Aufgabe 1: Zimmerbelegung

Lösungsidee

Bei dieser gesamten Aufgabe geht es darum, eine Zimmerkonstellation zu erstellen, die alle gegebenen Wünsche erfüllt. Dabei besitzt jede Schülerin eine Liste, auf der Personen stehen, mit denen Sie ein Zimmer teilen möchte („Plus-Liste“) und mit denen sie nicht ein Zimmer teilen möchte („Minus-Liste“). Dabei wurde die Aufgabe so vereinfacht, dass die Zahl der Zimmer und Zimmergrößen nicht beschränkt ist. Die Lösung darf demnach so viele Zimmer beinhalten wie möglich, dabei sind auch Einzelzimmer erlaubt.

Aufgrund dieser Vereinfachung lässt sich die Lösung dieser Aufgabe in zwei Schritten erklären:

1. Erfüllung des ersten Wunsches: „Mit wem möchte ich in einem Zimmer sein?“

Zuallererst müssen die Zimmer mit den Schülerinnen aufgestellt werden. Dies erfolgt, indem man die Schülerinnen, die jeweils zusammen in einem Zimmer sein wollen, auch einem Zimmer zuordnet. Man beginnt mit der ersten Schülerin und ordnet diese und diejenigen aus ihrer Plus-Liste einem Zimmer zu. Für die restlichen Schülerinnen wird folgende Überprüfung durchgeführt:

Bei der zu untersuchenden Schülerin muss man zuerst vergewissern, ob diese oder eine der Schülerinnen ihrer Plus-Liste bereits dem ersten Zimmer zugeordnet wurde. Ist dies der Fall, muss die Schülerin mitsamt denen ihrer Plus-Liste dem ersten Zimmer zugeteilt werden, wenn sie schon nicht bereits zugeteilt wurden. Dann wiederholt man die Überprüfung mit den restlichen Schülerinnen, die dem Zimmer noch nicht zugeordnet wurden, solange bei jeder Überprüfung mindestens eine Schülerin gefunden wird, die in dem ersten Zimmer vorkommt. Fällt die Überprüfung erstmals negativ aus, geht man zur nächsten Schülerin mit einer nicht-leeren Plus-Liste über und falls diese nicht zugeordnet wurde, wird sie einem neuen Zimmer zugeteilt und der vorherige Prozess wird für sie wiederholt.

Mit dieser Herangehensweise weiß man, dass in den erstellten Zimmern alle Schülerinnen vorkommen, die zusammen ein Zimmer teilen müssen. Die Wiederholung der Überprüfung verhindert, dass man indirekte Verbindungen vergisst (Schülerinnen, die erst nach dem Hinzufügen einer anderen Person zugeteilt werden wollen). Damit kann es keine Fälle geben, in denen zwei Schülerinnen aus zwei verschiedenen Zimmern dieselbe Wunschperson teilen.

Schülerinnen, die keine Personen in ihrer Plus-Liste stehen haben, werden danach überprüft. Falls diese nicht bereits einem Zimmer zugeordnet wurden, so weiß man dadurch, dass sie auch nicht mit den anderen Schülerinnen mit leeren Plus-Listen unbedingt ein Zimmer teilen möchte. Ich habe mich deswegen entschieden, diesen der Einfachheit halber ein Einzelzimmer zu geben.

Insgesamt erstellt man so die einfachste Möglichkeit der Zimmerbelegung, da nur die Mindestanforderung („Personen, mit denen man ein Zimmer teilen möchte“) erfüllt wird. Daher wird in meinem Lösungsansatz die Zimmeranzahl maximal sein, die einzelnen Zimmergrößen werden minimal sein.

2. Erfüllung des zweiten Wunsches: „Mit wem möchte ich nicht in einem Zimmer sein?“

Wenn der erste Wunsch erfüllt wird, heißt das nicht, dass die gesamte Zimmerbelegung funktioniert. Dafür muss auch der zweite Wunsch berücksichtigt werden. Zimmerbelegungen können nur dann nicht funktionieren, wenn Schülerinnen in ein Zimmer gebracht werden, obwohl sie das von dem zweiten Wunsch her nicht wollen. Das heißt, man betrachtet nun auch hier die vorher erstellten Zimmer und geht diese der Reihe nach durch. Für jedes dieser Zimmer untersucht man nun den zweiten Wunsch der Minus-Liste für jede der in dem Zimmer vorkommenden Schülerin. Nur dann, wenn der Fall auftritt, dass eine Schülerin ein Zimmer teilen muss mit jemandem aus ihrer Minus-Liste, so kann es keine Zimmerbelegung mit dieser Wunschkonstellation geben. Das wird damit begründet, dass beide Schülerinnen Personen aus ihren Plus-Listen teilen und auch voneinander nicht getrennt werden können. Bei weiteren regelkonformen Optimierungen der Zimmerbelegung würden nur die Zimmeranzahl verringert und die Zimmergrößen erhöht werden, Wunschkonflikte würden daher weiterhin bestehen.

