Praktische Informatik II | SoSe 2014 | Luca Raimondo ***DM***
TutorIn: Alexis Tsogias | Gruppe: 4 | Holger Illhardt ***INF***
Christiane Wolff ***DM***

Übungsblatt 2

93/100 Punkte

Aufgabenlösung
Abgabe: 06.06.2014

Inhaltsverzeichnis

Warum ist denn das Inhaltsverzeichnis so kaputt?

Aufgabe 1 Trockenübung

22/22

Aufgabe 1.1 StudentIn 4

4/4

Tut 1	st1	st2	**st3**	
Tut 2				
Tut 3				

Ausgangskonfiguration

$\Rightarrow$

Tut 1	st1	st2	st3	st4
Tut 2				
Tut 3				

$r_s < r$ und $j < t \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2	st3	
Tut 2	**st4**			
Tut 3				

Fall 2

Aufgabe 1.2 StudentIn 7? 6/6

Tut 1	st1	st2	st3	
Tut 2	st4	st5	**st6**	
Tut 3				

Ausgangskonfiguration

$\Rightarrow$

Tut 1	st1	st2	st3	**st7**
Tut 2	st4	st5	st6	
Tut 3				

$|T_1| < r$ und $j < t \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2	st3	
Tut 2	st4	st5	st6	**st7**
Tut 3				

$|T_2| < r$ und $j < t \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2	st3	
Tut 2	st4	st5	st6	
Tut 3	**st7**			

$r_s < r$ und $j = t \rightarrow$ Fall 4

$\Rightarrow$

Tut 1	st1	st2	st3	
Tut 2	st4	st5		
Tut 3	**st6**			

Fall 2

Aufgabe 1.3 StudentIn 7! 12/12

Tut 1	st1	st2	st4	
Tut 2	st5			
Tut 3	st3	**st6**		

Ausgangskonfiguration

$\Rightarrow$

Tut 1	st1	st2	st4	**st7**
Tut 2	st5			
Tut 3	st3	st6		

$|T_1| > st_{max} \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2	st4	
Tut 2	st5	**st7**		
Tut 3	st3	st6		

$r_s < r$ und $j < t \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2	st4	
Tut 2	st5			
Tut 3	st3	st6	**st7**	

$r_s < r$ und $j = t$ Fall 4

$\Rightarrow$

Tut 1	st1	st2	st4	
Tut 2				
Tut 3	st3	**st5**		

$r_s < r$ und $j = t \rightarrow$ Fall 4

$\Rightarrow$

Tut 1	st1	st2		
Tut 2	**st4**			
Tut 3	st3			

Fall 2

$\Rightarrow$

Tut 1	st1	st2	**st5**	
Tut 2	st4			
Tut 3	st3			

$r_s < r$ und $j < t \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2		
Tut 2	st4	**st5**		
Tut 3	st3			

Fall 2

$\Rightarrow$

Tut 1	st1	st2	**st6**	
Tut 2	st4	st5		
Tut 3	st3			

$r_s < r$ und $j < t \rightarrow$ Fall 3

$\Rightarrow$

Tut 1	st1	st2		
Tut 2	st4	st5	**st6**	
Tut 3	st3			

Fall 2

$\Rightarrow$

Tut 1	st1	st2	**st7**	
Tut 2	st4	st5	st6	
Tut 3	st3			

Fall 2

Aufgabe 2 TheaPlusPlus 59/60

Aufgabe 2.1 Studis

G: 8/10
C: 4/4 D: 4/4 T: 0/2

Um `Object.equals()` für unsere Zwecke zu überschreiben, muss `equals()` ein Object (also keinen Studi) entgegennehmen. Um die beiden Studi-Attribute *account* mittels `equals()` zu vergleichen, müssen wir das übergebene1 Object dann wieder nach Studi *casten*.

Tests fehlen T = 0

Aufgabe 2.2 TheaPPParser

G: 12/15
C: 6/6 D: 6/6 T: 0/3

Nach *javaDoc*-Vorgaben implementiert.

Tests fehlen T = 0

Aufgabe 2.3 TheaPP

G: 24/30
C: 12/12 D: 12/12 T: 0/6

Methode: `TheaPP.createInitialConfig`

Zuerst wird die äußere Liste von Tutorien erzeugt, die dann durchlaufen und mit Listen von Studis gefüllt wird.

Eine leere Konfiguration könnte auf zwei Arten umgesetzt werden:

1. Als Tutorien mit Listenelementen, die null-Daten haben, füllen und bei Veränderung der Konfiguration werden die Daten mit List.set() überschrieben.
2. In einer leeren Konfiguration sind die einzelnen Tutoriums-Listen leer und man fügt Studis mit List.add() hinzu.

Wir haben uns für die zweite Variante entschieden, weil wir uns daran orientiert haben, dass TheaPP.resetConfiguration ein clear() zum Löschen der Liste nutzt. (clear() löscht alle Elemente der Liste.)

