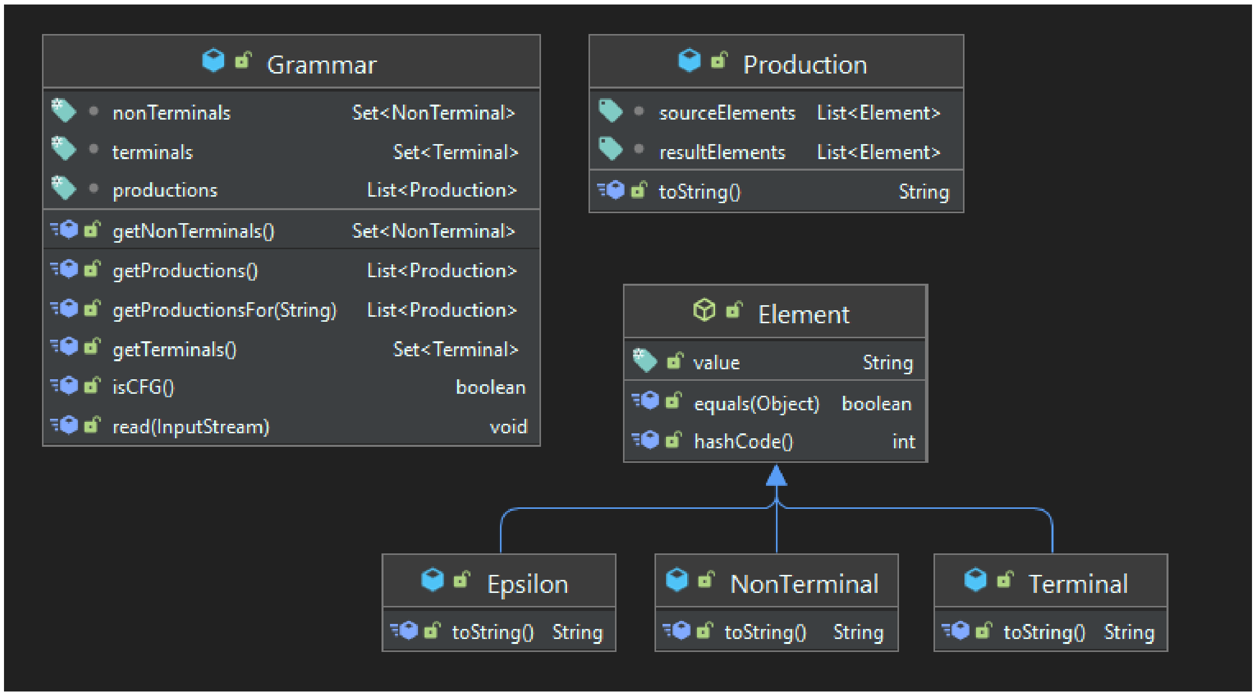
# Lab 5

## Requirements

Implement a grammar class for the LR(0) parsing algorithm.

## Architecture


Grammar
nonTerminals Set<NonTerminal>
terminals Set<Terminal>
productions List<Production>
getNonTerminals() Set<NonTerminal>
getProductions() List<Production>
getProductionsFor(String) List<Production>
getTerminals() Set<Terminal>
isCFG() boolean
read(InputStream) void
Production
sourceElements List<Element>
resultElements List<Element>
toString() String
Element
value String
equals(Object) boolean
hashCode() int
Epsilon
toString() String
NonTerminal
toString() String
Terminal
toString() String

## **Grammar** class

### read(InputStream input)

- Initializes a grammar instance based on an input stream.
- The input stream should contain a list of productions separated by newline.
- Throws:
    - IOException - Production source is empty
    - IOException - Production source only contains terminals
    - IOException - Production result is empty

### isCFG()

- Checks if each production has exactly one non-terminal symbol as source
- Returns:
    - True - the grammar is CFG
    - False - the grammar is not CFG

### **Production** class

Contains a list of source symbols (Element) and a list of result symbols

## Design Details

- Classes Element, Terminal, NonTerminal, which just encapsulate strings were used in order to make it easy to determine if a symbol is a terminal or a non terminal, in conjunction with using double quotes (“) in input files to denote terminal symbols;
- A Grammar contains a set of terminals, another one for nonterminals, and a list of productions; the first symbol of the first production (as entered in the input file) is considered to be the starting symbol;
- There is a reserved character (“$”) for denoting ε in the input file;
- “::=” is used as a separator between a production’s left side and right side, and a space is used to separate symbols

## Implementation

Github url: https://github.com/darius-calugar/flcd-parser-1

# Tests

Tests were ran on the following inputs:

## Test 1

```
E ::= E "+" T | T
T ::= T "*" F | F
F "a" ::= "(" E ")" | "a"
```

## Test 2

```
program ::= cmpdStmt
type ::= "integer"|"string"|userType|arrayType
arrayType ::= type "list"
userType ::= "identifier"
var ::= "identifier" | "identifier" "of" var | var "at" "const"
expression ::= numExpression|"const"
numExpression ::= expression operator expression|var|"const"
condition ::= expression relation expression
operator ::= "plus"|"minus"|"times"|"divided"|"mod"
relation ::= "smaller"|"larger"|"is"
def ::= "define" userType
declStmt ::= type "identifier"
declListStmt ::= arrayType "identifier"
stmt ::= cmpdStmt|assignStmt|ioStmt|ifStmt|declStmt|declListStmt|stopStmt
stmts ::= $ | stmt stmts
cmpdStmt ::= "begin" stmts "end"
assignStmt ::= var "becomes" expression
ioStmt ::= readStmt|printStmt
readStmt ::= "read" var
printStmt ::= "print" expression
ifStmt ::= "if" condition "then" stmt | "if" condition "then" stmt "else" stmt
loopStmt ::= "each" var "from" numExpression "to" numExpression stmt
stopStmt ::= "stop"
```