Java Platform, Standard Edition

What's New in Oracle JDK 9

Release 9

E77563-05

September 2017

Overview of What's New in JDK 9

Java Platform, Standard Edition 9 is a major feature release. The following summarizes features and enhancements in Java SE 9 and in JDK 9, Oracle's implementation of Java SE 9.

A JDK Enhancement Proposal (JEP) is a proposal to design and implement a

Feature	Description
Java Platform Module System	Introduces a new kind of Java programing component, the module, which is a named, self-describing collection of code and data. This module system: • Introduces a new optional phase, link time, which is in-between compile time and run time, during which a set of modules can be assembled and optimized into a custom runtime image; see the `jlink` tool in *Java Platform, Standard Edition Tools Reference*. • Adds options to the tools `javac`, `jlink`, and `java` where you can specify module paths, which locate definitions of modules. • Introduces the modular JAR file, which is a JAR file with a `module-info.class` file in its root directory. • Introduces the JMOD format, which is a packaging format similar to JAR except it can include native code and configuration files; see the `jmod` tool. The JDK itself has been divided into a set of modules. This change: • Enables you to combine the JDK's modules into a variety of configurations, including: – Configurations corresponding to the JRE and the JDK. – Configurations roughly equivalent in content to each of the Compact Profiles defined in Java SE 8. – Custom configurations that contain only a specified set of modules and their required modules. • Restructures the JDK and JRE runtime images to accommodate modules and improve performance, security, and maintainability. • Defines a new URI scheme for naming modules, classes, and resources stored in a runtime image without revealing the internal structure or format of the image. • Removes the endorsed-standards override mechanism and the extension mechanism. • Removes `rt.jar` and `tools.jar` from the Java runtime image. • Makes most of the JDK's internal APIs inaccessible by default but leaves a few critical, widely used internal APIs accessible until supported replacements exist for all or most of their functionality. Run the command `jdeps -jdkinternals` to determine if your code uses internal JDK APIs. For more information, see the following: • Java Platform Module System (JSR 376) • JEP 261: Module System • JEP 200: The Modular JDK • JEP 220: Modular Run-Time Images • JEP 260: Encapsulate Most Internal APIs

Feature	Description
JEP 223: New Version-String Scheme	Provides a simplified version-string format that helps to clearly distinguish major, minor, security, and patch update releases. The new version-string format is as follows: `$MAJOR.$MINOR.$SECURITY.$PATCH` • `$MAJOR` is the version number that is incremented for a major release, for example JDK 9, which contains significant new features as specified by the Java SE platform specification. A major release contains new features and changes to existing features, which are planned and announced well in advance. • `$MINOR` is the version number that is incremented for each minor update, such as bug fixes, revisions to standard APIs, or implementation of features outside the scope of the relevant platform specifications. • `$SECURITY` is the version number that is incremented for a security-update release, which contains critical fixes, including those necessary to improve security. • `$PATCH` is the version number that is incremented for a release containing security and high-priority customer fixes that have been tested together. See New Version String Format in *Java Platform, Standard Edition Installation Guide*.

What’s New for the JDK 9 Installer

Feature	Description
JEP 222: jshell: The Java Shell (Read-Eval-Print Loop)	Adds Read-Eval-Print Loop (REPL) functionality to the Java platform. The `jshell` tool provides an interactive command-line interface for evaluating declarations, statements, and expressions of the Java programming language. It facilitates prototyping and exploration of coding options with immediate results and feedback. The immediate feedback combined with the ability to start with expressions is useful for education—whether learning the Java language or just learning a new API or language feature. See `jshell` in *Java Platform, Standard Edition Tools Reference*, and Introduction to JShell in *Java Platform, Standard Edition Java Shell User's Guide*. The JShell API enables applications to leverage REPL functionality. See the `jdk.jshell` package.
JEP 228: Add More Diagnostic Commands	Defines additional diagnostic commands to improve the ability to diagnose issues with Hotspot and the JDK. See `jcmd` in *Java Platform, Standard Edition Tools Reference*.
JEP 231: Remove Launch-Time JRE Version Selection	Removes the ability to request a version of the JRE that is not the JRE being launched at launch time. Modern applications are typically deployed through Java Web Start (with a JNLP file), native OS packaging systems, or active installers. These technologies have their own methods to manage the JREs needed by finding or downloading and updating the required JRE as needed. This makes launch-time JRE version selection obsolete.
JEP 238: Multi-Release JAR Files	Extends the JAR file format to enable multiple, Java release-specific versions of class files to coexist in a single archive. A multirelease JAR (MRJAR) contains additional, versioned directories for classes and resources specific to particular Java platform releases. Specify versioned directories with the `jar` tool's `--release` option.