Methode: `TheaPP.nextConfiguration`

Der Algorithmus muss Fall 1 und Fall 4 als Abbruchkriterien erkennen. Zuerst wird im äußeren *if*-Zweig nach gültiger oder ungültiger Konfiguration unterschieden, im gültig-Zweig dann zwischen Fall 1 und 2, im ungültig-Zweig zwischen Fall 3 und 4. Bei Fall 2 wird vorausgesetzt, dass das Szenario $n > s$ nie eintreten kann.

In Fall 4 wird $j > t$ vernachlässigt, weil wir davon ausgehen, dass j nie größer als t wird, weil alle Studis bereits durch Fall 3 vorher in jedem Schritt neu zugeordnet/ausgetragen wurden.

Falls Fall 4 eintritt, ist sichergestellt, dass mindestens ein Studi zugeordnet ist und Studi S_n im letzten Tutorium T_t ist, daher kann man ohne weitere Bedingungen zu prüfen, damit beginnen, den Studi aus dem Tutorium zu entfernen (hier als *do-while*-Schleife umgesetzt).

Aufgabe 2.4 Main 5/5

- ***FiveTutorials.csv* mit Mindestbewertung $r = 5$:**
  Keine Lösung.
- ***FiveTutorials.csv* mit Mindestbewertung $r = 3$:**
  Keine Lösung.

- ***FiveTutorials.csv* mit Mindestbewertung $r = 2$:**

taykin[2]	lydsco[2]	kayfos[2]	coomil[3]	natcox[2]	taytho[2]	marjam[4]	wyapri[2]	serjen[5]	wyaher[5]
levlon[3]	joswri[5]	chajam[5]	brajoh[2]	gratho[2]	wilphi[2]	katbak[2]	kylben[3]	adamar[2]	concar[4]
trirog[2]	peybar[3]	carper[2]	tylcol[3]	haykin[3]	paylon[4]	parher[3]	isabut[3]	carbel[3]	liawal[2]
hencol[2]	zoetho[2]	jayflo[5]	bratur[5]	calhay[4]	zacmor[5]	faibut[3]	audsan[5]	morpet[5]	evasim[3]
juajoh[5]	jasree[2]	nevkin[2]	laulon[5]	sartay[3]	levlew[2]	avenel[4]	oweben[2]	noacol[5]	aritur[2]

- ***FiveTutorials.csv* mit Mindestbewertung $r = 1$:**

levlon[1]	taykin[2]	joswri[1]	lydsco[2]	kayfos[2]	chajam[1]	coomil[3]	natcox[2]	taytho[2]	gratho[1]
juajoh[1]	brajoh[2]	jasree[1]	hencol[1]	wilphi[2]	marjam[5]	wyapri[5]	katbak[2]	kylben[3]	serjen[1]
trirog[2]	wyaher[2]	peybar[3]	zoetho[1]	adamar[2]	concar[2]	carper[2]	tylcol[3]	haykin[3]	liawal[2]
jayflo[5]	paylon[5]	parher[3]	bratur[5]	isabut[3]	calhay[4]	faibut[3]	audsan[5]	morpet[5]	evasim[3]
carbel[4]	zacmor[2]	nevkin[2]	laulon[5]	sartay[3]	levlew[2]	avenel[4]	oweben[2]	noacol[5]	aritur[2]

Aufgabe 2.5 BONUS: Was ist denn bloß los? 2/5

Das Problem der *TenTutorials.csv* bei einer Mindestbewertung von $r = 4$ ist, dass einige StudentInnen dann in 0 Tutorien ein persönliches Rating $\geq r$ haben und der Algorithmus somit schon vor dem Start zum scheitern verurteilt ist.

Im konkreten Datensatz sind das die Einträge *(56)matpet*, *(67)luisco* und *(100)lydon*.

Durch eine Reduzierung von r auf 2 oder weniger, wäre aber auch dieser Datensatz wieder bearbeitbar.

Der Algorthmus kann keine gültige Lösung finden. Aber um das festzustelle braucht er ewigkeiten. Durch Umsortieren kann man den Aufwand reduzieren.

Aufgabe 2.6 BONUS: Pro Studi 8/10

Mit den gegebenen Daten aus Aufgabe 1 lassen sich einige der wesentlichsten Probleme des vorgegebenen Algorithmusses gut verdeutlichen, die anderen werden ohne explizites Beispiel als Gedankenspiel behandelt.

Optimierungsmöglichkeit 1 – Berücksichtigung zusätzlicher der Abbruchparameter

Es macht Sinn einige Fälle direkt vor "anlaufenlassen" des Algorithmus zu überprüfen und gegebenfalls den Start zu verhindern, sollte das eh offensichtlich zu keiner Lösung führen können.

- **Nicht genügend Plätze:**
  Sollte $s \leq st_{max} \cdot t$ sein, so ist keine Konfiguration möglich, bei der alle StudentInnen berücksichtigt werden können.

- **Zu viele Studenten wünschen sich lediglich ein Tutorium:**
  Bei einer zu großen Anzahl $x < st_{max}$ von unflexiblen StudentInnen, die alle in das selbe Tutorium T_i wollen, gibt es auch offensichtlich keinen Fall 1.

Optimierungsmöglichkeit 2 – Berücksichtigung der individuellen Flexibilität

Offensichtlich lohnt es sich extremst die individuelle *Flexibilität* der jeweiligen Studenten zu berücksichtigen.

