

Aufgabe 1: Superstar

Team-ID: 00126

Team-Name: GSO

Bearbeiter dieser Aufgabe: Lennart Sandbothe

Programmiersprache: Java

Inhaltsverzeichnis:

Lösungsidee	2
Umsetzung	2
Beispiele	3
1. 'superstar2.txt'	3
2. 'superstar3.txt'	3
Quelltext	4
Superstar.java: searchSuperstar	4
Superstar.java: main	5
Group.java:	5

Lösungsidee

Ein Superstar ist jemand, wie in der Aufgabenstellung gegeben, dem jedes andere Gruppenmitglied folgt, welcher selber jedoch niemandem folgt.

Man nehme an jedes Mitglied der Gruppe wäre ein möglicher Superstar und hat so eine List mit möglichen Namen.

Nun geht man die Mitglieder der Reihe nach durch und überprüft, ob der aktuelle Teilnehmer den Teilnehmern aus der Liste möglicher Superstars(außer ihm selbst) folgt.

Ist dies der Fall, kann der aktuelle Teilnehmer kein Superstar mehr sein, da er selber jemandem folgt und wird, insofern er selbst noch in der Superstar-Liste steht, aus dieser gestrichen.

Ist dies nicht der Fall, wird der andere Teilnehmer aus der Superstar-Liste gestrichen, da ihm mindestens eine Person nicht folgt.

Nach diesem Durchlauf wurden alle Mitglieder als möglicher Superstar gestrichen, denen nicht von allen anderen gefolgt wird. Wer jedoch selbst noch jemandem folgt, der vorher aus der Superstar-Liste gestrichen wurde, wurde noch nicht gestrichen.

Daher wird nun von allen noch in der Superstar-Liste befindlichen Mitglieder getestet, ob sie einer Person, bei der es noch nicht bereits geprüft wurde folgen. Ist dies der Fall können auch sie aus der Liste gestrichen werden.

Am Ende besteht die Liste entweder aus einem Namen, dem Superstar der Gruppe, oder ist komplett leer, dann gibt es keinen Superstar.

Die Anfragen, die bei diesem Ansatz benötigt werden, sind von der Reihenfolge der Mitglieder abhängig.

Umsetzung

Um die beschriebene Lösungsidee zu implementieren wurde eine Klasse 'Group' erstellt, welche eine Datei im Format der BwInf-Beispielaufgaben einliest und die Mitglieder speichert und wem die folgen speichert.

Weiterhin beinhaltet diese Klasse eine Methode zur Überprüfung, ob ein Teilnehmer einem anderen folgt.

Alle abfragen werden gezählt, sodass die Anzahl am Ende eingesehen werden kann.

Der Dateipfad zur Mitgliederliste wird in den Programm-Argumenten angegeben.

Nach dem Einlesen werden die Teilnehmer in eine Liste 'superstars' kopiert.

Nun wird das oben beschriebene Verfahren angewandt, wobei jeder gemachte vergleich in einer weiteren Liste 'checkedOn' festgehalten wird.

So kann nach dem ersten Durchlaufen der Teilnehmerliste eingesehen werden, welche Paarungen noch überprüft werden müssen.

Beispiele

1. 'superstar2.txt'

Startet man nun das Programm mit dem Pfad zur 'superstar2.txt'-Beispieldatei wird folgendes ausgegeben:

```
Request: Knuth -> Hoare: false
Request: Knuth -> Turing: true
Request: Hoare -> Turing: true
Request: Turing -> Codd: false
Request: Turing -> Dijkstra: true
Request: Codd -> Dijkstra: true
Request: Dijkstra -> Knuth: false
Request: Dijkstra -> Hoare: false
Request: Dijkstra -> Turing: false
Request: Dijkstra -> Codd: false

Results:
[Dijkstra] 10 requests with 5 members
```

alle durchgeführten Vergleiche werden ausgegeben

Format: Request: [x] -> [y]: bool
folgt Mitglied x Mitglied y? ja/nein

die Ergebnisse werden in Form der Superstar-Liste und den benötigten Anfragen bei gegebener Anzahl von Mitgliedern ausgegeben.

2. 'superstar3.txt'

Bei der Beispieldatei 'superstar3.txt' werden aufgrund der höheren Anzahl an Mitgliedern auch mehr Anfragen benötigt:

Hier kann man auch die 2 Phasen des Verfahrens erkennen:

1. Phase: - alle die mjd. folgen, werden gestrichen

2. Phase: - nur noch Edsger ist übrig
- Überprüfung ob auch er jmd. folgt

```
Request: Rineke -> Sjoukje: false
Request: Rineke -> Rinus: false
Request: Rineke -> Jitse: false
Request: Rineke -> Edsger: true
Request: Sjoukje -> Edsger: true
Request: Rinus -> Edsger: true
Request: Jitse -> Edsger: true
Request: Edsger -> Jorrit: false
Request: Edsger -> Pia: false
Request: Edsger -> Peter: false
Request: Jorrit -> Edsger: true
Request: Pia -> Edsger: true
Request: Peter -> Edsger: true
Request: Edsger -> Rineke: false
Request: Edsger -> Sjoukje: false
Request: Edsger -> Rinus: false
Request: Edsger -> Jitse: true

Results:
[] 17 requests with 8 members
```

Quelltext

Superstar.java: searchSuperstar

```
private static String[] searchSuperstar(Group group) {
    List<String> members = new LinkedList<>(group.getMembers());
    List<String> superstars = new LinkedList<>(members);
    Map<String, LinkedList<String>> checkedOn = new HashMap<>();

    for (String member : members) {
        checkedOn.putIfAbsent(member, new LinkedList<>());
        for (String superstar : new LinkedList<>(superstars)) {
            if (!member.equals(superstar)) {
                checkedOn.get(member).add(superstar);
                if (group.isXFollowingY(member, superstar)) {
                    superstars.remove(member);
                    break;
                } else {
                    superstars.remove(superstar);
                }
            }
        }
    }

    for (String superstar : new LinkedList<>(superstars)) {
        for (String member : members) {
            if (!superstar.equals(member) &&
                !checkedOn.get(superstar).contains(member)) {
                if (group.isXFollowingY(superstar, member)) {
                    superstars.remove(superstar);
                    break;
                }
            }
        }
    }

    return superstars.toArray(new String[]{});
}
```

Superstar.java: main

```
public static void main(String[] args) {
    Group group = new Group(args[0]);

    String[] superstars = searchSuperstar(group);

    System.out.println("\nResults: ");
    System.out.println(Arrays.toString(superstars) + "    " +
        group.getNumRequests() + " requests with " +
        group.getMembers().size() + " members");
}
```

Group.java:

```
public class Group {

    private HashMap<String, List<String>> members;
    private int numRequests = 0;

    public Group(String dataFilePath) {
        // Datei einlesen und Daten in members speichern
        ...
    }

    public boolean isXFollowingY(String x, String y) {
        boolean follows = this.members.get(x).contains(y);
        System.out.println("Request: "+ x + " -> " + y + ": " +
            follows);
        this.numRequests++;
        return follows;
    }

    ....
}
```