252-0027-00: Einführung in die Programmierung

Übungsblatt 6

Abgabe: 7. November 2023, 23:59

Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus. Importieren Sie beide Eclipse-Projekte (das Projekt für den Bonus und das Projekt für die restlichen Aufgaben). Beachten Sie, dass Sie mehrere unabhängige Programme im bonusunabhängigen Eclipse-Projekt haben werden. Bevor Sie ein Programm starten, achten Sie deshalb darauf, dass Sie die richtige Datei im Package Explorer ausgewählt oder im Editor geöffnet haben. *Vergessen Sie nicht, Ihren Programmcode zu kommentieren!*

xkcd: Margin
Randall Munroe
(CC BY-NC 2.5)

Aufgabe 1: Dreiecksmatrix (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.

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 $Z \times S$ Dreiecksmatrize hat Z Zeilen $X_0, X_1, ..., X_{Z-1}$, 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.

In der Datei `Triangle.java` finden Sie die Klasse `Triangle` mit einem Konstruktor `Triangle(int z, int s)`, der eine `z` × `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).

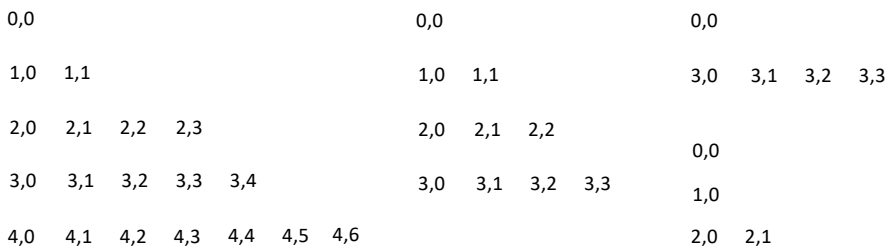


Abbildung 1: Beispiele von 5×7, 4×4, 2×4 und 3×2 Dreiecksmatrizen.

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: Datenanalyse mit Personen

In der letzten Übung haben Sie Körpergrössen analysiert. In dieser Aufgabe werten Sie einen Personen-Datensatz mit mehreren "Spalten" aus. Die Spalten enthalten Werte für Gewicht, Alter, Geschlecht, usw. Um einfach mit diesen Daten zu arbeiten, entwerfen Sie eine Klasse `Person`, welche alle Eigenschaften einer Person als Felder enthält. Weiter schreiben Sie ein Programm "PersonenAnalyse.java", welche die Personendaten aus einer Datei einliest und diese auswertet. Sie finden wie gewohnt Tests für Teile des Programms im "test"-Ordner.

Die Daten befinden sich in der Datei "body.dat.txt", welche wie folgt aufgebaut ist: Auf der ersten Zeile steht die Anzahl Datensätze in der Datei. Der Rest der Datei ist tabellarisch aufgebaut, wobei jede Zeile die Daten einer Person enthält. Die Beschreibung der Spalten und die Typen, den Sie für die Felder der Klasse `Person` verwenden sollen, sehen Sie in Tabelle 1.

a) Vervollständigen Sie die Klasse `Person` (in der Datei "Person.java"), indem Sie sie mit Feldern und Methoden ergänzen. Fügen Sie für jede Spalte im Datensatz ein Feld mit dem entsprechenden Java-Typ hinzu und implementieren Sie den (schon vorhandenen) Konstruktor so, dass er diese Felder initialisiert.

   Füllen Sie dann die Methode `beschreibung()` so aus, dass sie einen `String` zurück gibt, den die Person beschreibt. Beispielsweise könnte die Methode folgendes zurückgeben:

   ```
   Person (m, 43 Jahre, 179.7 cm, 86.4 kg)
   ```

Spalte	Beschreibung	Java-Typ
0	Schulterbreite in cm	`double`
1	Brusttiefe in cm	`double`
2	Brustbreite in cm	`double`
3	Alter in Jahren	`int`
4	Gewicht in kg	`double`
5	Grösse in cm	`double`
6	Geschlecht (1: männlich, 0: weiblich)	`boolean`

Tabelle 1: Datenbeschreibung von "body.dat.txt"

b) Als ersten Schritt der Datenanalyse sollten Sie die Daten einlesen und ein Array von `Person`-Objekten daraus erstellen. Implementieren Sie dazu `PersonenAnalyse.liesPersonen()` und rufen Sie die Methode mit dem richtigen Argument aus Ihrer `main()`-Methode auf.