Hierbei wird für jeden Schüler $S_n, n \in [1, s]$ die Anzahl seiner Tutorienbewertungen, welche über der Minimalbewertung r liegen gezählt und zu einer Summe f zusammengerechnet. f liegt hierbei offensichtlich im Bereich $[0, t]$, denn ein Schüler kann minimal 0 Tutorien ein Rating $< r$ und maximal allen Tutorien ein Rating $\geq r$ gegeben haben.

	Tut1	Tut2	Tut3	Flexibilität
st1:	4	0	5	2
st2:	3	4	5	3
st3:	3	0	4	2
st4:	4	3	3	3
st5:	0	5	0	1
st6:	0	4	5	2
st7:	5	0	0	1
st8:	0	5	5	2
st9:	0	4	5	2

Tabelle 1: Flexibilitäten

Wenn wir an dieser Stelle noch nicht auf humanitäre Aspekte wie Rücksicht auf empathische oder Sanktionierung von egoistischen StudentInnen eingehen, dann lohnt es sich die StudentInnen in 4 Gruppen zu unterteilen:

- **Gruppe 1 – Die Joker:** $f_S = t$
  Diese Personen haben allen Tutorien eine Bewertung $r_T \geq r$ gegeben und sind somit als Joker zu sehen. Es erspart Rechenschritte, diese StudentInnen aus der Menge der zu verteilenden StudentInnen herauszuhalten und lediglich am Ende mit ihnen "aufzufüllen".

- **Gruppe 2 – Die Menge:** $f_S \in [2, t-1]$
  Der Durchschnitt der Probanden dürfte in dieser Menge zu finden sein, in der zumindest 2 Tutorien ein Rating $\geq r$ haben, aber in keinem Fall alle. Das sind die Objekte, die der Algorithmus durch Einsortieren platzieren muss, da nicht von vornherein sichtbar ist, wo sie landen werden.

- **Gruppe 3 – Die Egoisten:** $f_S = 1$
  Diese Menge an StudentInnen hat aus für den Algorithmus irrelevanten Gründen lediglich 1 Tutorium $\geq r$ gerated und kann damit als "statisch" betrachtet werden. Hier offenbart sich nun ein zusätzlicher Abbruchparameter für die *Optimierungsmöglichkeit 1*, denn wenn die Anzahl aller StudentInnen dieser Gruppe, die das Tutorium T_i ausgewählt haben $> st_{max}$ ist, so ist wieder keine Konfiguration möglich, die nach den Parametern zufriedenstellend wäre.

- **Gruppe 4 – Die Nichtkönner:** $f_S = 0$
  Wie man mit der Gruppe derjenigen umgeht, die laut gesetztem r zu keinem der Tutorien kompatibel sind umgeht, ist eine Verfahrensfrage. Man könnte durchaus dafür plädieren, diese zu *Gruppe 1* hinzuzufügen. Alternativ könnte man diese Personen auch einfach komplett aus der Berechnung herauslassen und sie sich selbst überlassen. Letztere Verfahrensweise ist jedoch bei linearer Abarbeitung der Tutorien keine Empfehlenswerte Variante, da es unter Umständen zu einer Verdichtung von Tutanden in den Tutorien mit niedrigem Index führen könnte, da hierdurch die Anzahl der zu verteilenden StudentInnen reduziert wird.

Das würde folgende Mengen für diese Beispielkonstellation ergeben:

Gruppe	G_1	G_2	G_3	G_4
Anzahl	2	5	2	0

Das soll eine 2 sein, oder? :D

Diese Gruppenzuweisungen verkürzen nun den durchschnittlich zu erwartenden Arbeitsaufwand enorm, denn indem man alle S aus Gruppe 3 zuerst zuweisen lässt, umgeht man die vielen Fehler die ihre geringe Flexibilität hervorruft.

Danach widmet man sich der vermutlich größten Gruppe, der Nummer 2. Hier verfährt der Algorithmus wie ursprünglich vorgesehen.

Schlussendlich ergänzt man nur alle offengebliebenen Slots mit Jokern, also Personen aus Gruppe 1, um eine extrem einfach zu erreichende Konfiguration zu bekommen.

Schön wird sie aber nicht zwangsweise sein und ethisch betrachtet, ist es auch mehr als verwerlich sämtlichen mutmaßlichen Egoisten ihren Willen einzugestehen und eventuelle Abstufungen in den unterschiedlichen Ratings des einzelnen Jokers zu seinem Nachteil zu ignorieren.

Optimierungsmöglichkeit 3 – Berücksichtigung der individuellen Unterschiede in den Ratings

Eine "gute Sache" wäre auch ein Algorithmus, der nicht nur ein r berücksichtigt, sondern dabei noch individuelle Abstufungen in den Ratings berücksichtigt. So ist der Fall nicht auszuschließen, dass S_n Tutorium x höher bewertet hat, als Tutorium y aber dennoch auf Grund der linearen Vorgehensweise in nichtpräferierter Weise in der Konfiguration zugeteilt wurde.

