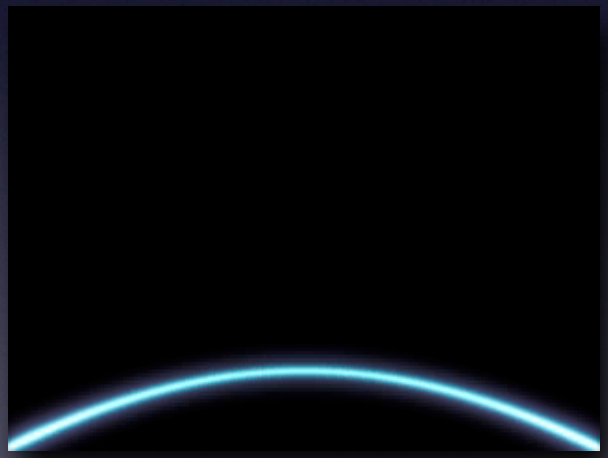
JPS cwiczenia 9.

Lekser i parser



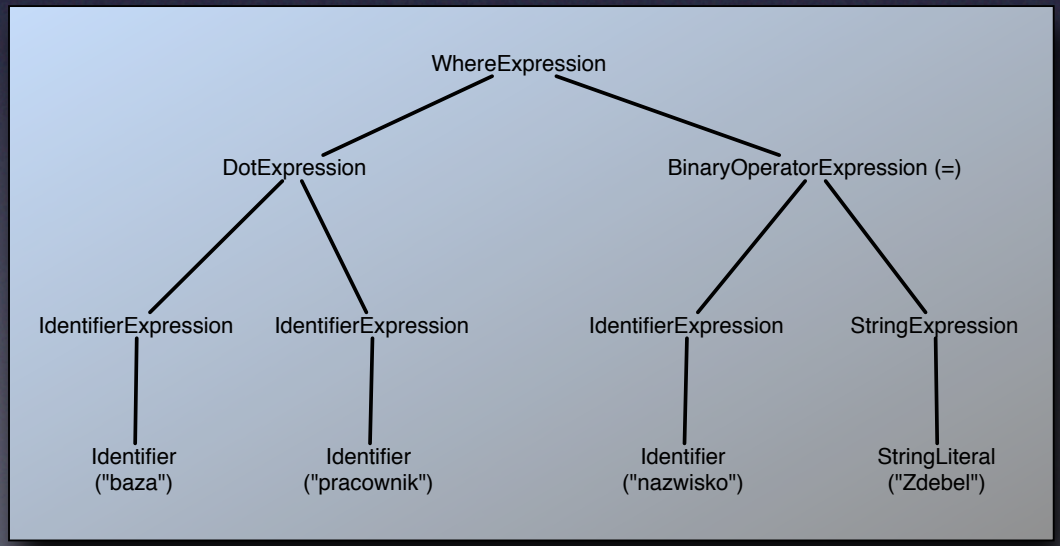
Polsko-Japońska Wyższa Szkoła Technik Komputerowych

Cel i funkcje

baza.pracownik where nazwisko = “Zdebel”

baza . pracownik where nazwisko = “Zdebel”

