Benjamin Schürmann

Auswertung vom 18.05.2012

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)	
				✔	Ausdruck	public String toString()	
✔					Konstante	public Konstante(int)	
✔					Konstante	public boolean equals(Object)	
✔					Konstante	public int gibWert(Variablenbelegung)	
				✔	Konstante	public String toString()	
				✔	Konstante	public int hashCode()	
✔					Variable	public Variable(String)	
✔					Variable	public boolean equals(Object)	
✔					Variable	public int gibWert(Variablenbelegung)	
				✔	Variable	public String toString()	
				✔	Variable	public int hashCode()	

Benjamin Schürmann

✔					OperatorAusdruck	public OperatorAusdruck(Ausdruck, char, Ausdruck)	
✔					OperatorAusdruck	public boolean equals(Object)	
✔					OperatorAusdruck	public int gibWert(Variablenbelegung)	
				✔	OperatorAusdruck	public void berechneErgebnis(Variablenbelegung)	
				✔	OperatorAusdruck	public String toString()	
				✔	OperatorAusdruck	public int hashCode()	
✔					Variablenbelegung	public Variablenbelegung()	
✔					Variablenbelegung	public int gibWert(String)	
✔					Variablenbelegung	public void belege(String, int)	
✔					Parser	public Parser()	
✔					Parser	public Ausdruck parse(String)	
				✔	Parser	private Ausdruck parseSummand()	
				✔	Parser	private Ausdruck parseAusdruck()	
				✔	Parser	private Ausdruck parseFaktor()	

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.

Beim Test sind keine Fehler aufgetreten.

3.1. Test der Klasse Ausdruck

3.1.1. Testsituation „Belegung“

Die Testsituation wird in folgenden Schritten aufgebaut:

1.	Erzeuge Variablenbelegung.
2.	Belege Variable i mit 3.
3.	Belege Variable k mit 4.
4.	Belege Variable j mit 9.

Tests der Methode gibWert(Variablenbelegung)

	Testfall	Fehlerhinweis
✔	Werte Ausdruck "10" aus.	
✔	Werte Ausdruck "k" aus.	
✔	Werte Ausdruck "10 + j" aus.	
✔	Werte Ausdruck "(10 + j) / k" aus.	
✔	Werte Ausdruck "1 + 2 * (i + 2 * k - 1) / 2 + j" aus.	

3.2. Test der Klasse Parser

3.2.1. Testsituation „Parser“

Die Testsituation wird in folgenden Schritten aufgebaut:

1.	Erzeuge Parser.

Tests der Methode parse(String)

	Testfall	Fehlerhinweis
✔	Parse Ausdruck "" und prüfe Meldung der ParseException.	
✔	Parse Ausdruck "2 * (s - t * 5" und prüfe Meldung der ParseException.	
✔	Parse Ausdruck "2 * (3 * (i + 4)" und prüfe Meldung der ParseException.	

Benjamin Schürmann

✔	Parse Ausdruck "( )" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "(+)" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "((10) 10)" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "1a" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "a 1" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "(a b * 2)" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "(1+2 a)?" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "2 -)" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "2 * (s - t * a_b)" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "2 * (s -)" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "2 * (s - t) (" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "2 * (s - t))" und prüfe Offset der ParseException.	
✔	Parse Ausdruck "(a b))" und prüfe Offset der ParseException.	
✔	Parse korrekten Ausdruck "10". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck "((10))". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck "betrag". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck "((betrag))". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck " 10 * betrag ". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck "(((10))) * ((betrag))". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck "10*betrag/anzahl". Es wird keine ParseException erwartet.	
✔	Parse korrekten Ausdruck "1 + 2 * (i + 2 * k - 1) / 2 + j". Es wird keine ParseException erwartet.	

