

Dokumentation 36. BWINF

Lars Müller & Kai Gerd Müller, Team : iub, Team-ID : 00782, Gymnasium : Clara-Schumann-Gymnasium Bonn

Vorwort :

In jedem „Aufgaben-Ordner“ finden sich zwei Ordner vor. Der eine heißt „Highlighted“. Dies sind die „gehighlighteten“ Quellcodes als .rtf. Der andere Ordner ist ein NetBeans-Projektordner. Die Lösungen liegen als NetBeans-Projekte vor. Diese sind als .zip komprimiert. Die kompilierte .jar Datei ist in Projekt/dist vorzufinden. Der Quellcode ist unter Projekt/src/projektname/ einsehbar. Die Testdateien sind unter Projekt/res/ Ich empfehle, NetBeans zu verwenden. Als Betriebssystem wurde Ubuntu 16.04 verwendet, aber da ich Java benutzt habe, sollte dies nicht wichtig sein.

△ Ich halte es für empfehlenswert, den kommentierten Quellcode zu lesen △

1. Junioraufgabe „Bücherregal“

Lösungsidee :

Zuerst ermitteln wir, wie viele Abschnitte es geben muss. Es muss mindestens so viele Abschnitte wie Dekofiguren geben, da sie ja alle Dekofiguren benutzen will.

Nun erstellen wir zuerst „Zwangsabschnitte“, das sind Abschnitte, die erstellt werden müssen, da sonst nicht die Regel eingehalten werden könnte, dass in jedem Abschnitt maximal $3\text{ cm} \triangleq 30\text{ mm}$ Höhenunterschied der Bücher sein dürfen.

Dies machen wir folgendermaßen :

- Wir sortieren die Bücher mithilfe von **Selectionsort**
(Anmerkung : Wir könnten hier auch den built-in Sortieralgorithmus von Java, einen Quicksort, benutzen, aber da ich diesen als einen wichtigen Teil der Aufgabe angesehen habe, habe ich selbst einen Selectionsort implementiert)
- Wir erstellen Abschnitte folgendermaßen :
 - Wir iterieren die Liste der sortierte Bücher durch
 - Man fängt mit dem ersten Buch an. Dieses ist gleichzeitig das kleinste Buch.

- Man erstellt nun einen Abschnitt für dieses Buch
- Alle Bücher, die daraufhin durchiteriert werden, werden darauf geprüft, ob sie einen Höhenunterschied von mehr als 30 mm aufweisen zum kleinsten Buch dieses Abschnittes.
- Sollte dies der Fall sein, muss ein neuer Abschnitt erstellt werden. Das kleinste Buch dieses Abschnittes ist das jetzt gefundene Buch, das nicht in den vorherigen Abschnitt passt.
- Nun wiederholt man das beschriebene Verfahren

Nun kann es jedoch vorkommen, dass wir weniger Zwangsabschnitte als Dekofiguren haben.

Dann müssen wir neue Abschnitte erstellen. Dies machen wir, indem wir immer ein Buch solange aus jedem Abschnitt, der mindestens zwei Bücher hat, herausholen, und in einen neuen Abschnitt tun, bis wir genug Abschnitte haben.

Haben wir irgendwann nur noch 1-er Abschnitte, aber immer noch nicht genug, ist die Aufgabe nicht lösbar

Umsetzung :

Bibliotheken :

- java.io.File : Dateien
- java.io.FileNotFoundException : Datei-nicht-gefunden-Fehler
- java.io.FileReader : Liest Textdateien ein
- java.io.IOException : Eingabe-Ausgabe-Fehler
- java.util.ArrayList : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können
- java.util.Scanner : Benutzereingabe, kann direkt aus der Befehlszeile lesen

Datenstrukturen :

Abschnitt : Speichert einen Abschnitt, ist also eigentlich eine Liste aus Bücherhöhen

Programmstruktur :

Wir implementieren zunächst die folgenden Funktionen : leseDatei und dateiFrage.

dateiFrage fragt den Benutzer nach dem Pfad zu einer Datei. Dies tut es so lange, bis der Benutzer einen validen Pfad angegeben hat.

leseDatei liest diese Datei ein. Dies ist die Aufgabe. Wir separieren sie nach Zeilenumbrüchen. Die erste Zeile sind die Dekofiguren.

Wir implementieren eine Funktion „sort“. Diese repräsentiert einen Selectionsort, und nimmt zwei Parameter : Sortierte Liste bis jetzt, und noch nicht sortierter Rest der Liste. Aus dem nicht sortierten Rest der Liste sucht diese Funktion dann das kleinste Element raus, hängt es an die sortierte Liste an, und entfernt es vom nicht sortierten Rest der Liste. Dann ruft sich die Funktion mit den neuen Listen wieder selbst auf, solange, bis der unsortierte „Rest“ leer ist. Mit einer Schleife erstellen wir nun die Zwangsabschnitte. Sind es dann noch nicht genug, also weniger Abschnitte als Figuren, tritt erneut eine Schleife ein, welche von den Abschnitten solange Bücher abschneidet, bis wir genug Abschnitte haben. Haben wir irgendwann nur noch 1er Abschnitte, und sind es immer noch nicht genug, ist die Aufgabe unlösbar. Ansonsten geben wir die Abschnitte in übersichtlicher Darstellung mit der `<Abschnitt>.toString()` Methode aus.

Beispiele :

/res/buecherregal1.txt	/res/buecherregal2.txt	/res/buecherregal3.txt	/res/buecherregal4.txt	/res/buecherregal5.txt	/res/buecherregal6.txt
Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/buecherregal1.txt	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/buecherregal2.txt	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/buecherregal3.txt	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/buecherregal4.txt	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/buecherregal5.txt	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/buecherregal6.txt
Abschnitt : 1 mit 2 Büchern 168 170 ----- Abschnitt : 2 mit 3 Büchern 202 211 229 ----- Abschnitt : 3 mit 3 Büchern 233 254 260 ----- Abschnitt : 4 mit 1 Buch 272 ----- Abschnitt : 5 mit 1 Buch 306	Abschnitt : 1 mit 2 Büchern 169 175 ----- Abschnitt : 2 mit 4 Büchern 203, 209 210, 229 ----- Abschnitt : 3 mit 3 Büchern 233 254 260 ----- Abschnitt : 4 mit 1 Buch 272 ----- Abschnitt : 5 mit 1 Buch 306	Abschnitt : 1 mit 2 Büchern 170 174 ----- Abschnitt : 2 mit 2 Büchern 202 229 ----- Abschnitt : 3 mit 1 Buch 235 ----- Abschnitt : 4 mit 1 Buch 279	Abschnitt : 1 mit 31 Büchern 160, 160, 161, 161, 162 165, 165, 166, 167, 167 167, 169, 170, 170, 171 173, 173, 174, 174, 177 180, 182, 183, 184, 184 185, 185, 187, 188, 189 190 ----- Abschnitt : 2 mit 21 Büchern 196, 197, 197, 199 200, 201, 202, 206 207, 207, 211, 212 212, 214, 215, 216 217, 218, 219, 224 225 ----- Abschnitt : 3 mit 20 Büchern 233, 235, 237, 238 238, 239, 240, 240 240, 245, 246, 246 247, 253, 254, 256 258, 259, 259, 261 ----- Abschnitt : 4 mit 22 Büchern 264, 266, 266, 267 268, 270, 270, 272 274, 275, 276, 277 278, 279, 286, 286 287, 288, 289, 290	Abschnitt : 1 mit 50 Büchern 160, 161, 161, 161, 162, 162, 162 163, 163, 164, 164, 164, 164, 164 165, 165, 165, 166, 167, 167, 168 168, 168, 168, 169, 169, 170, 170 171, 171, 171, 171, 172, 174, 174 174, 174, 175, 175, 176, 176, 176 176, 176, 177, 177, 178, 179, 180 180 ----- Abschnitt : 2 mit 22 Büchern 201, 202, 202, 202 203, 206, 206, 208 209, 210, 211, 212 216, 220, 220, 221 221, 229, 230, 230 230, 231 ----- Abschnitt : 3 mit 23 Büchern 232, 232, 233, 233 235, 237, 238, 240 241, 241, 241, 243 243, 246, 248, 248 250, 254, 257, 258 260, 261, 261 ----- Abschnitt : 4 mit 4 Büchern 263, 264 265, 265	Abschnitt : 1 mit 1 Buch 125 ----- Abschnitt : 2 mit 24 Büchern 160, 161, 162, 164 166, 166, 167, 168 169, 170, 171, 172 174, 175, 175, 175 175, 175, 177, 177 177, 178, 179, 180 ----- Abschnitt : 3 mit 17 Büchern 200, 200, 203, 204 207, 208, 208, 209 211, 211, 214, 214 218, 220, 221, 226 228 ----- Abschnitt : 4 mit 7 Büchern 247, 255 261, 262 263, 264 264 ----- Abschnitt : 5 mit 48 Büchern 281, 282, 282, 283, 283, 283 285, 286, 287, 287, 290, 290 291, 292, 292, 293, 293, 294 295, 295, 296, 298, 298, 300 304, 304, 304, 305, 305, 305

			293, 293 ----- Abschnitt : 5 mit 5 Büchern 295, 296 300, 301 303		305, 306, 306, 306, 306, 307 307, 308, 308, 308, 309, 309 309, 309, 310, 310, 310, 310 ----- Abschnitt : 6 mit 2 Büchern 340 341
--	--	--	---	--	--

Quellcode :

Buecherregal.java :

```

package buecherregal;

//Bibliotheken, um Dateien zu lesen
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
//Listen
import java.util.ArrayList;
//Benutzer-Eingabe
import java.util.Scanner;

/**
 *
 * @author lars
 */
public class Buecherregal {

    public static final Scanner EINGABE = new Scanner(System.in);

    public static String leseDatei(File pfad_zur_datei) throws FileNotFoundException, IOException { //Liest eine Datei ein, und gibt Text zurück
        FileReader datei = new FileReader(pfad_zur_datei);
        String r = "";
        int i = datei.read();
        while (i != -1) {
            r += (char) i;
            i = datei.read();
        }
        return r;
    }

    public static File dateiFrage(String frage) { //Fragt nach einem Pfad und prüft, ob dieser existiert
        while (true) {

```

```

        System.out.println(frage + "(Pfad zu .txt Datei) ? ");
        String s = EINGABE.nextLine();
        File f = new File(s);
        if (f.exists() && !f.isDirectory() && f.canRead()) { //Prüfe, ob : - existiert die Datei - ist es kein Ordner - ist sie lesbar
            return f; //Gebe Pfad zurück
        }
        System.out.println("Bitte antworten sie mit einem vorhandenen Pfad einer .txt Datei. Versuchen sie es erneut.");
    }
}

```

```

public static ArrayList<Integer> sort(ArrayList<Integer> liste, ArrayList<Integer> sortiert) { //Sortiert eine Liste
    //Das kleinste Element und dessen Position(Index) bestimmen
    int kleinster_wert_index = 0;
    int kleinster_wert = liste.get(0);
    for (int i = 1; i < liste.size(); i++) {
        int wert = liste.get(i);
        if (wert < kleinster_wert) {
            kleinster_wert_index = i;
            kleinster_wert = wert;
        }
    }
    if (liste.size() > 1) { //Falls es noch zu sortierende Elemente gibt
        liste.remove(kleinster_wert_index); //Den ermittelten Wert von der Liste entfernen
        sortiert.add(kleinster_wert); //Zur sortierten Liste hinzufügen
        return sort(liste, sortiert); //Rufe Funktion noch einmal auf, diesmal wurde der jetzt ermittelte kleinste Wert aber entfernt
    }
    return sortiert; //Wenn das letzte Element angehängt wurde, kann die sortierte Liste zurückgegeben werden
}

```

```

/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws IOException {
    String aufgabe = leseDatei(dateiFrage("Wo befindet sich die Aufgabe")); //Inhalt der Aufgabendatei
    String[] zeilen = aufgabe.split("\n"); //Sämtliche Zahlen in der Aufgabedatei, welche ja untereinander geschrieben waren(deswegen trennen wir nach Zeilenumbrüchen)
    ArrayList<Integer> hoehen_liste = new ArrayList(); //Liste, welche die Höhen aller Bücher speichert
    int figuren = Integer.parseInt(zeilen[0]); //Anzahl der Dekofiguren
    for (int i = 2; i < zeilen.length; i++) { //Mit Zeile 3 fangen die Höhen der Bücher an
        hoehen_liste.add(Integer.parseInt(zeilen[i]));
    }
    hoehen_liste = sort(hoehen_liste, new ArrayList()); //Wir sortieren die Bücher der Höhe nach
    //Anmerkung : Es könnte auch eine Die Built-In Sortierfunktion von Java(Arrays.sort(<hoehen>)) auf ein Array angewendet werden
    //Da ich jedoch den Sortieralgorithmus für einen wichtigen Teil der Aufgabe halte, habe ich diesen selbst implementiert
    byte a = 0; //Aktueller Abschnitt a
    int min = hoehen_liste.get(0); //Kleinstes Buch des aktuellen Abschnittes
    ArrayList<Abschnitt> abschnitte = new ArrayList();
    abschnitte.add(new Abschnitt());
    abschnitte.get(0).add(hoehen_liste.get(0));
    for (int i = 1; i < hoehen_liste.size(); i++) {
        int wert = hoehen_liste.get(i);
        if (wert - min > 30) { //Wenn das jetztige Buch mehr als 30 mm höher ist als das kleinste dieses Abschnittes, so muss zwangsweise ein neuer Abschnitt beginnen

```



```

public static byte ZAEHLER = 1;

public Abschnitt() {
    super();
}

public static void zaehlerZuruecksetzen() {
    ZAEHLER = 1;
}

public static String toString(ArrayList<Abschnitt> abschnitte) {
    String s = "";
    for (Abschnitt abschnitt : abschnitte) {
        s += abschnitt.toString();
        s += "\n";
    }
    return s;
}

@Override
public String toString() { //Optimierte Darstellung/Ausgabe
    String b = "Abschnitt : " + Byte.toString(ZAEHLER) + " mit " + Integer.toString(size()) + " ";
    if (size() != 1) {
        b += "Büchern";
    } else {
        b += "Buch";
    }
    String s = "";
    if (ZAEHLER != 1) {
        for (int i = 0; i < b.length(); i++) {
            s += "-";
        }
    }
    s += "\n" + b + "\n";
    b = null;
    byte index = 0;
    byte counter = 0;
    byte zeilenumbruch = (byte) (Math.sqrt(size()));
    for (Integer e : this) {
        s += Integer.toString(e);
        counter++;
        index++;
        if (index != size()) {
            if (counter == zeilenumbruch) {
                s += "\n";
                counter = 0;
            } else if (index != size()) {
                s += ", ";
            }
        }
    }
}

```

```
    ZAEHLER++;  
    return s;  
}  
}
```

2. Junioraufgabe „Wintervorrat“

Lösungsidee :

Zuerst ermitteln wir alle total sicheren Felder, indem wir erstmal davon ausgehen, dass alle Felder total sicher seien, und streichen dann alle Felder, die auf der „Fluglinie“ eines Adlers liegen.

Wir simulieren den Rechteckwald, und wie die Vögel ihn überfliegen. Jedes Feld speichert, wann es das letzte Mal von einem Adler überflogen wurde. Fliegt nun erneut ein Adler darüber, obwohl die letzte „Adler-Sichtung“ weniger als eine halbe Stunde her ist, ist der Zeitraum von letzter Überflug+1 Minute bis jetzt nicht als sicher einzustufen, sonst schon. Sollte ein Zeitraum als sicher eingestuft werden, wird er ausgegeben, und der Status des Feldes auf „sicher“ (1) gesetzt. Würde ein Adler über den Rand des Rechteckwaldes hinausfliegen, so kehrt er um. Nach durchsimulieren des Tages ermitteln wir dann noch alle Felder, deren letzte Adler-Sichtung eine halbe Stunde her ist, und fügen denen einen sicheren Zeitraum hinzu, da sie dann ja eine halbe Stunde lang sicher waren.

Umsetzung :

Bibliotheken :

- java.awt.Color : Farben
- java.awt.Font : Schriftarten
- java.awt.Graphics2D : Wird benutzt, um auf ein Bild zu zeichnen
- java.io.File : Dateien
- java.io.FileNotFoundException : Datei-nicht-gefunden-Fehler
- java.io.FileReader : Liest Textdateien ein
- java.io.FileWriter : Schreibt in Dateien
- java.io.BufferedWriter : Schreibt in Dateien
- java.io.IOException : Eingabe-Ausgabe-Fehler
- java.util.ArrayList : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können

- `java.util.Scanner` : Benutzereingabe, kann direkt aus der Befehlszeile lesen
- `java.awt.image.BufferedImage` : „Bild“-Klasse von Java. Wird benutzt, um Bilder zu erzeugen.
- `javax.imageio.ImageIO` : Kann `BufferedImages` speichern

Datenstrukturen :

Punkt : Speichert einen 2D-Punkt

Zeitraum : Speichert einen Startminute und Endminute eines Zeitraumes

Wald : Simuliert & speichert den kompletten Rechteckwald

Adler : Speichert einen Adler, also zwei Punkte : Dessen Position und Flugrichtung

Feld : Speichert ein Feld, also im Wesentlichen dessen sichere Zeiträume, und ob es total sicher ist.

Programmstruktur :

Wir implementieren zunächst die folgenden Funktionen : `leseDatei`, `dateiFrage`, `textFrage`, und `jaNeinFrage`.

`dateiFrage` fragt den Benutzer nach dem Pfad zu einer Datei. Dies tut es so lange, bis der Benutzer einen validen Pfad angegeben hat.

Mit `jaNeinFrage` erfahren wir vom Benutzer :

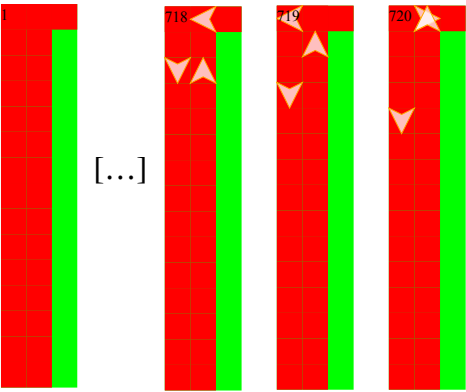
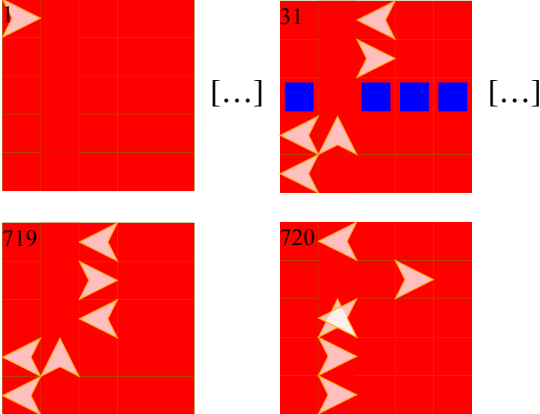
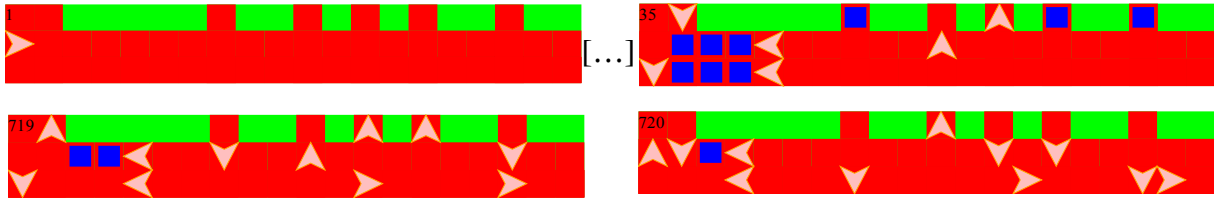
- Sollen Ausgabebilder generiert werden ?
- SVGs ?
- JPGs ?

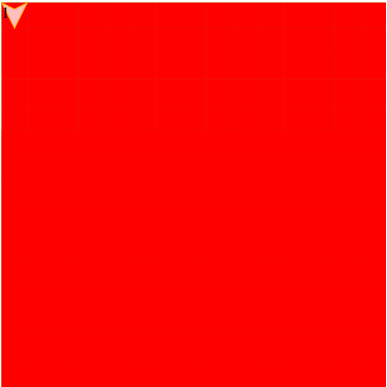
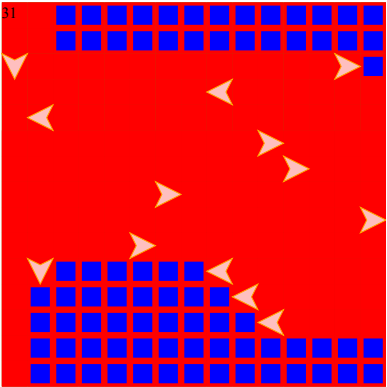
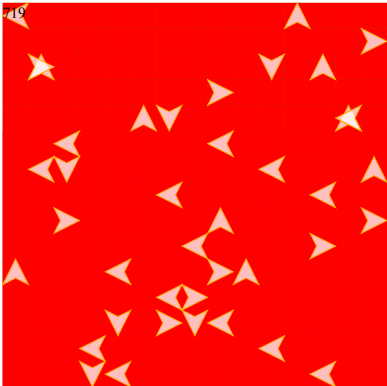
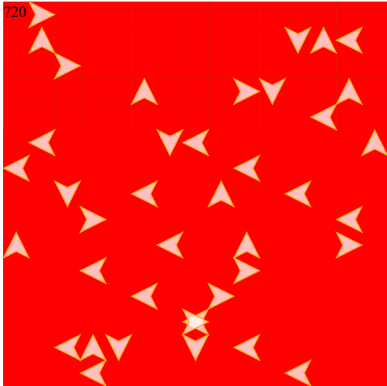
Und mit `textFrage` :

- Unter welchem Namen sollen diese gespeichert werden ?

`leseDatei` liest diese Datei ein. Dies ist die Aufgabe. Wir separieren sie nach Zeilenumbrüchen. Die erste Zeile sind die Dimensionen. Diese übergeben wir an den Rechteckwald-Konstruktor. Die zweite Zeile ist die Zahl der Adler. Diese ist nützlich, um ein Array zu initialisieren. Wir implementieren auch noch eine Funktion „`parseDir`“. Diese wandelt eine Himmelsrichtung in einen Punkt um. Somit erzeugen wir jetzt Adler aus den Zeilen und speichern diese im Adler-Array. Auch dieses wird dem Konstruktor des Waldes übergeben. Genauso wird dem Konstruktor der Speichername, und ob SVGs/JPGs generiert werden sollen, übergeben. Der Wald ist hauptsächlich ein zweidimensionales Array/Array-of-Arrays von Feldern. Nach durchsimulieren gibt er automatisch selbst aus temporär sichere Felder aus. Danach wird die Funktion `toString()` des Waldes aufgerufen, welche die Sicherheiten der Felder darstellt.

Beispiele :

<code>/res/wintervorrat1.txt</code>	<code>/test/wintervorrat1_loes.txt</code>	
<code>/res/wintervorrat2.txt</code>	<code>/test/wintervorrat2_loes.txt</code>	
<code>/res/wintervorrat3.txt</code>	<code>/test/wintervorrat3_loes.txt</code>	

/res/wintervorrat4.txt	/test/wintervorrat4_loes.txt	<div>  [...]  [...] </div> <div>   </div>
/res/wintervorrat5.txt	/test/wintervorrat5_loes.txt	[...]
/res/wintervorrat6.txt	/test/wintervorrat6_loes.txt	[...]

Quellcode :

Wald.java :

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package wintervorrat;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.imageio.ImageIO;

/**
 *
 * @author lars
 */
public class Wald { //Der Rechteckwald

    public static final short DAY = 60 * 12; //Ein Tag ist 12 Stunden(60 Minuten) lang
    public Adler[] adler; //Liste der Adler
    public Feld[][] felder; //Tabelle mit Feldern
    public String name; //Speichernname für Dateien
    public BufferedImage hintergrund; //Das Feld : grün - total sicher rot - nicht total sicher
    public String svg; //Als Vektorgrafik : Header und das Feld als Vektorgrafik : grün - total sicher rot - nicht total sicher
    public boolean svg_erstellen, jpg_erstellen; //Sollen Vektorgrafiken und Bilder als Ausgabe erzeugt werden ?
    public static final Punkt[] ADLER_SVG = new Punkt[]{new Punkt(0, 0), new Punkt(40, 20), new Punkt(0, 40), new Punkt(10, 20)}; //Ein Adler(Pfeil) als Punktmenge für die
    SVG-Darstellung

    public Wald(String name, int x, int y, Adler[] adler, boolean erstelle_svg, boolean erstelle_jpg) {
        this.svg_erstellen = erstelle_svg;
        this.jpg_erstellen = erstelle_jpg;
        if (erstelle_svg) { //Falls SVGs erzeugt werden sollen
            this.svg = "<?xml version='1.0' encoding='UTF-8' standalone='no' ?>\n" //SVG Header
                + "<!DOCTYPE svg PUBLIC '-//W3C//DTD SVG 20010904//EN'\n"
                + "\"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd\">\n"
                + "<svg xmlns='http://www.w3.org/2000/svg'\n"
                + "    xmlns:xlink='http://www.w3.org/1999/xlink'\n"
                + "    version='1.1' baseProfile='full'\n"
                + String.format("    width='%dpx' height='%dpx'\n", x * 40, y * 40) //Breite, Höhe
                + String.format("    viewBox='0 0 %d %d'\n", x * 40, y * 40); //Angezeigter Bereich
            svg += String.format("    <rect x='0' y='0' width='%d' height='%d' fill='rgb(0,255,0)'\n", x * 40, y * 40); //Hintergrund grün färben
        }
    }
}
```

```

Graphics2D hg = null;
if (erstelle_jpg) { //Falls JPGs erzeugt werden sollen
    hintergrund = new BufferedImage(x * 40, y * 40, BufferedImage.TYPE_INT_RGB); //Hintergrundbild erstellen
    hg = hintergrund.createGraphics();
    //Hintergrund grün füllen
    hg.setColor(Color.GREEN);
    hg.fillRect(0, 0, hintergrund.getWidth(), hintergrund.getHeight());
}
this.name = name; //Name, um die Dateien zu speichern
this.felder = new Feld[x][y]; //Speichert den alle Felder
for (int xw = 0; xw < x; xw++) {
    for (int yw = 0; yw < y; yw++) {
        felder[xw][yw] = new Feld(true); //Wir gehen erstmal davon aus, dass jedes Feld total sicher ist
    }
}
this.adler = adler; //Adler werden vorgegeben
for (Adler a : adler) { //Adler durchgehen, um nicht vollkommen sichere Felder zu ermitteln
    if (a.flugrichtung.x == 0) { //Sollte er in y-Richtung fliegen(Norden-Süden)
        for (int yw = 0; yw < y; yw++) { //So sind alle auf seiner Linie liegenden Felder...
            int xw = a.position.x;
            felder[xw][yw].totalSicher = false; //...nicht total sicher
            if (erstelle_jpg) { //Und werden dementsprechend rot markiert, falls erwünscht, hier im Bild...
                hg.setColor(Color.RED);
                hg.fillRect(xw * 40, yw * 40, 40, 40);
            }
            if (erstelle_svg) { //...und hier in der Vektorgrafik, falls erwünscht
                svg += String.format("    <rect x=\"%d\" y=\"%d\" width=\"40\" height=\"40\" fill=\"rgb(255,0,0)\" />\n", xw * 40, yw * 40);
            }
        }
    } else { //Sollte er in x-Richtung fliegen(Osten-Westen)
        for (int xw = 0; xw < x; xw++) { //So sind alle auf seiner Linie liegenden Felder...
            int yw = a.position.y;
            felder[xw][yw].totalSicher = false; //...nicht total sicher
            if (erstelle_jpg) { //Und werden dementsprechend rot markiert, falls erwünscht, hier im Bild...
                hg.setColor(Color.RED);
                hg.fillRect(xw * 40, yw * 40, 40, 40);
            }
            if (erstelle_svg) { //...und hier in der Vektorgrafik, falls erwünscht
                svg += String.format("    <rect x=\"%d\" y=\"%d\" width=\"40\" height=\"40\" fill=\"rgb(255,0,0)\" />\n", xw * 40, yw * 40);
            }
        }
    }
}
}
}

public void simulieren() throws IOException {
    for (int x = 0; x < felder.length; x++) {
        for (int y = 0; y < felder[0].length; y++) {
            felder[x][y].anfangSichererZeitraum = 1;
        }
    }
}

```

