ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „Софтуерни и интернет технологии“

ДОКУМЕНТАЦИЯ

по дисциплината „Обектно Ориентирано Програмиране I част”

на тема: „ Контекстно-свободна граматика “

Изготвил: Виктор Янев

Специалност: СИТ

Група: 3б

Факултетен номер: 23621633

Проверил:

Глава 1. Увод

1.1. Описание и идея на проекта

Проект 5: Контекстно-свободна граматика

Да се реализира програма, която поддържа операции с контекстно-свободна граматика, главни латински букви за променливи (нетерминали) и малки латински букви и цифри за терминали.

Граматиките да се сериализират по разработен от Вас формат. Всяка прочетена граматика да получава уникален идентификатор.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (open, close, save, save as, help и exit):

1.2. Цел и задачи на разработката

Целта на проекта е да се създаде програма със CLI за обработка на контекстно-свободни граматики като се спазват правилата на обектно ориентираното програмиране и се имплементират следните функции:

list -Списък с идентификаторите на всички прочетени граматики

print- <id> Извежда граматиката в подходящ формат. За всяко правило да се отпечата пореден номер

save -<id> <filename> Записва граматиката във файл

addRule- <id> <rule> Добавя правила

removeRule -<id> <n> Премахване на правило по пореден номер

union -<id1> <id2> Намира обединението на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика

concat -<id1> <id2> Намира конкатенацията на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика

chomsky- <id> Проверява дали дадена граматика е в нормална форма на Чомски

cyk- <id> Проверява дали дадена дума е в езика на дадена граматика (CYK алгоритъм)

iter -<id> Намира резултат от изпълнението на операцията “итерация” (звезда на Клини) над граматика и създава нова граматика. Отпечатва идентификатора на новата граматика

empty -<id> Проверява дали езикът на дадена контекстно-свободна граматика е празен

chomskify -<id> Преобразува граматика в нормална форма на Чомски. Отпечатва идентификатора на новата граматик

Както и общите команди (open, close, save, save as, help и exit)

1.3. Структура на документацията

4.1. Реализация на класове

4.2. Алгоритми и оптимизации.

Глава 5. Тестване

5.1. Планиране, описание и създаване на тестови сценарии (

Глава 6. Заключение

6.1. Обобщение на изпълнението на началните цел

6.2 Използвана литература

Глава 2. Преглед на предметната област

2.1. Основни концепции и алгоритми, които ще бъдат използвани

Една контекстно-свободна граматика се дефинира от следните данни:

- **V** е крайно множество; всеки елемент **v ∈ V** се нарича **нетерминален символ** или **променлива**.
- **Σ** е крайно множество от **терминални символи.**
- **R** е крайно множество от съотношения в **V × (V ∪ Σ)***, където '*' означава операцията на Клийн (Kleene star). Елементите на **R** се наричат правила за пренаписване

Ще бъде използван СУК алгоритъм, който проверява дали дадена дума съществува в дадена граматика

Ще бъде използван и алгоритъм са превръщане на граматика в Чомски Нормална Форма

2.2. Подходи, методи за решаване на поставените задачи

Подход:

Първо се създава имплементация на контекстно-свободна граматика чрез създаване на класове за нейните азбуки — множества от терминални или нетерминални символи —, създаване на клас за правило — съдържащ във себе си низове за лява и дясна част — и на края имплементация на самият клас за контекстно-свободна граматика в който има две азбуки — една за терминални и една за нетерминални символи — и множество правила, които важат за нея, както и низово поле в което се записва от кой файл е прочетена дадена граматика.

После се създава клас в който ще съдържаме всяка отворена контекстно-свободна граматика и нейн индентифициращ номер (id); Това се реализира чрез структурата от данни Map, където ключът се пада нейният индентифициращ номер (id), а стойността самата граматика.

2.3. Стандарти

Всяка една контекстно-свободна граматика се записва във файл завършващ на ".cfg".

Във всеки един .cfg файл може да има по само една контекстно-свободна граматика.

