36 Bundeswettbewerb Informatik
1. Runde / Aufgabe 1

DOKUMENTATION

Aufgabenstellung: Zimmerbelegung

Die Klasse 9c des Alan-Turing Gymnasiums, einer Mädchenschule, möchte eine Klassenfahrt unternehmen. Die wichtigste Frage dabei: Wer teilt sich mit wem ein Zimmer? Das ist leider gar nicht so einfach zu klären, denn die Wünsche der Schülerinnen können sich wiedersprechen.

Bei der letzten Klassenfahrt wollte zum Beispiel Anna gerne mit Paula ein Zimmer teilen, aber Paula nicht mit Anna. Und dann wollten sowohl Dani als auch Lotta gerne mit Steffi ein Zimmer teilen, aber auf keinen Fall miteinander. Da gab es natürlich keine Möglichkeit alle Wünsche zu erfüllen.

Die Klassenlehrerin der 9c möchte herausfinden, ob dieses Mal alle Wünsche erfüllt werden können und wie in diesem Fall die Zimmerbelegung aussieht. Hilf der Lehrerin!

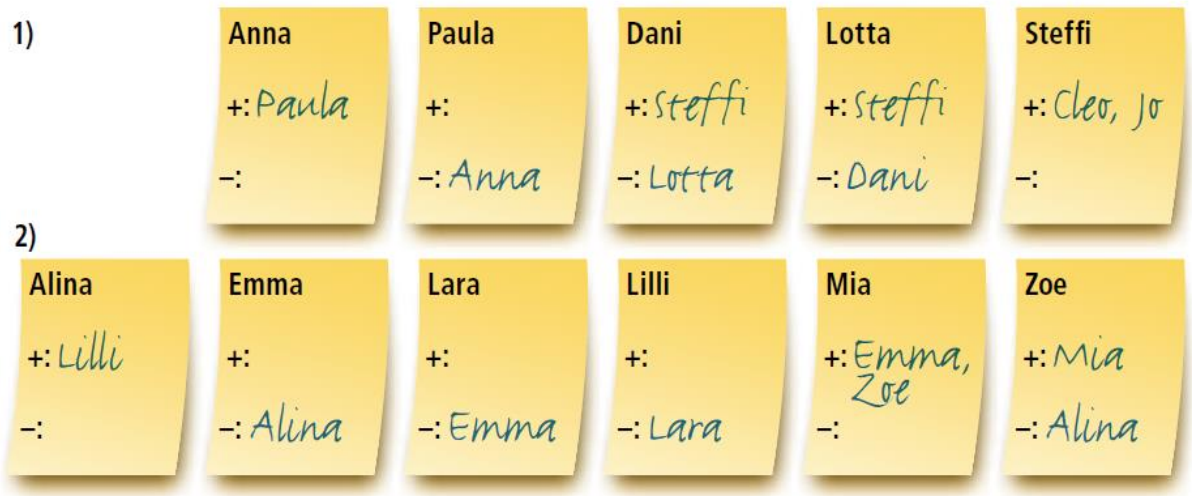
Aufgabe 1

Schreibe ein Programm, das ermittelt, ob alle Wünsche erfüllt werden können, wenn es genug Zimmer jeder Größe gibt. Als Eingabe erhält es für jede Schülerin zwei Listen der Mitschülerinnen, mit denen sie auf jeden Fall (+) bzw. auf keinen Fall (-) ein Zimmer teilen möchte.

Unten sind zwei Tests. In Beispiel 1 können nicht alle Wünsche erfüllt werden. Sehen die Angaben aber so aus wie in Beispiel 2, dann gelingt es.

Dein Programm soll ausgeben, ob eine Zimmerbelegung möglich ist, die alle Wünsche erfüllt. Falls ja, soll es zusätzlich eine solche Zimmerbelegung ausgeben. Zum Einstieg könntest du zunächst annehmen, dass jede Schülerin jede Mitschülerin in einer ihrer beiden Listen nennt.

Wende dein Programm auf die Tests an, die du auf den BwInf-Webseiten findest.



Lösungsidee

Als mögliche Lösungsansätze sind mir zwei grobe Methoden eingefallen. Zum einen könnte man alle Schülerinnen am Anfang in ein Zimmer setzen und sich dann für jede Schülerin, die Schülerinnen ansehen, mit denen jene nicht in ein Zimmer will.