```

for (short m = 1; m <= DAY; m++) { //Alle Minuten durchsimulieren
    Graphics2D grafik = null;
    BufferedImage img = null;
    if (jpg_erstellen) { //Falls Erstellung von Bildern gewünscht ist, Kopie des Hintergrunds erstellen
        img = new BufferedImage(felder.length * 40, felder[0].length * 40, BufferedImage.TYPE_INT_RGB);
        grafik = img.createGraphics();
        grafik.drawImage(hintergrund, 0, 0, null);
    }
    String svg_to_save = "";
    if (svg_erstellen) { //Falls ein SVG gewünscht ist, Kopie des Headers + des Hintergrundes als Vektorgrafik erstellen
        svg_to_save = svg;
    }
    for (Adler a : adler) { //Alle Adler durchgehen
        if (m >= a.startminute) { //Davon sind nur die, die schon gestartet sind, relevant
            if (svg_erstellen || jpg_erstellen) { //Falls eine grafische Darstellung gefordert ist...
                int xw = a.position.x * 40 + 5;
                int yw = a.position.y * 40 + 5;
                if (jpg_erstellen) {
                    //Adler als weiße Kreise anzeigen, Radius 30
                    grafik.setColor(Color.WHITE);
                    grafik.fillOval(xw, yw, 30, 30);
                }
                if (svg_erstellen) {
                    //Adler als festgelegten Pfeil anzeigen
                    //Adler richtig drehen
                    String t = "rotate(";
                    if (a.flugrichtung.x != 0) {
                        if (a.flugrichtung.x < 0) {
                            t += "180";
                        } else {
                            t += "0";
                        }
                    } else if (a.flugrichtung.y < 0) {
                        t += "270";
                    } else {
                        t += "90";
                    }
                    t += " ";
                    t += Integer.toString(xw + 15); //Drehungsmittelpunkt X
                    t += " ";
                    t += Integer.toString(yw + 15); //Drehungsmittelpunkt Y
                    t += ")";
                    //Koordinaten des Pfeils versetzen
                    String ps = "";
                    for (int p = 0; p < ADLER_SVG.length; p++) {
                        ps += Integer.toString(xw - 5 + ADLER_SVG[p].x);
                        ps += ",";
                        ps += Integer.toString(yw - 5 + ADLER_SVG[p].y);
                        if (p != ADLER_SVG.length - 1) {
                            ps += " ";
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        svg_to_save += "    <polygon points=\"" + ps + "\" style=\"" + fill + "; fill-opacity:0.75; stroke:rgb(255,165,0); stroke-width:0.5px\" transform=\"" + t +
        "\"/>\n"; //Adler(Pfeil) versetzt zum SVG, mit gegebener Richtung(Drehung), hinzufügen, ebenfalls in weiß, nun aber auch teilweise durchsichtig
    }
}
if (a.flugrichtung.x != 0) { //Sollte der Adler in x-Richtung fliegen(Westen-Osten)
    if (a.position.x + a.flugrichtung.x < 0 || a.position.x + a.flugrichtung.x > felder.length - 1) { //Würde er über den Rechteckwald hinausfliegen...
        a.flugrichtung.x = -a.flugrichtung.x; //...fliegt er wieder zurück, ab jetzt also in umgekehrter Richtung
    }
} else if (a.position.y + a.flugrichtung.y < 0 || a.position.y + a.flugrichtung.y > felder[0].length - 1) { //Sollte der Adler in y-Richtung fliegen(Norden-Süden), und würde
er über den Rechteckwald hinausfliegen...
    a.flugrichtung.y = -a.flugrichtung.y; //...fliegt er wieder zurück, ab jetzt also in umgekehrter Richtung
}
Feld f = felder[a.position.x][a.position.y]; //Das Feld, über dem er sich jetzt befindet
if (m - f.anfangSichererZeitraum >= 30) { //Sollte dieses mindestens satte 30 Minuten sicher gewesen sein :
    felder[a.position.x][a.position.y].add(new Zeitraum(f.anfangSichererZeitraum, m)); //Zeitraum zu sicheren Zeiträumen des Feldes hinzufügen
}
felder[a.position.x][a.position.y].anfangSichererZeitraum = (short) (m + 1); //Nach Überflug ist das Feld wieder sicher
a.position = a.position.add(a.flugrichtung); //Weiterfliegen !
}
}
if (svg_erstellen || jpg_erstellen) { //Für eine eventuell gewünschte Darstellung :
    for (int x = 0; x < felder.length; x++) {
        for (int y = 0; y < felder[0].length; y++) {
            if (!felder[x][y].totalSicher) { //Ist das Feld nicht total sicher
                if (m - felder[x][y].anfangSichererZeitraum >= 30) { //Sichere Felder einfärben, solange sie einen sicheren Zeitraum durchlaufen
                    int xw = x * 40 + 5;
                    int yw = y * 40 + 5;
                    if (jpg_erstellen) {
                        //Im Bild als blaue Quadrate darstellen
                        grafik.setColor(Color.BLUE);
                        grafik.fillRect(xw, yw, 30, 30);
                    }
                    if (svg_erstellen) {
                        //Genauso im SVG
                        svg_to_save += String.format("    <rect x=\"%d\" y=\"%d\" width=\"30\" height=\"30\" fill=\"rgb(0,0,255)\" />\n", xw, yw);
                    }
                }
            }
        }
    }
}
}
//Simultaionsminute in die Darstellung/en einfügen und diese dann speichern
if (jpg_erstellen) {
    //Simulationsminute einfügen, schwarz, Schriftgröße 24
    grafik.setColor(Color.BLACK);
    grafik.setFont(new Font("sans", Font.PLAIN, 24));
    grafik.drawString(Short.toString(m), 0, 24);
    File ausgabe = new File(name + Short.toString(m) + ".jpg"); //Ausgabe : name+minute+".jpg"
    ImageIO.write(img, "JPG", ausgabe); //Bild als JPG speichern
}
}

```

```

if (svg_erstellen) {
    //Simulationsminute einfügen, schwarz, Schriftgröße 24
    svg_to_save += String.format("<text x=\"0\" y=\"24\" fill=\"black\" style=\"font-size:24px\">%s</text>", Short.toString(m));
    svg_to_save += "\n</svg>";
    File svg_ausgabe = new File(name + Short.toString(m) + ".svg");
    if (svg_ausgabe.exists()) { //Existiert die Datei schon
        svg_ausgabe.delete(); //So wollen wir diese komplett löschen
    }
    svg_ausgabe.createNewFile(); //Wir erzeugen die Ausgabedatei
    //SVG speichern
    BufferedWriter w = new BufferedWriter(new FileWriter(svg_ausgabe));
    w.write(svg_to_save);
    w.close();
}
System.gc(); //Speicher freigeben
}
for (int x = 0; x < felder.length; x++) {
    for (int y = 0; y < felder[0].length; y++) {
        if (!felder[x][y].totalSicher) { //Alle nicht total sicheren Felder
            if (DAY - felder[x][y].anfangSichererZeitraum >= 30) { //Alles muss raus"-Felder, die vor Ende des Tages nur einmal überflogen wurden, was jetzt schon mehr als eine
1/2 h her ist
                felder[x][y].add(new Zeitraum(felder[x][y].anfangSichererZeitraum, DAY)); //Zeitraum zu sicheren Zeiträumen des Feldes hinzufügen
            }
            //Alle sicheren Zeiträume des Feldes ausgeben
            if (!felder[x][y].isEmpty()) { //Gibt es bei diesem Feld sichere Zeiträume
                System.out.print("X=" + Integer.toString(x) + " Y=" + Integer.toString(y));
                for (Zeitraum z : felder[x][y]) {
                    System.out.print(" SM=" + Short.toString(z.anfangSichererZeitraum) + " EM=" + Short.toString(z.endeSichererZeitraum));
                }
                System.out.println();
            }
        }
    }
}
}
}

@Override
public String toString() { //Sicherheiten der Felder angeben
    String s = "";
    for (int y = felder[0].length - 1; y > -1; y--) { //Y-Achse umdrehen
        for (int x = 0; x < felder.length; x++) {
            if (felder[x][y].totalSicher) { //Ist das Feld total sicher, 2 ausgeben an der Stelle
                s += "2";
            } else if (felder[x][y].isEmpty()) { //Gibt es keine sicheren Zeiträume für das Feld
                s += "0"; //0 ausgeben
            } else { //Ansonsten gibt es Sichere
                s += "1"; //1 ausgeben
            }
        }
        if (x != felder.length - 1) {
            s += " ";
        }
    }
}

```



```
    }  
    s += "\n";  
  }  
  return s; //Textdarstellung zurückgeben  
}
```

Punkt.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package wintervorrat;

/**
 *
 * @author lars
 */
public class Punkt { //Speichert eine 2D-Koordinate

    public int x; //X
    public int y; //Y

    public Punkt(int x, int y) { //X und Y werden vorgegeben
        this.x = x;
        this.y = y;
    }

    public Punkt subtract(Punkt p) { //Erzeugt einen neuen Punkt von diesem Punkt minus einen Punkt einen Punkt p
        return new Punkt(x - p.x, y - p.y);
    }

    public Punkt add(Punkt p) { //Erzeugt einen neuen Punkt von diesem Punkt plus einen Punkt einen Punkt p
        return new Punkt(x + p.x, y + p.y);
    }
}

```

Zeitraum.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package wintervorrat;

/**
 *
 * @author lars
 */
public class Zeitraum { //Speichert einen Zeitraum

    public short anfangSichererZeitraum; //Erste Minute des sicheren Zeitraumes
    public short endeSichererZeitraum; //Letzte Minute des sicheren Zeitraumes

    public Zeitraum(short anfangSichererZeitraum, short endeSichererZeitraum) { //Konstruktor - Neuen Zeitraum aus vorgegebener Anfangs-und Endminute erstellen
        this.anfangSichererZeitraum = anfangSichererZeitraum;
    }
}

```

```

    this.endeSichererZeitraum = endeSichererZeitraum;
}
}

```

Wintervorrat.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package wintervorrat;

//Dateien
import java.io.File;
import java.io.FileNotFoundException; //Datei-nicht-gefunden-Fehler
import java.io.FileReader; //Dateien lesen
import java.io.IOException; //Eingabe-Ausgabe-Fehler
import java.util.Scanner; //Benutzereingabe

/**
 *
 * @author lars
 */
public class Wintervorrat {

    public static final Scanner EINGABE=new Scanner(System.in); //Benutzereingabe
    public static final int DAY = 60 * 12; //Länge eines Tages in Minuten

    public static String leseDatei(File pfad_zur_datei) throws FileNotFoundException, IOException { //Liest eine Datei ein, und gibt Text zurück
        FileReader datei = new FileReader(pfad_zur_datei);
        String r = "";
        int i = datei.read();
        while (i != -1) {
            r += (char) i;
            i = datei.read();
        }
        return r;
    }

    public static File dateiFrage(String frage) { //Fragt nach einem Pfad und prüft, ob dieser existiert
        while (true) {
            System.out.println(frage + "(Pfad zu .txt Datei) ? ");
            String s = EINGABE.nextLine();
            File f = new File(s);
            if (f.exists() && !f.isDirectory() && f.canRead()) { //Prüfe, ob : - existiert die Datei - ist es kein Ordner - ist sie lesbar
                return f; //Gebe Pfad zurück
            }
            System.out.println("Bitte antworten sie mit einem vorhandenen Pfad einer .txt Datei. Versuchen sie es erneut.");
        }
    }
}

```

```

    }
}

public static String textFrage(String frage) { //Fragt nach einer Zeichenkette
    System.out.println(frage + "(Zeichenkette) ? ");
    String s = EINGABE.nextLine();
    return s;
}

public static boolean jaNeinFrage(String frage) { //Stellt eine ja/nein Frage
    while (true) {
        System.out.println(frage + "(j/n) ? ");
        String s = EINGABE.nextLine().toLowerCase();
        if (s.equals("j")) {
            return true;
        } else if (s.equals("n")) {
            return false;
        }
        System.out.println("Bitte antworten sie mit j/n beziehungsweise J/N. Versuchen sie es erneut.");
    }
}

public static Punkt parseDir(String dir) { //Wandelt eine Himmelsrichtung in einen "Vektor"(Punkt) um
    switch (dir.charAt(0)) {
        case 'N': //Norden : nach oben
            return new Punkt(0, 1);
        case 'O': //Osten : nach rechts
            return new Punkt(1, 0);
        case 'S': //Süden : nach unten
            return new Punkt(0, -1);
        default: //Westen : nach links
            return new Punkt(-1, 0);
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws IOException {
    String file = leseDatei(dateiFrage("Wo befindet sich die Aufgabe"));
    boolean bilder = jaNeinFrage("Sollen Ausgabebilder gespeichert werden");
    boolean svg=false;
    boolean jpg=false;
    String ausgabe="?";
    if (bilder) {
        ausgabe = textFrage("Wo sollen die Ausgabebilder gespeichert werden(ohne Endung & Nummer)");
        svg = jaNeinFrage("Als SVG");
        jpg = jaNeinFrage("Als JPG");
    }
    String[] zeilen = file.split("\n"); //Zeilen der Datei, split("\n") trennt den String bei Zeilenumbrüchen
    String[] dim = zeilen[0].split(" "); //Dimensionen des Rechteckwalds, split(" ") trennt den String bei Leerzeichen
}

```

```

Adler[] adler = new Adler[Integer.parseInt(zeilen[1])]; //Adler-Array
for (int i = 2; i < zeilen.length; i++) {
    String[] info = zeilen[i].split(" "); //Werte eines Adlers, split(" ") trennt den String bei Leerzeichen
    Adler a = new Adler(Short.parseShort(info[2]), new Punkt(Integer.parseInt(info[0])-1, Integer.parseInt(info[1])-1, parseDir(info[3])); //Neuen Adler erzeugen
    adler[i - 2] = a; //Und speichern
}
Wald wald = new Wald(ausgabe,Integer.parseInt(dim[0]), Integer.parseInt(dim[1]), adler, svg, jpg); //Wald mit gegebener Breite und Höhe erzeugen, sowie gegebenem
Speichernamen für die Bilder und ob svgs oder jpgs gespeichert werden sollen
wald.simulieren(); //Wald simulieren
System.out.println(wald); //Sicherheiten der Felder des Waldes ausgeben
}
}

```

Adler.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package wintervorrat;

/**
 *
 * @author lars
 */
public class Adler { //Speichert alles, was wir über einen Adler im Rechteckwald wissen müssen

    public short startminute; //Startminute
    public Punkt position; //Startposition
    public Punkt flugrichtung; //Flugrichtung

    public Adler(short startminute, Punkt position, Punkt flugrichtung) { //Konstruktor - Neuen Adler mit gegebenen Werten erzeugen
        this.startminute = startminute;
        this.position = position;
        this.flugrichtung = flugrichtung;
    }
}

```

Feld.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package wintervorrat;

```

```

import java.util.ArrayList;

/**
 *
 * @author lars
 */
public class Feld extends ArrayList<Zeitraum> { //Ein Feld : Enthält alles was wir über ein Feld wissen müssen,,so eine Liste von sicheren Zeiträumen
    public boolean totalSicher; //Speichert, ob das Feld total sicher ist
    public short anfangSichererZeitraum; //Speichert den letzten Adlerüberflug + 1, den Anfang eines sicheren Zeitraums
    public Feld(boolean totalSicher) { //Konstruktor, Variablen initialisieren
        super(); //Liste initialisieren, noch kein Zeitraum bekannt
        this.totalSicher=totalSicher; //Ob das Feld als total sicher gilt, wird vorgegeben
        this.anfangSichererZeitraum=-1; //Noch kein bekannter sicherer Zeitraum
    }
}

```

1. Aufgabe „Zimmerbelegung“

Lösungsidee :

Gegeben ist für jedes Mädchen, welche Mädchen sie sich wünscht.

Also lässt sich auch für jedes Mädchen bestimmen, welche Mädchen sich sie wünschen.

Alle Mädchen, die sich sie wünschen oder die von ihr erwünscht werden, bezeichnen wir als „Freunde“.

Alle Mädchen, die sie nicht mag, bezeichnen wir als „Feinde“.

Ein Mädchen habe also Feinde und Freunde.

Ein Zimmer sei eine gemeinsame Ansammlung von Mädchen, und habe also gemeinsame Feinde. Diese werden auch als Gegner bezeichnet.

Den Prozess, die *Feinde* eines Mädchens zu Gegnern eines Zimmers zu erklären, und sie als Bewohnerin selbiges zu bezeichnen, nennen wir ein Mädchen zu einem Zimmer „hinzufügen“.

„Prüfen“, ob ein Mädchen zu einem Zimmer hinzugefügt werden kann, ist, zu ermitteln, ob ein Feind des Mädchens im Zimmer vorzufinden ist, oder sie als Gegnerin des Zimmers geführt wird. Sollte eine von diesen beiden Möglichkeiten der Fall sein, kann das Mädchen leider nicht zum Zimmer hinzugefügt werden. Genauso kann allerdings auch geprüft werden, ob zwei Zimmer zusammengeführt werden können. Dies ist der Fall, wenn kein Bewohner des ersten Zimmers vom zweiten Zimmer als Gegner geführt wird, und auch andersherum.

„Zusammenführen“ bezeichnet das Vorgehen, zwei Zimmer zusammen zu tun. Es entsteht dann ein neues Zimmer, welches die Bewohner von beiden Zimmern enthält, sowie die Gegner beider Zimmer.

Nun fangen wir mit dem ersten Mädchen an. Für sie erstellen wir ein neues Zimmer.

Dann fügen wir sie zum Zimmer hinzu. Für alle ihre Freunde prüfen wir jetzt, ob diese auch zum Zimmer hinzugefügt werden können. Sollte dies nicht der Fall sein, sind leider nicht alle Wünsche erfüllbar und man ist fertig. Für besagte Freunde werden jetzt wieder alle Freunde geprüft, die noch nicht als Bewohner des Zimmers bekannt sind, usw. Wenn wir irgendwann einmal keine offenstehenden Wünsche mehr haben, sind wir mit diesem Zimmer fertig.

Nun nehmen wir solange das nächste Mädchen, welches noch nicht als Bewohnerin eines Zimmers gilt, und wiederholen den fürs erste Mädchen beschriebenen Prozess, bis alle Mädchen als Bewohner von Zimmern gelten.

Sollten alle Wünsche erfüllbar gewesen sein, resultiert eine Menge an Zimmern, die eine gültige Zimmerbelegung repräsentiert.

Allerdings gilt es jetzt noch, diese Zimmerbelegung zu optimieren. So werden die meisten Lehrer wohl keine Einzelzimmer dulden, oder zum Zwecke einer besseren Klassengemeinschaft versuchen, möglichst wenige Zimmer zu erstellen.

Die Zimmerbelegung muss also noch optimiert werden, indem man zur Auswahl stellt, Zimmer zusammenzuführen, und ob mit Priorität Einzelzimmer vermieden werden sollen.

Dies funktioniert folgendermaßen :

Momentan gilt die gewünschte Zimmerbelegung als ideale Zimmerbelegung.

Sollen mit Priorität Einzelzimmer vermieden werden, so werden probeweise nur Einzelzimmer mit anderen Zimmern zusammengeführt.

Wir haben eine Zimmerbelegung. Wir gehen alle Zimmer dieser durch. Für jedes prüfen wir, ob es mit einem anderen Zimmer zusammengeführt werden kann. Wenn ja, machen wir dies probeweise und wiederholen den bis jetzt beschriebenen Algorithmus so lange, bis es keine Zimmer mehr zusammenzuführen gibt. Dann prüfen wir, ob bei dieser fiktiven Zimmerbelegung weniger Zimmer vorhanden wären als bei der momentan als ideal geltenden. Falls ja, gilt nun diese als ideal.

Umsetzung :

Bibliotheken :

- java.io.File : Dateien
- java.io.FileNotFoundException : Datei-nicht-gefunden-Fehler
- java.io.FileReader : Liest Textdateien ein
- java.io.IOException : Eingabe-Ausgabe-Fehler

- `java.util.ArrayList` : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können
- `java.util.List` : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können
- `java.util.HashMap` : „Wörterbuch“ : Jedem Schlüssel „s“ wird ein Wert „w“ zugeordnet. Sehr schnell wegen binärer Suche nach Hash.
- `java.math.BigDecimal` : Dezimalzahlen mit vielen Nachkommastellen
- `java.text.DecimalFormat` : Dezimalzahlen darstellen
- `java.util.Scanner` : Benutzereingabe, kann direkt aus der Befehlszeile lesen

Datenstrukturen :

Mädchen : Speichert Freundes- und Feindeslisten

Zimmer : Speichert ein „Wörterbuch“ mit Namen von Freunden, und eins mit Namen von Feinden, sowie eine Zimmer-ID. Bei den Wörterbüchern sind die Werte nicht relevant, sondern nur die Schlüssel. Damit wir jedoch schnell Freundes/Feindeslisten zusammenführen können, ohne dass doppelte Einträge entstehen, benutzen wir hier ein „Wörterbuch“. Diese Klasse bietet die Funktionen `kannLeiden` und `zimmerZusammenführen`. Erstere prüft, ob zwei Zimmer sich leiden können, letztere führt zwei Zimmer zusammen.

Programmstruktur :

Wir speichern die Mädchen in einem „Wörterbuch“, also als `HashMap`. So können wir schnell ermitteln, welche Freunde(und Feinde) ein Mädchen hat.

Wir implementieren die folgenden Funktionen : `leseDatei`, `dateiFrage`, und `jaNeinFrage`.

`dateiFrage` fragt den Benutzer nach dem Pfad zu einer Datei. Dies tut es so lange, bis der Benutzer einen validen Pfad angegeben hat.

`leseDatei` liest diese Datei ein.

Mit `jaNeinFrage` werden dem Nutzer die Fragen gestellt, ob er Einzelzimmer mit Priorität vermeiden will, und ob er möglichst weniger Zimmer erstellen will.

Die Ausgabe von `leseDatei` wird mit nach zwei Zeilenumbrüchen separiert. Übrig bleiben Zettel.

Von jedem Zettel ist nun die 1. Zeile das Mädchen, die Zweite(wenn ein + am Anfang steht) ihre Freundesliste, und dementsprechend die Dritte ihre Feindesliste.

Dies speichern wir dann in der `HashMap` ein.

Das Kernstück des Algorithmus bildet die Funktion „`prüfeMädchen`“. Diese prüft, ob ein Mädchen ins jetzige Zimmer integrierbar ist, und integriert es, falls möglich. Sollte es nicht gehen, ist die Aufgabenstellung nicht bewältigbar. Integrieren bedeutet auch, dass diese Funktion für alle ihre Bewunderer und Freunde auch angewandt wird, sowie dass sie nunmehr als „verplant“ gilt.

Erweitert wird die Aufgabenstellung von der Funktion „zimmerAuflösen“, welche die Lösung nach den vorgegebenen Kriterien optimiert. Auch diese Funktion ruft sich selbst auf. Die beste Zimmerbelegung wird in der globalen, gleichnamigen Variable gespeichert.

Nun gehen wir die Einträge der HashMap, also die Mädchen, durch.

Stoßen wir auf ein noch nicht verplantes Mädchen, erzeugen wir für es ein neues Zimmer. Zuerst wird die Funktion prüfeMädchen auf sie angewandt. Der Rest ergibt sich dann. Wurde das Zimmer fertig generiert, fügen wir es zur Liste der Zimmer, also der Zimmerbelegung, hinzu.

Diese Zimmerbelegung gilt dann als beste Zimmerbelegung. Wurde vom Benutzer eine Optimierung gewählt, wird nun die Funktion zimmerAufloesen für jede gewählte Optimierung gestartet, also maximal 2x.

Schließlich wird von der Funktion gebeListeAus jedes Zimmer möglichst übersichtlich dargestellt.

Beispiele :

/res/zimmerbelegung1.txt	/res/zimmerbelegung2.txt	/res/zimmerbelegung3.txt	/res/zimmerbelegung4.txt	/res/zimmerbelegung5.txt	/res/zimmerbelegung6.txt
Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/zimmerbelegung1.txt Hat es für sie Priorität, Einzelzimmer zu vermeiden(j/n) ? j Soll versucht werden, möglichst wenige Zimmer zu erstellen(j/n) ? j Freundeswunsch nicht erfüllbar : Lotta von Steffi, da Lotta ein Gegner des Zimmers ist	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/zimmerbelegung2.txt Hat es für sie Priorität, Einzelzimmer zu vermeiden(j/n) ? j Soll versucht werden, möglichst wenige Zimmer zu erstellen(j/n) ? j Zimmer : 1 mit 1 Bewohner Lara ----- Zimmer : 2 mit 3 Bewohnern Zoe Mia Emma ----- Zimmer : 3 mit 2 Bewohnern Alina Lilli	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/zimmerbelegung3.txt Hat es für sie Priorität, Einzelzimmer zu vermeiden(j/n) ? j Soll versucht werden, möglichst wenige Zimmer zu erstellen(j/n) ? j Zimmer : 1 mit 11 Bewohnern Lara, Charlotte, Laura Celina, Jasmin, Jessica Hannah, Luisa, Nina Miriam, Merle ----- Zimmer : 2 mit 14 Bewohnern Michelle, Larissa, Sofia Jana, Lisa, Julia Marie, Emily, Carolin Nele, Johanna, Emma Antonia, Anna ----- Zimmer : 3 mit 5 Bewohnern Celine, Lina Clara, Lena Lea -----	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/zimmerbelegung4.txt Hat es für sie Priorität, Einzelzimmer zu vermeiden(j/n) ? j Soll versucht werden, möglichst wenige Zimmer zu erstellen(j/n) ? j Zimmer : 1 mit 19 Bewohnern Michelle, Celine, Lara, Charlotte Celina, Vanessa, Clara, Hannah Pia, Luisa, Nina, Kim Lisa, Annika, Nele, Carolin Lina, Emma, Merle ----- Zimmer : 2 mit 6 Bewohnern Julia, Pauline Jasmin, Jessica Miriam, Anna ----- Zimmer : 3 mit 17 Bewohnern Sophie, Larissa, Sofia, Leonie Lena, Lea, Jana, Lilli Melina, Marie, Emily, Laura Sarah, Katharina, Alina, Josephine Johanna	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/zimmerbelegung5.txt Hat es für sie Priorität, Einzelzimmer zu vermeiden(j/n) ? j Soll versucht werden, möglichst wenige Zimmer zu erstellen(j/n) ? j Maries Wunsch kann leider nicht erfüllt werden. Denn sie kann die Insassin Marie nicht leiden	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/zimmerbelegung6.txt Hat es für sie Priorität, Einzelzimmer zu vermeiden(j/n) ? j Soll versucht werden, möglichst wenige Zimmer zu erstellen(j/n) ? j Zimmer : 1 mit 7 Bewohnern Michelle, Annika Celina, Jasmin Sofia, Lea Merle ----- Zimmer : 2 mit 25 Bewohnern Charlotte, Clara, Luisa, Lisa, Katharina Jessica, Josephine, Anna, Lara, Pauline Leonie, Lena, Pia, Nina, Kim Julia, Marie, Laura, Lina, Carolin Nele, Alina, Miriam, Emma, Antonia ----- Zimmer : 3 mit 11 Bewohnern Celine, Emily, Larissa

		Zimmer : 4 mit 13 Bewohnern Sophie, Pauline, Vanessa Leonie, Pia, Lilli Kim, Melina, Annika Katharina, Sarah, Alina Josephine	----- Zimmer : 4 mit 1 Bewohner Antonia		Sophie, Sarah, Vanessa Hannah, Jana, Lilli Johanna, Melina
--	--	--	---	--	--

Quellcode :

Zimmerbelegung.java :

```
package zimmerbelegung;
```

```
//Nötige Bibliotheken importieren
```

```
//Dateien einlesen
```

```
import java.io.File;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
```

```
import java.io.IOException;
```

```
//Listen
```

```
import java.util.ArrayList;
```

```
//HashMap
```

```
import java.util.HashMap;
```

```
import java.util.Map.Entry;
```

```
import java.util.Set;
```

```
//Benutzereingabe
```

```
import java.util.Scanner;
```

```
/**
```

```
*
```

```
* @author lars
```

```
*/
```

```
public class Zimmerbelegung {
```

```
    public static final Scanner EINGABE=new Scanner(System.in); //Benutzereingabe
```

```
    public static Zimmer zimmer; //Aktuelles Zimmer
```

```
    public static HashMap<String, Maedchen> maedchen; //Liste aller Maedchen
```

```
    public static ArrayList<Zimmer> besteZimmerbelegung; //Die ideale Zimmerbelegung(nach der Vorstellung des Lehrers)
```

```
    public static String leseDatei(File pfad_zur_datei) throws FileNotFoundException, IOException { //Liest eine Datei ein, und gibt Text zurück
```

```
        FileReader datei = new FileReader(pfad_zur_datei);
```

```
        String r = "";
```

```
        int i = datei.read();
```

```
        while (i != -1) {
```

```
            r += (char) i;
```

```
            i = datei.read();
```

```
        }
```

```

    return r;
}

public static File dateiFrage(String frage) { //Fragt nach einem Pfad und prüft, ob dieser existiert
    while (true) {
        System.out.println(frage + "(Pfad zu .txt Datei) ? ");
        String s = EINGABE.nextLine();
        File f = new File(s);
        if (f.exists() && !f.isDirectory() && f.canRead()) { //Prüfe, ob : - existiert die Datei - ist es kein Ordner - ist sie lesbar
            return f; //Gebe Pfad zurück
        }
        System.out.println("Bitte antworten sie mit einem vorhandenen Pfad einer .txt Datei. Versuchen sie es erneut.");
    }
}

public static boolean jaNeinFrage(String frage) { //Stellt eine ja/nein Frage
    while (true) {
        System.out.println(frage + "(j/n) ? ");
        String s = EINGABE.nextLine().toLowerCase();
        if (s.equals("j")) {
            return true;
        } else if (s.equals("n")) {
            return false;
        }
        System.out.println("Bitte antworten sie mit j/n beziehungsweise J/N. Versuchen sie es erneut.");
    }
}

public static <T> String gebeListeAus(Set<T> k) { //Gibt die "Schlüssel" der Einträge einer HashMap benutzerfreundlich aus
    int zeilenumbruch = (int) (Math.sqrt(k.size()));
    int counter = 0;
    int index = 0;
    String s = "";
    for (T objekt : k) {
        Entry e=(Entry)objekt;
        s += e.getKey().toString();
        counter++;
        index++;
        if (index != k.size()) {
            if (counter == zeilenumbruch) {
                s += "\n";
                counter = 0;
            } else {
                s += ", ";
            }
        }
    }
    return s;
}

