ANTLR-2

COM S 319

PARSER

Hello World

```
grammar HelloWorld; // It is important to know the NAME of the grammer
    1


// PARSER RULES
// our grammar accepts only salutation followed by an end symbol
// The LEFT HAND SIDE is lowercase. That is a "variable" in the grammer
// The variable derives the right hand side which are in this case
// tokens.
start : (SALUTATION ENDSYMBOL)+;
  2
// LEXER RULES
// the grammar must contain at least one lexer rule
// The LEFT HAND SIDE is all CAPITALIZED. These are the token ids.
// The Right hand side is the "regular expression" rules to match tokens
// just like lex
SALUTATION: 'Hello world';
ENDSYMBOL: '!' ;
```

(1) Note: This generates BOTH lexer AND parser

(2) start is a VARIABLE

Simple

```
grammar simple;


// PARSER RULES
start   : stat NEWLINE + ;

stat  : ALPHA  | NUMERIC  | EXIT ;

// LEXER RULES
ALPHA   : ('a'..'z'|'A'..'Z')+
        { System.out.println("Found alpha: "+getText()); }
        ;

NUMERIC : ('0'..'9')+
        { System.out.println("Found numeric: "+getText()); }
        ;

EXIT    : '.' { System.out.println("EXITING!"); System.exit(0); } ;

NEWLINE:'\r'? '\n' ;

WS  :   (' '|'\t')+ {skip();} ;
```

1

(1) There are TWO variables.

(2) Lexer part is same as before

Boolean

```
grammar SimpleBoolean;

// LEXER RULES
//the grammar must contain at least one lexer rule
LPAREN : '(' ;
RPAREN : ')' ;
AND : '&&';
OR : '||';
NOT : '!';
NAME : ('a'..'z' | '0'..'9')+;
WS : ( ' ' | '\t' | '\r' | '\n' )+   {skip();};

// PARSER RULES

//start rule
start : (andexpression ';')+ ;
andexpression : orexpression (AND orexpression)*;
orexpression : notexpression (OR notexpression)*;
notexpression : NOT atom | atom;
atom : NAME | LPAREN andexpression RPAREN;
```

1

(1) Complex variable rules

CountInts

```
grammar Count;

@members {      1
  int count = 0;
}

// PARSE RULE
start    2
  @after {System.out.println("Total ints Count is " + count);}
   :  INT {count++; }  ( ',' INT    {count++;} )*
   ;

// LEXER RULE
INT : [0-9]+ ;
WS : [ \r\t\n]+ {skip();} ;
~
~
```

(1) can add variables to parser!

(2) see @after used to work on post processing

CountPairs

```
grammar CountPair;

@members {
  int count = 0;
}

// PARSE RULE
start
  @after {System.out.println("Total pair Count is " + count);}
   :  pair +
   ;

pair
  @after {count++;
    System.out.println("Found pair: ");  }
     : '('
       word {System.out.print("found word: " + $word.text);}
       ':'
       integer {System.out.println(" found number:" + $integer.text);}
       ')'
     ;

word : WORD;

integer : INT;

// LEXER RULE
WORD: [a-z]+;
INT : [0-9]+ ;
WS : [ \r\t\n]+ {skip();} ;
```

(1) and (2) can use $expr.text

ExprCalc

```
grammar ExprCalc;

@members {
  long exprVal;
  long v1, v2;
}

// PARSER RULES
start:  (expr {System.out.println("******* ANSWER = "+$expr.value);} NEWLINE)
* EOF ;
expr returns [long value]
    @after {
       System.out.println("after: "    $value);
    }
    :
      a=expr {v1 = $a.value;} '*' b=expr  {$value = v1 * $b.value;}
    | a=expr {v1 = $a.value;} '/' b=expr  {$value = v1 / $b.value;}
    | a=expr {v1 = $a.value;
       System.out.println("--v1: " + v1);
       System.out.println("--a.text: " + $a.text);
       System.out.println("--a.value: " + $a.value);

    } '+' b=expr  {$value = v1 + $b.value;
       System.out.println("--v1+v2: " + $value);
       System.out.println("--expr.text: " + $b.text);
       System.out.println("--expr.value: " + $b.value);
    }
    | a=expr {v1 = $a.value;} '-' b=expr  {$value = v1 - $b.value;}
    | INT { $value = Integer.parseInt($INT.text);
       System.out.println("INT: " + $value);

          }
    | '(' expr {$value = $expr.value;} ')'
    ;

// LEXER RULES
NEWLINE : ['\r''\n']+ ;
INT     : [0-9]+ ;
```

(1) members

(2) $expr,value

(3) a=expr and b=expr

SHOW TestGrammar.java

SHOW ANTLRWORKS

Summary

- grammar
- variables

- @after{}
- $expr.text in parser code segment
- $expr.value in parser code segment
- a=expr in right hand rule

- getText() in lexer code