Benjamin Schürmann

Auswertung vom 07.04.2012

Benjamin Schürmann

1. Übersetzen der Java-Klassen

Dieser Abschnitt enthält etwaige Fehlermeldungen oder Warnungen des Übersetzers während des Übersetzungsvorgangs. Falls Sie hier keine Meldungen finden, kann dies bedeuten, dass Ihre hochgeladene Lösung keine Java-Dateien enthält oder alle Klassen sich fehlerfrei übersetzen ließen.

Benjamin Schürmann

2. Formale Prüfung

Dieser Abschnitt enthält das Ergebnis der formalen Prüfung. Es wird geprüft, ob alle in der Aufgabe geforderten Klassen mit allen geforderten Methoden vorhanden sind.

2.1. Übersicht der Klassen und Schnittstellen

❶	❷	❸	❹	❺	❻	Klasse oder Schnittstelle	Fehlerhinweis
✔						Klasse labyrinth.Labyrinth	

Legende: Die Klasse oder Schnittstelle ...

❶	ist vorhanden und ohne formale Fehler.
❷	ist nicht vorhanden oder nicht compilierbar.
❸	ist vorhanden, hat aber formale Fehler. Sofern es sich um formale Fehler in Methoden handelt, finden Sie Details dazu in der Methodenübersicht.
❹	enthält Fehler bzgl. der darin definierten symbolischen Konstanten.
❺	enthält Fehler bzgl. ihrer Oberklasse.
❻	enthält Fehler bzgl. der implementierten Schnittstellen.

2.2. Übersicht der Methoden

❶	❷	❸	❹	❺	Klasse	Methode	Fehlerhinweis
✔					labyrinth.Labyrinth	public labyrinth.Labyrinth(int, int)	
✔					labyrinth.Labyrinth	public int gibAnzahlWandelemente()	
✔					labyrinth.Labyrinth	public Punkt gibEndpunkt(int)	
✔					labyrinth.Labyrinth	public Punkt gibStartpunkt(int)	
✔					labyrinth.Labyrinth	public int gibBreite()	
✔					labyrinth.Labyrinth	public int gibHoehe()	
				✔	labyrinth.Labyrinth	public boolean pruefeAusserhalb(int, int)	
				✔	labyrinth.Labyrinth	public void erzeugeLabyrinth()	
				✔	labyrinth.Labyrinth	public void erzeugeAussenwand()	
				✔	labyrinth.Labyrinth	public void erzeugeWand(int, int, int)	

Benjamin Schürmann

Legende: Die Methode ist ...

❶	vorhanden und ohne formale Fehler.
❷	nicht vorhanden.
❸	vorhanden, hat aber nicht die geforderten Modifikatoren. Die von Ihnen deklarierten Modifikatoren stehen im Fehlerhinweis.
❹	vorhanden, hat aber nicht den geforderten Ergebnistyp. Der von Ihnen deklarierte Ergebnistyp steht im Fehlerhinweis.
❺	zusätzlich von Ihnen definiert.

3. Funktionale Prüfung

Dieser Abschnitt enthält das Ergebnis der funktionalen Prüfung. Es wird geprüft, ob alle in der Aufgabe geforderten Methoden für bestimmte Testdaten die erwarteten Ergebnisse liefern. Fehlermeldungen finden Sie in diesem Abschnitt auch, wenn geforderte Klassen oder Methoden fehlen. In diesem Fall scheitert bereits der Aufruf der Methoden.

Beim Test sind keine Fehler aufgetreten.

3.1. Test der Klasse Labyrinth

3.1.1. Testsituation „Ohne Aufbau von Testdaten“

Die folgenden Tests werden ohne Aufbau von Testdaten ausgeführt.

Tests der Methode Labyrinth(int, int)

	Testfall	Fehlerhinweis
✔	Erzeuge Labyrinthe der Größen 200 x 200 und 400 x 200 und prüfe, ob das Verhältnis der Laufzeiten maximal 3 beträgt.	
✘	Erzeuge Labyrinthe der Größen 300 x 300 und 600 x 300 und prüfe, ob das Verhältnis der Laufzeiten maximal 3 beträgt.	Die Zeitdauer von 60000 ms für diesen Test wurde überschritten.

