Westfälische Hochschule

Fachbereich Informatik und Kommunikation
Prof. Dr. Marcel Luis

30. April 2014

Praktikum 5 zu Objektorientierte Programmierung

Diese Hausaufgabe bezieht sich auf die 3. Hausaufgabe. Es geht um das Parsen und Auswerten von Ausdrücken. Der Parser, dessen Aufgabe es ist, eine textuelle Darstellung eines Ausdrucks in eine Baumstruktur zu überführen, soll hierbei sowohl mit korrekt gebildeten, als auch mit fehlerhaften Ausdrücken umgehen können.

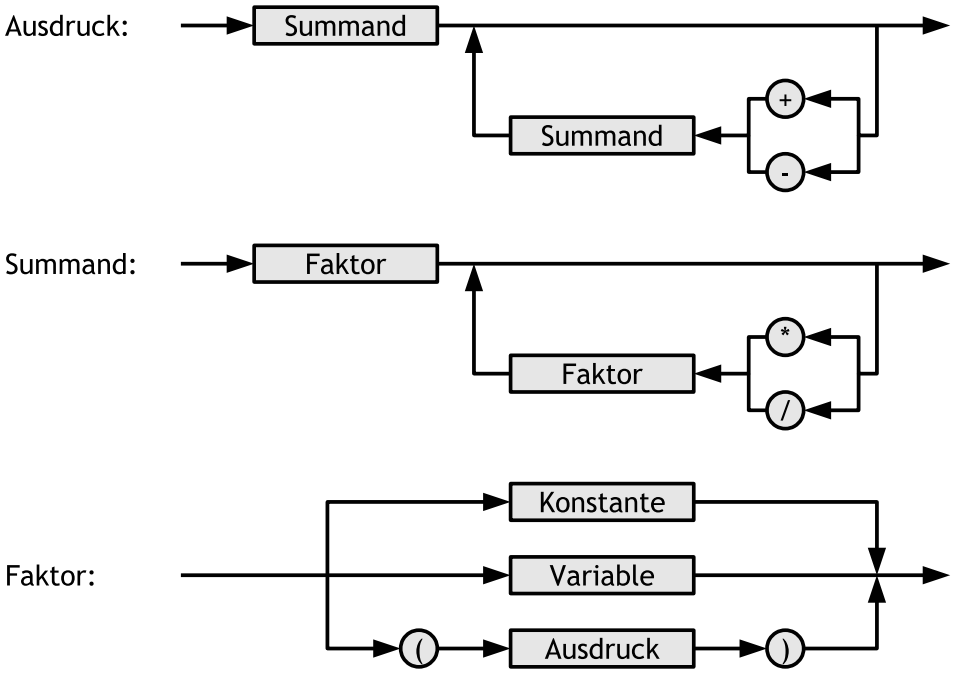
Aufgabe 5.1 (Aufwand ca. 10 Stunden, Abgabe bis 18.05.2014 um 18 Uhr)

Für die Entwicklung des Parsers ist es sinnvoll, sich zunächst nur um die Erkennung der *korrekten* Ausdrücke zu kümmern. Die Behandlung fehlerhafter Ausdrücke ist eine Erweiterung und bedeutet nicht, schon Programmiertes wieder verwerfen zu müssen.

Bearbeiten Sie diese Hausaufgabe also in zwei Schritten.

