Praktikum Grundlagen der Programmierung

Prof. Dr. Harald Räcke, Prof. Dr. Felix Brandt, J. Kranz, A. Reuss WS 2017/18

Übungsblatt 5 Abgabefrist: 27.11.2017 um 05:00 Uhr

Σ Punkte: 20

Aufgabe 5.1 (P) Quiz

	Wahr	Falsch

Folgende Programmstücke sind semantisch äquivalent:
`int x = read(); int y = x/x;`
und
`int x = read(); int y = 1;`
Lösung: Die Methode `read()` könnte die Zahl 0 zurückliefern, womit im ersten Programmstück eine `ArithmeticException` (division by zero) geworfen würde, was im zweiten Fall nicht passieren kann. Somit sind die Programme nicht semantisch äquivalent.
	☐	☒

Der Ausdruck `0.3 == 0.1 + 0.1 + 0.1` evaluiert zu `true` **Lösung:** Mathematisch betrachtet sollte der Ausdruck wahr sein. Die Gleitkommazahl 0.1 kann in Java aber nicht exakt dargestellt werden. D.h. es wird nur mit einer Approximation/Annäherung der Zahl 0.1 gerechnet. Somit kann das Ergebnis aber auch nicht exakt sein. Als Lehre nehmen wir mit, dass Gleitkommazahlen im Generellen nicht auf Gleichheit überprüft werden sollten, sondern ob diese in einem *Wertebereich* liegen.	☐	☒
Mit dem 32-bit-Datentyp `float` lassen sich größere Zahlen darstellen als mit dem 64-bit-Datentyp `long` **Lösung:** Obwohl für den Datentyp `float` weniger Bits zur Verfügung stehen als in einem `long`, können damit größere Zahlen dargestellt werden. Das liegt an der Darstellung von Gleitkommazahlen. Diese werden mittels Vorzeichen, Mantisse und Exponent dargestellt. Durch die Interpretation einiger Bits als Exponenten lassen sich somit große Zahlen darstellen.	☒	☐
Jeder 32-bit-`Integer` kann in einem 32-bit-`Float` dargestellt werden. **Lösung:** Try it out with the `Integer` `16777217`	☐	☒
Der Ausdruck `(x + y) - y == x` evaluiert immer zu `true` unter der Annahme, dass `x` und `y` vom selben Zahlentyp sind.	☐	☒

Wahr Falsch

Lösung: Für Gleitkommazahlen gilt dies nicht. Wenn `y` den Wert `NaN` oder `POSITIVE_INFINITY` bzw. `NEGATIVE_INFINITY` hat, muss die Gleichheit nicht gelten. Ein weiteres Argument ist ähnlich zu der Aufgabe weiter oben. Versuchen Sie es mit den Werten `0.3` für `x` und `0.1` für `y`.

☐ ☒

Angenommen, `x` und `y` sind vom Typen `int`, dann sind folgende Ausdrücke semantisch äquivalent:
`x + 10 > y + 10`
und
`x > y`

Lösung: Angenommen, dass `x + 10` zu einem *Integer-Overflow* führt und `y + 10` nicht, dann sind die Ausdrücke nicht äquivalent. Beispiel: `x = Integer.MAX_VALUE - 1; y = 0;` **Achtung:** Treten zwei Integer-Overflows auf, dann sind die Ausdrücke wieder äquivalent!

☒ ☐

Jede Menge von Wörtern, die sich mit einer kontextfreien Grammatik beschreiben lässt, die *nicht* rekursiv ist, lässt sich auch mit einem regulären Ausdruck beschreiben und umgekehrt.

☒ ☐

Die Grammatik mit den Regeln
S ::= aAb
A ::= ab | S
ist rekursiv.

