Java (programming language)

From Wikipedia, the free encyclopedia

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented,[14] and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA),[15] meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.[16] Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use,[17][18][19][20] particularly for client-server web applications, with a reported 9 million developers.[21] Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licences. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java (bytecode compiler), GNU Classpath (standard libraries), and IcedTea-Web (browser plugin for applets).

The latest version is Java 8 Update 121 which is the only version currently supported for free by Oracle, although earlier versions are supported both by Oracle and other companies on a commercial basis.

Java

Paradigm	Multi-paradigm: Object-oriented (class-based), structured, imperative, generic, reflective, concurrent
Designed by	James Gosling
Developer	Sun Microsystems (now owned by Oracle Corporation)
First appeared	May 23, 1995[1]
Typing discipline	Static, strong, safe, nominative, manifest
License	GNU General Public License, Java Community Process
Filename extensions	.java, .class, .jar
Website	www.oracle.com/java/ (https://www.oracle.com/java/)

Major implementations

OpenJDK, GNU Compiler for Java (GCJ), many others

Dialects

Generic Java, Pizza

Influenced by

Ada 83, C++,[2] C#,[3] Eiffel,[4] Generic Java, Mesa,[5] Modula-3,[6] Oberon,[7] Objective-C,[8] UCSD Pascal,[9][10] Object Pascal[11]

Influenced

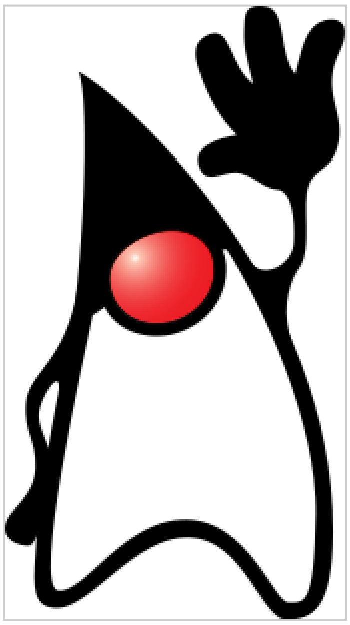
Contents

Ada 2005, BeanShell, C#, Chapel,[12] Clojure, ECMAScript, Fantom, Groovy, Hack,[13] Haxe, J#, JavaScript, Kotlin, PHP, Python, Scala, Seed7, Vala

Java Programming at Wikibooks

History

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.[22] Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time.[23] 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.[24] Gosling designed Java with a C/C++-style syntax that system and application programmers would find familiar.[25]


Duke, the Java mascot

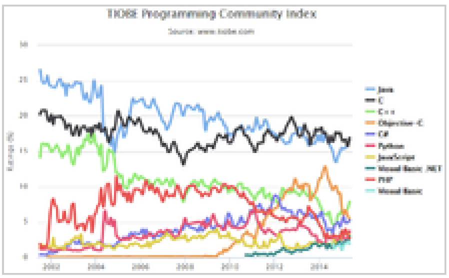
Sun Microsystems released the first public implementation as Java 1.0 in 1995.[26] It promised "Write Once, Run Anywhere" (WORA), providing no-cost run-times on popular platforms. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions. Major web browsers soon incorporated the ability to run *Java applets* within web pages, and Java quickly became popular, while mostly outside of browsers, that wasn't the original plan. In January 2016, Oracle announced that Java runtime environments based on JDK 9 will discontinue the browser plugin.[27] The Java 1.0 compiler was re-written in Java by Arthur van Hoff to comply strictly with the Java 1.0 language specification.[28] 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.[29][30][31] Java remains a *de facto* standard, controlled through the Java Community Process.[32] 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 GNU General Public License (GPL). 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.[33]

James Gosling, the creator of Java (2008)

Sun's vice-president Rich Green said that Sun's ideal role with regard to Java was as an "evangelist".[34] 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".[35] This did not prevent Oracle from filing a lawsuit against Google shortly after that for using Java inside the Android SDK (see Google section below). Java software runs on everything from laptops to data centers, game consoles to scientific supercomputers.[36] On April 2, 2010, James Gosling resigned from Oracle.[37]



The TIOBE programming language popularity index graph from 2002 to 2015. Over the course of a decade Java (blue) and C (black) competing for the top position.

