Содержание

[Формулировка задачи 2](#_Toc107067397)

[Ход работы 3](#_Toc107067398)

[Общие положения 3](#_Toc107067399)

[Лексер 5](#_Toc107067400)

[Синтаксический анализатор 6](#_Toc107067401)

[Конфликты грамматики 8](#_Toc107067402)

[Ошибки и предупреждения 9](#_Toc107067403)

[Вывод 10](#_Toc107067404)

[Приложения 11](#_Toc107067405)

# Формулировка задачи

Необходимо создать лексический анализатор, проверяющий корректность кода Java, находящегося в некотором текстовом файле. Поскольку данная работа выполняется мной одиночно, рассмотрению подлежит не весь синтаксис языка Java, а лишь его часть, содержащая наиболее популярные конструкции.

# Ход работы

## Общие положения

При построении инструмента парсинга нужно было выполнить следующий ряд задач:

* Найти/составить набор лексем языка Java
* Составить набор правил, соответствующий наиболее популярным (наиболее используемым) конструкциям данного языка
* Проверить созданный инструмент на ряде примеров

Первый шаг был достигнут путём нахождения информации в открытых источниках, а также при помощи изучения ряда программ. Эти же два подхода применялись и для выполнения второго пункта. Ряд тестовых выборок брался из интернета.

Охвату подлежит не весь синтаксис языка. Вот список случаев, токенов и выражений, подлежащих анализу:

* Команды «package» и «import». Является неким аналогом директив и библиотек в С++. Состоит из регулярного одноимённого выражения и ряда идентификаторов, разделяемых точками.
* Непосредственно, сами классы.
* Методы и переменные, находящиеся в теле класса. У них должны быть маркеры доступа, типы данных и идентификаторы – это минимальный набор.
* Работа с методами и переменными, находящимися внутри метода.
* Разнообразные выражения, как численные, так и работа с переменными. Так же рассмотрению подлежит множество регулярных выражений.
* Конструкции циклов и условных операторов.
* Другие команды (try-catch-finally, return и т.д.)
* Также стоит отметить некоторые тонкости, затронутые в данной грамматике: рассмотрению подлежали комбинации модификаторов доступа, некий особый вид выражений – тернарные выражения, оператор new, обработка массивов и применение приведения типов данных.

## Лексер

В первую очередь, до написания какого-либо кода, были изучены открытые источники, примеры программ, и был составлен список ключевых слов языка Java. Данный список (а также список сгруппированных ключевых слов по принадлежности к определённым классам) предоставлен в Приложении 1.

Изначально парсер строился на лексере, реализуемом мной самостоятельно в функции yylex(). Но в процессе проектирования объём и сложность лексера возрастали, поэтому было принято решение использовать лексер «flex» и его возможности. Соответственно, парсер стал разбит на две части – «.l» и «.y». Данные файлы содержат, соответственно, набор лексем (токенов) и набор правил для их обработки.

Соответственно, теперь реализации подлежал только набор правил грамматики. Токены же теперь получаются flex-ом в неком «автоматическом» режиме.

## Синтаксический анализатор

Синтаксический анализатор представляет собой набор правил, позволяющих производить разбор входящего потока данных. При некорректной последовательности входных токенов, на которые разбивается подаваемый текст, анализатор синтаксиса должен выдавать ошибку. Для быстрого построения эффективного анализатора необходимо составить хотя бы простейшую блок-схему общей структуры языка. Чем такая схема сложнее и детальнее – тем проще строить анализатор и тем эффективнее он будет.

Перед построением блок-схемы был произведён поверхностный анализ примеров кодов и были замечены следующие закономерности:

* Код языка Java можно разделить на уровни. На данных уровнях должны встречаться определённые типы токенов. Можно заметить, что у двух наиболее «приоритетных» структурах (а таковыми являются «объявления классов» и «объявления методов») наблюдаются сходства. А именно, последовательно идёт маркер доступа. После, у класса следует токен «class», а у метода – маркер «static». Далее следует токен «идентификатор», представляемый последовательностью букв и цифр, но обязательно начинающийся с буквы или нижнего подчёркивания. В редких случаях можно наблюдать добавление таких структур, как, например, «расширение» (Extend) или «имплементация» (Implements). Также можно наблюдать отсутствие модификатора доступа или же модификатора «static» в определённых случаях. Исходя из всех этих особенностей был сделан вывод о некой вложенности задаваемых модификаторов и маркеров. Причём вложенность является приоритетной и, например, маркер «static» никак не может идти перед модификатором доступа. Или же имя (идентификатор) класса/метода никак не может идти в самую первую очередь кроме случаев вызова метода или же обращения к классу.
* У Java есть возможность подключения пакетов к проекту при помощи директивы «import», а также возможность указания того, что класс принадлежит определенному пакету при помощи «package». Данные ключевые слова можно встретить только в начале файла. Они встречаются исключительно до обращения к классам, методам и переменным.
* К переменным, методам, классам зачастую приходится обращаться через ряд идентификаторов (System.out.println). Команды такого рода представляют собой иерархическое включение пакетов, классов, методов и переменных. Данные идентификаторы указываются через «точку». Поэтому при проектировании тел методов нужно учитывать то, что вызов метода или обращение к переменной может происходить посредством иерархического прохода через другие идентификаторы.