☒ ☐

Der Ausdruck `2 + 5 + ">=" + 1 + 1` evaluiert zu `"7>=11"`

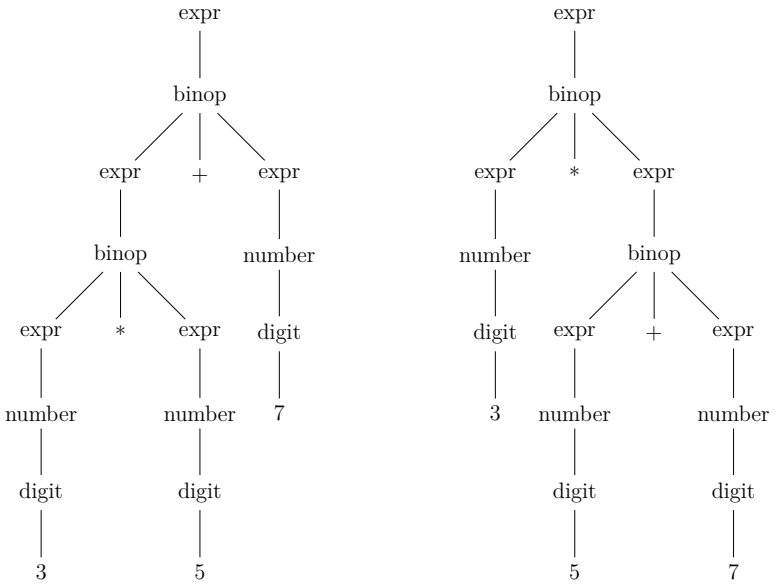
Lösung: *Operator Overloading* is evil! Machen Sie sich mit der Auswertungsreihenfolge von Ausdrücken bekannt: `((((2 + 5) + ">=") + 1) + 1)`. Vereinfacht gesagt wird bei Java wie folgt ein Additionsausdruck ausgewertet: Zahl + Zahl ergibt eine Zahl, Zahl + String oder String + Zahl ergibt einen String.

☐ ☒

Gegeben ist folgender Ausdruck `3*5+7`. Ist der Syntaxbaum/Ableitungsbaum, der mithilfe der MiniJava-Grammatik erzeugt werden kann, eindeutig?

Lösung: Kurze Antwort: Nein, es gibt zwei Ableitungsbäume:

Wahr Falsch



Lange Antwort: Für das Wort `3*5+7` existieren zwei unterschiedliche Linksableitungen. Somit können wir schließen, dass die MiniJava-Grammatik mehrdeutig ist :(

Aufgabe 5.2 (P) Auswertungssträucher

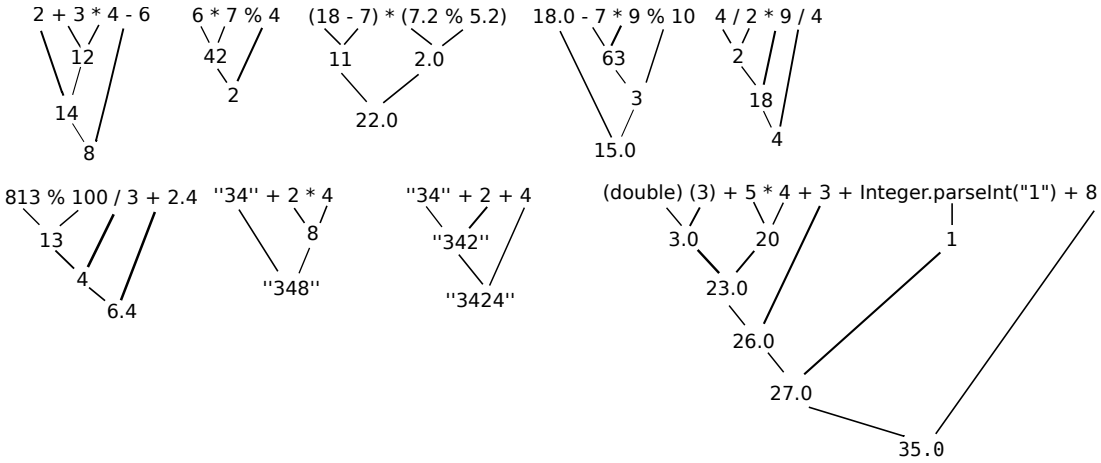
Gegeben sind folgende Ausdrücke.

1. `2 + 3 * 4 - 6`
2. `6 * 7 % 4`
3. `(18 - 7) * (7.2 % 5.2)`
4. `18.0 - 7 * 9 % 10`
5. `4 / 2 * 9 / 4`
6. `813 % 100 / 3 + 2.4`
7. `"34" + 2 * 4`
8. `"34" + 2 + 4`
9. `(double) 3 + 5 * 4 + 3 + Integer.parseInt("1") + 8`

- Geben Sie zu jedem Ausdruck den Auswertungsbaum an. Achten Sie darauf, dass anhand des Wertes auf den Typ geschlossen werden kann (String `"hallo"`, Integer `5`, Double `2.0`).
- Ändern Sie jeden Ausdruck (nur falls nötig) so ab, dass der Ausdruck den Typen
  - Integer,
  - Double,
  - String,

  hat. Verwenden Sie dabei nur die Casts `(int) (...)`, `(double) (...)` und die beiden Methoden `Integer.parseInt(String s)`, `Integer.toString(int i)`. Welche Werte haben nun die Ausdrücke? Beachten Sie, dass es verschiedene Möglichkeiten gibt, die Ausdrücke anzupassen, und sich daraus verschiedene Werte ergeben können.

