Report

CS 3512: Programming Languages

Programming Project 01

Prepared by Group 22

- **200481C Premathilaka G.G.G.S.C.**
- **200614N Somarathna W.A.N.M.**
- **200629N Sumathipala H.E.S.**
- **200695K Warshamana W.I.S.N.**

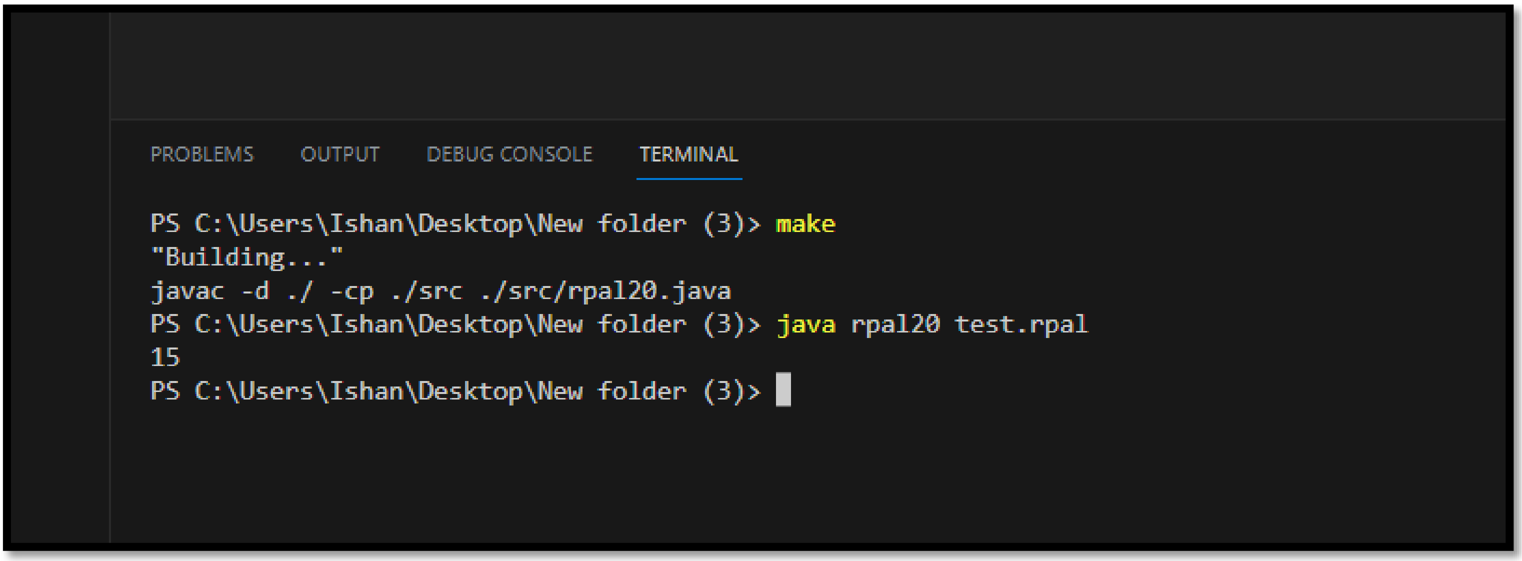
Date 2023/07/25

Table of content

Problem

You are required to implement a lexical analyzer and a parser for the RPAL language. Output of the parser should be the Abstract Syntax Tree (AST) for the given input program. Then you need to implement an algorithm to convert the Abstract Syntax Tree (AST) in to Standardize Tree (ST) and implement CSE machine. Your program should be able to read an input file which contains a RPAL program. Output of your program should match the output of “rpal.exe“ for the relevant program.

Run Programme

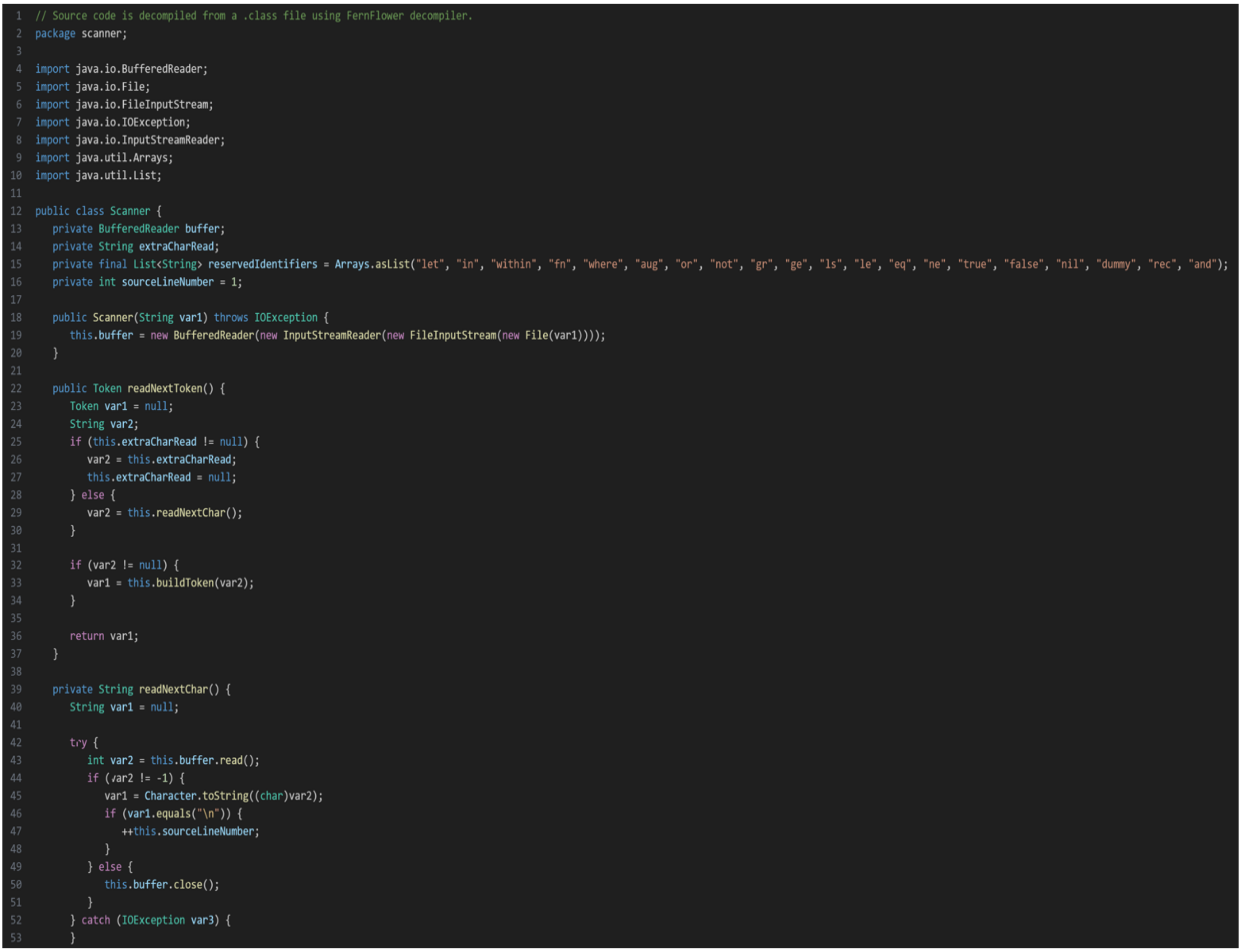


```
PS C:\Users\Ishan\Desktop\New folder (3)> make
"Building..."
javac -d ./ -cp ./src ./src/rpal20.java
PS C:\Users\Ishan\Desktop\New folder (3)> java rpal20 test.rpal
15
PS C:\Users\Ishan\Desktop\New folder (3)>
```

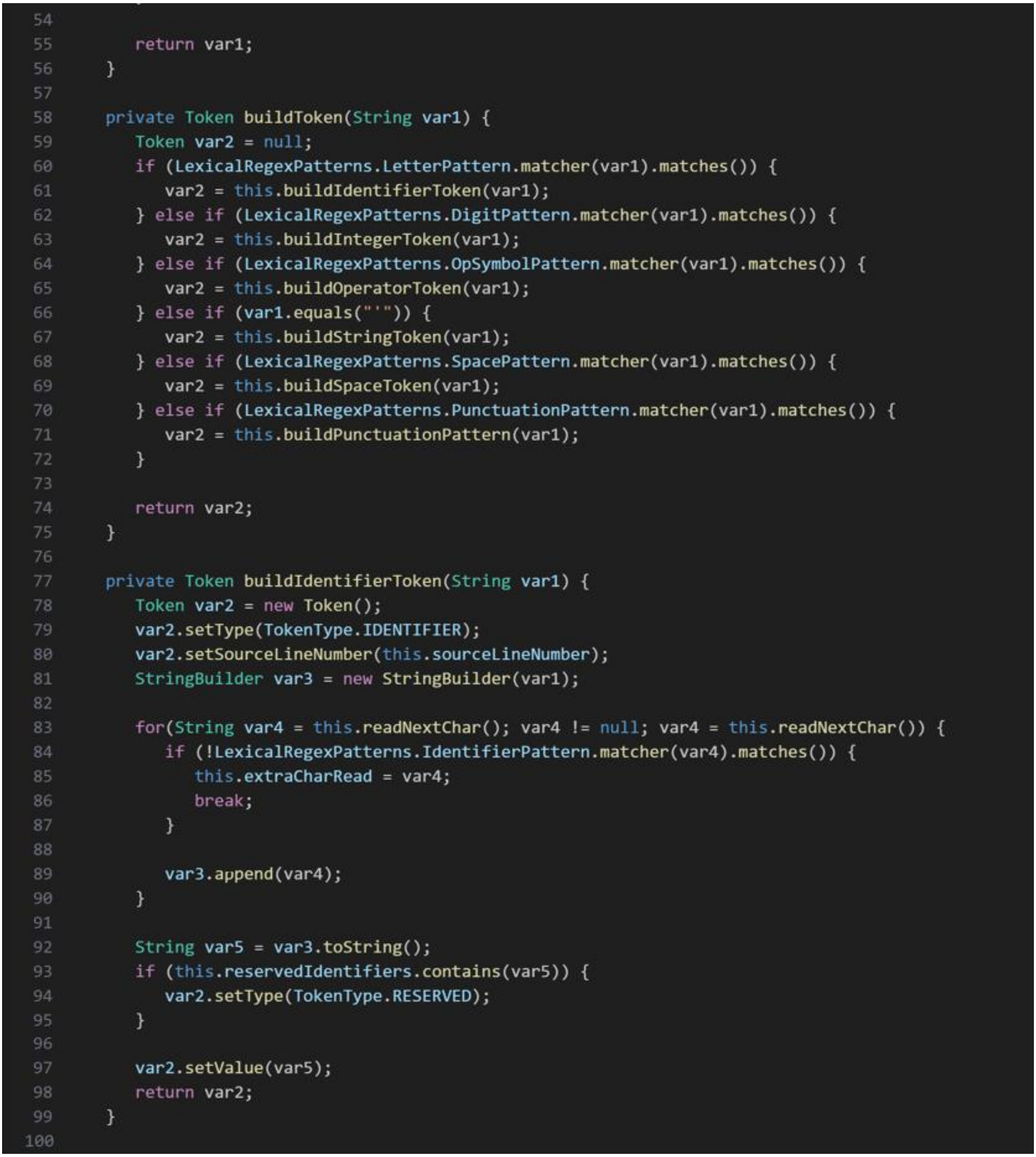
Solution

Lexical Analysis

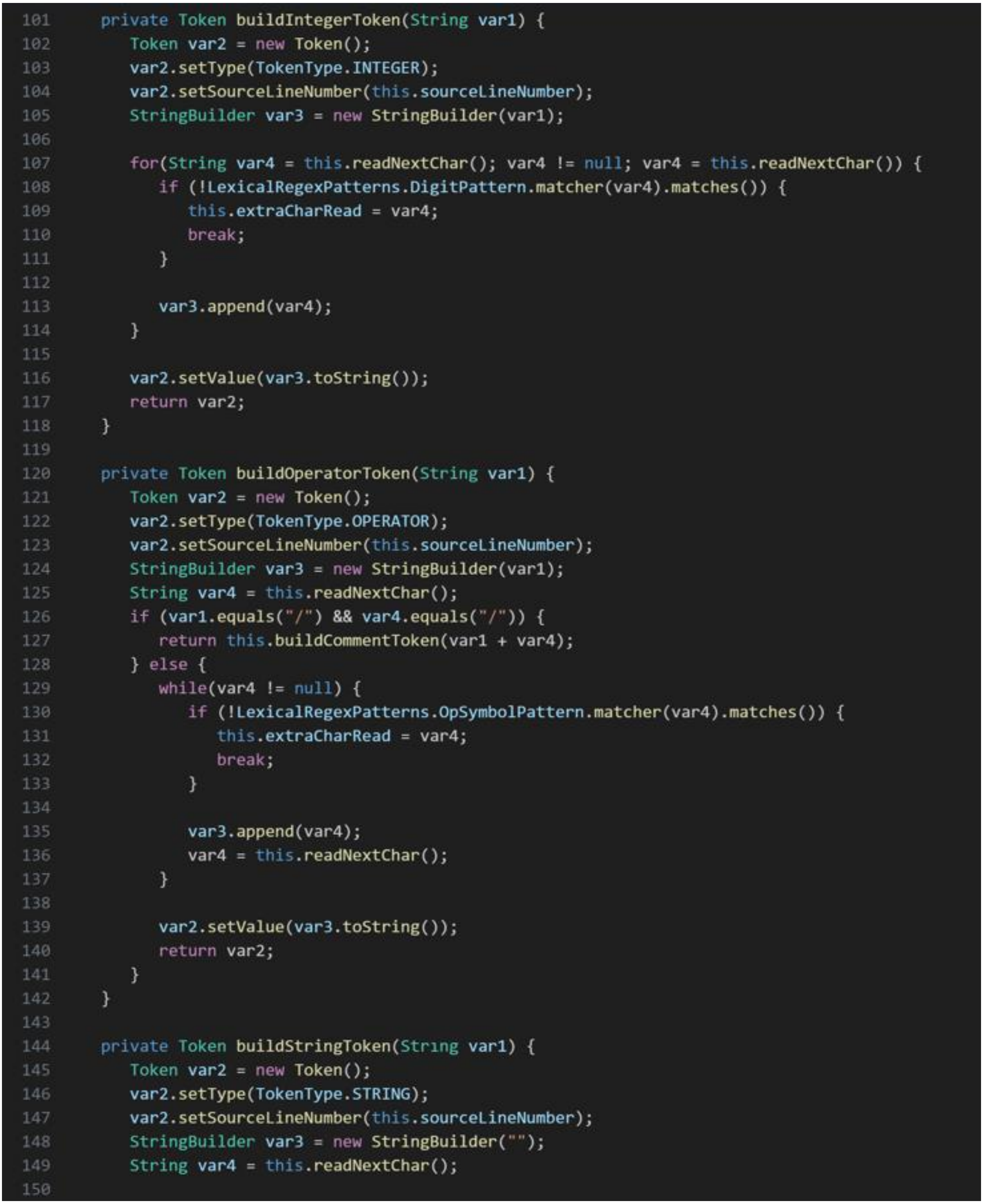
- Implement a lexical analyzer (also known as a lexer or scanner) that reads the input RPAL program and converts it into a stream of tokens. Tokens are the smallest meaningful units in a programming language (e.g., keywords, identifiers, operators, etc.).
- Define rules to identify different types of tokens based on the RPAL language specifications.