Всеки .cfg файл следва следната конструкция:

Ред 1: всички терминални и нетерминални символи, разделени с интервал

Ред 2 - край: на всеки нов ред се изписва едно правило, последвано със запетая

Правила: всяко едно правило е във форма "А->Б" където "А" е нетерминален символ, а "Б" ∈ (V ∪ Σ)*

За терминални символи се смятат малки букви; на пр. "а"

За нетерминални символи се смятат главни букви; на пр. "А"

Глава 3. Проектиране

3.1. Обща структура на проекта

Реализираните пакети са:

- commands - в него се съдържат всички команди, както и пакет с общите команди

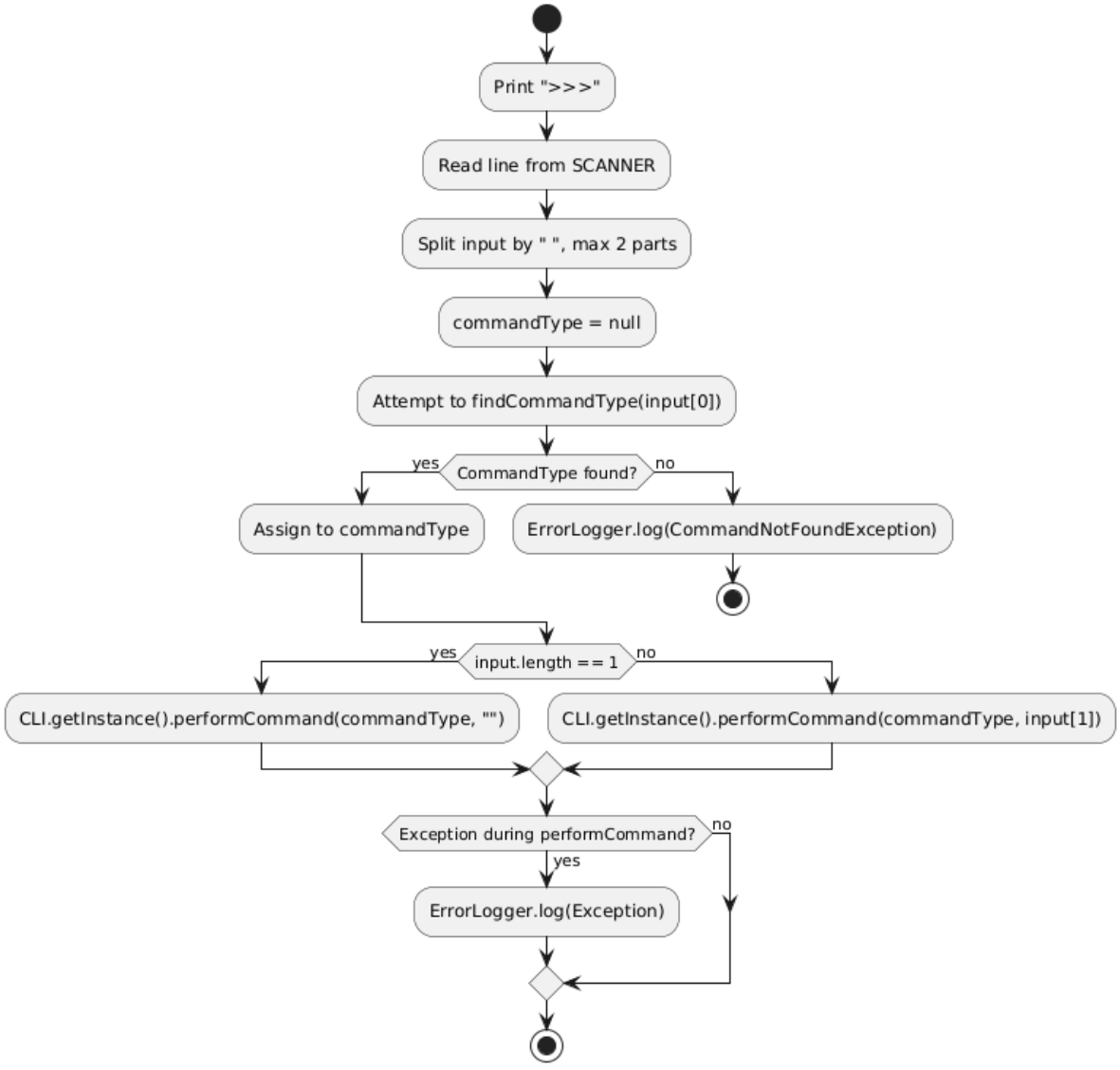
- exceptions - в този пакет се намират всичките възможни грешки които могат да възникнат

