Benjamin Schürmann

Auswertung vom 20.06.2012

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.

```
Note: C:\Temp\TestBench\opr8\submissions\19566\src\Indexierer.java uses unchecked or
unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

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 textverarbeitung.Textverarbeiter	
	✗					Klasse textverarbeitung.Indexierer	
	✗					Schnittstelle textverarbeitung.Wortverarbeiter	

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

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 Indexierer

✗	Die Klasse textverarbeitung/Indexierer ist nicht vorhanden oder konnte nicht fehlerfrei übersetzt werden. Deshalb war die Durchführung des Tests nicht möglich.

3.2. Test der Klasse Textverarbeiter

✗	Die Klasse textverarbeitung/Textverarbeiter ist nicht vorhanden oder konnte nicht fehlerfrei übersetzt werden. Deshalb war die Durchführung des Tests nicht möglich.

Benjamin Schürmann

4. Checkstyle-Prüfung

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

5. Quellcode der Java-Klassen

5.1. Wortverarbeiter.java

```
 1: public interface Wortverarbeiter {
 2:     public void verarbeite(String eingabeWort);
 3:     public void verarbeiteZeilenende();
 4: }
```

5.2. Textverarbeiter.java

```
 1: import java.io.Reader;
 2: import java.io.BufferedReader;
 3: import java.io.IOException;
 4: import java.util.StringTokenizer;
 5:
 6: public class Textverarbeiter {
 7:     public static final String TRENNZEICHEN = " .,:;!?-()";
 8:     private Wortverarbeiter wortverarbeiter;
 9:
10:     public Textverarbeiter(Wortverarbeiter eingabeWortverarbeiter) {
11:         wortverarbeiter = eingabeWortverarbeiter;
12:     }
13:
14:     void verarbeite(Reader eingabeDatenquelle) throws IOException {
15:         BufferedReader zeilenLeser = new BufferedReader(eingabeDatenquelle);
16:         String ausgeleseneZeile = zeilenLeser.readLine();
17:         StringTokenizer tokenizer;
18:
19:         while (ausgeleseneZeile != null) {
20:             tokenizer = new StringTokenizer(ausgeleseneZeile,
21:                 Textverarbeiter.TRENNZEICHEN);
22:             while (tokenizer.hasMoreTokens()) {
23:                 this.wortverarbeiter.verarbeite(tokenizer.nextToken());
24:             }
25:
26:             ausgeleseneZeile = zeilenLeser.readLine();
27:             this.wortverarbeiter.verarbeiteZeilenende();
28:         }
29:     }
30: }
```

5.3. Indexierer.java

```
 1: import java.util.ArrayList;
 2: import java.util.Collection;
 3: import static java.util.Collections.sort;
 4: import java.util.HashMap;
 5: import java.util.HashSet;
 6: import java.util.Iterator;
```

```
 7: import java.util.List;
 8:
 9: public class Indexierer implements Wortverarbeiter {
10:     private HashMap<String, HashSet<Integer>> gefundeneWoerter;
11:     private HashSet<String> ausschlussWoerter;
12:     private int zeilenIndex;
13:
14:     public Indexierer(Collection ausschlussWoerter) {
15:         this.gefundeneWoerter = new HashMap<String, HashSet<Integer>>();
16:         this.ausschlussWoerter = new HashSet<String>();
17:         this.zeilenIndex = 1;
18:
19:         Iterator<String> iterator = ausschlussWoerter.iterator();
20:         while (iterator.hasNext()) {
21:             this.ausschlussWoerter.add(iterator.next());
22:         }
23:     }
24:
25:     public List gibWoerter() {
26:         ArrayList<String> rueckgabeListe =
27:                 new ArrayList<String>(this.gefundeneWoerter.keySet());
28:         sort(rueckgabeListe);
29:         return rueckgabeListe;
30:     }
31:
32:     public String gibZeilennummern(String gesuchtesWort) {
33:         String rueckgabeZeichenkette = "";
34:         //voher pruefen ob tatsaechlich das gesuchte Wort enthalten ist
35:         if (this.gefundeneWoerter.containsKey(gesuchtesWort)) {
36:             HashSet<Integer> zeilenListe =
37:                     this.gefundeneWoerter.get(gesuchtesWort);
38:             Iterator<Integer> iterator = zeilenListe.iterator();
39:             while (iterator.hasNext()) {
40:                 rueckgabeZeichenkette += "" + iterator.next();
41:                 if (iterator.hasNext()) {
42:                     rueckgabeZeichenkette += ", ";
43:                 }
44:             }
45:
46:         }
47:
48:         return rueckgabeZeichenkette;
49:     }
50:
51:     public void verarbeite(String eingabeWort) {
52:         if (!this.ausschlussWoerter.contains(eingabeWort)) {
53:             if (this.gefundeneWoerter.containsKey(eingabeWort)) {
54:                 this.gefundeneWoerter.get(eingabeWort).add(this.zeilenIndex);
55:             } else {
```

