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 descendent

    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:

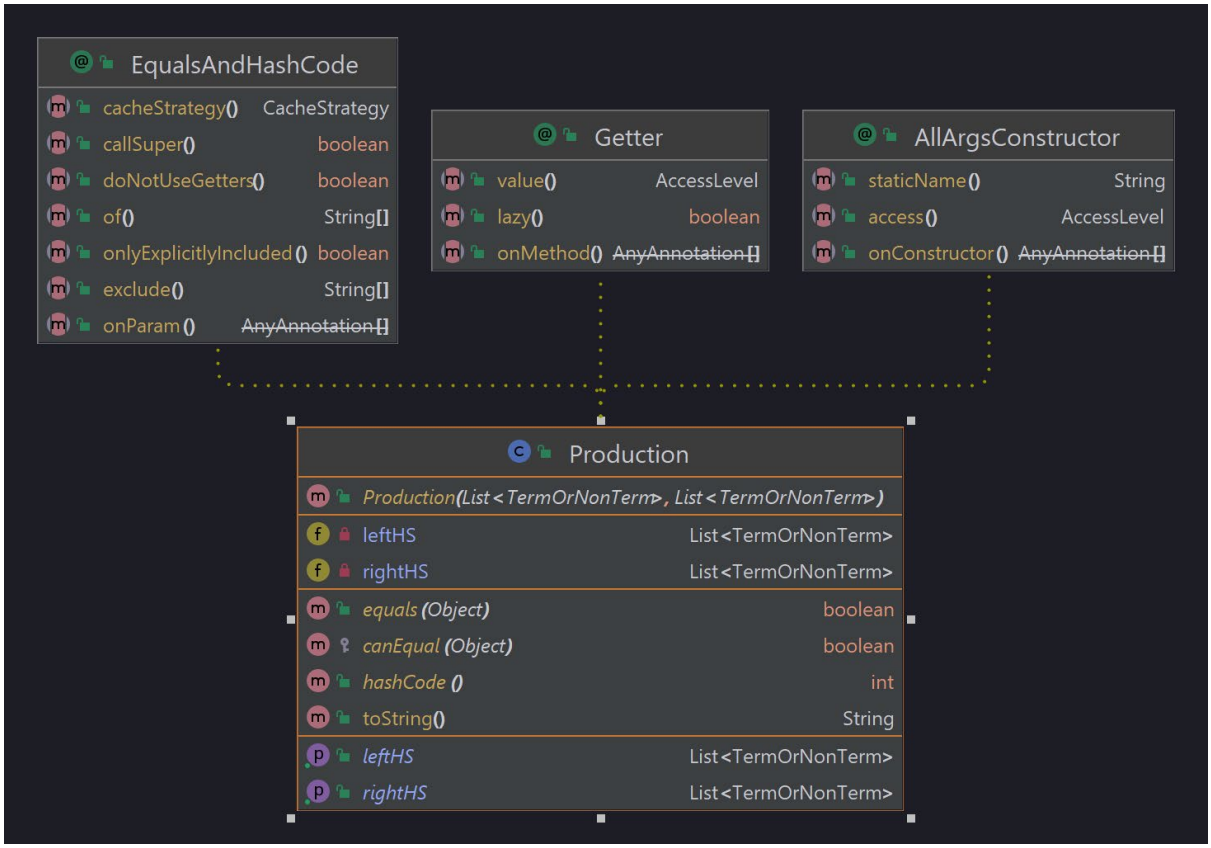
1. Class Grammar (required operations: read a grammar from file, print set of non-terminals, set of terminals, set of productions, productions for a given nonterminal, CFG check)
2. Input files: g1.txt (simple grammar from course/seminar), g2.txt (grammar of the minilanguage - syntax rules from Lab 1b)

