A Java virtual machine is a program which executes certain other programs, namely those containing Java bytecode instructions. JVMs are most often implemented to run on an existing [operating system](http://en.wikipedia.org/wiki/Operating_system), but can also be implemented to run directly on hardware. A JVM provides a [run-time environment](http://en.wikipedia.org/wiki/Run-time_environment) in which Java bytecode can be executed, enabling features such as [automated exception handling](http://en.wikipedia.org/wiki/Automated_exception_handling), which provides *root-cause* debugging information for every software error ([exception](http://en.wikipedia.org/wiki/Exception_handling)). A JVM is distributed along with [Java Class Library](http://en.wikipedia.org/wiki/Java_Class_Library), a set of [standard class libraries](http://en.wikipedia.org/wiki/Standard_library) (in Java bytecode) that implement the Java [application programming interface](http://en.wikipedia.org/wiki/Application_programming_interface) (API). These libraries, bundled together with the JVM, form the[Java Runtime Environment (JRE)](http://en.wikipedia.org/wiki/Java_Runtime_Environment).

JVMs are available for many hardware and software [platforms](http://en.wikipedia.org/wiki/Computing_platform). The use of the same bytecode for all JVMs on all platforms allows Java to be described as a [*write once, run anywhere*](http://en.wikipedia.org/wiki/Write_once,_run_anywhere) programming language, versus [*write once, compile anywhere*](http://en.wikipedia.org/wiki/Write_once,_compile_anywhere), which describes cross-platform [compiled languages](http://en.wikipedia.org/wiki/Compiled_language). Thus, the JVM is a crucial component of the [Java platform](http://en.wikipedia.org/wiki/Java_(software_platform)).

Java bytecode is an [intermediate language](http://en.wikipedia.org/wiki/Intermediate_language) which is typically compiled from Java, but it can also be compiled from other programming languages. For example, [Ada](http://en.wikipedia.org/wiki/Ada_(programming_language)" \o "Ada (programming language)) source code can be compiled to Java bytecode and executed on a JVM.

[Oracle Corporation](http://en.wikipedia.org/wiki/Oracle_Corporation), the owner of the *Java* [trademark](http://en.wikipedia.org/wiki/Trademark), produces the most widely used JVM,[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] named[HotSpot](http://en.wikipedia.org/wiki/HotSpot), that is written in the [C++](http://en.wikipedia.org/wiki/C%2B%2B) programming language.[[2]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-2) JVMs using the *Java* trademark may also be developed by other companies as long as they adhere to the JVM specification published by Oracle Corporation and to related contractual obligations.

Execution environment

Oracle's Java execution environment is termed the Java Runtime Environment, or JRE.

Programs intended to run on a JVM must be compiled into Java bytecode, a standardized portable binary format which typically comes in the form of [.class](http://en.wikipedia.org/wiki/Class_(file_format)) files ([Java class files](http://en.wikipedia.org/wiki/Java_class_file)). A program may consist of many classes in different files. For easier distribution of large programs, multiple class files may be packaged together in a [.jar](http://en.wikipedia.org/wiki/Jar_(file_format)) file (short for Java archive).

The Java application launcher, java, offers a standard way of executing Java code. Compare javaw.[[3]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-3)

The JVM [runtime](http://en.wikipedia.org/wiki/Run-time_system) executes .class or .jar files, [emulating](http://en.wikipedia.org/wiki/Emulator) the JVM [instruction set](http://en.wikipedia.org/wiki/Instruction_set) by [interpreting](http://en.wikipedia.org/wiki/Interpreter_(computing)) it or using a [just-in-time compiler](http://en.wikipedia.org/wiki/Just-in-time_compilation) (JIT) such as Oracle's [HotSpot](http://en.wikipedia.org/wiki/HotSpot" \o "HotSpot). JIT compiling, not interpreting, is used in most JVMs today to achieve greater speed. There are also [ahead-of-time compilers](http://en.wikipedia.org/wiki/AOT_compiler) that enable developers to precompile class files into native code for particular platforms.

Like most virtual machines, the Java virtual machine has a [stack](http://en.wikipedia.org/wiki/Stack_machine)-based architecture akin to a microcontroller/microprocessor. However, the JVM also has low-level support for Java-like classes and methods, which amounts to a highly idiosyncratic[*[clarification needed](http://en.wikipedia.org/wiki/Wikipedia:Please_clarify" \o "Wikipedia:Please clarify)*] [memory model](http://en.wikipedia.org/wiki/Java_Memory_Model) and capability-based architecture.