```

```
public static void zimmerAufloesen(ArrayList<Zimmer> aktuelleZimmerbelegung, boolean einzelzimmer) { //Versucht, Zimmer zusammenzuführen, wobei welche "aufgelöst" werden.
```

```
    boolean kein_zusammenfuehren = true; //Konnten Zimmer zusammengeführt werden ?
```

```
    for (int i = 0; i < aktuelleZimmerbelegung.size(); i++) { //Alle Zimmer durchgehen
```

```
        Zimmer z = aktuelleZimmerbelegung.get(i);
```

```
        //Für alle Zimmer, für die noch nicht geprüft wurde, ob sie mit diesem Zimmer zusammengeführt werden können, wird dies geprüft und weiterverfolgt
```

```
        for (int j = i + 1; j < aktuelleZimmerbelegung.size(); j++) {
```

```
            Zimmer k = aktuelleZimmerbelegung.get(j);
```

```
            //Will der Lehrer nur Einzelzimmer vermeiden, werden keine nicht-Einzelzimmer mit nicht-Einzelzimmern zusammengeführt
```

```
            if (einzelzimmer && (z.zimmerinsassen.size() != 1 && k.zimmerinsassen.size() != 1)) {
```

```
                continue;
```

```
            }
```

```
            if (k.kannLeiden(z)) { //Können die Zimmer zusammengeführt werden ?
```

```
                kein_zusammenfuehren = false; //Dann konnten offensichtlich noch Zimmer zusammengeführt werden.
```

```
                //Diese fiktive Zimmerbelegung wird weiter überlegt
```

```
                ArrayList<Zimmer> kopie = new ArrayList();
```

```
                kopie.add(k.zimmerZusammenfuehren(z)); //Zimmer k und z als neues Zimmer hinzufügen
```

```
                //Kopie der aktuellen Zimmerbelegung anfertigen, ausgenommen Zimmer k und z
```

```
                for (int n = 0; n < aktuelleZimmerbelegung.size(); n++) {
```

```
                    if (n != i && n != j) {
```

```
                        kopie.add(aktuelleZimmerbelegung.get(n));
```

```
                    }
```

```
                }
```

```
                zimmerAufloesen(kopie, einzelzimmer); //Versuchen, weitere Zimmer zusammenzuführen und alle Möglichkeiten dies zu tun erstellen
```

```
            }
```

```
        }
```

```
    }
```

```
    if (kein_zusammenfuehren) { //Falls keine Zimmer mehr zusammengeführt werden konnten
```

```
        if (aktuelleZimmerbelegung.size() < besteZimmerbelegung.size()) { //Gibt es bei dieser Zimmerbelegung weniger Zimmer als bei der als ideal geltenden
```

```
            besteZimmerbelegung = new ArrayList();
```

```
            besteZimmerbelegung.addAll(aktuelleZimmerbelegung); //So gilt diese nun als ideal !
```

```
        }
```

```
    }
```

```
}
```

```
public static void pruefeMaedchen(String girl) { //Prüft, ob ein Mädchen in einem Zimmer aufgenommen werden kann
```

```
    Maedchen g = maedchen.get(girl); //Freundes/Feindeslisten holen
```

```
    g.zimmer = zimmer.id; //Das Mädchen soll ab jetzt als Mitglied dieses Zimmers gelten
```

```
    maedchen.put(girl, g); //HashMap updaten
```

```
    zimmer.zimmerinsassen.put(girl, false); //Es gehört nun zu den Mitgliedern des Zimmers
```

```
    for (String s : g.feinde) { //Für alle unerwünschten Mädchen
```

```
        if (zimmer.zimmerinsassen.get(s) == null) { //Befindet sich das unerwünschte Mädchen in diesem Zimmer
```

```
            zimmer.zimmergegner.put(s, false); //Dieser Gegner wird zum Gegner des aktuellen Zimmers erklärt
```

```
        } else { //Befindet sich ein Gegner des Mädchens im Zimmer, sind nicht alle Wünsche erfüllbar, und das Programm stoppt
```

```
            System.out.println(girl + "s Wunsch kann leider nicht erfüllt werden. Denn sie kann die Insassin " + s + " nicht leiden");
```

```
            System.exit(0);
```

```
            return;
```

```
        }
```

```
    }
```

```
    ArrayList<String> freunde_und_bewunderer = new ArrayList();
```

```
    freunde_und_bewunderer.addAll(g.freunde);
```

```

for (Entry e : maedchen.entrySet()) { //Finde alle Bewunderer, die noch aufgenommen werden sollen
    Maedchen moeglicher_bewunderer = (Maedchen) e.getValue();
    if (moeglicher_bewunderer.zimmer == -1) { //Ist das Mädchen noch nicht in einem Zimmer, sonst hätte es nämlich schon alle dort aufnehmen lassen, die es mag
        for (String bewundert : moeglicher_bewunderer.freunde) {
            if (bewundert.equals(girl)) { //Wird dieses Mädchen bewundert
                freunde_und_bewunderer.add((String) e.getKey()); //Zu Freunden und Bewunderern hinzufügen
            }
        }
    }
}

for (String friend : freunde_und_bewunderer) { //Jetzt muss geprüft werden, ob besagte Freunde und Bewunderer aufgenommen werden können
    if (zimmer.zimmergegner.get(friend) != null) { //Wenn eine dieser Personen als Gegner bekannt ist, sind nicht alle Wünsche erfüllbar, und das Programm stoppt
        System.out.println("Freundeswunsch nicht erfüllbar : " + friend + " von " + girl + ", da " + friend + " ein Gegner des Zimmers ist");
        System.exit(0);
        return;
    } else if (maedchen.get(friend).zimmer == -1) { //Wenn das Mädchen noch nicht verplant ist, muss es neu geprüft werden
        pruefeMaedchen(friend); //Das Mädchen prüfen
    }
}
}

public static void main(String[] args) throws IOException {
    maedchen = new HashMap(); //HashMap mit Mädchen initialisieren
    zimmer = new Zimmer((byte) 1); //Aktuelles Zimmer initialisieren
    ArrayList<Zimmer> gewuenschte_zimmerbelegung = new ArrayList(); //Liste von Zimmern(erwünschte Zimmerbelegung)
    String[] zettel = leseDatei(dateiFrage("Wo befindet sich die Aufgabe")).split("\n\n"); //Liste aller Zettel. split("\n\n") teilt die Zeichenkette bei zwei Zeilenumbrüchen
    boolean einzelzimmerAufloesen = jaNeinFrage("Hat es für sie Priorität, Einzelzimmer zu vermeiden");
    boolean zimmerAufloesen = jaNeinFrage("Soll versucht werden, möglichst wenige Zimmer zu erstellen");
    for (String z : zettel) { //Alle Zettel durchgehen
        String[] infos = z.split("\n"); //Informationen des jeweiligen Mädchens
        String name = infos[0];
        String[] friends, enemies;
        friends = enemies = new String[]{};
        if (infos[1].charAt(0) == '+') { //Entspricht das erste Zeichen der zweiten Zeile des Zettels einem Plus, werden zuerst die Freunde, dann die Feinde aufgeführt
            if (infos[1].length() > 2) {
                friends = infos[1].substring(2).split(" ");
            }
            if (infos[2].length() > 2) {
                enemies = infos[2].substring(2).split(" ");
            }
        } else { //Sollte es nicht so sein, werden zuerst die Feinde, dann die Freunde aufgeführt
            if (infos[2].length() > 2) {
                friends = infos[2].substring(2).split(" ");
            }
            if (infos[1].length() > 2) {
                enemies = infos[1].substring(2).split(" ");
            }
        }
        maedchen.put(name, new Maedchen(friends, enemies));
    }
    for (Entry e : maedchen.entrySet()) {

```

```

        Maedchen mz = (Maedchen) e.getValue();
        if (mz.zimmer == -1) { //Sollte das Mädchen noch nicht verplant sein
            pruefeMaedchen((String) e.getKey()); //So wird es in ein neues Zimmer gepackt
            gewuenschte_zimmerbelegung.add(zimmer); //Das Zimmer wird zur Zimmerbelegung hinzugefügt
            zimmer = new Zimmer((byte) (zimmer.id + 1)); //Für's nächste Zimmer geht die Zimmernummer um 1 hoch
        }
    }
    besteZimmerbelegung = new ArrayList();
    besteZimmerbelegung.addAll(gewuenschte_zimmerbelegung);
    if (einzelzimmerAufloesen) {
        zimmerAufloesen(besteZimmerbelegung, true); //Falls gewünscht, wird die Zimmerbelegung optimiert indem zuerst Einzelzimmer aufgelöst werden.
    }
    if (zimmerAufloesen) {
        zimmerAufloesen(besteZimmerbelegung, false); //Falls gewünscht, wird die Zimmerbelegung optimiert indem möglichst wenige Zimmer entstehen sollen.
    }
    for (Zimmer z : besteZimmerbelegung) {
        System.out.println(z); //Ausgabe des Zimmers, hierbei werden die Zimmer automatisch mithilfe des Zählers durchnummeriert
    }
}
}
}

```

Zimmer.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zimmerbelegung;

import java.util.HashMap;
import java.util.Map.Entry;

/**
 *
 * @author lars
 */
public class Zimmer {

    public static byte ZAEHLER = 1; //Zähler fürs Durchnummerieren
    public byte id; //ID
    public HashMap<String, Boolean> zimmergegner; //Gegner, Wert ist hier egal
    public HashMap<String, Boolean> zimmerinsassen; //Insassen, Wert ist hier egal

    public Zimmer(byte id) { //Konstruktor, Zimmernummer wird vorgegeben
        this.id = id;
        zimmerinsassen = new HashMap();
        zimmergegner = new HashMap();
    }
}

```

```

public static void zaehlerZuruecksetzen() {
    ZAEHLER = 1;
}

public boolean kannLeiden(Zimmer z) { //Prüft, ob ein Zimmer und ein Anderes zusammengeführt werden können, also ob kein Zimmerinsasse einen Insassen des anderen
Zimmer als Gegner hat
    for (Entry e : z.zimmerinsassen.entrySet()) {
        String insasse=(String)e.getKey();
        for (Entry v : zimmergegner.entrySet()) {
            String gegner=(String)v.getKey();
            if (insasse.equals(gegner)) { //Ist ein Gegner dieses Zimmers im anderen Zimmer ?
                return false; //Gebe zurück : Leider können sich die Zimmer nicht leiden !
            }
        }
    }
    for (Entry e : zimmerinsassen.entrySet()) {
        String insasse=(String)e.getKey();
        for (Entry v : z.zimmergegner.entrySet()) {
            String gegner=(String)v.getKey();
            if (insasse.equals(gegner)) { //Ist ein Gegner des anderen Zimmers in diesem Zimmer ?
                return false; //Gebe zurück : Leider können sich die Zimmer nicht leiden !
            }
        }
    }
    return true; //Wenn nichts davon der Fall ist, können sich die Zimmer leiden.
}

public Zimmer zimmerZusammenfuehren(Zimmer z) { //Neues Zimmer aus diesem Zimmer + Zimmer z erstellen
    Zimmer g=new Zimmer(this.id);
    g.zimmerinsassen.putAll(z.zimmerinsassen);
    g.zimmergegner.putAll(z.zimmergegner);
    g.zimmerinsassen.putAll(zimmerinsassen);
    g.zimmergegner.putAll(zimmergegner);
    return g;
}

@Override
public String toString() { //Zimmer ausgeben, optimierte Darstellung
    String b = "Zimmer : " + Byte.toString(ZAEHLER) + " mit " + Integer.toString(zimmerinsassen.size()) + " Bewohner";
    if (zimmerinsassen.size() != 1) {
        b += "n";
    }
    String s = "";
    if (ZAEHLER != 1) {
        for (int i = 0; i < b.length(); i++) {
            s += "-";
        }
    }
    s += "\n" + b + "\n";
    s+=Zimmerbelegung.gebeListeAus(zimmerinsassen.entrySet());
    ZAEHLER++;
}

```

```
    return s;  
}  
}
```

Maedchen.java :

```
/*  
 * To change this license header, choose License Headers in Project Properties.  
 * To change this template file, choose Tools | Templates  
 * and open the template in the editor.  
 */  
package zimmerbelegung;  
  
import java.util.Arrays;  
import java.util.List;  
  
/**  
 *  
 * @author lars  
 */  
public class Maedchen {  
  
    public List<String> freunde; //Liste von Freunden  
    public List<String> feinde; //Liste von Feinden  
    public byte zimmer; //Nummer des Zimmers  
  
    public Maedchen(String[] friends, String[] enemies) { //Konstruktor  
        this.freunde = Arrays.asList(friends);  
        this.feinde = Arrays.asList(enemies);  
        zimmer = -1; //Unverplante Mädchen haben -1 als Zimmernummer  
    }  
}
```

2. Aufgabe „Schwimmbad“

Lösungsidee :

Überlegungen :

1. Da der Eintritt für Kleinkinder in Begleitung eines Erwachsenen gratis ist, muss man nur prüfen, ob bei dem Ausflug auch mindestens ein Erwachsener mitkommt, sonst kann nämlich das Schwimmbad nicht besucht werden. Ansonsten können die Kleinkinder weggelassen werden.
2. An Wochentagen gibt es 20 % Ermäßigung, also kostet dann Kinder der Eintritt 2 € und für Erwachsene der Eintritt 2,8 €.

3. An Wochentagen lohnen sich imperfekte, also nicht komplett gefüllte Familienkarten, nicht, da : $2 \text{ E}, 1 \text{ K (worst case)} : 5,6+2 = 7,6 < 8 \text{ €}$

4. Anders an Wochenenden, dann werden zuerst so viele Familienkarten wie möglich erstellt, um dann zu prüfen, ob noch imperfekte erstellt werden können. Hierbei gilt jedoch der Zusatz, dass es sich noch lohnt, falls :

- eine Familienkarte Typ 2 vorhanden ist (3 K, 1 E)

- min. 3 Erwachsene übrig bleiben

die Familienkarte Typ 2 aufzulösen, und eine imperfekte Familienkarte zu erstellen, Typ 1, mit 1 Erwachsenen, und 2 Kindern

5. Familienkarten / imperfekte Familienkarten verteilen wir, in dem wir alle Möglichkeiten, diese zu kaufen ausprobieren, und die, bei der die Einzelkarten für die übrig bleibenden Kinder am wenigsten kosten würden. Beispiel : 12 Personen sollen in Familienkarten untergebracht werden.

Also brauchen wir $12/4 = 3 \text{ FKs}$.

Alle Möglichkeiten sind also (1 = Typ 1, 2 = Typ 2) :

1 1 1 1

2 1 1 1

2 2 1 1

2 2 2 1

2 2 2 2

Bei diesen probieren wir jetzt aus, wie viele Familienkarten, vom Anfang an durchgehend, davon besorgt werden können. Übrig bleibt ein Rest. Wir probieren dies für alle Möglichkeiten, und ermitteln die, mit dem Rest, der am billigsten wäre, also restliche $\text{E} * 3,50 + \text{restliche K} * 2,50$.

6. Da sich Tageskarten am meisten lohnen, füllen wir an Wochentagen so viele von diesen, wie möglich.

Es gibt 5 Möglichkeiten, wie viele Personen übrig bleiben :

1 – Eine Person bleibt übrig. In dem Fall lohnt es sich am meisten, für diese einfach noch eine Einzelkarte zu besorgen, da diese ja auch noch 20 % ermäßigt ist.

2 – Zwei Personen bleiben übrig. Genauso wie (1).

3 – Drei Personen bleiben übrig. Genauso wie (1).

4 – Vier Personen bleiben übrig. Zuerst sollte versucht werden, noch eine FK reinzupacken. Falls dies nicht geht, wird eine imperfekte Tageskarte besorgt, wenn es alles nur Erwachsene sind – denn dann würde sich der Preis für Einzelkarten auf 20 Cent mehr belaufen. Ansonsten werden einfach Einzelkarten besorgt.

5 – Fünf Personen bleiben übrig. In diesem Falle sollte zuerst versucht werden, noch eine FK reinzupacken. Geht dies leider nicht, sollte eine imperfekte Tageskarte gekauft werden, wenn min. 2 Erwachsene vorhanden sind, da : $5,6+6=11,6 \text{ €} > 11 \text{ €}$.

Fragt sich nur noch, wie wir mit den Gutscheinen verfahren.

Liegen mindestens so viele Gutscheine wie zahlungspflichtige Personen vor, setzten wir einfach die Gutscheine ein.

Ansonsten geschieht Folgendes :

- Wir generieren alle Möglichkeiten, die Gutscheine einzusetzen, und probieren diese dann aus. Möge die Günstigste gewinnen ! Bsp. :

2 Gutscheine - Möglichkeiten :

- 1 Gutschein für die Ermäßigung einsetzen & 1 für ein Kind

- 1 Gutschein für die Ermäßigung einsetzen & 1 für einen Erwachsenen
- beide Gutscheine für Erwachsene einsetzen
- beide Gutscheine für Kinder einsetzen
- einen Gutschein für ein Kind, den anderen für einen Erwachsenen einsetzen.

Umsetzung :

Bibliotheken :

- java.util.Scanner : Benutzereingabe

Datenstrukturen :

Besuch : Speichert die Kartenverteilung, berechnet den Preis. Enthält : Einzelkarten E/K, Familienkarten, Tageskarten, Einsetzen der Gutscheine, etc. Greift dabei auf die globale Variable „wochentag“ zu.

Programmstruktur :

Wir implementieren die Funktionen mengenFrage und jaNeinFrage. Dann gilt es vom Benutzer zunächst folgendes zu erfahren :

jaNeinFrage :

Wochentag ?

Schultag(Schulzeit) ?

mengenFrage :

Anzahl Personen ?

Anzahl Kinder ?

Anzahl Kleinkinder ?

→ Anzahl Erwachsene

Wurde außerdem Schulzeit mit ja beantwortet, ist noch relevant :

Wie viele Gutscheine gibt es ?

Alle diese Informationen werden in globalen Variablen gespeichert, mit Ausnahme der Kleinkinder. Diese werden aus der Kinder und Personenzahl entfernt, da sie zur Berechnung der idealen Kartenverteilung irrelevant sind.

Nun haben wir eine Funktion fülleFamilienkarten. Diese bestimmt, wie in der Lösungsidee angesprochen, aus allen Möglichkeiten, Familienkarten zu erstellen, die Beste. Dasselbe gibt es für imperfekte FKs. Diese bauen wiederum auf Unterfunktionen auf, die ermitteln, wie viele Personen bei

anwenden der Möglichkeit untergebracht werden können, und wie viele übrig bleiben. Auch implementieren wir eine Funktion „fülleTageskarten“. Diese packt so viele Personen wie möglich in Tageskarten, bevorzugt dabei Erwachsene, da Einzelkarten für diese teurer sind.

Ableitend aus den bisher erstellten Funktionen, machen wir zwei weitere. „kartenKeinWochentag“, um eine Kartenverteilung für einen nicht-Wochentag zu ermitteln, und „kartenWochentag“, um eine Kartenverteilung für einen Wochentag zu bestimmen. Erstere baut auf sämtlichen „fülle“-Funktionen auf und arbeitet wie in der Lösungsidee erwähnt :

Zuerst werden die evtl. noch reinpassenden FKs ermittelt, danach erst wird fülleTKs angewandt, da für die FKs relevant ist, **wer** hinein kommt, während dies bei den Tks nicht der Fall ist. Imperfekte Familienkarten benötigen wir hier nicht. Es können aber imperfekte Tks erstellt werden. Bei „kartenKeinWochentag“ sind keine Tageskarten möglich. Also wird zuerst fülleFKs, und mit dem Personen-Rest dieses Aufrufs, fülleImperfekteFKs aufgerufen, wobei nach Aufrufen beider Funktionen zusätzlich allerdings noch der Fall aus Überlegung (4) der Lösungsidee berücksichtigt wird, und somit dann die Ergebnisse von „außen“ verändert werden. Als Summe dieser Funktionen existiert die Funktion „karten“. Diese erstellt die Kartenverteilung, wobei sie je nach globaler Variable „kartenKeinWochentag“ oder „kartenWochentag“ aufruft. Das Ergebnis wird in der globalen Variable „bester_besuch“ gespeichert, welche den idealen Besuch für die gegebenen Bedingungen repräsentiert.

Nun kommen wir zurück zum jetzt ausgeführten. Die Fragen sind gestellt worden. Falls es Gutscheine gibt, ermitteln wir jetzt, ob mindestens so viele Gutscheine wie Personen vorhanden sind. Falls ja, geben wir aus, dass so viele Gutscheine wie Personen zu benutzen sind, und sind fertig. Ansonsten werden in der Hauptfunktion, ähnlich dem Algorithmus der alle Möglichkeiten FKs zu kaufen ermittelt, alle Möglichkeiten, Gutscheine einzusetzen, mit einer for-Schleife bestimmt. Auch muss hier jedesmal noch die Möglichkeit berücksichtigt werden, einen Gutschein zur Ermäßigung einzusetzen. Die globalen Variablen, die „karten“ ließt, werden ständig, je nach Möglichkeit, angepasst, und so werden die Möglichkeiten durchprobiert. Die Billigste von allen wird dann final ausgegeben und das Programm ist fertig. Gibt es jedoch keine Gutscheine, bzw. ist kein Schultag, brauchen wir einfach nur karten auszuführen, mit gegebenen Variablen, und sind fertig.

Beispiele :

Beispiel 1 :

Wochentag(j/n) ? j Schultag(j/n) ? n Wie viele Personen(Zahl) ? 27 Wie viele sind Kinder(unter 16)(Zahl) ? 0 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 0 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 0 Familienkarten(1 Erwachsener, 3 Kinder) : 0 Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0	Wochentag(j/n) ? n Schultag(j/n) ? j Wie viele Personen(Zahl) ? 27 Wie viele Gutscheine(Zahl) ? 3 Wie viele sind Kinder(unter 16)(Zahl) ? 0 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 0 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 0 Familienkarten(1 Erwachsener, 3 Kinder) : 0
--	--

Tageskarten : 4 Einzelkarten Erwachsene : 3 Einzelkarten Kinder : 0 Es werden 0 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt. Es wird kein Gutschein für die Ermäßigung von 10 % eingesetzt. Preis : 52,4	Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0 Tageskarten : 0 Einzelkarten Erwachsene : 25 Einzelkarten Kinder : 0 Es werden 2 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt. Es wird ein Gutschein für die Ermäßigung von 10 % eingesetzt. Preis : 78.75
---	---

Da 78, 75 € beträchtlich mehr als 52,4 € sind, sollten sie ihre ursprüngliche Terminwahl beibehalten.

Beispiel 2 :

Es gibt 4 Erwachsene. Es gibt 6 Kinder. Eines davon ist ein Kleinkind.

Wochentag(j/n) ? j Schultag(j/n) ? n Wie viele Personen(Zahl) ? 10 Wie viele sind Kinder(unter 16)(Zahl) ? 6 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 1 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 0 Familienkarten(1 Erwachsener, 3 Kinder) : 0 Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0 Tageskarten : 1 Einzelkarten Erwachsene : 0 Einzelkarten Kinder : 3 Es werden 0 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt. Es wird kein Gutschein für die Ermäßigung von 10 % eingesetzt. Preis : 17.0	Wochentag(j/n) ? n Schultag(j/n) ? n Wie viele Personen(Zahl) ? 10 Wie viele sind Kinder(unter 16)(Zahl) ? 6 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 1 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 2 Familienkarten(1 Erwachsener, 3 Kinder) : 0 Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0 Tageskarten : 0 Einzelkarten Erwachsene : 0 Einzelkarten Kinder : 1 Es werden 0 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt. Es wird kein Gutschein für die Ermäßigung von 10 % eingesetzt. Preis : 18.5
---	---

Am Wochenende würde es 17 € kosten, ansonsten 18,5 €.

Beispiel 3 :

<p>Wochentag(j/n) ? n Schultag(j/n) ? j Wie viele Personen(Zahl) ? 32 Wie viele Gutscheine(Zahl) ? 1 Wie viele sind Kinder(unter 16)(Zahl) ? 0 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 0 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 0 Familienkarten(1 Erwachsener, 3 Kinder) : 0 Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0 Tageskarten : 0 Einzelkarten Erwachsene : 14 Einzelkarten Kinder : 0 Es werden 0 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt. Es wird ein Gutschein für die Ermäßigung von 10 % eingesetzt. Preis : 44.1</p>	<p>Wochentag(j/n) ? n Schultag(j/n) ? j Wie viele Personen(Zahl) ? 32 Wie viele Gutscheine(Zahl) ? 1 Wie viele sind Kinder(unter 16)(Zahl) ? 0 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 0 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 0 Familienkarten(1 Erwachsener, 3 Kinder) : 0 Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0 Tageskarten : 0 Einzelkarten Erwachsene : 18 Einzelkarten Kinder : 0 Es werden 0 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt. Es wird ein Gutschein für die Ermäßigung von 10 % eingesetzt. Preis : 56.699997</p>
--	---

44.1+56.7=100.8 €

<p>Wochentag(j/n) ? n Schultag(j/n) ? j Wie viele Personen(Zahl) ? 32 Wie viele Gutscheine(Zahl) ? 2 Wie viele sind Kinder(unter 16)(Zahl) ? 0 Wie viele davon sind Kleinkinder(unter 4)(Zahl) ? 0 Besuch beim Schwimmbad - Kosten Familienkarten(2 Erwachsene, 2 Kinder) : 0 Familienkarten(1 Erwachsener, 3 Kinder) : 0 Familienkarten(2 Erwachsene, 1 Kind) : 0 Familienkarten(1 Erwachsener, 2 Kinder) : 0 Tageskarten : 0 Einzelkarten Erwachsene : 31 Einzelkarten Kinder : 0 Es werden 1 Gutscheine für den freien Eintritt von Erwachsenen benutzt. Es werden 0 Gutscheine für den freien Eintritt von Kindern benutzt.</p>

Es wird ein Gutschein für die Ermäßigung von 10 % eingesetzt.
Preis : 97.649994

$100.80 - 97.65 = 3.15$ € Ersparnis, wenn beide Kegelveereine zusammen gehen.

Quellcode :

Schwimmbad.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package schwimmbad;

import java.util.Scanner; //Benutzereingabe

/**
 *
 * @author lars
 */
public class Schwimmbad {

    public static final Scanner EINGABE = new Scanner(System.in); //Benutzereingabe
    public static boolean wochentag; //Wochentag ?
    public static boolean schultag; //Schultag ?
    public static int personen; //Anzahl Personen, exklusive Kleinkinder
    public static int erwachsene; //Anzahl Erwachsene
    public static int kinder; //Anzahl Kinder, exklusive Kleinkinder
    public static int gutscheine; //Gutscheine, 0 wenn es kein Schultag ist, oder es einfach keine gibt
    public static Besuch bester_besuch; //Billigste Möglichkeit, Karten zu kaufen, für die gegebenen Infos(Gutscheine, E, K, Schul-/Wochentag, etc.)

    public static int[] familienKarten(int t1, int t2, int erwachsene, int kinder) { //Versucht für Erwachsene und Kinder t1 FKs Typ 1, sowie t2 FKs Typ 2 zu organisieren. Mit Priorität
        //Konzept identisch zu imperfekteFamilienkarten, nur eben mit normalen FKs
        int e = erwachsene;
        int k = kinder;
        int f1, f2;
        f1 = 0;
        f2 = 0;
        //Zuerst Typ 1 Familienkarten erzeugen, da diese mehr Ersparnis bringen, solange es geht
        for (int i = 0; i < t1; i++) {
            if (e < 2 || k < 2) {
                break;
            }
        }
    }
}
```

```

    e -= 2;
    k -= 2;
    f1++;
}
//Danach Typ 2 erzeugen, solange dass gut geht
for (int i = 0; i < t2; i++) {
    if (e < 1 || k < 3) {
        break;
    }
    e--;
    k -= 3;
    f2++;
}
return new int[]{f1, f2, e, k};
}

```

```

public static int[] illegaleFamilienKarten(int t1, int t2, int erwachsene, int kinder) {
    //Konzept identisch zu familienkarten, nur eben mit "nicht-vollen" aka "illegalen/imperfekten" FKs. Diese sind : Typ 1 : 2 E, 1 K, sowie 1 E, 2 K
    int e = erwachsene;
    int k = kinder;
    int f1, f2;
    f1 = 0;
    f2 = 0;
    for (int i = 0; i < t1; i++) {
        if (e < 2 || k < 1) {
            break;
        }
        e -= 2;
        k--;
        f1++;
    }
    for (int i = 0; i < t2; i++) {
        if (e < 1 || k < 2) {
            break;
        }
        e--;
        k -= 2;
        f2++;
    }
    return new int[]{f1, f2, e, k};
}

