

<https://github.com/roxanazachman01/FLCD>

Documentation:

Statement:

Implement a parser algorithm

1. One of the following parsing methods will be chosen (assigned by teaching staff):

1.a. recursive descent

1.b. LL(1)

1.c. LR(0)

2. The representation of the parsing tree (output) will be (decided by the team):

2.a. productions string (max grade = 8.5)

2.b. derivations string (max grade = 9)

2.c. table (using father and sibling relation) (max grade = 10)

Deliverables:

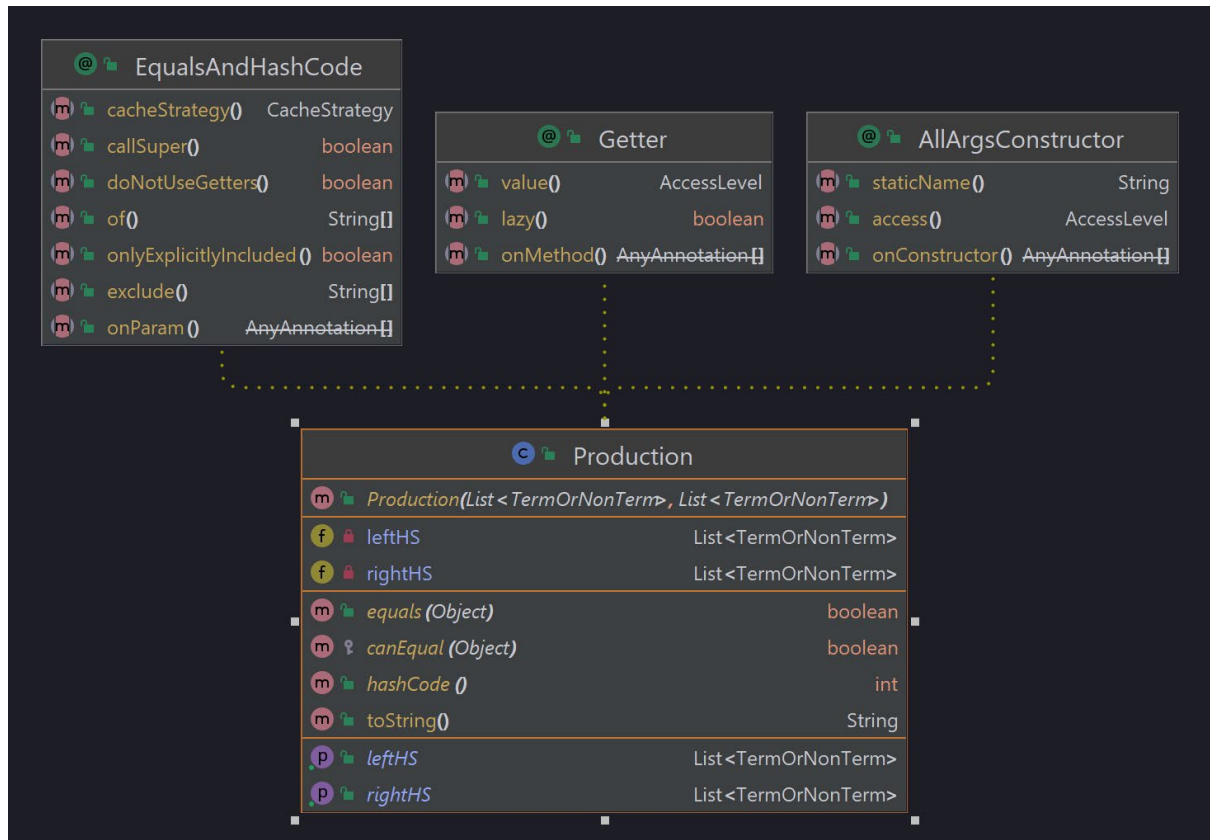
1. Class Grammar (required operations: read a grammar from file, print set of non-terminals, set of terminals, set of productions, productions for a given nonterminal, CFG check)
2. Input files: g1.txt (simple grammar from course/seminar), g2.txt (grammar of the minilanguage - syntax rules from [Lab 1b](#))

Implementation:

The input file is read in the constructor of the Grammar, then the loaded elements are validated. (throws exception if starting symbol is not a nonterminal or elements in productions are neither terminals or non-terminals).

