

# Projekt LinEqSysCalculator

## 1 Inhaltsverzeichnis

### Inhalt

1	Inhaltsverzeichnis .....	0
2	Vorwort .....	1
2.1	Ziel .....	1
2.2	Erforderliche class-Dateien .....	1
2.3	Datenmodell .....	1
3	Start .....	2
4	Einfacher Kommando-Interpreter .....	11
4.1	Ziel .....	11
4.2	Vorgehen .....	11
4.3	Gestaltung .....	12
5	Lösen von LGS mit LinEqSysCalculator .....	21
5.1	Vorbereitung .....	21
5.2	Exkurs: Rekursion und Iteration .....	24
5.3	Determinanten mit Dimension grösser als 3 .....	24
5.4	Determinantenmethode .....	28
6	Ende des Projekts .....	35
7	Lösungen .....	36
7.1	Zu Kap. 3: Start .....	36
7.2	Zu Kap. 4: Einfacher Kommando-Interpreter .....	38
7.3	Zu Kap. 5: Lösen von LGS mit LinEqSysCalculator .....	38

## 2 Vorwort

### 2.1 Ziel

Das grundlegende Ziel des Projektes «LinEqSysCalculator» ist die Erstellung eines Java-Programms zur Lösung von «beliebig» grossen linearen Gleichungssystemen (**Abkürzung: LGS**). Als Lösungsmethode wird die **Determinantenmethode** verwendet.

Alle dazu erforderlichen Eingaben und Ausgaben sollen in der Standard-Konsole, also textuell, durchgeführt werden.

### 2.2 Erforderliche class-Dateien

Da alle Ein- und Ausgaben in der Standard-Konsole geschehen sollen, werden wir dazu die statischen Methoden der Klasse `Kon` verwenden, die vor allem das Programmieren von Benutzer-Eingaben in der Standard-Konsole erleichtern aber auch für Text-Ausgaben verwendet werden können.

Im zweiten Teil des Projektes werden wir auch die statischen Methoden der Klasse `MatrixUtilt` verwenden, auf die wir später näher eingehen werden. Beide Klassendateien `Kon.class` und `MatrixUtilt.class` befinden sich schon im Unterordner «src» Ihres Projektordners «LinEqSysCalculator».

### 2.3 Datenmodell

Wir beschäftigen uns zuerst mit dem Datenmodell, das für die Speicherung und nachfolgende Bearbeitung eines LGS für unsere Zwecke gewählt werden soll.

Ein LGS in  $n$  Variablen und  $n$  Gleichungen ( $n \in \mathbb{N}$ ) kann folgendermassen dargestellt werden, wobei die Indices von 0 an beginnen und die Konstantenkolonne auch indiziert wird:

$$\begin{array}{cccccccl}
 a_{0,0} \cdot x_0 + & a_{0,1} \cdot x_1 + \cdots + & a_{0,n-1} \cdot x_{n-1} & = & a_{0,n} & & 0\text{-te Gleichung} \\
 a_{1,0} \cdot x_0 + & a_{1,1} \cdot x_1 + \cdots + & a_{1,n-1} \cdot x_{n-1} & = & a_{1,n} & & 1\text{-te Gleichung} \\
 \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
 a_{n-1,0} \cdot x_0 + & a_{n-1,1} \cdot x_1 + \cdots + & a_{n-1,n-1} \cdot x_{n-1} & = & a_{n-1,n} & & (n-1)\text{-te Gleichung}
 \end{array}$$

Das System ist also durch seine Koeffizienten  $a_{i,j}$ ,  $i, j = 0, \dots, n-1$ , und Konstanten  $a_{i,n}$ ,  $i = 0, \dots, n-1$ , vollständig gegeben. Diese können bequem in der Matrix

$$\begin{pmatrix}
 a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} & a_{0,n} \\
 a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} & a_{1,n} \\
 \vdots & \vdots & \ddots & \vdots & \vdots \\
 a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} & a_{n-1,n}
 \end{pmatrix}$$

zusammengefasst werden, die aus  $n$  Zeilen für die Gleichungen und  $n+1$  Spalten für die jeweiligen Koeffizienten und die Konstante besteht.

Als Datenmodell zur Speicherung dieser Matrix bietet sich sofort ein 2-dimensionales Zahlenfeld, also z.B. ein 2-dimensionales `double`-Array, an. Der **erste Index** des Arrays soll die **Zeile**, d.h. die Nummer der Gleichung, angeben und der **zweite Index** die **Spalte**, d.h. der Index der Variable, mit der der Koeffizient multipliziert wird, bzw.  $n$  bei den Konstanten.

Wir werden also Matrizen modellieren und uns insbesondere für den Fall von Matrizen interessieren, deren Anzahl Spalten um 1 grösser als die Anzahl Zeilen ist.

Dazu werden wir eine Java-Klasse `Matrix` erstellen, die wir im Laufe des Projekts immer weiter ausstatten werden.

Diese soll ein einziges Attribut `public double[][] elmt=new double[0][0]` enthalten, das zur Speicherung der Matrixelemente dient.

### 3 Start

Da das Programm in mehreren Schritten aufgebaut wird und die verschiedenen Stadien als eigene Versionen gespeichert werden sollen, ist es sinnvoll, zunächst eine Klasse `LinEqSysCalculator_0` zu erstellen, die vorerst nur die `main`-Methode enthält. Danach werden wir in der gleichen Datei die Klasse `Matrix` hinzufügen. So können wir die verschiedenen Versionen unter z.B. `LinEqSysCalculator_1_2` o. ä. speichern, ohne dass eine Umbenennung der Klasse `Matrix` nötig wäre.

Der *JavaEditor* ändert zudem nach dem Speichern unter einem neuen Namen den Namen der Klasse `LinEqSysCalculator...` im Java-Code automatisch, was sehr bequem ist.

#### 3.1.1 Aufgabe

Erstellen Sie mit *JavaEditor* (oder einem anderen Java-Editor) im Unterordner «src» Ihres Projekt-Ordners «LinEqSysCalculator» ein neues Konsole-Programm namens `LinEqSysCalculator_0`. Der Code sollte etwa so aussehen:

```
8
9 public class LinEqSysCalculator_0 {
10
11     public static void main(String[] args) {
12
13     } // end of main
14
15 } // end of class LinEqSysCalculator_0
16
17
```

Nun fügen Sie nach dem Code der Klasse `LinEqSysCalculator_0` den Code für die Klasse `Matrix` hinzu. Das Ergebnis sollte etwa so aussehen.

```
9
10 public class LinEqSysCalculator_0 {
11
12     public static void main(String[] args) {
13
14     } // end of main
15
16 } // end of class LinEqSysCalculator_0
17
18 class Matrix {
19     public double[][] elmt=new double[0][0];
20
21     public Matrix() {
22
23     }
24
25 } // end of class Matrix
26
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_0` ab.

`LinEqSysCalculator_0` ist das (noch sehr leere) Gerüst unseres Programms. Zu Beginn sind beide Dimensionen des 2-dimensionalen `double`-Array-Attribut `elmt` auf 0 initialisiert, `elmt` ist also noch leer. Wir werden sofort eine erste kleine Erweiterung vornehmen.

### 3.1.2 Aufgabe

Speichern Sie `LinEqSysCalculator_0` unter `LinEqSysCalculator_1_0` neu ab.

Ergänzen Sie dann die Klasse `Matrix` um die unten gezeigte Methode `public void init(int amtRows, int amtClmns)`, die die Dimensionen des Attributs `elmt` auf `amtRows` und `amtClmns` initialisiert und alle `double`-Array-Elemente auf den Wert `0` setzt.

Fügen Sie dann, wie unten auch gezeigt, einen zusätzlichen Konstruktor `public Matrix(int rowDim, int clmnDim)` hinzu, der die `init`-Methode geeignet aufruft.

Das Ergebnis sollte etwa so aussehen.

```

10 public class LinEqSysCalculator_1_0 {
11
12     public static void main(String[] args) {
13
14     } // end of main
15
16 } // end of class LinEqSysCalculator_1_0
17
18 class Matrix {
19     public double[][] elmt=new double[0][0];
20
21     public Matrix() {
22
23     }
24
25     public Matrix(int rowDim, int clmnDim) {
26         init(rowDim,clmnDim);
27     }
28
29     // - Initialisiert das 2-dim. Array elmt auf die Dimension amtRows x amtClmns
30     // - und setzt alle Array-Elemente auf den Wert 0.
31     // - Falls amtRows oder amtClmns kleiner oder gleich 0 sind, so wird nichts
32     // - gemacht, d.h. elmt bleibt unverändert.
33     public void init(int amtRows, int amtClmns) {
34         if (amtRows>0 && amtClmns>0) {
35             elmt=new double[amtRows][amtClmns];
36             for (int i=0; i<amtRows; ++i) {
37                 for (int j=0; j<amtClmns; ++j) {
38                     elmt[i][j]=0;
39                 } // end of for
40             } // end of for
41         }
42     }
43
44 } // end of class Matrix

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_0` ab.

Nun kümmern wir uns um eine mögliche Darstellung eines `Matrix`-Objekts in Textform. Es ist üblich in Java, eine Methode `public String toString()` zu schreiben, die eine einzeilige `String`-Repräsentation des Objektes zurückgibt.

Wir wählen dazu vorerst lediglich die Auflistung der Elemente des `Matrix`-Objektes und wollen sie mit eckigen Klammern '[' , ']' umschliessen, wobei wir darin die Zeilen mit vertikalen Strichen '|' trennen und die Elemente in den Zeilen mit Strichpunkten ';'.

### 3.1.3 Aufgabe

Speichern Sie `LinEqSysCalculator_1_0` unter `LinEqSysCalculator_1_1` neu ab.

Fügen Sie dann der Klasse `Matrix` die Methode `public String toString()`, die hier folgt, hinzu. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```
// -- Darstellungsmethoden -- //
```

```
// - Gibt eine einzeilige Darstellung der Matrix zurück.
```

```
public String toString() {  
    String ret="[";  
    for (int i=0; i<[ ]; ++i) {  
        for (int j=0; j<elmt[0].length; ++j) {  
            ret=ret+[ ];  
            if (j<elmt[0].length-1) {  
                ret=ret+";";  
            } // end of if  
        } // end of for  
        if (i<elmt.length-1) {  
            ret=ret+"|";  
        }  
    } // end of for  
    return ret+"]";  
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_1` ab.

In diesem ersten Teil des Projekts werden vor allem die Methoden der Klasse `Matrix`, die jeweils hinzugefügt werden, getestet.

Wir werden die Tests in der jeweiligen Klasse `LinEqSysCalculator_...` durchführen und wollen nun einige Vorbereitungen dafür treffen.

Wir wollen auch die `toString()`-Methode eines `Matrix`-Objekts mit den Informationen zu seinen Dimensionen ergänzen.

### 3.1.4 Aufgabe

Speichern Sie `LinEqSysCalculator_1_1` unter `LinEqSysCalculator_1_2` neu ab.

Ergänzen Sie dann zuerst die Klasse `Matrix` um die zwei getter-Methode `public int getAmtRows()` und `public int getAmtClmns()`, die die Anzahl Zeilen bzw. Spalten der Matrix zurückgeben, und ergänzen bzw. verändern Sie dann die `toString()`-Methode wie hier abgebildet.

```

60 | // - Gibt die Anzahl Zeilen zurück.
61 | public int getAmtRows() {
62 |     return elmt.length;
63 | }
64 |
65 | // - Gibt die Anzahl Spalten zurück. Falls die Anzahl Zeilen Null ist,
66 | // - so wird auch Null zurückgegeben.
67 | public int getAmtClmns() {
68 |     if (getAmtRows() > 0) {
69 |         return elmt[0].length;
70 |     } else {
71 |         return getAmtRows();
72 |     } // end of if-else
73 | }
74 |
75 |
76 | // -- Darstellungsmethoden -- //
77 |
78 | // - Gibt eine einzeilige Darstellung der Matrix zurück.
79 | public String toString() {
80 |     String ret = "" + getAmtRows() + "x" + getAmtClmns() + "-Matrix[";
81 |     for (int i = 0; i < getAmtRows(); ++i) {
82 |         for (int j = 0; j < getAmtClmns(); ++j) {
83 |             ret = ret + elmt[i][j];
84 |             if (j < getAmtClmns() - 1) {
85 |                 ret = ret + ",";
86 |             } // end of if
87 |         } // end of for
88 |         if (i < getAmtRows() - 1) {
89 |             ret = ret + "\n";
90 |         }
91 |     } // end of for
92 |     return ret + "]";
93 | }
94 |

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_2` ab.

### 3.1.5 Aufgabe

Fügen Sie nun der Klasse `LinEqSysCalculator_1_2`, wie unten gezeigt, die Methoden `public static Matrix getExample1()`, die eine Beispielsmatrix zurückgibt, und `public static void test()`, die die Ausgabe der Matrix-Methode `toString()` mit dieser Beispielsmatrix testet, hinzu und rufen Sie die Methode `test()` in der `main`-Methode auf.

```

public static void main(String[] args) {
    test();
} // end of main

public static Matrix getExample1() {
    Matrix ret = new Matrix(4, 5);
    ret.elmt[0][0] = 2; ; ret.elmt[0][1] = 2; ; ret.elmt[0][2] = -1; ; ret.elmt[0][3] = 1; ; ret.elmt[0][4] = 7; ;
    ret.elmt[1][0] = -3; ; ret.elmt[1][1] = 5; ; ret.elmt[1][2] = 4; ; ret.elmt[1][3] = -5; ; ret.elmt[1][4] = -1; ;
    ret.elmt[2][0] = 7; ; ret.elmt[2][1] = -6; ; ret.elmt[2][2] = 3; ; ret.elmt[2][3] = -2; ; ret.elmt[2][4] = -4; ;
    ret.elmt[3][0] = -10; ; ret.elmt[3][1] = 3; ; ret.elmt[3][2] = 5; ; ret.elmt[3][3] = 1; ; ret.elmt[3][4] = 15; ;
    return ret;
}

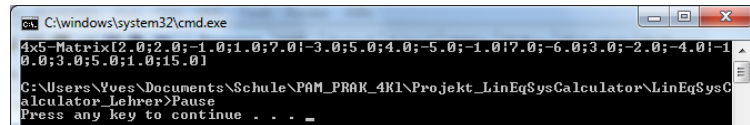
public static void test() {
    Matrix matr1 = getExample1();
    Kon.writeln(matr1.toString());
}

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_2` ab.