Mathematische Ergänzung:

Für jede der Schülerinnen existiert eine Menge S, die aus einem Element („Name der Schülerin“) und zwei weiteren Mengen L1 und L2 besteht, deren Elemente aus den Plus-/Minus-Listen der Namen der Schülerinnen stammen. Die Zimmer repräsentieren selbst Mengen aus den Namen der Schülerinnen aus dem jeweiligen Zimmer. In Schritt 1 wird bei der Überprüfung geschaut, ob die Menge S der jeweiligen Schülerin und die Menge Z des jeweiligen Zimmers disjunkt sind (ohne Berücksichtigung der Menge L2). In Schritt 2 wird bei der Überprüfung geschaut, ob die Menge L2 der jeweiligen Schülerin und die Menge Z des jeweiligen Zimmers disjunkt sind (ohne Berücksichtigung der Menge L1).

Umsetzung der Idee:

Meine Umsetzung der Lösungsidee ist an die mathematische Ergänzung angelehnt.

Als Programmiersprache wurde Java ausgesucht aufgrund der eigenen Präferenz. Als Entwicklungsumgebung wurde BlueJ benutzt, da der Quellcode hier farblich untermalt wird, was zu einer besseren Lesbarkeit führt.

Allgemein wurden für die Erstellung des Programms zwei Klassen erstellt: Eine Klasse „Zimmerbelegung“ und eine Klasse „Schüler“.

Die Klasse Schüler beschreibt, welche Informationen jede Schülerin beinhalten soll. Das wären einmal der Name (*private String* ***name***), ein Array bestehend aus den Namen der Schülerinnen aus der Plus-Liste (private String[] teilen), ein Array bestehend aus den Namen der Schülerinnen aus der Minus-Liste (*private String[] n****ichtTeilen***) und ein Statusflag, der anzeigt, ob eine Schülerin beretis zugeordnet wurde (*private boolean* ***zugeteilt***). Der Rest der Klasse besteht aus Konstruktoren, um diese Variablen zu initialisieren, und aus Setter/-Getter-Methoden, um die Variablen zu setzen und die Daten aus diesen zu entnehmen.

Die Klasse Zimmerbelegung enthält als Instanzvariablen ein Array bestehend aus den Schülerinnen (*private Schüler[]* ***schüler***) und eine Arrayliste bestehend aus einer Arrayliste, die die Namen der Schülerinnen aus dem jeweiligen Zimmer enthält (*ArrayList<ArrayList<String>>* ***zimmer***).

Die wichtigsten Methoden zur Erstellung und Ausgabe der Zimmerbelegung befinden sich in dieser Klasse. Insgesamt sind es sechs Methoden, die im Folgenden näher erläutert werden:

- public int zimmernummerVonSchüler (*Schüler* ***s***)

Diese Methode dient dazu, die Zimmernummer zurückzugeben, bei welcher die als Argument übergegebene Schülerin ***s*** beziehungsweise eine andere Schülerin ihrer Plus-Liste einem Zimmer zugeordnet wurde (beginnend ab 0) zurück bzw. -1, wenn sie nicht zugeordnet wurde. Hierzu werden in der ersten for-Schleife alle Zimmer durchlaufen. In einer if-Abfrage wird dann getestet, ob der Name der Schülerin (ein String-Literal) in der der Liste ***zimmer*** enthalten ist. Falls ja, soll der Index des Zimmers, bei welchem die if-Abfrage positiv verläuft, als Ganzzahl zurückgegeben werden. Wenn die Schülerin doch nicht in einem der Zimmer vorkommt, werden in einer zweiten for-Schleife die Schülerinnen aus ihrer Plus-Liste (sofern sie nicht leer ist) genauso auf ihre Vorkommen in einem der Zimmer überprüft. Falls in beiden for-Schleifen die if-Abfrage stets negativ verläuft, wird -1 zurückgegeben. Da die Indezes der Zimmer nie -1 betragen können, bedeutet eine Rückgabe von -1, dass keine der Schülerinnen in einem der Zimmer vorkommen. Weil diese Methode nach der Erstellung der Zimmerbelegung erst benutzt wird, wird nie -1 zurückgegeben.

- public boolean istSchülerInEinemZimmer(*Schüler* ***s****, ArrayList<String>* ***schülerImZimmer***)

Diese Methode überprüft, ob die Schülerin in dem gegebenen Zimmer vorkommt. Funktional gesehen gleicht sie der vorherigen Methode, nur dass man keine zusätzliche for-Schleife zum Durchlaufen aller Zimmer benötigt.

- public String zimmerBelegen()

Die Variable ***schalter*** des Typs Boolean wird erst mal deklariert. In einer for-Schleife durchläuft man alle Schülerinnen. In der darauffolgenden if-Abfrage werden Schülerinnen mit leeren Plus-Listen behandelt. Ihre Namen werden einem Einzelzimmer zugeordnet und man geht zur nächsten Schülerin über. Eine andere if-Abfrage überspringt Schülerinnen, die bereits zugeordnet wurden über das Statusflag ***zugeteilt***. Wenn eine Schülerin bis dahin nicht aussortiert wurde, so werden sie und diejenigen aus ihrer Plus-Liste in einer for-Schleife einem Zimmer zugeordnet. Jetzt wird bei der aktuellen Schülerin zugeteilt auf *true* gesetzt. Die folgende do-while-Schleife enthält selbst eine for-Schleife, die alle Schülerinnen erneut durchläuft. Die Variable ***schalter*** wird zu Beginn auf *false* gesetzt. Man überspringt Schülerinnen, die bereits zugeordnet wurden. Des Weiteren überprüft man, ob die neue Schülerin oder eine aus ihrer Plus-Liste in dem vorherigen erstellten Zimmer vorkommt. Falls ja, wird ihr Statusflag auch auf *true* gesetzt und die Namen der Schülerinnen, falls sie nicht bereits im Zimmer vorkommen, diesem Zimmer hinzugefügt. Die Variable Schalter wird nur dann auf *true* gesetzt. Die do-while-Schleife wird erst beendet, wenn in einem Schleifendurchgang der Schalter nicht auf *true* gesetzt wird.