3.1.2. Testsituation „Labyrinth 40 x 30“

Die Testsituation wird in folgenden Schritten aufgebaut:

1.	Erzeuge Labyrinth mit Breite 40 und Höhe 30.

Tests der Methode gibAnzahlWandelemente()

	Testfall	Fehlerhinweis
✔	Prüfe Anzahl Wandelemente des Labyrinths.	

Tests der Methode gibBreite()

	Testfall	Fehlerhinweis
✔	Prüfe Breite des Labyrinths.	

Tests der Methode gibHoehe()

	Testfall	Fehlerhinweis
✔	Prüfe Höhe des Labyrinths.	

Tests der Methode gibStartpunkt(int) und gibEndpunkt(int)

Benjamin Schürmann

	Testfall	Fehlerhinweis
✔	Prüfe Anzahl der verschiedenen Wandelemente des Labyrinths.	

Benjamin Schürmann

4. Checkstyle-Prüfung

```
Starting audit...
Audit done.
```

5. Quellcode der Java-Klassen

5.1. Labyrinth.java

```
 1: package labyrinth;
 2:
 3: import java.util.Random;
 4: import util.Punkt;
 5:
 6: /**
 7:  * Ein Objekt dieser Klasse dient der Erzeugung eines Labyrinths.
 8:  *
 9:  * @author Benjamin Schuermann, 201020367
10:  * @version 0.5rc, 06.04.2012
11:  */
12: public class Labyrinth {
13:
14:     /**
15:      * Konstanten zur Richtungsbestimmung im Raster
16:      */
17:     private static final int LINKS = 0;
18:     private static final int HOCH = 1;
19:     private static final int RECHTS = 2;
20:     private static final int RUNTER = 3;
21:     /**
22:      * Mögliche Richtungen vom Endpunkt eines jeden Wandelements Hoch(-1,0),
23:      * Runter(1,0), Rechts(0,1), Links(0,-1)
24:      */
25:     private static final Punkt[] RICHTUNG = {new Punkt(-1, 0), new Punkt(0, -1),
26:         new Punkt(1, 0), new Punkt(0, 1)};
27:     /**
28:      * Breite, Höhe, Länge des Labyrinths
29:      */
30:     private int breite;
31:     private int hoehe;
32:     private int laenge;
33:     /**
34:      * Index des nächst freien Punkts
35:      */
36:     private int wandIndex = 0;
37:     /**
38:      * Startpunkte, Endpunkte von Wandelementen
39:      */
40:     private Punkt[] startpunkte;
41:     private Punkt[] endpunkte;
42:     /**
43:      * Belegte Punkte
44:      */
```

```
45:     private boolean[][] istBelegt;
46:     /**
47:      * Aktuell gewählter Punkt
48:      */
49:     private Punkt aktuellerPunkt;
50:
51:     /**
52:      * Erzeugt ein Labyrinth aus der Größe Breite x Höhe
53:      *
54:      * @param breite Breite, des zu erzeugenden Labyrinths
55:      * @param hoehe Höhe, des zu erzeugenden Labyrinths
56:      */
57:     public Labyrinth(int breite, int hoehe) {
58:
59:         /*
60:          * Setze Breite, Höhe und Länge des Labyrinths
61:          */
62:         this.breite = breite;
63:         this.hoehe = hoehe;
64:         this.laenge = (breite + 1) * (hoehe + 1);
65:
66:         /*
67:          * Bestimme Größen der Arrays für Start-und Endpunkte
68:          */
69:         startpunkte = new Punkt[laenge];
70:         endpunkte = new Punkt[laenge];
71:
72:         /*
73:          * Bestimme Größe des istBelegt Arrays
74:          */
75:         istBelegt = new boolean[breite + 1][hoehe + 1];
76:
77:         /*
78:          * Erzeuge Außenwand
79:          */
80:         erzeugeAussenwand();
81:
82:         /*
83:          * Erzeuge Labyrinth
84:          */
85:         erzeugeLabyrinth();
86:     }
87:
88:     /**
89:      * Erzeugt die Außenwände des Labyrinths
90:      */
91:     public void erzeugeAussenwand() {
92:
93:         for (int i = 0; i <= breite - 1; i++) {
```

```
 94:
 95:             /*
 96:              * Erzeuge Obere Aussenwand
 97:              */
 98:             erzeugeWand(i, 0, RECHTS);
 99:
100:             /*
101:              * Erzeuge Untere Aussenwand
102:              */
103:             erzeugeWand(i, hoehe, RECHTS);
104:
105:         }
106:
107:         for (int i = 0; i <= hoehe - 1; i++) {
108:
109:             /*
110:              * Erzeuge Linke Aussenwand
111:              */
112:             erzeugeWand(0, i, RUNTER);
113:
114:             /*
115:              * Erzeuge Rechte Aussenwand
116:              */
117:             erzeugeWand(breite, i, RUNTER);
118:
119:         }
120:
121:     }
122:
123:     /**
124:      * Erzeugt ein neues Wandelement
125:      *
126:      * @param x x-Koordinate des Startpunkts
127:      * @param y y-Koordinate des Startpunkts
128:      * @param richtung Richtung, in welche die Wand gezogen werden soll
129:      */
130:     public void erzeugeWand(int x, int y, int richtung) {
131:
132:         /*
