

# Java (programming language)

**Java** is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers *write once, run anywhere* (WORA),<sup>[17]</sup> meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.<sup>[18]</sup> Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub,<sup>[19][20]</sup> particularly for client-server web applications, with a reported 9 million developers.<sup>[21]</sup>

Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GPL-2.0-only license. Oracle offers its own HotSpot Java Virtual Machine, however the official reference implementation is the OpenJDK JVM which is free open source software and used by most developers and is the default JVM for almost all Linux distributions.

As of March 2021, the latest version is Java 16, with Java 11, a currently supported long-term support (LTS) version, released on September 25, 2018. Oracle released the last zero-cost public update for the legacy version Java 8 LTS in January 2019 for commercial use, although it will otherwise still support Java 8 with public updates for personal use indefinitely. Other vendors have begun to offer zero-cost builds of OpenJDK 8 and 11 that are still receiving security and other upgrades.

Oracle (and others) highly recommend uninstalling outdated versions of Java because of serious risks due to unresolved security issues.<sup>[22]</sup> Since Java 9, 10, 12, 13, 14, and 15 are no longer supported, Oracle advises its users to immediately transition to the latest version (currently Java 16) or an LTS release.

## Java



<b>Paradigm</b>	Multi-paradigm: generic, <u>object-oriented</u> (class-based), functional, imperative, reflective
<b>Designed by</b>	<u>James Gosling</u>
<b>Developer</b>	Oracle Corporation
<b>First appeared</b>	May 23, 1995 <sup>[1]</sup>
<b>Stable release</b>	Java SE 16.0.2 <sup>[2]</sup> <div><div></div></div> / 20 July 2021
<b>Typing discipline</b>	Static, strong, safe, <u>nominative</u> , manifest
<b>Filename extensions</b>	.java, .class, .jar
<b>Website</b>	oracle.com/java/ (http://oracle.com/java/)
<b>Influenced by</b>	
CLU, <sup>[3]</sup> Simula67, <sup>[3]</sup> Lisp, <sup>[3]</sup> Smalltalk, <sup>[3]</sup> Ada 83, C++, <sup>[4]</sup> C#, <sup>[5]</sup> Eiffel, <sup>[6]</sup> Mesa, <sup>[7]</sup> Modula-3, <sup>[8]</sup> Oberon, <sup>[9]</sup> Objective-C, <sup>[10]</sup> UCSD Pascal, <sup>[11][12]</sup> Object Pascal <sup>[13]</sup>	
<b>Influenced</b>	
Ada 2005, BeanShell, C#, Chapel, <sup>[14]</sup> Clojure, ECMAScript, Fantom, Gambas, <sup>[15]</sup> Groovy,	

# Contents

---

## History

[Principles](#)

[Versions](#)

## Editions

## Execution system

[Java JVM and bytecode](#)

[Performance](#)

[Non-JVM](#)

[Automatic memory management](#)

## Syntax

[Hello world example](#)

[Example with methods](#)

## Special classes

[Applet](#)

[Servlet](#)

[JavaServer Pages](#)

[Swing application](#)

[JavaFX application](#)

[Generics](#)

## Criticism

## Class libraries

## Documentation

## Implementations

## Use outside the Java platform

[Android](#)

[Controversy](#)

## See also

[Comparison of Java with other languages](#)

## References

## Works cited

## External links

[Hack](#),<sup>[16]</sup> [Haxe](#), [J#](#), [Kotlin](#), [PHP](#),  
[Python](#), [Scala](#), [Seed7](#), [Vala](#),  
[JavaScript](#)



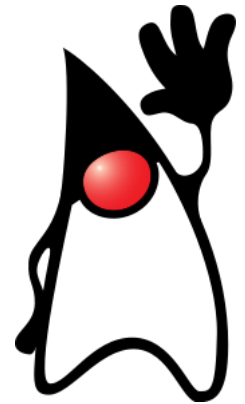
[Java Programming at Wikibooks](#)

# History

---

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.<sup>[23]</sup> Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time.<sup>[24]</sup> The language was initially called *Oak* after an oak tree that stood outside Gosling's office. Later the project went by the name *Green* and was finally renamed *Java*, from Java coffee, a type of coffee from [Indonesia](#).<sup>[25]</sup> Gosling designed Java with a [C/C++](#)-style syntax that system and application programmers would find familiar.<sup>[26]</sup>

Sun Microsystems released the first public implementation as Java 1.0 in 1996.<sup>[27]</sup> It promised **Write Once, Run Anywhere (WORA)** functionality, providing no-cost run-times on popular platforms. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions. Major web browsers soon incorporated the ability to run Java applets within web pages, and Java quickly became popular. The Java 1.0 compiler was re-written in Java by Arthur van Hoff to comply strictly with the Java 1.0 language specification.<sup>[28]</sup> With the advent of Java 2 (released initially as J2SE 1.2 in December 1998 – 1999), new versions had multiple configurations built for different types of platforms. J2EE included technologies and APIs for enterprise applications typically run in server environments, while J2ME featured APIs optimized for mobile applications. The desktop version was renamed J2SE. In 2006, for marketing purposes, Sun renamed new J2 versions as *Java EE*, *Java ME*, and *Java SE*, respectively.



Duke, the Java mascot

In 1997, Sun Microsystems approached the ISO/IEC JTC 1 standards body and later the Ecma International to formalize Java, but it soon withdrew from the process.<sup>[29][30][31]</sup> Java remains a *de facto* standard, controlled through the Java Community Process.<sup>[32]</sup> At one time, Sun made most of its Java implementations available without charge, despite their proprietary software status. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System.

On November 13, 2006, Sun released much of its Java virtual machine (JVM) as free and open-source software (FOSS), under the terms of the GPL-2.0-only license. On May 8, 2007, Sun finished the process, making all of its JVM's core code available under free software/open-source distribution terms, aside from a small portion of code to which Sun did not hold the copyright.<sup>[33]</sup>



James Gosling, the creator of Java, in 2008

Sun's vice-president Rich Green said that Sun's ideal role with regard to Java was as an *evangelist*.<sup>[34]</sup> Following Oracle Corporation's acquisition of Sun Microsystems in 2009–10, Oracle has described itself as the steward of Java technology with a relentless commitment to fostering a community of participation and transparency.<sup>[35]</sup> This did not prevent Oracle from filing a lawsuit against Google shortly after that for using Java inside the Android SDK (see the *Android* section).

On April 2, 2010, James Gosling resigned from Oracle.<sup>[36]</sup>

In January 2016, Oracle announced that Java run-time environments based on JDK 9 will discontinue the browser plugin.<sup>[37]</sup>

Java software runs on everything from laptops to data centers, game consoles to scientific supercomputers.<sup>[38]</sup>

## Principles

There were five primary goals in the creation of the Java language:<sup>[18]</sup>



The TIOBE programming language popularity index graph from 2002 to 2018. Java was steadily on the top from mid-2015 to early 2020.

1. It must be simple, object-oriented, and familiar.
2. It must be robust and secure.
3. It must be architecture-neutral and portable.
4. It must execute with high performance.
5. It must be interpreted, threaded, and dynamic.