Diesen Algorithmus habe ich aber nicht weiter ausgearbeitet, da ich mich für den anderen Ansatz entschieden habe. Zuerst wird jeder Schülerin ein eigenes Zimmer zugeordnet, dann kontrolliert man pro Schülerin die Schülerinnen, mit denen sie in ein Zimmer will und probiert die beiden Schülerinnen in dasselbe Zimmer zu verlegen. Zum Schluss überprüft man noch die Schülerinnen, die im ersten Teil nicht zugeordnet wurden (keine positive Präferenz für eine andere Schülerin und von keiner anderen Schülerin in ihr Zimmer gewünscht) und versucht diese noch anderen Zimmern zuzuordnen, um Einzelzimmer zu vermeiden.

Anfangs wird ein Textdokument eingelesen, welches die Schülerinnen und ihre jeweiligen Präferenzen enthält und am Ende wird die Zimmerkonstellation beziehungsweise eine Fehlermeldung über eine GUI ausgegeben.

Umsetzung

Das Programm wurde in der Programmiersprache Java mit der Entwicklungsumgebung IntelliJ IDEA 2017.2.5 von JetBrains entwickelt. Im Folgenden wird auf die einzelnen Klassen, ihre wichtigsten Attribute und Methoden, den Algorithmus zur Zimmerkonstellation und auf weitere Informationen zum Programm eingegangen.

Klasse: `Inhabitant`

Die Klasse `Inhabitant` (Bewohner) ist der Bauplan für eine Schülerin Instanz. Sie enthält zwei Listen vom Datentyp `Inhabitant` für die Schülerinnen, mit denen die Schülerin unbedingt (Attributbezeichner: `plusWishes`) beziehungsweise auf keinen Fall in ein Zimmer will (Attributbezeichner: `minusWishes`). Außerdem zwei String Attribute `forename` (Vorname) und `surename` (Nachname), die zur Identifizierung der Schülerinnen verwendet werden und ein Attribut der Klasse `Room`, das auf das Zimmer verweist, dem die Schülerin gerade zugewiesen ist.

Die Klasse enthält größtenteils nur Getter und Setter Methoden und eine überschriebene `equals()` Methode, um zwei `Inhabitant` Objekte anhand ihrer Attribute `forename` und `surename` zu vergleichen.

Klasse: `Room`

Die Klasse `Room` (Zimmer) enthält die Schülerinnen, die diesem Zimmer zugeordnet sind, und die Schülerinnen, die aufgrund der momentanen Bewohner nicht in das Zimmer dürfen, in zwei Listen des Typs `Inhabitant`.

Neben Gettern und Settern ist die wichtigste Methode der Klasse `Room` die Methode `checkOtherRoom()`, die als Parameter eine `Room` Instanz erwartet, und ein `Inhabitant` Objekt zurückgibt. Diese Methode überprüft, ob die `Inhabitant` Objekte, die den zwei `Room` Instanzen (dem, das die Methode ausführt und dem Übergebenen) sich gegenseitig nicht in ihrer `minusWishes` Liste haben und somit alle in ein Zimmer könnten. Ist dies nicht der Fall wird das `Inhabitant` Objekt zurückgegeben, das in der `minusWishes` Liste einer `Inhabitant` Instanz ist, ansonsten `null`.

Klasse: RoomAssignment

Diese Klasse, von der im Normalfall nur eine Instanz vorliegt, ist hauptsächlich für den Algorithmus der Zimmer Zuordnung verantwortlich. Sie beinhaltet eine Liste des Typs `Room` mit dem Bezeichner `allRooms`, die alle momentan verwendeten Zimmer referenziert und eine Liste `inhabitants` des Typs `Inhabitant`, die alle Schülerinnen, die die Textdatei fasst, enthält.

Des Weiteren beinhaltet die Klasse die Methode `searchInhabitant()`, die als Übergabewert einen String `name` und einen boolean `fullName` erwartet und ein `Inhabitant` Objekt zurückgibt. Diese Methode sucht in der Liste `inhabitants`, nach einer `Inhabitant` Instanz, deren Vorname (`fullName` ist `false`) beziehungsweise Vor- und Nachname (`fullName` ist `true`) gleich dem Übergebenen String `name` ist und gibt diese zurück. Wenn kein passendes Objekt gefunden wird, wird `null` zurückgegeben.

Die Methode `changeRooms()` mit den Parametern `Room oldRoom` und `Room newRoom` fügt der `inhabitants` Liste von `newRoom` alle `Inhabitant` Objekte aus der `inhabitants` Liste der `oldRoom` Instanz hinzu und entfernt anschließend das `oldRoom` Objekt aus der Liste `allRooms`.