Kompilieren Sie nun `LinEqSysCalculator_1_2` und lassen Sie das Programm laufen.

Sie sollten in etwa die folgende Ausgabe erhalten:



```
cmd C:\windows\system32\cmd.exe
4x5-Matrix[2.0;2.0;-1.0;1.0;7.0;-3.0;5.0;4.0;-5.0;-1.0;7.0;-6.0;3.0;-2.0;-4.0;-1.0;0.0;3.0;5.0;1.0;15.0]
C:\Users\Yves\Documents\Schule\PAM_PRAK_4KI\Projekt_LinEqSysCalculator>Lehren>Pause
Press any key to continue . . . _
```

Die einzeilige `Matrix`-Repräsentation funktioniert also. Da sie aber schwer zu lesen ist, möchten wir eine mehrzeilige Ausgabemöglichkeit haben, mit der die Matrixelemente in gewohnter Weise in Zeilen und Spalten dargestellt werden können.

Wir werden dazu der Klasse `Matrix` eine Methode `public String[] toStrings()` hinzufügen, die ein `Matrix`-Objekt als `String`-Array repräsentiert. Die einzelnen `Strings` des Arrays sollen die Zeilen der Matrix darstellen, d.h. die Zeilenelemente hintereinander durch ein Leerzeichen getrennt.

Falls nun die Zeilen, d.h. die `Strings` des Arrays, unter einander ausgegeben werden, sollen Elemente der gleichen Spalten auch sicher unter einander stehen. Da die Matrixelemente verschieden lang sein können, muss dazu vorher die nötige minimale Spaltenbreite in Zeichen in jeder Spalte ermittelt werden. Elemente einer Spalte müssen dann eventuell durch Hinzufügen von Leerzeichen auf die nötige Breite gebracht werden, bevor sie im jeweiligen Zeilen-`String` aufgenommen werden.

Wir werden nun also Methoden ergänzen, die die gestellten Aufgaben lösen, und damit die `toStrings()`-Methode erstellen.

Für eventuelle spätere Zwecke und auch zum besseren Verständnis der Methode für die Ermittlung der nötigen Spaltenbreiten pro Spalte werden wir auch eine Methode erstellen, die die maximale `String`-Breite aller Matrixelemente zurückgibt.

### 3.1.6 Aufgabe

Speichern Sie `LinEqSysCalculator_1_2` unter `LinEqSysCalculator_1_3` neu ab.

Fügen Sie nun der Klasse `Matrix`, wie unten gezeigt, die Methoden `public int getMaxElmtStringLength()`, die die maximale String-Breite aller Matrixelemente zurückgibt, und `public int[] getMaxElmtStringLengthProClmn()`, die die maximale String-Breite pro Matrixspalte als `int`-Array zurückgibt, hinzu. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```
// - Gibt die max. Länge der String-Versionen ""+elmt[i][j] aller Einträge zurück.
public int getMaxElmtStringLength() {
    int ret=0;
    int actlg=0;
    for (int i=0; i<getAmtRows(); ++i) {
        for (int j=0; j<getAmtClmns(); ++j) {
            actlg=(""+elmt[i][j]).length();
            if (ret<actlg) {
                ret=actlg;
            } // end of if
        } // end of for
    } // end of for
    return ret;
}

// - Gibt ein int-Array der Länge getAmtClmns() zurück, das die max. Länge
// - der String-Versionen ""+elmt[i][j] der Einträge pro Spalte j enthält.
public int[] getMaxElmtStringLengthProClmn() {
    int[] ret=new int[getAmtClmns()];
    int actlg=0;
    for (int j=0; j<getAmtClmns(); ++j) {
        ret[j]=0;
        for (int i=0; i<getAmtRows(); ++i) {
            actlg=(""+elmt[i][j]).length();
            if (  <actlg) {
                 =actlg;
            } // end of if
        } // end of for
    } // end of for
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_3` ab.

### 3.1.7 Aufgabe

Fügen Sie nun der Klasse `Matrix` von `LinEqSysCalculator_1_3` die statische Methode `public static String completeToLength(String s, int lg)`, die den String `s`, falls nötig von links mit Leerzeichen bis zur String-Breite `lg` ergänzt, zurückgibt.

```
// - Statische Methode: Falls lg grösser als die Länge des String s ist, so wird s links um so viele
// - Leerzeichen ergänzt bis die Länge lg erreicht ist, und zurückgegeben, sonst wird s zurückgegeben.
public static String completeToLength(String s, int lg) {
    String ret=s;
    for (int i=0; i<lg-s.length(); ++i) {
        ret=" "+ret;
    } // end of for
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_3` ab.



Nun fügen Sie der Klasse `Matrix`, wie unten gezeigt, die Methode `public String[] toStrings()` hinzu. Ergänzen Sie den Code im fehlenden, umrahmten Teil selbst.

```
// - Gibt eine mehrzeilige String-Repräsentation der Matrix zurück.
// - Die String-Elemente der Rückgabe enthalten die Elemente der Zeilen.
// - Pro Spalte werden die Elemente durch eventuelles Hinzufügen von
// - Leerzeichen links zu gleich langen Strings gemacht, damit eine rechteckige
// - Darstellung einfach möglich ist.
public String[] toStrings() {
    String[] ret=new String[getAmtRows()];
    int[] maxLgs=getMaxElmtStringLengthProClmn();
    for (int i=0; i<ret.length; ++i) {
        ret[i]="";
        for (int j=0; j<getAmtClmns(); ++j) {
            ret[i]=ret[i]+completeToLength(""+elmt[i][j],);
            if (j<getAmtClmns()-1) {
                ret[i]=ret[i]+" ";
            } // end of if
        } // end of for
    } // end of for
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_3` ab.

Fügen Sie nun in der Klasse `LinEqSysCalculator_1_3` die Methode `public static void display(Matrix matr)` hinzu, die das String-Array aus der `toString()`-Methode von `matr` mit zusätzlich umschliessenden eckigen Klammern pro Zeile ausgibt.

```
public static void display(Matrix matr) {
    String[] row=matr.toStrings();
    for (int i=0; i<row.length; ++i) {
        Kon.writeln("[ "+row[i]+" ]");
    } // end of for
}
```

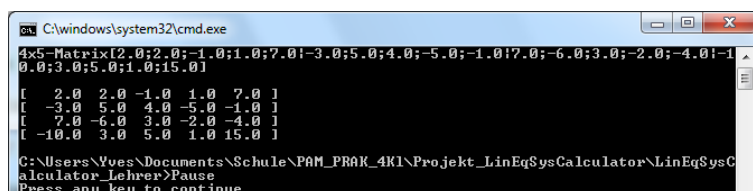
Ergänzen Sie schliesslich die `test()`-Methode durch folgende Codezeilen:

```
Kon.writeln();
display(matr1);
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_3` ab.

Kompilieren Sie nun `LinEqSysCalculator_1_3` und lassen Sie das Programm laufen.

Sie sollten in etwa die folgende Ausgabe erhalten:



```
C:\windows\system32\cmd.exe
4x5-Matrix[2.0;2.0;-1.0;1.0;7.0;-3.0;5.0;4.0;-5.0;-1.0;7.0;-6.0;3.0;-2.0;-4.0;-1.0;0.0;3.0;5.0;1.0;15.0]
[ 2.0 2.0 -1.0 1.0 7.0 ]
[ -3.0 5.0 4.0 -5.0 -1.0 ]
[ 7.0 -6.0 3.0 -2.0 -4.0 ]
[ -1.0 3.0 5.0 1.0 15.0 ]

C:\Users\Yves\Documents\Schule\PAM_PRAK_4K1\Projekt_LinEqSysCalculator\Lehrer>Pause
Press any key to continue . . .
```

Nun können wir also eine Matrix geeignet darstellen.

Wir wollen auch die Möglichkeit haben, eine Matrix vom Benutzer des Programms eingeben lassen zu können. Der Benutzer soll zuerst die Dimensionen der Matrix eingeben müssen und dann die Elemente zeilenweise oder spaltenweise.

### 3.1.8 Aufgabe

Speichern Sie `LinEqSysCalculator_1_3` unter `LinEqSysCalculator_1_4` neu ab.

Fügen Sie nun der Klasse `LinEqSysCalculator_1_4`, wie unten gezeigt, die Methoden `public static Matrix inputMatrix(boolean rowWise)`, die die Eingabe einer Matrix durch den Benutzer realisiert und als `Matrix`-Objekt zurückgibt, hinzu. Je nach Wert des Booleschen Parameters `rowWise` muss der Benutzer die Elemente der Matrix zeilenweise oder spaltenweise eingeben. Die parameterlose Methode `public static Matrix inputMatrix()` führt lediglich die zeilenweise-Variante aus. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```
public static Matrix inputMatrix() {
    return inputMatrix(true);
}

public static Matrix inputMatrix(boolean rowWise) {
    Matrix ret=new Matrix();
    int rows=0, clmns=0;
    do {
        Kon.write("Anzahl Zeilen ( > 0!): ");
        rows=Kon.readInt(true);
    } while (rows<=0); // end of do-while
    do {
        Kon.write("Anzahl Spalten ( > 0!): ");
        clmns=Kon.readInt(true);
    } while (clmns<=0); // end of do-while
    ret.init(rows,clmns);
    if (rowWise) {
        Kon.writeln("Matrixelemente zeilenweise eingeben (jede Eingabe mit [Enter] bestätigen): ");
        for (int i=0; i<ret.getAmtRows(); ++i) {
            Kon.writeln("Zeile "+i+": ");
            for (int j=0; j<ret.getAmtClmns(); ++j) {
                Kon.write(" a["+i+"]["+j+"] = ");
                ret.elmt[i][j]=Kon.readDouble(true);
            } // end of for
        } // end of for
    } else {
        Kon.writeln("Matrixelemente spaltenweise eingeben (jede Eingabe mit [Enter] bestätigen): ");
        for (int j=0; j< ; ++j) {
            Kon.writeln("Spalte "+j+": ");
            for (int i=0; i< ; ++i) {
                Kon.write(" a["+i+"]["+j+"] = ");
                ;
            } // end of for
        } // end of for
    } // end of if-else
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_4` ab.

Als letztes in diesem ersten Teil wollen wir diese Eingabemöglichkeit testen und ändern die `test()`-Methode von `LinEqSysCalculator_1_4` so ab, dass der Benutzer wählen kann, ob er die Beispielsmatrix laden will oder selber eine Matrix eingeben möchte.

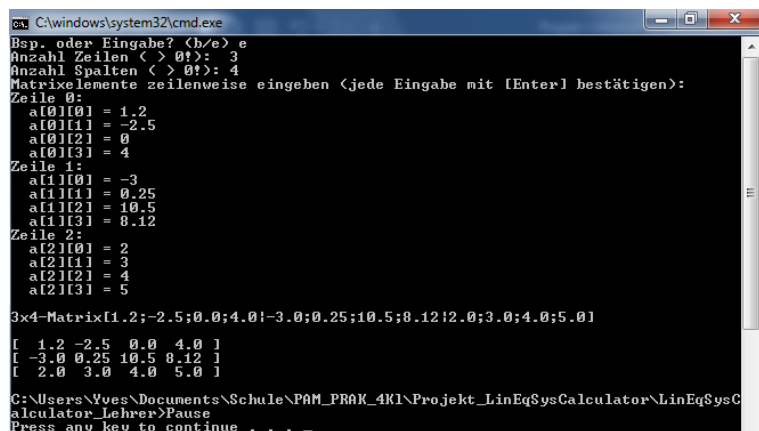
### 3.1.9 Aufgabe

Ergänzen bzw. ändern Sie den Code der Methode `test()` der Klasse `LinEqSysCalculator_1_4`, wie unten gezeigt.

```
public static void test() {
    Matrix matr1=new Matrix();
    char answ1=' ';
    Kon.write("Bsp. oder Eingabe? (b/e) ");
    answ1=Kon.readChar();
    if (answ1=='b' || answ1=='B') {
        matr1=getExample1();
    } else {
        matr1=inputMatrix(true);
    } // end of if-else
    Kon.writeln();
    Kon.writeln(matr1.toString());
    Kon.writeln();
    display(matr1);
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_1_4` ab.

Kompilieren Sie nun `LinEqSysCalculator_1_4` und lassen Sie das Programm laufen. Geben Sie nach der Frage 'e' ein und dann irgendeine Matrix. Das Ergebnis könnte so aussehen:



```
ca: C:\windows\system32\cmd.exe
Bsp. oder Eingabe? (b/e) e
Anzahl Zeilen ( > 0!): 3
Anzahl Spalten ( > 0!): 4
Matrixelemente zeilenweise eingeben (jede Eingabe mit [Enter] bestätigen):
Zeile 0:
a[0][0] = 1.2
a[0][1] = -2.5
a[0][2] = 0
a[0][3] = 4
Zeile 1:
a[1][0] = -3
a[1][1] = 0.25
a[1][2] = 10.5
a[1][3] = 8.12
Zeile 2:
a[2][0] = 2
a[2][1] = 3
a[2][2] = 4
a[2][3] = 5
3x4-Matrix[1.2;-2.5;0.0;4.0|-3.0;0.25;10.5;8.12|2.0;3.0;4.0;5.0]
[ 1.2 -2.5 0.0 4.0 ]
[ -3.0 0.25 10.5 8.12 ]
[ 2.0 3.0 4.0 5.0 ]
C:\Users\Yves\Documents\Schule\PAM_PRAK_4K1\Projekt_LinEqSysCalculator\LinEqSysC
calculator_Lehrer>Pause
Press any key to continue . . .
```

Ändern Sie dann zu Testzwecken den Wert des Parameters im Aufruf der `inputMatrix`-Methode in der Methode `test()` von `true` zu `false`. Kompilieren Sie das Programm wieder und lassen Sie es laufen. Nach Beantwortung der Frage mit 'e' sollten Sie nun die Matrixelemente spaltenweise eingeben müssen.

Nun können wir Matrizen eingeben und anzeigen lassen.

Für die weitere Entwicklung unseres Programms werden wir auf die `test()`-Methode verzichten. Dafür werden wir `LinEqSysCalculator` zu einem einfachen Kommando-Interpreter ausbauen.

## 4 Einfacher Kommando-Interpreter

In diesem zweiten Teil des Projekts wird `LinEqSysCalculator` schrittweise zu einem Kommando-Interpreter erweitert.

### 4.1 Ziel

Der Benutzer des Programms soll in der Standard-Konsole gewisse vorgefertigte Kommandos eingeben können, die das Programm anschließend interpretiert, indem es eine gewisse Handlung ausführt. Diese Handlung wird meistens eine Textausgabe beinhalten und manchmal auch anderes, wie z.B. das Beenden des Programms.

