Aufgabe 1: Zimmerbelegung

Team: Hier könnte ihre Werbung stehen.
Team-ID: 00093

24. November 2017

Inhaltsverzeichnis

1 Lösungsidee

Wir bekommen zunächst eine Liste von Personen mit den Verhältnissen der Personen zueinander, also mit welchen Personen sie ein Zimmer teilen möchten oder nicht teilen möchten. Als erstes werden die Personen in Einzelzimmer verteilt, es sei denn es gibt schon eine Person in einem Zimmer, die die aktuelle Person ebenfalls im Zimmer haben möchte, oder ein Zimmer, welches eine Person enthält, mit der die aktuelle Person in einem Zimmer sein möchte. Im nächsten Schritt wird für alle Zimmer überprüft, ob es denn Wünsche aus einem Zimmer gibt, mit einer Person eines anderen Zimmer das Zimmer zu teilen. Ist dies der Fall so werden die Zimmer zusammengelegt. Dabei haben wir uns gegen eine Zusammenlegung der neutral zueinander gesinnten Zimmer (also Zimmer bei denen keine Person des einen Zimmers eine Beziehung zu einer Person des anderen Zimmers hat) entschieden, da die Anzahl der Personen in den jeweiligen Zimmern teilweise sehr gross und damit unrealistisch werden würde. Zum Schluss wird für jedes Zimmer überprüft, ob jede Person mit den gewünschten Personen zusammen ist und ob keine nicht gewollte Person im selben Zimmer ist.

2 Umsetzung

2.1 Klassen

Es werden die Klassen *Person*, *Room* und *Distribution* implementiert. Die Klasse *Person* besitzt die Attribute *name*, *likes* und *dislikes* (siehe Abbildung 1). *name* stellt den Namen der Person, *likes* eine Liste mit den Personen mit denen die Person ein Zimmer teilen möchte und *dislikes* eine Liste mit den Personen mit denen die Person nicht im Zimmer sein möchte. Die Klasse *Room* besitzt die Attribute *persons*, *personNames*, *likes* und *dislikes* (siehe Abbildung 2). Die Liste *persons* enthält alle aktuell in diesem Zimmer untergebrachte Personen, die drei anderen Listen enthalten jeweils alle Items der entprechenden Listen von *Person*. *personNames* enthält also alle Namen der Personen, *likes* alle Namen der Personen mit denen ein Zimmer geteilt werden möchte und *dislikes* alle Namen der Personen, die nicht in diesem Zimmer sein sollen. *Distribution* dient hier als Main-Klasse, führt also die wesentlichen Bestandteile des Programms aus und wird im Folgenden behandelt.

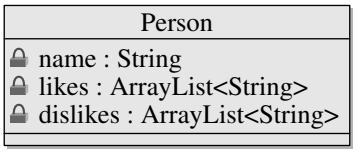

Abbildung 1: UML-Diagramm der Klasse Person

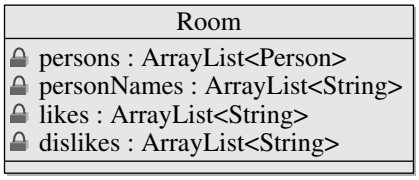

Abbildung 2: UML-Diagramm der Klasse Room

2.2 Main-Methode

Die Main-Methode verwirktlicht die Umsetzung der Lösungsidee indem sie die einzelnen Methoden des Programms ausführt und miteinander verknüpft. Zuerst wird überprüft, ob die Aufgabendatei als Parameter gegeben wurde und bei zu wenigen / zu vielen Parametern eine entsprechende Fehldermeldung ausgegeben. Anschliessend werden die Personen eingelesen, in Zimmer verteilt und das Ergebnis in der Konsole ausgegeben und in einer Datei gespeichert (siehe Listing 5).

2.3 Einlesen der Aufgabendatei

Zuerst werden die Personen und deren Attribute eingelesen. Diese werden mithilfe der Klasse *Person* gespeichert (siehe Listing 6). Auffällig hierbei ist, dass beim Einlesen einer Datei die ersten drei Zeichen manchmal kryptisch sind und durch eine Überprüfung der ersten Buchstaben auf ihr Vorkommen im Alphabet entfernt werden müssen.

2.4 Erste Verteilung

Bei der ersten Verteilung der Personen werden, wie in 1 Lösungsidee erwähnt, die Personen in Einzelzimmer verteilt, es sei denn es gibt schon eine Person in einem Zimmer, die die aktuelle Person ebenfalls im Zimmer haben möchte, oder ein Zimmer, welches eine Person enthält, die die mit der die aktuelle Person in einem Zimmer haben möchte. Dazu wird zunächst die Liste *rooms* initialisiert. Anschliessend wird für jede Person eine ForEach-Schleife ausgeführt, bei der mittels *java.util.Collections.disjoint* für jede Person überprüft wird, ob eine der oben genannten Bedingungen zutrifft. Wenn dem so ist, wird die Person dem entsprechenden Zimmer hinzugefügt. Trifft keine dieser Bedingungen auf ein Zimmer zu, wird ein neues Zimmer erstellt und die Person zu diesem hinzugefügt (siehe Listing 7, Zeile 4-18).