Principles

There were five primary goals in the creation of the Java language:[16]

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 2015, only Java 8 is officially supported. Major release versions of Java, along with their release dates:

- JDK 1.0 (January 23, 1996)[38]
- 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)

Practices

Java platform

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 runtime 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 own 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 makes interpreted programs almost always run more slowly than native executables. However, just-in-time (JIT) compilers that compile bytecodes 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 for it, which translates the Java bytecode into the platform's machine language.[39]

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 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 organization, 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.

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.[40] 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.

Performance

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++.[41][42] However, Java programs' execution speed improved significantly with the introduction of just-in-time compilation in 1997/1998 for Java 1.1,[43] 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 the default for Sun's 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 Java 1.6.

Some platforms offer direct hardware support for Java; there are microcontrollers that can run Java in hardware instead of a software Java virtual machine, 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 nonexistent object are called, a "null pointer exception" is thrown.[44][45]

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 and/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 and unsigned integers (usually long integers) can be used interchangeably. 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. By default, HotSpot uses the parallel scavenge garbage collector. However, there are also several other garbage collectors that can be used to manage the heap. For 90% of applications in Java, the Concurrent Mark-Sweep (CMS) garbage collector is sufficient.[46] Oracle aims to replace CMS with the Garbage-First collector (G1).[47]

Syntax

The syntax of Java is largely influenced by C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object-oriented language.[16] 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 printf() method).

Unlike C++, Java does not support operator overloading[48] or multiple inheritance for *classes*, though multiple inheritance is supported for interfaces.[49] This simplifies the language and aids in preventing potential errors and anti-pattern design.

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.

Example:

```
// 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.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.
 */
public class FibCalculator extends Fibonacci implements Calculator {
    private static Map<Integer, Integer> memoized = new HashMap<Integer, Integer>();

    /*
     * 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;
        }
    }
}
```

"Hello world" example

The traditional "Hello, world!" program can be written in Java as:[50]

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

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 named `HelloWorldApp.class`. 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 other than public access and any number of public inner classes. When the source file contains multiple classes, make one class "public" 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`** and **`protected`**.

The keyword **`static`** 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/8/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 (the `args` parameter 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 (https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)` 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/8/docs/api/java/lang/System.html)`** class defines a public static field called **`out (https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#out)`**. The `out` object is an instance of the `PrintStream (https://docs.oracle.com/javase/8/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/8/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.

Special classes

Applet

Java applets are programs that are embedded in other applications, typically in a Web page displayed in a web browser.

```java
// Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {
    public void paintComponent(final Graphics g) {
        g.drawString("Hello, world!", 65, 95);
    }
}
```

The **import** statements direct the Java compiler to include the **javax.swing.JApplet (https://docs.oracle.com/javase/8/docs/api/javax/swing/JApplet.html)** and **java.awt.Graphics (https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html)** classes in the compilation. The import statement allows these classes to be referenced in the source code using the *simple class name* (i.e. `JApplet`) instead of the *fully qualified class name* (*FQCN*, i.e. `javax.swing.JApplet`).

The `Hello` class **extends** (subclasses) the **JApplet** (Java Applet) class; the `JApplet` class provides the framework for the host application to display and control the lifecycle of the applet. The `JApplet` class is a JComponent (Java Graphical Component) which provides the applet with the capability to display a graphical user interface (GUI) and respond to user events.

The `Hello` class overrides the **paintComponent(Graphics) (https://docs.oracle.com/javase/8/docs/api/java/awt/Container.html#paint(java.awt.Graphics))** method (additionally indicated with the annotation, supported as of JDK 1.5, `Override`) inherited from the `Container (https://docs.oracle.com/javase/8/docs/api/java/awt/Container.html)` superclass to provide the code to display the applet. The `paintComponent()` method is passed a **Graphics** object that contains the graphic context used to display the applet. The `paintComponent()` method calls the graphic context **drawString(String, int, int) (https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html#drawString(java.lang.String,%20int,%20int))** method to display the **"Hello, world!"** string at a pixel offset of (**65, 95**) from the upper-left corner in the applet's display.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<!-- Hello.html -->
<html>
    <head>
        <title>Hello World Applet</title>
    </head>
    <body>
        <applet code="Hello.class" width="200" height="200">
        </applet>
    </body>
</html>
```

An applet is placed in an HTML document using the **<applet>** HTML element. The `applet` tag has three attributes set: **code="Hello"** specifies the name of the `JApplet` class and **width="200" height="200"** sets the pixel width and height of the applet. Applets may also be embedded in HTML using either the `object` or `embed` element,[51] although support for these elements by web browsers is inconsistent.[52] However, the `applet` tag is deprecated, so the `object` tag is preferred where supported.

The host application, typically a Web browser, instantiates the **Hello** applet and creates an `AppletContext (https://docs.oracle.com/javase/8/docs/api/java/applet/AppletContext.html)` for the applet. Once the applet has initialized itself, it is added to the AWT display hierarchy. The `paintComponent()` method is called by the AWT event dispatching thread whenever the display needs the applet to draw itself.

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 (typically HTML pages) to requests (typically HTTP requests) from clients. A servlet can almost be thought of as an applet that runs on the server side—without a face.

