|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | |
| федеральное государственное бюджетное образовательное учреждение | | | | |
| высшего образования | | | | |
| «Балтийский государственный технический университет | | | | |
| «ВОЕНМЕХ» им. Д.Ф. Устинова» | | | | |
| (БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова») | | | | |
| БГТУ.СМК-Ф-4.2-К5-01 | | | | |
| Факультет | | | О |  | Информационные и управляющие системы |
|  |  |  | шифр |  | наименование |
| Кафедра | | | О5 |  | Информационные системы и программная инженерия |
|  |  |  | шифр |  | наименование |
| Дисциплина | |  | Системное программное обеспечение | | |

КУРСОВАЯ РАБОТА

на тему

«Интерпретатор учебного языка»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | | | | | | |  | ЗИ591 |
|  |  | Митин А.Г. | | | | | | |
|  |  | Фамилия И.О. | | | | |  |  |
|  |  | **РУКОВОДИТЕЛЬ** | | | | | | |
| Никитин С. С. | | | | | | |  |  |
| Фамилия И.О. | | | | |  |  | Подпись | |
| Оценка | | | | | | |  |  |
| « » | | |  |  |  |  |  | 2021 г. |

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc84206973)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 3](#_Toc84206974)

[ВВЕДЕНИЕ 4](#_Toc84206975)

[1 Описание исходного языка 5](#_Toc84206976)

[2 Структура программы 7](#_Toc84206977)

[3 Техническая реализация и описание алгоритмов 10](#_Toc84206978)

[4 Результат тестирования программы 13](#_Toc84206979)

[ЗАКЛЮЧЕНИЕ 14](#_Toc84206980)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15](#_Toc84206981)

[ПРИЛОЖЕНИЕ 16](#_Toc84206982)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете применяют следующие сокращения и обозначения.

**КА** – конечный автомат.

# ВВЕДЕНИЕ

Целью курсовой работы является реализация интерпретатора учебного языка. Интерпретатор должен представлять собой упрощённую версию калькулятора, поддерживающего помимо основных арифметических операций, возможность изменения приоритетов операций (вычисление подвыражений), а также простые тригонометрические функции.

# 1 Описание исходного языка

Исходный язык представляет собой простейший калькулятор с возможностью вычисления выражений простых арифметический операций: сложения, вычитания, умножения и деления, а также вычисления выражений и тригонометрических функций синуса и косинуса.

Алфавит языка ограничен символами:

* Цифры: ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘0’
* Разделитель целой и дробной части: ‘.’
* Знаки арифметических операций: ‘+’, ‘-’, ‘\*’, ‘/’
* Символы изменения приоритета вычислений: ‘(’, ‘)’
* Символы, составляющие наименование тригонометрических функций:
  + Синус: ‘с’, ‘и’, ‘н’
  + Косинус: ‘к’, ’о’, ’с’

Одно выражение должно быть написано на одной строке, допускается несколько строк в одном файле. Выражения не должны содержать пробелов и пустых строк. Поддерживается как запись целых чисел, так и чисел с целой и дробной частью. Результат вычисления выражений всегда выводится с дробной частью.

Язык поддерживает правильный (арифметический) приоритет операторов и тригонометрических функций.

Грамматика исходного языка описывается следующими продукциями

expr -> expr + term | expr - term | term

term -> term \* func | term / func | func

func -> func(factor) | factor

factor -> digit | (expr)

Правило factor описывает число, либо подвыражение в скобках.

Правило func описывает либо тригонометрическую функцию, в которой в скобках должно содержаться число либо выражение.

Правило term содержит вычисления умножения и сложения в порядке приоритета.

Правило expr содержит самые низкие по приоритету операции - сложения и вычитания. Является стартовой продукцией грамматики.

# 2 Структура программы

Интерпретатор реализован на языке программирования Java SE 11 версии. Структура интерпретатора реализована с учётом требований и ограничений объектно-ориентированной парадигмы. Каждая сущность, задействованная в работе интерпретатора, описана с помощью классов, их атрибутов и методов.

Классы организованы в пакеты (package) для упорядочивания и соблюдения общей иерархии программы. Помимо стандартных пакетов Java определены следующие:

* ru.voenmeh.amitin.interpreter - содержит основные классы интерпретатора, представляющие собой различные фазы трансляции.
* ru.voenmeh.amitin.interpreter.exceptions - содержит специфичные для интерпретатора исключения.
* ru.voenmeh.amitin.interpreter.helpers - содержит вспомогательные классы, необходимые для функционирования программы. К ним относятся перечисление идентификаторов токенов и класс, описывающий структуру токена.
* ru.voenmeh.amitin.stateMachines - содержит описание всех конечных автоматов, а также менеджера конечных автоматов, управляющих их состоянием и считывающим результат.