2.5 Zweite Verteilung

Bei der zweiten Verteilung werden Zimmer zusammengeführt, in denen eine Person des einen Zimmers mit einer Person des anderen Zimmers zusammen in einem Zimmer sein möchte. Dafür wird mittels zweier ForEach-Schleifen überprüft, ob diese Bedingung für zwei Zimmer zutrifft. Wichtige Kriterien für eine Zusammenlegung von zwei Zimmern sind: In jedem der beiden Zimmer befindet sich mindestens eine Person; Es handelt sich um unterschiedliche Zimmer; Die Zimmer können überhaupt zusammengelegt werden; Eine Zusammenlegung macht überhaupt Sinn. Letzteres wird wieder mittels *java.util.Collections.disjoint* überprüft. Des weiteren ist es wichtig, dass bei dem nach der Zusammenlegung leeren Zimmer alle Attribute bereinigt werden, um eine doppelte Existenz der Personen zu vermeiden. Schliesslich werden die leeren Zimmer nach Abschluss der zweiten Verteilung komplett aus der Liste entfernt (siehe Listing 7, Zeile 21-42).

2.6 Überprüfung der Zimmer

Um herauszufinden, ob eine für alle Personen zufriedenstellende Verteilung möglich ist, ist eine Überprüfung der Zimmer auf Unvollständigkeiten oder Widersprüche nötig. Diese wird durch die Methode *isPossible* implementiert. Diese überprüft, ob alle Personen mit den Personen, mit denen sie ein Zimmer teilen wollen, zusammen sind und ob sie mit keiner Person, mit der sie nicht in einem Zimmer sein möchten, zusammen sind (siehe Listing 8).

2.7 Ausgabe und Speichern des Ergebnisses

Die letztendliche Zimmerverteilung wird in der Konsole ausgegeben und in eine *-loes.txt* geschrieben (zimmerverteilungX.txt -> zimmerverteilungX-loes.txt). Dabei wird zwischen einer erfolgreichen und einer nicht erfolgreichen Verteilung unterschieden. Bei einer erfolgreichen Verteilung wird in jede Zeile ein Zimmer bzw. die darin untergebrachten Personen geschrieben. Bei einer nicht erfolgreichen Verteilung wird ein einfaches *Verteilung nicht möglich!* in die Datei geschrieben (siehe Listing 9).

3 Beispiele

Um zu überprüfen, ob das Programm fehlerfrei funktioniert, werden diesem zwei Aufgaben gestellt, die manuell leicht zu überprüfen sind:

3.1 Beispiel 1

Personenkonstellation mit einer erfolgreichen Verteilung (beigelegt als *test1.txt*):

```
1 Neele
  + Lara Lina
3 -
```

```
Lara
+
-

Charlotte
+
- Lina

Lina
+ Neele
-
```

Listing 1: Aufgabe mit erfolgreicher Verteilung

Das Programm gibt folgende Verteilung als *test1-loes.txt* aus:

```
Neele Lara Lina
Charlotte
```

Listing 2: Erfolgreiche Verteilung

3.2 Beispiel 2

Personenkonstellation mit einer nicht erfolgreichen Verteilung (beigelegt als *test2.txt*):

```
Julia
+ Lea Celine
-

Lea
+
- Clara

Yasmin
+ Julia Clara
-

Celine
+ Yasmin
-
Clara
+
- Julia
```

Listing 3: Aufgabe mit nicht erfolgreicher Verteilung

Das Programm gibt folgende Ausgabe als *test2-loes.txt*:

```
Verteilung nicht moeglich!
```

Listing 4: Nicht erfolgreiche Verteilung

4 Quellcode

```
    public static void main(String[] args) {
        // check the parameter
        if (args.length < 1) {
            System.out.println("Zu wenige Parameter!");
            System.out.println("Benutzung: java -jar Distribution.jar <dateiname>");
            System.exit(1);
        } else if (args.length > 2) {
            System.out.println("Zu viele Parameter!");
            System.out.println("Benutzung: java -jar Distribution.jar <dateiname>");
            System.exit(1);
        }
        // get all persons
        ArrayList<Person> persons = addPersons(args[0]);
        // distribute the persons into rooms
        ArrayList<Room> rooms = distributePersons(persons);
        // calculate if this distribution is possible
        boolean possible = isPossible(rooms);
```

```
        if (possible) {
            System.out.println("Verteilung ist moeglich!");
            int numPersons = persons.size() + 1;
            int numRooms = rooms.size() + 1;
            System.out.println("Anzahl aller Personen: " + numPersons + "\n");
            System.out.println("Anzahl aller Zimmer: " + numRooms + "\n");
            for (Room z : rooms) {
                System.out.println(z.getInformation() + "\n");
            }
        } else {
            System.out.println("Verteilung ist nicht moeglich!");
        }
        // save the result
        writeIntoFile(args[0].substring(0, args[0].length() - 4) + "-loes.txt", rooms, possible);
    }
```

