Programmierung 2 (SS 2021)
Universität des Saarlandes
Fakultät MI
Compiler Design Lab

Musterlösung 8
Semantik, Testen & Java

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Thorsten Klößner, M. Sc.
Julia Wichlacz, M. Sc.
Maximilian Fickert, M. Sc.

Im Vorlesungskalender finden Sie Informationen über die Kapitel des Skripts, die parallel zur Vorlesung bearbeitet werden sollen bzw. dort besprochen werden. Die Übungsaufgaben dienen der Vertiefung des Wissens, das in der Vorlesung vermittelt wird und als Vorbereitung auf Minitests und Klausur.
Weitere Aufgaben zu den Themen finden Sie jeweils am Ende der Skriptkapitel.
Die Schwierigkeitsgrade sind durch Steine des 2048-Spiels gekennzeichnet, von 512 „leicht“ bis 2048 „schwer“. 4096 steht für Knobelaufgaben.

Semantik & Testen

Aufgabe 8.0: Mixed Bag: Testen

1. Welche Fehler können mit black- bzw. white-box testing gefunden werden?
2. Ist die Abdeckung von Anweisungen und Zweigen ein gutes Kriterium für black-box Tests? Warum (nicht)?
3. Was ist der Unterschied zwischen Anweisungsabdeckung, Zweigabdeckung und Pfadabdeckung in einem C Programm? Können Sie Zweigabdeckung erreichen, ohne Anweisungsabdeckung zu erfüllen?
4. Können Sie ein Programm konstruieren, bei dem keine der Abdeckungsvarianten erfüllt werden kann?

Lösung

1. Funktionsbasiertes Testen (black-box testing) testet auf Korrektheit der Spezifikation, also ob das Programm in seiner Gesamtheit zu einem Input den erwartetem Output liefert. Es wird keinerlei Wissen über die Implementierung vorausgesetzt. Hierbei kann fehlende Funktionalität gefunden werden, was beim white-box testing nicht der Fall ist. Beim white-box testing werden auch einzelne Teile des Programms getestet, um eine möglichst hohe Abdeckung des geschriebenen Programmcodes zu bekommen. Fehler die hierdurch gefunden werden sind oft Fehler in internen Methoden die durch Black-Box Testen oft nur schwer zu finden sind.
2. Nein, da hier nicht die Abdeckung des Codes im Vordergrund steht, sondern die Entsprechung des Programmverhaltens im Vergleich zur Spezifikation. Es fehlt beim Black-Box Testen das Wissen über einzelne interne Methoden und Implementierungsdetails, eine hohe Abdeckung korreliert hierbei nicht mit der Genauigkeit der Black-Box Tests.

   Ein Beispiel:

   ```
   void f(int x) {
     if (x < 0)
       error();
     else
       ok();
   }
   ```

   Wird `f` innerhalb des Programms nur an genau einer Stelle aufgerufen, z.B. als `f(4)`, dann kann ein Black-Box Test den then-branch nie abdecken, und hätte in dieser Methode eine sehr geringe Abdeckung, obwohl die Spezifikation gut abgedeckt sein kann.
3. Bei der Anweisungsabdeckung, muss jede Anweisung im Programm mindestens einmal zur Ausführung kommen. Bei der Zweigabdeckung muss jeder Zweig mindestens einmal genommen werden. Die Pfadabdeckung fordert nicht nur, dass jeder Zweig und jede Anweisung ausgeführt wird, sondern auch, dass jeder mögliche Pfad durch das Programm genommen wird, also alle möglichen Sequenzen von Anweisungen ausgeführt werden. Es ist nicht möglich, alle Zweige abgedeckt zu haben, ohne alle Anweisungen auch zu erreichen. Hat man alle Zweige genommen, war man auch bei jeder Anweisung.

4. Für Programme die *dead code* enthalten, kann keine vollständige Abdeckung erreicht werden. Beispiel:

```
void f(int x) {
  if (false)
    x = 42;
  else
    x = 1;
  return;
}
```

Aufgabe 8.1: Assert

Schreiben Sie Zusicherungen für die mit ... gekennzeichnete Lücke, sodass die Funktion `supersum` immer terminiert.

```
1 int supersum(int a, int b, int c) {
2   assert(c >= 0);
3
4   ...
5
6   int x = 0;
7   for (int i = a; x < b; i = i + c) {
8     x = x + i;
9   }
10  return x;
11 }
```

