Einführung in Software Engineering WS 10/11

Fachbereich Informatik

Dr. Michael Eichberg
eichberg@informatik.tu-darmstadt.de

Assistent: Ralf Mitschke
mitschke@st.informatik.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungsblatt 5 (10 Punkte): Automatisiertes Testen

Abgabeformat: Reichen Sie ihre Lösung per SVN ein. Jede Übung muss in einen eigenen Ordner ***ex<Number>*** *(<Number> = 01, 02, ...)* in Ihrem Gruppenverzeichnis gespeichert werden. Während der Übungsbearbeitung können Sie Ihre Lösungen beliebig oft in das SVN hochladen (per Commit). Wir prüfen die Zeit der Einreichung Ihrer Lösungen unter der Benutzung des SVN Zeitstempels.

Erstellen Sie für Lösungen der Aufgaben, die keinen Quelltext erfordern, eine PDF-Datei mit dem Dateinamen ***solution.pdf***. Dies gilt auch für UML-Diagramme die Sie erstellen. Die Basisanwendung wird als Eclipse-Projekt vorgegeben. Ihr eigener Code muss entsprechend in den dafür vorgesehenen Verzeichnissen (***/src*** oder ***/test***) erstellt werden.

Abgabetermin: 01.12.2010 - 24:00 Uhr

Aufgabe 1 (4 Punkte)

Ziel: *Systematisches Testen erlernen*

In dieser Aufgabe soll die Flashcards-Anwendung mit Hilfe von JUnit getestet werden. Ein Eclipse-Projekt mit den Rumpfklassen, in denen Sie die Tests schreiben sollen finden Sie im EiSE SVN Repository unter ***public/exercise/ex05***. Der zu testende Code ist Ihre eigene Code-Basis der Flashcards-Anwendung, die Sie in der vierten Übung erhalten und weiterentwickelt haben. Nutzen Sie das Eclipse-Plugin **EclEmma** (http://www.eclemma.org) zur Unterstützung beim automatisierten Testen. EclEmma errechnet die aktuell erreichte Testabdeckung und zeigt diese an.

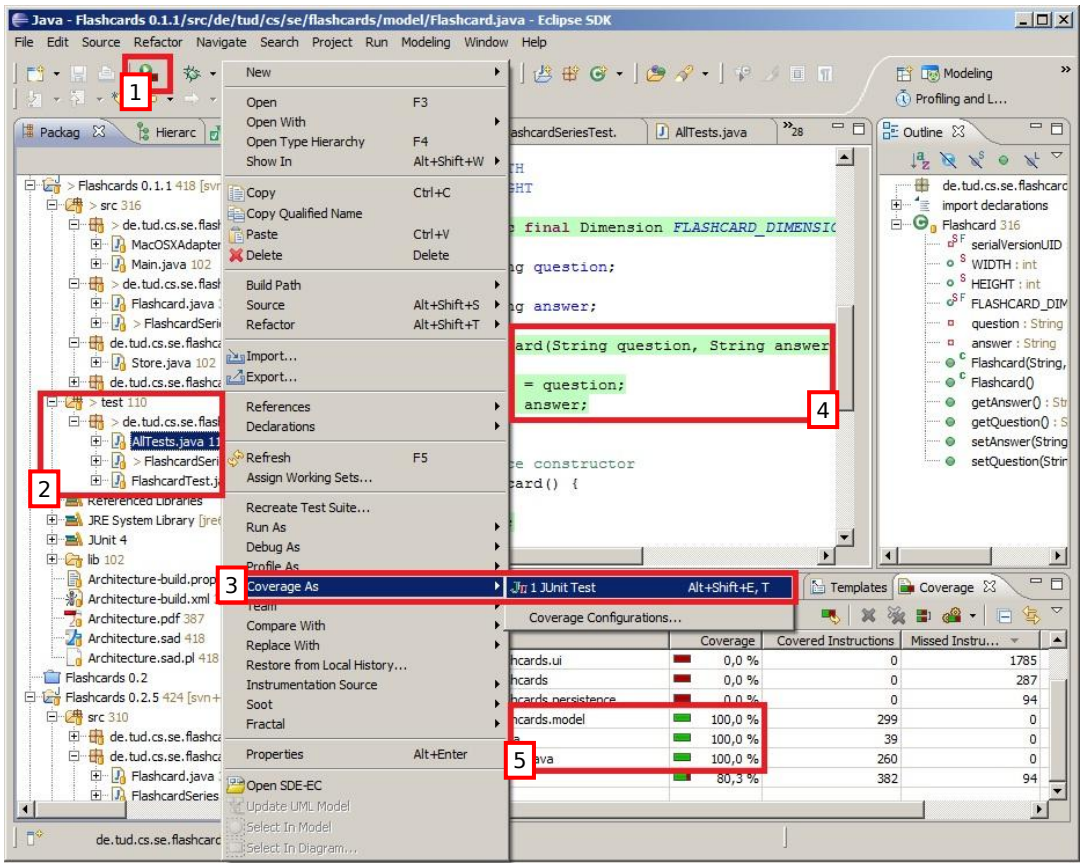
a) Basic Block Coverage (3 Punkte)