Eine optimale Konfiguration $K_{optimal}$ mag vielleicht existieren, allerdings ist es bereits bei kleinen Werten für s und t mit enorm vielen zusätzlichen Arbeitsschritten verbunden, die individuellen Präferenzen der einzelnen StudentInnen zu berücksichtigen. Hinzu kommt, dass man sich dem Optimum in der Theorie dabei meist nur nähern kann und es nicht realistisch ist die Anzahl der Konfigurationen $|K|$ komplett zu berechnen und gegeneinander abzuwiegen.

Die einzige Berücksichtigung individueller Präferenzen der StudentInnen, die sowohl vom Aufwand als auch von ihrem Ergebnis her interessante Ergebnisse bringt, ist die *nachträgliche Anpassung*. Hierbei werden nach einer ermittelten gültigen Konfiguration K eine Anzahl Durchläufe gestartet, die das von S_n vergebene Rating für T_j mit den Ratings von in den anderen Tutorien $\neq T_j$ gelegenen StudentInnen, welche mit Ihrer Platzierung NICHT vollständig zufrieden sind, also keine 5 als Rating für ihre aktuelle Platzierung gegeben haben.

Die genaue Anzahl der Durchläufe d abhängig von einer beliebig zu wählenden Konstante x sollte dabei bei steigenden s und t auf Grund der exponentiell größeren Anzahl an Arbeitsschritten antiproportional angepasst werden.

$$d = \frac{x}{s \cdot t}$$

Optimierungsmöglichkeit 4 – Vorsortierung der *.csv-Dateien*

Eine chronologische Zuordnung der Tutorien von 1 zu erstem Tutorium am Montag und t letztem Tutorium am Freitag vorausgesetzt existiert noch eine weitere Möglichkeit zur Optimierung:

StudentInnen, die für Tutorien $\in [T_1, T_{\frac{t}{2}}]$ wesentlich höhere Bewertungen abgegeben haben, als für die Tutorien in der zweiten Hälfte der Woche $\in [T_{\frac{t}{2}+1}, T_t]$ sollten nämlich auch früher vom Algorithmus bearbeitet werden, da sie sonst später die Listen "verschlacken" und unnötig viele Arbeitsschritte provozieren.

Aufgabe 3 GUI – GUI, GUI, GUI

G: 12/18
C: 7/7 D: 5/7 T: 0/4

Nach *javaDoc*-Vorgaben implementiert.

Doku für displayResultDialog() nicht vollständig D-2

Quellcode

Es folgt der Quellcode der implementierten Klassen, bzw. ergänzten Methoden in bereits existierenden Klassen.

Klasse `Studi`

```
1  package pi.uebung02;
2
3  import pi.uebung02.spec.IStudi;
4  import pi.uebung02.spec.ITheaPP;
5
6  /**
7   * Stellt die Klasse für Studierende dar. Eine StudentIn hat einen Account (Namen) und
      eine bestimmte Anzahl
8   * von Bewertungen (zwischen 0 und {@link ITheaPP#MAX_RATING} inklusive) für eine
      bestimmte Anzahl von Tutorien.
9   * Zusätzlich hat eine StudentIn den Index des Tutoriums, dem sie aktuell zugeordnet ist.
       Dieser Wert darf -1 sein um
10  * anzuzeigen, dass die StudentIn aktuell keinem Tutorium zugeordnet ist.
11  */
12 public class Studi implements IStudi {
13
14   /** der Name des Studi */
15   private String account;
16
17   /** die Bewertungen, die der Studi für alle Tutorien abgegeben hat */
18   private int[] bewertungen;
19
20   /**
21    * Nummer des Tutoriums, dem der/die Studi zugeordnet ist.
22    * Wird als Index im Array {@code bewertungen} verwendet.
23    */
24   private int tutoriumNr = -1;
25
26   /**
27    * Der Konstruktor erwartet den Account des neuen Studierenden und ein Array mit den
        Tutoriumsbewertungen.
28    * Falls mindestens einer dieser Parameterwerte {@code null} oder leer ist oder das
        Array weniger als
29    * {@link ITheaPP#MIN_NO_OF_TUTORIALS} oder mehr als {@link ITheaPP#MAX_NO_OF_TUTORIALS
        } Elemente enthält, wird eine
30    * {@link IllegalArgumentException} ausgelöst. Falls eine Bewertung in dem Array
        negativ oder größer als
31    * {@link ITheaPP#MAX_RATING} ist, wird ebenfalls eine {@link IllegalArgumentException}
         ausgelöst.
32    *
33    * @param pAccount Name des Studi
34    * @param pBewertungen die Bewertungen, die der Studi für alle Tutorien abgegeben hat
35    */
36   public Studi(final String pAccount, final int[] pBewertungen) {
37     if (pAccount==null || pAccount.equals("") || pBewertungen==null)
38       throw new IllegalArgumentException();
39     int l = pBewertungen.length;
40     if (l==0 || l<ITheaPP.MIN_NO_OF_TUTORIALS || ITheaPP.MAX_NO_OF_TUTORIALS<l)
41       throw new IllegalArgumentException();
42     for (int b: pBewertungen) {
43       if (b<0 || ITheaPP.MAX_RATING<b)
44         throw new IllegalArgumentException();
45     }
46
```