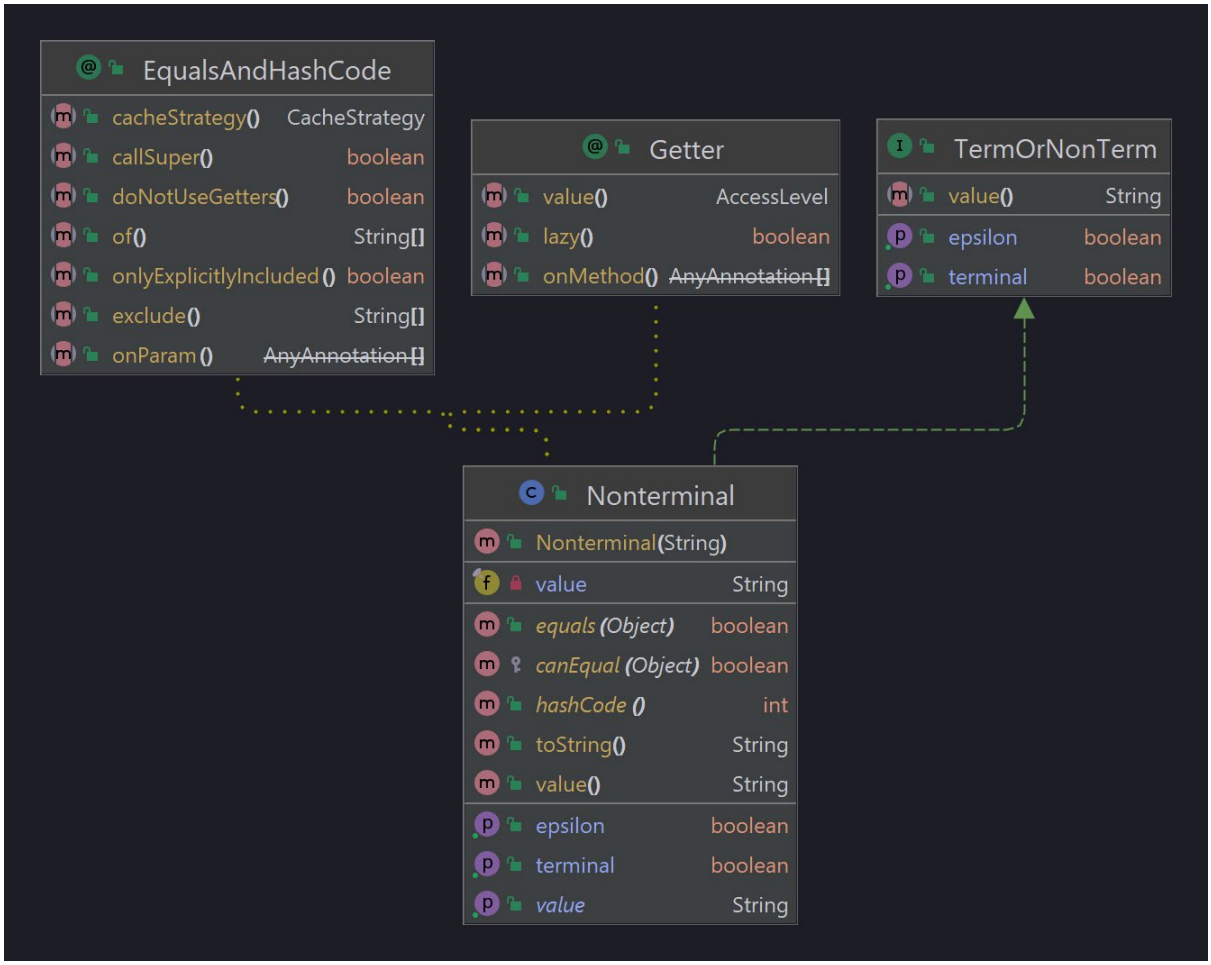
Implementation:

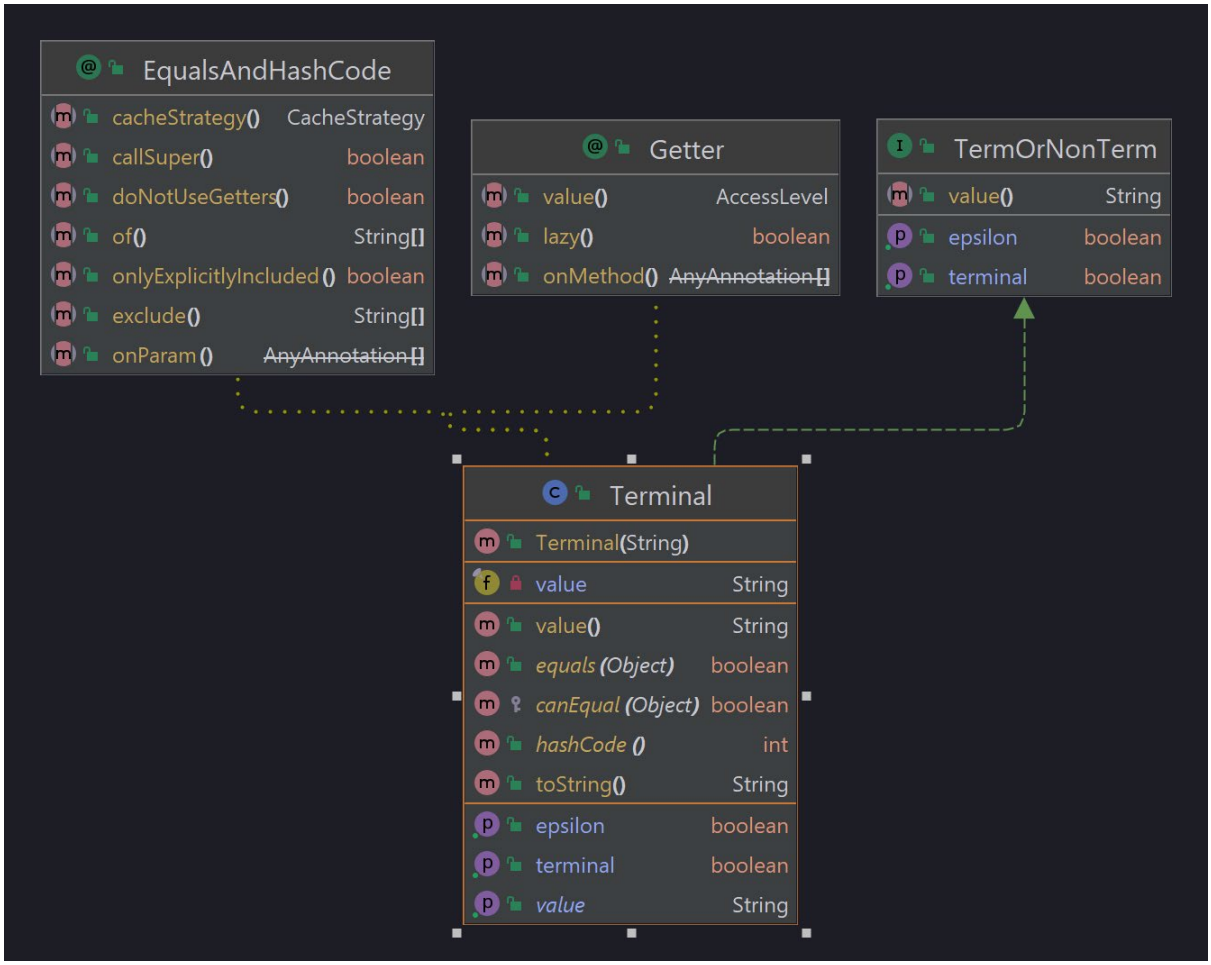
The input file is read in the constructor of the Grammar, then the loaded elements are validated. (throws exception if starting symbol is not a nonterminal or elements in productions are neither terminals or non-terminals).

