```
compilationUnit          -> packageDeclaration?
importDeclaration* typeDeclaration*

packageDeclaration     -> "package"
qualifiedName ";"

importDeclaration       -> "import"
qualifiedName ( "." "*" )? ";"

typeDeclaration          -> classDeclaration |
interfaceDeclaration | enumDeclaration |
annotationDeclaration

classDeclaration        -> modifiers? "class"
identifier typeParameters? superclass?
superinterfaces? classBody
```

```
interfaceDeclaration    -> modifiers? "interface" identifier typeParameters? extendsInterfaces? interfaceBody

enumDeclaration         -> modifiers? "enum" identifier interfaces? enumBody

annotationDeclaration   -> modifiers? "@" "interface" identifier annotationBody

modifiers               -> modifier*

modifier                -> "public" | "protected" | "private" | "static" | "abstract" | "final" | "native" | "synchronized" | "transient" | "volatile" | "strictfp"

typeParameters          -> "<" typeParameterList ">"

typeParameterList       -> typeParameter ( ","
```

typeParameter)*

typeParameter   -> typeBound? identifier

typeBound   -> "extends" type ("&" type)*

superclass   -> "extends" classType

superinterfaces   -> "implements" interfaceTypeList

interfaceTypeList   -> interfaceType ("," interfaceType)*

classBody   -> "{" classBodyDeclaration* "}"

interfaceBody   -> "{" interfaceBodyDeclaration* "}"

enumBody   -> "{" enumConstants? (","

```
enumBodyDeclarations?)? "}"

enumConstants          -> enumConstant ( ","
enumConstant )*

enumConstant           -> annotations?
identifier ( arguments? )? classBody?

enumBodyDeclarations   -> ";"
classBodyDeclaration*

arguments              -> "(" ( expressionList )? ")"

classBodyDeclaration   -> block |
fieldDeclaration | methodDeclaration |
classDeclaration | interfaceDeclaration |
enumDeclaration | annotationDeclaration

fieldDeclaration       -> modifiers? type
variableDeclarators ";"

variableDeclarators    -> variableDeclarator (
```

"," variableDeclarator)*

variableDeclarator   -> variableDeclaratorId ("=" variableInitializer)?

variableDeclaratorId   -> identifier dimensions?

variableInitializer   -> expression | arrayInitializer

methodDeclaration   -> modifiers? typeParameters? type identifier formalParameters (dims)? throws? methodBody

formalParameters   -> "(" (formalParameterList)? ")"

formalParameterList   -> formalParameter ("," formalParameter)*

formalParameter -> variableModifier* type variableDeclaratorId

methodBody -> block | ";"

constantDeclaration -> modifiers? type variableDeclarator "=" variableInitializer ";"

interfaceMemberDeclaration -> constantDeclaration | methodDeclaration | classDeclaration | interfaceDeclaration | annotationDeclaration

block -> "{" blockStatement* "}"

blockStatement -> localVariableDeclarationStatement | statement

localVariableDeclarationStatement -> localVariableDeclaration ";"

localVariableDeclaration -> modifiers? type variableDeclarators

statement   -> block | ifStatement | whileStatement | doStatement | forStatement | switchStatement | returnStatement | breakStatement | continueStatement | throwStatement | tryStatement | expressionStatement | emptyStatement | assertStatement | labeledStatement

ifStatement   -> "if" "(" expression ")" statement ("else" statement)?

whileStatement   -> "while" "(" expression ")" statement

doStatement   -> "do" statement "while" "("
expression ")" ";"

forStatement   -> "for" "(" forControl ")" statement

forControl -> enhancedForControl | forInit? ";" expression? ";" forUpdate?

forInit -> statementExpressionList | localVariableDeclaration

forUpdate -> statementExpressionList

enhancedForControl -> modifiers? type identifier ":" expression

switchStatement -> "switch" "(" expression ")" switchBlock

switchBlock -> "{" switchBlockStatementGroup* switchLabel* "}"

switchBlockStatementGroup -> switchLabels blockStatements

```
switchLabels           -> switchLabel (
switchLabel )*

switchLabel            -> "case"
constantExpression ":" | "default" ":"

returnStatement        -> "return" expression? ";"

breakStatement         -> "break" identifier? ";"

continueStatement      -> "continue"
identifier? ";"

throwStatement         -> "throw" expression ";"

tryStatement           -> "try" block catches
finallyBlock?

catches                -> catchClause (
catchClause )*

catchClause            -> "catch" "("
```

catchFormalParameter ")" block

catchFormalParameter   -> modifiers?
catchType variableDeclaratorId

catchType   -> qualifiedType |
unionType

finallyBlock   -> "finally" block

expressionStatement   ->
statementExpression ";"

emptyStatement   -> ";"

assertStatement   -> "assert" expression (
":" expression)? ";"

labeledStatement   -> identifier ":"
statement

statementExpressionList ->

statementExpression (","
statementExpression)*

statementExpression -> assignment |
methodInvocation |
classInstanceCreationExpression

assignment -> leftHandSide
assignmentOperator expression

leftHandSide -> expressionName |
fieldAccess | arrayAccess

assignmentOperator -> "=" | "+=" | "-=" | "*="
| "/=" | "%=" | "&=" | "|=" | "^=" | "<<=" | ">>=" |
">>>="

constantExpression -> expression

primary -> primaryNoNewArray |
arrayCreationExpression

primaryNoNewArray   -> literal | "this" | "super" | "(" expression ")" | classInstanceCreationExpression | fieldAccess | methodInvocation | arrayAccess

classInstanceCreationExpression -> "new" classOrInterfaceTypeToInstantiate (arguments? classBody?)

fieldAccess   -> primary "." identifier

methodInvocation   -> methodName "(" argumentList? ")"

arrayAccess   -> expressionName "[" expression "]"

arguments   -> "(" (expressionList)? ")"

expressionList   -> expression ("," expression)*

```
classOrInterfaceTypeToInstantiate ->
classOrInterfaceType

classOrInterfaceType    -> classType |
interfaceType

classType               -> identifier
typeArguments?

interfaceType           -> identifier
typeArguments?

typeArguments           -> "<" typeArgumentList
">"

typeArgumentList        -> typeArgument ( ","
typeArgument )*

typeArgument            -> referenceType | "?" ( (
"extends" | "super" ) referenceType )?

referenceType           -> classType |
```

interfaceType | arrayType

arrayType   -> primitiveType dims | classOrInterfaceType dims | typeVariable dims

dims   -> "[" "]" ("[" "]")*

primitiveType   -> "byte" | "short" | "int" | "long" | "char" | "float" | "double" | "boolean"