Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Peer Kröger
Michael Fromm, Florian Richter

Einführung in die Programmierung

WS 2018/19

Übungsblatt 10: Objekt-orientierte Programmierung

Besprechung 11.01 - 18.01.2019

Aufgabe 10-1 *Datenkapselung*

In einem Seminarraum stehen eine gewisse Anzahl an Stühlen für eine Veranstaltung. Entwerfen Sie eine entsprechende Klasse `Seminarraum`. Diese soll folgenden Anforderungen entsprechen:

- Alle Klassenattribute sollen von außen weder sichtbar noch veränderbar sein.
- Ein Seminarraum hat eine obere Schranke: es dürfen nur begrenzt viele Stühle in den Raum gestellt werden. Der Konstruktor muss dies berücksichtigen.
- Eine getter-Methode ermöglicht die Abfrage der Anzahl der Stühle, die sich im Raum befinden.
- Eine setter-Methode ermöglicht das Ändern der Bestuhlung in ihren erlaubten Grenzen.
- Zwei Methoden zum Erhöhen und Verringern der Bestuhlung erleichtern die Handhabung.
- Eine Überladung der toString-Methode gibt eine textuelle Repräsentation des Raumes zurück.
- Die Vergleichsfunktion equals vergleicht die Anzahl der Stühle zweier Räume.

Lösungsvorschlag:

```
1  public class Seminarraum {
2
3      private int anzahlStuehle = 0;
4      private final int maxStuehle;
5
6      public Seminarraum(int maxStuehle) {
7          if(maxStuehle < 0)
8              maxStuehle = 0;
9          this.maxStuehle = maxStuehle;
10     }
11
12     public int getStuehle() {
13         return anzahlStuehle;
14     }
15
16     public void setStuehle(int anzahl) {
17         if(anzahl <= maxStuehle && anzahl >= 0)
18             anzahlStuehle = anzahl;
19     }
20
21     public void inc() {
22         if(anzahlStuehle < maxStuehle)
23             anzahlStuehle++;
24     }
```

```
25
26      public void dec() {
27          if(anzahlStuehle > 0)
28              anzahlStuehle--;
29      }
30
31      public String toString() {
32          return "Der Raum hat " + anzahlStuehle +
33              " von maximal " + maxStuehle + " Stuehlen.";
34      }
35
36      public boolean equals(Seminarraum r) {
37          return this.anzahlStuehle == r.getStuehle();
38      }
39
40  }
```

Aufgabe 10-2 *Vektoren*

Ein Vektor $v = \begin{pmatrix} 1 & 4 & 3 & 8 \end{pmatrix}$ kann als Array `[1 4 3 8]` umgesetzt werden. Implementieren Sie eine Vektorklasse, die ein Integer-Array (ja, die Wrapperklasse) als privates Klassenattribut besitzt. Gemäß des objektorientierten Paradigmas soll diese Klasse nun mit Funktionalität versehen werden:

(a) Implementieren Sie einen Konstruktor, der einen Null-Vektor (Ein Vektor, der nur Nullen als Komponenten enthält) einer definierbaren Größe initialisiert.

(b) Implementieren Sie außerdem einen Konstruktor, der zusätzlich den Vektor mit einem wählbaren Wert initialisiert, also z.B. einen Vektor $\begin{pmatrix} 2 & 2 & 2 & 2 & 2 \end{pmatrix}$ erstellt.

(c) Implementieren Sie lesende und schreibende Zugriffsmethoden (Getter/Setter) für einzelne Komponenten. Achten Sie auf die Dimension des Arrays. Wird auf ungültige Arraypositionen zugegriffen, so geben Sie `null` zurück.

(d) Implementieren Sie eine Methode `boolean equals(Vektor other)`, der zwei Vektoren auf Gleichheit prüft. Sie benötigen dafür einen weiteren Getter. Überlegen Sie, wofür und implementieren ihn.

(e) Implementieren Sie eine Methode `boolean contains(int k)`, der einen Wert im Vektor sucht. Wenn der Wert enthalten ist, wird true zurückgegeben. Ist der Wert nicht enthalten, geben Sie false zurück.

(f) Implementieren Sie eine toString-Methode, die das Array als String (z.B. `[1 4 3 8]`) zurückgibt.

(g) Implementieren Sie eine Vektoraddition `Vektor addiere(Vektor other)`, die einen Vektor mit einem anderen Vektor komponentenweise addiert. Bei ungleichen Dimensionen geben sie `null` zurück.

(h) Implementieren Sie ein Skalarprodukt `Integer skalarprodukt(Vektor other)`, das bei gleichen Dimensionen die i-ten Komponenten paarweise multipliziert und diese Produkte aufsummiert und sonst `null` zurückgibt.

Achten Sie in der gesamten Aufgabe darauf, dass ein Objekt immer auch `null` sein kann!