Lösungsvorschlag 5.2




```
(18 - 7) * (int) (7.2 % 5.2) = 22
(int) (18.0) - 7 * 9 % 10 = 15
813 % 100 / 3 + (int) 2.4 = 6
Integer.parseInt("34") + 2 * 4 = 42
Integer.parseInt("34") + 2 + 4 = 40
(int) ((double) (3)) + 5 * 4 + 3 + Integer.parseInt("1") + 8 = 35


(double) 2 + 3 * 4 - 6 = 8.0
(double) 6 * 7 % 4 = 2.0
(double) 4 / 2 * 9 / 4 = 4.5
(double) (Integer.parseInt("34")) + 2 * 4 = 42.0
(double) (Integer.parseInt("34") + 2 + 4) = 40.0

Integer.toString(2 + 3 * 4 - 6) = "8"
Integer.toString(6 * 7 % 4) = "2"
Integer.toString((18 - 7) * (int) (7.2 % 5.2)) = "22"
Integer.toString((int) (18.0) - 7 * 9 % 10) = "15"
Integer.toString(4 / 2 * 9 / 4) = "4"
Integer.toString(813 % 100 / 3 +  (int) (2.4)) = "6"
3 + 5 * 4 + 3 + "1" + 8= "2618"
```

Aufgabe 5.3 (P) Methoden zur Feldbearbeitung

Verwenden Sie für diese Aufgabe nur die Methoden aus MiniJava. (Insbesondere reicht die Methode `void writeLineConsole()` aus, um alle Teilaufgaben zu lösen.)

Implementieren Sie in einer Klasse `Arrays` die folgenden Methoden:

- `public static void print(int[] array)` - gibt das Array `array` auf der Konsole aus. Das Array soll mit einer öffnenden geschweiften Klammer beginnen und mit einer schließenden geschweiften Klammer enden; die einzelnen Elemente des Arrays sollen durch ein Komma und ein Leerzeichen getrennt sein.

  Beispiel: `print(new int[] {1, 2, 3, 4, 5})` liefert auf der Konsole die Ausgabe `{1, 2, 3, 4, 5}`.

- `public static int[] invert(int[] array)` - gibt ein neues Array zurück, das die Elemente von `array` in umgekehrter Reihenfolge enthält.

  Beispiel: `invert(new int[] {0, 1, 2, 3})` liefert ein Array `{3, 2, 1, 0}` zurück.

- `public static int[] cut(int[] array, int length)` - gibt ein neues Array zurück, das `length` Zeichen lang ist und die Elemente von `array` in der gleichen Reihenfolge und so viele wie möglich enthält. Sollte das zurückgegeben Feld größer sein als das übergebene, sollen die zusätzlichen Positionen den Wert 0 haben.

  Beispiel: `cut(new int[] {1, 2, 3}, 2)` liefert ein Array `{1, 2}` und `cut(new int[] {1, 2, 3}, 5)` liefert ein Array `{1, 2, 3, 0, 0}`.

- `public static int[] linearize(int[][] array)` - gibt ein neues eindimensionales Array zurück, das die Werte des übergebenen zweidimensionalen Arrays `array` enthält. Die Zeilen des Arrays `array` sollen dabei nacheinander in der ihrem Zeilenindex entsprechenden Reihenfolge in dem eindimensionalem Array abgelegt werden. Beachten Sie, dass Zeilen nicht gleich lang sein müssen.

  Zum Beispiel liefert

```
1 linearize(new int[][] {{1, 3}, {25}, {7, 4, 6, 9}})
```

  ein Array `{1, 3, 25, 7, 4, 6, 9}`.

Lösungsvorschlag 5.3

Die Lösung befindet sich in der Datei `Methods.java`.