Lösung

```
1 int supersum(int a, int b, int c) {
2   assert(c >= 0);
3
4   assert(
5     // das Programm terminiert direkt ohne die Schleife auszuführen
6     (b <= 0)  ||
7
8     // die Schleife endet bereits nach der ersten Iteration
9     (a >= b) ||
10
11    // i wird irgendwann positiv und x wird somit irgendwann größer als b
12    (c > 0)  ||
13
14    // x wird in jeder Iteration um den positiven Wert a erhöht
15    (c == 0 && a > 0)
16  );
17
18  int x = 0;
19  for (int i = a; x < b; i = i + c) {
20    x = x + i;
21  }
22  return x;
23 }
```

Tipp: Um möglichst detaillierte Fehlermeldungen zu bekommen, ist es bei Konjunktionen sinnvoll, mehrere Asserts zu schreiben, anstatt eines, das alle Bedingungen prüft. Wenn wir beispielsweise für die variablen x und y prüfen, ob x,y>=0, dann schreiben wir ‘assert(x>=0); assert(y>=0);‘ anstatt ‘assert(x>=0 && y>=0);‘.

Aufgabe 8.2: Abdeckung

Betrachten Sie folgendes Programm:

```
1 char* convert (int x){
2
3     char* string = null;
4     if (x == 1) {
5         string = "ONE";
6     }
7     else if (x == 2) {
8         string = "TWO";
9     }
10    return string;
11 }
```

1. Schreiben Sie eine Testsuite, die alle Anweisungen abdeckt.
2. Erweitern Sie Ihre Testsuite so, sodass nun auch alle Zweige abgedeckt werden.

Lösung

```
1 void test_one() {
2     char* one = convert(1);
3     assert(strcmp(one, "ONE") == 0);
4 }
5
6 void test_two() {
7     char* two = convert(2);
8     assert(strcmp(two, "TWO") == 0);
9 }
```

```
1 void test_three() {
2     char* empty = convert(3);
3     assert( empty == null);
4 }
```

Aufgabe 8.3: Abdeckung

Betrachten Sie folgendes Programm:

```
1  int substr(char * dest, int begin, int end, char * data) {
2    int delta = 0;
3    if (end < 0) {
4      delta = -end;
5    }
6
7    if (delta == 0) {
8      if (begin <= end) {
9        for (int i = begin; i < end; ++i) {
10         dest[i - begin] = data[i];
11       }
12     }
13     dest[end - begin] = '\0';
14   } else {
15     int i = begin;
16     if (data[i + delta] != '\0') {
```

```
17       for (; data[i + delta] != '\0'; ++i) {
18         dest[i - begin] = data[i];
19       }
20     }
21     dest[i - begin] = '\0';
22   }
23 }
```

1. Beschreiben Sie die Funktion des Programms in ein bis zwei Sätzen.

Lösung

Das Programm kopiert einen Substring von data nach dest beginnend bei begin bis ende. Ist der Index in end negativ so wird er relativ zum Ende des Strings interpretiert.

2. Schreiben Sie eine Testsuite, die alle Anweisungen abdeckt.

Lösung

```
 1 // end ist negativ
 2 void test_neg_end() {
 3   char data[16];
 4   char * input = "Testnachricht!";
 5   substr(data, 4, -1, input);
 6   printf("%s\n", data);
 7   assert(! strcmp("nachricht", data));
 8 }
 9
10 // end ist positiv
11 void test_pos_end() {
12   char data[16];
13   char * input = "Testnachricht!";
14   substr(data, 0, 4, input);
15   printf("%s\n", data);
16   assert(! strcmp("Test", data));
17 }
```

3. Erweitern Sie die Testsuite ggf., so dass alle Zweige abdeckt werden.

Lösung

Wir ergänzen folgende Tests

```
 1 void test_branch_pos_end_noloop() {
 2   char data[16];
 3   char * input = "Testnachricht!";
 4   substr(data, 5, 5, input);
 5   printf("%s\n", data);
 6   assert(! strcmp("", data));
 7 }
 8
 9 void test_branch_neg_end_noloop() {
10   char data[16];
11   char * input = "Testnachricht!";
12   substr(data, 4, -10, input);
13   printf("%s\n", data);
14   assert(! strcmp("", data));
15 }
```

Aufgabe 8.4: Funktionale Tests

Die im Skript in Beispiel 6.9 (S.161) angegebene Testsuite für das Unterprogramm `roots` deckt nicht alle Fälle ab. Schreiben Sie weitere Tests für die noch nicht behandelten Fälle.

