

**Programmierung**  
**Prof. Dr.-Ing. Nobert Fuhr**

Gudrun Fischer  
Sascha Kriewel  
programmierung@is.informatik.uni-duisburg.de

**Übungsblatt Nr. 4**

**Aufgabe 8: Vereinfachte arithmetische Ausdrücke – Schleifen, Rekursion und eine Baumstruktur**

Ein mathematischer Ausdruck setzt sich aus einem Operator (z.B.  $*$  für Multiplikation) und einem oder mehreren Operanden zusammen. Auf diesem Übungsblatt soll eine Datenstruktur für arithmetische Ausdrücke aus natürlichen Zahlen und dem Operator  $*$  entwickelt werden.

Betrachte dazu zunächst die Eigenschaften eines solchen arithmetischen Ausdrucks:

- Eine natürliche Zahl für sich genommen ist schon ein Ausdruck. Beispiel: 42
- Zwei Ausdrücke, die mit  $*$  verbunden sind, bilden wiederum einen Ausdruck. Beispiel:  $6 * 7$

Betrachtet man Ausdrücke nun als Objekte einer zu definierenden Java-Klasse `Ausdruck`, dann wären dies Eigenschaften der Klasse:

- ein Basisblock, der entweder eine natürliche Zahl oder ein Operator sein kann
- zwei Operanden, die wiederum Ausdrücke sind, aber auch leer sein können

Wenn der Basisblock eine natürliche Zahl ist, so sind keine Operanden erlaubt. Ist der Basisblock ein Operator, dann hängt von der Art des Operators ab, wieviele Operanden vorhanden sein müssen (z.B. genau zwei für den Operator  $*$ ).

Die Datei `Ausdruck.java` auf der Webseite enthält ein Gerüst für die folgenden Aufgaben. In der Methode `main` sind für jede Teilaufgabe Testaufrufe vorgesehen, aber noch auskommentiert. Diese kannst du zum Testen deiner Lösungen verwenden. Entwirf zusätzlich auch eigene Testaufrufe.

### a) Definition der Variablen für die Klasse Ausdruck

Erstelle eine Klasse `Ausdruck` und definiere für sie drei Instanzvariablen:

- eine Variable `basisBlock` vom Typ `String`
- eine Variable `linkerOperand` vom Typ `Ausdruck`
- eine Variable `rechterOperand` vom Typ `Ausdruck`

Warum sollen die beiden Variablen `linkerOperand` und `rechterOperand` den Datentyp `Ausdruck` haben?

### b) Definition passender Konstruktoren

Erstelle zwei Konstruktoren für die Klasse `Ausdruck`.

`Ausdruck(String zahl)`

Das soll dem Fall entsprechen, dass der zu repräsentierende Ausdruck aus einer einzelnen natürlichen Zahl besteht.

`Ausdruck(String t, Ausdruck lOperand, Ausdruck rOperand)`

Das soll einem zusammengesetzten Ausdruck entsprechen, bei dem der Operator als Text abgespeichert wird, und die Operanden wiederum Ausdrücke sind.

Achte darauf, in beiden Konstruktoren die Instanzvariablen `basisBlock`, `linkerOperand` und `rechterOperand` passend zu setzen. Was ist ein sinnvoller Wert für `linkerOperand` und `rechterOperand`, wenn es sich bei dem Ausdruck um eine Zahl handelt?

### c) Konsistenzprüfung

Schreibe zwei Hilfsmethoden:

`boolean istErlaubterOperator(String zeichenkette)`

...soll genau dann `true` zurückgeben, wenn die Zeichenkette `zeichenkette` der erlaubte Operator „\*“ ist.

`boolean istNatuerlicheZahl(String zeichenkette)`

...soll genau dann `true` zurückgeben, wenn die Zeichenkette `zeichenkette` eine natürliche Zahl (inklusive 0, ohne Vorzeichen) ist.

Verwende diese beiden Methoden in den Konstruktoren, um folgende Punkte sicherzustellen:

- Wenn `basisBlock` eine Zahl ist, dann darf der Ausdruck keine Operanden besitzen. Anderfalls soll eine Fehlermeldung ausgegeben werden.
- Wenn `basisBlock` ein erlaubter Operator ist, dann muss der Ausdruck die korrekte Anzahl an Operanden besitzen, die nicht den Wert `null` haben. Anderfalls soll eine Fehlermeldung ausgegeben werden.
- Wenn `basisBlock` weder eine Zahl, noch ein erlaubter Operator ist, handelt es sich um keinen gültigen Ausdruck und es soll ebenfalls eine Fehlermeldung ausgegeben werden.

#### d) Ausgabe

Schreibe eine Methode, mit der sich ein Ausdruck komplett mit allen *relevanten* Instanzvariablen selber zu einem String wandelt:

```
public String toString()
```

Diese Methode soll für die folgenden Ausdrücke die folgenden Strings liefern:

Für  $3 * 4 * 2$

```
„Ausdruck( op = *, links = Ausdruck( 3 ), rechts = Ausdruck( op = *, links =  
Ausdruck( 4 ), rechts = Ausdruck( 2 ) ) )”
```

Für  $(1 * 2) * (3 * 4)$

```
„Ausdruck( op = *, links = Ausdruck( op = *, links = Ausdruck( 1 ), rechts =  
Ausdruck( 2 ) ), rechts = Ausdruck( op = *, links = Ausdruck( 3 ), rechts =  
Ausdruck( 4 ) ) )”
```

Warum braucht man hier eine rekursive Methode?

#### Anmerkung:

Hier wird bei jedem Ausdruck zuerst der Operator und dann die beiden Operanden ausgegeben. Diese Reihenfolge nennt man *Präfix-Notation*. Analog gibt es noch die *Infix-Notation* (zuerst der linke Operand, dann der Operator, und zuletzt der rechte Operand) und die *Postfix-Notation* (zuerst die beiden Operanden, dann der Operator).

#### e) Auswertung

Schreibe eine Methode, mit der sich ein Ausdruck selbst auswertet:

```
int berechneWert()
```

#### f) Schleifen

Angenommen, wir wollen einen zusätzlichen Operator „\*\*“ zulassen. Der Ausdruck  $3 * 4$  soll dann als „3 hoch 4“ (also  $3 * 3 * 3 * 3$ ) interpretiert werden.

Schreibe eine *statische* Methode `exp`, die zwei `int`-Parameter nimmt und die entsprechende Potenz zurückliefert:

```
static int exp(int a, int b)
```

...soll  $a * b$  zurückgeben.

Verwende bei der Berechnung keine sonstigen Klassen, sondern stattdessen eine `while-do`-Schleife.

Versuche nun, die Methode `exp` stattdessen mit einer `do-while`-Schleife zu formulieren.

Ist auch eine `for`-Schleife möglich?

Kannst du dir eine Lösung ohne Schleifen, aber auch ohne Verwendung anderer Klassen vorstellen?

**g) Baumstruktur**

Die hier entwickelte Klasse **Ausdruck** ist ein Beispiel für einen „Baum“, eine Struktur, auf die man in der Informatik häufiger trifft. Kannst du dir vorstellen, wieso diese und ähnliche Strukturen „Baum“ heißen?