Benjamin Schürmann

Auswertung vom 02.05.2012

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 Ausdruck	
✔						Klasse Konstante	
✔						Klasse Variable	
✔						Klasse OperatorAusdruck	
✔						Klasse Variablenbelegung	
✔						Klasse Parser	

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
				✔	Ausdruck	public Ausdruck()	
✔					Ausdruck	public abstract int gibWert(Variablenbelegung)	
✔					Konstante	public Konstante(int)	
✔					Konstante	public boolean equals(Object)	
✔					Konstante	public int gibWert(Variablenbelegung)	
✔					Variable	public Variable(String)	
✔					Variable	public boolean equals(Object)	
✔					Variable	public int gibWert(Variablenbelegung)	
✔					OperatorAusdruck	public OperatorAusdruck(Ausdruck, char, Ausdruck)	
✔					OperatorAusdruck	public boolean equals(Object)	
✔					OperatorAusdruck	public int gibWert(Variablenbelegung)	

Benjamin Schürmann

✔					Variablenbelegung	public Variablenbelegung()	
✔					Variablenbelegung	public int gibWert(String)	
✔					Variablenbelegung	public void belege(String, int)	
✔					Parser	public Parser()	
✔					Parser	public Ausdruck parse(String)	

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 Ausdruck

3.1.1. Testsituation „Ohne Aufbau von Testdaten“

Die folgenden Tests werden ohne Aufbau von Testdaten ausgeführt.

Tests der Methode equals(Object)

	Testfall	**Fehlerhinweis**	
✗	Prüfe, ob zwei erzeugte Konstanten mit Wert 0 gleich sind.	Soll	true
		Ist	false
✔	Prüfe, ob Konstante mit Wert 0 gleich String "" ist.		
✗	Prüfe, ob zwei erzeugte Variablen mit Namen "ab" gleich sind.	Soll	true
		Ist	false
✔	Prüfe, ob Variable mit Namen "ab" gleich Variablen mit Namen "cd" ist.		
✔	Prüfe, ob Variable mit Namen "ab" gleich String "" ist.		
✗	Prüfe, ob zwei Operatorausdrücke für (a + 1) gleich sind.	Soll	true
		Ist	false
✔	Prüfe, ob Operatorausdruck für (a + 1) gleich Operatorausdruck für (1 + a) ist.		
✔	Prüfe, ob Operatorausdruck für (a + 1) gleich String "" ist.		

4. Checkstyle-Prüfung

```
Starting audit...
Ausdruck.java:15:5: Javadoc-Kommentar fehlt.
AusdruckTest.java:11:5: Javadoc-Kommentar fehlt.
AusdruckTest.java:11:14: Variable 'sollAusdruck' muss private sein.
AusdruckTest.java:12:5: Javadoc-Kommentar fehlt.
AusdruckTest.java:12:23: Variable 'belegung' muss private sein.
AusdruckTest.java:14:5: Javadoc-Kommentar fehlt.
AusdruckTest.java:47: Zeile länger als 80 Zeichen
AusdruckTest.java:82: Kind von operator new bei Einrücktiefe 16 nicht an korrekter
Einrücktiefe 20
Konstante.java:14:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:15: Zeile länger als 80 Zeichen
OperatorAusdruck.java:15:5: Javadoc-Kommentar fehlt.
Parser.java:16:5: Javadoc-Kommentar fehlt.
ParserTest.java:11:5: Javadoc-Kommentar fehlt.
ParserTest.java:11:14: Variable 'sollAusdruck' muss private sein.
ParserTest.java:12:5: Javadoc-Kommentar fehlt.
ParserTest.java:12:12: Variable 'parser' muss private sein.
ParserTest.java:14:5: Javadoc-Kommentar fehlt.
ParserTest.java:37: Zeile länger als 80 Zeichen
ParserTest.java:41: Zeile länger als 80 Zeichen
ParserTest.java:67: Kind von operator new bei Einrücktiefe 16 nicht an korrekter
Einrücktiefe 20
Variable.java:14:5: Javadoc-Kommentar fehlt.
Variablenbelegung.java:19:5: Javadoc-Kommentar fehlt.
Variablenbelegung.java:22:5: Javadoc-Kommentar fehlt.
Variablenbelegung.java:25:5: Javadoc-Kommentar fehlt.
Audit done.
```

5. Quellcode der Java-Klassen

5.1. Ausdruck.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5:
 6: /**
 7:  * Enthält abstrakte Instanzmethode int gibWert(Variablenbelegung), die den Wert
 8:  * dieses Ausdrucks basierend auf der Variablenbelegung liefert.
 9:  *
10:  * @author apex
11:  */
12: public abstract class Ausdruck {
13:
14:      //NOTE: skript s.96
15:     public abstract int gibWert(Variablenbelegung belegung);
16: }
```

5.2. Konstante.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5:
 6: /**
 7:  * Enthält Konstruktor Konstante(int), durch den ein konstanter Ausdruck mit dem
 8:  * angegebenen Wert erzeugt wird.
 9:  *
10:  * @author apex
11:  */
12: public class Konstante extends Ausdruck {
13:
14:     public Konstante(int wert) {
15:     }
16:
17:     @Override
18:     public int gibWert(Variablenbelegung belegung) {
19:         return 0;
20:     }
21: }
```

