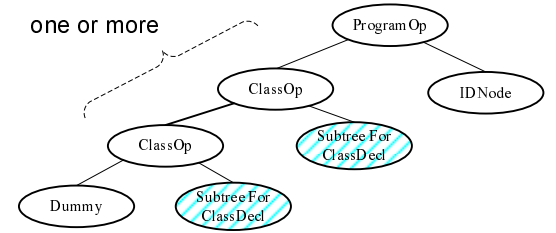
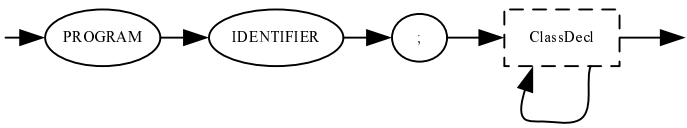
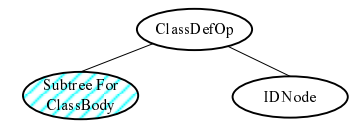
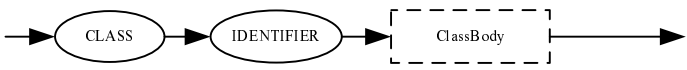


Legend:	dashed boxes → nonterminal symbols	eclipse → normal nodes
	solid ellipsis → terminal symbols (tokens)	shaded eclipse → subtree

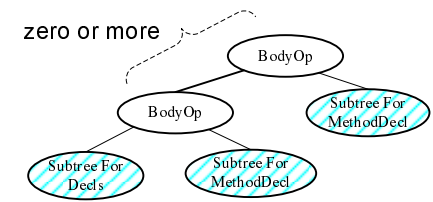
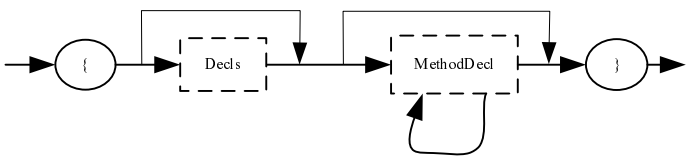
Program



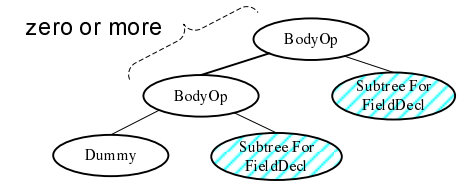
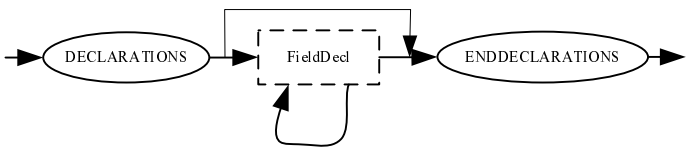
ClassDecl



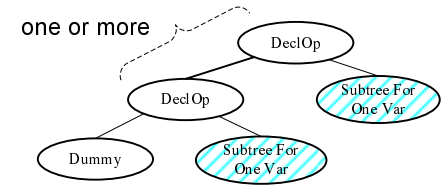
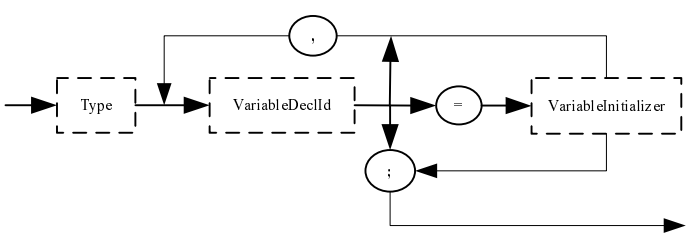
ClassBody



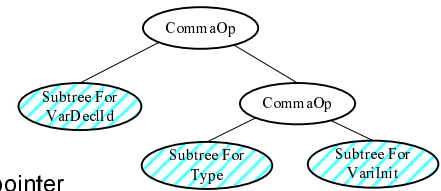
Decls



FieldDecl

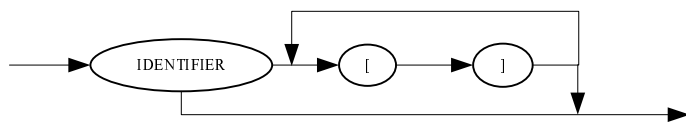


Each Var has the following subtree

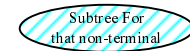
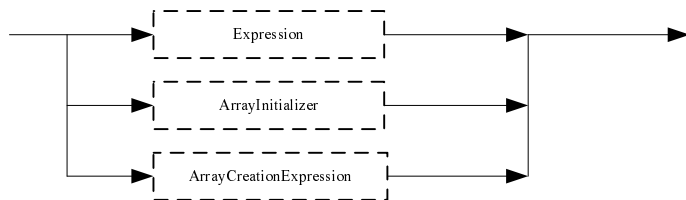


Type should be stored in a separate pointer (global variable) such that it may be used in building the VariableInitializer subtree.

