```
 1 parser grammar DecafParser;
 2 options { tokenVocab = DecafLexer; }
 3
 4 program: CLASS ID LCURLY field_decl* method_decl* RCURLY EOF;
 5 field_decl: type field_name (COMMA field_name)* SEMICOLON;
 6 field_name: ID | (ID LSQUARE INT_LITERAL RSQUARE);
 7 type: INT | BOOLEAN;
 8 method_decl: method_header block;
 9 method_header: (type | VOID) ID (LPAREN params? RPAREN);
10 params: param (COMMA param)*;
11 param: type param_name;
12 param_name: ID;
13 block: LCURLY var_decl* statement* RCURLY;
14 var_decl: type param_name (COMMA param_name)* SEMICOLON;
15 statement: location assign_op expr SEMICOLON
16     | method_call SEMICOLON
17     | IF LPAREN expr RPAREN block (ELSE block)?
18     | FOR ID ASSIGNMENT expr COMMA expr block
19     | RETURN (expr)? SEMICOLON
20     | BREAK SEMICOLON
21     | CONTINUE SEMICOLON
22     | block
23 ;
24 assign_op: ASSIGNMENT
25     | INCREMENT
26     | DECREMENT
27 ;
28 method_call: method_name LPAREN (expr (COMMA expr)*)? RPAREN
29     | CALLOUT LPAREN STRING_LITERAL (COMMA callout_arg (COMMA
   callout_arg)*)? RPAREN
30 ;
31 method_name: ID;
32 location: ID
33     | ID LSQUARE expr RSQUARE
34 ;
35 expr: location
36     | method_call
37     | literal
38     | SUBTRACT expr
39     | NOT expr
40     | expr mul_div_mod_op expr
41     | expr add_sub_op expr
42     | expr rel_op expr
43     | expr eq_op expr
44     | expr cond_op expr
45     | LPAREN expr RPAREN
46 ;
```

```
47 callout_arg: expr | STRING_LITERAL;
48 mul_div_mod_op: MULTIPLY | DIVIDE | MOD;
49 add_sub_op: ADD | SUBTRACT;
50 rel_op: LTHAN | GTHAN | LTHANEQUAL | GTHANEQUAL;
51 eq_op: EQUALITY | NEQUAL;
52 cond_op: (AND) | OR;
53 literal: INT_LITERAL | CHAR_LITERAL | bool_literal;
54 bool_literal: TRUE | FALSE;
```