Программа содержит следующие классы

* Основные классы:
* **Initializer** - класс-инициализатор, содержащий точку входа программы, а также осуществляющий первичную проверку файла на возможность чтения
* **Interpreter** - основной класс программы, инициализирующий фазы компиляции и выводящий результат вычисления выражения на экран
* **Lexer** - лексический анализатор
* **Parser** - синтаксический анализатор. Также непосредственно осуществляет вычисление выражения
* **StateMachineManager** - класс, управляющий конечными автоматами. В его обязанности входит переключение состояния автоматов и считывание результата работы с каждого
* Конечные автоматы:
* **StateMachine** - абстрактный класс-шаблон, содержащий описание атрибутов и методов конечного автомата. Все классы, реализующие КА, наследуют от этого класса.
* **DigitsStateMachine** - числа
* **Operators1StateMachine** - арифметические операторы сложения и вычитания
* **Operators2StateMachine** - арифметические операторы умножения и деления
* **LeftBracketStateMachine** - левая (открывающая скобка)
* **RightBracketStateMachine** - правая (закрывающая) скобка
* **SineStateMachine** - тригонометрическая функция вычисления синуса
* **CosineStateMachine** - тригонометрическая функция вычисления косинуса
* Вспомогательные классы:
* **Token** - описание структуры токена, используется в качестве строки таблицы символов
* **TokenIdentificators** - перечисление, содержащее все возможные идентификаторы токенов
* Исключения:
* **UnrecognizedTokenException** - возникает когда ни один автомат не распознал текущую лексему, то есть выявлена лексическая ошибка
* **SyntaxErrorException** - возникает в работе синтаксического анализатора при несоответствии потока токенов заданной продукции, то есть выявлена синтаксическая ошибка

Структура классов и отношения между ними представлены на диаграмме на рисунке 1.

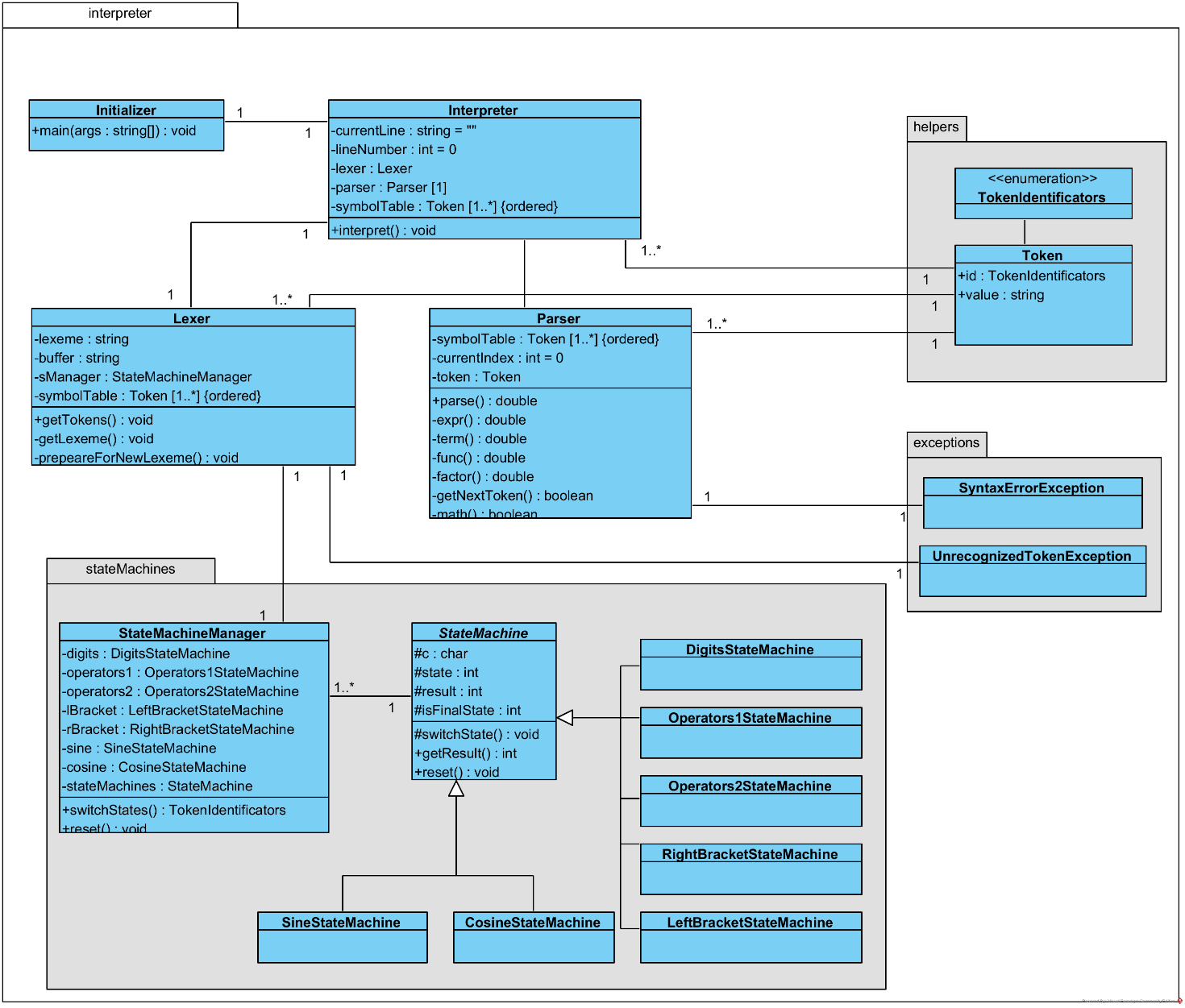


Рисунок 1 - Диаграмма классов

# 3 Техническая реализация и описание алгоритмов

Выражения исходного языка должны быть определены в файле “lib/Source.myscript”. Точкой входа программы является метод main() класса Initializer. При запуске приложения определяется существование файла по заданному пути и осуществляется проверка на возможность чтения файла. Если все проверки прошли успешно, то путь к файлу передаётся в основной класс программы Interpreter.