## Versions

As of September 2020, Java 8 and 11 are supported as Long Term Support (LTS) versions, and one later non-LTS version is supported.<sup>[39]</sup> Major release versions of Java, along with their release dates:

Version	Date
JDK Beta	1995
JDK1.0	January 23, 1996 <sup>[40]</sup>
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11	September 25, 2018 <sup>[41]</sup>
Java SE 12	March 19, 2019
Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020 <sup>[42]</sup>
Java SE 16	March 16, 2021

## Editions

Sun has defined and supports four editions of Java targeting different application environments and segmented many of its APIs so that they belong to one of the platforms. The platforms are:

- Java Card for smart-cards.<sup>[43]</sup>
- Java Platform, Micro Edition (Java ME) – targeting environments with limited resources.<sup>[44]</sup>
- Java Platform, Standard Edition (Java SE) – targeting workstation environments.<sup>[45]</sup>
- Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.<sup>[46]</sup>

The classes in the Java APIs are organized into separate groups called packages. Each package contains a set of related interfaces, classes, subpackages and exceptions.

Sun also provided an edition called Personal Java that has been superseded by later, standards-based Java ME configuration-profile pairings.

## Execution system

---

### Java JVM and bytecode

One design goal of Java is portability, which means that programs written for the Java platform must run similarly on any combination of hardware and operating system with adequate run time support. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to architecture-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be executed by a virtual machine (VM) written specifically for the host hardware. End users commonly use a Java Runtime Environment (JRE) installed on their machine for standalone Java applications, or in a web browser for Java applets.

Standard libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

The use of universal bytecode makes porting simple. However, the overhead of interpreting bytecode into machine instructions made interpreted programs almost always run more slowly than native executables. Just-in-time (JIT) compilers that compile byte-codes to machine code during runtime were introduced from an early stage. Java itself is platform-independent and is adapted to the particular platform it is to run on by a Java virtual machine (JVM) for it, which translates the Java bytecode into the platform's machine language.<sup>[47]</sup>

### Performance

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++.<sup>[48][49]</sup> However, Java programs' execution speed improved significantly with the introduction of just-in-time compilation in 1997/1998 for Java 1.1,<sup>[50]</sup> the addition of language features supporting better code analysis (such as inner classes, the StringBuilder class, optional assertions, etc.), and optimizations in the Java virtual machine, such as HotSpot becoming Sun's default JVM in 2000. With Java 1.5, the performance was improved with the addition of the java.util.concurrent package, including lock free implementations of the ConcurrentMaps and other multi-core collections, and it was improved further with Java 1.6.

### Non-JVM

Some platforms offer direct hardware support for Java; there are micro controllers that can run Java bytecode in hardware instead of a software Java virtual machine,<sup>[51]</sup> and some ARM-based processors could have hardware support for executing Java bytecode through their Jazelle option, though support has mostly been dropped in current implementations of ARM.

### Automatic memory management

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable memory becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a non-existent object are called, a null pointer exception is thrown.<sup>[52][53]</sup>

One of the ideas behind Java's automatic memory management model is that programmers can be spared the burden of having to perform manual memory management. In some languages, memory for the creation of objects is implicitly allocated on the stack or explicitly allocated and deallocated from the heap. In the latter case, the responsibility of managing memory resides with the programmer. If the program does not deallocate an object, a memory leak occurs. If the program attempts to access or deallocate memory that has already been deallocated, the result is undefined and difficult to predict, and the program is likely to become unstable or crash. This can be partially remedied by the use of smart pointers, but these add overhead and complexity. Note that garbage collection does not prevent logical memory leaks, i.e. those where the memory is still referenced but never used.

Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Explicit memory management is not possible in Java.

Java does not support C/C++ style pointer arithmetic, where object addresses can be arithmetically manipulated (e.g. by adding or subtracting an offset). This allows the garbage collector to relocate referenced objects and ensures type safety and security.

As in C++ and some other object-oriented languages, variables of Java's primitive data types are either stored directly in fields (for objects) or on the stack (for methods) rather than on the heap, as is commonly true for non-primitive data types (but see escape analysis). This was a conscious decision by Java's designers for performance reasons.

Java contains multiple types of garbage collectors. Since Java 9, HotSpot uses the Garbage First Garbage Collector (G1GC) as the default.<sup>[54]</sup> However, there are also several other garbage collectors that can be used to manage the heap. For most applications in Java, G1GC is sufficient. Previously, the Parallel Garbage Collector (<https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/parallel.html>) was used in Java 8.

Having solved the memory management problem does not relieve the programmer of the burden of handling properly other kinds of resources, like network or database connections, file handles, etc., especially in the presence of exceptions.

## Syntax

---

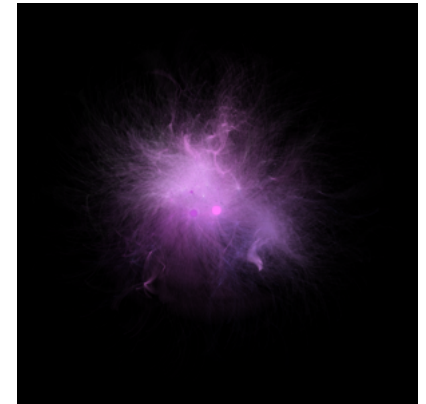
The syntax of Java is largely influenced by C++ and C. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object-oriented language.<sup>[18]</sup> All code is written inside classes, and every data item is an object, with the exception of the primitive data types, (i.e. integers, floating-point numbers, boolean values, and characters), which are not objects for performance reasons. Java reuses some popular aspects of C++ (such as the `printf` method).

Unlike C++, Java does not support operator overloading<sup>[55]</sup> or multiple inheritance for classes, though multiple inheritance is supported for interfaces.<sup>[56]</sup>

Java uses comments similar to those of C++. There are three different styles of comments: a single line style marked with two slashes (//), a multiple line style opened with /\* and closed with \*/, and the Javadoc commenting style opened with /\*\* and closed with \*/. The Javadoc style of commenting allows the user to run the Javadoc executable to create documentation for the program and can be read by some integrated development environments (IDEs) such as Eclipse to allow developers to access documentation within the IDE.

## Hello world example

The traditional Hello world program can be written in Java as:<sup>[57]</sup>



Dependency graph of the Java Core classes (created with jdeps and Gephi)

```

1 public class HelloWorldApp {
2     public static void main(String[] args) {
3         System.out.println("Hello World!"); // Prints the string to the console.
4     }
5 }

```

All source files must be named after the public class they contain, appending the suffix `.java`, for example, `HelloWorldApp.java`. It must first be compiled into bytecode, using a Java compiler, producing a file with the `.class` suffix (`HelloWorldApp.class`, in this case). Only then can it be executed or launched. The Java source file may only contain one public class, but it can contain multiple classes with a non-public access modifier and any number of public inner classes. When the source file contains multiple classes, it is necessary to make one class (introduced by the **class** keyword) public (preceded by the **public** keyword) and name the source file with that public class name.

A class that is not declared public may be stored in any `.java` file. The compiler will generate a class file for each class defined in the source file. The name of the class file is the name of the class, with `.class` appended. For class file generation, anonymous classes are treated as if their name were the concatenation of the name of their enclosing class, a `$`, and an integer.

The keyword **public** denotes that a method can be called from code in other classes, or that a class may be used by classes outside the class hierarchy. The class hierarchy is related to the name of the directory in which the `.java` file is located. This is called an access level modifier. Other access level modifiers include the keywords **private** (a method that can only be accessed in the same class) and **protected** (which allows code from the same package to access). If a piece of code attempts to access private methods or protected methods, the JVM will throw a `SecurityException`.

The keyword **static**<sup>[19]</sup> in front of a method indicates a static method, which is associated only with the class and not with any specific instance of that class. Only static methods can be invoked without a reference to an object. Static methods cannot access any class members that are not also static. Methods that are not designated static are instance methods and require a specific instance of a class to operate.

The keyword **void** indicates that the main method does not return any value to the caller. If a Java program is to exit with an error code, it must call `System.exit()` explicitly.



