1. Aufgabe
„Zimmerbelegung“- Dokumentation

36. Bundeswettbewerb Informatik 2017/18 - 1.Runde

Sebastian Baron, Simon Fiebich, Lukas Rost und Max Woithe

Team-ID: 00020

27. November 2017

Inhaltsverzeichnis

1 Lösungsidee

Die Lösungsidee lässt sich am besten folgendermaßen formulieren:

Tue für jede Schülerin:

1. Hat diese einen oder mehr Namen auf der Plus-Liste?
   a) Ja: Ist die Schülerin oder jemand von ihrer Plus-Liste schon einem Zimmer zugeteilt?
      i. Ja:
         A. Sind die Personen der Plus-Liste in unterschiedlichen Zimmern? (Ja: Füge diese Zimmer zu einem neuen zusammen [1])
         B. Teile sie und alle Personen ihrer Plus-Liste diesem (bei (A) dem neu erstellten) Zimmer zu
      ii. Nein: Teile sie und alle Personen ihrer Plus-Liste einem neuen Zimmer zu
   b) Überprüfe, ob zwischen Zimmerbelegung und den Minuslisten der Leute auf dem gerade „bearbeiteten“ Zimmer Konflikte bestehen
      i. Ja: Abbruch
      ii. Nein: Mache mit der nächsten Schülerin weiter

Tue für jede Schülerin:

1. Hat diese leere Plus-Liste und ist diese noch in keinem Zimmer?
   a) Teile sie einem Einzelzimmer zu
2. Hat diese sich selbst auf der Minus-Liste?
   a) Ja: Abbruch[2]

Tue für jedes entstandene Zimmer [3]:

1. Überprüfe, ob kein Schüler, der auf der Minus-Liste eines der Schüler im Zimmer steht, sich in diesem Zimmer befindet
2. Überprüfe, ob sich alle Schüler, die auf der Plus-Liste eines der Schüler im Zimmer stehen, in diesem Zimmer befinden

Diese Idee entstand aus der Überlegung, dass es sinnvoll ist, zuerst von den Plus-Listen auszugehen und zu versuchen, deren Anforderungen zu erfüllen. Wenn bei diesem Versuch Konflikte zwischen den Minus-Listen der Schüler auf einem Zimmer und der Belegung dieses Zimmers auftreten, weiß man, dass eine Zimmerbelegung nicht möglich ist, ohne mindestens eine Anforderung durch die Plus-Listen nicht erfüllen zu können.
Anschließend, wenn alle Bedingungen durch die Plus-Listen ohne Probleme erfüllt wurden und noch Schüler ohne Zimmer übrig sind (welche dann im allgemeinen eine leere Plus-Liste haben müssen), müssen diese auch noch auf Zimmer aufgeteilt werden. Die einfachste Methode, dies zu erreichen, ist, ihnen Einzelzimmer zuzuteilen, was gefahrlos durchgeführt werden kann, da ja diese Schüler mit keinem anderen Schüler ein Zimmer

[1] siehe Sonderfall 1
[2] siehe Sonderfall 2
[3] optional, dient der Verifikation des Ergebnisses

unbedingt teilen wollen und auch von keinem Schüler auf dessen Plus-Liste angefordert werden. Menschlich mag diese Lösung suboptimal sein; sie stellt aber für die folgende Implementierung eine nützliche Vereinfachung dar.
Ein Ansatz, der zuerst von den Minus-Listen ausginge, wäre hierbei untauglich, da es genügen würde, jedem Schüler ein Einzelzimmer zuzuweisen, um die Anforderungen der Minus-Listen zu erfüllen. Die Plus-Listen müssten dann wieder mittels des oben genannten Algorithmus berücksichtigt werden und insgesamt würde dies nur den Programmcode verlängern.

Der Algorithmus muss zwei Sonderfälle beachten:

1. Die Personen auf der Plus-Liste eines Schülers sind bereits unterschiedlichen Zimmern zugeteilt:
   Die einzige Lösung, um diesen Fall zu beheben und der Plus-Liste dieses Schülers zu genügen, ist, all diese Zimmer zu einem zu vereinigen.

2. Eine Person hat auf ihrer Minus-Liste sich selbst angegeben:
   Dies weist entweder auf einen Fehler in den Eingabedaten hin oder schlimmstenfalls darauf, dass diese Person schizophren ist bzw. mehrere Persönlichkeiten besitzt. Das Programm sollte deshalb abgebrochen werden.

Bezüglich der Zeitkomplexität und Schnelligkeit des Algorithmus sei darauf hingewiesen, dass dieser für Beispiele mit normalen Klassen- bzw. Klassenstufengrößen in unseren Versuchen nur wenige Sekunden benötigte.

