

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Кафедра «Информатики и защиты информации»

**Компилятор подмножества
процедурного языка в ассемблер**

Специальность: 10.05.04 – Информационно-аналитические системы
безопасности



Сергеев Никита Николаевич, ст. гр. ИСБ-118
Научный руководитель: к. т. н., доц. каф. ИЗИ
Монахов Ю. М.

г. Владимир 2021



Введение

Цель

Реализовать компилятор
подмножества
процедурного языка

Задачи

- Описание языка
- Контекстно свободная грамматика
- Лексический анализатор
- Таблица символов
- Синтаксический анализатор
- Генератор промежуточного кода

1

ЭТАПЫ РАЗРАБОТКИ КОМПИЛЯТОРА

Write once, run everywhere



Описание языка

Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под ее управлением.

Ключевые слова подмножества языка:

break; continue; while; double; int; if; else; public; static; class;
return; void; new; true; false; null.



Контекстно-свободная грамматика

```
program: mainClass classDeclaration*;

mainClass: 'class' identifier '{' mainMethod '}';

mainMethod: 'public' 'static' 'void' 'main' '(' 'String' ('[' ']'|'...') identifier ')' '{' statement+ '}';

classDeclaration: 'class' identifier '{' fieldDeclaration* methodDeclaration* '}';

parameter: type identifier;

fieldDeclaration: type identifier SC;

localDeclaration: type identifier SC;

methodDeclaration: 'public'? (type|'void') identifier '(' parameterList? ')' '{' methodBody '}';

parameterList: parameter(',' parameter)*;

methodBody: localDeclaration* statement* (returnStatement)?;

type: 'int' '[' ']'| 'boolean'| 'int' | 'char' | 'String' | 'double' | identifier;

identifier: Identifier;
```

Контекстно-свободная грамматика — частный случай формальной грамматики, у которой левые части всех продукций являются одиночными нетерминалами.



Лексический анализатор

Лексический анализ («токенизация») — процесс аналитического разбора входной последовательности символов на распознанные группы — лексемы — с целью получения на выходе идентифицированных последовательностей, называемых «токенами».

ANTLR — это мощный генератор парсеров для чтения, обработки, выполнения или перевода структурированных текстовых или двоичных файлов. Он широко используется для создания языков, инструментов и фреймворков.



Лексический анализатор

```
private static String[] makeLiteralNames() {  
    return new String[] {  
        null, "class", "{", "}", "public", "static", "void", "main",  
        "String", "...", ",", "int", "boolean", "char", "double",  
        "do", "while", "break", "continue", "System.out.println", "if",  
        "else", ".length", ".charAt", ".", "new", "this", "/",  
        "||", ">", "&&", "<", "+", "-", "*", "!", "[", "]",  
        "(", ")", "return", "=", null, ";"  
    };  
}
```



Синтаксический анализатор

```
private static String[] makeRuleNames() {  
    return new String[] {  
        "program", "mainClass", "mainMethod", "classDeclaration", "parameter",  
        "fieldDeclaration", "localDeclaration", "methodDeclaration", "parameterList",  
        "methodBody", "type", "identifier", "statement", "doWhileStatement",  
        "breakStatement", "continueStatement", "arrayAssignmentStatement", "variableAssignmentStatement",  
        "printStatement", "whileStatement", "ifElseStatement", "nestedStatement",  
        "returnStatement", "methodCallStatement", "expression", "methodCallParams"  
    };  
}
```

