Einzelprüfung „Datenbanksysteme / Softwaretechnologie (vertieft)“

Einzelprüfungsnummer 66116 / 2019 / Herbst

Thema 2 / Teilaufgabe 1 / Aufgabe 2

(*Assertions*)

Stichwörter: Formale Verifikation

Methoden in Programmen funktionieren nicht immer für alle möglichen Eingaben - klassische Beispiele sind die Quadratwurzel einer negativen Zahl oder die Division durch Null. Zulässige (bzw. in negierter Form unzulässige) Eingabewerte sollten dann spezifiziert werden, was mit Hilfe sog. assertions geschehen kann. Assertions prüfen zur Laufzeit den Wertebereich der Parameter einer Methode, bevor der Methodenkörper ausgeführt wird. Wenn ein oder mehrere Argumente zur Laufzeit unzulässig sind, wird eine Ausnahme geworfen.

Betrachten Sie das folgende Java-Programm:

```java
public static double[][] magic(double[][] A, double[][] B, int m) {
  double[][] C = new double[m][m];
  for (int i = 0; i < m; i++)
    for (int j = 0; j < m; j++)
      for (int k = 0; k < m; k++)
        C[i][j] += A[i][k] * B[k][j];
  return C;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/Assertion.java`

(a) Beschreiben Sie kurz, was dieses Programm tut.

Lösungsvorschlag

Das Programm erzeugt ein neues zweidimensionales Feld mit $m \times m$ Einträgen.

Zelle an Kreuzung Zeile Z und Spalte S =
$A[\text{1. Wert in Z}] \times B[\text{1. Wert in S}] +$
$A[\text{2. Wert in Z}] \times B[\text{2. Wert in S}] +$
$A[\text{3. Wert in Z}] \times B[\text{3. Wert in S}]$

Zelle an Kreuzung 1. Zeile 1. Spalte =
$1 \times 11 +$
$2 \times 16 +$
3×17

A

1	2	3
4	5	6
7	8	9

B

11	12	13
16	15	14
17	18	19

`magic(A, B, 3);`:

94	96	98
226	231	236
358	366	374

```
C[0][0] = C[0][0] + A[0][0] * B[0][0] = 0.0 + 1.0 * 11.0 = 11.0
C[0][0] = C[0][0] + A[0][1] * B[1][0] = 11.0 + 2.0 * 16.0 = 43.0
C[0][0] = C[0][0] + A[0][2] * B[2][0] = 43.0 + 3.0 * 17.0 = 94.0

C[0][1] = C[0][1] + A[0][0] * B[0][1] = 0.0 + 1.0 * 12.0 = 12.0
C[0][1] = C[0][1] + A[0][1] * B[1][1] = 12.0 + 2.0 * 15.0 = 42.0
C[0][1] = C[0][1] + A[0][2] * B[2][1] = 42.0 + 3.0 * 18.0 = 96.0

C[0][2] = C[0][2] + A[0][0] * B[0][2] = 0.0 + 1.0 * 13.0 = 13.0
C[0][2] = C[0][2] + A[0][1] * B[1][2] = 13.0 + 2.0 * 14.0 = 41.0
C[0][2] = C[0][2] + A[0][2] * B[2][2] = 41.0 + 3.0 * 19.0 = 98.0

C[1][0] = C[1][0] + A[1][0] * B[0][0] = 0.0 + 4.0 * 11.0 = 44.0
C[1][0] = C[1][0] + A[1][1] * B[1][0] = 44.0 + 5.0 * 16.0 = 124.0
C[1][0] = C[1][0] + A[1][2] * B[2][0] = 124.0 + 6.0 * 17.0 = 226.0

C[1][1] = C[1][1] + A[1][0] * B[0][1] = 0.0 + 4.0 * 12.0 = 48.0
C[1][1] = C[1][1] + A[1][1] * B[1][1] = 48.0 + 5.0 * 15.0 = 123.0
C[1][1] = C[1][1] + A[1][2] * B[2][1] = 123.0 + 6.0 * 18.0 = 231.0