The starting symbol is stored as a Nonterminal type, which implements the interface TermOrNonTerm. The set of non-terminals are stored as Nonterminal type and terminals as Terminal type. The list of production is stored as an array list of type Production, which is a class that contains a list of TermOrNonTerm as left hand-side and right hand-side. The reason for using the interface is to be able to store both terminals and non-terminals in the same list, since they have the same behaviour. (methods for getting the value and checking whether it is a terminal or nonterminal).

The code checks whether the grammar is a cfg by checking the size of the left hand-side in the list of productions. If all productions have size 1, then it is a cfg.



@ EqualsAndHashCode	
m cacheStrategy()	CacheStrategy
m callSuper()	boolean
m doNotUseGetters()	boolean
m of()	String[]
m onlyExplicitlyIncluded()	boolean
m exclude()	String[]
m onParam()	AnyAnnotation[]

@ Getter	
m value()	AccessLevel
m lazy()	boolean
m onMethod()	AnyAnnotation[]

I TermOrNonTerm	
m value()	String
p epsilon	boolean
p terminal	boolean

C Nonterminal	
m Nonterminal(String)	
f value	String
m equals(Object)	boolean
m canEqual(Object)	boolean
m hashCode()	int
m toString()	String
m value()	String
p epsilon	boolean
p terminal	boolean
p value	String



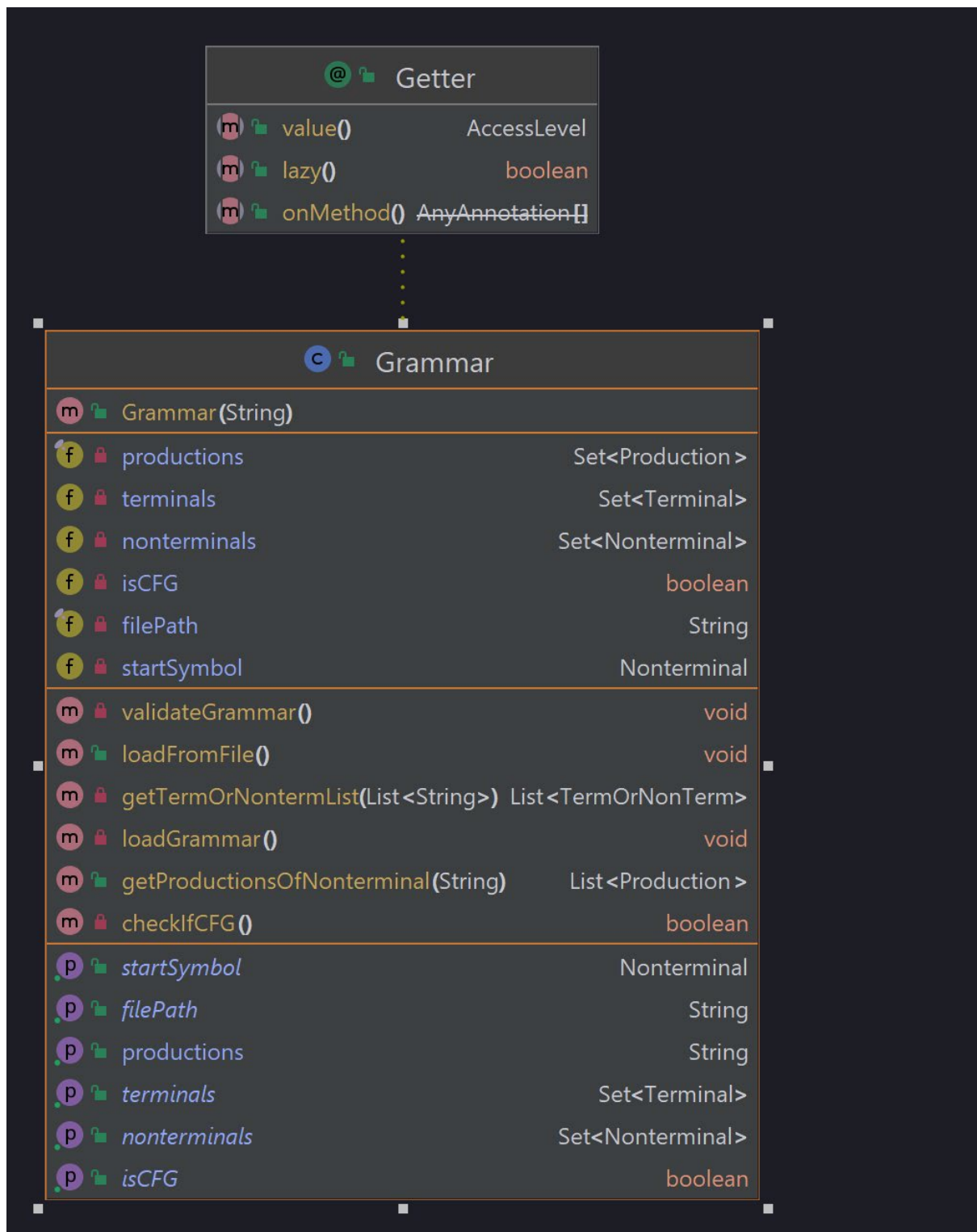
@	EqualsAndHashCode
m	cacheStrategy() CacheStrategy
m	callSuper() boolean
m	doNotUseGetters() boolean
m	of() String[]
m	onlyExplicitlyIncluded() boolean
m	exclude() String[]
m	onParam() AnyAnnotation[]