The method name `main` is not a keyword in the Java language. It is simply the name of the method the Java launcher calls to pass control to the program. Java classes that run in managed environments such as applets and Enterprise JavaBeans do not use or need a `main()` method. A Java program may contain multiple classes that have `main` methods, which means that the VM needs to be explicitly told which class to launch from.

The `main` method must accept an array of **String** (<https://docs.oracle.com/javase/10/docs/api/java/lang/String.html>) objects. By convention, it is referenced as `args` although any other legal identifier name can be used. Since Java 5, the `main` method can also use variable arguments, in the form of `public static void main(String... args)`, allowing the `main` method to be invoked with an arbitrary number of `String` arguments. The effect of this alternate declaration is semantically identical (to the `args` parameter which is still an array of `String` objects), but it allows an alternative syntax for creating and passing the array.

The Java launcher launches Java by loading a given class (specified on the command line or as an attribute in a JAR) and starting its `public static void main(String[])` method. Stand-alone programs must declare this method explicitly. The `String[] args` parameter is an array of `String` objects containing any arguments passed to the class. The parameters to `main` are often passed by means of a command line.

Printing is part of a Java standard library: The **System** (<https://docs.oracle.com/javase/10/docs/api/java/lang/System.html>) class defines a `public static` field called `out` (<https://docs.oracle.com/javase/10/docs/api/java/lang/System.html#out>). The `out` object is an instance of the `PrintStream` (<https://docs.oracle.com/javase/10/docs/api/java/io/PrintStream.html>) class and provides many methods for printing data to standard out, including `println(String)` ([https://docs.oracle.com/javase/10/docs/api/java/io/PrintStream.html#println\(java.lang.String\)](https://docs.oracle.com/javase/10/docs/api/java/io/PrintStream.html#println(java.lang.String))) which also appends a new line to the passed string.

The string "Hello World!" is automatically converted to a `String` object by the compiler.

## Example with methods

```
// This is an example of a single line comment using two slashes

/*
 * This is an example of a multiple line comment using the slash and asterisk.
 * This type of comment can be used to hold a lot of information or deactivate
 * code, but it is very important to remember to close the comment.
 */

package fibsandlies;

import java.util.Map;
import java.util.HashMap;

/**
 * This is an example of a Javadoc comment; Javadoc can compile documentation
 * from this text. Javadoc comments must immediately precede the class, method,
 * or field being documented.
 * @author Wikipedia Volunteers
 */
public class FibCalculator extends Fibonacci implements Calculator {
    private static Map<Integer, Integer> memoized = new HashMap<>();

    /*
     * The main method written as follows is used by the JVM as a starting point
     * for the program.
     */
    public static void main(String[] args) {
```



```

memoized.put(1, 1);
memoized.put(2, 1);
System.out.println(fibonacci(12)); // Get the 12th Fibonacci number and print to console
}

/**
 * An example of a method written in Java, wrapped in a class.
 * Given a non-negative number FIBINDEX, returns
 * the Nth Fibonacci number, where N equals FIBINDEX.
 *
 * @param fibIndex The index of the Fibonacci number
 * @return the Fibonacci number
 */
public static int fibonacci(int fibIndex) {
    if (memoized.containsKey(fibIndex)) return memoized.get(fibIndex);
    else {
        int answer = fibonacci(fibIndex - 1) + fibonacci(fibIndex - 2);
        memoized.put(fibIndex, answer);
        return answer;
    }
}
}

```

## Special classes

### Applet

Java applets were programs that were embedded in other applications, typically in a Web page displayed in a web browser. The Java applet API is now deprecated since Java 9 in 2017.<sup>[58][59]</sup>

### Servlet

Java servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. Servlets are server-side Java EE components that generate responses to requests from clients. Most of the time, this means generating HTML pages in response to HTTP requests, although there are a number of other standard servlet classes available, for example for WebSocket communication.

The Java servlet API has to some extent been superseded (but still used under the hood) by two standard Java technologies for web services:

- the Java API for RESTful Web Services (JAX-RS 2.0) useful for AJAX, JSON and REST services, and
- the Java API for XML Web Services (JAX-WS) useful for SOAP Web Services.

Typical implementations of these APIs on Application Servers or Servlet Containers use a standard servlet for handling all interactions with the HTTP requests and responses that delegate to the web service methods for the actual business logic.

### JavaServer Pages

JavaServer Pages (JSP) are server-side Java EE components that generate responses, typically HTML pages, to HTTP requests from clients. JSPs embed Java code in an HTML page by using the special delimiters `<%` and `%>`. A JSP is compiled to a Java *servlet*, a Java application in its own right, the first

time it is accessed. After that, the generated servlet creates the response.<sup>[60]</sup>

## Swing application

Swing is a graphical user interface library for the Java SE platform. It is possible to specify a different look and feel through the pluggable look and feel system of Swing. Clones of Windows, GTK+, and Motif are supplied by Sun. Apple also provides an Aqua look and feel for macOS. Where prior implementations of these looks and feels may have been considered lacking, Swing in Java SE 6 addresses this problem by using more native GUI widget drawing routines of the underlying platforms.<sup>[61]</sup>

## JavaFX application

JavaFX is a software platform for creating and delivering desktop applications, as well as rich web applications that can run across a wide variety of devices. JavaFX is intended to replace Swing as the standard GUI library for Java SE, but since JDK 11 JavaFX has not been in the core JDK and instead in a separate module.<sup>[62]</sup> JavaFX has support for desktop computers and web browsers on Microsoft Windows, Linux, and macOS. JavaFX does not have support for native OS look and feels.<sup>[63]</sup>

## Generics

In 2004, generics were added to the Java language, as part of J2SE 5.0. Prior to the introduction of generics, each variable declaration had to be of a specific type. For container classes, for example, this is a problem because there is no easy way to create a container that accepts only specific types of objects. Either the container operates on all subtypes of a class or interface, usually Object, or a different container class has to be created for each contained class. Generics allow compile-time type checking without having to create many container classes, each containing almost identical code. In addition to enabling more efficient code, certain runtime exceptions are prevented from occurring, by issuing compile-time errors. If Java prevented all runtime type errors (ClassCastExceptions) from occurring, it would be type safe.

In 2016, the type system of Java was proven unsound.<sup>[64]</sup>

## Criticism

---

Criticisms directed at Java include the implementation of generics,<sup>[65]</sup> speed,<sup>[66]</sup> the handling of unsigned numbers,<sup>[67]</sup> the implementation of floating-point arithmetic,<sup>[68]</sup> and a history of security vulnerabilities in the primary Java VM implementation HotSpot.<sup>[69]</sup>

## Class libraries

---

The Java Class Library is the standard library, developed to support application development in Java. It is controlled by Oracle in cooperation with others through the Java Community Process program.<sup>[70]</sup> Companies or individuals participating in this process can influence the design and development of the APIs. This process has been a subject of controversy during the 2010s.<sup>[71]</sup> The class library contains features such as:

- The core libraries, which include:

- [IO/NIO \(https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html\)](https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html)
- [Networking \(https://docs.oracle.com/javase/8/docs/technotes/guides/net/index.html\)](https://docs.oracle.com/javase/8/docs/technotes/guides/net/index.html) (NOTE: new HTTP Client (<https://docs.oracle.com/en/java/javase/11/docs/api/java.net.http/java/net/http/HttpClient.html>) since Java 11)
- [Reflection](#)
- [Concurrency](#)
- [Generics](#)
- [Scripting/Compiler](#)
- [Functional programming \(Lambda, Streaming\)](#)
- [Collection libraries that implement data structures such as lists, dictionaries, trees, sets, queues and double-ended queue, or stacks<sup>\[72\]</sup>](#)
- [XML Processing \(Parsing, Transforming, Validating\) libraries](#)
- [Security<sup>\[73\]</sup>](#)
- [Internationalization and localization libraries<sup>\[74\]</sup>](#)
- The integration libraries, which allow the application writer to communicate with external systems. These libraries include:
  - The [Java Database Connectivity \(JDBC\) API](#) for database access
  - [Java Naming and Directory Interface \(JNDI\)](#) for lookup and discovery
  - [RMI](#) and [CORBA](#) for distributed application development
  - [JMX](#) for managing and monitoring applications
- [User interface libraries](#), which include:
  - The (heavyweight, or [native](#)) [Abstract Window Toolkit \(AWT\)](#), which provides GUI components, the means for laying out those components and the means for handling events from those components
  - The (lightweight) [Swing](#) libraries, which are built on AWT but provide (non-native) implementations of the AWT widgetry
  - APIs for audio capture, processing, and playback
  - [JavaFX](#)
- A platform dependent implementation of the Java virtual machine that is the means by which the bytecodes of the Java libraries and third party applications are executed
- [Plugins](#), which enable [applets](#) to be run in web browsers
- [Java Web Start](#), which allows Java applications to be efficiently distributed to [end users](#) across the Internet
- [Licensing and documentation](#)

## Documentation

---

Javadoc is a comprehensive documentation system, created by [Sun Microsystems](#). It provides developers with an organized system for documenting their code. Javadoc comments have an extra asterisk at the beginning, i.e. the delimiters are `/**` and `*/`, whereas the normal multi-line comments in Java are set off with the delimiters `/*` and `*/`, and single-line comments start off the line with `//`.<sup>[75]</sup>

## Implementations

---

Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of [Sun Microsystems](#) on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for [Microsoft Windows](#) (still works for XP, while only later versions are currently officially supported), [macOS](#), [Linux](#), and [Solaris](#). Because Java lacks any formal standardization recognized by [Ecma International](#), [ISO/IEC](#), [ANSI](#), or other third-party standards organizations, the Oracle implementation is the [de facto standard](#).

The Oracle implementation is packaged into two different distributions: The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end users, and the [Java Development Kit](#) (JDK), which is intended for software developers and includes development tools such as the [Java compiler](#), [Javadoc](#), [Jar](#), and a [debugger](#). Oracle has also released [GraalVM](#), a high performance Java dynamic compiler and interpreter.

[OpenJDK](#) is another notable Java SE implementation that is licensed under the GNU GPL. The implementation started when Sun began releasing the Java source code under the GPL. As of Java SE 7, OpenJDK is the official Java reference implementation.

The goal of Java is to make all implementations of Java compatible. Historically, Sun's trademark license for usage of the Java brand insists that all implementations be *compatible*. This resulted in a legal dispute with [Microsoft](#) after Sun claimed that the Microsoft implementation did not support [RMI](#) or [JNI](#) and had added platform-specific features of their own. Sun sued in 1997, and, in 2001, won a settlement of US\$20 million, as well as a court order enforcing the terms of the license from Sun.<sup>[76]</sup> As a result, Microsoft no longer ships Java with [Windows](#).

Platform-independent Java is essential to Java EE, and an even more rigorous validation is required to certify an implementation. This environment enables portable server-side applications.

## Use outside the Java platform

---

The Java programming language requires the presence of a software platform in order for compiled programs to be executed.

Oracle supplies the [Java platform](#) for use with Java. The [Android SDK](#) is an alternative software platform, used primarily for developing [Android applications](#) with its own [GUI system](#).

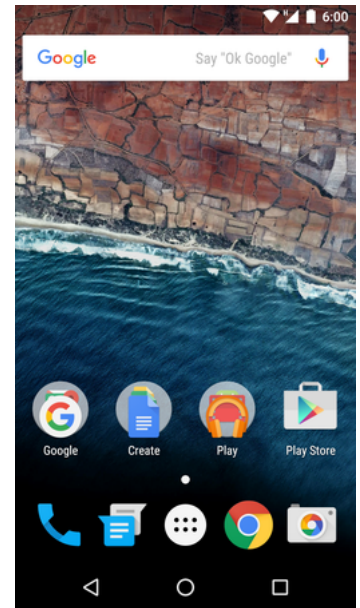
### Android

The Java language is a key pillar in Android, an [open source](#) mobile operating system. Although Android, built on the [Linux kernel](#), is written largely in C, the [Android SDK](#) uses the Java language as the basis for Android applications but does not use any of its [standard GUI](#), [SE](#), [ME](#) or other established Java standards.<sup>[77]</sup> The bytecode language supported by the Android SDK is incompatible with Java bytecode and runs on its own virtual machine, optimized for low-memory devices such as [smartphones](#) and [tablet computers](#). Depending on the Android version, the bytecode is either interpreted by the [Dalvik virtual machine](#) or compiled into native code by the [Android Runtime](#).

Android does not provide the full Java SE standard library, although the Android SDK does include an independent implementation of a large subset of it. It supports Java 6 and some Java 7 features, offering an implementation compatible with the standard library ([Apache Harmony](#)).

### Controversy

The use of Java-related technology in Android led to a legal dispute between Oracle and Google. On May 7, 2012, a San Francisco jury found that if APIs could be copyrighted, then Google had infringed Oracle's copyrights by the use of Java in Android devices.<sup>[78]</sup> District Judge William Alsup ruled on May 31, 2012, that APIs cannot be copyrighted,<sup>[79]</sup> but this was reversed by the United States Court of Appeals for the Federal Circuit in May 2014.<sup>[80]</sup> On May 26, 2016, the district court decided in favor of Google, ruling the copyright infringement of the Java API in Android constitutes fair use.<sup>[81]</sup> In March 2018, this ruling was overturned by the Appeals Court, which sent down the case of determining the damages to federal court in San Francisco.<sup>[82]</sup> Google filed a petition for writ of certiorari with the Supreme Court of the United States in January 2019 to challenge the two rulings that were made by the Appeals Court in Oracle's favor.<sup>[83]</sup> On April 5, 2021 the Court ruled 6-2 in Google's favor, that its use of Java APIs should be considered fair use. However, the court refused to rule on the copyrightability of APIs, choosing instead to determine their ruling by considering Java's API copyrightable "purely for argument's sake."<sup>[84]</sup>



The Android operating system makes extensive use of Java-related technology

## See also

- C#
- C++
- Dalvik, used in old Android versions, replaced by non-JIT Android Runtime
- Deterministic Parallel Java
- List of Java virtual machines
- List of Java APIs
- List of JVM languages

## Comparison of Java with other languages

- Comparison of C# and Java
- Comparison of Java and C++

## References

- Binstock, Andrew (May 20, 2015). "Java's 20 Years of Innovation" (<https://www.forbes.com/sites/oracle/2015/05/20/javas-20-years-of-innovation/>). *Forbes*. Archived (<https://web.archive.org/web/20160314102242/http://www.forbes.com/sites/oracle/2015/05/20/javas-20-years-of-innovation/>) from the original on March 14, 2016. Retrieved March 18, 2016.
- "Consolidated JDK 16 Release Notes" (<https://www.oracle.com/java/technologies/javase/16-all-relnotes.html>). July 20, 2021. Retrieved July 27, 2021.
- Barbara Liskov with John Guttag (2000). *Program Development in Java - Abstraction, Specification, and Object-Oriented Design*. USA, Addison Wesley. ISBN 9780201657685.
- Chaudhary, Harry H. (July 28, 2014). "Cracking The Java Programming Interview :: 2000+ Java Interview Que/Ans" (<https://books.google.com/books?id=0rUtBAAQBAJ&pg=PAPA133>). Retrieved May 29, 2016.

