*Identifier:*

*IDENTIFIER*

*QualifiedIdentifier:*

*Identifier { . Identifier }*

*Literal:*

*IntegerLiteral*

*FloatingPointLiteral*

*CharacterLiteral*

*StringLiteral*

*BooleanLiteral*

*NullLiteral*

*Expression:*

*Expression1 [AssignmentOperator Expression1]]*

*AssignmentOperator:*

=

+=

-=

\*=

/=

&=

|=

^=

%=

<<=

>>=

>>>=

*Type:*

*Identifier [TypeArguments]{ . Identifier [TypeArguments]} {*[]*}*

*BasicType*

*TypeArguments:*

*< TypeArgument {, TypeArgument} >*

*TypeArgument:*

*Type*

? *[(* extends |super ) Type]

StatementExpression:

Expression

ConstantExpression:

Expression

Expression1:

Expression2 [Expression1Rest]

Expression1Rest:

?  *Expression*  : *Expression1*

*Expression2 :*

*Expression3 [Expression2Rest]*

*Expression2Rest:*

*{InfixOp Expression3}*

*Expression3 instanceof Type*

*InfixOp:*

||

&&

|

^

&

==

!=

<

>

<=

>=

<<

>>

>>>

+

-

\*

/

%

*Expression3:*

*PrefixOp Expression3*

( *Expression* | *Type*  )  *Expression3*

*Primary {Selector} {PostfixOp}*

*Primary:*

*ParExpression*

*NonWildcardTypeArguments (ExplicitGenericInvocationSuffix* | this

*Arguments)*

this *[Arguments]*

super *SuperSuffix*

*Literal*

new *Creator*

*Identifier { . Identifier }[ IdentifierSuffix]*

*BasicType {*[]*} .*class

void.class

*IdentifierSuffix:*

[ *(* ] *{*[]*}* . class | *Expression* ]*)*

*Arguments*

*. (* class | *ExplicitGenericInvocation* | this | super *Arguments* | new

*[NonWildcardTypeArguments] InnerCreator )*

*ExplicitGenericInvocation:*

*NonWildcardTypeArguments ExplicitGenericInvocationSuffix*

*NonWildcardTypeArguments:*

*< TypeList >*

*ExplicitGenericInvocationSuffix:*

super *SuperSuffix*

*Identifier Arguments*

*PrefixOp:*

++

--

!

~

+

-

*PostfixOp:*

++

--

*Selector: Selector:*

*. Identifier [Arguments]*

*. ExplicitGenericInvocation*

*.* this

. super *SuperSuffix*

. new *[NonWildcardTypeArguments] InnerCreator*

*[ Expression ]*

*SuperSuffix:*

*Arguments*

*. Identifier [Arguments]*

*BasicType:*

byte

short

char

int

long

float

double

boolean

*Arguments:*

*( [Expression { , Expression }] )*

*Creator:*

*[NonWildcardTypeArguments] CreatedName ( ArrayCreatorRest |*

*ClassCreatorRest )*

*CreatedName:*

*Identifier [NonWildcardTypeArguments] {. Identifier*

*[NonWildcardTypeArguments]}*

*InnerCreator:*

*Identifier ClassCreatorRest*

*ArrayCreatorRest:*

[ *(* ] *{*[]*} ArrayInitializer* | *Expression* ] *{*[ *Expression* ]*} {*[]*} )*

*ClassCreatorRest:*

*Arguments [ClassBody]*

*ArrayInitializer:*

*{ [VariableInitializer {, VariableInitializer} [,]] }*

*VariableInitializer:*

*ArrayInitializer*

*Expression*

*ParExpression:*

*( Expression )*

*Block:*

*{ BlockStatements }*

*BlockStatements:*

*{ BlockStatement }*

*BlockStatement :*

*LocalVariableDeclarationStatement*

*ClassOrInterfaceDeclaration*

*[Identifier :] Statement*

*LocalVariableDeclarationStatement:*

*[*final*] Type VariableDeclarators* ;

