Übungen zur Vorlesung

Softwaretechnologie

-Wintersemester 2009/2010-

Dr. Günter Kniesel, Pascal Bihler

Übungsblatt 12

Zu bearbeiten bis: 24.01.2010

Bitte fangen Sie **frühzeitig** mit der Bearbeitung an, damit wir Ihnen bei Bedarf helfen können.
Checken Sie die Lösungen zu den Aufgaben in Ihr SVN-Repository ein, Texte als Textdatei.

Aufgabe 1. *Blackbox Test* (12 Punkte)

Gegeben sei folgende Schnittstelle:

```
interface DateParser {
     java.util.Date parseDate(String input);
}
```

und folgender Dokumentation:

> Interpretiert gutmütig einen Datumsstring der Form *Tag-Monat-Jahr*
>
> *Tag* und *Monat* können ein- order zweistellig sein, evtl. mit führender 0
> *Jahr* kann zwei- oder vierstellig sein, zweistellige Angaben beziehen sich auf das 21. Jahrhundert
>
> Wenn *Monat* kleiner *1* ist, wird Januar angenommen, bei Werten über *12* Dezember.
> Wenn *Tag* kleiner *1* ist wird der Monatserste angenommen, bei Werten größer dem zuletzt gültigen Tag der Monatsletzte.
>
> **Spezifiziert durch:** parseDate(...) in DateParser
> **Parameter:**
> **input** Der Datumsstring, der interpretiert werden soll
> **Liefert zurück:**
> Das interpretierte Datum (mit dem Uhrzeitwert 12:00:00) oder *null* bei ungültiger Eingabe

a) Überlegen Sie, welche Äquivalenzklassen auftreten können. Geben Sie für fünf typische korrekte Eingaben und Fehlerarten einen kritischen Eingabestring und das korrekte Methoden-Ergebnis an.

b) In Ihrem SVN-Repository finden Sie das Projekt „DateParser“, dass eine Klasse enthält, die obiges Interface implementiert. Vervollständigen Sie mit *JUnit 4* den `GemaltoDateParserTest`, mit der Sie die Methode `parseDate` in der Klasse `GemaltoParser` testen. Setzten Sie dabei jede in (a) identifizierte Fehlerart in einen Testfall um.
Hinweis: *Sie können in den Tests die Hilfs-Methode* `makeDate(…)` *verwenden.*
Bonus: Welchen Fehler hat die getestete Klasse?

c) Schreiben Sie nun eine Klasse `MyDateParserTest`, welche von `GemaltoDateParserTest` erbt, aber die `setUp`-Methode so überschreibt, dass in `systemUnderTest` eine Instanz der Klasse `MyDateParser` instantiiert wird. Entwickeln Sie nun die Klasse `MyDateParser`, indem Sie nach jedem Entwicklungsschritt den JUnit-Test `MyDateParser` ausführen, bis alle Testmethoden erfolgreich sind (Zur Dokumentation Ihrer Vorgehensweise sollen Sie nach jeder Modifikation Ihre Klasse in Ihr SVN-Repository einchecken).

Aufgabe 2. *Whitebox Test* (10 Punkte)

Gegeben sei die folgende Methode `sortiere`, welche mittels Bubblesort ein Feld von Variablen des Typs `int` sortiert.

```
public int[] sortiere(int[] bestand) {                    // Anweisungsnr.
      boolean change = true;                              //  1
      if (bestand.length > 1) {                           //  2
            while (change) {                              //  3
                  change = false;                         //  4
                  for (int i = bestand.length - 1;        //  5
                              i > 0;                      //  6
                              i--) {                      //  7
                        int i1 = bestand[i];              //  8
                        int i2 = bestand[i - 1];          //  9
                        if (i1 < i2) {                    // 10
                              bestand[i] = i2;            // 11
                              bestand[i - 1] = i1;        // 12
                              change = true;              // 13
                        }
                  }
            }
      }
      return bestand;                                     // 14
}
```

a) Entwerfen Sie für die Methode `sortiere` einen Kontrollflussgraphen.

b) Geben Sie ein Feld mit Eingabewerten an, das nötig ist, um eine Anweisungsüberdeckung zu erreichen. Schreiben Sie die Reihenfolge auf, in der die Anweisungen getestet werden.

c) Erreichen Sie mit diesem Feld an Eingabewerten auch eine Verzweigungsabdeckung? Begründen Sie kurz Ihre Antwort. Falls ja, geben Sie eine Codemodifikation an, mit der die Verzweigungsabdeckung nicht mehr gegeben wäre. Falls nicht, geben Sie ein weiteres Eingabewerte-Feld an, sodass auch die übrigen Zweige überdeckt werden. Notieren Sie zu diesem neuen Testfall wieder die Reihenfolge der durchgeführten Anweisungen.

d) Wie viele Pfade gibt es? Kurze Begründung.

e) Formulieren Sie ein minimales Programm, für das mindestens zwei verschiedene Testfälle notwendig sind, um eine Anweisungsüberdeckung zu erreichen.

Aufgabe 3. *Refactorings* (8 Punkte)

a) Überlegen Sie sich welche Probleme beim Verlagern einer Methode in eine andere Klasse (Move Method Refactoring) und beim Extrahieren von Programmcode aus einer Methode (Extract Method Refactoring) auftreten können. Beachten Sie auch Sichtbarkeiten und Vererbung!

b) Erkunden Sie auch die Refactoring-Fähigkeiten von Eclipse. Schreiben Sie dafür ein kleines Programm, und führen Sie mit Eclipse drei verschiedene automatisierte Refactorings durch. Beschreiben Sie was Sie tun, und was das System tut.

Tipp: Wenn Sie im Java-Editor von Eclipse mit dem Cursor auf eine Klasse / Variable / Methode gehen und die rechte Maustaste drücken, wird Ihnen unter „Refactor“ ein Katalog möglicher Refactorings angeboten.

Das SWT-Team wünscht gutes Gelingen und viel Erfolg für die Prüfung!