C[1][2] = C[1][2] + A[1][0] * B[0][2] = 0.0 + 4.0 * 13.0 = 52.0
C[1][2] = C[1][2] + A[1][1] * B[1][2] = 52.0 + 5.0 * 14.0 = 122.0
C[1][2] = C[1][2] + A[1][2] * B[2][2] = 122.0 + 6.0 * 19.0 = 236.0

C[2][0] = C[2][0] + A[2][0] * B[0][0] = 0.0 + 7.0 * 11.0 = 77.0
C[2][0] = C[2][0] + A[2][1] * B[1][0] = 77.0 + 8.0 * 16.0 = 205.0
C[2][0] = C[2][0] + A[2][2] * B[2][0] = 205.0 + 9.0 * 17.0 = 358.0

C[2][1] = C[2][1] + A[2][0] * B[0][1] = 0.0 + 7.0 * 12.0 = 84.0
C[2][1] = C[2][1] + A[2][1] * B[1][1] = 84.0 + 8.0 * 15.0 = 204.0
C[2][1] = C[2][1] + A[2][2] * B[2][1] = 204.0 + 9.0 * 18.0 = 366.0

C[2][2] = C[2][2] + A[2][0] * B[0][2] = 0.0 + 7.0 * 13.0 = 91.0
C[2][2] = C[2][2] + A[2][1] * B[1][2] = 91.0 + 8.0 * 14.0 = 203.0
C[2][2] = C[2][2] + A[2][2] * B[2][2] = 203.0 + 9.0 * 19.0 = 374.0
```

(b) Implementieren Sie drei nützliche Assertions, die zusammen verhindern, dass das Programm abstürzt.

Lösungsvorschlag

```java
private static void assert2DArray(double[][] a, int m) {
  assert a.length < m : "Das 2D-Feld muss mindestens m Zeilen/Felder haben.";
  for (int i = 0; i < a.length; i++) {
    double[] row = a[i];
    assert row.length < m:
     ↪  "Jede Zeile im 2D-Feld muss mindestens m Einträge/Spalten haben.";
  }
}

public static double[][] magicWithAssertions(double[][] A, double[][] B, int m)
 ↪  {
  assert m < 0: "m darf nicht negativ sein.";
  assert2DArray(A, m);
  assert2DArray(B, m);
  double[][] C = new double[m][m];
  for (int i = 0; i < m; i++)
    for (int j = 0; j < m; j++)
      for (int k = 0; k < m; k++) {
        C[i][j] += A[i][k] * B[k][j];
      }
  return C;
}

/**
 * Duplikat der magic-Methode mit einer println-Anweisung, um die Funktionsweise
 * besser untersuchen zu können.
 *
 * @param A Ein zwei-dimensionales Feld des Datentyps double. Sollte mindestens
 *          m x m Einträge haben.
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/Assertion.java`

(c) Wann und warum kann es sinnvoll sein, keine expliziten Assertions anzugeben? Erläutern Sie, warum das potentiell gefährlich ist.

Lösungsvorschlag

Bei sicherheitskritischen Anwendungen, z. B. bei Software im Flugzeug, kann es möglicherweise besser sein, einen Fehler zu ignorieren, als die Software abstürzen zu lassen. In Java beispielsweise sind die Assertions standardmäßig deaktiviert und müssen erst mit `-enableassertions` aktiviert werden.

(d) Assertions können wie oben sowohl für Vorbedingungen am Anfang einer Methode als auch für Nachbedingungen am Ende einer Methode verwendet werden. Solche Spezifikationen bilden dann sog. Kontrakte. Kontrakte werden insbesondere auch statisch verwendet, d. h. nicht zur Laufzeit. Skizzieren Sie ein sinnvolles Anwendungsgebiet und beschreiben Sie kurz den Vorteil der Verwendung von Kontrakten in diesem Anwendungsgebiet.