Класс Interpreter открывает файл на чтение и запускает цикл построчное считывание. Если строка успешно прочитана, то она передаётся в лексический анализатор Lexer.

Лексический анализатор копирует переданную строку в буфер, и начинает распознавание лексем, посимвольно передавая содержание буфера в менеджер конечных автоматов StateMachineManager.

Менеджер КА создаёт экземпляры классов всех конечных автоматов, определённых в программе. При получении очередного символа менеджер передаёт этот символ в каждый конечный автомат, а затем получает результат работы каждого КА. Если все автоматы вернули признак невозможности перехода в следующее состояние согласно полученному символу, причём текущее состояние не является финальным, то менеджер КА передаёт в лексический анализатор специальный идентификатор UNIDENTIFIED, обозначающий, что лексему не удалось распознать. Затем менеджер производит попытку определить наличие конечных автоматов, которые не смогли перейти в следующее состояние, но завершили работу в финальном. Если такой автомат был найден, то менеджер определяет тип автомата и возвращает в лексический анализатор идентификатор токена распознанной лексемы. Если предыдущие два условия не были выполнены, то это означает что автоматы ещё не завершили работу, и менеджер ожидает следующего символа от лексического анализатора.

Лексический анализатор принимает результат работы от менеджера КА. Если вернулся специальный идентификатор UNIDENTIFIED, обозначающий невозможность распознавания лексемы, то вызывается исключение UnrecognizedTokenException, которое отображается интерпретатором как результат вычисления текущей строки. Если от менеджера не был получен ни один идентификатор токена, то анализатор перемещает последний переданный символ из буфера в специальную переменную, предназначенную для хранения лексемы, и передаёт в менеджер конечных автоматов следующий символ. Если от менеджера вернулся идентификатор (отличный от UNIDENTIFIED), то лексический анализатор добавляет новую строку в таблицу символов, в которую записывает полученный идентификатор токена и его значение (саму лексему). Таким образом, токены записываются в таблицу символов по мере последовательного распознавания лексем, что позволяет считать таблицу символов упорядоченным списком (массивом) токенов.

После окончания считывания всех символов из буфера, лексический анализатор возвращает управление интерпретатору.

Интерпретатор получает из лексического анализатора ссылку на таблицу символов, которую передаёт в класс Parser, представляющий собой реализацию синтаксического анализатора.

Синтаксический анализатор реализован по методу рекурсивного спуска. Каждая продукция грамматики реализована в виде отдельной функции, которая в свою очередь вызывает функцию следующей продукции, и так далее до самой последней функции, обрабатывающей терминальные символы. Синтаксический анализатор принимает на вход таблицу символов, последовательно извлекает из неё токены, и направляет их в функцию expr(), представляющую собой реализацию стартового нетерминала грамматики исходного языка. Каждая функция также вычисляет значение выражения, и возвращает вычисленное значение обратно в вызвавшую функцию, которая в свою очередь вычисляет значение выражения на своём уровне. При возврате управления в главную функцию parse() анализируется все ли токены были обработаны. Если имеются необработанные токены, то вызывается синтаксическая ошибка. Если таких токенов нет, то управление вместе с вычисленным значение передаётся обратно в интерпретатор.

Интерпретатор получает значение выражения из синтаксического анализатора и выводит его на экран вместе с исходным выражением. В случае возврата исключения на любой фазе анализа отображается текст исключения. Независимо от полученного результата (вычисленное значение или исключение), интерпретатор переходит на следующую строку исходного файла. При обработке последней строки интерпретатор освобождает ресурсы, использовавшихся для чтения файла, и завершает работу программы.

# 4 Результат тестирования программы

Программа была проверена на тестовом наборе данных, содержащем как лексически и синтаксически верные выражения, так и заведомо ошибочные. Результаты тестирования представлены в таблице 1.

|  |  |  |
| --- | --- | --- |
| **Выражение** | **Ожидаемый результат** | **Результат работы программы** |
| 25+38\*син(18) | -3.5375153773236896 | 25+38\*син(18) = -3.5375153773236896 |
| (16+2.35)-50 | -31.65 | (16+2.35)-50 = -31.65 |
| 23+кос13) | Синтаксическая ошибка | Error at line 3: Syntax error: "(" expected |
| sin(90) | Лексическая ошибка | Error at line 4: Unrecognized token "sin(" |
| кос(3)-2 | -2.989992496600445 | кос(3)-2 = -2.989992496600445 |
| 2+ | Синтаксическая ошибка | Error at line 6: Syntax error |
| 45-(94.6\*45 | Синтаксическая ошибка | Error at line 7: Syntax Error: ")" expected |
| 56.3+-10 | Синтаксическая ошибка | Error at line 8: Syntax error |

Таблица 1 - Результаты тестирования программы

# ЗАКЛЮЧЕНИЕ

Любой вид транслятора является, пожалуй, одним из самых сложных видов программ в современном программировании. Так как трансляторы являются по сути фундаментом процесса разработки без которых невозможно представить программирование на языках высокого уровня, то к ним предъявляются повышенные требования в части производительности и качества, что порождает решения с использование довольно сложных алгоритмов. Тем не менее решения и алгоритмы, использующиеся для реализации трансляторов, также могут быть полезны и в других предметных областях.

