```
/*
 * A VERY minimal skeleton for your parser, provided by Emma Norling.
 *
 * Your parser should use the tokens provided by your lexer in rules.
 * Even if your lexer appeared to be working perfectly for stage 1,
 * you might need to adjust some of those rules when you implement
 * your parser.
 *
 * Remember to provide documentation too (including replacing this
 * documentation).
 *
 */
parser grammar DecafParser;
options { tokenVocab = DecafLexer; }

// This rule says that a program consists of the tokens CLASS ID LCURLY RCURLY
    EOF nothing more nothing less,
// in exactly that order. However obviously something (quite a lot of something)
     needs to go between the curly
// brackets. You need to write the rules (based on the provided grammar) to
    capture this.
program: CLASS ID LCURLY (field_decl)* (method_decl)* RCURLY EOF;

field_name: ID | (ID LSQUARE NUMBER RSQUARE);
field_decl: type field_name (COMMA field_name)* END;

method_decl: (meth_type) meth_name LBRACE ((arg_type ID COMMA)* arg_type ID)?
    RBRACE block;
meth_name: ID;
meth_type: type | VOID;
arg_type: type;

block: LCURLY (var_decl)* (statement)* RCURLY;

var_decl: type var_name (COMMA var_name)* END;
var_name: ID;

type: INT | BOOLEAN;

statement: location assign_op expr END # Assign
| method_call END # MC
| IF LBRACE expr RBRACE block (ELSE block)? # If
| FOR ID ASSIGN expr COMMA expr block # For
| RETURN expr? END # Return
| BREAK END # Break
| CONTINUE END # Continue
| block # Bl;

assign_op: ASSIGN | math_assign;
math_assign: PLUSASSIGN | MINUSASSIGN;

method_call: method_name LBRACE ((expr COMMA)* expr)?  RBRACE | CALLOUT LBRACE
    STRING (COMMA callout_arg)* RBRACE;
method_name: ID;

location:  ID | ID LSQUARE expr RSQUARE;
```

```
// Could call the "bin_op" here, but have separated to ensure order of
    precedence is preserved
expr: MINUS expr
| NOT expr # Not
| expr (MULT | DIV | MOD) expr
| expr (PLUS | MINUS) expr
| expr (rel_op) expr
| expr (eq_op) expr
| expr (AND) expr
| expr (OR) expr
| location
| method_call
| literal
| LBRACE expr RBRACE;

callout_arg: expr | STRING;

bin_op: arith_op | rel_op | eq_op | cond_op;

arith_op: MULT | DIV | MOD | PLUS | MINUS;

rel_op: LT | LTE | GTE | GT;

eq_op: EQ | NEQ;

cond_op: AND | OR;

literal: NUMBER | CHAR | bool_literal;

bool_literal: TRUE | FALSE;
```