Benjamin Schürmann

Auswertung vom 31.05.2012

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 Textfinder	
✔						Klasse TextfinderTest	

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
✔					Textfinder	public Textfinder(InputStream, int)	
✔					Textfinder	public int gibHaeufigkeit(String)	
✔					Textfinder	public Set gibWoerter()	
				✔	Textfinder	private boolean istValide(int)	
				✔	TextfinderTest	public void setUp()	
				✔	TextfinderTest	public void testGibHaeufigkeit()	
				✔	TextfinderTest	public void testGibWoerter()	

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.

Benjamin Schürmann

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.

3.1. Test der Klasse Textfinder

3.1.1. Testsituation „A“

Die Testsituation wird in folgenden Schritten aufgebaut:

1.	Erzeuge Textfinder für Bytefolge {-20, 65} und Mindestlänge 1.

Tests der Methode gibHaeufigkeit(String)

	Testfall	Fehlerhinweis
✔	Prüfe, wie oft "A" gefunden wurde.	

Tests der Methode gibWoerter()

	Testfall	Fehlerhinweis
✔	Prüfe Menge der gefundenen Wörter.	

3.1.2. Testsituation „B“

Die Testsituation wird in folgenden Schritten aufgebaut:

1.	Erzeuge Textfinder für Bytefolge {0, 60, -4, 66} und Mindestlänge 1.

Tests der Methode gibHaeufigkeit(String)

	Testfall	Fehlerhinweis
✔	Prüfe, wie oft "B" gefunden wurde.	

Tests der Methode gibWoerter()

	Testfall	Fehlerhinweis
✔	Prüfe Menge der gefundenen Wörter.	

3.1.3. Testsituation „Berg“

Die Testsituation wird in folgenden Schritten aufgebaut:

1.	Erzeuge Textfinder für Bytefolge {'b', 'e', 'r', 'g', '-', '-', 'B', 'e', 'r', 'g', '2', '-', '-', 'b', 'e', 'r', 'g', '-', '-', 'B', 'e', 'r', 'g', '-', '-', '2', 'B', 'e', 'r', 'g', '-', '-', 'B', 'e', 'r', 'g', '-', '-', 'B', 'e', 'r', '-', '-', 'B', 'e', 'r', 'g', '1'} und Mindestlänge 4.

Benjamin Schürmann

Tests der Methode gibHaeufigkeit(String)

	Testfall	**Fehlerhinweis**	
✗	Prüfe, wie oft "Berg" gefunden wurde.	Soll	3
		Ist	2
✔	Prüfe, wie oft "Berg1" gefunden wurde.		
✔	Prüfe, wie oft "Berg2" gefunden wurde.		
✔	Prüfe, wie oft "berg" gefunden wurde.		

Tests der Methode gibWoerter()

	Testfall	**Fehlerhinweis**	
✗	Prüfe Menge der gefundenen Wörter.	Soll	[berg, Berg1, Berg2, Berg]
		Ist	[Berg1, berg, Berg, Berg2, 2Berg]

4. Checkstyle-Prüfung

```
Starting audit...
Textfinder.java:17:5: Javadoc-Kommentar fehlt.
Textfinder.java:25:35: Erwartete Tag @param für 'data'.
Textfinder.java:25:45: Erwartete Tag @param für 'laenge'.
Textfinder.java:61: Fehlender @return-Tag.
Textfinder.java:70: Fehlender @return-Tag.
Textfinder.java:70:38: Erwartete Tag @param für 'wort'.
Textfinder.java:86: Fehlender @return-Tag.
Textfinder.java:86:35: Erwartete Tag @param für 'symbol'.
TextfinderTest.java:9:8: Nicht benutztes import - java.util.Set.
TextfinderTest.java:10: Die Form '.*' für import-Anweisungen sollte vermieden werden -
org.junit.*.
TextfinderTest.java:11: Die Form '.*' für import-Anweisungen sollte vermieden werden -
org.junit.Assert.*.
TextfinderTest.java:19:5: Javadoc-Kommentar fehlt.
TextfinderTest.java:20:5: Javadoc-Kommentar fehlt.
TextfinderTest.java:21:5: Javadoc-Kommentar fehlt.
TextfinderTest.java:22:5: Javadoc-Kommentar fehlt.
TextfinderTest.java:23:5: Javadoc-Kommentar fehlt.
Audit done.
```