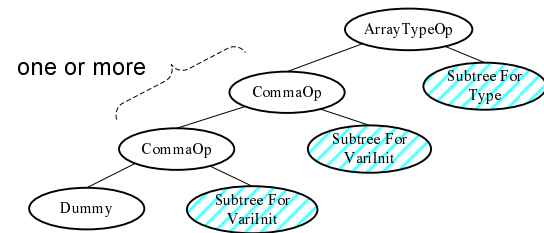
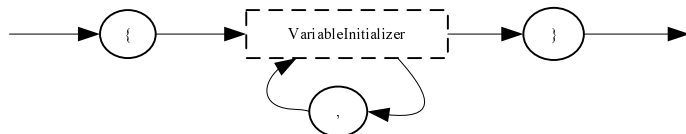
VariableDeclId



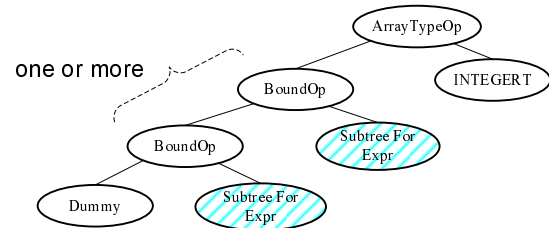
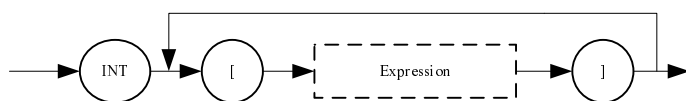
VariableInitializer



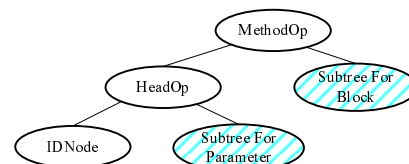
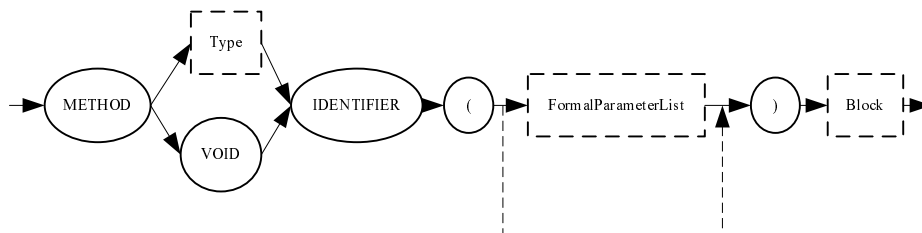
ArrayInitializer



ArrayCreationExpression

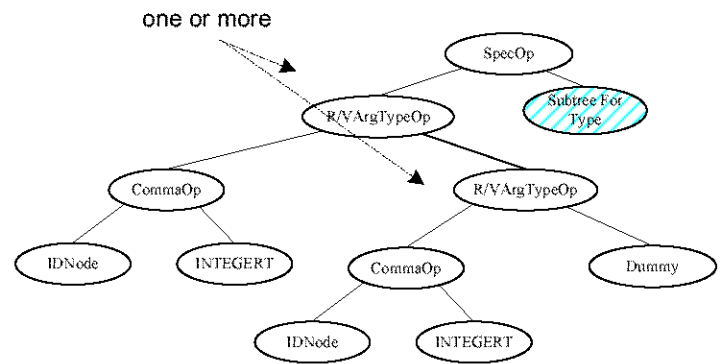
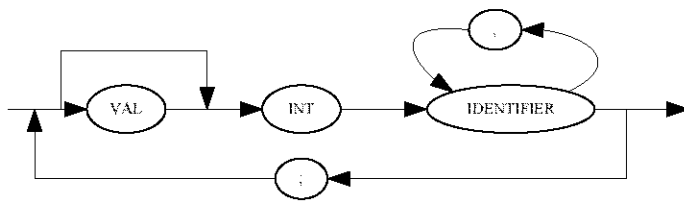


MethodDecl

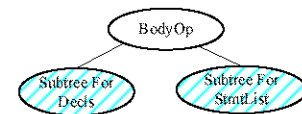
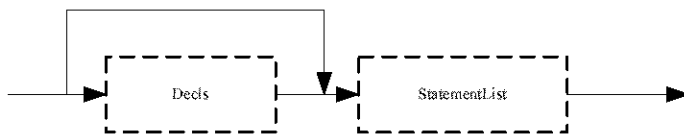


Type should be stored in a separate pointer (global variable) such that it may be used in building the *Parameter* and *Block* subtrees.