4. Checkstyle-Prüfung

```
Starting audit...
Ausdruck.java:20:5: 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.
Konstante.java:13:5: Javadoc-Kommentar fehlt.
Konstante.java:15:5: 4 return-Anweisungen (Obergrenze ist 1).
Konstante.java:42:5: Javadoc-Kommentar fehlt.
Konstante.java:42:5: Konstruktordefinition in falscher Reihenfolge.
OperatorAusdruck.java:15:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:16:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:25:5: 7 return-Anweisungen (Obergrenze ist 1).
OperatorAusdruck.java:37: Zeile länger als 80 Zeichen
OperatorAusdruck.java:43: Zeile länger als 80 Zeichen
OperatorAusdruck.java:53: Zeile länger als 80 Zeichen
OperatorAusdruck.java:55: Zeile länger als 80 Zeichen
OperatorAusdruck.java:58:5: Instanzvariablendefinition in falscher Reihenfolge.
OperatorAusdruck.java:58:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:59:5: Instanzvariablendefinition in falscher Reihenfolge.
OperatorAusdruck.java:59:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:62: Zeile länger als 80 Zeichen
OperatorAusdruck.java:62:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:62:5: Konstruktordefinition in falscher Reihenfolge.
OperatorAusdruck.java:68:5: Javadoc-Kommentar fehlt.
OperatorAusdruck.java:70:9: switch ohne "default".
OperatorAusdruck.java:72: Zeile länger als 80 Zeichen
OperatorAusdruck.java:75: Zeile länger als 80 Zeichen
OperatorAusdruck.java:78: Zeile länger als 80 Zeichen
OperatorAusdruck.java:81: Zeile länger als 80 Zeichen
Parser.java:35: assign bei Einrücktiefe 12 nicht an korrekter Einrücktiefe 8
Parser.java:108: Kind von while bei Einrücktiefe 16 nicht an korrekter Einrücktiefe 12
Parser.java:140: Kind von while bei Einrücktiefe 16 nicht an korrekter Einrücktiefe 12
Parser.java:160:11: Nach 'if' folgt kein Leerzeichen.
Parser.java:181: Zeile länger als 80 Zeichen
ParserTest.java:32: Kind von method def bei Einrücktiefe 9 nicht an korrekter
Einrücktiefe 8
ParserTest.java:167: Kind von method call bei Einrücktiefe 8 nicht an korrekter
Einrücktiefe 12
ParserTest.java:167: Kind von try bei Einrücktiefe 8 nicht an korrekter Einrücktiefe 12
Variable.java:19:5: 4 return-Anweisungen (Obergrenze ist 1).
Variable.java:43:5: Instanzvariablendefinition in falscher Reihenfolge.
Variable.java:43:5: Javadoc-Kommentar fehlt.
Variable.java:45:5: Javadoc-Kommentar fehlt.
Variable.java:45:5: Konstruktordefinition in falscher Reihenfolge.
```

```
Variable.java:46: Kind von ctor def bei Einrücktiefe 9 nicht an korrekter Einrücktiefe
8
Variablenbelegung.java:22:5: Javadoc-Kommentar fehlt.
Variablenbelegung.java:22:37: Variable 'variablen' muss private sein.
Variablenbelegung.java:24:5: Javadoc-Kommentar fehlt.
Variablenbelegung.java:25:32: Vor '<' befindet sich ein Leerzeichen.
Variablenbelegung.java:28:5: Javadoc-Kommentar fehlt.
Variablenbelegung.java:32:5: 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:     @Override
15:     public String toString() {
16:         return "Ausdruck{" + '}';
17:     }
18: 
19:      //NOTE: skript s.96
20:     public abstract int gibWert(Variablenbelegung belegung);
21: }
```

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:     private int wert;
14: 
15:     @Override
16:     public boolean equals(Object obj) {
17:         if (obj == null) {
18:             return false;
19:         }
20:         if (getClass() != obj.getClass()) {
21:             return false;
```

```
22:         }
23:         final Konstante other = (Konstante) obj;
24:         if (this.wert != other.wert) {
25:             return false;
26:         }
27:         return true;
28:     }
29:
30:     @Override
31:     public int hashCode() {
32:         int hash = 5;
33:         hash = 43 * hash + this.wert;
34:         return hash;
35:     }
36:
37:     @Override
38:     public String toString() {
39:         return "Konstante{" + "wert=" + wert + '}';
40:     }
41:
42:     public Konstante(int wert) {
43:         this.wert = wert;
44:     }
45:
46:     @Override
47:     public int gibWert(Variablenbelegung belegung) {
48:         return wert;
49:     }
50: }
```

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:     @Override
15:     public String toString() {
16:         return "Variable{" + "titel=" + titel + '}';
17:     }
18:
```

```
19:     @Override
20:     public boolean equals(Object obj) {
21:         if (obj == null) {
22:             return false;
23:         }
24:         if (getClass() != obj.getClass()) {
25:             return false;
26:         }
27:         final Variable other = (Variable) obj;
28:         if ((this.titel == null)
29:                 ? (other.titel != null)
30:                 : !this.titel.equals(other.titel)) {
31:             return false;
32:         }
33:         return true;
34:     }
35:
36:     @Override
37:     public int hashCode() {
38:         int hash = 3;
39:         hash = 83 * hash + (this.titel != null ? this.titel.hashCode() : 0);
40:         return hash;
41:     }
42:
43:     private String titel;
44:
45:     public Variable(String titel) {
46:          this.titel = titel;
47:     }
48:
49:     @Override
50:     public int gibWert(Variablenbelegung belegung) {
51:         return belegung.gibWert(this.titel);
52:     }
53:
54:
55: }
```