c) Ihre erste Analyse soll "ungesunde" Personen anhand des Body-Mass-Index (BMI) finden. (Dass der BMI nicht unbedingt der beste Ansatz ist, den Gesundheitszustand einer Person zu ermitteln, ist ein anderes Thema.) Schreiben Sie eine Methode `bodyMassIndex()` in der Klasse `Person`, welche den BMI dieser Person nach folgender Formel berechnet:

$$\text{BMI} = \frac{\text{Gewicht}_{kg}}{\text{Grösse}^2_m}$$

Die Weltgesundheitsorganisation (WHO) definiert folgende Gewichtsklassifikation:

$$\begin{array}{rl} \text{BMI} < 18.5 & \text{untergewichtig} \\ 18.5 \leq \text{BMI} < 25.0 & \text{normalgewichtig} \\ 25.0 \leq \text{BMI} < 30.0 & \text{übergewichtig} \\ 30.0 \leq \text{BMI} & \text{fettleibig} \end{array}$$

In der Method `druckeUngesunde()` sollen Sie nun alle Personen, welche nicht normalgewichtig sind, mithilfe des gegebenen `PrintStreams` ausgeben. Geben Sie für jede solche Person zuerst ihre `beschreibung()` und dann die entsprechende Gewichtsklasse aus. Eine Zeile in der Ausgabe könnte etwa so aussehen:

```
Person (m, 34 Jahre, 167.6 cm, 75.5 kg) ist übergewichtig
```

Um die Ausgabe anzusehen, rufen Sie die Methode `druckeUngesunde()` in der `main()`-Methode mit `System.out` als Argument auf. Ausserdem können Sie die Methode mit den Tests in "PersonenAnalyseTest.java" testen. Nachdem Sie überprüft haben, dass die Ausgabe korrekt ist, ändern Sie Ihr Programm schliesslich so ab, dass die Ausgabe in eine Datei "ungesund.txt" geleitet wird.

d) Glücklicherweise sind die Daten ideal um ein weiteres wichtiges Problem zu lösen: Sie ermöglichen uns, zu jeder Person einen geeigneten Trainingspartner fürs Fitnessstudio zu ermitteln. Da die Geräte im Studio immer auf die Person eingestellt sein müssen, sollten Trainingspartner ähnliche "Dimensionen" aufweisen. Wir beschreiben die Ähnlichkeit von zwei Personen p_1

und p_2 betreffend dieses Kriteriums als *Partner-Qualität* $Q(p_1, p_2)$, die wir wie folgt definieren:

$$\begin{aligned}
\mathit{grössenDiff}(p_1, p_2) &= \mathit{grösse}(p_1) - \mathit{grösse}(p_2) \\
\mathit{brustDiff}(p_1, p_2) &= \mathit{brustTiefe}(p_1) \cdot \mathit{brustBreite}(p_1) - \mathit{brustTiefe}(p_2) \cdot \mathit{brustBreite}(p_2) \\
\mathit{schulterDiff}(p_1, p_2) &= \mathit{schulterBreite}(p_1) - \mathit{schulterBreite}(p_2) \\
Q(p_1, p_2) &= \frac{1}{1 + \mathit{grössenDiff}(p_1, p_2)^2 + \frac{\mathit{abs}(\mathit{brustDiff}(p_1, p_2))}{5} + \frac{\mathit{schulterDiff}(p_1, p_2)^2}{2}}
\end{aligned}$$

i) Implementieren Sie diese Formel in der Methode `PersonenAnalyse.partnerQualitaet()` und überprüfen Sie die Korrektheit mithilfe der Tests in "PersonenAnalyseTest.java". Folgende Methoden könnten dabei nützlich sein: `Math.pow()` und `Math.abs()`.

ii) Schreiben Sie nun eine Methode `druckeGuteTrainingsPartner()`, welche die Qualität aller möglichen Paare berechnet. Sofern die Qualität eines Paares 0.8 übersteigt, sollen die Partner-Qualität, sowie die Beschreibungen und Gewichtsklassen beider Personen ausgegeben werden. Achten Sie darauf, dass kein Paar doppelt ausgegeben wird (eine bestimmte Person darf jedoch in mehreren Paaren auftreten). Ein Beispiel eines solchen Paares ist:

```
Person (m, 21 Jahre, 177.8 cm, 79.5 kg), übergewichtig
Person (m, 19 Jahre, 177.8 cm, 76.6 kg), normalgewichtig
Qualität: 1.0
```

Kennen Sie eine bessere Formel für Partner-Qualität?