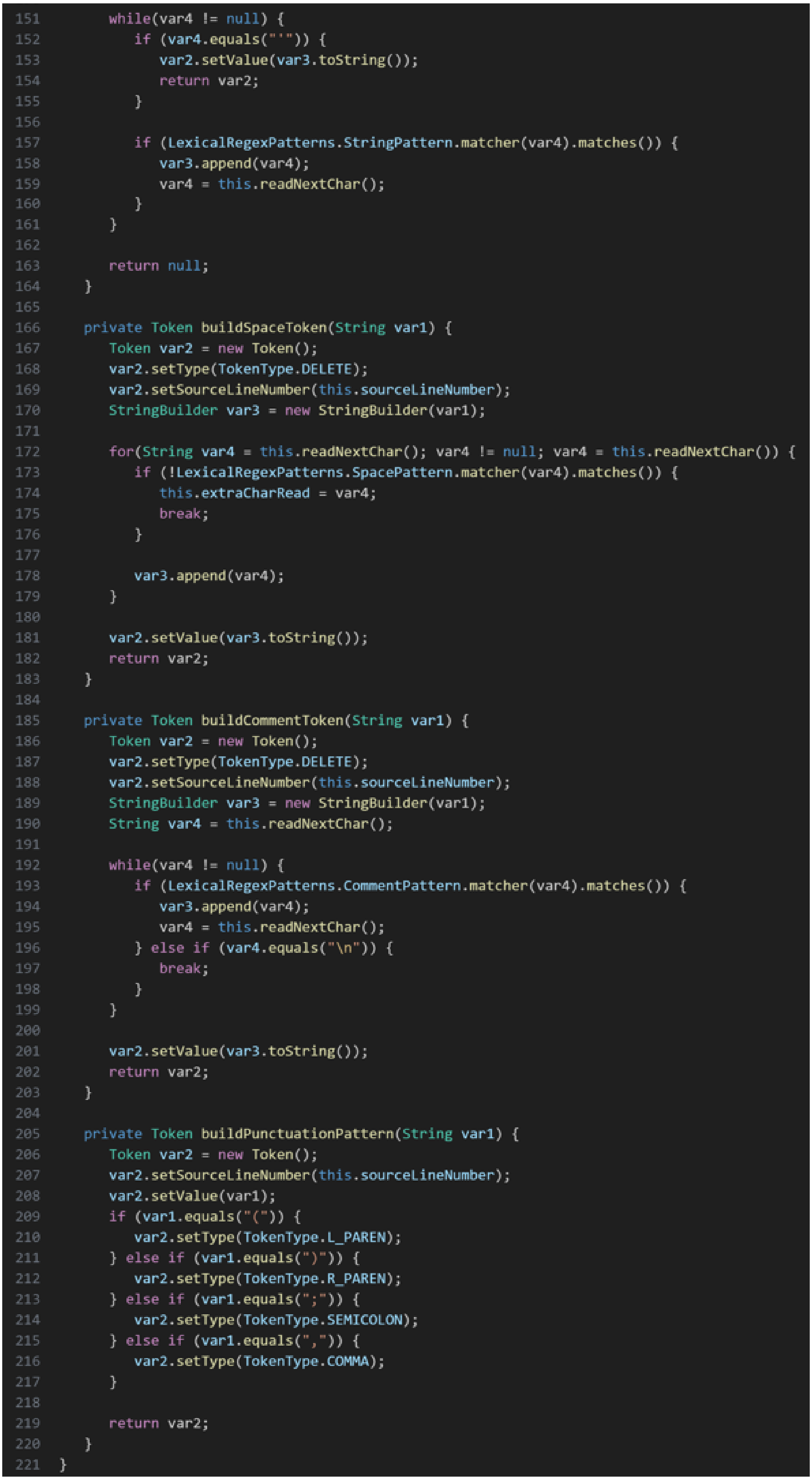
```java
// Source code is decompiled from a .class file using FernFlower decompiler.
package scanner;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;
import java.util.List;

public class Scanner {
   private BufferedReader buffer;
   private String extraCharRead;
   private final List<String> reservedIdentifiers = Arrays.asList("let", "in", "within", "fn", "where", "aug", "or", "not", "gr", "ge", "ls", "le", "eq", "ne", "true", "false", "nil", "dummy", "rec", "and");
   private int sourceLineNumber = 1;

   public Scanner(String var1) throws IOException {
      this.buffer = new BufferedReader(new InputStreamReader(new FileInputStream(new File(var1))));
   }

   public Token readNextToken() {
      Token var1 = null;
      String var2;
      if (this.extraCharRead != null) {
         var2 = this.extraCharRead;
         this.extraCharRead = null;
      } else {
         var2 = this.readNextChar();
      }

      if (var2 != null) {
         var1 = this.buildToken(var2);
      }

      return var1;
   }

   private String readNextChar() {
      String var1 = null;

      try {
         int var2 = this.buffer.read();
         if (var2 != -1) {
            var1 = Character.toString((char)var2);
            if (var1.equals("\n")) {
               ++this.sourceLineNumber;
            }
         } else {
            this.buffer.close();
         }
      } catch (IOException var3) {
      }
```



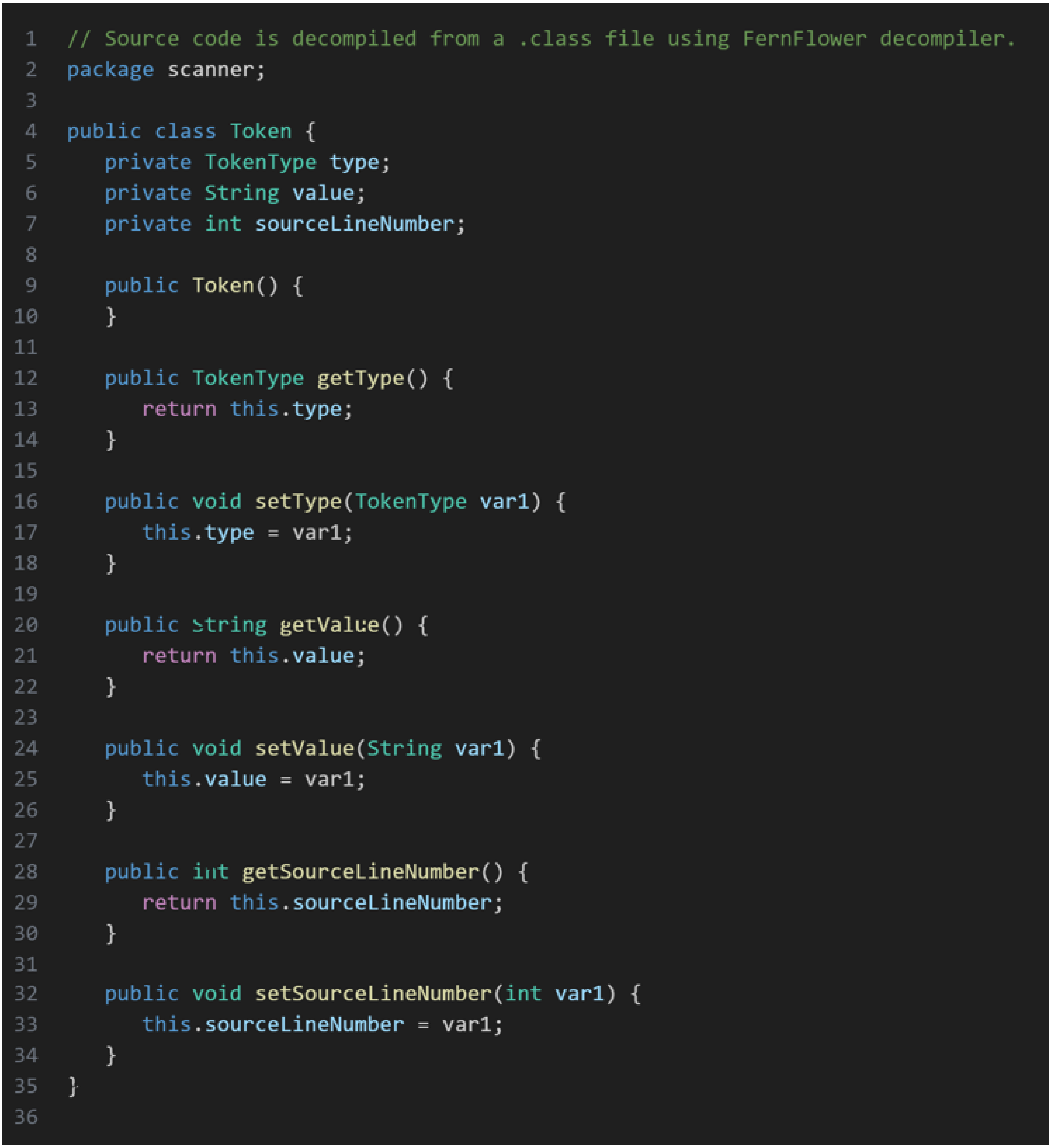
```
      return var1;
   }

   private Token buildToken(String var1) {
      Token var2 = null;
      if (LexicalRegexPatterns.LetterPattern.matcher(var1).matches()) {
         var2 = this.buildIdentifierToken(var1);
      } else if (LexicalRegexPatterns.DigitPattern.matcher(var1).matches()) {
         var2 = this.buildIntegerToken(var1);
      } else if (LexicalRegexPatterns.OpSymbolPattern.matcher(var1).matches()) {
         var2 = this.buildOperatorToken(var1);
      } else if (var1.equals("'")) {
         var2 = this.buildStringToken(var1);
      } else if (LexicalRegexPatterns.SpacePattern.matcher(var1).matches()) {
         var2 = this.buildSpaceToken(var1);
      } else if (LexicalRegexPatterns.PunctuationPattern.matcher(var1).matches()) {
         var2 = this.buildPunctuationPattern(var1);
      }

      return var2;
   }

   private Token buildIdentifierToken(String var1) {
      Token var2 = new Token();
      var2.setType(TokenType.IDENTIFIER);
      var2.setSourceLineNumber(this.sourceLineNumber);
      StringBuilder var3 = new StringBuilder(var1);

      for(String var4 = this.readNextChar(); var4 != null; var4 = this.readNextChar()) {
         if (!LexicalRegexPatterns.IdentifierPattern.matcher(var4).matches()) {
            this.extraCharRead = var4;
            break;
         }

         var3.append(var4);
      }

      String var5 = var3.toString();
      if (this.reservedIdentifiers.contains(var5)) {
         var2.setType(TokenType.RESERVED);
      }

      var2.setValue(var5);
      return var2;
   }

```



```java
    private Token buildIntegerToken(String var1) {
        Token var2 = new Token();
        var2.setType(TokenType.INTEGER);
        var2.setSourceLineNumber(this.sourceLineNumber);
        StringBuilder var3 = new StringBuilder(var1);

        for(String var4 = this.readNextChar(); var4 != null; var4 = this.readNextChar()) {
            if (!LexicalRegexPatterns.DigitPattern.matcher(var4).matches()) {
                this.extraCharRead = var4;
                break;
            }

            var3.append(var4);
        }

        var2.setValue(var3.toString());
        return var2;
    }

    private Token buildOperatorToken(String var1) {
        Token var2 = new Token();
        var2.setType(TokenType.OPERATOR);
        var2.setSourceLineNumber(this.sourceLineNumber);
        StringBuilder var3 = new StringBuilder(var1);
        String var4 = this.readNextChar();
        if (var1.equals("/") && var4.equals("/")) {
            return this.buildCommentToken(var1 + var4);
        } else {
            while(var4 != null) {
                if (!LexicalRegexPatterns.OpSymbolPattern.matcher(var4).matches()) {
                    this.extraCharRead = var4;
                    break;
                }

                var3.append(var4);
                var4 = this.readNextChar();
            }

            var2.setValue(var3.toString());
            return var2;
        }
    }

    private Token buildStringToken(String var1) {
        Token var2 = new Token();
        var2.setType(TokenType.STRING);
        var2.setSourceLineNumber(this.sourceLineNumber);
        StringBuilder var3 = new StringBuilder("");
        String var4 = this.readNextChar();

