

Praktikum

Formale Entwicklung objektorientierter Software

Übungsblatt 3: JML

Aufgabe 5 — Methodenverträge

Geben Sie in natürlicher Sprache die Bedeutung der folgenden JML-Spezifikation an:

```
//@ assignable x;  
//@ signals_only IllegalArgumentException, IllegalStateException;  
//@ signals(IllegalArgumentException e) e.getMessage().equals("Oops");  
//@ diverges false;  
//@ also normal_behavior  
//@ requires x >= 0 && i >= 0;  
//@ ensures x == \old(x) + i;  
Object m(int i);
```

Aufgabe 6 — Invarianten

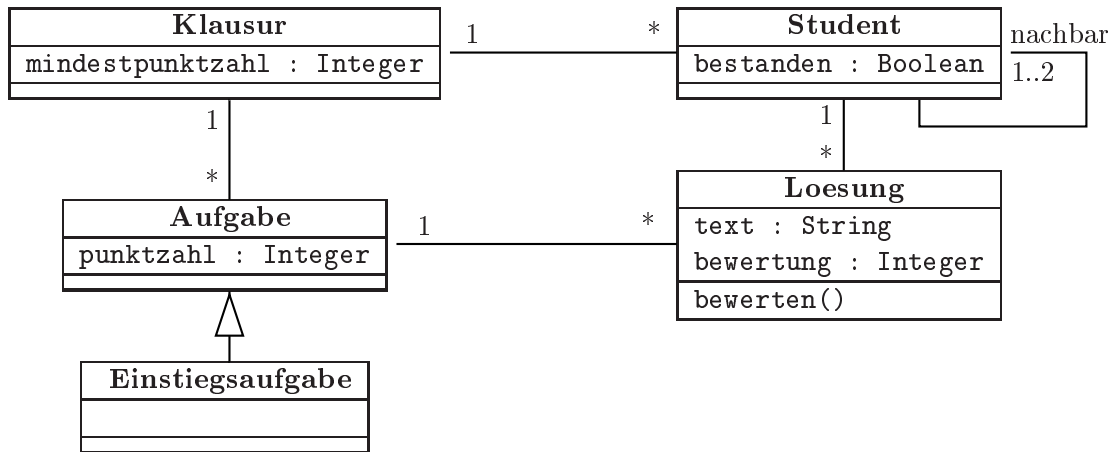
Betrachten Sie die auf der Praktikums-Webseite verfügbare Datei `Invarianten.java`. Welche der Klassen `C1` bis `C4` erfüllen nach der “Visible State Semantics” für Invarianten ihre Spezifikation, und welche nicht? Begründen Sie!

Zur Erinnerung: Die “Visible State Semantics” von JML verlangt, dass eine Instanz-Invariante für ein Objekt o mindestens in folgenden Programzuständen erfüllt sein muss:

- Am Ende eines Konstruktoraufrufs, der o initialisiert,
- am Anfang und Ende eines Aufrufs einer nicht-statischen Methode auf o ,
- am Anfang und Ende eines Aufrufs einer statischen Methode aus der Klasse von o oder einer ihrer Oberklassen, und
- immer wenn *kein* Aufruf einer der drei beschriebenen Arten im Gang ist.

Aufgabe 7 — 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 Null ist.
 - ii. 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.
 - iii. 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.
 - iv. Für jede Einstiegsaufgabe gilt: Ihre Punktzahl ist kleiner oder gleich der Punktzahl aller Aufgaben der Klausur, zu der sie gehört.
 - v. 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.
 - vi. *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 Invarianten nicht erfüllt und geben Sie an, welche Invariante verletzt wird.

Aufgabe 8 — Entwickeln mit Java und JML

Im letzten Übungsblatt haben Sie die “Event-Bank” in Event-B modelliert. Dabei konnten Kunden mit Bankkarten verschiedene Dienste an Bankautomaten in Anspruch nehmen. Hierzu mussten sie sich zunächst authentifizieren. Der Authentifizierungsprozess soll in dieser Aufgabe mit JML spezifiziert und in Java implementiert werden.

Der Authentifizierungsprozess wird durch folgende Anforderungen beschrieben:

Jede Karte hat eine unveränderliche, vierstellige PIN.	REQ-29
Karten können in Bankautomaten eingeführt werden, zu jedem Zeitpunkt ist höchstens eine Karte im Automaten.	REQ-34
Zu jedem Zeitpunkt ist eine Karte in höchstens einem Automaten.	REQ-35
Ein Benutzer kann seine eingeführte Karte jederzeit vom Automaten ausgeben lassen.	REQ-36
Der Karteninhaber muss sich am Automaten nach Einführen der Karte durch Eingabe der PIN authentifizieren.	REQ-37
Bei Eingabe der korrekten PIN ist der Benutzer authentifiziert, bis die Karte wieder ausgegeben wird.	REQ-38
Wird für eine Karte dreimal in Folge (also bei drei aufeinanderfolgenden Authentifizierungsversuchen, auch an verschiedenen Automaten) eine falsche PIN eingegeben, so ist die Karte danach gesperrt.	REQ-39
Eine gesperrte Karte wird vom Automaten sofort eingezogen und nicht wieder ausgegeben. Der Automat ist dann wieder kartenfrei.	REQ-40

Was mit eingezogenen Karten passiert, interessiert uns nicht weiter.

Ein Automat kann die PIN einer Karte nicht auslesen. Die Funktionalität zum Authentifizieren des Benutzers und zum Sperren der Karte muss von der Karte selbst bereitgestellt werden.	REQ-41
Automaten können von Karten abfragen, ob sie gesperrt sind.	REQ-42

Setzen Sie diese Anforderungen möglichst vollständig in JML um, und implementieren Sie das System mit Java.

Die Signaturen der beteiligten Klassen sind in den mitgelieferten Dateien `ATM.java` (Bankautomaten) und `BankCard.java` (Bankkarten) vorgegeben. Sie können die Klassen um weitere Attribute und Methoden ergänzen, behalten Sie aber die gegebenen Methodensignaturen bei. Zum Ausprobieren Ihrer Implementierung steht in `ATMGui.java` eine GUI bereit.

Abgabe bis Mittwoch, 02.12.

Die Abgabe erfolgt über die Praktikums-Webseite. 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://lfm.iti.kit.edu/keyprakt0910.php>

Christian Engel: R. 106, Tel. 608-4338, E-Mail: engelc@ira.uka.de

David Faragó: R. 308, Tel. 608-7322, E-Mail: farago@ira.uka.de

Roman Krenický: E-Mail: krenicky@ira.uka.de

Mattias Ulbrich: R. 106, Tel. 608-4338, E-Mail: mulbrich@ira.uka.de

Benjamin Weiß: R. 309, Tel. 608-4324, E-Mail: bweiss@ira.uka.de