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:     private int ergebnis;
 16:     private Ausdruck erstAusdruck;
 17:
 18:     @Override
 19:     public String toString() {
 20:         return "OperatorAusdruck{" + "ergebnis=" + ergebnis + ", erstAusdruck="
 21:                 + erstAusdruck + ", operator=" + operator + ", zweitAusdruck="
 22:                 + zweitAusdruck + '}';
 23:     }
 24:
 25:     @Override
 26:     public boolean equals(Object obj) {
 27:         if (obj == null) {
 28:             return false;
 29:         }
 30:         if (getClass() != obj.getClass()) {
 31:             return false;
 32:         }
 33:         final OperatorAusdruck other = (OperatorAusdruck) obj;
 34:         if (this.ergebnis != other.ergebnis) {
 35:             return false;
 36:         }
 37:         if (this.erstAusdruck != other.erstAusdruck && (this.erstAusdruck == null
|| !this.erstAusdruck.equals(other.erstAusdruck))) {
 38:             return false;
 39:         }
 40:         if (this.operator != other.operator) {
 41:             return false;
 42:         }
 43:         if (this.zweitAusdruck != other.zweitAusdruck && (this.zweitAusdruck ==
null || !this.zweitAusdruck.equals(other.zweitAusdruck))) {
 44:             return false;
 45:         }
 46:         return true;
 47:     }
 48:
 49:     @Override
 50:     public int hashCode() {
 51:         int hash = 7;
 52:         hash = 71 * hash + this.ergebnis;
 53:         hash = 71 * hash + (this.erstAusdruck != null ?
this.erstAusdruck.hashCode() : 0);
 54:         hash = 71 * hash + this.operator;
 55:         hash = 71 * hash + (this.zweitAusdruck != null ?
this.zweitAusdruck.hashCode() : 0);
```

```
 56:         return hash;
 57:     }
 58:     private char operator;
 59:     private Ausdruck zweitAusdruck;
 60:
 61:
 62:     public OperatorAusdruck(Ausdruck erstAusdruck, char operator, Ausdruck
zweitAusdruck) {
 63:         this.erstAusdruck = erstAusdruck;
 64:         this.zweitAusdruck = zweitAusdruck;
 65:         this.operator = operator;
 66:     }
 67:
 68:     public void berechneErgebnis(Variablenbelegung belegung) {
 69:
 70:         switch (this.operator) {
 71:             case '+':
 72:                 this.ergebnis = this.erstAusdruck.gibWert(belegung) +
this.zweitAusdruck.gibWert(belegung);
 73:                 break;
 74:             case '-':
 75:                 this.ergebnis = this.erstAusdruck.gibWert(belegung) -
this.zweitAusdruck.gibWert(belegung);
 76:                 break;
 77:             case '*':
 78:                 this.ergebnis = this.erstAusdruck.gibWert(belegung) *
this.zweitAusdruck.gibWert(belegung);
 79:                 break;
 80:             case '/':
 81:                 this.ergebnis = this.erstAusdruck.gibWert(belegung) /
this.zweitAusdruck.gibWert(belegung);
 82:                 break;
 83:         }
 84:
 85:
 86:     }
 87:
 88:     @Override
 89:     public int gibWert(Variablenbelegung belegung) {
 90:         berechneErgebnis(belegung);
 91:         return this.ergebnis;
 92:     }
 93: }
```

5.5. Variablenbelegung.java