- public void setSchülerliste(String datei) throws IOException

Diese Methode wird verwendet, um das Schüler-Array schüler zu initialisieren. Zuerst werden hier eine temporäre Schülerliste und eine temporäre Plus-/Minus-Liste (wird in dieser Methode alternierend erst als Plus-Liste, dann als Minus-Liste verwendet) des Datentyps *ArrayList<Schüler>* und *ArrayList<String>* erstellt. Dies hat den Vorteil, dass die Listen dynamisch während der Laufzeit erweitert und zum Schluss in ein Array umgewandelt werden können. Zudem wird eine Variable des Typs *BufferedReader* mit der Textdatei über den Dateinamen ***datei*** (als Argument übergeben) als String-Literal initialisiert (die Textdatei wird so geöffnet; falls dies nicht möglich ist, wird eine Ausnahme ausgelöst, die in er Main-Methode behandelt wird), damit die einzelnen Textzeilen von der Textdatei eingelesen werden können. Außerdem werden zwei Variablen ***zeile*** und ***schülername*** des Datentyps *String* erstellt, die jeweils die Textzeile und den Namen einer Schülerin enthalten. Zuletzt wird die Integer-Variable ***indexSchüler*** (bezeichnet die Stelle der Schülerin in der Schülerliste) erstellt und mit 0 initialisiert.

In einer while-Schleife wird in der Schleifenbedingung die Textzeile in der Variablen ***zeile*** eingelesen und dabei gleichzeitig überprüft, ob das Ende der Textdatei erreicht wurde („null“ wird der Variablen ***zeile*** zugewiesen). Innerhalb dieser Schleife werden zu Beginn die temporäre Plus-/Minus-Liste und die Variable ***schülername*** „geleert“, damit bei jedem Schleifendurchlauf sowohl eine neue Plus-/Minus-Liste als auch ein neuer Schülername gebildet werden kann. Nun folgen vier if-Abfragen:

1.

Ist die Textzeile leer, so weiß man, dass eine neue Schülerin aufgenommen werden muss. Deshalb erweitert man die Schülerliste um eine neue Schülerin und ***indexSchüler*** wird um eins erhöht, um zur nächsten Schülerin überzugehen.

2.

Ist das erste Zeichen der Textzeile ein „+“, so werden in dieser Zeile die Schülernamen der Plus-Liste aufgelistet. Eine if-Abfrage zuvor überprüft, ob die Textlänge der Zeile größer als zwei ist, also ob die Plus-Liste nicht leer ist. Nur wenn dies der Fall ist, werden in einer for-Schleife alle Zeichen der Textzeile durchlaufen. In dieser Schleife wird mittels einer if-Abfrage kontrolliert, ob das aktuelle Zeichen ein Leerzeichen ist. Wenn dies zutrifft, weiß man, dass der Schülername vollständig in der Variablen ***schülername*** steht, sodass dieser in die Plus-/Minus-Liste aufgenommen werden kann. Man „leert“ ***schülername***, bevor man im nachfolgenden Schritt zur nächsten Schülerin über den continue-Befehl übergeht. Solange das Zeichen kein Leerzeichen ist, wird der Schülername um das aktuelle Zeichen erweitert. Sind alle Zeichen durchlaufen worden, wird ***schülername*** ein letztes Mal in die Plus-/Minus-Liste aufgenommen (dieser Extrabefehl versichert, dass auch die letzte Schülerin in die Schülerliste aufgenommen wird). Als letztes wird die Plus-Liste der Schülerin an der Stelle ***indexSchüler*** mit den Schülernamen der Plus-/Minus-Liste (die zuvor in ein Array umgewandelt wird) initialisiert.

3.

Ist das erste Zeichen der Textzeile ein „-“, so werden in dieser Zeile die Schülernamen der Minus-Liste aufgelistet. Es passiert hier nahezu dasselbe wie in der zweiten if-Abfrage. Der Unterschied ist, dass die Plus-/Minus-Liste nun die Minusliste der Schülerin darstellt.

4.

Fallen alle vorherigen if-Abfragen negativ aus, so weiß man, dass in der Textzeile der Namen der Schülerin steht. Dieser wird der Schülerliste an der Stelle ***indexSchüler*** hinzugefügt.

Nun ist die Schülerliste komplettiert worden. Deshalb wird die Textdatei geschlossen und dem Schüler-Array ***schüler*** wird die Schülerliste (zuvor in ein Array umgewandelt) zugewiesen.