5.3. Variable.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
```

```
 4:  */
 5:
 6: /**
 7:  * Enthält Konstruktor Variable(String), durch den eine Variable mit dem
 8:  * angegebenen Namen erzeugt wird.
 9:  *
10:  * @author apex
11:  */
12: public class Variable extends Ausdruck {
13:
14:     public Variable(String titel) {
15:     }
16:
17:     @Override
18:     public int gibWert(Variablenbelegung belegung) {
19:         return 0;
20:     }
21: }
```

5.4. OperatorAusdruck.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5:
 6: /**
 7:  * Enthält Konstruktor OperatorAusdruck(Ausdruck, char, Ausdruck), durch den ein
 8:  * arithmetischer Ausdruck mit den angegebenen Teilausdrücken und dem
 9:  * Operatorsymbol erzeugt wird.
10:  *
11:  * @author apex
12:  */
13: public class OperatorAusdruck extends Ausdruck {
14:
15:     public OperatorAusdruck(Ausdruck erstAusdruck, char operator, Ausdruck
zweitAusdruck) {
16:     }
17:
18:     @Override
19:     public int gibWert(Variablenbelegung belegung) {
20:         return 0;
21:     }
22: }
```

5.5. Variablenbelegung.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
```

```
 5:
 6: /**
 7:  * Enthält Konstruktor Variablenbelegung(), durch den eine Variablenbelegung
 8:  * erzeugt wird, in der zunächst keiner Variablen ein Wert zugeordnet ist.
 9:  * Enthält Instanzmethode void belege(String, int), durch die einer Variablen
10:  * (1.Parameter) ein Wert (2. Parameter) zugeordnet wird. Ein evtl. vorhandener
11:  * alter Wert wird dabei überschrieben. Enthält Instanzmethode int
12:  * gibWert(String), die den Wert liefert, der der angegebenen Variable
13:  * zugeordnet ist.
14:  *
15:  * @author apex
16:  */
17: public class Variablenbelegung {
18:
19:     public Variablenbelegung() {
20:     }
21:
22:     public void belege(String var, int wert) {
23:     }
24:
25:     public int gibWert(String var) {
26:         return 0;
27:     }
28: }
```

5.6. Parser.java

```
 1: /*
 2:  * To change this template, choose Tools | Templates
 3:  * and open the template in the editor.
 4:  */
 5:
 6: /**
 7:  * Enthält Instanzmethode Ausdruck parse(String), die die Textdarstellung
 8:  * (gewöhnliche geklammerte Infixdarstellung eines Ausdrucks parst und ein
 9:  * entsprechendes Ausdruck- Objekt zurückgibt. Beliebig viele Leerzeichen
10:  * zwischen den Komponenten des Ausdrucks sind zulässig.
11:  *
12:  * @author apex
13:  */
14: public class Parser {
15:
16:     public Ausdruck parse(String ausdruck) {
17:         throw new UnsupportedOperationException("Not yet implemented");
18:     }
19:
20: }
```

5.7. AusdruckTest.java

```
 1:
```

```
 2: import org.junit.Assert;
 3: import org.junit.Test;
 4:
 5: /**
 6:  *
 7:  * @author apex
 8:  */
 9: public class AusdruckTest {
10:
11:     Ausdruck sollAusdruck;
12:     Variablenbelegung belegung;
13:
14:     @Test
15:     public void testGibWert() {
16:
17:         belegung = new Variablenbelegung();
18:
19:         belegung.belege("a", 5);
20:
21:         // (a + 1) * 5
22:         sollAusdruck = new OperatorAusdruck(
23:                 new OperatorAusdruck(new Variable("a"), '+', new Konstante(1)),
24:                 '*',
25:                 new Konstante(5));
26:
27:         Assert.assertEquals(30, sollAusdruck.gibWert(belegung));
28:
29:         belegung.belege("b", 7);
30:
31:         // (b * 13) + 5
32:         sollAusdruck = new OperatorAusdruck(
33:                 new OperatorAusdruck(new Variable("b"), '*', new Konstante(13)),
34:                 '+',
35:                 new Konstante(5));
36:
37:         Assert.assertEquals(96, sollAusdruck.gibWert(belegung));
38:
39:         belegung.belege("a", 59);
40:         belegung.belege("b", 53);
41:
42:         // ((a + 1) / (b - 23)) + 42
43:         sollAusdruck = new OperatorAusdruck(
44:                 new OperatorAusdruck(
45:                 new OperatorAusdruck(new Variable("a"), '+', new Konstante(1)),
46:                 '/',
47:                 new OperatorAusdruck(new Variable("b"), '-', new Konstante(23))),
48:                 '+',
49:                 new Konstante(42));
50:
```