Aufgabe 5.4 (P) Verflixte Matrix

Vektoren und Matrizen lassen sich durch Arrays darstellen. Eine Matrix ist hier durch einen Vektor von Zeilen gegeben. D.h. der Eintrag `m[2][0]` steht in der dritten Zeile an erster Stelle. Vektoren werden durch eindimensionale Arrays dargestellt, d.h. es wird nicht zwischen Zeilen- und Spaltenvektoren unterschieden. Im Folgenden sollen Methoden zur Multiplikation von Matrizen und Vektoren erstellt werden. Sie können in allen Methoden davon ausgehen, dass die Dimensionen der übergebenen Arrays entsprechend der Operation zueinander passen.

1. Die Multiplikation zweier Vektoren ist folgendermaßen definiert:

$$\begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \sum_{j=1}^{n} a_j \cdot b_j$$

   Schreiben Sie eine Methode `public static int vecVecMul(int[] a, int[] b)`, welche die Multiplikation zweier Vektoren implementiert.

2. Die Multiplikation einer Matrix mit einem Vektor ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{pmatrix}$$

wobei die Einträge c_i gegeben sind durch

$$c_i = \sum_{j=1}^{n} a_{ij} \cdot b_j.$$

Schreiben Sie eine Methode `public static int[] matVecMul(int[][] a, int[] b)`, welche die Multiplikation einer Matrix und eines Vektors implementiert.

3. Die Transposition einer Matrix ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}^{\top} = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix}$$

Schreiben Sie eine Methode `public static int[][] transpose(int[][] a)`, welche die Transponierte zur Matrix a zurückgibt.

4. Die Multiplikation zweier Matrizen ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{n2} & \dots & c_{mk} \end{pmatrix}$$

wobei die Einträge c_{ij} gegeben sind durch

$$c_{ij} = \sum_{l=1}^{n} a_{il} \cdot b_{lj}.$$

Schreiben Sie eine Methode `public static int[][] matMatMult(int[][] a, int[][] b)`, welche das Produkt der Matrizen a und b zurückgibt.

5. Schreiben Sie eine Methode `public static void main(String[] args)`, um Ihre Methoden zu testen.

Lösungsvorschlag 5.4

Die Lösung befindet sich in der Datei `Matrix2D.java`.

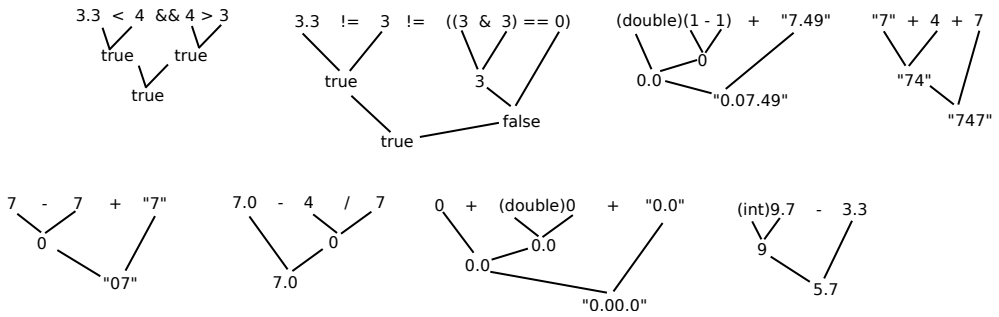
Die Hausaufgabenabgabe erfolgt über Moodle. Bitte geben Sie Ihren Code als UTF8-kodierte (ohne BOM) Textdatei(en) mit der Dateiendung `.java` ab. Geben Sie **keine** Projektdateien Ihrer Entwicklungsumgebung ab. Geben Sie **keinen** kompilierten Code ab (`.class`-Dateien). Geben Sie Ihren Code **nicht** als Archiv (z.B. als `.zip`-Datei) ab. Nutzen Sie **keine** Ordner in Moodle. Nutzen Sie **keine** Packages. Achten Sie darauf, dass Ihr Code kompiliert. Bitte vermerken Sie aus Datenschützgründen nicht Ihren Namen oder Ihre Matrikelnummer im Code. Auf diesem Blatt gibt es auch Aufgaben, zu denen die Lösung kein Programm ist. Geben Sie diese Aufgaben in einer einzelnen, korrekt orientierten und gut lesbaren DIN-A4-PDF-Datei ab. Hausaufgaben, die sich nicht im vorgegebenen Format befinden, werden nur mit Punktabzug oder gar nicht bewertet.

