

# 1 Lexical Grammar

```
1 // Whitespace -- ignored
2 " " | "\t" | "\n" | "\r" | "\f"
3
4 // Single line comment -- ignored
5 "/*" { ~( "\n" | "\r" ) } ( "\n" | "\r" ["\n"] )
6
7 // Reserved words
8 ABSTRACT      ::= "abstract"
9 BOOLEAN       ::= "boolean"
10 CHAR          ::= "char"
11 CLASS         ::= "class"
12 ELSE          ::= "else"
13 EXTENDS       ::= "extends"
14 FALSE         ::= "false"
15 IF            ::= "if"
16 IMPORT        ::= "import"
17 INSTANCEOF    ::= "instanceof"
18 INT           ::= "int"
19 NEW           ::= "new"
20 NULL          ::= "null"
21 PACKAGE       ::= "package"
22 PRIVATE       ::= "private"
23 PROTECTED     ::= "protected"
24 PUBLIC        ::= "public"
25 RETURN        ::= "return"
26 STATIC        ::= "static"
27 SUPER         ::= "super"
28 THIS          ::= "this"
29 TRUE          ::= "true"
30 VOID          ::= "void"
31 WHILE         ::= "while"
32
33 // Separators
34 COMMA         ::= ","
35 DOT           ::= "."
36 LBRACK        ::= "["
37 LCURLY        ::= "{"
38 LPAREN        ::= "("
39 RBRACK        ::= "]"
40 RCURLY        ::= "}"
41 RPAREN        ::= ")"
42 SEMI          ::= ";"
43
44 // Operators
45 ASSIGN        ::= "="
46 DEC           ::= "--"
47 EQUAL         ::= "=="
48 GT            ::= ">"
49 INC           ::= "++"
50 LAND          ::= "&&"
51 LE            ::= "<="
52 LNOT          ::= "!"
```

```

53 MINUS      ::= "-"
54 PLUS       ::= "+"
55 PLUS_ASSIGN ::= "+="
56 STAR       ::= "*"
57
58 // Identifiers
59 IDENTIFIER ::= ( "a"... "z" | "A"... "Z" | "_" | "$" )
60             { "a"... "z" | "A"... "Z" | "_" | "0"... "9" | "$" }
61
62 // Literals
63 INT_LITERAL ::= ( "0"... "9" ) { "0"... "9" }
64 ESC        ::= "\\\" ( "n" | "r" | "t" | "b" | "f" | "\"" | "\\\" )
65 STRING_LITERAL ::= "\\\" { ESC | ~( "\\\" | "\\\" | "\\n" | "\\r" ) } "\\\"
66 CHAR_LITERAL ::= "\"" ( ESC | ~( "\"" | "\\n" | "\\r" | "\\\" ) ) "\""
67
68 // End of file
69 EOF        ::= "<end of file>"

```

## 2 Syntactic Grammar

```

1 compilationUnit ::= [ PACKAGE qualifiedIdentifier SEMI ]
2                 { IMPORT qualifiedIdentifier SEMI }
3                 { typeDeclaration }
4                 EOF
5
6 qualifiedIdentifier ::= IDENTIFIER { DOT IDENTIFIER }
7
8 typeDeclaration ::= modifiers classDeclaration
9
10 modifiers ::= { ABSTRACT | PRIVATE | PROTECTED | PUBLIC | STATIC }
11
12 classDeclaration ::= CLASS IDENTIFIER [ EXTENDS qualifiedIdentifier ] classBody
13
14 classBody ::= LCURLY { modifiers memberDecl } RCURLY
15
16 memberDecl ::= IDENTIFIER formalParameters block
17             | ( VOID | type ) IDENTIFIER formalParameters ( block | SEMI )
18             | type variableDeclarators SEMI
19
20 block ::= LCURLY { blockStatement } RCURLY
21
22 blockStatement ::= localVariableDeclarationStatement
23                | statement
24
25 statement ::= block
26            | IF parExpression statement [ ELSE statement ]
27            | RETURN [ expression ] SEMI
28            | SEMI
29            | WHILE parExpression statement
30            | statementExpression SEMI
31
32 formalParameters ::= LPAREN [ formalParameter { COMMA formalParameter } ] RPAREN
33

