WIKIPEDIA

List of Java keywords

In the Java programming language, a **Keyword** is any one of 51 reserved words[1] that have a predefined meaning in the language; because of this, programmers cannot use keywords as names for variables, methods, classes, or as any other identifier.[2] Of these 51 keywords, 49 are in use and 2 are not in use. Due to their special functions in the language, most integrated development environments for Java use syntax highlighting to display keywords in a different colour for easy identification.

```
public void processData() {
   do {
      int data = getData();

      if (data < 0)
         performOperation1(data);
      else
         performOperation2(data);
   } while (hasMoreData());
}
```

A snippet of Java code with keywords highlighted in blue and bold font

Contents

List

`abstract`
: A method with no definition must be declared as abstract and the class containing it must be declared as abstract. Abstract classes cannot be instantiated. Abstract methods must be implemented in the sub classes. The abstract keyword cannot be used with variables or constructors. Note that an abstract class isn't required to have an abstract method at all.

`assert` **(added in J2SE 1.4)**[3]
: Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort. Optionally enable by ClassLoader method.

`boolean`
: Defines a boolean variable for the values "true" or "false" only. By default, the value of boolean primitive type is false. This keyword is also used to declare that a method returns a value of the primitive type `boolean`.

`break`
: Used to end the execution in the current loop body.

`byte`
: The `byte` keyword is used to declare a field that can hold an 8-bit signed two's complement integer.[4][5] This keyword is also used to declare that a method returns a value of the primitive type `byte`.[6][7]

case
A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, then executes all statements that follow the matching case label; see switch.[8][9]

catch
Used in conjunction with a try block and an optional finally block. The statements in the catch block specify what to do if a specific type of exception is thrown by the try block.

char
Defines a character variable capable of holding any character of the java source file's character set.

class
A type that defines the implementation of a particular kind of object. A class definition defines instance and class fields, methods, and inner classes as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass is implicitly Object (https://docs.oracle.com/javase/10/docs/api/java/lang/Object.html). The class keyword can also be used in the form Class**.class** to get a Class object without needing an instance of that class. For example, **String.class** can be used instead of doing **new String().getClass()**.

const
Unused but reserved.

continue
Used to resume program execution at the end of the current loop body. If followed by a label, continue resumes execution at the end of the enclosing labeled loop body.

default
The default keyword can optionally be used in a switch statement to label a block of statements to be executed if no case matches the specified value; see *switch*.[8][9] Alternatively, the default keyword can also be used to declare default values in a Java annotation. From Java 8 onwards, the default keyword can be used to allow an interface to provide an implementation of a method.

do
The do keyword is used in conjunction with while to create a do-while loop, which executes a block of statements associated with the loop and then tests a boolean expression associated with the while. If the expression evaluates to true, the block is executed again; this continues until the expression evaluates to false.[10][11]

double
The double keyword is used to declare a variable that can hold a 64-bit double precision IEEE 754 floating-point number.[4][5] This keyword is also used to declare that a method returns a value of the primitive type double.[6][7]

else
The else keyword is used in conjunction with if to create an if-else statement, which tests a boolean expression; if the expression evaluates to true, the block of statements associated with the if are evaluated; if it evaluates to false, the block of statements associated with the else are evaluated.[12][13]

enum (added in J2SE 5.0)[3]
A Java keyword used to declare an enumerated type. Enumerations extend the base class Enum (https://docs.oracle.com/javase/10/docs/api/java/lang/Enum.html).

extends

Used in a class declaration to specify the superclass; used in an interface declaration to specify one or more superinterfaces. Class X extends class Y to add functionality, either by adding fields or methods to class Y, or by overriding methods of class Y. An interface Z extends one or more interfaces by adding methods. Class X is said to be a subclass of class Y; Interface Z is said to be a subinterface of the interfaces it extends.
Also used to specify an upper bound on a type parameter in Generics.

`final`
Define an entity once that cannot be changed nor derived from later. More specifically: a final class cannot be subclassed, a final method cannot be overridden, and a final variable can occur at most once as a left-hand expression on an executed command. All methods in a final class are implicitly `final`.

`finally`
Used to define a block of statements for a block defined previously by the `try` keyword. The `finally` block is executed after execution exits the `try` block and any associated `catch` clauses regardless of whether an exception was thrown or caught, or execution left method in the middle of the `try` or `catch` blocks using the `return` keyword.