Aufgabe 5.5 (H) Neues aus dem Gewächshaus [4 Punkte]

Geben Sie zu jedem der folgenden Ausdrücke den Auswertungsbaum an. Achten Sie darauf, dass anhand des Wertes auf den Typ geschlossen werden kann (String `"hallo"`, Integer `5`, Double `2.0`).

1. `3.3 < 4 && 4 > 3`
2. `3.3!=3!=((3&3)==0)`
3. `(double)(1-1)+"7.49"`
4. `"7"+4+7`
5. `7-7+"7"`
6. `7.0-4/7`
7. `0+(double)0+"0.0"`
8. `(int)9.7-3.3`

Lösungsvorschlag 5.5



Aufgabe 5.6 (H) Student, ärgere dich! [8 Punkte]

In dieser Aufgabe kann die Methode `int dice()` von MiniJava hilfreich sein. Die Methode liefert eine Zufallszahl zwischen 1 und 6 (jeweils inklusive).

In dieser Aufgabe sollen Sie eine vereinfachte Variante des Spiels "~~Mensch~~ Studentin, ärgere dich ~~nicht~~!" für *zwei* Spieler programmieren. Das Ziel jedes Spielers ist es, alle seine Figuren vom Haus in den Garten zu bewegen. Das Spielfeld besteht aus einem Ring mit den Feldern $0 - 39$. Alle 10 Felder verzweigt sich der Weg entweder in den Garten eines Spielers oder er geht weiter entlang des Rings. Die Grundregeln des Spiels sind:

- Jeder Spieler besitzt vier Spielsteine, die sich zu Spielbeginn im jeweiligen Haus befinden.

- Die Spieler starten jeweils um 10 Felder versetzt (also an den Positionen 0 bzw. 10).
- Nach dem 39. Feld geht es mit dem 0. Feld weiter.
- Im Wechsel wird gewürfelt. Ein eigener Spielstein wird um die entsprechende Augenzahl vorgerückt. Der Spieler entscheidet, welcher Spielstein.
- Wer zuerst mit all seinen Spielsteinen in seinem Garten ankommt, gewinnt.
- Zwei Steine dürfen nicht gleichzeitig auf demselben Feld stehen.
- Ein fremder Spielstein kann geschlagen werden. Landet ein Spieler mit seinem Stein auf einem Feld, auf dem ein fremder Spielstein steht, muss dieser zurück nach Hause.
- Sollte das Zielfeld eines Spielsteins s von einem eigenen Spielstein besetzt sein, darf s nicht gezogen werden.

Obige Grundregeln entsprechen womöglich nicht den offiziellen Regeln oder den Ihnen bekannten Hausregeln, bitte implementieren Sie die Regeln trotzdem so, wie sie beschrieben sind!

Hinweis: Im Gegensatz zum Originalspiel darf das Haus mit jeder Zahl verlassen werden und der Garten muss nicht exakt getroffen werden.

Erstellen Sie ein Programm namens `Mensch.java` in folgenden Schritten:

1. Ermöglichen Sie, dass *ein einzelner* Spieler einen Stein vom Haus in den Garten bringen kann.
2. Erweitern Sie das Programm so, dass ein Spieler alle vier Steine steuern kann. Arrays eigenen sich dazu, um mehrere Steine zu verwalten.
3. Achten Sie darauf, dass der Zug eines Spielsteins durch die Position der anderen Spielsteine blockiert werden kann. Fordern Sie in diesem Fall die Wahl eines anderen Steins.
4. Erweitern Sie ihr Programm so, dass auch ein zweiter Spieler am Spiel teilnehmen kann. Achten Sie dabei auf die Überschreitung des 39. Feldes.
5. Achten Sie darauf, dass sich Steine zweier verschiedener Spieler nicht blockieren, sondern evtl. geschlagen werden können.

Das Spiel soll über eine grafische Benutzeroberfläche verfügen, die von uns bereits vorgegeben ist. Fügen Sie die Datei `Aerger.java` zu Ihrem Projekt hinzu und ersetzen Sie `extends MiniJava` durch `extends Aerger`.