5. Java 5.0 added several new language features (the enhanced for loop, autoboxing, varargs and annotations), after they were introduced in the similar (and competing) C# language. [1] (<http://www.barrycornelius.com/papers/java5/>) Archived (<https://web.archive.org/web/20110319065438/http://www.barrycornelius.com/papers/java5/>) March 19, 2011, at the Wayback Machine [2] (<http://www.levenez.com/lang/>) Archived (<https://web.archive.org/web/20060107162045/http://www.levenez.com/lang/>) January 7, 2006, at the Wayback Machine
6. Gosling, James; McGilton, Henry (May 1996). "The Java Language Environment" (<https://www.oracle.com/technetwork/java/langenv-140151.html>). Archived (<https://web.archive.org/web/20140506214653/http://www.oracle.com/technetwork/java/langenv-140151.html>) from the original on May 6, 2014. Retrieved May 6, 2014.
7. Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad. "The Java Language Specification, 2nd Edition" ([https://java.sun.com/docs/books/jls/second\\_edition/html/intro.doc.html#237601](https://java.sun.com/docs/books/jls/second_edition/html/intro.doc.html#237601)). Archived ([https://web.archive.org/web/20110805051057/http://java.sun.com/docs/books/jls/second\\_edition/html/intro.doc.html#237601](https://web.archive.org/web/20110805051057/http://java.sun.com/docs/books/jls/second_edition/html/intro.doc.html#237601)) from the original on August 5, 2011. Retrieved February 8, 2008.
8. "The A-Z of Programming Languages: Modula-3" (<https://web.archive.org/web/20090105145818/http://www.computerworld.com.au/index.php/id%3B1422447371%3Bpp%3B3%3Bfp%3B4194304%3Bfpid%3B1>). Computerworld.com.au. Archived from the original (<http://www.computerworld.com.au/index.php/id;1422447371;pp;3;fp;4194304;fpid;1>) on January 5, 2009. Retrieved June 9, 2010.
9. Niklaus Wirth stated on a number of public occasions, e.g. in a lecture at the Polytechnic Museum, Moscow in September 2005 (several independent first-hand accounts in Russian exist, e.g. one with an audio recording: Filippova, Elena (September 22, 2005). "Niklaus Wirth's lecture at the Polytechnic Museum in Moscow" (<http://www.delphikingdom.com/asp/viewitem.asp?catalogid=1155>).), that the Sun Java design team licensed the Oberon compiler sources a number of years prior to the release of Java and examined it: a (relative) compactness, type safety, garbage collection, no multiple inheritance for classes – all these key overall design features are shared by Java and Oberon.
10. Patrick Naughton cites Objective-C as a strong influence on the design of the Java programming language, stating that notable direct derivatives include Java interfaces (derived from Objective-C's protocol) and primitive wrapper classes. [3] (<http://cs.gmu.edu/~sean/stuff/java-objc.html>) Archived (<https://web.archive.org/web/20110713014816/http://cs.gmu.edu/~sean/stuff/java-objc.html>) July 13, 2011, at the Wayback Machine
11. TechMetrix Research (1999). "History of Java" (<https://web.archive.org/web/20101229090912/http://www.fscript.org/prof/javapassport.pdf>) (PDF). *Java Application Servers Report*. Archived from the original (<http://www.fscript.org/prof/javapassport.pdf>) (PDF) on December 29, 2010. "The project went ahead under the name *green* and the language was based on an old model of UCSD Pascal, which makes it possible to generate interpretive code."
12. "A Conversation with James Gosling – ACM Queue" (<http://queue.acm.org/detail.cfm?id=1017013>). Queue.acm.org. August 31, 2004. Archived (<https://web.archive.org/web/20150716194245/http://queue.acm.org/detail.cfm?id=1017013>) from the original on July 16, 2015. Retrieved June 9, 2010.
13. In the summer of 1996, Sun was designing the precursor to what is now the event model of the AWT and the JavaBeans component architecture. Borland contributed greatly to this process. We looked very carefully at Delphi Object Pascal and built a working prototype of bound method references in order to understand their interaction with the Java programming language and its APIs. White Paper About Microsoft's Delegates (<https://web.archive.org/web/20120627043929/http://java.sun.com/docs/white/delegates.html>)
14. "Chapel spec (Acknowledgements)" (<http://chapel.cray.com/spec/spec-0.98.pdf>) (PDF). Cray Inc. October 1, 2015. Archived (<https://web.archive.org/web/20160205114946/http://chapel.cray.com/spec/spec-0.98.pdf>) (PDF) from the original on February 5, 2016. Retrieved January 14, 2016.
15. "Gambas Documentation Introduction" (<http://gambaswiki.org/wiki/doc/intro?nh&l=en>). Gambas Website. Archived (<https://web.archive.org/web/20171009041815/http://gambaswiki.org/wiki/doc/intro?nh&l=en>) from the original on October 9, 2017. Retrieved October 9, 2017.

16. "Facebook Q&A: Hack brings static typing to PHP world" (<http://www.infoworld.com/article/2610885/facebook-q-a--hack-brings-static-typing-to-php-world.html>). *InfoWorld*. March 26, 2014. Archived (<http://web.archive.org/web/20150213220946/http://www.infoworld.com/article/2610885/facebook-q-a--hack-brings-static-typing-to-php-world.html>) from the original on February 13, 2015. Retrieved January 11, 2015.
17. "Write once, run anywhere?" (<http://www.computerweekly.com/Articles/2002/05/02/186793/write-once-run-anywhere.htm>). *Computer Weekly*. May 2, 2002. Retrieved July 27, 2009.
18. "1.2 Design Goals of the Java™ Programming Language" (<https://www.oracle.com/technetwork/java/intro-141325.html>). Oracle. January 1, 1999. Archived (<https://web.archive.org/web/20130123204103/http://www.oracle.com/technetwork/java/intro-141325.html>) from the original on January 23, 2013. Retrieved January 14, 2013.
19. McMillan, Robert (August 1, 2013). "Is Java Losing Its Mojo?" (<https://www.wired.com/2013/01/java-no-longer-a-favorite/>). *wired.com*. Archived (<https://web.archive.org/web/20170215115409/https://www.wired.com/2013/01/java-no-longer-a-favorite/>) from the original on February 15, 2017. Retrieved March 8, 2017. "Java is on the wane, at least according to one outfit that keeps an eye on the ever-changing world of computer programming languages. For more than a decade, it has dominated the TIOBE Programming Community Index, and is back on top – a snapshot of software developer enthusiasm that looks at things like internet search results to measure how much buzz different languages have. But lately, Java has been slipping."
20. Chan, Rosalie (January 22, 2019). "The 10 most popular programming languages, according to the 'Facebook for programmers'" (<https://www.businessinsider.de/the-10-most-popular-programming-languages-according-to-github-2018-10?op=1>). *Business Insider*. Archived (<https://archive.today/20190629083530/https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10?op=1&r=DE&IR=T>) from the original on June 29, 2019. Retrieved June 29, 2019.
21. "JavaOne 2013 Review: Java Takes on the Internet of Things" (<https://www.oracle.com/technetwork/articles/java/afterglow2013-2030343.html>). *www.oracle.com*. Archived (<https://web.archive.org/web/20160419213058/http://www.oracle.com/technetwork/articles/java/afterglow2013-2030343.html>) from the original on April 19, 2016. Retrieved June 19, 2016. Alt URL (<https://www.imarslan.com/javaone-2013-review-java-takes-on-the-internet-of-things>)
22. "Why should I uninstall older versions of Java from my system?" ([https://www.java.com/en/download/faq/remove\\_olderversions.xml](https://www.java.com/en/download/faq/remove_olderversions.xml)). Oracle. Retrieved September 9, 2016.
23. Byous, Jon (c. 1998). "Java technology: The early years" (<https://web.archive.org/web/20050420081440/http://java.sun.com/features/1998/05/birthday.html>). *Sun Developer Network*. Sun Microsystems. Archived from the original (<https://java.sun.com/features/1998/05/birthday.html>) on April 20, 2005. Retrieved April 22, 2005.
24. Object-oriented programming "The History of Java Technology" (<http://www.java.com/en/javahistory/>). *Sun Developer Network*. c. 1995. Archived (<https://web.archive.org/web/20100210225651/http://www.java.com/en/javahistory/>) from the original on February 10, 2010. Retrieved April 30, 2010.
25. Murphy, Kieron (October 4, 1996). "So why did they decide to call it Java?" (<https://www.infoworld.com/article/2077265/so-why-did-they-decide-to-call-it-java-.html>). *JavaWorld*. Retrieved 2020-07-13.
26. Kabutz, Heinz; *Once Upon an Oak* (<http://www.artima.com/weblogs/viewpost.jsp?thread=7555>) Archived (<https://web.archive.org/web/20070413072630/http://www.artima.com/weblogs/viewpost.jsp?thread=7555>) April 13, 2007, at the *Wayback Machine*. Artima. Retrieved April 29, 2007.
27. "JAVASOFT SHIPS JAVA 1.0" (<https://web.archive.org/web/20070310235103/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml>). Archived from the original (<http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml>) on March 10, 2007. Retrieved May 13, 2018.
28. *Object-oriented Programming with Java: Essentials and Applications* (<https://books.google.com/books?id=rXGMFYXFDwMC>). Tata McGraw-Hill Education. p. 34.



