Übungen zur Vorlesung

Softwaretechnologie

-Wintersemester 2010/2011-

Dr. Günter Kniesel

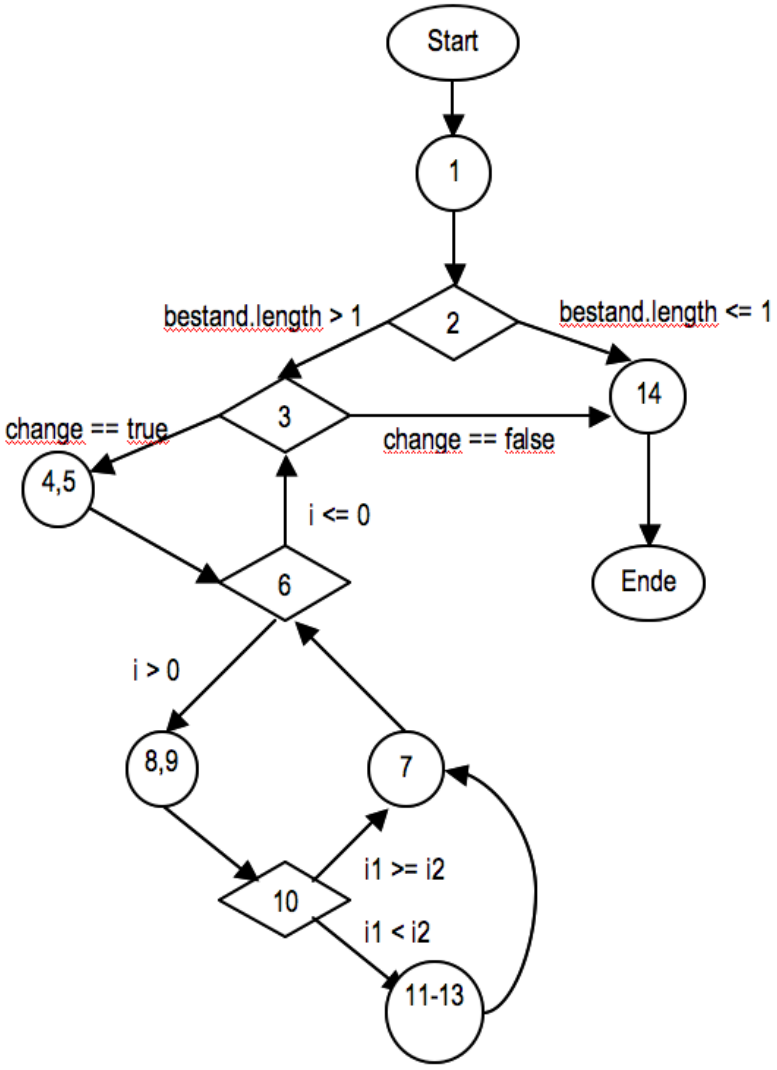
Übungsblatt 13 - Lösungshilfe

Aufgabe 1. *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.

[4,3] oder jedes andere Feld mit mindestens zwei Zahlen und für das gilt, dass mindestens zwei Zahlen in der falschen Reihenfolge vorliegen.

Die Anweisungsreihenfolge ist (von 1 bis 14):

1 change wird auf true gesetzt
2 if: bestand.length > 1 wird zu true ausgewertet
3 while: change wird zu true ausgewertet
4 change wird auf false gesetzt
5 for: i wird initialisiert (i = 1)
6 for: i > 0 wird zu true ausgewertet
8 i1 wird auf 3 gesetzt
9 i2 wird auf 4 gesetzt
10 if: i1 < i2 wird zu true ausgewertet
11-13 4 und 3 vertauschen; change wird true
7 for: i wird angepasst (i = 0)

6 for: i > 0 wird zu false ausgewertet
3 while: change wird zu true ausgewertet
4 change wird auf false gesetzt
5 for: i wird initialisiert (i = 1)
6 for: i > 0 wird zu true ausgewertet
8 i1 wird auf 4 gesetzt
9 i2 wird auf 3 gesetzt
10 if: i1 < i2 wird zu false ausgewertet
7 for: i wird angepasst (i = 0)
6 for: i > 0 wird zu false ausgewertet
3 while: change wird zu false ausgewertet
14 bestand wird zurückgegeben

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.

