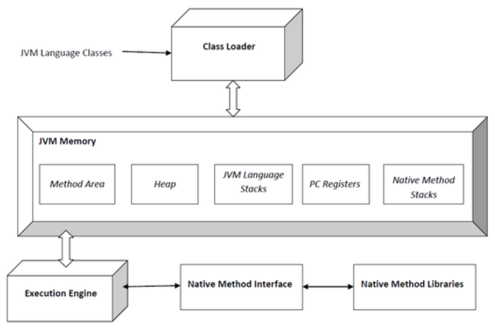
Java virtual machine

“JRE” redirects here. For the podcast, see Joe Rogan Experience.

A **Java virtual machine** (**JVM**) is an abstract computing machine that enables a computer to run a Java program. There are three notions of the JVM: specification, implementation, and instance. The specification is a document that formally describes what is required of a JVM implementation. Having a single specification ensures all implementations are interoperable. A JVM implementation is a computer program that meets the requirements of the JVM specification. An instance of a JVM is an implementation running in a process that executes a computer program compiled into Java bytecode.



Overview of a Java virtual machine (JVM) architecture based on The Java Virtual Machine Specification Java SE 7 Edition

Java Runtime Environment (**JRE**) is a software package that contains what is required to run a Java program. It includes a Java Virtual Machine implementation together with an implementation of the Java Class Library. The Oracle Corporation, which owns the Java trademark, distributes a Java Runtime environment with their Java Virtual Machine called HotSpot.

Java Development Kit (**JDK**) is a superset of a JRE and contains tools for Java programmers, e.g. a javac compiler. The Java Development Kit is provided free of charge either by Oracle Corporation directly, or by the OpenJDK open source project, which is governed by Oracle.

1 JVM specification

The Java virtual machine is an abstract (virtual) computer defined by a specification. This specification omits implementation details that are not essential to ensure interoperability: the memory layout of run-time data areas, the garbage-collection algorithm used, and any internal optimization of the Java virtual machine instructions (their translation into machine code). The main reason for this omission is to not unnecessarily constrain implementers. Any Java application can be run only inside some concrete implementation of the abstract specification of the Java virtual machine.[1]

Starting with Java Platform, Standard Edition (J2SE) 5.0, changes to the JVM specification have been developed under the Java Community Process as JSR 924.[2] As of 2006, changes to specification to support changes proposed to the class file format (JSR 202)[3] are being done as a maintenance release of JSR 924. The specification for the JVM was published as the *blue book*,[4] The preface states:

> We intend that this specification should sufficiently document the Java Virtual Machine to make possible compatible clean-room implementations. Oracle provides tests that verify the proper operation of implementations of the Java Virtual Machine.

One of Oracle’s JVMs is named HotSpot, the other, inherited from BEA Systems is JRockit. Clean-room Java implementations include Kaffe and IBM J9. Oracle owns the Java trademark and may allow its use to certify implementation suites as fully compatible with Oracle’s specification.

2 Class loader

Main article: Java Class loader

One of the organizational units of JVM byte code is a class. A class loader implementation must be able to recognize and load anything that conforms to the Java class file format. Any implementation is free to recognize other binary forms besides *class* files, but it must recognize *class* files.

The class loader performs three basic activities in this strict order:

1. Loading: finds and imports the binary data for a type
2. Linking: performs verification, preparation, and (optionally) resolution

    - Verification: ensures the correctness of the imported type
    - Preparation: allocates memory for class variables and initializing the memory to default values
    - Resolution: transforms symbolic references from the type into direct references.

3. Initialization: invokes Java code that initializes class variables to their proper starting values.

In general, there are two types of class loader: bootstrap class loader and user defined class loader.

Every Java virtual machine implementation must have a bootstrap class loader, capable of loading trusted classes. The Java virtual machine specification doesn't specify how a class loader should locate classes.

3 Bytecode instructions

Main article: Java bytecode

The JVM has instructions for the following groups of tasks:

- Load and store
- Arithmetic
- Type conversion
- Object creation and manipulation
- Operand stack management (push / pop)
- Control transfer (branching)
- Method invocation and return
- Throwing exceptions
- Monitor-based concurrency

The aim is binary compatibility. Each particular host operating system needs its own implementation of the JVM and runtime. These JVMs interpret the bytecode semantically the same way, but the actual implementation may be different. More complex than just emulating bytecode is compatibly and efficiently implementing the Java core API that must be mapped to each host operating system.

These instructions operate on a set of common abstracted data types rather the native data types of any specific processor architecture.