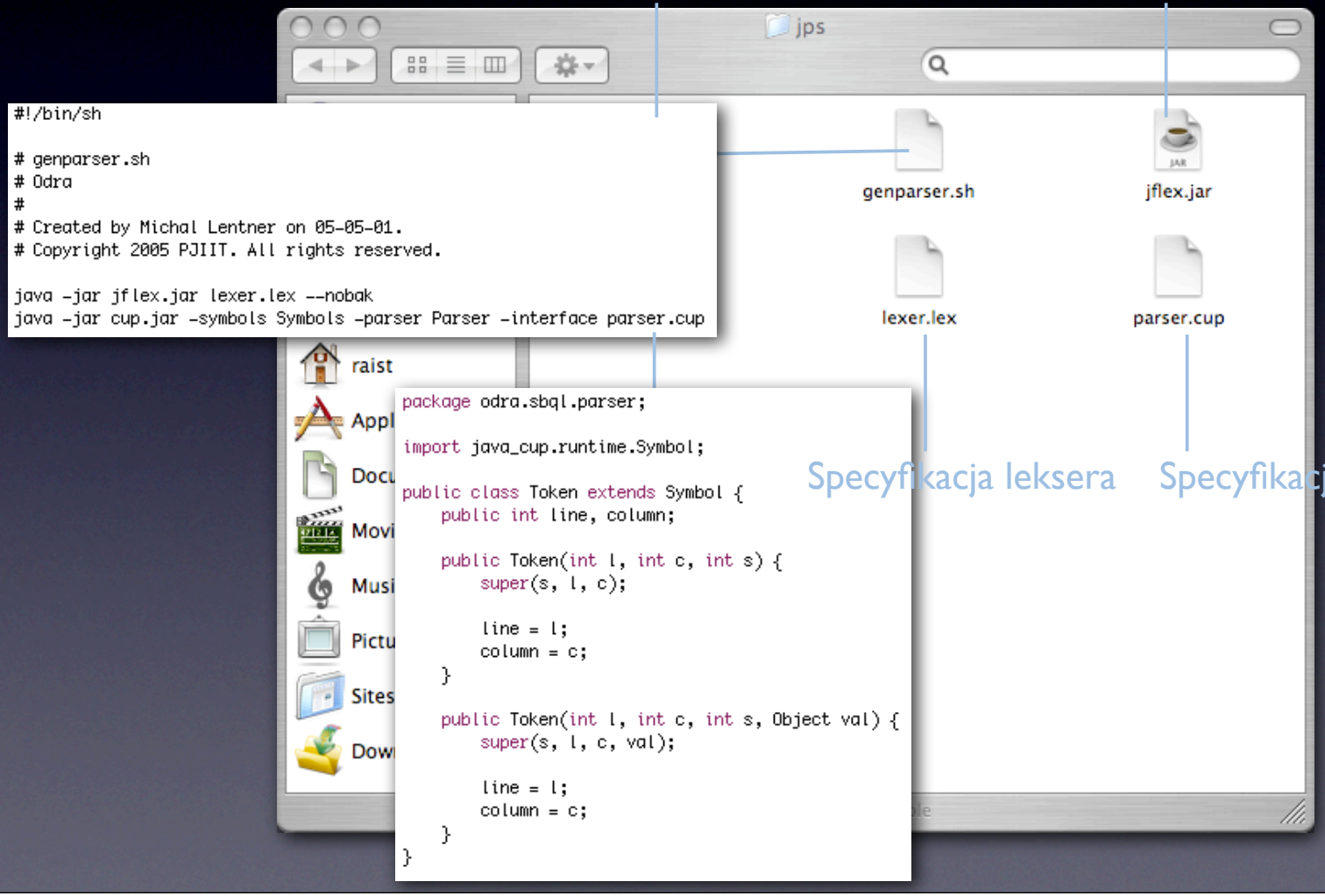
```
program  ::=
          wyrazenie ;
        | instrukcja ;
        ;

wyrazenie ::=
          wyrazenie . wyrazenie
        | wyrazenie where wyrazenie
        | wyrazenie = wyrazenie
        | wyrazenie + wyrazenie
        | identyfikator
        | literal
        | ( wyrazenie )
      ...
```



Potrzebne pliki

Program generatora parserów

Program generatora lekserów




```
#!/bin/sh

# genparser.sh
# Odra
#
# Created by Michal Lentner on 05-05-01.
# Copyright 2005 PJIIT. All rights reserved.

java -jar jflex.jar lexer.lex --nobak
java -jar cup.jar -symbols Symbols -parser Parser -interface parser.cup
```

```
package odra.sbql.parser;

import java_cup.runtime.Symbol;

public class Token extends Symbol {
    public int line, column;

    public Token(int l, int c, int s) {
        super(s, l, c);

        line = l;
        column = c;
    }

    public Token(int l, int c, int s, Object val) {
        super(s, l, c, val);

        line = l;
        column = c;
    }
}
```