`float`
The `float` keyword is used to declare a variable that can hold a 32-bit single precision IEEE 754 floating-point number.[4][5] This keyword is also used to declare that a method returns a value of the primitive type `float`.[6][7]

`for`
The `for` keyword is used to create a for loop, which specifies a variable initialization, a boolean expression, and an incrementation. The variable initialization is performed first, and then the boolean expression is evaluated. If the expression evaluates to `true`, the block of statements associated with the loop are executed, and then the incrementation is performed. The boolean expression is then evaluated again; this continues until the expression evaluates to `false`.[14]

As of J2SE 5.0, the `for` keyword can also be used to create a so-called "enhanced for loop",[15] which specifies an array or `Iterable (https://docs.oracle.com/javase/10/docs/api/java/lang/Iterable.html)` object; each iteration of the loop executes the associated block of statements using a different element in the array or `Iterable`.[14]

`goto`
Unused

`if`
The `if` keyword is used to create an if statement, which tests a boolean expression; if the expression evaluates to `true`, the block of statements associated with the if statement is executed. This keyword can also be used to create an if-else statement; see *`else`*.[12][13]

`implements`
Included in a class declaration to specify one or more interfaces that are implemented by the current class. A class inherits the types and abstract methods declared by the interfaces.

`import`
Used at the beginning of a source file to specify classes or entire Java packages to be referred to later without including their package names in the reference. Since J2SE 5.0, `import` statements can import `static` members of a class.

`instanceof`
A binary operator that takes an object reference as its first operand and a class or interface as its second operand and produces a boolean result. The `instanceof` operator evaluates to true if and only if the runtime type of the object is assignment compatible with the class or interface.

int
: The `int` keyword is used to declare a variable that can hold a 32-bit signed two's complement integer.[4][5] This keyword is also used to declare that a method returns a value of the primitive type `int`.[6][7]

interface
: Used to declare a special type of class that only contains abstract or default methods, constant (`static final`) fields and `static` interfaces. It can later be implemented by classes that declare the interface with the `implements` keyword. As multiple inheritance is not allowed in Java, interfaces are used to circumvent it. An interface can be defined within another interface.

long
: The `long` keyword is used to declare a variable that can hold a 64-bit signed two's complement integer.[4][5] This keyword is also used to declare that a method returns a value of the primitive type `long`.[6][7]

native
: Used in method declarations to specify that the method is not implemented in the same Java source file, but rather in another language.[7]

new
: Used to create an instance of a class or array object. Using keyword for this end is not completely necessary (as exemplified by Scala), though it serves two purposes: it enables the existence of different namespace for methods and class names, it defines statically and locally that a fresh object is indeed created, and of what runtime type it is (arguably introducing dependency into the code).

package
: Java package is a group of similar classes and interfaces. Packages are declared with the `package` keyword.

private
: The `private` keyword is used in the declaration of a method, field, or inner class; private members can only be accessed by other members of their own class.[16]

protected
: The `protected` keyword is used in the declaration of a method, field, or inner class; protected members can only be accessed by members of their own class, that class's subclasses or classes from the same package.[16]

public
: The `public` keyword is used in the declaration of a class, method, or field; public classes, methods, and fields can be accessed by the members of any class.[16]

return
: Used to finish the execution of a method. It can be followed by a value required by the method definition that is returned to the caller.

short
: The `short` keyword is used to declare a field that can hold a 16-bit signed two's complement integer.[4][5] This keyword is also used to declare that a method returns a value of the primitive type `short`.[6][7]

static
: Used to declare a field, method, or inner class as a class field. Classes maintain one copy of class fields regardless of how many instances exist of that class. `static` also is used to define a method as a class method. Class methods are bound to the class instead of to a specific

instance, and can only operate on class fields. (Classes and interfaces declared as `static` members of another class or interface are actually top-level classes and are *not* inner classes.)

`strictfp` (added in J2SE 1.2)[3]
A Java keyword used to restrict the precision and rounding of floating point calculations to ensure portability.[7]

`super`
Inheritance basically used to achieve dynamic binding or run-time polymorphism in java. Used to access members of a class inherited by the class in which it appears. Allows a subclass to access overridden methods and hidden members of its superclass. The `super` keyword is also used to forward a call from a constructor to a constructor in the superclass.
Also used to specify a lower bound on a type parameter in Generics.

`switch`
The `switch` keyword is used in conjunction with `case` and `default` to create a switch statement, which evaluates a variable, matches its value to a specific `case`, and executes the block of statements associated with that `case`. If no `case` matches the value, the optional block labelled by `default` is executed, if included.[8][9]

`synchronized`
Used in the declaration of a method or code block to acquire the mutex lock for an object while the current thread executes the code.[7] For static methods, the object locked is the class's `Class`. Guarantees that at most one thread at a time operating on the same object executes that code. The mutex lock is automatically released when execution exits the synchronized code. Fields, classes and interfaces cannot be declared as *synchronized*.

`this`
Used to represent an instance of the class in which it appears. `this` can be used to access class members and as a reference to the current instance. The `this` keyword is also used to forward a call from one constructor in a class to another constructor in the same class.

`throw`
Causes the declared exception instance to be thrown. This causes execution to continue with the first enclosing exception handler declared by the `catch` keyword to handle an assignment compatible exception type. If no such exception handler is found in the current method, then the method returns and the process is repeated in the calling method. If no exception handler is found in any method call on the stack, then the exception is passed to the thread's uncaught exception handler.