2 Umsetzung

Das Programm wurde in der Programmiersprache Java umgesetzt und ist konsolenbasiert. Es besteht dabei aus drei Klassen:

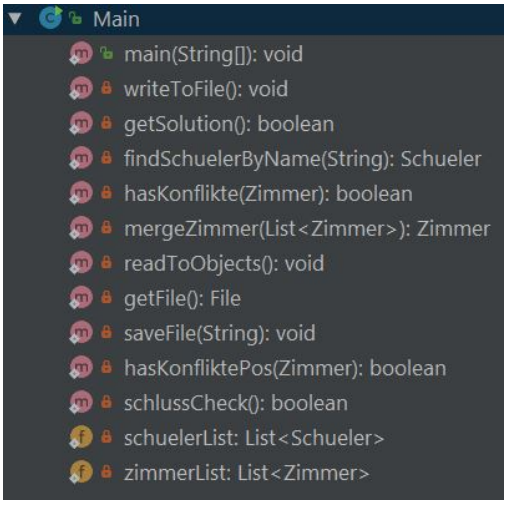

Abbildung 1: `Funktionen und Klassenvariablen der` *`Main`*`-Klasse`

Die *Schueler*-Klasse repräsentiert im Programm einen Schüler. Sie dient dazu, die drei zusammengehörigen Angaben Name (als *String*), Plus-Liste und Minus-Liste (beide als *ArrayList* von *Strings*) in einer gemeinsamen Datenstruktur bzw. einem gemeinsamen Objekt zu speichern. Sie bietet nur Getter- und Setter-Funktionen für diese drei Variablen, wobei die Setter bei den Listen durch Funktionen zum Hinzufügen ersetzt wurden.

Die *Zimmer*-Klasse repräsentiert ein Zimmer. Sie speichert die Liste der Mitglieder des Zimmers (als *ArrayList* von *Strings*) und bietet Funktionen zum Auslesen dieser und zum Hinzufügen zu ihr. Weiterhin ist eine Hilfsfunktion (`isInZimmer()`) enthalten, die dazu dient, anhand dessen Namen herauszufinden, ob ein bestimmter Schüler Mitglied des Zimmers ist.

Die dritte Klasse ist die Main-Klasse, die ausschließlich aus statischen Funktionen besteht und für den eigentlichen Ablauf des Programms verantwortlich ist. Diese besitzt zwei wichtige Klassenvariablen: die schuelerList, die nach dem Einlesen alle Schüler als Objekte enthält, und die zimmerList, die die im Programmablauf erzeugten Zimmer speichert und am Ende die endgültige Zimmeraufteilung (sofern möglich) enthält.

Sie enthält folgende Funktionen:

- `getFile()` fragt vom Benutzer mithilfe eines *JFileChooser* die einzulesende Datei ab und gibt diese als *File* zurück.

- `readToObjects()` liest die von `getFile()` übergebene Datei mithilfe eines *BufferedReader* in einer while-Schleife ein. Dabei erfolgt über die Variable *lineNumber* die Entscheidung, welchen Typs die aktuell eingelesene Zeile ist (1:Name, 2:Plus-Liste, 3:Minus-Liste, 4:Leerzeile) und *index* speichert, um den wievielten Schüler es sich aktuell handelt.

- `getSolution()` enthält die Hauptschleife(n) des Programms. Diese ist weitestgehend selbsterklärend und folgt dem in der Lösungsidee beschriebenen Algorithmus, dennoch hierzu einige Bemerkungen: Die Variable *jemandVonPlusListInZimmer* speichert, ob mindestens ein Schüler von der Plus-Liste einem Zimmer zugeteilt wurde. In der Liste *zimmerVonPlusListLeuten* wird für jeden solchen Schüler sein Zimmer abgelegt.
  Diese wird in Zeile 19 zur Überprüfung, wie viele unterschiedliche Zimmer es darunter gibt, in einen *Stream* umgewandelt und mittels `distinct()` werden nur noch unterschiedliche Zimmer behalten, die dann mittels `count()` gezählt werden. In der nächsten Zeile läuft ein ähnlicher Prozess noch einmal ab, nur dass der Stream danach wieder mittels `collect(Collectors.toList())` in eine Liste umgewandelt wird. Diese wird `mergeZimmer()` übergeben. Die Variable *currentZimmer* speichert das aktuell bearbeitete Zimmer, u.a. zur späteren Prüfung mit `hasKonflikte()`.
  In der zweiten Schleife speichert *isinAZimmer* für jeden Schüler, ob dieser schon einem Zimmer zugeordnet ist. Für den oben beschriebenen zweiten Sonderfall wird eine entsprechende Meldung auf der Konsole ausgegeben.