Nein. Verzweigungsabdeckung erfordert zusätzlich ein Feld der Größe 1 als Eingabe.

d) Wie viele Pfade gibt es? Kurze Begründung.

- Es gibt theoretisch 5 Pfade:
  - 1 → 2 → 14
  - 1 → 2 → 3 → 14
  - 1 → 2 → 3 → 4 → 5 → 6 → 3 → 14
  - 1 → 2 → 3 → 4 → 5 → 6 → 8 → 9 → 10 → 7 → 6 → 3 → 14
  - 1 → 2 → 3 → 4 → 5 → 6 → 8→ 9 → 10 → 11→ 12→ 13 → 7 → 6 → 3 → 14
- Allerdings kommen 2. und 3. nie vor:
  - zu 2.) weil change == true
  - zu 3.) weil i > 0

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

```
public void myMeth(int a)
{
   if (a >= 0)
      a--;
   else
      a++;
}
```

Aufgabe 2. *Refactoring* (8 Punkte)

a) Überlegen Sie sich welche Probleme beim Verlagern einer Methode in eine andere Klasse (Move Method Refactoring) auftreten können. Geben Sie 5 Beispiele an.

Tip: beachten Sie auch Sichtbarkeiten und Vererbung!

Move Method Refactoring:

- Zugriff auf privaten Methoden oder Attribute der Ursprungsklasse
  - Sichtbarkeiten sinnvoll?
  - Refactoring der Sichtbarkeiten?
    - ⇨ Encapsulate field
    - ⇨ Move field
- Zugriff auf *package-weit sichtbare oder protected Methoden oder* Attribute o.k.
  - Methoden und Attribute gleich mitverschieben?
- Methode existiert bereits in anderer Klasse
  - Gleicher Name und Signatur
  - Evtl. Umbenennen
- Beachte beim Verschieben
  - Sichtbarkeit der Methode durch Clients
- private-Zugriffe:
  - nicht ohne weitere Anpassungen
- package:
  - nur innerhalb des Packages, Aufruf anpassen
  - ansonsten weitere Anpassungen nötig (z.B. Delegation)
- protected:
  - im gleichen Package oder in Oberklassen des Zugriffs verschiebbar
  - Aufrufe anpassen
- public:
  - Keine Probleme durch Sichtbarkeit
  - Aufrufe anpassen
- m() aus B nach E verschieben
  - Wenn E ebenfalls eine finale Methode m() mit gleicher Signatur besitzt
    - m() nicht nach E
- B ist abstrakte Klasse mit abstrakter Methode m(), oder Interface das m() definiert
  - m() nicht aus B herausnehmen
  - Methodendefinition m() in B, die auf E delegiert (falls B Zugriff auf E hat)

- Meist liegen solche Probleme nicht vor. Denn sonst würde die Methode in der Ursprungsklasse mehr Sinn machen, als in der Zielklasse.

Extract Method Refactoring:

- Zugriff auf lokale Variablen der ursprünglichen Methode
  - Mit verschieben
  - Nicht mit verschieben & Parameterübergabe
- Per Parameter übergebene lokale Variable
  - Falls sie in neuer Methode verändert wird
    - ⇨ Rückgabeparameter, return
- Per Parameter übergebene lokale Variablen
  - Falls mehr als eine in neuer Methode verändert werden

- ⇨ Klasse für Rückgabewerte schreiben (evtl. unschön)
- ⇨ Feld für Rückgabewerte anlegen (evtl. unschön)
- ⇨ Call by Reference

- Neue lokale Variable wird in extrahierter Methode erzeugt
  - Falls nötig Variable zurückgeben

- Wenn man derartige Probleme nicht auflösen kann, kann dies ein Indiz dafür sein, dass die Aufteilung der Codeteile vielleicht doch nicht optimal ist.

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.