Feature	Description
JEP 240: Remove the JVM TI hprof Agent	Removes the `hprof` agent from the JDK. The `hprof` agent was written as demonstration code for the JVM Tool Interface and not intended to be a production tool. The useful features of the `hprof` agent have been superseded by better alternatives.

Note:

While the `hprof` agent has been removed, it is still possible to create heap dumps in the `hprof` format using `jmap` or other diagnostic tools. See Diagnostic Tools in *Java Platform, Standard Edition Troubleshooting Guide*.

JEP 241: Remove the jhat Tool	Removes the `jhat` tool from the JDK. The `jhat` tool was an experimental and unsupported tool added in JDK 6. It is out of date; superior heap visualizers and analyzers have been available for many years.
JEP 245: Validate JVM Command-Line Flag Arguments	Validates arguments to all numerical JVM command-line flags to avoid failures and instead displays an appropriate error message if they are found to be invalid. Range and optional constraint checks have been implemented for arguments that require a user-specified numerical value. See `java` and Validate Java Virtual Machine Flag Arguments in *Java Platform, Standard Edition Tools Reference*.
JEP 247: Compile for Older Platform Versions	Enhances `javac` so that it can compile Java programs to run on selected earlier versions of the platform. When using the `-source` or `-target` options, the compiled program might accidentally use APIs that are not supported on the given target platform. The `--release` option will prevent accidental use of APIs. See `javac` in *Java Platform, Standard Edition Tools Reference*.
JEP 282: jlink: The Java Linker	Assembles and optimizes a set of modules and their dependencies into a custom runtime image as defined in JEP 220. The `jlink` tool defines a plug-in mechanism for transformation and optimization during the assembly process, and for the generation of alternative image formats. It can create a custom runtime optimized for a single program. JEP 261 defines *link time* as an optional phase between the phases of compile time and run time. Link time requires a linking tool that assembles and optimizes a set of modules and their transitive dependencies to create a runtime image or executable. See `jlink` in *Java Platform, Standard Edition Tools Reference*.

What's New for Security in JDK 9

Feature	Description
JEP 219: Datagram Transport Layer Security (DTLS)	Enables Java Secure Socket Extension (JSSE) API and the SunJSSE security provider to support DTLS Version 1.0 and DTLS Version 1.2 protocols. See Datagram Transport Layer Security (DTLS) in *Java Platform, Standard Edition Security Developer's Guide*.
JEP 244: TLS Application-Layer Protocol Negotiation Extension	Enables the client and server in a Transport Layer Security (TLS) connection to negotiate the application protocol to be used. With Application-Layer Protocol Negotiation (ALPN), the client sends the list of supported application protocols as part of the TLS ClientHello message. The server chooses a protocol and returns the selected protocol as part of the TLS ServerHello message. The application protocol negotiation can be accomplished within the TLS handshake, without adding network round-trips. See TLS Handshake and Application Layer Protocol Negotiation in *Java Platform, Standard Edition Security Developer's Guide*.
JEP 249: OCSP Stapling for TLS	Enables the server in a TLS connection to check for a revoked X. 509 certificate revocation. The server does this during TLS handshaking by contacting an Online Certificate Status Protocol (OCSP) responder for the certificate in question. It then attaches or "staples" the revocation information to the certificate that it returns to the client so that the client can take appropriate action. Enables the client to request OCSP stapling from a TLS server. The client checks stapled responses from servers that support the feature. See OCSP Stapling in *Java Platform, Standard Edition Security Developer's Guide*.
JEP 246: Leverage CPU Instructions for GHASH and RSA	Improves performance ranging from 34x to 150x for `AES/GCM/NoPadding` using GHASH HotSpot intrinsics. GHASH intrinsics are accelerated by the `PCLMULQDQ` instruction on Intel x64 CPU and the `xmul/xmulhi` instructions on SPARC. Improves performance up to 50% for `BigInteger squareToLen` and `BigInteger mulAdd` methods using RSA HotSpot intrinsics. RSA intrinsics apply to the `java.math.BigInteger` class on Intel x64. A new security property `jdk.security.provider.preferred` is introduced to configure providers that offer significant performance gains for specific algorithms. See Configuring the Preferred Provider for Specific Algorithms in *Java Platform, Standard Edition Security Developer's Guide*.
JEP 273: DRBG-Based SecureRandom Implementations	Provides the functionality of Deterministic Random Bit Generator (DRBG) mechanisms as specified in NIST SP 800-90Ar1 in the `SecureRandom` API. The DRBG mechanisms use modern algorithms as strong as SHA-512 and AES-256. Each of these mechanisms can be configured with different security strengths and features to match

	enhanced with several new constraints that allow greater control over the types of certificates that can be disabled. See JEP 288.
JEP 229: Create PKCS12 Keystores by Default	Modifies the default keystore type from JKS to PKCS12. PKCS#12 is an extensible, standard, and widely supported format for storing cryptographic keys. PKCS12 keystores improve confidentiality by storing private keys, trusted public key certificates, and secret keys. This feature also opens opportunities for interoperability with