256 25 1024 4

Lösung

- Das Polynom ist quadratisch und hat keine reellen Wurzeln.

```
1 void test_quadratic_none(void) {
2     double r[2];
3     int res = roots(1, 0, 1, r);
4     assert(res == 0);
5 }
```

- Das Polynom ist quadratisch und hat genau eine Wurzel.

```
1 void test_quadratic_one(void) {
2     double r[2];
3     int res = roots(3, 0, 0, r);
4     assert(res == 1 && r[0] == 0.0);
5 }
```

- Das Polynom ist quadratisch und hat zwei reelle Wurzeln.

```
1 void test_quadratic_two(void) {
2     double r[2];
3     int res = roots(1, 0, -1, r);
4     assert(res == 2);
5     assert((r[0] == 1.0 && r[1] == -1.0)
6         || (r[1] == 1.0 && r[0] == -1.0));
7 }
```

Aufgabe 8.5: Nach Spezifikation

1. Schreiben Sie ein C0 Programm, das folgende Spezifikation implementiert:

$$\begin{aligned} S = \{(\sigma,\sigma') | \sigma\, x > 1 &\implies \sigma'\, y = \sigma\, b \wedge \\ \sigma\, x < 0 &\implies \sigma'\, y = \sigma\, a \wedge \\ \sigma\, x \geq 0 \wedge \sigma\, x \leq 1 &\implies \sigma'\, y = \sigma\, b \cdot \sigma\, x + (1 - \sigma\, x) \cdot \sigma\, a\} \end{aligned}$$

2. Beweisen Sie, dass Ihr Programm korrekt ist. Machen Sie hierzu eine Fallunterscheidung über die Eingabe ($\sigma\, x > 1$, etc.) und führen Sie das Programm dann für jeden Fall mit der Semantik von C0 aus.

Lösung

1.
```
if (x > 1)
  y = b;
else if (x < 0)
  y = a;
else
  y = x * b + (1 - x) * a
```

2. Wir betrachten die drei möglichen Fälle einzeln. Dafür sei $(\sigma, \sigma') \in S$ beliebig aber fest gewählt.

- **Fall** $\sigma\, x > 1$
  Damit (σ, σ') in diesem Fall die Spezifikation erfüllt, muss nach der Ausführung gelten, dass $\sigma'\, y = \sigma b$.

  (a)
  $$[\text{IfTrue}]\ \frac{\sigma\, x > 1 \quad \text{(nach Fallannahme)}}{\langle \texttt{if (x > 1) y = b; else ...} \mid \sigma\rangle \to \langle \texttt{y = b;} \mid \sigma\rangle}$$

  (b)
  $$[\text{Assign}]\ \frac{}{\langle \texttt{y = b;} \mid \sigma\rangle \to \sigma[y \mapsto \sigma\, b]}$$

  Aus dem letzten Zustand $\sigma' = \sigma[y \mapsto \sigma\, b]$ folgt
  $$\sigma'\, y = \sigma\, b$$

- **Fall** $\sigma\, x < 0$
  Damit (σ, σ') in diesem Fall die Spezifikation erfüllt, muss nach der Ausführung gelten, dass $\sigma'\, y = \sigma\, a$.

  (a)
  $$[\text{IfFalse}]\ \frac{\sigma\, x < 0 \quad \text{(nach Fallannahme)}}{\langle \texttt{if (x > 1) y = b; else if (x < 0) y = a; else ...} \mid \sigma\rangle \to \langle \texttt{if (x < 0) y = a; else ...} \mid \sigma\rangle}$$

  (b)
  $$[\text{IfTrue}]\ \frac{\sigma\, x < 0 \quad \text{(nach Fallannahme)}}{\langle \texttt{if (x < 0) y = a; else ...} \mid \sigma\rangle \to \langle \texttt{y = a;} \mid \sigma\rangle}$$

  (c)
  $$[\text{Assign}]\ \frac{}{\langle \texttt{y = a;} \mid \sigma\rangle \to \sigma[y \mapsto \sigma\, a]}$$

  Aus dem letzten Zustand $\sigma' = \sigma[y \mapsto \sigma\, a]$ folgt
  $$\sigma' y = \sigma\, a$$

- **Fall** $\sigma\, x \geq 0 \wedge \sigma\, x \leq 1$
  Damit (σ, σ') in diesem Fall die Spezifikation erfüllt, muss nach der Ausführung gelten, dass $\sigma' y = \sigma\, x \cdot \sigma\, b + (1 - \sigma\, x) \cdot \sigma\, a$.