```



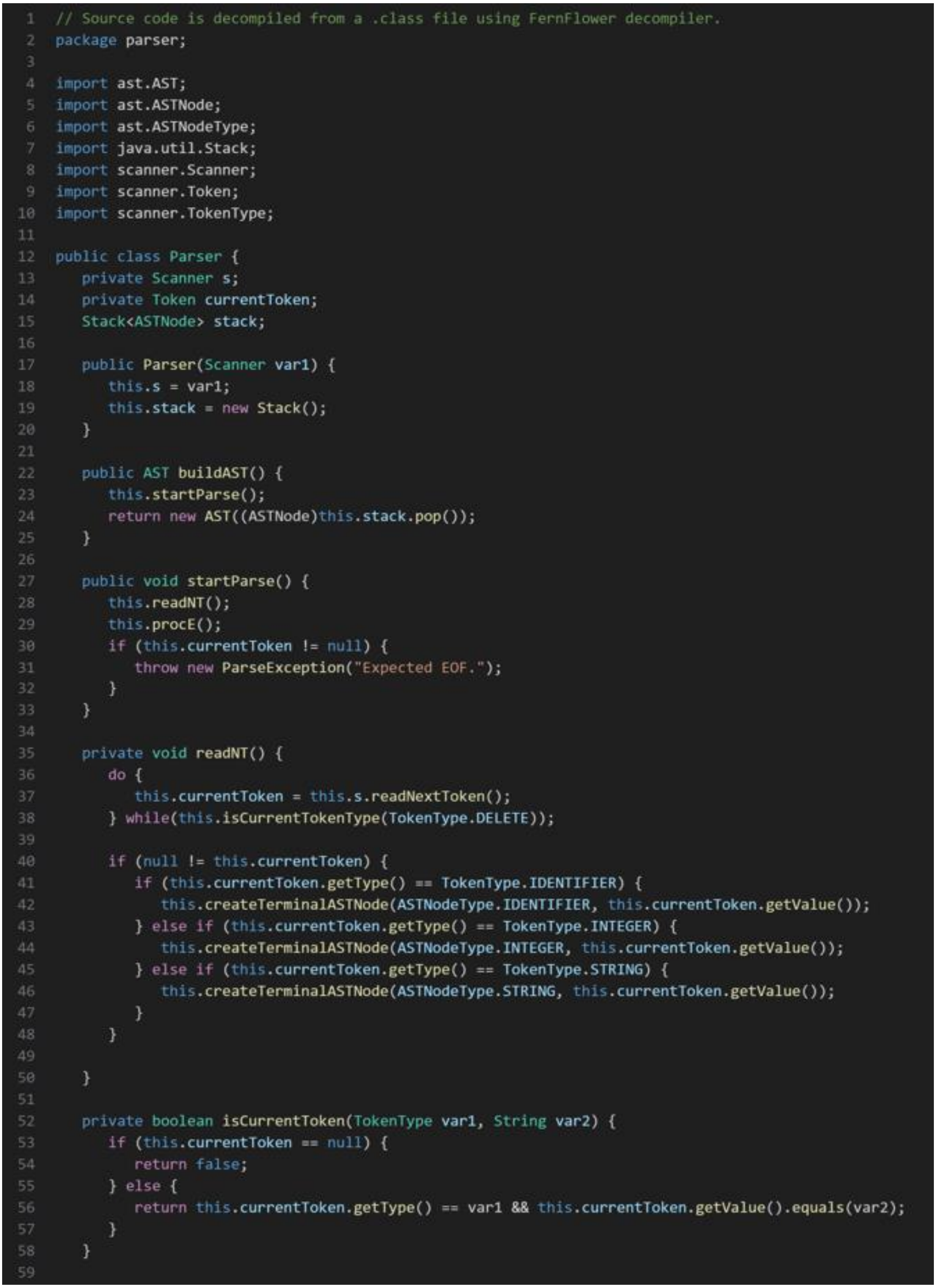
```java
      while(var4 != null) {
         if (var4.equals("'")) {
            var2.setValue(var3.toString());
            return var2;
         }

         if (LexicalRegexPatterns.StringPattern.matcher(var4).matches()) {
            var3.append(var4);
            var4 = this.readNextChar();
         }
      }

      return null;
   }

   private Token buildSpaceToken(String var1) {
      Token var2 = new Token();
      var2.setType(TokenType.DELETE);
      var2.setSourceLineNumber(this.sourceLineNumber);
      StringBuilder var3 = new StringBuilder(var1);

      for(String var4 = this.readNextChar(); var4 != null; var4 = this.readNextChar()) {
         if (!LexicalRegexPatterns.SpacePattern.matcher(var4).matches()) {
            this.extraCharRead = var4;
            break;
         }

         var3.append(var4);
      }

      var2.setValue(var3.toString());
      return var2;
   }

   private Token buildCommentToken(String var1) {
      Token var2 = new Token();
      var2.setType(TokenType.DELETE);
      var2.setSourceLineNumber(this.sourceLineNumber);
      StringBuilder var3 = new StringBuilder(var1);
      String var4 = this.readNextChar();

      while(var4 != null) {
         if (LexicalRegexPatterns.CommentPattern.matcher(var4).matches()) {
            var3.append(var4);
            var4 = this.readNextChar();
         } else if (var4.equals("\n")) {
            break;
         }
      }

      var2.setValue(var3.toString());
      return var2;
   }

   private Token buildPunctuationPattern(String var1) {
      Token var2 = new Token();
      var2.setSourceLineNumber(this.sourceLineNumber);
      var2.setValue(var1);
      if (var1.equals("(")) {
         var2.setType(TokenType.L_PAREN);
      } else if (var1.equals(")")) {
         var2.setType(TokenType.R_PAREN);
      } else if (var1.equals(";")) {
         var2.setType(TokenType.SEMICOLON);
      } else if (var1.equals(",")) {
         var2.setType(TokenType.COMMA);
      }

      return var2;
   }
}
```



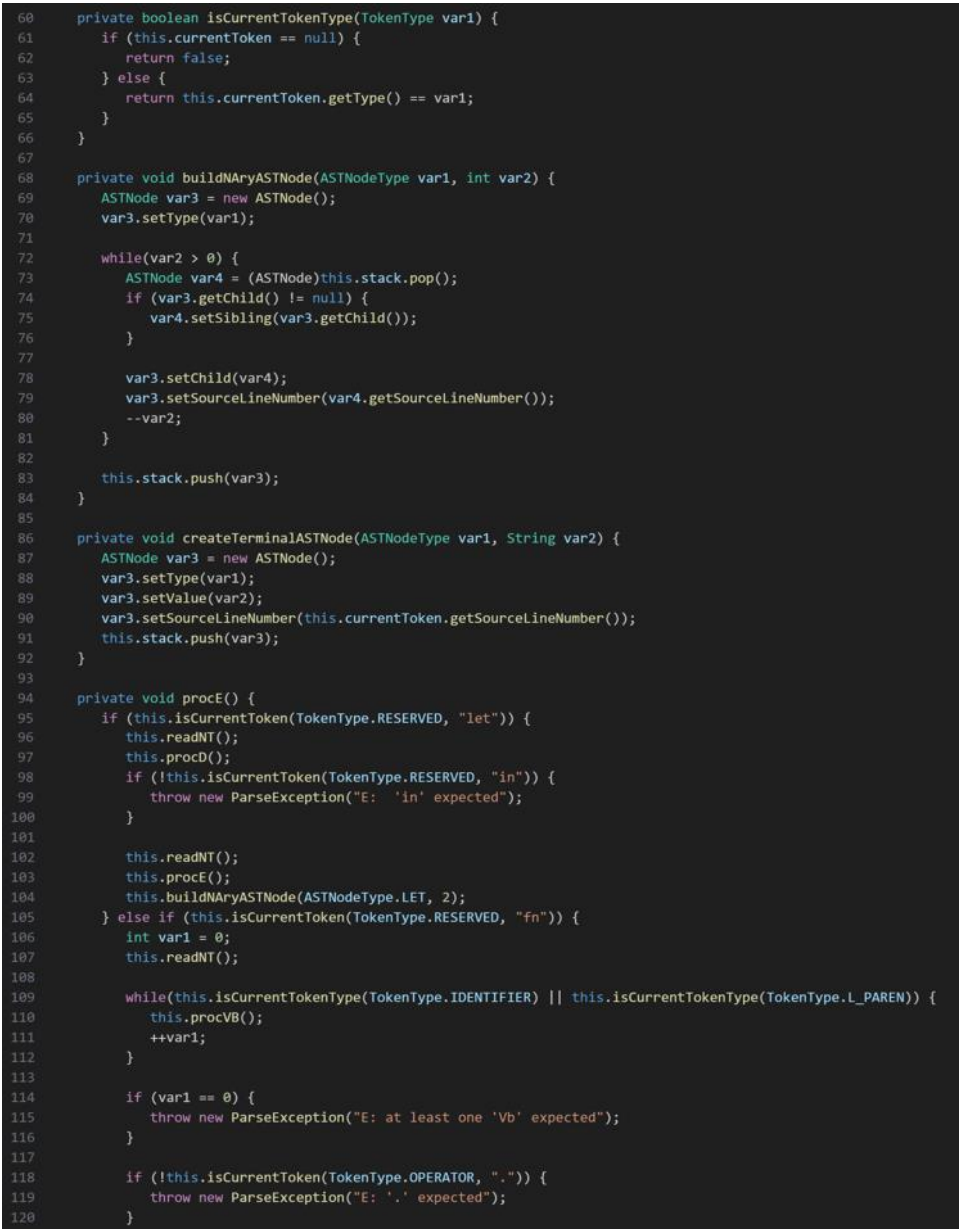
```java
// Source code is decompiled from a .class file using FernFlower decompiler.
package scanner;

public class Token {
   private TokenType type;
   private String value;
   private int sourceLineNumber;

   public Token() {
   }

   public TokenType getType() {
      return this.type;
   }

   public void setType(TokenType var1) {
      this.type = var1;
   }

   public String getValue() {
      return this.value;
   }

   public void setValue(String var1) {
      this.value = var1;
   }

   public int getSourceLineNumber() {
      return this.sourceLineNumber;
   }

   public void setSourceLineNumber(int var1) {
      this.sourceLineNumber = var1;
   }
}

```

Syntax Analysis (Parsing)

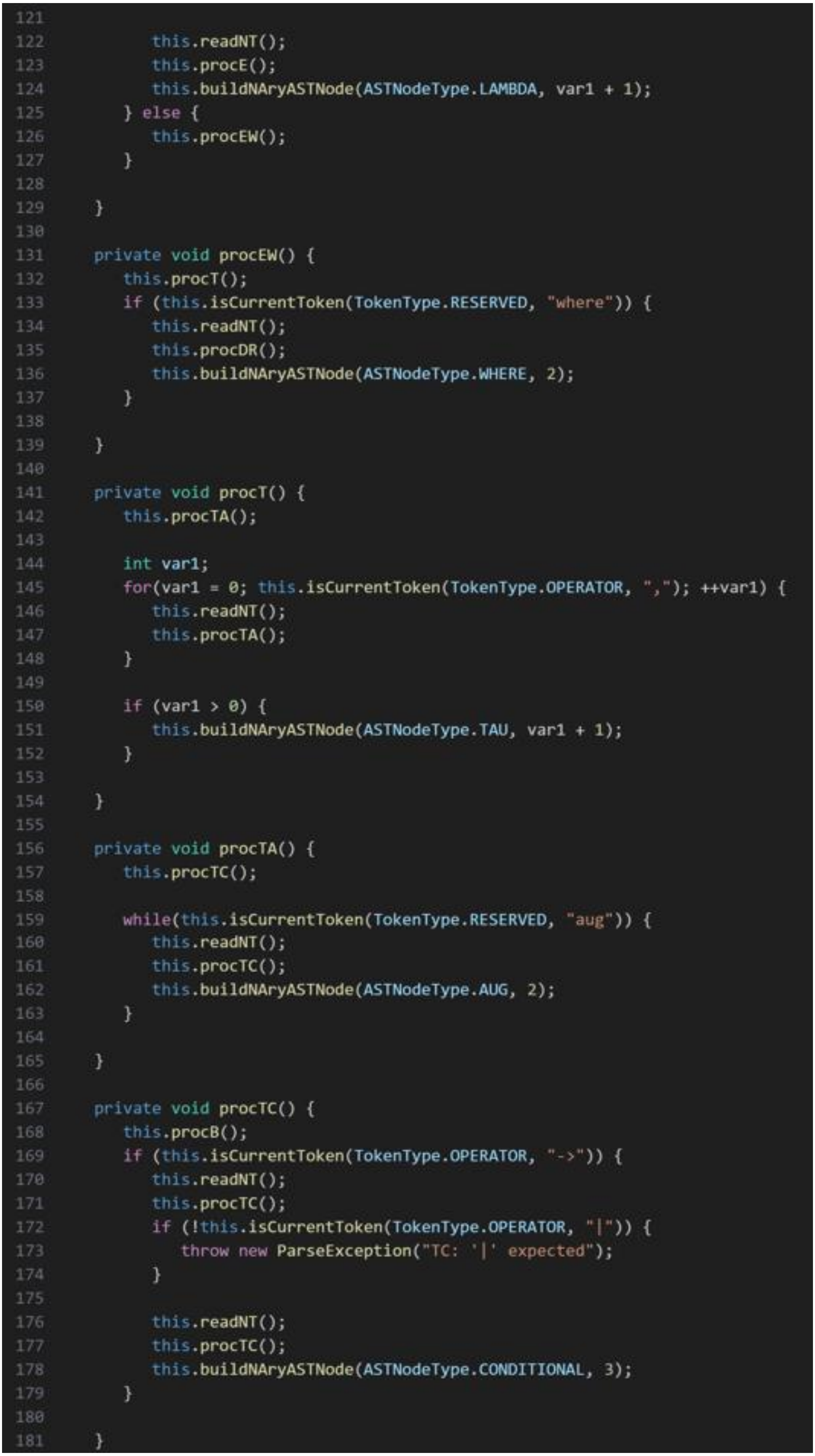
- Implement a parser that takes the token stream from the lexer and constructs the Abstract Syntax Tree (AST).
- The parser should follow the grammar rules of the RPAL language to validate the syntax and build the corresponding AST nodes.
- If there is no file name, an exception is thrown.
- If the file specified in the argument exists, first Read the text file mentioned above each by line, trimming any extra spaces, then create a list for each line.
- All of the following escape sequences can be found using a function.
    - BS
    - FF
    - NL
    - CR
    - TAB
    - double quotation
    - single quote
- To determine the depth of the root, we must count the dots at the beginning of each line.
- Use the CreateTree class to build a tree in accordance with the lines that were read.



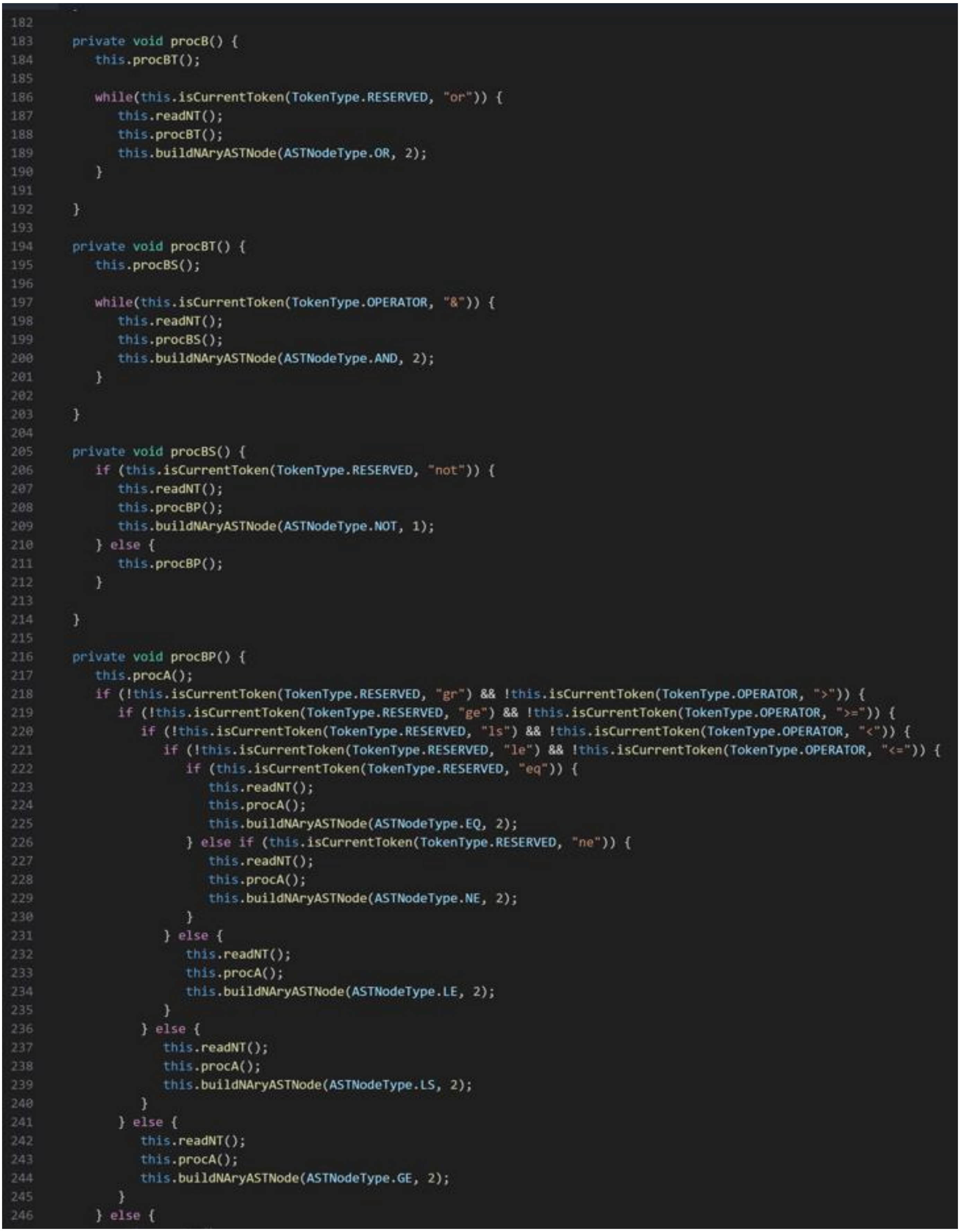
```
// Source code is decompiled from a .class file using FernFlower decompiler.
package parser;

import ast.AST;
import ast.ASTNode;
import ast.ASTNodeType;
import java.util.Stack;
import scanner.Scanner;
import scanner.Token;
import scanner.TokenType;

public class Parser {
   private Scanner s;
   private Token currentToken;
   Stack<ASTNode> stack;

   public Parser(Scanner var1) {
      this.s = var1;
      this.stack = new Stack();
   }

   public AST buildAST() {
      this.startParse();
      return new AST((ASTNode)this.stack.pop());
   }

   public void startParse() {
      this.readNT();
      this.procE();
      if (this.currentToken != null) {
         throw new ParseException("Expected EOF.");
      }
   }

   private void readNT() {
      do {
         this.currentToken = this.s.readNextToken();
      } while(this.isCurrentTokenType(TokenType.DELETE));

      if (null != this.currentToken) {
         if (this.currentToken.getType() == TokenType.IDENTIFIER) {
            this.createTerminalASTNode(ASTNodeType.IDENTIFIER, this.currentToken.getValue());
         } else if (this.currentToken.getType() == TokenType.INTEGER) {
            this.createTerminalASTNode(ASTNodeType.INTEGER, this.currentToken.getValue());
         } else if (this.currentToken.getType() == TokenType.STRING) {
            this.createTerminalASTNode(ASTNodeType.STRING, this.currentToken.getValue());
         }
      }

   }

   private boolean isCurrentToken(TokenType var1, String var2) {
      if (this.currentToken == null) {
         return false;
      } else {
         return this.currentToken.getType() == var1 && this.currentToken.getValue().equals(var2);
      }
   }

