Universität Koblenz-Landau WS ’03/04
Institut für Informatik
Bernhard Beckert · `www.uni-koblenz.de/~beckert`
Vladimir Klebanov · `www.uni-koblenz.de/~vladimir`

Praktikum

Entwicklung objektorientierter Software mit formalen Methoden

Aufgabenblatt 2

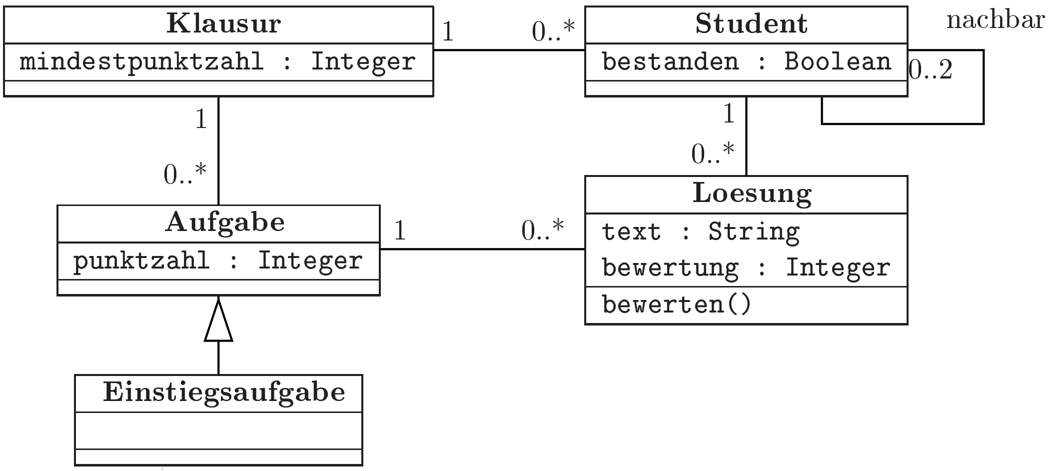
Aufgabe 5

Für die folgenden Aufgaben können Sie das Skriptum zur Vorlesung „Formale Spezifikation und Verifikation von Software“ verwenden, das von

`http://www.uni-koblenz.de/~beckert/Lehre/Spezifikation-Verifikation/`

herunterladbar ist, und sich ab Seite 49 mit OCL befasst.

Das folgende UML-Klassendiagramm sei gegeben:



Dieses Diagramm finden Sie als Together-Projekt herunterladbar von der Praktikumswebseite. Entpacken Sie das Projekt vor dem Laden mit TogetherCC, indem Sie

`tar −xzvf aufg5.tgz`

eingeben.

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

i. **context** Aufgabe
inv: self.punktzahl > 0

ii. **context** Student
inv: loesung->size = klausur.aufgabe->size

iii. **context** Loesung
inv: 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. Geben Sie die Constraints (wie besprochen) in TogetherCC ein, nachdem Sie das oben angegebene Projekt geladen haben. Die Constraints sollten wie besprochen vom OCL-Parser eingelesen werden können.

   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 Ausführung von `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 mithilfe des `iterate`-Konstrukts.*

(d) Betrachten Sie das Java-Programm `Blatt2.java` von der Praktikumswebseite. Es implementiert die Klassen des obigen Klassendiagramms.

   i. Erfüllt es auch die in den vorherigen Teilaufgaben angegebenen bzw. erstellten Constraints?

   ii. Wenn ja, (wie) könnten Sie das nachweisen? Wenn nein, implementieren Sie die `main`-Methode so, dass nach ihrer Ausführung ein Systemzustand eintritt, der die OCL-Constraints nicht erfüllt und geben Sie an, welcher Constraint verletzt wird.

Aufgabe 6

Präzisieren Sie Ihr in Aufgabe 3 (b) erstelltes Klassendiagramm mit Hilfe von OCL-Constraints. Ergänzen Sie dazu gegebenenfalls Ihr damals erstelltes Diagramm (z.B. um Attribute und Operationen) und stellen Sie folgendes sicher:

(a) Die Anzahl der von einem Kunden ausgeliehenen Gegenstände beschränkt sich auf höchstens 5.