- `mergeZimmer()` ist eine Hilfsfunktion, die dafür verantwortlich ist, mehrere Zimmer zu einem zu vereinigen. Dazu wird ein neues Zimmer erstellt und alle Mitglieder der alten zu diesem hinzugefügt. Anschließend werden die Änderungen in die *zimmerList* übernommen und das neue Zimmer zurückgegeben.

- `hasKonflikte()` überprüft für ein gegebenes Zimmer, ob Konflikte zwischen den Minuslisten der Mitglieder und der Zimmerbelegung bestehen. Dabei wird für jeden Schüler in diesem Zimmer überprüft, ob irgendein Mitglied seiner Minus-Liste in dem Zimmer ist. Dann wird *true* zurückgegeben, sonst *false*. Außerdem wird die durch `findSchuelerByName()` geworfene *Exception* abgefangen.

- `hasKonfiktePos()` erfüllt eine ähnliche Aufgabe wie `hasKonflikte()`, jedoch bezogen auf die Überprüfung, ob alle Wünsche der Plus-Listen erfüllt wurden. Die Funktion wird nur aus `schlussCheck()` aufgerufen.

- `schlussCheck()` ruft `hasKonflikte()` und `hasKonfliktePos()` für jedes Zimmer auf und überprüft somit, ob alle Zimmer die Anforderungen erfüllen. Diese Funktion dient zur Verifikation des Endergebnisses.

- `findSchuelerByName()` gibt mithilfe der *schuelerList* zu einem gegebenen Namen das entsprechende Schüler-Objekt zurück. Schlägt dies fehl, wird eine *Exception* geworfen. Dieser Fall tritt jedoch nie auf, sofern die Eingabedaten dem erwarteten Format entsprechen.

- `writeToFile()` wandelt die *zimmerList* mithilfe eines *StringBuilder* in eine für die Ausgabe geeignete Form um. Dabei wird, wie vom Bwinf empfohlen, pro Zimmer eine Zeile verwendet und die einzelnen Schüler durch Kommas getrennt.

- `saveFile()` speichert die Ausgabedatei an einen vom Benutzer per *JFileChooser* ausgewählten Ort und benutzt dazu einen *PrintWriter*.

- Die immer zuerst aufgerufene Funktion `main()` startet die Funktionen für Eingabe, Verarbeitung, Verifikation des Ergebnisses und (falls nötig) Ausgabe.

3 Beispiele und Ausgaben

Ausgabe für Beispiel 1: Zimmerbelegung nicht möglich

Ausgabe für Beispiel 2

```
Alina, Lilli
Mia, Zoe, Emma
Lara

```

Ausgabe für Beispiel 3

```
Lea, Celine, Lena, Lina, Clara
Sarah, Sophie, Alina, Josephine, Vanessa, Leonie, Lilli, Pia, Annika, Melina, Kim,
↪  Pauline, Katharina
Laura, Charlotte, Lara, Jasmin, Merle, Celina, Nina, Miriam, Luisa, Jessika
Anna, Julia, Lisa, Emily, Sofia, Marie, Jana, Johanna, Carolin, Nele, Michelle, Nele,
↪  Antonia, Emma, Larissa
Hannah

```

Ausgabe für Beispiel 4

```
Anna, Julia, Miriam, Jessika, Jasmin, Pauline
Lea, Laura, Katharina, Leonie, Larissa, Alina, Jana, Josephine, Lilli, Sarah, Marie,
↪  Johanna, Lena, Melina, Emily, Sophie, Sofia
Hannah, Celine, Annika, Hannah, Pia, Lisa, Carolin, Celina, Lara, Nele, Michelle,
↪  Charlotte, Emma, Luisa, Nina, Clara, Kim, Lina, Merle, Vanessa
Antonia

```

Ausgabe für Beispiel 5: Zimmerbelegung nicht möglich

Ausgabe für Beispiel 6

```
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
Lea, Michelle
Annika, Celina
Jasmin, Merle, Sofia
Hannah, Sophie, Sarah, Vanessa, Jana, Johanna, Melina, Lilli, Emily, Celine, Larissa

```

eigenes Beispiel 7 (selbst_zimmerbelegung7.txt)

```
Anna
+ Berta Carolin
- Marie

Berta
+ Anna
- Tina

Tina
+ Marie
- Irene

Marie
+ Tina
- Berta

Carolin
+ Berta
- Irene

Irene
+
-
```

Ausgabe für Beispiel 7

```
Anna, Berta, Carolin
Tina, Marie
Irene

```

Bei den Beispielen fällt auf, dass Bsp. 5 den Sonderfall, dass ein Schüler mit sich selbst nicht auf einem Zimmer sein will, beinhaltet. Die Beispiele 3,4 und 6 beinhalten den Sonderfall der Zuteilung zu unterschiedlichen Zimmern. Soweit für uns ersichtlich, sind die Ausgaben zu allen Beispielen korrekt. Dies wurde auch durch das Programm verifiziert.