```



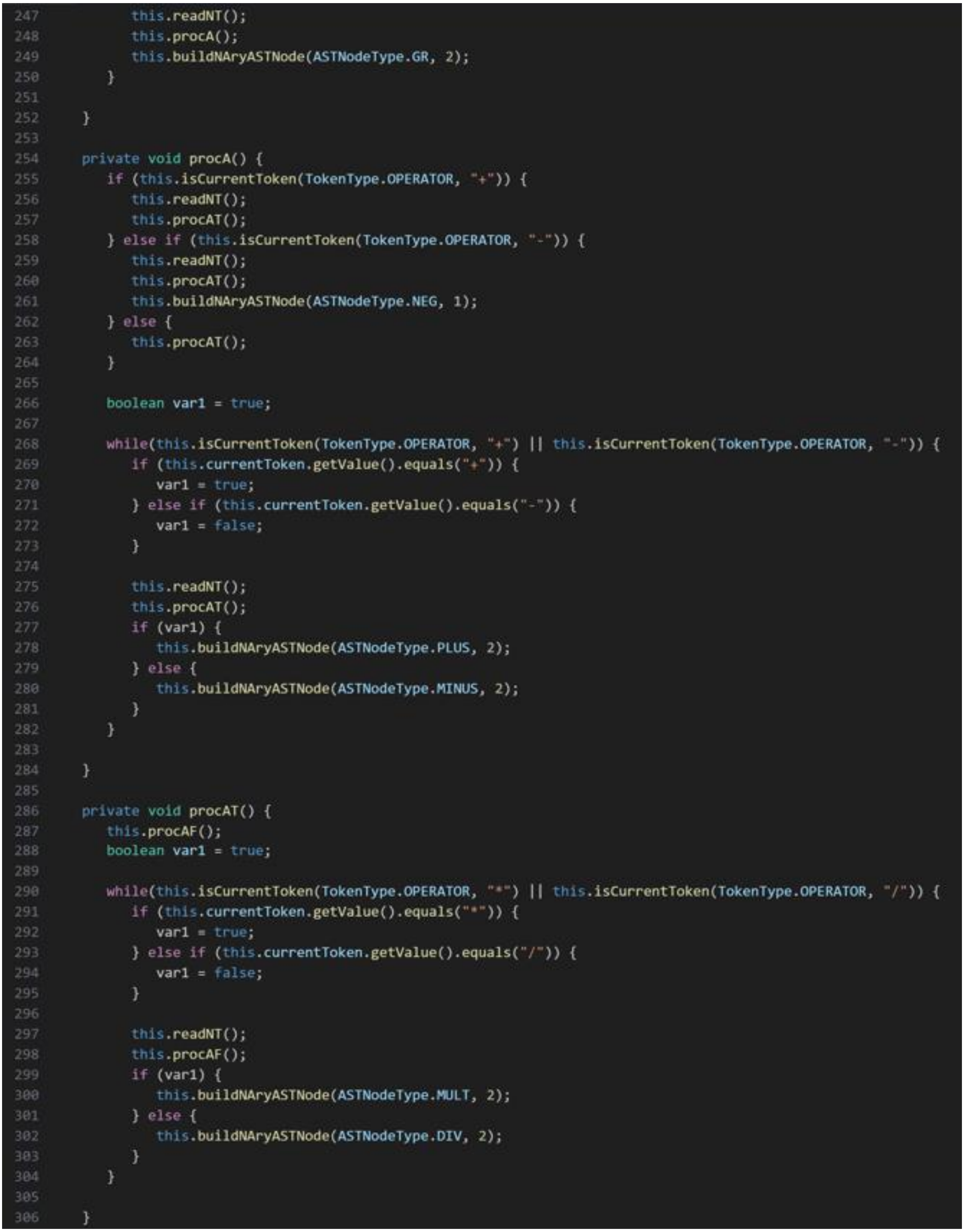
```
   private boolean isCurrentTokenType(TokenType var1) {
      if (this.currentToken == null) {
         return false;
      } else {
         return this.currentToken.getType() == var1;
      }
   }

   private void buildNAryASTNode(ASTNodeType var1, int var2) {
      ASTNode var3 = new ASTNode();
      var3.setType(var1);

      while(var2 > 0) {
         ASTNode var4 = (ASTNode)this.stack.pop();
         if (var3.getChild() != null) {
            var4.setSibling(var3.getChild());
         }

         var3.setChild(var4);
         var3.setSourceLineNumber(var4.getSourceLineNumber());
         --var2;
      }

      this.stack.push(var3);
   }

   private void createTerminalASTNode(ASTNodeType var1, String var2) {
      ASTNode var3 = new ASTNode();
      var3.setType(var1);
      var3.setValue(var2);
      var3.setSourceLineNumber(this.currentToken.getSourceLineNumber());
      this.stack.push(var3);
   }

   private void procE() {
      if (this.isCurrentToken(TokenType.RESERVED, "let")) {
         this.readNT();
         this.procD();
         if (!this.isCurrentToken(TokenType.RESERVED, "in")) {
            throw new ParseException("E:  'in' expected");
         }

         this.readNT();
         this.procE();
         this.buildNAryASTNode(ASTNodeType.LET, 2);
      } else if (this.isCurrentToken(TokenType.RESERVED, "fn")) {
         int var1 = 0;
         this.readNT();

         while(this.isCurrentTokenType(TokenType.IDENTIFIER) || this.isCurrentTokenType(TokenType.L_PAREN)) {
            this.procVB();
            ++var1;
         }

         if (var1 == 0) {
            throw new ParseException("E: at least one 'Vb' expected");
         }

         if (!this.isCurrentToken(TokenType.OPERATOR, ".")) {
            throw new ParseException("E: '.' expected");
         }
```



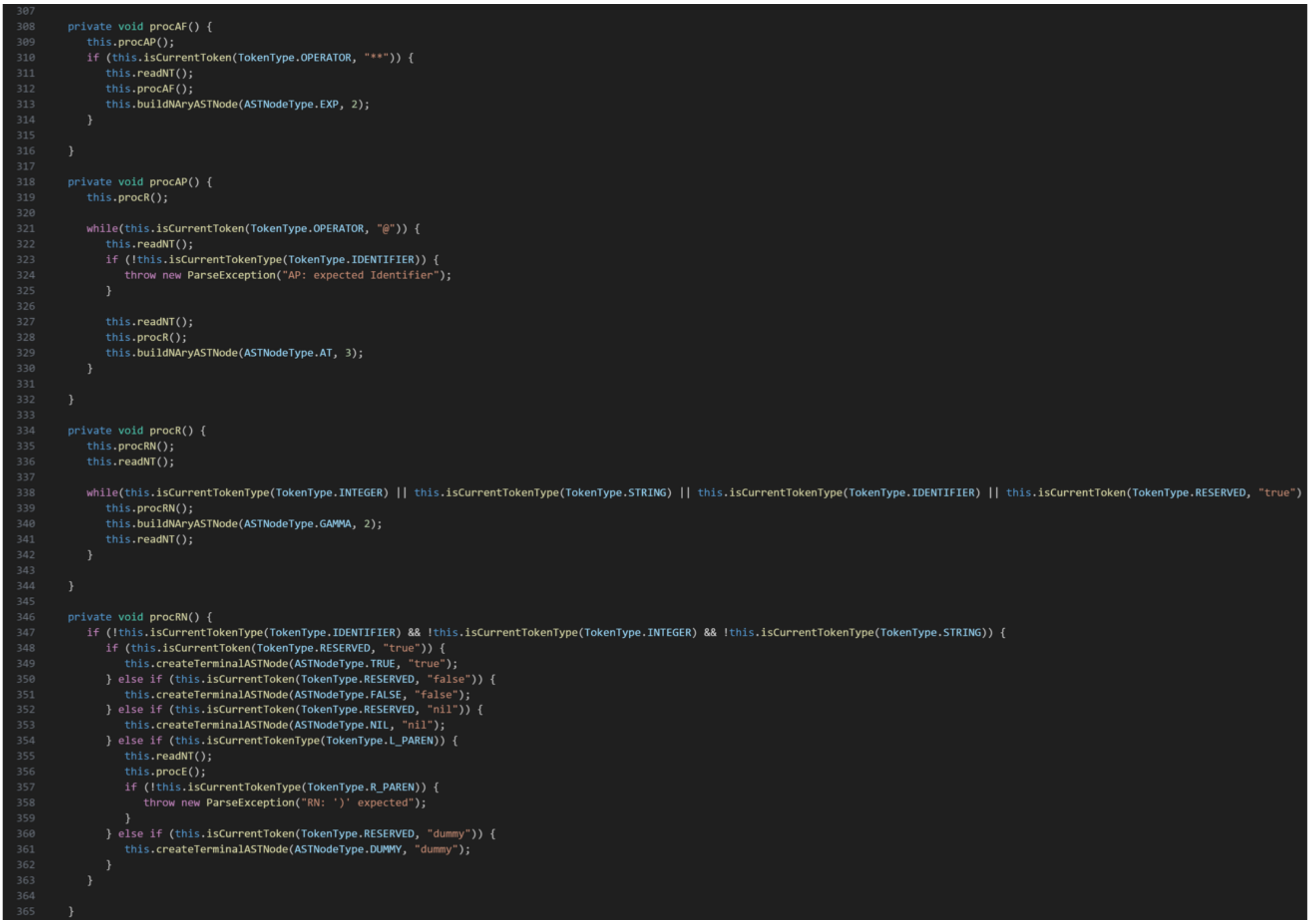
```java

            this.readNT();
            this.procE();
            this.buildNAryASTNode(ASTNodeType.LAMBDA, var1 + 1);
        } else {
            this.procEW();
        }

    }

    private void procEW() {
        this.procT();
        if (this.isCurrentToken(TokenType.RESERVED, "where")) {
            this.readNT();
            this.procDR();
            this.buildNAryASTNode(ASTNodeType.WHERE, 2);
        }

    }

    private void procT() {
        this.procTA();

        int var1;
        for(var1 = 0; this.isCurrentToken(TokenType.OPERATOR, ","); ++var1) {
            this.readNT();
            this.procTA();
        }

        if (var1 > 0) {
            this.buildNAryASTNode(ASTNodeType.TAU, var1 + 1);
        }

    }

    private void procTA() {
        this.procTC();

        while(this.isCurrentToken(TokenType.RESERVED, "aug")) {
            this.readNT();
            this.procTC();
            this.buildNAryASTNode(ASTNodeType.AUG, 2);
        }

    }

    private void procTC() {
        this.procB();
        if (this.isCurrentToken(TokenType.OPERATOR, "->")) {
            this.readNT();
            this.procTC();
            if (!this.isCurrentToken(TokenType.OPERATOR, "|")) {
                throw new ParseException("TC: '|' expected");
            }

            this.readNT();
            this.procTC();
            this.buildNAryASTNode(ASTNodeType.CONDITIONAL, 3);
        }

    }
```