133:          * Erzeuge Startpunkt
134:          */
135:         startpunkte[wandIndex] = new Punkt(x, y);
136:
137:         /*
138:          * Erzeuge Endpunkt
139:          */
140:         endpunkte[wandIndex] = startpunkte[wandIndex].addiere(RICHTUNG[richtung]);
141:
142:         /*
```

```
143:          * Füge Startpunkt zum Array istBelegt hinzu
144:          */
145:         istBelegt[x][y] = true;
146:
147:         /*
148:          * Füge Endpunkt zum Array istBelegt
149:          */
150:         istBelegt[x + RICHTUNG[richtung].gibX()][y + RICHTUNG[richtung].gibY()] =
true;
151:
152:         /*
153:          * Erhöhe Anzahl der Wandelemente
154:          */
155:         wandIndex++;
156:
157:     }
158:
159:     /**
160:      * Generiert ein neues Labyrinth, innerhalb der Außenwände
161:      */
162:     public void erzeugeLabyrinth() {
163:
164:         /*
165:          * Erzeuge Zufallsgenerator
166:          */
167:         Random random = new Random();
168:
169:         while (wandIndex < laenge) {
170:
171:             /*
172:              * Wähle beliebigen Punkt aus
173:              */
174:             aktuellerPunkt = endpunkte[(int) (Math.random() * wandIndex)];
175:
176:             /*
177:              * Wähle beliebige RIchtung
178:              */
179:             int richtung = random.nextInt(4);
180:
181:             /*
182:              * Setze nächste Koordinate
183:              */
184:             int naechstesX = aktuellerPunkt.addiere(RICHTUNG[richtung]).gibX();
185:             int naechstesY = aktuellerPunkt.addiere(RICHTUNG[richtung]).gibY();
186:
187:             /*
188:              * Prüfe auf möglichen Punkt
189:              */
190:             if (pruefeAusserhalb(naechstesX, naechstesY)) {
```

```
191:
192:                 /*
193:                  * Erzeuge neue Wand
194:                  */
195:                 erzeugeWand(aktuellerPunkt.gibX(), aktuellerPunkt.gibY(),
richtung);
196:
197:             }
198:
199:         }
200:
201:     }
202:
203:     /**
204:      * Prüft, ob Zielpunkt ein möglicher Punkt ist
205:      *
206:      * @param naechstesX x-Koordinate des Zielpunkts
207:      * @param naechstesY y-Koordinate des Zielpunkts
208:      * @return Rückgabe, ob Zielpunkt ein möglicher Punkt ist
209:      */
210:     public boolean pruefeAusserhalb(int naechstesX, int naechstesY) {
211:
212:         /*
213:          * Rückgabewert, ob Punkt außerhalb möglicher Punkte liegt
214:          */
215:         boolean istAusserhalb = true;
216:
217:         /*
218:          * Prüfung, ob Punkt noch innerhalb des Feldes liegt
219:          */
220:         if (naechstesX < 0 || naechstesX > breite
221:                 || naechstesY < 0 || naechstesY > hoehe) {
222:
223:             istAusserhalb = false;
224:
225:         } else {
226:
227:             /*
228:              * Prüfung, ob Punkt schon belegt ist
229:              */
230:             istAusserhalb = !istBelegt[naechstesX][naechstesY];
231:
232:         }
233:
234:         return istAusserhalb;
235:
236:     }
237:
238:     /**
```

```
239:      * Liefert die Anzahl aller Wandelemente des Labyrinths
240:      *
241:      * @return Rückgabe der Anzahl an Wandelementen
242:      */
243:     public int gibAnzahlWandelemente() {
244:
245:         return wandIndex;
246:
247:     }
248:
249:     /**
250:      * Gibt die Höhe des Labyrinths zurück
251:      *
252:      * @return Rückgabe der Höhe des Labyrinths
253:      */
254:     public int gibHoehe() {
255:
256:         return this.hoehe;
257:
258:     }
259:
260:     /**
261:      * Gibt die Breite des Labyrinths zurück
262:      *
263:      * @return Rückgabe der Breite des Labyrinths
264:      */
265:     public int gibBreite() {
266:
267:         return this.breite;
268:
269:     }
270:
271:     /**
272:      * Gibt den Startpunkt eines Wandelements zurück
273:      *
274:      * @param anfang Gewählter Startpunkt
275:      * @return Rückgabe des Startpunkts
276:      */
277:     public Punkt gibStartpunkt(int anfang) {
278:
279:         return startpunkte[anfang];
280:
281:     }
282:
283:     /**
284:      * Gibt den Endpunkt eines Wandelements zurück
285:      *
286:      * @param ende Gewählter Endpunkt
287:      * @return Rückgabe des Endpunkts
```

```
288:      */
289:     public Punkt gibEndpunkt(int ende) {
290: 
291:         return endpunkte[ende];
292: 
293:     }
294: }
```

