WS 2016/17

Simon Kaiser, 354692
Philipp Hochmann, 356148
Felix Kiunke, 357322
Giacomo Klingen, 356778
Daniel Schleiz, 356092

Softwaretechnik
Aufgabenblatt 7

Aufgabe 7.1

a)

Ohne Betrachtung des Programmcodes wäre das Black-Box Verfahren eine geeignete Weise zum Testen. In dem Rahmen möchte man sich möglichst geschickt Eingabedaten überlegen, womit man die Methode testen könnte. Dazu ist es sinnvoll, sich Äquivalenzklassen für die Eingabedaten zu entwickeln, um mit so wenig wie möglich Eingaben die Methode so umfangreich wie es geht zu testen. Die Äquivalenzklassen von den Eingabedaten für diese Methode könnten wie folgt aussehen:

- $x < 0$
- $x > 0$
- $x = 0$

Insbesondere sollte die 0 geprüft werden, da unter Umständen in der Methode Divisionen vorkommen könnten, wobei 0 als Nenner problematisch wäre. Außerdem sollte man noch explizit Integer.MAX_VALUE und Integer.MIN_VALUE testen, da die Methode arithmetische Operationen verwenden könnte, welche Überläufe ausgelöst werden könnten, was das Ergebnis verfälschen würde. Da außerdem auch null in die Methode übergeben werden könnte, sollte geprüft werden, wie die Methode damit umgeht.

b)

- $x < 0$: Eingabemenge: { Integer.MIN_VALUE, -49783, -14, -1}. Neben dem oben genannten Randfall Integer.MIN_VALUE sollte auch noch -1 als Randfall betrachtet werden, da eventuell arithmetische Operationen durchgeführt werden, wodurch die Eingabe in den positiven Bereich rutscht. Außerdem sollen mit -14 und -49783 noch ein zufälliger “kleiner” und “großer” Wert getestet werden, da nur Randfallbetrachtungen nicht ausreichen.

- x > 0: Eingabemenge: { 1, 15, 59285, Integer.MAX_VALUE}. Neben dem oben genannten Randfall Integer.MAX_VALUE sollte auch noch 1 als Randfall betrachtet werden, da eventuell arithmetische Operationen durchgeführt werden, wodurch die Eingabe in den negativen Bereich rutscht. Außerdem sollen mit 15 und 59285 noch ein zufälliger "kleiner" und "großer" Wert getestet werden, da nur Randfallbetrachtungen nicht ausreichen.
- x = 0: Eingabemenge: { 0 }.

Zudem sollte auch noch null als Eingabe getestet werden.

Aufgabe 7.2

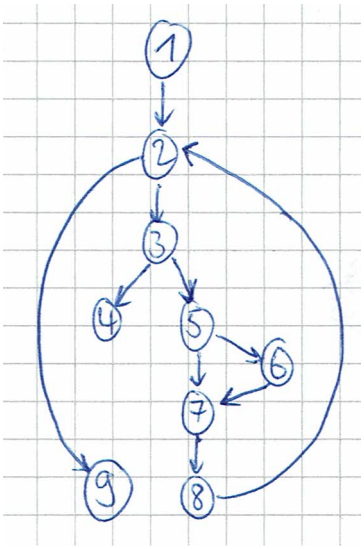
```
   public static int binary2decimal(String string) {
1:   int decimalNumber = 0;
     int exponent = 0;
     char currentChar;
     CharacterIterator stringIterator =
     new StringCharacterIterator(string);
     currentChar = stringIterator.last();

2:   while ((currentChar != CharacterIterator.DONE)
     && (decimalNumber < Integer.MAX_VALUE)) {
3:     if (currentChar != '0' && currentChar != '1')
4:       return -1;

5:     if (currentChar == '1')
6:       decimalNumber += Math.pow(2, exponent);

7:     exponent++;
8:     currentChar = stringIterator.previous();
     }
9:   return decimalNumber;
   }
```

Es soll die 1 die ganzen Zuweisungen vor der while-loop umfassen. Der Kontrollflussgraph sieht wie folgt aus:



Für die Anweisungsabdeckung reichen 2 Fälle:

- eine gültige Eingabe, z.B. "3"(Ablauf: 1,2,3,4)
- eine gültige Eingabe mit mindestens einer 1, zum Beispiel "1"(Ablauf: 1,2,3,5,6,7,8,2,9)

Für die Zweigabdeckung genügen auch 2 Fälle:

- eine ungültige Eingabe, z.B. "3"(Ablauf: 1,2,3,4)
- eine gültige Eingabe mit Einsen und 0en, z.B. "10"(Ablauf: 1,2,3,5,6,7,8,2,3,5,7,8,2,9)

Für die Pfadabdeckung genügen 10 Fälle:

- die Eingaben ”(leerer String),"2","02","12","00","01","10","11","0","1"

Aufgabe 7.3

a)

Der Zweigüberdeckungstest ist stärker, da nicht nur alle Anweisungen ausgeführt werden, sondern auch alle möglichen Entscheidungen (false & true) durchlaufen werden. Hier kann sich herausstellen, dass manche Verzweigungen nie benutzt werden und somit eine unnötige Abfrage darstellen.