4 JVM languages

Main article: List of JVM languages

A JVM language is any language with functionality that can be expressed in terms of a valid class file which can be hosted by the Java Virtual Machine. A class file contains Java Virtual Machine instructions (Java byte code) and a symbol table, as well as other ancillary information. The class file format is the hardware- and operating system-independent binary format used to represent compiled classes and interfaces.[5]

There are several JVM languages, both old languages ported to JVM and completely new languages. JRuby and Jython are perhaps the most well-known ports of existing languages, i.e. Ruby and Python respectively. Of the new languages that have been created from scratch to compile to Java bytecode, Clojure, Groovy and Scala may be the most popular ones. A notable feature with the JVM languages is that they are compatible with each other, so that, for example, Scala libraries can be used with Java programs and vice versa.[6]

Java 7 JVM implements *JSR 292: Supporting Dynamically Typed Languages*[7] on the Java Platform, a new feature which supports dynamically typed languages in the JVM. This feature is developed within the Da Vinci Machine project whose mission is to extend the JVM so that it supports languages other than Java.[8][9]

5 Bytecode verifier

A basic philosophy of Java is that it is inherently *safe* from the standpoint that no user program can *crash* the host machine or otherwise interfere inappropriately with other operations on the host machine, and that it is possible to protect certain methods and data structures belonging to *trusted* code from access or corruption by *untrusted* code executing within the same JVM. Furthermore, common programmer errors that often led to data corruption or unpredictable behavior such as accessing off the end of an array or using an uninitialized pointer are not allowed to occur. Several features of Java combine to provide this safety, including the class model, the garbage-collected heap, and the verifier.

The JVM *verifies* all bytecode before it is executed. This verification consists primarily of three types of checks:

- Branches are always to valid locations
- Data is always initialized and references are always type-safe
- Access to *private* or *package private* data and methods is rigidly controlled

The first two of these checks take place primarily during the *verification* step that occurs when a class is loaded and made eligible for use. The third is primarily performed dynamically, when data items or methods of a class are first accessed by another class.

The verifier permits only some bytecode sequences in valid programs, e.g. a jump (branch) instruction can only target an instruction within the same method. Furthermore, the verifier ensures that any given instruction operates on a fixed stack location,[10] allowing the JIT compiler to transform stack accesses into fixed register accesses. Because of this, that the JVM is a stack architecture does not imply a speed penalty for emulation on register-based architectures when using a JIT compiler. In the face of the code-verified JVM architecture, it makes no difference to a JIT compiler whether it gets named imaginary registers or imaginary stack positions that must be allocated to the target architecture's registers. In fact, code verification makes the JVM different from a classic stack architecture, of which efficient emulation with a JIT compiler is more complicated and typically carried out by a slower interpreter.

The original specification for the bytecode verifier used natural language that was *incomplete or incorrect in some respects*. A number of attempts have been made to specify the JVM as a formal system. By doing this, the security of current JVM implementations can more thoroughly be analyzed, and potential security exploits prevented. It will also be possible to optimize the JVM by skipping unnecessary safety checks, if the application being run is proven to be safe.[11]

5.1 Secure execution of remote code

A virtual machine architecture allows very fine-grained control over the actions that code within the machine is permitted to take. It assumes the code is “semantically” correct, that is, it successfully passed the (formal) bytecode verifier process, materialized by a tool, possibly off-board the virtual machine. This is designed to allow safe execution of untrusted code from remote sources, a model used by Java applets, and other secure code downloads. Once bytecode-verified, the downloaded code runs in a restricted *sandbox*, which is designed to protect the user from misbehaving or malicious code. As an addition to the bytecode verification process, publishers can purchase a certificate with which to digitally sign applets as *safe*, giving them permission to ask the user to break out of the sandbox and access the local file system, clipboard, execute external pieces of software, or network.

Formal proof of bytecode verifiers have been done by the Javacard industry (Formal Development of an Embedded Verifier for Java Card Byte Code[12])

6 Bytecode interpreter and just-in-time compiler

For each hardware architecture a different Java bytecode interpreter is needed. When a computer has a Java bytecode interpreter, it can run any Java bytecode program, and the same program can be run on any computer that has such an interpreter.

When Java bytecode is executed by an interpreter, the execution will always be slower than the execution of the same program compiled into native machine language. This problem is mitigated by just-in-time (JIT) compilers for executing Java bytecode. A JIT compiler may translate Java bytecode into native machine language while executing the program. The translated parts of the program can then be executed much more quickly than they could be interpreted. This technique gets applied to those parts of a program frequently executed. This way a JIT compiler can significantly speed up the overall execution time.

There is no necessary connection between the Java programming language and Java bytecode. A program written in Java can be compiled directly into the machine language of a real computer and programs written in other languages than Java can be compiled into Java bytecode.