Lösungsvorschlag

a

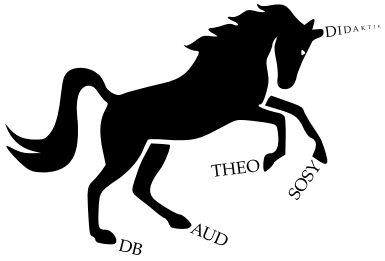
[a] https://de.wikipedia.org/wiki/Design_by_contract

Additum: Die komplette Klasse

```
public class Assertion {
  public static double[][] magic(double[][] A, double[][] B, int m) {
    double[][] C = new double[m][m];
    for (int i = 0; i < m; i++)
      for (int j = 0; j < m; j++)
        for (int k = 0; k < m; k++)
          C[i][j] += A[i][k] * B[k][j];
    return C;
  }

  private static void assert2DArray(double[][] a, int m) {
    assert a.length < m : "Das 2D-Feld muss mindestens m Zeilen/Felder haben.";
    for (int i = 0; i < a.length; i++) {
      double[] row = a[i];
      assert row.length < m:
       ↪  "Jede Zeile im 2D-Feld muss mindestens m Einträge/Spalten haben.";
    }
  }

  public static double[][] magicWithAssertions(double[][] A, double[][] B, int m) {
    assert m < 0: "m darf nicht negativ sein.";
    assert2DArray(A, m);
    assert2DArray(B, m);
    double[][] C = new double[m][m];
    for (int i = 0; i < m; i++)
      for (int j = 0; j < m; j++)
        for (int k = 0; k < m; k++) {
          C[i][j] += A[i][k] * B[k][j];
        }
    return C;
  }

  /**
   * Duplikat der magic-Methode mit einer println-Anweisung, um die Funktionsweise
   * besser untersuchen zu können.
   *
   * @param A Ein zwei-dimensionales Feld des Datentyps double. Sollte mindestens
   *          m x m Einträge haben.
   * @param B Ein zwei-dimensionales Feld des Datentyps double. Sollte mindestens
   *          m x m Einträge haben.
   * @param m Ein nicht negative Ganzzahl.
   *
   * @return Ein zwei-dimensionales Feld mit m x m Einträgen.
   */
  public static double[][] magicPrint(double[][] A, double[][] B, int m) {
    assert m < 0: "m darf nicht negativ sein.";
```

```
    assert2DArray(A, m);
    assert2DArray(B, m);
    double[][] C = new double[m][m];
    for (int i = 0; i < m; i++)
      for (int j = 0; j < m; j++)
        for (int k = 0; k < m; k++) {
          double altesC = C[i][j];
          C[i][j] += A[i][k] * B[k][j];

          ↪ System.out.println(String.format("C[%d][%d] = C[%d][%d] + A[%d][%d] * B[%d][%d] = %s +
          ↪ i, j, i,
              j, i, k, k, j, altesC, A[i][k], B[k][j], altesC + A[i][k] * B[k][j]));
        }
    return C;
  }

  public static void print2DArray(double[][] A) {
    for (int i = 0; i < A.length; i++) {
      double[] reihe = A[i];
      for (int j = 0; j < reihe.length; j++) {
        System.out.print(reihe[j] + " ");
      }
      System.out.println();
    }
  }

  public static void printMagic(double[][] A, double[][] B, int m) {
    try {
      print2DArray(magicPrint(A, B, m));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    printMagic(new double[][] {
      { 1, 2, 3 },
      { 4, 5, 6 },
      { 7, 8, 9 },
    }, new double[][] {
      { 11, 12, 13 },
      { 16, 15, 14 },
      { 17, 18, 19 },
    }, -3);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/Assertion.java`



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Diese Materialsammlung unterliegt den Bestimmungen der Creative Commons Namensnennung-Nicht kommerziell-Share Alike 4.0 International-Lizenz.

Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.Der TEX-Quelltext dieses Dokuments kann unter folgender URL aufgerufen werden: `https://github.com/bschlangaul-sammlung/examens-aufgaben/blob/main/Staatsexamen/66116/2019/09/Thema-2/Teilaufgabe-1/Aufgabe-2.tex`