```
56:                 HashSet zeilenListe = new HashSet<Integer>();
57:                 zeilenListe.add(this.zeilenIndex);
58:                 this.gefundeneWoerter.put(eingabeWort, zeilenListe);
59:             }
60:         }
61:     }
62:
63:     public void verarbeiteZeilenende() {
64:         this.zeilenIndex++;
65:     }
66:
67: }
```

5.4. IndexiererTest.java

```
 1: import java.util.ArrayList;
 2: import java.util.HashSet;
 3: import org.junit.Test;
 4: import static org.junit.Assert.*;
 5:
 6: public class IndexiererTest {
 7:
 8:     @Test
 9:     public void testGibWoerter() {
10:         Indexierer instanz;
11:         ArrayList<String> sollErgebnis;
12:         HashSet<String> ausschlussWoerter;
13:
14:         //Test1
15:         ausschlussWoerter = new HashSet<String>();
16:         ausschlussWoerter.add("groß");
17:         ausschlussWoerter.add("sehr");
18:         ausschlussWoerter.add("und");
19:         ausschlussWoerter.add("wunderschoen");
20:
21:         instanz = new Indexierer(ausschlussWoerter);
22:
23:         sollErgebnis = new ArrayList<String>();
24:         sollErgebnis.add("Alle");
25:         sollErgebnis.add("Heim");
26:         sollErgebnis.add("bauen");
27:         sollErgebnis.add("das");
28:         sollErgebnis.add("eigene");
29:
30:
31:         instanz.verarbeite("Alle");
32:         instanz.verarbeite("bauen");
33:         instanz.verarbeite("das");
34:         instanz.verarbeite("eigene");
35:         instanz.verarbeite("Heim");
```

```
36:         instanz.verarbeite("sehr");
37:         instanz.verarbeite("groß");
38:         instanz.verarbeite("und");
39:         instanz.verarbeite("wunderschoen");
40:
41:         assertEquals(sollErgebnis, instanz.gibWoerter());
42:     }
43:
44:     @Test
45:     public void testGibZeilennummern() {
46:         Indexierer instanz;
47:         HashSet<String> ausschlussWoerter;
48:
49:         //Test2
50:         ausschlussWoerter = new HashSet<String>();
51:         ausschlussWoerter.add("Ja");
52:         ausschlussWoerter.add("das");
53:         ausschlussWoerter.add("ist");
54:         ausschlussWoerter.add("Gut");
55:
56:         instanz = new Indexierer(ausschlussWoerter);
57:
58:
59:         instanz.verarbeite("Mama");
60:         instanz.verarbeite("sucht");
61:         instanz.verarbeite("Papa");
62:         instanz.verarbeite("Papa");
63:         instanz.verarbeiteZeilenende();
64:         instanz.verarbeite("sucht");
65:         instanz.verarbeite("Mama");
66:         instanz.verarbeite("Ja");
67:         instanz.verarbeite("das");
68:         instanz.verarbeite("ist");
69:         instanz.verarbeite("gut");
70:
71:         assertEquals("1, 2", instanz.gibZeilennummern("Mama"));
72:         assertEquals("1", instanz.gibZeilennummern("Papa"));
73:         assertEquals("1, 2", instanz.gibZeilennummern("sucht"));
74:
75:         assertEquals("", instanz.gibZeilennummern("Ja"));
76:         assertEquals("", instanz.gibZeilennummern("das"));
77:         assertEquals("", instanz.gibZeilennummern("ist"));
78:         assertEquals("2", instanz.gibZeilennummern("gut"));
79:     }
80: }
```

5.5. TextverarbeiterTest.java