```

```

public static int[] fuehleFamilienkarten(int b, int erwachsene, int kinder) {
    //Konzept identisch zu fülleImperfekteFamilienkarten, nur eben mit "normalen" Familienkarten
    //Generiert alle Möglichkeiten, "b" FKs zu erzeugen. Die Möglichkeit, bei der der Rest am billigsten wäre, wird zurückgegeben.
    //Achtung : 20 % Ermässigung sind hierbei egal, da das Verhältnis wichtig ist
    int[] best_combo = new int[4];
    int min_rest = Integer.MAX_VALUE;
    float min_preis = Float.MAX_VALUE;
    for (int i = 0; i <= b; i++) {
        int j = (b - i);
    }
}

```

```

    int[] combo = familienKarten(i, j, erwachsene, kinder); //Familienkarten, soviel es geht, nach dieser Kombi erzeugen
    if (combo[2] + combo[3] < min_rest) {
        System.arraycopy(combo, 0, best_combo, 0, 4);
        min_rest = (combo[2] + combo[3]);
        min_preis = combo[2] * 3.50f + combo[3] * 2.50f;
    } else if (combo[2] + combo[3] == min_rest) {
        float preis = combo[2] * 3.50f + combo[3] * 2.50f;
        if (preis < min_preis) { //Wäre der Preis des Restes "geringer" ?
            //Gilt dies nun als die beste Kombi !
            System.arraycopy(combo, 0, best_combo, 0, 4);
            min_preis = combo[2] * 3.50f + combo[3] * 2.50f;
        }
    }
}
}
return best_combo;
}

public static int[] fuelleImperfekteFamilienkarten(int karten, int e, int k) {
    //Konzept identisch zu fülleFamilienkarten, nur eben mit "imperfekten/illegalen" Familienkarten
    //Achtung : 20 % Ermässigung sind hierbei egal, da das Verhältnis wichtig ist
    int[] best_combo = new int[4];
    int min_rest = Integer.MAX_VALUE;
    float min_preis = Float.MAX_VALUE;
    for (int i = 0; i <= karten; i++) {
        int j = (karten - i);
        int[] combo = illegaleFamilienKarten(i, j, e, k); //Illegale Familienkarten, soviel es geht, nach dieser Kombi erzeugen
        if (combo[2] + combo[3] < min_rest) {
            System.arraycopy(combo, 0, best_combo, 0, 4);
            min_rest = (combo[2] + combo[3]);
            min_preis = combo[2] * 3.50f + combo[3] * 2.50f;
        } else if (combo[2] + combo[3] == min_rest) {
            float preis = combo[2] * 3.50f + combo[3] * 2.50f;
            if (preis < min_preis) {
                System.arraycopy(combo, 0, best_combo, 0, 4);
                min_preis = combo[2] * 3.50f + combo[3] * 2.50f;
            }
        }
    }
}
return best_combo;
}

public static int[] fuelleImperfekteFamilienkarten(int e, int k) {
    return fuelleImperfekteFamilienkarten((e + k) / 3, e, k); //Imperfekte Familienkarten erzeugen, soviel es geht
}

public static int[] fuelleTageskarten(int b, int erwachsene, int kinder) { //Versucht, b Tageskarten zu erstellen. Gibt zurück, was übrig bleibt.
    int e = erwachsene;
    int k = kinder;
    for (int i = 0; i < b; i++) {
        for (int j = 0; j < 6; j++) {
            if (e != 0) { //Solange es noch Erwachsene gibt, diese in Tageskarten packen, da sie teurer sind

```



```

    e--;
} else if (k != 0) {
    k--; //Ansonsten Kinder in Tageskarten packen
} else { //Es gibt keine Erwachsenen und keine Kinder mehr
    break; //Man kann aufhören
}
}
}
return new int[]{e, k};
}

public static void kartenWochentag() { //Kartenverteilung an einem Wochentag, Achtung : 20 % REDUKTION, also lohnen sich imperfekte Familienkarten nicht mehr !
    int[] result = new int[3];
    result[0] = (personen / 6); //Tageskarten, die verteilt werden werden
    int rest = personen - (result[0] * 6); //Zahl der Personen, die übrig bleiben werden
    if (rest <= 2) { //Gibt es zwei, oder gar weniger als zwei Personen Rest
        int[] restliche_einzelkarten = fuelleTageskarten(result[0], erwachsene, kinder); //Tageskarten erstellen
        bester_besuch = new Besuch(restliche_einzelkarten[0] /*Restliche Erwachsene*/, restliche_einzelkarten[1] /*Restliche Kinder*/, new int[4] /*TKs*/, result[0]); //Ergebnis
        //zurückgeben
    } else if (rest == 3) { //Gibt es 3 Personen Rest
        int[] restliche_einzelkarten = fuelleTageskarten(result[0], erwachsene, kinder); //Tageskarten erstellen
        bester_besuch = new Besuch(restliche_einzelkarten[0], restliche_einzelkarten[1], new int[4], result[0]);
    } else if (rest == 4) { //Gibt es 4 P Rest
        int[] restliche_personen = fuelleFamilienkarten(1, erwachsene, kinder); //Wir versuchen, eine Familienkarte unterzubringen
        if (restliche_personen[0] + restliche_personen[1] == 0) { //Hat dies nicht geklappt
            if (kinder == 0) { //Worst case : Alle sind erwachsen
                bester_besuch = new Besuch(0, 0, new int[4], result[0] + 1); //Nur dann lohnt sich doch eine imperfekte Tageskarte !
            } else {
                int[] restliche_einzelkarten = fuelleTageskarten(result[0], erwachsene, kinder); //Ansonsten einfach billige, ermässigte Einzelkarten
                bester_besuch = new Besuch(restliche_einzelkarten[0], restliche_einzelkarten[1], new int[4], result[0]); //Ansonsten lohnen sich wohl doch die Einzelkarten am
                //meisten !
            }
        } else { //Hat's geklappt
            bester_besuch = new Besuch(0, 0, new int[]{restliche_personen[0], restliche_personen[1], 0, 0}, result[0]);
        }
    } else if (rest == 5) { //Gibt es 5 P Rest
        int[] restliche_personen = fuelleFamilienkarten(1, erwachsene, kinder); //Erstmal ermitteln wir, ob sich noch eine Familienkarte unterbringen ließe
        if (restliche_personen[0] + restliche_personen[1] == 0) { //Ist dies der Fall
            bester_besuch = new Besuch(restliche_personen[2], restliche_personen[3], new int[4], result[0] + 1); //Haben wir ein Ergebnis !
        } else if (restliche_personen[2] > 1) { //Ist dies nicht der Fall, gebe es aber mindestens zwei Erwachsene, lohnt sich eine imperfekte TK
            bester_besuch = new Besuch(0, 0, new int[]{restliche_personen[0], restliche_personen[1], 0, 0}, result[0]);
        }
    }
}

}

public static void kartenWochenende() { //Kartenverteilung an keinem Wochentag : Keine Tageskarten
    int[] fakas = fuelleFamilienkarten(personen / 4, erwachsene, kinder); //Zuerst werden Familienkarten erstellt
    int rest_fakas[] = fuelleImperfekteFamilienkarten(fakas[2], fakas[3]); //Aus dem Rest werden imperfekte Familienkarten erstellt
    if (rest_fakas[2] > 2) { //Gibt es mindestens drei übrig bleibende Erwachsene
        if (fakas[1] > 0) { //Gibt es mindestens eine Familienkarte vom Typ 2, also 1 E, 3 K
            fakas[1]--; //Dann lösen wir diese auf
        }
    }
}

```

```

        rest_fakas[0]++; //Und erstellen eine neue, imperfekte Familienkarte mit 2 Erwachsenen und einem Kind
        fakas[0]++; //Sowie eine mit 2 E, 2 K (Typ 1)
        rest_fakas[2] -= 2; //Jetzt sind die Erwachsenen verplant
    }
}
fakas[2] = rest_fakas[0];
fakas[3] = rest_fakas[1];
bester_besuch = new Besuch(rest_fakas[2] /*Erwachsene*/, rest_fakas[3] /*Kinder*/, fakas /*Familienkarten*/, 0 /*Tageskarten*/); //Besuch einspeichern in globaler Variable
}

public static boolean jaNeinFrage(String frage) { //Stellt eine jaNeinFrage
    while (true) {
        System.out.println(frage + "(j/n) ? ");
        String s = EINGABE.nextLine().toLowerCase();
        if (s.equals("j")) {
            return true;
        } else if (s.equals("n")) {
            return false;
        }
        System.out.println("Bitte antworten sie mit j/n beziehungsweise J/N. Versuchen sie es erneut.");
    }
}

public static int mengenFrage(String frage) { //Fragt nach einer ganzen Zahl > 0
    while (true) {
        System.out.println(frage + "(Zahl) ? ");
        String s = EINGABE.nextLine().toLowerCase();
        try {
            int so = Integer.parseInt(s);
            if (so >= 0) {
                return so;
            }
        } catch (Exception e) {
        }
        System.out.println("Bitte antworten sie mit einer ganzen Zahl > 0. Versuchen sie es erneut.");
    }
}

public static void karten() { //Verteilt Karten
    if (wochentag) { //Ist Wochentag ?
        kartenWochentag();
    } else { //Ansonsten
        kartenWochenende();
    }
}

public static void main(String[] args) {
    wochentag = jaNeinFrage("Wochentag");
    schultag = jaNeinFrage("Schultag");
    personen = mengenFrage("Wie viele Personen");
}

```

```

if (schultag) { //Nur an Schultagen
    gutscheine = mengenFrage("Wie viele Gutscheine"); //Sind Gutscheine relevant
}
kinder = mengenFrage("Wie viele sind Kinder(unter 16)"); //Kleinkinder eingeschlossen
int kleinkinder = mengenFrage("Wie viele davon sind Kleinkinder(unter 4)");
erwachsene = personen - kinder; //Alle Personen, die keine Kinder(Kleinkinder inklusive) sind, sind erwachsen
personen -= kleinkinder; //Kleinkinder sind irrelevant
kinder -= kleinkinder; //Kleinkinder sind irrelevant
if (erwachsene == 0 && kleinkinder != 0) { //Gibt es keine Erwachsenen, aber Kleinkinder
    System.out.println("Die " + Integer.toString(kleinkinder) + " Kleinkinder dürfen nicht ohne die Aufsicht einer Erwachsenen Person das Schwimmbad besuchen.");
    System.exit(0); //Wir sind fertig !
}
int kinderecht = kinder;
int erwachseneecht = erwachsene;
if (gutscheine >= personen) { // Gibt es mehr Gutscheine als Personen
    //Können einfach alle Gutscheine eingesetzt werden
    System.out.println("Setzen sie " + Integer.toString(personen) + " Gutscheine ein, und sie werden nichts bezahlen müssen.");
    System.exit(0); //Wir sind fertig !
}
if (gutscheine > 0) { //Gibt es mindestens einen Gutschein
    //Alle Möglichkeiten, die Gutscheine einzusetzen, generieren
    //Erstmal, falls ein Gutschein für die Ermässigung von 10 % eingesetzt werden soll
    Besuch resultat = null;
    if (gutscheine > 1) { //Wenn es genau einen Gutschein gibt, und dieser für 10 % Ermässigung eingesetzt wird, brauchen wir keine "Einsatzmöglichkeiten" für 0 Gutscheine generieren
        for (int i = 0; i < gutscheine; i++) {
            kinder = kinderecht;
            erwachsene = erwachseneecht;
            int ge = (gutscheine - 1) - i;
            kinder -= i;
            erwachsene -= ge;
            erwachsene = Math.max(erwachsene, 0);
            kinder = Math.max(kinder, 0);
            personen = kinder + erwachsene;
            karten();
            if (resultat == null || bester_besuch.preis < resultat.preis) { //Gibt es noch keinen Rekordhalter, oder ist diese Variante günstiger
                //Ist diese nun der "Rekordhalter"
                resultat = bester_besuch;
                resultat.gutscheine_kinder = i;
                resultat.gutscheine_erwachsene = ge;
            }
        }
    }
    else {
        karten();
    }
    resultat.setErmassigung(); //Ermässigung
    //Und ohne einen Gutschein für die Ermässigung von 10 % einzusetzen
    for (int i = 0; i <= gutscheine; i++) {
        kinder = kinderecht;
        erwachsene = erwachseneecht;
    }
}

```

```

    int ge = gutscheine - i;
    kinder -= i;
    erwachsene -= ge;
    erwachsene = Math.max(erwachsene, 0);
    kinder = Math.max(kinder, 0);
    personen = kinder + erwachsene;
    karten();
    if (bester_besuch.preis < resultat.preis) { //Ist diese Variante günstiger
        //Ist diese nun der "Rekordhalter"
        resultat = bester_besuch;
        resultat.gutscheine_kinder = i;
        resultat.gutscheine_erwachsene = ge;
    }
}
System.out.println(resultat); //Ergebnis ausgeben
System.exit(0); //Wir sind fertig !
}
karten(); //Gibt es keine Gutscheine, können wir einfach normal Karten kaufen
System.out.println(bester_besuch); //Ergebnis ausgeben
}
}
}

```

Besuch.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package schwimmbad;
/**
 *
 * @author lars
 */
public class Besuch { //Besuch, speichert alles, was wir über einen Besuch wissen wollen

    public float preis;
    public int einzelkarten_kinder;
    public int einzelkarten_erwachsene;
    public int gutscheine_kinder;
    public int gutscheine_erwachsene;
    public boolean ermaessigung;
    public int[] familienkarten;
    public int tageskarten;

    public Besuch(int einzelkarten_erwachsene, int einzelkarten_kinder, int[] familienkarten, int tageskarten) {
        this.einzelkarten_kinder = einzelkarten_kinder;
        this.einzelkarten_erwachsene = einzelkarten_erwachsene;
        this.familienkarten = new int[4];
        System.arraycopy(familienkarten, 0, this.familienkarten, 0, 4);
        this.tageskarten = tageskarten;
    }
}

```

```

float einzelkarten_preis=einzelkarten_kinder * 2.50f + einzelkarten_erwachsene * 3.50f;
if (Schwimmbad.wochentag) { //An Wochentagen
    einzelkarten_preis*=0.8f; //Ist der Einzelkartenpreis um 20 % reduziert
}
this.preis = einzelkarten_preis + (familienkarten[0] + familienkarten[1] + familienkarten[2] + familienkarten[3]) * 8.0f + tageskarten * 11.0f;
}

@Override
public String toString() { //ALLES ausgeben
    String s = "";
    s+="Besuch beim Schwimmbad - Kosten";
    s+="\n";
    s+="Familienkarten(2 Erwachsene, 2 Kinder) : "+Integer.toString(familienkarten[0]);
    s+="\n";
    s+="Familienkarten(1 Erwachsener, 3 Kinder) : "+Integer.toString(familienkarten[1]);
    s+="\n";
    s+="Familienkarten(2 Erwachsene, 1 Kind) : "+Integer.toString(familienkarten[2]);
    s+="\n";
    s+="Familienkarten(1 Erwachsener, 2 Kinder) : "+Integer.toString(familienkarten[3]);
    s+="\n";
    s+="Tageskarten : "+Integer.toString(tageskarten);
    s+="\n";
    s+="Einzelkarten Erwachsene : "+Integer.toString(einzelkarten_erwachsene);
    s+="\n";
    s+="Einzelkarten Kinder : "+Integer.toString(einzelkarten_kinder);
    s+="\n";
    s+="Es werden "+Integer.toString(gutscheine_erwachsene)+" Gutscheine für den freien Eintritt von Erwachsenen benutzt.";
    s+="\n";
    s+="Es werden "+Integer.toString(gutscheine_kinder)+" Gutscheine für den freien Eintritt von Kindern benutzt.";
    s+="\n";
    s+="Es wird ";
    if (ermaessigung) {
        s+="ein";
    }
    else {
        s+="kein";
    }
    s+=" Gutscheine für die Ermäßigung von 10 % eingesetzt.";
    s+="\n";
    s+="Preis : "+Float.toString(preis);
    return s;
}

public void setErmaessigung() {
    ermaessigung=true;
    this.preis*=0.9f; //-10 %
}
}

```

3. Aufgabe „Dreiecke zählen“

Lösungsidee :

1. Wir müssen ermitteln, wo zwei Strecken sich schneiden. Dazu ermitteln wir zuerst, in welchem Punkt sich die Geraden der Strecken schneiden würden. Liegt dieser Punkt auf beiden Strecken, schneiden sich die Strecken in dem ermittelten Punkt.

Jede Gerade lässt sich in die folgende Form bringen :

$$y = mx + n$$

Nun haben wir zwei solche Geraden. Wir setzen (I) = (II) :

$$m_1x_1 + n_1 = m_2x_2 + n_2$$

Durch umformen ergibt sich :

$$m_1x_1 - m_2x_2 = n_2 - n_1$$

2. Ein Dreieck besteht, wenn eine Strecke a vorhanden ist, welche eine Strecke b schneidet, wobei Gerade a und b beide noch zusätzlich eine Strecke c schneiden müssen. Vorausgesetzt wird, dass alle Schnittpunkte verschieden sind, also nicht 2-3 von ihnen identisch sind.

3. Zwei Dreiecke entsprechen einander, wenn sie aus den gleichen Strecken bestehen, egal in welcher Reihenfolge diese angegeben sind.

Nun müssen wir für alle Strecken (2) prüfen, und dann gefundene Dreiecke aufschreiben. Allerdings prüfen wir (2) nicht mehr für Strecken, die schon als gewesen sind, um Strecken nicht mehrmals aufzuschreiben.

Umsetzung :

Bibliotheken :

- java.io.File : Dateien
- java.io.FileNotFoundException : Datei-nicht-gefunden-Fehler
- java.io.FileReader : Liest Textdateien ein
- java.io.FileWriter : Schreibt in Dateien
- java.io.BufferedWriter : Schreibt in Dateien
- java.io.IOException : Eingabe-Ausgabe-Fehler
- java.util.ArrayList : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können
- java.util.Scanner : Benutzereingabe, kann direkt aus der Befehlszeile lesen

- java.util.HashMap : „Wörterbuch“ : Jedem Schlüssel „s“ wird ein Wert „w“ zugeordnet. Sehr schnell wegen binärer Suche nach Hash.
- java.util.Map.Entry : „Einträge“ eines „Wörterbuches“. Enthalten je einen Schlüssel und den ihm zugeordneten Wert.
- java.util.Set : „Menge“ der „Einträge“ eines Wörterbuches

Datenstrukturen :

Punkt : Speichert einen 2D-Punkt
 Strecke : Speichert eine Strecke zwischen zwei Punkten, sowie ein umfassendes Rechteck. Bietet eine Funktion, die den Schnitt zweier Strecken berechnet, und einen Schnittpunkt zurückgibt.
 Rechteck : Speichert ein Rechteck
 Schnittpunkt : Ein Punkt, der zusätzlich noch speichert, ob ein Schnitt vorliegt

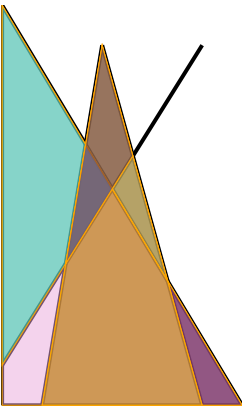
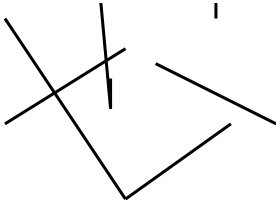
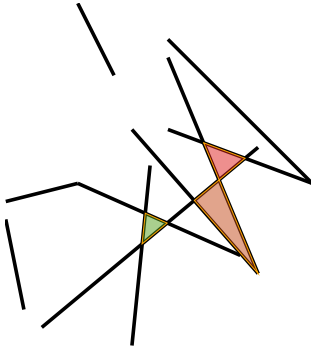
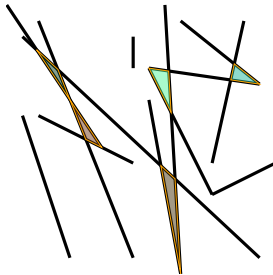
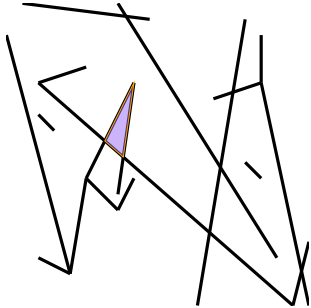
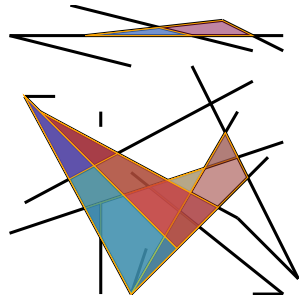
Programmstruktur :

Wir implementieren die folgenden Funktionen : leseDatei, dateiFrage, und textFrage.
 dateiFrage fragt den Benutzer nach dem Pfad zu einer Datei. Dies tut es so lange, bis der Benutzer einen validen Pfad angegeben hat.
 leseDatei liest diese Datei ein. Dies ist die Aufgabe. Wir separieren sie nach Zeilenumbrüchen.
 Mit textFrage wird vom Nutzer erfragt, wo die veranschaulichende Vektorgrafik gespeichert werden soll.
 Nun kommen wir zum Herzstück des Programms. Dies sind 3 ineinander geschachtelte for-Schleifen. Die erste zählt die Indexes aller Strecken durch, die zweite fängt schon beim Index der Ersten an, und die Dritte fängt beim Index der Zweiten an. So wird kein Dreieck mehrfach gezählt. Dann prüfen wir, ob (2) aus der Lösungsidee für die Strecken mit a=Index 1. Schleife, b=Index 2. Schleife, c=Index 3. Schleife, gegeben ist. Falls ja, geben wir das Dreieck aus, und zeichnen es in der Vektorgrafik ein. Außerdem zeichnen wir in der 1. Schleife alle Strecken in die Grafik ein. Wir speichern in den Strecken jeweils eine HashMap, welche Schnitte sie mit anderen Strecken aufweisen, um diese nicht immer neu berechnen zu müssen. Ist ein Schnitt noch nicht verzeichnet, berechnen wir ihn, und tragen ihn dann bei beiden Strecken ein. Nachdem alle Schleifen durchgelaufen sind, speichern wir die Vektorgrafik und geben die Zahl der gefundenen Dreiecke aus.

Beispiele :

/res/dreiecke1.txt	/res/dreiecke2.txt	/res/dreiecke3.txt	/res/dreiecke4.txt	/res/dreiecke5.txt	/res/dreiecke6.txt
Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/dreiecke1.txt Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg) (Zeichenkette) ? dreiecke1_loesungen.svg	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/dreiecke2.txt Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg) (Zeichenkette) ? dreiecke2_loesungen.svg	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/dreiecke3.txt Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg) (Zeichenkette) ? dreiecke3_loesungen.svg	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/dreiecke4.txt Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg) (Zeichenkette) ? dreiecke4_loesungen.svg	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/dreiecke5.txt Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg) (Zeichenkette) ? dreiecke5_loesungen.svg	Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ? res/dreiecke6.txt Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg) (Zeichenkette) ? dreiecke6_loesungen.svg

<p>Neues Dreieck : A=(0 0), B=(120 0), C=(0 200)</p> <p>Neues Dreieck : A=(0 200), B=(55.102041 108.163265), C=(0 20)</p> <p>Neues Dreieck : A=(120 0), B=(41.73913 130.434783), C=(20 0)</p> <p>Neues Dreieck : A=(120 0), B=(82.758621 62.068966), C=(100 0)</p> <p>Neues Dreieck : A=(20 0), B=(50 180), C=(100 0)</p> <p>Neues Dreieck : A=(55.102041 108.163265), B=(31.818182 70.909091), C=(41.73913 130.434783)</p> <p>Neues Dreieck : A=(55.102041 108.163265), B=(65.384615 124.615385), C=(82.758621 62.068966)</p> <p>Neues Dreieck : A=(41.73913 130.434783), B=(50 180), C=(82.758621 62.068966)</p> <p>Neues Dreieck : A=(31.818182 70.909091), B=(50 180), C=(65.384615 124.615385)</p> <p>Es konnten 9 Dreiecke gefunden werden.</p>	<p>Es konnten 0 Dreiecke gefunden werden.</p>	<p>Neues Dreieck : A=(104.578313 80.481928), B=(118.350515 91.958763), C=(140 40)</p> <p>Neues Dreieck : A=(75.636364 56.363636), B=(77.340426 73.404255), C=(89.565217 67.971014)</p> <p>Neues Dreieck : A=(132.758621 103.965517), B=(109.753086 112.592593), C=(118.350515 91.958763)</p> <p>Es konnten 3 Dreiecke gefunden werden.</p>	<p>Neues Dreieck : A=(60 110), B=(38.823529 141.764706), C=(52.340426 129.148936)</p> <p>Neues Dreieck : A=(60 110), B=(80 80), C=(70 85)</p> <p>Neues Dreieck : A=(117.591241 68.248175), B=(126.473029 59.958506), C=(130 0)</p> <p>Neues Dreieck : A=(124 102), B=(122.457627 128.220339), C=(110 130)</p> <p>Neues Dreieck : A=(161.692308 122.615385), B=(180 120), C=(163.962264 132.830189)</p> <p>Es konnten 5 Dreiecke gefunden werden.</p>	<p>Neues Dreieck : A=(61.73913 103.478261), B=(80 140), C=(73.333333 93.333333)</p> <p>Es konnten 1 Dreiecke gefunden werden.</p>	<p>Neues Dreieck : A=(91.764706 70.588235), B=(111.818182 54.545455), C=(128.275862 82.758621)</p> <p>Neues Dreieck : A=(128.275862 82.758621), B=(118.926829 66.731707), C=(104.117647 74.705882)</p> <p>Neues Dreieck : A=(128.275862 82.758621), B=(80 0), C=(49.565217 56.521739)</p> <p>Neues Dreieck : A=(128.275862 82.758621), B=(141.923077 106.153846), C=(150 90)</p> <p>Neues Dreieck : A=(128.275862 82.758621), B=(102.105263 37.894737), C=(75 65)</p> <p>Neues Dreieck : A=(104.117647 74.705882), B=(20 120), C=(75 65)</p> <p>Neues Dreieck : A=(49.565217 56.521739), B=(10 130), C=(75 65)</p> <p>Neues Dreieck : A=(49.565217 56.521739), B=(60 37.142857), C=(60 60)</p> <p>Neues Dreieck : A=(111.818182 54.545455), B=(80 0), C=(124.444444 44.444444)</p> <p>Neues Dreieck : A=(80 0), B=(137 57), C=(118.926829 66.731707)</p> <p>Neues Dreieck : A=(80 0), B=(156.666667 76.666667), C=(141.923077 106.153846)</p> <p>Neues Dreieck : A=(80 0), B=(110 30), C=(102.105263 37.894737)</p> <p>Neues Dreieck : A=(118.926829 66.731707), B=(20 120), C=(102.105263 37.894737)</p> <p>Neues Dreieck : A=(80 0), B=(10 130), C=(102.105263 37.894737)</p> <p>Neues Dreieck : A=(137 57), B=(20 120), C=(110 30)</p> <p>Neues Dreieck : A=(80 0), B=(10 130), C=(110 30)</p> <p>Neues Dreieck : A=(71.004785 92.535885), B=(55.652174 84.347826), C=(20 120)</p> <p>Neues Dreieck : A=(39.282869 75.61753), B=(55.652174 84.347826), C=(10 130)</p> <p>Neues Dreieck : A=(160 170), B=(50 170), C=(140 180)</p>
---	---	--	--	---	--

					Neues Dreieck : A=(50 170), B=(98.461538 175.384615), C=(120 170) Es konnten 20 Dreiecke gefunden werden.
					

Quellcode :

DreieckeZaehlen.java :

```

package dreieckezaehlen;

//Dateien & Ordner
import java.io.BufferedWriter; //Dateien schreiben
import java.io.File; //Dateien & Ordner
import java.io.FileNotFoundException; //Datei-nicht-gefunden-Fehler
import java.io.FileReader; //Dateien lesen
import java.io.FileWriter; //Dateien schreiben
import java.io.IOException; //Eingabe/Ausgabe-Fehler
import java.util.Scanner; //Benutzereingabe

public class DreieckeZaehlen {

    public static final Scanner EINGABE=new Scanner(System.in); //Benutzereingabe
    public static Strecke[] strecken; //Alle Strecken

    public static String leseDatei(File pfad_zur_datei) throws FileNotFoundException, IOException { //Liest eine Datei ein, und gibt Text zurück
        FileReader datei = new FileReader(pfad_zur_datei);
  
```