  (a)
  $$[\text{IfFalse}]\ \frac{\sigma\, x \geq 0 \wedge \sigma\, x \leq 1 \quad \text{(nach Fallannahme)}}{\langle \texttt{if (x > 1) y = b; else if (x < 0) y = a; else ...} \mid \sigma\rangle \to \langle \texttt{if (x < 0) y = a; else ...} \mid \sigma\rangle}$$

  (b)
  $$[\text{IfFalse}]\ \frac{\sigma\, x \geq 0 \wedge \sigma\, x \leq 1 \quad \text{(nach Fallannahme)}}{\langle \texttt{if (x < 0) y = a; else y = x * b + (1 - x) * a;} \mid \sigma\rangle \to \langle \texttt{y = x * b + (1 - x) * a;} \mid \sigma\rangle}$$

  (c)
  $$[\text{Assign}]\ \frac{}{\langle \texttt{y = x * b + (1 - x) * a;} \mid \sigma\rangle \to \sigma[y \mapsto \sigma\, x \cdot \sigma\, b + (1 - \sigma\, x) \cdot \sigma\, a]}$$

  Aus dem letzten Zustand $\sigma' = \sigma[y \mapsto \sigma\, x \cdot \sigma\, b + (1 - \sigma\, x) \cdot \sigma\, a]$ folgt
  $$\sigma' y = \sigma\, x \cdot \sigma\, b + (1 - \sigma\, x) \cdot \sigma\, a$$

Aufgabe 8.6: Spezifikation zum Programm schreiben

Schreiben Sie eine formale Spezifikation zu den unten stehenden Programmen. Geben Sie zudem die Menge der kompatiblen Variablen an.

1.
```
if (n < m) {
    k = n;
} else {
    k = m;
}
```

2.
```
x = n/m;
if(x <= 0){
    x = -n/m
}
```

3.
```
if (x > 1) {
    y = 4 * x + 2;
    z = y / 2;
    x = y + z + 1;
    x--;
    x = x / 3;
} else {
    x++;
}
```

Lösung

1. Das Programm speichert $\min\{m, n\}$ in k:
   $\{(\sigma, \sigma') | \sigma' k \leq \sigma m \wedge \sigma' k \leq \sigma n \wedge (\sigma' k = \sigma n \vee \sigma' k = \sigma m)\}$, $K = \{n, m\}$
2. Das Programm berechnet den Betrag des Bruches $\frac{n}{m}$:
   $\{(\sigma, \sigma') | \sigma m \neq 0 \Rightarrow (((\sigma m < 0 \wedge \sigma n \geq 0) \vee (\sigma m \geq 0 \wedge n < 0)) \wedge \sigma' x = -\frac{\sigma n}{\sigma m}) \vee (((\sigma m \geq 0 \wedge \sigma n \geq 0) \vee (\sigma m < 0 \wedge \sigma n < 0)) \wedge \sigma' x = \frac{\sigma n}{\sigma m})\}$, $K = \{n, m\}$

   Alternativ:
   $\{(\sigma, \sigma') | \sigma m \neq 0 \Rightarrow \sigma' x = |\frac{\sigma n}{\sigma m}|\}$, $K = \{n, m\}$
3. $\{(\sigma, \sigma') | (\sigma x \leq 1 \wedge \sigma' x = \sigma x + 1) \vee (\sigma x > 1 \wedge \sigma' x = \sigma' z = 2 * \sigma x + 1 \wedge \sigma' y = 4 * \sigma x + 2)\}$, $K = \{\}$

Aufgabe 8.7: Programm zu Spezifikation

1. Konstruieren Sie zu den folgenden Spezifikationen jeweils ein *C0b*-Programm, das die Spezifikation erfüllt.

   (a) $S_1 = \{(\sigma, \sigma') \mid \sigma' x = \sigma x + \sigma y\}$

   (b) $S_2 = \{(\sigma, \sigma') \mid \sigma' x = (\sigma x)^{100}\}$

   (c) $S_3 = \{(\sigma, \sigma') \mid \sigma' x > \sigma' y\}$

   (d) $S_4 = \{(\sigma, \sigma') \mid \sigma' x = (\sigma x)!\}$ mit $n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{k=1}^{n} k$

   (e) $S_5 = \{(\sigma, \sigma') \mid \sigma' g = \mathrm{ggT}(\sigma x, \sigma y)\}$ mit ggT = größter gemeinsamer Teiler