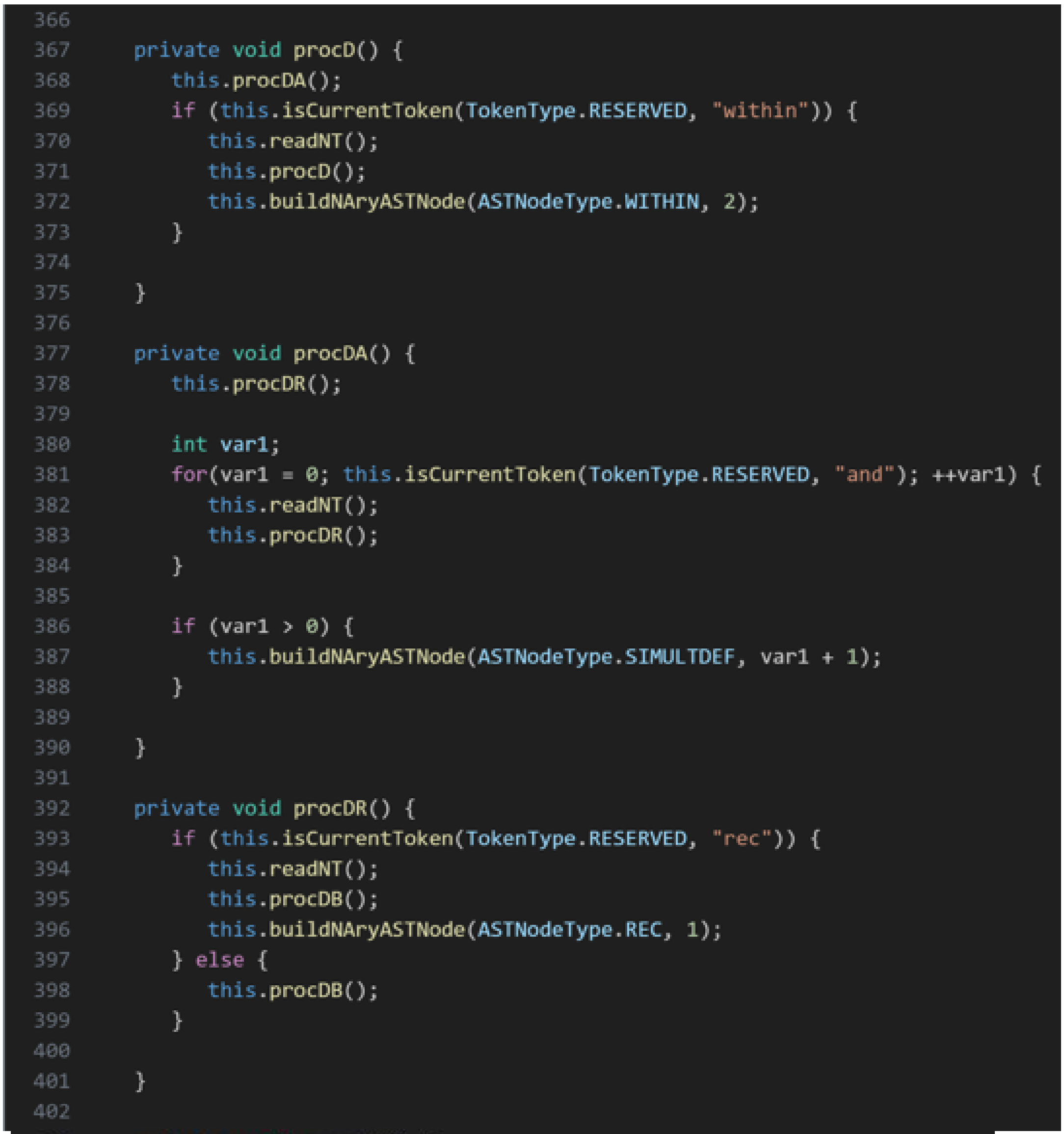
```
   private void procB() {
      this.procBT();

      while(this.isCurrentToken(TokenType.RESERVED, "or")) {
         this.readNT();
         this.procBT();
         this.buildNAryASTNode(ASTNodeType.OR, 2);
      }

   }

   private void procBT() {
      this.procBS();

      while(this.isCurrentToken(TokenType.OPERATOR, "&")) {
         this.readNT();
         this.procBS();
         this.buildNAryASTNode(ASTNodeType.AND, 2);
      }

   }

   private void procBS() {
      if (this.isCurrentToken(TokenType.RESERVED, "not")) {
         this.readNT();
         this.procBP();
         this.buildNAryASTNode(ASTNodeType.NOT, 1);
      } else {
         this.procBP();
      }

   }

   private void procBP() {
      this.procA();
      if (!this.isCurrentToken(TokenType.RESERVED, "gr") && !this.isCurrentToken(TokenType.OPERATOR, ">")) {
         if (!this.isCurrentToken(TokenType.RESERVED, "ge") && !this.isCurrentToken(TokenType.OPERATOR, ">=")) {
            if (!this.isCurrentToken(TokenType.RESERVED, "ls") && !this.isCurrentToken(TokenType.OPERATOR, "<")) {
               if (!this.isCurrentToken(TokenType.RESERVED, "le") && !this.isCurrentToken(TokenType.OPERATOR, "<=")) {
                  if (this.isCurrentToken(TokenType.RESERVED, "eq")) {
                     this.readNT();
                     this.procA();
                     this.buildNAryASTNode(ASTNodeType.EQ, 2);
                  } else if (this.isCurrentToken(TokenType.RESERVED, "ne")) {
                     this.readNT();
                     this.procA();
                     this.buildNAryASTNode(ASTNodeType.NE, 2);
                  }
               } else {
                  this.readNT();
                  this.procA();
                  this.buildNAryASTNode(ASTNodeType.LE, 2);
               }
            } else {
               this.readNT();
               this.procA();
               this.buildNAryASTNode(ASTNodeType.LS, 2);
            }
         } else {
            this.readNT();
            this.procA();
            this.buildNAryASTNode(ASTNodeType.GE, 2);
         }
      } else {
```



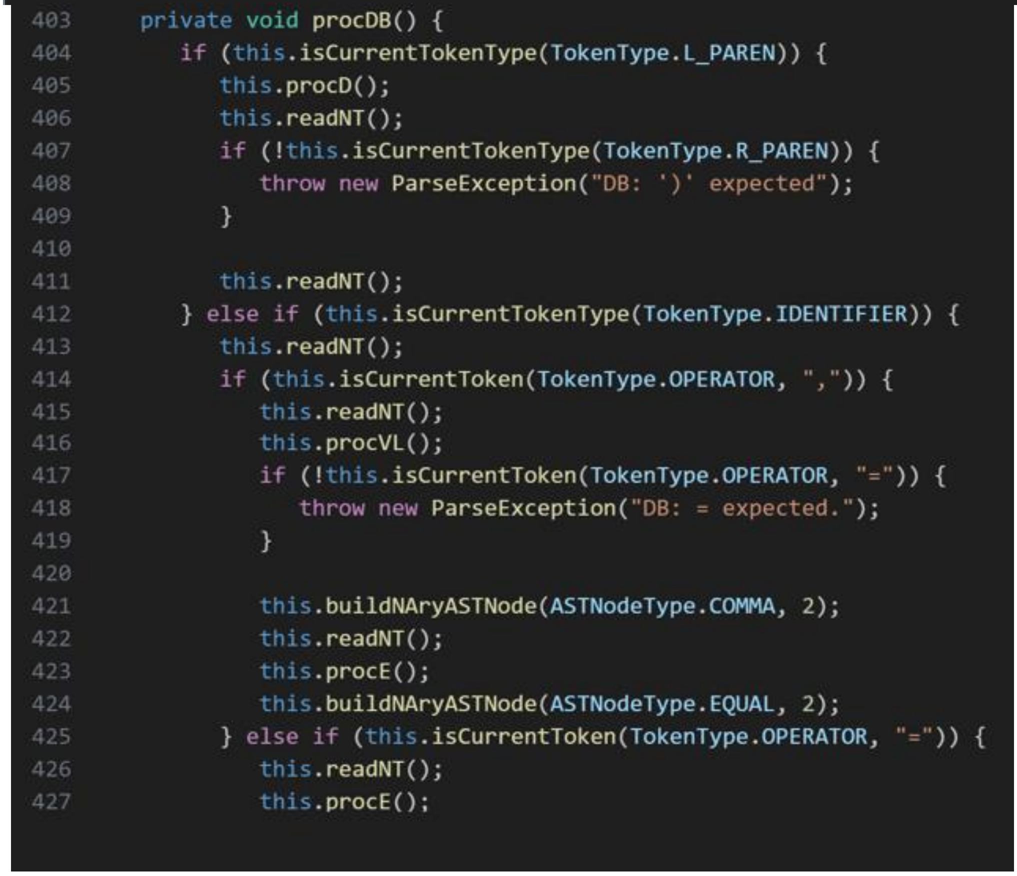
```
         this.readNT();
         this.procA();
         this.buildNAryASTNode(ASTNodeType.GR, 2);
      }

   }

   private void procA() {
      if (this.isCurrentToken(TokenType.OPERATOR, "+")) {
         this.readNT();
         this.procAT();
      } else if (this.isCurrentToken(TokenType.OPERATOR, "-")) {
         this.readNT();
         this.procAT();
         this.buildNAryASTNode(ASTNodeType.NEG, 1);
      } else {
         this.procAT();
      }

      boolean var1 = true;

      while(this.isCurrentToken(TokenType.OPERATOR, "+") || this.isCurrentToken(TokenType.OPERATOR, "-")) {
         if (this.currentToken.getValue().equals("+")) {
            var1 = true;
         } else if (this.currentToken.getValue().equals("-")) {
            var1 = false;
         }

         this.readNT();
         this.procAT();
         if (var1) {
            this.buildNAryASTNode(ASTNodeType.PLUS, 2);
         } else {
            this.buildNAryASTNode(ASTNodeType.MINUS, 2);
         }
      }

   }

   private void procAT() {
      this.procAF();
      boolean var1 = true;

      while(this.isCurrentToken(TokenType.OPERATOR, "*") || this.isCurrentToken(TokenType.OPERATOR, "/")) {
         if (this.currentToken.getValue().equals("*")) {
            var1 = true;
         } else if (this.currentToken.getValue().equals("/")) {
            var1 = false;
         }

         this.readNT();
         this.procAF();
         if (var1) {
            this.buildNAryASTNode(ASTNodeType.MULT, 2);
         } else {
            this.buildNAryASTNode(ASTNodeType.DIV, 2);
         }
      }

   }
```



```java
private void procAF() {
    this.procAP();
    if (this.isCurrentToken(TokenType.OPERATOR, "**")) {
        this.readNT();
        this.procAF();
        this.buildNAryASTNode(ASTNodeType.EXP, 2);
    }

}

private void procAP() {
    this.procR();

    while(this.isCurrentToken(TokenType.OPERATOR, "@")) {
        this.readNT();
        if (!this.isCurrentTokenType(TokenType.IDENTIFIER)) {
            throw new ParseException("AP: expected Identifier");
        }

        this.readNT();
        this.procR();
        this.buildNAryASTNode(ASTNodeType.AT, 3);
    }

}

private void procR() {
    this.procRN();
    this.readNT();

    while(this.isCurrentTokenType(TokenType.INTEGER) || this.isCurrentTokenType(TokenType.STRING) || this.isCurrentTokenType(TokenType.IDENTIFIER) || this.isCurrentToken(TokenType.RESERVED, "true")
        this.procRN();
        this.buildNAryASTNode(ASTNodeType.GAMMA, 2);
        this.readNT();
    }

}

private void procRN() {
    if (!this.isCurrentTokenType(TokenType.IDENTIFIER) && !this.isCurrentTokenType(TokenType.INTEGER) && !this.isCurrentTokenType(TokenType.STRING)) {
        if (this.isCurrentToken(TokenType.RESERVED, "true")) {
            this.createTerminalASTNode(ASTNodeType.TRUE, "true");
        } else if (this.isCurrentToken(TokenType.RESERVED, "false")) {
            this.createTerminalASTNode(ASTNodeType.FALSE, "false");
        } else if (this.isCurrentToken(TokenType.RESERVED, "nil")) {
            this.createTerminalASTNode(ASTNodeType.NIL, "nil");
        } else if (this.isCurrentTokenType(TokenType.L_PAREN)) {
            this.readNT();
            this.procE();
            if (!this.isCurrentTokenType(TokenType.R_PAREN)) {
                throw new ParseException("RN: ')' expected");
            }
        } else if (this.isCurrentToken(TokenType.RESERVED, "dummy")) {
            this.createTerminalASTNode(ASTNodeType.DUMMY, "dummy");
        }
    }

}
```



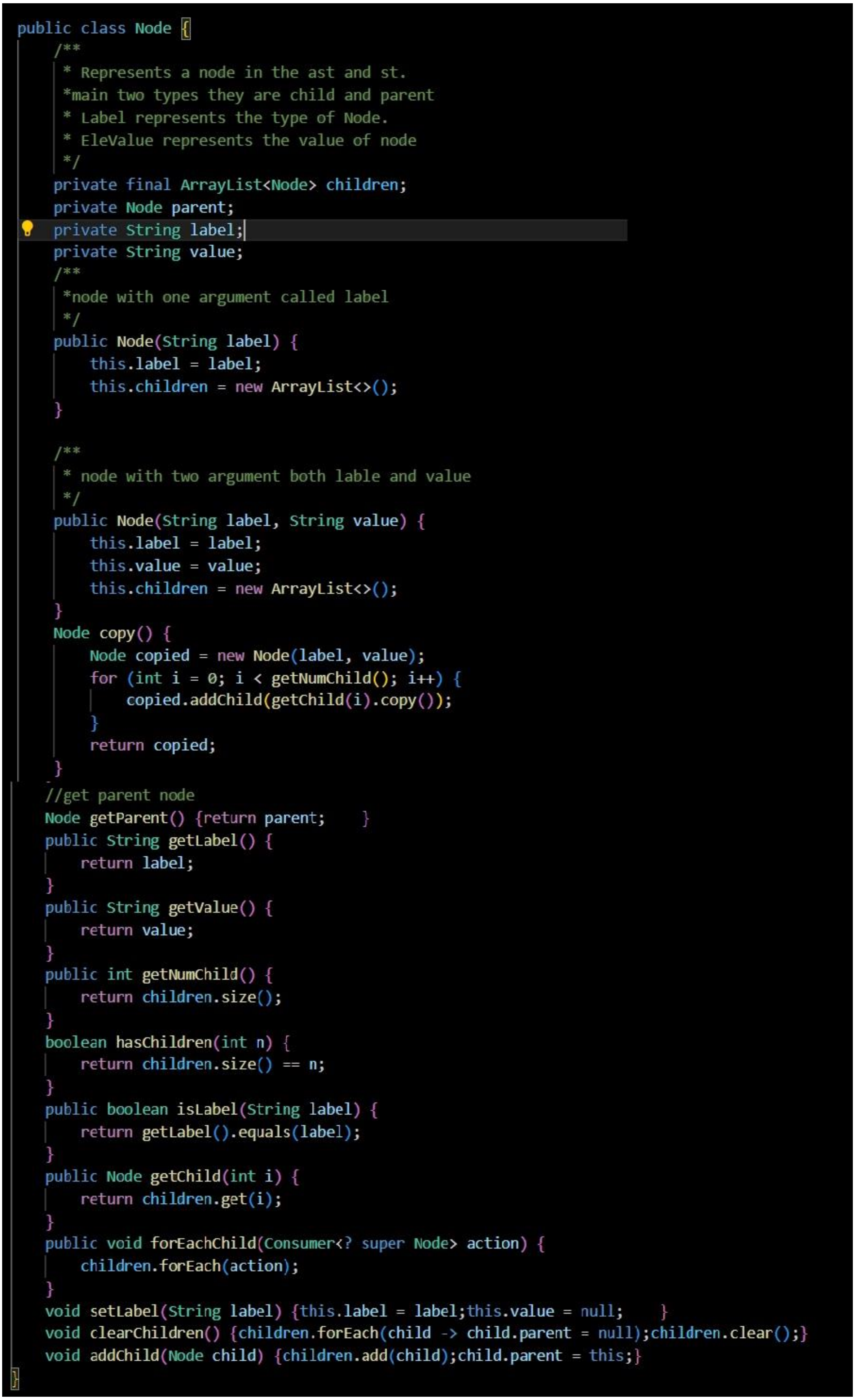
```java

   private void procD() {
      this.procDA();
      if (this.isCurrentToken(TokenType.RESERVED, "within")) {
         this.readNT();
         this.procD();
         this.buildNAryASTNode(ASTNodeType.WITHIN, 2);
      }

   }

   private void procDA() {
      this.procDR();

      int var1;
      for(var1 = 0; this.isCurrentToken(TokenType.RESERVED, "and"); ++var1) {
         this.readNT();
         this.procDR();
      }

      if (var1 > 0) {
         this.buildNAryASTNode(ASTNodeType.SIMULTDEF, var1 + 1);
      }

   }

   private void procDR() {
      if (this.isCurrentToken(TokenType.RESERVED, "rec")) {
         this.readNT();
         this.procDB();
         this.buildNAryASTNode(ASTNodeType.REC, 1);
      } else {
         this.procDB();
      }

   }

