Project Specifications: Build Compiler using JavaCC for Java programming language

Project Overview

This project involves creating a compiler module using JavaCC to implement lexical analysis , parser grammar and Build AST for the Java programming language. The module will:

1. Perform lexical analysis by breaking the source code into tokens.
2. Parse the tokenized input to validate the syntax according to Java grammar.
3. Construct an Abstract Syntax Tree (AST) representing the program structure.
4. Take Java source code from text file as input.
5. Output either syntax validation results or error reports for invalid code and print the AST.

Objectives

- Develop a lexical analyzer using JavaCC to tokenize Java source code.
- Implement parser rules in JavaCC to validate Java grammar.
- Construct a clear AST representing program structure.
- Provide meaningful error messages for syntax violations.

Project Requirements

Functional Requirements

1. **Input File**: The program will take a Java source code from text file as input.
2. **Tokenization**:
   - Recognize and generate tokens for keywords (e.g., class, if, while), literals (e.g., integers, strings), identifiers, operators, and delimiters.
   - Handle comments (single-line // and multi-line /* ... */) and whitespace correctly by ignoring them.
3. **Parsing**:
   - Parse input using grammar rules for programming language like:
     - Variable declarations
     - Expressions and statements (e.g., if, for, while, return)
   - Validate syntactic correctness and ensure that the input conforms to Java grammar.
4. **AST Construction:**
   - Build an AST during parsing.
   - Represent nodes for variable declarations, expressions, statements, and control structures.
   - Provide a method to print the AST in a readable format.
5. **Error Reporting**:

- Display detailed error messages **specifying the line and column of syntax errors**.

6. **Output**:
   - If valid source code, print a success message (e.g., "Syntax validation successful.") then print **Symbol Table to each token produced Like in sheet 1 and the AST.**
   - If invalid, display detailed error messages **specifying the line and column of syntax errors**.

Project Milestones

1. **Phase 1: Lexical Analysis**
   - Define tokens in JavaCC.
   - Implement rules for keywords, identifiers, literals, operators, delimiters, and comments.
   - Validate tokenizer with sample input files.
2. **Phase 2: Parser Grammar**
   - Implement grammar rules for:
     - Variable declarations and assignments
     - Control structures (if, for, while)
     - Expressions
   - Test parsing functionality with valid and invalid Java source code.
3. **Phase 3: AST Construction**
   - Build AST nodes corresponding to grammar rules.
   - Implement visitor or print method to traverse and display AST.
   - Integrate AST printing with syntax validation output.

(BOUNS FEATURE)

Semantic Enhancements:

Simple Examples:

- Detect **unused variables** across the program.
- Type checking for expressions and variable assignments.
- Scope resolution for variables across functions or blocks.
- And more optimizations…