Besser:
p.Account.trim().equals("");

```
47      this.account = pAccount;
48      this.bewertungen = pBewertungen;
49    }
50
51    /** {@inheritDoc} */
52    @Override
53    public int getTutorial() {
54      return tutoriumNr;
55    }
56
57      /**
58       * Setzt den Index des aktuellen Tutoriums dieser StudentIn auf den übergebenen Index
            . Um eine StudentIn keinem
59       * Tutorium zuzuordnen, ist der Index -1 zu verwenden. Wenn der Parameterwert kleiner
             als -1 oder größer als die
60       * zuvor festgelegte Anzahl von Tutorien ist, wird eine {@link
            IllegalArgumentException} ausgelöst.
61       *
62       * @param pTutorialIndex
63       *            der Index des Tutoriums, dem diese StudentIn zugeordnet werden soll,
            oder -1 falls diese StudentIn in
64       *            keinem Tutorium sein soll
65       */
66    @Override
67    public void setTutorial(int pTutorialIndex) {
68      if (pTutorialIndex<-1 || bewertungen.length<=pTutorialIndex) {
69        throw new IllegalArgumentException();
70      }
71      tutoriumNr = pTutorialIndex;
72    }
73
74    /** {@inheritDoc} */
75    @Override
76    public int getRating(int pTutorialIndex) {
77      if (pTutorialIndex<0 || bewertungen.length<=pTutorialIndex) {
78        throw new IllegalArgumentException();
79      }
80      return bewertungen[pTutorialIndex];
81    }
82
83    /**
84     * Vergleicht zwei Studis.
85     * Zwei Studis sind gleich, wenn ihre Accounts übereinstimmen.
86     * @param andererStudi der Studi, mit dem verglichen wird
87     * @return {@code true} falls die Studis namensgleich sind
88     */
89    @Override
90    public boolean equals(Object andererStudi) {
91      Studi anderer = (Studi)andererStudi;
92      return account.equals(anderer.account);
93    }
94
95    /**
96     * Gibt eine Zeichenkettendarstellung des Studi zurück.
97     * Diese enthält den Account der/des Studierenden und dahinter in
98     * eckigen Klammern die Bewertung für das aktuell zugeordnete Tutorium.
99     * @return Name und Zufriedenheit des Studi, wenn er einem Tutorium
100    *         zugeordnet ist, Name{@code [-1]} sonst
101    */
102   @Override
103   public String toString() {
104     int zufriedenheit = tutoriumNr==-1 ? -1 : bewertungen[tutoriumNr];
105     return account+"["+zufriedenheit+"]";
106   }
107
108   /**
109    * Gibt die Anzahl der vom Studi abgegebenen TutoriumsBewertungen zurück.
110    * @return Anzahl der Bewertungen
111    */
112   int gibAnzahlBewertungen() {
113     return bewertungen.length;
```

Wenn man equals überschreibt sollte man als aller erstes auf null prüfen und dann ob das übergebene Objekt überhaußt vom eigenen Typ ist. Das geht mit: <variable> instanceof <Typ>

```
114   }
115
116 }
```

