CHAPTER 18

Syntax

Is there grammar in a title. There is grammar in a title. Thank you.
—Gertrude Stein, *Arthur a Grammar*, in *How to Write* (1931)

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

TypeArguments:
    < *TypeArgument {* , *TypeArgument}* >

TypeArgument:
    Type
    ? *[(* `extends` | `super` *) Type]*

RawType:
    Identifier { . *Identifier } BracketsOpt*

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

IdentifierSuffix:
    `[` *(* `]` *BracketsOpt* `. 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`

ArgumentsOpt:
    [Arguments]

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

BracketsOpt:
    {`[]`*}*

Creator:
    [NonWildcardTypeArguments] CreatedName (ArrayCreatorRest | ClassCreatorRest)

CreatedName:
    Identifier [NonWildcardTypeArguments] {. Identifier [NonWildcardTypeArguments]}

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
    `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]*
    `;`
    ExpressionStatement
    Identifier `:` *Statement*

Catches:
    CatchClause {CatchClause}

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

SwitchBlockStatementGroups:
    { SwitchBlockStatementGroup }

SwitchBlockStatementGroup:
    SwitchLabel BlockStatements

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

MoreStatementExpressions:
    { `,` *StatementExpression }*

ForControl:
    `;` *[Expression]* `;` *ForUpdateOpt*
    StatementExpression MoreStatementExpressions `;` *[Expression]* `;`

ForUpdateOpt
    [`final`*] [Annotations] Type Identifier ForControlRest*

ForControlRest:
    VariableDeclaratorsRest `;` *[Expression]* `;` *ForUpdateOpt*
    `:` *Expression*

ForUpdate:
    StatementExpression MoreStatementExpressions

ModifiersOpt:
    { Modifier }

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:
  BracketsOpt [`=` *VariableInitializer]*

ConstantDeclaratorRest:
  BracketsOpt `=` *VariableInitializer*

VariableDeclaratorId:
  Identifier BracketsOpt

CompilationUnit:
  [Annotations$_{opt}$ `package` *QualifiedIdentifier* `;` *] {ImportDeclaration}*
{TypeDeclaration}

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

TypeDeclaration:
  ClassOrInterfaceDeclaration
  `;`

ClassOrInterfaceDeclaration:
  ModifiersOpt (ClassDeclaration | InterfaceDeclaration)

ClassDeclaration:
  NormalClassDeclaration
  EnumDeclaration

NormalClassDeclaration:
  `class` *Identifier TypeParameters*$_{opt}$ *[*`extends` *Type] [*`implements`
TypeList] ClassBody

TypeParameters:
  `<` *TypeParameter {*`,` *TypeParameter}* `>`

TypeParameter:
  Identifier [`extends` *Bound]*

Bound:
  Type {`&` *Type}*

EnumDeclaration:
    ClassModifiers$_{opt}$ `enum` *Identifier [*`implements` *TypeList] EnumBody*

EnumBody:
    { EnumConstants$_{opt}$,$_{opt}$ EnumBodyDeclarations$_{opt}$ }

EnumConstants:
    EnumConstant
    EnumConstants , EnumConstant

EnumConstant:
    Annotations Identifier Arguments$_{opt}$ ClassBody$_{opt}$

Arguments:
    (ArgumentList$_{opt}$)

EnumBodyDeclarations:
    ; ClassBodyDeclarations$_{opt}$

InterfaceDeclaration:
    NormalInterfaceDeclaration
    AnnotationTypeDeclaration

NormalInterfaceDeclaration:
    `interface` *Identifier TypeParameters$_{opt}$ [*`extends` *TypeList]*
InterfaceBody

TypeList:
    Type {   ,   Type}

AnnotationTypeDeclaration:
    InterfaceModifiers$_{opt}$ @ interface Identifier AnnotationTypeBody

AnnotationTypeBody:
    { AnnotationTypeElementDeclarations$_{opt}$ }

AnnotationTypeElementDeclarations:
    AnnotationTypeElementDeclaration
    AnnotationTypeElementDeclarations AnnotationTypeElementDeclaration

AnnotationTypeElementDeclaration:
AbstractMethodModifiers$_{opt}$ *Type Identifier ( ) DefaultValue*$_{opt}$ *;*
ConstantDeclaration
ClassDeclaration
InterfaceDeclaration
EnumDeclaration
AnnotationTypeDeclaration
;

DefaultValue:
`default` *ElementValue*

ClassBody:
{ *{ClassBodyDeclaration}* }

InterfaceBody:
{ *{InterfaceBodyDeclaration}* }

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

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

GenericMethodOrConstructorDecl:
TypeParameters GenericMethodOrConstructorRest

GenericMethodOrConstructorRest:
Type Identifier MethodDeclaratorRest
Identifier ConstructorDeclaratorRest

MethodOrFieldDecl:
Type Identifier MethodOrFieldRest

MethodOrFieldRest:
VariableDeclaratorRest
MethodDeclaratorRest

InterfaceBodyDeclaration:
  `;`
  ModifiersOpt InterfaceMemberDecl

InterfaceMemberDecl:
  InterfaceMethodOrFieldDecl
  InterfaceGenericMethodDecl
  `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]* `;`

InterfaceGenericMethodDecl:
  TypeParameters Type 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