Java Syntax Specification

Programs
1. <goal> ::= <compilation unit>
2. <compilation unit> ::= <package declaration>? <import declarations>? <type declarations>?

Declarations

3. <package declaration> ::= **package** <package name> **;**
4. <import declarations> ::= <import declaration>
            | <import declarations> <import declaration>
5. <import declaration> ::= <single type import declaration>
            | <type import on demand declaration>
6. <single type import declaration> ::= **import** <type name> **;**
7. <type import on demand declaration> ::= **import** <package name> **. * ;**
8. <type declarations> ::= <type declaration> | <type declarations> <type declaration>
9. <type declaration> ::= <class declaration> | <interface declaration> | **;**
10. <class declaration> ::= <class modifiers>? **class** <identifier> <super>?
            <interfaces>? <class body>
11. <class modifiers> ::= <class modifier> | <class modifiers> <class modifier>
12. <class modifier> ::= **public** | **abstract** | **final**
13. <super> ::= **extends** <class type>
14. <interfaces> ::= **implements** <interface type list>
15. <interface type list> ::= <interface type> | <interface type list> **,** <interface type>
16. <class body> ::= **{** <class body declarations>? **}**
17. <class body declarations> ::= <class body declaration>
            | <class body declarations> <class body declaration>
18. <class body declaration> ::= <class member declaration>
            | <static initializer> | <constructor declaration>
19. <class member declaration> ::= <field declaration> | <method declaration>
20. <static initializer> ::= **static** <block>
21. <constructor declaration> ::= <constructor modifiers>? <constructor declarator>
            <throws>? <constructor body>
22. <constructor modifiers> ::= <constructor modifier>
            | <constructor modifiers> <constructor modifier>
23. <constructor modifier> ::= **public** | **protected** | **private**
24. <constructor declarator> ::= <simple type name> **(** <formal parameter list>? **)**
25. <formal parameter list> ::= <formal parameter>
            | <formal parameter list> **,** <formal parameter>
26. <formal parameter> ::= <type> <variable declarator id>
27. <throws> ::= **throws** <class type list>
28. <class type list> ::= <class type> | <class type list> **,** <class type>
29. <constructor body> ::= **{** <explicit constructor invocation>? <block statements>? **}**
30. <explicit constructor invocation>::= **this (** <argument list>? **)**
            | **super (** <argument list>? **)**
31. <field declaration> ::= <field modifiers>? <type> <variable declarators> **;**
32. <field modifiers> ::= <field modifier>
            | <field modifiers> <field modifier>
33. <field modifier> ::= **public** | **protected** | **private** | **static** | **final** | **transient** | **volatile**
34. <variable declarators> ::= <variable declarator>
            | <variable declarators> **,** <variable declarator>
35. <variable declarator> ::= <variable declarator id>
            | <variable declarator id> **=** <variable initializer>
36. <variable declarator id> ::= <identifier> | <variable declarator id> **[ ]**
37. <variable initializer> ::= <expression> | <array initializer>

| <while statement no short if> | <for statement no short if>

77.   <statement without trailing substatement> ::= <block> | <empty statement>
  | <expression statement> | <switch statement> | <do statement>
  | <break statement> | <continue statement> | <return statement>
  | <synchronized statement> | <throws statements> | <try statement>
78.   <empty statement> ::= **;**
79.   <labeled statement> ::= <identifier> **:** <statement>
80.   <labeled statement no short if> ::= <identifier> **:** <statement no short if>
81.   <expression statement> ::= <statement expression> **;**
82.   <statement expression> ::= <assignment> | <preincrement expression>
  | <postincrement expression> | <predecrement expression>
  | <postdecrement expression> | <method invocation>
  | <class instance creation expression>
83.   <if then statement>::= **if (** <expression> **)** <statement>
84.   <if then else statement>::= **if (** <expression> **)** <statement no short if> **else** <statement>
85.   <if then else statement no short if> ::= **if (** <expression> **)** <statement no short if>
  else <statement no short if>
86.   <switch statement> ::= **switch (** <expression> **)** <switch block>
87.   <switch block> ::= **{** <switch block statement groups>? <switch labels>? **}**
88.   <switch block statement groups> ::= <switch block statement group>
  | <switch block statement groups> <switch block statement group>
89.   <switch block statement group> ::= <switch labels> <block statements>
90.   <switch labels> ::= <switch label> | <switch labels> <switch label>
91.   <switch label> ::= **case** <constant expression> **:** | **default :**
92.   <while statement> ::= **while (** <expression> **)** <statement>
93.   <while statement no short if> ::= **while (** <expression> **)** <statement no short if>
94.   <do statement> ::= **do** <statement> **while (** <expression> **) ;**
95.   <for statement> ::= **for (** <for init>? **;** <expression>? **;** <for update>? **)** <statement>
96.   <for statement no short if> ::= **for (** <for init>? **;** <expression>? **;** <for update>? **)**
  <statement no short if>
97.   <for init> ::= <statement expression list> | <local variable declaration>
98.   <for update> ::= <statement expression list>
99.   <statement expression list> ::= <statement expression>
  | <statement expression list> **,** <statement expression>
100. <break statement> ::= **break** <identifier>? **;**
102. <continue statement> ::= **continue** <identifier>? **;**
103. <return statement> ::= **return** <expression>? **;**
104. <throws statement> ::= **throw** <expression> **;**
105. <synchronized statement> ::= **synchronized (** <expression> **)** <block>
106. <try statement> ::= **try** <block> <catches> | **try** <block> <catches>? <finally>
107. <catches> ::= <catch clause> | <catches> <catch clause>
108. <catch clause> ::= **catch (** <formal parameter> **)** <block>
109. <finally > ::= **finally** <block>