Feature	Description
Deprecate the Java Plug-in	Deprecates the Java Plug-in and associated applet technologies in Oracle's JDK 9 builds. While still available in JDK 9, these technologies will be considered for removal from the Oracle JDK and JRE in a future release. Applets and JavaFX applications embedded in a web page require
	See Migrating Java Applets to Java Web Start and JNLP and Self-Contained Application Packaging in *Java Platform, Standard Edition Deployment Guide*.
Enhanced Java Control Panel	Improves the grouping and presentation of options within the Java Control Panel. Information is easier to locate, a search field is available, and modal dialog boxes are no longer used. Note that the location of some options has changed from previous versions of the Java Control Panel. See Java Control Panel in *Java Platform, Standard Edition Deployment Guide*.
JEP 275: Modular Java Application Packaging	Integrates features from Project Jigsaw into the Java Packager, including module awareness and custom runtime creation. Leverages the `jlink` tool to create smaller packages. Creates applications that use the JDK 9 runtime only. Cannot be used to package applications with an earlier release of the JRE.
	See Customization of the JRE and Packaging for Modular Applications in *Java Platform, Standard Edition Deployment Guide*.
JEP 289: Deprecate the Applet API	Deprecates the Applet API, which is becoming less useful as web browser vendors remove support for Java browser plug-ins. While still available in JDK 9, the `Applet` class will be considered for removal in a future release. Consider rewriting applets as Java Web Start or self-contained applications.

Feature	Description
JEP 213: Milling Project Coin	Identifies a few small changes: • Allow `@SafeVargs` on private instance methods. • Allow effectively final variables to be used as resources in the `try-with-resources` statement. • Allow the diamond with anonymous classes if the argument type of the inferred type is denotable. • Complete the removal, begun in Java SE 8, of the underscore from the set of legal identifier names. • Add support for private interface methods. See Java Language Changes for Java SE 9 in *Java Platform, Standard Edition Java Language Updates*.

Feature	Description
JEP 221: Simplified Doclet API	Replaces the old Doclet API with a new simplified API that leverages other standard, existing APIs. The standard doclet has been rewritten to use the new Doclet API. **Note:** The existing API and old standard doclet are available, but have not been updated to support new language features, such as modules.
JEP 224: HTML5 Javadoc	Supports generating HTML5 output. To get fully compliant HTML5 output, ensure that any HTML content provided in documentation comments are compliant with HTML5.
JEP 225: Javadoc Search	Provides a search box to the generated API documentation. Use this search box to find program elements, tagged words, and phrases within the documentation.
JEP 261: Module System	Supports documentation comments in module declarations. Includes new command-line options to configure the set of modules to be documented and generates a new summary page for any modules being documented.

Feature	Description
JEP 165: Compiler Control	Provides a way to control JVM compilation through compiler directive options. The level of control is runtime-manageable and method-specific. Compiler Control supersedes, and is backward compatible, with CompileCommand. See Compiler Control in *Java Platform, Standard Edition Java Virtual Machine Guide*.
JEP 197: Segmented Code Cache	Divides the code cache into distinct segments, each of which contains compiled code of a particular type, to improve performance and enable future extensions. See `java` in *Java Platform, Standard Edition Tools Reference*.
JEP 276: Dynamic Linking of Language-Defined Object Models	Dynamically links high-level object operations at run time, such as read a property, write a property, and invoke a function, to the appropriate target method handles. It links these operations to target method handles based on the actual types of the values passed. These object operations are expressed as `invokedynamic` sites. While `java.lang.invoke` provides a low-level API for dynamic linking of `invokedynamic` call sites, it doesn't provide a way to express higher level operations on objects nor methods that implement them. With the package `jdk.dynalink`, you can implement programming languages whose expressions contain dynamic types (types that cannot be determined statically) and whose operations on these dynamic types are expressed as `invokedynamic` call sites (because the language's object model or type system doesn't closely match that of the JVM).