Dieser Zyklus bestehend aus der Benutzereingabe eines Kommandos und der darauffolgenden Interpretation durch das Programm soll sich wiederholen, bis der Benutzer das Kommando zum Beenden des Programms eingibt.

### 4.2 Vorgehen

Wir werden `LinEqSysCalculator_1_4` neu als `LinEqSysCalculator_2_0` abspeichern und zuerst nur in der Klasse `LinEqSysCalculator_2_0` Ergänzungen vornehmen. Die Klasse `Matrix` wird vorerst also noch nicht erweitert.

Als erstes werden wir nur Kommandos erstellen, die von uns schon vorbereitete Aufgaben übernehmen, d.h. das Laden der Beispielmatrix und das Eingeben einer Matrix, und natürlich ein Kommando zum Beenden des Programms. Später werden wir weitere Kommandos hinzufügen.

Da das Laden der Beispielmatrix und auch die Matrix-Eingabe immer ein Objekt der Klasse `Matrix` zurückgeben müssen, ist es sinnvoll, der Klasse `LinEqSysCalculator_2_0` ein Attribut der Klasse `Matrix` hinzuzufügen, das die aktuelle Matrix darstellt. Damit kann diese aktuelle Matrix jeweils als Rückgabe der Methoden für das Laden der Beispielmatrix bzw. die Benutzer-Eingabe verwendet werden und das Programm kann immer auf sie zugreifen. Das wird für spätere Zwecke nützlich sein.

Bei den Kommandos selbst werden wir uns an die englische Sprache halten bzw. anlehnen und z.B. das Kommando 'quit' für das Beenden des Programms wählen.

Die Kommandos werden wir immer in konstanten `String`-Attributen der jeweiligen Klasse `LinEqSysCalculator_...` speichern und die Methoden jeweils, falls nötig, nur auf diese Konstanten beziehen. Auf diese Weise können die Kommandos selbst, falls gewünscht, in einem späteren Zeitpunkt einfach verändert werden. Z.B. könnten die Kommandos eher an die deutsche Sprache angelehnt werden wollen, etwa 'beenden' statt 'quit'.

Die `test()`-Methode wird aus `LinEqSysCalculator_2_0` entfernt. Dafür wird eine neue Methode `public static void start()` hinzugefügt, die im Wesentlichen die Schleife für den Zyklus Benutzereingabe/Interpretation ausführt.

Die Eingabe-Einforderung vom Benutzer wird in der Methode `public static String getInputCmd()` realisiert, die Interpretation der Kommandos in der Methode `public static void interpret(String cmd)`.

Die `main`-Methode ruft nun die `start()`-Methode auf.

### 4.3 Gestaltung

Um unserem Kommando-Interpreter ein einigermaßen strukturiertes Aussehen zu verleihen, werden wir der Klasse `LinEqSysCalculator_2_0` einige Methoden zur Erzeugung und Darstellung von «Zeichenlinien» in der Standard-Konsole hinzufügen, d.h. von Textzeilen, die nur aus einem «Strich»-Zeichen wie z.B. '-' bestehen, mit eventuell eingebautem Text.

Diese sollen zur Ausgabe von Titeln und für die Gestaltung der Kommando-Eingabe durch den Benutzer und der Ausgabe durch das Programm verwendet werden.

Wir werden auch `String`-Konstanten für einen Eingabe-Prompt und einen Ausgabe-Prompt erstellen.

Die Methoden `public static Matrix inputMatrix(boolean rowWise)` bzw. `public static Matrix inputMatrix()` und `public static void display(Matrix matr)` werden zudem um einen Parameter `String prompt` ergänzt, der jeder Textzeile, die die Methode ausgibt, vorangestellt wird. Damit ist es möglich, die ganze Matriceingabe und -ausgabe nach rechts zu verschieben.

Da all diese Änderungen und Ergänzungen zusammengehören, werden wir nun das Ganze in einer einzigen Aufgabe schrittweise erledigen.

### 4.3.1 Aufgabe

Speichern Sie `LinEqSysCalculator_1_4` unter `LinEqSysCalculator_2_0` neu ab.

Fügen Sie nun der Klasse `LinEqSysCalculator_2_0` die unten gezeigten Konstanten und Attribute hinzu.

```
public class LinEqSysCalculator_2_0 {
// -- KONSTANTEN -- //
// - Programmname
public static final String PROGNAME="LinEqSysCalculator";

// - Kommandos
public static final String CMD_QUIT="quit";
public static final String CMD_EXAMPLE="exmpl";
public static final String CMD_INPUT="input";

// - char-Linien für die Darstellung
// - Max. Anz. Zeichen pro Zeile
public static final int MAXCHARLNLENGTH=78;
// - Linien
public static final String STD_LINE=charLine('-',MAXCHARLNLENGTH); // - Standardlinie
public static final String INPUT_LINE=titledLine("input",'_',MAXCHARLNLENGTH);
public static final String OUTPUT_LINE=titledLine("output", (char)175,MAXCHARLNLENGTH);
public static final String END_LINE=charLine('_',MAXCHARLNLENGTH);

// - Prompts für Ein- und Ausgabe
public static final String INPUT_PROMPT=">> ";
public static final String OUTPUT_PROMPT=":: ";
public static final String IN_PROMPT=charLine(' ',INPUT_PROMPT.length());
public static final String OUT_PROMPT=charLine(' ',OUTPUT_PROMPT.length());

// -- ATTRIBUTE -- //
// - Gibt an, ob das Programm beendet werden soll
private static boolean terminate=false;

// - Aktuelle Matrix
public static Matrix actualMatrix=new Matrix();
...
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_2_0` ab.

Fügen Sie nun der Klasse `LinEqSysCalculator_2_0`, wie unten gezeigt, im unteren Code-Bereich die Methoden `public static String charLine(char lineSegment, int length)`, `public static String centerInCharLine(char lineSegment, int lineLength, String center)`, `public static String blowUpByCharForSpace(String s, char charForSpace)` und `public static String titledLine(String title, char lineSegment, int lineLength)` hinzu. Lesen Sie dabei die Beschreibung in den Kommentaren des Codes und analysieren Sie danach den Code, bis Sie ihn verstehen.

```
...
// -- Methoden für die Darstellung von char-Linien und Titel
// - Gibt einen String zurück, der aus genau length Zeichen lineSegment besteht.
public static String charLine(char lineSegment, int length) {
    String ret="";
    for (int i=0; i<length; ++i) {
        ret=ret+lineSegment;
    } // end of for
    return ret;
}

// - Mittet den String center innerhalb von charLine(lineSegment,lineLength) ein und
// - gibt das Resultat zurück.
public static String centerInCharLine(char lineSegment, int lineLength, String center) {
    String ret=center;
    int amtLineSegments=lineLength-center.length();
    return charLine(lineSegment,amtLineSegments-amtLineSegments/2)+ret+charLine(lineSegment,amtLineSegments/2);
}
}
```

```

// - Zuerst werden eventuelle Leerzeichen in s durch das Zeichen charForSpace ersetzt.
// - Nach jedem Zeichen wird dann ein Zeichen charForSpace zusätzlich eingefügt.
// - s wird also durch charForSpace "aufgeblasen".
public static String blowUpByCharForSpace(String s, char charForSpace) {
    String sh=s.replace(' ',charForSpace);
    String ret="";
    for (int i=0; i<sh.length(); ++i) {
        ret=ret+sh.charAt(i)+charForSpace;
    } // end of for
    return ret;
}

// - Gibt den Titel title eingemittet und "aufgeblasen" in einer char-Linie
// - aus Zeichen lineSegment zurück.
public static String titledLine(String title, char lineSegment, int lineLength) {
    return centerInCharLine(lineSegment,lineLength,blowUpByCharForSpace(title,lineSegment));
}

} // end of class LinEqSysCalculator_2_0

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_2_0` ab.

Ergänzen bzw. ändern Sie den Code der Methode `public static Matrix inputMatrix(boolean rowWise)` bzw. `public static Matrix inputMatrix()` und `public static void display(Matrix matr)` der Klasse `LinEqSysCalculator_2_0`, wie unten angedeutet. Untersuchen Sie selbst im nicht abgedruckten Teil des Codes (mit 3 Pünktchen «...» gekennzeichnet), wo der `String`-Parameter `prompt` noch vorangestellt werden muss.

```

// - Fordert die Eingabe einer Matrix auf und gibt diese zurück. Falls der Parameter
// - rowWise true ist, so müssen die Matricelemente zeilenweise eingegeben werden,
// - sonst spaltenweise. Vor jeder ausgegebenen Zeile wird der Parameter prompt
// - vorangestellt.
public static Matrix inputMatrix(boolean rowWise, String prompt) {
    Matrix ret=new Matrix();
    int rows=0, clmns=0;
    do {
        Kon.write(prompt+"Anzahl Zeilen ( > 0!): ");
        ...
    } while (rows<=0); // end of do-while
    do {
        Kon.write(prompt+"Anzahl Spalten ( > 0!): ");
        ...
    } while (clmns<=0); // end of do-while
    ...
    return ret;
}

// - Fordert die Eingabe einer Matrix auf (Elemente zeilenweise) und gibt diese zurück.
// - Vor jeder ausgegebenen Zeile wird der Parameter prompt vorangestellt.
public static Matrix inputMatrix(String prompt) {
    return inputMatrix(true,prompt);
}

// - Gibt die Matrix matr geeignet aus; vor jeder Zeile wird der Parameter prompt
// - vorangestellt.
public static void display(Matrix matr, String prompt) {
    String[] row=matr.toStrings();
    for (int i=0; i<row.length; ++i) {
        Kon.writeln(prompt+"["+row[i]+" ]");
    } // end of for
}

// - Gibt die Matrix matr geeignet aus (ohne prompt).
public static void display(Matrix matr) {
    display(matr,"");
}
}

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_2_0` ab.

Fügen Sie schliesslich der Klasse `LinEqSysCalculator_2_0`, wie unten gezeigt, im oberen Code-Bereich die Methoden `public static void start()`, `public static String getInputCmd()`, `public static void output(String s)`, `public static void interpret(String cmd)` und `public static String getCmdInfo()` hinzu und rufen Sie die `start()`-Methode in der `main`-Methode auf. Die Methode `output(String s)` gibt den `String s` geeignet in der Standard-Konsole aus, die Methode `getCmdInfo()` gibt eine Liste der verfügbaren Kommandos als `String` zurück, die am Anfang zur Information angezeigt werden soll. Der Zweck der anderen Methoden wurde schon vorher besprochen. Schauen Sie sich dabei den Code wieder genau an, um ihn gut zu verstehen.

```
...
// -- METHODEN -- //
// - main-Methode
public static void main(String[] args) {
    start();
} // end of main

// - Startet das Programm und führt eine Ein-/Ausgabe-Schleife aus.
public static void start() {
    Kon.writeln(STD_LINE);
    Kon.writeln(titledLine(PROGNAME, ' ', MAXCHARLNLENGTH));
    Kon.writeln(STD_LINE);
    Kon.writeln(getCmdInfo());
    do {
        interpret(getInputCmd());
    } while (!terminate); // end of do-while
}

// - Stellt den Eingabebereich dar, wartet auf eine Eingabe und gibt diese zurück.
public static String getInputCmd() {
    Kon.writeln(INPUT_LINE);
    Kon.write(INPUT_PROMPT);
    return Kon.readString();
}

// - Stellt den Ausgabebereich dar und gibt s aus.
public static void output(String s) {
    Kon.writeln(OUTPUT_LINE);
    Kon.write(OUTPUT_PROMPT);
    Kon.writeln(s);
}

// - Interpretiert das Kommando cmd
public static void interpret(String cmd) {
    if (cmd.equals(CMD_QUIT)) {
        terminate=true;
        output(PROGNAME+" beendet!");
        Kon.writeln(END_LINE);
    } else if (cmd.equals(CMD_EXAMPLE)) {
        actualMatrix=getExample1();
        output("Beispielsmatrix: ");
        display(actualMatrix,OUT_PROMPT);
    } else if (cmd.equals(CMD_INPUT)) {
        actualMatrix=inputMatrix(IN_PROMPT);
        output("Eingegebene Matrix: ");
        display(actualMatrix,OUT_PROMPT);
    } else {
        output("Kommando '"+cmd+"' existiert nicht!");
    } // end of if-else
}

// - Gibt die Liste der Kommandos zurück.
public static String getCmdInfo() {
    String ret="Kommandos: "+CMD_QUIT+", "+CMD_EXAMPLE+", "+CMD_INPUT;
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_2_0` ab.



Kompilieren Sie nun `LinEqSysCalculator_2_0` und lassen Sie das Programm laufen. Geben Sie z.B. das Kommando 'exmpl' ein und danach 'quit'. Das Ergebnis sollte etwa so aussehen:

```

C:\windows\system32\cmd.exe

LinEqSysCalculator

Kommandos: quit, exmpl, input

i_n_p_u_t
>> exmpl
o_u_t_p_u_t
:: Beispielsmatrix:
[ 2.0 2.0 -1.0 1.0 7.0 ]
[ -3.0 5.0 4.0 -5.0 -1.0 ]
[ 7.0 -6.0 3.0 -2.0 -4.0 ]
[ -10.0 3.0 5.0 1.0 15.0 ]
i_n_p_u_t
>> quit
o_u_t_p_u_t
:: LinEqSysCalculator beendet!

C:\Users\Vves\Documents\Schule\PAM_PRAK_4K1\Projekt_LinEqSysCalculator\LinEqSysCalculator_Lehrer>Pause
Press any key to continue . . .

```

Lassen Sie `LinEqSysCalculator_2_0` wieder laufen und testen Sie auch das Kommando 'input'. Das Ergebnis könnte so aussehen:

```

C:\windows\system32\cmd.exe

LinEqSysCalculator

Kommandos: quit, exmpl, input

i_n_p_u_t
>> input
Anzahl Zeilen (> 0?): 2
Anzahl Spalten (> 0?): 3
Matrixelemente zeilenweise eingeben (jede Eingabe mit [Enter] bestätigen):
Zeile 0:
a[0][0] = -1
a[0][1] = 2.3
a[0][2] = 4.01
Zeile 1:
a[1][0] = 3
a[1][1] = -2.5
a[1][2] = 10
o_u_t_p_u_t
:: Eingegebene Matrix:
[ -1.0 2.3 4.01 ]
[ 3.0 -2.5 10.0 ]
i_n_p_u_t
>> quit
o_u_t_p_u_t
:: LinEqSysCalculator beendet!

C:\Users\Vves\Documents\Schule\PAM_PRAK_4K1\Projekt_LinEqSysCalculator\LinEqSysCalculator_Lehrer>Pause
Press any key to continue . . .