Expressions

110. <constant expression> ::= <expression>
111. <expression> ::= <assignment expression>
112. <assignment expression> ::= <conditional expression> | <assignment>
113. <assignment> ::= <left hand side> <assignment operator> <assignment expression>
114. <left hand side> ::= <expression name> | <field access> | <array access>
115. <assignment operator> ::= **=** | ***=** | **/=** | **%=** | **+=** | **-=** | **<<=** | **>>=** | **>>>=** | **&=** | **^=** | **|=**
116. <conditional expression> ::= <conditional or expression>
  | <conditional or expression> **?** <expression> **:** <conditional expression>
117. <conditional or expression> ::= <conditional and expression>
  | <conditional or expression> **||** <conditional and expression>
118. <conditional and expression> ::= <inclusive or expression>

        | <conditional and expression> **&&** <inclusive or expression>

119.   <inclusive or expression> ::= <exclusive or expression>
        | <inclusive or expression> **|** <exclusive or expression>

120.   <exclusive or expression> ::= <and expression>
        | <exclusive or expression> **^** <and expression>

121.   <and expression> ::= <equality expression>
        | <and expression> **&** <equality expression>

122.   <equality expression> ::= <relational expression>
        | <equality expression> **==** <relational expression>
        | <equality expression> **!=** <relational expression>

123.   <relational expression> ::= <shift expression>
        | <relational expression> **<<**<shift expression>
        | <relational expression> **>** <shift expression>
        | <relational expression> **<=** <shift expression>
        | <relational expression> **>=** <shift expression>
        | <relational expression> **instanceof** <reference type>

124.   <shift expression> ::= <additive expression>
        | <shift expression> **<<** <additive expression>
        | <shift expression> **>>** <additive expression>
        | <shift expression> **>>>** <additive expression>

125.   <additive expression> ::= <multiplicative expression>
        | <additive expression> + <multiplicative expression>
        | <additive expression> - <multiplicative expression>

126.   <multiplicative expression> ::= <unary expression>
        | <multiplicative expression> ***** <unary expression>
        | <multiplicative expression> **/** <unary expression>
        | <multiplicative expression> **%** <unary expression>

127.   <cast expression> ::= **(** <primitive type> **)** <unary expression>
        | **(** <reference type> **)** <unary expression not plus minus>

128.   <unary expression> ::= <preincrement expression> | <predecrement expression>
        | **+** <unary expression> | **-** <unary expression>
        | <unary expression not plus minus>

129.   <predecrement expression> ::= **--** <unary expression>

130.   <preincrement expression> ::= **++** <unary expression>

131.   <unary expression not plus minus> ::= <postfix expression> | **~** <unary expression>
        | **!** <unary expression> | <cast expression>

132.   <postdecrement expression> ::= <postfix expression> **--**

133.   <postincrement expression> ::= <postfix expression> **++**

134.   <postfix expression> ::= <primary> | <expression name>
        | <postincrement expression> | <postdecrement expression>

135.   <method invocation> ::= <method name> **(** <argument list>? **)**
        | <primary> **.** <identifier> **(** <argument list>? **)**
        | **super .** <identifier> **(** <argument list>? **)**

136.   <field access> ::= <primary> **.** <identifier> | **super .** <identifier>

137.   <primary> ::= <primary no new array> | <array creation expression>

138.   <primary no new array> ::= <literal> | **this** | **(** <expression> **)**
        | <class instance creation expression> | <field access>
        | <method invocation> | <array access>

139.   <class instance creation expression> ::= **new** <class type> **(** <argument list>? **)**

140.   <argument list> ::= <expression> | <argument list> **,** <expression>

141.   <array creation expression> ::= **new** <primitive type> <dim exprs> <dims>?
        | **new** <class or interface type> <dim exprs> <dims>?

