Formal Languages and Compiler Design
Fifth laboratory
LL(1) Parser

Part 1

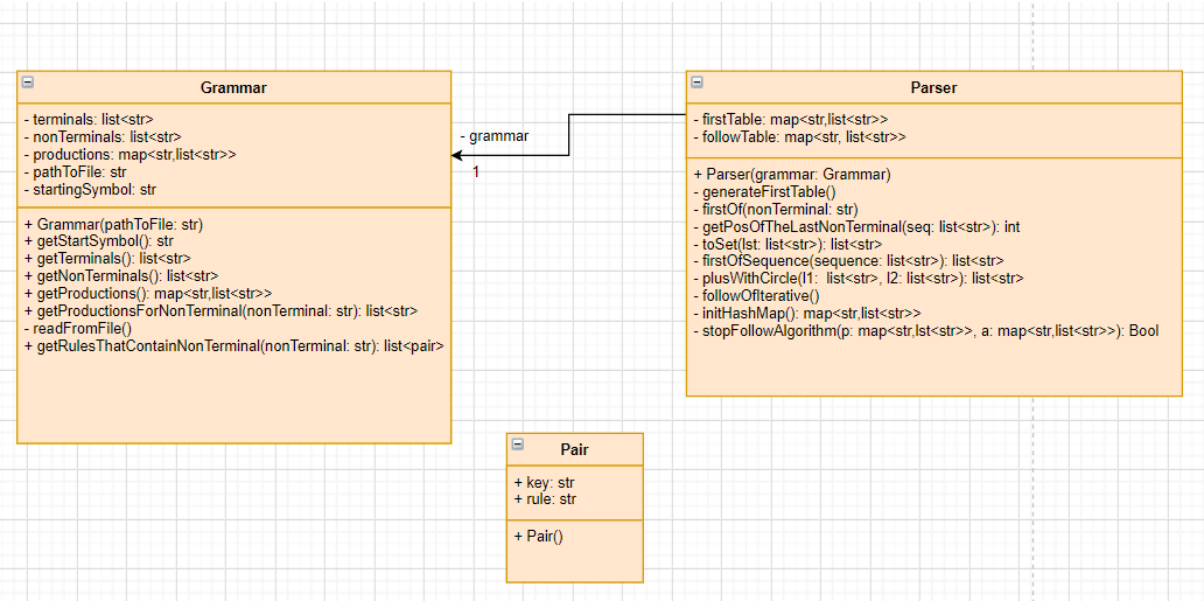
Implemented classes:

1. *Pair*: simply represent a key, value pair
2. *Grammar*:
   - represents the grammar for a language described in a grammar.in file (does not require to have this name)
   - holds in "nonTerminals" the set of non-terminals from that grammar, also in the list "terminals" we keep the list for terminal symbols from that list
   - as for the productions, we keep them in a HashMap, so that we can map for each non-terminal the productions it is involved in.
   - the starting symbol is stored in a String field named "startingSymbol"
3. *Parser*:
   - represents an LL(1) parses
   - it contains a grammar, the one for which we build the parses
   - the First Table, containing the first values for the non-terminals
   - the Follow Table, containing the follow values for the non-terminals
   - the first values are computed using recursion and the follow values are computed in an iterative manner

Class Diagram:

- see next page

Bacotiu Denisa-Cristina
Alb Dan-Cristian
LL(1) Parser
Gr.931



GRAMMAR.in file structure:

1. Description:
   - first line, contains separated by “,” the set of non-terminals
   - second line, contains separated by “,” the set of terminals
   - third line, flag which marks the presence of “,” as a terminal
   - last lines: productions
2. EBNF for grammar.in:

   grammar ::= terminals “\n” nonTerminals “\n” flag “\n” productions
   terminals ::= terminal | terminal “,” terminals
   terminal::= word
   nonTerminals::= nonTerminal | nonTerminal “,” nonTerminals
   nonTerminal::= word
   flag::= “true” | “false”
   productions::= production | production “\n” productions

Bacotiu Denisa-Cristina
Alb Dan-Cristian
LL(1) Parser
Gr.931

production::= nonTerminal “->” ruleList “\n”
ruleList::=rule | rule “|” ruleList
rule::= compoundRule | ε
compoundRule::= terminal “ ” | nonTerminal “ ” | terminal “ ” compoundRule | nonTerminal “ ” compoundRule

3. File Example:

```
S,A,B,C,D
a,(,),+,*
false
S -> B A
A -> + B A | ε
B -> D C
C -> * D C | ε
D -> ( S ) | a
```