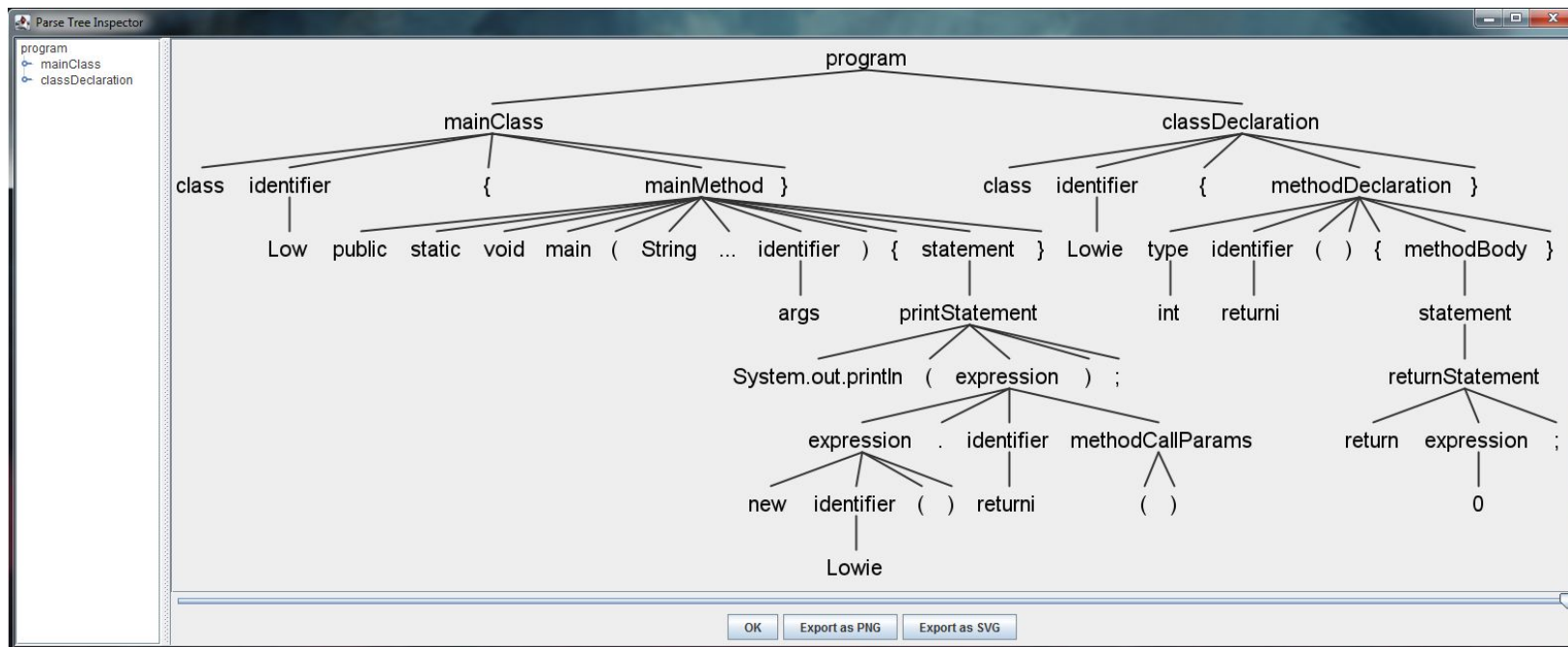



Синтаксический анализатор

```
Low.java x
1  class Low {
2
3      public static void main(String... args) {
4          System.out.println(new Lowie().returni());
5      }
6  }
7
8  class Lowie {
9
10     int returni() {
11         return 0;
12     }
13 }
14
```