(b) Die Anzahl der von einem Studenten ausgeliehenen Gegenstände beschränkt sich auf höchstens 3.

(c) Ein Student darf sich höchstens eine Zeitschrift ausleihen.

(d) Ist ein Gegenstand von einem Kunden ausgeliehen, so kann er durch Ausführung einer "verlängere"-Operation um höchstens 4 Wochen verlängert werden.

(e) Überschreitet ein Kunde mit seinem ausgeliehenden Gegenstand die (für seine Kundengruppe spezifische) maximale Ausleihfrist, so ist dieser ausgeliehene Gegenstand als 'Rückgabe angemahnt' und der Kunde mit 'gemahnt' gekennzeichnet.

Annotieren Sie die Constraints direkt an die Klassen und Operationen in Ihrem TogetherCC-Projekt.

Aufgabe 7

Spezifizieren sie die folgende Methode der Klasse `Math` in OCL

`Real nullstelle(a:Real, b:Real, .., g:Real)`,

welche Ihnen, sofern vorhanden, eine Nullstelle der folgenden Gleichung zurückliefert:

$$x^7 + a * x^6 + b * x^5 + ... + f * x + g,$$

Wie schwer wäre es, diese Methode zu implementieren? Könnte man eine Implementierung aus der Spezifikation ableiten?

Aufgabe 8

Häufig will man vermeiden, dass die Kompilate von Quelldateien oder andere erzeugte Dateien von CVS erfasst werden. Ein Grund, neben dem Vermeiden unnötigen Speicherplatzes, ist, dass CVS anderweitig versucht verschiedene Versionen von Binärdateien miteinander zu vereinen. Sollten Sie zum Beispiel unabhängig voneinander dieselbe Quelldatei `A.java` geändert und das erzeugte Kompilat `A.class` eingecheckt haben, wird CVS versuchen auch die verschiedenen Versionen der `.class` Dateien zu einer gemeinsamen zu verbinden. Bei Binärdateien führt dies in der Regel zu unerwünschten Resultaten.Um das zu vermeiden, gibt es in CVS die Möglichkeit bestimmte Dateitypen mithilfe einer `.cvsignore` Datei im Heimverzeichnis auszuschliessen.

Lesen Sie den entsprechenden Abschnitt in dem in Aufgabe 2 erwähntem Buch durch und erstellen Sie eine funktionsfähige `.cvsignore` Datei, die alle Binärdateien vom Typ `.class`, `.tmf` (ein Together Dateityp) ignoriert und weiter alle typischen Sicherheitskopien, wie <*dateiname*>~ oder #<*dateiname*>#.

Abgabe bis 24.11.
Es muss pro Gruppe nur *eine* Lösung abgegeben werden.
Die Abgabe der Übungsblätter erfolgt mit dem CVS System. Dazu legen Sie die abzugebenden Dateien im CVS ab und markieren die abzugebende Version der Dateien mit “`LoesungsBlatt`<*nr*>” wie in Aufgabe 2 beschrieben. Die Lösungen sollten vorzugsweise im dazugehörigen Unterverzeichnis `uebeungsblaetter/nr/` vorzufinden sein, zumindest aber ein Hinweis auf den Ort der Lösungen.
Einige Aufgaben verlangen eine schriftliche Bearbeitung, diese ist dann je nach Komplexität als ASCII, html, ps- oder pdf-Dokument abzugeben. Auf *keinen* Fall im MS Word `doc`-Format.

Materialien

`http://www.uni-koblenz.de/~beckert/Lehre/Praktikum-Formale-Entwicklung/`

Bernhard Beckert: Zi. MB 221, Tel. 287-2775, Email: `beckert@uni-koblenz.de`
Vladimir Klebanov: Zi. MB 230, Tel. 287-2781, Email: `vladimir@uni-koblenz.de`