В ходе работы были получены навыки проектирования и реализации простейшего интерпретатора учебного языка. Несмотря на относительную простоту реализации, в ходе проекта использовались основные принципы построения трансляторов, включая разделение на основные подпрограммы начальной фазы трансляции – лексический и синтаксический анализаторы.

В итоге удалось реализовать программу согласно заданию, приложение успешно прошло итоговое тестирование. Таким образом, цели курсовой работы были достигнуты.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Джеффри Д. Ульман, Альфред Ахо, Моника С. Лам, Рави Сети (перевод Красиков И.В.).* Компиляторы: принципы, технологии и инструментарий. – М.: Диалектика, 2018. – 1186 с.
2. Компиляция.1: лексер [Электронный ресурс] URL: https://habr.com/en/post/99162/ (дата обращения 01.09.2021)
3. Компиляция.2: грамматики [Электронный ресурс] URL: https://habr.com/en/post/99298/ (дата обращения 02.09.2021)
4. Еще раз про разбор выражений методом рекурсивного спуска [Электронный ресурс] URL: https://habr.com/en/post/489534/ (дата обращения 07.09.2021)

# ПРИЛОЖЕНИЕ

*Текст программы*

Файл Initializer.java

*package ru.voenmeh.amitin.interpreter;*

*import java.io.File;*

*import java.nio.file.Paths;*

*/\*Инициализатор. Отвечает за работу с файлами и вызов интерпретатора*

*Вход - путь до файла*

*\*/*

*public class Initializer {*

*public static void main(String[] args) {*

*File file;*

*file = new File(Paths.get("").toAbsolutePath() + "/lib/Source.myscript"); //определяем абсолютный путь до файла*

*if(file.canRead()){ //проверяем на возможность чтения*

*Interpreter interpreter = new Interpreter();*

*interpreter.interpret(file); //передаём файл в интерпретатор*

*}*

*else System.out.println("Can't read file");*

*}*

*}*

Файл Interpreter.java

*package ru.voenmeh.amitin.interpreter;*

*import java.io.BufferedReader;*

*import java.io.File;*

*import java.io.FileReader;*

*import java.util.ArrayList;*

*import ru.voenmeh.amitin.interpreter.helpers.Token;*

*/\*Интерпретатор. Инициализирует фазы анализа и разбора, и обрабатывает результат*

*Вход - файл со строками*

*Выход - вывод на экран результата вычисления*

*\*/*

*public class Interpreter {*

*private String currentLine = "";*

*private Integer lineNumber = 0;*

*private Lexer lexer;*

*private Parser parser;*

*private ArrayList<Token> symbolTable;*

*public void interpret(File file) {*

*try (FileReader fileReader = new FileReader(file);*

*BufferedReader bufferedReader = new BufferedReader(fileReader)) { //открываем файл на чтение*

*while ((currentLine = bufferedReader.readLine())!=null) { //построчное считывание*

*lineNumber++; //счётчик строк для диагностики ошибок*

*if (currentLine.isBlank()) continue; //пропускаем пустые строки*

*try {*

*lexer = new Lexer(); //инициализируем лексический анализатор*

*symbolTable = lexer.getTokens(currentLine); //получаем таблицу символов*

*parser = new Parser(symbolTable); //инициализируем синтаксический анализатор*

*Double result = parser.parse(); //получаем результат разбора и вычисления от синтаксического анализатора*

*System.out.println(currentLine + " = " + result); //выводим результат*

*} catch (Exception e) { //пробрасываем исключения из лексера и парсера для вывода ошибки с номером строки*

*System.out.println("Error at line " + lineNumber + ": " + e.getMessage());*

*}*

*}*

*} catch (Exception e) { //если файл не открылся*

*System.out.println(e.getMessage());*

*}*

*}*

*}*

Файл Lexer.java

*package ru.voenmeh.amitin.interpreter;*

*import java.util.ArrayList;*

*import ru.voenmeh.amitin.interpreter.exceptions.\*;*

*import ru.voenmeh.amitin.interpreter.helpers.\*;*

*import ru.voenmeh.amitin.stateMacnines.StateMachineManager;*

*/\*Лексический анализатор*

*Вход - строка символов*

*Выход - таблица символов, содержащую идентификаторы токенов и их значения*

*\*/*

*public class Lexer {*

*private StringBuilder lexeme = new StringBuilder(); //текущая лексема*

*private StringBuilder buffer; //поток нераспознанных символов*

*private StateMachineManager sManager = new StateMachineManager(); //менеджер КА*

*private ArrayList<Token> symbolTable = new ArrayList<>(); //таблица символов - результат работы лексера*

*/\*Основной метод лексера. Выделяет токены и записывает их в таблицу символов\*/*

*public ArrayList<Token> getTokens(String valueString) throws UnrecognizedTokenException{*

*buffer = new StringBuilder(valueString); //считываем строку в буфер*

*buffer.append("$"); //добавляем символ не в алфавите языка для обработки конечных лексем в буфере*

*while (buffer.length() > 1) { //пока в буфере есть символы для распознавания*

*prepareForNewLexeme(); //подготавливаем КА*

*getLexeme(); //получаем следующую лексему*

*}*

*return symbolTable; //возвращаем таблицу символов*

*}*

*/\*Получение следующей лексемы\*/*

*private void getLexeme() throws UnrecognizedTokenException{*

*char symbol;*

*TokenIdentificators currentID;*

*while (true) { //цикл по символам, пока не распознана лексема*

*symbol = buffer.charAt(0); //считываем первый символ из буфера*

*currentID = sManager.switchStates(symbol); //передаём символ для обработки в менеджер КА*