`throws`
Used in method declarations to specify which exceptions are not handled within the method but rather passed to the next higher level of the program. All uncaught exceptions in a method that are not instances of `RuntimeException` must be declared using the `throws` keyword.

`transient`
Declares that an instance field is not part of the default serialized form of an object. When an object is serialized, only the values of its non-transient instance fields are included in the default serial representation. When an object is deserialized, transient fields are initialized only to their default value. If the default form is not used, e.g. when a *serialPersistentFields* table is declared in the class hierarchy, all `transient` keywords are ignored.[17][18]

`try`
Defines a block of statements that have exception handling. If an exception is thrown inside the `try` block, an optional `catch` block can handle declared exception types. Also, an optional `finally` block can be declared that will be executed when execution exits the `try` block and `catch` clauses, regardless of whether an exception is thrown or not. A `try` block must have at least one `catch` clause or a `finally` block.

`void`
: The `void` keyword is used to declare that a method does not return any value.[6]

`volatile`
: Used in field declarations to guarantee visibility of changes to variables across threads. Every read of a volatile variable will be read from main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.[19] Methods, classes and interfaces thus cannot be declared *volatile*, nor can local variables or parameters.

`while`
: The `while` keyword is used to create a while loop, which tests a boolean expression and executes the block of statements associated with the loop if the expression evaluates to `true`; this continues until the expression evaluates to `false`. This keyword can also be used to create a do-while loop; see *do*.[10][11]

Reserved words for literal values

`true`
: A boolean literal value.

`false`
: A boolean literal value.

`null`
: A reference literal value.

Special identifiers

`var`
: A special identifier that cannot be used as a type name (since Java 10).[20]

`_`
: Added in Java 11, the underscore has become a keyword and cannot be used as a variable name anymore.[21]

Unused

`const`
: Although reserved as a keyword in Java, `const` is not used and has no function.[2][1] For defining constants in Java, see the `final` keyword.

`goto`
: Although reserved as a keyword in Java, `goto` is not used and has no function.[2][1]

See also

- Java annotation

References

1. Flanagan 2005, p. 20.

2. "Java Language Specification - Section 3.9: Keywords" (https://docs.oracle.com/javase/specs/jls/se11/html/jls-3.html#jls-3.9). *The Java Language Specification*. Oracle. 2018-08-21. Retrieved 2018-12-25.
3. "Java Language Keywords" (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html). *The Java Tutorials*. Sun Microsystems, Inc. Retrieved 2017-07-24.
4. "Primitive Data Types" (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2008-12-03.
5. Flanagan 2005, p. 22.
6. "Returning a Value from a Method" (https://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2008-12-03.
7. Flanagan 2005, pp. 66-67.
8. "The switch Statement" (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2014-12-18.
9. Flanagan 2005, pp. 46-48.
10. "The while and do-while Statements" (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2008-12-03.
11. Flanagan 2005, pp. 48-49.
12. "The if-then and if-then-else Statements" (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2008-12-03.
13. Flanagan 2005, pp. 44-46.
14. "The for Statement" (https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2008-12-03.
15. Flanagan 2005, pp. 50-54.
16. "Controlling Access to Members of a Class" (https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html). *The Java Tutorials*. Sun Microsystems, Inc. February 14, 2008. Retrieved 2008-12-03.
17. "Java Object Serialization Specification version 1.5.0" (http://download-llnw.oracle.com/javase/1.5.0/docs/guide/serialization/spec/serial-arch.html#6250). Sun/Oracle. 2004. 1.5 Defining Serializable Fields for a Class. Retrieved 2010-09-16.
18. Grosso, William (November 21, 2001). "Java RMI: Serialization" (http://onjava.com/pub/a/onjava/excerpt/JavaRMI_10/index.html?page=3). *ONJava*. O'Reilly Media. Declaring serialPersistentFields. Retrieved 2010-09-16.
19. "Java Volatile Keyword" (http://tutorials.jenkov.com/java-concurrency/volatile.html).
20. "Chapter 3. Lexical Structure" (https://docs.oracle.com/javase/specs/jls/se11/html/jls-3.html#jls-3.9). *docs.oracle.com*. Retrieved 2018-12-25.
21. Goetz, Brian. "Warning about single underscore identifier" (http://openjdk.5641.n7.nabble.com/Warning-about-single-underscore-identifier-td145974.html#a146426). *OpenJDK Lambda Development*.

External links

- Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad (June 2005). *Java Language Specification* (http://java.sun.com/docs/books/jls/index.html) (Third ed.). Addison-Wesley Professional. ISBN 978-0-321-24678-3. Retrieved 2008-12-03.
- Flanagan, David (March 2005). *Java in a Nutshell* (https://archive.org/details/javainnutshell00flan) (Fifth ed.). O'Reilly Media. ISBN 978-0-596-00773-7. Retrieved 2010-03-03.

Retrieved from "https://en.wikipedia.org/w/index.php?title=List_of_Java_keywords&oldid=954521615"

This page was last edited on 2 May 2020, at 21:25 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.