Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №11
«Разработка парсеров на языке Java»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-21Б
Старовойтов А. И.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Овладение методом рекурсивного спуска для разработки парсеров по грамматике некоторого формального языка.

Задачи

№46:

```
<Stmt> ::= if ( IDENT ) <Stmt>
         | IDENT = NUMBER ;
         | { <Seq> }
<Seq>  ::= <Stmt> <Seq> | ε
```

Пример:

```
if (x) {
    x = 0;
    if (y) z = 10;
}
```

Решение

Test.java

parser/Parser.java

```
package parser;

import java.util.function.IntPredicate;

public class Parser {
    private Token sym;
    private String text;

    public static class SyntaxError extends Exception {
        public SyntaxError(String msg, Position pos) {
```

```
            super(String.format("Syntax error at %s: %s",
             ↪  pos, msg));
        }
    }

    public Parser(String text) {
        this.text = text;
    }

    public void parse() throws SyntaxError {
        sym = new Token(text);
        parseStmt();
    }

    private void expect(Tag tag) throws SyntaxError {
        if (sym.getTag() != tag) {
            sym.throwError(String.format("expected %s",
↪   tag));
        }
    }

    private void parseStmt() throws SyntaxError {
        switch (sym.getTag()) {
        case IF -> {
            System.out.println("<Stmt> ::= if ( IDENT )
             ↪  <Stmt>");
            sym = sym.next();
            expect(Tag.LPAREN);
            sym = sym.next();
            expect(Tag.IDENT);
            sym = sym.next();
            expect(Tag.RPAREN);
            sym = sym.next();
            parseStmt();
        }
        case IDENT -> {
            System.out.println("<Stmt> ::= IDENT = NUMBER
             ↪  ;");
            sym = sym.next();
            expect(Tag.EQUALSIGN);
```

```
                sym = sym.next();
                expect(Tag.NUMBER);
                sym = sym.next();
                expect(Tag.SEMICOLON);
                sym = sym.next();
            }
            case LCURLY -> {
                System.out.println("<Stmt> ::= { <Seq> }");
                sym = sym.next();
                parseSeq();
                expect(Tag.RCURLY);
                sym = sym.next();
            }
            default -> {
                sym.throwError("expected if statement,
 ↪  identifier or {");
            }
            }
        }

        private void parseSeq() throws SyntaxError {
            Tag tag = sym.getTag();
            if (tag == Tag.IF || tag == Tag.IDENT || tag ==
             ↪  Tag.LCURLY) {
                System.out.println("<Seq> ::= <Stmt> <Seq>");
                parseStmt();
                parseSeq();
            } else {
                System.out.println("<Seq> ::= ε");
            }
        }
    }

    enum Tag {
        LPAREN, RPAREN, IF, IDENT, NUMBER, LCURLY, RCURLY,
 ↪  SEMICOLON, EQUALSIGN, EOF;

        @Override
        public String toString() {
            return switch (this) {
```

```
            case LPAREN -> "'('";
            case RPAREN -> "')'";
            case IF -> "if";
            case IDENT -> "identifier";
            case NUMBER -> "number";
            case LCURLY -> "'{'";
            case RCURLY -> "'}'";
            case SEMICOLON -> "';'";
            case EQUALSIGN -> "'='";
            case EOF -> "end of file";
            };
        }
    }

    class Position {
        private String text;
        private int line;
        private int col;
        private int index;

        public static final int EOF = -1;

        public Position(String text) {
            this.text = text;
            this.line = 0;
            this.col = 0;
            this.index = 0;
        }

        private Position(String text, int line, int col, int
         ↪  index) {
            this.text = text;
            this.line = line;
            this.col = col;
            this.index = index;
        }

        public int getChar() {
            return index < text.length() ?
             ↪  text.codePointAt(index) : EOF;
```

```
    }

    public Position skip() {
        if (getChar() > 0xFFFF) {
            return new Position(text, line, col + 2, index +
             ↪  2);
        } else if (getChar() == '\n') {
            return new Position(text, line + 1, 0, index +
             ↪  1);
        } else {
            return new Position(text, line, col + 1, index +
             ↪  1);
        }
    }

    public Position skipWhile(IntPredicate p) {
        Position res = this;
        while (p.test(res.getChar())) {
            res = res.skip();
        }
        return res;
    }

    @Override
    public String toString() {
        return String.format("%d, %d", line, col);
    }
}

class Token {
    private Tag tag;
    private Position start;
    private Position end;

    private Token(Position start) throws Parser.SyntaxError
     ↪  {
        start = start.skipWhile(Character::isWhitespace);
        this.start = start;
        this.end = start.skip();
        switch (start.getChar()) {
```

```
case Position.EOF -> {
    this.tag = Tag.EOF;
}
case '(' -> {
    this.tag = Tag.LPAREN;
}
case ')' -> {
    this.tag = Tag.RPAREN;
}
case '{' -> {
    this.tag = Tag.LCURLY;
}
case '}' -> {
    this.tag = Tag.RCURLY;
}
case ';' -> {
    this.tag = Tag.SEMICOLON;
}
case '=' -> {
    this.tag = Tag.EQUALSIGN;
}
default -> {
    int c = start.getChar();
    if (Character.isAlphabetic(c)) {
        if (end.getChar() == 'f' &&
         ↪  !Character.isLetterOrDigit(end.skip().getChar()))
         ↪  {
            this.end = end.skip();
            this.tag = Tag.IF;
        } else {
            this.end =
             ↪  end.skipWhile(Character::isLetterOrDigit);
            this.tag = Tag.IDENT;
        }
    } else if (Character.isDigit(c)) {
        this.end =
         ↪  end.skipWhile(Character::isDigit);
        this.tag = Tag.NUMBER;
    } else {
        throwError("invalid character " +
         ↪  start.getChar());
```

```
            }
        }
        }
    }

    public Token(String text) throws Parser.SyntaxError {
        this(new Position(text));
    }

    public Token next() throws Parser.SyntaxError {
        return new Token(end);
    }

    public Tag getTag() {
        return tag;
    }

    public void throwError(String msg) throws
     ↪  Parser.SyntaxError {
        throw new Parser.SyntaxError(msg, start);
    }
}
```

tests.sh

```
#!/usr/bin/env bash

cat <<"EOF" | make
if (x12) {
    aoaoa = 01010;
    dadaya = 123345;
}
EOF

cat <<"EOF" | make
x = 1;
EOF

cat <<"EOF" | make
if (a)
```

```
   if (b)
       if (c)
           x = 123;
EOF

cat <<"EOF" | make
if (x123) {
   x123 = ;
}
EOF
```