```

String r = "";
int i = datei.read();
while (i != -1) {
    r += (char) i;
    i = datei.read();
}
return r;
}

public static File dateiFrage(String frage) { //Fragt nach einem Pfad und prüft, ob dieser existiert
    while (true) {
        System.out.println(frage + "(Pfad zu .txt Datei) ? ");
        String s = EINGABE.nextLine();
        File f = new File(s);
        if (f.exists() && !f.isDirectory() && f.canRead()) { //Prüfe, ob : - existiert die Datei - ist es kein Ordner - ist sie lesbar
            return f; //Gebe Pfad zurück
        }
        System.out.println("Bitte antworten sie mit einem vorhandenen Pfad einer .txt Datei. Versuchen sie es erneut.");
    }
}

public static String textFrage(String frage) { //Fragt nach einer Zeichenkette
    System.out.println(frage + "(Zeichenkette) ? ");
    String s = EINGABE.nextLine();
    return s;
}

public static Schnittpunkt holeSchnittpunkt(Strecke e, Strecke f) { //Gibt den Schnittpunkt zweier Strecken zurück
    Schnittpunkt c = e.schnitte.get(f); //Ist der Schnittpunkt gespeichert
    if (c == null) { //Falls leider nicht, müssen wir ihn ermitteln
        c = f.schnittpunkt(e); //Ermittelt den Schnittpunkt
        //Den Schnitt einspeichern
        e.schnitte.put(f, c);
        f.schnitte.put(e, c);
    }
    return c; //Schnittpunkt zurückgeben
}

public static void main(String[] args) throws IOException {
    String[] aufgabe = leseDatei(dateiFrage("Wo befindet sich die Aufgabe")).split("\n");
    String ausgabe = textFrage("Wo soll das Ausgabebild gespeichert werden(Pfad zu .svg)");
    int d = 0; //Noch wurden 0 Dreiecke gezählt
    strecken = new Strecke[aufgabe.length - 1]; //Es gibt so viele Strecken, wie die Aufgabe Zeilen hat abzüglich 1, wegen der obersten Zeile
    for (int i = 1; i < aufgabe.length; i++) { //Alle Strecken von Textform in Streckenform umwandeln
        String s = aufgabe[i];
        String[] p = s.split(" ");
        Punkt a = new Punkt(new Dezimalzahl(Double.parseDouble(p[0])), new Dezimalzahl(Double.parseDouble(p[1]))); //Punkt a lesen
        Punkt b = new Punkt(new Dezimalzahl(Double.parseDouble(p[2])), new Dezimalzahl(Double.parseDouble(p[3]))); //Punkt b lesen
        strecken[i - 1] = new Strecke(a, b); //Strecke speichern
    }
    double minx = Double.MAX_VALUE; //Kleinster X-Wert von allen Strecken
}

```

```

double miny = Double.MAX_VALUE; //Kleinsten Y-Wert von allen Strecken
double maxx = Double.MIN_VALUE; //Größter X-Wert von allen Strecken
double maxy = Double.MIN_VALUE; //Größter Y-Wert von allen Strecken
for (Strecke s : strecken) { //Ermittelt die darüber aufgeführten Werte
    double ax = s.a.x.wert();
    double ay = s.a.y.wert();
    double bx = s.b.x.wert();
    double by = s.b.y.wert();
    if (ax > maxx) {
        maxx = ax;
    }
    if (ax < minx) {
        minx = ax;
    }
    if (ay > maxy) {
        maxy = ay;
    }
    if (ay < miny) {
        miny = ay;
    }
    if (bx > maxx) {
        maxx = bx;
    }
    if (bx < minx) {
        minx = bx;
    }
    if (by > maxy) {
        maxy = by;
    }
    if (by < miny) {
        miny = by;
    }
}
double dimx = Math.abs(maxx - minx); //Breite des Bildes
double dimy = Math.abs(maxy - miny); //Höhe des Bildes
//SVG Header
String svg = "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\" ?>\n"
    + "<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG 20010904//EN\" \"\n"
    + "\"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd\">\n"
    + "<svg xmlns=\"http://www.w3.org/2000/svg\" \n"
    + "    xmlns:xlink=\"http://www.w3.org/1999/xlink\" \n"
    + "    version=\"1.1\" baseProfile=\"full\" \n"
    + String.format("    width=\"%fpx\" height=\"%fpx\" \n", dimx, dimy)
    + String.format("    viewBox=\"%f %f %f %f\">\n", minx, miny, maxx, maxy);
String lines = ""; //Linien als SVG-Elemente
String triangles = ""; //Dreiecke als SVG-Elemente
for (int i = 0; i < strecken.length; i++) { //Alle Strecken durchgehen
    Strecke a = strecken[i];
    //Strecke in der Vektorgrafik darstellen
    double[] xl = new double[]{a.a.x.wert(), a.b.x.wert()}; //Punkte a und b der Strecke, X-Koordinaten
    double[] yl = new double[]{dimy - a.a.y.wert(), dimy - a.b.y.wert()}; //Punkte a und b der Strecke, Y-Koordinaten, gespiegelt

```

```
//Koordinaten in SVG-gerechten Text umwandeln
String ls = ""; //Koordinaten als Text
for (int l = 0; l < 2; l++) {
    ls += Double.toString(xl[l]);
    ls += ","; //Immer ein Kommata zwischen ein Koordinatenpaar
    ls += Double.toString(yl[l]);
    if (l != 1) {
        ls += " ";
    }
}
lines += "\n   <polyline points=\"" + ls + "\" style=\"fill:rgb(0,0,0); stroke:rgb(0,0,0); stroke-width:2px\" />"; //Linie im SVG speichern
for (int j = i + 1; j < strecken.length; j++) { //Alle Strecken durchgehen, von i ausgehend, da i Strecken alle schon "benutzt" worden sind
    Strecke b = strecken[j];
    Schnittpunkt A = holeSchnittpunkt(a, b); //Schnittpunkt zwischen a und b ermitteln...
    if (A.schnitt) { //...schneiden die sich überhaupt ?
        for (int k = j + 1; k < strecken.length; k++) { //Falls ja, kann Ausschau nach einer Strecke c, die das Dreieck vervollständigt, gehalten werden, von j ausgehend, da für dieses Element j Strecken alle schon "benutzt" worden sind, Die benutzten i Strecken sind schon enthalten
            Strecke c = strecken[k];
            Schnittpunkt B = holeSchnittpunkt(c, b); //Schnittpunkt zwischen b und c ermitteln...
            if (B.schnitt && !A.e(B)) { //...schneiden die sich überhaupt, und : sind A und B verschiedene Punkte ?
                Schnittpunkt C = holeSchnittpunkt(c, a); //Schnittpunkt C-A : Nötig, damit das Dreieck geschlossen ist
                if (C.schnitt && !A.e(C) && !B.e(C)) { //...ist es das überhaupt, und : sind A und C bzw. B und C verschiedene Punkte ?
                    //Dreieck im SVG speichern
                    double[] x = new double[]{A.x.wert(), C.x.wert(), B.x.wert()}; //Eckpunkte des Dreiecks, X-Koordinaten
                    double[] y = new double[]{dimy - A.y.wert(), dimy - C.y.wert(), dimy - B.y.wert()}; //Eckpunkte des Dreiecks, Y-Koordinaten, gespiegelt
                    //Koordinaten in SVG-gerechten Text umwandeln
                    String ps = ""; //Koordinaten als Text
                    for (int l = 0; l < 3; l++) {
                        ps += Double.toString(x[l]);
                        ps += ","; //Immer ein Kommata zwischen ein Koordinatenpaar
                        ps += Double.toString(y[l]);
                        if (l != 2) {
                            ps += " ";
                        }
                    }
                    String color = String.format("rgb(%s,%s,%s)", (int) (Math.random() * 255), (int) (Math.random() * 255), (int) (Math.random() * 255)); //Zufällige Farbe
                    triangles += "\n   <polygon points=\"" + ps + "\" style=\"fill:" + color + "; fill-opacity:0.5; stroke:rgb(255,165,0); stroke-width:0.5px\" />"; //Dreieck im SVG speichern
                    System.out.println("Neues Dreieck : A=" + A.toString() + ", B=" + B.toString() + ", C=" + C.toString()); //Dreieck(dessen Eckpunkte) ausgeben
                    d++; //Ein weiteres Dreieck !
                }
            }
        }
    }
}
svg += lines; //Linien im SVG speichern
svg += triangles; //Dreiecke im SVG speichern
svg += "\n</svg>";
//Neue Datei erzeugen
File f = new File(ausgabe);
```

```

f.createNewFile();
//SVG speichern
BufferedWriter w = new BufferedWriter(new FileWriter(f));
w.write(svg);
w.close();
System.out.println("Es konnten " + Integer.toString(d) + " Dreiecke gefunden werden."); //Anzahl der gefundenen Dreiecke ausgeben
}
}

```

Strecke.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package dreieckezaehlen;

import java.util.HashMap; // "Wörterbücher" : Jedem "Schlüssel" s wird ein "Wert" w zugeordnet

/**
 *
 * @author lars
 */
public class Strecke { //Speichert eine Strecke

    public HashMap<Strecke, Schnittpunkt> schnitte; //Schnittpunkt mit anderen Strecken
    public Rechteck r; //Umfassendes Rechteck
    public Punkt a, b; //Startpunkt/Endpunkt
    public Dezimalzahl steigungX; //Steigung per x-Wert
    public Dezimalzahl steigungY; //Steigung per y-Wert
    public Dezimalzahl Y; //Y-Achsenabschnitt
    public Dezimalzahl X; //X-Achsenabschnitt
    public boolean waagerecht; //Speichert für Strecken ohne Steigung die Ausrichtung
    public boolean sonderfall; //Speichert, ob diese Strecke solch eine ist

    public Strecke(Punkt a, Punkt b) { //Konstruktor - Start/Endpunkt werden vorgegeben
        schnitte = new HashMap();
        r = new Rechteck(a, b); //Umfassendes Rechteck erzeugen
        this.a = a;
        this.b = b;
        Punkt temp = b.minus(a); //b-a=(delta x|delta y)
        if (!temp.x.isNull() && !temp.y.isNull()) { //Falls diese kein Sonderfall ist(nicht waagerecht oder senkrecht)
            sonderfall = false; //Kein Sonderfall
            steigungX = temp.y.geteilt(temp.x); //Steigung X bestimmen
            Y = this.a.y.minus(this.a.x.mal(steigungX)); //X-Achsenabschnitt berechnen
            steigungY = temp.x.geteilt(temp.y); //Steigung Y bestimmen
            X = this.a.x.minus(this.a.y.mal(steigungY)); //Y-Achsenabschnitt berechnen
        } else {
            sonderfall = true; //Ansonsten ist es ein Sonderfall, dann gilt es nur zu ermitteln, ob waagerecht oder senkrecht
        }
    }
}

```

```

    steigungX = null;
    steigungY = null;
    X = null;
    Y = null;
    waagerecht = !temp.x.isNull(); //Sollte die x-Achsendifferenz null sein, ist die Strecke senkrecht also !waagerecht
}
}

```

```

public Schnittpunkt schnittpunkt(Strecke s) { //Ermittelt den Schnittpunkt zweier Strecken, und damit auch, ob sie sich schneiden
    Dezimalzahl schnitt_x, schnitt_y; //Schnitt X-und Y-Koordinate
    schnitt_x = null;
    schnitt_y = null;
    if (!s.sonderfall && !this.sonderfall) { //Sollten beide Strecken keine Sonderfälle sein
        if (s.steigungY.e(this.steigungY)) { //Sind die Steigungen identisch
            return new Schnittpunkt(); //Gibt es keinen Schnittpunkt
        }
        Dezimalzahl steigungsdifferenz = this.steigungX.minus(s.steigungX); //Formel : Differenz der Steigungen
        Dezimalzahl offsetdifferenz = s.Y.minus(this.Y); //Formel : Differenz der Y-Achsenabschnitte
        schnitt_x = offsetdifferenz.geteilt(steigungsdifferenz); //Formel : Schnittpunkt X berechnen
        schnitt_y = this.Y.plus(schnitt_x.mal(this.steigungX)); //Schnittpunkt X einsetzen in eine der beiden Geradengleichungen
    } else if (s.sonderfall && this.sonderfall) { //Sollten beide Strecken Sonderfälle sein
        if (s.waagerecht != this.waagerecht) { //Sind sie nicht parallel zueinander
            if (s.waagerecht) { //Ist s waagerecht ?
                schnitt_y = s.a.y; //Die Y-Koordinate des Schnittes ist dann von s
                schnitt_x = this.a.x; //Entsprechend ist die X-Koordinate von dieser
            } else { //Sollte es andersrum sein...
                //Werden auch die Koordinaten des Schnittes andersrum belegt
                schnitt_x = s.a.x;
                schnitt_y = this.a.y;
            }
        } else { //Wenn sie parallel zueinander sind, schneiden sie sich nicht
            return new Schnittpunkt();
        }
    } else if (s.sonderfall) { //Ist nur s ein Sonderfall
        if (s.waagerecht) { //Ist s waagerecht
            schnitt_y = s.a.y; //Ist die Y-Koordinate des Schnittpunkts klar von s
            Dezimalzahl offsetdifferenz = schnitt_y.minus(this.Y);
            schnitt_x = offsetdifferenz.geteilt(this.steigungX); //Nach der Formel wird dann die X-Koordinate ermittelt
        } else {
            schnitt_x = s.a.x; //Ansonsten ist es die X-Koordinate, welche von s ist
            Dezimalzahl offsetdifferenz = schnitt_x.minus(this.X);
            schnitt_y = offsetdifferenz.geteilt(this.steigungY); //Nach der Formel wird dann die Y-Koordinate ermittelt
        }
    } else if (this.sonderfall) { //Sollte diese Gerade von beiden der Sonderfall sein, geschieht das Selbe wie oben; lediglich s und diese Gerade müssen vertauscht werden
        if (this.waagerecht) {
            schnitt_y = this.a.y;
            Dezimalzahl offsetdifferenz = schnitt_y.minus(s.Y);
            schnitt_x = offsetdifferenz.geteilt(s.steigungX);
        } else {
            schnitt_x = this.a.x;
            Dezimalzahl offsetdifferenz = schnitt_x.minus(s.X);

```

```

        schnitt_y = offsetdifferenz.getteilt(s.steigungY);
    }
}
Punkt p = new Punkt(schnitt_x, schnitt_y);
if (s.r.schneidetPunkt(p) && this.r.schneidetPunkt(p)) { //Ist der Schnittpunkt "valid", liegt er also auf beiden Strecken
    return new Schnittpunkt(p.x, p.y); //Kann er zurückgegeben werden
}
return new Schnittpunkt(); //Sonst liegt wohl kein Schnitt vor...
}
}
}

```

Dezimalzahl.java :

```

/*
 * To change value license header, choose License Headers in Project Properties.
 * To change value template file, choose Tools | Templates
 * and open the template in the editor.
 */
package dreieckezaehlen;

import java.math.BigDecimal; //Dezimalzahlen mit vielen Nachkommastellen
import java.text.DecimalFormat; //Textausgabe

/**
 *
 * @author lars
 */
public class Dezimalzahl {

    public static final Dezimalzahl ZERO=new Dezimalzahl(new BigDecimal("0")); //Dezimalzahl 0
    public static final DecimalFormat DARSTELLUNG=new DecimalFormat("#.#####"); //Ausgabeformat/Darstellung : 6 Nachkommastellen
    public BigDecimal wert; //Wert der Dezimalzahl

    public Dezimalzahl(double w) {
        wert=new BigDecimal(w);
    }

    public Dezimalzahl(BigDecimal w) {
        wert=w;
    }

    public boolean isNull() { //Ermittelt, ob diese Dezimalzahl 0 ist
        return this.wert().compareTo(BigDecimal.ZERO) == 0;
    }

    public boolean e(Dezimalzahl b) { //Ermittelt, ob zwei Dezimalzahlen einander entsprechen
        return this.wert().compareTo(b.wert())==0;
    }
}

```

```

public BigDecimal w() { //Gerundet auf 12 Nachkommastellen
    return wert.setScale(12, BigDecimal.ROUND_HALF_UP);
}

public Dezimalzahl minus() { //Vorzeichen umdrehen
    return new Dezimalzahl(wert.negate());
}

public boolean gg(Dezimalzahl b) { //Ist diese Dezimalzahl größer-gleich b ?
    return this.w().compareTo(b.w()) >= 0;
}

public boolean kg(Dezimalzahl b) { //Ist diese Dezimalzahl kleiner-gleich b ?
    return this.w().compareTo(b.w()) <= 0;
}

public double wert() { //Gibt diese Dezimalzahl im "double"-Format zurück
    return this.w().doubleValue();
}

public Dezimalzahl plus(Dezimalzahl b) {
    return new Dezimalzahl(wert.add(b.wert));
}

public Dezimalzahl minus(Dezimalzahl b) {
    return new Dezimalzahl(wert.subtract(b.wert));
}

public Dezimalzahl mal(Dezimalzahl b) {
    return new Dezimalzahl(wert.multiply(b.wert));
}

public Dezimalzahl geteilt(Dezimalzahl b) {
    return new Dezimalzahl(wert.divide(b.wert, 64, BigDecimal.ROUND_HALF_UP)); //Mit 64 Nachkommastellen Genauigkeit rechnen
}

public static Dezimalzahl abs(Dezimalzahl b) { //Absolutwert von einer Dezimalzahl b
    return new Dezimalzahl(b.wert.abs());
}

public static Dezimalzahl min(Dezimalzahl g, Dezimalzahl b) { //Kleinster Wert von b und g
    return new Dezimalzahl(g.wert.min(b.wert));
}

public static Dezimalzahl max(Dezimalzahl g, Dezimalzahl b) { //Größter Wert von b und g
    return new Dezimalzahl(g.wert.max(b.wert));
}

@Override
public String toString() { //Darstellung : 6 Nachkommastellen
    String s=DARSTELLUNG.format(this.w()); //Mit 6 Nachkommastellen anzeigen
    if (s.length() == 2 && s.charAt(0) == '-' && s.charAt(1) == '0') { //-0 wird zu 0
        return "0";
    }
    return s;
}
}

```


Rechteck.java :

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package dreieckezaehlen;

/**
 *
 * @author lars
 */
public class Rechteck {
    public Dezimalzahl x;
    public Dezimalzahl y;
    public Dezimalzahl w;
    public Dezimalzahl h;
    public Punkt[] eckpunkte;
    public Rechteck(Dezimalzahl x, Dezimalzahl y, Dezimalzahl w, Dezimalzahl h) {
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
        eckpunkte=new Punkt[] {
            new Punkt(x,y),
            new Punkt(x.plus(w),y),
            new Punkt(x.plus(w),y.plus(h)),
            new Punkt(x.y.plus(h)),
        };
    }
    public Rechteck(Punkt a, Punkt b) {
        this.w=Dezimalzahl.abs(a.x.minus(b.x));
        this.h=Dezimalzahl.abs(a.y.minus(b.y));
        this.x=Dezimalzahl.min(a.x,b.x);
        this.y=Dezimalzahl.min(a.y,b.y);
        eckpunkte=new Punkt[] {
            new Punkt(x,y),
            new Punkt(x.plus(w),y),
            new Punkt(x.plus(w),y.plus(h)),
            new Punkt(x.y.plus(h)),
        };
    }
    public boolean schneidetPunkt(Punkt p) {
        return p.x.gg(eckpunkte[0].x) && p.x.kg(eckpunkte[1].x) && p.y.gg(eckpunkte[0].y) && p.y.kg(eckpunkte[3].y);
    }
    @Override
    public String toString() {
        String s="X : "+x.toString()+"Y : "+y.toString()+" W : "+w.toString()+" H : "+h.toString();
    }
}
```

```

    }
    return s;
}

```

Punkt.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package dreieckezaehlen;

/**
 *
 * @author lars
 */
public class Punkt { //Speichert einen 2d-Punkt
    public Dezimalzahl x; //X-Koordinate
    public Dezimalzahl y; //Y-Koordinate
    public Punkt(Dezimalzahl x, Dezimalzahl y) { //Konstruktor-die Koordinaten werden vorgegeben
        this.x=x;
        this.y=y;
    }
    public boolean e(Punkt p) { //Entspricht dieser Punkt einem anderen Punkt ?
        return this.x.e(p.x) && this.y.e(p.y); //Sind X-und Y-Koordinate gleich ?
    }
    public Punkt minus(Punkt p) {
        return new Punkt(x.minus(p.x),y.minus(p.y)); //Zieht einen Punkt p von diesem Punkt ab
    }
    @Override
    public String toString() { //Textausgabe
        return "("+x.toString()+"|"+y.toString()+")";
    }
}

```

Schnittpunkt.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package dreieckezaehlen;

/**
 *
 * @author lars
 */

```

```

public class Schnittpunkt extends Punkt { //Speichert einen Schnitt zweier Strecken, erweitert die Klasse "Punkt"
    public boolean schnitt; //Schneiden sich die Strecken überhaupt ?
    public Schnittpunkt() { //Erzeugt einen Schnittpunkt, der speichert, dass kein Schnitt vorliegt
        super(null,null); //Keine X und Y-Koordinaten
        schnitt=false; //Kein Schnitt
    }
    public Schnittpunkt(Dezimalzahl x, Dezimalzahl y) { //Erzeugt einen Schnittpunkt, der einen vorhandenen Schnitt speichert
        super(x,y); //X-und Y-Koordinate speichern
        schnitt=true; //Ein Schnitt
    }
}

```

4. Aufgabe „Auto-Scrabble“

Lösungsidee :

1. + 2.)

Um zu prüfen, ob ein Wort auf ein Kennzeichen passt, muss berücksichtigt werden :

- Ein Kürzel muss schon am Wortanfang zu finden sein, so z.B. :
BIN → Städtekürzel „B“ für Berlin am Wortanfang
- Das Kürzel darf das Wort nicht vollkommen „einnehmen“, es muss mindestens 1 Zeichen Anhang bleiben :
BN → Städtekürzel „BN“ für Bonn nicht zulässig, da dann kein Anhang bliebe
- Der daraus folgende Anhang darf keinen Umlaut enthalten. Beispiel :
BUÄ → nicht möglich, da „Ä“ ein Umlaut ist und somit nicht im Anhang stehen darf
- Der Anhang darf maximal 2 Zeichen lang sein. Bsp :
BINA → Kennzeichen B : INA nicht möglich, da INA 3 Buchstaben sind.

Da Städtekürzel maximal 3 Zeichen lang sind, und Anhänge maximal 2 Zeichen lang sind, sind Wörter mit mehr als 5 Stellen nicht schreibbar.

- Nach diesen Regeln ist TIMO also nicht mit einem Kennzeichen schreibbar.

3. + 4.)

Um zu prüfen, ob ein Wort sich aus Kennzeichen, generieren wir rekursiv alle Möglichkeiten, wie wir verfahren können. Sobald wir in einen „Deadlock“ kommen, oder eine Lösung gefunden worden ist, wird die Möglichkeit nicht mehr weiter generiert. An einem Beispiel :

BNOX

Möglichkeit 1 :

Kürzel „B“ verwenden.

Mit zwei Buchstaben Anhang : B|NO, ein Buchstabe bleibt übrig → Deadlock

Mit einem Buchstaben Anhang : B|N, zwei Buchstaben bleiben übrig, in OB passt kein Kennzeichen mehr → Deadlock

Möglichkeit 2 :

Kürzel „BN“ verwenden.

Mit einem Buchstaben Anhang : BN|O, ein Buchstabe bleibt übrig → Deadlock

Mit zwei Buchstaben Anhang : BN|OX, kein Buchstabe bleibt übrig → Fertig !

Umsetzung :

Bibliotheken :

- java.io.File : Dateien
- java.io.FileNotFoundException : Datei-nicht-gefunden-Fehler
- java.io.FileReader : Liest Textdateien ein
- java.io.IOException : Eingabe-Ausgabe-Fehler
- java.util.ArrayList : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können
- java.util.HashMap : „Wörterbuch“ : Jedem Schlüssel „s“ wird ein Wert „w“ zugeordnet. Sehr schnell wegen binärer Suche nach Hash.
- java.util.Scanner : Benutzereingabe, kann direkt aus der Befehlszeile lesen

Datenstrukturen :

Kennzeichen : Speichert ein Kennzeichen, also Kürzel, Anhang und Nummer

Programmstruktur :

Wir implementieren, um Aufgabenteil 1 und 2 zu lösen, eine Funktion prüfe, die (1) alle Kombinationen mit n Buchstaben generiert, und (2) diese dann durch Aufruf einer Funktion „schreibbar“ darauf prüft, ob sie nicht schreibbar sind. Falls ja werden sie ausgegeben. Die Funktion „schreibbar“ ist wie in der Lösungsidee beschrieben implementiert. Diese Funktion rufen wir dann mit $n=2$, $n=3$, sowie falls erwünscht, $n=4$ auf.

Wir implementieren die folgenden Funktionen : leseDatei, dateiFrage, und jaNeinFrage.

dateiFrage fragt den Benutzer nach dem Pfad zu einer Datei. Dies tut es so lange, bis der Benutzer einen validen Pfad angegeben hat.

leseDatei liest diese Datei ein.

Mit diesen Funktionen fragen wir einmal nach dem Pfad zu der Kürzelliste, und einmal nach dem Pfad zur Aufgabenstellung. Beides wird eingelesen.

Mit jaNeinFrage wird dem Nutzer die Frage gestellt, ob er alle Wörter mit 4 Buchstaben, die nicht mit einem Kennzeichen schreibbar sind, anzeigen lassen will. Dies ist nicht empfehlenswert, da diese sehr viele sind, und es je nach Rechner durchaus dauern kann, und mir auch nicht besonders sinnvoll erscheint.

Die Kürzelliste separieren wir einfach nach Zeilenumbrüchen, dann haben wir sie schon im Array-Format.

Die Aufgabe braucht ebenfalls nur nach Zeilenumbrüchen separiert werden, um sie ins Array-Format zu bringen.

Wir speichern die in einem Wort gefundenen Kürzel in einem „Wörterbuch“, also als HashMap. Schlüssel ist hierbei die Stelle im Wort, und Wert das an der Stelle anfangende Kürzel. Wir haben eine Funktion „findeKürzel“, welche diese HashMap mit gefundenen Kürzeln füllt.

Nun haben wir eine Funktion „schreibeWort“, die wie in der Lösungsidee erklärt vorgeht. Diese wird mit folgenden Parametern aufgerufen :

(1) Die bisherige Kennzeichenkombination und (2) Die Stelle im Wort, bei der man sich momentan befindet. Darin holt man aus der HashMap der im Wort gefunden Kürzel alle an der momentanen Stelle vorzufindenden. Für diese probieren wir dann jeweils beide Möglichkeiten aus : 2er Anhang / 1er Anhang, und rufen dann wieder diese Funktion aus, mit Stelle + Kürzellänge + 1/2(Anhang), und einem Kennzeichen mehr in der Kennzeichenkombination. Sollte ein Funktionsaufruf feststellen, dass er sich in einem Deadlock befindet, so hört er auf, und gibt „null“ zurück.

Ebenfalls kommt dies vor, wenn eine Lösung schon gefunden worden ist, was in einer globalen Variable gespeichert wird. Sollte ein Funktionsaufruf das Kennzeichen beendet kriegen, so reicht er die Lösung an den Funktionsaufruf, der ihn aufgerufen hat, weiter, etc. So kommt die Lösung am obersten Funktionsaufruf an, welcher diese dann zurückgibt. Sollte kein Aufruf das Kennzeichen beendet gekriegt haben, wir auch hier „null“ zurückgegeben. Das Resultat ist nun also die finale Kennzeichenkombination. Ist diese „null“, so geben wir „nicht schreibbar aus“. Ansonsten geben wir die Kennzeichenliste übersichtlich mit der Funktion gebeListeAus aus.

Beispiele :

1. + 2.)

TIMO ist nicht schreibbar, da es kein Kürzel TI, oder T, oder TIM gibt.

Mit vier Buchstaben ließ sich ÜBER ausmachen(es gibt kein Städtekürzel, dass mit Ü beginnt).

Mit drei Buchstaben konnte ich noch BÄR finden, da es kein Kennzeichen BÄ gibt, und Ä ja nicht im Anhang stehen darf.

Bei zwei Buchstaben gibt es das Wort OB, da es kein Städtekürzel O gibt.

3. + 4.)

Auszug aus der Programmausgabe :

```
Wo befindet sich die Liste aller Städtekürzel(Pfad zu .txt Datei) ?
/res/kuerzelliste.txt
```

```
-----
TIMO ist
nicht
schreibbar
```

```
-----
Nicht schreibbare Wörter mit zwei Buchstaben :
AA, AO, AU, BA, BO, BU, CA, CO, CU, DA, DO, DU, EA, EO, EU, FA, FO, FU
GA, GO, GU, HA, HO, HU, IA, IB, IC, ID, IE, IF, IG, IH, II, IJ, IK, IL
IM, IN, IO, IP, IQ, IR, IS, IT, IU, IV, IW, IX, IY, IZ, IÄ, IÖ, IÜ, JÄ
JÖ, JÜ, KÄ, KÖ, KÜ, LÄ, LÖ, LÜ, MÄ, MÖ, MÜ, NÄ, NÖ, NÜ, OA, OB, [...]
```

```
-----
Nicht schreibbare Wörter mit drei Buchstaben :
[...]
```

```
-----
Wollen sie alle Kombinationen von Wörtern mit 4 Buchstaben, die nicht mit einem Kennzeichen schreibbar sind, anzeigen lassen(j/n) ?
n
```