- public void schülerlisteSortieren()

Das Schüler-Array wird so sortiert, dass zuerst die Schülerinnen mit nicht-leeren Plus-Listen an erster Stelle stehen So kann es keine Konflikte geben mit einer Schülerin, deren Plus-Liste leer ist und einem Einzelzimmer zugeordnet wird, obwohl die nächste Schülerin ein Zimmer mit ihr teilen möchte. Zum Sortieren wird der BubbleSort-Algorithmus angewendet: Zwei Schülerinnen tauschen ihre Plätze im Array, wenn die erste Schülerin eine leere, die zweite eine nicht-leere Plus-Liste besitzt.

- public void zimmerbelegungAusgeben()

Diese Methode ist dafür zuständig, dass die Zimmerbelegung in der Lösungsdatei ausgegeben wird. Dazu wird die Variable ***pWriter*** des Typs *PrintWriter* angelegt, um in die Lösungsdatei zu schreiben. In einem try-Block wird die Datei erst geöffnet und mit zwei for-Schleifen werden die Zimmer und die Schüler durchlaufen, sodass erst die Namen der Schülerinnen (abgetrennt durch ein Komma) nacheinander in die Datei geschrieben werden und erst dann ein Zeilenumbruch erfolgt. Im darauffolgenden catch-Block wird die Ausnahme so behandelt, dass eine Fehlermeldung in der Konsole ausgegeben wird. Zuletzt wird der Stream der Variablen *pWriter* geleert und die Datei geschlossen.

- public static void main(String[] args)

Die Main-Methode startet das gesamte Programm. Der Programmablauf besteht aus den folgenden Schritten:

1.

Initialisieren des Schüler-Arrays ***schüler***; Falls die Textdatei nicht geöffnet werden kann, wird eine Fehlermeldung über die Konsole ausgegeben.

2.

Sortieren des Schüler-Arrays ***schüler***.

3.

Belegen der Zimmer; Bei einem Wunschkonflikt werden die Namen der Schülerinnen, deren Wünsche konfligieren, über die Konsole ausgegeben. Falls kein Wunschkonflikt auftritt, wird die Belegung der Zimmer in der Lösungsdatei angegeben.

Um die main-Methode zu starten, müssen die Verzeichnisse der Schülerlisten und der Lösungsdatei eingegeben werden.

Beispiele

1.

Das erste Beispiel zeigt, wie es zu einem Wunschkonflikt kommt.

Mein Programm gibt folgendes aus: „Es gibt einen Wunschkonflikt, weil die Wünsche von Dani und Lotta konfligieren!“

Das Problem hierbei ist, dass Dani und Lotta mit derselben Person ein Zimmer teilen möchten, aber nicht gleichzeitig mit dem jeweils anderen. Dieser Wunschkonflikt kann nicht behoben werden.

2.

Es gibt keinen Wunschkonflikt. Die Ausgabe ist:

Zimmer 1

Alina, Lilli

Zimmer 2

Mia, Zoe, Emma

Zimmer 3

Lara

3.

Es gibt keinen Wunschkonflikt. Die Ausgabe ist:

Zimmer 1

Anna, Julia, Lisa, Emily, Sofia, Marie, Jana, Johanna, Carolin, Michelle, Emma, Larissa, Nele, Antonia

Zimmer 2

Lea, Celine, Lena, Lina, Clara

Zimmer 3

Sarah, Sophie, Alina, Josephine, Vanessa, Leonie, Lilli, Pia, Annika, Melina, Kim, Pauline, Katharina

Zimmer 4

Laura, Charlotte, Lara, Jasmin, Merle, Celina, Nina, Miriam, Luisa, Jessika

Zimmer 5

Hannah

Man merkt jetzt schon, dass die Zimmer sehr groß werden, weil sich die Wünsche sehr häufig überschneiden. Deshalb ist es nur sinnvoll, die Zimmer nicht mit Arrays sondern mit Listen dynamisch zu erstellen. Zum ersten Mal treten hier auch komische Eingaben seitens der Aufgabe auf. Beispielsweise möchte Nele mit sich selber ein Zimmer teilen, was ja sowieso gelten muss, da mein Programm nicht mit Schizophrenie umgehen kann.

4.

Es gibt keinen Wunschkonflikt. Die Ausgabe ist:

Zimmer 1

Anna, Julia, Miriam, Jessika, Jasmin, Pauline

Zimmer 2

Hannah, Celine, Annika, Pia, Lisa, Carolin, Celina, Lara, Nele, Michelle, Charlotte, Clara, Nina, Kim, Lina, Merle, Vanessa, Luisa, Emma

Zimmer 3

Lea, Laura, Katharina, Leonie, Larissa, Alina, Jana, Josephine, Lilli, Marie, Lena, Sarah, Emily, Sophie, Sofia, Johanna, Melina

Zimmer 4

Antonia

5.

Es gibt einen Wunschkonflikt aufgrund des vorher erwähnten Schizophrenie-Problems: Marie möchte nicht mit sich selber ein Zimmer teilen. Mein Programm kann mit diesem Problem nicht umgehen und gibt daher einen Wunschkonflikt aus. Ich frage mich auch hier schon, ob solche Eingaben überhaupt realistisch erscheinen.

6.