```



```java
   private void procDB() {
      if (this.isCurrentTokenType(TokenType.L_PAREN)) {
         this.procD();
         this.readNT();
         if (!this.isCurrentTokenType(TokenType.R_PAREN)) {
            throw new ParseException("DB: ')' expected");
         }

         this.readNT();
      } else if (this.isCurrentTokenType(TokenType.IDENTIFIER)) {
         this.readNT();
         if (this.isCurrentToken(TokenType.OPERATOR, ",")) {
            this.readNT();
            this.procVL();
            if (!this.isCurrentToken(TokenType.OPERATOR, "=")) {
               throw new ParseException("DB: = expected.");
            }

            this.buildNAryASTNode(ASTNodeType.COMMA, 2);
            this.readNT();
            this.procE();
            this.buildNAryASTNode(ASTNodeType.EQUAL, 2);
         } else if (this.isCurrentToken(TokenType.OPERATOR, "=")) {
            this.readNT();
            this.procE();
```

Create the ST to the provided AST using a node-based standardization tree

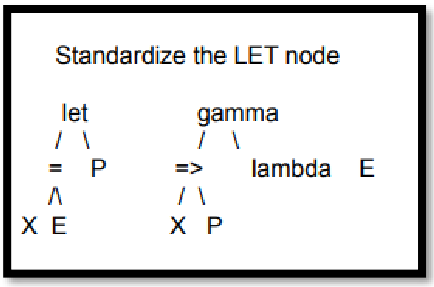
- Implementing the tree node is necessary for creating a ST or AST tree. Create a node class containing the parent, children, value, and label for that job.
- There are functions to clone a node, add a child, obtain a child, see if there are any children, clear the node, and add a child to it.



```
public class Node {
    /**
     * Represents a node in the ast and st.
     *main two types they are child and parent
     * Label represents the type of Node.
     * EleValue represents the value of node
     */
    private final ArrayList<Node> children;
    private Node parent;
    private String label;
    private String value;
    /**
     *node with one argument called label
     */
    public Node(String label) {
        this.label = label;
        this.children = new ArrayList<>();
    }

    /**
     * node with two argument both lable and value
     */
    public Node(String label, String value) {
        this.label = label;
        this.value = value;
        this.children = new ArrayList<>();
    }
    Node copy() {
        Node copied = new Node(label, value);
        for (int i = 0; i < getNumChild(); i++) {
            copied.addChild(getChild(i).copy());
        }
        return copied;
    }
    //get parent node
    Node getParent() {return parent;    }
    public String getLabel() {
        return label;
    }
    public String getValue() {
        return value;
    }
    public int getNumChild() {
        return children.size();
    }
    boolean hasChildren(int n) {
        return children.size() == n;
    }
    public boolean isLabel(String label) {
        return getLabel().equals(label);
    }
    public Node getChild(int i) {
        return children.get(i);
    }
    public void forEachChild(Consumer<? super Node> action) {
        children.forEach(action);
    }
    void setLabel(String label) {this.label = label;this.value = null;    }
    void clearChildren() {children.forEach(child -> child.parent = null);children.clear();}
    void addChild(Node child) {children.add(child);child.parent = this;}
}
```

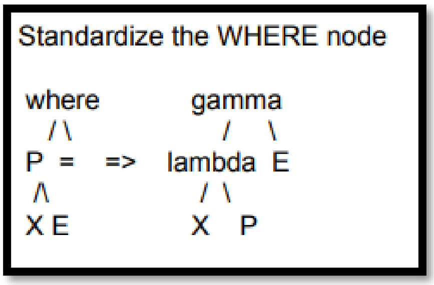
Using the AST to ST class, the raw AST from the text file is transformed into the ST. The converters for the AST node labels listed below need first be implemented.

Let



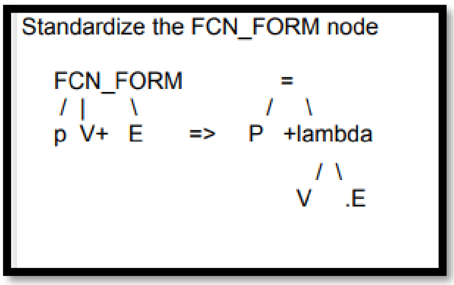
```
  Standardize the LET node

   let            gamma
   /  \            /   \
  =    P        =>       lambda    E
  /\              / \
X  E            X   P
```

Where



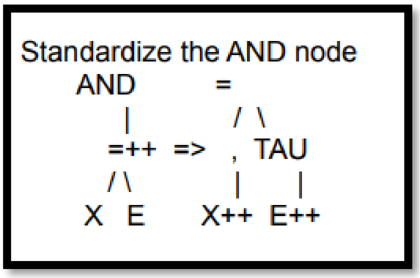
```
Standardize the WHERE node

where          gamma
  /\            /    \
P  =    =>   lambda  E
 /\            / \
X E           X    P
```

Function_form



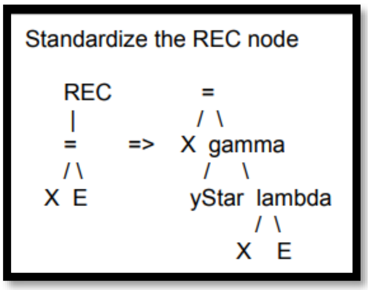
```
Standardize the FCN_FORM node

   FCN_FORM            =
   / |    \          /   \
   p  V+   E     =>   P   +lambda
                           / \
                          V    .E
```

And



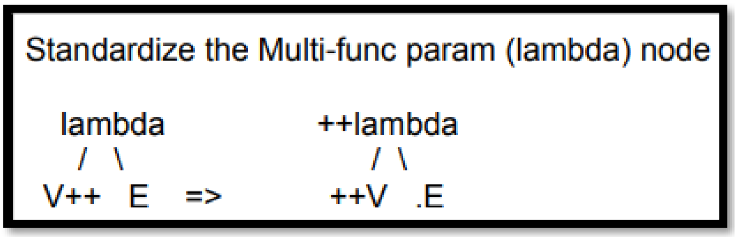
```
Standardize the AND node
     AND          =
       |         / \
     =++  =>   ,  TAU
     /\         |     |
    X  E      X++  E++
```

Rec



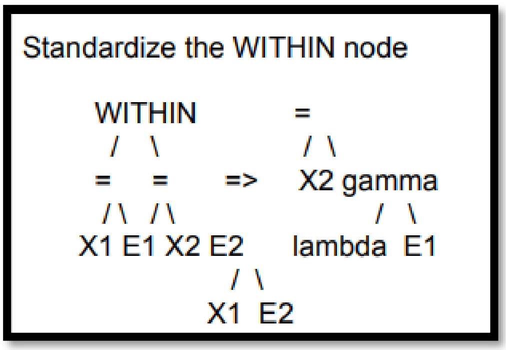
```
Standardize the REC node

    REC           =
     |           / \
     =     =>   X  gamma
    / \           /    \
   X   E      yStar  lambda
                       / \
                      X   E
```

Lambda



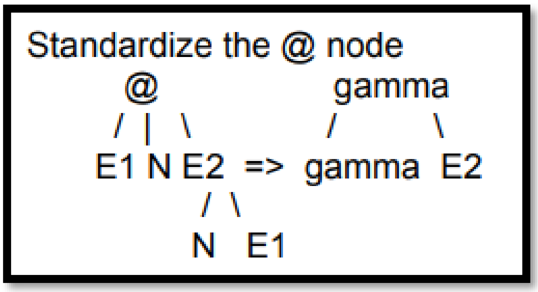
```
Standardize the Multi-func param (lambda) node

   lambda               ++lambda
     /  \                  /  \
  V++   E    =>         ++V   .E
```

Within



```
Standardize the WITHIN node

      WITHIN            =
       /   \           / \
      =     =     =>  X2 gamma
     / \   / \            /  \
    X1 E1 X2 E2       lambda  E1
                         / \
                        X1  E2
```

At



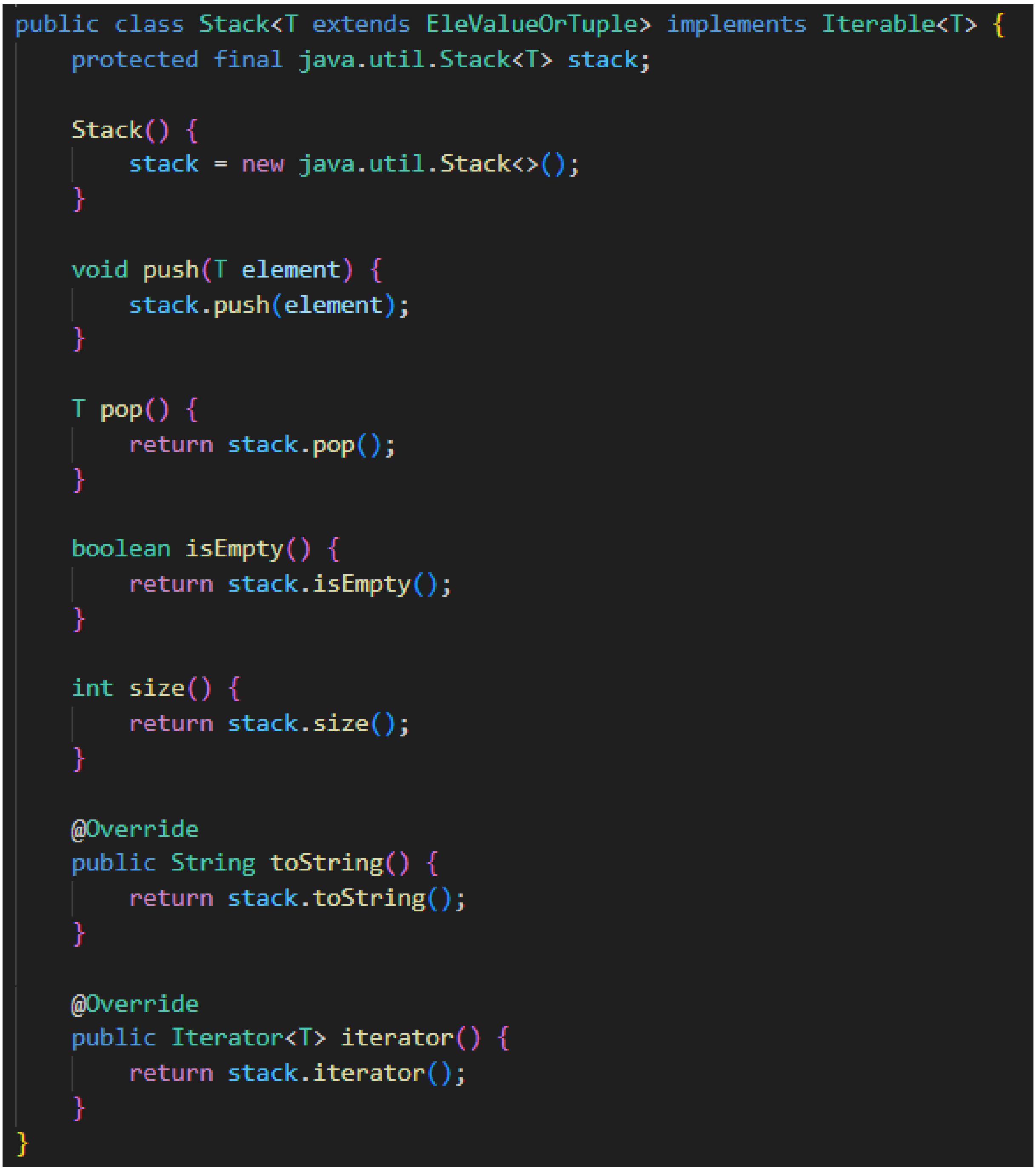
```
Standardize the @ node
        @              gamma
      / | \            /      \
     E1 N E2  =>  gamma   E2
              / \
             N   E1
```

Operation Handling

- Three interfaces in the OperationHandler class distinguish boolean, arithmetic, and array type operations. This class includes implementations of every form of tree implementation.

Make a stack with all the components

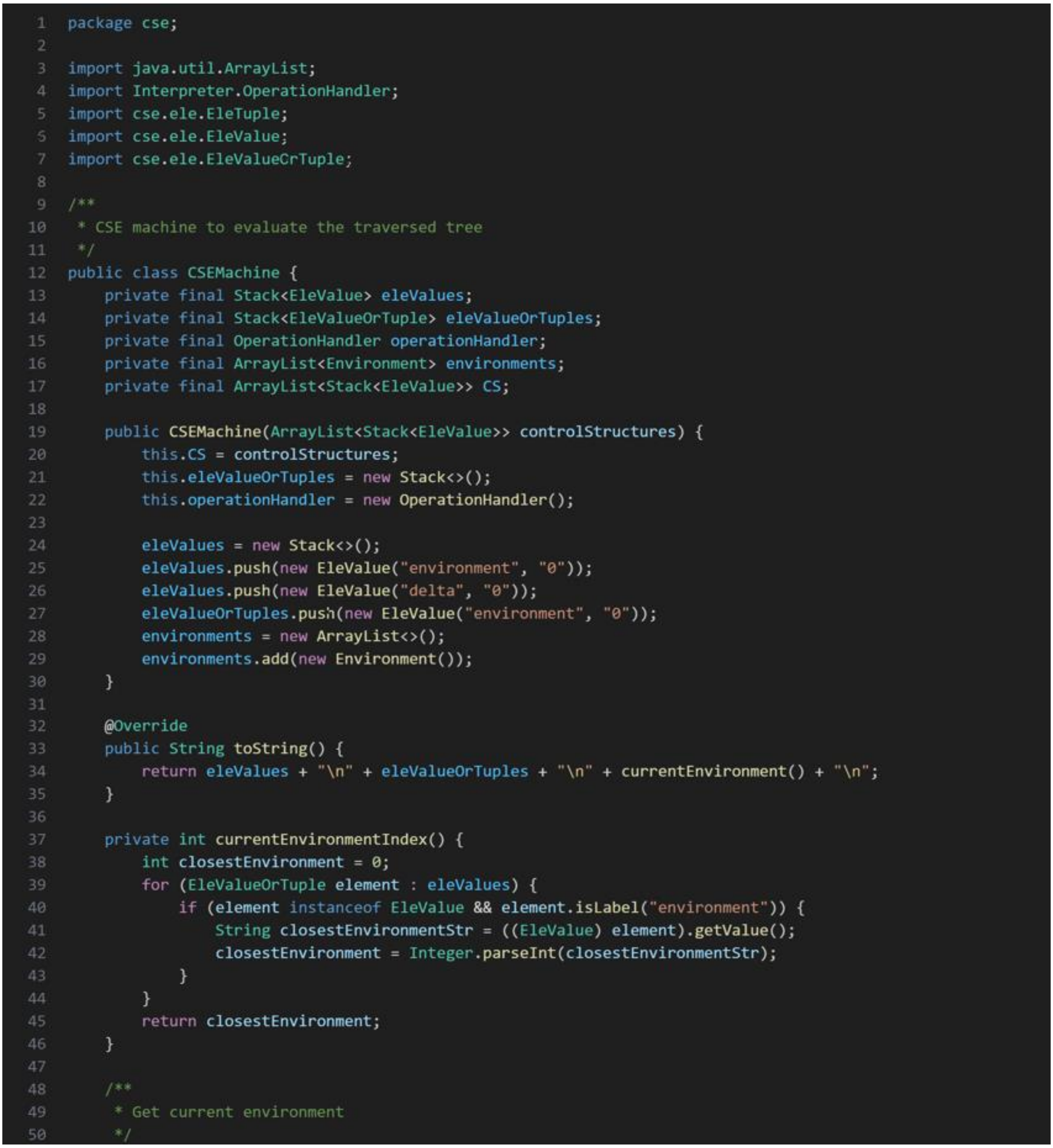
- Create the next stack class to store the components.