```
Wo befindet sich die Aufgabe(Pfad zu .txt Datei) ?
/res/autoscrabble.txt
```

```
-----
BIBER
|B:I 1|
|B:ER 2|
```

```
-----
BUNDESWETTBEWERB
|B:U 1|, |N:D 2|
|E:S 3|, |WE:TT 4|
|B:E 5|, |W:E 6|
|R:B 7|
```

```
-----
CLINTON
|C:LI 1|
|NT:ON 2|
```

```
-----
DONAUDAMPFSCHIFFFAHRTSKAPITÄNSMÜTZE
```

Nicht schreibbar

ETHERNET

|E:T 1|, |H:E 2|

|R:N 3|, |E:T 4|

INFORMATIK

|IN:FO 1|

|R:M 2|

|AT:IK 3|

LLANFAIRPWLLGWYNGYLLGOGERYCHWYRNDROBWLLLLANTYSILIOGOGOGOCH

|L:L 1|, |A:N 2|, |F:AI 3|, |R:P 4|, |W:L 5|

|L:G 6|, |W:Y 7|, |N:GY 8|, |L:L 9|, |G:O 10|

|G:E 11|, |R:Y 12|, |C:H 13|, |W:Y 14|, |R:N 15|

|D:R 16|, |OB:W 17|, |L:L 18|, |L:L 19|, |AN:TY 20|

|S:I 21|, |L:I 22|, |OG:O 23|, |G:O 24|, |G:O 25|

|C:H 26|

RINDFLEISCHETIKETTIERUNGSÜBERWACHUNGSAUFGABENÜBERTRAGUNGSGESETZ

Nicht schreibbar

SOFTWARE

|S:O 1|, |F:T 2|

|W:A 3|, |R:E 4|

TRUMP

|TR:U 1|

|M:P 2|

TSCHÜSS

Nicht schreibbar

VERKEHRSWEGEPLANUNGSBESCHLEUNIGUNGSGESETZ

|V:E 1|, |R:K 2|, |E:H 3|, |R:S 4|

|W:E 5|, |G:E 6|, |P:L 7|, |A:N 8|

|UN:G 9|, |S:B 10|, |E:S 11|, |C:H 12|

|L:E 13|, |UN:I 14|, |G:U 15|, |N:G 16|

|S:G 17|, |E:S 18|, |E:TZ 19|

Quellcode :

AutoScrabble.java :

package autoscrabble;

```

import java.io.File; //Für Dateien
import java.io.FileNotFoundException; //Fehler, entsteht, wenn eine Datei nicht aufzufinden ist
import java.io.FileReader; //Liest Dateien
import java.io.IOException; //Eingabe/Ausgabe Fehler, entsteht, wenn eine Datei nicht lesbar ist
import java.util.ArrayList; //Listen
import java.util.HashMap; //{"Wörterbuch" : jedem Schlüssel(wert) s wird ein Wert w zugeordnet
import java.util.Scanner; //Terminal : Eingabe

/**
 *
 * @author lars
 */
public class AutoScrabble {

    public static final Scanner EINGABE=new Scanner(System.in); //Benutzereingabe
    public static String[] kuerzel; //Alle Kürzel
    public static String[] woerter; //Alle zu prüfenden Wörter
    public static String wort; //Aktuell geprüftes Wort
    public static HashMap positionen; //Stellen, an denen Kürzel im Wort auftauchen
    public static char[] buchstaben; //Großbuchstaben
    public static ArrayList<String> kombinationen; //Alle möglichen Zeichenketten
    public static int n; //Von n Zeichen
    public static final String SEPARATOR = fuelle('-', 50); //Separator

    public static String leseDatei(File pfad_zur_datei) throws FileNotFoundException, IOException { //Liest eine Datei ein, und gibt Text zurück
        FileReader datei = new FileReader(pfad_zur_datei);
        String r = "";
        int i = datei.read();
        while (i != -1) {
            r += (char) i;
            i = datei.read();
        }
        return r;
    }

    public static File dateiFrage(String frage) { //Fragt nach einem Pfad und prüft, ob dieser existiert
        while (true) {
            System.out.println(frage + "(Pfad zu .txt Datei) ? ");
            String s = EINGABE.nextLine();
            File f = new File(s);
            if (f.exists() && !f.isDirectory() && f.canRead()) {
                return f;
            }
            System.out.println("Bitte antworten sie mit einem vorhandenen Pfad einer .txt Datei. Versuchen sie es erneut.");
        }
    }

    public static boolean jaNeinFrage(String frage) { //Stellt eine ja/nein Frage
        while (true) {
            System.out.println(frage + "(j/n) ? ");

```



```

String s = EINGABE.nextLine().toLowerCase();
if (s.equals("j")) {
    return true;
} else if (s.equals("n")) {
    return false;
}
System.out.println("Bitte antworten sie mit j/n beziehungsweise J/N. Versuchen sie es erneut.");
}
}

public static <T> String gebeListeAus(ArrayList<T> k) { //Gibt eine Liste benutzerfreundlich aus
    int zeilenumbruch = (int) (Math.sqrt(k.size()));
    int counter = 0;
    int index = 0;
    String s = "";
    for (T objekt : k) {
        s += objekt.toString();
        counter++;
        index++;
        if (index != k.size()) {
            if (counter == zeilenumbruch) {
                s += "\n";
                counter = 0;
            } else {
                s += ", ";
            }
        }
    }
    return s;
}

public static String fuehle(char c, int m) { //Erzeugt einen Text aus m mal Schriftzeichen c
    String e = "";
    for (int i = 0; i < m; i++) {
        e += c;
    }
    return e;
}

public static boolean umlaut(char c) { //Prüft, ob ein Schriftzeichen ein Umlaut ist
    return c == 'Ä' || c == 'Ö' || c == 'Ü';
    //Gebe "Wahr" zurück, wenn das Schriftzeichen Ö, Ä oder Ü ist, ansonsten gebe "Falsch" zurück.
    //Die Kleinbuchstaben müssen wir hier nicht prüfen, da wir nur mit Großbuchstaben arbeiten
}

public static void moeglichkeiten(int n, String aktuell) { //Rekursive Funktion, generiert alle Möglichkeiten mit n Zeichen aus Großbuchstaben
    for (int i = 0; i < buchstaben.length; i++) {
        if (n == AutoScrabble.n) { //Wenn diese Kombination fertig generiert wurde
            kombinationen.add(aktuell + buchstaben[i]); //Kann sie an die Liste der Kombis angehängt werden
        } else {
            moeglichkeiten(n + 1, aktuell + buchstaben[i]); //Sonst muss wieder gestartet werden, diesmal mit aktueller Kombi als Startwert
        }
    }
}

```

```

    }
}
}

```

public static HashMap<Integer, ArrayList<String>> **findKuerzels**(String s) { *//Findet Städtekuerzel in einem Kennzeichen und speichert, an welchen Indexes welche Kürzel zu finden sind.*

```

    HashMap<Integer, ArrayList<String>> result = new HashMap();
    char[] chars = s.toCharArray();
    int kplace;
    for (String k : kuerzel) { //Für alle Kürzel
        kplace = 0;
        for (int i = 0; i < chars.length; i++) { //Buchstaben des Wortes durchgehen
            if (chars[i] == k.charAt(kplace)) { //Entspricht der aktuelle Buchstabe dem Buchstaben, der jetzt kommen müsste
                kplace++;
            } else { //Falls nicht
                i -= kplace; //Gehe zurück, weil sich das Kennzeichen hier finden lassen könnte
                kplace = 0; //Wir fangen an, den ersten Buchstaben des Wortes zum Vergleich zu stellen
            }
            if (kplace == k.length()) { //Wenn das komplette Wort aufzufinden ist
                int key = i + 1 - k.length(); //Anfangsposition im Wort
                ArrayList<String> value = result.get(key); //Gibt es schon eine Kürzel an dieser Stelle ?
                if (value == null) { //Wenn nicht, erstellen wir eine neue Liste und fügen das eben gefundene Kürzel hinzu
                    ArrayList<String> a = new ArrayList();
                    a.add(k);
                    result.put(i + 1 - k.length(), a);
                } else { //Es gibt schon eine Liste
                    value.add(k); //Fügt das Kennzeichen zu der Liste hinzu
                }
                kplace = 0; //Wir fangen an, den ersten Buchstaben des Wortes zum Vergleich zu stellen
            }
        }
    }
    return result; //Gefunden Kürzel als "Wörterbuch" zurückgeben
}

```

public static boolean **schreibbar**(String s) { *//Gibt aus, ob eine ein Wort mit einem Kennzeichen schreibbar ist. Sinnvoll für Wörter aus 2-5 Buchstaben*

```

    char[] c = s.toCharArray();
    for (String k : kuerzel) {
        if (k.length() < c.length && c.length - 2 <= k.length()) { //Wäre der Anhang maximal zwei Buchstaben, und mindestens einen lang
            boolean unmoeglich = false;
            for (int j = c.length - 1; j >= k.length(); j--) { //Würde der Anhang Umlaute enthalten, ist es nicht mit diesem Kürzel möglich
                if (umlaut(c[j])) {
                    unmoeglich = true;
                    break;
                }
            }
            if (unmoeglich) {
                continue;
            }
            for (int i = 0; i < k.length(); i++) { //Prüfen, ob das Kürzel auch tatsächlich ab Wortanfang vorzufinden ist, falls nicht, ist es mit diesem Kürzel nicht möglich
                if (k.charAt(i) != c[i]) {

```

```

        unmöglich = true;
        break;
    }
}
if (unmöglich) {
    continue;
}
return true; //Sollte es mit diesem Kürzel gehen, ist das Wort offensichtlich schreibbar
}
}
return false; //Sollte es mit keinem Kürzel geklappt haben, ist es wohl leider nicht möglich
}

public static ArrayList<String> pruefe(int z) { //Gibt alle Wörter mit z Buchstaben zurück, die mit Kennzeichen nicht schreibbar sind
    ArrayList<String> ergebnis = new ArrayList(); //Liste nicht schreibbarer Wörter
    kombinationen = new ArrayList(); //Alle Kombinationen aus z Buchstaben
    n = z - 1;
    moeglichkeiten(0, ""); //Alle Wörter generieren
    System.gc(); //Speicher freigeben
    for (String c : kombinationen) {
        if (!schreibbar(c)) { //Falls nicht schreibbar
            ergebnis.add(c); //Gilt dieses Wort als nicht schreibbar
        }
    }
    return ergebnis; //Liste der nicht schreibbaren Wörter mit z Buchstaben zurückgeben
}

public static void pruefeMoegliche() { //Ermittelt nicht schreibbare Wörter
    System.out.println(SEPARATOR);
    boolean timo_schreibbar = schreibbar("TIMO"); //Prüfen, ob "TIMO" schreibbar ist
    System.out.println("TIMO ist ");
    if (!timo_schreibbar) { //Wenn "TIMO" nicht schreibbar ist, wird zwischen "TIMO" und "ist schreibbar" nicht ausgegeben
        System.out.println("nicht ");
    }
    System.out.println("schreibbar");
    System.out.println(SEPARATOR); //Trennlinie
    System.out.println("Nicht schreibbare Wörter mit zwei Buchstaben : ");
    System.out.println(gebeListeAus(pruefe(2))); //Alle nicht schreibbaren Wörter mit zwei Buchstaben ausgeben
    System.gc(); //Speicher freigeben
    System.out.println(SEPARATOR); //Trennlinie
    System.out.println("Nicht schreibbare Wörter mit drei Buchstaben : ");
    System.out.println(gebeListeAus(pruefe(3))); //Alle nicht schreibbaren Wörter mit drei Buchstaben ausgeben
    System.gc(); //Speicher freigeben
    System.out.println(SEPARATOR); //Trennlinie
    if (jaNeinFrage("Wollen sie alle Kombinationen von Wörtern mit 4 Buchstaben, die nicht mit einem Kennzeichen schreibbar sind, anzeigen lassen")) { //Falls erwünscht...
        System.out.println("Nicht schreibbare Wörter mit vier Buchstaben : ");
        System.out.println(gebeListeAus(pruefe(4))); //...alle nicht schreibbaren Wörter mit vier Buchstaben ausgeben(dauert allerdings)
        System.out.println(SEPARATOR); //Trennlinie
    }
}
}

```

```

public static ArrayList<Kennzeichen> schreibeWort(ArrayList<Kennzeichen> a, int stelle) { //Schreibt ein Wort aus Kennzeichen, falls das Wort nicht schreibbar ist, wird "null"
zurückgegeben
    if (wort==null) { //STOP ! Es ist schon eine Lösung gefunden worden.
        return null;
    }
    if (stelle == wort.length()) {
        wort=null; //Bedeutet : Es ist eine Lösung gefunden worden, sämtliche Prozesse, die gestartet wurden und auch nach einer Lösung suchen, werden dann angehalten
        return a; //Kennzeichenliste zurückgeben
    }
    ArrayList<String> kuerzel_bei_stelle = (ArrayList<String>) positionen.get(stelle); //Alle Kürzel einholen, die sich an dem Anfang des noch zu schreibenden Teils befinden
    if (kuerzel_bei_stelle == null || wort.length()-stelle == 1) { //Dieser Prozess fand keine Lösung, wenn es im jetzt noch zu schreibenden Teil kein Kürzel am Anfang gibt oder
noch ein Buchstabe übrig bleibt
        return null;
    } else {
        for (String s : kuerzel_bei_stelle) { //Unter Verwendung dieses Kürzels
            int pos = s.length() + stelle; //Stelle im Wort + Kuerzel
            if (wort.length() - pos >= 1 && !umlaut(wort.charAt(pos))) { //Bleibt mindestens noch ein Buchstabe für den Anhang unter Verwendung dieses Kürzels, und wäre dies kein
Umlaut ?
                //Kopie der Kennzeichen-Liste erstellen
                ArrayList b = new ArrayList();
                b.addAll(a);
                //Neues Kennzeichen hinzufügen (Aktuell geprüftes Kürzel+ein Buchstabe Anhang, durchnummerieren)
                b.add(new Kennzeichen(s, "" + wort.charAt(pos), a.size() + 1));
                ArrayList c = schreibeWort(b, pos + 1); //Weiter "schreiben"
                if (c != null) { //Konnte es zuendegeschrieben werden ?
                    return c; //Gebe das Ergebnis zurück
                }
            }
            if (wort.length() - pos >= 2 && !umlaut(wort.charAt(pos)) && !umlaut(wort.charAt(pos + 1))) { //Bleiben mindestens noch zwei Buchstabe für den Anhang unter
Verwendung dieses Kürzels, und wäre unter diesen kein Umlaut ?
                //Kopie der Kennzeichen-Liste erstellen
                ArrayList b = new ArrayList();
                b.addAll(a);
                //Neues Kennzeichen hinzufügen (Aktuell geprüftes Kürzel+zwei Buchstaben Anhang, durchnummerieren)
                b.add(new Kennzeichen(s, "" + wort.charAt(pos) + wort.charAt(pos + 1), a.size() + 1));
                ArrayList c = schreibeWort(b, pos + 2); //Weiter "schreiben"
                if (c != null) { //Konnte es zuendegeschrieben werden ?
                    return c; //Gebe das Ergebnis zurück
                }
            }
        }
    }
}

return null;
}

public static void main(String[] args) throws IOException {
    buchstaben = new char[29];
    for (int c = 0; c < 26; c++) { //Generiert alle Buchstaben
        buchstaben[c] = (char) (65 + c); //Wandelt Zahl in Buchstaben um
    }
    buchstaben[26] = 'Ä';
}

```

```

buchstaben[27] = 'Ö';
buchstaben[28] = 'Ü';
//Das Buchstaben-Array enthält jetzt : buchstaben = [A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, Ä, Ö, Ü]
kuerzel = leseDatei(dateiFrage("Wo befindet sich die Liste aller Städte Kürzel")).split("\n");
pruefeMoegliche(); //Ermittle nicht-schreibbare Wörter
woerter = leseDatei(dateiFrage("Wo befindet sich die Aufgabe")).split("\n");
System.gc(); //Speicher befreien
for (String w : woerter) { //Wörter durchgehen
    System.out.println(SEPARATOR); //Trennlinie
    wort = w;
    System.out.println(wort); //Wort ausgeben
    positionen = findKuerzels(wort); //Alle Kürzel im Wort ausmachen
    ArrayList<Kennzeichen> k = schreibeWort(new ArrayList(), 0); //Ermittle eine Liste aus Kennzeichen, mit denen das Wort schreibbar ist
    if (k != null) { //Sollte eine gefunden worden sein
        System.out.println(gebeListeAus(k));
    } else { //Ansonsten war's wohl nicht schreibbar
        System.out.println("Nicht schreibbar");
    }
    System.gc(); //Speicher freigeben
}
}
}

```

Kennzeichen.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package autoscrabble;

/**
 *
 * @author lars
 */
public class Kennzeichen { //Speichert Kennzeichen

    public String kuerzel; //Städtekürzel
    public String wert; //Anhang
    public int zahl; //Nummer

    public Kennzeichen(String kuerzel, String wert, int zahl) { //Konstruktor, alle Variablen werden vorgegeben
        this.kuerzel = kuerzel;
        this.wert = wert;
        this.zahl = zahl;
    }

    @Override
    public String toString() { //Benutzerfreundliche Darstellung

```

```
    }  
    return String.format("|%s:%s %d|", kuerzel, wert, zahl); //String formatieren : Text vor(Kuerzel) und hinter(Anhang) dem Doppelpunkt, Nummer einfügen  
}
```

5. Aufgabe „Bauernopfer“

Lösungsidee :

- 1) Die acht Bauern müssen sich in einer Reihe/Diagonale aufstellen. Dann müssen sie sich dem Turm nähern, ohne ihm Freiheiten zu geben, also könnte man sagen „geschlossen“, bis der Turm keine Bewegungsmöglichkeiten mehr hat, wo kein Bauer hinschlagen kann. Dann schlägt ihn einfach ein Bauer.
- 2) Nein, die sieben Bauern können den Turm nicht fangen, da sie ihn nicht „eingrenzen“ können. Der Turm kann sich immer „auf Lücke“ positionieren, welche dann von den Bauern geschlossen werden muss. So kann der Turm dann sein Spielchen ewig treiben, er geht auf Lücke, und die Bauern müssen schließen, damit er ihnen nicht entkommt.
- 3) Ich habe durch Benutzung meiner Applikation feststellen können, dass bei $k+l \geq 9$, vorausgesetzt $k > 0$ sowie $l > 0$, die Bauern den Turm fangen können. So können beispielsweise 8 Bauern, wenn sie immer einen Zug haben, den Turm fangen. Genauso fangen jedoch 7 Bauern mit zwei Zügen den Turm, da sie in der Lage sind, sich so zu positionieren, dass sie alle Felder „kontrollieren“ wo der Turm hinziehen könnte. Ab 6 Bauern mit 3 Zügen können sich die Bauern so verteilen, dass sie das komplette Spielfeld kontrollieren, und der Turm ist dann definitiv chancenlos.
- 4) Die Dame kann diagonal ziehen, also dürfen sich die Bauern keineswegs diagonal aufstellen. Sie müssen sich in einer Reihe aufstellen. Dann sind sie in der Lage, die Dame zu fangen. Wird nämlich ein Bauer nach vorne gezogen, entsteht eine Lücke auf Diagonale. Auf diese geht die Dame dann auch. Diese kann jedoch sofort geschlossen werden, indem ein anderer Bauer nach vorne zieht. So geht das dann ewig weiter. Schließlich verengt sich das Feld immer weiter, und die Dame wird am Ende gefangen. In allen Kombinationen, wo die Bauern das komplette Spielfeld kontrollieren, ist auch die Dame chancenlos. Genauso verhält es sich, wenn die Bauern alle von der Dame erreichbaren Felder kontrollieren. Es gilt also das Gleiche wie für den Turm, mit einer Ausnahme : 7 Bauern mit 2 Zügen können die Dame nicht fangen. Auch dies habe ich mithilfe meiner Applikation ermittelt.

Umsetzung :

Bibliotheken :

- java.awt.Color : Farben
- java.awt.Graphics2D : Wird benutzt, um auf ein Bild zu zeichnen

- java.awt.Graphics : Wird benutzt, um auf den Bildschirm zu zeichnen
- java.util.ArrayList : Listen, wo dynamisch Elemente hinzugefügt und entfernt werden können
- java.util.Scanner : Benutzereingabe, kann direkt aus der Befehlszeile lesen
- java.awt.image.BufferedImage : „Bild“-Klasse von Java. Wird benutzt, um Bilder zu speichern
- javax.imageio.ImageIO : Kann BufferedImages laden
- javax.swing.JPanel : Zeichenfläche auf dem Fenster
- javax.swing.JFrame : Fenster
- java.awt.event.WindowEvent : Speichert Fenster-Ereignisse
- java.awt.event.WindowAdapter : Fängt Fenster-Ereignisse ab : Beispielsweise das Schließen eines Fensters
- java.awt.event.MouseEvent : Speichert Maus-Ereignisse, so wie das Drücken eines Maus-Knopfes
- java.awt.event.MouseListener : Fängt Maus-Ereignisse ab

Datenstrukturen :

Punktmenge : Speichert eine Menge von Punkten auf dem Spielbrett. Bietet außerdem logische Operatoren, so wie `nund`, was bedeutet : Alle Punkte übernehmen, die in Punktmenge a, nicht aber in Punktmenge b vorkommen.

Brett : Speichert das Spielbrett, mit Bauern & Turm. Zeichnet dieses außerdem auch.

Punkt : Ein Punkt auf dem Spielbrett

ALs :

BauernAL : AL für die Bauern

Funktionen :

- `bestimmeFelder(Punktmenge bauern, Punkt bauer)` : Bestimmt die Felder, wo ein Bauer innerhalb eines Zuges hinziehen kann
- `bestimmeFelder(Punktmenge bauern)` : Bestimmt die Felder, die alle Bauern zusammen erreichen können, indem für jeden Bauer die oben genannte Methode ausgeführt wird, und dann die Ergebnisse zusammengekommen werden.
- `bestimmeFelder(Punktmenge bauern, byte züge)` : Bestimmt die Felder, die alle Bauern zusammen innerhalb von „zügen“ Zügen erreichen können. Basiert auf der vorhin erwähnten Funktion. Ermittelt mit einer Schleife, welche Felder im ersten Durchlauf erreicht werden können, und von diesen ausgehend dann, welche im zweiten Durchgang erreicht werden können etc..., bis „züge“ erreicht ist, oder die Bauern keine weiteren Zugmöglichkeiten mehr haben.
- `platziereBauern` : Platziert die Bauern gleichmäßig in einer Waagerechten falls mit Dame gespielt wird, und wenn ohne gespielt wird, in einer Diagonale.
- `zieheBauern(Punktmenge bauern, Punkt turm)` : Ermittelt, welcher Bauer wo hinziehen soll. Zuerst wird geprüft, ob die Bauern innerhalb der bleibenden Züge, welche in einer globalen Variable gespeichert sind, den Turm fangen können. Falls ja, geht der dem Turm nächste Bauer los, den

Turm zu fangen. Ansonsten geschieht dies, aufbauend auf Funktionen aus der TurmAL, indem alle Möglichkeiten für die Bauern, zu ziehen, durchgespielt werden, und daraus dann diverse Werte, wie (1) dem Turm übrig bleibende Freiheiten, oder (2) die Gewichtung der übrig bleibenden Freiheiten, oder gar (3) die insgesamt Entfernung der Bauern zum Turm, ermittelt werden. Jenachdem, wie diese Werte ausfallen, wird der für die Bauern beste Zug ermittelt. Priorität hat hierbei, dem Turm am wenigsten Freiheiten zu lassen

TurmAL : AI für den Turm/die Dame

Funktionen :

- bestimmeFelder(Punktmenge bauern, Punkt turm) : Bestimmt die Felder, wo der Turm innerhalb eines Zuges hinziehen kann. Falls mit Dame gespielt wird, wird dies berücksichtigt.
- sichereFelder(Punktmenge bauern, Punkt turm) : Bestimmt die Felder, wo der Turm innerhalb eines Zuges hinziehen kann, sich aber auch keine Schlagmöglichkeit für einen Bauern innerhalb der gegebenen Zugzahl befindet. Baut auf der 3.) bestimmeFelder Methode aus der BauernAL auf, um zu bestimmen, wo die Bauern hinziehen könnten, sowie auf der oben genannten Methode der TurmAL, um zu bestimmen, wo hingezogen werden könnte. Auch wird hier die „nund“ Methode der Punktmenge benutzt, damit wir alle Punkte ausschließen, die von Bauern betreten werden können.
- gewichteteFelder(Punktmenge bauern, Punkt turm) : Bestimmt alle sicheren Felder, die vom Turm erreicht werden können. Baut auf sichereFelder auf. Ermittelt für jedes Feld eine Gewichtung, welche 4-(Minimale Zugzahl zum Erreichen des Feldes) entspricht. Ermittelt mit einer Schleife, welche sicheren Felder im ersten Durchlauf erreicht werden können, und von diesen ausgehend dann, welche im zweiten Durchgang erreicht werden können etc..., bis der Turm/die Dame keine weiteren sicheren Zugmöglichkeiten mehr hätte.
- smarteFelder(Punktmenge bauern, Punkt turm) : Bestimmt alle Felder, die der Turm zum Hinziehen in Betracht ziehen sollte. Verfährt dafür ähnlich wie gewichteteFelder, nur dass eben noch die Ursprünge der in späteren Zügen erreichbaren Felder verzeichnet werden. Somit lässt sich ermitteln, welche Felder Ursprung von in 3 Zügen erreichbaren Feldern sind. Diese geben wir dann zurück, es sei denn, es gibt keine solchen. Dann ermitteln wir die Ursprünge von in zwei Zügen erreichbaren Feldern. Gibt es nun Resultate, geben wir diese zurück. Ansonsten geben wir alle in einem Zug erreichbaren Felder zurück. Gibt es leider Gottes jedoch kein einziges sicheres Feld, so ist es egal was der Turm tut. Er kann also getrost stehenbleiben. Es wird die eigene Position zurückgegeben.
- platziereTurm : Platziert den Turm. Dazu wird für jeden „Bereich“ ermittelt, wie viele Freiheiten der Turm da hätte, und wie „wertvoll“ diese sind. Im besten Bereich wird dann der Turm am fernsten aller Bauern platziert, und in einem Feld, wo er sicher ist. Dies wird mit sichereFelder ermittelt. Gibt es kein solches, wird auch der Punkt am fernsten aller Bauern genommen.
- zieheTurm(Punktmenge bauern, Punkt turm) : Ermittelt, wo der Turm/die Dame hinziehen soll. Hierzu wird smarteFelder aufgerufen, und die Ergebnisse werden nach den Kriterien (1, Priorität) Insgesamt-Gewichtung und (2) Insgesamt-Entfernung zu den Bauern bewertet. Das Beste wird zurückgegeben.

Sonstiges :

Maus : Helferklasse zum Verarbeiten von Maus-Ereignissen

Programmstruktur :

In dieser Applikation benutzen wir erstmal zwei eigene Funktionen : mengenFrage, sowie jaNeinFrage.

Mit der ersten bekommen wir folgende Infos vom Benutzer :

- Mit wievielen Bauern wird gespielt ?
- Wieviele Züge sollen sie pro Runde haben ?

Und mit der zweiten :

- Wird mit Dame gespielt ?
- Sollen die Bauern/der Turm vom Rechner gesteuert werden ?

Dann erzeugen wir ein Fenster, und setzen ein neues Brett-Objekt als Inhalt dieses Fensters. Eine „Maus“ wird erzeugt, und zum Fenster „hinzugefügt“. Jenachdem, was gewählt worden ist, wird evtl. dem Benutzer die Möglichkeit gegeben, (eine) Spielpartei(en) zu platzieren. Danach fängt die Hauptschleife an. Diese „managt“ gewissermaßen alles.