Zimmer 1

Anna, Carolin, Clara, Lena, Nina, Antonia, Leonie, Lina, Kim, Lisa, Nele, Luisa, Marie, Julia, Laura, Katharina, Pauline, Emma, Miriam, Lara, Jessika, Alina, Charlotte, Josephine, Pia

Zimmer 2

Hannah, Sophie, Sarah, Vanessa, Jana, Johanna, Melina, Lilli, Emily, Larissa, Celine

Zimmer 3

Lea, Michelle

Zimmer 4

Annika, Celina

Zimmer 5

Jasmin, Merle, Sofia

Quelltext

Schüler-Klasse:

/\*

\* Die Klasse "Schüler" dient zur Beschreibung der Schülerinnen.

\* Daher sind nur Setter/Getter existent.

\*/

public class Schüler

{

// Eigenschaften einer Schülerin:

// (Name, Plus-Liste, Minus-Liste, Zugeteilt-Sein Flag)

private String name;

private String[] teilen;

private String[] nichtTeilen;

private boolean zugeteilt;

/\*

\* Konstruktoren

\*/

public Schüler(String schülerName, String[] mitSchülerTeilen, String[] nichtMitSchülerTeilen)

{

// Instanzvariable initialisieren

this.name = schülerName;

this.teilen = mitSchülerTeilen;

this.nichtTeilen = nichtMitSchülerTeilen;

this.zugeteilt = false;

}

public Schüler()

{

// Instanzvariable initialisieren

this.name = "";

this.teilen = null;

this.nichtTeilen = null;

this.zugeteilt = false;

}

/\*

\* Setter- und Getter-Methoden

\*/

// Gibt den Namen der Schülerin zurück

public String getName()

{

return this.name;

}

// Legt den Namen der Schülerin fest

public void setName(String name)

{

this.name = name;

}

// Gibt den Namen einer Schülerin der Plus-Liste zurück

public String getTeilen(int stelle)

{

return this.teilen[stelle];

}

// Legt die Plus-Liste fest

public void setTeilen(String[] teilen)

{

this.teilen = teilen;

}

// Gibt die Anzahl der Schülerinnen in der Plus-Liste zurück

public int getTeilenGröße()

{

return this.teilen.length;

}

// Überprüft, ob die Plus-Liste leer ist

public boolean istTeilenNull()

{

return this.teilen == null;

}

/\*

\* Jetzt folgen dieselben Methoden, nur dass die Plus-Liste

\* mit der Minus-Liste ausgetauscht wurde

\*/

public String getNichtTeilen(int stelle)

{

return this.nichtTeilen[stelle];

}

public void setNichtTeilen(String[] nichtTeilen)

{

this.nichtTeilen = nichtTeilen;

}

public int getNichtTeilenGröße()

{

return this.nichtTeilen.length;

}

public boolean istNichtTeilenNull()

{

return this.nichtTeilen == null;

}

/\*

\* Nun folgen Methoden, um den Zugeteilt-Sein-Flag

\* benutzen zu können.

\*/

// Die Schülerin wird einem Zimmer zugeteilt.

void wirdZugeteilt()

{

this.zugeteilt = true;

}

// Überprüft, ob die Schülerin bereits einem Zimmer

// zugeteilt wurde.

public boolean istZugeteilt()

{

return this.zugeteilt;

}

}

Zimmerbelegung-Klasse:

/\*

\* Folgende Klasse dient zur Fertigung und Überprüfung

\* der Zimmerbelegung. Innerhalb dieser Klasse wird die

\* Aufgabe gelöst.

\*/

/\*

\* Hier werden Klassen importiert zur Erweiterung der

\* der Funktionalität meines Programms.

\*/

// Dynamische Listen (Listen während der Laufzeit verändern)

import java.util.ArrayList;

// Auslesen von Textdateien

import java.io.FileReader;

// Reinschreiben in Textdateien

import java.io.FileWriter;

// Verbessert Effizienz von FileReader durch Buffer

import java.io.BufferedReader;

// Verbessert Effizienz von FileWriter durch Buffer

import java.io.BufferedWriter;

// Ausnahmebehandlung (Fehler beim Bearbeiten der Textdatei)

import java.io.IOException;

// Erweitert Funktion von BufferedWriter (println())

// Der Code wird dadurch besser lesbar.

import java.io.PrintWriter;

public class Zimmerbelegung

{

// Informationen zur Fertigung einer Zimmerbelegung

// Schülerliste, Zimmerliste (enthält nur die Namen der Schülerinnen),

private Schüler[] schüler;

private ArrayList<ArrayList<String>> zimmer;

/\*\*

\* Konstruktoren

\*/

public Zimmerbelegung(Schüler[] schülerListe)

{

// Instanzvariable initialisieren

schüler = schülerListe;

zimmer = new ArrayList<ArrayList<String>>();

}

public Zimmerbelegung()

{

// Instanzvariable initialisieren

schüler = null;

zimmer = new ArrayList<ArrayList<String>>();

}

// Gibt die Zimmernummer (beginnend ab 0) zurück, bei welcher

// die Schülerin gefunden wurde bzw. -1, wenn die Schülerin

// noch nicht zugeordnet wurde.

public int zimmernummerVonSchüler(Schüler s)

{

for(int indexZimmer = 0; indexZimmer < zimmer.size();indexZimmer++)

// Taucht der Name der Schülerin in einem der Zimmer auf?