What’s New for JVM Tuning in JDK 9

Feature	Description
Improve G1 Usability, Determinism, and Performance	Enhances the Garbage-First (G1) garbage collector to automatically determine several important memory-reclamation settings. Previously these settings had to be set manually to obtain optimal results. In addition, fixes issues with the usability, determinism, and performance of the G1 garbage collector.
JEP 158: Unified JVM Logging	Introduces a common logging system for all components of the JVM. See the `-Xloggc` java option in *Java Platform, Standard Edition Tools Reference*.

Feature	Description
JEP 214: Remove GC Combinations Deprecated in JDK 8	Removes garbage collector (GC) combinations that were deprecated in JDK 8. This means that the following GC combinations no longer exist: • DefNew + CMS • ParNew + SerialOld • Incremental CMS The "foreground" mode for Concurrent Mark Sweep (CMS) has also been removed. The following command-line flags have been removed: • `-Xincgc` • `-XX:+CMSIncrementalMode` • `-XX:+UseCMSCompactAtFullCollection` • `-XX:+CMSFullGCsBeforeCompaction` • `-XX:+UseCMSCollectionPassing` The command line flag `-XX:+UseParNewGC` no longer has an effect. ParNew can only be used with CMS and CMS requires ParNew. Thus, the `-XX:+UseParNewGC` flag has been deprecated and will likely be removed in a future release.
JEP 248: Make G1 the Default Garbage Collector	Makes Garbage-First (G1) the default garbage collector (GC) on 32- and 64-bit server configurations. Using a low-pause collector such as G1 provides a better overall experience, for most users, than a throughput-oriented collector such as the Parallel GC, which was previously the default. See Garbage-First Garbage Collector in *Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide*
JEP 271: Unified GC Logging	Reimplements Garbage Collection (GC) logging using the unified JVM logging framework introduced in JEP 158. GC logging is re-implemented in a manner consistent with the current GC logging format; however, some differences exist between the new and old formats. See Enable Logging with the JVM Unified Logging Framework in *Java Platform, Standard Edition Tools Reference*.
JEP 291: Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector	Deprecates the Concurrent Mark Sweep (CMS) garbage collector. A warning message is issued when it is requested on the command line, using the `-XX:+UseConcMarkSweepGC` option. The Garbage-First (G1) garbage collector is intended to be a replacement for most uses of CMS.

What’s New for Core Libraries in JDK 9

Feature	Description
JEP 102: Process API Updates	Improves the API for controlling and managing operating system processes. The `ProcessHandle` class provides the process's native process ID, arguments, command, start time, accumulated CPU time, user, parent process, and descendants. The class can also monitor processes' liveness and destroy processes. With the `ProcessHandle.onExit` method, the asynchronous mechanisms of the `CompletableFuture` class can perform an action when the process exits. See Process API in *Java Platform, Standard Edition Java Core Libraries Developer's Guide*, `java.lang.Process`, and `java.lang.ProcessHandle`.
JEP 193: Variable Handles	Defines a standard means to invoke the equivalents of `java.util.concurrent.atomic` and `sun.misc.Unsafe` operations upon object fields and array elements. Defines a standard set of fence operations, which consist of `VarHandle` static methods that enable fine-grained control of memory ordering. This is an alternative to `sun.misc.Unsafe`, which provides a nonstandard set of fence operations. Defines a standard reachability fence operation to ensure that a referenced object remains strongly reachable.
JEP 254: Compact Strings	Adopts a more space-efficient internal representation for strings. Previously, the `String` class stored characters in a `char` array, using two bytes (16 bits) for each character. The new internal representation of the `String` class is a byte array plus an encoding-flag field. This is purely an implementation change, with no changes to existing public interfaces. See the `CompactStrings` option of the `java` command in *Java Platform, Standard Edition Tools Reference*.
JEP 264: Platform Logging API and Service	Defines a minimal logging API that platform classes can use to log messages, together with a service interface for consumers of those messages. A library or application can provide an implementation of this service to route platform log messages to the logging framework of its choice. If no implementation is provided, then a default implementation based on the `java.util.logging` API is used.
JEP 266: More Concurrency Updates	Adds further concurrency updates to those introduced in JDK 8 in JEP 155: Concurrency Updates, including an interoperable publish-subscribe framework and enhancements to the `CompletableFuture` API.
JEP 268: XML Catalogs	Adds a standard XML Catalog API that supports the Organization for the Advancement of Structured Information Standards (OASIS) XML Catalogs version 1.1 standard. The API defines catalog and catalog-resolver abstractions that can be used as an intrinsic or external resolver with the JAXP processors that accept resolvers. Existing libraries or applications that use the internal catalog API will need to migrate to the new API to take advantage of the new features. See XML Catalog API in *Java Platform, Standard Edition Java Core Libraries Developer's Guide*.

