# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc84447033)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 2](#_Toc84447034)

[1 Описание исходного языка 3](#_Toc84447035)

[2 Структура программы 5](#_Toc84447036)

[3 Техническая реализация и описание алгоритмов 8](#_Toc84447037)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете применяют следующие сокращения и обозначения.

**КА** – конечный автомат.

# 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.

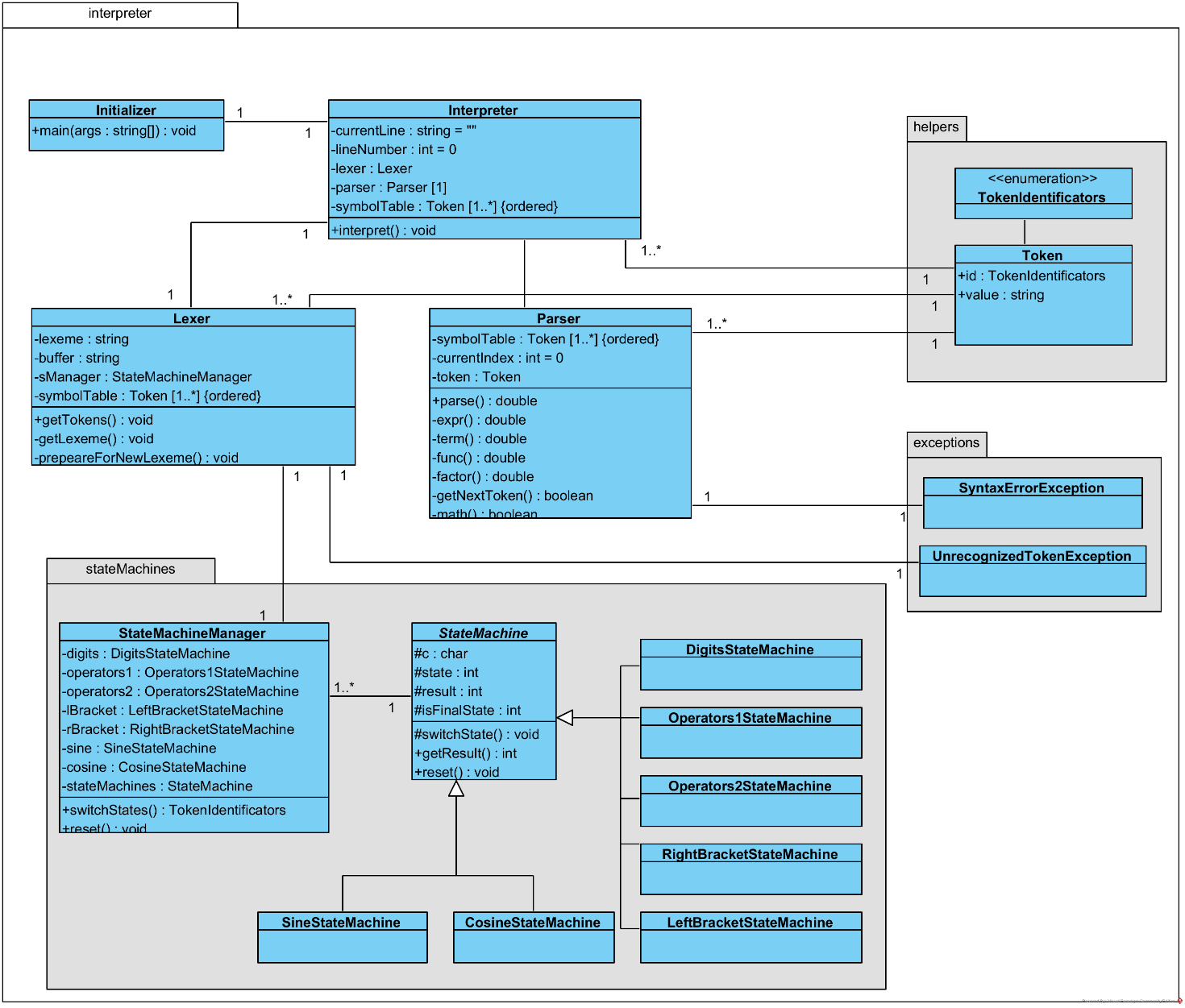


Рисунок 1 - Диаграмма классов

# 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.