@	Getter
m	value() AccessLevel
m	lazy() boolean
m	onMethod() AnyAnnotation[]

I	TermOrNonTerm
m	value() String
p	epsilon boolean
p	terminal boolean

C	Terminal
m	Terminal(String)
f	value String
m	value() String
m	equals (Object) boolean
m	canEqual (Object) boolean
m	hashCode () int
m	toString() String
p	epsilon boolean
p	terminal boolean
p	value String





Testing: tests the grammar loads correctly the input file for the mini language

Input file:

```

1  program, decllist, cmpdstmt, declaration, type, identifier, type1, arraydecl, arrayitem, constnr, stmtlist, stmt, simplstmt, structstmt, assignstmt, iostmt, expression, term, f
2  {, }, ;, int, char, string, [, ], begin, end, =, +, -, *, %, /, (, ), read, write, if, else, while, <, <=, >, >=, !=
3
4  21
5  program#{, decllist, cmpdstmt, }
6  decllist#declaration|declaration, decllist
7  declaration#type, identifier, ;
8  type1#int|char|string
9  arraydecl#type1, [, constnr, ]
10 arrayitem#identifier, [, constnr, ]
11 type#type1|arraydecl
12 cmpdstmt#begin, stmtlist, end
13 stmtlist#stmt|stmt, stmtlist
14 stmt#simplstmt, ;|structstmt, ;
15 simplstmt#assignstmt|iostmt
16 assignstmt#identifier, =, expression|arrayitem, =, expression
17 expression#expression, +, term|expression, -, term|term
18 term#term, *, factor|term, %, factor|term, /, factor|factor
19 factor#(, expression, )|identifier|constnr
20 iostmt#read, (, identifier, )|read, (, arrayitem, )|write, (, arrayitem, )|write, (, identifier, )|write, (, constnr, )|write, (, constchar, )|write, (, conststring, )
21 structstmt#cmpdstmt|ifstmt|whilestmt
22 ifstmt#if, (, condition, ), {, stmtlist, }, else, {, stmtlist, }
23 while#while, (, condition, ), {, stmtlist, }
24 condition#expression, relation, expression
25 relation#<|<=|>|>=|!=
26
27 program

```

Tests:

```

1  public class GrammarTest {
2      @Test
3      public void testGrammar() {
4          final Grammar grammar = new Grammar( filePath: "in/g2.txt");
5
6          assertThat(grammar.getStartSymbol()).isEqualTo(new Nonterminal( value: "program"));
7
8          assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "program"));
9          assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "stmt"));
10         assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "term"));
11         assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "term"));
12
13         assertThat(grammar.getTerminals()).contains(new Terminal( value: "{"));
14         assertThat(grammar.getTerminals()).contains(new Terminal( value: "while"));
15         assertThat(grammar.getTerminals()).contains(new Terminal( value: "if"));
16
17         assertThat(grammar.isCFG()).isTrue();
18
19         assertThat(grammar.getProductionsOfNonterminal( str: "condition").get(0)
20             .getRightHS()).contains(new Nonterminal( value: "relation"));
21     }
22 }

```