Laboratory work nr.3
Lexer Scanner

Course: Formal Languages & Finite Automata

Student: Munteanu Ecaterina

Objectives:

1. Understand what lexical analysis is.
2. Get familiar with the inner workings of a lexer/scanner/tokenizer.
3. Implement a sample lexer and show how it works.

Implementation:

A lexer is the initial stage of a compiler or interpreter. It takes the source code as input and breaks it down into meaningful units called tokens. Tokens represent the smallest units of syntax in the programming language, such as keywords, identifiers, operators, literals, etc.

A token is a categorized block of text in the source code of a programming language. It is generated by the lexer and serves as input to the parser. Each token represents a lexical unit with a specific meaning in the programming language.

Defining an enumeration **TokenType** was the first step in this implementation. It will be used to represent the different types of tokens recognized by the lexer. Each token type corresponds to a specific lexical unit in the programming language, such as keywords, identifiers, operators, etc.

```
public enum TokenType {
    1 usage
    VAR,    // var keyword
    2 usages
    ID,     // Identifier
    1 usage
    ASSIGN, // =
    3 usages
    SEMI,   // ;
    1 usage
    OP,     // Arithmetic operator (+, -, *, /)
    1 usage
    LPAREN, // (
    1 usage
    RPAREN, // )
    1 usage
    PRINT,  // print keyword
    1 usage
    COMMA,  // ,
    1 usage
    STRING  // String literal
}
```

Implementing a **Token** class was the second step. It represents individual tokens generated by the lexer. Each token consists of a type (TokenType) and an optional value.

```
public class Token {
    3 usages
    TokenType type;
    3 usages
    String value;

    12 usages
    public Token(TokenType type, String value) {
        this.type = type;
        this.value = value;
    }

    @Override
    public String toString() {
        return String.format("%s:\'%s\'", type.name(), value != null ? value : "");
    }
}
```

Implementing a **Lexer** class was a more voluminous step. It will help us tokenize the input source code. The lexer scans the input text character by character and generates tokens based on predefined lexical rules. The main method here is **tokenize()**:

```
public List<Token> tokenize() {
    List<Token> tokens = new ArrayList<>();
    Token token = getNextToken();
    while (token.type != TokenType.SEMI) {
        tokens.add(token);
        token = getNextToken();
    }
    tokens.add(token);
    return tokens;
}
```

It is a petty straightforward method, analyzing one statement at a time, and stopping when a semicolon is met. The **getNextToken()** method helps us move along the statement and add new tokens to our list.

The **Main** class imitates a console, thus we can introduce as many statements as we want, but they will be analyzed separately, by pressing Enter to let the Lexer do its work.

```java
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter your statements (press Enter after each statement), and type 'exit' to end:");
        System.out.print("console> ");

        StringBuilder inputBuilder = new StringBuilder();
        String line;
        while (!(line = scanner.nextLine()).equals("exit")) {
            inputBuilder.append(line).append("\n");

            String text = inputBuilder.toString();
            Lexer lexer = new Lexer(text);
            List<Token> tokens = lexer.tokenize();

            System.out.println("Tokens: " + tokens);

            inputBuilder.setLength(0); // Clear the inputBuilder for the next statement

            System.out.print("\nconsole> ");
        }

        scanner.close();
    }
}
```

The result in the console looks as follows:

```
Enter your statements (press Enter after each statement), and type 'exit' to end:
console> var x = 101;
Tokens: [VAR:'var', ID:'x', ASSIGN:'=', ID:'101', SEMI:';']

console> print(y);
Tokens: [PRINT:'print', LPAREN:'(', ID:'y', RPAREN:')', SEMI:';']

console> var z = "haha";
Tokens: [VAR:'var', ID:'z', ASSIGN:'=', STRING:'haha', SEMI:';']

console> exit

Process finished with exit code 0
```

Conclusions:

In conclusion, the laboratory work focused on implementing a lexer for a simple programming language. The lexer serves as the first step in the compilation or interpretation process, breaking down the input source code into tokens for further processing. By understanding the theoretical concepts of lexers and applying them in the implementation, I successfully wrote a lexer capable of tokenizing input text according to predefined lexical rules. This practical exercise helped me understand lexers better and will help me work better on my semester project.