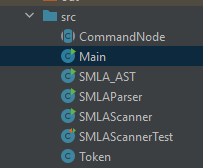
**A short explaining about program:**

1. **Common principle:**

* Scan: use scanning word by word
* Input: low or up character, having blank between word, number or operator. Each command on each line, don’t use semicolon “;” at the end of command
* Special character: “(”, “)”, “,” written as:

SETUP SIMULATION (Example1) WITH 4 AS (Group1, Group2, Group3, ABC)

1. **Constructor program:**

* CommandNode: consists of classes of commandNode using to create AST
* Main(): main function, when it is run the flow of stream:

Input data: from file input.txt → SMLAScanner class → SMLAParse class → SMLA\_AST class → output data: console, file output.txt only shows list tokens from scanner (don’t update)

Note: have to change folder address in computer to can see it on desktop.

* SMLA\_AST class: check and save tokens as AST

. input: from SMLAParser class

. output: node AST shown on console

* SMLAParser class: check grammatical commands

. input: token list from SMLAScanner class

. output: token list to SMLA\_AST class

* SMLAScanner: scan input data in string form (String, console or .txt file)

. input: from file input.txt on desktop

. output: token list to SMLAParser

* SMLAScanner: consists of unit test
* Token: main constructor of data

1. **Operation of classes:**
2. **Scanning class SMLAScanner:**

To scan a command using two functions:

* Specific functions for commands:

SETUP SIMULATION (Example1) WITH 4 AS (Group1, Group2, Group3, ABC):

. scanSetupRunReport(lineScanner, token): collect keyword ‘SETUP SIMULATION’

. scanNextToken(inputScanner, nextToken); scan the rest of command

WHERE PREF IS (10, 20, 30, 40):

. scanWhere(Scanner inputScanner, String token): collect keyword ‘WHERE PREF IS’

. scanNextToken(inputScanner, nextToken); scan the rest of command

WHERE VACANT IS Vacant:

. scanWhere(Scanner inputScanner, String token): collect keyword ‘WHERE VACANT IS’

. scanNextToken(inputScanner, nextToken); scan the rest of command

USING RANDOM MOVE:

. scanUsing(Scanner inputScanner, String token): collect keyword ‘USING RANDOM’

. scanNextToken(inputScanner, nextToken); scan the rest of command

RUN SIMULATION Example1 FOR 50 TICKS:

. scanSetupRunReport(lineScanner, token): collect keyword ‘RUN SIMULATION’

. scanNextToken(inputScanner, nextToken); scan the rest of command

REPORT SIMULATION Example1

. scanSetupRunReport(lineScanner, token): collect keyword ‘REPORT SIMULATION’

. scanNextToken(inputScanner, nextToken); scan the rest of command

X = 10 or Y = abcd…

.scanVariable(lineScanner, token); identify variable

. scanNextToken(inputScanner, nextToken); scan the rest of command

Note:

Apart from the two main functions groups, all the remaining functions are just subprograms of these two function groups

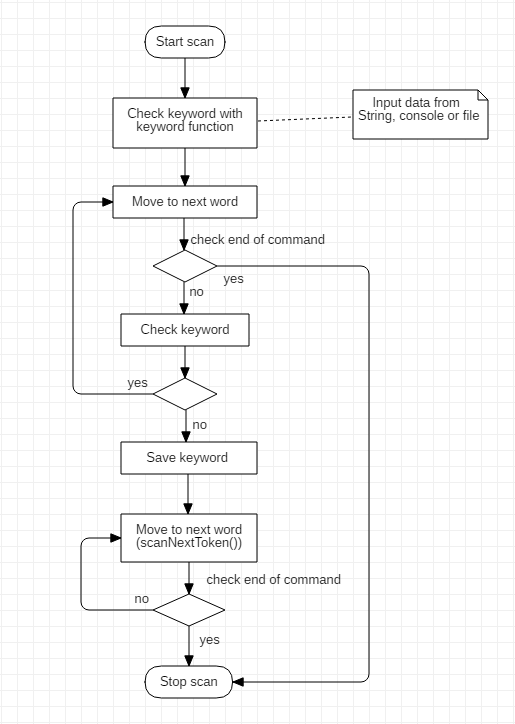
With this way there can add or modify keyword of each command but does not affect the rest of the program

**Scanner's mission:**

1. Scan, check and collect key word, if it is incorrect throwing exception
2. All type of the rest data, it only scans and adds type and value, saves to the list ‘tokenList’

(doesn’t check grammar)

1. Keyword is saved in one token, other words one word in one token. End of command is token (“end\_command”, “end of command”), it will be automatically added when starting the new line.
2. Check and delete “(”, “)”, “,” before save token
3. Scan process is described by figure below



1. **Parser class SMLAParser:**

**Parser's mission:**

1. Check command grammar based on each command
2. Process goes two steps:

. Check type of token by matchType(TokenType expectedType), if is correct going to step 2

. Check value of token, using functions:

- matchKeywordValue(TokenType expectedType)

- matchStringValue(TokenType expectedType)

- matchPhraseValue(TokenType expectedType)

- matchIntegerValue(TokenType expectedType)

- matchVariableValue(TokenType expectedType)

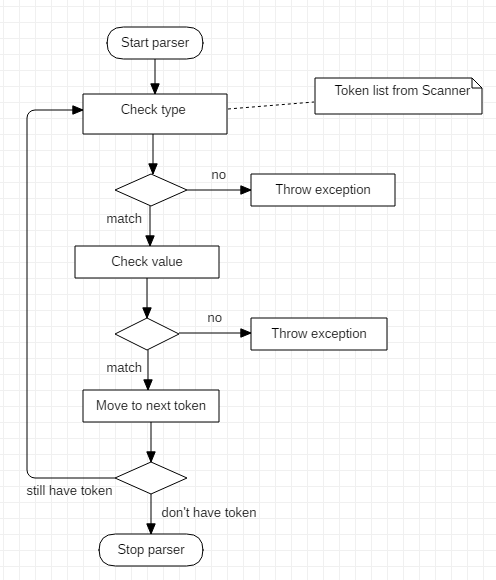
- matchOtherValue(TokenType expectedType)

Type and value of token will be compared with enum class TokenType

3. If all tokens match with type, value and grammar, they will be saved into token list ‘tokenList’ send to SMLA\_AST.

4. Parser process is described by figure below

Note: Parser doesn’t save token ‘end of command’



1. **AST class SMLA\_AST:**

**Parser's mission:**

1. Continue check type and value of tokens again but don’t check grammar
2. Save them to Node following each command
3. **Test directly from program:**

* There main() function to check each class separately, they are in each class: scanner, parser and AST.
* Public main() is used to check for whole process from read input file to output AST