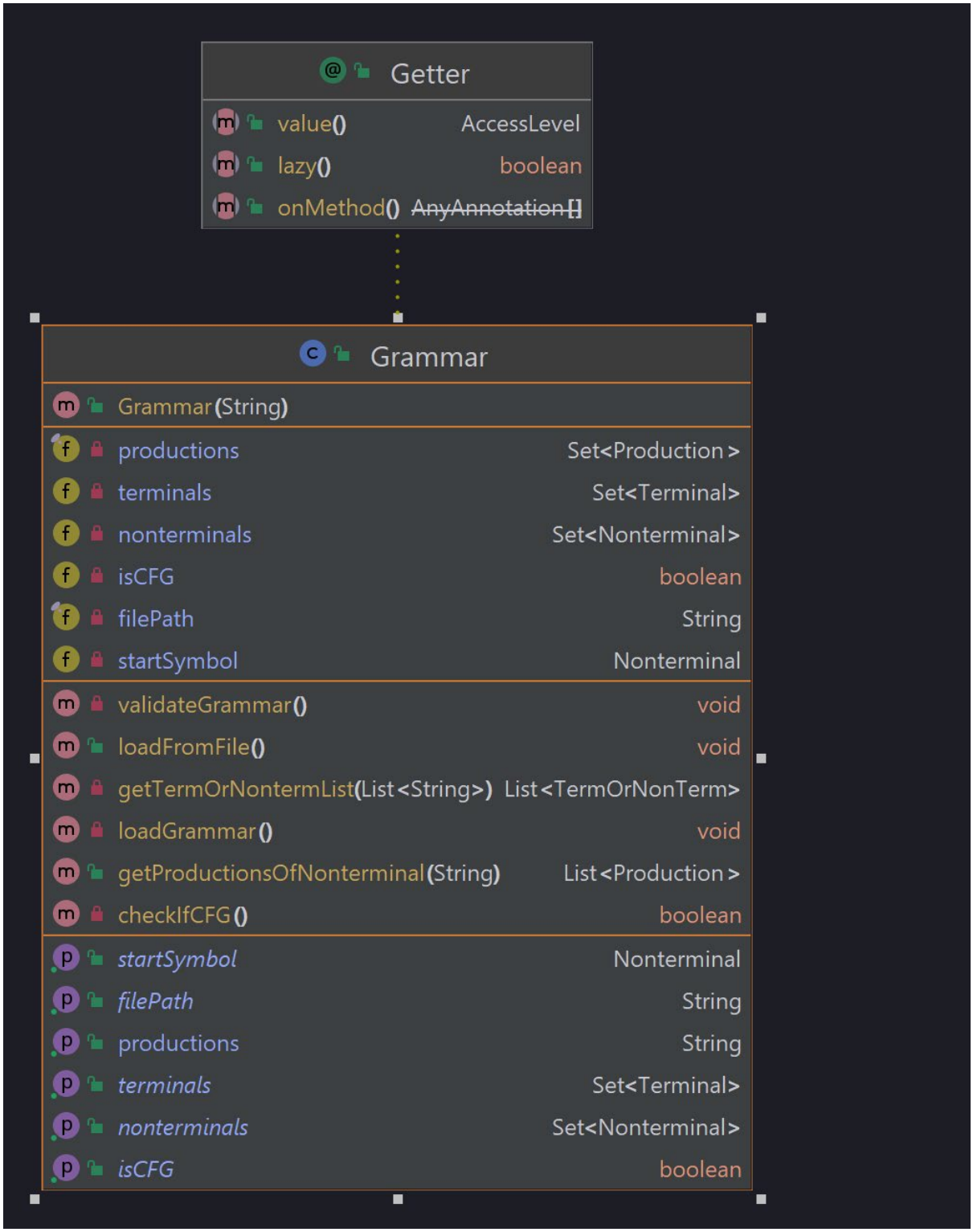
The starting symbol is stored as a Nonterminal type, which implements the interface TermOrNonTerm. The set of non-terminals are stored as Nonterminal type and terminals as Terminal type. The list of production is stored as an array list of type Production, which is a class that contains a list of TermOrNonTerm 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.



