| expression: expression '+' term  {  if ($1.type != $3.type) {  printf("Type error: Mismatched types in expression.\n");  } else {  $$ = $1 + $3; // Perform addition if types match  }  }  | term; |
| --- |

| declaration: TYPE ID{if (symbol\_table.lookup($2) != NULL) {printf("Error: Variable '%s' is already declared.\n", $2);} else {symbol\_table.insert($2, $1); // Insert the variable name and type into the symbol table}}; |
| --- |

| block: '{' declarations statements '}'{symbol\_table.enter\_scope(); // Start a new scope for the block// Process declarations and statementssymbol\_table.exit\_scope(); // End the scope when the block ends}; |
| --- |

| assignment: ID '=' expression{printf("Generated code: %s = %s\n", $1, $3.code);$$ = $1 + " = " + $3.code; // Generate intermediate code for the assignment}; |
| --- |

| grammar Arithmetic;    @header {  package example;  }    @members {  int result;  }    expr: term ((PLUS|MINUS) term)\* ;  term: factor ((MUL|DIV) factor)\* ;  factor: NUMBER | '(' expr ')' ;    PLUS: '+' ;  MINUS: '-' ;  MUL: '\*' ;  DIV: '/' ;  NUMBER: [0-9]+ ;  WS: [ \t\r\n]+ -> skip ; |
| --- |

| import org.antlr.v4.runtime.\*;  import org.antlr.v4.runtime.tree.\*;    public class ParserExample {  public static void main(String[] args) throws Exception {  // Create an input stream from a string  CharStream input = CharStreams.fromString("sample input");    // Initialize lexer and parser  YourGrammarLexer lexer = new YourGrammarLexer(input);  CommonTokenStream tokens = new CommonTokenStream(lexer);  YourGrammarParser parser = new YourGrammarParser(tokens);    // Parse the input  ParseTree tree = parser.yourStartRule();    // Print the parse tree  System.out.println(tree.toStringTree(parser));  }  } |
| --- |