```

Unser Kommando-Interpreter `LinEqSysCalculator` funktioniert also grundsätzlich und hat ein vernünftiges Aussehen. Es kann doch schon drei Kommandos korrekt ausführen und bei einer Fehleingabe eine Fehlermeldung ausgeben.

Bevor wir daran gehen, die Klasse `Matrix` zu erweitern, werden wir uns zuerst um etwas Benutzerfreundlichkeit bemühen. Es ist gute Sitte, dem Benutzer eines Kommando-Interpreters einen kurzen Hilfetext zu den zur Verfügung stehenden Kommandos anzubieten.

Wir werden dazu eine Methode `public static String[] getCMDHelp()` hinzufügen und das dazugehörige Kommando 'cmdhelp', mit dem der Hilfetext ausgegeben werden kann. Zudem werden wir das Kommando 'cmd' ergänzen, mit dem der Benutzer die Liste der Kommandos anzeigen lassen kann. Für die Ausgabe des Hilfetexts werden wir auch überladene Methoden `public static void display(String[] line, String prompt)` und `public static void display(String[] line)` ergänzen.

Da wir schon dabei sind, wäre es wünschenswert, ein Kommando zu haben, mit dem die aktuelle Matrix angezeigt werden kann. Da dieses Kommando vom Benutzer sehr wahrscheinlich häufig eingegeben wird, entscheiden wir uns, die leere Eingabe dafür zu verwenden. Wenn also der Benutzer nichts eingibt und lediglich die [Enter]-Taste drückt, soll die aktuelle Matrix angezeigt werden. Trotzdem werden wir auch ein Kommando 'out' hinzufügen, das dasselbe bewirkt, damit jedes Kommando auch eine nicht-leere Bezeichnung hat. Der Benutzer kann also 'out' eingeben oder nichts, beides soll die Anzeige der aktuellen Matrix bewirken.

### 4.3.2 Aufgabe

Speichern Sie LinEqSysCalculator\_2\_0 unter LinEqSysCalculator\_2\_1 neu ab.

Fügen Sie nun der Klasse LinEqSysCalculator\_2\_1, wie unten gezeigt, die neuen, markierten Kommandos und die Methoden `public static String[] getCMDHelp()`, die einen Hilfetext zu den Kommandos als String-Array zurückgibt, hinzu. Ändern Sie auch die Methode `public static String getCmdInfo()`, wie im unten markierten Teil gezeigt, ab.

```
public class LinEqSysCalculator_2_1 {
    // -- KONSTANTEN -- //
    // - Programmname
    public static final String PROGNAME="LinEqSysCalculator";

    // - Kommandos
    public static final String CMD_QUIT="quit";
    public static final String CMD_EXAMPLE="exmpl";
    public static final String CMD_INPUT="input";
    public static final String CMD_OUT="out";
    public static final String CMD_CMDINFO="cmd";
    public static final String CMD_CMDHELP="cmdhelp";

    ...
    // - Gibt die Liste der Kommandos zurück.
    public static String getCmdInfo() {
        String ret="Kommandos: "+CMD_QUIT+" "+CMD_CMDHELP+" "+CMD_CMDINFO+" "+CMD_EXAMPLE+" "+CMD_INPUT+" "+CMD_OUT;
        return ret;
    }

    // - Gibt einen Hilfetext zu den Kommandos zurück.
    public static String[] getCMDHelp() {
        String[] ret=new String[6];
        ret[0]=CMD_QUIT+": "+PROGNAME+" wird beendet. ";
        ret[1]=CMD_CMDHELP+": Dieser Hilfetext wird ausgegeben. ";
        ret[2]=CMD_CMDINFO+": Die Liste der verfügbaren Kommandos wird ausgegeben. ";
        ret[3]=CMD_EXAMPLE+": Die Beispielsmatrix wird als aktuelle Matrix gesetzt. ";
        ret[4]=CMD_INPUT+": Eine neue Matrix kann als aktuelle Matrix eingegeben werden. ";
        ret[5]=CMD_OUT+": Die aktuelle Matrix wird ausgegeben (auch bei leerer Eingabe). ";
        return ret;
    }
}
```

Speichern Sie die Datei wieder unter LinEqSysCalculator\_2\_1 ab.

Ergänzen Sie nun die `interpret`-Methode, wie unten im markierten Teil gezeigt, so, dass die neuen Kommandos nun auch korrekt interpretiert werden, und die Methoden `public static void display(String[] line, String prompt)` und `public static void display(String[] line)`.

```
// - Interpretiert das Kommando cmd
public static void interpret(String cmd) {
    if (cmd.equals(CMD_QUIT)) {
        terminate=true;
        output(PROGNAME+" beendet!");
        Kon.writeln(END_LINE);
    } else if (cmd.equals(CMD_EXAMPLE)) {
        actualMatrix=getExample1();
        output("Beispielsmatrix: ");
        display(actualMatrix,OUT_PROMPT);
    } else if (cmd.equals(CMD_INPUT)) {
        actualMatrix=inputMatrix(IN_PROMPT);
        output("Eingegebene Matrix: ");
        display(actualMatrix,OUT_PROMPT);
    } else if (cmd.equals(CMD_OUT) || cmd.equals("")) {
        output("Aktuelle Matrix: ");
        display(actualMatrix,OUT_PROMPT);
    } else if (cmd.equals(CMD_CMDINFO)) {
        output(getCmdInfo());
    } else if (cmd.equals(CMD_CMDHELP)) {
        output("Hilfe zu den Kommandos: ");
        display(getCMDHelp(),OUT_PROMPT);
    } else {

```

```

        output("Kommando '"+cmd+"' existiert nicht!");
    } // end of if-else
}

...
// - Gibt das String-Array line zeilenweise aus; or jeder Zeile wird der
// - Parameter promptvorangestellt.
public static void display(String[] line, String prompt) {
    for (int i=0; i<line.length; ++i) {
        Kon.writeln(prompt+line[i]);
    } // end of for
}

// - Gibt as String-Array line zeilenweise aus (ohne prompt).
public static void display(String[] line) {
    display(line, "");
}

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_2_1` ab.

Kompilieren Sie nun `LinEqSysCalculator_2_1`, lassen Sie das Programm laufen und testen Sie die neuen Kommandos.

Wir sind bald soweit, dass wir unser Hauptziel, nämlich die Lösung von LGS mit der Determinantenmethode, weiterverfolgen können.

Vorher wollen wir aber unserem Kommando-Interpreter `LinEqSysCalculator` noch zwei sehr nützliche Fähigkeiten hinzufügen, die Möglichkeit nämlich, die aktuelle Matrix in eine Datei speichern und sie von einer solchen Datei wieder laden zu können. Damit können z.B. eingegebene Matrizen als Datei gespeichert werden und zu einem später Zeitpunkt bequem wieder geladen werden, anstatt sie erneut eingeben zu müssen.

Dazu wurde die Klasse `MatrixUtl` bereitgestellt. Sie exportiert statische Methoden zum Speichern von Objekten der Klasse `Matrix` in Textdateien und zum Laden von `Matrix`-Objekten aus solchen Dateien. Die Dateien haben die Dateierweiterung `'.data'`, sind aber lediglich Textdateien und können mit jedem Texteditor gelesen werden.

Die Klassendatei `MatrixUtl.class` ist im Unterordner `«src»` Ihres Projekt-Ordners `«LinEqSysCalculator»` schon vorhanden. In Ihrem Projekt-Ordner `«LinEqSysCalculator»` befindet sich auch der noch leere Unterordner `«MatrixData»`. Alle `'.data'`-Dateien, die von unserem Kommando-Interpreter erstellt werden, sollen in diesen Unterordner `«MatrixData»` gespeichert werden.

Die Klasse `MatrixUtl` stellt die Methoden `public static void save(Matrix matr)` und `public static Matrix load()` zum Speichern eines `Matrix`-Objekts in eine `'.data'`-Datei bzw. zum Laden einer `'.data'`-Datei in ein `Matrix`-Objekt zur Verfügung.

Sie bietet auch Methoden zur Erkennung von Fehlern bei den Dateioperationen und für dazugehörige Fehlermeldungen an. Mit der `MatrixUtl`-Methode `public static boolean errorOccured()` kann überprüft werden, ob ein Fehler geschehen ist, mit der Methode `public static int getError()` die dazugehörige Fehlernummer zurückgeholt werden und mit der Methode `public static String errorMsg(int e)` die Fehlermeldung. Zudem kann, falls kein Fehler aufgetreten ist, mit der Methode `public static String getActualFileName()` der von aktuellen Dateioperation verwendete Dateinamen zurückgeholt werden.

Mit diesen `MatrixUtl`-Methoden ist es einfach, unseren Kommando-Interpreter geeignet zu erweitern.

### 4.3.3 Aufgabe

Speichern Sie LinEqSysCalculator\_2\_1 unter LinEqSysCalculator\_2\_2 neu ab. Fügen Sie nun der Klasse LinEqSysCalculator\_2\_2, wie unten gezeigt, die neuen markierten Kommandos hinzu.

```
public class LinEqSysCalculator_2_2 {
    // -- KONSTANTEN -- //
    // - Programmname
    public static final String PROGNAME="LinEqSysCalculator";

    // - Kommandos
    public static final String CMD_QUIT="quit";
    public static final String CMD_EXAMPLE="exmpl";
    public static final String CMD_INPUT="input";
    public static final String CMD_OUT="out";
    public static final String CMD_CMDINFO="cmd";
    public static final String CMD_CMDHELP="cmdhelp";
    public static final String CMD_SAVE="save";
    public static final String CMD_LOAD="load";
    ...
}
```

Ergänzen Sie nun, wie im unten markierten Teil gezeigt, die interpret-Methode so, dass die neuen Kommandos geeignet interpretiert werden. Beachten Sie dabei, wie die Ausgabe auch im Fall eines Datei-Fehlers geeignet mit Hilfe der MatrixUtlt-Methoden gestaltet ist. Ergänzen Sie auch die Methoden public static String getCmdInfo() und public static String[] getCMDHelp(), wie im markierten Teil gezeigt.

```
// - Interpretiert das Kommando cmd
public static void interpret(String cmd) {
    if (cmd.equals(CMD_QUIT)) {
        ...
    } else if (cmd.equals(CMD_CMDHELP)) {
        output("Hilfe zu den Kommandos: ");
        display(getCMDHelp(), OUT_PROMPT);
    } else if (cmd.equals(CMD_SAVE)) {
        MatrixUtlt.save(actualMatrix);
        if (MatrixUtlt.errorOccurred()) {
            output("Datei-Fehler: "+MatrixUtlt.errorMsg(MatrixUtlt.getError()));
        } else {
            output("Aktuelle Matrix in "+MatrixUtlt.getActualFileName()+" gespeichert.");
        } // end of if-else
    } else if (cmd.equals(CMD_LOAD)) {
        actualMatrix=MatrixUtlt.load();
        if (MatrixUtlt.errorOccurred()) {
            output("Datei-Fehler: "+MatrixUtlt.errorMsg(MatrixUtlt.getError()));
        } else {
            output("Aktuelle Matrix von "+MatrixUtlt.getActualFileName()+" geladen.");
        } // end of if-else
    } else {
        output("Kommando '"+cmd+"' existiert nicht!");
    } // end of if-else
}

...
// - Gibt die Liste der Kommandos zurück.
public static String getCmdInfo() {
    String ret="Kommandos: "+CMD_QUIT+", "+CMD_CMDHELP+", "+CMD_CMDINFO+", "+CMD_EXAMPLE+", "+CMD_INPUT+",
    "+CMD_OUT+", "+CMD_SAVE+", "+CMD_LOAD;
    return ret;
}

// - Gibt einen Hilfetext zu den Kommandos zurück.
public static String[] getCMDHelp() {
    String[] ret=new String[8];
    ret[0]=CMD_QUIT+": "+PROGNAME+" wird beendet. ";
    ...
    ret[6]=CMD_SAVE+": Die akt. Matrix wird in eine "+MatrixUtlt.FILE_EXT+"-Datei gespeichert. ";
    ret[7]=CMD_LOAD+": Die akt. Matrix wird von einer "+MatrixUtlt.FILE_EXT+"-Datei geladen. ";
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_2_2` ab.

Kompilieren Sie nun `LinEqSysCalculator_2_2` und lassen Sie das Programm laufen. Geben Sie das Kommando `'exmpl'` ein und danach `'save'`. Ein *FileSelection*-Dialog-Fenster zum Speichern sollte sich öffnen. Navigieren Sie in diesem Dialog-Fenster bis zum Unterordner «MatrixData» Ihres Projektordners und speichern Sie darin die Datei unter dem Namen `'example'` ab. Beachten Sie die darauf in `LinEqSysCalculator` angezeigte Vollzugsmeldung.

Geben Sie nun `'quit'` ein und beenden Sie `LinEqSysCalculator`.

Lassen Sie danach `LinEqSysCalculator_2_2` wieder laufen. Drücken Sie dann die [Enter]-Taste, um die aktuelle Matrix, die leer sein sollte, anzeigen zu lassen.

Geben Sie dann `'load'` ein. Ein *FileSelection*-Dialog-Fenster zum Öffnen sollte aufgehen und normalerweise gleich den Inhalt Ihres Ordners «MatrixData» anzeigen. Ansonsten navigieren Sie wieder zu «MatrixData». Wählen Sie nun im Dialog-Fenster die Datei `'example'` aus.

In `LinEqSysCalculator` wird wieder eine Vollzugsmeldung angezeigt. Drücken Sie nun erneut die [Enter]-Taste, um die aktuelle Matrix anzeigen zu lassen und kontrollieren Sie das Ergebnis der Dateioperationen.

Geben Sie dann wieder `'load'` ein und testen Sie die Ausgabe bei einem Datei-Fehler, indem Sie im *FileSelection*-Dialog auf «Abbrechen» klicken. In `LinEqSysCalculator` sollte darauf eine diesbezügliche Fehlermeldung erscheinen.

Verlassen Sie dann `LinEqSysCalculator` durch die Eingabe von `'quit'`.

