Aufgabenblatt 07

Bearbeitungsende: 27.11.2022

Aufgabe 1 [Theorie]

Was ist die aus der folgenden lexikalischen Grammatik mit Startsymbol *DecimalNumeral* erzeugte Sprache?

```
DecimalNumeral:
  0
  NonZeroDigit [ { DigitOrUnderscore } Digit ]

DigitOrUnderscore:
  Digit
  _

Digit:
  0
  NonZeroDigit

NonZeroDigit:
  eines von 1 2 3 4 5 6 7 8 9
```

Produzieren Sie zur Beantwortung zunächst möglichst unterschiedliche Beispiele für Tokens, die sich aus der Grammatik erzeugen lassen.

Beschreiben Sie dann in eigenen Worten, wie solche Tokens (Java-Dezimalzahlliterale) im Allgemeinen aufgebaut sind.

Hinweise:

- [ ] und { } sind Klammern der Grammatik, nicht des beschriebenen Codes selbst. (Die Unterstreichungen dienen der Verdeutlichung, da die Kursivschrift ggf. nicht klar zu erkennen ist.)
- Diese Grammatikregel wird *lexikalisch* betrachtet: Zwischen den Elementen (Nichtterminalen und Terminalen) in einer Ersetzung entstehen keine Trennungen („Leerzeichen“).
- Es handelt sich bei dieser Grammatik um einen vereinfachten Auszug aus der Java-Sprachspezifikation für Dezimalzahlliterale.

Aufgabe 2 [Theorie]

(a) Gegeben sei die folgende Grammatik/-regel:

```
Ausdruck:
  Ausdruck + Ausdruck
  Ausdruck - Ausdruck
  Ausdruck * Ausdruck
  Ausdruck / Ausdruck
  ( Ausdruck )
  DecimalNumeral
```

Wie lassen sich daraus die folgenden Terme ableiten?

(1) `6`

(2) `6 - 2`

(3) `6 - 2 + 5`

(4) `6 - 2 * 5`

(5) `(6 - 2) * 5`

Welche Probleme stellen Sie dabei fest?

(b) Gegeben sei die folgende Grammatik:

```
Ausdruck:
  StrichAusdruck

StrichAusdruck:
  StrichAusdruck + StrichAusdruck
  StrichAusdruck - StrichAusdruck
  PunktAusdruck

PunktAusdruck:
  PunktAusdruck * PunktAusdruck
  PunktAusdruck / PunktAusdruck
  EinzelAusdruck

EinzelAusdruck:
  ( Ausdruck )
  DecimalNumeral
```

Wie lassen sich daraus die oben gegebenen Terme ableiten?

Welche Unterschiede zu der vorherigen Version stellen Sie dabei fest?

(c) Gegeben sei die folgende Grammatik:

```
Ausdruck:
  StrichAusdruck

StrichAusdruck:
  StrichAusdruck + PunktAusdruck
  StrichAusdruck - PunktAusdruck
  PunktAusdruck

PunktAusdruck:
  PunktAusdruck * EinzelAusdruck
  PunktAusdruck / EinzelAusdruck
  EinzelAusdruck

EinzelAusdruck:
  ( Ausdruck )
  DecimalNumeral
```

Wie lassen sich daraus die oben gegebenen Terme ableiten?

Welche Unterschiede zu den vorherigen Versionen stellen Sie dabei fest?

Aufgabe 3 [Theorie]

Erstellen Sie eine Grammatik für Zugfahrpläne mit folgendem Aufbau:

```
Stuttgart              ab 16:01
Ulm           an 17:04 ab 17:19
Biberach      an 17:40 ab 17:40 Gleis 1
Durlesbach    an 17:59 ab 17:59
Meckenbeuren  an 18:18          Gleis 2
```

Leerzeichen wurden hier zur Verdeutlichung teils vervielfacht, was die Grammatik jedoch nicht beschreiben soll. Gleise sind nur angegeben, wenn sie für diesen Zug festgelegt sind.

Führen Sie dazu geeignete Nichtterminale ein wie z.B. *`Uhrzeit`*.

Sie müssen keine komplexeren Fälle abdecken als hier angegeben (z.B. Ortsnamen mit Bindestrich). Natürlich kann eine Verbindung beliebig viele Zwischenstationen enthalten.

Aufgabe 4 [Programmierung]

Schreiben Sie eine Klasse `Interpreter` mit einer Methode `verarbeiteEingabe`, die einen `Scanner` und einen `PrintStream` als Argumente annimmt. Sie liest über den `Scanner` wiederholt, bis zum Eingabeende, Eingaben in folgendem Format (als Syntaxregel formuliert) ein:

`Eingabe`:
  `{ EinzelEingabe }`

`EinzelEingabe`:
  `Position Zeichen` `in` *`Zeichenfolge`*

`Position`:
  `vor`
  `nach`

`Zeichen` bezeichnet je ein von der Nutzerin gewähltes Zeichen und *`Zeichenfolge`* je eine nichtleere, von der Nutzerin gewählte Zeichenfolge ohne Leerzeichen.

`vor`, `nach` und `in` sind fest vorgegebene Zeichenfolgen (Terminale) – wie Schlüsselwörter in Java.