- grammar - съдържа имплементациите за правило, азбука, граматика и картата на граматиките

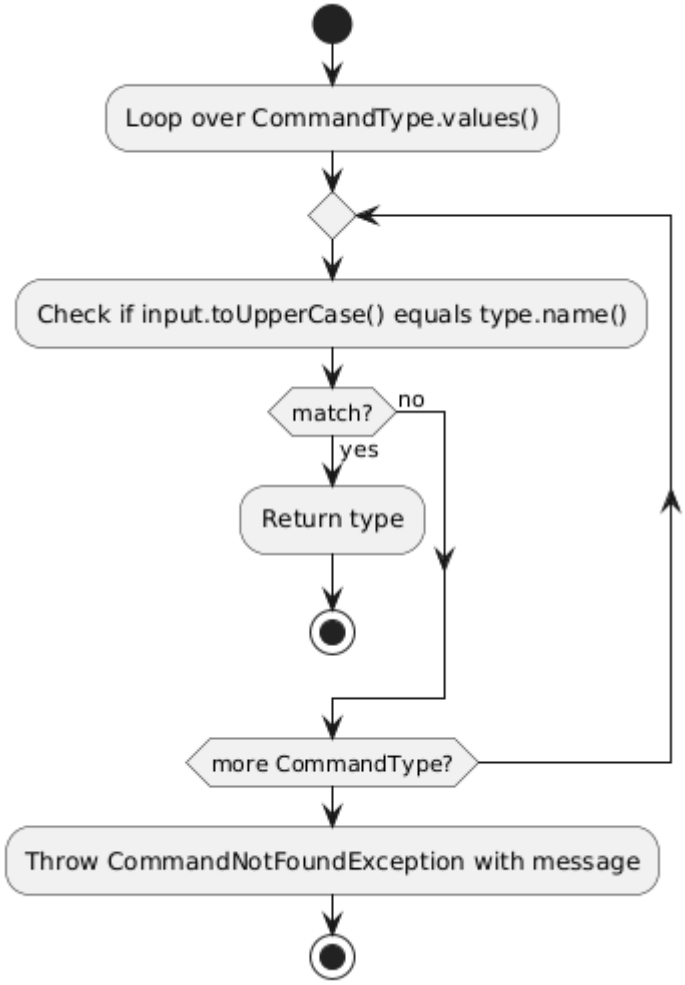
- util - съдържа неспецифични за програмата класове, както и класът който изпълнява главният цикъл на програмата (ProgramManager)

3.2. Диаграми/Блок схеми

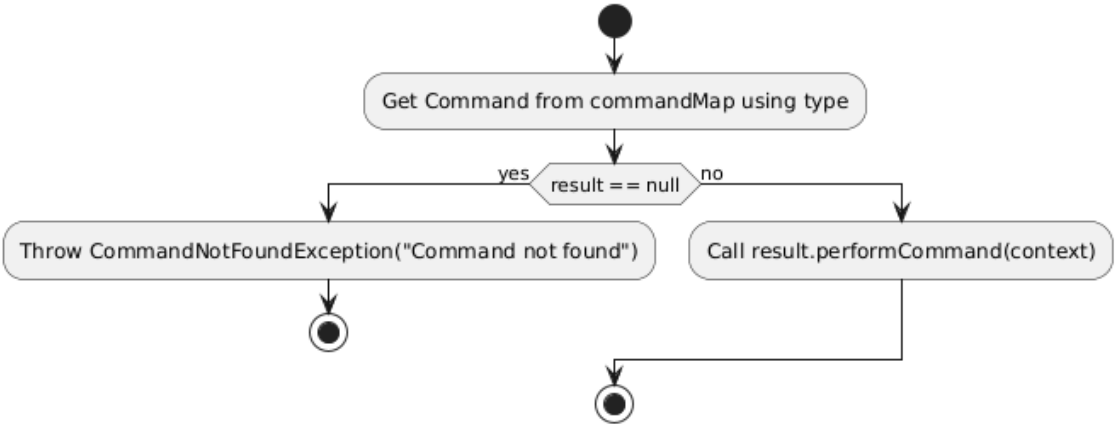
Диаграма за ProgramManager.run():