```
  1:
  2: import java.util.HashMap;
  3:
  4: /*
```

```
 5:  * To change this template, choose Tools | Templates
 6:  * and open the template in the editor.
 7:  */
 8:
 9: /**
10:  * Enthält Konstruktor Variablenbelegung(), durch den eine Variablenbelegung
11:  * erzeugt wird, in der zunächst keiner Variablen ein Wert zugeordnet ist.
12:  * Enthält Instanzmethode void belege(String, int), durch die einer Variablen
13:  * (1.Parameter) ein Wert (2. Parameter) zugeordnet wird. Ein evtl. vorhandener
14:  * alter Wert wird dabei überschrieben. Enthält Instanzmethode int
15:  * gibWert(String), die den Wert liefert, der der angegebenen Variable
16:  * zugeordnet ist.
17:  *
18:  * @author apex
19:  */
20: public class Variablenbelegung {
21:
22:     public HashMap<String, Integer> variablen;
23:
24:     public Variablenbelegung() {
25:         variablen = new HashMap <String, Integer>();
26:     }
27:
28:     public void belege(String var, int wert) {
29:         this.variablen.put(var, wert);
30:     }
31:
32:     public int gibWert(String var) {
33:         return this.variablen.get(var);
34:     }
35: }
```

5.6. Parser.java

```
 1: import java.text.ParseException;
 2: import java.util.LinkedList;
 3: import java.util.List;
 4: import java.util.StringTokenizer;
 5:
 6: /**
 7:  * Parser zum parsen von Termausdruecken.
 8:  */
 9: public class Parser {
10:
11:     /** Konstante fuer Rechenzeichen + .*/
12:     private static final char PLUS = '+';
13:     /** Konstante fuer Rechenzeichen - .*/
14:     private static final char MINUS = '-';
15:     /** Konstante fuer Rechenzeichen * .*/
16:     private static final char MAL = '*';
```

```
17:     /** Konstante fuer Rechenzeichen / .*/
18:     private static final char GETEILT = '/';
19:     /** Regex Ausdruck fuer eine Konstante. */
20:     private static final String KONSTANTE = "[0-9]+";
21:     /** Regex Ausdruck fuer eine Variable. */
22:     private static final String VARIABLE = "[a-zA-Z][a-zA-Z0-9]*";
23:     /** Konstante fuer ( .*/
24:     private static final char KLAMMER_AUF = '(';
25:     /** Konstante fuer ) .*/
26:     private static final char KLAMMER_ZU = ')';
27:     /** Trennzeichen fuer das Zerlegen des Terms. */
28:     private static final String TRENNZEICHEN = "+-*/ ()";
29:     /** Konstante fuer ein Leerzeichen. */
30:     private static final char LEERZEICHEN = ' ';
31:     /** Fehlermeldung fuer unerwartetes Ende. */
32:     private static final String MELDUNG_UNERWARTETES_ENDE = "unerwartetes Ende";
33:     /** Fehlermeldung fuer ungueltiges Token. */
34:     private static final String MELDUNG_UNGUELTIGES_TOKEN
35:             = "ungueltiges Token ";
36:     /** Liste in der alle Tokens gespeichert werden. */
37:     private List<String> tokens;
38:     /** Die Anzahl der Toekens die eingelesen wurden. */
39:     private int tokenAnzahl;
40:
41:
42:     /**
43:      * Erzeugt ein Parser Objekt.
44:      */
45:     public Parser() {
46:     }
47:
48:     /**
49:      * Parst den uerbergeben Term.
50:      * @param term Term als String
51:      * @return erzeugter Operatorausdruck
52:      * @throws ParseException Fehler beim Parsen
53:      */
54:     public Ausdruck parse(String term) throws ParseException {
55:         Ausdruck geparsterAusdruck;
56:         /* Dafuer sorgen das die Liste auf jeden Fall leer ist, also wird sie
57:          * erst hier initialsiert und nicht im Konstruktor */
58:         this.tokens = new LinkedList<String>();
59:         StringTokenizer tokenizer = new StringTokenizer(term, TRENNZEICHEN,
60:                                                         true);
61:
62:         while (tokenizer.hasMoreTokens()) {
63:             String element = tokenizer.nextToken();
64:             if (element.charAt(0) != LEERZEICHEN) {
65:                 this.tokens.add(element);
```