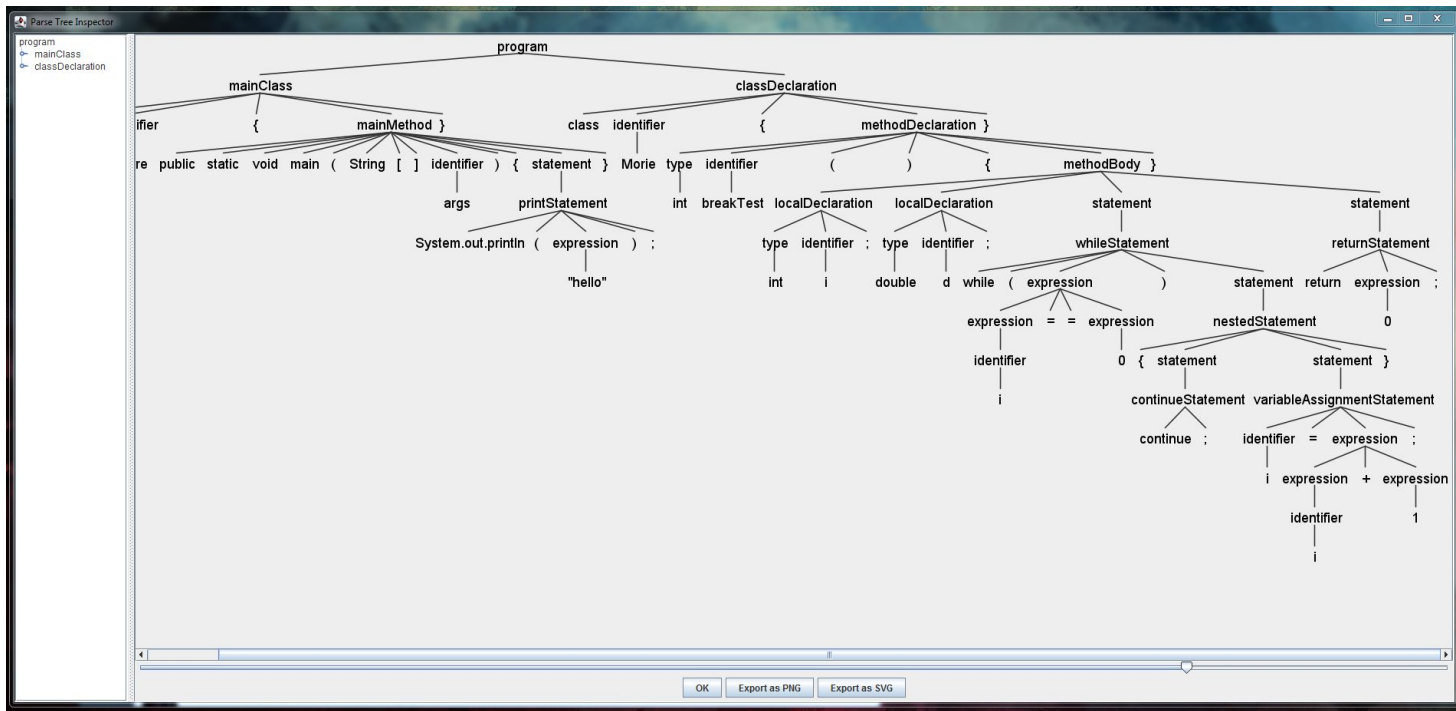


Синтаксический анализатор





Синтаксический анализатор





Синтаксический анализатор

```
Fail.java x TypeCheckVis.java x
1  class Fail {
2
3      public static void main(String[] args) {
4          System.out.println(new Failie().fail());
5      }
6
7
8  class Failie {
9
10     int fail() {
11         int i;
12         if (i > true) {}
13         return 0
14     }
15 }
16
```

Программа с ошибками
Fail.java

```
Run: Generation x
C:\Program Files\AdoptOpenJDK\jdk-15.0.2.7-hotspot\bin\java.exe
line 8:0 missing '}' at 'class'
line 14:4 no viable alternative at input 'return0}'
[err#0 - @12:12] >, >= operations can be done on 'int' types only!
```

Работа класса
TypeCheckVisitor.java



Таблица символов

Run: Generation x

"C:\Program Files\AdoptOpenJDK\jdk-15.0.2.7-hotspot\bin\java.exe" --enable-preview

Symbol Table:

ID	TYPE	SCOPE/NOTE
Lowie	Lowie	ClassNote
Low	Low	ClassNote
Low.main	null	MethodNote
Lowie.returni	int	MethodNote



Генерация промежуточного кода

Генератор промежуточного кода преобразует выходные данные синтаксического анализатора в Java Bytecode, который в последствии будет обработан интерпретатором в JVM.

Write once, run everywhere

Выходные данные из синтаксического анализатора передаются в транслятор, который продолжает обработку. Итогом работы транслятора является файл «название файла.tjp», в котором записывается промежуточный Java bytecode.



Генерация промежуточного кода

Байт-код Java — набор инструкций, исполняемых виртуальной машиной Java. Каждый код операции байт-кода — один байт; используются не все 256 возможных значений кодов операций, 51 из них зарезервированы для использования в будущем.

```
Codes.java x
1 package compiler.generation;
2
3 public interface Codes {
4
5     int CONST = 1;           // Push int
6     int STORE = 2;          // Pop valu
7     int LOAD = 3;           // Push int
8     int ADD = 4;             // Pop valu
9     int MULTI = 5;          // Pop valu
10    int SUB = 6;             // Pop valu
11    int DIV = 7;             // Pop valu
12    int ILT = 8;             // Pop valu
13    int NO = 9;              // Pop valu
14    int AND = 10;            // Pop valu
15    int OR = 11;             // Pop valu
16    int GOTO = 12;           // Jump to
17    int IF_FALSE = 13;       // Pop valu
18    int INVOKE = 14;         // Push cur
19    int RETURN = 15;         // Pop acti
20    int PRINT = 16;          // Pop valu
21    int STOP = 17;           // Executio
22 }
23
```









Трансляция в целевой код

```
case AND:
    if (data.pop() * data.pop() == 0)
        dataZeroPush();
    else dataOnePush();
    break;
case OR:
    if (data.pop() + data.pop() == 0)
        dataZeroPush();
    else dataOnePush();
    break;
case GOTO: // pc = argument
    number = Integer.parseInt(argument.toString());
    next = number - 1;
    break; // в конце происходит добавление переключателя
           // pc-1
case IF_FALSE: // v = pop(), let pc = argument if v = 0
    number = Integer.parseInt(argument.toString());
    if (data.pop() == 0)
        next = number - 1;
```