1. Schritt (Parsen fehlerfreier Ausdrücke und Auswerten von Ausdrücken)

Die „im Alltag“ übliche Darstellung arithmetischer Ausdrücke, die aus *Konstanten*, *Variablen* und den *vier Grundrechenarten* bestehen, ist die geklammerte Infixdarstellung. Solche Darstellungen lassen sich durch eine Grammatik erzeugen, deren Regeln wie folgt aussehen:



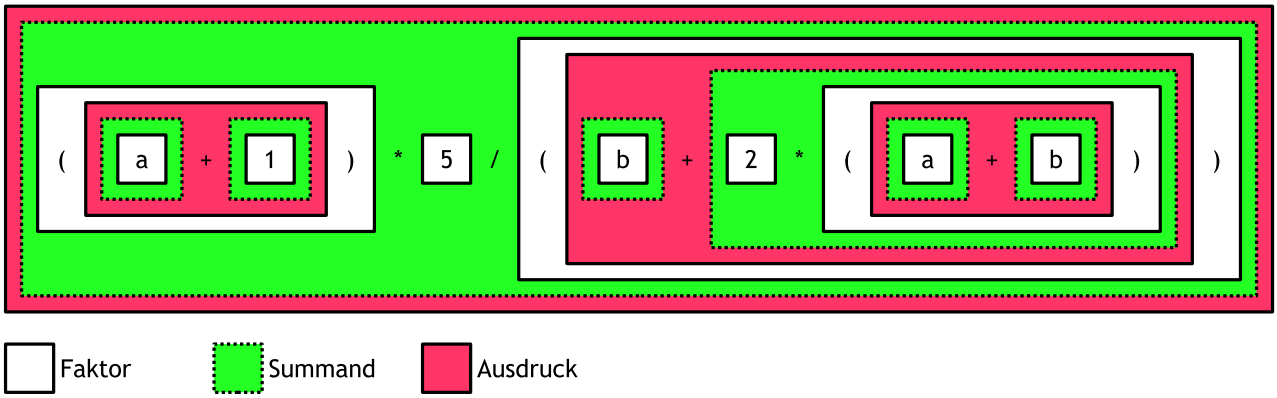
Diese Regeln besagen:

- Ein *Ausdruck* ist ein Summand oder eine Folge von Summanden, die durch + oder – verknüpft sind.
- Ein *Summand* ist ein Faktor oder eine Folge von Faktoren, die durch `*` oder `/` verknüpft sind.
- Ein *Faktor* ist eine Konstante, eine Variable oder ein geklammerter Ausdruck.

Für Konstanten und Variablen soll gelten:

- Eine *Konstante* ist eine nicht-leere Folge von Dezimalziffern.
- Eine *Variable* ist eine nicht-leere Folge von Dezimalziffern und Buchstaben (a bis z, A bis Z), die mit einem Buchstaben beginnt.

Am Beispiel des Ausdrucks `(a + 1) * 5 / (b + 2 * (a + b))` soll die Beziehung zwischen den Regeln und der geklammerten Infixdarstellung verdeutlicht werden.



Die farbige Darstellung ist wie folgt zu lesen:

- `(a + 1) * 5 / (b + 2 * (a + b))` ist ein Ausdruck (der äußere rote Kasten), denn `(a + 1) * 5 / (b + 2 * (a + b))` ist ein Summand (der darin liegende grüne Kasten).
- `(a + 1) * 5 / (b + 2 * (a + b))` ist ein Summand, denn der Text besteht aus den drei Faktoren `(a + 1)`, 5 und `(b + 2 * (a + b))` (die darin liegenden weißen Kästen).
- `(a + 1)` ist ein Faktor, denn der Text beginnt mit `(`, gefolgt von dem Ausdruck `a + 1` (roter Kasten) und der schließenden Klammer `)`.
- `a + 1` ist ein Ausdruck, denn der Text besteht aus den Summanden `a` und `1`.
- `a` ist ein Summand, denn `a` ist ein Faktor (analog für `1`).

- `a` ist ein Faktor, denn `a` ist eine Variable.
- `1` ist ein Faktor, denn `1` ist eine Konstante.
- usw.

Realisieren Sie nun für diese Aufgabe im Paket `ausdruck` folgende Klassen und Methoden:

Klasse `Ausdruck`:

- Enthält abstrakte Instanzmethode `int gibWert(Variablenbelegung)`, die den Wert dieses Ausdrucks basierend auf der Variablenbelegung liefert.