*if(currentID == null){ //если менеджер не вернул идентификатор, то лексема пока не распознана*

*lexeme.append(symbol); //добавляем символ к текущей распознаваемой лексеме*

*buffer.delete(0, 1); //удаляем символ из буфера*

*continue; //продолжаем распознвание, передаём следующий символ в менеджер КА*

*}*

*switch (currentID) { //если менеджер вернул идентификатор*

*case UNIDENTIFIED: //лексема не распознана ни одни КА, возвращаем лексическую ошибку*

*throw new UnrecognizedTokenException("Unrecognized token \"" + lexeme.append(symbol).toString() + "\"");*

*default: //лексема распознала*

*symbolTable.add(new Token(currentID, lexeme.toString())); //добавляем её идентификатор и значение в таблицу символов*

*break;*

*}*

*break;*

*}*

*}*

*/\*метод, подготавливающий лексический анализатор для распознавания следующей лексемы\*/*

*private void prepareForNewLexeme(){*

*lexeme.setLength(0); //очищаем текущую лексему*

*sManager.reset(); //сбрасываем все КА до начального состояния*

*}*

*}*

Файл Parser.java

*package ru.voenmeh.amitin.interpreter;*

*import java.util.ArrayList;*

*import ru.voenmeh.amitin.interpreter.helpers.Token;*

*import ru.voenmeh.amitin.interpreter.helpers.TokenIdentificators;*

*import ru.voenmeh.amitin.interpreter.exceptions.SyntaxErrorException;*

*/\* Синтаксический анализатор. Проверяет поток лексем на соответствие продукциям грамматики методом рекурсивного спуска*

*Также вычисляет результат*

*Вход - таблица символов*

*Выход - значение выражения текущей строки*

*\*/*

*public class Parser {*

*private ArrayList<Token> symbolTable; //таблица символов*

*private Integer currentIndex; //номер текущей строки таблицы символов (нумерация с 0, т.к. таблица символов в виде массива строк)*

*private Token token; //текущий токен (строка таблицы символов)*

*public Parser(ArrayList<Token> symbolTable){ //при создании экземпляра класса*

*this.symbolTable = symbolTable; //присваиваем ссылку на таблицу символов*

*currentIndex = 0; //устанавливаем указать на первую строку таблицы для начала разбора*

*token = symbolTable.get(currentIndex); //получаем первую строку таблицы символов*

*}*

*/\* Основной метод синтаксического анализатора. Начинает разбор потока токенов, вычисляет и возвращает результат выражения \*/*

*public Double parse() throws SyntaxErrorException{*

*Double result; //результат вычисления*

*result = expr(); //переходим к стартовой продукции*

*if(currentIndex < symbolTable.size()) //если были разобраны не все токены таблицы*

*throw new SyntaxErrorException("Syntax error"); //значит в строке есть продукции, не принадлежащие грамматике*

*return result; //возвращаем общий результат вычисления выражения строки*

*}*

*/\* expr -> term+digit | term-digit | term \*/*

*private Double expr() throws SyntaxErrorException{*

*Double result, right;*

*String operation;*

*result = term(); //спускаемся на продукцию ниже*

*while (currentIndex < symbolTable.size()) { //пока не кончились токены*

*operation = token.value; //сохраняем значение токена*

*if (match(TokenIdentificators.OP1)) { // если это сложение или вычитание*

*right = term(); //вычисляем правый операнд*

*switch (operation) { //выполняем арифметическую операцию*

*case "+":*

*result += right;*

*break;*

*case "-":*

*result -= right;*

*default:*

*break;*

*}*

*}else break;*

*}*

*return result;*

*}*

*/\* term-> func\*digit | func/digit | func \*/*

*private Double term() throws SyntaxErrorException{*

*Double result, right;*

*String operation;*

*result = func(); //спускаемся на продукцию ниже*

*while (currentIndex < symbolTable.size()) {// пока не кончились токены*

*operation = token.value; //сохраняем значение токена*

*if(match(TokenIdentificators.OP2)){ //если это операция умножения или деления*

*right = func(); //вычисляем правый операнд*

*switch (operation) { //выполняем артифметическую операцию*

*case "\*":*

*result \*= right;*

*break;*

*case "/":*

*result /= right;*

*break;*

*default:*

*break;*

*}*

*}*

*else break;*

*}*

*return result;*

*}*

*/\* func -> func(digit) | factor \*/*

*private Double func() throws SyntaxErrorException{*

*Double result;*

*String operation = token.value;*

*if(match(TokenIdentificators.TRIGONOMETRIC)){ //если идентификатор токена указывает на тригнометрическую функцию*

*if(match(TokenIdentificators.LBRACKET)){ //проверяем на следующий символ в продукции - открывающую скобку*

*result = factor(); //вычисляем значение выражения в скобках*

*if (match(TokenIdentificators.RBRACKET)){ //после выражения ожидает закрывающая скобка*

*switch (operation) { //вычисляем результат тригонометрической функции*

*case "син":*

*result = Math.sin(result);*

*break;*

*default: //только две функции реализовано, если это не синус, то вычисляем косинус*

*result = Math.cos(result);*

*break;*

*}*

*} else throw new SyntaxErrorException("Syntax error: \")\" expected"); //если скобки нет - возвращаем синтаксическую ошибку*