Schreiben Sie JUnit 4 Tests für die Klassen **`Flashcard`** und **`FlashcardSeries`**. In der Vorgabe aus dem Repository finden Sie im Verzeichnis **`test`** zwei Test-Klassen (**`FlashcardTest`**, **`FlashcardSeriesTest`**). Implementieren Sie Ihre Tests innerhalb dieser vorgegebenen Klassen. Sie können die vorgegebene JUnit 4 – TestSuite (**`AllTests`**) nutzen, um alle Tests für diese Aufgabe in einem Durchlauf zu starten. Stellen Sie mit Hilfe von EclEmma sicher, dass Ihre Tests für die genannten Klassen 100% *Basic Block Coverage* erreichen.

b) Condition Coverage (1 Punkte)

Erstellen Sie in der Testklasse **`ConditionCoverageTest`** JUnit Tests, die 100% Simple Condition Coverage für die Methode **`public synchronized void removeListDataListener(ListDataListener l)`** der Klasse **`FlashcardSeries`** erreichen.

Hinweis: *Simple Condition Coverage wird von EclEmma nicht errechnet. Überprüfen Sie deshalb mit Hilfe des Vorlesungsmaterials selbst, ob ihre Tests 100% Simple Condition Coverage erreichen.*



1: Start der Analyse

2: hier liegen die Testfälle im Projekt

3: Start der Analyse

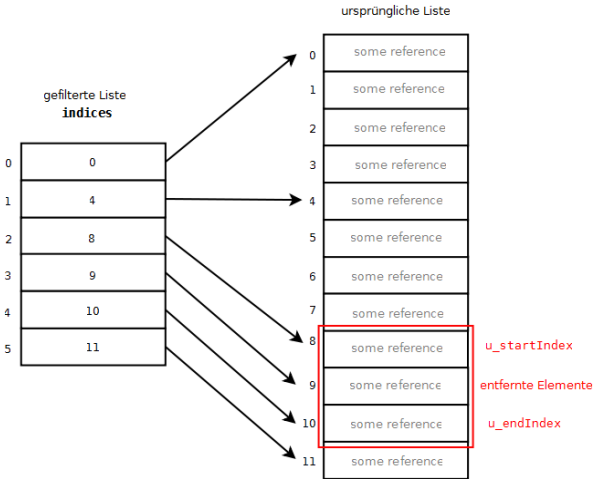
4: Markierung des durch Testfälle abgedeckten Source-Code (Basic Block Coverage)

5: Prozentuale Source-Code Abdeckung

Aufgabe 2 (6 Punkte)

Ziel: *Systematisches Testen von komplexen Methoden*

Die unten gelistete Methode **`intervalRemoved(int u_startIndex, int u_endIndex)`** ist Bestandteil Klasse, die das Filtern von Listenelementen unterstützt. Die Methode ist dafür verantwortlich, dass Elemente, die aus einer zugrundeliegenden Liste von Daten gelöscht werden, auch im zugehörigen Filterergebnis entfernt werden. Die Daten und die gefilterte Liste basieren hier auf Arrays. Die Argumente **`u_startIndex`** und **`u_endIndex`** geben den ersten und den letzten Index der Elemente an, die aus der zugrundeliegenden Liste gelöscht wurden. Die Klasse zum das Filtern von Listenelementen enthält ein Array **`indices`**, das die Indizes der Elemente beinhaltet, die aus der zugrundeliegenden Gesamtliste ausgewählt wurden. Die nachfolgende Abbildung verdeutlicht die Daten der gefilterten Liste. Die Methode **`intervalRemoved`** wird aufgerufen, wenn auf der unterliegenden Liste eine Löschoperationen durchgeführt wurde. Die Liste der **`indices`** des Filters wird innerhalb dieser Methode aktualisiert. Die Indizes in der zugrundeliegenden Liste können sich beim Entfernen von Elementen auch verschieben. So ist z.B. nach der im Bild dargestellten Löschung, das Element das vorher Index 11 hatte auf Index 8 aufgerückt. Die Methode **`intervalRemoved`** beinhaltet daher auch ein update der betroffenen Indizes. Nachdem die gefilterte Liste aktualisiert wurde, wird zusätzlich die Methode **`fireIntervalRemoved(ElementFilter filter, int startIndex, int endIndex)`** aufgerufen. Diese Methode erzeugt ein Ereignis, das andere Komponenten über die Änderung informiert. Dies wird z.B. vom User-Interface genutzt, um anschließend das GUI-Framework anzuweisen, die Ausgabe neu zu zeichnen.