Die Klassen `Konstante`, `Variable` und `OperatorAusdruck` sind Unterklassen von `Ausdruck`.

Klasse `Konstante`:

- Enthält Konstruktor `Konstante(int)`, durch den ein konstanter Ausdruck für den angegebenen Wert erzeugt wird.

Klasse `Variable`:

- Enthält Konstruktor `Variable(String)`, durch den eine Variable mit dem angegebenen Namen erzeugt wird.

Klasse `OperatorAusdruck`:

- Enthält Konstruktor `OperatorAusdruck(Ausdruck, char, Ausdruck)`, durch den ein arithmetischer Ausdruck mit den angegebenen Teilausdrücken und dem Operatorsymbol erzeugt wird.

Klasse `Variablenbelegung`:

- Enthält Konstruktor `Variablenbelegung()`, durch den eine Variablenbelegung erzeugt wird, in der zunächst keiner Variablen ein Wert zugeordnet ist.
- Enthält Instanzmethode `void belege(String, int)`, durch die einer Variablen (1. Parameter) ein Wert (2. Parameter) zugeordnet wird. Ein evtl. vorhandener alter Wert wird dabei überschrieben.
- Enthält Instanzmethode `int gibWert(String)`, die den Wert liefert, der der angegebenen Variable zugeordnet ist.

Wozu benötigt man diese Klasse? Stellen Sie sich vor, Sie sollen den Ausdruck $a+2+x1$ auswerten. Sie werden schnell feststellen, dass dies nicht möglich ist, ohne die Werte von a und $x1$ zu kennen. Ist die Belegung dieser Variablen z. B. $a \mapsto 5$ und $x1 \mapsto -20$, dann lässt sich der Ausdruck zu -13 auswerten.

Genau hierum geht es bei der Klasse `Variablenbelegung`. Ein Objekt dieser Klasse repräsentiert die Beziehung zwischen (vielen) Variablen und ihren Werten.

Die obige beispielhafte Belegung von Variablen a und $x1$ wird wie folgt erzeugt:

1. Objekt der Klasse `Variablenbelegung` erzeugen.
2. Auf dieses Objekt `belege("a", 5)` anwenden.
3. Auf dasselbe Objekte `belege("x1", -20)` anwenden.

Eine Frage, die in diesem Zusammenhang häufig gestellt wird: Warum wird der Wert einer Variablen nicht schon im Objekt der Klasse `Variable` verwaltet? Weil ein Objekt dieser Klasse lediglich das *Auftreten* der Variablen innerhalb eines Ausdrucks repräsentiert (also die syntaktische Seite der Variablen).

Klasse `Parser`:

- Enthält Instanzmethode `Ausdruck parse(String)`, die die Textdarstellung (gewöhnliche geklammerte Infixdarstellung) eines Ausdrucks parst und ein entsprechendes `Ausdruck`-Objekt zurückgibt. Beliebig viele Leerzeichen (einschließlich 0 Leerzeichen!) zwischen den Komponenten des Ausdrucks sind zulässig. Gehen Sie für diesen ersten Schritt davon aus, dass nur *gültige* Darstellungen von Ausdrücken übergeben werden. (Die Behandlung fehlerhafter Darstellungen ist Gegenstand des zweiten Schritts. Dabei soll durch eine Ausnahme die am weitesten „links“ stehende Fehlerstelle angezeigt werden.)

Eine Testklasse für den ersten Schritt der Entwicklung haben Sie in der 3. Hausaufgabe realisiert.

Einige Beispiele zum Auswerten von Ausdrücken:

Testmuster	Sollergebnis
Belegung ohne Zuordnungen und Ausdruck `"10"` auswerten	10
Belegung i=3 und Ausdruck `"7 + i"` auswerten	10
Belegung i=3 und Ausdruck `"(i + 18)/2"` auswerten	10 (ganzzahlige Division!)
Belegung i=3, k=4, j=9 und Ausdruck `"1 + 2 * (i + 2 * k - 1) / 2 + j"` auswerten	20

Für die Auswertung der Ausdrücke dürfen Sie davon ausgehen, dass alle im Ausdruck vorkommenden Variablen in der Variablenbelegung einen Wert besitzen. Die Ausnahme, die bei Division durch 0 auftritt, müssen Sie nicht behandeln.