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 lead 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](http://en.wikipedia.org/wiki/Java_virtual_machine#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](http://en.wikipedia.org/wiki/Branch_(computer_science)) can only target an instruction within the same [method](http://en.wikipedia.org/wiki/Method_(computer_programming)). Furthermore, the verifier ensures that any given instruction operates on a fixed stack location,[[9]](http://en.wikipedia.org/wiki/Java_virtual_machine" \l "cite_note-9) 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](http://en.wikipedia.org/wiki/Register_machine) 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 which efficient emulation with a JIT compiler is more complicated and typically carried out by a slower interpreter.

Code verification also ensures that arbitrary bit patterns cannot get used as an address. [Memory protection](http://en.wikipedia.org/wiki/Memory_protection) is achieved without the need for a [memory management unit](http://en.wikipedia.org/wiki/Memory_management_unit) (MMU). Thus, JVM is an efficient way to get memory protection on simple architectures that lack an MMU. This is analogous to [managed code](http://en.wikipedia.org/wiki/Managed_code) in Microsoft's .NET [Common Language Runtime](http://en.wikipedia.org/wiki/Common_Language_Runtime), and conceptually similar to[capability architectures](http://en.wikipedia.org/wiki/Capability_architecture) such as the [Plessey 250](http://en.wikipedia.org/wiki/Plessey_250), and IBM [System/38](http://en.wikipedia.org/wiki/System/38).

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 proved to be safe.[[10]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-10)

Bytecode instructions

*Main article:*[*Java bytecode*](http://en.wikipedia.org/wiki/Java_bytecode)

The JVM has [instructions](http://en.wikipedia.org/wiki/Instruction_(computer_science)) for the following groups of tasks:

* Load and store
* [Arithmetic](http://en.wikipedia.org/wiki/Arithmetic)
* [Type conversion](http://en.wikipedia.org/wiki/Type_conversion)
* [Object creation and manipulation](http://en.wikipedia.org/wiki/Dynamic_memory_allocation)
* [Operand stack management (push / pop)](http://en.wikipedia.org/wiki/Stack_(data_structure))
* [Control transfer (branching)](http://en.wikipedia.org/wiki/Branch_(computer_science))
* [Method invocation and return](http://en.wikipedia.org/wiki/Subroutine)
* [Throwing exceptions](http://en.wikipedia.org/wiki/Exception_handling)
* [Monitor-based concurrency](http://en.wikipedia.org/wiki/Monitor_(synchronization))

The aim is binary compatibility. Each particular host [operating system](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Java_Class_Library) that must be mapped to each host operating system.

[[edit](http://en.wikipedia.org/w/index.php?title=Java_virtual_machine&action=edit&section=6)]Heap

The *Java virtual machine heap* is the area of memory used by the JVM, specifically [HotSpot](http://en.wikipedia.org/wiki/HotSpot" \o "HotSpot), for [dynamic memory allocation](http://en.wikipedia.org/wiki/Dynamic_memory_allocation).[[11]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-hotspotfaq-11) The heap is divided into *generations*:

* The *young generation* stores short-lived [objects](http://en.wikipedia.org/wiki/Object_(computer_science)) that are created and immediately [garbage collected](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)).
* Objects that persist longer are moved to the *old generation* (also called the *tenured generation*).

The *permanent generation* (or *permgen*) is used for [class](http://en.wikipedia.org/wiki/Class_(file_format)) definitions and associated metadata. Permanent generation is not part of the heap.[[12]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-permgen-12)[[13]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-13)

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 allows significant performance improvements.[[12]](http://en.wikipedia.org/wiki/Java_virtual_machine#cite_note-permgen-12)

[[edit](http://en.wikipedia.org/w/index.php?title=Java_virtual_machine&action=edit&section=7)]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. This is designed to allow safe execution of untrusted code from remote sources, a model used by [Java applets](http://en.wikipedia.org/wiki/Java_applet). Applets run within a VM incorporated into a user's browser, executing code downloaded from a remote [HTTP](http://en.wikipedia.org/wiki/HTTP) server. The remote code runs in a restricted [*sandbox*](http://en.wikipedia.org/wiki/Sandbox_(security)), which is designed to protect the user from misbehaving or malicious code. Publishers can purchase a certificate with which to [digitally sign](http://en.wikipedia.org/wiki/Digital_signature) applets as *safe*, giving them permission to ask the user to break out of the sandbox and access the local file system, [clipboard](http://en.wikipedia.org/wiki/Clipboard_(software)), execute external pieces of software, or network.