`startRoomAssignment()` hat als Rückgabewert ein Objekt der Klasse `Exception`. Die Methode besteht aus einem `try` Block, in dem die Methoden `checkWishes()`, `checkAloneInhabitants()` ausgeführt und anschließend `null` zurückgegeben wird und einen `catch` Block, der ein Objekt der Klasse `Exception` als Übergabewert erwartet und dieses dann zurückgibt.
Mit dieser Methode wird der Algorithmus gestartet.

Die wichtigste Methode checkWishes(), die eine Exception werfen kann, ist Hauptverantwortlich für die verlangte Zimmerverteilung mit Berücksichtigung der Wünsche der Schülerinnen. Mit einer `foreach`-Schleife werden alle `Inhabitant` Objekte in der Liste `inhabitants` durchgegangen. Auf den aktuellen `Inhabitant` wird über `curInhabitantToCheck` verwiesen. Zuerst wird überprüft, ob das `curInhabitantToCheck` Objekt irgendein `Inhabitant` in seiner `plusWishes` Liste hat, der `curInhabitantToCheck` in seiner `minusWishes` Liste hat. Mit einer `for`-Schleife wird jedes `Inhabitant` Objekt, welches in der `plusWishes` Liste des `curInhabitantToCheck` ist durchgegangen:

```
for (Inhabitant plusWishInhabitant : inhabitantPlusWishes) {
        //Jede Schuelerin, die ein Plus Wunsch der aktuellen Schuelerin ist

        //Liste mit Minus Wuenschen der Schuelerin, die in der Plus Wunsch Liste der
        //aktuellen Schuelerin ist
        List<Inhabitant> plusWishInhabitantMinusWishes = plusWishInhabitant.getMinusWishes();

        for (Inhabitant p : plusWishInhabitantMinusWishes) {
            //Jede Schuelerin in der Minus Wunsch Liste der Schuelerin, die in der
            //Plus Wunsch Liste der aktuellen Schuelerin ist
            if (p.equals(curInhabitantToCheck)) {
                //Aktuelle Schuelerin ist in der Minus Wunsch Liste!
                //Exception
                throw new Exception(curInhabitantToCheck.getName() + " will mit " +
                        plusWishInhabitant.getName() + " in einem Zimmer sein, " +
                        "aber " + plusWishInhabitant.getName() + " nicht mit " +
                        curInhabitantToCheck.getName() + ".");
            }
        }
        //Aktuelle Schuelerin ist kein Minus Wunsch  ihrer  Plus Wunsch Schuelerin
        ...
```

Erster Teil des Zimmerkonstellations-Algorithmus Stand 25.11

Ist dies der Fall wird eine `Exception` geworfen, ansonsten wird weiter überprüft, ob das Zimmer (`Room` Objekt), indem das `curInhabitantToCheck` Objekt ist, mit dem `Zimmer` des aktuell überprüften `Inhabitant` aus der `plusWishes` Liste zusammengelegt werden kann. Dies passiert über die `checkOtherRoom()` Methode der `Room` Klasse. Ist das zurückgegebene `Inhabitant` Objekt `null`, werden die zwei Zimmer über die Methode `changeRoom()` zu einem Zimmer geändert, ansonsten wird eine `Exception` geworfen.

```
//Pruefen, ob beide Zimmer zusammengelegt werden koennten
Inhabitant errorInhabitant = curInhabitantToCheck.getRoom().checkOtherRoom(plusWishInhabitant.getRoom());

    if(errorInhabitant == null) {
        //Die Schuelerinnen im einen Zimmer in das andere Zimmer verlegen
        //und das leere Zimmer loeschen
        changeRooms(plusWishInhabitant.getRoom(), curInhabitantToCheck.getRoom());
    } else {
        throw new Exception(curInhabitantToCheck.getName() + " will mit " +
                plusWishInhabitant.getName() + " in ein Zimmer, " + "aber " +
                errorInhabitant.getName() + " kann nicht ins andere Zimmer.");
    }
```

Zweiter Teil des Zimmerkonstellations-Algorithmus Stand 25.11

Die letzte wichtige Methode ist `checkAloneInhabitants()`. Sie sortiert, nachdem `checkWishes()` und somit der erste Algorithmus ausgeführt wurde, alle `Inhabitant` Instanzen, die in jenem nicht zugeordnet wurden, weil sie weder `Inhabitant` Objekte in ihrer `plusWishes` Liste haben, noch selber in irgendeiner anderen `plusWishes` Liste waren. Für diese Objekte wird jeder `Room` in der `allRooms` Liste durchgegangen und über die Methode `checkOtherRoom()` der `Room` Klasse überprüft, ob jener `Inhabitant` noch in dieses Zimmer passt.