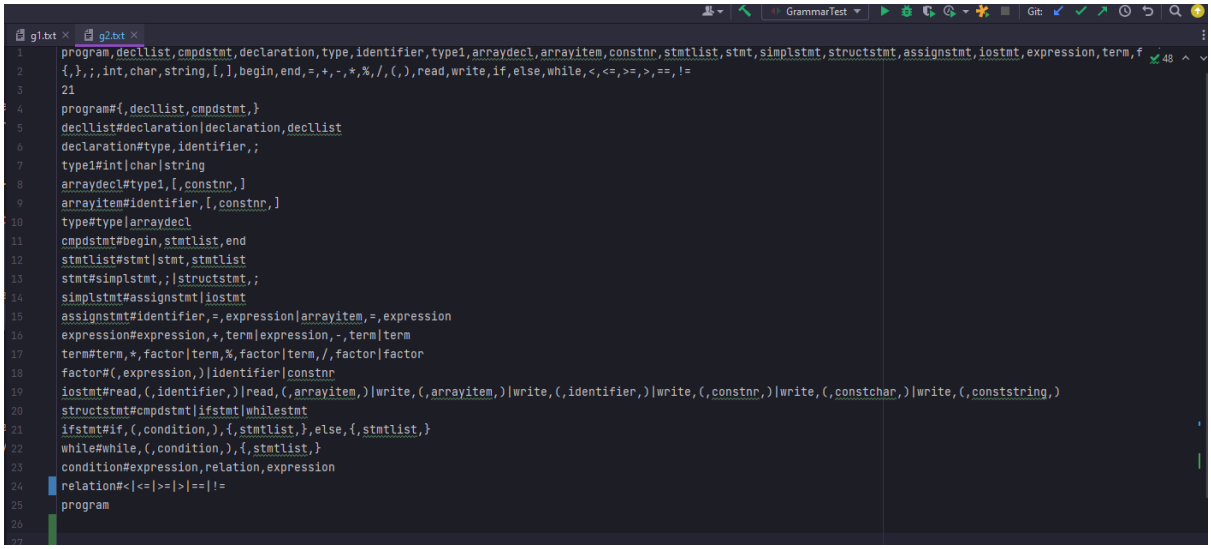

EqualsAndHashCode
cacheStrategy() CacheStrategy
callSuper() boolean
doNotUseGetters() boolean
of() String[]
onlyExplicitlyIncluded() boolean
exclude() String[]
onParam() AnyAnnotation[]
Getter
value() AccessLevel
lazy() boolean
onMethod() AnyAnnotation[]
TermOrNonTerm
value() String
epsilon boolean
terminal boolean
Nonterminal
Nonterminal(String)
value String
equals (Object) boolean
canEqual (Object) boolean
hashCode () int
toString() String
value() String
epsilon boolean
terminal boolean
value String


EqualsAndHashCode
cacheStrategy() CacheStrategy
callSuper() boolean
doNotUseGetters() boolean
of() String[]
onlyExplicitlyIncluded() boolean
exclude() String[]
onParam() AnyAnnotation[]
Getter
value() AccessLevel
lazy() boolean
onMethod() AnyAnnotation[]
TermOrNonTerm
value() String
epsilon boolean
terminal boolean
Terminal
Terminal(String)
value String
value() String
equals (Object) boolean
canEqual (Object) boolean
hashCode () int
toString() String
epsilon boolean
terminal boolean
value String



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

Input file:



Tests:

```java
public class GrammarTest {
    @Test
    public void testGrammar() {
        final Grammar grammar = new Grammar( filePath: "in/g2.txt");

        assertThat(grammar.getStartSymbol()).isEqualTo(new Nonterminal( value: "program"));

        assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "program"));
        assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "stmt"));
        assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "term"));
        assertThat(grammar.getNonterminals()).contains(new Nonterminal( value: "term"));

        assertThat(grammar.getTerminals()).contains(new Terminal( value: "{"));
        assertThat(grammar.getTerminals()).contains(new Terminal( value: "while"));
        assertThat(grammar.getTerminals()).contains(new Terminal( value: "if"));

        assertThat(grammar.isCFG()).isTrue();

        assertThat(grammar.getProductionsOfNonterminal( str: "condition").get(0)
                .getRightHS()).contains(new Nonterminal( value: "relation"));
    }
}
```