29. "JSG – Java Study Group" (<http://www.open-std.org/JTC1/SC22/JSG/>). *open-std.org*. Archived (<http://web.archive.org/web/20060825082008/http://www.open-std.org/JTC1/SC22/JSG/>) from the original on August 25, 2006. Retrieved August 2, 2006.
30. "Why Java™ Was – Not – Standardized Twice" (<http://www.computer.org/csdl/proceedings/hicss/2001/0981/05/09815015.pdf>) (PDF). Archived (<https://web.archive.org/web/20140113101235/http://www.computer.org/csdl/proceedings/hicss/2001/0981/05/09815015.pdf>) (PDF) from the original on January 13, 2014. Retrieved June 3, 2018.
31. "What is ECMA—and why Microsoft cares" (<https://www.zdnet.com/news/what-is-ecma-and-why-microsoft-cares/298821>). Archived (<https://web.archive.org/web/20140506215226/http://www.zdnet.com/news/what-is-ecma-and-why-microsoft-cares/298821>) from the original on May 6, 2014. Retrieved May 6, 2014.
32. "Java Community Process website" (<http://www.jcp.org/en/home/index>). Jcp.org. May 24, 2010. Archived (<https://web.archive.org/web/20060808070528/http://www.jcp.org/en/home/index>) from the original on August 8, 2006. Retrieved June 9, 2010.
33. "JAVAONE: Sun – The bulk of Java is open sourced" (<http://grnlight.net/index.php/programming-articles/115-javaone-sun-the-bulk-of-java-is-open-sourced>). GrnLight.net. Archived (<https://web.archive.org/web/20140527220942/http://grnlight.net/index.php/programming-articles/115-javaone-sun-the-bulk-of-java-is-open-sourced>) from the original on May 27, 2014. Retrieved May 26, 2014.
34. "Sun's Evolving Role as Java Evangelist" (<http://onjava.com/pub/a/onjava/2002/04/17/evangelism.html>). O'Reilly Media. Archived (<https://web.archive.org/web/20100915162748/http://onjava.com/pub/a/onjava/2002/04/17/evangelism.html>) from the original on September 15, 2010. Retrieved August 2, 2009.
35. "Oracle and Java" (<https://web.archive.org/web/20100131091008/http://www.oracle.com/us/technologies/java/index.html>). *oracle.com*. Oracle Corporation. Archived from the original (<https://www.oracle.com/us/technologies/java/index.html>) on January 31, 2010. Retrieved August 23, 2010. "Oracle has been a leading and substantive supporter of Java since its emergence in 1995 and takes on the new role as steward of Java technology with a relentless commitment to fostering a community of participation and transparency."
36. Gosling, James (April 9, 2010). "Time to move on..." ([https://web.archive.org/web/20101105031239/http://nighthacks.com/roller/jag/entry/time\\_to\\_move\\_on](https://web.archive.org/web/20101105031239/http://nighthacks.com/roller/jag/entry/time_to_move_on)) *On a New Road*. Archived from the original ([http://nighthacks.com/roller/jag/entry/time\\_to\\_move\\_on](http://nighthacks.com/roller/jag/entry/time_to_move_on)) on November 5, 2010. Retrieved November 16, 2011.
37. Topic, Dalibor. "Moving to a Plugin-Free Web" ([https://blogs.oracle.com/java-platform-group/entry/moving\\_to\\_a\\_plugin\\_free](https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free)). Archived ([https://web.archive.org/web/20160316164325/https://blogs.oracle.com/java-platform-group/entry/moving\\_to\\_a\\_plugin\\_free](https://web.archive.org/web/20160316164325/https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free)) from the original on March 16, 2016. Retrieved March 15, 2016.
38. "Learn About Java Technology" (<http://www.java.com/en/about/>). Oracle. Archived (<https://web.archive.org/web/20111124090716/http://www.java.com/en/about/>) from the original on November 24, 2011. Retrieved November 21, 2011.
39. "Oracle Java SE Support Roadmap" (<https://www.oracle.com/java/technologies/java-se-support-roadmap.html>). Oracle. May 13, 2020. Retrieved December 11, 2020.
40. "JAVASOFT SHIPS JAVA 1.0" (<https://web.archive.org/web/20070310235103/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml>). *sun.com*. Archived from the original (<http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml>) on March 10, 2007. Retrieved February 5, 2008.
41. Chander, Sharat. "Introducing Java SE 11" (<https://blogs.oracle.com/java-platform-group/introducing-java-se-11>). *oracle.com*. Archived (<https://web.archive.org/web/20180926093144/https://blogs.oracle.com/java-platform-group/introducing-java-se-11>) from the original on September 26, 2018. Retrieved September 26, 2018.
42. "The Arrival of Java 15!" (<https://blogs.oracle.com/java-platform-group/the-arrival-of-java-15>). Oracle. September 15, 2020. Retrieved September 15, 2020.