В общем и целом, это все особенности языка Java. Все остальные конструкции в той или иной мере совпадают с общими взглядами на структуру сложных высокоуровневых языков.

Полученная в результате анализа блок-схема имеет следующий вид:

Diagram

Description automatically generated

Рисунок 1. Составленное общее древо граматики.

## Конфликты грамматики

В созданной грамматике присутствовало довольно большое количество конфликтов: 43 сдвиг/свёртка и 286 свёртка/свёртка. Так как грамматика изначально не задумывалась, как грамматика с минимальным или полностью отсутствующим количеством конфликтов, в процессе её создания этому не уделялось особого внимания, но изменениям подлежал ряд основных правил, содержащих в себе петли и неоднозначности. Результатом стало гораздо меньшее количество конфликтов: 53 сдвиг/свёртка.

Все эти конфликты возникают только в случаях неоднозначности просмотра следующей лексемы или свёртки. Один из популярных конфликтов этого типа – конфликт «кочующего else». Суть его заключается в том, что компилятору не понятно то, нужно ли сворачивать «if Expression» до правила «If» или же нужно заглянуть на лексему дальше. Ведь есть возможность иметь «else» в следующей лексеме. В таком случае, само собой, ветка разбора пойдёт по совершенно другому пути. Данный конфликт имеется в созданной грамматике, как и ряд других подобных ему. Разница лишь заключается в получаемых идентификаторах (это может быть и «try-catch-finally» и неоднозначные выражения, работы с переменными и тому подобные ситуации).

## Ошибки и предупреждения

Выводимым ошибкам и предупреждениям, с одной стороны, было выделено внимание. С другой стороны, его было выделено недостаточно много для того, чтобы иметь развёрнутый отчёт о каждой из ошибок. Поскольку по заданию необходимо лишь сообщать о том, корректен ли подаваемый код или нет – использовалась лишь стандартная функция yyerror(), которая срабатывает, если подаваемая лексема не подходит ни под одно рассматриваемое правило/состояние – тут всё стандартно.

Однако, номер строки с ошибкой, инкрементируемый при встрече терминала «\n», не всегда был корректен и можно было видеть сообщение об ошибке на строке, на которой её нет. Этому было уделено некоторое внимание и время и теперь номер строки с ошибкой сохраняется корректно. Более того, было замечено, что ошибки чаще всего возникают в токене «Идентификатор» – то есть при появлении каких-либо произвольных структур в языке. Поэтому считанная лексема сохраняется в строку и последняя считанная лексема выводится, как возможная ошибка.

# Вывод

В процессе выполнения данной курсовой работы мной были подробно изучены основные элементы синтаксиса языка Java, наиболее популярные конструкции. Все эти знания были применены с целью написания парсера входных данных, позволяющего определить корректность кода в большинстве случаев.

К сожалению, в языке Java, как и в любом другом языке высокого уровня имеются ситуации, порождающие конфликты (например, «кочующий else»). Также весьма проблематичной является безконфликтная грамматика выражений языка Java. Такие ситуации послужили причиной ряда конфликтов «сдвиг/свёртка». На основе множественных тестов был сделан вывод о том, что данные конфликты не влияют на корректность обработки.

# Приложения

abstract — абстрактный метод, абстрактный класс;

assert — отладка программы;

boolean — булев тип;

break — оператор для выхода из цикла или оператора switch;

byte — целочисленный тип;

case — переключатель оператора switch;

catch — оператор обработки исключений;

char — символьный тип;

class — класс;

const;

continue — прекращение итерации цикла;

default — ветвь оператора switch, метод по умолчанию;

do — оператор цикла do/while;

double — тип числа с плавающей точкой;

else — оператор принятия решений;

enum — перечисление;

extends — родитель класса;

final — класс, который нельзя расширить, метод, который нельзя переопределить или завершённый член данных;

finally — оператор обработки исключений;

float — тип числа с плавающей точкой;

for — тип цикла;

goto;

if — оператор принятия решений;

implements — интерфейсы, реализуемые классом;

import — импорт пакета;

instaceof — является ли объект экземпляром класса;

int — целочисленный тип;

interface — интерфейс;

long — целочисленный тип;

native — метод с кодом, написанным на другом языке;

new — выделение памяти для нового массива или объекта;

package — пакет классов;

private — модификатор доступа;

protected — модификатор доступа;

public — модификатор доступа;

return — выход из метода и возвращение значения;

short — целочисленный тип;

static — переменная или метод, общий для всех экземпляров класса;

strictfp — строгие правила для вычислений с плавающей точкой;

super — объект или конструктор суперкласса;

switch — оператор управления выполнения программы;

synchronized — доступ к участку кода только одному потоку;

this — неявный аргумент метода или конструктора класса;

throw — оператор обработки исключений;

throws — оператор обработки исключений;

transient — данные, которые не должны быть постоянными;

try — оператор обработки исключений;

void — метод не возвращает значений;

volatile — поле доступно нескольким потокам;

while — тип цикла.

Директивы:

import

package

типы данных:

byte

char

double

float

int

long

short

boolean

String

тип переменных:

var

тип метода:

void

модификаторы доступа:

public

private

protected

Модификаторы класса:

static

final

abstract

Модификаторы метода:

static

final

abstract

Модификаторы переменной:

static

final

Модификаторы потока:

synchronized

volatile

Циклы, условия и т.п.:

break

continue

do

for

while

case

default

switch