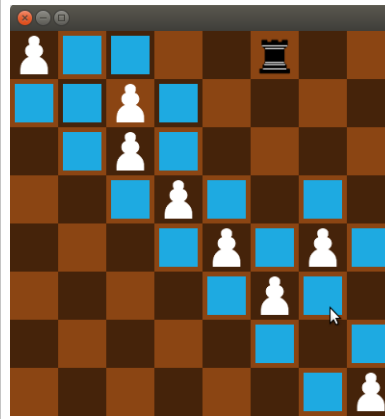
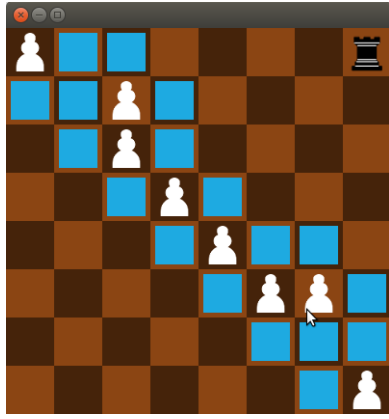
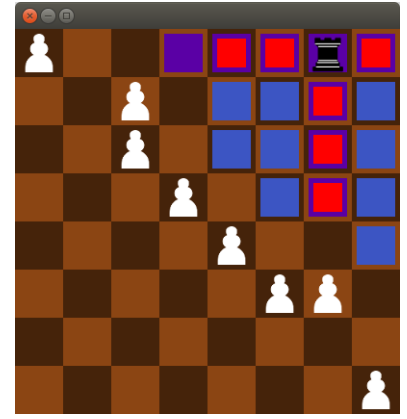
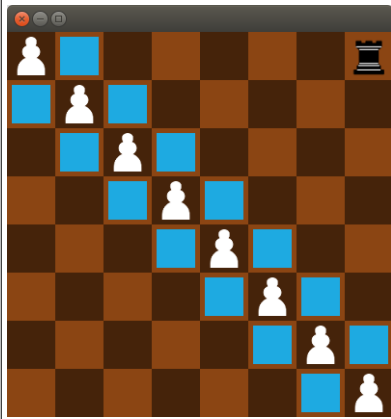
Sobald die Maus gedrückt wird, wird jenachdem die AL, oder der Benutzer zum Zug gelassen, in dem die Funktion der AL aufgerufen wird, und das Ergebnis der „bewegeBauern“ Funktion des Brettes bzw. der „bewegeTurm“ Funktion übergeben wird. Dies geschieht mit 20 FPS. Sobald gezogen wurde und nach jeder Platzierung wird das Brett neugezeichnet. Das Brett bemerkt, wenn die Bauern den Turm gefangen haben. Dann wird „Die Bauern haben gewonnen !“ ausgegeben. Genauso gibt es die Ausgabe „Das Programm wurde beendet.“, wenn das Schließen-Symbol des Fensters betätigt wird.

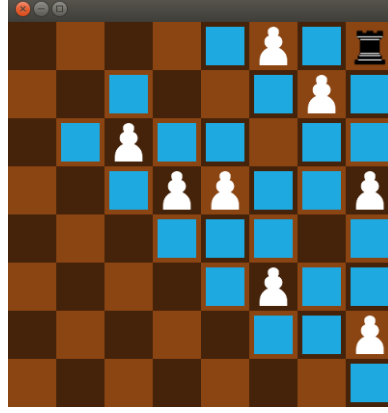
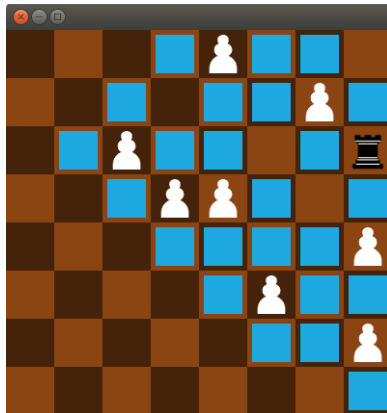
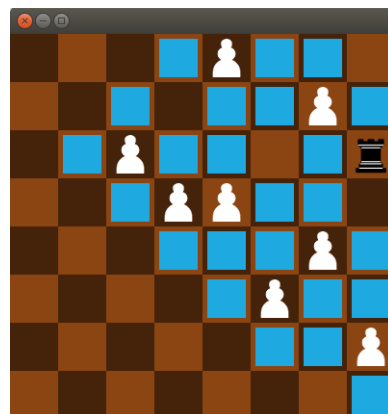
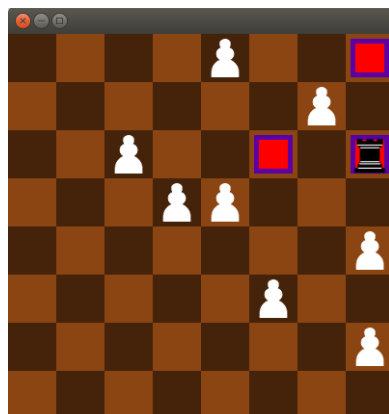
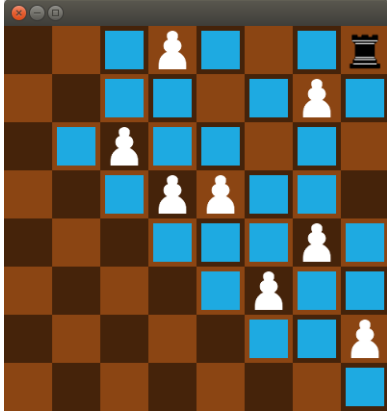
Beispiele :

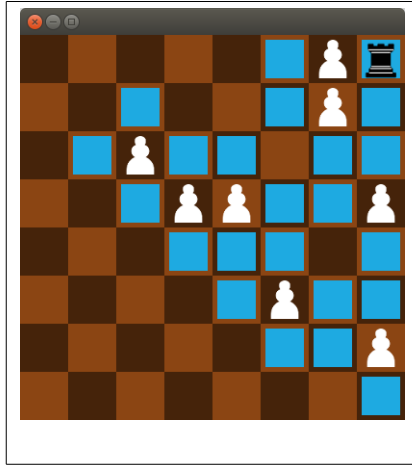
Auszug aus einem AL vs AL Spiel :

Anzahl Bauern : 8

Anzahl Züge : 1



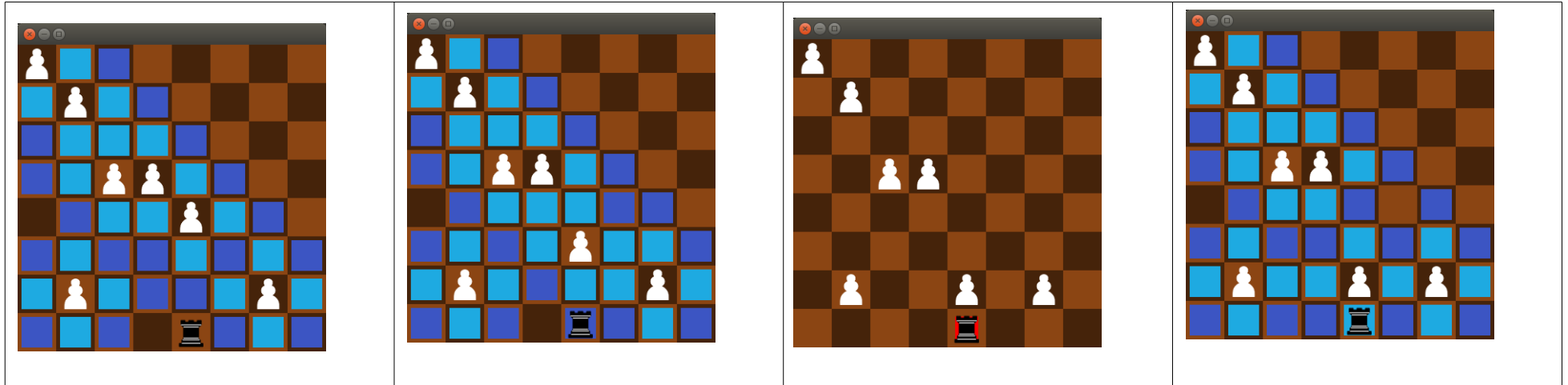




Auszug aus einem AL vs AL Spiel :

Anzahl Bauern : 7

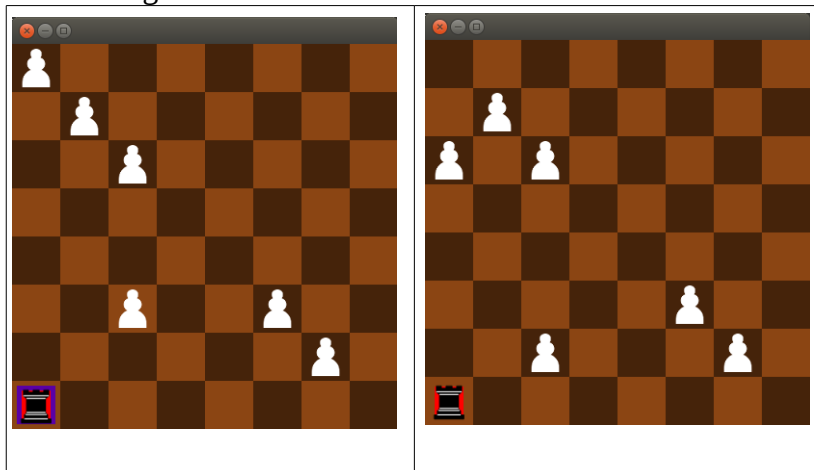
Anzahl Züge : 2



Auszug aus einem AL vs AL Spiel :

Anzahl Bauern : 6

Anzahl Züge : 3

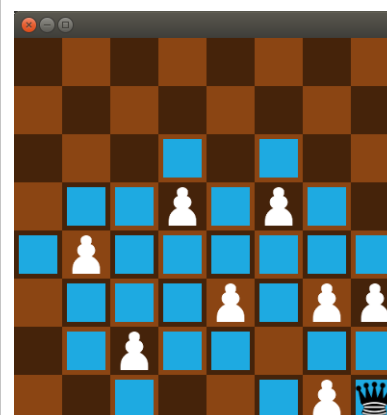
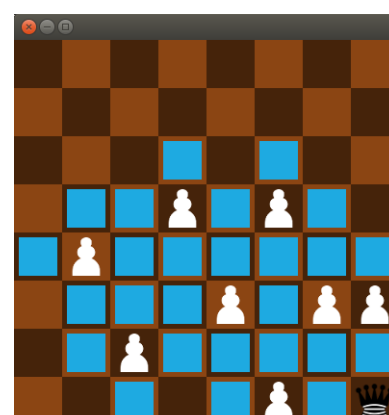
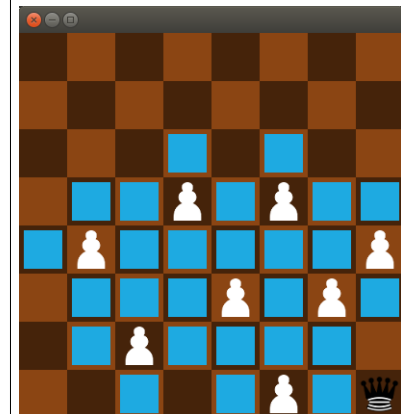
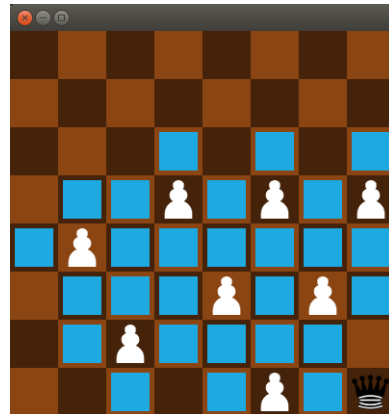


Auszug aus einem AL vs AL Spiel :

- Dame

Anzahl Bauern : 8

Anzahl Züge : 1



Quellcode :

Brett.java :

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package bauernopfer;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.io.File;
import javax.imageio.ImageIO;
import javax.swing.JPanel;

/**
 *
 * @author lars
 */
public class Brett extends JPanel {
    public static final BufferedImage SCHACHBRETT;
    public static final BufferedImage BAUER_BILD=ladeBild("res/bauer.png", Color.WHITE); //Bild eines Bauern
    public static final BufferedImage TURM_BILD; //Bild des Turmes/der Dame
    static {
        //Bild der Dame/des Turmes laden
        if (!Bauernopfer.dame) {
            TURM_BILD=ladeBild("res/turm.png", Color.BLACK); //Bild des Turmes
        }
        else {
            TURM_BILD=ladeBild("res/dame.png", Color.BLACK); //Bild der Dame
        }
        //Schachbrett zeichnen
        SCHACHBRETT=new BufferedImage(400,400, BufferedImage.TYPE_INT_RGB);
        Graphics2D g=SCHACHBRETT.createGraphics();
        //Hintergrund hellbraun füllen
        g.setColor(new Color(139, 69, 19));
        g.fillRect(0,0,400,400);
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                //Jedes 2. Feld dunkelbraun füllen
                if ((i + j) % 2 == 0) {
                    g.setColor(new Color(69, 35, 10));
                    g.fillRect(i * 50, j * 50, 50, 50);
                }
            }
        }
    }
}
```

```

    }
}
public Punktmenge bauern; //Speichert die Positionen aller Bauern
public Punkt turm; //Speichert die Position des Turms/der Dame

public void platziereTurm(Punkt position) { //Platziert den Turm an gegebener Position
    turm=position;
}

public void platziereBauern(Punktmenge positionen) { //Platziert alle Bauern an gegebenen Positionen
    bauern=positionen;
}

public void bewegeTurm(Punkt position) { //Bewegt den Turm zu einer gegebenen Position
    turm=position;
}

public void bewegeBauern(Punkt bauer, Punkt position) { //Bewegt den Bauern am Punkt "bauer" zu dem Punkt "position"
    bauern.raster[bauer.x][bauer.y]=false; //Den Bauern an alter Stelle löschen
    bauern.raster[position.x][position.y]=true; //Und an neuer platzieren
    if (position.x == turm.x && position.y == turm.y) { //Konnte die Dame/der Turm geschlagen werden
        System.out.println("Die Bauern haben gewonnen ! ");
        System.exit(0); //Programm beenden
    }
    bauern=new Punktmenge(bauern.raster); //Bauern aktualisieren
}

public static BufferedImage ladeBild(String pfad, Color farbe) {
    try {
        return ImageIO.read(new File(pfad)); //Versuche, das Bild zu laden
    } catch (Exception fehler) {
        //Ansonsten wird ein farbiger Kreis als Ersatz benutzt
        BufferedImage ziel = new BufferedImage(50, 50, BufferedImage.TYPE_INT_ARGB);
        Graphics2D zielg = ziel.createGraphics();
        zielg.setBackground(new Color(0, 0, 0, 0));
        zielg.setColor(farbe);
        zielg.fillOval(5, 5, 40, 40);
        return ziel;
    }
}

public Brett() { //Konstruktor - noch nichts vorgegeben
    bauern=new Punktmenge();
    turm=null;
}

@Override
public void paintComponent(Graphics g) { //Brett "zeichnen"
    g.drawImage(SCHACHBRETT,0,0,null);
    if (Bauernopfer.bauern_dran) {
        byte[][] erreichbar=BauernAL.bestimmeFelder(bauern,Bauernopfer.zuege); //Alle Felder, die von den Bauern erreichbar sind
    }
}

```



```

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (erreichbar[i][j] > 0) { //Wenn das Feld erreichbar ist
                g.setColor(new Color(Math.min(255,30*erreichbar[i][j]),Math.max(0,255-(85*erreichbar[i][j])),255-Math.min(255,30*erreichbar[i][j]))); //Jenachdem, wie viele Züge
                die Bauern minimal zum Erreichen eines Feldes benötigen, färben wir dies ein
                g.fillRect(i * 50+5, j * 50+5, 40, 40);
            }
        }
    }
} else {
    byte[][] erreichbar=TurmAL.gewichteteFelder(bauern, turm); //Alle Felder, die vom Turm erreichbar sind
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (erreichbar[i][j] > 0) {
                g.setColor(new Color(Math.min(255,30*erreichbar[i][j]),Math.max(0,255-(85*erreichbar[i][j])),255-Math.min(255,30*erreichbar[i][j]))); //Jenachdem, wie viele Züge
                der Turm/die Dame minimal zum Erreichen eines Feldes benötigen, färben wir dies ein
                g.fillRect(i * 50+5, j * 50+5, 40, 40);
            }
        }
    }
    for (Punkt p : TurmAL.smarteFelder(bauern, turm).punkte) { //Felder, welche von der TurmAL als Möglichkeiten, wo der Turm/die Dame hingehen sollte, betrachtet werden,
        //Werden rot eingefärbt
        g.setColor(Color.RED);
        g.fillRect(p.x * 50 + 10, p.y * 50 + 10, 30, 30);
    }
}
if (turm != null) { //Wenn der Turm schon platziert wurde,
    g.drawImage(TURM_BILD, turm.x * 50, turm.y * 50, this); //Dann können wir ihn auch anzeigen lassen
}
for (Punkt Bauer : bauern.punkte) {
    g.drawImage(BAUER_BILD, Bauer.x * 50, Bauer.y * 50, this); //Alle schon platzierten Bauern können wir auch anzeigen lassen
}
}
}
}

```

Bauernopfer.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Bauernopfer;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList; //Listen
import java.util.Scanner; //Benutzereingabe
import javax.swing.JFrame; //Fenster

```

```

public class Bauernopfer {

    public static boolean bauern_dran = true; //Sind die Bauern momentan dran ?
    public static boolean dame = false; //Wird mit Dame gespielt ?
    public static byte zuege; //Wie viele Züge haben die Bauern ?
    public static byte bleibende_zuege; //Wie viele Züge bleiben den Bauern noch
    public static final Scanner EINGABE = new Scanner(System.in); //Benutzereingabe
    public static JFrame fenster;

    public static boolean jaNeinFrage(String frage) { //Stellt eine ja/nein Frage an den Benutzer
        while (true) {
            System.out.println(frage + "(j/n) ? ");
            String s = EINGABE.nextLine().toLowerCase();
            if (s.equals("j")) {
                return true;
            } else if (s.equals("n")) {
                return false;
            }
            System.out.println("Bitte antworten sie mit j/n beziehungsweise J/N. Versuchen sie es erneut.");
        }
    }

    public static byte mengenFrage(String frage) { //Fragt nach einer Zahl von 1-8
        while (true) {
            System.out.println(frage + "(Zahl von 1-8) ? ");
            String s = EINGABE.nextLine().toLowerCase();
            try {
                byte so = Byte.parseByte(s);
                if (so > 0 && so < 9) {
                    return so;
                }
            } catch (NumberFormatException e) {
                System.out.println("Bitte antworten sie mit einer Zahl von 1 bis 8. Versuchen sie es erneut.");
            }
        }
    }

    public static void main(String[] args) {
        dame = jaNeinFrage("Soll mit Dame gespielt werden");
        boolean turmal = jaNeinFrage("Soll der Turm/die Dame vom Rechner gesteuert werden");
        boolean bauernal = jaNeinFrage("Sollen die Bauern vom Rechner gesteuert werden");
        byte bauern = mengenFrage("Wie viele Bauern soll es geben");
        zuege = mengenFrage("Wie viele Züge sollen sie haben");
        bleibende_zuege = zuege;
        fenster = new JFrame(); //Neues Fenster erzeugen
        fenster.setSize(400, 400); //Größe auf 400x400 = 50*8 x 50*8 Schachbrett
        Brett brett = new Brett(); //Neues Spielbrett erzeugen
        fenster.setContentPane(brett); //Brett im Fenster anzeigen
        fenster.setVisible(true); //Fenster sichtbar machen
        fenster.setFocusable(true); //Für die Maus fokusierbar machen
        fenster.requestFocus(); //Maus auf dieses Fenster fokussieren
    }
}

```

```

fenster.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("Programm beendet.");
        System.exit(0); //Wird das Fenster geschlossen, soll das komplette Programm anhalten
    }
});
long now; //FPS-Zähler
Maus maus = new Maus(); //Neue "Maus" erzeugen, die alle für dieses Programm relevanten "Maus-Infos" speichert
fenster.addMouseListener(maus); //Maus zum Fenster hinzufügen, um Position etc. bei Änderung und pro Frame abzufangen
if (bauer.al) { //Werden die Bauern vom Rechner gesteuert ?
    brett.platziereBauern(Bauern.al.platziereBauern((byte) bauer)); //Bauern von der Bauern.al platzieren lassen
    brett.repaint(); //Anzeigen, wie sie platziert wurden
} else { //Ansonsten kann man die Bauern selbst platzieren
    ArrayList<Punkt> plaetze = new ArrayList();
    byte platzieren = 0;
    now = System.currentTimeMillis();
    while (true) {
        if (System.currentTimeMillis() - now > 50) { //20 FPS
            now = System.currentTimeMillis();
            maus.infosHolen(); //Maus-Infos abfangen
            if (maus.LMB > 1) { //Wurde die linke Maustaste gedrückt
                Punkt p = new Punkt((byte) (maus.mouseX / 50), (byte) (maus.mouseY / 50)); //Ausgewähltes Feld
                if (!brett.bauern.raster[p.x][p.y]) { //Ist an dem ausgewählten Feld noch kein Bauer
                    plaetze.add(p); //So fügen wir diesen hinzu
                    brett.platziereBauern(new Punktmenge(plaetze)); //Und updaten das Brett
                    brett.repaint(); //Brett neuzeichnen
                    platzieren++;
                    if (platzieren == bauern) { //Wurden alle Bauern platziert
                        break; //Sind wir hier fertig
                    }
                }
            }
            maus.LMB = 0;
        }
    }
}
}
if (turm.al) { //Falls der Turm/die Dame vom Computer gesteuert werden soll
    brett.platziereTurm(Turm.al.platziereTurm(brett.bauern)); //Automatisch platzieren
    brett.repaint(); //Brett neuzeichnen
} else {
    now = System.currentTimeMillis();
    while (true) {
        maus.infosHolen(); //Maus-Infos abfangen
        if (System.currentTimeMillis() - now > 50) { //20 FPS
            now = System.currentTimeMillis();
            if (maus.LMB > 1) { //Wurde die linke Maustaste gedrückt
                Punkt p = new Punkt((byte) (maus.mouseX / 50), (byte) (maus.mouseY / 50)); //Ausgewähltes Feld
                if (!brett.bauern.raster[p.x][p.y]) { //Ist an dem ausgewählten Feld noch kein Bauer
                    brett.platziereTurm(p); //So kann dort der Turm/die Dame platziert werden
                    brett.repaint(); //Brett neuzeichnen
                }
            }
        }
    }
}

```

```

        break; //Der Turm/die Dame wurde platziert, hier sind wir fertig
    }
    maus.LMB = 0;
}
}
}
}
Punkt b = null; //Koordinaten vom aktuell ausgewählten Bauern. Ist "null" wenn keiner ausgewählt wurde.
//K und L : WENN K+L == 9 ?
now = System.currentTimeMillis();
while (true) {
    if (System.currentTimeMillis() - now > 50) { //20 FPS
        maus.infosHolen(); //Maus-Infos abfangen
        if (maus.LMB > 1) { //Wurde die linke Maustaste gedrückt
            if (bauern_dran) { //Sind die Bauern dran
                if (bauernal) { //Werden die Bauern vom Computer gesteuert
                    Punkt[] bewegen = BauernAL.zieheBauern(brett.bauern, brett.turm); //Bauern ziehen, gibt (1) die Ausgangsposition des Bauern, also welcher Bauer gezogen wird,
und (2) dessen Zielposition zurück
                    brett.bewegeBauern(bewegen[0], bewegen[1]); //Brett aktualisieren
                    bleibende_zuege--;
                    if (bleibende_zuege == 0) { //Sollten den Bauern keine Züge mehr bleiben,
                        bauern_dran = false; //Ist wieder der Turm dran
                        bleibende_zuege = zuege; //Und die Bauern haben wieder volle beliebende Zugzahl
                    }
                } else if (b == null) { //Wenn der Mensch(der Spieler) spielt, und noch keinen Bauern ausgewählt hat,
                    Punkt p = new Punkt((byte) (maus.mouseX / 50), (byte) (maus.mouseY / 50)); //Ausgewähltes Feld
                    if (brett.bauern.raster[p.x][p.y]) { //Befindet sich dort ein Bauer
                        b = p; //Wird dieser ausgewählt
                    }
                } else { //Wenn der Mensch(der Spieler) spielt, und schon einen Bauern ausgewählt hat,
                    ArrayList<Punkt> begehbar = BauernAL.bestimmeFelder(brett.bauern, b); //Ermitteln wir, welche Felder alle von dem Bauer begangen werden können
                    Punkt q = new Punkt((byte) (maus.mouseX / 50), (byte) (maus.mouseY / 50)); //Ausgewähltes Feld
                    boolean c = false; //Kontrolle c : Ist das ausgewählte Feld begehbar ?
                    for (Punkt k : begehbar) {
                        if (k.x == q.x && k.y == q.y) { //Es ist begehbar
                            c = true;
                            break; //Schleife beenden
                        }
                    }
                    if (c) {
                        brett.bewegeBauern(b, q); //Da es ja begehbar ist, können wir nun den Bauern ziehen
                        bleibende_zuege--;
                        if (bleibende_zuege == 0) { //Sollten den Bauern keine Züge mehr bleiben,
                            bauern_dran = false; //Ist wieder der Turm dran
                            bleibende_zuege = zuege; //Und die Bauern haben wieder volle beliebende Zugzahl
                        }
                        b = null; //Nach dem Zug muss der menschliche Spieler erneut einen Bauern auswählen
                    } else if (brett.bauern.raster[q.x][q.y]) { //Sollte sich an der gewünschten Position ein Bauer befinden, c also "Falsch" sein,
                        b = q; //So ist dieser nun ausgewählt
                    }
                }
            }
        }
    }
}

```



```

        //Punkt übernehmen
        this.raster[x][y]=true;
        punkte.add(new Punkt(x,y));
    }
}
}
}
}
public Punktmenge(byte[][] raster) { //Konstruktor - Erzeugt eine Punktmenge aus einem Spielbrett, wo Punkte als != 0 eingetragen sind
    this.punkte=new ArrayList();
    this.raster=new boolean[8][8];
    for (byte x=0; x < 8; x++) {
        for (byte y=0; y < 8; y++) {
            if (raster[x][y] != 0) { //Falls sich dort ein Punkt befindet,
                //Punkt übernehmen
                this.raster[x][y]=true;
                punkte.add(new Punkt(x,y));
            }
        }
    }
}
public Punktmenge(ArrayList<Punkt> punkte) { //Konstruktor - Erzeugt eine Punktmenge aus einer Liste
    this.punkte=new ArrayList();
    this.raster=new boolean[8][8];
    for (Punkt p:punkte) {
        //jeden Punkt übernehmen
        this.raster[p.x][p.y]=true;
        this.punkte.add(p);
    }
}
public Punktmenge oder(Punktmenge feld) { //Logischer "oder" Operator
    boolean[][] ergebnis=new boolean[8][8];
    for (byte x=0; x < 8; x++) {
        for (byte y=0; y < 8; y++) {
            if (this.raster[x][y] || feld.raster[x][y]) { //Ist der Punkt in mindestens einer der beiden Punktmenge vorhanden
                ergebnis[x][y]=true; //So wird er ins Ergebnis übernommen
            }
        }
    }
    return new Punktmenge(ergebnis);
}
public Punktmenge nund(Punktmenge feld) { //Logischer (1), aber nicht (2) Operator, also werden die Überschneidungen beider Punktmenge entfernt
    boolean[][] ergebnis=new boolean[8][8];
    for (byte x=0; x < 8; x++) {
        for (byte y=0; y < 8; y++) {
            if (this.raster[x][y] && !feld.raster[x][y]) {
                ergebnis[x][y]=true; //Wenn sich der Punkt nicht mit dem anderen überschneidet, dann kann er ins Ergebnis übernommen werden
            }
        }
    }
    return new Punktmenge(ergebnis);
}
}

```

```

@Override
public String toString() { //Benutzerfreundliche Ausgabe des Feldes, "x" für verzeichnete Punkt, "o" für nicht Verzeichnete
    String s="";
    for (byte x=0; x < 8; x++) {
        for (byte y=0; y < 8; y++) {
            if (raster[x][y]) {
                s+="x";
            }
            else {
                s+="o";
            }
        }
        s+="\n";
    }
    return s;
}
}

```

BauernAL.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package bauernopfer;

import java.util.ArrayList;

/**
 *
 * @author lars
 */
public class BauernAL {

    public static final Punkt[] RICHTUNGEN = new Punkt[]{new Punkt((byte) 0, (byte) -1), new Punkt((byte) 0, (byte) 1), new Punkt((byte) -1, (byte) 0), new Punkt((byte) 1, (byte) 0)}; //Bewegungsrichtungen Bauer

    public static Punktmenge platziereBauern(byte bauern) { //Platziert "bauern" Bauern, falls eine Dame im Spiel ist, in einer Reihe, sonst in einer Diagonale
        ArrayList<Punkt> result = new ArrayList();
        float faktor = 8.0f / (float) bauern; //Sorgt für eine gleichmäßige Verteilung der Bauern auf der 8 Felder langen Diagonalen/Reihe
        if (!Bauernopfer.dame) {
            for (byte b = 0; b < bauern; b++) {
                result.add(new Punkt((byte) (b * faktor), (byte) (b * faktor))); //X und Y-Koordinate ändern sich
            }
        } else {
            for (byte b = 0; b < bauern; b++) {
                result.add(new Punkt((byte) (b * faktor), (byte) 3)); //Nur die X-Koordinate ändert sich, die Y-Koordinate bleibt 3(möglichst mittig)
            }
        }
    }
}

```

