A Study of Syntactic Rule Usage in Java

Dong Qiu* Bixin Li* Earl T. Barr† Zhendong Su‡
* School of Computer Science and Engineering, Southeast University, China
† Department of Computer Science, University College London, UK
‡ Department of Computer Science, University of California, Davis, USA
Email: {dongqiu, bx.li}@seu.edu.cn, e.barr@ucl.ac.uk, su@ucdavis.edu

Appendix

This appendix describes the syntactic grammars for the Java programming language in this study. We employ the following notations to describe the grammars.

Terminals are written in **bold**; non-terminals are written in the form of Java (mixed-case) identifiers; [x] denotes zero or one occurrence of x; {x} denotes zero or more occurrences of x. (x | y) indicates that either x or y appears. Our grammar refers to the grammar defined in the Eclipse JDT compiler.

TABLE I: Java Syntactic Rules

Syntax Name	Rules		Since	Update
CompilationUnit	CompilationUnit	→ [PackageDeclaration]{ ImportDeclaration }{ BodyDeclaration }	JLS1	JLS3
PackageDeclaration	PackageDeclaration	→ { Annotation } **package** Name **;**	JLS1	JLS3
ImportDeclaration	ImportDeclaration	→ **import** [**static**] Name [. *] **;**	JLS1	JLS3
ClassDeclaration	BodyDeclaration	→ { ExtendedModifier } **class SimpleName** [**<** TypeParameter { , TypeParameter } **>**] [**extends** Type] [**implements** Type { , Type }] **{** { BodyDeclaration | ; } **}**	JLS1	JLS3
InterfaceDeclaration	BodyDeclaration	→ { ExtendedModifier } **interface SimpleName** [**<** TypeParameter , TypeParameter **>**] [**extends** Type { , Type }] **{** { BodyDeclaration | ; } **}**	JLS1	JLS3
EnumDeclaration	BodyDeclaration	→ { ExtendedModifier } **enum SimpleName** [**implements** Type { , Type }] **{** [EnumConstantDeclaration { , EnumConstantDeclaration }] [,] [; { BodyDeclaration | ; }] **}**	JLS3	
EnumConstantDeclaration	BodyDeclaration	→ { ExtendedModifier } **SimpleName** [**(** [Expression { , Expression }])] [AnonymousClassDeclaration]	JLS3	
MethodDeclaration	BodyDeclaration	→ { ExtendedModifier } [**<** TypeParameter { , TypeParameter } **>**] (Type | **void**) **SimpleName (**[FormalParameter { , FormalParameter }]) { **[ ]** } [**throws** Name { , Name }] (Block | ;)	JLS1	JLS3
ConstructorDeclaration	BodyDeclaration	→ { ExtendedModifier } [**<** TypeParameter { , TypeParameter } **>**] **SimpleName** ([FormalParameter { , FormalParameter }] **)** [**throws** Name { , Name }] Block	JLS1	JLS3
FieldDeclaration	BodyDeclaration	→ { ExtendedModifier } Type VariableDeclarationFragment { , VariableDeclarationFragment } **;**	JLS1	
Initializer	BodyDeclaration	→ [**static**] Block	JLS1	
AnnotationTypeDeclaration	BodyDeclaration	→ { ExtendedModifier } @ **interface SimpleName {** { BodyDeclaration | ; } **}**	JLS3	
AnnotationTypeMemberDeclaration	BodyDeclaration	→ { ExtendedModifier } Type **SimpleName** ( ) [**default** Expression] **;**	JLS3	
AssertStatement	Statement	→ **assert** Expression [**:** Expression] **;**	JLS2	
Block	Statement	→ **{** { Statement } **}**	JLS1	
BreakStatement	Statement	→ **break** [**SimpleName**] **;**	JLS1	
ConstructorInvocation	Statement	→ [**<** Type { , Type } **>**] **this (** [Expression { , Expression }]) **;**	JLS1	JLS3
ContinueStatement	Statement	→ **continue** [**SimpleName**] **;**	JLS1	
DoStatement	Statement	→ **do** Statement **while** (Expression **)** **;**	JLS1	
EmptyStatement	Statement	→ **;**	JLS1	
EnhancedForStatement	Statement	→ **for** (FormalParameter **:** Expression **)** Statement	JLS3	
ExpressionStatement	Statement	→ Expression **;**	JLS1	
ForStatement	Statement	→ **for** ([Expression { , Expression }]**;** [Expression] **;** [Expression { , Expression }]) Statement	JLS1	
IfStatement	Statement	→ **if (** Expression **)** Statement [**else** Statement]	JLS1	
LabeledStatement	Statement	→ **SimpleName :** Statement	JLS1	
ReturnStatement	Statement	→ **return** [Expression] **;**	JLS1	
SuperConstructorInvocation	Statement	→ [Expression**.**] [**<** Type { , Type } **>**] **super (** [Expression { , Expression}] **)** **;**	JLS1	JLS3
