CHAPTER 18

Syntax

THIS chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters is much better for exposition, but it is not ideally suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *x* | *y* means one of either *x* or *y*.

18.1 The Grammar of the Java Programming Language

Identifier:
    IDENTIFIER

QualifiedIdentifier:
    Identifier { . Identifier }

Literal:
    IntegerLiteral
    FloatingPointLiteral
    CharacterLiteral
    StringLiteral
    BooleanLiteral
    NullLiteral

Expression:
    Expression1 [AssignmentOperator Expression1]]

AssignmentOperator:
= += -= *= /= &= |= ^= %= <<= >>= >>>=

Type:
Identifier { . *Identifier* } *BracketsOpt*
BasicType

StatementExpression:
Expression

ConstantExpression:
Expression

Expression1:
Expression2 [*Expression1Rest*]

Expression1Rest:
[? *Expression* : *Expression1*]

Expression2 :
Expression3 [*Expression2Rest*]

Expression2Rest:
{*Infixop Expression3*}
Expression3 `instanceof` *Type*

Infixop:
|| && | ^ & == !=

```
<
>
<=
>=
<<
>>
>>>
+
-
*
/
%
```

Expression3:

PrefixOp Expression3
`(` *Expr* | *Type* `)` *Expression3*
Primary {Selector} {PostfixOp}

Primary:

`(` *Expression* `)`
`this` *[Arguments]*
`super` *SuperSuffix*
Literal
`new` *Creator*
Identifier { `.` *Identifier* }[*IdentifierSuffix]*
BasicType BracketsOpt `.class`
`void.class`

IdentifierSuffix:

`[` (`]` *BracketsOpt* `.` `class` | *Expression* `]`)
Arguments
`.` (`class` | `this` | `super` *Arguments* | `new` *InnerCreator*)

PrefixOp:

```
++
--
!
~
+
-
```

PostfixOp:

```
++
--
```

Selector:
`.` *Identifier [Arguments]*
`.` `this`
`.` `super` *SuperSuffix*
`.` `new` *InnerCreator*
`[` *Expression* `]`

SuperSuffix:
Arguments
`.` *Identifier [Arguments]*

BasicType:
`byte`
`short`
`char`
`int`
`long`
`float`
`double`
`boolean`

ArgumentsOpt:
[Arguments]

Arguments:
`(` *[Expression { , Expression }]* `)`

BracketsOpt:
{`[]`*}*

Creator:
QualifiedIdentifier (ArrayCreatorRest | ClassCreatorRest)

InnerCreator:
Identifier ClassCreatorRest

ArrayCreatorRest:
`[` *( ]* *BracketsOpt ArrayInitializer | Expression* `]` *{*`[` *Expression* `]`*}* *BracketsOpt)*

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
`if` *ParExpression Statement [*`else` *Statement]*
`for` `(` *ForInitOpt* `;` *[Expression]* `;` *ForUpdateOpt* `)` *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]*
`;`
ExpressionStatement
Identifier `:` *Statement*

Catches:
CatchClause {CatchClause}

CatchClause:
`catch` `(` *FormalParameter* `)` *Block*

SwitchBlockStatementGroups:
{ SwitchBlockStatementGroup }

SwitchBlockStatementGroup:
SwitchLabel BlockStatements

SwitchLabel:
`case` *ConstantExpression* `:`
`default :`

MoreStatementExpressions:
{ `,` *StatementExpression }*

ForInit:
StatementExpression MoreStatementExpressions
[`final`*] Type VariableDeclarators*

ForUpdate:
StatementExpression MoreStatementExpressions

ModifiersOpt:
{ Modifier }

Modifier:
`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:
BracketsOpt [`=` *VariableInitializer*]

ConstantDeclaratorRest:
BracketsOpt `=` *VariableInitializer*

VariableDeclaratorId:
Identifier BracketsOpt

CompilationUnit:
[`package` *QualifiedIdentifier* `;`] {*ImportDeclaration*} {*TypeDeclaration*}

ImportDeclaration:
`import` *Identifier* { `.` *Identifier* } [`.` `*`] `;`

TypeDeclaration:
ClassOrInterfaceDeclaration
`;`

ClassOrInterfaceDeclaration:
ModifiersOpt (*ClassDeclaration* | *InterfaceDeclaration*)

ClassDeclaration:
`class` *Identifier* [`extends` *Type*] [`implements` *TypeList*] *ClassBody*

InterfaceDeclaration:
`interface` *Identifier* [`extends` *TypeList*] *InterfaceBody*

TypeList:
Type { `,` *Type*}

ClassBody:
`{` {*ClassBodyDeclaration*} `}`

InterfaceBody:
`{` {*InterfaceBodyDeclaration*} `}`

ClassBodyDeclaration:
`;`
[`static`] *Block*
ModifiersOpt MemberDecl

MemberDecl:
MethodOrFieldDecl
`void` *Identifier MethodDeclaratorRest*
Identifier ConstructorDeclaratorRest
ClassOrInterfaceDeclaration

MethodOrFieldDecl:
Type Identifier MethodOrFieldRest

MethodOrFieldRest:
VariableDeclaratorRest
MethodDeclaratorRest

InterfaceBodyDeclaration:
;
ModifiersOpt InterfaceMemberDecl

InterfaceMemberDecl:
InterfaceMethodOrFieldDecl
`void` *Identifier VoidInterfaceMethodDeclaratorRest*
ClassOrInterfaceDeclaration

InterfaceMethodOrFieldDecl:
Type Identifier InterfaceMethodOrFieldRest

InterfaceMethodOrFieldRest:
ConstantDeclaratorsRest ;
InterfaceMethodDeclaratorRest

MethodDeclaratorRest:
FormalParameters BracketsOpt [`throws` *QualifiedIdentifierList] (*
MethodBody | ; *)*

VoidMethodDeclaratorRest:
FormalParameters [`throws` *QualifiedIdentifierList] (MethodBody* | ; *)*

InterfaceMethodDeclaratorRest:
FormalParameters BracketsOpt [`throws` *QualifiedIdentifierList]* ;

VoidInterfaceMethodDeclaratorRest:
FormalParameters [`throws` *QualifiedIdentifierList]* ;

ConstructorDeclaratorRest:
FormalParameters [`throws` *QualifiedIdentifierList] MethodBody*

QualifiedIdentifierList:
QualifiedIdentifier { , *QualifiedIdentifier}*

FormalParameters:
(*[FormalParameter {* , *FormalParameter}]*)

FormalParameter:
[`final`*] Type VariableDeclaratorId*

MethodBody:
Block