Specyfikacja leksera

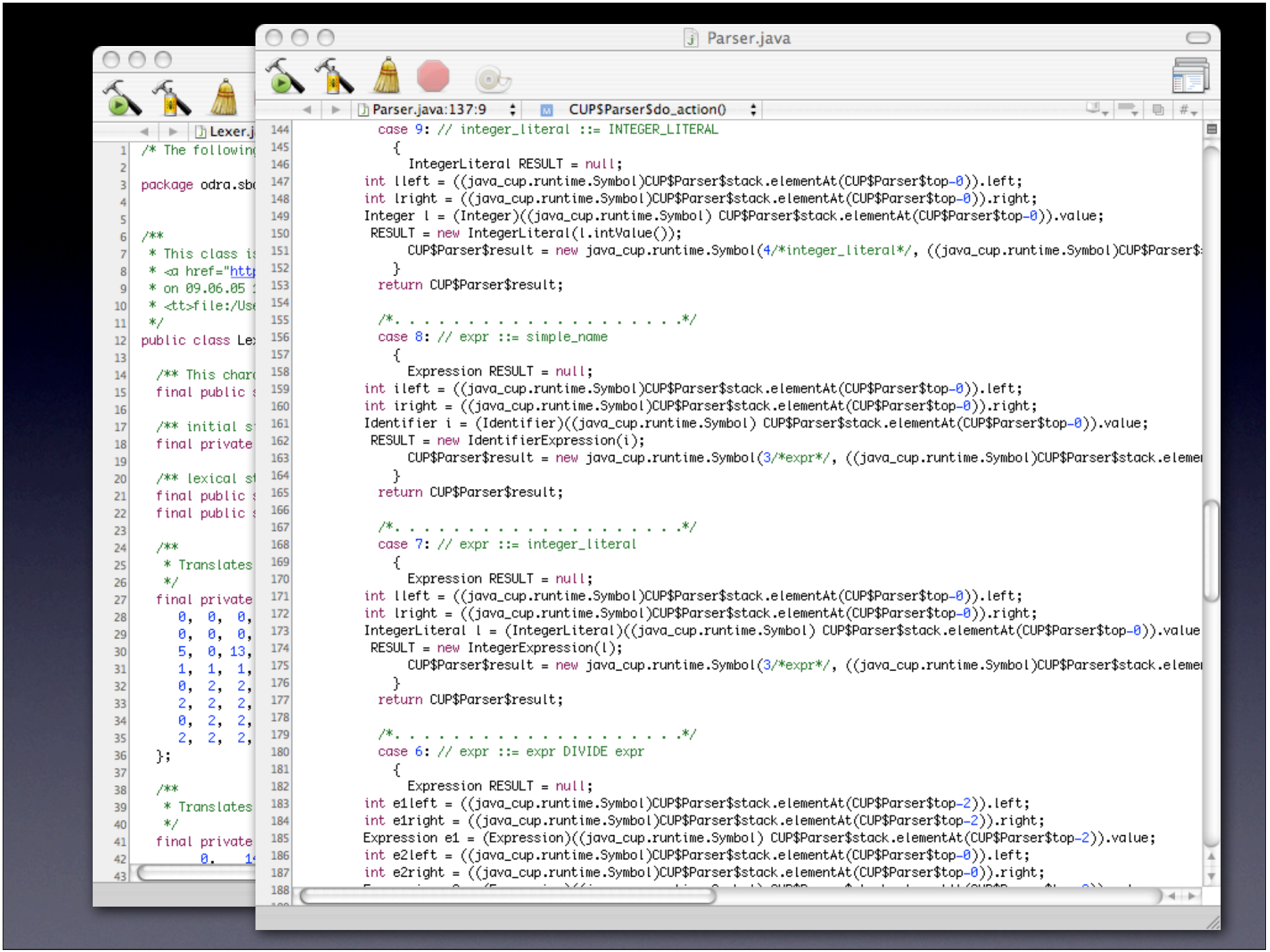
Specyfikacja parsera

parser.cup

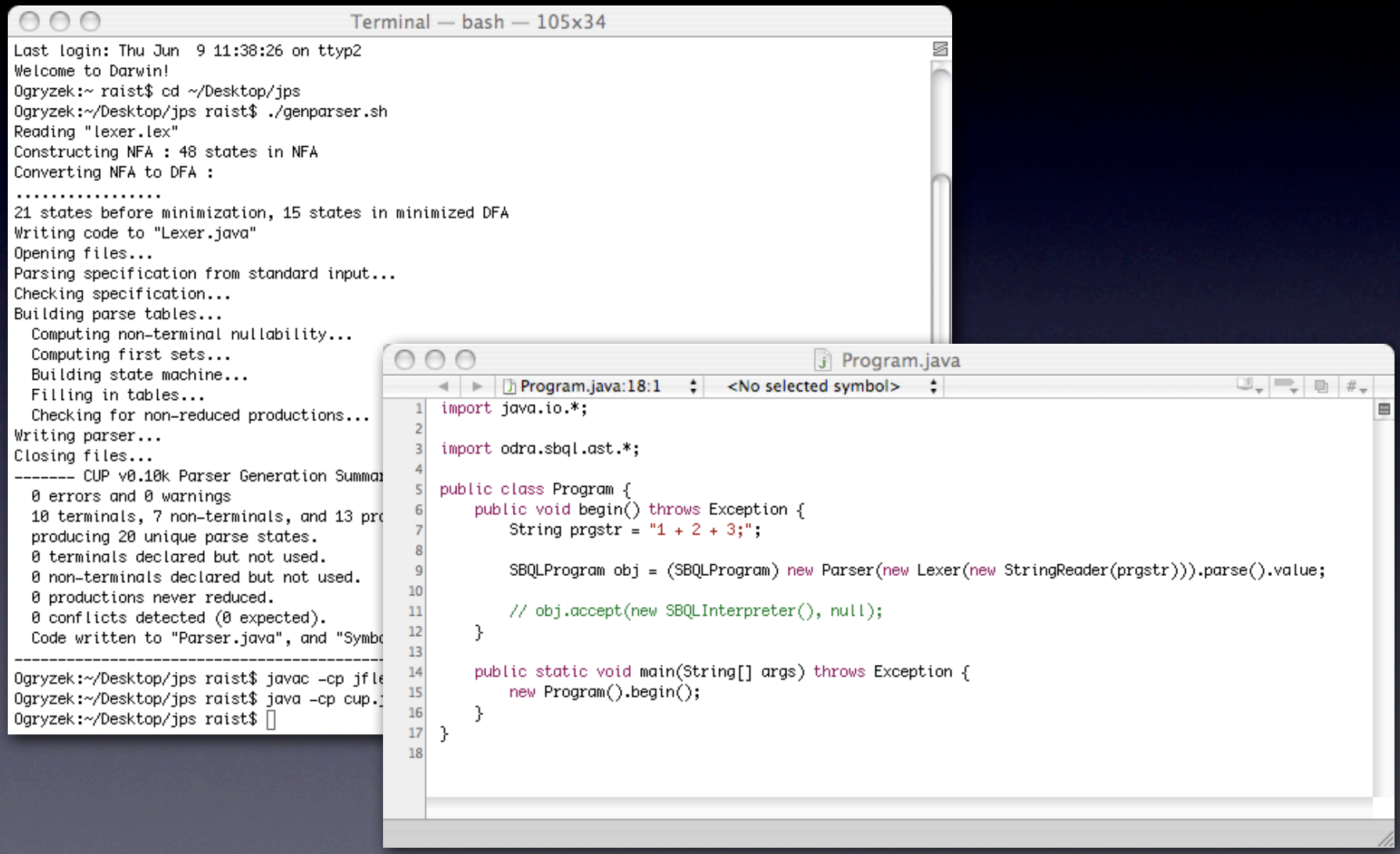
```
import java_cup.runtime.Symbol;

import odra.sbql.ast.*;
import odra.sbql.ast.expressions.*;
import odra.sbql.ast.statements.*;
import odra.sbql.ast.terminals.*;

terminal Integer        INTEGER_LITERAL;
terminal String         STRING_LITERAL;
terminal String         IDENTIFIER;

terminal                SEMICOLON;
terminal                PLUS;
terminal                MINUS;
terminal                TIMES;
terminal                DIVIDE;

non terminal SBQLProgram                goal;
non terminal ExpressionStatement        expr_stmt;
non terminal Expression                 expr;
non terminal IntegerLiteral             integer_literal;
non terminal StringLiteral              string_literal;
non terminal Identifier                 simple_name;

precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE;

start with goal;

goal                                    ::=
                                                expr_stmt:s             {: RESULT = new SBQLProgram(s); :}
                                        ;

expr_stmt                               ::=
                                                expr:e SEMICOLON        {: RESULT = new ExpressionStatement(e); :}
                                        ;

expr                                    ::=
                                                expr:e1 PLUS expr:e2    {: RESULT = new BinaryExpression(e1, e2, Operator.opPlus); :}
                                        |       expr:e1 MINUS expr:e2   {: RESULT = new BinaryExpression(e1, e2, Operator.opMinus); :}
                                        |       expr:e1 TIMES expr:e2   {: RESULT = new BinaryExpression(e1, e2, Operator.opMultiply); :}
                                        |       expr:e1 DIVIDE expr:e2  {: RESULT = new BinaryExpression(e1, e2, Operator.opDivide); :}
                                        |       integer_literal:l       {: RESULT = new IntegerExpression(l); :}
                                        |       string_literal:l        {: RESULT = new StringExpression(l); :}
                                        |       simple_name:i           {: RESULT = new IdentifierExpression(i); :}
                                        ;

integer_literal                         ::=
                                                INTEGER_LITERAL:l       {: RESULT = new IntegerLiteral(l.intValue()); :}
                                        ;

string_literal                          ::=
                                                STRING_LITERAL:l        {: RESULT = new StringLiteral(l); :}
                                        ;

simple_name                             ::=
                                                IDENTIFIER:i            {: RESULT = new Identifier(i); :}
                                        ;
```