2. Konstruieren Sie nun ein Progamm, welches für folgende V, N, K die Spezifikation $S_{V,N,K}$ erfüllt:

   (a) V= $x \neq 0$ N= $d = \frac{y}{x}$ K= $\{x, y\}$

   (b) V= true N= $d = \frac{y}{x} \vee x = 0$ K= $\{x, y\}$

   (c) V= true N= $a = x + y$ K= $\{\}$

   (d) V= true N= $a = X + Y$ K= $\{\}$

3. Überlegen Sie sich ein Programm, welches die Zusicherung 2.c erfüllt, jedoch nicht die folgende Spezifikation:
   V= true N= $a = x + y$ K= $\{x, y\}$
   Erfüllt jedes Programm, welches 2.d erfüllt, auch die hier gegebene Spezifikation?

Lösung

1. (a)

```
x = x + y;
```

   (b)

```
{
    int y;
    int x0;
    x0 = x;
    y = 100;
    while (y != 1) {
        x = x * x0;
        y = y - 1;
    }
}
```

   (c)

```
x = y + 1;
```

   (d)

```
{
    int y;
    y = x - 1;
    while (y != 0) {
        x = x * y;
        y = y - 1;
    }
}
```

   (e)

```
while(y != 0) {
    int h;
    h = x % y;
    x = y;
    y = h;
}
g = x;
```

2. (a)

```
d = y / x;
```

(b)

```
if (x == 0)
    ;
else
    d = y / x;
```

(c)

```
a = x + y;
```

(d)

```
a = x + y;
```

3.

```
a = x + y;
x = x + 1;
y = y + 1;
```

Dieses Programm erfüllt $S_{V,N,K}$, da $x, y \notin K$ und somit während der Ausführung verändert werden dürfen.

Ja, da die Verwendung von Großbuchstaben für Variablen impliziert, dass diese in der Kompatiblitätsmenge enthalten sind.

Aufgabe 8.8: Erstellen von Spezifikationen

Um den Umgang mit der C0-Semantik zu üben, sollen Sie die unten stehenden Programme formal beschreiben, indem Sie die passenden Spezifikationen angeben. Versuchen Sie dabei, die Spezifikation möglichst kurz zu formulieren.

1.

```
1         if (n >= 0) {
2             n = n;
3         } else {
4             n = (-7) * 3 + n * (-1) + 21;
5         }
```

2.

```
1         while (b > 0) {
2             x += a;
3             b--;
4         }
```

3.

```
1          if (a < b) {
2              if (a < 10) {
3                  max = a;
4              } else {
5                  max = 10;
6              }
7          } else {
8              if (b < 10) {
9                  max = b;
10             } else {
11                 max = 10;
12             }
13         }
```

Lösung

1. $S_{abs} = \{(\sigma, \sigma') \mid \sigma n = |\sigma' n|\}$
2. $S_{mul} = \{(\sigma, \sigma') \mid (\sigma b > 0 \Rightarrow \sigma' x = \sigma x + \sigma a * \sigma b \wedge \sigma' b = 0) \wedge (\sigma b \leq 0 \Rightarrow \sigma' x = \sigma x \wedge \sigma b = \sigma' b)\}$
3. $S_{min} = \{(\sigma, \sigma') \mid ((\sigma a \geq 10 \wedge \sigma b \geq 10) \Rightarrow \sigma' max = 10) \wedge ((\sigma a < 10 \vee \sigma b < 10) \Rightarrow \sigma' max = min(\sigma a, \sigma b))\}$

   Dabei soll $min(a, b)$ der Intuition nach wie folgt definiert sein: $min(a, b) = \begin{cases} a & \text{wenn } a < b \\ b & \text{sonst} \end{cases}$

Aufgabe 8.9: Testsuite

Betrachten Sie folgendes Programm:

```
1 public class Compute {
2   public int rec(int n, int x, int y) {
3     if (n == 0) {
4       return x + y;
5     }
6     if (y == 0) {
7       return x;
8     }
9     return rec(n - 1, rec(n, x, y - 1), rec(n, x, y - 1) + y);
10  }
11 }
```

1. Schreiben Sie eine Testsuite, die alle Anweisungen abdeckt.
2. Erweitern Sie Ihre Testsuite gegebenenfalls, sodass alle Zweige abgedeckt werden.
3. Beschreiben Sie kurz die Funktion des Programms.