Nun steht Ihnen der Aufruf `paintField(a, b, c, d);` zur Verfügung, mit dem Sie ein Spielfeld zeichnen können. Die Spielsteine stehen dabei auf den Feldern, die durch die `int`-Arrays `a` bis `d` beschrieben werden. Der Wert -1 für eine Spielsteinposition steht dabei für das Haus, ein Wert > 39 für den Garten.

```
1  public static void main(String[] args) {
2    int[] arrYellow = {40, 15, 2, -1};
3    int[] arrBlue =  {0, -1, 42, 30};
4    int[] arrRed = {5, 10, -1, 42};
5    int[] arrGreen = {42, 25, 35, -4};
6    Aerger.paintField(arrYellow, arrBlue, arrRed, arrGreen);
7  }
```

Spieler, die am Spiel nicht teilnehmen (laut Aufgabenstellung müssen Sie hier nur zwei Spieler implementieren) sollen alle Steine im Haus stehen haben.

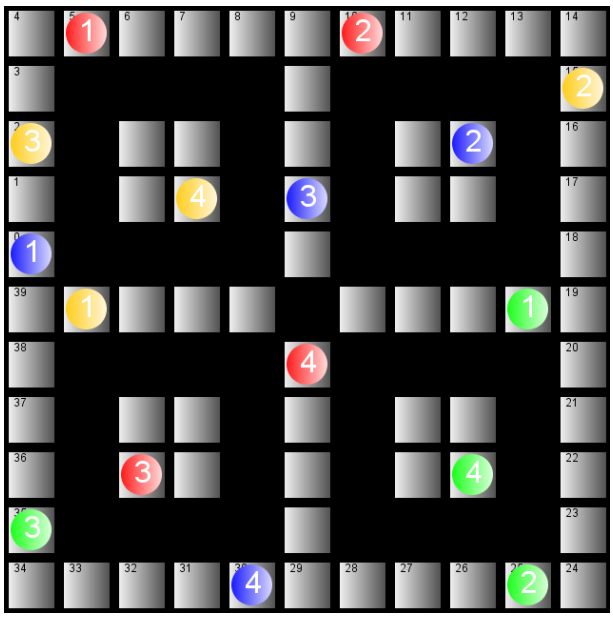


Abbildung 1: Spielbrett-GUI nach dem Beispielcode

Lösungsvorschlag 5.6

Die Lösung befindet sich in `Mensch.java`.

Aufgabe 5.7 (H) Der Gauß ist dem Student ein Graus [8 Punkte]

In dieser Aufgabe soll eine vereinfachte Version des Gaußschen Eliminationsverfahrens[1] implementiert werden. Das Verfahren kann dazu verwendet werden, lineare Gleichungssysteme zu lösen. In dieser Aufgabe gehen wir davon aus, dass das gegebene Gleichungssystem genau eine eindeutige Lösung hat. Das Verhalten Ihrer Lösung ist für alle anderen Fälle nicht weiter definiert, diese können also ignoriert werden.

Das Gleichungssystem wird als sog. *erweiterte Koeffizientenmatrix* eingelesen und vom Programm verwaltet. Es sei z.B. folgendes Gleichungssystem $A * x = b$ gegeben:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 2 \\ x_1 + x_2 + x_3 &= 2 \\ 3x_1 + 3x_2 + x_3 &= 0 \end{aligned}$$

Die erweiterte Koeffizientenmatrix enthält die Koeffizienten A und den b-Vektor:

$$\begin{pmatrix} 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 2 \\ 3 & 3 & 1 & 0 \end{pmatrix}$$

[1] `https://de.wikipedia.org/wiki/Gau%C3%9Fsches_Eliminationsverfahren`

Die Matrix hat in dieser Aufgabe immer genau eine Spalte mehr als Zeilen. Im Rahmen des Gaußschen Verfahrens muss die Matrix nun in die sog. *Zeilenstufenform* überführt werden. In Zeilenstufenform befinden sich unterhalb der Diagonalen des vorderen Teils der Matrix nur 0er; für den Spezialfall von drei Gleichungen hat die Matrix in Zeilenstufenform also die folgende Form:

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots \end{pmatrix}$$

Die Zeilenstufenform wird durch eine Reihe von Umformungen erreicht, die weiter unten genauer beschrieben sind. Hier sei zunächst eine gültige Matrix in Zeilenstufenform für obiges Beispiel gegeben:

$$\begin{pmatrix} 1 & 2 & 3 & 2 \\ 0 & -1 & -2 & 0 \\ 0 & 0 & -2 & -6 \end{pmatrix}$$

Die Matrix lässt sich zum besseren Verständnis wieder zurück in die ursprüngliche Schreibweise eines Gleichungssystems überführen:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= \quad 2 \\ 0 - \quad x_2 - 2x_3 &= \quad 0 \\ 0 + \quad 0 - 2x_3 &= -6 \end{aligned}$$