```
51:         Assert.assertEquals(44, sollAusdruck.gibWert(belegung));
52:
53:         belegung.belege("a", 5);
54:         belegung.belege("b", 4);
55:
56:         // 205 * ((a - 1) + b)
57:         sollAusdruck = new OperatorAusdruck(
58:                 new Konstante(205),
59:                 '*',
60:                 new OperatorAusdruck(
61:                 new OperatorAusdruck(new Variable("a"), '-', new Konstante(1)),
62:                 '+',
63:                 new Variable("b")));
64:
65:         Assert.assertEquals(1640, sollAusdruck.gibWert(belegung));
66:
67:         // (2300 / 23) * 567
68:         sollAusdruck = new OperatorAusdruck(
69:                 new OperatorAusdruck(
70:                 new Konstante(2300), '/', new Konstante(23)),
71:                 '*',
72:                 new Konstante(567));
73:
74:         Assert.assertEquals(56700, sollAusdruck.gibWert(belegung));
75:
76:         belegung.belege("a", 5);
77:
78:         // ((a * 1) + 6) / 12
79:         sollAusdruck = new OperatorAusdruck(
80:                 new OperatorAusdruck(
81:                 new OperatorAusdruck(
82:                 new Variable("a"), '*', new Konstante(1)),
83:                 '+',
84:                 new Konstante(5)),
85:                 '/',
86:                 new Konstante(5));
87:
88:         Assert.assertEquals(1, sollAusdruck.gibWert(belegung));
89:     }
90: }
```

5.8. ParserTest.java

```
1:
2: import org.junit.Assert;
3: import org.junit.Test;
4:
5: /**
6:  *
7:  * @author apex
```

```
 8:  */
 9: public class ParserTest {
10:
11:     Ausdruck sollAusdruck;
12:     Parser parser = new Parser();
13:
14:     @Test
15:     public void testParse() {
16:         // (a + 1) * 5
17:         sollAusdruck = new OperatorAusdruck(
18:                 new OperatorAusdruck(new Variable("a"), '+', new Konstante(1)),
19:                 '*',
20:                 new Konstante(5));
21:
22:         Assert.assertEquals(sollAusdruck, parser.parse("(a + 1) * 5"));
23:
24:         // (b * 13) + 5
25:         sollAusdruck = new OperatorAusdruck(
26:                 new OperatorAusdruck(new Variable("b"), '*', new Konstante(13)),
27:                 '+',
28:                 new Konstante(5));
29:
30:         Assert.assertEquals(sollAusdruck, parser.parse("(b * 13) + 5"));
31:
32:         // ((a + 1) / (b - 23)) + 42
33:         sollAusdruck = new OperatorAusdruck(
34:                 new OperatorAusdruck(
35:                 new OperatorAusdruck(new Variable("a"), '+', new Konstante(1)),
36:                 '/',
37:                 new OperatorAusdruck(new Variable("b"), '-', new Konstante(23))),
38:                 '+',
39:                 new Konstante(42));
40:
41:         Assert.assertEquals(sollAusdruck, parser.parse("((a + 1) / (b - 23)) +
42"));
42:
43:         // 205 * ((a - 1) + b)
44:         sollAusdruck = new OperatorAusdruck(
45:                 new Konstante(205),
46:                 '*',
47:                 new OperatorAusdruck(
48:                 new OperatorAusdruck(new Variable("a"), '-', new Konstante(1)),
49:                 '+',
50:                 new Variable("b")));
51:
52:         Assert.assertEquals(sollAusdruck, parser.parse("205 * ((a - 1) + b)"));
53:
54:         // (2300 / 23) * 567
55:         sollAusdruck = new OperatorAusdruck(
```

```
56:                 new OperatorAusdruck(
57:                 new Konstante(2300), '/', new Konstante(23)),
58:                 '*',
59:                 new Konstante(567));
60:
61:         Assert.assertEquals(sollAusdruck, parser.parse("(2300 / 23) * 567"));
62:
63:         // ((a * 1) + 5) / 12
64:         sollAusdruck = new OperatorAusdruck(
65:                 new OperatorAusdruck(
66:                 new OperatorAusdruck(
67:                 new Variable("a"), '*', new Konstante(1)),
68:                 '+',
69:                 new Konstante(5)),
70:                 '/',
71:                 new Konstante(5));
72:
73:         Assert.assertEquals(sollAusdruck, parser.parse("((a * 1) + 5) / 12"));
74:     }
75: }
```