

reflection APIs added in Java 5.

- JVM Improvements - Most of the run time library is now mapped into memory as a memory image, as opposed to being loaded from a series of class files. Large portion of the runtime libraries will now be shared among multiple JVM instances.

(from [3])

Mustang (version 1.6.0; Java SE 6)

Released on 11 December 2006.^[4]

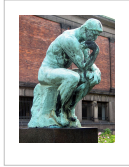
What's New in Java SE 6:

- Web Services - First-class support for writing XML web service client applications.
 - Scripting - You can now mix in JavaScript technology source code, useful for prototyping. Also useful when you have teams with a variety of skill sets. More advanced developers can plug in their own scripting engines and mix their favorite scripting language in with Java code as they see fit.
 - Database - No more need to find and configure your own JDBC database when developing a database application! Developers will also get the updated JDBC 4.0, a well-used API with many important improvements, such as special support for XML as an SQL datatype and better integration of Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) into the APIs.
 - More Desktop APIs - GUI developers get a large number of new tricks to play like the ever popular yet newly incorporated SwingWorker utility to help you with threading in GUI apps, JTable sorting and filtering, and a new facility for quick splash screens to quiet impatient users.
 - Monitoring and Management - The really big deal here is that you don't need do anything special to the startup to be able to attach on demand with any of the monitoring and management tools in the Java SE platform.
 - Compiler Access - Really aimed at people who create tools for Java development and for frameworks like JavaServer Pages (JSP) or Personal Home Page construction kit (PHP) engines that need to generate a bunch of classes on demand, the compiler API opens up programmatic access to javac for in-process compilation of dynamically generated Java code. The compiler API is not directly intended for the everyday developer, but for those of you deafened by your screaming inner geek, roll up your sleeves and give it a try. And the rest of us will happily benefit from the tools and the improved Java frameworks that use this.
 - Pluggable Annotations - ???
 - Desktop Deployment - At long last, Java SE 6 unifies the Java Plug-in technology and Java WebStart engines, which just makes sense. Installation of the Java WebStart application got a much needed makeover.
 - Security - Java SE 6 has simplified the job of its security administrators by providing various new ways to access platform-native security services, such as native Public Key Infrastructure (PKI) and cryptographic services on Microsoft Windows for secure authentication and communication, Java Generic Security Services (Java GSS) and Kerberos services for authentication, and access to LDAP servers for authenticating users.
 - The -lities: Quality, Compatibility, Stability - Bug fixes ...
-

Dolphin (version 1.7.0; Java SE 7)

Anticipated for 2010

The Java Platform



Points to ponder

The newer versions of Java, i.e., version 6.0 and 7.0 are being developed and are known by their code-names Mustang^[5] and Dolphin^[6] respectively.

You can test-ride their nightly builds, available online via the given links.

The **Java platform** is the name for a computing environment, or platform, from Sun Microsystems which can run applications developed using the Java programming language and set of development tools. In this case, the platform is not a specific hardware or operating system, but rather an execution engine called a virtual machine, and a set of standard libraries which provide common functionality.

The platform is properly called the **Java 2 Platform** (although the "2" is to be dropped [7]), and includes a Standard Edition or J2SE (now Java SE), an Enterprise Edition or J2EE (now Java EE), and a Micro Edition or J2ME (now Java ME). The current version of the Java 2 platform is alternatively specified as version 1.6 or version 6 (both refer to the same version). A good overview of the myriad of technologies that makes up the Java 2 Platform can be found on the JDK Documentation Page^[8].

Java technologies

The Java platform consists of a wide array of technologies, each of which provides a distinct portion of the overall development or runtime environment. For example, end-users typically interface with the Java virtual machine and the standard set of class libraries. In addition, there are numerous ways for Java applications to be deployed, including being embedded into a web page. Lastly, developers who are creating applications for the platform use a set of development tools called the Java Development Kit.

Java Runtime Environment

A program targeting the Java platform needs two components to be present on its host: a Java virtual machine, and a set of class libraries providing any services on which it depends. Sun's distribution of their JVM and their implementation of the standard classes is known as the Java Runtime Environment (JRE).

Java Virtual Machine

The heart of the Java platform is the concept of a common "virtual" processor that executes Java bytecode programs. This bytecode is the same no matter what hardware or operating system the program is running under. The Java platform provides an interpreter called the Java virtual machine (JVM), which translates the Java bytecode into native processor instructions at run-time. This permits the same application to be run on any platform that has a virtual machine available.

Since JRE version 1.2, Sun's JVM implementation has also included a just-in-time compiler. Instead of interpreting the bytecode one instruction at a time, this converts the bytecode for a program into equivalent native machine code as the program is loaded into the virtual machine, allowing it to execute much faster at the cost of a small delay whenever new bytecode is loaded. This allows the JIT compiler to target a specific host platform and hardware, even potentially optimizing the output code in different ways based on observations of the program's behaviour.

This is not to say that one can truly compile Java code to its fullest extent (in order to reap the benefits of speedy native machine code). Yes, there are "compilers" available that will attempt this feat, but not all Java libraries have a

machine code equivalent. For instance, the "reflect" library, which allows Java programmers to delve into instructions only available at runtime, is not well represented (if at all) by machine code.

Java was not the first virtual-machine-based platform, though it is by far the most successful and well-known. Previous uses for virtual machine technology primarily involved emulators to aid development for not-yet-developed hardware or operating systems, but the JVM was designed to be implemented entirely in software, while making it easy to efficiently port an implementation to hardware of all kinds.

Class libraries

In most modern operating systems, a large body of reusable code is provided to simplify the programmer's job. This code is typically provided as a set of dynamically loadable libraries that applications can call at runtime. Because the Java platform is not dependent on any specific operating system, applications cannot rely on any of the existing libraries. Instead, the Java platform provides a comprehensive set of standard class libraries, containing much of the same reusable functions commonly found in modern operating systems.

The Java class libraries serve three purposes within the Java platform. Like other standard code libraries, they provide the programmer with a well-known set of functions to perform common tasks, such as maintaining lists of items or performing complex string parsing. In addition, the class libraries provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system. Tasks such as network access and file access are often heavily dependent on the native capabilities of the platform. The Java `java.net` and `java.io` libraries implement the required native code internally, then provide a standard interface for the Java applications to perform those tasks. Finally, some underlying platforms may not support all of the features a Java application expects. In these cases, the class libraries can either emulate those features using whatever is available, or provide a consistent way to check for the presence of a specific feature.

Languages

The word Java, by itself, usually refers to the Java programming language which was designed for use with the Java platform. Programming languages are typically outside of the scope of the phrase "platform". However, Sun does not encourage the use of any other languages with the platform, and lists the Java programming language as a core part of the Java 2 platform. The language and runtime are therefore commonly considered a single unit.

Nevertheless, third parties have produced a number of compilers which target the JVM. Some of these are for existing languages, while others are for extensions to the Java language itself. These include:

- Groovy
- Pizza
- GJ (Generic Java), which was incorporated into official Java as of Sun's version 1.5.
- NetREXX

Another option is to use a more interface approach like the Jython.