*}else throw new SyntaxErrorException("Syntax error: \"(\" expected"); //если открывающей скобки нет - возвращаем синтаксическую ошибку*

*}else result = factor(); //если это не функция, то переходим на продукцию ниже*

*return result;*

*}*

*/\* factor -> digit | (expr) \*/*

*private Double factor() throws SyntaxErrorException{*

*Double tokenValue;*

*Double result;*

*if (match(TokenIdentificators.LBRACKET)) { //если текущий токен - открывающая скобка*

*result = expr(); //то переходим к вычислению подвыражения*

*if (!match(TokenIdentificators.RBRACKET)) //выражение должно заканчиваться закрывающей скобкой*

*throw new SyntaxErrorException("Syntax Error: \")\" expected"); //если не так, то выводим синтаксическую ошибку*

*return result;*

*}*

*if (token.id == TokenIdentificators.NUM){ //если это число*

*tokenValue = Double.parseDouble(token.value); //преобразовываем в double*

*getNextToken(); //получаем следующий токен*

*return tokenValue; //возвращаем результат*

*}*

*else throw new SyntaxErrorException("Syntax error"); //спускаться ниже некуда, возвращаем синтаксическую ошибку*

*}*

*/\* перехода на следующую строку таблицы символов и взятие следующего токена \*/*

*private boolean getNextToken(){*

*if (++currentIndex+1 > symbolTable.size()) return false; //если это последний токен, то вернуть false*

*token = symbolTable.get(currentIndex); //взять токен по следующему индексу*

*return true;*

*}*

*/\* Проверка на соответствие текущего токена ожидаемому и взятие следующего токена \*/*

*private boolean match(TokenIdentificators tid){*

*if(tid == token.id){ //если текущий токен соответствует ожидаемому*

*getNextToken(); //то взять следующий токен*

*return true; //вернуть признак успешного сравнения*

*}*

*return false;*

*}*

*}*

Файл StateMachineManager.java

*package ru.voenmeh.amitin.stateMacnines;*

*import java.util.HashMap;*

*import java.util.Map;*

*import java.util.stream.Collectors;*

*import ru.voenmeh.amitin.interpreter.helpers.TokenIdentificators;*

*/\*Менеджер конечных автоматов. Управляет переключением КА и считывает результат с каждого для определения лексемы*

*Вход - символ*

*Выход - идентификатор токена, соответствующий успешно отработавшему КА*

*\*/*

*public class StateMachineManager {*

*/\* Определение всех конечных автоматов \*/*

*private DigitsStateMachine digits = new DigitsStateMachine(); //числа*

*private Operators1StateMachine operators1 = new Operators1StateMachine(); //арифметические операции сложения и вычитания*

*private Operators2StateMachine operators2 = new Operators2StateMachine(); //арифметические операторы умножения и деления*

*private LeftBracketStateMachine lBracket = new LeftBracketStateMachine(); //левая (открывающая) скобка*

*private RightBracketStateMachine rBracket = new RightBracketStateMachine(); //правая (закрывающая) скобка*

*private SineStateMachine sine = new SineStateMachine(); //тригонометрическая функция синус*

*private CosineStateMachine cosine = new CosineStateMachine(); //тригонометрическая функция косинус*

*private Map<StateMachine, Integer> stateMachines = new HashMap<>(); //таблица для отслеживания текущего состояния всех КА*

*public StateMachineManager(){*

*/\*Заполнение таблицы КА\*/*

*stateMachines.put(digits, 0);*

*stateMachines.put(operators1, 0);*

*stateMachines.put(operators2, 0);*

*stateMachines.put(lBracket, 0);*

*stateMachines.put(rBracket, 0);*

*stateMachines.put(sine, 0);*

*stateMachines.put(cosine, 0);*

*}*

*/\* Основной метод менеджера КА, меняет состояние КА в зависимости от символа, и считывает результат \*/*

*public TokenIdentificators switchStates(char c){*

*Map<StateMachine, Integer> filteredMap; //вспомогательная структура, для отслеживания состояния автоматов*

*stateMachines.forEach((stateMachine, state) ->{ //для каждого КА*

*if (state!=1 && state!=-1) //который ещё не завершил работу*

*stateMachines.put(stateMachine, stateMachine.getResult(c)); //получить результат обработки символа*

*});*

*/\* Все автоматы сломались, лексема не распознана \*/*

*filteredMap = stateMachines.entrySet().stream().*

*filter(a -> a.getValue() != -1).*

*collect(Collectors.toMap(e->e.getKey(), e -> e.getValue()));*

*if (filteredMap.isEmpty()) {*

*return TokenIdentificators.UNIDENTIFIED; //возвращаем для проброса ошибки*

*}*

*/\* Ожидаемый результат - один автомат завершился в финальном состоянии, лексема распознана \*/*

*filteredMap = stateMachines.entrySet().stream(). //фильтр по успешно отработавшим КА*

*filter(a -> a.getValue() == 1).*

*collect(Collectors.toMap(e -> e.getKey(), e -> e.getValue()));*

*if (filteredMap.size() == 1) { //если есть только один успешно отработавший КА*

*// то возвращаем идентификатор токена в зависимости от того какой КА успешно распознал лексему*