```
 66:             }
 67:         }
 68:         this.tokenAnzahl = this.tokens.size();
 69:
 70: //        if (this.tokenAnzahl == 0) {
 71: //            throw new ParseException(MELDUNG_UNERWARTETES_ENDE, 0);
 72: //        }
 73:
 74:         geparsterAusdruck = parseAusdruck();
 75:
 76:
 77:         /* Der Parser lief ohne Probleme durch, jedoch sind noch Tokens
 78:          * vorhanden, dies ist dann jedoch ein Token, welches nicht folgen darf
 79:          */
 80:         if (!this.tokens.isEmpty()) {
 81:             throw new ParseException(MELDUNG_UNGUELTIGES_TOKEN
 82:                                      + this.tokens.get(0),
 83:                                    this.tokenAnzahl - this.tokens.size() + 1);
 84:         }
 85:
 86:         return geparsterAusdruck;
 87:     }
 88:
 89:
 90:     /**
 91:      * Abschnitt des Parsers, der einen Ausdruck parst.
 92:      * @return gibt einen OperatorAusdruck zurueck.
 93:      * @throws ParseException Fehler beim Parsen.
 94:      */
 95:     private Ausdruck parseAusdruck() throws ParseException {
 96:         Ausdruck ausdruckLinks;
 97:         char operator;
 98:         Ausdruck ausdruckRechts;
 99:
100:         ausdruckLinks = parseSummand();
101:
102:         while (!this.tokens.isEmpty() && (this.tokens.get(0).charAt(0) == PLUS
103:                                       || this.tokens.get(0).charAt(0)
104:                                             == MINUS)) {
105:             operator = this.tokens.remove(0).charAt(0);
106:
107: //            if (!this.tokens.isEmpty()) {
108:                 ausdruckRechts = parseSummand();
109: //            } else {
110: //                throw new ParseException(MELDUNG_UNERWARTETES_ENDE, 0);
111: //            }
112:
113:             ausdruckLinks = new OperatorAusdruck(ausdruckLinks, operator,
114:                                                  ausdruckRechts);
```

```
115:
116:         }
117:
118:         return ausdruckLinks;
119:     }
120:
121:
122:     /**
123:      * Abschnitt des Parsers, der einen Summanden parst.
124:      * @return gibt einen OperatorAusdruck zurueck.
125:      * @throws ParseException Fehler beim Parsen.
126:      */
127:     private Ausdruck parseSummand() throws ParseException {
128:         Ausdruck ausdruckLinks;
129:         char operator;
130:         Ausdruck ausdruckRechts;
131:
132:         ausdruckLinks = parseFaktor();
133:
134:         while (!this.tokens.isEmpty() && (this.tokens.get(0).charAt(0) == MAL
135:                                       || this.tokens.get(0).charAt(0)
136:                                             == GETEILT)) {
137:             operator = this.tokens.remove(0).charAt(0);
138:
139: //            if (!this.tokens.isEmpty()) {
140:                 ausdruckRechts = parseFaktor();
141: //            } else {
142: //                throw new ParseException(MELDUNG_UNERWARTETES_ENDE, 0);
143: //            }
144:
145:             ausdruckLinks = new OperatorAusdruck(ausdruckLinks, operator,
146:                                                  ausdruckRechts);
147:         }
148:
149:         return ausdruckLinks;
150:     }
151:
152:     /**
153:      * Abschnitt des Parsers, der einen Faktor parst.
154:      * @return gibt einen einzelnen Ausdruck zurueck.
155:      * @throws ParseException Fehler beim Parsen.
156:      */
157:     private Ausdruck parseFaktor() throws ParseException {
158:         Ausdruck rueckgabe;
159:
160:         if(!this.tokens.isEmpty()) {
161:             String aktuellesToken = this.tokens.remove(0);
162:             if (aktuellesToken.matches(KONSTANTE)) {
163:                 rueckgabe = new Konstante(Integer.parseInt(aktuellesToken));
```