5.2. Labyrinthdarstellung.java

```
  1: package labyrinth;
  2: 
  3: import java.awt.Color;
  4: import java.awt.Dimension;
  5: import java.awt.Graphics;
  6: import javax.swing.JPanel;
  7: import javax.swing.border.EmptyBorder;
  8: import util.Punkt;
  9: 
 10: /**
 11:  * Ein Objekt dieser Klasse stellt ein Labyrinth2 grafisch dar.
 12:  */
 13: public class Labyrinthdarstellung extends JPanel {
 14: 
 15:     /**
 16:      * Innenabstand der Darstellung vom Fensterrand.
 17:      */
 18:     private static final int INNENABSTAND = 2;
 19: 
 20:     /**
 21:      * Pixel pro Rastereinheit des Labyrinths.
 22:      */
 23:     private int rastermass;
 24: 
 25:     /**
 26:      * Labyrinth2, das dargestellt wird.
 27:      */
 28:     private Labyrinth labyrinth;
 29: 
 30:     /**
 31:      * Erzeugt ein Objekt dieser Klasse zur Darstellung des
 32:      * übergebenen Labyrinths.
 33:      */
 34:     public Labyrinthdarstellung(Labyrinth labyrinth, int rastermass) {
 35: 
 36:         this.labyrinth = labyrinth;
 37:         this.rastermass = rastermass;
 38: 
 39:         this.setBorder(new EmptyBorder(INNENABSTAND, INNENABSTAND,
 40:                                        INNENABSTAND, INNENABSTAND));
```

```
41:     }
42:
43:     /**
44:      * Liefert bevorzugte Größe dieser Darstellung.
45:      *
46:      * @return bevorzugte Größe basierend auf Rastermaß und
47:      *         Größe des Labyrinths
48:      */
49:     @Override
50:     public Dimension getPreferredSize() {
51:
52:         return (labyrinth == null)
53:                ? new Dimension(0, 0)
54:                : new Dimension(rastermass * labyrinth.gibBreite()
55:                                + getInsets().left + getInsets().right,
56:                                rastermass * labyrinth.gibHoehe()
57:                                + getInsets().top + getInsets().bottom);
58:     }
59:
60:     /**
61:      * Zeichnet das Labyrinth2 in die angegebene Grafikumgebung.
62:      *
63:      * @param graphics  Grafikumgebung, in die gezeichnet wird
64:      */
65:     @Override
66:     public void paintComponent(Graphics graphics) {
67:
68:         super.paintComponent(graphics);
69:
70:         graphics.setColor(Color.BLACK);
71:
72:         for (int i = 0; i < labyrinth.gibAnzahlWandelemente(); i++) {
73:             Punkt startpunkt = labyrinth.gibStartpunkt(i);
74:             Punkt zielpunkt = labyrinth.gibEndpunkt(i);
75:             graphics.drawLine(
76:                     getInsets().left + rastermass * startpunkt.gibX(),
77:                     getInsets().top + rastermass * startpunkt.gibY(),
78:                     getInsets().left + rastermass * zielpunkt.gibX(),
79:                     getInsets().top + rastermass * zielpunkt.gibY());
80:         }
81:     }
82: }
```