Die Lösung ($x_1 = 5$, $x_2 = -6$ und $x_3 = 3$) lässt sich nun einfach von unten nach oben ablesen. Wir überführen eine Matrix in Zeilenstufenform, indem wir folgende Operationen auf der Matrix ausführen:

- **Multiplikation einer Zeile mit einem Faktor**
  Im folgenden Beispiel wird die zweite Zeile der Matrix mit dem Faktor 2 multipliziert:

$$\begin{pmatrix} 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 2 \\ 3 & 3 & 1 & 0 \end{pmatrix} \mid \times 2 \; = \begin{pmatrix} 1 & 2 & 3 & 2 \\ 2 & 2 & 2 & 4 \\ 3 & 3 & 1 & 0 \end{pmatrix}$$

- **Addition einer Zeile, die vorher mit einem Faktor multipliziert wird**
  Im folgenden Beispiel wird die zweite Zeile der Matrix mit dem Faktor -3 multipliziert und auf die dritte Zeile addiert:

$$\begin{pmatrix} 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 2 \\ 3 & 3 & 1 & 0 \end{pmatrix} \begin{matrix} \\ \\ \longrightarrow \\ \hookleftarrow \end{matrix} \begin{matrix} \\ \\ \times(-3) \\ + \end{matrix} = \begin{pmatrix} 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 2 \\ 0 & 0 & -2 & -6 \end{pmatrix}$$

- **Vertauschen zweier Zeilen**
  Im folgenden Beispiel wird die zweite mit der dritten Zeile vertauscht:

$$\begin{pmatrix}1 & 2 & 3 & 2\\1 & 1 & 1 & 2\\3 & 3 & 1 & 0\end{pmatrix} \begin{matrix}\leftarrow\!\urcorner\\ \leftarrow\!\lrcorner\end{matrix} = \begin{pmatrix}1 & 2 & 3 & 2\\3 & 3 & 1 & 0\\1 & 1 & 1 & 2\end{pmatrix}$$

Alle gezeigten Operationen können beliebig angewendet werden, da sie die Lösung des beschriebenen Gleichungssystems nicht ändern. Unser Algorithmus folgt schrittweise der Diagonalen der Koeffizientenmatrix von links oben nach rechts unten und führt für jeden Eintrag die folgenden Operationen aus.

1. Ist der Eintrag auf der Diagonalen gleich 0, so wird die Zeile mit einer darunterliegenden Zeile getauscht, sodass der Eintrag in der Diagonalen nicht mehr gleich 0 ist. Dies ist wegen der Voraussetzung einer eindeutigen Lösung immer möglich.

2. Es müssen nun alle Einträge e_i unterhalb des Diagonaleneintrags d in der gleichen Spalte zu 0 verändert werden. Um dies zu erreichen, berechnen wir für jeden Eintrag e_i das KGV mit d; es sei $kgv_i = kgv(e_i, d)$. Wir multiplizieren nun die Zeile von e_i mit $\frac{kgv_i}{e_i}$ und addieren die Zeile von d multipliziert mit $\frac{-kgv_i}{d}$ auf das Ergebnis. Der entsprechende Eintrag e_i hat nun den Wert 0.

   Es sei z.B. folgende Matrix gegeben:

   $$\begin{pmatrix}6 & 2 & 3 & 2\\4 & 2 & 2 & 4\\3 & 3 & 1 & 0\end{pmatrix}$$

   Der Algorithmus befindet sich in der ersten Zeile, es ist also $d = 6$. Um direkt darunter in der zweiten Zeile eine 0 zu erzeugen, bilden wir zunächst das KGV mit 4, also 12. Die Multiplikation der zweiten Zeile mit $\frac{12}{4} = 3$ resultiert in folgender Matrix:

   $$\begin{pmatrix}6 & 2 & 3 & 2\\4 & 2 & 2 & 4\\3 & 3 & 1 & 0\end{pmatrix} \begin{matrix}\\ \mid \times 3\\ \\\end{matrix} = \begin{pmatrix}6 & 2 & 3 & 2\\12 & 6 & 6 & 12\\3 & 3 & 1 & 0\end{pmatrix}$$