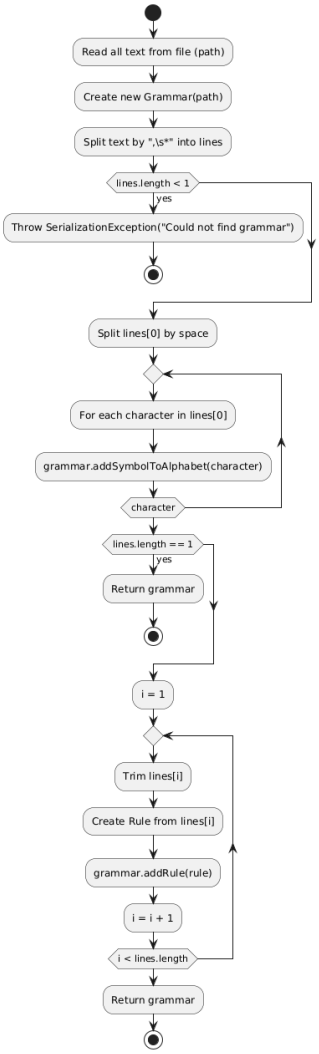
Диаграма за намиране на команден тип ProgramManager.findCommandType(String input)



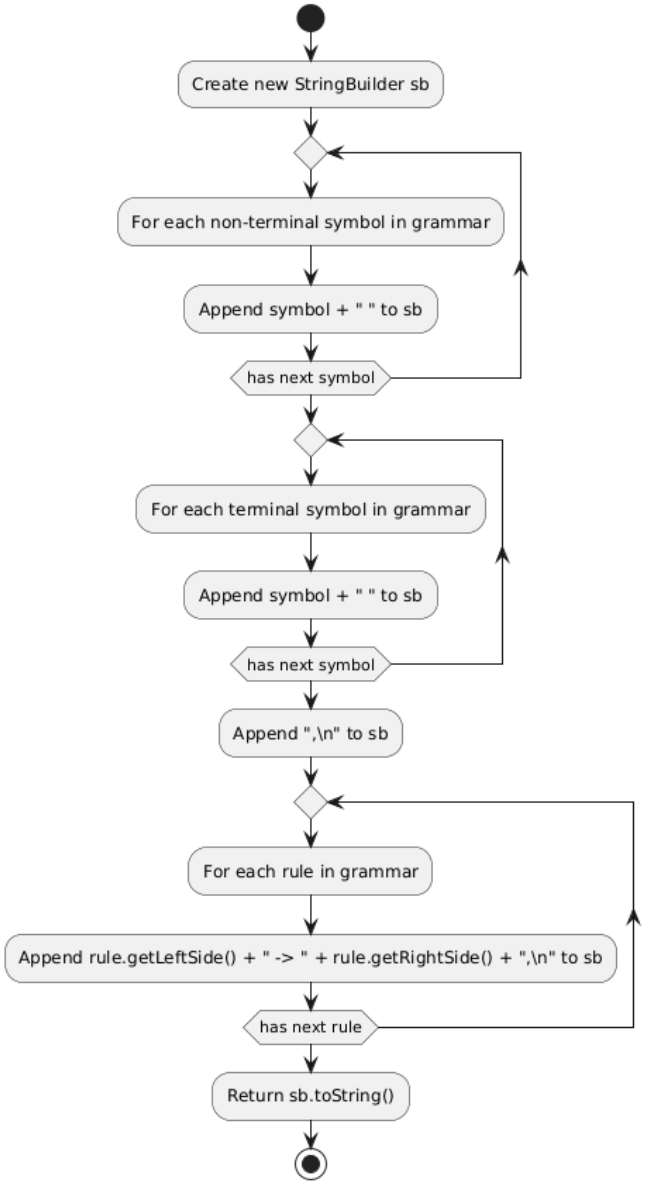
Диаграма за извикване на команда CLI.performCommand(CommandType type, String context)