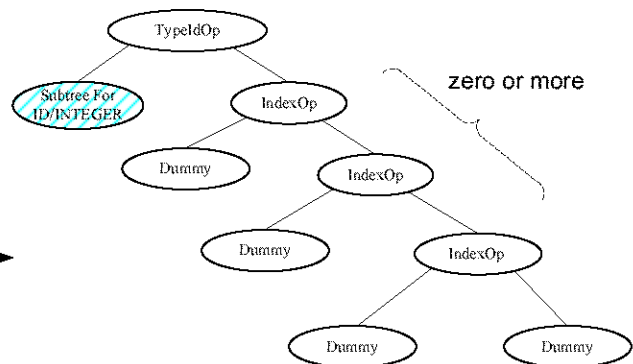
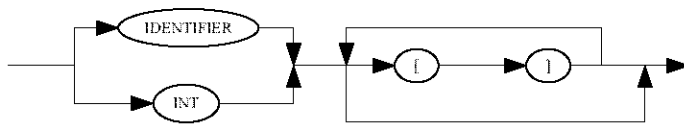
FormalParameterList



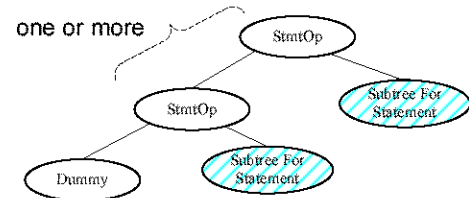
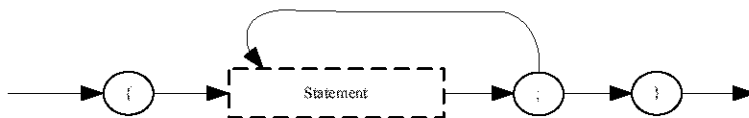
Block



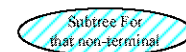
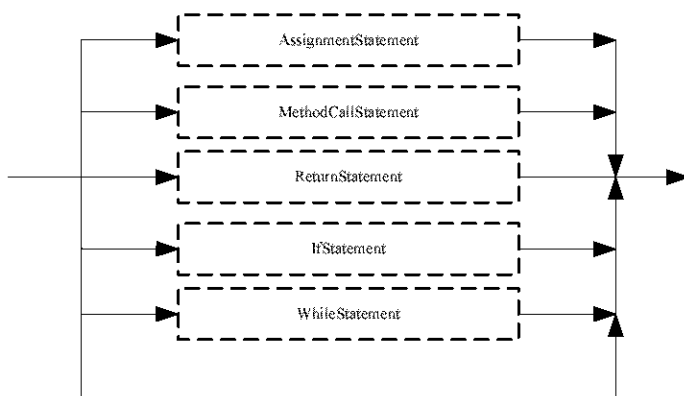
Type



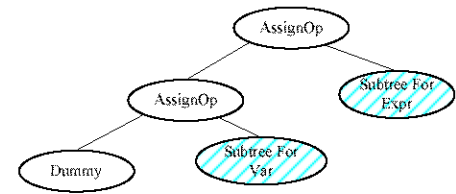
StatementList



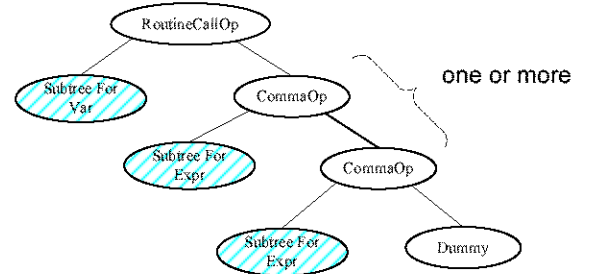
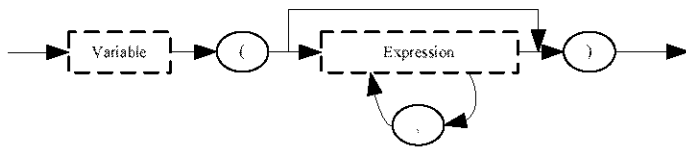
Statement



