Java Syntax Specification

http://www.daimi.au.dk/dRegAut/JavaBNF.html

Programs

> *<compilation unit>* ::= *<package declaration>*? *<import declarations>*? *<type declarations>*?

Declarations

> *<package declaration>* ::= **package** *<package name>* **;**
>
> *<import declarations>* ::= *<import declaration>* | *<import declarations>* *<import declaration>*
>
> *<import declaration>* ::= *<single type import declaration>* | *<type import on demand declaration>*
>
> *<single type import declaration>* ::= **import** *<type name>* **;**
>
> *<type import on demand declaration>* ::= **import** *<package name>* **. * ;**
>
> *<type declarations>* ::= *<type declaration>* | *<type declarations>* *<type declaration>*
>
> *<type declaration>* ::= *<class declaration>* | *<interface declaration>* | **;**
>
> *<class declaration>* ::= *<class modifiers>*? **class** *<identifier>* *<super>*? *<interfaces>*? *<class body>*
>
> *<class modifiers>* ::= *<class modifier>* | *<class modifiers>* *<class modifier>*
>
> *<class modifier>* ::= **public** | **abstract** | **final**
>
> *<super>* ::= **extends** *<class type>*

<*interfaces*> ::= **implements** <*interface type list*>

<*interface type list*> ::=<*interface type*> | <*interface type list*> , <*interface type*>

<*class body*> ::= { <*class body declarations*>? }

<*class body declarations*> ::= <*class body declaration*> | <*class body declarations*> <*class body declaration*>

<*class body declaration*> ::=<*class member declaration*> | <*static initializer*> | <*constructor declaration*>

<*class member declaration*> ::= <*field declaration*> | <*method declaration*>

<*static initializer*> ::= **static** <*block*>

<*constructor declaration*> ::= <*constructor modifiers*>? <*constructor declarator*> <*throws*>? <*constructor body*>

<*constructor modifiers*> ::=<*constructor modifier*> | <*constructor modifiers*> <*constructor modifier*>

<*constructor modifier*> ::= **public** | **protected** | **private**

<*constructor declarator*> ::=<*simple type name*> (<*formal parameter list*>?)

<*formal parameter list*> ::=<*formal parameter*> | <*formal parameter list*> , <*formal parameter*>

<*formal parameter*> ::= <*type*> <*variable declarator id*>

<*throws*> ::= **throws** <*class type list*>

<*class type list*> ::=<*class type*> | <*class type list*> , <*class type*>

<*constructor body*> ::= { <*explicit constructor invocation*>? <*block statements*>? }

<*explicit constructor invocation*>::= **this** (<*argument list*>?) | **super** (<*argument list*>?)

<*field declaration*> ::= <*field modifiers*>? <*type*> <*variable declarators*> ;

<*field modifiers*> ::= <*field modifier*> | <*field modifiers*> <*field modifier*>

<*field modifier*> ::= public | protected | private | static | final | transient | volatile

<*variable declarators*> ::= <*variable declarator*> | <*variable declarators*> , <*variable declarator*>

<*variable declarator*> ::= <*variable declarator id*> | <*variable declarator id*> = <*variable initializer*>

<*variable declarator id*> ::= <*identifier*> | <*variable declarator id*> [ ]

<*variable initializer*> ::= <*expression*> | <*array initializer*>

<*method declaration*> ::= <*method header*> <*method body*>

<*method header*> ::= <*method modifiers*>? <*result type*> <*method declarator*> <*throws*>?

<*result type*> ::= <*type*> | void

<*method modifiers*> ::= <*method modifier*> | <*method modifiers*> <*method modifier*>

<*method modifier*> ::= public | protected | private | static | abstract | final | synchronized | native

<*method declarator*> ::= <*identifier*> (<*formal parameter list*>?)

