

Meine Aufgaben Ratings:



- Best Case: Ihr macht alle Aufgaben (, wenn die Zeit reicht)
- Falls nicht: Ich mache Ratings zur Wichtigkeit der einzelnen Aufgaben
- (Keine offizielle Empfehlung, meine subjektive Meinung auf Basis meiner eigenen Erfahrung als Student in diesem Kurs)
- ! Sagen nichts über die Schwierigkeit der Aufgaben aus !

Wichtig für die Prüfung:
(Prüfungs- oder
prüfungsähnliche Aufgaben)



Wichtig für euer Verständnis:
(Aber nicht in Prüfungsform)



Wichtig für tiefes Verständnis/
Zusatzaufgaben:
(Bspw. viele vom gleichen Typ)



Discord: timostucki



Aufgabe 1: Dreiecksmatrix

Die Klasse `Triangle` erlaubt die Darstellung von $Z \times S$ Dreiecksmatrizen (von `int` Werten). Z und S sind immer strikt grösser als 1 (d.h., > 1). Eine $Z \times S$ Dreiecksmatrix hat Z Zeilen $X_0, X_1, \dots, X_{Z-1}$, wobei Zeile X_i genau $(i * (S - 1) / (Z - 1)) + 1$ viele Elemente hat. Dieser Ausdruck wird nach den Regeln für `int` Ausdrücke in Java ausgewertet. Für eine Dreiecksmatrix D ist $D_{i,j}$ das $(j + 1)$ -te Element in der $(i + 1)$ -ten Zeile. $D_{0,0}$ ist das erste Element in der ersten Zeile [die immer genau 1 Element hat]. Abbildung 1 zeigt Beispiele von Dreiecksmatrizen. Beachten Sie, dass es möglich ist, dass zwei (aufeinanderfolgende) Zeilen die selbe Anzahl Elemente haben.

0,0	0,0	0,0
1,0 1,1	1,0 1,1	3,0 3,1 3,2 3,3
2,0 2,1 2,2 2,3	2,0 2,1 2,2	0,0
3,0 3,1 3,2 3,3 3,4	3,0 3,1 3,2 3,3	1,0
4,0 4,1 4,2 4,3 4,4 4,5 4,6		2,0 2,1

Abbildung 1: Beispiele von 5×7 , 4×4 , 2×4 und 3×2 Dreiecksmatrizen.

Aufgabe 1: Dreiecksmatrix



In der Datei `Triangle.java` finden Sie die Klasse `Triangle` mit einem Konstruktor `Triangle(int z, int s)`, der eine $z \times s$ Dreiecksmatrix erstellt. Dieser Konstruktor setzt die Werte aller Elemente auf 0. Vervollständigen Sie diese Klasse, so dass die folgenden Methoden unterstützt werden:

1. `int get(int i, int j)` gibt das Element $D_{i,j}$ zurück.
2. `void put(int i, int j, int value)` setzt das Element $D_{i,j}$ auf den Wert `value`.
3. `int[] linear()` liefert die Elemente in der kanonischen Reihenfolge (die Elemente jeder Zeile mit steigendem Index, und die Zeilen in steigender Reihenfolge).
4. `void init(int[] data)` ersetzt die Elemente von `D` durch die Werte in `data`. Sie dürfen annehmen, dass `data` genauso viele Elemente hat wie `D`. Die Methode setzt die Elemente von `D`, so dass die Folge `D.init(data); int[] y = D.linear();` in einen Array `y` resultiert für den `Arrays.equals(y, data)` den Wert `true` ergibt.
5. `void add(Triangle t)` Ein Aufruf `D.add(t)` addiert zu jedem Element $D_{i,j}$ den Wert von $t_{i,j}$, falls $t_{i,j}$ existiert. Falls $t_{i,j}$ nicht existiert, dann bleibt $D_{i,j}$ unverändert.

Tests finden Sie in der Datei `"TriangleTest.java"`. Die Datei `"TriangleGradingTest.java"` enthält die Tests, welche wir bei der Prüfung für die Korrektur verwendet haben. Wir empfehlen, diese Tests erst zu verwenden, wenn Sie denken, dass Ihre Lösung korrekt ist, damit Sie sehen können, wie Sie bei einer Prüfung abgeschnitten hätten.

Aufgabe 2: Hoare Tripel



Welche dieser Hoare Tripel sind (un)gültig? Bitte geben Sie für ungültige Tripel ein Gegenbeispiel an. Die Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1. `{ x >= 0 || y >= 0 }    z = x * y;    { z > 0 }`
2. `{ x > 10 }    z = x % 10;    { z > 0 }`
3. `{ x > 0 }    y = x * x; z = y / 2;    { z > 0 }`
4. `{ x > 0 }    y = x * x; sum = y % (x + 1); { sum > 1 }`
5. `{ b > c }`

```
if (x > b) {  
    a = x;  
} else {  
    a = b;  
}
```

```
{ a > c }
```

Aufgabe 3: Weakest Precondition



Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest precondition) an. Bitte verwenden Sie Java-Syntax. Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1.

```
P: { ?? }  
S:  if (x < 5) {  
    y = x * x;  
  } else {  
    y = x + 1;  
  }  
Q: { y >= 9 }
```

2.

```
P: { ?? }  
S:  if (x != y) {  
    x = y;  
  } else {  
    x = y + 1;  
  }  
Q: { x != y }
```

Aufgabe 4: Blackbox Testing



Im letzten Übungsblatt haben Sie Testautomatisierung mit JUnit kennengelernt. In dieser Aufgabe sollen Sie nun Tests für eine Methode schreiben, deren Implementierung Sie nicht kennen. Dadurch werden Sie weniger durch möglicherweise falsche Annahmen beeinflusst, die bei einer Implementierung getroffen wurden. Sie müssen sich also überlegen, wie sich *jede* fehlerfreie Implementierung verhalten muss. Diesen Ansatz nennt man auch **Black-Box Testing** da die Details der Implementation verdeckt sind.

In Ihrem "U06"-Projekt befindet sich eine "blackbox.jar"-Datei, welche eine kompilierte Klasse `BlackBox` enthält. Den Code dieser Klasse können Sie nicht sehen, aber sie enthält eine Methode `void rotateArray(int[] values, int steps)`, welche Sie aus einer eigenen Klasse oder einem Unit-Test aufrufen können. Diese Methode "rotiert" ein `int`-Array um eine gegebene Anzahl Schritte.

Vereinfacht macht die Methode `rotateArray()` Folgendes: Eine Rotation mit `steps=1` bedeutet, dass alle Elemente des Arrays um eine Position nach rechts verschoben werden. Das letzte Element wird dabei zum ersten. Mit `steps=2` wird alles um zwei Positionen nach rechts rotiert, usw. Eine Rotation nach links kann mit einer negativen Zahl für `steps` erreicht werden. Das folgende Beispiel ist der erste, einfache Test, den Sie in der Datei "BlackBoxTest.java" finden:

```
int[] values = new int[] { 1, 2 };
int[] expected = new int[] { 2, 1 };
BlackBox.rotateArray(values, 1);
assertArrayEquals(expected, values);
```


Aufgabe 5: Levels

- Bonus

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe “Leistungskontrolle” im www.vvz.ethz.ch). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich [die allgemeinen Regeln](#).