5. Quellcode der Java-Klassen

5.1. Textfinder.java

```
 1:
 2: import java.io.IOException;
 3: import java.io.InputStream;
 4: import java.util.HashMap;
 5: import java.util.Set;
 6:
 7: /*
 8:  * To change this template, choose Tools | Templates and open the template in
 9:  * the editor.
10:  */
11: /**
12:  *
13:  * @author apex
14:  */
15: public class Textfinder {
16:
17:     private HashMap<String, Integer> woerter;
18:
19:     /**
20:      *
21:      * @param data
22:      * @param laenge
23:      * @throws IOException
24:      */
25:     public Textfinder(InputStream data, int laenge) throws IOException {
26:
27:         this.woerter = new HashMap<String, Integer>();
28:
29:         int symbol = data.read();
30:
31:         while (symbol != -1) {
32:
33:             StringBuilder wort = new StringBuilder();
34:
35:             while (istValide(symbol)) {
36:
37:                 wort.append((char) symbol);
38:                 symbol = data.read();
39:
40:             }
41:
42:             if (wort.length() >= laenge) {
43:                 if (!woerter.containsKey(wort.toString())) {
44:                     woerter.put(wort.toString(), 1);
```

```
45:                 } else {
46:                     this.woerter.put(wort.toString(),
47:                             this.woerter.get(wort.toString()).intValue() + 1);
48:                 }
49:             }
50:
51:             symbol = data.read();
52:
53:         }
54:
55:     }
56:
57:     /**
58:      *
59:      * @return
60:      */
61:     public Set gibWoerter() {
62:         return this.woerter.keySet();
63:     }
64:
65:     /**
66:      *
67:      * @param wort
68:      * @return
69:      */
70:     public int gibHaeufigkeit(String wort) {
71:         int ergebnis = 0;
72:
73:         if (woerter.containsKey(wort)) {
74:             ergebnis = woerter.get(wort);
75:         }
76:
77:         return ergebnis;
78:
79:     }
80:
81:     /**
82:      *
83:      * @param symbol
84:      * @return
85:      */
86:     private boolean istValide(int symbol) {
87:         boolean ergebnis = false;
88:
89:         if ((symbol >= 'a' && symbol <= 'z')
90:                 || (symbol >= 'A' && symbol <= 'Z')
91:                 || (symbol >= '0' && symbol <= '9')) {
92:             ergebnis = true;
93:         }
```

```
94:
95:         return ergebnis;
96:     }
97: }
```

5.2. TextfinderTest.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5:
 6: import java.io.ByteArrayInputStream;
 7: import java.io.IOException;
 8: import java.util.HashMap;
 9: import java.util.Set;
10: import org.junit.*;
11: import static org.junit.Assert.*;
12:
13: /**
14:  *
15:  * @author apex
16:  */
17: public class TextfinderTest {
18:
19:     private String s;
20:     private Textfinder tf3;
21:     private Textfinder tf4;
22:     private Textfinder tf5;
23:     private HashMap<String, Integer> woerter;
24:
25:     /**
26:      *
27:      * @throws IOException
28:      */
29:     @Before
30:     public void setUp() throws IOException {
31:         s = "H>-p>Chd>(9&i@90P6=X,(cb0E(eh9#Chd";
32:         tf3 = new Textfinder(new ByteArrayInputStream(s.getBytes()), 3);
33:         tf4 = new Textfinder(new ByteArrayInputStream(s.getBytes()), 4);
34:         tf5 = new Textfinder(new ByteArrayInputStream(s.getBytes()), 5);
35:     }
36:
37:     /**
38:      * Test of gibWoerter method, of class Textfinder.
39:      */
40:     @Test
41:     public void testGibWoerter() {
42:         woerter = new HashMap<String, Integer>();
43:         woerter.put("90P6", 1);
```

```
44:         woerter.put("Chd", 2);
45:         woerter.put("cb0E", 1);
46:         woerter.put("eh9", 1);
47:         assertEquals(woerter.keySet(), tf3.gibWoerter());
48:
49:         woerter = new HashMap<String, Integer>();
50:         woerter.put("90P6", 1);
51:         woerter.put("cb0E", 1);
52:         assertEquals(woerter.keySet(), tf4.gibWoerter());
53:
54:         woerter = new HashMap<String, Integer>();
55:         assertEquals(woerter.keySet(), tf5.gibWoerter());
56:     }
57:
58:     /**
59:      * Test of gibHaeufigkeit method, of class Textfinder.
60:      */
61:     @Test
62:     public void testGibHaeufigkeit() {
63:         woerter = new HashMap<String, Integer>();
64:         woerter.put("90P6", 1);
65:         woerter.put("Chd", 2);
66:         woerter.put("cb0E", 1);
67:         woerter.put("eh9", 1);
68:         assertEquals(2, tf3.gibHaeufigkeit("Chd"));
69:         assertEquals(0, tf3.gibHaeufigkeit("hf2z"));
70:         assertEquals(1, tf3.gibHaeufigkeit("cb0E"));
71:
72:         woerter = new HashMap<String, Integer>();
73:         woerter.put("90P6", 1);
74:         woerter.put("cb0E", 1);
75:         assertEquals(1, tf4.gibHaeufigkeit("90P6"));
76:         assertEquals(1, tf4.gibHaeufigkeit("cb0E"));
77:         assertEquals(0, tf4.gibHaeufigkeit(".,mnadfk4kjb4"));
78:
79:         woerter = new HashMap<String, Integer>();
80:         assertEquals(0, tf5.gibHaeufigkeit("hf2z"));
81:         assertEquals(0, tf5.gibHaeufigkeit("cb0E"));
82:
83:     }
84: }
```