Institut für Theoretische Informatik
Logik und Formale Methoden
Prof. Dr. B. Beckert, Prof. Dr. P. H. Schmitt, D. Bruns, D. Faragó, Dr. C. Gladisch, C. Scheben

22.10.2012

Praktikum
Formale Entwicklung objektorientierter Software
Übungsblatt 1: Java Modeling Language

Aufgabe 1 — Methodenverträge

Gegeben sei folgende JML-Spezifikation:

```
private int[] a;

/*@ ensures a == \old(a);
  @ signals_only ArrayIndexOutOfBoundsException;
  @ assignable a[*];
  @ also
  @ normal_behavior
  @ requires a.length > 0;
  @ ensures (\forall int i; 0 <= i && i < a.length;
  @                         a[i] <= \result);
  @ ensures (\exists int i; 0 <= i && i < a.length;
  @                         a[i] == \result);
  @ diverges a[0] == 0;
  @*/
public int foo();
```

(a) Geben Sie ihre Bedeutung in natürlicher Sprache wieder.

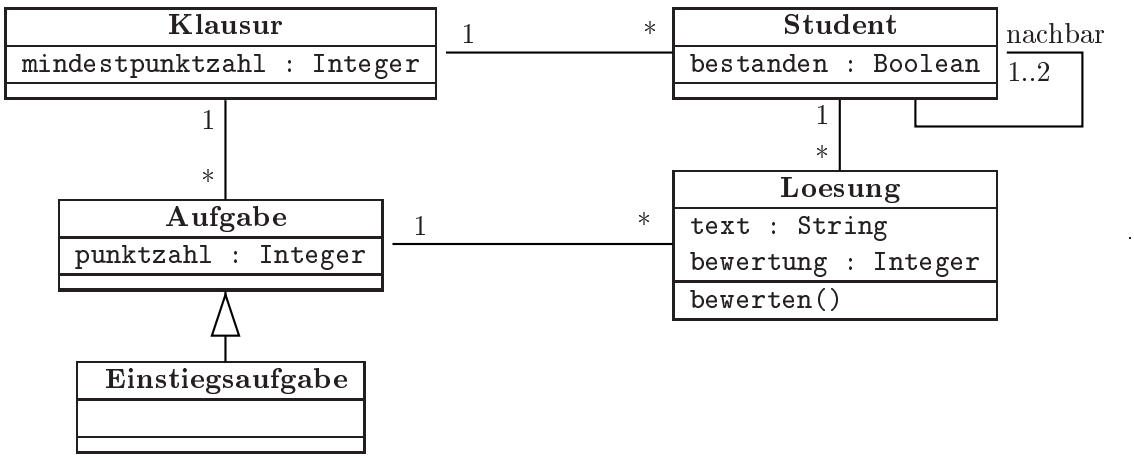
(b) Welche Eigenschaft wird nicht explizit aufgeführt, die trotzdem Teil von Vor- und Nachbedingungen ist (und auch notwendig für die Wohldefiniertheit selbiger ist)?

Aufgabe 2 — Implementieren von Spezifikationen

Auf der Praktikums-Webseite finden Sie eine Datei `Contracts.java`. Geben Sie für die dort enthaltenen Spezifikationen jeweils Implementierungen an — oder begründen Sie weshalb es keine Implementierung geben kann, die die Spezifikation erfüllt.

Aufgabe 3 — Spezifizieren von existierendem Code mit JML

Das folgende UML-Klassendiagramm sei gegeben:



Betrachten Sie das auf der Praktikums-Webseite verfügbare Programm `Klausuren.java`. Es implementiert die Klassen des obigen Klassendiagramms.

