21AIE112 Elements of Computing Systems-2

END PROJECT REPORT

SYNTAX ANALYZER

Submitted By:

ANNAPOORNA A K

(AM.EN.U4AIE21114)

AMRITA VISHWA VIDYAPEETHAM

DEEMED TO BE UNIVERSITY

INTRODUCTION

This project is based on Syntax Analyzer. It is a part of the compiler which converts Jack language into VM code. A Syntax Analyzer can parse Jack programmes in accordance with the Jack language and generate an XML(extensible markup language) file that uses marked-up text to display the program's structure. This syntax analyser which we have done has three components , a Jack Tokenizer, a Compilation Engine and a Jack Analyzer class which is the top-most module.

Jack Tokenizer converts the jack program into a list of tokens while the Compilation Engine or parser verifies that the string can be the grammar for the source language. It detects and reports any syntax errors and produces a parse tree from which intermediate code can be generated. Later on, the code generator can be used to generate VM codes which haven’t been done yet in this project.

PACKAGES, CLASSES AND FUNCTIONS USED

Here, we have two packages: syntax_analyzer package with three classes: JackAnalyzer.java, JackTokenizer.java and CompiationEngine.java and util package with two classes Keyword and TokenType.

syntax_analyzer

I. **JackAnalyzer.java:** is the top-most module/ main module. The main function of this class invokes the Tokenizer and Compilation Engine classes after receiving the file path to the jack file or the directory containing the jack files.

II. **JackTokenizer.java** :

1. **JackTokenizer(Path path):** is a constructor where the single-line and block comments are removed and then the tokens are matched with the regex of keywords, symbols, integers, strings, and identifiers before being added to an array list of tokens.
2. **reset() :** is used to reset tokenizer to beginning of the file.
3. **tokenIndex():** is the method which is used to return index of the current token.
4. **hasMoreTokens():** used to check if there are any tokens left. It returns true if the index of the current token is less than size of tokens - 1.
5. **advance():** helps to advance to the next token,if there are more tokens, set currToken to next token. Set token to the token's value first, and then set tokenType to the token's type (keyword, symbol, int, string, identifier).
6. **previousToken():** used to return to the previous token. Set token to the token's value first, and then set tokenType to the token's type (keyword, symbol, int, string, identifier) just as given above.
7. **tokenType():** used to return the type of the token.
8. **keyword() :**is used to return the keyword value if the passed token is a KEYWORD. This checking is done by using the keyword enum. . Returns the statement "NOT A KEYWORD" otherwise.
9. **symbol():** is used to return the symbol if the passed token is of a SYMBOL type. . Returns the statement "NOTA SYMBOL!" otherwise.
10. **identifier():** is used to return the identifier if the passed token is of an IDENTIFIER type. Returns the statement "NOT AN IDENTIFIER!" otherwise.

11. **intVal():**is used to return the integer value of the token of type String if the passed token is an INT_CONST. It returns -1 if it's not an integer constant.
12. **stringVal:** is used to return the string value of the token if the token is of the type STRING_CONST. Returns the statement "NOT A STRINGVAL!" otherwise.
13. **removeBlockComments(String s):** is used to return the file after eliminating the input file's block comments.

III. **CompilationEngine.java:** The parser or compilation engine analyzes to see if the string matches the grammar of the source language.

1. **compilationEngine (JackTokenizer t, FileOutputStream o, FileOutputStream ot):** It is the function that will invoke the tokenizer class, generate a single-token XML file, and start the XML file writer.
2. **writeError(String parameter):** used to write error, if any, to the output file.
3. **nextIs (String tokenType, String tokenValue):** is used to check if the next token in the file is of the passed token type.
4. **compileClass():** is used to compile a complete class. It starts with the class tag and ends with the ending tag.
5. **compileClassVarDec():** is used to compile the variables declared after the class.
6. **compileSubroutine():** is used to compile the complete method, function or the constructor and also the subroutine's body.
7. **compileParameterList():** is used to compile a parameter list. It does not handle the normal parenthesis '**(**' , '**)**'.
8. **compileVarDec():** is used to compile variable declarations by looking at the 'var' keyword and writes the XML code under the varDec start and end tags.
9. **compileStatements():** is used to compile a sequence of statements. It identifies which kind of statement is, then calls the respective functions which are given below and writes into the output file. Does not handle the '**{**' , '**}**' parenthesis.
10. **compileDo():** is used to compile a do statement by searching for the keyword '**do**' and then writes it into the output file. It is used to call other functions.
11. **compileLet():** is used to compile a let statement by searching for the keyword '**let**' and then writes the XML code into the output file. letStatements are used to assign a value to a variable.