Damit haben wir die Grundausstattung unseres Kommando-Interpreters `LinEqSysCalculator` zunächst einmal beendet. Wir können mit `LinEqSysCalculator` die Beispielsmatrix in die aktuelle Matrix laden, die aktuelle Matrix von Hand eingeben, die aktuelle Matrix in eine Datei speichern und aus einer Datei laden und die aktuelle Matrix anzeigen lassen. Zudem können wir die Kommandoliste ausgeben lassen und die Hilfe zu den Kommandos.

## 5 Lösen von LGS mit LinEqSysCalculator

Nun werden wir auch die Klasse `Matrix` erweitern. Da wir für die Lösung von LGS die Determinantenmethode verwenden wollen, werden wir uns zuerst um die Berechnung der Determinante einer quadratischen Matrix kümmern.

Wir beschränken uns zunächst auf höchstens 3-dimensionale quadratische Matrizen, deren Determinante nach der bekannten, expliziten «Diagonalmethode» berechnet werden kann:

$$|a_{0,0}| = a_{0,0}$$

$$\begin{vmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{vmatrix} = a_{0,0} \cdot a_{1,1} - a_{0,1} \cdot a_{1,0}$$

$$\begin{vmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{vmatrix} = a_{0,0} \cdot a_{1,1} \cdot a_{2,2} + a_{0,1} \cdot a_{1,2} \cdot a_{2,0} + a_{0,2} \cdot a_{1,0} \cdot a_{2,1} \\ - a_{0,2} \cdot a_{1,1} \cdot a_{2,0} - a_{0,1} \cdot a_{1,0} \cdot a_{2,2} - a_{0,0} \cdot a_{1,2} \cdot a_{2,1}$$

### 5.1 Vorbereitung

Zuerst werden wir die Klasse `LinEqSysCalculator_2_2` neu unter `LinEqSysCalculator_3_0` abspeichern.

Dann werden wir der Klasse `Matrix` eine neue Methode `public double det()` hinzufügen, die die Determinante des `Matrix`-Objekt zurückgeben soll. Da die Determinante nur für quadratische Matrizen definiert ist, werden wir auch eine Methode `public boolean isSquare()` ergänzen, die überprüft, ob das `Matrix`-Objekt quadratisch ist, d.h. gleiche Anzahl Zeilen wie Spalten hat. Sie wird in der `det()`-Methode aufgerufen werden.

Falls nun das `Matrix`-Objekt nicht quadratisch oder leer ist, so soll die `det()`-Methode die `Double`-Konstante `Double.NaN` zurückgeben. 'NaN' steht für 'Not a Number', d.h. dass diese Konstante sich von allen anderen `Double`-Objekten bzw. `double`-Zahlen unterscheidet und für «undefiniert» verwendet werden kann. Mit der statischen `Double`-Methode `public static boolean isNaN(double v)` kann überprüft werden, ob eine `double`-Zahl gleich `Double.NaN` ist.

Da wir uns vorerst auf höchstens 3-dimensionale Matrizen beschränken, wird die `det()`-Methode auch bei grösseren Matrizen zunächst noch `Double.NaN` zurückgeben.

Schliesslich werden wir unserem Kommando-Interpreter ein neues Kommando 'det' hinzufügen, das die Ausgabe der Determinante der aktuellen `Matrix` in `LinEqSysCalculator` bewirken soll, falls jene quadratisch und klein genug ist. Sonst soll eine Fehlermeldung erscheinen.

### 5.1.1 Aufgabe

Speichern Sie `LinEqSysCalculator_2_2` unter `LinEqSysCalculator_3_0` neu ab.

Fügen Sie nun der Klasse `Matrix` in `LinEqSysCalculator_3_0`, wie unten gezeigt, die Methoden `public boolean isSquare()` und `public double det()` hinzu. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```
class Matrix {
    ...
    ...
    // - Prüft, ob die Matrix quadratisch ist.
    public boolean isSquare() {
        return getAmtRows() == getAmtClnms();
    }

    // - Falls die Matrix nicht quadratisch oder leer ist, so wird Double.NaN
    // - zurückgegeben. Andernfalls wird die Determinante der quadratischen Matrix
    // - berechnet und zurückgegeben.
    public double det() {
        double ret = Double.NaN;
        if (isSquare()) {
            int dim = getAmtRows();
            if (dim > 0) {
                if (dim == 1) {
                    ret = ;
                } else if (dim == 2) {
                    ret = ;
                } else if (dim == 3) {
                    ret = elmt[0][0]*elmt[1][1]*elmt[2][2] + elmt[0][1]*elmt[1][2]*elmt[2][0]
                        + elmt[0][2]*elmt[1][0]*elmt[2][1] - elmt[2][0]*elmt[1][1]*elmt[0][2]
                        - elmt[2][1]*elmt[1][2]*elmt[0][0] - elmt[2][2]*elmt[1][0]*elmt[0][1];
                } else {
                    // - Code für dim >= 4
                } // end of if-else
            } // end of if
        } // end of if
        return ret;
    }

    // -- Darstellungsmethoden -- //
    ...
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_0` ab.

Fügen Sie nun der Klasse `LinEqSysCalculator_3_0`, wie unten gezeigt, das neue markierte Kommando hinzu.

```
public class LinEqSysCalculator_3_0 {
    // -- KONSTANTEN -- //
    // - Programmname
    public static final String PROGNAME = "LinEqSysCalculator";

    // - Kommandos
    public static final String CMD_QUIT = "quit";
    ...
    public static final String CMD_LOAD = "load";
    public static final String CMD_DET = "det";

    ...
}
```

Ergänzen Sie nun, wie im unten markierten Teil gezeigt, die `interpret`-Methode so, dass das neue Kommando geeignet interpretiert wird, und auch die Methoden `public static String getCmdInfo()` und `public static String[] getCMDHelp()`.

```
// - Interpretiert das Kommando cmd
public static void interpret(String cmd) {
    if (cmd.equals(CMD_QUIT)) {
        ...
    } else if (cmd.equals(CMD_LOAD)) {
        ...
    } else if (cmd.equals(CMD_DET)) {
        double det1=actualMatrix.det();
        if (Double.isNaN(det1)) {
            output("Die Determinante der akt. Matrix ist nicht definiert! ");
        } else {
            output("Determinante der akt. Matrix: "+det1);
        } // end of if-else
    } else {
        output("Kommando '"+cmd+"' existiert nicht!");
    } // end of if-else
}

...
// - Gibt die Liste der Kommandos zurück.
public static String getCmdInfo() {
    String ret="Kommandos: "+CMD_QUIT+", "+CMD_CMDHELP+", "+CMD_CMDINFO+", "+CMD_EXAMPLE+", "+CMD_INPUT+",
    "+CMD_OUT+", "+CMD_SAVE+", "+CMD_LOAD+", "+CMD_DET;
    return ret;
}

// - Gibt einen Hilfetext zu den Kommandos zurück.
public static String[] getCMDHelp() {
    String[] ret=new String[9];
    ret[0]=CMD_QUIT+": "+PROGNAME+" wird beendet. ";
    ...
    ret[7]=CMD_LOAD+": Die akt. Matrix wird von einer "+MatrixUtl.FILE_EXT+"-Datei geladen. ";
    ret[8]=CMD_DET+": Die Determinante der akt. Matrix wird, falls möglich, berechnet. ";
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_0` ab.

Kompilieren Sie nun `LinEqSysCalculator_3_0`, lassen Sie das Programm laufen. Geben Sie in `LinEqSysCalculator` das Kommando 'input' ein und dann folgende 3x3-Matrix:

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

Geben Sie nun das Kommando 'det' ein. Die Determinante sollte  $-20$  ergeben.

Speichern Sie die aktuelle Matrix mittels Eingabe des Kommandos 'save' unter 'matrix\_3x3\_1' in Ihrem Ordner «MatrixData» ab und verlassen Sie `LinEqSysCalculator` durch die Eingabe von 'quit'. Die Ausgabe sollte etwa so aussehen:

```
C:\windows\system32\cmd.exe
LinEqSysCalculator
Kommandos: quit, cmdhelp, cmd, exampl, input, out, save, load, det
input
Anzahl Zeilen (> 0): 3
Anzahl Spalten (> 0): 3
Matrixelemente zeilenweise eingeben (jede Eingabe mit [Enter] bestätigen):
Zeile 0:
a[0][0] = 2
a[0][1] = -1
a[0][2] = 2
Zeile 1:
a[1][0] = -3
a[1][1] = 2
a[1][2] = 4
Zeile 2:
a[2][0] = 1
a[2][1] = 1
a[2][2] = 2
:: Eingebene Matrix:
[ 2.0 -1.0 2.0 ]
[ -3.0 2.0 4.0 ]
[ 1.0 1.0 2.0 ]
det
:: Determinante der akt. Matrix: -20.0
save
:: Aktuelle Matrix in matrix_3x3_1.data gespeichert.
quit
:: LinEqSysCalculator beendet!
```

Wir können also Determinanten von quadratischen, höchstens 3-dimensionale Matrizen berechnen. Bevor wir zur Berechnung von grösseren Determinanten übergehen, müssen wir uns vorher mit den Begriffen «Rekursion» und «Iteration» beschäftigen.



## 5.2 Exkurs: Rekursion und Iteration

Hier wird ein Exkurs über das Thema «Rekursion und Iteration» stattfinden, der in einem eigenen kleinen Projekt realisiert ist.

Laden Sie von Share-Point den Projekt-Ordner «Rekursion» in ihren Sammelordner «projects» herunter.

Öffnen Sie dann das Projekt-Dokument «Projekt\_Rekursion», das sich im Unterordner «Dokumente» Ihres Projekt-Ordners «Rekursion» befindet. Lesen Sie den Text von Anfang an aufmerksam durch und befolgen Sie die Anweisungen in den gestellten Aufgaben sorgfältig.

Nachdem wir nun einiges über Rekursion und Iteration wissen, können wir mit unserem Projekt `LinEqSysCalculator` weitermachen.

## 5.3 Determinanten mit Dimension grösser als 3

Zuerst werden wir den Begriff der Determinante einer quadratischen Matrix genauer definieren.

Es sei also  $A = (a_{i,j})_{i,j=0}^{n-1}$  eine quadratische,  $n$ -dimensionale Matrix. Wir betrachten das Produkt ihrer Diagonalelemente  $a_{0,0} \cdot a_{1,1} \cdot \dots \cdot a_{n-1,n-1}$  oder kurz mit dem Produktzeichen  $\prod_{i=0}^{n-1} a_{i,i}$ .

Nun ordnen wir die Zeilen von  $A$  anders an und betrachten auch das neue Diagonalprodukt.

Im Kapitel «2.2. Die Fakultät einer natürlichen Zahl» des Dokumentes «Projekt\_Rekursion» des Projekts «Rekursion» wird gezeigt, dass es genau  $n!$  Permutationen von  $n$  Objekten gibt, also genau  $n!$  Möglichkeiten, die  $n$  Matrixzeilen anders anzuordnen. Zudem besitzt jede Permutation ein bestimmtes Vorzeichen Plus oder Minus, je nachdem ob sie eine sogenannte «gerade» oder «ungerade» Permutation ist. Dies hat mit der Anzahl sogenannter Transpositionen zu tun, d.h. Vertauschungen von je genau 2 der Objekte, die nötig sind, um die betreffende Permutation herzustellen. Man kann zeigen, dass die Parität («Gerade- oder Ungerade sein») dieser Anzahl Transpositionen unabhängig von der gewählten Transpositionsfolge ist und also eine Eigenschaft der Permutation selbst ist. Das Vorzeichen der Permutation ist Plus, falls die benötigte Anzahl Transpositionen gerade ist, und Minus, falls sie ungerade ist.

Wir bilden nun die Diagonalprodukte der Matrix über alle möglichen Permutationen ihrer Zeilen und multiplizieren die Produkte noch mit dem Vorzeichen der jeweiligen Permutation. Falls wir all diese  $n!$  erweiterten Diagonalprodukte addieren, so erhalten wir nach Definition die Determinante der Matrix.

Um die Definition kompakter zu formulieren, bezeichnen wir mit  $S_n$  die Menge der Permutationen der  $n$  Indices  $\{0, 1, \dots, n-1\}$ . Es gilt  $|S_n| = n!$ .

Eine Permutation  $\sigma \in S_n$  ist nichts anderes als eine bijektive (eindeutige) Abbildung von  $\{0, 1, \dots, n-1\}$  nach  $\{0, 1, \dots, n-1\}$ , also  $\sigma: \{0, 1, \dots, n-1\} \rightarrow \{0, 1, \dots, n-1\}; i \mapsto \sigma(i)$ .

$\sigma(i)$  ist also der Platz, den die ursprüngliche  $i$ -te Zeile der Matrix nach Anwendung der Permutation  $\sigma$  nun hat. Weiter bezeichnen wir mit  $sign$  die Vorzeichenfunktion von  $S_n$ . Es gilt also  $sign: S_n \rightarrow \{-1, +1\}; \sigma \mapsto sign(\sigma)$ . Damit können wir die Definition einer Determinante kompakter formulieren.

### 5.3.1 Def.:

Es sei  $n \in \mathbb{N}$  und  $(a_{i,j})_{i,j=0}^{n-1}$  eine quadratische,  $n$ -dimensionale Matrix.

Ihre **Determinante**  $\left| (a_{i,j})_{i,j=0}^{n-1} \right|$  ist folgendermassen definiert:

$$\left| (a_{i,j})_{i,j=0}^{n-1} \right| := \sum_{\sigma \in S_n} \text{sign}(\sigma) \cdot \prod_{i=0}^{n-1} a_{\sigma(i),i}$$

Wenden wir Def. 5.3.1 auf die uns schon bekannten Fälle für  $n$  von 1 bis 3 an, so folgen natürlich die gleichen Berechnungsmethoden:

$$\left| (a_{i,j})_{i,j=0}^0 \right| = \sum_{\sigma \in S_1} \text{sign}(\sigma) \cdot \prod_{i=0}^0 a_{\sigma(i),i} = (+1) \cdot a_{0,0} = a_{0,0}$$

$$\begin{aligned} \left| (a_{i,j})_{i,j=0}^1 \right| &= \sum_{\sigma \in S_2} \text{sign}(\sigma) \cdot \prod_{i=0}^1 a_{\sigma(i),i} = (+1) \cdot a_{0,0} \cdot a_{1,1} + (-1) \cdot a_{1,0} \cdot a_{0,1} \\ &= a_{0,0} \cdot a_{1,1} - a_{0,1} \cdot a_{1,0} \end{aligned}$$

$$\begin{aligned} \left| (a_{i,j})_{i,j=0}^2 \right| &= \sum_{\sigma \in S_3} \text{sign}(\sigma) \cdot \prod_{i=0}^2 a_{\sigma(i),i} = (+1) \cdot a_{0,0} \cdot a_{1,1} \cdot a_{2,2} + (-1) \cdot a_{1,0} \cdot a_{0,1} \cdot a_{2,2} \\ &\quad + (-1) \cdot a_{2,0} \cdot a_{1,1} \cdot a_{0,2} + (-1) \cdot a_{0,0} \cdot a_{2,1} \cdot a_{1,2} + (+1) \cdot a_{1,0} \cdot a_{2,1} \cdot a_{0,2} + (+1) \cdot a_{2,0} \cdot a_{0,1} \cdot a_{1,2} \\ &= a_{0,0} \cdot a_{1,1} \cdot a_{2,2} + a_{0,1} \cdot a_{1,2} \cdot a_{2,0} + a_{0,2} \cdot a_{1,0} \cdot a_{2,1} \\ &\quad - a_{0,2} \cdot a_{1,1} \cdot a_{2,0} - a_{0,1} \cdot a_{1,0} \cdot a_{2,2} - a_{0,0} \cdot a_{1,2} \cdot a_{2,1} \end{aligned}$$

Also gut, nun wissen wir endlich genau, was die Determinante einer quadratischen Matrix ist. Sie ist die Summe über alle Permutationen aus  $S_n$  von den mit dem Vorzeichen der Permutation multiplizierten Diagonalprodukten der zeilen-permutierten Matrix.

Eine Berechnung der Determinante direkt nach Def. 5.3.1 ist aber nicht einfach zu implementieren, d.h. zu programmieren. Es müssten alle Permutationen aus  $S_n$  erzeugt werden können und auch ihr Vorzeichen. Es ist zwar sicher möglich, systematisch alle Permutationen zu erzeugen und damit eine iterative Methode, die wahrscheinlich mehrere verschachtelte Schleifen verwenden muss, zu programmieren, es ist aber nicht einfach und erfordert kombinatorische Überlegungen.

Es gibt aber eine Möglichkeit (sogar mehrere), die Berechnung einer Determinante rekursiv durchzuführen. Wir wählen dazu die Methode der «Entwicklung in Unterdeterminanten nach der nullten Zeile». Damit kann die Berechnung der Determinante einer  $n \times n$ -Matrix  $A$  auf die Summe der  $n$  Determinanten der  $(n-1) \times (n-1)$ -Untermatrizen zurückgeführt werden, die entstehen, wenn aus der Matrix  $A$  die nullte Zeile eliminiert wird und dann aufeinanderfolgend jeweils eine Kolonne von der nullten an mit abwechselndem Vorzeichen (beginnend bei Plus), die zusätzlich jeweils mit dem Element der nullten Zeile von  $A$  der aktuellen Kolonne multipliziert werden.

Es sei also  $A = (a_{i,j})_{i,j=0}^{n-1}$  eine  $n \times n$ -Matrix und es sei  $A^{[l,k]} = (a_{i,j})_{i,j=0}^{n-1}$  die  $(n-1) \times (n-1)$ -Matrix, die aus  $A$  entsteht, wenn die  $l$ -te Zeile und die  $k$ -te Spalte eliminiert werden.

$$A = \begin{pmatrix} a_{0,0} & \cdots & a_{0,k-1} & a_{0,k} & a_{0,k+1} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{l-1,0} & \cdots & a_{l-1,k-1} & a_{l-1,k} & a_{l-1,k+1} & \cdots & a_{l-1,n-1} \\ a_{l,0} & \cdots & a_{l,k-1} & a_{l,k} & a_{l,k+1} & \cdots & a_{l,n-1} \\ a_{l+1,0} & \cdots & a_{l+1,k-1} & a_{l+1,k} & a_{l+1,k+1} & \cdots & a_{l+1,n-1} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,k-1} & a_{n-1,k} & a_{n-1,k+1} & \cdots & a_{n-1,n-1} \end{pmatrix} \Rightarrow A^{[l,k]} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,k-1} & a_{0,k+1} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{l-1,0} & \cdots & a_{l-1,k-1} & a_{l-1,k+1} & \cdots & a_{l-1,n-1} \\ a_{l+1,0} & \cdots & a_{l+1,k-1} & a_{l+1,k+1} & \cdots & a_{l+1,n-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,k-1} & a_{n-1,k+1} & \cdots & a_{n-1,n-1} \end{pmatrix}$$

Nach dem Satz über die «Entwicklung in Unterdeterminanten nach der nullten Zeile» gilt also

$$|A| = (+1) \cdot a_{0,0} \cdot |A^{[0,0]}| + (-1) \cdot a_{0,1} \cdot |A^{[0,1]}| + \dots + (\pm 1) \cdot a_{0,n-1} \cdot |A^{[0,n-1]}|$$

wobei je nach Parität von  $n$  als letztes Vorzeichen eine Plus oder eine Minus steht. Um diese Notation eindeutig zu machen, ist es besser, das abwechselungsweise Vorzeichen mit Potenzen von  $-1$  zu realisieren, also  $|A| = (-1)^0 \cdot |A^{[0,0]}| + (-1)^1 \cdot |A^{[0,1]}| + \dots + (-1)^{n-1} \cdot |A^{[0,n-1]}|$  oder kurz mit dem Summenzeichen  $|A| = \sum_{k=0}^{n-1} (-1)^k \cdot a_{0,k} \cdot |A^{[0,k]}|$ .

Da für die Anfangswerte  $n = 1$ ,  $n = 2$  und  $n = 3$  explizite Berechnungsmethoden vorliegen, ist damit eine rekursive Definition der Determinante möglich.

### 5.3.2 Satz:

Es sei  $n \in \mathbb{N}$  und  $(a_{i,j})_{i,j=0}^{n-1}$  eine quadratische,  $n$ -dimensionale Matrix. Ihre Determinante kann folgendermassen rekursiv definiert werden:

$$\begin{aligned} |(a_{i,j})_{i,j=0}^{n-1}| &= \sum_{k=0}^{n-1} (-1)^k \cdot a_{0,k} \cdot |(a_{i,j})_{\substack{i,j=0 \\ i \neq 0, j \neq k}}^{n-1}|, \quad n > 3 \\ |(a_{i,j})_{i,j=0}^{n-1}| &= \sum_{\sigma \in S_n} \text{sign}(\sigma) \cdot \prod_{i=0}^{n-1} a_{\sigma(i),i}, \quad n \leq 3 \end{aligned}$$

Nach diesen eher theoretischen Ausführungen kehren wir wieder zu unserem Programm zurück.

Satz 5.3.2 gibt uns eine einfache Möglichkeit, unsere Berechnungsmethode `det()` in `LinEqSysCalculator` durch rekursive Aufrufe auf Dimensionen grösser als 3 zu erweitern.

Die Summe  $\sum_{k=0}^{n-1} \dots$  für  $n > 3$  kann in einer `for`-Schleife mitsamt abwechselungweisem Vorzeichen elegant und einfach realisiert werden.

Wir brauchen aber noch die Möglichkeit, aus der Matrix  $A = (a_{i,j})_{i,j=0}^{n-1}$  die um die 0-te Zeile und  $k$ -te Kolonne reduzierten Untermatrizen  $A^{[0,k]} = (a_{i,j})_{\substack{i,j=0 \\ i \neq 0, j \neq k}}^{n-1}$  herzustellen.

Dazu werden wir der Klasse `Matrix` eine Methode `public Matrix reduced(int rowInd, int clmnInd)` hinzufügen, die genau die um die Zeile `rowInd` und Kolonne `clmnInd` reduzierte Untermatrix zurückgibt.

### 5.3.3 Aufgabe

Öffnen Sie nun wieder Ihr Programm `LinEqSysCalculator_3_0` und speichern Sie es unter `LinEqSysCalculator_3_1` neu ab.

Fügen Sie nun der Klasse `Matrix` in `LinEqSysCalculator_3_1`, wie unten gezeigt, die Methoden `public Matrix reduced(int rowInd, int clmnInd)` hinzu und ergänzen Sie den Code der Methode `public double det()`, wie im markierten Teil gezeigt. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```

// - Falls die Matrix nicht quadratisch oder leer ist, so wird Double.NaN
// - zurückgegeben. Andernfalls wird die Determinante der quadratischen Matrix
// - berechnet und zurückgegeben.
public double det() {
    double ret=Double.NaN;
    if (isSquare()) {
        int dim=getAmtRows();
        if (dim>0) {
            if (dim==1) {
                ret=elmt[0][0];
            } else if (dim==2) {
                ret=elmt[0][0]*elmt[1][1]-elmt[1][0]*elmt[0][1];
            } else if (dim==3) {
                ret=elmt[0][0]*elmt[1][1]*elmt[2][2]+elmt[0][1]*elmt[1][2]*elmt[2][0]
                +elmt[0][2]*elmt[1][0]*elmt[2][1]-elmt[2][0]*elmt[1][1]*elmt[0][2]
                -elmt[2][1]*elmt[1][2]*elmt[0][0]-elmt[2][2]*elmt[1][0]*elmt[0][1];
            } else { // - Rekursiver Aufruf; "Entwicklung nach der nullten Zeile"
                int signl=1;
                ret=0;
                for (int j=0; j<getAmtClmns(); ++j) {
                    ret=ret+signl*elmt[0][j]*reduced(0,j).det();
                    signl=-signl;
                } // end of for
            } // end of if-else
        } // end of if
    } // end of if
    return ret;
}

// - Gibt die Matrix zurück, die aus der Elimination der Zeile mit Index rowInd
// - und der Spalte mit Index clmnInd aus der aktuellen Matrix entsteht.
// - Falls für den Parameter rowInd bzw. clmnInd kein möglicher Index angegeben
// - wird (z.B. ein negativer Wert), dann wird keine Zeile bzw. keine Spalte
// - entfernt.
// - Bsp.:
// - m2=m1.reduced(1,2) Zeile mit Index 1 und Spalte mit Index 2 wird entfernt
// - m2=m1.reduced(-1,2) nur Spalte mit Index 2 wird entfernt
// - m2=m1.reduced(1,-2) nur Zeile mit Index 1 wird entfernt
// - m2=m1.reduced(-1,-2) nichts wird entfernt
public Matrix reduced(int rowInd, int clmnInd) {
    Matrix ret=new Matrix();
    int retAmtRows= getAmtRows();
    int retAmtClmns= getAmtClmns();
    if (0<=rowInd && rowInd< getAmtRows()) {
        retAmtRows=retAmtRows-1;
    } // end of if
    if (0<=clmnInd && clmnInd< getAmtClmns()) {
        retAmtClmns=retAmtClmns-1;
    } // end of if
    ret.init(retAmtRows,retAmtClmns);
    int n=0, k=0;
    for (int i=0; i< getAmtRows(); ++i) {
        if (i!=rowInd) {
            k=0;
            for (int j=0; j< getAmtClmns(); ++j) {
                if (j!=clmnInd) {
                    ret.elmt[n][k]=elmt[i][j];
                    ++k;
                } // end of if
            } // end of for
            ++n;
        } // end of if
    } // end of for
    return ret;
}

```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_1` ab.

Kompilieren Sie nun `LinEqSysCalculator_3_1`, lassen Sie das Programm laufen. Geben Sie in `LinEqSysCalculator` das Kommando 'input' ein und dann folgende 4x4-Matrix:

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

Geben Sie nun das Kommando 'det' ein. Die Determinante sollte  $-103$  ergeben.

Speichern Sie die aktuelle Matrix mittels Eingabe des Kommandos 'save' unter 'matrix\_4x4\_1' in Ihrem Ordner «MatrixData» ab und verlassen Sie LinEqSysCalculator durch die Eingabe von 'quit'.

Wir können nun also Determinanten von beliebigen quadratischen Matrizen berechnen.

#### 5.4 Determinantenmethode

Nun können wir uns endlich der Determinantenmethode zur Lösung von LGS widmen. Wir erinnern uns, dass wir ein LGS in  $n$  Variablen und  $n$  Gleichungen

$$\begin{aligned} a_{0,0} \cdot x_0 + a_{0,1} \cdot x_1 + \cdots + a_{0,n-1} \cdot x_{n-1} &= a_{0,n} \\ a_{1,0} \cdot x_0 + a_{1,1} \cdot x_1 + \cdots + a_{1,n-1} \cdot x_{n-1} &= a_{1,n} \\ \vdots & \\ a_{n-1,0} \cdot x_0 + a_{n-1,1} \cdot x_1 + \cdots + a_{n-1,n-1} \cdot x_{n-1} &= a_{n-1,n} \end{aligned}$$

durch die  $n \times (n+1)$ -Matrix

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} & a_{0,n} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} & a_{n-1,n} \end{pmatrix}$$

darstellen wollen. Wir nennen diese Matrix die **LGS-Matrix**.

Die Determinantenmethode erfordert die Berechnung der Nennerdeterminante  $N$  und der  $n$  Zählerdeterminanten  $Z_i, i = 0, \dots, n-1$ .

Dazu muss aus der LGS-Matrix zuerst die Koeffizientenmatrix extrahiert werden, die aus den ersten  $n$  Spalten der LGS-Matrix besteht. Ihre Determinante ist die Nennerdeterminante  $N$ .

Für die Berechnung der  $n$  Zählerdeterminanten muss  $n$ -mal hintereinander eine Spalte der Koeffi-

zientenmatrix mit der Konstantenspalte  $\begin{pmatrix} a_{0,n} \\ a_{1,n} \\ \vdots \\ a_{n-1,n} \end{pmatrix}$  der LGS-Matrix ersetzt werden. Die Determinan-

ten der so entstehenden Matrizen sind die Zählerdeterminanten  $Z_i$ .

Falls die Nennerdeterminante ungleich Null ist, so ist das  $n$ -Tupel  $\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$  der Lösungen eindeutig

und man erhält die Lösungen durch Division der jeweiligen Zählerdeterminante durch die Nennerdeterminante, also  $x_i = \frac{Z_i}{N}, i = 0, \dots, n-1$ .

Falls die Nennerdeterminante gleich Null ist, so liegt ein Sonderfall vor. Ist in diesem Fall mindestens eine Zählerdeterminante ungleich Null, so hat das LGS keine Lösungen. Sind hingegen auch alle Zählerdeterminanten gleich Null, so gibt es unendlich viele Lösungen. Weiter werden wir hier nicht gehen und insbesondere noch auf die Bestimmung der Lösungsdimension in Fall von unendlich vielen Lösungen verzichten.