```
// Hello.java
import java.io.*;
import javax.servlet.*;

public class Hello extends GenericServlet {
    public void service(final ServletRequest request, final ServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        final PrintWriter pw = response.getWriter();
        try {
            pw.println("Hello, world!");
        } finally {
            pw.close();
        }
    }
}
```

The **`import`** statements direct the Java compiler to include all the public classes and interfaces from the **`java.io (https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html)`** and **`javax.servlet (https://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html)`** packages in the compilation. Packages make Java well suited for large scale applications.

The **`Hello`** class **`extends`** the **`GenericServlet (https://docs.oracle.com/javaee/7/api/javax/servlet/GenericServlet.html)`** class; the `GenericServlet` class provides the interface for the server to forward requests to the servlet and control the servlet's lifecycle.

The `Hello` class overrides the **`service(ServletRequest, ServletResponse) (https://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html#service(javax.servlet.ServletRequest,javax.servlet.ServletResponse))`** method defined by the `Servlet (https://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html)` interface to provide the code for the service request handler. The `service()` method is passed: a **`ServletRequest (https://docs.oracle.com/javaee/7/api/javax/servlet/ServletRequest.html)`** object that contains the request from the client and a **`ServletResponse (https://docs.oracle.com/javaee/7/api/javax/servlet/ServletResponse.html)`** object used to create the response returned to the client. The `service()` method declares that it **`throws`** the exceptions `ServletException (https://docs.oracle.com/javaee/7/api/javax/servlet/ServletException.html)` and `IOException (https://docs.oracle.com/javase/8/docs/api/java/io/IOException.html)` if a problem prevents it from responding to the request.

The **`setContentType(String) (https://docs.oracle.com/javaee/7/api/javax/servlet/ServletResponse.html#setContentType(java.lang.String))`** method in the response object is called to set the MIME content type of the returned data to **"text/html"**. The **`getWriter() (https://docs.oracle.com/javaee/7/api/javax/servlet/ServletResponse.html#getWriter())`** method in the response returns a **`PrintWriter (https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html)`** object that is used to write the data that is sent to the client. The **`println(String) (https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html#println(java.lang.String))`** method is called to write the **"Hello, world!"** string to the response and then the **`close() (https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html#close())`** method is called to close the print writer, which causes the data that has been written to the stream to be returned to the client.

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.

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.

This example Swing application creates a single window with "Hello, world!" inside:

```
// Hello.java (Java SE 5)
import javax.swing.*;

public class Hello extends JFrame {
    public Hello() {
        super("hello");
        super.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        super.add(new JLabel("Hello, world!"));
        super.pack();
        super.setVisible(true);
    }