// Wenn ja, wird die Stelle des Funds zurückgegeben

if(zimmer.get(indexZimmer).indexOf(s.getName()) != -1)

return indexZimmer;

// Nur überprüfen, wenn die Plus-Liste nicht leer ist

if(!s.istTeilenNull())

{for(int indexSchülerAusPlusliste = 0; indexSchülerAusPlusliste < s.getTeilenGröße(); indexSchülerAusPlusliste++)

for(int indexZimmer = 0; indexZimmer < this.zimmer.size(); indexZimmer++)

// Taucht der Name einer Schülerin aus der Plus-Liste auf?

// Wenn ja, wird die Stelle des Funds zurückgegeben

if(this.zimmer.get(indexZimmer).indexOf(s.getTeilen(indexSchülerAusPlusliste)) != -1)

return indexZimmer;

}

// Keine Schülerin taucht auf

return -1;

}

// Überprüft, ob die Schülerin oder eine Schülerin aus ihrer Plus-Liste

// in einem beliebigen Zimmer vorhanden ist.

public boolean istSchülerInEinemZimmer(Schüler s, ArrayList<String> schülerImZimmer)

{

// Taucht der Name der Schülerin in einem der Zimmer auf?

// Wenn ja, wird die Stelle des Funds zurückgegeben.

// Überprüft, ob die Schülerin vorhanden ist.

if(schülerImZimmer.indexOf(s.getName()) != -1)

return true;

if(!s.istTeilenNull())

{

for(int indexSchülerAusPlusliste = 0; indexSchülerAusPlusliste < s.getTeilenGröße(); indexSchülerAusPlusliste++)

// Überprüft, ob eine Schülerin aus der Plus-Liste vorhanden ist.

if(schülerImZimmer.indexOf(s.getTeilen(indexSchülerAusPlusliste)) != -1)

return true;

}

// Keine der vorherigen, überprüften Schülerinnen ist vorhanden.

return false;

}

// Die Zimmerbelegung wird kreiert.

public void zimmerBelegen()

{

// Ein Statusflag, der auf true gesetzt wird, wenn eine Schülerin oder

// eine aus ihrer Plus-Liste in einem Zimmer gefunden wird.

boolean schalter;

for(int indexSchüler1 = 0; indexSchüler1 < schüler.length; indexSchüler1++)

{

// Die Schülerinnen, die leere Plus-Listen haben und noch keinem

// Zimmer zugeordnet wurden, bekommen ein Einzelzimmer für sich.

// Weil die Schülerliste vorher sortiert wurde, wird garantiert,

// dass diese Schülerin nicht bereits in einem vorherigen Zimmer

// vorkommt.

if(!schüler[indexSchüler1].istZugeteilt() && schüler[indexSchüler1].istTeilenNull())

{

// Ein neues Zimmer wird hinzugefügt.

zimmer.add(new ArrayList<String>());

// Die Schülerin wird dem Einzelzimmer zugeordnet.

zimmer.get(zimmer.size()-1).add(schüler[indexSchüler1].getName());

continue;

}

// Wenn die Schülerin schon einem Zimmer zugeordnet wurde, wird

// zu einer neuen Schülerin übergegangen.

if(schüler[indexSchüler1].istZugeteilt())

continue;

// Falls die Schülerin nicht zugeordnet wurde, werden sie und

// diejenigen aus ihrer Plus-Liste einem neuen Zimmer zugeordnet.

zimmer.add(new ArrayList<String>());

// Die Schülerin wird hinzugefügt.

zimmer.get(zimmer.size()-1).add(schüler[indexSchüler1].getName());

for(int indexSchülerAusPlusliste1 = 0; indexSchülerAusPlusliste1 < schüler[indexSchüler1].getTeilenGröße(); indexSchülerAusPlusliste1++)

{

if(!zimmer.get(zimmer.size()-1).contains(schüler[indexSchüler1].getTeilen(indexSchülerAusPlusliste1)))

// Die Schülerinnen aus der Plus-Liste werden hinzugefügt.

zimmer.get(zimmer.size()-1).add(schüler[indexSchüler1].getTeilen(indexSchülerAusPlusliste1));

}

// Die Schülerin kann jetzt in einem neuen Schleifendurchgang

// nicht mehr hinzugefügt werden.

schüler[indexSchüler1].wirdZugeteilt();

do

{ // Noch wurde keine Schülerin im Zimmer gefunden.

schalter = false;

for(int indexSchüler2 = 0; indexSchüler2 < schüler.length; indexSchüler2++)

{

// Wurde eine Schülerin zugeteilt, soll sie nicht nochmals

// einem Zimmer hinzugefügt werden.

if(schüler[indexSchüler2].istZugeteilt())

continue;

// Wird Schülerin oder eine aus ihrer Plus-Liste gefunden:

if(istSchülerInEinemZimmer(schüler[indexSchüler2],zimmer.get(zimmer.size()-1)))

{

schüler[indexSchüler2].wirdZugeteilt();

if(!zimmer.get(zimmer.size()-1).contains(schüler[indexSchüler2].getName()))

// Nur wenn der Name der Schülerin nicht bereits im Zimmer vorkommt, soll er

// hinzugefügt werden.

zimmer.get(zimmer.size()-1).add(schüler[indexSchüler2].getName());

// Überspringt leere Plus-Listen

if(!schüler[indexSchüler2].istTeilenNull())