a) Testplanerstellung (3 Punkte)

Erstellen Sie analog zum Beispiel aus der Vorlesung einen Testplan für die Methode **`intervalRemoved(int u_startIndex, u_endIndex)`** und stellen Sie sicher, dass Sie mit diesem 100% Simple *Condition Coverage* erreichen. Geben Sie für jeden Testfall die Eingabedaten und das erwartete Ergebnis an. Markieren Sie zusätzlich, wie die einzelnen Bedingungen in den Testfällen abgedeckt werden und wie sie dabei auswerten. Sie können die Zeilennummern verwenden, um die entsprechenden Conditions zu benennen. Eine Angabe der genutzten Methode `ArrayUtils.remove` finden Sie auf der nächsten Seite.

```
   public class FilteredList {
           …
           private int[] indices;

01         public void intervalRemoved(int u_startIndex, int u_endIndex) {
02                 if( u_startIndex > u_endIndex) throw new IllegalArgumentException();
03
04                 int endIndex = -1;
05                 int startIndex = indices.length;
06
07                 // the list of indices is traversed in reverse order
08                 for (int index = indices.length - 1; index >= 0; index--) {
09
10                         if (indices[index]>=u_startIndex && indices[index]<=u_endIndex) {
11
12                                 // update startIndex since we moved a new (lower) element
13                                 startIndex = index;
14
15                                 // update the endIndex only if we did not do so before
16                                 if (index > endIndex)
17                                         endIndex = index;
18
19                                 // removes the element at index by creating a new array
20                                 indices = ArrayUtils.remove(indices, index);
21
22                                 // update indices to reflect removal in underlying array
23                                 for (int j = index; j < indices.length; j++) {
24                                         indices[j] = indices[j] - 1;
25                                 }
26                         }
27                 }
28                 if (startIndex < endIndex) {
29                         // we did remove some elements ...
30                         fireIntervalRemoved(this, startIndex, endIndex);
31                 }
32         }
           …
           private void fireIntervalRemoved(ElementFilter f, int s, int e) { ... }
   }
```

```java
public class ArrayUtils {
        …

        /**
         * Returns a new array where the item with the specified index is removed from
         * the array <tt>ts</tt>.
         * @param ts the base array for the creation of a new array; this is not changed.
         * @param index the index; must be valid.
         * @return the newly created array which contains all elements of <tt>ts</tt>
         * without the element at the given index.
         */
        public static int[] remove(int[] ts, int index)
                throws ArrayIndexOutOfBoundsException {

                if (ts.length == 1) {
                        if (index != 0) {
                                throw new ArrayIndexOutOfBoundsException(index);
                        }
                        return new int[0];
                }

                int[] newts = new int[ts.length - 1];

                System.arraycopy(ts, 0, newts, 0, index);
                if (index < newts.length) {
                        System.arraycopy(ts, index +1, newts, index, newts.length -index);
                }

                return newts;
        }

}
```

b) Fehlerbestimmung (2 Punkte)

Die in a) gelistete Methode **`intervalRemoved(int u_startIndex, u_endIndex)`** enthält einen Fehler. Erläutern Sie unter welchen Umständen der Fehler im Programm auftreten wird. Diskutieren Sie, wie sich der Fehler in einer laufenden Applikation auswirken wird. Diskutieren Sie außerdem, ob sie diesen Fehler zwingend mit Ihrem Testplan für *Simple Condition Coverage* aus Teilaufgabe a) gefunden hätten.

c) Fehlerbehebung (1 Punkt)

Zwei Entwickler haben unabhängig voneinander eine neue Implementierung geliefert, die den Fehler aus Teilaufgabe a) behebt. Eine der beiden unten angegebenen Varianten der Methode **`intervalRemoved(int u_startIndex, u_endIndex)`** ist korrekt, die andere beinhaltet einen neuen Fehler. Identifizieren Sie die fehlerhafte Variante und diskutieren Sie, wie sich der Fehler in der laufenden Applikation auswirken wird, und, ob sie diesen Fehler zwingend mit Ihrem Testplan für *Simple Condition Coverage* aus Teilaufgabe a) gefunden hätten.

Variante I

```
01  public void intervalRemoved(int u_startIndex, int u_endIndex) {
            …

28          if (endIndex > -1) {
29                  // we did remove some elements ...
30                  fireIntervalRemoved(this, startIndex, endIndex);
31          }
32  }
```

Variante II

```
01  public void intervalRemoved(int u_startIndex, int u_endIndex) {
            …

28          if (startIndex < -1) {
29                  // we did remove some elements ...
30                  fireIntervalRemoved(this, startIndex, endIndex);
31          }
32  }
```