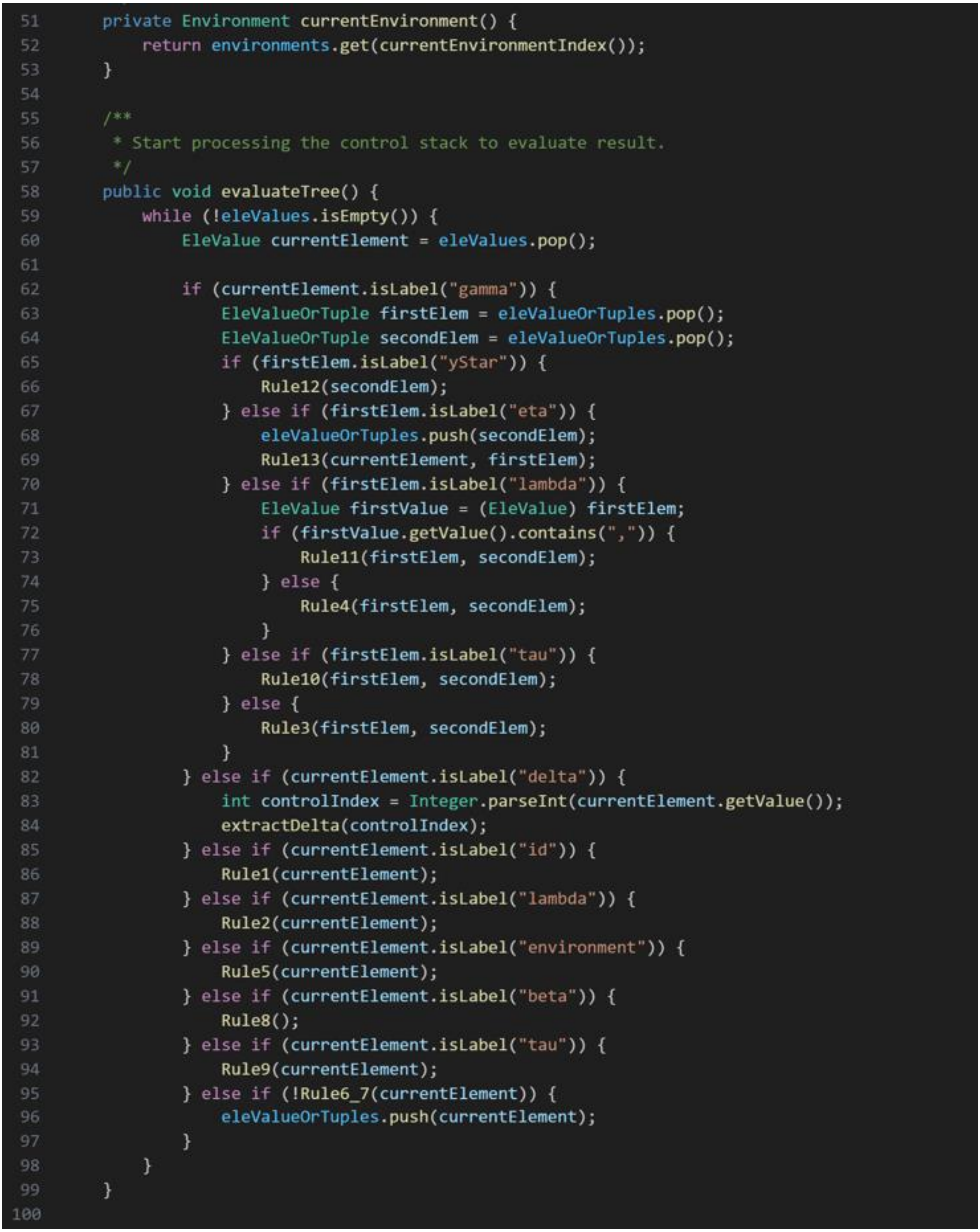
```java
public class Stack<T extends EleValueOrTuple> implements Iterable<T> {
    protected final java.util.Stack<T> stack;

    Stack() {
        stack = new java.util.Stack<>();
    }

    void push(T element) {
        stack.push(element);
    }

    T pop() {
        return stack.pop();
    }

    boolean isEmpty() {
        return stack.isEmpty();
    }

    int size() {
        return stack.size();
    }

    @Override
    public String toString() {
        return stack.toString();
    }

    @Override
    public Iterator<T> iterator() {
        return stack.iterator();
    }
}
```

- Implement operations in the operation handler class

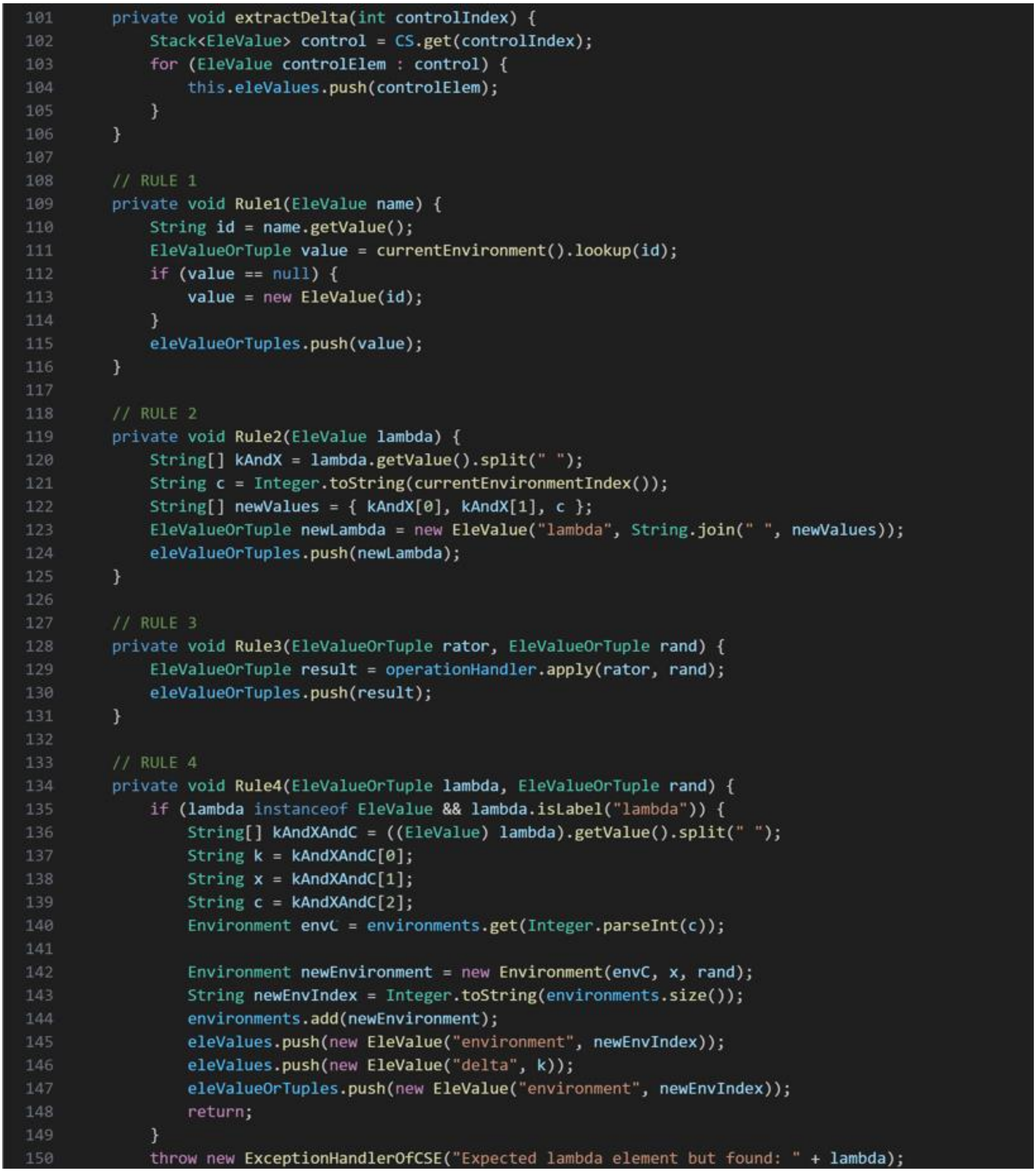
Generate the CSE machine for created ST



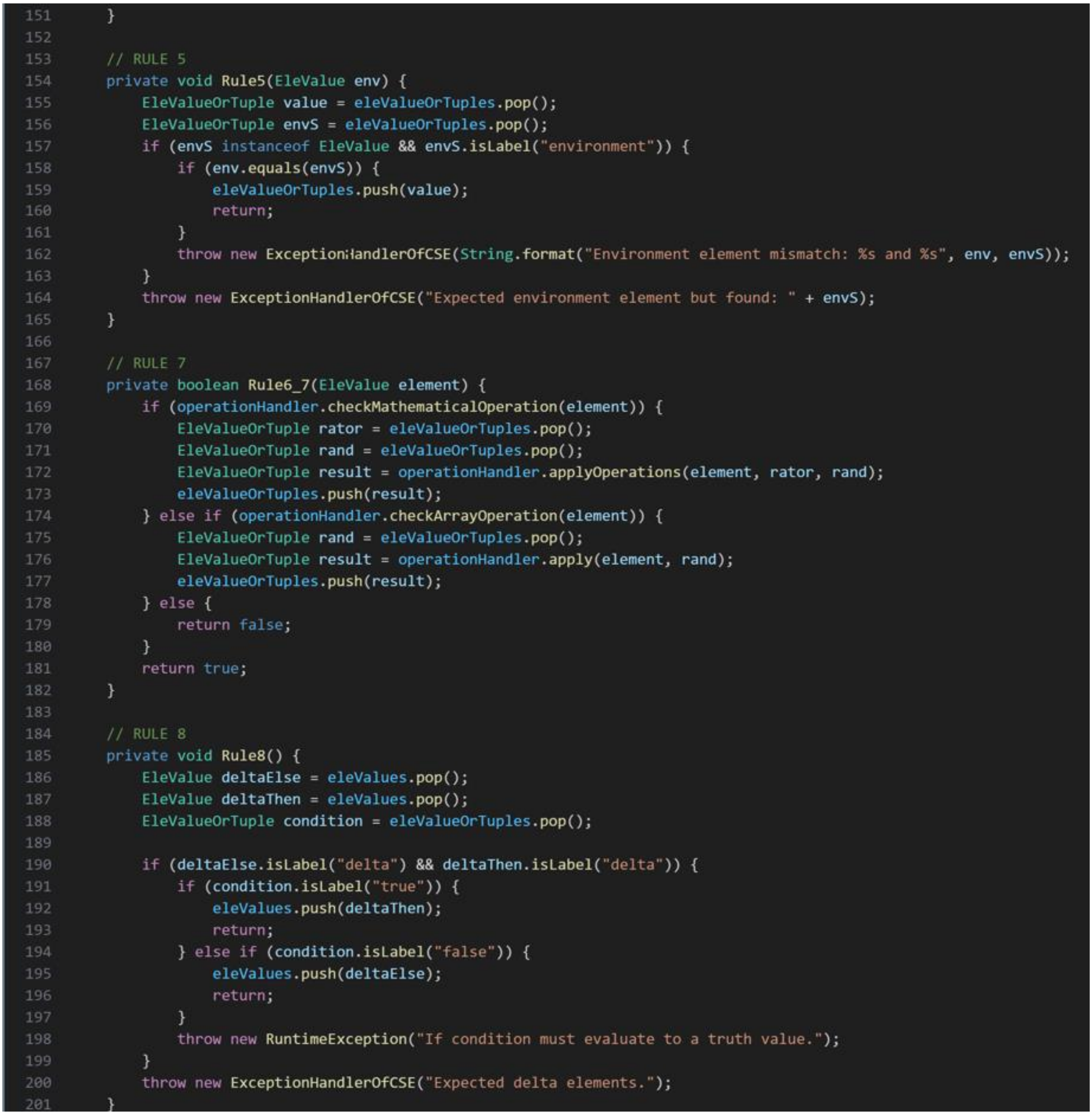
```java
package cse;

import java.util.ArrayList;
import Interpreter.OperationHandler;
import cse.ele.EleTuple;
import cse.ele.EleValue;
import cse.ele.EleValueCrTuple;

/**
 * CSE machine to evaluate the traversed tree
 */
public class CSEMachine {
    private final Stack<EleValue> eleValues;
    private final Stack<EleValueOrTuple> eleValueOrTuples;
    private final OperationHandler operationHandler;
    private final ArrayList<Environment> environments;
    private final ArrayList<Stack<EleValue>> CS;

    public CSEMachine(ArrayList<Stack<EleValue>> controlStructures) {
        this.CS = controlStructures;
        this.eleValueOrTuples = new Stack<>();
        this.operationHandler = new OperationHandler();

        eleValues = new Stack<>();
        eleValues.push(new EleValue("environment", "0"));
        eleValues.push(new EleValue("delta", "0"));
        eleValueOrTuples.push(new EleValue("environment", "0"));
        environments = new ArrayList<>();
        environments.add(new Environment());
    }

    @Override
    public String toString() {
        return eleValues + "\n" + eleValueOrTuples + "\n" + currentEnvironment() + "\n";
    }

    private int currentEnvironmentIndex() {
        int closestEnvironment = 0;
        for (EleValueOrTuple element : eleValues) {
            if (element instanceof EleValue && element.isLabel("environment")) {
                String closestEnvironmentStr = ((EleValue) element).getValue();
                closestEnvironment = Integer.parseInt(closestEnvironmentStr);
            }
        }
        return closestEnvironment;
    }

    /**
     * Get current environment
     */
```