Hinweise zum Parse-Algorithmus

Die wesentliche Aufgabe des Parsens ist es, die Struktur des dargestellten Ausdrucks zu erkennen. Für die Struktur spielen aber weder die Stellenanzahl einer Konstanten, noch die Länge eines Variablenbezeichners eine Rolle. Auch die Leerzeichen sind ohne Bedeutung.

Deshalb empfiehlt es sich, zum Parsen des Ausdrucks nicht unmittelbar auf den einzelnen Zeichen zu arbeiten, sondern zunächst die Liste der *Tokens* zu ermitteln, aus denen der Ausdruck besteht und die zur Erkennung seiner Struktur erforderlich sind.

Für den Ausdruck `10 + 200 * (i + 2*betrag1 - 1) / 2 + j` sind dies zum Beispiel die Tokens `10`, `+`, `200`, `*`, `(`, `i`, `+`, `2`, `*`, `betrag1`, `-`, `1`, `)`, `/`, `2`, `+` und `j`.

Es ist sinnvoll, den Algorithmus in Methoden `parseAusdruck`, `parseSummand` und `parseFaktor` aufzuteilen. `parseFaktor` liefert ein `Ausdruck`-Objekt, das einen Faktor repräsentiert. `parseSummand` liefert ein `Ausdruck`-Objekt, das einen Summanden repräsentiert, usw. Die Implementierung der Methoden orientiert sich zweckmäßig an der obigen Syntaxdefinition.

Für `parseSummand` bedeutet dies z. B.: Ein Summand ist ein Faktor oder eine Folge von Faktoren, die durch `*` oder `/` verknüpft sind. Also stützt sich `parseSummand` auf `parseFaktor` ab. Nach dem Parsen des ersten Faktors fährt die Methode so lange fort, wie noch Faktoren für diesen einen Summanden folgen. Folgt kein weiterer Faktor, liefert die Methode das `Ausdruck`-Objekt für den geparsten Summanden, d. h. das `Ausdruck`-Objekt für alle durch `*` oder `/` verknüpften Faktoren.

2. Schritt (Ergänzung der Lösung um Behandlung fehlerhafter Ausdrücke)

Erweitern Sie die Instanzmethode `Ausdruck parse(String) throws ParseException` in der Klasse `Parser`, sodass sie sich für korrekte Textdarstellungen genauso verhält wie für den ersten Schritt beschrieben, und fehlerhafte Textdarstellungen durch eine `java.text.ParseException` anzeigt.

- Enthält die Textdarstellung ein ungültiges Token, ist die Meldung der Exception `"ungueltiges Token ..."`, wobei `...` für das erste ungültige Token steht. Der Error-Offset (siehe Konstruktor der Klasse `ParseException`) gibt den Index dieses Tokens an (siehe Testfälle).

- Endet die Textdarstellung vorzeitig, d. h. ließe sie sich durch Anhängen von Tokens zu einer gültigen Darstellung vervollständigen, ist die Meldung der Exception `"unerwartetes Ende"` (siehe Testfälle). Der Error-Offset ist in diesem Fall ohne Bedeutung und kann beliebig sein.

