Java Grammares:

```
ConstructorDeclaration ::= { Modifier } SimpleName "(" { Parameter } ")" { Throws }
ConstructorBody
ConstructorBody ::= BlockStmt
MethodDeclaration ::= { Modifier } TypeOrVoid SimpleName "(" { Parameter } ")" {
Throws } MethodBody
Modifier ::= Modifier
TypeOrVoid ::= Type | VoidType
SimpleName ::= SimpleName
Parameter ::= ReceiverParameter | Parameter
ReceiverParameter ::= [ Modifier ] Type [ Identifier ]
Parameter ::= [ Modifier ] Type Identifier [ "..." ]
Throws ::= "throws" { ReferenceType }
MethodBody ::= BlockStmt | ";"
BlockStmt ::= "{" { Statement } "}"
Statement ::= AssertStmt
        | BlockStmt
        | BreakStmt
        | ContinueStmt
        | DoStmt
        | EmptyStmt
        | ExplicitConstructorInvocationStmt
        | ExpressionStmt
        | ForEachStmt
        | ForStmt
        | IfStmt
        | LabeledStmt
        | LocalClassDeclarationStmt
        | LocalRecordDeclarationStmt
        | ReturnStmt
        | SwitchStmt
        | SynchronizedStmt
        | ThrowStmt
        | TryStmt
        | UnparsableStmt
        | WhileStmt
        | YieldStmt
Expression ::= ArrayAccessExpr
        | ArrayCreationExpr
        | ArrayInitializerExpr
        | AssignExpr
        | BinaryExpr
        | CastExpr
        | ClassExpr
        | ConditionalExpr
        | EnclosedExpr
        | FieldAccessExpr
        | InstanceOfExpr
        | LambdaExpr
        | LiteralExpr
        | MethodCallExpr
        | MethodReferenceExpr
        | NameExpr
        | ObjectCreationExpr
        | PatternExpr
        | SuperExpr
        | ThisExpr
        | TypeExpr
        | UnaryExpr
        | VariableDeclarationExpr
LiteralExpr ::= BooleanLiteralExpr
         | CharLiteralExpr
         | DoubleLiteralExpr
         | IntegerLiteralExpr
         | LongLiteralExpr
         | NullLiteralExpr
         | StringLiteralExpr
         | TextBlockLiteralExpr
AssertStmt ::= "assert" Expression [ ":" Expression ] ";"
BreakStmt ::= "break" [ SimpleName ] ";"
ContinueStmt ::= "continue" [ SimpleName ] ";"
DoStmt ::= "do" Statement "while" "(" Expression ")" ";"
EmptyStmt ::= ";"
ExplicitConstructorInvocationStmt ::= [ Expression "." ] "super" "(" [ { Expression [ "," ] } ]
")" ";"
                        | [ Expression "." ] "this" "(" [ { Expression [ "," ] } ] ")" ";"
ExpressionStmt ::= Expression ";"
ForEachStmt ::= "for" "(" VariableDeclarator ":" Expression ")" Statement
ForStmt ::= "for" "(" [ ForInit ] ";" [ Expression ] ";" [ ForUpdate ] ")" Statement
ForInit ::= VariableDeclarationExpr | Expression
ForUpdate ::= Expression
IfStmt ::= "if" "(" Expression ")" Statement [ "else" Statement ]
LabeledStmt ::= SimpleName ":" Statement
LocalClassDeclarationStmt ::= ClassOrInterfaceDeclaration
LocalRecordDeclarationStmt ::= RecordDeclaration
ReturnStmt ::= "return" [ Expression ] ";"
SwitchStmt ::= "switch" "(" Expression ")" "{" { SwitchEntry } "}"
SwitchEntry ::= CaseOrDefault ":" { Statement }
CaseOrDefault ::= "case" Expression | "default"
SynchronizedStmt ::= "synchronized" "(" Expression ")" BlockStmt
ThrowStmt ::= "throw" Expression ";"
TryStmt ::= "try" BlockStmt { CatchClause } [ "finally" BlockStmt ]
CatchClause ::= "catch" "(" Parameter ")" BlockStmt
UnparsableStmt ::= <invalid input>
WhileStmt ::= "while" "(" Expression ")" Statement
YieldStmt ::= "yield" Expression ";"
ArrayAccessExpr ::= Expression "[" Expression "]"
ArrayCreationExpr ::= "new" Type { ArrayCreationLevel } [ ArrayInitializerExpr ]
ArrayCreationLevel ::= "[" [ Expression ] "]"
ArrayInitializerExpr ::= "{" [ Expression { "," Expression } ] "}"
AssignExpr ::= Expression AssignOperator Expression
ClassExpr ::= Type "." "class"
ConditionalExpr ::= Expression "?" Expression ":" Expression
EnclosedExpr ::= "(" Expression ")"
FieldAccessExpr ::= Expression "." SimpleName
LambdaExpr ::= "(" [ { Parameter { "," Parameter } } ] ")" "->" ( Expression | BlockStmt )
MethodCallExpr ::= [ Expression "." ] SimpleName "(" [ { Expression { "," Expression } } ]
")"
MethodReferenceExpr ::= Expression "::" SimpleName
NameExpr ::= SimpleName
ObjectCreationExpr ::= "new" Type "(" [ { Expression { "," Expression } } ] ")" [ ClassBody ]
ClassBody ::= "{" { BodyDeclaration } "}"
InstanceOfExpr ::= Expression "instanceof" Type [ PatternExpr ]
PatternExpr ::= TypePatternExpr
          | RecordPatternExpr
TypePatternExpr ::= Type SimpleName
RecordPatternExpr ::= Type "(" { VariableDeclarator { "," VariableDeclarator } } ")"
SuperExpr ::= [ Expression "." ] "super"
ThisExpr ::= [ Expression "." ] "this"
TypeExpr ::= Type
Type ::= ReferenceType | PrimitiveType | VoidType
ReferenceType ::= ClassOrInterfaceType | ArrayType | TypeParameter | WildcardType
PrimitiveType ::= "int" | "boolean" | "char" | "byte" | "short" | "long" | "float" | "double"
VoidType ::= "void"
ClassOrInterfaceType ::= SimpleName [ TypeArgumentList ]
ArrayType ::= Type [ ArrayCreationLevel ]
TypeParameter ::= SimpleName [ "extends" ReferenceType { "&" ReferenceType } ]
WildcardType ::= "?" [ "extends" ReferenceType ] | "?" [ "super" ReferenceType ]
TypeArgumentList ::= "<" TypeArgument { "," TypeArgument } ">"
TypeArgument ::= Type | WildcardType
UnaryExpr ::= UnaryOperator Expression
UnaryOperator ::= "+" | "-" | "++" | "--" | "~" | "!"
VariableDeclarationExpr ::= { Modifier } Type VariableDeclarator { "," VariableDeclarator
}
VariableDeclarator ::= SimpleName [ "=" Expression ]
SwitchExpr ::= "switch" "(" Expression ")" "{" { SwitchEntry } "}"
SwitchEntry ::= CaseOrDefault ":" { Expression }
CaseOrDefault ::= "case" Expression | "default"
```