Sonstige Klassen:

Es gibt noch zwei weiter Klassen `FileChooser` und `ResultOutputDialog`, die größtenteils nur für die GUI zuständig sind und auf die deswegen hier nur kurz eingegangen werden soll.

Klasse: `FileChooser`

Diese Klasse öffnet im Konstruktor ein File Explorer Fenster. Öffnet man eine Textdatei, werden `Inhabitant` Objekte basierend auf dem Inhalt der Datei, mit `plusWishes` und `minusWishes` Listen erzeugt und einem `RoomAssignment` Objekt hinzugefügt. Am Schluss wird `startRoomAssignment()` in der Klasse `RoomAssignment` ausgeführt und die Ergebnisse über die `ResultOutputDialog` Instanz ausgegeben.

Klasse: `ResultOutputDialog`

Dies ist die gebundene Klasse zu der `ResultOutputDialog` Form.

Die Methode `showResultOutputDialog()` zeigt die gesammelten `output Strings` der `RoomAssignment` Klasse in einer `JList` an.

Als letztes gibt es die `Main` Klasse, deren `main()` Methode ein neues `FileChooser` Objekt erstellt und somit das Programm startet.

Klassendiagramm und Ausführung des Programms

Das Programm wurde als .jar-Datei abgegeben und kann entweder über die Konsole (Java 1.8 erforderlich) über den Befehl

```
java -jar Zimmerbelegung.jar
```

oder unter Windows durch einfaches Doppelklicken auf die .jar-Datei ausgeführt werden.

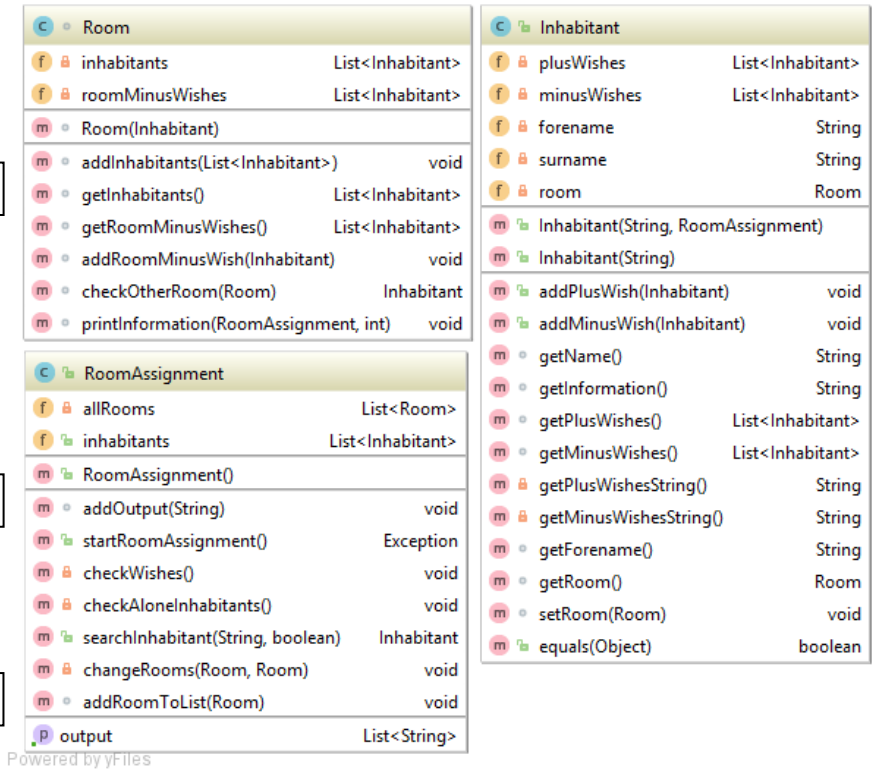
Der gesamte Quellcode ist unter

```
Zimmerbelegung_IntelliJ\src
```

und die Testfälle der BwInf-Website und der hier angegebenen Testfälle sind im Ordner

```
Zimmerbelegung_IntelliJ\Tests
```

zu finden.