(a) Übersetzen Sie folgende natürlichsprachliche Beschreibungen in JML-Spezifikationen (Invarianten, Methodenverträge) und fügen Sie diese in das obige Java-Programm ein:

i. Für jede Aufgabe gilt, dass ihre Punktzahl größer als 0 ist.

ii. Für jeden Studenten gilt, dass alle Lösungen zu genau der bearbeiteten Klausur gehören.

iii. Die Zahl der Lösungen eines Studenten ist gleich der Zahl der Aufgaben seiner Klausur. Drücken Sie diesen Sachverhalt als Vertrag für die Methode `Student.macheLoesungen(Loesung[] loesungen)` aus.

iv. Die folgende Bedingung soll sich ebenfalls in einem Vertrag für diese Methode wiederfinden: Ein Student hat nicht bestanden, wenn er eine Lösung hat, deren Text mit dem Text einer Lösung eines seiner Nachbarn übereinstimmt.

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

vi. Nachdem eine Lösung bewertet worden ist (d.h., nach Ausführung von `bewerten()`), ist ihre Bewertung größer oder gleich 0 und kleiner oder gleich der Punktzahl der Aufgabe, zu der sie gehört.

vii. *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.

Stellen Sie mit Hilfe des JML-Syntaxprüfers (Befehl „`jml Klausuren.java`“) sicher, dass Ihre Spezifikationen syntaktisch korrekt sind. Achten Sie auch darauf, dass nur Kommentare, deren *erstes* Zeichen ein „`@`“ ist, als JML interpretiert werden.

(b) Erfüllt die Implementierung die in den vorherigen Teilaufgaben erstellten Spezifikationen?

Wenn ja, (wie) könnten Sie das nachweisen? Wenn nein, implementieren Sie die `main`-Methode so, dass nach ihrer Ausführung ein Systemzustand eintritt, der eine der spezifizierten Eigenschaften nicht erfüllt und geben Sie an, welche Eigenschaft verletzt wird.

Aufgabe 4 — Schleifen und Rekursion

(a) Geben Sie für die folgenden Methoden jeweils (sinnvolle) Schleifeninvarianten und -varianten an.

(b) Schreiben Sie die Methoden so um, dass sie anstelle von Schleifen rekursive Aufrufe verwenden. Geben Sie dazu Kontrakte an.

```
public int foo ( int[] a ) {
    int i = 0;
    int x = a[0];

    while ( ++i < a.length )
        if ( a[i] > x ) x = a[i];
    return x;
}

public int bar ( int x, int y ) {
    int z = 0;
    while ( y-- > 0 )
        z += x;
    return z;
}
```

Aufgabe 5 (Zusatzaufgabe)

Gegeben sei ein Array von Ganzzahlen der Länge $n+2$ mit $n >= 2$. Mindestens zwei verschiedene Werte tauchen doppelt auf (d.h. es gibt mindestens zwei Duplikate). Implementieren und spezifizieren Sie ein (nicht notwendigerweise lauffähigees) Java-Programm, das zwei solche Werte findet.[1]

Abgabe bis Montag, 05.11.2012

Abgabe (als Java-, ASCII- oder PDF-Dateien, oder tar-Archiv solcher) per E-Mail an Daniel Bruns. Es braucht pro Gruppe und Aufgabe nur *eine* Lösung abgegeben werden. Bitte dokumentieren Sie Ihre Lösungen ausreichend und seien Sie darauf vorbereitet, sie auf Nachfrage zu erklären.

Praktikums-Webseite: `http://formal.iti.kit.edu/teaching/keypraktWS1213/`

Daniel Bruns:	R. 223,	Tel. 608-45268,	E-Mail: `bruns@kit.edu`
David Faragó:	R. 308,	Tel. 608-47322,	E-Mail: `farago@ira.uka.de`
Christoph Gladisch:	R. 223,	Tel. 608-45268,	E-Mail: `gladisch@ira.uka.de`
Christoph Scheben:	R. 106,	Tel. 608-44338,	E-Mail: `scheben@ira.uka.de`

[1] Falls es hilfreich ist, dürfen Sie annehmen, dass das Array nur Werte zwischen 0 und $n-1$ enthält.