Машинный код — система команд (набор кодов операций) конкретной вычислительной машины, которая интерпретируется непосредственно процессором или микропрограммами этой вычислительной машины



Трансляция в целевой код

Имя	Дата изменения	Тип	Размер
 FacFib	21.05.2021 10:17	IntelliJ IDEA Com...	1 КБ
 FacFib.tjp	21.05.2021 10:51	Файл "TJP"	3 КБ
 Generation	21.05.2021 12:14	Файл "JAR"	2 182 КБ
 Interpreter	21.05.2021 12:14	Файл "JAR"	2 182 КБ
 While	21.05.2021 11:06	IntelliJ IDEA Com...	1 КБ
 While.tjp	21.05.2021 10:59	Файл "TJP"	2 КБ

```
C:\Windows\system32\cmd.exe

C:\Users\NIKITA SERGEEV\IdeaProjects\ZFULL\SergeevCompiler\
FacFib.tjp
120
55
1
```

```
Приложения  Места  Терминал

nikita@NIKITA: ~/Загрузки

(nikita@NIKITA)-[~/Загрузки]
$ java -jar --enable-preview Interpreter.jar While.tjp
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
0
0
0
0
```

2

РАБОТОСПОСОБНОСТЬ КОМПИЛЯТОРА И ПРИМЕРЫ

```
FacFib.java x
1  class FacFib {
2
3      public static void main(String[] a) {
4          System.out.println(new FacFibImpl().compute(5,10));
5      }
6  }
7
8  class FacFibImpl {
9      int compute(int first, int second) {
10         int fac;
11         int fib;
12         int returnValue;
13
14         if ( !(first < 1)  && !(second < 1) ) {
15             fac = this.computeFac(first);
16             System.out.println(fac);
17
18             fib = this.computeFib(second);
19             System.out.println(fib);
20             returnValue = 1;
21         }
22         else returnValue = 0;
23         return returnValue;
24     }
25
26     int computeFac(int num){
27         int fac;
28         if (num < 1) fac = 1;
29         else fac = num * (this.computeFac( num: num-1));
```



Примеры

```
Run: Interpreter x
"C:\Program Files\AdoptOpenJDK\jdk-15.0.2.7-hotspot\bin\"
720
89
1
Process finished with exit code 0
```

```
Run: Interpreter x
"C:\Program Files\AdoptOpenJDK\jdk-15.
120
55
1
Process finished with exit code 0
```

```
Run: Generation x
" C:\Program Files\AdoptOpenJDK\jdk-15.0.2.7-hotspot\bin\java.exe" --enable-preview "-javaagent:D:\JetBrains
How to run generation: java -jar <Generation.jar> <filePath.java>
How to run interpreter: java -jar <Interpreter.jar> <filePath.tjp>

Grammar of language:
program: mainClass classDeclaration*;

mainClass: 'class' identifier '{' mainMethod '}';
mainMethod: 'public' 'static' 'void' 'main' '(' 'String' '[' ']' '|' '...' identifier ')' '{' statement+ '}';
classDeclaration: 'class' identifier '{' fieldDeclaration* methodDeclaration* '}';
parameter: type identifier;
fieldDeclaration: type identifier SC;
localDeclaration: type identifier SC;
methodDeclaration: 'public'? (type 'void') identifier '(' parameterList? ')' '{' methodBody '}';
parameterList: parameter(',' parameter)*;
methodBody: localDeclaration* statement* (returnStatement)?;
type: 'int' '[' ']' | 'boolean' | 'int' | 'char' | 'String' | 'double' | identifier;
identifier: Identifier;
doWhileStatement: 'do' statement 'while' LP expression RP SC;
breakStatement: 'break' SC;
continueStatement: 'continue' SC;
arrayAssignmentStatement: identifier LSB expression RSB EQ expression SC;
variableAssignmentStatement: identifier EQ expression SC;
printStatement: 'System.out.println' LP(expression) RP SC;
whileStatement: 'while' LP expression RP statement;
ifElseStatement: 'if' LP expression RP statement ('else' statement)?;
nestedStatement: '{' statement* '}';
returnStatement: 'return' expression SC;
methodCallStatement: expression SC;

Process finished with exit code 0
```



**Спасибо за
внимание!**