Java 15 Language Specification ↗

Java 18 Language Specification ↗

Java 19 Language Specification ↗

### Definition

A lexeme is a sequence of characters constituting a fundamental unit of a programming language somewhat like words in natural language used to build more complex and significant grammatical constructs.

A Unicode escape, like `\u0065`, is an alternative name or representation for a character using the Unicode standard's codepoint in hexadecimal digits. It is not a lexeme, to Java it is just another character out of which lexemes are built.

Like words and punctuation in an English composition, every little character of a Java program is part of one of seven things. These things are called tokens or lexemes. The seven different kinds of lexical tokens in Java:

1. white space
2. comments and Java Doc comments
3. punctuation aka separators or delimiters
4. identifiers (Unicode letters)
5. Java 18 keywords and Contextual Keywords
6. Java 18 literals
7. operators

> *White space is defined as the ASCII space character, horizontal tab character, form feed character, and line terminator characters.*

A line terminator is one of:

- the ASCII LF character, also known as "newline"
- the ASCII CR character, also known as "return"
- the ASCII CR character followed by the ASCII LF character

“As a special concession for compatibility with certain operating systems, the ASCII SUB character (`\u001a`, or control-Z) is ignored if it is the last character” in the input.

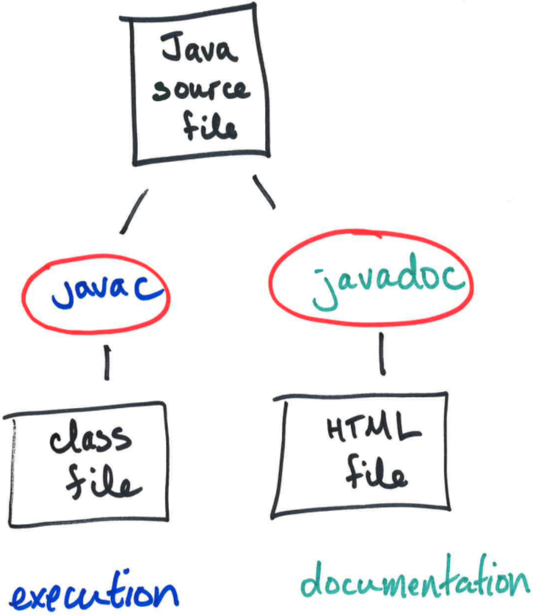
Don’t use tab, form feed, or sub.

- “end of line” comments: // . . . line-terminator. Both single-line and trailing.
- block: /∗ . . . ∗/ May include line-terminators, but not nested block comments.
  - `javadoc` comments: /∗∗ . . . ∗/

Java supports literate programming by having tools the make the source code executable and extract the documentation as structured HTML files.

**Definition**

The practice of literate programming combines source code and documentation into a single source file.


Java source file
javac
javadoc
class file
HTML file
execution
documentation

```java
// import statements

/**
 * @author      Firstname Lastname <address @ example.com>
 * @version     1.6                 (current version number
 * @since       1.2          (the version of the package this
 */
public class Test {
    // class body
}
```

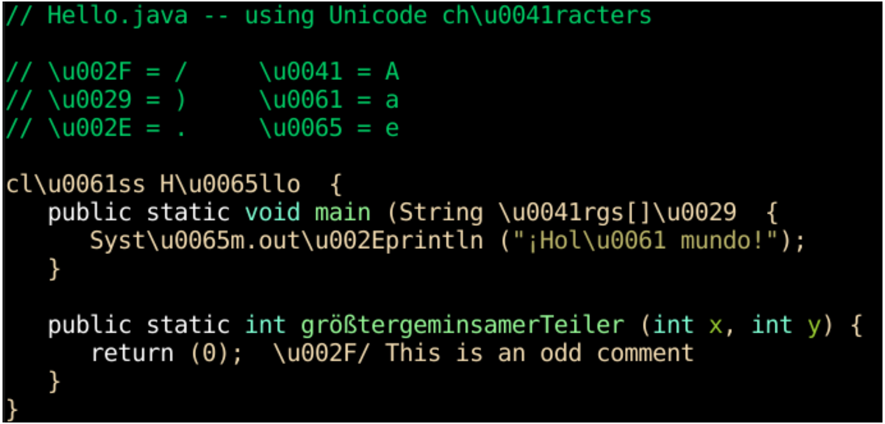
```
/**
 * Short one line description.
 * <p>
 * Longer description. If there were any, it
 * would be here.
 * <p>
 * More explanations follow in consecutive
 * paragraphs separated by HTML paragraph breaks.
 *
 * @param  variable Description text text text.
 * @return Description text text text.
 */
public int methodName (...) {
    // method body with a return statement
}
```

# Identifiers

Identifiers: letters (of Unicode), digits, _, $. The $ is intended for use in computer generated Java code or to access names in legacy code, not but for use in ordinary programming.