Диаграма за четене на контекстно-свободна граматика от файл Parser.readGrammarFromFile(String path):



Диаграма за превеждане на граматика в низ Parser.grammarToString(Grammar grammar)



Глава 4. Реализация

4.1. Реализация на класове

CloseCommand**: javadoc/commands/common/CloseCommand.html**

```
if (context.isEmpty()) throw new CommandContextException("Empty
command context");
```

```
        //context is grammar ID

        int id = Integer.parseInt(context);
```

```
if (GrammarMap.getInstance().removeGrammarByID(id)){
            System.out.println("Removed grammar with id " + id);

        }else{

            throw new GrammarNotFoundException("Could not find
grammar with id " + id);

        }
```

ExitCommand: javadoc/commands/common/ExitCommand.html

```
System.out.println("Exiting");

        System.exit(0);
```

HelpCommand: javadoc/commands/common/HelpCommand.html

```
Map<CommandType, Command>  commandMap =
CLI.getInstance().getCommandMap();

        StringBuilder sb = new StringBuilder();



        for(Map.Entry<CommandType, Command> entry:
commandMap.entrySet()){



            sb.append(entry.getValue().getDesc());

            sb.append("\n\n");



        }

        System.out.println(sb.toString());
```

OpenCommand: javadoc/commands/common/OpenCommand.html

```
if (context.isEmpty()) throw new CommandContextException("Empty
command context");

        try{


GrammarMap.getInstance().addGrammar(Parser.readGrammarFromFile(conte
xt));

            System.out.println("Grammar loaded from " + context);

        } catch (Exception e){

            ErrorLogger.log(e);

        }
```

SaveAsCommand: javadoc/commands/common/SaveAsCommand.html

```
if (context.isEmpty()) throw new CommandContextException("Empty
command context");



        String[] keyWords = context.split(" ", 2);

        if(keyWords.length < 2) throw new
CommandContextException("Not enough context given");



        //context is grammar ID

        int id = Integer.parseInt(keyWords[0]);
```

```
        Grammar grammar =
GrammarMap.getInstance().getGrammarByID(id);

        if(grammar == null) throw new
GrammarNotFoundException("Could not find grammar with id " + id);



        String path = keyWords[1];

        WriteToFile.write(path,false,
Parser.grammarToString(grammar));



        System.out.println("Saved grammar with id " + id + " to file
" + path);
```

SaveCommand: javadoc/commands/common/SaveCommand.html

```
if (context.isEmpty()) throw new CommandContextException("Empty
command context");



        //context is grammar ID

        int id = Integer.parseInt(context);



        Grammar grammar =
GrammarMap.getInstance().getGrammarByID(id);

        if(grammar == null) throw new
GrammarNotFoundException("Could not find grammar with id " + id);



        String path = grammar.getOriginalFile();
```

```
        if(path.isEmpty()) throw new PathException("This grammar
does not have a default path");

        WriteToFile.write(path,false,
Parser.grammarToString(grammar));



        System.out.println("Saved grammar with id " + id + " to file
" + path);
```

AddRuleCommand: javadoc/commands/AddRuleCommand.html

```
String rule = keyWords[1];



        Grammar grammar =
GrammarMap.getInstance().getGrammarByID(id);

        if(grammar == null) throw new
GrammarNotFoundException("Failed to find grammar with id: " + id);

        grammar.addRule(new Rule(rule));

        System.out.println("Added rule " + rule);
```