    public static void main(final String[] args) {
        new Hello();
    }
}
```

The first **`import`** includes all the public classes and interfaces from the **`javax.swing (https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html)`** package.

The **`Hello`** class **`extends`** the **`JFrame (https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html)`** class; the `JFrame` class implements a window with a title bar and a close control.

The **`Hello()`** constructor initializes the frame by first calling the superclass constructor, passing the parameter `"hello"`, which is used as the window's title. It then calls the **`setDefaultCloseOperation(int) (https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html#setDefaultCloseOperation(int))`** method inherited from `JFrame` to set the default operation when the close control on the title bar is selected to **`WindowConstants.EXIT_ON_CLOSE (https://docs.oracle.com/javase/8/docs/api/javax/swing/WindowConstants.html#EXIT_ON_CLOSE)`** – this causes the `JFrame` to be disposed of when the frame is closed (as opposed to merely hidden), which allows the Java virtual machine to exit and the program to terminate. Next, a **`JLabel (https://docs.oracle.com/javase/8/docs/api/javax/swing/JLabel.html)`** is created for the string **"Hello, world!"** and the **`add(Component) (https://docs.oracle.com/javase/8/docs/api/java/awt/Container.html#add(java.awt.Component))`** method inherited from the `Container (https://docs.oracle.com/javase/8/docs/api/java/awt/Container.html)` superclass is called to add the label to the frame. The **`pack() (https://docs.oracle.com/javase/8/docs/api/java/awt/Window.html#pack())`** method inherited from the `Window (https://docs.oracle.com/javase/8/docs/api/java/awt/Window.html)` superclass is called to size the window and lay out its contents.

The **`main()`** method is called by the Java virtual machine when the program starts. It instantiates a new **`Hello`** frame and causes it to be displayed by calling the **`setVisible(boolean) (https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html#setVisible(boolean))`** method inherited from the `Component (https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html)` superclass with the boolean parameter **`true`**. Once the frame is displayed, exiting the `main` method does not cause the program to terminate because the AWT event dispatching thread remains active until all of the Swing top-level windows have been disposed.

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 was shown not to be safe at all, it was proven unsound.[53]

Criticism

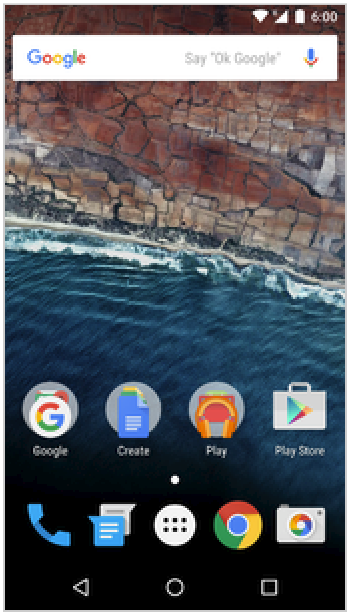
Criticisms directed at Java include the implementation of generics,[54] speed,[55] the handling of unsigned numbers,[56] the implementation of floating-point arithmetic,[57] and a history of security vulnerabilities in the primary Java VM implementation HotSpot.[58]

Use outside of 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.

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. 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.



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

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.[59] District Judge William Haskell Alsup ruled on May 31, 2012, that APIs cannot be copyrighted,[60] but this was reversed by the United States Court of Appeals for the Federal Circuit in May 2014.[61] 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.[62]

Class libraries

The Java Class Library is the standard library, developed to support application development in Java. It is controlled by Sun Microsystems in cooperation with others through the Java Community Process program. Companies or individuals participating in this process can influence the design and development of the APIs. This process has been a subject of controversy. The class library contains features such as:

- The core libraries, which include:
  - IO/NIO
  - Networking
  - 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[63]
    - XML Processing (Parsing, Transforming, Validating) libraries
    - Security[64]
    - Internationalization and localization libraries[65]
- 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, used by many Java developers. 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 `*/`.[66]

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 smartcards.[67]
- Java Platform, Micro Edition (Java ME) – targeting environments with limited resources.[68]
- Java Platform, Standard Edition (Java SE) – targeting workstation environments.[69]
- Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.[70]

The classes in the Java APIs are organized into separate groups called packages. Each package contains a set of related interfaces, classes and exceptions. Refer to the separate platforms for a description of the packages available.

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

See also

- Dalvik – used in old Android versions, replaced by non-JIT Android Runtime
- JavaOne
- Javapedia
- List of Java virtual machines
- List of Java APIs
- List of JVM languages
- Graal (compiler), a project aiming to implement a high performance Java dynamic compiler and interpreter
- Spring Framework

Comparison of Java with other languages

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

Notes

1. Binstock, Andrew (20 May 2015). "Java's 20 Years Of Innovation". Forbes. Retrieved 18 March 2016.
2. Harry. H. Chaudhary (28 July 2014). "Cracking The Java Programming Interview :: 2000+ Java Interview Que/Ans". Retrieved 29 May 2016.
3. 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/) [2] (http://www.levenez.com/lang/)
4. Gosling, James; McGilton, Henry (May 1996). "The Java Language Environment".
5. Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad. "The Java Language Specification, 2nd Edition".
6. "The A-Z of Programming Languages: Modula-3". Computerworld.com.au. Retrieved 2010-06-09.
7. 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".), 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.
8. 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)
9. TechMetrix Research (1999). "History of Java" (PDF). *Java Application Servers Report*. "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"
10. "A Conversation with James Gosling – ACM Queue". Queue.acm.org. 2004-08-31. Retrieved 2010-06-09.
11. In the summer of 1996, Sun was designing the precursor to what is now the event model of the AWT and the JavaBeans TM 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)
12. "Chapel spec (Acknowledgements)" (PDF). Cray Inc. 2015-10-01. Retrieved 2016-01-14.
13. "Facebook Q&A: Hack brings static typing to PHP world". InfoWorld. 2014-03-26. Retrieved 2015-01-11.
14. Gosling et al. 2014, p. 1.
15. "Write once, run anywhere?". Computer Weekly. 2002-05-02. Retrieved 2009-07-27.
16. "1.2 Design Goals of the Java™ Programming Language". Oracle. 1999-01-01. Retrieved 2013-01-14.
17. McMillan, Robert (2013-08-01). "Is Java Losing Its Mojo?". wired.com. "Java is on the wane, at least according to one outfit that keeps on 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."
18. RedMonk Index (http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/) on redmonk.com (Stephen O'Grady, January 2015)
19. "Programming Language Popularity". langpop.com. 2013-10-25. Retrieved 2015-04-02. "Normalized Comparison: 1st C, 2nd Java, 3rd PHP"

20. "TIOBE Programming Community Index". 2015. Retrieved 2015-04-03.
21. "JavaOne 2013 Review: Java Takes on the Internet of Things". *www.oracle.com*. Retrieved 2016-06-19.
22. Byous, Jon (c. 1998). "Java technology: The early years". *Sun Developer Network*. Sun Microsystems. Archived from the original on 2005-04-20. Retrieved 2005-04-22.
23. Object-oriented programming "The History of Java Technology". *Sun Developer Network*. c. 1995. Retrieved 2010-04-30.
24. "So why did they decide to call it Java? (http://www.javaworld.com/jw-10-1996/jw-10-javaname.html)", Kieron Murphy, JavaWorld.com, 10/04/96
25. Kabutz, Heinz; *Once Upon an Oak* (http://www.artima.com/weblogs/viewpost.jsp?thread=7555). Artima. Retrieved April 29, 2007.
26. "The History of Java Technology". Retrieved October 6, 2012.
27. https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free
28. *Object-oriented Programming with Java: Essentials and Applications*. Tata McGraw-Hill Education. p. 34.
29. "JSG – Java Study Group". *open-std.org*.
30. "Why Java™ Was – Not – Standardized Twice" (PDF).
31. "What is ECMA—and why Microsoft cares".
32. "Java Community Process website". Jcp.org. 2010-05-24. Retrieved 2010-06-09.
33. "JAVAONE: Sun – The bulk of Java is open sourced". GrnLight.net. Retrieved 2014-05-26.
34. "Sun's Evolving Role as Java Evangelist". O'Reilly Media.
35. "Oracle and Java". *oracle.com*. Oracle Corporation. Retrieved 2010-08-23. "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. "Learn About Java Technology". Oracle. Retrieved 21 November 2011.
37. Gosling, James (April 9, 2010). "Time to move on...". *On a New Road*. Retrieved 2011-11-16.
38. JAVASOFT SHIPS JAVA 1.0 (https://web.archive.org/web/20070310235103/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml) at the Wayback Machine (archived March 10, 2007)