Klasse TheaPP

```
1  /*
2   * Copyright 2014 AG Softwaretechnik, University of Bremen, Germany
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *   http://www.apache.org/licenses/LICENSE-2.0
9   *
10  *   Unless required by applicable law or agreed to in writing, software
11  *   distributed under the License is distributed on an "AS IS" BASIS,
12  *   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  *   See the License for the specific language governing permissions and
14  *   limitations under the License.
15  */
16 package pi.uebung02;
17
18 import pi.uebung02.exceptions.TheaPPParseException;
19 import pi.uebung02.spec.IStudi;
20 import pi.uebung02.spec.ITheaPP;
21 import pi.uebung02.spec.ITheaPPParser;
22
23 import java.io.IOException;
24 import java.util.ArrayList;
25 import java.util.List;
26
27 /**
28  * Realisiert die nötige Logik, um eine gegebene Liste von Studierenden mit ihren
      Bewertungen gemäß einer
29  * konfigurierbaren Beschränkung auf Tutorien zu verteilen.
30  */
31 public class TheaPP implements ITheaPP {
32
33     /** die Anzahl der Tutorien */
34   private final int noOfTutorials;
35
36   /** die maximale Anzahl an Studierenden pro Tutorium */
37   private final int maxTutorialSize;
38
39     /**
40      * Die aktuelle Konfiguration als Liste von Tutorien.
41      * Ein Tutorium wird hier als Liste von Studi-Objekten realisiert.
42      */
43     private final List<List<IStudi>> tutorials;
44
45     /**
46      * Der Konstruktor erwartet die Anzahl an Tutorien und die maximale Anzahl von
          Studierenden pro Tutorien. Falls die
47      * Anzahl der Tutorien nicht innerhalb der durch {@linkplain ITheaPP#
          MIN_NO_OF_TUTORIALS} und
48      * {@linkplain ITheaPP#MAX_NO_OF_TUTORIALS}) vorgegebenen Grenzen liegt, wird eine {
          @link IllegalArgumentException}
49      * ausgelöst, ebenso falls die maximale Anzahl der Studierenden pro Tutorium kleiner
          als 2 oder größer als
50      * {@link ITheaPP#MAX_STUDENTS_IN_TUTORIAL} ist.
51      *
52      * @param pNoOfTutorials
53      *            die Anzahl der Tutorien
54      * @param pMaxTutorialSize
55      *            die maximale Anzahl an Studierenden pro Tutorium
56      */
57     public TheaPP(final int pNoOfTutorials, final int pMaxTutorialSize) {
58     if (pNoOfTutorials<MIN_NO_OF_TUTORIALS
59         || MAX_NO_OF_TUTORIALS<pNoOfTutorials
60         || pMaxTutorialSize<2
```

```
61          || MAX_STUDENTS_IN_TUTORIAL<pMaxTutorialSize) {
62        throw new IllegalArgumentException();
63        }
64
65        noOfTutorials = pNoOfTutorials;
66        maxTutorialSize = pMaxTutorialSize;
67          tutorials = createInitialConfiguration();
68      }
69
70      /**
71       * Erzeugt die Startkonfiguration, d.h. eine Liste von Tutorien.
72       * Ein Tutorium ist dabei eine (anfangs leere) Liste von Studi-Objekten.
73       *
74       * @return die Startkonfiguration als Liste von Studi-Listen
75       */
76      private List<List<IStudi>> createInitialConfiguration() {
77        List<List<IStudi>> konfig = new ArrayList<List<IStudi>>();
78        for (int i = 0; i < noOfTutorials; i++) {
79          konfig.add((List<IStudi>) new ArrayList<IStudi>());
80        }
81          return konfig;
82      }
83
84      /** {@inheritDoc} */
85      @Override
86      public final boolean distributeStudis(final List<IStudi> studis, final int pMinRating
          ) {
87      if (pMinRating<1 || MAX_RATING<pMinRating || studis==null) {
88        throw new IllegalArgumentException();
89        }
90      resetConfiguration();
91      for (IStudi s: studis) s.setTutorial(-1);
92
93          int lastStudiIndex = -1;
94          do {
95            lastStudiIndex = nextConfiguration(tutorials, studis, lastStudiIndex,
                pMinRating);
96          } while (lastStudiIndex >= 0);
97      return lastStudiIndex == -1;
98      }
99
100     /**
101      * Berechnet die Folgekonfiguration der gegebenen Konfiguration für die gegebene
           Liste von Studierenden, den Index
102      * des zuletzt zugeordneten Studis der gegebenen Konfiguration und die
           Mindestbewertung. Die Berechnung der
103      * Folgekonfiguration geschieht "in-place", d.h. die gegebene Konfiguration wird
           direkt durch diese Methode
104      * verändert. Gibt den Index des bei der Berechnung der Folgekonfiguration zuletzt
           zugeordneten Studis zurück.
105      *
106      * Die Methode ist nicht private, sondern package-private, da sie durch JUnit-Tests
           getestet werden soll. Da das
107      * Paket vor einer Auslieferung versiegelt wird, ist sie damit von außen nicht
           aufrufbar. Aus diesem Grund werden
108      * die Parameterwerte auch nicht auf sinnvolle Werte überprüft. Wenn es keine
           Folgekonfiguration gibt, weil eine Lösung
109      * gefunden wurde (Fall 1) oder es keine Lösung gibt (Fall 4b), drückt die Methode
           das mit ihrem Rückgabewert aus.
110      *
111      * @param configuration
112      *            die Konfiguration, deren Folgekonfiguration errechnet werden soll
113      * @param students
114      *            die Liste der Studierenden
115      * @param lastStudiIndex
116      *            der Index der zuletzt zugeordneten StudentIn der gegebenen
           Konfiguration
117      * @param minRating
118      *            die Mindestbewertung
119      * @return -1 wenn eine Lösung gefunden ist, -2 wenn es keine Lösung gibt,
120      *            sonst den Index des zuletzt zugeordneten Studis
```

```
121      */
122     final int nextConfiguration(final List<List<IStudi>> configuration, final List<IStudi
          > students,
123             final int lastStudiIndex, final int minRating) {
124       int anzahlStudis = students.size();
125       int n = lastStudiIndex;
126 
127       if (constraintsSatisfied(configuration, minRating)) { // Konfiguration ist gültig
128         if (n == anzahlStudis-1) { // Fall 1 "n = s"
129            n = -1;
130         } else { // Fall 2: "n < s"
131           IStudi s = students.get(++n);
132           configuration.get(0).add(s);
133           s.setTutorial(0);
134         }
135 
136       } else { // Konfiguration ist ungültig
137         IStudi s = students.get(n);
138         int j = s.getTutorial();
139       List<IStudi> tj = configuration.get(j);
140         if (j < noOfTutorials-1) { // Fall 3: aktuelles Tutorium ist kleiner als größter
            gültiger Index
141           tj.remove(tj.size()-1);
142           configuration.get(j+1).add(s);
143           s.setTutorial(j+1);
144         } else { // Fall 4: aktuelles Tutorium ist letztes Tutorium
145           do {
146             tj.remove(tj.size()-1);
147             s.setTutorial(-1);
148             n--;
149             if (n<0) return -2; // Fall 4b: Es gibt keinen Studi, der nicht im letzten
                  Tutorium ist.
150             s = students.get(n);
151             j = s.getTutorial();
152             tj = configuration.get(j);
153           } while (j == noOfTutorials-1);
154           tj.remove(tj.size()-1); // Fall 4a: Der "größte" Studi, der nicht im letzten T.
                ist, wird weitergesetzt.
155           configuration.get(j+1).add(s);
156           s.setTutorial(j+1);
157         }
158       }
159 
160       return n;
161   }
162 
163     /**
164      * Prüft, ob die gegebene Konfiguration hinsichtlich der gegebenen
165      * Mindestbewertung und den Tutoriumsgrößen gültig ist. Wenn das
166      * so ist, wird {@code true} zurückgegeben, sonst {@code false}.
167      *
168      * Die Methode ist nicht private, sondern package-private, da sie durch JUnit-Tests
           getestet werden soll. Da das
169      * Paket vor einer Auslieferung versiegelt wird, ist sie damit von außen nicht
           aufrufbar.
170      *
171      * @param configuration
172      *            die zu prüfende Konfiguration als Liste von Studi-Listen
173      * @param pMinRating
174      *            die Mindestbewertung der Studis für ihre Tutorien
175      * @return {@code true} falls kein Tutorium mehr als die erlaubte Maximalzahl von
           Studis enthält und kein Studi
176      *         einem Tutorium zugeordnet ist, für das sie oder er eine geringere
           Bewertung als die gegebene
177      *         Mindestbewertung vergeben hat, ansonsten {@code false}.
178      */
179     final boolean constraintsSatisfied(final List<List<IStudi>> configuration, final int
          pMinRating) {
180       for (List<IStudi> t : configuration) {
181         if (t.size() > maxTutorialSize) return false;
182         for (IStudi s: t) {
```

```
183           if (s.getRating(s.getTutorial()) < pMinRating) return false;
184         }
185       }
186       return true;
187     }
188
189   /** {@inheritDoc} */
190     @Override
191     public final Object getValueAt(final int row, final int column) {
192       try {
193       return tutorials.get(row).get(column);
194     } catch (IndexOutOfBoundsException e) {
195       return null;
196     }
197     }
198
199     /** Setzt die Konfiguration auf die Startkonfiguration zurück. */
200     private void resetConfiguration() {
201         for (List<IStudi> tutorial: tutorials) {
202             tutorial.clear();
203         }
204     }
205
206     /**
207      * Liest eine der gegebenen Dateien ein und versucht, Tutoriumszuordnungen
208      * mit verschiedenen MindestBewertungen vorzunehmen. Gibt die Ergebnisse
209      * einer Zuordnung auf der Konsole aus.
210      *
211      * @param args wird ignoriert
212    */
213     public static void main(String[] args) {
214       String dateiName = "FiveTutorials.csv";
215       List<IStudi> studis;
216       try {
217       studis = new TheaPPParser().parseStudents(dateiName);
218     } catch (TheaPPParseException e) {
219       System.out.println("Fehler beim Parsen der Bewertungen in "+dateiName+"."); return;
220     } catch (IOException e) {
221       System.out.println("Fehler beim Einlesen der Datei "+dateiName+"."); return;
222     }
223
224       int anzahlTutorien = ((Studi)studis.get(0)).gibAnzahlBewertungen();
225       int tutoriumsGroesse = (int) Math.ceil(((double)studis.size())/anzahlTutorien);
226
227     TheaPP theapp = new TheaPP(anzahlTutorien, tutoriumsGroesse);
228
229       for (int minRating: new int[]{5,3,2,1}) {
230       System.out.println(dateiName+" mit Mindestbewertung "+minRating+":");
231         if (!theapp.distributeStudis(studis, minRating)) {
232           System.out.println("Keine Lösung.");
233         } else {
234           for (int zeile=0; zeile<anzahlTutorien; zeile++) {
235               for (int spalte=0; spalte<tutoriumsGroesse; spalte++) {
236                 Studi s = (Studi)theapp.getValueAt(zeile, spalte);
237                 System.out.print((s==null ? "          " : s) +" ");
238               }
239               System.out.println();
240           }
241         }
242       System.out.println();
243       }
244     }
245
246 }
```