Lösung

1.

```
1 class Tests {
2   Compute comp = new Compute();
3
4   @Test
5   public void n_zero_test() {
6     int result = comp.rec(0, 1, 2);
7     int expected = 3;
8     assertEquals(expected, result);
9   }
10
11  @Test
12  public void y_zero_test() {
13    int result = comp.rec(100, 5, 0);
14    int expected = 5;
15    assertEquals(expected, result);
16  }
17
18  @Test
19  public void y_not_zero_test() {
20    int result = comp.rec(1, 2, 3);
21    int expected = 27;
22    assertEquals(expected, result);
23  }
24
25  @Test
26  public void one_step_test() {
27    int result = comp.rec(1, 2, 0);
28    int expected = 2;
29    assertEquals(expected, result);
30  }
31 }
```

2. In diesem Fall ist es unmöglich eine vollständige Anweisungsabdeckung zu erhalten, ohne bereits alle Zweige zu erreichen. Daher genügen die vorhandenen Tests auch für die Zweigabdeckung.

3. Die gegebene Klasse kann verwendet werden, um die sogenannte Sudanfunktion zu berechnen. Ähnlich wie die Ackermannfunktion können wir diese mathematisch nicht primitiv rekursiv darstellen. Die Funktionswerte wachsen schneller als beliebige Exponentialfunktion, so hat `rec(1, 1, 2)` lediglich den Wert 8, aber `rec(2, 1, 2)` bereits den Wert 10228.

Java

Aufgabe 8.10: Java Einstieg

Unter Materialien finden Sie eine Datei *ExerciseSheet.zip*, die ein Java-Projekt beinhaltet. Installieren Sie unzip, indem Sie den Befehl `sudo pacman -S unzip` auf der Konsole ausführen. Das sudo Passwort für die vm lautet `prog2`. Entpacken Sie als nächstes das Projekt mit dem Befehl unzip ExerciseSheet.zip, den Sie ebenfalls auf der Konsole ausführen müssen. Öffnen Sie nun den entpackten Ordner in Visual Studio Code.

1. Legen Sie eine Klasse `MyCounter` an. Verwenden Sie dazu Visual Studio Code, indem Sie eine neue Datei `MyCounter.java` anlegen. Achten Sie darauf, dass Sie im selben Ordner liegt wie die vorhandene Datei `MyFirstJava.java`. In der ersten Zeile sollte `package firstJavaExercise;` stehen.
2. Fügen Sie Ihrer Klasse eine Methode `countToN` ein, die die Zahlen von `0` bis `n` auf der Konsole ausgibt. Das verwendete `n` soll nicht als Argument der Methode übergeben werden, sondern im Konstruktor der Klasse.
3. Nutzen Sie jetzt die `main` Methode der gegebenen Klasse `MyFirstJava` um ein Objekt der Klasse `MyCounter` anzulegen.
4. Rufen Sie auf dem gerade erzeugten Objekt die Methode `countToN` auf.
5. Was passiert, wenn jemand eines Ihrer Objekte mit `MyCounter(-1)` erzeugt und dann `countToN` aufruft? Beheben Sie eventuelle Probleme mit Ihrer Implementierung, indem Sie eine sinnvolle Invariante für die Nutzung Ihrer Klasse definieren.

Lösung

Datei `MyFirstJava.java`:

```
1  package firstJavaExercise;
2
3  public class MyFirstJava {
4    public static void main(String[] args) {
5      MyCounter counter = new MyCounter(10);
6      counter.countToN();
7    }
8  }
```

Datei `MyCounter.java`:

```
1  package firstJavaExercise;
2
3  public class MyCounter {
4    private int n;
5
6    public MyCounter(int n){
7      assert n>=0 : "MyCounter␣expects␣a␣non-negative␣value!";
8      this.n = n;
9    }
10
11   public void countToN() {
12     for (int i=0; i<=this.n; i++){
13       System.out.println(i);
14     }
15   }
16
17 }
```

Hinweis: Damit Assertions beim Ausführen beachtet werden, muss Visual Studio Code entsprechend konfiguriert werden. Erstellen Sie dazu im Ordner *ExerciseSheet* ein neuer Ordner *.vscode* mit einer Datei *launch.json*. Folgendes ist der Inhalt von *launch.json*:

```
{
  "configurations": [
    {
      "type": "java",
      "name": "Launch firstJavaExercise",
      "request": "launch",
      "mainClass": "firstJavaExercise.MyFirstJava",
      "projectName": "ExerciseSheet",
      "vmArgs": "-enableassertions"
    }
  ]
}
```

Aufgabe 8.11: Mixed Bag

1. Welche Sichtbarkeitsmodifizierer gibt es in Java? Was sind die Eigenschaften dieser? Welchen Sichtbarkeitsbereich hat eine Variable die in einer Klasse als Feld definiert ist, aber keinen expliziten Modifizierer verwendet? Wie können Sie ihre Vermutung überprüfen?