Lösungsvorschlag:

```
1   public class Vektor {
2       private Integer[] vek;
3
4       public static void main(String[] args) {
5           Vektor v1 = new Vektor(3,4);
```

```
6           Vektor v2 = new Vektor(3,1);
7           v2.set(4,9);
8           System.out.println(v1);
9           System.out.println(v1.addiere(v2));
10          System.out.println(v1.skalarprodukt(v2));
11          Vektor v3 = null;
12          System.out.println(v3);
13      }
14
15      public Vektor(int dimension) {
16          if(dimension <= 0)
17              dimension = 1;
18          vek = new Integer[dimension];
19          for(int i = 0; i < vek.length; i++) {
20              vek[i] = 0;
21          }
22      }
23
24      public Vektor(int dimension, int defaultValue) {
25          this(dimension);
26          for(int i = 0; i < vek.length; i++) {
27              vek[i] = defaultValue;
28          }
29      }
30
31      public Integer get(int i) {
32          if(i < 0)
33              return null;
34          if(i > vek.length - 1)
35              return null;
36          return vek[i];
37      }
38
39      public int getDim() {
40          return vek.length;
41      }
42
43      public void set(int i, int value) {
44          if(0 <= i && i < vek.length)
45              vek[i] = value;
46      }
47
48      public boolean equals(Vektor v) {
49          if(v == null)
50              return false;
51          if(v.getDim() != this.getDim())
52              return false;
53          for(int i = 0; i < vek.length; i++) {
54              if(vek[i] != v.get(i))
55                  return false;
56          }
57          return true;
58      }
59
60      public boolean contains(int k) {
61          for(int i = 0; i < vek.length; i++) {
62              if(vek[i] == k)
63                  return true;
64          }
```

```
65          return false;
66      }
67
68      public String toString() {
69          String str = "[";
70          for(int i = 0; i < vek.length; i++) {
71              if(i != 0)
72                  str += " ";
73              str += vek[i];
74          }
75          str += "]";
76          return str;
77      }
78
79      public Vektor addiere(Vektor v) {
80          if(v == null)
81              return null;
82          if(v.getDim() != this.getDim())
83              return null;
84          Vektor sum = new Vektor(vek.length);
85          for(int i = 0; i < vek.length; i++) {
86              sum.set(i,vek[i]+v.get(i));
87          }
88          return sum;
89      }
90
91      public Integer skalarprodukt(Vektor v) {
92          if(v == null)
93              return null;
94          if(v.getDim() != this.getDim())
95              return 0;
96          int sum = 0;
97          for(int i = 0; i < vek.length; i++) {
98              sum += vek[i]*v.get(i);
99          }
100         return sum;
101     }
102
103 }
```

Aufgabe 10-3 *Sudoku (optional)*

Sudoku ist ein Logikrätsel, das sich gut für das Spielen unterwegs eignet. Es besteht in der klassischen Variante aus einer 9×9-großen Tabelle, die nochmal etwas feiner in 9 3×3 große Blöcke partitioniert ist. Gesucht ist eine Belegung der Tabelle, sodass folgende drei Regeln erfüllt sind:

- Eine bestimmte Menge von Zellen ist je nach Schwierigkeitsgrad unveränderlich vorgegeben.
- In jeder Reihe muss jede Ziffer genau einmal vorkommen.
- In jeder Spalte muss jede Ziffer genau einmal vorkommen.
- In jedem `3x3`-Block (von denen es 9 Stück gibt) muss jede Ziffer genau einmal vorkommen.
- Alle Zellen müssen genau eine Ziffer enthalten.

In dieser Aufgabe sollen Sie einen Generator für solche Rätsel implementieren.

(a) Implementieren Sie eine Klasse `Sudoku`. Als Attribut soll diese Klasse ein zweidimensionales Array passenden Typs enthalten. Es genügt, wenn Sie sich auf 9×9-Sudokus beschränken.

Implementieren Sie einen Konstruktor, der ohne Argumente aufgerufen wird. Dieser soll das Spielfeld mit der richtigen Größe neu anlegen. Außerdem soll er alle Felder mit einer gültigen Startbelegung initialisieren. Diese kann ein Ihnen bekanntes Sudoku sein oder einer einfachen Generierungsregel folgen, die die Sudoku-Eigenschaften nicht verletzt. Einen Vorschlag finden Sie hier:

```
-------------------------
| 1 2 3 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |
-------------------------
| 2 3 4 | 5 6 7 | 8 9 1 |
| 5 6 7 | 8 9 1 | 2 3 4 |
| 8 9 1 | 2 3 4 | 5 6 7 |
-------------------------
| 3 4 5 | 6 7 8 | 9 1 2 |
| 6 7 8 | 9 1 2 | 3 4 5 |
| 9 1 2 | 3 4 5 | 6 7 8 |
-------------------------
```

Hinweis: Die obige Matrix m kann mithilfe der Formel $m_{i,j} = (3i + i/3 + j) \mod 9 + 1$ generiert werden.