SwitchCase	Statement	→ **case** Expression **:** | **default :**	JLS1	
SwitchStatement	Statement	→ **switch (** Expression **) {** { Statement } **}**	JLS1	
SynchronizedStatement	Statement	→ **synchronized (** Expression) Block	JLS1	
ThrowStatement	Statement	→ **throw** Expression **;**	JLS1	
TryStatement	Statement	→ **try** [**(** { VariableDeclarationExpression } **)**] Block [{ CatchClause }] [**finally Block**]	JLS1	JLS4
TypeDeclarationStatement	Statement	→ ClassDeclaration | InterfaceDeclaration | EnumDeclaration	JLS2	
VariableDeclarationStatement	Statement	→ { ExtendedModifier } Type VariableDeclarationFragment { , VariableDeclarationFragment} **;**	JLS1	JLS3
WhileStatement	Statement	→ **while (** Expression **)** Statement	JLS1	
NormalAnnotation	Annotation	→ @ Name **(** [MemberValuePair { , MemberValuePair }])	JLS3	
MarkerAnnotation	Annotation	→ @ Name	JLS3	
SingleMemberAnnotation	Annotation	→ @ Name **(** Expression)	JLS3	
ArrayAccess	Expression	→ Expression **[** Expression **]**	JLS1	
ArrayCreation	Expression	→ **new** ArrayType { **[** Expression **]** } { **[ ]** } [ArrayInitializer]	JLS1	JLS3
ArrayInitializer	Expression	→ **{** [Expression , Expression [,]] **}**	JLS1	
Assignment	Expression	→ Expression Operator Expression	JLS1	
BooleanLiteral	Expression	→ **true** | **false**	JLS1	
CastExpression	Expression	→ (Type **)** Expression	JLS1	
ClassInstanceCreation	Expression	→ [Expression.] **new** [**<** Type { , Type } **>**] Type **(** [Expression { , Expression }]) [AnonymousClassDeclaration]	JLS1	JLS3
ConditionalExpression	Expression	→ Expression **?** Expression **:** Expression	JLS1	
ExpName	Expression	→ Name	JLS1	
FieldAccess	Expression	→ Expression**. SimpleName**	JLS1	
InfixExpression	Expression	→ Expression Operator Expression { Operator Expression }	JLS1	
InstanceofExpression	Expression	→ Expression **instanceof** Type	JLS1	
MethodInvocation	Expression	→ [Expression **.**] [**<** Type { , Type } **>**] **SimpleName (** [Expression { , Expression }])	JLS1	JLS3
ParenthesizedExpression	Expression	→ (Expression **)**	JLS1	
PostfixExpression	Expression	→ Expression Operator	JLS1	
PrefixExpression	Expression	→ Operator Expression	JLS1	
SuperFieldAccess	Expression	→ [Name**.**] **super. SimpleName**	JLS1	
SuperMethodInvocation	Expression	→ [Name**.**] **super.** [**<** Type , Type **>**] **SimpleName (** [Expression { , Expression }])	JLS1	JLS3
ThisExpression	Expression	→ [Name**.**] **this**	JLS1	
TypeLiteral	Expression	→ (Type | **void**) **. class**	JLS1	
VariableDeclarationExpression	Expression	→ { ExtendedModifier } Type VariableDeclarationFragment { , VariableDeclarationFragment }	JLS1	JLS3
ArrayType	Type	→ Type **[ ]**	JLS1	
ParameterizedType	Type	→ Type **<** Type { , Type } **>**	JLS3	
PrimitiveType	Type	→ **boolean** | **byte** | **char** | **double** | **float** | **int** | **long** | **short** | **void**	JLS1	
QualifiedType	Type	→ Type **. SimpleName**	JLS3	
SimpleType	Type	→ Name	JLS1	
UnionType	Type	→ Type | Type { | Type }	JLS4	
WildcardType	Type	→ **?** [(**extends** | **super**) Type]	JLS3	
TypeParameter	TypeParameter	→ **SimpleName** [**extends** Type { **&** Type }]	JLS3	
FormalParameter	FormalParameter	→ { ExtendedModifier } Type [**...**] **SimpleName** { **[]** } [**=** Expression]	JLS1	
VariableDeclarationFragment	VariableDeclarationFragment	→ **SimpleName** { **[]** } [**=** Expression]	JLS1	JLS3
AnonymousClassDeclaration	AnonymousClassDeclaration	→ { BodyDeclaration }	JLS2	
QualifiedName	Name	→ Name **. SimpleName**	JLS1	
CatchClause	CatchClause	→ **catch** (FormalParameter) Block	JLS1	
Modifier	ExtendedModifier	→ **abstract** | **final** | **native** | **private** | **protected** | **public** | **static** | **strictfp** | **synchronized** | **transient** | **volatile**	JLS1	
ExAnnotation	ExtendedModifier	→ Annotation	JLS3	
Operator	Operator	→ = | += | -= | *= | /= | &= | |= | ˆ= | %= | <<= | >>= | >>>= | ++ | – | + | - | ∼ | ! | * | / | % | << | >> | >>> | < | > | <= | >= | == | != | ˆ | & | | | && | ||	JLS1	