{for(int indexSchülerAusPlusliste2 = 0; indexSchülerAusPlusliste2 < schüler[indexSchüler2].getTeilenGröße(); indexSchülerAusPlusliste2++)

if(!zimmer.get(zimmer.size()-1).contains(schüler[indexSchüler2].getTeilen(indexSchülerAusPlusliste2)))

// Wenn der Name einer Schülerin aus der Plus-Liste nicht bereits im Zimmer vorkommt,

// wird er hinzugefügt.

zimmer.get(zimmer.size()-1).add(schüler[indexSchüler2].getTeilen(indexSchülerAusPlusliste2));

}

// Eine Schülerin wurde gefunden. Daher muss die Schleife

// erneut durchlaufen werden, um die restlichen hinzuzufügen.

schalter = true;

}

}

// Solange eine Schülerin im Zimmer gefunden wird, wird die Schleife

// wiederholt.

}while(schalter);

}

}

// Die Belegung wird überprüft.

// Existieren Wunschkonflikte,

// so werden die Namen der Schülerinnen zurückgegeben, deren

// Wünsche konfligieren

public String zimmerbelegungÜberprüfen()

{// Zimmerbelegung wird anhand der Minus-Liste überprüft für jede Schülerin

for(int indexSchüler = 0; indexSchüler<schüler.length;indexSchüler++)

{

// Soll nur durchgeführt werden, wenn die Minus-Liste nicht leer ist

if(!schüler[indexSchüler].istNichtTeilenNull())

{

// Zimmernummer, in welchem die Schülerin vorkommt

int zimmernummer = zimmernummerVonSchüler(schüler[indexSchüler]);

for(int indexSchülerAusMinusliste = 0; indexSchülerAusMinusliste < schüler[indexSchüler].getNichtTeilenGröße();indexSchülerAusMinusliste++)

{

// Wenn eine Schülerin aus der Minus-Liste vorkommt, dann sollen die

// die Schülerin selbst und die Schülerin aus der Minus-Liste zurückgegeben werden

if(zimmer.get(zimmernummer).contains(schüler[indexSchüler].getNichtTeilen(indexSchülerAusMinusliste)))

{

return schüler[indexSchüler].getName() + " und " + schüler[indexSchüler].getNichtTeilen(indexSchülerAusMinusliste);

}

}

}

}

// Zimmerbelegung hat geklappt.

return "";

}

// Extrahiert die Daten der Schülerinnen aus zimmerbelegung.txt

// und speichert sie in das Schüler-Array ein

public void setSchülerliste(String datei) throws IOException

{

// Eine temporäre Schülerliste wird erstellt

// und eine Schülerin wird dieser Liste hinzugefügt

// (Arrayliste kann dynamisch während der Laufzeit erweitert werden)

ArrayList<Schüler> schülerListe = new ArrayList<Schüler>();

schülerListe.add(new Schüler());

// zimmerbelegung.txt wird geöffnet;

// Textzeilen können eingelesen werden

BufferedReader br = new BufferedReader(new FileReader(datei));

// Inhalt der Textzeile aus der Datei

String zeile = "";

int indexSchüler = 0;

// Der Schülername, der aus der Textzeile extrahiert wird

String schülername;

// Liste, die die Schülernamen aus jeweils einer Liste (Minus-/Plus-Liste)

// enthält

ArrayList<String> schülernameListe = new ArrayList<String>();

// Solange das Zeilenende der Datei erreicht wird, soll die Textzeile

// eingelesen werden

while( (zeile = br.readLine()) != null)

{

// Liste der Schülernamen wird gelöscht, damit die neue

// Liste (Minus-/Plus-Liste) aufgenommen werden kann

schülernameListe.clear();

// Ein neuer Schülernamen soll aufgenommen werden

schülername = "";

//Schülerinnen werden durch einen Zeilenumbruch voneinander

// getrennt

if(zeile.equals(""))

{

// Fügt eine neue Schülerin hinzu und geht deswegen

// zur nächsten Schülerin über

schülerListe.add(new Schüler());

indexSchüler++;

}

// Das "+"-Zeichen markiert eine Zeile, in der die Namen

// der Plus-Liste vorkommen; die Plus-Liste wird erstellt

else if(zeile.charAt(0)== '+')

{

// Überspringt leere Plus-Listen

if(zeile.length() > 2){

for(int indexBuchstabe = 2; indexBuchstabe < zeile.length(); indexBuchstabe++)

{

//Schülerinnen werden durch ein Leerzeichen voneinander

// getrennt

if(zeile.charAt(indexBuchstabe) == ' ')

{

// Die Schülerin aus der Plus-Liste wird der Schülerliste

// hinzugefügt und ein neuer Schülernamen soll aufgenommen werden

schülernameListe.add(schülername);

schülername = "";

// Es wird zur nächsten Schülerin übergegangen

continue;

}

// Der Schülername wird aus der Zeile extrahiert

schülername += zeile.charAt(indexBuchstabe);

}

// Dieser Extrabefehl versichert, dass auch die letzte Schülerin

// in die Schülerliste aufgenommen wird

schülernameListe.add(schülername);

// Die Plus-Liste wird der Schülerliste hinzugefügt

schülerListe.get(indexSchüler).setTeilen(schülernameListe.toArray(new String[0]));