Um die eben beschriebenen Operationen in unserem Programm `LinEqSysCalculator` bequem durchführen zu können, werden wir zuerst der Klasse `Matrix` eine Methode `public Matrix extract(int startRowInd, int startClmnInd, int endRowInd, int endClmnInd)` hinzufügen, die die Matrix zurückgibt, die aus den Elementen der aktuellen Matrix mit Indices  $(i, j)$  zwischen  $(\text{startRowInd}, \text{startClmnInd})$  und  $(\text{endRowInd}, \text{endClmnInd})$  besteht, genauer mit  $\text{startRowInd} \leq i < \text{endRowInd}$  und  $\text{startClmnInd} \leq j < \text{endClmnInd}$ .

Die Methode `extract` ermöglicht es, einen beliebigen zusammenhängenden Block einer Matrix als eigene Matrix zu extrahieren. Damit wird es einfach sein, die drei nützlichen Methoden `public Matrix getRow(int ind)`, die die  $\text{ind}$ -te Zeile der Matrix als  $1 \times n$ -Matrix zurückgibt, `public Matrix getClmn(int ind)`, die die  $\text{ind}$ -te Spalte der Matrix als  $n \times 1$ -Matrix zurückgibt, und `public Matrix copy()`, die eine Kopie der Matrix als eigene Matrix zurückgibt, zu implementieren. Sie rufen alle lediglich die Methode `extract(...)` mit geeigneten Parametern auf.

Weiter werden wir der Klasse `Matrix` die Methode `public Matrix replaceFromPos(int startRowInd, int startClmnInd, Matrix source)` hinzufügen, die eine Kopie der Matrix zurückgibt, deren Elemente aber ab Position  $(\text{startRowInd}, \text{startClmnInd})$  durch die Elemente der Matrix `source` ersetzt werden (soweit es geht).

Wir führen all diese Vorbereitungen in der nächsten Aufgabe durch.

### 5.4.1 Aufgabe

Speichern Sie `LinEqSysCalculator_3_1` unter `LinEqSysCalculator_3_2` neu ab.

Fügen Sie nun der Klasse `Matrix` in `LinEqSysCalculator_3_2`, wie unten gezeigt, die Methode `public Matrix extract(int startRowInd, int startClmnInd, int endRowInd, int endClmnInd)` hinzu. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```
class Matrix {
    . . .
    . . .
    // - Die Elemente mit Indices (row,clmn), die startRowInd <= row < endRowInd
    // - und startClmnInd <= clmn < endClmnInd erfüllen, werden aus der Matrix
    // - extrahiert und als Elemente einer neuen Matrix zurückgegeben.
    // - Es muss startRowInd<=endRowInd und startClmnInd<=endClmnInd gelten, sonst
    // - wird die leere Matrix zurückgegeben.
    // - Falls startRowInd oder startClmnInd negativ ist, so wird es auf 0 gesetzt.
    // - Falls endRowInd grösser als getAmtRows() oder endClmnInd grösser als
    // - getAmtClmns(), so wird es auf getAmtRows() bzw. getAmtClmns() gesetzt.
    public Matrix extract(int startRowInd, int startClmnInd, int endRowInd, int endClmnInd) {
        Matrix ret=new Matrix();
        if (startRowInd<=endRowInd && startClmnInd<=endClmnInd) {
            if (startRowInd<0) {
                startRowInd=0;
            } // end of if
            if (startClmnInd<0) {
                startClmnInd=0;
            } // end of if
            if (endRowInd>getAmtRows()) {
                endRowInd=getAmtRows();
            } // end of if
            if (endClmnInd>getAmtClmns()) {
                endClmnInd=getAmtClmns();
            } // end of if
            ret.init(endRowInd-[ ],endClmnInd-startClmnInd);
            int retAmtRows=ret.getAmtRows();
            int retAmtClmns=ret.getAmtClmns();
            for (int i=0; i<retAmtRows; ++i) {
                for (int j=0; j<retAmtClmns; ++j) {
                    ret.elmt[i][j]=elmt[startRowInd+i][[ ]+j];
                } // end of for
            } // end of for
        } // end of if
        return ret;
    }

    // -- Darstellungsmethoden -- //
    . . .
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_2` ab.

Ergänzen Sie nun in der Klasse `Matrix` in `LinEqSysCalculator_3_2` nach dem Code der Methode `extract(...)`, wie unten gezeigt, die Methoden `public Matrix getRow(int ind)`, `public Matrix getClmn(int ind)` und `public Matrix copy()`, die alle lediglich die Methode `extract(...)` mit geeigneten Parametern aufrufen. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst.

```
// - Gibt die Zeile der Matrix mit Index ind als einzelige Matrix zurück.
// - Ruft extract(...) auf.
public Matrix getRow(int ind) {
    return extract(ind, 0,  ,  );
}

// - Gibt die Spalte der Matrix mit Index ind als einspaltige Matrix zurück.
// - Ruft extract(...) auf.
public Matrix getClmn(int ind) {
    return extract(0, ind, getAmtRows(), ind+1);
}

public Matrix copy() {
    return extract(0, 0, getAmtRows(), getAmtClmns());
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_2` ab.

Fügen Sie schliesslich der Klasse `Matrix` Klasse `Matrix` in `LinEqSysCalculator_3_2` nach dem Code der Methode `copy()` die Methoden `private boolean intersects(int startRowInd, int startClmnInd, Matrix source)` und `public Matrix replaceFromPos(int startRowInd, int startClmnInd, Matrix source)`, wie unten gezeigt, hinzu. Die Methode `intersects(...)` überprüft, ob überhaupt eine Überschneidung der Indexbereiche für die Ersetzung der Elemente durch die Quellmatrix `source` vorhanden ist. Sie wird in der Methode `replaceFromPos(...)` verwendet, um festzustellen, ob die Doppelschleife über alle Elemente von `source` ausgeführt werden soll. Ergänzen Sie den Code im fehlenden, umrahmten Teil selbst.

```
// - Überprüft, ob sich der Indexpaar-Bereich von (startRowInd, startClmnInd) bis
// - (startRowInd+source.getAmtRows(), startClmnInd+source.getAmtClmns()) mit
// - dem Indexpaar-Bereich der Elemente der Matrix überschneidet.
private boolean intersects(int startRowInd, int startClmnInd, Matrix source) {
    return startRowInd<getAmtRows() && startClmnInd<getAmtClmns()
        && startRowInd+source.getAmtRows()>=0 && startClmnInd+source.getAmtClmns()>=0;
}

// - Gibt eine Matrix zurück, die eine Kopie der akt. Matrix ist, wobei ab
// - Indexposition (startRowInd, startClmnInd), falls möglich, die Elemente der
// - Rückgabe-Matrix durch die Elemente der Quellmatrix source ersetzt werden.
public Matrix replaceFromPos(int startRowInd, int startClmnInd, Matrix source) {
    Matrix ret=copy();
    if (intersects(startRowInd, startClmnInd, source)) { // - Wird nur ausgeführt, falls es eine
        int amtRows=ret.getAmtRows(); // - Überschneidung der Indexbereiche gibt.
        int amtClmns=ret.getAmtClmns();
        int amtRowsSource=source.getAmtRows();
        int amtClmnsSource=source.getAmtClmns();
        for (int i=0; i<amtRowsSource; ++i) {
            for (int j=0; j<amtClmnsSource; ++j) {
                if (0<=startRowInd+i && startRowInd+i<amtRows && 0<=  &&
startClmnInd+j<amtClmns) {
                    ret.elmt[startRowInd+i][startClmnInd+j]=source.elmt[i][j];
                } // end of if
            } // end of for
        } // end of for
    } // end of if
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_2` ab.

Nun sind alle Vorbereitungen getroffen. Es fehlt nur noch ein Kommando 'solve' und eine `Matrix`-Methode, die die Lösung zurückgibt.

Wir werden der Klasse `Matrix` also eine Methode `public Matrix solve()` hinzufügen. Da diese Methode auf beliebige Objekte der Klasse `Matrix` angewendet werden kann, also auch z.B. auf solche, deren Anzahl Spalten nicht genau um 1 grösser ist als die Anzahl Zeilen und deren Koeffizientenmatrix also nicht quadratisch ist, müssen diese Fälle auch berücksichtigt werden. Es gibt grundsätzlich folgende Möglichkeiten:

- 1) Anzahl Spalten = Anzahl Zeilen + 1  $\Leftrightarrow$  Anzahl Gleichungen = Anzahl Unbekannte
  - a) Eindeutig lösbar
  - b) Sonderfall:
    - i) Unlösbar
    - ii) Unendlich viele Lösungen
- 2) Anzahl Spalten > Anzahl Zeilen + 1  $\Leftrightarrow$  Anzahl Gleichungen > Anzahl Unbekannte  
LGS ist überbestimmt
- 3) Anzahl Spalten < Anzahl Zeilen + 1  $\Leftrightarrow$  Anzahl Gleichungen < Anzahl Unbekannte  
LGS ist unterbestimmt

Wir werden diese 5 Fälle durch die Anzahl Spalten der Rückgabe-Matrix der Methode `solve()` kennzeichnen:

- 1) Anzahl Spalten = Anzahl Zeilen + 1  $\Leftrightarrow$  Anzahl Gleichungen = Anzahl Unbekannte
  - a) **Eindeutig lösbar  $\Rightarrow$  Anz. Spalten der Rückgabe = 1**, enthält die Lösungen als einspaltige Matrix mit Anzahl Zeilen = Anzahl Unbekannte
  - b) Sonderfall:
    - i) **Unlösbar  $\Rightarrow$  Anz. Spalten der Rückgabe = 0**, es wird also die leere Matrix zurückgegeben
    - ii) **Unendlich viele Lösungen  $\Rightarrow$  Anz. Spalten der Rückgabe = 2**, nur 1 Zeile, Elemente alle Null
- 2) Anzahl Spalten > Anzahl Zeilen + 1  $\Leftrightarrow$  Anzahl Gleichungen > Anzahl Unbekannte  
LGS ist **überbestimmt  $\Rightarrow$  Anz. Spalten der Rückgabe = 3**, nur 1 Zeile, Elemente alle Null
- 3) Anzahl Spalten < Anzahl Zeilen + 1  $\Leftrightarrow$  Anzahl Gleichungen < Anzahl Unbekannte  
LGS ist **unterbestimmt  $\Rightarrow$  Anz. Spalten der Rückgabe = 4**, nur 1 Zeile, Elemente alle Null

Auf diese Weise ist an dieser Anzahl Spalten der Rückgabe-Matrix der Methode `solve()` erkennbar, welcher Fall eingetreten ist, und wir können die `interpret(...)`-Methode von `LinEqSysCalculator` mit einer geeigneten Ausgabe dazu erweitern.



### 5.4.2 Aufgabe

Speichern Sie `LinEqSysCalculator_3_2` unter `LinEqSysCalculator_3_3` neu ab. Fügen Sie nun der Klasse `Matrix` in `LinEqSysCalculator_3_3`, wie unten gezeigt, die Methode `public Matrix solve()` nach dem Code der Methode `replaceFromPos(...)` hinzu. Ergänzen Sie den Code in den fehlenden, umrahmten Teilen selbst. Lesen Sie dabei den Code aufmerksam durch und versuchen Sie, alles zu verstehen.

```
// - Interpretiert die Matrix als Darstellung eines lin. Gl.systems.
// - Die letzte Spalte wird als die Konstantenkolonne nach dem Gleichheitszeichen
// - betrachtet und alle anderen Spalten ergeben die Koeffizientenmatrix.
// - Falls die Koeffizientenmatrix quadratisch ist, so wird, falls das System
// - eindeutig lösbar ist, die Lösung nach der Determinantenmethode ermittelt
// - und als 1-spaltige Matrix zurückgegeben. Falls das System nicht lösbar ist,
// - so wird die leere Matrix zurückgegeben und, falls es unendlich viele Lösungen
// - hat, eine 2-spaltige Matrix mit lauter Nullen als Elemente.
// - Ist die Koeffizientenmatrix nicht quadratisch, so ist das System entweder
// - überbestimmt, dann wird eine 3-spaltige Matrix mit lauter Nullen zurückgegeben,
// - oder unterbestimmt und dann wird eine 4-spaltige Matrix mit lauter Nullen zurückgegeben. D.h.
// - Rückgabe-Matrix 1-spaltig <=> Koeff.matrix quadrat. und Sys. eindeutig lösbar, Lsgen in den Zeilen
// - Rückgabe-Matrix leer <=> Koeff.matrix quadratisch und System unlösbar
// - Rückgabe-Matrix 2-spaltig <=> Koeff.matrix quadratisch und System mit unendl. vielen Lösungen
// - Rückgabe-Matrix 3-spaltig <=> Koeff.matrix nicht quadratisch, System überbestimmt
// - Rückgabe-Matrix 4-spaltig <=> Koeff.matrix nicht quadratisch, System unterbestimmt

public Matrix solve() {
    Matrix ret=new Matrix();
    int amtRows=getAmtRows();
    int amtClmns=getAmtClmns();
    Matrix constClmn=getClmn(amtClmns-1);
    Matrix coeffMatr=extract(0,0,amtRows,amtClmns-1);
    if (coeffMatr.isSquare()) { // - Anz. Glgen = Anz. Unbekannte
        double nennerDet=coeffMatr.det();
        if (nennerDet!=0.0) { // - Normalfall
            ret.init(amtRows,1);
            for (int i=0; i<amtRows; ++i) {
                ret.elmt[i][0]=coeffMatr.replaceFromPos(0,i,constClmn).det()/[ ];
            } // end of for
        } else { // - Sonderfall
            boolean hasSolutions=true;
            for (int i=0; i<amtRows && hasSolutions; ++i) {
                hasSolutions=coeffMatr.replaceFromPos(0,i,constClmn).det()==0;
            } // end of for
            if (hasSolutions) { // - Unendliche Lösungsmenge
                ret.init(1,2);
            } // end of if
        } // end of if-else
    } else { // - Anz. Glgen != Anz. Unbekannte
        if (amtClmns<amtRows+1) { // - Gl.system überbestimmt
            ret.init(1,3);
        } else { // - Gl.system unterbestimmt
            ret.init(1,4);
        } // end of if-else
    } // end of if-else
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_3` ab.

Fügen Sie nun der Klasse `LinEqSysCalculator_3_3`, wie unten gezeigt, das neue markierte Kommando hinzu.