*Statement:*

*Block*

assert *Expression [* : *Expression]* ;

if *ParExpression Statement [*else *Statement]*

for *( ForControl ) Statement*

while *ParExpression Statement*

do *Statement* while *ParExpression ;*

try *Block ( Catches* | *[Catches]* finally *Block )*

switch *ParExpression* { *SwitchBlockStatementGroups* }

synchronized *ParExpression Block*

return *[Expression]* ;

throw *Expression*  ;

break *[Identifier]*

continue *[Identifier]*

;

*StatementExpression* ;

*Identifier* :  *Statement*

*Catches:*

*CatchClause {CatchClause}*

*CatchClause:*

catch *( FormalParameter ) Block*

*SwitchBlockStatementGroups:*

*{ SwitchBlockStatementGroup }*

*SwitchBlockStatementGroup:*

*SwitchLabel BlockStatements*

*SwitchLabel:*

case *ConstantExpression* :

case *EnumConstantName* :

*default :*

*MoreStatementExpressions:*

*{ , StatementExpression }*

*ForControl:*

*ForVarControl*

*ForInit; [Expression] ; [ForUpdate]*

*ForVarControl*

*[*final*] [Annotations] Type Identifier ForVarControlRest*

*Annotations:*

*Annotation [Annotations]*

*Annotation:*

@ *TypeName [( [Identifier =] ElementValue)]*

*ElementValue:*

*ConditionalExpression*

*Annotation*

*ElementValueArrayInitializer*

*ConditionalExpression:*

*Expression2 Expression1Rest*

*ElementValueArrayInitializer:*

{ *[ElementValues] [,]* }

*ElementValues:*

*ElementValue [ElementValues]*

*ForVarControlRest:*

*VariableDeclaratorsRest; [Expression] ; [ForUpdate]*

*: Expression*

*ForInit:*

*StatementExpression Expressions*

*Modifier:*

*Annotation*

public

protected

private

static

abstract

final

native

synchronized

transient

volatile

strictfp

*VariableDeclarators:*

*VariableDeclarator { , VariableDeclarator }*

*VariableDeclaratorsRest:*

*VariableDeclaratorRest { , VariableDeclarator }*

*ConstantDeclaratorsRest:*

*ConstantDeclaratorRest { , ConstantDeclarator }*

*VariableDeclarator:*

*Identifier VariableDeclaratorRest*

*ConstantDeclarator:*

*Identifier ConstantDeclaratorRest*

*VariableDeclaratorRest:*

*{*[]*} [ = VariableInitializer]*

*ConstantDeclaratorRest:*

*{*[]*} = VariableInitializer*

*VariableDeclaratorId:*

*Identifier {*[]*}*

*CompilationUnit:*

*[[Annotations]* package *QualifiedIdentifier ; ] {ImportDeclaration}*

*{TypeDeclaration}*

*ImportDeclaration:*

import *[* static*] Identifier { . Identifier } [ . \* ] ;*

*TypeDeclaration:*

*ClassOrInterfaceDeclaration*

*;*

*ClassOrInterfaceDeclaration:*

*{Modifier} (ClassDeclaration* | *InterfaceDeclaration)*

*ClassDeclaration:*

*NormalClassDeclaration*

*EnumDeclaration*

*NormalClassDeclaration:*

class *Identifier [TypeParameters] [*extends *Type] [*implements *TypeList]*

*ClassBody*

*TypeParameters:*

*< TypeParameter {, TypeParameter} >*

*TypeParameter:*

*Identifier [*extends *Bound]*

*Bound:*

*Type {& Type}*

*EnumDeclaration:*

enum *Identifier [*implements *TypeList] EnumBody*

*EnumBody:*

*{ [EnumConstants] [,] [EnumBodyDeclarations] }*

*EnumConstants:*

*EnumConstant*

*EnumConstants , EnumConstant*

*EnumConstant:*

*Annotations Identifier [Arguments] [ClassBody]*

*EnumBodyDeclarations:*