}

}

// Das "-"-Zeichen markiert eine Zeile, in der die Namen

// der Minus-Liste vorkommen; die Minus-Liste wird erstellt;

// Derselbe Aufbau wie der vorherige Programmblock zur Erstellung

// der Plus-Liste, nur das die Plus-Liste mit der Minus-Liste

// ausgetauscht wurde

else if(zeile.charAt(0) == '-')

{

if(zeile.length() > 2){

for(int indexBuchstabe = 2; indexBuchstabe < zeile.length(); indexBuchstabe++)

{

if(zeile.charAt(indexBuchstabe) == ' ')

{

schülernameListe.add(schülername);

schülername = "";

continue;

}

schülername += zeile.charAt(indexBuchstabe);

}

schülernameListe.add(schülername);

schülerListe.get(indexSchüler).setNichtTeilen(schülernameListe.toArray(new String[0]));

}

}

// Fügt den Namen der Schülerin zur der Schülerliste hinzu

else

{

schülerListe.get(indexSchüler).setName(zeile);

}

}

// Die zimmerbelegung.txt - Datei wird geschlossen, da

// sie nun nicht mehr benötigt wird

br.close();

// Das Schüler-Array wird hier erstellt;

// Dazu wird der Inhalt der Schülerliste reinkopiert

schüler = schülerListe.toArray(new Schüler[0]);

}

// Das Schüler-Array wird so sortiert, dass zuerst die

// Schülerinnen mit nicht-leeren Plus-Listen an erster Stelle

// stehen

public void schülerlisteSortieren()

{

// Eine Schülerin wird für den folgenden Tausch reinkopiert

Schüler schülerkopie;

for(int indexSchüler1=1; indexSchüler1<schüler.length; indexSchüler1++) {

for(int indexSchüler2=0; indexSchüler2<schüler.length-indexSchüler1; indexSchüler2++) {

// Wenn die nächste Schülerin eine nicht-leere Plus-Liste hat

// und die vorherige eine leere, so tauschen sie

// ihre Plätze im Array

if(!schüler[indexSchüler2+1].istTeilenNull() && schüler[indexSchüler2].istTeilenNull()) {

schülerkopie=this.schüler[indexSchüler2];

this.schüler[indexSchüler2]=this.schüler[indexSchüler2+1];

this.schüler[indexSchüler2+1]=schülerkopie;

}

}

}

}

// Die Zimmerbelegung wird ausgegeben

public void zimmerbelegungAusgeben(String lösungsdatei)

{

// Vorbereitung des Writers, damit in die Lösungsdatei

// geschrieben werden kann

PrintWriter pWriter = null;

try {

// Die Lösungsdatei soll jetzt beschrieben werden

pWriter = new PrintWriter(new BufferedWriter(new FileWriter(lösungsdatei)));

for(int indexZimmer = 0; indexZimmer < zimmer.size(); indexZimmer++)

{

// Zimmernummer ausgeben

pWriter.println("Zimmer " + Integer.toString(indexZimmer+1));

for(int indexSchülerAusZimmer = 0; indexSchülerAusZimmer < zimmer.get(indexZimmer).size(); indexSchülerAusZimmer++)

// Die Schülernamen werden durch ein Komma voneinander getrennt;

{

// Damit nicht ein zusätzliches Komma nach der letzten Schülerin steht

if(!(indexSchülerAusZimmer == zimmer.get(indexZimmer).size() - 1))

pWriter.print(zimmer.get(indexZimmer).get(indexSchülerAusZimmer) + ", ");

else

pWriter.print(zimmer.get(indexZimmer).get(indexSchülerAusZimmer));

}

// Die Zimmer werden durch ein Zeilenumbruch getrennt

pWriter.println("");

}

}

// Falls das Schreiben in die Lösungsdatei nicht funktioniert,

// wird eine Fehlermeldung ausgegeben

catch (IOException ioe) {

System.out.println("Die Zimmerbelegung kann nicht ausgegeben werden!");

}

// Leeren des Streams und Schließen der Lösungsdatei

finally {

if (pWriter != null){

pWriter.flush();

pWriter.close();

}

}

}

// Main-Methode

public static void main(String[] args)

{

Zimmerbelegung belegung = new Zimmerbelegung();

try{

// Die Schülerliste wird erstellt

belegung.setSchülerliste(args[0]);

}

// Falls die Datei nicht geöffnet werden kann, wird eine Fehlermeldung ausgegeben

// und das Programm wird beendet

catch(IOException e)

{

System.out.println("Datei konnte nicht geöffnet werden!");

return;

}

// Die Schülerliste wird sortiert

belegung.schülerlisteSortieren();

// Die Zimmerbelegung wird erstellt und überprüft

belegung.zimmerBelegen();

String belegungNichtMöglich = belegung.zimmerbelegungÜberprüfen();

// Wenn die Zimmerbelegung nicht möglich ist, dann soll auch der

// Grund der Unmöglichkeit ausgegeben werden

if(!belegungNichtMöglich.equals(""))

System.out.println("Es gibt einen Wunschkonflikt, weil die Wünsche von " + belegungNichtMöglich + " konfligieren!");

else

{

// Ausgabe der Zimmerbelegung in der Lösungsdatei

belegung.zimmerbelegungAusgeben(args[1]);

}

}

}