*if(filteredMap.containsKey(digits)) return TokenIdentificators.NUM;*

*if(filteredMap.containsKey(operators1)) return TokenIdentificators.OP1;*

*if(filteredMap.containsKey(operators2)) return TokenIdentificators.OP2;*

*if(filteredMap.containsKey(lBracket)) return TokenIdentificators.LBRACKET;*

*if(filteredMap.containsKey(rBracket)) return TokenIdentificators.RBRACKET;*

*if(filteredMap.containsKey(sine)) return TokenIdentificators.TRIGONOMETRIC;*

*if(filteredMap.containsKey(cosine)) return TokenIdentificators.TRIGONOMETRIC;*

*}*

*return null; //если нет успешно завершённых КА (нужно больше символов для распознания лексемы), то передаём лексеру сигнал для приёма ещё одного символа*

*}*

*/\* сброс всех КА до начального состояния\*/*

*public void reset() {*

*stateMachines.forEach((stateMachine, state) ->{*

*stateMachine.reset(); //вызываем метод сброса для конкретного КА*

*stateMachines.put(stateMachine, 0); //обнуляем текущее состояние КА в таблице*

*});*

*}*

*}*

Файл StateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\* Класс-шаблон для КА. Все автоматы являются наследниками класса \*/*

*public abstract class StateMachine {*

*protected Character c; //текущий распознаваемый символ*

*protected Integer state; //текущее состояние*

*protected Integer result; //результат обработки*

*protected Boolean isFinalState; //признак нахождения КА в финальном состоянии*

*StateMachine(){*

*reset(); //инициализация автомата при создании*

*}*

*//Основной метод всех автоматов. Переключает состояние автомата в зависимости от символа*

*protected abstract void switchState(char c);*

*/\**

*Результаты:*

*-1 - автомат не перешёл в следующее состояние, текущее состояние - не финальное*

*0 - автомат успешно перешёл в следующее состояние*

*1 - автомат не перешёл в следующее состояние, текущее состояние - финальное*

*\*/*

*public Integer getResult(char sym){*

*switchState(sym); //переключить состояние автомата*

*if (state == -1) { //если автомат не перешёл в следующее состояние*

*if(isFinalState) //то оценить последнее состояние*

*return 1; // если оно финальное, то передать информацию менеджеру КА об успешно распознанной лексеме*

*else return -1; //если не финальное, то лексема не известна автомату*

*} else return 0; // если автомат успешно перешёл в следующее состояние, то вернуть признак готовности приёма следующего символа*

*}*

*/\*сброс автомата для приёма следующей лексемы\*/*

*public void reset(){*

*state = 0; //установить состояние на начальное*

*result = -1; //установить результат работы на неопредеённый*

*isFinalState = false; //сбросить признак финального состояния*

*}*

*}*

Файл DigitsStateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*КА, распознающий числа. Распознаёт числа с дробной частью, не поддерживает экспоненциальную форму*

*Состояния автомата:*

*0 - начальное*

*1- цифра в целой части, может перейти в 1 и 2, финальное*

*2 - "." (разделитель между целой и дробной частью), может перейти в 3*

*3 - цифра в дробной части, может перейти к 3, финальное*

*\*/*

*public class DigitsStateMachine extends StateMachine{*

*public DigitsStateMachine(){*

*super();*

*}*

*protected void switchState(char sym){*

*if (Character.isDigit(sym)) {*

*switch (state) {*

*case 0:*

*state = 1;*

*isFinalState = true;*

*case 1:*

*state = 1;*

*isFinalState = true;*

*break;*

*case 2:*

*state = 3;*

*isFinalState = true;*

*break;*

*case 3:*

*state = 3;*

*isFinalState = true;*

*break;*

*default:*

*state = null;*

*}*

*}else{*

*if (sym == '.') {*

*switch (state) {*

*case 0:*

*state = -1;*

*isFinalState = true;*

*break;*

*case 1:*

*state = 2;*

*isFinalState = false;*

*break;*

*case 2:*

*state = -1;*

*isFinalState = false;*

*break;*

*case 3:*

*state = -1;*

*break;*

*default:*

*state = null;*

*}*

*}else state = -1;*

*}*

*}*

*}*

Файл Operators1StateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*ДКА, распознающий операторы сложения и вычитания*

*Состояния автомата:*

*0 - начальное*

*1- "+" или "-", перехода нет, финальное*

*\*/*

*public class Operators1StateMachine extends StateMachine{*

*public Operators1StateMachine(){*

*super();*

*}*

*protected void switchState(char sym) {*

*if (isOperator(sym)) {*

*switch (state) {*

*case 0:*

*state = 1;*

*isFinalState = true;*

*break;*

*case 1:*

*state = -1;*

*isFinalState = true;*

*break;*

*default:*

*state = -1;*

*break;*

*}*

*} else {*

*state = -1;*

*}*

*}*

*private boolean isOperator(char sym) {*

*switch (sym) {*

*case '+':*

*case '-':*

*return true;*

*default:*

*return false;*

*}*

*}*

*}*

Файл Operators2StateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*ДКА, распознающий операторы умножения и деления*

*Состояния автомата:*

*0 - начальное*

*1- "\*" или "/", перехода нет, финальное*

*\*/*

*public class Operators2StateMachine extends StateMachine{*

*public Operators2StateMachine(){*

*super();*

*}*

*protected void switchState(char sym) {*

*if (isOperator(sym)) {*

*switch (state) {*

*case 0:*

*state = 1;*

*isFinalState = true;*

*break;*

*case 1:*

*state = -1;*

*isFinalState = true;*

*break;*

*default:*

*state = -1;*

*break;*

*}*

*} else {*

*state = -1;*

*}*

*}*

*private boolean isOperator(char sym) {*

*switch (sym) {*

*case '\*':*

*case '/':*

*return true;*

*default:*

*return false;*

*}*

*}*

*}*

Файл LeftBracketStateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*ДКА, распознающий терминал открывающей скобки "("*