(b) Implementieren Sie eine `toString`-Methode. Diese soll das Sudoku-Objekt als einen String zurückgeben, der der oben gezeigten Ausgabe entspricht. Für später sollen nicht belegte Felder (repräsentiert durch den int-Wert 0) ein Leerzeichen ausgeben.

(c) Es gibt ein paar Operationen auf einem gültigen Sudoku, welche dessen Eigenschaften erhalten. Eine Spaltenvertauschung innerhalb eines 9×3 Blocks (1-3,4-6,7-9) ist daher eine gültige Transformation. Sie erweitert die Menge der generierbaren Sudokus um $3!^3$ soviele Möglichkeiten. Implementieren Sie eine Funktion `permutateRows(int a, int b)`, die zwei Zahlen entgegennimmt, diese auf ihre Gültigkeit prüft (überlegen Sie, was das bedeutet), und die beiden Spalten vertauscht. Implementieren Sie dies ebenfalls für eine Reihenvertauschung `permutateColumns(int a, int b)`.

(d) Auch eine Vertauschung zweier ganzer Blockspalten oder Blockreihen zerstört keine der Eigenschaften. Zum Beispiel kann im obigen Beispiel der Block der Reihen 1-3 mit dem Reihenblock 4-6 getauscht werden. Implementieren Sie also analog zur vorherigen Teilaufgabe die Methoden `permutateStack(int a, int b)` und `permutateBand(int a, int b)`. Achten Sie auf gültige Eingaben. Beide Methoden erweitern Ihre Sudokuanzahl um den Faktor 3!.

(e) Implementieren Sie 4 Methoden `permutateRows()`, `permutateColumns()`, `permutateStack()` sowie `permutateBand()`. Diese sollen zufällig entsprechende Parameter bestimmen und die zugehörige Permutation aufrufen. Sie können dazu die Klasse `java.util.Random` importieren. Fügen Sie ein Random-Object Ihrer Klasse hinzu und benutzen Sie die Methode `int nextInt(int n)`.

(f) Zusätzlich können Sie das Spielfeld an einer der Diagonalen spiegeln. Implementieren Sie eine Methode `transpose()`, die dies realisiert. Eine Methode genügt, eine weitere Spiegelung kann durch Kombination der bisherigen Operationen erreicht werden.

(g) Theoretisch kann Ihre Sudokuklasse nun $3!^3 * 3!^3 * 3! * 3! * 2$ viele verschiedene Sudokus generieren. Implementieren Sie eine Methode `randomPermutation()`, die zufällig eine der fünf Transformationen ausführt. Überladen Sie diese Methode mit einer weiteren Methode `randomPermutation(int n)`, die n zufällige Permutationen ausführt. Überladen Sie außerdem den Konstruktor, sodass auch er mit der Anzahl der gewünschten Permutationen ein Sudoku erstellt.

(h) Zusätzlich zu den Transformationen können wir auch die Ziffern in der Startbelegung tauschen. Implementieren Sie eine Methode `randomRow()`, welche ein Array mit den Ziffern 1-9 in zufälliger Reihenfolge zurückgibt. Nutzen Sie dieses Array in Ihrem Konstruktor, um die Startbelegung zu permutieren. Das Array `[3 4 2 8 7 1 9 5 6]` liefert dann

```
-------------------------
| 3 4 2 | 8 7 1 | 9 5 6 |
| 8 7 1 | 9 5 6 | 3 4 2 |
| 9 5 6 | 3 4 2 | 8 7 1 |
-------------------------
| 4 2 8 | 7 1 9 | 5 6 3 |
| 7 1 9 | 5 6 3 | 4 2 8 |
| 5 6 3 | 4 2 8 | 7 1 9 |
-------------------------
| 2 8 7 | 1 9 5 | 6 3 4 |
| 1 9 5 | 6 3 4 | 2 8 7 |
| 6 3 4 | 2 8 7 | 1 9 5 |
-------------------------
```

(i) Zuletzt implementieren Sie eine Methode `hide(int n)`, die n Ziffern verbirgt. Die Ausführung Ihres Hauptprogramms

```
public static void main(String[] args){
  Sudoku s = new Sudoku(100000);
  s.hide(50);
  System.out.println(s);
}
```

könnte dann folgende Ausgabe haben:

```
-------------------------
|   2   |   5   |   7   |
|     7 |   4 8 |       |
|     5 |   7 9 |     2 |
-------------------------
|       | 3   4 |     9 |
| 4 3   |     5 |   2 8 |
|       | 8   7 |     3 |
-------------------------
|       | 5     | 1   7 |
| 6     |   8   |   3   |
| 1   8 | 4     |   9   |
-------------------------
```