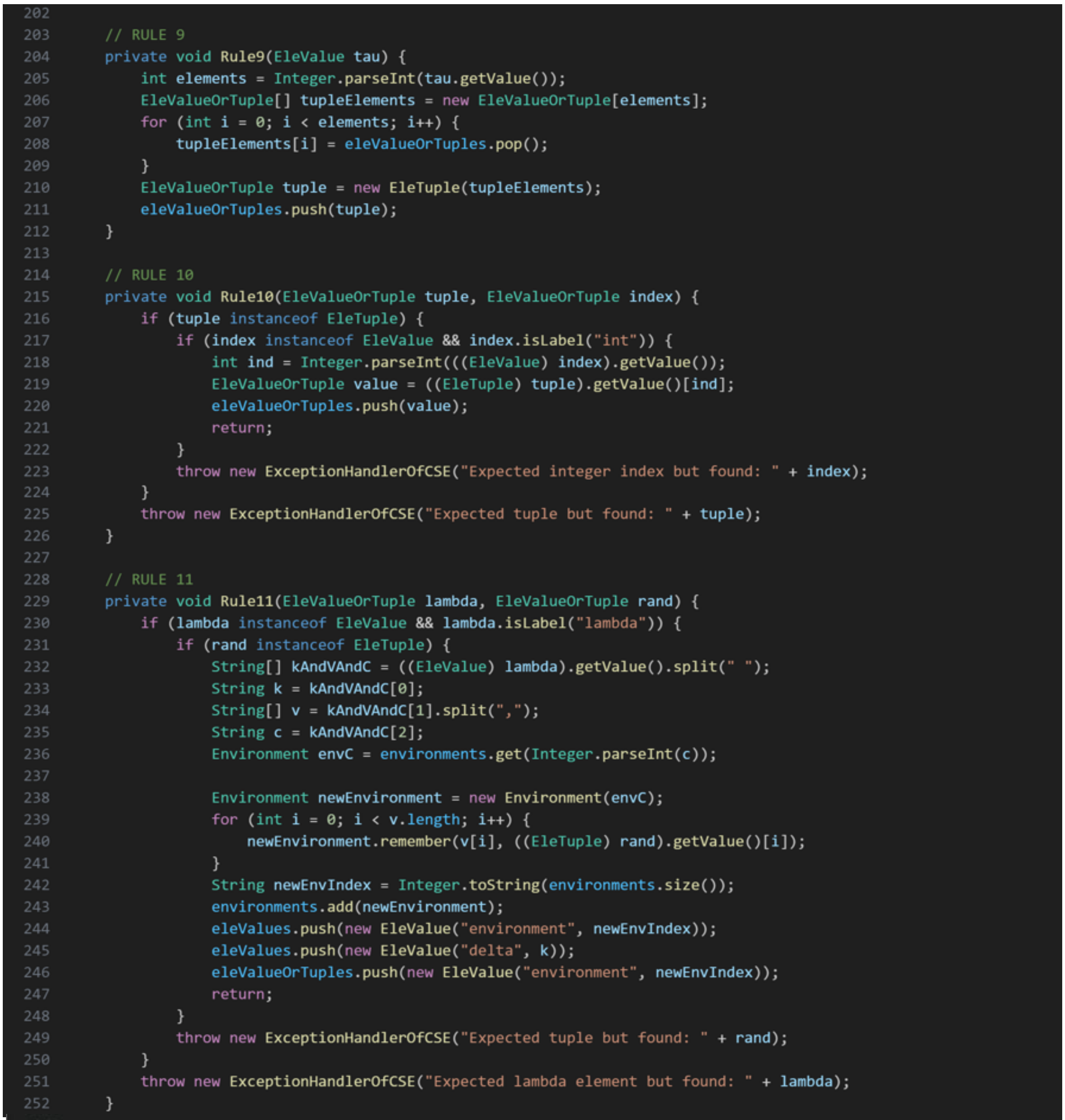
```java
    private Environment currentEnvironment() {
        return environments.get(currentEnvironmentIndex());
    }

    /**
     * Start processing the control stack to evaluate result.
     */
    public void evaluateTree() {
        while (!eleValues.isEmpty()) {
            EleValue currentElement = eleValues.pop();

            if (currentElement.isLabel("gamma")) {
                EleValueOrTuple firstElem = eleValueOrTuples.pop();
                EleValueOrTuple secondElem = eleValueOrTuples.pop();
                if (firstElem.isLabel("yStar")) {
                    Rule12(secondElem);
                } else if (firstElem.isLabel("eta")) {
                    eleValueOrTuples.push(secondElem);
                    Rule13(currentElement, firstElem);
                } else if (firstElem.isLabel("lambda")) {
                    EleValue firstValue = (EleValue) firstElem;
                    if (firstValue.getValue().contains(",")) {
                        Rule11(firstElem, secondElem);
                    } else {
                        Rule4(firstElem, secondElem);
                    }
                } else if (firstElem.isLabel("tau")) {
                    Rule10(firstElem, secondElem);
                } else {
                    Rule3(firstElem, secondElem);
                }
            } else if (currentElement.isLabel("delta")) {
                int controlIndex = Integer.parseInt(currentElement.getValue());
                extractDelta(controlIndex);
            } else if (currentElement.isLabel("id")) {
                Rule1(currentElement);
            } else if (currentElement.isLabel("lambda")) {
                Rule2(currentElement);
            } else if (currentElement.isLabel("environment")) {
                Rule5(currentElement);
            } else if (currentElement.isLabel("beta")) {
                Rule8();
            } else if (currentElement.isLabel("tau")) {
                Rule9(currentElement);
            } else if (!Rule6_7(currentElement)) {
                eleValueOrTuples.push(currentElement);
            }
        }
    }

```



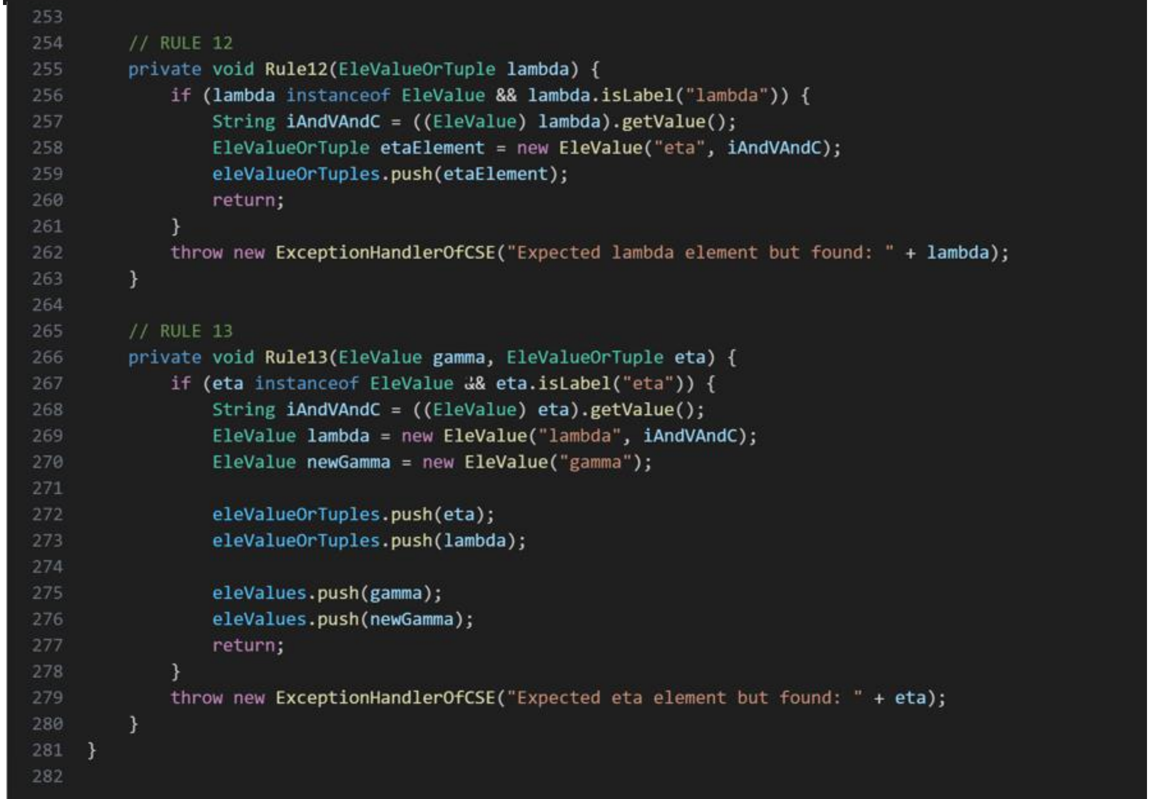
```
    private void extractDelta(int controlIndex) {
        Stack<EleValue> control = CS.get(controlIndex);
        for (EleValue controlElem : control) {
            this.eleValues.push(controlElem);
        }
    }

    // RULE 1
    private void Rule1(EleValue name) {
        String id = name.getValue();
        EleValueOrTuple value = currentEnvironment().lookup(id);
        if (value == null) {
            value = new EleValue(id);
        }
        eleValueOrTuples.push(value);
    }

    // RULE 2
    private void Rule2(EleValue lambda) {
        String[] kAndX = lambda.getValue().split(" ");
        String c = Integer.toString(currentEnvironmentIndex());
        String[] newValues = { kAndX[0], kAndX[1], c };
        EleValueOrTuple newLambda = new EleValue("lambda", String.join(" ", newValues));
        eleValueOrTuples.push(newLambda);
    }

    // RULE 3
    private void Rule3(EleValueOrTuple rator, EleValueOrTuple rand) {
        EleValueOrTuple result = operationHandler.apply(rator, rand);
        eleValueOrTuples.push(result);
    }

    // RULE 4
    private void Rule4(EleValueOrTuple lambda, EleValueOrTuple rand) {
        if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
            String[] kAndXAndC = ((EleValue) lambda).getValue().split(" ");
            String k = kAndXAndC[0];
            String x = kAndXAndC[1];
            String c = kAndXAndC[2];
            Environment envC = environments.get(Integer.parseInt(c));

            Environment newEnvironment = new Environment(envC, x, rand);
            String newEnvIndex = Integer.toString(environments.size());
            environments.add(newEnvironment);
            eleValues.push(new EleValue("environment", newEnvIndex));
            eleValues.push(new EleValue("delta", k));
            eleValueOrTuples.push(new EleValue("environment", newEnvIndex));
            return;
        }
        throw new ExceptionHandlerOfCSE("Expected lambda element but found: " + lambda);
```



```
    }

    // RULE 5
    private void Rule5(EleValue env) {
        EleValueOrTuple value = eleValueOrTuples.pop();
        EleValueOrTuple envS = eleValueOrTuples.pop();
        if (envS instanceof EleValue && envS.isLabel("environment")) {
            if (env.equals(envS)) {
                eleValueOrTuples.push(value);
                return;
            }
            throw new ExceptionHandlerOfCSE(String.format("Environment element mismatch: %s and %s", env, envS));
        }
        throw new ExceptionHandlerOfCSE("Expected environment element but found: " + envS);
    }

    // RULE 7
    private boolean Rule6_7(EleValue element) {
        if (operationHandler.checkMathematicalOperation(element)) {
            EleValueOrTuple rator = eleValueOrTuples.pop();
            EleValueOrTuple rand = eleValueOrTuples.pop();
            EleValueOrTuple result = operationHandler.applyOperations(element, rator, rand);
            eleValueOrTuples.push(result);
        } else if (operationHandler.checkArrayOperation(element)) {
            EleValueOrTuple rand = eleValueOrTuples.pop();
            EleValueOrTuple result = operationHandler.apply(element, rand);
            eleValueOrTuples.push(result);
        } else {
            return false;
        }
        return true;
    }

    // RULE 8
    private void Rule8() {
        EleValue deltaElse = eleValues.pop();
        EleValue deltaThen = eleValues.pop();
        EleValueOrTuple condition = eleValueOrTuples.pop();

        if (deltaElse.isLabel("delta") && deltaThen.isLabel("delta")) {
            if (condition.isLabel("true")) {
                eleValues.push(deltaThen);
                return;
            } else if (condition.isLabel("false")) {
                eleValues.push(deltaElse);
                return;
            }
            throw new RuntimeException("If condition must evaluate to a truth value.");
        }
        throw new ExceptionHandlerOfCSE("Expected delta elements.");
    }
```



```
    // RULE 9
    private void Rule9(EleValue tau) {
        int elements = Integer.parseInt(tau.getValue());
        EleValueOrTuple[] tupleElements = new EleValueOrTuple[elements];
        for (int i = 0; i < elements; i++) {
            tupleElements[i] = eleValueOrTuples.pop();
        }
        EleValueOrTuple tuple = new EleTuple(tupleElements);
        eleValueOrTuples.push(tuple);
    }

    // RULE 10
    private void Rule10(EleValueOrTuple tuple, EleValueOrTuple index) {
        if (tuple instanceof EleTuple) {
            if (index instanceof EleValue && index.isLabel("int")) {
                int ind = Integer.parseInt(((EleValue) index).getValue());
                EleValueOrTuple value = ((EleTuple) tuple).getValue()[ind];
                eleValueOrTuples.push(value);
                return;
            }
            throw new ExceptionHandlerOfCSE("Expected integer index but found: " + index);
        }
        throw new ExceptionHandlerOfCSE("Expected tuple but found: " + tuple);
    }

    // RULE 11
    private void Rule11(EleValueOrTuple lambda, EleValueOrTuple rand) {
        if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
            if (rand instanceof EleTuple) {
                String[] kAndVAndC = ((EleValue) lambda).getValue().split(" ");
                String k = kAndVAndC[0];
                String[] v = kAndVAndC[1].split(",");
                String c = kAndVAndC[2];
                Environment envC = environments.get(Integer.parseInt(c));

                Environment newEnvironment = new Environment(envC);
                for (int i = 0; i < v.length; i++) {
                    newEnvironment.remember(v[i], ((EleTuple) rand).getValue()[i]);
                }
                String newEnvIndex = Integer.toString(environments.size());
                environments.add(newEnvironment);
                eleValues.push(new EleValue("environment", newEnvIndex));
                eleValues.push(new EleValue("delta", k));
                eleValueOrTuples.push(new EleValue("environment", newEnvIndex));
                return;
            }
            throw new ExceptionHandlerOfCSE("Expected tuple but found: " + rand);
        }
        throw new ExceptionHandlerOfCSE("Expected lambda element but found: " + lambda);
    }
```



```
    // RULE 12
    private void Rule12(EleValueOrTuple lambda) {
        if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
            String iAndVAndC = ((EleValue) lambda).getValue();
            EleValueOrTuple etaElement = new EleValue("eta", iAndVAndC);
            eleValueOrTuples.push(etaElement);
            return;
        }
        throw new ExceptionHandlerOfCSE("Expected lambda element but found: " + lambda);
    }

    // RULE 13
    private void Rule13(EleValue gamma, EleValueOrTuple eta) {
        if (eta instanceof EleValue && eta.isLabel("eta")) {
            String iAndVAndC = ((EleValue) eta).getValue();
            EleValue lambda = new EleValue("lambda", iAndVAndC);
            EleValue newGamma = new EleValue("gamma");

            eleValueOrTuples.push(eta);
            eleValueOrTuples.push(lambda);

            eleValues.push(gamma);
            eleValues.push(newGamma);
            return;
        }
        throw new ExceptionHandlerOfCSE("Expected eta element but found: " + eta);
    }
}
```

By using the executeTree function, we can obtain the definitive response to the AST

Exception handling

- The command line parameter is checked first; if there are any invalid arguments, an exception is thrown.
- The exceptionHandlerOfCSE handles any exception that arises during the construction and evaluation of the CSE machine.
- The exceptionHandlerOfAST handles any exception that arises when producing an AST and ST.

References

[1] Wikipedia contributors, “Abstract syntax tree,” Wikipedia, The Free Encyclopedia, 10-Aug-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Abstract_syntax_tree&oldid =1103 626323.

[2] “Abstract syntax tree (AST) in java,” GeeksforGeeks, 11-Aug-2021. [Online]. Available: https://www.geeksforgeeks.org/abstract-syntax-tree-ast-in-java/. [Accessed: 08-Dec-2022].

[3] “What is Syntax Tree?,” Tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/what-is-syntax-tree. [Accessed: 08-Dec-2022].