<*method body*> ::= <*block*> | ;

<*interface declaration*> ::= <*interface modifiers*>? interface <*identifier*> <*extends interfaces*>? <*interface body*>

<*interface modifiers*> ::= <*interface modifier*> | <*interface modifiers*> <*interface modifier*>

<*interface modifier*> ::= public | abstract

<*extends interfaces*> ::= extends <*interface type*> | <*extends interfaces*> , <*interface type*>

<*interface body*> ::= { <*interface member declarations*>? }

<*interface member declarations*> ::= <*interface member declaration*>
| <*interface member declarations*> <*interface member declaration*>

<*interface member declaration*> ::= <*constant declaration*> |
<*abstract method declaration*>

<*constant declaration*> ::= <*constant modifiers*> <*type*> <*variable
declarator*>

<*constant modifiers*> ::= public | static | final

<*abstract method declaration*>::= <*abstract method modifiers*>?
<*result type*> <*method declarator*> <*throws*>? ;

<*abstract method modifiers*> ::= <*abstract method modifier*> |
<*abstract method modifiers*> <*abstract method modifier*>

<*abstract method modifier*> ::= public | abstract

<*array initializer*> ::= { <*variable initializers*>? , ? }

<*variable initializers*> ::= <*variable initializer*> | <*variable
initializers*> , <*variable initializer*>

<*variable initializer*> ::= <*expression*> | <*array initializer*>

Types

<*type*> ::= <*primitive type*> | <*reference type*>

<*primitive type*> ::= <*numeric type*> | boolean

<*numeric type*> ::= <*integral type*> | <*floating-point type*>

<*integral type*> ::= byte | short | int | long | char

<*floating-point type*> ::= float | double

<*reference type*> ::= <*class or interface type*> | <*array type*>

<*class or interface type*> ::= <*class type*> | <*interface type*>

<*class type*> ::= <*type name*>

<*interface type*> ::= <*type name*>

<*array type*> ::= <*type*> [ ]

Blocks and Commands

<*block*> ::= { <*block statements*>? }

<*block statements*> ::= <*block statement*> | <*block statements*> <*block statement*>

<*block statement*> ::= <*local variable declaration statement*> | <*statement*>

<*local variable declaration statement*> ::= <*local variable declaration*> ;

<*local variable declaration*> ::= <*type*> <*variable declarators*>

<*statement*> ::= <*statement without trailing substatement*> | <*labeled statement*> | <*if then statement*> | <*if then else statement*> | <*while statement*> | <*for statement*>

<*statement no short if*> ::= <*statement without trailing substatement*> | <*labeled statement no short if*> | <*if then else statement no short if*> | <*while statement no short if*> | <*for statement no short if*>

<*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*>

<*empty statement*> ::= ;

<*labeled statement*> ::= <*identifier*> : <*statement*>

<*labeled statement no short if*> ::= <*identifier*> : <*statement no short if*>

<*expression statement*> ::= <*statement expression*> ;

<*statement expression*> ::= <*assignment*> | <*preincrement expression*> | <*postincrement expression*> | <*predecrement expression*> | <*postdecrement expression*> | <*method invocation*> | <*class instance creation expression*>

<*if then statement*>::= if (<*expression*>) <*statement*>

<*if then else statement*>::= if (<*expression*>) <*statement no short if*> else <*statement*>

<*if then else statement no short if*> ::= if (<*expression*>) <*statement no short if*> else <*statement no short if*>

<*switch statement*> ::= switch (<*expression*>) <*switch block*>

<*switch block*> ::= { <*switch block statement groups*>? <*switch labels*>? }

<*switch block statement groups*> ::= <*switch block statement group*> | <*switch block statement groups*> <*switch block statement group*>

<*switch block statement group*> ::= <*switch labels*> <*block statements*>

<*switch labels*> ::= <*switch label*> | <*switch labels*> <*switch label*>

<*switch label*> ::= case <*constant expression*> : | default :

<*while statement*> ::= while (<*expression*>) <*statement*>

<*while statement no short if*> ::= while (<*expression*>) <*statement no short if*>

<*do statement*> ::= do <*statement*> while (<*expression*>) ;

<*for statement*> ::= for (<*for init*>? ; <*expression*>? ; <*for update*>?) <*statement*>

<*for statement no short if*> ::= for (<*for init*>? ; <*expression*>? ; <*for update*>?) <*statement no short if*>

<*for init*> ::= <*statement expression list*> | <*local variable declaration*>

<*for update*> ::= <*statement expression list*>

<*statement expression list*> ::= <*statement expression*> | <*statement expression list*> , <*statement expression*>

<*break statement*> ::= break <*identifier*>? ;

<*continue statement*> ::= continue <*identifier*>? ;

<*return statement*> ::= return <*expression*>? ;

<*throws statement*> ::= throw <*expression*> ;