```
164:             } else if (aktuellesToken.matches(VARIABLE)) {
165:                 rueckgabe = new Variable(aktuellesToken);
166:             } else if (aktuellesToken.charAt(0) == KLAMMER_AUF) {
167:                 rueckgabe = parseAusdruck();
168:                 if (this.tokens.isEmpty()) {
169:                     throw new ParseException(MELDUNG_UNERWARTETES_ENDE, 0);
170:                 } else if (this.tokens.get(0).charAt(0) != KLAMMER_ZU) {
171:                     throw new ParseException(MELDUNG_UNGUELTIGES_TOKEN
172:                                                 + this.tokens.get(0),
173:                                             this.tokenAnzahl - this.tokens.size()
174:                                                 + 1);
175:                 } else {
176:                     /* Es war eine Klammer, also entfernen */
177:                     this.tokens.remove(0);
178:                 }
179:             } else {
180:                 throw new ParseException(MELDUNG_UNGUELTIGES_TOKEN
181:                         + aktuellesToken, this.tokenAnzahl - this.tokens.size());
182:             }
183:         } else {
184:             throw new ParseException(MELDUNG_UNERWARTETES_ENDE, 0);
185:         }
186:         return rueckgabe;
187:     }
188: 
189: }
```

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"), '-',
48:                         new Konstante(23))),
49:                 '+',
50:                 new Konstante(42));
51:
52:         Assert.assertEquals(44, sollAusdruck.gibWert(belegung));
53:
54:         belegung.belege("a", 5);
55:         belegung.belege("b", 4);
56:
57:         // 205 * ((a - 1) + b)
58:         sollAusdruck = new OperatorAusdruck(
59:                 new Konstante(205),
60:                 '*',
61:                 new OperatorAusdruck(
62:                 new OperatorAusdruck(new Variable("a"), '-', new Konstante(1)),
63:                 '+',
64:                 new Variable("b")));
65:
66:         Assert.assertEquals(1640, sollAusdruck.gibWert(belegung));
67:
68:         // (2300 / 23) * 567
69:         sollAusdruck = new OperatorAusdruck(
70:                 new OperatorAusdruck(
```

```
 71:                 new Konstante(2300), '/', new Konstante(23)),
 72:                 '*',
 73:                 new Konstante(567));
 74:
 75:         Assert.assertEquals(56700, sollAusdruck.gibWert(belegung));
 76:
 77:         belegung.belege("a", 6);
 78:
 79:         // ((a * 1) + 6) / 12
 80:         sollAusdruck = new OperatorAusdruck(
 81:                 new OperatorAusdruck(
 82:                     new OperatorAusdruck(
 83:                     new Variable("a"), '*', new Konstante(1)),
 84:                 '+',
 85:                 new Konstante(6)),
 86:                 '/',
 87:                 new Konstante(12));
 88:
 89:         Assert.assertEquals(1, sollAusdruck.gibWert(belegung));
 90:     }
 91: }
```