ChomskyCommand: javadoc/commands/ChomskifyCommand.html

```
for (Rule r : grammar.getRules()) {

            String rightSide = r.getRightSide();

            if (rightSide.length() == 1 &&
grammar.getTerminalSymbols().contains(rightSide)) {

                continue;
```

```
            }

            if (rightSide.length() == 2 &&
grammar.getNonTerminalSymbols().contains(String.valueOf(rightSide.ch
arAt(0))) &&
grammar.getNonTerminalSymbols().getSymbols().contains(String.valueOf
(rightSide.charAt(1)))) {

                continue;

            }

            System.out.println("Grammar is not in Chomsky Normal
Form");

            return;

        }
```

ConcatCommand: javadoc/commands/ConcatCommand.html

```
//Add Alphabets from Grammar<id1>


newGrammar.getTerminalSymbols().addAll(grammar1.getTerminalSymbols()
.getSymbols());


newGrammar.getNonTerminalSymbols().addAll(grammar1.getNonTerminalSym
bols().getSymbols());

        //Add Alphabets from Grammar<id2>


newGrammar.getTerminalSymbols().addAll(grammar2.getTerminalSymbols()
.getSymbols());


newGrammar.getNonTerminalSymbols().addAll(grammar2.getNonTerminalSym
bols().getSymbols());
```

```
        //Add Rules from both grammars

        newGrammar.getRules().addAll(grammar1.getRules());

        newGrammar.getRules().addAll(grammar2.getRules());



        System.out.println("New grammar saved with id - " +
GrammarMap.getInstance().addGrammar(newGrammar));
```

EmptyCommand: javadoc/commands/EmptyCommand.html

```
if(grammar.getNonTerminalSymbols().size() == 0 &&
grammar.getTerminalSymbols().size() == 0){

            System.out.println("Grammar is empty");

        }else{

            System.out.println("Grammar is not empty");

        }
```

IterCommand: javadoc/commands/IterCommand.html

```
newGrammar.getNonTerminalSymbols().addAll(grammar.getNonTerminalSymb
ols().getSymbols());


newGrammar.getNonTerminalSymbols().addAll(grammar.getNonTerminalSymb
ols().getSymbols());


newGrammar.getTerminalSymbols().addAll(grammar.getTerminalSymbols().
getSymbols());
```

```
newGrammar.getTerminalSymbols().addAll(grammar.getTerminalSymbols().
getSymbols());

        for(Rule rule : grammar.getRules()){

            newGrammar.addRule(rule);

        }



newGrammar.getNonTerminalSymbols().addSymbol(String.valueOf(Alphabet
.EPSILON));


newGrammar.getTerminalSymbols().addSymbol(String.valueOf(Alphabet.EP
SILON));



        for(String c1 : grammar.getTerminalSymbols().getSymbols()){

            for(String c2 :
grammar.getTerminalSymbols().getSymbols()){

                newGrammar.getTerminalSymbols().addSymbol(c1 + c2);

            }

        }

        for(String c1 :
grammar.getNonTerminalSymbols().getSymbols()){

            for(String c2 :
grammar.getNonTerminalSymbols().getSymbols()){

                newGrammar.getNonTerminalSymbols().addSymbol(c1 +
c2);

            }
```

```
        }


        GrammarMap.getInstance().addGrammar(newGrammar);

        System.out.println("Grammar iterated. Created new grammar
with id " + (GrammarMap.getInstance().getIdCounter() - 1));
```

ListCommand: javadoc/commands/ListCommand.html

```
for(Map.Entry<Integer,Grammar> entry : grammarMap.entrySet()){

            sb.append(entry.getKey());

            sb.append("\n");

        }

        System.out.println(sb.toString());
```

