Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Fachbereich Informatik und Kommunikation
Prof. Dr. Ulrike Griefahn

Minijava-Projekt (ÜSB)

Aufgabenblatt 1

WS 2021/22

Aufgabe 1 (Scanner-Generator)

Verwenden Sie den Scanner-Generator JavaCC, um einen Scanner für Minijava zu realisieren.

Genauer gesagt, Sie sollen in dieser Aufgabe die Tokens und Non-Tokens von Minijava mit Hilfe von JavaCC definieren und daraus einen Scanner automatisch erzeugen lassen.

Abschnitt 2.7 des Vorlesungskapitels zur Lexikalischen Analyse enthält Folien, auf denen die Arbeitsweise von Scanner-Generatoren beschrieben ist und die für diese Aufgabe notwendige JavaCC-Syntax vorgestellt wird.

In dieser Aufgabe sollen Sie die JavaCC-Datei `MiniJavaParser.jj` erweitern. Sie finden die Datei im Package `parser` des Minijava-Projekts. Verändern Sie bitte nicht den Namen des Parsers oder den der Datei. Verändern Sie bitte auch nicht den Inhalt der Methode `main`.

In der Datei `README.md` finden Sie den Befehl, mit dem Sie aus der `jj`-Datei den lexikalischen Scanner generieren können. In der `main`-Methode des Scanners können Sie über die Variable `sample` das zu testende MiniJava-Programm festlegen. Bei Ausführung des Scanners werden die erkannten Tokens in eine `.tok`-Datei ausgegeben, die Sie in `./samples/out/` finden. Testen Sie Ihren Scanner mit allen vorhandenen und auch eigenen Minijava-Programmen!

Hinweise zur Definition von Tokens in JavaCC

Infos zu JavaCC finden Sie unter `https://javacc.github.io/javacc/`.

- *Token-Definition*: Die Definition von Non-Tokens und Tokens mit JavaCC erfolgt wie im Folgenden dargestellt. Tokens, die nicht an den Parser weitergegeben werden sollen, definieren Sie im Bereich `SKIP` und Tokens, die im Parser weiterverarbeitet werden sollen, im Bereich `TOKEN`. Tokennamen, denen ein `#` vorangestellt ist, können als Abkürzungen in anderen Definitionen genutzt werden. Sie werden nicht an den Parser geliefert.

```
SKIP : {
    < "//" ( ["a"-"z"] )* "\n" >
    | < (" " | "\t" | "\n")+ >
}

TOKEN : {
    < IF      : "if" >
    | < #DIGIT  : ["0"-"9"] >
    | < NUM     : ( <DIGIT> )+ >
}
```

- *Mehrdeutigkeit*: Passt ein Lexem zu mehreren Tokens, bspw. passt das Schlüsselwort **`if`** auch zur Spezifikation von Bezeichnern, entscheidet die Reihenfolge der Definitionen über die Priorität. Die erste Definition hat Vorrang gegenüber allen nachfolgenden.
- *Negation*: Für mehrzeilige Kommentare müssen Sie im regulären Ausdruck definieren, dass Sie bestimmte Zeichen ausschließen. In JavaCC stellt man hierzu die Tilde `~` vor die Liste der Zeichen, die nicht erlaubt sind, bspw. `[~*/]`.