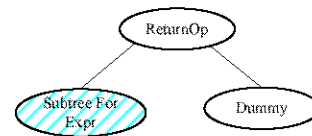
AssignmentStatement



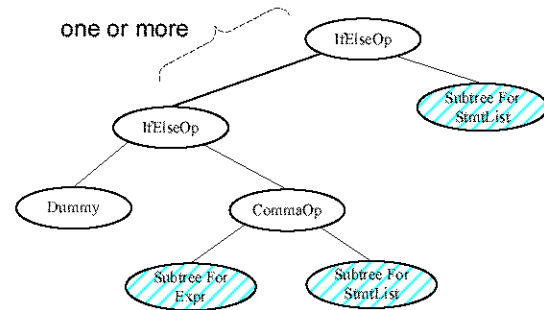
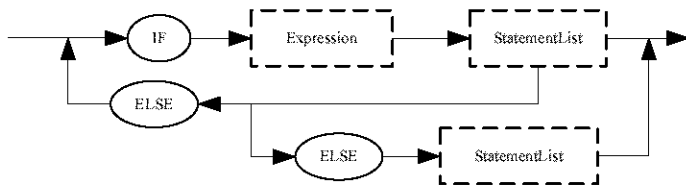
MethodCallStatement



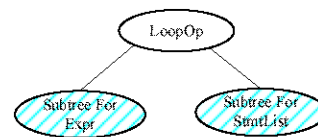
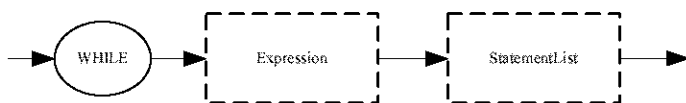
ReturnStatement



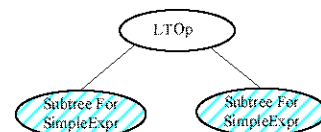
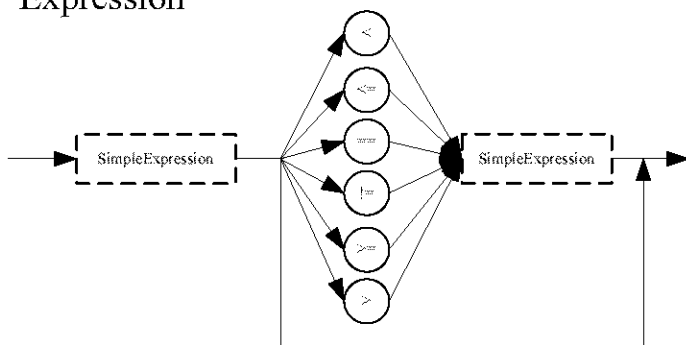
IfStatement



WhileStatement



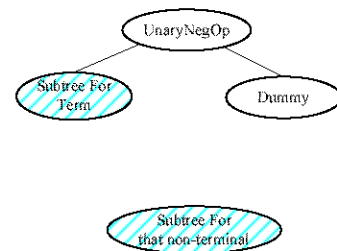
Expression



The figure consists of two parts, (a) and (b), illustrating the evaluation of the postfix expression 3 2 + 5 *.

(a) Initial state: The input stream shows the token '3'. An arrow points from '3' to a stack (represented by a vertical rectangle). The stack now contains '3'.

(b) Next step: The input stream shows the token '2'. An arrow points from '2' to the stack. The stack now contains '3' and '2'.



```

graph LR
    Input(( )) --> UnsignedConstant[UnsignedConstant]
    Input --> Variable[Variable]
    Input --> MethodCallStatement[MethodCallStatement]
    Input --> ExpressionRule[Expression]
    Input --> Factor[Factor]
    
    ExpressionRule --> LP("(")
    LP --> ExpressionRule
    ExpressionRule --> RP(")")
    
    Factor --> NOT("NOT")
    NOT --> Factor
    Factor --> ExpressionRule
    
    UnsignedConstant --> Output(( ))
    Variable --> Output
    MethodCallStatement --> Output
    ExpressionRule --> Output
    Factor --> Output
    Output --> FinalOutput(( ))
  
```

```

graph LR
    Start(( )) --> IDENTIFIER1([IDENTIFIER])
    IDENTIFIER1 --> Join(( ))
    Join --> L1(( ))
    L1 --> LBracket([ [ ])
    LBracket --> Expression[Expression]
    Expression --> RBracket([ ])
    RBracket --> Join
    Join --> L2(( ))
    L2 --> Dot([. ])
    Dot --> IDENTIFIER2([IDENTIFIER])
    IDENTIFIER2 --> Exit(( ))
    style Start fill:none,stroke:none
    style Exit fill:none,stroke:none
  
```

The flowchart illustrates the derivation of the grammar rule $\text{IDENTIFIER} \rightarrow [\text{Expression}] \mid \cdot \text{IDENTIFIER}$. It starts with an entry arrow pointing to an oval labeled "IDENTIFIER". From this oval, an arrow leads to a vertical line that splits into two paths. The top path leads to a circle containing "[", followed by a dashed rectangle labeled "Expression", and then a circle containing "]", all connected by arrows. The bottom path leads to a circle containing ".", followed by an oval labeled "IDENTIFIER", also connected by arrows. Both paths merge back into a single arrow that exits the flowchart to the right.