PrintCommand: javadoc/commands/PrintCommand.html

```
sb.append("\tTerminal Symbols: {");

        for (String c : grammar.getTerminalSymbols().getSymbols()){

            sb.append(c).append(", ");

        }

        sb.append("}\n\tNon-terminal Symbols: {");


        for (String c :
grammar.getNonTerminalSymbols().getSymbols()){

            sb.append(c).append(", ");
```

```
        }

        sb.append("}\n\n");



        sb.append("\tRules:\n");

        int counter = 0;

        for(Rule rule : grammar.getRules()){


sb.append("\t\t").append("(").append(counter++).append(")
").append(rule.getLeftSide()).append(" ->
").append(rule.getRightSide()).append("\n");

        }

        sb.append("\n");



        sb.append("\tOriginal file:
").append(grammar.getOriginalFile().isBlank() ? "N/A" :
grammar.getOriginalFile());

        System.out.println(sb.toString());
```

RemoveRuleCommand: javadoc/commands/RemoveRuleCommand.html

```
if(grammar == null) throw new GrammarNotFoundException("Failed to
find grammar with id: " + id);

        try{

            grammar.removeRule(ruleId);

        }catch (RuleNotFoundException e){

            ErrorLogger.log(e);
```

```
        }
```

UnionCommand: javadoc/commands/UnionCommand.html

```
for(String c : grammar1.getTerminalSymbols().getSymbols()){
            if(grammar2.getTerminalSymbols().contains(c)){
                newGrammar.getTerminalSymbols().addSymbol(c);
            }
        }
        for(String c :
grammar1.getNonTerminalSymbols().getSymbols()){
            if(grammar2.getNonTerminalSymbols().contains(c)){
                newGrammar.getNonTerminalSymbols().addSymbol(c);
            }
        }

        for(Rule r : grammar1.getRules()){
            if(grammar2.getRules().contains(r)){
                newGrammar.addRule(r);
            }
        }
```

```
        System.out.println("New grammar saved with id - " +
GrammarMap.getInstance().addGrammar(newGrammar));
```

4.2. Алгоритми и оптимизации.

Алгоритъм за превръщане на една граматика в Чомски нормална форма - Целта е да се преобразува дадена контекстно-свободна граматика така, че всяко правило да е в една от следните форми:

A → BC (два нетерминални символа)

A → a (един терминален символ)

Стъпки за превръщане в нормална форма на Чомски:

1.Премахване на правила без лява или дясна страна

2. Премахване на правила с дясна страна 'Є' (епсилон)

3.Заместване на терминални символи в правила с дясна страна > 1 с нов нетерминален символ и създаваме ново правило със новия нетерминален символ и терминалния символ

4.Вече всяко правило или има 1 терминален символ от дясната страна или n на брой нетерминални символа от дясната страна. Създаваме и променяме правила така, че за всяко правило да е вярно A → BC (два нетерминални символа)

СУК алгоритъм - Използва се, за да провери дали дадена дума принадлежи на езика, генериран от контекстно-свободна граматика в нормална форма на Чомски.

Работи чрез динамично програмиране.

Глава 5. Тестване

5.1. Планиране, описание и създаване на тестови сценарии

CloseCommand:

```
>>>close 0
Removed grammar with id 0
```

ExitCommand:

```
>>>exit
Exiting

Process finished with exit code 0
```

HelpCommand:

```
>>>help
list - List every loaded grammar's id

removerule <id> <rule id> - Removes given rule from given grammar

help - Prints the description of every command

chomsky <id> - Checks if given grammar is in Chomsky Normal Form

union <id1> <id2> - Gets the union between two grammars and saves it to the grammar map

chomskify <id> - Converts given grammar into Chomsky Normal Form and saves it to the grammar map

concat <id1> <id2> - Gets the concatenation between two grammars and saves it to the grammar map

exit - Exits the program

iter <id> - Iterates over given grammar and creates a new one

CYK <id> <word> - Checks if given word is in given grammar using CYK algorithm

save <id> - Saves grammar with the given id to its original file

close <id> - Removes the grammar with the given key(id) from the grammar map

print <id> - Prints given command to console

saveas <id> <filename> - Saves grammar with he given id to the given path

open <filename> - Reads a grammar from a .cfg file and adds it to the GrammarMap singleton's map

addrule <id> <rule> - Adds given rule to given grammar
```