Java bytecode is intended to be platform-independent and secure.[13] Some JVM implementations do not include an interpreter, but consist only of a just-in-time compiler.[14]

7 JVM in the web browser

Since very early stages of the design process, Java (and JVM) has been marketed as a web technology for creating Rich Internet Applications.

7.1 Java applets

Main article: Java applet

On the client side, web browsers may be extended with a NPAPI Java plugin which executes so called Java applets embedded into HTML pages. The applet is allowed to draw into a rectangular region on the page assigned to it and use a restricted set of APIs that allow for example access to user's microphone or 3D acceleration. Java applets were superior to JavaScript both in performance and features until approximately 2011, when JavaScript engines in browsers were made significantly faster and the HTML 5 suite of web technologies started enhancing JavaScript with new APIs. Java applets are not able to modify the page outside its rectangular region which is not true about JavaScript. Adobe Flash Player, the main competing technology, works in the same way in this respect. Java applets are not restricted to Java and in general can be created in any JVM language.

As of April 2014, Google Chrome does not allow the use of any NPAPI plugins.[15] Mozilla Firefox will also ban NPAPI plugins by the end of 2016. This means that Java applets can no longer be used in either browser.[16] Oracle ultimately announced in January 2016 that it will discontinue the Java web browser plugin effective Java 9.[17]

As of June 2015 according to W3Techs, Java applet use had fallen to 0.1% of all web sites. Flash had fallen to 10.8% and Silverlight to 0.1% of web sites.[18]

7.2 JavaScript JVMs and interpreters

As of May 2016, JavaPoly allows users to import unmodified Java libraries, and invoke them directly from JavaScript. JavaPoly allows websites to use run unmodified Java libraries, even if the user does not have Java installed on their computer.[19]

7.3 Compilation to JavaScript

With the continuing improvements in JavaScript execution speed, combined with the increased use of mobile devices whose web browsers do not implement support for plugins, there are efforts to target those users through compilation to JavaScript. It is possible to either compile the source code or JVM bytecode to JavaScript. Compiling the JVM bytecode which is universal across JVM languages allows building upon the existing compiler to bytecode.

Main JVM bytecode to JavaScript compilers are TeaVM,[20] the compiler contained in Dragome Web SDK,[21] Bck2Brwsr,[22] and j2js-compiler.[23]

Leading compilers from JVM languages to JavaScript include the Java to JavaScript compiler contained in Google Web Toolkit, Clojurescript (Clojure), GrooScript (Groovy), Scala.js (Scala) and others.[24]

8 Java Runtime Environment from Oracle

Main article: HotSpot

The Java Runtime Environment (JRE) released by Oracle is a software distribution containing a stand-alone Java VM (HotSpot), browser plugin, Java standard libraries and a configuration tool. It is the most common Java environment installed on Windows computers. It is freely available for download at the website java.com.

8.1 Performance

Main article: Java performance

The JVM specification gives a lot of leeway to implementors regarding the implementation details. Since Java 1.3, JRE from Oracle contains a JVM called HotSpot. It has been designed to be a high-performance JVM.

To speed-up code execution, HotSpot relies on just-in-time compilation. To speed-up object allocation and garbage collection, HotSpot uses generational heap.

8.2 Generational heap

The *Java virtual machine heap* is the area of memory used by the JVM for dynamic memory allocation.[25]

In HotSpot the heap is divided into *generations*:

- The *young generation* stores short-lived objects that are created and immediately garbage collected.
- Objects that persist longer are moved to the *old generation* (also called the *tenured generation*). This memory is subdivided into (two) Survivors spaces where the objects that survived the first and next garbage collections are stored.

The *permanent generation* (or *permgen*) was used for class definitions and associated metadata prior to Java 8. Permanent generation was not part of the heap.[26][27] The *permanent generation* was removed from Java 8.[28]

Originally there was no permanent generation, and objects and classes were stored together in the same area. But as class unloading occurs much more rarely than objects are collected, moving class structures to a specific area allowed significant performance improvements.[26]

8.3 Security

Oracle's JRE is installed on a large number of computers. Since any web page the user visits may run Java applets, Java provides an easily accessible attack surface to malicious web sites that the user visits. Kaspersky Labs reports that the Java web browser plugin is the method of choice for computer criminals. Java exploits are included in many exploit packs that hackers deploy onto hacked web sites.[29]

In the past, end users were often using an out-of-date version of JRE which was vulnerable to many known attacks. This led to the widely shared belief between users that Java is inherently insecure.[30] Since Java 1.7, Oracle's JRE for Windows includes automatic update functionality.