Testfälle

Testmuster	Sollergebnis
Textdarstellung `"( )"`	ParseException mit Meldung `"ungueltiges Token )"` und Error-Offset 2 (denn das 2-te Token ist nicht korrekt)
`"( + )"`	ParseException mit Meldung `"ungueltiges Token +"` und Error-Offset 2
`"(( 10) 10)"`	ParseException mit Meldung `"ungueltiges Token 10"` und Error-Offset 5
`"1a"`	ParseException mit Meldung `"ungueltiges Token 1a"` und Error-Offset 1
`"a 1"`	ParseException mit Meldung `"ungueltiges Token 1"` und Error-Offset 2
`"2 - )"`	ParseException mit Meldung `"ungueltiges Token )"` und Error-Offset 3
`"2 * (s - t * a_b)"`	ParseException mit Meldung `"ungueltiges Token a_b"` und Error-Offset 8
`"2 * (s - )"`	ParseException mit Meldung `"ungueltiges Token )"` und Error-Offset 6
`"2 * (s – t) ("`	ParseException mit Meldung `"ungueltiges Token ("` und Error-Offset 8
`""`	ParseException mit Meldung `"unerwartetes Ende"`
`"2 * (s - t * 5"`	ParseException mit Meldung `"unerwartetes Ende"`
`"2 * (3 * (i + 4)"`	ParseException mit Meldung `"unerwartetes Ende"`

Realisieren Sie im Paket `ausdruck` außerdem eine Testklasse `ParserTest` basierend auf JUnit. Erzeugen Sie in der mit `@Before` annotierten Methode einen Parser und realisieren Sie drei Testmethoden, jeweils eine zum Test des Parsens von

- korrekten Textdarstellungen,
- Textdarstellungen mit ungültigem Token,
- Textdarstellungen mit unerwartetem Ende.

In der ersten Testmethode müssen Sie testen, dass die erwarteten Ausdrücke geliefert werden, in den letzten beiden, ob die erwarteten Ausnahmen geworfen werden. Verwenden Sie als Testmuster mindestens die korrekten Textdarstellungen aus der 3. Hausaufgabe und die oben angegebenen fehlerhaften Textdarstellungen.

Damit die Funktion des Parsers automatisiert testbar ist, müssen Ihre Klassen einige Voraussetzungen erfüllen. Dies war u. a. Gegenstand der 3. Hausaufgabe.

Hinweise

- Achten Sie darauf, dass in den Eigenschaften Ihres NetBeans-Projekts die Zeichenkodierung UTF-8 eingestellt ist. Ist dies nicht der Fall, kann es bei der automatischen Auswertung Ihrer Lösung zu Fehlern beim Compilieren kommen. Die Lösung wird dann mit einer Erfolgsquote von 0% gewertet.

- Verwenden Sie nur den Vorlesungsstoff bis einschließlich Kapitel 8.

- Sie dürfen selbstverständlich zusätzliche Klassen und Methoden realisieren. Zusätzliche Klassen müssen ebenfalls im Paket `ausdruck` liegen.

- Denken Sie an die ausreichende Dokumentation und Kommentierung Ihrer Lösung. Beachten Sie die unterschiedliche Bedeutung der *externen Dokumentation* `/** ... */` *vor* einer Klasse oder Methode und des *Implementierungskommentars* `/* ... */` *innerhalb* einer Methode. Die externe Dokumentation sagt, *was* eine Klasse oder Methode leistet, der Implementierungskommentar hilft zu verstehen, *wie* es gemacht wird. Verwenden Sie Implementierungskommentare vor allem, um den Berechnungsablauf verständlich zu machen.

- Erzeugen Sie die HTML-Dokumentation Ihrer Klasse und überzeugen Sie sich, ob Ihre externe Dokumentation sinnvoll und ohne Kenntnis des Quellcodes der Klasse hilfreich ist.

- Erstellen Sie je ein `zip`-Archiv des Quellordners `ausdruck` unter `src` und des gleichnamigen Ordners unter `test` und laden Sie beide zu Moodle hoch.

- Im Veranstaltungskalender finden Sie die Termine, an denen diese Lösung im Praktikum besprochen wird. Bringen Sie zu diesen Terminen bitte die Auswertung Ihrer Lösung mit, entweder ausgedruckt oder unmittelbar auf Ihrem Rechner verfügbar.