SUBJECT-JAVA PROGRAMMING

[bca –Iind yr]

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 programmers *write once, run anywhere* (WORA),[16] meaning that compiled Java code can run on all platforms that support Java without the need to recompile.[17] 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.

Java gained popularity shortly after its release, and has been a very popular programming language since then.[18] Java was the third most popular programming language in 2022 according to GitHub.[19] Although still widely popular, there has been a gradual decline in use of Java in recent years with other languages using JVM gaining popularity.[20]

Java was originally developed by James Gosling at Sun Microsystems. It was released in May 1995 as a core component of Sun's 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As of September 2024, Java 23 is the latest version (Java 22, and 20 are no longer maintained). Java 8, 11, 17, and 21 are previous LTS versions still officially supported.

History

Duke, the Java mascot

James Gosling, the creator of Java, in 2008

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.[21] Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time.[22] 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.[23] Gosling designed Java with a C/C++-style syntax that system and application programmers would find familiar.[24]

Sun Microsystems released the first public implementation as Java 1.0 in 1996.[25] 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.[26] 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.

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.[27][28][29] Java remains a *de facto* standard, controlled through the Java Community Process.[30] 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.[31]

Sun's vice-president Rich Green said that Sun's ideal role with regard to Java was as an *evangelist*.[32] 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.[33] 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.[34]

In January 2016, Oracle announced that Java run-time environments based on JDK 9 will discontinue the browser plugin.[35]

Java software runs on everything from laptops to data centers, game consoles to scientific supercomputers.[36]

Oracle (and others) highly recommend uninstalling outdated and unsupported versions of Java, due to unresolved security issues in older versions.[37]

Principles

There were five primary goals in creating the Java language:[17]

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

.[38]

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 such as Adoptium continue to offer free builds of OpenJDK's long-term support (LTS) versions. These builds may include additional security patches and bug fixes.[39]

Major release versions of Java, along with their release dates:

Version	Date
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8 (LTS)	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11 (LTS)	September 25, 2018[41]
Java SE 12	March 19, 2019

Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020[42]
Java SE 16	March 16, 2021
Java SE 17 (LTS)	September 14, 2021
Java SE 18	March 22, 2022
Java SE 19	September 20, 2022
Java SE 20	March 21, 2023
Java SE 21 (LTS)	September 19, 2023[43]
Java SE 22	March 19, 2024
Java SE 23	September 17, 2024

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.[44]
- Java Platform, Micro Edition (Java ME) – targeting environments with limited resources.[45]
- Java Platform, Standard Edition (Java SE) – targeting workstation environments.[46]
- Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.[47]

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 device for standalone Java applications or 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's Hotspot compiler is actually two compilers in one; and with GraalVM (included in e.g. Java 11, but removed as of Java 16) allowing tiered compilation.[48] Java itself is platform-independent and is adapted to the particular platform it is to run on by a Java virtual machine (JVM), which translates the Java bytecode into the platform's machine language.[49]

Performance

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++.[50][51] However, Java programs' execution speed improved significantly with the introduction of just-in-time compilation in 1997/1998 for Java 1.1,[52] 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,[53] 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.[54] If methods for a non-existent object are called, a null pointer exception is thrown.[55][56]

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.[54] 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. Garbage collection does not prevent logical memory leaks, i.e. those where the memory is still referenced but never used.[54]

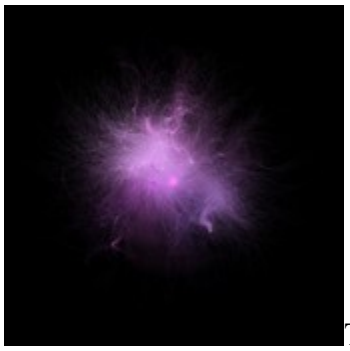
Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Explicit memory management is not possible in Java.

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.[57] 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 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.

 This dependency graph of the Java Core classes was created with jdeps and Gephi.

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.[17] 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[58] or multiple inheritance for classes, though multiple inheritance is supported for interfaces.[59]

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

The following is a simple example of a "Hello, World!" program that writes a message to the standard output:

```
public class Example {
    public static void main(string[] args) {
        System.out.println("Hello World!");
    }
}
```

Applet

Java applets are programs 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.[60][61]

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.[62]

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.[63]

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.[64] 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.[65]

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 (`ClassCastException`s) from occurring, it would be type safe.

In 2016, the type system of Java was proven unsound in that it is possible to use generics to construct classes and methods that allow assignment of an instance one class to a variable of another unrelated class. Such code is accepted by the compiler, but fails at run time with a class cast exception.[66]

: Criticism of Java

Criticisms directed at Java include the implementation of generics,[67] speed,[50] the handling of unsigned numbers,[68] the implementation of floating-point arithmetic,[69] and a history of security vulnerabilities in the primary Java VM implementation HotSpot.[70] Developers have criticized the complexity and verbosity of the Java Persistence API (JPA), a standard part of Java EE. This has led to increased adoption of higher-level abstractions like Spring Data JPA, which aims to simplify database operations and reduce boilerplate code. The growing popularity of such frameworks suggests limitations in the standard JPA implementation's ease-of-use for modern J*:*
[71]

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.[72] 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.[73] The class library contains features such as:

- The core libraries, which include:
  - Input/output (I/O or IO)[74] and non-blocking I/O (NIO), or IO/NIO[75]
  - Networking[76] (new user agent (HTTP client) since Java 11[77])
  - Reflective programming (reflection)
  - Concurrent computing (concurrency)[74]
  - 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[78]
  - XML Processing (Parsing, Transforming, Validating) libraries
  - Security[79]

    - Internationalization and localization libraries[80]
- 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
    - Java remote method invocation (RMI) and Common Object Request Broker Architecture (CORBA) for distributed application development
    - Java Management Extensions (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 delimited by `/*` and `*/`, and single-line comments start with `//`.[81]

Implementations

Oracle Corporation owns the official implementation of the Java SE platform, due to its 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 Windows, 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 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 Java remote method invocation (RMI) or Java Native Interface (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.[82] 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.[83] 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).

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.[84] District Judge William Alsup ruled on May 31, 2012, that APIs cannot be copyrighted,[85] but this was reversed by the United States Court of Appeals for the Federal Circuit in May 2014.[86] 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.[87] 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.[88] 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.[89] 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."[90]