Abstract SyntaxTrees for DECAF

Production	AST	Notes
Start -> Class+	ClassList (Class ...)	
Class -> **class** ***id*** Super? { Member* }	Class (SuperClass MemberList) MemberList (Member ...)	Atom id;
Super -> **extends** ***id***	Super (Type)	use Object
Member -> Field | Method | Ctor		
Field -> Modifier* Type VarDeclList	FieldDecl (Field ...) Field (Initializer)	int modifiers Type type; Atom id;
Method -> Modifier* Type ***id*** FormalArgs Block	Method(Type,MethodBody) MethodBody(FormalList StatementList)	Atom id; int modifiers;
Ctor -> Modifier* ***id*** FormalArgs Block	Constructor(Type,MethodBoty)	Atom id; int modifiers;
Modifer -> ...		int
FormalArgs -> **(** FormalArgList? **)**	FormalList (Formal ...)	
FormalArgList -> FormalArg | FormalArg **,** FormalArgList		
FormalArg -> Type VarDeclId	Formal (Type)	Atom id;
Type -> PrimitiveType		
Type -> ***id***	ClassType	Atom id;
Type -> Type **[ ]**	ArrayType	
PrimitiveType -> **boolean** | **char** | **int** | **void**	Boolean, Char, Int, Void (also Null, Init, Meta)	
VarDeclList -> VarDecl { **,** VarDecl}*		
VarDecl -> VarDeclId [**=** Expression]		
VarDeclId -> ***id*** { **[ ]** }		int count;
Block -> **{** Statement* **}**	StatementList (Statement ...)	
Statement -> **;**	EmptyStatement	
Statement -> Type VarDeclList **;**	DeclStatement (Local ...) Local(Expression)	Atom id; int count; init expr

Statement -> **if (** Expression **)** Statement [**else** Statement]	IfStatement(Expression,Statement) IfStatement(Expression,Statement,Statement)	
Statement -> Expression **;**	ExpressionStatement(Expression)	
Statement-> **while (** Expression **)** Statement	WhileStatement(Expression,Statement)	
Statement-> **return** [Expression] **;**	ReturnStatement ReturnStatement(Expression)	
Statement-> **continue ;**	ContinueStatement()	
Statement -> **break ;**	BreakStatement()	
Statement ->Block	BlockStatement(Block(Statement ...))	
Statement -> **super** actual_args **;**	SuperStatement(Call(Name(Name[super],[<init>],ExpressionList))	
Expression -> Expression BinaryOp Expression	OpExpression(Expression,Expression)	OP op;
Expression-> UnaryOp Expression	OpExpression(Expression)	Op op;
Expression -> Primary		
Primary -> NewArrayExpr		
Primary ->NonNewArrayExpr		
Primary -> ***id***	Name	Atom id;
NewArrayExpr -> **new** ***id*** Dimension+	NewArray(Type,Expression...)	
| new PrimitiveType Dimension+		
Dimension -> **[** Expression **]**		
NonNewArrayExpr -> Literal		
NonNewArrayExpr -> **this**	Name	
NonNewArrayExpr -> **(** Expression **)**		
NonNewArrayExpr -> **new** ***id*** ActualArgs	Call(Name(New(Type),[CTOR]),ExpressionList)	
NonNewArrayExpr -> ***id*** ActualArgs	Call(Name(Name[CLASS},[id]) ExpressionList)	
NonNewArrayExpr -> Primary **.** ***id*** ActualArgs	Call(Name(Expression,[id]),ExpressionList)	
NonNewArrayExpr -> **super .** ***id*** ActualArgs	Call(Name(Name[super],[id]),ExpressionList)	
NonNewArrayExpr -> ArrayExpr		
NonNewArrayExpr -> FieldExpr		

FieldExpr -> Primary **.** ***id***	Name[Expression,[id])	
FieldExpr -> **super .** ***id***	Name[Name[super],[id])	
ArrayExpr -> id Dimension	ArrayRef(Name[id],Expression)	
ArrayExpr -> NonNewArrayExpr Dimension	ArrayRef(Expression,Expression)	
Literal -> **null** | **true** | **false** | intLiteral | charLiteral | stringLiteral	LiteralNull(), LiteralBoolean(), LiteralInt(), LiteralChar(), LiteralString()	value
ActualArgs -> **(** ExprList? **)**	ExpressionList(Expression...)	
ExprList -> Expression { **,** Expression }		