sun.misc.Unsafe has been removed from JDK in Java 9.
Apache Spark makes heavy use of this class
Consequently does not work with JDK9

Feature	Description
JEP 269: Convenience Factory Methods for Collections	Makes it easier to create instances of collections and maps with small numbers of elements. New static factory methods on the `List`, `Set`, and `Map` interfaces make it simpler to create immutable instances of those collections. For example: `Set<String> alphabet = Set.of("a", "b", "c");` See Creating Immutable Lists, Sets, and Maps in *Java Platform, Standard Edition Java Core Libraries Developer's Guide*. For API documentation, see Immutable Set Static Factory Methods, Immutable Map Static Factory Methods, and Immutable List Static Factory Methods.
JEP 274: Enhanced Method Handles	Enhances the `MethodHandle`, `MethodHandles`, and `MethodHandles.Lookup` classes of the `java.lang.invoke` package to ease common use cases and enable better compiler optimizations. Additions include: • In the `MethodHandles` class in the `java.lang.invoke` package, provide new `MethodHandle` combinators for loops and `try`/`finally` blocks. • Enhance the `MethodHandle` and `MethodHandles` classes with new `MethodHandle` combinators for argument handling. • Implement new lookups for interface methods and, optionally, super constructors in the `MethodHandles.Lookup` class.
JEP 277: Enhanced Deprecation	Revamps the `@Deprecated` annotation to provide better information about the status and intended disposition of an API in the specification. Two new elements have been added: • `@Deprecated(forRemoval=true)` indicates that the API will be removed in a future release of the Java SE platform. • `@Deprecated(since="version")` contains the Java SE version string that indicates when the API element was deprecated, for those deprecated in Java SE 9 and beyond. For example: `@Deprecated(since="9", forRemoval=true)` `@Deprecated` annotations in the core platform have been updated. You can use a new tool, `jdeprscan`, to scan a class library (JAR file) for uses of deprecated JDK API elements. See Enhanced Deprecation in *Java Platform, Standard Edition Java Core Libraries Developer's Guide*. See `jdeprscan` in *Java Platform, Standard Edition Tools Reference*.
JEP 285: Spin-Wait Hints	Defines an API that enables Java code to hint that a spin loop is executing. A spin loop repeatedly checks to see if a condition is true, such as when a lock can be acquired, after which some computation can be safely performed followed by the release of the lock. This API is purely a hint, and carries no semantic behavior requirements. See the method `Thread.onSpinWait`.

Feature	Description
JEP 290: Filter Incoming Serialization Data	Allows incoming streams of object-serialization data to be filtered to improve both security and robustness. Object-serialization clients can validate their input more easily, and exported Remote Method Invocation (RMI) objects can validate invocation arguments more easily as well. Serialization clients implement a filter interface that is set on an `ObjectInputStream`. For RMI, the object is exported through a `RemoteServerRef` that sets the filter on the `MarshalInputStream` to validate the invocation arguments as they are unmarshalled.
JEP 259: Stack-Walking API	Provides a stack-walking API that allows easy filtering and lazy access to the information in stack traces. The API supports both short walks that stop at a frame that matches given criteria, and long walks that traverse the entire stack. Stopping at a frame that matches a given criteria avoids the cost of examining all the frames if the caller is interested only in the top frames on the stack. The API enables access to Class objects when the stack walker is configured to do so. See the class `java.lang.Stackwalker`.
JEP 255: Merge Selected Xerces 2.11.0 Updates into JAXP	Updates the JDK to support the 2.11.0 version of the Xerces parser. There is no change to the public JAXP API. The changes are in the following categories of Xerces 2.11.0: Datatypes, DOM L3 Serializer, XPointer, Catalog Resolver, and XML Schema Validation (including bug fixes, but not the XML Schema 1.1 development code).

	to the package `javax.imageio`. The new package `javax.imageio.plugins.tiff` provides classes that simplify the optional manipulation of TIFF metadata.
JEP 263: HiDPI Graphics on Windows and Linux	Automatically scales and sizes AWT and Swing components for High Dots Per Inch (HiDPI) displays on Windows and Linux. The JDK already supports HiDPI "retina displays" on OS X. Prior to this release, on Windows and Linux, Java applications were sized and rendered based on pixels, even on HiDPI displays that can have pixel densities two to three times as high as traditional displays. This led to GUI components and windows that were too small to read or use.