*Состояния автомата:*

*0 - начальное*

*1- "(", перехода нет, финальное*

*\*/*

*public class LeftBracketStateMachine extends StateMachine{*

*public LeftBracketStateMachine(){*

*super();*

*}*

*protected void switchState(char sym){*

*if(sym == '('){*

*switch (state) {*

*case 0:*

*state = 1;*

*isFinalState = true;*

*break;*

*default:*

*state = -1;*

*}*

*return;*

*}*

*state = -1;*

*}*

*}*

Файл LeftBracketStateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*ДКА, распознающий терминал закрывающей скобки ")"*

*Состояния автомата:*

*0 - начальное*

*1- ")", перехода нет, финальное*

*\*/*

*public class RightBracketStateMachine extends StateMachine{*

*public RightBracketStateMachine(){*

*super();*

*}*

*protected void switchState(char sym){*

*if(sym == ')'){*

*switch (state) {*

*case 0:*

*state = 1;*

*isFinalState = true;*

*break;*

*default:*

*state = -1;*

*}*

*return;*

*}*

*state = -1;*

*}*

*}*

Файл SineStateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*ДКА, распознающий лексему "син"*

*Состояния автомата:*

*0 - начальное*

*1- "с", можете перейти к 2*

*2 - "и", может перейти к 3*

*3 - "н", финальное, перехода нет*

*\*/*

*public class SineStateMachine extends StateMachine{*

*public SineStateMachine(){*

*super();*

*}*

*protected void switchState(char sym){*

*if (Character.isLetter(sym)) {*

*if(sym == 'с'){*

*switch (state) {*

*case 0:*

*state = 1;*

*//isFinalState = false;*

*break;*

*default:*

*state = -1;*

*//isFinalState = false;*

*break;*

*}*

*}*

*if(sym == 'и'){*

*switch (state) {*

*case 1:*

*state = 2;*

*//isFinalState = false;*

*break;*

*default:*

*state = -1;*

*//isFinalState = false;*

*break;*

*}*

*}*

*if(sym == 'н'){*

*switch (state) {*

*case 2:*

*state = 3;*

*isFinalState = true;*

*break;*

*default:*

*state = -1;*

*isFinalState = true;*

*break;*

*}*

*}*

*return;*

*}*

*state = -1;*

*}*

*}*

Файл CosineStateMachine.java

*package ru.voenmeh.amitin.stateMacnines;*

*/\*ДКА, распознающий лексему "кос"*

*Состояния автомата:*

*0 - начальное*

*1- "к", можете перейти к 2*

*2 - "о", может перейти к 3*

*3 - "с", финальное, перехода нет*

*\*/*

*public class CosineStateMachine extends StateMachine{*

*public CosineStateMachine(){*

*super();*

*}*

*protected void switchState(char sym){*

*if (Character.isLetter(sym)) {*

*if(sym == 'к'){*

*switch (state) {*

*case 0:*

*state = 1;*

*isFinalState = false;*

*break;*

*default:*

*state = -1;*

*isFinalState = false;*

*break;*

*}*

*}*

*if(sym == 'о'){*

*switch (state) {*

*case 1:*

*state = 2;*

*break;*

*default:*

*state = -1;*

*break;*

*}*

*}*

*if(sym == 'с'){*

*switch (state) {*

*case 2:*

*state = 3;*

*isFinalState = true;*

*break;*

*default:*

*state = -1;*

*break;*

*}*

*}*

*return;*

*}*

*state = -1;*

*}*

*}*

Файл Token.java

*package ru.voenmeh.amitin.interpreter.helpers;*

*/\* Строка таблицы символов, содержит идентификатор и значение токена \*/*

*public class Token {*

*public final TokenIdentificators id;*

*public final String value;*

*public Token(TokenIdentificators id, String value){*

*this.id = id;*

*this.value = value;*

*}*

*}*

Файл TokenIdentificators.java

*package ru.voenmeh.amitin.interpreter.helpers;*

*/\* Перечисление, содержащее идентификаторы токенов \*/*

*public enum TokenIdentificators {*

*NUM, //число*

*OP1, //операторы сложения и вычитания*

*OP2, //операторы умножения и деления*

*LBRACKET, //левая (открывающия скобка)*

*RBRACKET, //правая (закрывающая) скобка*

*TRIGONOMETRIC, //тригонометрическая функция*

*UNIDENTIFIED; //нераспознанный токен для выброса исключения*

*}*

Файл SyntaxErrorException.java

*package ru.voenmeh.amitin.interpreter.exceptions;*

*/\* Ошибки в работе синтаксического анализатора \*/*

*public class SyntaxErrorException extends Exception{*

*public SyntaxErrorException(String message){*

*super(message);*

*}*

*}*

Файл UnrecognizedTokenException.java

*package ru.voenmeh.amitin.interpreter.exceptions;*

*/\* Ошибки в работе лексического анализатора \*/*

*public class UnrecognizedTokenException extends Exception{*

*public UnrecognizedTokenException(String message){*

*super(message);*

*}*

*}*