Git : https://github.com/915-Narita-Andrei/Parser

Grammar

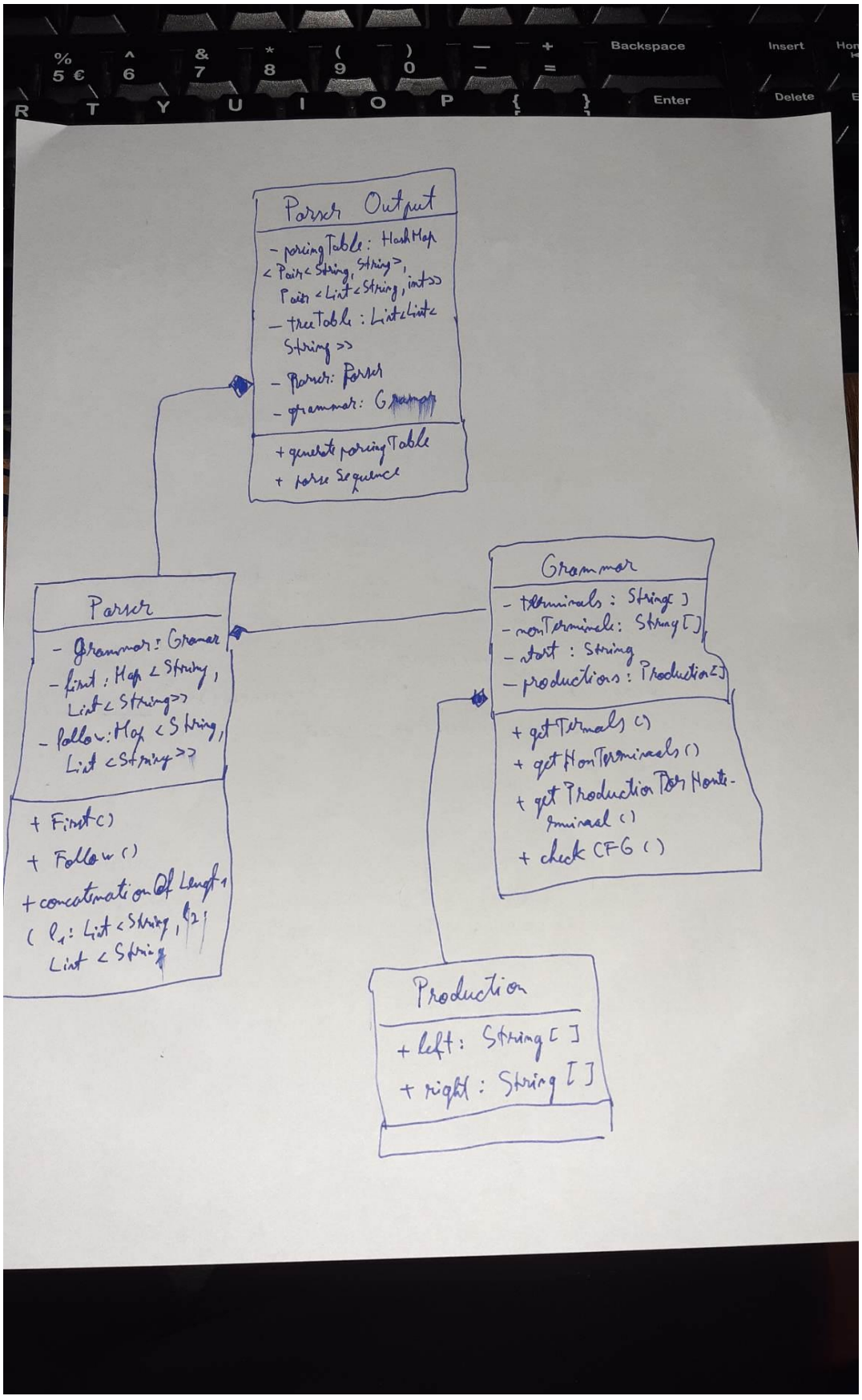
Grammar class has a 2 lists of strings representing the lists of terminals and nonterminals, a string representing the starting symbol and a list of productions. A production is just a class that has 2 lists of Strings representing the left and right part of a production. The grammar class is able to read a grammar, check if it is a CFG and it has some function in order to return some useful information about the grammar, like returning all the terminals or nonterminals or just return all productions that are containing a specific nonterminal.

Parser

Parser class has a grammar and 2 HashMaps representing the First and Follow sets. The Hash maps has as key a String and as a value a List of Strings. The Parser has the function to calculate the first and follow for a given grammar. Also in order to calculate these 2 sets it has an auxiliary method, concatenation of length 1 which receive 2 languages an return the concatenation of length one for them.

Parser Ouptut

Parser Output class has a Parser injected as a attribute. This class has the parsing table represented as a HashMap with the key being a Pair of String, String and a value being a Pair of a List of Strings and Integer. The key can hold a combination of 2 symbols from terminals and nonterminals sets and the value will hold all the prediction that was deduced and also the production index. This class can also print a nice parsing table after it was computed it has a function to parse a sequence and based on Parser class and parsing table it will tell if the sequence is accepted by the grammar or not and also find if there is any error and print it. While constructing the parsing table it will also generate the parsing tree and for this we used as a representation the table with father and siblings.


Parser Output
- parsingTable : HashMap < Pair< String, String >, Pair < List < String, int>>
- treeTable : List<List< String >>
- Parser: Parser
- grammar: Grammar
+ generateParsingTable
+ parseSequence
Parser
- Grammar: Grammar
- first : Map < String, List < String>>
- follow: Map < String, List < String >>
+ First()
+ Follow()
+ concatenationOfLength1 (l1: List < String, l2: List < String
Grammar
- terminals : String[]
- nonTerminals: String[]
- start : String
- productions : Production[]
+ getTerminals()
+ getNonTerminals()
+ getProductionForNonterminal()
+ checkCFG()
Production
+ left: String[]
+ right : String[]