```

```
34 formalParameter ::= type IDENTIFIER
35
36 parExpression ::= LPAREN expression RPAREN
37
38 localVariableDeclarationStatement ::= type variableDeclarators SEMI
39
40 variableDeclarators ::= variableDeclarator { COMMA variableDeclarator }
41
42 variableDeclarator ::= IDENTIFIER [ ASSIGN variableInitializer ]
43
44 variableInitializer ::= arrayInitializer | expression
45
46 arrayInitializer ::= LCURLY [ variableInitializer { COMMA variableInitializer } [ COMMA ] ] RCURLY
47
48 arguments ::= LPAREN [ expression { COMMA expression } ] RPAREN
49
50 type ::= referenceType | basicType
51
52 basicType ::= BOOLEAN | CHAR | INT
53
54 referenceType ::= basicType LBRACK RBRACK { LBRACK RBRACK }
55                 | qualifiedIdentifier { LBRACK RBRACK }
56
57 statementExpression ::= expression
58
59 expression ::= assignmentExpression
60
61 assignmentExpression ::= conditionalAndExpression [ ( ASSIGN | PLUS_ASSIGN ) assignmentExpression ]
62
63 conditionalAndExpression ::= equalityExpression { LAND equalityExpression }
64
65 equalityExpression ::= relationalExpression { EQUAL relationalExpression }
66
67 relationalExpression ::= additiveExpression [ ( GT | LE ) additiveExpression
68                       | INSTANCEOF referenceType ]
69
70 additiveExpression ::= multiplicativeExpression { ( MINUS | PLUS ) multiplicativeExpression }
71
72 multiplicativeExpression ::= unaryExpression { STAR unaryExpression }
73
74 unaryExpression ::= INC unaryExpression
75                 | MINUS unaryExpression
76                 | simpleUnaryExpression
77
78 simpleUnaryExpression ::= LNOT unaryExpression
79                       | LPAREN basicType RPAREN unaryExpression
80                       | LPAREN referenceType RPAREN simpleUnaryExpression
81                       | postfixExpression
82
83 postfixExpression ::= primary { selector } { DEC }
84
85 selector ::= DOT qualifiedIdentifier [ arguments ]
86           | LBRACK expression RBRACK
87
```

```

88 primary ::= parExpression
89           | NEW creator
90           | THIS [ arguments ]
91           | SUPER ( arguments | DOT IDENTIFIER [ arguments ] )
92           | qualifiedIdentifier [ arguments ]
93           | literal
94
95 creator ::= ( basicType | qualifiedIdentifier )
96             ( arguments
97               | LBRACK RBRACK { LBRACK RBRACK } [ arrayInitializer ]
98               | newArrayDeclarator
99             )
100
101 newArrayDeclarator ::= LBRACK expression RBRACK { LBRACK expression RBRACK } { LBRACK RBRACK }
102
103 literal ::= CHAR_LITERAL | FALSE | INT_LITERAL | NULL | STRING_LITERAL | TRUE

```

### 3 Semantics

```

1  JArrayExpression:
2  - The thing indexed must be an array
3  - The index must be an integer
4
5  JArrayInitializer:
6  - A non-array object must not be initialized with the array sequence {...}
7  - Each initializer must have the same type as the component type
8
9  JAssignment:
10 - JAssignOp:
11   - lhs must be legal
12   - lhs and rhs must have the same type
13 - JPlusAssignOp:
14   - lhs must be legal
15   - lhs and rhs must be integers (addition) or lhs must be a string (concatenation)
16
17 JBinaryExpression:
18 - JMultiplyOp, JSubtractOp
19   - lhs and rhs must be integers
20 - JPlusOp
21   - lhs and rhs must be integers (addition) or one of them must be a string (concatenation)
22
23 JBooleanBinaryExpression:
24 - JEqualOp:
25   - lhs and rhs must have the same type
26 - JLogicalAndOp:
27   - lhs and rhs must be booleans
28
29 JCastOp:
30 - Source type must be compatible with the target type
31
32 JClassDeclaration:
33 - Super type must be accessible from the base type
34 - Super type must not be final

```

35 - A non-abstract class must not declare abstract methods  
36  
37 JComparisonExpression:  
38 - lhs and rhs must be integers  
39  
40 JCompilationUnit:  
41 - Imports must be valid  
42  
43 JConstructorDeclaration:  
44 - A constructor must not be static or abstract  
45 - Signature must not exist already  
46  
47 JFieldDeclaration:  
48 - A field must not be abstract  
49 - Name must not exist already  
50  
51 JFieldSelection:  
52 - The target must be a reference type  
53 - The field must be declared  
54 - The field must be accessible  
55 - A non-static field must not be referenced from a static context  
56 - A final field must not be assigned a value  
57  
58 JIfStatement:  
59 - The condition must be a boolean  
60  
61 JInstanceOfOp:  
62 - lhs and rhs must be reference types and assignable from one to the other  
63  
64 JMessageExpression:  
65 - The target must be a reference type  
66 - The message must exist  
67 - The message must be accessible  
68 - A non-static message must not be referenced from a static context  
69  
70 JMethodDeclaration:  
71 - An abstract method cannot have a body  
72 - A method without body must be abstract  
73 - A private method cannot be abstract  
74 - A static method cannot be abstract  
75 - Signature must not exist already  
76 - A non-void method must have a return statement  
77  
78 JNewArrayOp:  
79 - Dimensions must be integers  
80  
81 JNewOp:  
82 - The constructor being invoked must not instantiate an abstract type  
83 - The constructor being invoked must exist  
84  
85 JReturnStatement:  
86 - Must not return a value from a constructor  
87 - Must not return a value from a void method  
88 - The type of return value in a non-void method must match return type of the method

89 - A non-void method must have a return value  
90  
91 JSuperConstruction:  
92 - super(...) must be the first statement in the constructor's body  
93 - A super constructor with the given argument types must exist  
94  
95 JThisConstruction:  
96 - this(...) must be the first statement in the constructor's body  
97 - A constructor with the given argument types must exist  
98  
99 JVariable:  
100 - The variable name must exist  
101 - The variable must be initialized  
102 - The variable must be a valid lhs to =  
103  
104 JVariableDeclaration:  
105 - The variable must not shadow another local variable  
106  
107 JUnaryExpression:  
108 - JLogicalNotOp:  
109 - The operand must be a boolean  
110 - JNegateOp, JUnaryPlusOp:  
111 - The operand must be an integer  
112 - JPostDecrementOp, JPreIncrementOp:  
113 - The operand must have an LValue  
114 - The operand must be an integer  
115  
116 JWhileStatement:  
117 - The condition must be a boolean