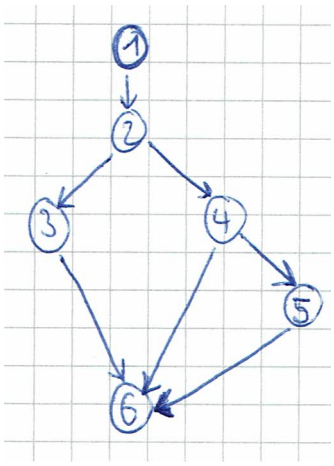
b)

Die folgende Funktion soll das Vorzeichen des übergebenen Parameters berechnen. Für eine Eingabe x sei $f(x)$ wie folgt definiert:

$$f(x) = \begin{cases} 1, & \text{falls } x > 0 \\ -1, & \text{falls } x < 0 \\ 0, & \text{falls } x = 0 \end{cases}$$

Nun die Methode f mit einem kleinen Fehler:

```
      public int f(int x){
1:        int res = 42;
2:        if(x<0){
3:            res = -1;
4:        } else if(x>0) {
5:            res = 1;
          }
6:        return res;
      }
```



1. Testfall: $x = -2$, überdeckt Anweisungen 1,2,3,6
2. Testfall: $x = 2$, überdeckt Anweisungen 1,2,4,5,6

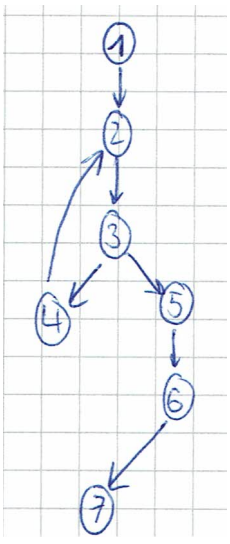
Zusammen erhält man eine volle Anweisungsüberdeckung, auf den Testeingaben erhält man die erwarteten Ergebnisse. Bei der Anweisungsüberdeckungstest fällt also der Fehler, dass die Methode bei Eingabe 0 ein falsches Ergebnis ausgibt, nicht auf.
Erst ein Zweigüberdeckungstest würde den Fehler aufzeigen, da so noch der implizite else-Teil, in welchem x==0 wäre, betrachtet wird. Für einen Zweigüberdeckungstest könnte man dann die Eingaben der beiden Testfälle aus dem Anweisungsüberdeckungstest übernehmen und noch zsätzlich einen dritten Testfall mit $x = 0$ einführen, bei welchem auffällt, dass das Ergebnis 42 ist, also nicht das erwartete Ergebnis 0.

c)

Die folgende Methode posTimesTwo soll als Eingabe einen int x bekommen, und das Zweifache von x zurückgeben, jedoch soll das Ergebnis strikt positiv, also immer echt größer 0 sein, weshalb bei negativen Eingaben zuerst mal -1 gerechnet wird.
Der Fehler im Programm liegt darin, dass die 0 nicht richtig berücksichtigt wurde, kommt zu einer Endlosschleife, jedoch sollte das Programm terminieren und die Rückgabe stets strikt positiv sein, zum Beispiel könnte man für die Eingabe 0 den Wert 1 returnen.

```
      public int posTimesTwo(int x){
1:        int y=x;
2:        while(true){
3:            if(y<=0){
4:                y=(-1)*y;
              } else {
5:                y=y*2;
6:                break;
              }
          }
7:        return y;
      }
```

Der Kontrollflussgraph sieht wie folgt aus:



Bereits mit einer Eingabe < 0 erreicht man schon eine Zweigabdeckung, zum Beispiel der Eingabe -1. Es ergibt sich die Abarbeitungsfolge 1,2,3,4,2,3,5,6,7.

Dabei werden alle Zweige im Kontrollflussgraphen besucht und man erhält das korrekte Ergebnis 2.

Bei einem Pfadabdeckungstest sieht dies anders aus: Man muss sich zunächst überlegen, welche Pfade möglich sind. Einmal wäre der Pfad aus dem Zweigabdeckungstest möglich, außerdem noch der Pfad 1,2,3,5,6,7 für eine Eingabe > 0, zum Beispiel 1. Dabei wird das Ergebnis 2 geliefert, dies ist ok.
Nun ist aber noch ein dritter Pfad möglich, nämlich für die Eingabe 0, dieser sieht so aus: 1,2,3,4,2,3,4,... , wobei sich 2,3,4 periodisch wiederholt und eine Endlosschleife vorliegt, welche vom Zweigabdeckungstest nicht zwingend gefunden wird. Somit ist die Pfadabdeckung mächtiger.