5.8. ParserTest.java

```
  1: import java.text.ParseException;
  2: import junit.framework.TestCase;
  3:
  4: /**
  5:  * Test Klasse zum Testen der Parser Klasse.
  6:  */
  7: public class ParserTest extends TestCase {
  8:     /** Parser fuer dir Testautrufe. */
  9:     private Parser parser;
 10:
 11:     /**
 12:      * Erzeuge neue Instanz der Test Klasse.
 13:      * @param name Name der Klasse.
 14:      */
 15:     public ParserTest(String name) {
 16:         super(name);
 17:     }
 18:
 19:     /**
 20:      * startet die Test Klasse.
 21:      * @param args nicht benutzt.
 22:      */
 23:     public static void main(String[] args) {
 24:         junit.textui.TestRunner.run(ParserTest.class);
 25:     }
 26:
```

```
27:     /**
28:      * Initialisierung der Testklasse.
29:      */
30:     @Override
31:     public void setUp() {
32:          this.parser = new Parser();
33:     }
34:
35:     /**
36:      * Testet die Methode parse aus der Klasse Parser fuer die Faelle das ein
37:      * ungueltiges Token vorhanden ist.
38:      */
39:     public void testParseUngueltigesToken() {
40:         try {
41:             parser.parse("( )");
42:             fail("( )");
43:         } catch (ParseException ex) {
44:             assertEquals("ungueltiges Token )", ex.getMessage());
45:             assertEquals(2, ex.getErrorOffset());
46:         }
47:
48:         try {
49:             parser.parse("( + )");
50:             fail("( + )");
51:         } catch (ParseException ex) {
52:             assertEquals("ungueltiges Token +", ex.getMessage());
53:             assertEquals(2, ex.getErrorOffset());
54:         }
55:
56:
57:         try {
58:             parser.parse("(( 10) 10)");
59:             fail("(( 10) 10)");
60:         } catch (ParseException ex) {
61:             assertEquals("ungueltiges Token 10", ex.getMessage());
62:             assertEquals(5, ex.getErrorOffset());
63:         }
64:
65:         try {
66:             parser.parse("1a");
67:             fail("1a");
68:         } catch (ParseException ex) {
69:             assertEquals("ungueltiges Token 1a", ex.getMessage());
70:             assertEquals(1, ex.getErrorOffset());
71:         }
72:
73:         try {
74:             parser.parse("a 1");
75:             fail("a 1");
```

```
 76:         } catch (ParseException ex) {
 77:             assertEquals("ungueltiges Token 1", ex.getMessage());
 78:             assertEquals(2, ex.getErrorOffset());
 79:         }
 80:
 81:         try {
 82:             parser.parse("2 - )");
 83:             fail("2 - )");
 84:         } catch (ParseException ex) {
 85:             assertEquals("ungueltiges Token )", ex.getMessage());
 86:             assertEquals(3, ex.getErrorOffset());
 87:         }
 88:
 89:         try {
 90:             parser.parse("2 * (s - t * a_b)");
 91:             fail("2 * (s - t * a_b)");
 92:         } catch (ParseException ex) {
 93:             assertEquals("ungueltiges Token a_b", ex.getMessage());
 94:             assertEquals(8, ex.getErrorOffset());
 95:         }
 96:
 97:         try {
 98:             parser.parse("2 * (s - )");
 99:             fail("2 * (s - )");
100:         } catch (ParseException ex) {
101:             assertEquals("ungueltiges Token )", ex.getMessage());
102:             assertEquals(6, ex.getErrorOffset());
103:         }
104:
105:         try {
106:             parser.parse("2 * (s - t) (");
107:             fail("2 * (s - t) (");
108:         } catch (ParseException ex) {
109:             assertEquals("ungueltiges Token (", ex.getMessage());
110:             assertEquals(8, ex.getErrorOffset());
111:         }
112:
113:     }
114:
115:     /**
116:     * Testet die Methode parse aus der Klasse Parser fuer die Faelle das ein
117:     * unerwartetes Ende auftritt.
118:     */
119:     public void testParseUnerwartetesEnde() {
120:         try {
121:             this.parser.parse("");
122:             fail("");
123:         } catch (ParseException ex) {
124:             assertEquals("unerwartetes Ende", ex.getMessage());
```

```
125:         }
126:
127:         try {
128:             this.parser.parse("2 * (s - t * 5");
129:             fail("2 * (s - t * 5");
130:         } catch (ParseException ex) {
131:             assertEquals("unerwartetes Ende", ex.getMessage());
132:         }
133:
134:         try {
135:             this.parser.parse("2 * (3 * (i + 4)");
136:             fail("");
137:         } catch (ParseException ex) {
138:             assertEquals("unerwartetes Ende", ex.getMessage());
139:         }
140:     }
141:
142:
143:     /**
144:     * Testet die Methode parse aus der Klasse Parser fuer die Faelle das ein
145:     * gueltiger Ausdruck vorliegt.
146:     */
147:     public void testParseGueltig() {
148:         try {
149:             assertEquals(new OperatorAusdruck(
150:                             new Variable("a"), '+', new Konstante(1)),
151:                         this.parser.parse("a + 1"));
152:         } catch (ParseException ex) {
153:             fail(ex.getMessage());
154:         }
155:
156:         try {
157:             assertEquals(new OperatorAusdruck(
158:                             new OperatorAusdruck(
159:                                 new Variable("ab"), '+', new Variable("b")),
160:                                     '/' , new Konstante(2)),
161:                         this.parser.parse("(ab   +    b) / 2"));
162:         } catch (ParseException ex) {
163:             fail(ex.getMessage());
164:         }
165:
166:         try {
167:         assertEquals(new OperatorAusdruck(new OperatorAusdruck(
168:                                             new Konstante(1), '+',
169:                                             new Variable("a")),
170:                                         '*', new OperatorAusdruck(
171:                                                 new Konstante(2), '-',
172:                                                 new Variable("b"))),
173:                     this.parser.parse("(1   +a)*   (2-   b)"));
```

```
174:         } catch (ParseException ex) {
175:             fail(ex.getMessage());
176:         }
177:     }
178: }
```