lexer.lex

```
package odra.sbql.parser;

%%
%{
	private StringBuffer str;
%}

%public
%class Lexer
%cup
%line
%column
%state STRING
%eofval{
	return new Token(yyline, yycolumn, Symbols.EOF);
%eofval}

INTEGER		= [0-9]+
IDENTIFIER	= [a-zA-Z_][a-zA-Z_|0-9]*
TEXT		= [^\r\n\"\\]
WHITESPACE	= {LINETERM} | [ \t\r\f]
LINETERM	= \r|\n|\r\n

%%

<YYINITIAL> {
	";"	{ return new Token(yyline, yycolumn, Symbols.SEMICOLON); }
	"+"	{ return new Token(yyline, yycolumn, Symbols.PLUS); }
	"-"	{ return new Token(yyline, yycolumn, Symbols.HYPHEN); }
	"*"	{ return new Token(yyline, yycolumn, Symbols.TIMES); }
	"/"	{ return new Token(yyline, yycolumn, Symbols.DIVIDE); }

	{INTEGER} {
			int val;
			try {
				val = Integer.parseInt(yytext());
			}
			catch (Exception e) {
				throw new Exception(e.getMessage());
			}
			return new Token(yyline, yycolumn, Symbols.INTEGER_LITERAL, new Integer(val));
		}

	{IDENTIFIER}	{
				return new Token(yyline, yycolumn, Symbols.IDENTIFIER, yytext());
			}

	\"		{ str = new StringBuffer(); yybegin(STRING); break; }

	{WHITESPACE}	{ break; }

	.		{ throw new Exception("Illegal character: " + yytext(), yyline + 1, yycolumn + 1); }
}

<STRING> {
	\"		{ yybegin(YYINITIAL); return new Token(yyline, yycolumn, Symbols.STRING_LITERAL, str.toString()); }
	{TEXT}		{ str.append(yytext()); break; }
}
```


Parser.java
Parser.java:137:9
CUP$Parser$do_action()
Lexer.j
/* The following
package odra.sb
/**
* This class is
* on 09.06.05
public class Le
/** This char
final public
/** initial s
final private
/** lexical s
final public
final public
/**
* Translates
final private
0, 0, 0,
0, 0, 0,
5, 0, 13,
1, 1, 1,
0, 2, 2,
2, 2, 2,
0, 2, 2,
2, 2, 2,
};
/**
* Translates
final private
0, 1
case 9: // integer_literal ::= INTEGER_LITERAL
{
IntegerLiteral RESULT = null;
int lleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
int lright = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
Integer l = (Integer)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value;
RESULT = new IntegerLiteral(l.intValue());
CUP$Parser$result = new java_cup.runtime.Symbol(4/*integer_literal*/, ((java_cup.runtime.Symbol)CUP$Parser$
}
return CUP$Parser$result;
/*. . . . . .*/
case 8: // expr ::= simple_name
{
Expression RESULT = null;
int ileft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
int iright = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
Identifier i = (Identifier)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value;
RESULT = new IdentifierExpression(i);
CUP$Parser$result = new java_cup.runtime.Symbol(3/*expr*/, ((java_cup.runtime.Symbol)CUP$Parser$stack.eleme
}
return CUP$Parser$result;
/*. . . . . .*/
case 7: // expr ::= integer_literal
{
Expression RESULT = null;
int lleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
int lright = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;
IntegerLiteral l = (IntegerLiteral)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-0)).value
RESULT = new IntegerExpression(l);
CUP$Parser$result = new java_cup.runtime.Symbol(3/*expr*/, ((java_cup.runtime.Symbol)CUP$Parser$stack.eleme
}
return CUP$Parser$result;
/*. . . . . .*/
case 6: // expr ::= expr DIVIDE expr
{
Expression RESULT = null;
int e1left = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
int e1right = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
Expression e1 = (Expression)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
int e2left = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).left;
int e2right = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-0)).right;

Jak z tego korzystać ?



Ćwiczenia