- `Identifier.java`↗

```
String
i3
MAX_VALUE
```



```
// Hello.java -- using Unicode ch\u0041racters

// \u002F = /     \u0041 = A
// \u0029 = )     \u0061 = a
// \u002E = .     \u0065 = e

cl\u0061ss H\u0065llo  {
   public static void main (String \u0041rgs[]\u0029  {
      Syst\u0065m.out\u002Eprintln ("¡Hol\u0061 mundo!");
   }

   public static int größtergeminsamerTeiler (int x, int y) {
      return (0);  \u002F/ This is an odd comment
   }
}
```

# Java 19; 51 Keywords

keywords ↗

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

The keywords const and goto are reserved, even though they are not currently used. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in programs. The keyword strictfp is obsolete and should be used anymore. The keyword _ (underscore) is reserved for possible future use in parameter declarations.

# Java 19; 16 Contextual Keywords

`exports module non-sealed open opens permits provides record requires sealed to transitive uses var with yield`

# Literals

1. taking words in their usual or most basic sense without metaphor or allegory. “dreadful in its literal sense, full of dread”
2. (of a translation) representing the exact words of the original text. “a literal translation from the Spanish”
3. (in a programming language) tokens or lexemes representing directly a specific value

The Java tutorial at Oracle is a good reference.
data types ↗ tutorial

1. literals of type int, long
2. literals of type float, double
3. literals of type character (character escapes)
4. literals of type String, text block `"""`
5. literals of type boolean (`true` and `false`)
6. the literal `null`
7. Java 18 literals of type `Class`↗

# 1. Literals of type int, long

Decimal, octal hexidecimal, and binary `int` literals:

```
0   2   0372   0xDadaCafe   1997   0X00FF00FF   0xCAFE_BABE 0b0010_0101
```

Leading 0's indicate an integer base 8. So `010` is actually $10_8$ and so equals $8_{10}$. But don't use them anyway.
The data type `int` is the default, i.e., you get an `int` if you omit the "l" or "L" at the end of the literal.

```
0l   0777L 0x1000L   2145L   0xC0B0L   1l
```

Don't use the letter "l" as it is confused with the digit "1" sometimes.

# 2. Literals of type float, long

The data type double is the default, ie., you get an double if you omit the letter “d,” “D,” “f,” or “F” at the end.

```
1e1f  2.F  .3f  0f  2F  3f  3.14f  6.02E23f
```

```
1E1  2.   .3   0.0   3.14   1e-9d   1e135
```

Hexadecimal floating-points are less-well know, but significant (see the discussion on data later)

```
0XabaceP-1F  0Xabace.abcdeP+3F  0Xabace.abcdeP99F 0xABACE.ABCDEp99f
```

The exponent is marked by the letter “p” or “P” (for power) is in *decimal*.

# Hexadecimal Floating-Point Literals

[This discussion belongs under data!]
Hexadecimal floating-point literals originated in C99 and were later included in a revision of the IEEE 754 floating-point standard. These literals are represented without loss in standard hardware (unlike decimal literals).

```
0x1.8p1
```

to be to used represent the value 3; $1.8_{16} \times 2^1 = 1.5_{10} \times 2 = 3$. More usefully, the maximum value of can be written as `0x1.fffffffffffffp1023` and the minimum value of $2^{-1074}$ can be written as `0x1.0P-1074` or `0x0.0000000000001P-1022`, which maps easily to the various fields of the floating-point representation and is more readable than the raw-bit encoding. In addition, "printf" facility including the `%a` format for hexadecimal floating-point.

# 3. Character literals and Character Escapes

```
char ch = ’a’;
char unicodeCh = ’\u03A9’; // uppercase Greek Omega
char es = ’\’’;            // single quote
```

Character escape sequences are not unicode escape sequecnes, nor are they format specificiers (for `printf`).

| Character Escape | Description |
|---|---|
| `\t` | a tab character |
| `\b` | a backspace character |
| `\n` | a newline character |
| `\r` | a carriage return character |
| `\f` | a form feed character |
| `\’` | a single quote character |
| `\"` | a double quote character |
| `\\` | a backslash character |

# 4. Strings and Text Blocks

String literals are unremarkable.

```
String greeting = "Hello world!";
String address = "575 Hibiscus #5, Melbourne, FL 32901";
```

- Block.java, an example of text blocks

Know not only literals, but the Java API for string methods.

# 5,6. Boolean literals and Null

Three literals are composed of just letters: `true`, `false`, and `null`. These are not keywords, but they are not legal identifiers either. They are literals.

```
int true = 20;      // syntax error: "not a statement"
int false = 30;     // syntax error: "not a statement"
float null = 23.6f; // syntax error: "not a statement"

// contextual keywords CAN be used as identifiers
int record = 10;
```

# 7. Literals of type Class

# Delimiters/Separators/Punctuation

`( ) { } [ ] ; , . ... @ ::`

# Operators

```
=   >    <  !   ~   ?   :   ->
==  >=  <=  !=  &&  ||  ++  --
+   -   *   /   &   |   ^   %   <<   >>   >>>
+=  -=  *=  /=  &=  |=  ^=  %=  <<=  >>=  >>>=
```

A sequence of Java lexemes. Completely legal, but pointless.

```
class     Main      {
public    static    void
main(
final     String    []
args){
// Do not make useless comments!
System    .         out
.         println   (
null=="Hello world"+"!"+42)
;}}
```