```

    return new Punktmenge(result);
}

public static boolean imFeld(Punkt p) { //Ermittelt, ob ein Punkt p innerhalb des Spielfeldes liegt
    return p.x > 0 && p.y > 0 && p.x < 8 && p.y < 8;
}

public static ArrayList<Punkt> bestimmeFelder(Punktmenge bauern, Punkt bauer) { //Bestimmt die Punkte, wo ein Bauer innerhalb eines Zuges hinziehen kann
    ArrayList<Punkt> ergebnis = new ArrayList();
    if (bauer.x != 0) { //Befindet er sich nicht am linken Rand
        ergebnis.add(new Punkt((byte) (bauer.x - 1), bauer.y)); //Kann er nach links ziehen
    }
    if (bauer.y != 0) { //Befindet er sich nicht am rechten Rand
        ergebnis.add(new Punkt(bauer.x, (byte) (bauer.y - 1))); //Kann er nach rechts ziehen
    }
    if (bauer.x != 7) { //Befindet er sich nicht am unteren Rand
        ergebnis.add(new Punkt((byte) (bauer.x + 1), bauer.y)); //Kann er nach unten ziehen
    }
    if (bauer.y != 7) { //Befindet er sich nicht am oberen Rand
        ergebnis.add(new Punkt(bauer.x, (byte) (bauer.y + 1))); //Kann er nach oben ziehen
    }
    Punktmenge feld = new Punktmenge(ergebnis);
    Punktmenge felder = feld.nund(bauern); //Alle Bewegungsmöglichkeiten, wo sich anderer Bauer befindet, entfernen
    return felder.punkte;
}

public static Punktmenge bestimmeFelder(Punktmenge bauern) { //Bestimmt alle Felder, die von allen "bauern" zusammen innerhalb eines Zuges erreicht werden können
    ArrayList<Punkt> insgesamt = new ArrayList(); //Ergebnis
    for (Punkt bauer : bauern.punkte) {
        insgesamt.addAll(bestimmeFelder(bauern, bauer)); //Alle Zugmöglichkeiten jedes Bauern zum Ergebnis hinzufügen
    }
    return new Punktmenge(new Punktmenge(insgesamt).raster); //(1) Gibt das Ergebnis als Punktmenge zurück, und (2) entfernt doppelt vorkommende Punkte
}

public static byte[][] bestimmeFelder(Punktmenge bauern, byte zuege) { //Bestimmt alle Felder, die von allen "bauern" zusammen innerhalb von "zuege" Zügen erreicht werden können
    byte[][] ergebnis = new byte[8][8]; //Ergebnis als Spielfeld. Jedes Feld speichert eine Nummer, die angibt, wie viele Züge die Bauern brauchen, um das Feld zu erreichen
    ArrayList<Punkt> neu_checken; //Punkte, die danach neu hinzukommen, und geprüft werden sollen
    ArrayList<Punkt> checken = new ArrayList(); //Punkte, für die ermittelt werden soll, welche Felder die Bauern von ihnen aus erreichen können
    checken.addAll(bauern.punkte); //Es fängt bei den Bauern an
    byte durchlauf = 1; //Gibt an, der wievielte Zug momentan simuliert wird
    while (true) {
        if (checken.isEmpty() || durchlauf > zuege) { //Wenn es keine weitere von den Bauern erreichbare Felder gibt, oder schon genug Züge simuliert worden sind
            break; //Dann sind wir fertig
        }
        neu_checken = new ArrayList();
        for (Punkt p : checken) { //Für alle zu prüfenden Punkte
            ArrayList<Punkt> vpe = bestimmeFelder(new Punktmenge(), p); //Die von ihnen aus erreichbaren Punkte bestimmen
            for (Punkt q : vpe) { //Von den erreichbaren Punkten
                if (ergebnis[q.x][q.y] == 0 && !bauern.raster[q.x][q.y]) { //Wenn der Punkt nicht schon erreicht wurde, und dort kein Bauer ist
                    ergebnis[q.x][q.y] = (byte) (durchlauf); //Dann speichern wir, das er im Zug "durchlauf" erreicht werden kann
                }
            }
        }
        neu_checken.addAll(neu_checken);
        checken = neu_checken;
        durchlauf++;
    }
    return ergebnis;
}

```



```

        neu_checken.add(q); //Für diesen Punkt soll dann im nächsten Durchlauf geprüft werden, welche Felder von ihm aus erreichbar sind
    }
}
//Im nächsten Durchlauf sollen die Punkte geprüft werden, die im jetzigen Durchlauf als "im nächsten Durchlauf zu prüfen" deklariert wurden
checken = new ArrayList();
for (Punkt k : neu_checken) {
    checken.add(k);
}
durchlauf++; //Nächster Durchlauf !
}
return ergebnis;
}

public static Punkt[] zieheBauern(Punktmenge bauern, Punkt turm) { //Gibt ein Punkt-Array zurück. Der erste Punkt gibt den gewählten Bauern an, der zweite den Punkt, wo
dieser hinzieht
    byte[][] schlagbar = BauernAL.bestimmeFelder(bauern, Bauernopfer.bleibende_zuege); //Bestimmt alle Felder, die von allen Bauern zusammen innerhalb von den bleibenden
Zügen erreicht werden können
    byte bester_bauer = 0;
    byte beste_zugzahl = Byte.MAX_VALUE;
    if (schlagbar[turm.x][turm.y] != 0) { //Sollte der Turm mithilfe der übrig bleibenden Züge erreichbar sein, wird zum Turm hin gezogen
        //So ermitteln wir zuerst den Bauern, der am nächsten am Turm liegt
        for (byte i = 0; i < bauern.punkte.size(); i++) {
            Punkt d = turm.minus(bauern.punkte.get(i)); //Abstand in x und y Richtung zum Turm
            byte f = (byte) (Math.abs(d.x) + Math.abs(d.y)); //Benötigte Züge
            if (f < beste_zugzahl) { //Wenn dieser Bauer weniger Züge benötigen würde
                beste_zugzahl = f; //Die beste Zugzahl ist dann f
                bester_bauer = i; //Der beste Bauer ist dann dieser
            }
        }
        Punkt d = turm.minus(bauern.punkte.get(bester_bauer)); //Weg, der zum Turm gegangen werden muss
        byte r;
        if (d.x != 0) { //Ist der noch nicht auf der gleichen x-Koordinate wie der Turm
            //Dann muss er da erstmal hin gehen
            if (d.x < 0) { //Ist er rechts vom Turm
                r = 2; //Muss er nach links gehen
            } else {
                r = 3; //Sonst nach rechts
            }
        } else if (d.y < 0) { //Ansonsten ist er auf der gleichen x-Koordinate wie der Turm, in diesem Fall unter ihm
            r = 0; //Dann muss er nach oben gehen
        } else {
            r = 1; //Ansonsten geht's nach unten
        }
        return new Punkt[] {bauern.punkte.get(bester_bauer), bauern.punkte.get(bester_bauer).plus(RICHTUNGEN[r])}; //Geben wir zurück : Der nächste Bauer in die bestimmte
Richtung
    }
    short beste_gewichtung = Short.MAX_VALUE;
    byte meiste_erreichbare_felder = Byte.MAX_VALUE;
    double beste_entfernung = Double.MAX_VALUE;
    byte richtung = 0;

```

```

//Brute-Force ! Wir ermitteln, welcher Zug sich aus Sicht der Bauern am meisten lohnt
for (byte i = 0; i < bauern.punkte.size(); i++) { //Alle Bauern durchgehen
    for (byte r = 0; r < 4; r++) { //Alle möglichen 4 Bewegungsrichtungen für jeden durchgehen
        Punkt bauer = bauern.punkte.get(i);
        Punkt neue_position = bauer.plus(RICHTUNGEN[r]); //Position des Bauern nach ziehen in die Richtung r
        if (!BauernAL.imFeld(neue_position) || bauern.raster[neue_position.x][neue_position.y]) {
            continue; //Dann simulieren wir hier nicht weiter
        }
        //Wir simulieren nun
        Punktmenge bauern_moved = new Punktmenge(bauern.raster); //Die Bauern, nach dem Zug
        bauern_moved.raster[bauer.x][bauer.y] = false; //Der Bauer wird an der alten Stelle gelöscht
        bauern_moved.raster[neue_position.x][neue_position.y] = true; //Und an der neuen gesetzt
        bauern_moved = new Punktmenge(bauern_moved.raster); //Punktmenge aktualisieren
        byte[][] gewichtungen = TurmAL.gewichteteFelder(bauern_moved, turm); //Spielbrett, auf dem eingetragen ist, 4-wieviele Züge der Turm bräuchte, um das jeweilige Feld
        //zu erreichen, also die "Gewichtungen". Felder, die er nicht erreichen kann, sind 0.
        Punktmenge p = new Punktmenge(gewichtungen); //Punktmenge, repräsentiert erreichbare Felder
        byte e = (byte) p.punkte.size(); //Wie viele Felder sind überhaupt erreichbar ?
        short g = 0; //Insgesamt-Gewichtung
        for (byte[] spalte : gewichtungen) {
            for (byte zeile : spalte) {
                g += zeile; //Von jedem Feld die Gewichtung hinzufügen
            }
        }
        double entfernung = TurmAL.bestimmeEntfernung(bauern_moved, turm); //Insgesamt-Entfernung der Bauern zum Turm
        if (e < meiste_erreichbare_felder) { //Würde dieser Zug dem Turm mehr Freiheiten wegnehmen ?
            //Alle "Rekord-Variablen" auf den Stand vom jetzigen Rekordhalter setzen, und speichern, das diesen Bauer in diese Richtung zu ziehen die beste Wahl ist
            richtung = r;
            bester_bauer = i;
            meiste_erreichbare_felder = e;
            beste_entfernung = entfernung;
            beste_gewichtung = g;
        } else if (e == meiste_erreichbare_felder) { //Wäre dieser Zug in Hinsicht auf die erreichbaren Felder equivalent zum jetzigen Rekordhalter ?
            if (g < beste_gewichtung) { //Würde dieser Zug im Vergleich zum jetzigen Rekordhalter dem Turm "wertvollere" Freiheiten wegnehmen ?
                //Alle "Rekord-Variablen" auf den Stand vom jetzigen Rekordhalter setzen, und speichern, das diesen Bauer in diese Richtung zu ziehen die beste Wahl ist
                beste_gewichtung = g;
                richtung = r;
                bester_bauer = i;
                meiste_erreichbare_felder = e;
                beste_entfernung = entfernung;
            } else if (g == beste_gewichtung) { //Wäre dieser Zug AUCH in Hinsicht auf die "wertvollen" Freiheiten equivalent zum jetzigen Rekordhalter ?
                if (entfernung < beste_entfernung) { //Jedoch bei diesem Zug die Bauern näher am Turm dran
                    //Alle "Rekord-Variablen" auf den Stand vom jetzigen Rekordhalter setzen, und speichern, das diesen Bauer in diese Richtung zu ziehen die beste Wahl ist
                    beste_gewichtung = g;
                    richtung = r;
                    bester_bauer = i;
                    meiste_erreichbare_felder = e;
                    beste_entfernung = entfernung;
                }
            }
        }
    }
}
}

```

```

    }
    return new Punkt[] {bauern.punkte.get(bester_bauer), bauern.punkte.get(bester_bauer).plus(RICHTUNGEN[richtung])}; //Am Ende geben wir den nach den Prinzipien
    gefundenen besten Zug zurück
}
}

```

TurmAL.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package bauernopfer;

import java.util.ArrayList;

/**
 *
 * @author lars
 */
class TurmAL {
    public static final Punkt[] DAME_RICHTUNGEN=new Punkt[] {new Punkt((byte)1,(byte)1),new Punkt((byte)-1,(byte)-1),new Punkt((byte)-1,(byte)1),new Punkt((byte)1,(byte)-
1)}; //Richtungen, in die eine Dame ziehen kann

    public static Punkt platziereTurm(Punktmenge bauern) { //Platziert den Turm
        Punktmenge unsicher = BauernAL.bestimmeFelder(bauern).oder(bauern); //Felder, wo sich (1) ein Bauer befindet, oder (2) ein Bauer hin schlagen kann, gelten alle als
        "unsicher"
        ArrayList<Punkt> fmmf = new ArrayList(); //Felder, die vorerst zur Auswahl stehen
        byte rekord = 0;
        for (byte x = 0; x < 8; x++) {
            for (byte y = 0; y < 8; y++) {
                if (!unsicher.raster[x][y]) { //Wenn das Feld sicher, also nicht unsicher ist
                    byte[][] a=gewichteteFelder(bauern, new Punkt(x,y)); //Ermitteln wir, welche Felder der Turm von hier aus in wievielen Zügen erreichen könnte, als Gewichtung
                    byte wert = 0;
                    for (byte x2 = 0; x2 < 8; x2++) {
                        for (byte y2 = 0; y2 < 8; y2++) {
                            if (a[x2][y2] != 0) { //Wenn das Feld gewichtet ist
                                unsicher.raster[x2][y2] = true; //Alle Felder, die der Turm erreichen kann, brauchen wir nicht darauf zu prüfen, wie viele Felder von dort aus erreichbar sind
                                wert+=(byte) (4-a[x2][y2]); //Felder, für die es mehr Züge braucht, gelten als wertvoller
                            }
                        }
                    }
                }
            }
        }
        if (wert > rekord) { //Wenn diese Gewichtung den Rekordhalter schlägt
            fmmf = new ArrayList(); //Ist der alte Bereich nicht mehr relevant
        }
        if (wert >= rekord) { //Wenn dieser Bereich äquivalent dem Rekordhalter, oder besser ist...
            for (byte x2 = 0; x2 < 8; x2++) {
                for (byte y2 = 0; y2 < 8; y2++) {
                    if (a[x2][y2] != 0) {

```

```

        fmmf.add(new Punkt(x2, y2)); //Kommen dessen Punkte alle hinzu
    }
}
}
}
}
}
}
}
Punkt rhalter = new Punkt((byte) 0, (byte) 0); //Bester Punkt
double rdis = 0; //Rekorddistanz Turm zum nächsten Bauern
if (fmmf.isEmpty()) { //Es konnte kein sicheres Feld gefunden werden
    //So ermitteln wir von allen Punkten den, wo der Turm vom nächsten Bauern entfernt ist
    for (byte x = 0; x < 8; x++) {
        for (byte y = 0; y < 8; y++) {
            if (!bauern.raster[x][y]) { //Alle Punkte, wo kein Bauer ist
                Punkt k = new Punkt(x, y);
                double mindis = Double.MAX_VALUE;
                for (Punkt bauer : bauern.punkte) {
                    double dis = bauer.entfernung(k);
                    if (dis < mindis) {
                        mindis = dis;
                    }
                }
                if (mindis > rdis) {
                    rdis = mindis;
                    rhalter = k;
                }
            }
        }
    }
    fmmf.add(rhalter); //Der Rekordhalter kommt infrage
}
//Besten Platzierungspunkt aus den infrage kommenden bestimmen
Punkt rekordhalter = fmmf.get(0);
double rekord_distanz = bestimmeEntfernung(bauern, rekordhalter);
for (byte i = 1; i < fmmf.size(); i++) {
    double d = bestimmeEntfernung(bauern, fmmf.get(i));
    if (d > rekord_distanz) {
        rekord_distanz = d;
        rekordhalter = fmmf.get(i);
    }
}
return rekordhalter; //Diesen zurückgeben
}

public static double bestimmeEntfernung(Punktmenge bauern, Punkt t) { //Bestimmt die insgesamte Entfernung des Turmes t zu den "bauern"
    double result = 0;
    for (Punkt bauer : bauern.punkte) {
        result += bauer.entfernung(t);
    }
    return result;
}

```

```

}

public static Punktmenge bestimmeFelder(Punktmenge bauern, Punkt turm) { //Bestimmt alle Felder, die der Turm/die Dame innerhalb eines Zuges erreichen kann
    ArrayList<Punkt> ergebnis = new ArrayList();
    byte minx = -1;
    byte maxx = 8;
    byte miny = -1;
    byte maxy = 8;
    for (Punkt bauer : bauern.punkte) {
        //Welche Bauern beschränken den Turm in der Waagerechten ?
        if (bauer.x == turm.x) {
            if (bauer.y < turm.y) {
                if (bauer.y > miny) {
                    miny = bauer.y;
                }
            } else if (bauer.y < maxy) {
                maxy = bauer.y;
            }
        }
        //Und welche in der Senkrechten ?
        else if (bauer.y == turm.y) {
            if (bauer.x < turm.x) {
                if (bauer.x > minx) {
                    minx = bauer.x;
                }
            } else if (bauer.x < maxx) {
                maxx = bauer.x;
            }
        }
    }
    //Dann kann er vom Bauern, der am wenigsten links von ihm steht, bis zum Bauern, der am wenigsten rechts von ihm steht
    for (byte y = (byte) (miny + 1); y < maxy; y++) {
        ergebnis.add(new Punkt(turm.x, y));
    }
    //Dann kann er vom Bauern, der am wenigsten über ihm steht, bis zum Bauern, der am wenigsten unter ihm steht
    for (byte x = (byte) (minx + 1); x < maxx; x++) {
        ergebnis.add(new Punkt(x, turm.y));
    }
    if (Bauernopfer.dame) { //Wird mit Dame gespielt
        boolean[] nicht_checken=new boolean[4]; //Welche Bewegungsrichtungen der Dame brauchen wir nicht mehr prüfen ?
        for (int d=1; d < 8; d++) { //Wir gehen bis zu 7 mögliche Diagonalzüge durch
            boolean b=true;
            for (int i=0; i < 4; i++) { //Alle Bewegungsmöglichkeiten der Dame durchgehen
                if (!nicht_checken[i]) { //Wenn sie in die Richtung noch weiterlaufen kann
                    Punkt c=DAME_RICHTUNGEN[i];
                    Punkt cpos=new Punkt((byte)(turm.x+c.x*d),(byte)(turm.y+c.y*d)); //Dann ermitteln wir, wo sie da hin käme
                    if (!BauernAL.imFeld(cpos) || bauern.raster[cpos.x][cpos.y]) { //Wurde sie (1) außerhalb des Feldes laufen oder (2) auf einen Bauern laufen
                        nicht_checken[i]=true; //Geht's in die Richtung nicht mehr weiter
                        continue; //Diese Richtung wird nicht weiter verfolgt
                    }
                }
                b=false; //Wenn noch eine weitere Zugmöglichkeit gefunden wurde, simulieren wir weiter
            }
        }
    }
}

```

```

        ergebnis.add(cpos); //Und fügen die Zugmöglichkeit zum Ergebnis zurück
    }
}
if (b) { //Konnte sich die Dame nicht weiter bewegen
    break; //Sind wir fertig
}
}
}
return new Punktmenge(ergebnis); //Ergebnis zurückgeben
}

public static Punktmenge sichereFelder(Punktmenge bauern, Punkt turm) { //Ermittelt die Felder, die vom Turm, aber nicht von den Bauern in der ihnen bleibenden Zugzahl
betreten werden können
    Punktmenge begehbbareFelder = TurmAL.bestimmeFelder(bauern, turm); //Vom Turm begehbbare Felder
    Punktmenge unsicher = new Punktmenge(BauernAL.bestimmeFelder(bauern, Bauernopfer.bleibende_zuege)); //Von den Bauern innerhalb der bleibenden Züge erreichbare
Felder
    Punktmenge sichereFelder = begehbbareFelder.nund(unsicher); //Alle Felder, wo der Turm aber kein Bauer hinkommt
    return sichereFelder;
}

public static Punkt zieheTurm(Punktmenge bauern, Punkt turm) { //Zieht den Turm
    Punktmenge moeglichkeiten = smarteFelder(bauern, turm); //Alle vorgeschlagenen Zugmöglichkeiten
    Punkt best_punkt = null;
    double best_entfernung = 0.0f;
    byte max_sum = 0;
    for (Punkt p : moeglichkeiten.punkte) { //Von allen vorgeschlagenen Zugmöglichkeiten
        byte sum = 0;
        byte[][] gewichtete_felder = gewichteteFelder(bauern, p);
        for (byte x = 0; x < 8; x++) {
            for (byte y = 0; y < 8; y++) {
                sum += gewichtete_felder[x][y];
            }
        }
        double e = 0;
        for (Punkt b : bauern.punkte) {
            e += p.entfernung(b);
        }
        if (sum > max_sum) { //Resultieren aus diesem Zug "bessere" / "wertvollere" Freiheiten ?
            //Gilt dieser jetzt als Bester
            max_sum = sum;
            best_entfernung = e;
            best_punkt = p;
        } else if (sum == max_sum && e > best_entfernung) { //Ist dieser Zug äquivalent, aber wäre der Turm hiermit weiter entfernt
            //Gilt dieser jetzt als Bester
            best_entfernung = e;
            best_punkt = p;
        }
    }
    return best_punkt;
}

```

```

public static byte[][] gewichteteFelder(Punktmenge bauern, Punkt turm) {
    byte[][] ergebnis = new byte[8][8]; //Ergebnis als Spielfeld. Jedes Feld speichert eine Nummer, die angibt, wie viele Züge die Bauern brauchen, um das Feld zu erreichen, als
Gewichtung : 4-n
    ArrayList<Punkt> checken = new ArrayList(); //Punkte, die danach neu hinzukommen, und geprüft werden sollen
    ArrayList<Punkt> neu_checken; //Punkte, für die ermittelt werden soll, welche Felder der Turm von ihnen aus erreichen können
    checken.add(turm); //Beim Turm fangen wir an
    byte durchlauf = 1;
    while (true) {
        if (checken.isEmpty()) { //Gibt's keine weiteren zu prüfenden Felder
            break; //Sind wir fertig
        }
        neu_checken = new ArrayList();
        for (Punkt p : checken) { //Für alle zu prüfenden Punkte
            Punktmenge vpe = TurmAL.sichereFelder(bauern, p); //Die von ihnen aus erreichbaren(sicheren) Punkte bestimmen
            for (Punkt q : vpe.punkte) { //Von den erreichbaren Punkten
                if (ergebnis[q.x][q.y] == 0) { //Wenn der Punkt nicht schon erreicht wurde
                    ergebnis[q.x][q.y] = (byte) (4 - durchlauf); //Dann speichern wir, dass er mit Gewichtung 4-"durchlauf", also 4-Gewichtung Zügen erreicht werden kann
                    neu_checken.add(q); //Für diesen Punkt soll dann im nächsten Durchlauf geprüft werden, welche Felder von ihm aus erreichbar sind
                }
            }
        }
        //Im nächsten Durchlauf sollen die Punkte geprüft werden, die im jetzigen Durchlauf als "im nächsten Durchlauf zu prüfen" deklariert wurden
        checken = new ArrayList();
        for (Punkt q2 : neu_checken) {
            checken.add(q2);
        }
        durchlauf++; //Nächster Durchlauf !
    }
    return ergebnis;
}

```

```

public static Punktmenge smarteFelder(Punktmenge bauern, Punkt turm) { //Felder, die für den Turm zum hingehen in Betracht gezogen werden
    Punkt[][] links = new Punkt[8][8]; //Speichert, von welchen Punkt aus ein Feld erreicht wurde
    byte[][] w = new byte[8][8]; //Speichert die Gewichtungen der erreichbaren Felder
    ArrayList<Punkt> checken = new ArrayList(); //Punkte, die danach neu hinzukommen, und geprüft werden sollen
    ArrayList<Punkt> neu_checken; //Punkte, für die ermittelt werden soll, welche Felder der Turm von ihnen aus erreichen können
    checken.add(new Punkt(turm.x, turm.y)); //Beim Turm fangen wir an
    byte durchlauf = 1;
    while (true) {
        if (checken.isEmpty()) { //Gibt's keine weiteren zu prüfenden Felder
            break; //Sind wir fertig
        }
        neu_checken = new ArrayList();
        for (Punkt p : checken) { //Für alle zu prüfenden Punkte
            Punktmenge vpe = TurmAL.sichereFelder(bauern, p); //Die von ihnen aus erreichbaren(sicheren) Punkte bestimmen
            for (Punkt q : vpe.punkte) { //Von den erreichbaren Punkten
                if (links[q.x][q.y] == null) { //Wenn der Punkt nicht schon erreicht wurde
                    w[q.x][q.y] = durchlauf; //Dann speichern wir, dass das Feld im "durchlauf" Durchlauf erreicht werden könnte
                    links[q.x][q.y] = p; //Wir speichern, von welchem Punkt aus er erreicht wurde
                    neu_checken.add(q); //Für diesen Punkt soll dann im nächsten Durchlauf geprüft werden, welche Felder von ihm aus erreichbar sind
                }
            }
        }
    }
}

```

```

    }
}
//Im nächsten Durchlauf sollen die Punkte geprüft werden, die im jetzigen Durchlauf als "im nächsten Durchlauf zu prüfen" deklariert wurden
checken = new ArrayList();
for (Punkt q2 : neu_checken) {
    checken.add(q2);
}
durchlauf++; //Nächster Durchlauf !
}
//Jetzt ermitteln wir die Ursprungspunkte für die in 3 Zügen erreichbaren Felder, diese sind das Ergebnis
ArrayList<Punkt> ergebnis = new ArrayList();
for (byte x = 0; x < 8; x++) {
    for (byte y = 0; y < 8; y++) {
        if (w[x][y] == 3) {
            Punkt p = links[x][y];
            Punkt r = links[p.x][p.y];
            ergebnis.add(r);
        }
    }
}
if (ergebnis.isEmpty()) { //Gab es keine in 3 Zügen erreichbare Felder
    //So ermitteln wir die Ursprungspunkte für die in zwei Zügen Erreichbaren
    for (byte x = 0; x < 8; x++) {
        for (byte y = 0; y < 8; y++) {
            if (w[x][y] == 2) {
                Punkt p = links[x][y];
                ergebnis.add(p);
            }
        }
    }
}
if (ergebnis.isEmpty()) { //Gab es SOGAR keine in 2 Zügen erreichbare Felder
    //So ermitteln wir alle in einem Zug erreichbaren
    for (byte x = 0; x < 8; x++) {
        for (byte y = 0; y < 8; y++) {
            if (w[x][y] == 1) {
                ergebnis.add(new Punkt(x, y));
            }
        }
    }
}
if (ergebnis.isEmpty()) { //Gab es keine sicheren Felder, die in einem Zug erreicht werden konnten
    ergebnis.add(turm); //Ist es egal, was der Turm/die Dame, einfach stehenbleiben !
}
return new Punktmenge(ergebnis);
}
}

```

Punkt.java :


```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package bauernopfer;

/**
 *
 * @author lars
 */
public class Punkt { //Speichert einen Punkt auf dem Spielbrett

    public byte x; //X-Koordinate(von 0-7), deshalb reicht 1 Byte
    public byte y; //Y-Koordinate(von 0-7), deshalb reicht 1 Byte

    public Punkt(byte x, byte y) { //Konstruktor - X und Y werden gegeben
        this.x = x;
        this.y = y;
    }

    public double entfernung(Punkt p) { //Bestimmt nach dem Satz des Pythagoras die Entfernung zweier Punkte
        byte xw=(byte) (p.x-x);
        byte yw=(byte) (p.y-y);
        return Math.sqrt(xw * xw + yw * yw);
    }

    public Punkt plus(Punkt p) { //Addiert auf diesen Punkt einen Punkt p
        return new Punkt((byte)(this.x+p.x),(byte)(this.y+p.y));
    }

    public Punkt minus(Punkt p) { //Zieht von diesem Punkt einen Punkt p ab
        return new Punkt((byte)(this.x-p.x),(byte)(this.y-p.y));
    }

    @Override
    public String toString() { //Gibt diesen Punkt in Tupleschreibweise aus
        return "(" + Byte.toString(x) + "|" + Byte.toString(y) + ")";
    }
}

```

Maus.java :

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package bauernopfer;

```

```

/**
 *
 * @author lars
 */
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

/**
 *
 * @author lars
 */
public class Maus implements MouseListener {
    public int mouseX; //Speichert die X-Position der Maus
    public int mouseY; //Speichert die Y-Position der Maus
    public int LMB; //Speichert, ob die linke Maustaste gedrückt worden ist
    public int RMB; //Speichert, ob die rechte Maustaste gedrückt worden ist
    public Maus() {
        RMB=0;
        LMB=0;
        Bauernopfer.fenster.requestFocus(); //Alle Maus-Ereignisse von diesem Fenster abfangen lassen
    }
    @Override
    public void mousePressed(MouseEvent m) { //Mausknopf gedrückt
        if (m.getButton() == MouseEvent.BUTTON3) { //Rechte Maustaste
            RMB=1;
        }
        if (m.getButton() == MouseEvent.BUTTON1) { //Linke Maustaste
            LMB=1;
        }
    }
    @Override
    public void mouseClicked(MouseEvent m) { //Mausknopf geklickt
        if (m.getButton() == MouseEvent.BUTTON3) { //Rechte Maustaste
            RMB=2;
        }
        if (m.getButton() == MouseEvent.BUTTON1) {
            LMB=2;
        }
    }
    @Override
    public void mouseReleased(MouseEvent m) { //Mausknopf losgelassen
        if (m.getButton() == MouseEvent.BUTTON3) { //Rechte Maustaste
            RMB=3;
        }
        if (m.getButton() == MouseEvent.BUTTON1) { //Linke Maustaste
            LMB=3;
        }
    }
    @Override
    public void mouseEntered(MouseEvent m) {
    }
}

```

```
@Override
public void mouseExited(MouseEvent m) {
}
public void infosHolen() {
    Bauernopfer.fenster.requestFocus(); //Alle Maus-Ereignisse von diesem Fenster abfangen lassen
    try {
        //Versucht, die Maus-Koordinaten relativ zum Fenster zu erfahren, erzeugt einen Fehler, wenn die Maus außerhalb des Fensters ist
        mouseX=Bauernopfer.fenster.getMousePosition().x;
        mouseY=Bauernopfer.fenster.getMousePosition().y;
    } catch (Exception e) {}
}
}
```