Klasse TheaPPParser

```
1 package pi.uebung02;
2
3 import java.io.IOException;
4 import java.util.List;
```

```
5
6  import pi.uebung02.exceptions.TheaPPParseException;
7  import pi.uebung02.spec.IStudi;
8  import pi.uebung02.spec.ITheaPP;
9  import pi.uebung02.spec.ITheaPPParser;
10
11 import java.io.*;
12 import java.util.ArrayList;
13
14 public class TheaPPParser implements ITheaPPParser {
15
16   public TheaPPParser() {
17
18   }
19
20 @Override
21 public List<String> readFile(final String filename) throws IOException {
22   if(filename == null || filename == ""){
23     throw new IllegalArgumentException();
24   }
25   List<String> output = new ArrayList<String>();
26   try{
27   BufferedReader in = new BufferedReader(new FileReader(filename));
28   String line = in.readLine();
29   while(line !=null) {
30     output.add(line);
31     line = in.readLine();
32   }
33   }
34   catch(IOException e){
35     System.out.println("Error while Reading file ...");
36   }
37   return output;
38 }
39 @Override
40 public IStudi parseStudent(final String line) throws TheaPPParseException {
41   if(line == null || line == "" ){
42     throw new IllegalArgumentException();
43   }
44
45   String[] parameters = line.split(";");
46   int[] ratings = new int[parameters.length-1];
47   try{
48   if((parameters.length-1) < ITheaPP.MIN_NO_OF_TUTORIALS || (parameters.length-1) >
       ITheaPP.MAX_NO_OF_TUTORIALS ){
49     throw new TheaPPParseException("Line: '"+line+"'");
50   }}
51   catch(TheaPPParseException e){
52     System.out.println("Line out of Date!");
53   }
54   try{
55   for(int i=1; i<parameters.length;i++){
56     if(Integer.parseInt(parameters[i]) < 0 || Integer.parseInt(parameters[i]) > ITheaPP.
         MAX_RATING){
57       throw new TheaPPParseException("Line: '"+line+"'");
58     }
59     ratings[i-1]= Integer.parseInt(parameters[i]);
60   }}
61   catch(TheaPPParseException e){
62     System.out.println("Parseing failed!");
63   }
64   return new Studi(parameters[0],ratings);
65 }
66 @Override
67 public List<IStudi> parseStudents(final String filename) throws TheaPPParseException,
     IOException {
68   List<String> lines = readFile(filename);
69   List<IStudi> output= new ArrayList<IStudi>();
70   for(int i=0;i<lines.size();i++){
71     output.add(parseStudent(lines.get(i)));
72   }
```