<*synchronized statement*> ::= synchronized (<*expression*>) <*block*>

<*try statement*> ::= try <*block*> <*catches*> | try <*block*> <*catches*>? <*finally*>

<*catches*> ::= <*catch clause*> | <*catches*> <*catch clause*>

<*catch clause*> ::= catch (<*formal parameter*>) <*block*>

<*finally* > ::= finally <*block*>

Expressions

<*constant expression*> ::= <*expression*>

<*expression*> ::= <*assignment expression*>

<*assignment expression*> ::= <*conditional expression*> | <*assignment*>

<*assignment*> ::= <*left hand side*> <*assignment operator*> <*assignment expression*>

<*left hand side*> ::= <*expression name*> | <*field access*> | <*array access*>

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

<*conditional expression*> ::= <*conditional or expression*> | <*conditional or expression*> ? <*expression*> : <*conditional expression*>

<*conditional or expression*> ::= <*conditional and expression*> | <*conditional or expression*> || <*conditional and expression*>

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

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

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

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

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

<*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*>

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

<*additive expression*> ::= <*multiplicative expression*> | <*additive expression*> + <*multiplicative expression*> | <*additive expression*> - <*multiplicative expression*>

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

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

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

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

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

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

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

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

<*postfix expression*> ::= <*primary*> | <*expression name*> | <*postincrement expression*> | <*postdecrement expression*>

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

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

<*primary*> ::= <*primary no new array*> | <*array creation expression*>

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

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

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

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

<*dim exprs*> ::= <*dim expr*> | <*dim exprs*> <*dim expr*>

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

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

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

Tokens

<*package name*> ::= <*identifier*> | <*package name*> . <*identifier*>

<*type name*> ::= <*identifier*> | <*package name*> . <*identifier*>

<*simple type name*> ::= <*identifier*>

<*expression name*> ::= <*identifier*> | <*ambiguous name*> . <*identifier*>

<*method name*> ::= <*identifier*> | <*ambiguous name*>. <*identifier*>

<*ambiguous name*>::= <*identifier*> | <*ambiguous name*>. <*identifier*>

<*literal*> ::= <*integer literal*> | <*floating-point literal*> | <*boolean literal*> | <*character literal*> | <*string literal*> | <*null literal*>

<*integer literal*> ::= <*decimal integer literal*> | <*hex integer literal*> | <*octal integer literal*>

<*decimal integer literal*> ::= <*decimal numeral*> <*integer type suffix*>?

<*hex integer literal*> ::= <*hex numeral*> <*integer type suffix*>?

<*octal integer literal*> ::= <*octal numeral*> <*integer type suffix*>?

<*integer type suffix*> ::= l | L

<*decimal numeral*> ::= 0 | <*non zero digit*> <*digits*>?

<*digits*> ::= <*digit*> | <*digits*> <*digit*>

<*digit*> ::= 0 | <*non zero digit*>

<*non zero digit*> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<*hex numeral*> ::= 0 x <*hex digit*> | 0 X <*hex digit*> | <*hex numeral*> <*hex digit*>

<*hex digit*> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F

<*octal numeral*> ::= 0 <*octal digit*> | <*octal numeral*> <*octal digit*>

<*octal digit*> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<*floating-point literal*> ::= <*digits*> . <*digits*>? <*exponent part*>? <*float type suffix*>?

<*digits*> <*exponent part*>? <*float type suffix*>?

<*exponent part*> ::= <*exponent indicator*> <*signed integer*>

<*exponent indicator*> ::= e | E

<*signed integer*> ::= <*sign*>? <*digits*>

<*sign*> ::= + | -

<*float type suffix*> ::= f | F | d | D

<*boolean literal*> ::= **true** | **false**

<*character literal*> ::=' <*single character*>' |' <*escape sequence*> '

<*single character*> ::= <*input character*> except ' and \

<*string literal*> ::= " <*string characters*>?"

<*string characters*> ::= <*string character*> | <*string characters*> <*string character*>

<*string character*> ::= <*input character*> except " and \ | <*escape character*>

<*null literal*> ::= **null**

<*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 iterative appearance of X is written {X}.

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 above - these are *reserved words* in Java.

The only BNF extention used here is the optional construct which is written with '?' added as a suffix to a terminal or non-terminal. 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.