

# Javacc

- Overview
- Caveats
- Grammar File Syntax
- Regular Expression Productions
- Example Regular Expression Productions
- Tokens
- Grammar Productions
- Example Grammar Productions for Calculator
- Calculator javacc Options
- Calculator Main Parsing Class
- Calculator Execution Log
- Evaluating Expressions
- Evaluating Expressions Continued
- Evaluating Expressions Log
- Error Recovery
- Log of Calculator with Error Recovery

# Overview

- A LL parser generator which produces recursive-descent parsers.
- Produces parsers in Java.
- Possible to vary lookahead: i.e., can produce  $LL(k)$  parsers for  $k > 1$ .
- Single specification file for both scanner and parser.
- CFG allows EBNF constructs (using RE quantifiers like `?`, `*` and `+`).
- Non-terminal return values can be assigned to a variable.
- Actions within grammar rules allow arbitrary Java code.
- AST building package using `jtree` preprocessor.

# Caveats

- Basic regular expressions with a somewhat unusual syntax (over-parenthesized, different syntax for character classes).
- Not many extensions to basic regular expressions (unlike `lex`).
- Not open-source (yet).
- Scattered documentation. See

<http://www.experimentalstuff.com/Technologies/JavaCC/docindex.html>.

# Grammar File Syntax

```
javacc_input ::= javacc_options
               "PARSER_BEGIN" "(" <IDENTIFIER> ")"
               java_compilation_unit
               "PARSER_END" "(" <IDENTIFIER> ")"
               ( production )*
               <EOF>
```

4 kinds of productions:

## Regular Expression Production

A specification for one or more tokens.

## BNF Production

A definition for a grammar non-terminal.

## Javacode Production

Arbitrary Java code which may be called on the RHS of a grammar rule if the lookahead is non-ambiguous.

## Token Manager Declarations

A set of Java declarations starting with `TOKEN_MGR_DECLS`: which are copied into generated scanner.

# Regular Expression Productions

- Allows lexical states like `lex`'s start states.
- Disposition of token controlled by `regexpr_kind`:

## TOKEN

A token to be delivered to the parser (like a reserved word or number).

## SKIP

A token to be ignored (like whitespace or comments).

## MORE

Do not deliver a token to the parser. Instead, concatenate this token with the next one. Like `yymore()` in `lex`.

## SPECIAL\_TOKEN

A token not delivered to the parser directly but can be accessed from the parser. Can be useful for handling comments when

building tools like syntax-directed editors.

- Can have a lexical action associated with each token.

# Example Regular Expression Productions

SKIP :

```
{  
    " "  
    |  "\t"  
}
```

TOKEN:

```
{  
    < NUMBER: (<DIGIT>)+ ( "." (<DIGIT>)+ )? >  
    |  < #DIGIT: [ "0"-"9" ] >  
}
```

# Tokens

Some of the fields in a token:

`int kind`

Number associated with token. Referred to symbolically by a constant which is generated in a `*Constants.java` file.

`String image`

The token lexeme.

`int beginLine, beginColumn, endLine, endColumn`

These indicate the beginning and ending positions of the token as it appeared in the input stream.

The scanner implements a `TokenManager` interface with a method `Token getNextToken()` throws `ParseError`.

# Grammar Productions

- Each production looks similar to a Java function named the same as the LHS non-terminal, with a return value and parameter list.
- The parameter list is followed by a { } enclosed block which contains Java code for setting up the parsing of that non-terminal. Typically used for declaring local variables.
- The first { } block is followed by a second { } block which contain all the rules for the LHS non-terminal.
- Terminals are enclosed within < > or " " (the latter result in corresponding patterns being added to the scanner).
- Nonterminals look like function calls.
- Regular expression quantifiers \*, + and ? are available, with ( ' )' used for grouping.

- Actions enclosed within { } can contain arbitrary Java code.

# Example Grammar Productions for Calculator

```
void program():
{
}
{
    ( expr() "\n" ) *
}

void expr():
{
}
{
    term() ( LOOKAHEAD(2) "+" term() | "-" term() ) *
}

void term():
{
}
{
    unary() ( LOOKAHEAD(2) "*" unary() | "/" unary() ) *
}

void unary():
{
}
{
    "-" element() | element()
}

void element():
{
```

```
}  
{  
  <NUMBER> | "(" expr() "  
}
```

# Calculator javacc Options

```
options {
    LOOKAHEAD = 1;
    CHOICE_AMBIGUITY_CHECK = 2;
    OTHER_AMBIGUITY_CHECK = 1;
    STATIC = true;
    DEBUG_PARSER = false;
    DEBUG_LOOKAHEAD = false;
    DEBUG_TOKEN_MANAGER = false;
    ERROR_REPORTING = true;
    JAVA_UNICODE_ESCAPE = false;
    UNICODE_INPUT = false;
    IGNORE_CASE = true;
    USER_TOKEN_MANAGER = false;
    USER_CHAR_STREAM = false;
    BUILD_PARSER = true;
    BUILD_TOKEN_MANAGER = true;
    SANITY_CHECK = true;
    FORCE_LA_CHECK = false;
}
```

