Simple Lexer in Java - Detailed Explanation

Overview of the Lexer

The lexer is responsible for converting a sequence of characters (source code) into a sequence of tokens. Tokens are the basic building blocks of the language, representing keywords, identifiers, operators, literals, etc.

Custom Modules and Types

1. **Token.java**
   - **Purpose**: Represents a single token with its type and value.
   - **Code**:

```
public class Token {
    private final TokenType type;
    private final String value;
    public Token(TokenType type, String value) {
        this.type = type;
        this.value = value;
    }
    public TokenType getType() {
        return type;
    }
    public String getValue() {
        return value;
    }
    @Override
    public String toString() {
        return "Token{" + "type=" + type + ", value='" + value + '\'' + '}';
    }
}
```

2. **TokenType.java**
    - **Purpose**: Enum defining different types of tokens.
    - **Code**:

```
public enum TokenType {
    IDENTIFIER,
    NUMBER,
    LET,
    EQUALS,
    SEMICOLON,
    EOF
}
```

Detailed Explanation of the Lexer

SimpleLexer.java

- **Purpose**: Tokenizes the input source code.
- **Code**:

```
public class SimpleLexer {
    private final String input;
    private int currentPosition = 0;
    public SimpleLexer(String input) {
        this.input = input;
    }
    public List<Token> tokenize() {
        List<Token> tokens = new ArrayList<>();
        while (currentPosition < input.length()) {
            char currentChar = input.charAt(currentPosition);
```

```
        if (Character.isWhitespace(currentChar)) {
            currentPosition++;
            continue;
        }
        if (Character.isLetter(currentChar)) {
            String identifier = readWhile(Character::isLetterOrDigit);
            tokens.add(new Token(TokenType.IDENTIFIER, identifier));
            continue;
        }
        if (Character.isDigit(currentChar)) {
            String number = readWhile(Character::isDigit);
            tokens.add(new Token(TokenType.NUMBER, number));
            continue;
        }
        switch (currentChar) {
            case '=':
                tokens.add(new Token(TokenType.EQUALS, "="));
                break;
            case ';':
                tokens.add(new Token(TokenType.SEMICOLON, ";"));
                break;
            default:
                throw new RuntimeException("Unexpected character: " + currentChar);
        }
        currentPosition++;
    }
    tokens.add(new Token(TokenType.EOF, ""));
    return tokens;
}
```

```
    private String readWhile(java.util.function.Predicate<Character> predicate) {
        StringBuilder result = new StringBuilder();
        while (currentPosition < input.length() && predicate.test(input.charAt(currentPosition))) {
            result.append(input.charAt(currentPosition));
            currentPosition++;
        }
        return result.toString();
    }
}
```

Explanation of Each Component

1. **Constructor: SimpleLexer(String input)**
    - **Purpose**: Initializes the lexer with the input source code.
    - **Input**: input - The source code to be tokenized.
    - **Usage**: SimpleLexer lexer = new SimpleLexer(sourceCode);
2. **Method: List<Token> tokenize()**
    - **Purpose**: Main method to tokenize the input source code.
    - **Input**: No direct input; operates on the input string provided during initialization.
    - **Output**: List<Token> - A list of tokens generated from the source code.
    - **Process**:
        - Iterates through each character of the input string.
        - **Whitespace Handling**: Skips whitespace characters.
        - **Identifier Handling**:
            - Uses Character.isLetter(currentChar) to detect the start of an identifier.
            - Calls readWhile(Character::isLetterOrDigit) to read the full identifier.
            - Adds a new Token of type IDENTIFIER to the token list.
        - **Number Handling**:
            - Uses Character.isDigit(currentChar) to detect the start of a number.
            - Calls readWhile(Character::isDigit) to read the full number.

        - Adds a new Token of type NUMBER to the token list.
    - **Operators and Punctuation**:
        - Directly identifies characters like = and ;.
        - Adds corresponding tokens to the token list.
    - **Unexpected Characters**: Throws an exception for any unexpected character.
    - **End of File**: Adds an EOF token at the end of the input.

3. **Method: String readWhile(java.util.function.Predicate<Character> predicate)**
    - **Purpose**: Reads characters from the input while they match a given condition.
    - **Input**: Predicate<Character> predicate - A functional interface that defines the condition.
    - **Output**: String - The string of characters that match the condition.
    - **Usage**: Used to read identifiers and numbers:
        - readWhile(Character::isLetterOrDigit)
        - readWhile(Character::isDigit)

Example Usage

Here's a practical example of how to use the SimpleLexer:

```
public class TinyLangDriver {
  public static void main(String[] args) {
    String sourceCode = "let x = 42;";
    SimpleLexer lexer = new SimpleLexer(sourceCode);
    List<Token> tokens = lexer.tokenize();

    for (Token token : tokens) {
      System.out.println(token);
    }
  }
}
```

Expected Output:

```
Token{type=IDENTIFIER, value='let'}
Token{type=IDENTIFIER, value='x'}
Token{type=EQUALS, value='='}
Token{type=NUMBER, value='42'}
Token{type=SEMICOLON, value=';'}
Token{type=EOF, value=''}
```

This output shows the sequence of tokens generated from the source code "let x = 42;".

In summary, we can define the whole process like this:

- The lexer is responsible for breaking down the source code into tokens.
- It uses character classification to identify different types of tokens.
- Tokens are created using the Token class, with types defined in the TokenType enum.
- The tokenize method performs the main tokenization process, iterating through the input string and generating tokens.
- The readWhile method helps read sequences of characters that match a given condition.