Listing 5: Main-Methode

```
    // reads the file, processes, saves and returns the persons
    public static ArrayList<Person> addPersons(String filepath) {
        // read the data from the file
        ArrayList<String> data = readDataFromFile(filepath);
        ArrayList<Person> persons = new ArrayList<>();
        int numPersons = data.size();

        // loop through the file
        // 1. line = Name
        // 2. line = liked persons
        // 3. line = disliked persons
        // 4. line = blank
        for (int i = 0; i < numPersons; i += 4) {
            String name = data.get(i);
            // get the liked persons and remove the prefix (+ ; - )
            String raw_likes = data.get(i + 1).substring(1);
            String raw_dislikes = data.get(i + 2).substring(1);
            // remove the space if there's one
            if (raw_likes.length() > 0 && Character.toString(raw_likes.charAt(0)) == " ") {
                raw_likes = raw_likes.substring(1);
            }
            if (raw_dislikes.length() > 0 && Character.toString(raw_dislikes.charAt(0)) == " ") {
                raw_dislikes = raw_dislikes.substring(1);
            }
            // convert the liked persons to an arraylist
            ArrayList<String> likes;
            ArrayList<String> dislikes;
            if (raw_likes.length() > 0) {
                likes = new ArrayList<String>(Arrays.asList(raw_likes.split(" ")));
            } else {
                likes = new ArrayList<String>();
            }
            if (raw_dislikes.length() > 0) {
                dislikes = new ArrayList<String>(Arrays.asList(raw_dislikes.split(" ")));
            } else {
                dislikes = new ArrayList<String>();
            }
            // add the person
            persons.add(new Person(likes, dislikes, name));
        }
```

Listing 6: Einlesen und Speichern der Aufgaben-Datei

```
    // distributes the persons into rooms
    public static ArrayList<Room> distributePersons(ArrayList<Person> personen) {
        ArrayList<Room> rooms = new ArrayList<Room>();
        A:
        for (Person person : personen) {
            // check if a rooms contains a liked person or the person is liked in a room
            // if so add the current person to that room
            for (Room z : rooms) {
                if (z.getLikes().contains(person.getName()) || !Collections.disjoint(z.getPersonNames()
                    z.addPerson(person);
                    continue A;
                }
```

```
            }
            // if no liked person is distibuted yet add this person into a single room
            Room z = new Room();
            z.addPerson(person);
            rooms.add(z);
        }
        // initialize a list for empty rooms
        ArrayList<Room> toRemove = new ArrayList<Room>();
        // Distribution of the not satisfied room (not all liked persons are in the room)
        B:
        for (Room zi : rooms) {
            for (Room zy : rooms) {
                if (zi.getPersons().size() > 0 && zy.getPersons().size() > 0 && zi != zy && !Collection
                    for (Person p : zy.getPersons()) {
                        zi.addPerson(p);
                    }
                    // clear the old room zy
                    zy.getPersons().clear();
                    zy.getPersonNames().clear();
                    zy.getLikes().clear();
                    zy.getDislikes().clear();
                    toRemove.add(zy);
                    continue B;
                }
            }
        }
        for (Room z : toRemove) {
            rooms.remove(z);
        }
        toRemove.clear();
        return rooms;
    }
```

Listing 7: Verteilen der Personen

```
    // checks whether all wishes are fulfilled
    public static boolean isPossible(ArrayList<Room> rooms) {
        if (rooms == null) {
            System.out.println("rooms is null");
            return false;
        }
        for (Room z : rooms) {
            for (Person p : z.getPersons()) {
                // check that all liked persons of a person are in the same room
                for (String s : p.getLikes()) {
                    if (!z.getPersonNames().contains(s)) {
                        return false;
                    }
                }
                // check that all disliked persons of a person are not in the room
                for (String s : p.getDislikes()) {
                    if (z.getPersonNames().contains(s)) {
                        return false;
                    }
                }
            }
        }
        return true;
    }
```

Listing 8: Überprüfung der Zimmer auf Unvollständigkeiten und Widersprüche

```
    // writes the result into a file
    public static void writeIntoFile(String path, ArrayList<Room> rooms, boolean distributionIsPossible
        try {
            // open a file to save the result into
            FileWriter fw = new FileWriter(path);
            BufferedWriter bw = new BufferedWriter(fw);
            if (distributionIsPossible) {
                for (Room z : rooms) {
                    // initializing a string which will contain all persons names in the current room
                    String s = "";
                    for (String p : z.getPersonNames()) {
```

```
12                          // add the current person to the string
                            s += p + "␣";
14                      }
                        // add a new line and write into the file
16                      bw.write(s + "\n");
                    }
18              } else {
                    bw.write("Verteilung␣nicht␣moeglich!");
20              }
                bw.close();
22          } catch (Exception e) {
                e.printStackTrace();
24          }

26      }
```

Listing 9: Speichern des Ergebnisses in eine Datei