Gramática

Keywords

```
class  struct true   false  void   if     else
while  return int    char   boolean
```

```
program           ::= 'class' 'Program' '{' {declaration} '}'
declaration       ::= varDeclaration
                    | structDeclaration
                    | methodDeclaration
varDeclaration    ::= varType ident '[' numericLit ']' ';'
                    | varType ident ';'
                    | structVarDecl
                    | structDeclaration ident ';'
                    | structDeclaration ident '[' numericLit ']'
                      ';'
structVarDecl     ::= 'struct' ident ident ';'
structDeclaration ::= 'struct' ident {varDeclarations}
varType           ::= primitiveType
                    | 'void'
primitiveType     ::= 'int'
                    | 'char'
                    | 'boolean'
methodDeclaration ::= varType ident parameterList block
parameterList     ::= '(' parameter [ ',' parameter ] ')'
parameter         ::= primitiveType ident
                                    | primitiveType ident '[' ']'
block             ::= '{' {varDeclaration} '}'
statement         ::= ifStatement
                    | whileStatement
                    | returnStatement
                    | methodCall ';'
                    | block
                    | assignment
                    | expression ';'
ifStatement       ::= 'if' '(' expression ')' block 'else' block
                    | 'if' '(' expression ')' block
whileStatement    ::= 'while' '(' expression ')' block
```

```
returnStatement   ::= 'return' [expression] ';'
methodCall        ::= ident '(' arguments ')'
arguments         ::= expression [ ',' expression ]
assignment        ::= location '=' expression
expression        ::= expressionOp ['&&' | '||' expression ]
expressionOp      ::= expressionSum [valueComparators
   expressionSum]
valueComparators  ::= '<='|'<'|'>'|'>='|'=='|'!='
expressionSum     ::= expressionMult [ '+'|'-' expressionSum ]
expressionMult    ::= unaryOpExpression ['/'|'*','%'
   expressionMult]
unaryOpExpression ::= '-' simpleExpression
                    | '!' simpleExpression
                    | simpleExpression
simpleExpression  ::= literal
                    | '(' expression ')'
                    | methodCall
                    | location
literal           ::= numericLit
                    | charLit
                    | 'false'
                    | 'true'
location          ::= ident '[' expression ']' optLocation
                    | ident optLocation
optLocation       ::= ['.' location]
```

Type System

```
program ::= 'class' 'Program' '{' {declaration} '}'
```