39. "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?". Programmer Interview. Retrieved 2015-01-19.
40. Niccolai, James (January 23, 2001). "Sun, Microsoft settle Java lawsuit". *JavaWorld*. International Data Group. Retrieved 2008-07-09.
41. Jelovic, Dejan. "Why Java will always be slower than C++". Retrieved 2008-02-15.
42. Google. "Loop Recognition in C++/Java/Go/Scala" (PDF). Retrieved 2012-07-12.
43. "Symantec's Just-In-Time Java Compiler To Be Integrated into Sun JDK 1.1".
44. "NullPointerException". Oracle. Retrieved 2014-05-06.
45. "Exceptions in Java". Artima.com. Retrieved 2010-08-10.
46. "Java HotSpot VM Options". Oracle.com. 2010-09-07. Retrieved 2012-06-30.
47. http://docs.oracle.com/javase/7/docs/technotes/guides/vm/G1.html
48. "Operator Overloading (C# vs Java)". *C# for Java Developers*. Microsoft. Retrieved 10 December 2014.
49. "Multiple Inheritance of State, Implementation, and Type". *The Java™ Tutorials*. Oracle. Retrieved 10 December 2014.
50. "Lesson: A Closer Look at the "Hello World!" Application". *The Java™ Tutorials > Getting Started*. Oracle Corporation. Retrieved 2011-04-14.
51. "Using applet, object and embed Tags". oracle.com. Retrieved 2010-10-14.
52. "Deploying Applets in a Mixed-Browser Environment". oracle.com. Retrieved 2010-10-14.
53. "Java and Scala's Type Systems are Unsound" (PDF).
54. Arnold, Ken. "Generics Considered Harmful". java.net. Retrieved 10 September 2015.. More comments to the original article available at earlier archive snapshots like this one from 2007 (http://web.archive.org/web/20071010002142/http://weblogs.java.net/blog/arnold/archive/2005/06/generics_consid_1.html).
55. Jelovic, Dejan. "Why Java Will Always Be Slower than C++". www.jelovic.com. Retrieved 17 October 2012.
56. Owens, Sean R. "Java and unsigned int, unsigned short, unsigned byte, unsigned long, etc. (Or rather, the lack thereof)". Archived from the original on 2009-02-20. Retrieved 2011-07-04.
57. Kahan, William. "How Java's Floating-Point Hurts Everyone Everywhere" (PDF). Electrical Engineering & Computer Science, University of California at Berkeley. Retrieved 4 June 2011.
58. "Have you checked the Java?".
59. Mullin, Joe. "Google guilty of infringement in Oracle trial; future legal headaches loom". *Law & Disorder*. Ars Technica. Retrieved 8 May 2012.
60. Joe Mullin (May 31, 2012). "Google wins crucial API ruling, Oracle's case decimated". *Ars Technica*. Retrieved 2012-06-01.
61. Rosenblatt, Seth (May 9, 2014). "Court sides with Oracle over Android in Java patent appeal". *CNET*. Retrieved 2014-05-10.

62. Mullin, Joe (26 May 2016). "Google beats Oracle—Android makes "fair use" of Java APIs". Ars Technica. Retrieved 26 May 2016.
63. "Collections Framework Overview". *Java Documentation*. Oracle. Retrieved 18 December 2014.
64. "Java™ Security Overview". *Java Documentation*. Oracle. Retrieved 18 December 2014.
65. "Trail: Internationalization". *The Java™ Tutorials*. Oracle. Retrieved 18 December 2014.
66. "How to Write Doc Comments for the Javadoc Tool". *Oracle Technology Network*. Oracle. Retrieved 18 December 2014.
67. "Java Card Overview". *Oracle Technology Network*. Oracle. Retrieved 18 December 2014.
68. "Java Platform, Micro Edition (Java ME)". *Oracle Technology Network*. Oracle. Retrieved 18 December 2014.
69. "Java SE". *Oracle Technology Network*. Oracle. Retrieved 18 December 2014.
70. "Java Platform, Enterprise Edition (Java EE)". *Oracle Technology Network*. Oracle. Retrieved 18 December 2014.

References

- Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). *The Java® Language Specification* (PDF) (Java SE 8 ed.).
- Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). *The Java Language Specification* (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.
- Lindholm, Tim; Yellin, Frank (1999). *The Java Virtual Machine Specification* (2nd ed.). Addison-Wesley. ISBN 0-201-43294-3.

External links

Wikiversity has learning resources about ***Java Platform, Enterprise Edition/Java EE Tutorial***

Retrieved from "https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=769528656"

Categories: Java (programming language) | Java platform | C programming language family | Class-based programming languages | Concurrent programming languages | Cross-platform software | Java specification requests | JVM programming languages | Object-oriented programming languages | Programming languages | Programming languages created in 1995 | Computer-related introductions in 1995 | Statically typed programming languages | Sun Microsystems