2. Was sind Basisdatentypen in Java? Welche Typen werden automatisch in andere konvertiert, welche nicht?

3. Wozu wird in Java die Methode *equals* verwendet? Warum reicht die Nutzung von == in vielen Fällen nicht aus?

Lösung

1. Die Sichtbarkeitsmodifizierer sind `public`, `private` und `protected`. Abbildung 1 zeigt die Eigenschaften.

	Klasse	Paket	Unterklasse	Rest
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	–
kein Modifier spezifiziert	✓	✓	–	–
`private`	✓	–	–	–

Abbildung 1: Sichtbarkeitsmodifizierer in Java

   Wie aus der Tabelle ersichtlich ist, sind Variablen ohne Modifizierer ein Sonderfall, der den Zugriff innerhalb der Klasse und innerhalb des Pakets zulässt, sonst aber die Variable abschirmt. Herausfinden kann man dies, indem man im Internet, Skript oder Büchern nachschlägt, oder auch einfach ein kleines Javaprojekt anlegt. Dieses beinhaltet eine Klasse mit dem Feld, eine Klasse im selben Paket und zwei Klassen in einem anderen Paket wobei eine der Klassen eine Unterklasse ist. Nun kann man versuchen von all diesen Orten auf die Variable zuzugreifen, und somit experimentell den Sichtbarkeitsbereich bestimmen.

2. Die Basisdatentypen und möglichen Konvertierungen sind in Abbildung 2 dargestellt.

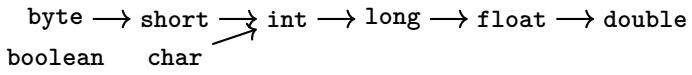

Abbildung 2: Automatische Typumwandlung zwischen Basistypen

3. Die Methode `equals` dient dazu, Objekt auf semantische Gleichheit testen zu können und dabei selbst zu bestimmen, wann zwei Objekte als gleich gelten sollen.

```
Fraction A = new Fraction(1, 2);
Fraction B = new Fraction(2, 4);

boolean eq = (A == B); \\ wertet zu false aus
```

Im oben stehenden Beispiel stellen beide Brüche den gleichen Zahlenwert da, der Vergleich mit == wertet aufgrund der unterschiedlichen Objekte `A` und `B` allerdings zu `false` aus. Eine selbst geschriebene `equals`-Methode kann hier aber diese beiden Objekte als gleich werten und `true` zurückliefern.

Aufgabe 8.12: JUnit-Taschenrechner

1. Implementieren Sie einen einfachen Taschenrechner in Java! Ihr Programm soll drei Argumente erhalten:
   - `int op` stellt den Operator dar, mit dem Ihr Programm das Ergebnis `res` berechnen soll.
   - `int l` und `int r` stellen Ihre Operanden dar. Diese dürfen Werte zwischen 0 und 10000 annehmen.

   Es gibt die folgenden Operatoren:

Operator	Ergebnis
1	$res = l + r$
2	$res = l - r$
3	$res = l * r$
4	$res = l/r$

   Ihr Programm soll als Methode implementiert werden, hierfür ist bereits folgendes Codegerüst gegeben:

```java
public static int compute(int op, int l, int r)
  throws IllegalArgumentException {

    res = 0;
    ...
    return res;
}
```

   Falls Ihr Programm ungültige Werte übergeben bekommt, soll es eine `IllegalArgumentException` werfen. Beachten Sie, dabei alle sinnvollen Randfälle abzudecken (z.B. eine ungültige Zahl, die als Operator übergeben wurde).

2. Schreiben sie für das in a) beschriebene Programm eine JUnit-Testsuite, welche maximale Anweisungsabdeckung (Code Coverage) erreicht.