Den leeren Konstruktor kann man auch weg lassen, da kein anderer Konstruktor vorhanden ist.

auch hier besser: filename.trim() == ""

Siehe oben

```
73   return output;
74 }
75 }
```

Methode `main()` in Klasse `TheaPPParser`

```
213     public static void main(String[] args) {
214       String dateiName = "FiveTutorials.csv";
215       List<IStudi> studis;
216       try {
217       studis = new TheaPPParser().parseStudents(dateiName);
218     } catch (TheaPPParseException e) {
219       System.out.println("Fehler beim Parsen der Bewertungen in "+dateiName+"."); return;
220     } catch (IOException e) {
221       System.out.println("Fehler beim Einlesen der Datei "+dateiName+"."); return;
222     }
223
224       int anzahlTutorien = ((Studi)studis.get(0)).gibAnzahlBewertungen();
225       int tutoriumsGroesse = (int) Math.ceil(((double)studis.size())/anzahlTutorien);
226
227     TheaPP theapp = new TheaPP(anzahlTutorien, tutoriumsGroesse);
228
229       for (int minRating: new int[]{5,3,2,1}) {
230       System.out.println(dateiName+" mit Mindestbewertung "+minRating+":");
231         if (!theapp.distributeStudis(studis, minRating)) {
232           System.out.println("Keine Lösung.");
233         } else {
234           for (int zeile=0; zeile<anzahlTutorien; zeile++) {
235               for (int spalte=0; spalte<tutoriumsGroesse; spalte++) {
236                 Studi s = (Studi)theapp.getValueAt(zeile, spalte);
237                 System.out.print((s==null ? "          " : s) +" ");
238               }
239               System.out.println();
240           }
241         }
242       System.out.println();
243       }
244     }
245
246 }
```

GUI

Javadoc wurde nicht ergänzt

```
243     private void displayResultDialog(final TheaPP theaPP) {
244       String[][] accs = new String[(int)noOfTutorialsSpinner.getValue()][(int)
            maxTutorialSizeSpinner.getValue()];
245       String[] tutNames = new String[(int)noOfTutorialsSpinner.getValue()];
246       for(int i=0;i<(int)noOfTutorialsSpinner.getValue();i++){
247         for(int y=0;y<(int)maxTutorialSizeSpinner.getValue();y++){
248           if(theaPP.getValueAt(i, y) != null){
249           accs[i][y] = theaPP.getValueAt(i, y).toString();
250           }
251           else{
252             accs[i][y] = "";
253           }
254         }
255       tutNames[i] = "Tut: " + i;
256       }
257       JTable table = new JTable(accs ,tutNames);
258       TheaPPResultDialog dialog = new TheaPPResultDialog(table);
259         dialog.setVisible(true);
260     }
261
262     /**
263      * Erzeugt einen Parser zum Einlesen der Studierendendaten und die
            Benutzungsschnittstelle und zeigt diese an.
264      *
265      * @param args
266      *            werden ignoriert
267      *
```

```
268      */
269     public static void main(final String[] args) {
270         SwingUtilities.invokeLater(new Runnable() {
271             public void run() {
272                 ITheaPPParser parser = new TheaPPParser();
273                 TheaPPMainFrame mainFrame = new TheaPPMainFrame(parser);
274                 mainFrame.pack();
275                 mainFrame.setVisible(true);
276             }
277         });
278     }
279
280 }
```