4 Quelltext

```
1  private static boolean getSolution() { //Hauptschleife zur Zimmeraufteilung
2      for (Schueler schueler: schuelerList) {
3          boolean jemandVonPlusListinZimmer = false;
4          List<Zimmer> zimmerVonPlusListLeuten = new ArrayList<>();
5          if (schueler.getPlusList().size() > 0){ //Plus-Liste vorhanden
6              for (Zimmer z: zimmerList) {
7                  if(z.isInZimmer(schueler.getName())){
8                      jemandVonPlusListinZimmer = true;
9                      zimmerVonPlusListLeuten.add(z);
10                 }
11                 for (String s:schueler.getPlusList()) {
12                     if(z.isInZimmer(s)){
13                         jemandVonPlusListinZimmer = true;
14                         zimmerVonPlusListLeuten.add(z);
15                     }
16                 }
17             }
18             Zimmer currentZimmer = (jemandVonPlusListinZimmer) ?
               ↪ zimmerVonPlusListLeuten.get(0) : null;
19             if (jemandVonPlusListinZimmer &&
               ↪ zimmerVonPlusListLeuten.stream().distinct().count() != 1){
               ↪ //Leute von Plus-Liste in untersch. Zimmern?
20                 currentZimmer = mergeZimmer(zimmerVonPlusListLeuten.
                   ↪ stream().distinct().collect(Collectors.toList()));
21             }
22             if (jemandVonPlusListinZimmer){ //in bestehendes Zimmer aufnehmen
23                 if(!currentZimmer.isInZimmer(schueler.getName())) { //Schueler
                   ↪ schon in diesem Zimmer?
24                     currentZimmer.addMitglied(schueler.getName());
25                 }
26                 for (String s:schueler.getPlusList()) {
27                     if(!currentZimmer.isInZimmer(s)) { //Schueler schon in
                       ↪ diesem Zimmer?
28                         currentZimmer.addMitglied(s);
29                     }
30                 }
31             } else{ //neues Zimmer erstellen
32                 currentZimmer = new Zimmer();
33                 currentZimmer.addMitglied(schueler.getName());
34                 for (String s:schueler.getPlusList()) {
35                     currentZimmer.addMitglied(s);
36                 }
37                 zimmerList.add(currentZimmer);
38             }
39             if (hasKonflikte(currentZimmer)){ //Konflikte mit Minus-Listen
               ↪ ueberpruefen
40                 return false;
41             }
42         }
43     }
```

```
    for (Schueler schueler: schuelerList){
        boolean isInAZimmer = false;
        for (Zimmer z: zimmerList) {
            if (z.isInZimmer(schueler.getName())) {
                isInAZimmer = true;
            }
        }
        if (schueler.getPlusList().size() == 0 && !isInAZimmer){
            Zimmer zi = new Zimmer();
            zi.addMitglied(schueler.getName());
            zimmerList.add(zi);
        }
        if (schueler.getMinusList().contains(schueler.getName())){
        ↪ //Sonderfall: Schueler will mit sich selbst nicht auf Zimmer sein
            System.out.println("Eingabedatenfehler oder schizophrener
            ↪ Schueler!");
            return false;
        }
    }
    return true;
}
```

Quelltext 1: Die *getSolution*-Funktion

```
private static boolean hasKonflikte(Zimmer zimmer){
    for (String schueler : zimmer.getMitglieder()) {
        try {
            for (String minusListenSchueler:
            ↪ findSchuelerByName(schueler).getMinusList()) {
                if (zimmer.isInZimmer(minusListenSchueler)){
                    return true;
                }
            }
        } catch (Exception e) {
            System.out.println("Schwerer Fehler:" + e.getLocalizedMessage() +
            ↪ "Klassenfahrt muss ausfallen!");
        }
    }
    return false;
}
```

Quelltext 2: Die *hasKonflikte*-Funktion

```
private static Zimmer mergeZimmer(List<Zimmer> zimmersToMerge){
    Zimmer mergedZimmer = new Zimmer();
    for (Zimmer z : zimmersToMerge) {
        for (String schueler: z.getMitglieder()) {
            mergedZimmer.addMitglied(schueler);
        }
    }
```

```
8       zimmerList.add(mergedZimmer);
9       zimmerList.removeAll(zimmersToMerge);
10      return mergedZimmer;
11  }
```

Quelltext 3: Die *mergeZimmer*-Funktion