9 See also

- List of Java virtual machines
- Comparison of Java virtual machines
- Comparison of application virtual machines
- Automated exception handling
- Java performance
- List of JVM languages
- Java processor
- Common Language Runtime

10 Notes

[1] Bill Venners, *Inside the Java Virtual Machine* Chapter 5

[2] "The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 924". Jcp.org. Retrieved 2015-06-26.

[3] "The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 202". Jcp.org. Retrieved 2015-06-26.

[4] *The Java Virtual Machine Specification* (the first and second editions are also available online).

[5] "The Java Virtual Machine Specification : Java SE 7 Edition" (PDF). Docs.oracle.com. Retrieved 2015-06-26.

[6] "Frequently Asked Questions - Java Interoperability". *scala-lang.org*. Retrieved 2015-11-18.

[7] "The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 292". Jcp.org. Retrieved 2015-06-26.

[8] "Da Vinci Machine project". Openjdk.java.net. Retrieved 2015-06-26.

[9] "New JDK 7 Feature: Support for Dynamically Typed Languages in the Java Virtual Machine". Oracle.com. Retrieved 2015-06-26.

[10] "The Verification process". *The Java Virtual Machine Specification*. Sun Microsystems. 1999. Retrieved 2009-05-31.

[11] Stephen N. Freund and John C. Mitchell. 1999. A formal framework for the Java bytecode language and verifier. In *Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '99)*, A. Michael Berman (Ed.) Association for Computing Machinery, New York, pp.147–166. doi:10.1145/320384.320397

[12] http://www-sop.inria.fr/everest/Lilian.Burdy/CBR02dsn.pdf

[13] David J. Eck, *Introduction to Programming Using Java*, Seventh Edition, Version 7.0, August 2014 at Section 1.3 "The Java Virtual Machine"

[14] *Oracle JRockit Introduction* Release R28 at 2. "Understanding Just-In-Time Compilation and Optimization"

[15] "Chrome starts pushing Java off the Web by disabling plugins". *Ars Technica*. Retrieved 9 December 2015.

[16] "Firefox will stop supporting plugins by end of 2016, following Chrome's lead". *PC World*. Retrieved 9 December 2015.

[17] "Oracle deprecates the Java browser plugin, prepares for its demise". *Ars Technica*. Retrieved 15 April 2016.

[18] "Historical yearly trends in the usage of client-side programming languages, June 2015". W3techs.com. Retrieved 2015-06-26.

[19] Krill, Paul (13 May 2016). "JavaPoly.js imports existing Java code and invokes it directly from JavaScript". InfoWorld. Retrieved 18 July 2016.

[20] "TeaVM project home page". Teavm.org. Retrieved 2015-06-26.

[21] "Dragome Web SDK". Dragome.com. Retrieved 2015-06-26.

[22] "Bck2Brwsr - APIDesign". Wiki.apidesign.org. Retrieved 2015-06-26.

[23] Wolfgang Kuehn (decatur). j2js-compiler GitHub

[24] "List of languages that compile to JS · jashkenas/coffeescript Wiki · GitHub". Github.com. 2015-06-19. Retrieved 2015-06-26.

[25] "Frequently Asked Questions about Garbage Collection in the Hotspot Java Virtual Machine". Sun Microsystems. 6 February 2003. Retrieved 7 February 2009.

[26] Masamitsu, Jon (28 November 2006). "Presenting the Permanent Generation". Retrieved 7 February 2009.

[27] Nutter, Charles (11 September 2008). "A First Taste of InvokeDynamic". Retrieved 7 February 2009.

[28] "JEP 122: Remove the Permanent Generation". Oracle Corporation. 2012-12-04. Retrieved 2014-03-23.

[29] "Is there any protection against Java exploits? | Kaspersky Lab". Kaspersky.com. 2013-09-09. Retrieved 2015-06-26.

[30] "What Is Java, Is It Insecure, and Should I Use It?". Lifehacker.com. 2013-01-14. Retrieved 2015-06-26.

11 References

- *Clarifications and Amendments to the Java Virtual Machine Specification, Second Edition* includes list of changes to be made to support J2SE 5.0 and JSR 45

- JSR 45, specifies changes to the class file format to support source-level debugging of languages such as JavaServer Pages (JSP) and SQLJ that are translated to Java

11.1 External links

- The Java Virtual Machine Specification
- Java implementations at DMOZ
- How to download and install prebuilt OpenJDK packages
- **How to Install Java?** (JRE from Oracle)

12 Text and image sources, contributors, and licenses

12.1 Text

12.2 Images

12.3 Content license