*; {ClassBodyDeclaration}*

*InterfaceDeclaration:*

*NormalInterfaceDeclaration*

*AnnotationTypeDeclaration*

*NormalInterfaceDeclaration:*

interface *Identifier [ TypeParameters] [*extends *TypeList] InterfaceBody*

*TypeList:*

*Type { , Type}*

*AnnotationTypeDeclaration:*

@ *interface Identifier AnnotationTypeBody*

*AnnotationTypeBody:*

*{ [AnnotationTypeElementDeclarations] }*

*AnnotationTypeElementDeclarations:*

*AnnotationTypeElementDeclaration*

*AnnotationTypeElementDeclarations AnnotationTypeElementDeclaration*

*AnnotationTypeElementDeclaration:*

*{Modifier} AnnotationTypeElementRest*

*AnnotationTypeElementRest:*

*Type Identifier AnnotationMethodOrConstantRest;*

*ClassDeclaration*

*InterfaceDeclaration*

*EnumDeclaration*

*AnnotationTypeDeclaration*

*AnnotationMethodOrConstantRest:*

*AnnotationMethodRest*

*AnnotationConstantRest*

*AnnotationMethodRest:*

*( ) [DefaultValue]*

*AnnotationConstantRest:*

*VariableDeclarators*

*DefaultValue:*

default *ElementValue*

*ClassBody:*

{ *{ClassBodyDeclaration}* }

*InterfaceBody:*

{ *{InterfaceBodyDeclaration}* }

*ClassBodyDeclaration:*

;

*[*static*] Block*

*{Modifier} MemberDecl*

*MemberDecl:*

*GenericMethodOrConstructorDecl*

*MethodOrFieldDecl*

void *Identifier VoidMethodDeclaratorRest*

*Identifier ConstructorDeclaratorRest*

*InterfaceDeclaration*

*ClassDeclaration*

*GenericMethodOrConstructorDecl:*

*TypeParameters GenericMethodOrConstructorRest*

*GenericMethodOrConstructorRest:*

*(Type* | void*) Identifier MethodDeclaratorRest*

*Identifier ConstructorDeclaratorRest*

*MethodOrFieldDecl:*

*Type Identifier MethodOrFieldRest*

*MethodOrFieldRest:*

*VariableDeclaratorRest*

*MethodDeclaratorRest*

*InterfaceBodyDeclaration:*

;

*{Modifier} InterfaceMemberDecl*

*InterfaceMemberDecl:*

*InterfaceMethodOrFieldDecl*

*InterfaceGenericMethodDecl*

void *Identifier VoidInterfaceMethodDeclaratorRest*

*InterfaceDeclaration*

*ClassDeclaration*

*InterfaceMethodOrFieldDecl:*

*Type Identifier InterfaceMethodOrFieldRest*

*InterfaceMethodOrFieldRest:*

*ConstantDeclaratorsRest ;*

*InterfaceMethodDeclaratorRest*

*MethodDeclaratorRest:*

*FormalParameters {*[]*} [*throws *QualifiedIdentifierList] ( MethodBody* | ;

*)*

*VoidMethodDeclaratorRest:*

*FormalParameters [*throws *QualifiedIdentifierList] ( MethodBody* | ; *)*

*InterfaceMethodDeclaratorRest:*

*FormalParameters {*[]*} [*throws *QualifiedIdentifierList]*  ;

*InterfaceGenericMethodDecl:*

*TypeParameters (Type* | void*) Identifier InterfaceMethodDeclaratorRest*

*VoidInterfaceMethodDeclaratorRest:*

*FormalParameters [*throws *QualifiedIdentifierList]* ;

*ConstructorDeclaratorRest:*

*FormalParameters [*throws *QualifiedIdentifierList] MethodBody*

*QualifiedIdentifierList:*

*QualifiedIdentifier { , QualifiedIdentifier}*

*FormalParameters:*

*( [FormalParameterDecls] )*

*FormalParameterDecls:*

*[*final*] [Annotations] Type FormalParameterDeclsRest]*

*FormalParameterDeclsRest:*

*VariableDeclaratorId [ , FormalParameterDecls]*

*... VariableDeclaratorId*

*MethodBody:*

*Block*

*EnumConstantName:*

*Identifier*