5.3. UILabyrinth.java

```
1: package labyrinth;
2:
3: import java.awt.Container;
4: import java.awt.GridLayout;
5: import javax.swing.JFrame;
```

```
 6:
 7: /**
 8:  * Ein Objekt dieser Klasse dient der Darstellung eines Labyrinths.
 9:  */
10: public class UILabyrinth extends JFrame {
11: //20x15
12:     /**
13:      * Breite des Labyrinths.
14:      */
15:     private static final int BREITE = 300;
16:
17:     /**
18:      * Höhe des Labyrinths.
19:      */
20:     private static final int HOEHE = 200;
21:
22:     /**
23:      * Pixel pro Rastereinheit des Labyrinths.
24:      */
25:     private static final int PIXEL_PRO_RASTER = 2;
26:
27:     /**
28:      * Komponente zur Darstellung des Labyrinths.
29:      */
30:     private Labyrinthdarstellung darstellung;
31:
32:     /**
33:      * Erzeugt die Oberfläche zur Darstellung des übergebenen Labyrinths.
34:      *
35:      * @param labyrinth  Labyrinth2, das dargestellt wird
36:      * @param rastermass  Anzahl Pixel pro Rastereinheit
37:      */
38:     public UILabyrinth(Labyrinth labyrinth, int rastermass) {
39:
40:         super("Labyrinth");
41:
42:         /* Erzeugt die Komponenten dieses Frame.
43:          */
44:         erzeugeKomponenten(labyrinth, rastermass);
45:
46:         /* Anwendung beim Schließen dieses Frame beenden.
47:          */
48:         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49:     }
50:
51:     /**
52:      * Erzeugt die Komponenten dieses Frame.
53:      */
54:     private void erzeugeKomponenten(Labyrinth labyrinth, int rastermass) {
```

```
55:
56:         Container container = this.getContentPane();
57:         container.setLayout(new GridLayout(1, 1));
58:
59:         /*
60:          * Objekt zur Darstellung des Labyrinths erzeugen und dem
61:          * Container diesen Frame hinzufügen.
62:          */
63:         darstellung = new Labyrinthdarstellung(labyrinth, rastermass);
64:         container.add(darstellung);
65:     }
66:
67:     /**
68:      * Start der Anwendung.
69:      *
70:      * @param args  wird nicht verwendet
71:      */
72:     public static void main(String[] args) {
73:
74:         UILabyrinth fenster =
75:                 new UILabyrinth(new Labyrinth(BREITE, HOEHE),
76:                                                 PIXEL_PRO_RASTER);
77:         fenster.pack();
78:         fenster.setResizable(true);
79:         fenster.setVisible(true);
80:     }
81: }
```

5.4. Wandelement.java

```
 1: package labyrinth;
 2: import util.Punkt;
 3:
 4: public class Wandelement {
 5:
 6:     private Punkt start;
 7:     private Punkt ende;
 8:     private Wandelement[] wandelemente;
 9:
10: }
```