```
public class LinEqSysCalculator_3_3 {
    // -- KONSTANTEN -- //
    // - Programmname
    public static final String PROGNAME="LinEqSysCalculator";

    // - Kommandos
    public static final String CMD_QUIT="quit";
    . . .
    public static final String CMD_SOLVE="solve";
    ...
}
```

Ergänzen Sie nun, wie im unten markierten Teil gezeigt, die `interpret`-Methode so, dass das neue Kommando geeignet interpretiert wird. Beachten Sie dabei, wie die Ausgabe je nach Fall, der auftreten kann, gestaltet ist. Ergänzen Sie auch die Methoden `public static String getCmdInfo()` und `public static String[] getCMDHelp()`, wie im markierten Teil gezeigt.

```
// - Interpretiert das Kommando cmd
public static void interpret(String cmd) {
    if (cmd.equals(CMD_QUIT)) {
        . . .
    } else if (cmd.equals(CMD_SOLVE)) {
        Matrix solution=actualMatrix.solve();
        int solAmtClmns=solution.getAmtClmns();
        if (solAmtClmns==1) {
            output("Die Lösung des GL.systems der akt. Matrix ist: ");
            display(solution,OUT_PROMPT);
        } else if (solAmtClmns==0) {
            output("Das GL.systems der akt. Matrix ist unlösbar! ");
        } else if (solAmtClmns==2) {
            output("Das GL.systems der akt. Matrix hat unendlich viele Lösungen! ");
        } else if (solAmtClmns==3) {
            output("Das GL.systems der akt. Matrix ist überbestimmt! ");
        } else if (solAmtClmns==4) {
            output("Das GL.systems der akt. Matrix ist unterbestimmt! ");
        }
    } else {
        output("Kommando '"+cmd+"' existiert nicht!");
    } // end of if-else
}

...
// - Gibt die Liste der Kommandos zurück.
public static String getCmdInfo() {
    String ret="Kommandos: "+CMD_QUIT+", "+CMD_CMDHELP+", "+CMD_CMDINFO+", "+CMD_EXAMPLE+", "+CMD_INPUT+",
"+CMD_OUT+", "+CMD_SAVE+", "+CMD_LOAD +", "+CMD_SOLVE;
    return ret;
}

// - Gibt einen Hilfetext zu den Kommandos zurück.
public static String[] getCMDHelp() {
    String[] ret=new String[10];
    ret[0]=CMD_QUIT+":      "+PROGNAME+" wird beendet. ";
    ...
    ret[8]=CMD_DET+":      Die Determinante der akt. Matrix wird, falls möglich, berechnet. ";
    ret[9]=CMD_SOLVE+":    Löst, falls eindeutig lösbar, das GL.system der akt. Matrix. ";
    return ret;
}
```

Speichern Sie die Datei wieder unter `LinEqSysCalculator_3_3` ab.

Kompilieren Sie nun `LinEqSysCalculator_3_3` und lassen Sie das Programm laufen. Geben Sie

das Kommando 'exmpl' ein und danach 'solve'. Die Lösung solle  $\begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{pmatrix}$  sein und die Ausgabe ungefähr so aussehen:

```
LinEqSysCalculator
Kommandos: quit, cmdhelp, cmd, exmpl, input, out, save, load, det, solve
>> exmpl
:: Beispielsmatrix:
[ 2.0 2.0 -1.0 1.0 7.0 ]
[ -3.0 5.0 4.0 -5.0 -1.0 ]
[ 7.0 -6.0 3.0 -2.0 -4.0 ]
[ -10.0 3.0 5.0 1.0 15.0 ]
>> solve
:: Die Lösung des GL.systems der akt. Matrix ist:
[ 1.0 ]
[ 2.0 ]
[ 3.0 ]
[ 4.0 ]
```

Testen Sie das 'solve'-Kommando selbst durch Eingabe von verschiedenen LGS. Versuchen Sie auch solche einzugeben, die keine Lösung oder unendlich viele haben, oder solche die unter- oder überbestimmt sind. Speichern Sie interessante LGS-Matrizen durch Eingabe des 'save'-Kommandos jeweils im Ihrem Unterordner «MatrixData» unter einem geeigneten Dateinamen ab, z.B. 'lgs\_4x5\_1' oder ähnliches.

Verlassen Sie schliesslich LinEqSysCalculator durch die Eingabe von 'quit'.

Das grundlegende Ziel des Projektes «LinEqSysCalculator», also die Erstellung eines Java-Programms zur Lösung von «beliebig» grossen LGS, das als Lösungsmethode die Determinantenmethode verwendet, ist also mit unserem Kommando-Interpreter LinEqSysCalculator gelungen. Alle dazu erforderlichen Eingaben und Ausgaben erfolgen, wie verlangt, in der Standard-Konsole.

Das Ziel ist also erreicht und das Projekt damit abgeschlossen.

## 6 Ende des Projekts

Das Projekt ist fertig. 😊

## 7 Lösungen

### 7.1 Zu Kap. 3: Start

#### 3.1.3 Aufgabe

LinEqSysCalculator\_1\_1:

```
public String toString() {
    String ret="[";
    for (int i=0; i<elmt.length; ++i) {
        for (int j=0; j<elmt[0].length; ++j) {
            ret=ret+elmt[i][j];
            if (j<elmt[0].length-1) {
                ret=ret+";";
            } // end of if
        } // end of for
        if (i<elmt.length-1) {
            ret=ret+"|";
        }
    } // end of for
    return ret+"]";
}
```

#### 3.1.6 Aufgabe

LinEqSysCalculator\_1\_3:

```
public int[] getMaxElmtStringLengthProClmn() {
    int[] ret=new int[getAmtClmns()];
    int actlg=0;
    for (int j=0; j<getAmtClmns(); ++j) {
        ret[j]=0;
        for (int i=0; i<getAmtRows(); ++i) {
            actlg=(""+elmt[i][j]).length();
            if (ret[j]<actlg) {
                ret[j]=actlg;
            } // end of if
        } // end of for
    } // end of for
    return ret;
}
```

## 3.1.7 Aufgabe

LinEqSysCalculator\_1\_3:

```

public String[] toStrings() {
    String[] ret=new String[getAmtRows()];
    int[] maxLgs=getMaxElmtStringLengthProClmn();
    for (int i=0; i<ret.length; ++i) {
        ret[i]="";
        for (int j=0; j<getAmtClmns(); ++j) {
            ret[i]=ret[i]+completeToLength(""+elmt[i][j], maxLgs[j]);
            if (j<getAmtClmns()-1) {
                ret[i]=ret[i]+" ";
            } // end of if
        } // end of for
    } // end of for
    return ret;
}

public static void test() {
    Matrix matr1=getExample1();
    Kon.writeln(matr1.toString());
    Kon.writeln();
    display(matr1);
}

```

## 3.1.8 Aufgabe

LinEqSysCalculator\_1\_4:

```

public static Matrix inputMatrix(boolean rowWise) {
    Matrix ret=new Matrix();
    int rows=0, clmns=0;
    do {
        Kon.write("Anzahl Zeilen ( > 0!): ");
        rows=Kon.readInt(true);
    } while (rows<=0); // end of do-while
    do {
        Kon.write("Anzahl Spalten ( > 0!): ");
        clmns=Kon.readInt(true);
    } while (clmns<=0); // end of do-while
    ret.init(rows, clmns);
    if (rowWise) {
        Kon.writeln("Matrixelemente zeilenweise eingeben (jede Eingabe mit [Enter] bestätigen): ");
        for (int i=0; i<ret.getAmtRows(); ++i) {
            Kon.writeln("Zeile "+i+": ");
            for (int j=0; j<ret.getAmtClmns(); ++j) {
                Kon.write(" a["+i+"]["+j+"] = ");
                ret.elmt[i][j]=Kon.readDouble(true);
            } // end of for
        } // end of for
    } else {
        Kon.writeln("Matrixelemente spaltenweise eingeben (jede Eingabe mit [Enter] bestätigen): ");
        for (int j=0; j<ret.getAmtClmns(); ++j) {
            Kon.writeln("Spalte "+j+": ");
            for (int i=0; i<ret.getAmtRows(); ++i) {
                Kon.write(" a["+i+"]["+j+"] = ");
                ret.elmt[i][j]=Kon.readDouble(true);
            } // end of for
        } // end of for
    } // end of if-else
    return ret;
}

```

## 7.2 Zu Kap. 4: Einfacher Kommando-Interpreter

## 4.3.1 Aufgabe

LinEqSysCalculator\_2\_0:

```

public static Matrix inputMatrix(boolean rowWise, String prompt) {
    Matrix ret=new Matrix();
    int rows=0, clmns=0;
    do {
        Kon.write(prompt+"Anzahl Zeilen ( > 0!): ");
        rows=Kon.readInt(true);
    } while (rows<=0); // end of do-while
    do {
        Kon.write(prompt+"Anzahl Spalten ( > 0!): ");
        clmns=Kon.readInt(true);
    } while (clmns<=0); // end of do-while
    ret.init(rows,clmns);
    if (rowWise) {
        Kon.writeln(prompt+"Matrixelemente zeilenweise eingeben (jede Eingabe mit [Enter] bestätigen): ");
        for (int i=0; i<ret.getAmtRows(); ++i) {
            Kon.writeln(prompt+"Zeile "+i+": ");
            for (int j=0; j<ret.getAmtClmns(); ++j) {
                Kon.write(prompt+" a["+i+"]["+j+"] = ");
                ret.elmt[i][j]=Kon.readDouble(true);
            } // end of for
        } // end of for
    } else {
        Kon.writeln(prompt+"Matrixelemente spaltenweise eingeben (jede Eingabe mit [Enter] bestätigen): ");
        for (int j=0; j<ret.getAmtClmns(); ++j) {
            Kon.writeln(prompt+"Spalte "+j+": ");
            for (int i=0; i<ret.getAmtRows(); ++i) {
                Kon.write(prompt+" a["+i+"]["+j+"] = ");
                ret.elmt[i][j]=Kon.readDouble(true);
            } // end of for
        } // end of for
    } // end of if-else
    return ret;
}

```

## 7.3 Zu Kap. 5: Lösen von LGS mit LinEqSysCalculator

## 5.1.1 Aufgabe

LinEqSysCalculator\_3\_0:

```

public double det() {
    double ret=Double.NaN;
    if (isSquare()) {
        int dim=getAmtRows();
        if (dim>0) {
            if (dim==1) {
                ret=elmt[0][0];
            } else if (dim==2) {
                ret=elmt[0][0]*elmt[1][1]-elmt[1][0]*elmt[0][1];
            } else if (dim==3) {
                ret=elmt[0][0]*elmt[1][1]*elmt[2][2]+elmt[0][1]*elmt[1][2]*elmt[2][0]
                    +elmt[0][2]*elmt[1][0]*elmt[2][1]-elmt[2][0]*elmt[1][1]*elmt[0][2]
                    -elmt[2][1]*elmt[1][2]*elmt[0][0]-elmt[2][2]*elmt[1][0]*elmt[0][1];
            } else {
                // - Code für dim>=4
            } // end of if-else
        } // end of if
    } // end of if
    return ret;
}

```

## 5.3.3 Aufgabe

```

LinEqSysCalculator_3_1:
public double det() {
    double ret=Double.NaN;
    if (isSquare()) {
        int dim=getAmtRows();
        if (dim>0) {
            . . .
        } else { // - Rekursiver Aufruf; "Entwicklung nach der nullten Zeile"
            int sign1=1;
            ret=0;
            for (int j=0; j<getAmtClmns(); ++j) {
                ret=ret+sign1*elmt[0][j]*reduced(0, j).det();
                sign1=-sign1;
            } // end of for
        } // end of if-else
    } // end of if
} // end of if
return ret;
}

public Matrix reduced(int rowInd, int clmnInd) {
    Matrix ret=new Matrix();
    . . .
    int n=0, k=0;
    for (int i=0; i< getAmtRows(); ++i) {
        if (i!=rowInd) {
            k=0;
            for (int j=0; j< getAmtClmns(); ++j) {
                if (j!=clmnInd) {
                    ret.elmt[n][k]=elmt[i][j];
                    ++k;
                } // end of if
            } // end of for
            ++n;
        } // end of if
    } // end of for
    return ret;
}

```

## 5.4.1 Aufgabe

```

LinEqSysCalculator_3_2:
public Matrix extract(int startRowInd, int startClmnInd, int endRowInd, int endClmnInd) {
    Matrix ret=new Matrix();
    if (startRowInd<=endRowInd && startClmnInd<=endClmnInd) {
        . . .
        ret.init(endRowInd-startRowInd, endClmnInd-startClmnInd);
        int retAmtRows=ret.getAmtRows();
        int retAmtClmns=ret.getAmtClmns();
        for (int i=0; i<retAmtRows; ++i) {
            for (int j=0; j<retAmtClmns; ++j) {
                ret.elmt[i][j]=elmt[startRowInd+i][startClmnInd+j];
            } // end of for
        } // end of for
    } // end of if
    return ret;
}

public Matrix getRow(int ind) {
    return extract(ind, 0, ind+1, getAmtClmns());
}

```



```

public Matrix replaceFromPos(int startRowInd, int startClmnInd, Matrix source) {
    Matrix ret=copy();
    if (intersects(startRowInd,startClmnInd,source)) { // - Wird nur ausgeführt, falls es eine
        . . .
        for (int i=0; i<amtRowsSource; ++i) {
            for (int j=0; j<amtClmnsSource; ++j) {
                if (0<=startRowInd+i && startRowInd+i<amtRows && 0<=startClmnInd+j &&
startClmnInd+j<amtClmns) {
                    ret.elmt[startRowInd+i][startClmnInd+j]=source.elmt[i][j];
                } // end of if
            } // end of for
        } // end of for
    } // end of if
    return ret;
}

```

### 5.4.2 Aufgabe

```

LinEqSysCalculator_3_3:
public Matrix solve() {
    . . .
    if (coeffMatr.isSquare()) { // - Anz. Glgen = Anz. Unbekannte
        double nennerDet=coeffMatr.det();
        if (nennerDet!=0.0) { // - Normalfall
            ret.init(amtRows,1);
            for (int i=0; i<amtRows; ++i) {
                ret.elmt[i][0]=coeffMatr.replaceFromPos(0,i,constClmn).det()/nennerDet;
            } // end of for
        } else { // - Sonderfall
            . . .
        } // end of if-else
    } else { // - Anz. Glgen != Anz. Unbekannte
        . . .
    } // end of if-else
    return ret;
}

```