Aufgabe 3: Opaque 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);
```

Dieser Test prüft, dass das Array `{ 1, 2 }` nach einer Rotation mit `steps=1` aussieht wie `{ 2, 1 }`. Die Methode `assertArrayEquals(expected, values)` prüft, dass die beiden Arrays `expected` und `values` die selben Elemente enthalten. Wenn nicht, schlägt der Test fehl.

Die genaue Definition von `rotateArray` lautet wie folgt: Nach einem Aufruf ist das Element am Index i gleich dem Element, das zuvor am Index $(i - \texttt{steps}) \mod \texttt{values.length}$ war, für alle i zwischen 0 und $\texttt{values.length} - 1$, inklusive. "mod" steht für *modulo* und bezeichnet den Rest einer Ganzzahl-Division (siehe Wikipedia). Mit diesem Wissen sollen Sie nun weitere Tests schreiben, die möglichst gut prüfen, ob sich eine Implementierung wunschgemäss verhält. Überlegen Sie sich genau, was für die Parameter `values` und `steps` angegeben werden kann, und wie `values` nach dem Aufruf von `rotateArray()` aussieht. Der gegebene Test prüft beispielsweise nur, dass bei einem Array mit zwei Elementen nach einer Rotation um `1` die Elemente vertauscht sind. Eine Implementierung, die ein Array einfach umkehrt, würde den Test auch bestehen.

Um die Stärke Ihrer Tests zu beurteilen, werden wir verschiedene, teilweise fehlerhafte Implementierungen mithilfe Ihrer Tests prüfen. Je mehr Fehler Ihre Tests aufdecken, desto besser. Tests sollten fehlschlagen, falls die Implementierung fehlerhaft ist, und erfolgreich durchlaufen, falls keine Fehler vorhanden sind.

Aufgabe 4: Loop Invariante

Gegeben sind die Precondition und Postcondition für das folgende Programm

```
public int compute(int n) {
  // Precondition:  n >= 0
  int x;
  int res;

  x = 0;
  res = x;

  // Loop Invariante:
  while (x <= n) {
    res = res + x;
    x = x + 1;
  }
  // Postcondition:  res == ((n + 1) * n) / 2
  return res;
}
```

Schreiben Sie die Loop Invariante in die Datei "LoopInvariante.txt".

Aufgabe 5: Schweizerfahne (GUI)

Obwohl die Konsole für viele Programme als Benutzerschnittstelle ausreicht, ist sie für graphische Anwendungen nur begrenzt geeignet. Java enthält deshalb auch Klassen, mit denen man eine *GUI* (graphical user interface) erstellen kann. Da diese Klassen allerdings nicht für Anfänger geeignet sind, benutzen Sie in diesem Kurs eine einfacher zu verwendende Klasse namens `Window`. Das "U05"-Projekt ist bereits so aufgesetzt, dass Sie diese Klasse in Ihren Programmen verwenden können. Ausserdem befinden sich dort vier Beispiel-Programme, welche die `Window`-Klasse verwenden, und welche Sie als Referenz für die korrekte Verwendung der `Window` Klasse nehmen können. In dieser Übung sollen Sie ein erstes, ganz einfaches Programm mit dieser Klasse schreiben, welches die Schweizerfahne in einem Fenster anzeigt.

1. Erstellen Sie ein neues Programm "RealSwissFlag.java", inklusive `main`-Methode.

2. Erstellen Sie eine Instanz der `Window`-Klasse. Sie machen dies analog zur `Scanner`-Klasse:

   ```
   Window window = new Window(Fahne", 400, 400);
   ```

   Das erste Argument, `Fahne"`, entspricht dem Titel des Fensters und die beiden anderen Argumente, `400` und `400`, entsprechen der Fenstergrösse.

3. Sie bestimmen den Inhalt des Fensters, indem Sie Zeichnungsbefehle auf das `Window`-Objekt aufrufen. Zeichnungsbefehle bestehen meist aus zwei oder mehr Methodenaufrufen: Zuerst wird die Zeichenfarbe festgelegt und dann wird eine Form gezeichnet. Zum Beispiel:

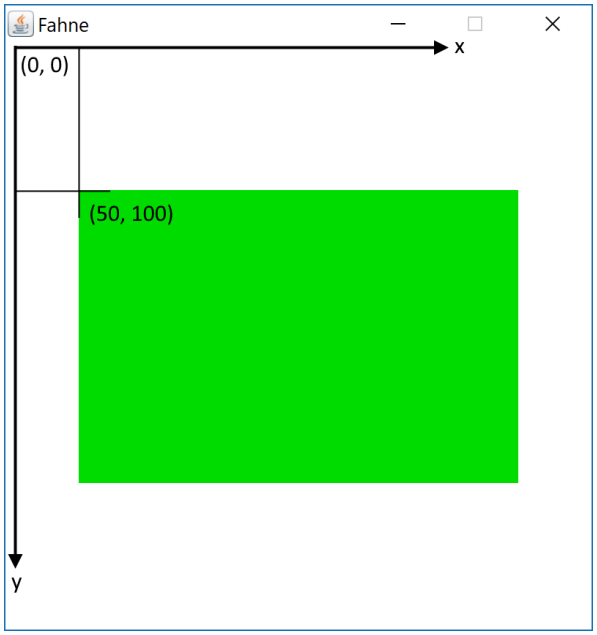
   ```
   window.setColor(0, 220, 0);
   window.fillRect(50, 100, 300, 200);
   ```

   Der erste Aufruf setzt die Zeichenfarbe auf einen mittelhellen Grünton und der zweite zeichnet ein ausgefülltes Rechteck mit der linken oberen Ecke bei der Koordinate $(50, 100)$, mit einer Breite von 300 Pixel und einer Höhe von 200 Pixel.

   Die Farbe wird als sogenannter **RGB**-Wert angegeben. Das erste Argument entspricht dem **R**ot-Kanal, das zweite dem **G**rün-Kanal und das dritte dem **B**lau-Kanal. Alle Werte sollten zwischen 0 und 255 sein. Beispiele:

`(255,  0,  0)`: rot	`(255, 255,  0)`: gelb	`(  0,  0,  0)`: schwarz
`(  0, 255,  0)`: grün	`(255,  0, 255)`: magenta	`(127, 127, 127)`: grau
`(  0,  0, 255)`: blau	`(  0, 255, 255)`: cyan	`(255, 255, 255)`: weiss

   Das Koordinatensystem von GUIs ist ein wenig anders als das, welches Sie von der Mathematik kennen. Statt von unten nach oben verläuft die y-Achse von oben nach unten und der Nullpunkt befindet sich in der linken oberen Ecke des Fensters:



Zeichnen Sie also die Schweizerfahne in das Fenster, indem Sie mehrere Rechtecke in der richtigen Farbe und den Abmessungen zeichnen. Auf Wikipedia finden Sie (teilweise widersprüchliche) Spezifikationen zu Form und Farbe.

4. Um das Fenster anzuzeigen, verwenden Sie als letztes die folgenden beiden Aufrufe:

```
window.open();
window.waitUntilClosed();
```

Der erste Aufruf öffnet das Fenster und der zweite verhindert, dass das Programm sofort wieder beendet und das Fenster dadurch geschlossen wird. Erst wenn der Benutzer das Fenster schliesst, erreicht das Programm das Ende der `main`-Methode und wird beendet.

Optional: Im Allgemeinen ist guter Code ("Good Code") so geschrieben, dass bei kleinen Änderungswünschen möglichst wenige Stellen im Code angepasst werden müssen. Ihr Programm sollte daher möglichst *parametrisiert* geschrieben sein. Ein wichtiger Parameter ist beispielsweise die Grösse der Fahne. Ändern Sie nun Ihr Programm so ab, dass es zuerst von der Konsole die gewünschte Grösse einliest und danach das Fenster und die Fahne in der gewünschten Grösse darstellt.

Aufgabe 6: Analoge Uhr

In dieser Übung geht es darum, eine "Analoge Uhr" zu programmieren, die die aktuelle Zeit inklusive Sekunden anzeigt. Dafür kommt die bereits bekannte Klasse "Window" zum Zuge. Sie befindet sich bereits in Ihrem Projektordner. Im Programm "Uhr.java" finden Sie bereits ein Template für die Aufgabe vor.

1. Zuerst sollen Sie die Zeiger Ihrer Uhr für eine gegebene Uhrzeit richtig zeichnen (z.B. für 13:37). Überlegen Sie sich, wie Sie die Start- und Endpunkte der Zeiger berechnen können. Dazu benötigen Sie für jeden Zeiger den Winkel zwischen 12Uhr und der gegebenen Uhrzeit.

Hinweis: Benutzen Sie trigonometrische Funktionen (`Math.cos(x)` und `Math.sin(x)`). Die Zeiger können Sie als Linien per `drawLine(x1,y1,x2,y2)` zeichnen und mit der Methode `setStrokeWidth(width)` die Breite der Linien verändern. Die `Window`-Klasse bietet auch noch andere Gestaltungsmöglichkeiten. Suchen Sie nach weiteren `draw...()`- und `fill...()`-Methoden mithilfe der Autovervollständigung von Eclipse.

2. Nun soll die Uhr natürlich die aktuelle Zeit anzeigen (Stunden, Minuten und Sekunden seit Mitternacht). Diese können Sie mithilfe der Methode `System.currentTimeMillis()` ermitteln. Diese Methode gibt zurück, wie viele Millisekunden seit Mitternacht, 01.01.1970 vergangen sind. Beachten Sie auch, dass die Zeit in UTC berechnet wird, weshalb Sie momentan noch eine Stunde dazu addieren müssen.