Lexer mit ANTLR generieren

Carsten Gips (HSBI)

Lexer: Erzeugen eines Token-Stroms aus einem Zeichenstrom

```
/* demo */
a= [5  , 6]     ;
```

Lexer: Erzeugen eines Token-Stroms aus einem Zeichenstrom

```
/* demo */
a= [5  , 6]     ;
```

```
<ID, "a"> <ASSIGN> <LBRACK> <NUM, 5> <COMMA> <NUM, 6> <RBRACK> <SEMICOL>
```

Definition wichtiger Begriffe

- **Token**: Tupel (Tokenname, optional: Wert)
- **Lexeme**: Sequenz von Zeichen im Eingabestrom, die auf ein Tokenpattern matcht und vom Lexer als Instanz dieses Tokens identifiziert wird.
- **Pattern**: Beschreibung der Form eines Lexems

Typische Muster für Erstellung von Token

1. Schlüsselwörter
   - Ein eigenes Token (RE/DFA) für jedes Schlüsselwort, oder
   - Erkennung als Name und Vergleich mit Wörterbuch
2. Operatoren
   - Ein eigenes Token für jeden Operator, oder
   - Gemeinsames Token für jede Operatoren-Klasse
3. Bezeichner: Ein gemeinsames Token für alle Namen
4. Zahlen: Ein gemeinsames Token für alle numerischen Konstante
5. String-Literale: Ein gemeinsames Token
6. Komma, Semikolon, Klammern, ...: Je ein eigenes Token
7. Regeln für White-Space und Kommentare etc. ...

Hello World

```
grammar Hello;

start       : 'hello' GREETING ;

GREETING    : [a-zA-Z]+ ;
WHITESPACE  : [ \t\n]+ -> skip ;
```

Konsole: Hello (Classpath, Aliase, grun, Main, Dateien, Ausgabe)

Verhalten des Lexers: 1. Längster Match

Primäres Ziel: Erkennen der längsten Zeichenkette

```
CHARS   : [a-z]+ ;
DIGITS  : [0-9]+ ;
FOO     : [a-z]+ [0-9]+ ;
```

Verhalten des Lexers: 2. Reihenfolge

Reihenfolge in Grammatik definiert Priorität

```
FOO     : 'f' .*? 'r' ;
BAR     : 'foo' .*? 'bar' ;
```

Verhalten des Lexers: 3. Non-greedy Regeln

Non-greedy Regeln versuchen *so wenig* Zeichen wie möglich zu matchen

```
FOO     : 'foo' .*? 'bar' ;
BAR     : 'bar' ;
```

Verhalten des Lexers: 3. Non-greedy Regeln

Non-greedy Regeln versuchen *so wenig* Zeichen wie möglich zu matchen

```
FOO     : 'foo' .*? 'bar' ;
BAR     : 'bar' ;
```

Achtung: Nach einer non-greedy Sub-Regel gilt "*first match wins*"

`.*? ('4' | '42')`

=> `'42'` ist "toter Code" (wegen der non-greedy Sub-Regel `.*?`)!

Attribute und Aktionen

```
grammar Demo;

@header {
import java.util.*;
}

@members {
String s = "";
}

start   : TYPE ID '=' INT ';' ;

TYPE    : ('int' | 'float') {s = getText();} ;
INT     : [0-9]+            {System.out.println(s+":"+Integer.valueOf(getText()));};
ID      : [a-z]+            {setText(String.valueOf(getText().charAt(0)));} ;
WS      : [ \t\n]+ -> skip ;
```

Wrap-Up

Lexer mit ANTLR generieren: Lexer-Regeln werden mit **Großbuchstaben** geschrieben

- Längster Match gewinnt, Gleichstand: zuerst definierte Regel
- *non greedy*-Regeln: versuche so *wenig* Zeichen zu matchen wie möglich
- Aktionen beim Matchen

LICENSE