# Calculator Main Parsing Class

Source in `./programs/calc1/src/calc1.jj`. Generated files in `./programs/calc1/build/java/edu/binghamton/cs572s03/calc1/`.

```
PARSER_BEGIN(Calcl)

package edu.binghamton.cs572s03.calc1;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

public class Calcl
{

    static public void main(String args[]) {
        if (args.length != 1) {
            System.err.println("usage: java ClassName SOURCE_FILE");
            System.exit(1);
        }
        try {
            InputStream in = (args[0].equals("-"))
                ? System.in
                : new BufferedInputStream(new FileInputStream(args[0]));
            Calcl parser = new Calcl(in);
            parser.program();
        }
        catch (FileNotFoundException e) {
            System.err.println(e); System.exit(1);
        }
        catch (ParseException e) {
            System.err.println(e); System.exit(1);
        }
    }
}

PARSER_END(Calcl)
```

# Calculator Execution Log

```
[umrigar@bing_sun2 calc1]$ java -classpath calc1.jar \
    edu.binghamton.cs572s03.calc1.Calc1 -
1 + 2 * 3
22.0 +3.0/-6
22.0 +3.0/--6
edu.binghamton.cs572s03.calc1.ParseException:
    Encountered "-" at line 3, column 12.
Was expecting one of:
    <NUMBER> ...
    "(" ...

[umrigar@bing_sun2 calc1]$
```

# Evaluating Expressions

Source in ./programs/calc2/src/calc2.jj. Generated files in

./programs/calc2/build/java/edu/binghamton/cs572s03/calc2/.

```
void program():
{
    double v;
}
{
    ( v = expr()
      { System.out.println(v); System.out.flush(); }
      "\n"
    )*
}

double expr():
{
    double v, a;
}
{
    v = term()
    ( LOOKAHEAD(2) "+" a = term() { v += a; }
      | "-" a = term() { v -= a; }
    )*
    { return v; }
}
```

# Evaluating Expressions Continued

Source in `./programs/calc2/src/calc2.jj`. Generated files in `./programs/calc2/build/java/edu/binghamton/cs572s03/calc2/`.

```
double term():
{
    double v, a;
}
{
    v = unary()
    ( LOOKAHEAD(2) "*" a = unary() { v *= a; }
    | "/" a = unary() { v /= a; }
    )*
    { return v; }
}

double unary():
{
    double v, a;
}
{
    (
        "-" a = element() { v = - a; }
    | v = element()
    )
    { return v; }
}
```

```
double element():
{
    double v;
    Token t;
}
{
    (
        t = <NUMBER> { v = Double.parseDouble(t.image); }
    | "(" v = expr() ")"
    )
    { return v; }
}
```

# Evaluating Expressions Log

```
[umrigar@bingusun2 calc2]$ java -classpath calc2.jar \  
                                edu.binghamton.cs572s03.calc2.Calc2 -
```

```
1 + 2*3
```

```
7.0
```

```
1 + 2 - 3 + 2*8 - 4/-3
```

```
17.333333333333332
```

```
1 + 2 - 3 + 2*8 - 4/--3
```

```
edu.binghamton.cs572s03.calc2.ParseException:
```

```
    Encountered "-" at line 3, column 22.
```

```
Was expecting one of:
```

```
    <NUMBER> ...
```

```
    "(" ...
```

```
[umrigar@bingusun2 calc2]$ echo $?
```

```
1
```

```
[umrigar@bingusun2 calc2]$
```

# Error Recovery

Source in ./programs/calc3/src/calc3.jj. Generated files in ./programs/calc3/build/java/edu/binghamton/cs572s03/calc3/.

```
void program():
{
    double v;
}
{
    ( try {
        v = expr() { System.out.println(v); System.out.flush(); } <NL>
    }
    catch (ParseException e) {
        errorSkipTo(NL);
    }
    )*
}

...
```

JAVACODE

```
void errorSkipTo(int kind) {
    ParseException e = generateParseException();
    System.err.println(e.toString()); // print the error message
    Token t;
    do {
        t = getNextToken();
    } while (t.kind != kind);
}
```

# Log of Calculator with Error Recovery

```
[umrigar@bingusun3 calc3]$ java -classpath calc3.jar \
    edu.binghamton.cs572s03.calc3.Calc3 -
1 + 2*3
7.0
1 + 2 - 3 + 2*8 - 4/-3
17.333333333333332
1 + 2 - 3 + 2*8 - 4/--3
edu.binghamton.cs572s03.calc3.ParseException:
  Encountered "-" at line 3, column 22.
Was expecting one of:
    <NUMBER> ...
    "(" ...

55.0/11.0 + 4
9.0
[umrigar@bingusun2 calc3]$
```