1, S_2, \ldots, S_n$ are arbitrary statements and $n \geq 0$.
>
> Statements are defined inductively. The "assignment statements" form the basis step, while the induction step involves the formation of "**while** statements" and "compound statements".
>
> In **while**-programs, the values assigned to variables will be non-negative integers. An *assignment statement* has one of the following forms:
>
> 1. $X := 0$
> 2. $X := succ(Y)$
> 3. $X := pred(Y)$
>
> where *succ* is the "successor" function and *pred* is the "predecessor" function. [...]
>
> X and Y in the above instructions stand for arbitrary variable names, possibly identical. [...]
>
> A **while** statement has the following form:
>
> $$\textbf{while}\ X \neq Y\ \textbf{do}\ S \quad ,$$
>
> which means "while the value of variable X is not equal to the value of variable Y, repeat the action specified by statement S. [...]
>
> Finally, a "compound statement" has the form:
>
> $$\textbf{begin}\ S_1; S_2; \ldots; S_m\ \textbf{end}$$
>
> where $S_1, S_2, \ldots, S_m$ are arbitrary statements and $m \geq 0$. A compound statement such as the one given here simply means that we have to carry out the actions prescribed by $S_1, S_2, \ldots,$ and S_m in succession and in the given order.

Geben sie ein UML-Klassendiagramm an, das die *abstrakte Syntax* der **while**-Sprache modelliert. Die Schlüsselwörter **begin**, **end** etc. und die Semikolons brauchen nicht modelliert zu werden. Auch die Semantik braucht nicht beachtet zu werden. Das Klassendiagramm soll lediglich die strukturellen Beziehungen der syntaktischen Elemente "program", "statement", "variable" etc. widerspiegeln.

Aufgabe 2

Das folgende UML-Klassendiagramm sei gegeben:



(a) Geben Sie natürlichsprachliche Übersetzungen der folgenden OCL-Constraints an:

i. <u>`Aufgabe`</u>

```
punktzahl > 0
```

ii. <u>`Student`</u>

```
loesung->size  =  klausur.aufgabe->size
```

iii. <u>`Loesung`</u>

```
student.nachbar->collect( n | n.loesung )->
    exists( l | l.text = self.text )
implies  not student.bestanden
```

(b) Geben Sie einen OCL-Constraint an, der äquivalent ist zu dem Constraint aus Aufgabenteil (a) iii., und der statt der Operatoren `collect` und `exists` den Operator `iterate` benutzt.

(c) Geben Sie OCL-Constraints an, die die folgenden Sachverhalte modellieren:

i. Für jede Einstiegsaufgabe gilt: Ihre Punktzahl ist kleiner oder gleich der Punktzahl aller Aufgaben der Klausur, zu der sie gehört.

ii. Nachdem eine Lösung bewertet worden ist (d.h., nach Ausführen der Methode `bewerten()`), ist ihre Bewertung größer oder gleich Null und kleiner oder gleich der Punktzahl der Aufgabe zu der sie gehört.

iii. *Verwenden Sie zur Modellierung des folgenden Sachverhalts das* `sum`*-Konstrukt*

Die Summe der Punktzahlen aller Aufgaben einer Klausur ist größer als die Mindestpunktzahl der Klausur.

iv. *Modellieren sie den selben Sachverhalt wie in der vorigen Teilaufgabe mithilfe des* `iterate`*-Konstrukts.*