Erweitertes Klassendiagramm der Klassen `Room`, `Inhabitant` und `RoomAssignment` Stand 25.11

Optimierungsmöglichkeiten

1. **Möglichst wenig Zimmer**
   a. Schülerinnen ohne positive Präferenzen werden vor Beendigung der Zimmerzuteilung wenn möglich anderen Zimmern zugeteilt.
   (In meiner Version implementiert)
   b. Die Bewohner eines Zimmers, die keine Minus Wünsche gegen die Bewohner eines anderen Zimmers haben, können mit in das andere Zimmer verlegt werden.
2. **Nachnamen**
   Das Programm unterscheidet Schüler anhand ihrer Namen. Da es in der Aufgabenstellung und den Test Beispielen nur Vornamen gibt, kann es zu Fehlern bei der Zimmerzuteilung kommen, wenn mehrere Schüler den gleichen Vornamen haben.
   (In meiner Version fast fertig implementiert)
3. **Zimmer Vorgaben**
   Das Programm kann theoretisch unendlich viele Zimmer mit unendlich viel Platz für Schüler erstellen. In der Realität, gibt es aber bei jeder Jugendherberge etc. eine Obergrenze an verfügbaren Zimmern und Platz pro Zimmer. Vor der Zimmerzuteilung sollte man angeben können, wie viele Zimmer verfügbar sind und wieviel Platz jedes Zimmer hat. Dann wird überprüft, ob eine Zimmerverteilung auf die verfügbaren Zimmer möglich ist.
4. **Fehler Ausgabe**
   Das Programm gibt bei nicht erfolgreicher Zimmerzuordnung aus, woran es gescheitert ist.
   (In meiner Version implementiert)

Tests

Tests Level 1:

1. Schülerin A will mit Schülerin B in ein Zimmer aber Schülerin B nicht mit Schülerin A.

Ergebnis: Fehler

Exception Message: Anna will mit Steffi in einem Zimmer sein, aber Steffi nicht mit Anna.

Gelöst durch: Ersten Teil des Algorithmus `checkWishes()`

```
Anna
+ Steffi
-

Steffi
+
- Anna
```

2. Schülerin A und Schülerin B wollen mit Schülerin C in ein Zimmer, aber nicht zusammen.

Ergebnis: Fehler

Exception Message: Lisa will mit Steffi in ein Zimmer, aber Lisa kann nicht ins andere Zimmer.

Gelöst durch: Zweiten Teil des Algorithmus `checkWishes()`

```
Anna
+ Steffi
- Lisa

Lisa
+ Steffi
-

Steffi
+
-
```

Tests Level 2:

3. Schülerin A will mit Schülerin C aber nicht mit Schülerin B in ein Zimmer. Schülerin C will aber mit Schülerin B in ein Zimmer.

Ergebnis: Fehler

Exception Message: Steffi will mit Lisa in ein Zimmer, aber Lisa kann nicht ins andere Zimmer.

Gelöst durch: Zweiten Teil des Algorithmus `checkWishes()`

```
Anna
+ Steffi
- Lisa

Lisa
+
-

Steffi
+ Lisa
-
```

4. Schülerin A und Schülerin C haben keine Präferenzen. Schülerin B will mit Schülerin C in ein Zimmer.

Ergebnis: Mögliche Zimmerkonstellation

Optimal Lösung: Alle kommen in ein Zimmer.

Gelöst durch: Extra Algorithmus `checkAloneInhabitants()`

```
Anna
+
-

Lisa
+ Steffi
-

Steffi
+
-
```

Sonderfälle:

5. Schülerin A will mit sich selbst nicht in ein Zimmer.

```
Anna
+
- Anna
```

Ergebnis: Fehler

Exception Message: Anna will nicht mit sich selbst ins Zimmer.

Gelöst durch: Prüfung beim Hinzufügen eines `Inhabitant` Objekts zur `minusWishes` Liste

6. Schülerin A will nicht mit Schülerin D in ein Zimmer. Schülerin B, C und D haben keine Präferenzen.

```
Anna
+
- Tina

Steffi
+
-

Lisa
+
-

Tina
+
-
```

Ergebnis: Mögliche Zimmerkonstellation

Mögliche Lösung: Zwei zweier Zimmer
(Bsp.: Anna mit Steffi und Lisa mit Tina)

Gelöst durch: Extra Algorithmus `checkAloneInhabitants()`