Dr. Holger Blaar
Dipl.-Inform. Annett Thüring
Dipl.-Inform. Steffen Schiele

Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Wintersemester 2017/ 2018

Übungsblatt 10: Objektorientierte Programmierung

Ausgabe: 15.12.2017 Abgabe: 27.12.2017

Aufgabe 1: Labyrinth (20 Punkte)

Das Ziel dieser Aufgabe ist es, einen sog. Agenten (oder auch Software-Agenten) zu programmieren, welcher eigenständig durch ein gegebenes Labyrinth manövrieren kann, um vom Eingang zum Ausgang zu gelangen. Das Labyrinth ist innerhalb einer Datei als Matrix aus „O“ und „X“ codiert, die „O“s bilden dabei die begehbaren Wege. Der Eingang ist stets oben links, der Ausgang stets unten rechts. Die Größe des Labyrinthes ist variabel, die Anzahl der Zeilen und Spalten wird der Datei vornean mitgeliefert:

```
5
5
OXXOX
OOOOX
XXXOX
XOOOO
XXXXO
```

Die Datei soll als Kommandozeilenparameter an die main-Methode übergeben werden. In der .zip Datei auf der Übungsplattform befinden sind drei Testdateien, Sie können selbstverständlich auch selber welche erstellen.

Zur Erfüllung dieser Aufgabenstellung, werden drei Klassen benötigt: `Labyrinth`, `Feld` und `Agent`. Falls nicht anders spezifiziert, sind ihre Methoden von außen zugreifbar.

Labyrinth

1. **Instanzvariablen** - Von außen nicht zugreifbare, ganzzahlige Variablen `zeilen` und `spalten` sowie ein zweidimensionales Array vom Typ `Feld` mit Namen `labyrinth`, auf welches von außen zugegriffen werden darf.
2. **Konstruktor** - Erhält einen Parameter vom Typ String, in welchem der Dateipfad steht. Diese Datei soll geöffnet (siehe Blatt 08, Sudoku Vorlage), daraus die Zeilen und Spalten extrahiert (und den Instanzvariablen zugewiesen), sowie das Labyrinth selber erstellt und eingelesen werden.
3. **getZeilen(), getSpalten()** - Sollen die getter-Methoden der entsprechenden Instanzvariablen werden.
4. **toString()** - Soll einen String zurückgeben (kein Array von Strings!), welcher das Labyrinth aus der Instanzvariable, in „O“s und „X“s codiert, enthält. Zeilenumbrüche werden mit „\n“ erzeugt. Diese Methode dient der Kontrolle, dass das Labyrinth korrekt eingelesen wurde.
5. **main** -

```java
import java.io.BufferedReader;
import java.io.FileReader;

public class Labyrinth {
  // zeilen und spalten
  // labyrinth
```

```
7
8     public Labyrinth(String datei) throws Exception {
9       // Datei oeffnen
10      // Zeilen und Spalten auslesen
11      // Array erstellen und fuellen
12    }
13
14    public static void main(String[] args) throws Exception {
15      // Labyrinth Objekt L aus args[0] erzeugen
16      // Evtl. Labyrinth ausgeben
17      Feld fStartFeld = new Feld(0,0,true);
18      Agent zeroZeroSeven = // Agent erzeugen
19      Feld fZielfeld = new Feld(L.getZeilen() - 1, L.getSpalten() - 1, true);
20      while (!zeroZeroSeven.aktuellesFeld().equals(fZielfeld)) {
21        zeroZeroSeven.geheWeiter();
22      }
23      System.out.println(zeroZeroSeven);
24    }
25  }
```

Feld

1. **Instanzvariablen** - Zwei ganzzahlige Variablen, welche die Koordinaten des Feldes angeben und eine boolesche Variable, welcher den Status des Feldes beinhaltet. Dabei ist dieser `true`, wenn das Feld begehbar ist. Sie sollen nicht außerhalb der Klasse sichtbar sein.
2. **Konstruktor** - Erhält die Zeilenkoordinate und die Spaltenkoordinate als ganzzahligen Wert, sowie den Status als `boolean` und weist sie den Instanzvariablen zu.
3. **getter** - Methoden, welche die Instanzvariablen zurückliefern sollen.
4. **equals** - Erhält ein Objekt des Typs Feld und vergleicht die Zeilen- und Spaltenkoordinaten. Bei Gleichheit liefert sie `true` zurück, sonst `false`.
5. **setStatus** - Weist der entsprechenden Instanzvariable den neuen, übergebenen Wert zu und liefert nichts zurück.
6. **toString()** - Gibt einen String zurück, welcher die Zeilen- und Spaltenkoordinate (z.B. 1 und 0) wie folgt enthält: (1,0)

In der letzten Klasse haben Sie gewisse Freiheiten, was die Lösung angeht. Falls Sie zusätzliche Methoden und Variablen benötigen, so ist ihnen dies erlaubt. Es gibt mehrere Möglichkeiten, einen Agenten zu programmieren, eine Möglichkeit wäre z.B. die umliegenden vier Felder abzufragen und die bereits besuchten zu markieren. Beachten Sie jedoch Sackgassen, Kreuzungen und den Rand des Feldes (um keine Zugriffsfehler zu erhalten). Sie können davon ausgehen, dass keine Kreuzungen auftauchen, die ausschließlich in Sackgassen enden.

Agent

1. **Instanzvariablen** - Eine Variable vom Typ `Labyrinth`, welche das zu begehende Labyrinth enthält. Eine vom Typ `Feld`, die das Feld beschreibt, auf dem der Agent gerade steht, und einen `String`, welcher die begangenen Felder speichert. Sie sollen nur für diese Klasse sichtbar sein.
2. **Konstruktor** - Erhält ein Labyrinth im entsprechenden Datentyp und das Startfeld mit dazugehörigem Typ. Hier sollen alle Instanzvariablen ihre entsprechende Initialisierung erhalten.
3. **aktuellesFeld()** - Ist die getter-Methode für das aktuelle Feld, auf dem sich der Agent derzeit befindet.

4. **toString()** - Gibt die begangenen Felder und das aktuelle Feld (in seiner Repräsentation als String) mit Zeilenumbruch am Ende zurück.
5. **geheWeiter()** - Diese Methode erhält keinen Parameter und gibt nichts zurück. Sie dient dem Zweck des Voranschreitens des Agenten auf dem Labyrinth nach der Suche des nächsten, begehbaren Feldes. Wurde solch ein Feld gefunden, soll das bisherige Feld als unbegehbar markiert und an den `String` der begangenen Felder angehangen werden. Desweiteren muss die Variable für das aktuelle Feld aktualisiert werden. Die Logik dieser Methode ist Ihnen überlassen.

Zu beachten ist, dass bei dieser Art der künstlichen Intelligenz kein Einwirken von Außen erfolgen soll. Das bedeutet, es dürfen keine Benutzereingaben getätigt werden. Falls die Methoden richtig implementiert wurden, ist folgende Ausgabe eine mögliche Lösung:

```
OXXOX
OOOOX
XXXOX
XOOOO
XXXXO

(0,0)
(1,0)
(1,1)
(1,2)
(1,3)
(2,3)
(3,3)
(3,2)
(3,1)
(3,4)
(4,4)
```