   Die Addition der ersten Zeile multipliziert mit $\frac{-12}{6} = -2$ ergibt eine Matrix, die unsere gewünschte Bedingung erfüllt:

   $$\begin{pmatrix}6 & 2 & 3 & 2\\12 & 6 & 6 & 12\\3 & 3 & 1 & 0\end{pmatrix} \begin{matrix}\urcorner^{\times(-2)}\\ \leftarrow\!\lrcorner_{+}\\ \\\end{matrix} = \begin{pmatrix}6 & 2 & 3 & 2\\0 & 2 & 0 & 8\\3 & 3 & 1 & 0\end{pmatrix}$$

Es sei folgendes Gerüst für die Aufgabe gegeben:

```java
public class GrausGauß extends MiniJava {
  private static int lines;

  public static void main(String[] args) {
    lines = read("Geben Sie die Anzahl der Gleichungen ein.");
  }
}
```

Die Variable `lines` ist hier eine *globale* Variable, die in allen Methoden verwendet werden kann. Gehen Sie bei Ihrer Implementierung wie folgt vor:

1. Schreiben Sie eine Methode `public static int[] readMatrix()`, die die erweiterte Koeffizientenmatrix eines Gleichungssystems vom Benutzer als eindimensionales Feld einliest.
2. Implementieren Sie die Java-Methode `public static void printMatrix(int[] matrix)`, die eine Matrix ausgibt.
3. Implementieren Sie die Methode `public static int get(int[] matrix, int line, int column)`, die ein Element der Matrix in einer bestimmten Zeile und einer bestimmten Spalte zurückliefert sowie die Methode `public static void set(int[] matrix, int line, int column, int value)`, die es erlaubt, ein Element der Matrix zu überschreiben.
4. Implementieren Sie die Methode `public static void multLine(int[] matrix, int line, int factor)`, die eine Zeile der Matrix mit einem Faktor multipliziert.
5. Implementieren Sie die Methode `public static void multAddLine(int[] matrix, int line, int otherLine, int factor)`, die zu einer Zeile `line` einer Matrix eine andere Zeile `otherLine` multipliziert mit einem Faktor `factor` addiert.
6. Implementieren Sie die Methode `public static void swap(int[] matrix, int lineA, int lineB)`, die zwei Zeilen einer Matrix tauscht.
7. Implementieren Sie die Methode `public static void searchSwap(int[] matrix, int fromLine)`, die sicherstellt, dass ein bestimmter Eintrag der Diagonalen, nämlich in Zeile `fromLine`, nicht den Wert 0 enthält. Sie soll dazu, wenn nötig, darunterliegende Zeilen nach oben tauschen. **Hinweis:** Es ist aufgrund der oben genannten Voraussetzungen nicht möglich, dass alle betrachteten Einträge den Wert 0 enthalten und ein Tauschen unmöglich ist.
8. Implementieren Sie die Methode `public static int kgv(int a, int b)`, die das KGV zweier Zahlen berechnet. **Hinweis**: Sollten Sie die Methode nicht implementieren können, nutzen Sie statt des KGVs von a und b in den folgenden Teilaufgaben den Wert $a * b$.
9. Implementieren Sie eine Methode `public static int[] rowEchelonToResult(int[] matrix)`, die aus einer Matrix in Zeilenstufenform das Ergebnis des Gleichungssystems als Vektor von Zahlen berechnet. Gehen Sie dabei davon aus, dass die x_i der Lösung ganzzahlig sind.
10. Implementieren Sie die Methode `public static int[] solveSystem(int[] matrix)`, die ein gegebenes Gleichungssystem mithilfe der bisherigen Methoden löst und das Ergebnis als Vektor von Zahlen zurückliefert.
11. Implementieren Sie ein Hauptprogramm, welches ein Gleichungssystem vom Benutzer als Matrix einliest und die Lösung des Systems ausgibt.

Hinweis: Sie dürfen in dieser Aufgabe nur eindimensionale Arrays verwenden.

Lösungsvorschlag 5.7
Die Lösung befindet sich in `GrausGauß.java`.

Bewertungsschema:

1. 0.25 Punkte
2. 0.25 Punkte
3. 0.25 Punkte
4. 0.75 Punkt
5. 1 Punkte
6. 0.5 Punkte
7. 1.5 Punkte
8. 0.25 Punkte
9. 1.5 Punkte
10. 1.5 Punkte
11. 0.25 Punkte

Korrekturhinweise:

- Studenten, die die erste Zeile freigelassen haben, weil sie nicht ab 0 zählen möchten, erhalten nur einen deutlichen Hinweis, keinen Abzug.