43. "Java Card Overview" (<https://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html>). *Oracle Technology Network*. Oracle. Archived (<https://web.archive.org/web/20150107034738/http://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html>) from the original on January 7, 2015. Retrieved December 18, 2014.
44. "Java Platform, Micro Edition (Java ME)" (<https://www.oracle.com/technetwork/java/embedded/javame/index.html>). *Oracle Technology Network*. Oracle. Archived (<https://web.archive.org/web/20150104210546/http://www.oracle.com/technetwork/java/embedded/javame/index.html>) from the original on January 4, 2015. Retrieved December 18, 2014.
45. "Java SE" (<https://www.oracle.com/technetwork/java/javase/overview/index.html>). *Oracle Technology Network*. Oracle. Archived (<https://web.archive.org/web/20141224184532/http://www.oracle.com/technetwork/java/javase/overview/index.html>) from the original on December 24, 2014. Retrieved December 18, 2014.
46. "Java Platform, Enterprise Edition (Java EE)" (<https://www.oracle.com/technetwork/java/javaee/overview/index.html>). *Oracle Technology Network*. Oracle. Archived (<https://web.archive.org/web/20141217155326/http://www.oracle.com/technetwork/java/javaee/overview/index.html>) from the original on December 17, 2014. Retrieved December 18, 2014.
47. "Is the JVM (Java Virtual Machine) platform dependent or platform independent? What is the advantage of using the JVM, and having Java be a translated language?" (<http://www.programmerinterview.com/index.php/java-questions/jvm-platform-dependent/>). Programmer Interview. Archived (<https://web.archive.org/web/20150119144223/http://www.programmerinterview.com/index.php/java-questions/jvm-platform-dependent/>) from the original on January 19, 2015. Retrieved January 19, 2015.
48. Jelovic, Dejan. "Why Java will always be slower than C++" ([https://web.archive.org/web/200802111923/http://www.jelovic.com/articles/why\\_java\\_is\\_slow.htm](https://web.archive.org/web/200802111923/http://www.jelovic.com/articles/why_java_is_slow.htm)). Archived from the original ([http://www.jelovic.com/articles/why\\_java\\_is\\_slow.htm](http://www.jelovic.com/articles/why_java_is_slow.htm)) on February 11, 2008. Retrieved February 15, 2008.
49. Google. "Loop Recognition in C++/Java/Go/Scala" (<https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf>) (PDF). Retrieved July 12, 2012.
50. "Symantec's Just-In-Time Java Compiler To Be Integrated into Sun JDK 1.1" ([http://www.symantec.com/about/news/release/article.jsp?prid=19970407\\_03](http://www.symantec.com/about/news/release/article.jsp?prid=19970407_03)). Archived ([https://web.archive.org/web/20100628171748/http://www.symantec.com/about/news/release/article.jsp?prid=19970407\\_03](https://web.archive.org/web/20100628171748/http://www.symantec.com/about/news/release/article.jsp?prid=19970407_03)) from the original on June 28, 2010. Retrieved August 1, 2009.
51. Salcic, Zoran; Park, Heejong; Teich, Jürgen; Malik, Avinash; Nadeem, Muhammad (July 22, 2017). "Noc-HMP: A Heterogeneous Multicore Processor for Embedded Systems Designed in SystemJ". *ACM Transactions on Design Automation of Electronic Systems*. **22** (4): 73. doi:10.1145/3073416 (<https://doi.org/10.1145%2F3073416>). ISSN 1084-4309 (<https://www.worldcat.org/issn/1084-4309>). S2CID 11150290 (<https://api.semanticscholar.org/CorpusID:11150290>).
52. "NullPointerException" (<http://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html>). Oracle. Archived (<https://web.archive.org/web/20140506214735/http://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html>) from the original on May 6, 2014. Retrieved May 6, 2014.
53. "Exceptions in Java" (<http://www.artima.com/designtechniques/exceptions.html>). Artima.com. Archived (<https://web.archive.org/web/20090121152332/http://www.artima.com/designtechniques/exceptions.html>) from the original on January 21, 2009. Retrieved August 10, 2010.
54. "Java HotSpot™ Virtual Machine Performance Enhancements" (<https://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html>). Oracle.com. Archived (<https://web.archive.org/web/20170529071720/http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html>) from the original on May 29, 2017. Retrieved April 26, 2017.
55. "Operator Overloading (C# vs Java)" (<http://msdn.microsoft.com/en-us/library/ms228498%28v=vs.90%29.aspx>). *C# for Java Developers*. Microsoft. Archived ([https://web.archive.org/web/20150107190007/http://msdn.microsoft.com/en-us/library/ms228498\(v=vs.90\).aspx](https://web.archive.org/web/20150107190007/http://msdn.microsoft.com/en-us/library/ms228498(v=vs.90).aspx)) from the original on January 7, 2015. Retrieved December 10, 2014.

56. "Multiple Inheritance of State, Implementation, and Type" (<https://docs.oracle.com/javase/tutorial/java/landl/multipleinheritance.html>). *The Java™ Tutorials*. Oracle. Archived (<https://web.archive.org/web/20141109034520/https://docs.oracle.com/javase/tutorial/java/landl/multipleinheritance.html>) from the original on November 9, 2014. Retrieved December 10, 2014.
57. "Lesson: A Closer Look at the Hello World Application" (<https://docs.oracle.com/javase/tutorial/getStarted/application/index.html>). *The Java™ Tutorials > Getting Started*. Oracle Corporation. Archived (<https://web.archive.org/web/20110317072804/http://download.oracle.com/javase/tutorial/getStarted/application/index.html>) from the original on March 17, 2011. Retrieved April 14, 2011.
58. "Deprecated APIs, Features, and Options" (<https://www.oracle.com/technetwork/java/javase/9-deprecated-features-3745636.html#JDK-8074165>). *www.oracle.com*. Retrieved May 31, 2019.
59. "Applet (Java Platform SE 7 )" (<https://docs.oracle.com/javase/7/docs/api/java/applet/Applet.html>). *docs.oracle.com*. Retrieved May 1, 2020.
60. "What Is a JSP Page? - The Java EE 5 Tutorial" (<https://docs.oracle.com/javaee/5/tutorial/doc/bnagy.html>). *docs.oracle.com*. Retrieved May 1, 2020.
61. "Trail: Creating a GUI With JFC/Swing (The Java™ Tutorials)" (<https://docs.oracle.com/javase/tutorial/uiswing/index.html>). *docs.oracle.com*. Retrieved May 1, 2020.
62. "Removed from JDK 11, JavaFX 11 arrives as a standalone module" (<https://www.infoworld.com/article/3305073/removed-from-jdk-11-javafx-11-arrives-as-a-standalone-module.html>). Retrieved October 13, 2020.
63. "Getting Started with JavaFX: Hello World, JavaFX Style | JavaFX 2 Tutorials and Documentation" ([https://docs.oracle.com/javafx/2/get\\_started/hello\\_world.htm](https://docs.oracle.com/javafx/2/get_started/hello_world.htm)). *docs.oracle.com*. Retrieved May 1, 2020.
64. "Java and Scala's Type Systems are Unsound" (<https://raw.githubusercontent.com/namin/unsound/master/doc/unsound-oopsla16.pdf>) (PDF). Archived (<https://web.archive.org/web/20161128174902/https://raw.githubusercontent.com/namin/unsound/master/doc/unsound-oopsla16.pdf>) (PDF) from the original on November 28, 2016. Retrieved February 20, 2017.
65. Arnold, Ken. "Generics Considered Harmful" ([https://web.archive.org/web/20071010002142/http://weblogs.java.net/blog/arnold/archive/2005/06/generics\\_considered\\_harmful\\_1.html](https://web.archive.org/web/20071010002142/http://weblogs.java.net/blog/arnold/archive/2005/06/generics_considered_harmful_1.html)). *java.net*. Archived from the original ([https://weblogs.java.net/blog/arnold/archive/2005/06/generics\\_considered\\_harmful\\_1.html](https://weblogs.java.net/blog/arnold/archive/2005/06/generics_considered_harmful_1.html)) on October 10, 2007. Retrieved September 10, 2015. More comments to the original article available at earlier archive snapshots.
66. Jelovic, Dejan. "Why Java Will Always Be Slower than C++" ([https://web.archive.org/web/20080211111923/http://www.jelovic.com/articles/why\\_java\\_is\\_slow.htm](https://web.archive.org/web/20080211111923/http://www.jelovic.com/articles/why_java_is_slow.htm)). *www.jelovic.com*. Archived from the original ([http://www.jelovic.com/articles/why\\_java\\_is\\_slow.htm](http://www.jelovic.com/articles/why_java_is_slow.htm)) on February 11, 2008. Retrieved October 17, 2012.
67. Owens, Sean R. "Java and unsigned int, unsigned short, unsigned byte, unsigned long, etc. (Or rather, the lack thereof)" (<https://web.archive.org/web/20090220171410/http://darksleep.com/player/JavaAndUnsignedTypes.html>). Archived from the original (<http://darksleep.com/player/JavaAndUnsignedTypes.html>) on February 20, 2009. Retrieved July 4, 2011.
68. Kahan, William. "How Java's Floating-Point Hurts Everyone Everywhere" (<http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf>) (PDF). Electrical Engineering & Computer Science, University of California at Berkeley. Archived (<https://web.archive.org/web/20120905004527/http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf>) (PDF) from the original on September 5, 2012. Retrieved June 4, 2011.
69. "Have you checked the Java?" (<https://web.archive.org/web/20120921140402/http://blogs.technet.com/b/mmpc/archive/2010/10/18/have-you-checked-the-java.aspx>). Archived from the original (<http://blogs.technet.com/b/mmpc/archive/2010/10/18/have-you-checked-the-java.aspx>) on September 21, 2012. Retrieved December 23, 2011.
70. Cadenhead, Rogers (November 20, 2017), *Understanding How Java Programs Work* (<http://www.informit.com/articles/article.aspx?p=2832404&seqNum=4>), retrieved March 26, 2019