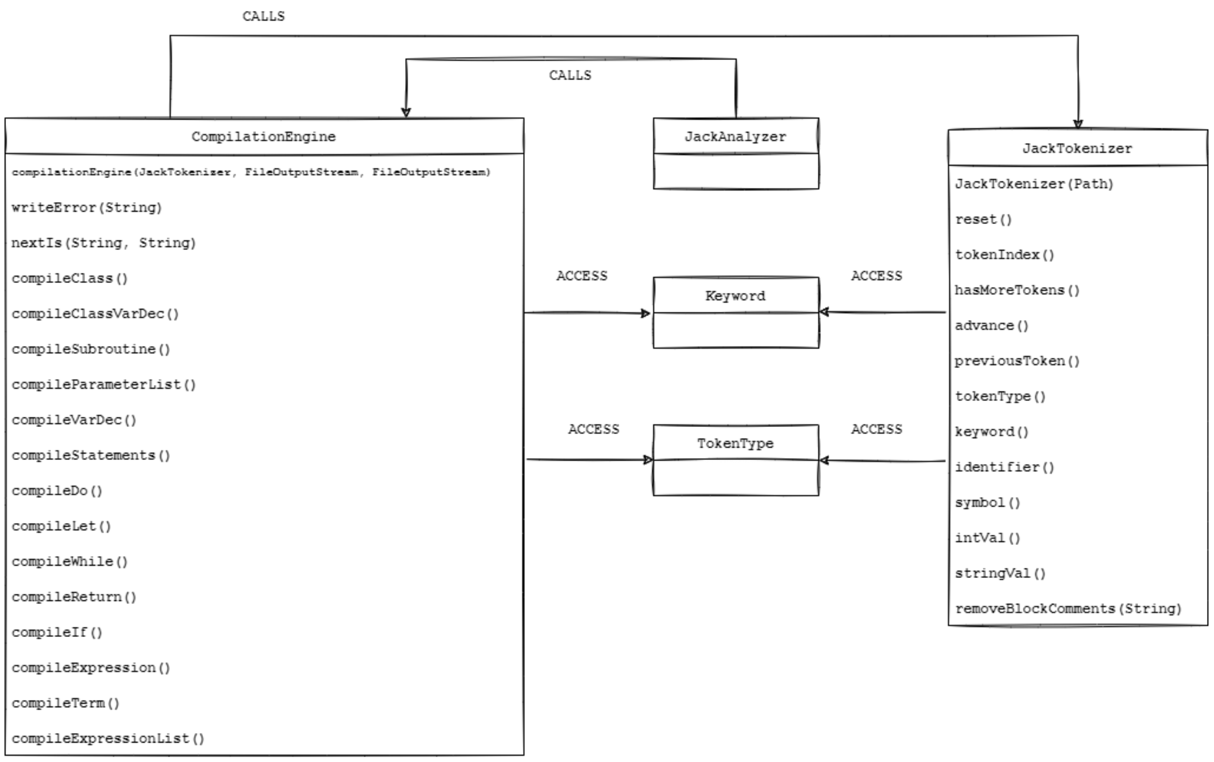
12. **compileWhile():** is used to compile a while statement by searching for the '**while'** keyword and then writes the 'whileStatement' in the XML file .whileStatement iis a while loop used to perform loops continuously until when a condition is met.
13. **compileReturn():** is used to compile a return statement. It writes the return statement to the XML file with the keyword '**return**' and then the return value if any and the ';' symbol.
14. **compileIf():** is used to compile if statements followed by the else statements, if present. It also handles the statements within the if-else ladder (or the if statement).
15. **compileExpression():** is used to compile expressions. Expressions include mathematical operations, constants, and many more. It also compiles all the symbols, identifiers etc that are present in the expression.
16. **compileTerm()**: is used to compile a term. Terms can be a combination of different or just a single tag.
17. **compileExpressionList():** is used to compile a comma-separated list of tokens. For each it expression, it calls the compileExpression() function.

util

I. **Keyword.java:** This class is an enum (class that represents a group of constants) for keywords in Jack Language.

II. **TokenType.java:** This class is the enum (class that represents a group of constants) for tokens in Jack Language.

CLASS DIAGRAM



COMMANDS

Call JackAnalyzer.main first and provide the .jack file path or the path of the directory in the code.