OpenCommand:

```
>>>open test.cfg
Grammar loaded from test.cfg
```

SaveAsCommand:

```
>>>saveas 0 something.cfg
Saved grammar with id 0 to file something.cfg
```

SaveCommand:

```
>>>save 0
Saved grammar with id 0 to file test.cfg
```

AddRuleCommand:

```
>>>addrule 0 D->a
Added rule D->a
```

ChomskifyCommand:

```
>>>chomskify 0
Chomskyfied. New Grammar id: 1
>>>print 1
Grammar with id 1

    Terminal Symbols: {a, b, }
    Non-terminal Symbols: {A, B, C, D, E, F, G, }

    Rules:
        (0) C -> b
        (1) G -> FD
        (2) F -> DC
        (3) E -> CB
        (4) B -> GD
        (5) D -> a
        (6) C -> CC
        (7) A -> ED

    Original file: N/A
```

ChomskyCommand:

```
>>>chomsky 1
Grammar is in Chomsky Normal Form
```

ConcatCommand:

```
>>>open test.cfg
Grammar loaded from test.cfg
>>>open something.cfg
Grammar loaded from something.cfg
>>>concat 0 1
New grammar saved with id - 2
>>>print 2
Grammar with id 2

    Terminal Symbols: {bb, ff, a, b, d, f, abaa, }
    Non-terminal Symbols: {A, B, C, D, F, }

    Rules:
        (0) A -> bBa
        (1) F -> Dd
        (2) B -> abaa
        (3) D -> ff
        (4) C -> bb

    Original file: N/A
```

CYKCommand:

```
>>>chomskify 0
Chomskyfied. New Grammar id: 1
>>>cyk 1 abaa
True
```

EmptyCommand:

```
>>>open test.cfg
Grammar loaded from test.cfg
>>>empty 0
Grammar is not empty
```

IterCommand:

```
>>>iter 0
Grammar iterated. Created new grammar with id 1
>>>print 1
Grammar with id 1

    Terminal Symbols: {bb, bba, abb, aa, a, ab, b, bbb, Є, abaaa, abaab, aabaa, abaa, bbabaa, bbbb, babaa, abaabb, abaaabaa, ba, }
    Non-terminal Symbols: {AA, BB, CC, A, AB, BC, B, AC, C, Є, CA, BA, CB, }

    Rules:
        (0) A -> bBa
        (1) B -> abaa
        (2) C -> bb

    Original file: N/A
```

ListCommand:

```
>>>list
0
1
```

PrintCommand:

```
>>>print 0
Grammar with id 0

    Terminal Symbols: {bb, a, b, abaa, }
    Non-terminal Symbols: {A, B, C, }

    Rules:
        (0) A -> bBa
        (1) B -> abaa
        (2) C -> bb

    Original file: test.cfg
```

RemoveRuleCommand:

```
>>>removerule 0 1
Removed rule 1
>>>print 0
Grammar with id 0

    Terminal Symbols: {bb, a, b, abaa, }
    Non-terminal Symbols: {A, B, C, }

    Rules:
        (0) A -> bBa
        (1) C -> bb

    Original file: test.cfg
```

UnionCommand:

```
>>>union 0 1
New grammar saved with id - 2
>>>print 2
Grammar with id 2

    Terminal Symbols: {bb, a, b, abaa, }
    Non-terminal Symbols: {A, B, C, }

    Rules:
        (0) A -> bBa
        (1) C -> bb

    Original file: N/A
```

Глава 6. Заключение

6.1. Обобщение на изпълнението на началните цели

Имплементирани са решения на зададените изисквания и команди за работене с контекстно-свободна граматика и е създадена програма с конзолен интерфейс чрез който потребителя взаимодейства с програмата и може да изпълнява зададените команди

6.2 Използвана литература

Sankar V. Understanding Automata, Formal Languages and Grammar-Alpha Science International, 2020

Cocke–Younger–Kasami (CYK) Algorithm. 2023. https:www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/