Ein Beispiel einer einzelnen Eingabe wäre:

```
nach a in Grammatik
```

Es soll dann jeweils diejenige Zeichenfolge in den `PrintStream` geschrieben werden, gefolgt von einem Zeilenumbruch, die **nach dem ersten** Vorkommen des Zeichens *`Zeichen`* in der Zeichenfolge *`Zeichenfolge`* steht.

Im oben gegebenen Beispiel wäre dies `"mmatik"`.

Bei Eingabe von `vor` statt `nach` soll entsprechend die Zeichenfolge geschrieben werden, die **vor dem letzten** Vorkommen des Zeichens *`Zeichen`* in der Zeichenfolge *`Zeichenfolge`* steht.

Für die Einzeleingabe

```
vor a in Grammatik
```

wäre diese also `"Gramm"`.

Kommt das Zeichen *`Zeichen`* nicht in *`Zeichenfolge`* vor, soll im `vor`-Fall eine leere Zeichenkette, im `nach`-Fall die ganze *`Zeichenfolge`* geschrieben werden, ebenfalls gefolgt von einem Zeilenumbruch.

Die Methode liefert keinen Ergebniswert.

Implementieren Sie die Lösung mit Hilfe der folgenden weiteren Methoden, die jeweils das entsprechende Nichtterminal verarbeiten:

- `verarbeiteEingabeEinzel`
- `verarbeiteEingabeEinzelVor`
- `verarbeiteEingabeEinzelNach`

Jede Methode nimmt den `Scanner` und den `PrintStream` als Argumente an. Die zweite und dritte Methode nehmen zudem das *`Zeichen`* und die *`Zeichenkette`* als Argumente an, die eingegeben wurden.

Gehen Sie davon aus, dass die Eingaben korrektes Format haben.

Hinweise: Verwenden Sie aus der Vorlesung bekannte Methoden der Klasse `String`.

Aufgabe 5 [Programmierung]

Schreiben Sie eine Klasse `Eingabe` mit einer Methode `filter`. Die Methode nimmt einen `Scanner`, einen `java.io.PrintStream` und drei Parameter `start`, `stop` und `skip` an, welche Zeichenketten sind.

Die Methode soll aus dem `Scanner` lesen, bis die Eingabe endet. Die gelesenen Zeichenketten sollen in den `PrintStream` geschrieben werden, allerdings in *gefilterter* Form:

- Beliebig viele aufeinanderfolgende „White Spaces“ werden durch ein einzelnes Leerzeichen ersetzt.
- Wenn eine Zeichenkette gleich `start` ist, werden diese Zeichenkette und die folgenden ignoriert, bis eine Zeichenkette gelesen wird, die gleich `stop` ist.
- Wenn eine Zeichenkette gleich `skip` ist, wird diese Zeichenkette und der Rest der Zeile ignoriert.

Nach der letzten Zeichenkette folgt ein Leerzeichen. Am Ende folgt stets ein Zeilenumbruch.

Die Methode liefert die Anzahl der ausgegebenen Zeichenketten.

Beispiel: Die Eingabe

```
/*
 * Programm: gibt "Hi!" aus
 */
public class Hi {
  public static void main(String[] args) {
    // Ausgabe der Nachricht
    System.out.println("Hi!");
  }
}
```

wird durch Aufruf von `filter` mit den Argumenten `"/*"`, `"*/"` und `"//"` verwandelt in die Ausgabe:

```
public class Hi { public static void main(String[] args) { System.out.println("Hi!"); } }
```

(Am Ende folgen ein Leerzeichen und ein Zeilenumbruch.)

Das Ergebnis ist 13.

Hinweis: Sie müssen/sollten keine `String`-Methoden außer `equals` verwenden.

Zusatzfrage: Wie könnte man die Lösung erweitern, so unmittelbar nach/vor dem Start-/Skip-/Stop-String weitere Zeichen stehen können, also auch Kommentare wie `/** ... **/` oder `/// ...` erkannt werden?

Aufgabe 6 [Theorie]

Machen Sie sich mit Style Guidelines zu Java vertraut (Beispiele in LEA unter „Software“), die auch im Praktomat (unverbindlich) überprüft werden.

Ignorieren sie dabei zunächst Regeln, die sich auf Sprachelemente beziehen, die Sie noch nicht kennen. Sehen Sie im weiteren Verlauf der Veranstaltung nach, welche Konventionen für neu vorgestellte Sprachelemente gelten.

Gewöhnen Sie sich an diese Konventionen – zuerst muss man sich dazu etwas zwingen, aber nach einer Weile empfindet man sie als selbstverständlich.

Falls Sie mit einer IDE arbeiten, informieren Sie sich, in welchen Weisen sie Sie bei der Formatierung von Java-Code (z.B. durch automatische Einrückung) unterstützen kann. Entscheiden Sie, in welchen Punkten Sie sich helfen lassen wollen, in welchen (zumindest zu Lernzwecken) eher nicht (z.B. automatische Codeergänzung).

Lösungen zu mit [**Programmierung**] markierten Aufgaben sind im **Praktomat** einzureichen.

Lösungen zu mit [**Programmierung – nicht bewertet**] markierten Aufgaben können ebenfalls im **Praktomat** eingereicht werden, werden jedoch nicht bewertet.

Allgemeine **Fragen** zu den Aufgaben können Sie im **LEA-Forum „Übungsaufgaben“** stellen.

Hilfe bei der Lösung der Aufgaben erhalten Sie in den **Übungen** und in der **Studierwerkstatt** .