71. Woolf, Nicky (May 26, 2016). "Google wins six-year legal battle with Oracle over Android code copyright" (<https://www.theguardian.com/technology/2016/may/26/google-wins-copyright-lawsuit-oracle-java-code>). *The Guardian*. ISSN 0261-3077 (<https://www.worldcat.org/issn/0261-3077>). Retrieved March 26, 2019.
72. "Collections Framework Overview" (<http://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>). *Java Documentation*. Oracle. Archived (<https://web.archive.org/web/20141231132540/http://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>) from the original on December 31, 2014. Retrieved December 18, 2014.
73. "Java™ Security Overview" (<http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html>). *Java Documentation*. Oracle. Archived (<https://web.archive.org/web/20150103045031/http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html>) from the original on January 3, 2015. Retrieved December 18, 2014.
74. "Trail: Internationalization" (<http://docs.oracle.com/javase/tutorial/i18n/>). *The Java™ Tutorials*. Oracle. Archived (<https://web.archive.org/web/20141231053232/http://docs.oracle.com/javase/tutorial/i18n/>) from the original on December 31, 2014. Retrieved December 18, 2014.
75. "How to Write Doc Comments for the Javadoc Tool" (<https://www.oracle.com/technetwork/articles/java/index-137868.html>). *Oracle Technology Network*. Oracle. Archived (<https://web.archive.org/web/20141218182906/http://www.oracle.com/technetwork/articles/java/index-137868.html>) from the original on December 18, 2014. Retrieved December 18, 2014.
76. Niccolai, James (January 24, 2001). "Sun, Microsoft settle Java lawsuit" (<https://www.infoworld.com/article/2074908/sun-microsoft-settle-java-lawsuit.html>). *JavaWorld*. IDG News Service. Retrieved 2020-07-13.
77. van Gorp, Jilles (November 13, 2007). "Google Android: Initial Impressions and Criticism" ([http://www.javalobby.org/nl/archive/jlnews\\_20071113o.html](http://www.javalobby.org/nl/archive/jlnews_20071113o.html)). *Javalobby*. Retrieved March 7, 2009. *"Frankly, I don't understand why Google intends to ignore the vast amount of existing implementation out there. It seems like a bad case of "not invented here" to me. Ultimately, this will slow adoption. There are already too many Java platforms for the mobile world and this is yet another one"*
78. Mullin, Joe. "Google guilty of infringement in Oracle trial; future legal headaches loom" (<https://arstechnica.com/tech-policy/news/2012/05/jury-rules-google-violated-copyright-law-google-moves-for-mistrial.ars>). *Law & Disorder*. Ars Technica. Archived (<https://web.archive.org/web/20120508134916/http://arstechnica.com/tech-policy/news/2012/05/jury-rules-google-violated-copyright-law-google-moves-for-mistrial.ars>) from the original on May 8, 2012. Retrieved May 8, 2012.
79. Mullin, Joe (May 31, 2012). "Google wins crucial API ruling, Oracle's case decimated" (<https://arstechnica.com/tech/2012/05/google-wins-crucial-api-ruling-oracles-case-decimated/>). *Ars Technica*. Archived (<https://web.archive.org/web/20170312065520/https://arstechnica.com/tech-policy/2012/05/google-wins-crucial-api-ruling-oracles-case-decimated/>) from the original on March 12, 2017. Retrieved June 1, 2012.
80. Rosenblatt, Seth (May 9, 2014). "Court sides with Oracle over Android in Java patent appeal" (<http://www.cnet.com/news/court-sides-with-oracle-over-android-in-java-patent-appeal/>). *CNET*. Archived (<https://web.archive.org/web/20140510203805/http://www.cnet.com/news/court-sides-with-oracle-over-android-in-java-patent-appeal/>) from the original on May 10, 2014. Retrieved May 10, 2014.
81. Mullin, Joe (May 26, 2016). "Google beats Oracle—Android makes "fair use" of Java APIs" (<https://arstechnica.com/tech-policy/2016/05/google-wins-trial-against-oracle-as-jury-finds-android-is-fair-use/>). *Ars Technica*. Archived (<https://web.archive.org/web/20170120164551/http://arstechnica.com/tech-policy/2016/05/google-wins-trial-against-oracle-as-jury-finds-android-is-fair-use/>) from the original on January 20, 2017. Retrieved May 26, 2016.
82. Farivar, Cyrus (March 27, 2018). "'Google's use of the Java API packages was not fair,' appeals court rules" (<https://arstechnica.com/tech-policy/2018/03/googles-use-of-the-java-api-packages-was-not-fair-appeals-court-rules/>). *Ars Technica*. Retrieved August 6, 2019.

83. Lee, Timothy (April 23, 2019). "Google asks Supreme Court to overrule disastrous ruling on API copyrights" (<https://arstechnica.com/tech-policy/2019/01/google-asks-supreme-court-to-overrule-disastrous-ruling-on-api-copyrights/>). *Ars Technica*. Retrieved April 23, 2019.
84. *Google LLC v. Oracle America, Inc* 593 U. S. \_\_\_\_ (2021) ([https://www.supremecourt.gov/opinions/20pdf/18-956\\_d18f.pdf](https://www.supremecourt.gov/opinions/20pdf/18-956_d18f.pdf))





## Works cited

---

- Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). *The Java® Language Specification* (<https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf>) (PDF) (Java SE 8 ed.).
- Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). *The Java Language Specification* (<https://java.sun.com/docs/books/jls/index.html>) (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.
- Lindholm, Tim; Yellin, Frank (1999). *The Java Virtual Machine Specification* (<https://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html>) (2nd ed.). Addison-Wesley. ISBN 0-201-43294-3.

## External links

---

-  The dictionary definition of *Java* at Wiktionary
  -  Media related to *Java* at Wikimedia Commons
  -  *Java Programming* at Wikibooks
  -  Learning materials related to *Java* at Wikiversity
- 

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Java\\_\(programming\\_language\)&oldid=1035737849](https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=1035737849)"

---

**This page was last edited on 27 July 2021, at 11:41 (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.