Lösung

1.

```java
package calculator;

public class Calculator {

    public static int compute(int op, int l, int r)
    throws IllegalArgumentException {
        int res = 0;
        if (l < 0 || r < 0 || l > 10000 || r > 10000) {
            throw new IllegalArgumentException("Werte␣ist␣zu␣gross/klein");
        }
        if (op == 4 && r == 0) {
            throw new IllegalArgumentException("Division␣durch␣0␣" +
                "ist␣nicht␣erlaubt");
        }
        switch (op) {
            case 1:
                res = l + r;
                break;
```

```
19              case 2:
20                  res = l - r;
21                  break;
22              case 3:
23                  res = l * r;
24                  break;
25              case 4:
26                  res = l / r;
27                  break;
28              default:
29                  throw new IllegalArgumentException("Op existiert nicht");
30          }
31          return res;
32      }
33 }
```

2.

```java
package calculator;

import static org.junit.Assert.assertEquals;

import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

public class CalculatorTests {


    @Rule
    public ExpectedException thrown = ExpectedException.none();

    @Test
    public void test_basic_op() {

        assertEquals(42, Calculator.compute(1,32,10));
        assertEquals(22, Calculator.compute(2, 32, 10));
        assertEquals(120, Calculator.compute(3, 12, 10));
        assertEquals(10, Calculator.compute(4, 100, 10));
    }

    @Test
    public void test_div_invalid() {

        thrown.expect(IllegalArgumentException.class);
        Calculator.compute(4, 20, 0);
    }

    @Test
    public void test_invalid_l_range() {

        thrown.expect(IllegalArgumentException.class);
        Calculator.compute(3, -10, 2);
    }

    @Test
    public void test_invalid_op_range() {

        thrown.expect(IllegalArgumentException.class);
        Calculator.compute(7, 10, 4);
    }
}
```

Aufgabe 8.13: Phones

Dieter Schlau hat eine Kiste voller Handys gefunden und möchte diese nun katalogisieren um ihre Eigentümer zu finden. Dazu hat er eine Klasse `Mobilephone` und eine Klasse `Smartphone` erstellt. Helfen sie ihm, diese Klassen zu vervollständigen:

1. Fügen sie in beiden Klassen einen sinnvollen Konstruktor hinzu. (Die Akkulaufzeit eines Mobiltelefons liegt bei 1500 Minuten, die eines Smartphones bei 800 Minuten.)

2. Erstellen sie für die Klasse `Mobilephone` zwei Methoden zum hinzufügen und löschen von Kontakten. Dabei wird bei jedem dieser Vorgänge 2 Minuten des Akkus verbraucht.

3. Erstellen sie für die Klasse `Smartphone` zwei Methoden, um Apps zu installieren und deinstallieren. Eine Installation benötigt 5 Minuten der Akkulaufzeit, eine Deinstallation nur 2 Minuten.

```java
public class Mobilephone {
  int contacts;
  int number;
  int battery;
  //...
}
```

```java
public class Smartphone extends Mobilephone {
  int apps;
  //...
}
```

Lösung

Mobilephone.java

```java
public class Mobilephone {
  int contacts;
  int number;
  int battery;

  public Mobilephone(int contacts, int number){
    this.contacts = contacts;
    this.number = number;
    this.battery = 1500;
  }

  public void addContact(){
    contacts++;
    battery -= 2;
  }

  public void removeContact(){
    contacts--;
    battery -= 2;
  }
}
```

Smartphone.java

```java
public class Smartphone extends Mobilephone {
  int apps;

  public Smartphone (int contacts, int number, int apps){
    super(contacts, number);
    this.apps = apps;
    this.battery = 800;
  }

  public void installApp(){
    apps++;
    battery -= 5;
  }

  public void deinstallApp(){
    apps--;
    battery -= 2;
  }
}
```

Aufgabe 8.14: Modulo in Java

In dieser Aufgabe lernen Sie eine Besonderheit über den Modulusoperator in Java. Nehmen wir an, wir betrachten natürliche Zahlen modulo n, also die Zahlen im Interval $[0, n-1]$. Für $n = 10$ erwarten wir, dass $3-5 \equiv 8 \mod 10$ gilt, da $-2 - \left\lfloor \frac{-2}{10} \right\rfloor \cdot 10 = 8$. Was liefert die Berechung `(3-5) % 10` in Java? Geben Sie einen Ausdruck an, der zu einer möglicherweise negativen Zahl a den korrekten, positiven Modulus bezüglich n liefert. Der Ausdruck soll beispielsweise für $n = 10$ und $a = -2$ den Wert 8 liefern.

Lösung

`(3-5) % 10` ergibt -2 in Java.

```java
/**
 * Gibt a mod n zurück.
 *
 * Berechnet den standard Modulo, falls a positiv ist,
 * ansonsten wandelt er die negative Zahl in einen Positiven Modulo.
 *
 * @param a - Dividend
 * @param n - Divisor
 * @return
 */
public static int modMinus(int a, int n) {
  while (a < 0) {
            a = a + n;
        }

  return a % n;
}
```