142.   <dim exprs> ::= <dim expr> | <dim exprs> <dim expr>

143.   <dim expr> ::= **[** <expression> **]**

144.   <dims> ::= **[ ]** | <dims> **[ ]**

145.   <array access> ::= <expression name> **[** <expression> **]** | <primary no new array> **[** <expression>**]**

Tokens

146.   <package name> ::= <identifier> | <package name> **.** <identifier>
147.   <type name> ::= <identifier> | <package name> **.** <identifier>
148.   <simple type name>> ::= <identifier>
149.   <expression name> ::= <identifier> | <ambiguous name> **.** <identifier>
150.   <method name> ::= <identifier> | <ambiguous name>**.** <identifier>
151.   <ambiguous name>::= <identifier> | <ambiguous name>**.** <identifier>
152.   <literal> ::= <integer literal> | <floating-point literal> | <boolean literal>
           | <character literal> | <string literal> | <null literal>
153.   <integer literal> ::= <decimal integer literal> | <hex integer literal> | <octal integer literal>
154.   <decimal integer literal> ::= <decimal numeral> <integer type suffix>?
155.   <hex integer literal> ::= <hex numeral> <integer type suffix>?
156**.**   <octal integer literal> ::= <octal numeral> <integer type suffix>?
157.   <integer type suffix> ::= � | **L**
158.   <decimal numeral> ::= 0 | <non zero digit> <digits>?
159.   <digits> ::= <digit> | <digits> <digit>
160.   <digit> ::= 0 | <non zero digit>
161.   <non zero digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
162.   <hex numeral> ::= 0 x <hex digit> | 0 X <hex digit> | <hex numeral> <hex digit>
163.   <hex digit> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F
164.   <octal numeral> ::= 0 <octal digit> | <octal numeral> <octal digit>
165.   <octal digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
166.   <floating-point literal> ::= <digits> **.** <digits>? <exponent part>? <float type suffix>?
           | **.** <digits> <exponent part>? <float type suffix>?
           | <digits> <exponent part> <float type suffix>?
           | <digits> <exponent part>? <float type suffix>
167.   <exponent part> ::= <exponent indicator> <signed integer>
168.   <exponent indicator> ::= e | E
169.   <signed integer> ::= <sign>? <digits>
170.   <sign> ::= + | �
171.   <float type suffix> ::= f | F | d | D
172.   <boolean literal> ::= **true** | **false**
173.   <character literal> ::= ' <single character> ' | ' <escape sequence> '
174.   <single character> ::= <input character> except ' and \
175.   <string literal> ::= " <string characters>?"
176.   <string characters> ::= <string character> | <string characters> <string character>
177.   <string character> ::= <input character> except " and \ | <escape character>
178.   <null literal> ::= **null**
179.   <keyword> ::=

abstract	| boolean	| break	| byte	| case	| catch
| char	| class	| const	| continue	| default	| do
| double	| else	| extends	| final	| finally	| float
| for	| goto	| if	| implements	| import	| instanceof
| int	| interface	| long	| native	| new	| package
| private	| protected	| public	| return	| short	| static
| super	| switch	| synchronized	| this	| throw	| throws
| transient	| try	| void	| volatile	| while	

The character set for Java is *Unicode*, a 16-bit character set. This is the set denoted by <input character>. Unicode effectively contains the familiar 7-bit ASCII characters as a subset, and includes �escape code� designations of the form \udddd (where each d is from <hex digit>). In the extended BNF for Java the optional appearance of X is written X?, and the symbols { and } are used only as terminals (i.e., the �star� operation is not used in this BNF).

The syntax category <identifier> consists of strings that must start with a letter � including underscore (_) and dollar sign ($) �

followed by any number of letters and digits. Characters of numerous international languages are recognized as �letters� in Java. A Java *letter* is a character for which the method Character.isJavaLetter returns **true**. A Java letter-or-digit is a character for which the method Character.isJaveLetterOrDigit returns **true**. Also, <identifier> includes none of the keywords given in line 179 � these are *reserved words* in Java.

Note that '*', '{', and '}' are all terminal symbols. This BNF definition does not address such pragmatic issues as comment conventions and the use of �white space� to delimit tokens. This BNF also does not express numerous �context-sensitive� restrictions on syntax. For instance, type use of identifiers must be consistent with the required declarations, there are size limitations on numerical literals, etc.