```
1: import java.io.StringReader;
2: import java.util.ArrayList;
```

```
 3: import org.junit.Test;
 4: import static org.junit.Assert.*;
 5:
 6: class MockupWortverarbeiter implements Wortverarbeiter {
 7:     public ArrayList<String> verarbeiteteWoerter;
 8:     public int zeilenIndex;
 9:
10:     public MockupWortverarbeiter() {
11:         this.zeilenIndex = 1;
12:         this.verarbeiteteWoerter = new ArrayList<String>();
13:     }
14:
15:     public void verarbeite(String eingabeWort) {
16:         this.verarbeiteteWoerter.add(eingabeWort);
17:     }
18:
19:     public void verarbeiteZeilenende() {
20:         this.zeilenIndex++;
21:     }
22: }
23:
24: public class TextverarbeiterTest {
25:     @Test
26:     public void testVerarbeite() throws Exception {
27:         Textverarbeiter instanz;
28:         MockupWortverarbeiter mockupWortverarbeiter;
29:         Indexierer indexierer;
30:         StringReader datenQuelle;
31:         ArrayList<String> sollWortListe;
32:         ArrayList<String> ausschlussWortliste;
33:
34:         //Test1
35:         sollWortListe = new ArrayList<String>();
36:         sollWortListe.add("Wir");
37:         sollWortListe.add("sind");
38:         sollWortListe.add("toll");
39:         sollWortListe.add("Und");
40:         sollWortListe.add("Fahren");
41:         sollWortListe.add("gerne");
42:         sollWortListe.add("NachHause");
43:         sollWortListe.add("Toll");
44:         sollWortListe.add("wa");
45:
46:         mockupWortverarbeiter = new MockupWortverarbeiter();
47:
48:         instanz = new Textverarbeiter(mockupWortverarbeiter);
49:         datenQuelle = new StringReader(
50:                 "Wir sind toll.Und   Fahren gerne-NachHause!Toll wa?");
51:         instanz.verarbeite(datenQuelle);
```

```
 52:
 53:         assertEquals(2, mockupWortverarbeiter.zeilenIndex);
 54:         assertEquals(sollWortListe, mockupWortverarbeiter.verarbeiteteWoerter);
 55:
 56:         //Test2
 57:         sollWortListe = new ArrayList<String>();
 58:         sollWortListe.add("Wir");
 59:         sollWortListe.add("sind");
 60:         sollWortListe.add("toll");
 61:         sollWortListe.add("Und");
 62:         sollWortListe.add("Fahren");
 63:         sollWortListe.add("gerne");
 64:         sollWortListe.add("Nach");
 65:         sollWortListe.add("Hause");
 66:         sollWortListe.add("Toll");
 67:         sollWortListe.add("wa");
 68:
 69:         mockupWortverarbeiter = new MockupWortverarbeiter();
 70:
 71:         instanz = new Textverarbeiter(mockupWortverarbeiter);
 72:         datenQuelle = new StringReader(
 73:                 "Wir\n\n\n sind\n toll!?-;..Und\n\n\n   Fahren gerne-
Nach\nHause\n!\nToll wa?\n\n");
 74:         instanz.verarbeite(datenQuelle);
 75:
 76:         assertEquals(13, mockupWortverarbeiter.zeilenIndex);
 77:         assertEquals(sollWortListe, mockupWortverarbeiter.verarbeiteteWoerter);
 78:
 79:         //Test3 (Verbundstest um zu prüfen ob die Komponenten auch zusammen
arbeiten
 80:         sollWortListe = new ArrayList<String>();
 81:         sollWortListe.add("Wir");
 82:         sollWortListe.add("sind");
 83:         sollWortListe.add("Und");
 84:         sollWortListe.add("gerne");
 85:         sollWortListe.add("Nach");
 86:         sollWortListe.add("Hause");
 87:         sollWortListe.add("toll");
 88:         sollWortListe.add("wa");
 89:         java.util.Collections.sort(sollWortListe);
 90:
 91:         ausschlussWortliste = new ArrayList<String>();
 92:         ausschlussWortliste.add("Fahren");
 93:
 94:         indexierer = new Indexierer(ausschlussWortliste);
 95:
 96:         instanz = new Textverarbeiter(indexierer);
 97:         datenQuelle = new StringReader(
 98:                 "Wir\n\n\n sind\n toll!?-;..Und\n\n\n   Fahren gerne-
Nach\nHause\n!\ntoll wa?\n\n");
```

```
 99:         instanz.verarbeite(datenQuelle);
100:
101:         assertEquals(sollWortListe, indexierer.gibWoerter());
102:         assertEquals("1", indexierer.gibZeilennummern("Wir"));
103:         assertEquals("9", indexierer.gibZeilennummern("Hause"));
104:         assertEquals("5, 11", indexierer.gibZeilennummern("toll"));
105:     }
106: }
```