ANTLR + ANTLRWorks

nie za długie wprowadzenie

ANTLR krótko

- Generuje gramatyki LL(k)
  - k jest dobierane automatycznie
- Wygenerowany kod ma postać 1:1 produkcja : metoda w pliku
- Do działającego parsera trzeba użyć jara antlr
- ANTLRWorks – narzędzie do budowania gramatyk, sprawdzania, rysowania diagramów itp. itd.

Definicja gramatyki

```
/** This is a grammar doc comment */
grammar-type grammar name;
options { name1 = value; name2 = value2; ... }
tokens { token-name1; token-name2 = value; ... }
scope global-scope-name-1 { «attribute-definitions» }
scope global-scope-name-2 { «attribute-definitions» }
...
@header {...}
@lexer::header {...}
@members {...}

«rules»
```

Definicja gramatyki

- @header – nagłówek klasy analizatora składniowego.
  - importy, pakiet itp.
- @lexer::header – nagłówek klasy analizatora leksykalnego
  - importy, pakiet itp.
- @members – pola w analizatorze składniowym
  - Zmienne pomocnicze

Produkcje i terminale

```
/** rule comment */
access-modifier rule-name[«arguments»] returns [«return-values»]
throws name1, name2, ...
options {...}
scope {...}
scope global-scope-name;
@init {...}
@after {...}
    : «alternative-1» -> «rewrite-rule-1»
    | «alternative-2» -> «rewrite-rule-2»
    ...
    | «alternative-n» -> «rewrite-rule-n»
    ;
    catch [«exception-arg-1»] {...}
    catch [«exception-arg-2»] {...}
    finally {...}
```

Leksemy 1/2

Podstawowy token definiujemy jako ciąg znaków

```
ID : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')* ;
```

Białe znaki, komentarze, komentarze w liniach

```
WS  :  (' '|'\r'|'\t'|'\u000C'|'\n') {$channel=HIDDEN;}
    ;

COMMENT
    :   '/*' .* '*/' {$channel=HIDDEN;}
    ;
LINE_COMMENT
    : '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    ;
```

Leksemy 2/2

„Stałe” w definicjach leksemów

```
HexLiteral : '0' ('x'|'X') HexDigit+ ;

fragment
HexDigit : ('0'..'9'|'a'..'f'|'A'..'F') ;
```

Uwaga ! Ważna jest kolejność definicji symboli nieterminalnych !

Produkcje

Definicje produkcji. Terminale definiujemy WIELKIMI literami. Nieterminale małymi

```
decl : ^(DECL type declarator) {System.out.println($type.text+"
"+$declarator.text);}
     ;
```

Widoki ANTLRWorks

- Syntax diagram – pokazuje diagramy terminali i nieterminali (jako automaty skończone)
- Interpreter – pozwala zbudować drzewo składni z kodu wejściowego
- Debugger – debugowanie kodu krok po kroku od dowolnej produkcji
- Console – wydruk konsolowy. Błędy, ostrzeżenia itp.

Uruchamianie logiki 1/2

- W dowolnym miejscu w produkcji można wstawić kod do uruchomienia pomiędzy znakami { ... }
- Produkcja może zwracać wartość. Po nazwie returns [values] np returns [int value]
- Aby zwrócić wartość podstawiamy do zmiennej $value
- Symbole można przyporządkowywać do zmiennych np a=jakis(Nie)terminal
- Do zmiennych i terminali dostajemy się poprzez znak $

Uruchamianie logiki 2/2

- Do zmiennych i terminali dostajemy się poprzez $TERM.text bądź $TERM.value, gdzie TERM to nazwa terminala bądź zmiennej

```
equation returns [int value]
   :   a=eqParam OP b=eqParam {Integer aI = $a.value;
            Integer bI = $b.value;
            if ($OP.text.equals("+")) {
               $value = aI + bI;
            } else if ($OP.text.equals("-")) {
               $value = aI - bI;
            } else if ($OP.text.equals("/")) {
               $value = aI / bI;
            } else if ($OP.text.equals("*")) {
               $value = aI * bI;
            }};
```

Uruchomienie parsera

- W ustawieniach programu podajemy ścieżkę do naszego projektu
- Generujemy kod poleceniem Generate->Generate Code
- Piszemy kod uruchamiający parser

```
public static void main(String[] str) throws IOException,
RecognitionException {
    CharStream stream = new ANTLRInputStream(System.in);

    CommonTokenStream tokens = new CommonTokenStream(new CalcLexer(stream));

    CalcParser parser = new CalcParser(tokens);
    parser.s();   // uruchomienie produkcji startowej
}
```

Przykłady

- Kalkulator
- Jakiś skrypt...