

ORACLE

ProductsIndustriesResourcesCustomersPartnersDevelopersCompany


Q

👤

View Accounts

🗨️

Contact Sales



JDK 19 Release Notes

All JDK Release Notes

Java Development Kit 19 Release Notes

JDK 19 Release Notes

The Java Platform, Standard Edition 19 Development Kit (JDK 19) is a feature release of the Java SE platform. It contains new features and enhancements in many functional areas. The Release Notes below describe the important changes, enhancements, removed APIs and features, deprecated APIs and features, and other information about JDK 19 and Java SE 19. Links to other sources of information about JDK 19 are also provided below.

Note: The Release Notes files are located only on our website.

- [JDK 19 ReadMe](#)
- [JDK 19 Guides and Reference Documentation](#)
- [JDK 19 Specifications](#)
- [Oracle JDK 19 Certified System Configurations](#)
- [JDK 19 Supported Locales](#)
- [Submitting a Bug Report and Available Support Options](#)
- [Copyright and License Terms for Documentation](#)

The following sections are included in these Release Notes:

Jump to category:

▼

Java™ SE Development Kit 19

These notes describe important changes, enhancements, removed APIs and features, deprecated APIs and features, and other information about JDK 19 and Java SE 19. In some cases, the descriptions provide links to additional detailed information about an issue or a change. This page does not duplicate the descriptions provided by the [Java SE 19 \(JSR 394\) Platform Specification](#), which provides informative background for all specification changes and might also include the identification of removed or deprecated APIs and features not described here. The Java SE 19 (JSR 394) specification provides links to:

- **Annex 1:** The complete [Java SE 19 API Specification](#).
- **Annex 2:** An [annotated API specification](#) showing the exact differences between Java SE 17 and Java SE 19. Informative background for these changes may be found in the list of approved Change Specification Requests for this release.
- **Annex 3:** Java SE 19 Editions of [The Java Language Specification](#) and [The Java Virtual Machine Specification](#). The Java SE 19 Editions contain all corrections and clarifications made since the Java SE 17 Editions, as well as additions for new features.

You should be aware of the content in the Java SE 19 (JSR 394) specification as well as the items described in this page.

The descriptions on this Release Notes page also identify potential compatibility issues that you might encounter when migrating to JDK 19. The [Kinds of Compatibility](#) page on the OpenJDK wiki identifies the following three types of potential compatibility issues for Java programs that might be used in these release notes:

- **Source:** Source compatibility preserves the ability to compile existing source code without error.
- **Binary:** Binary compatibility is defined in The Java Language Specification as preserving the ability to link existing class files without error.
- **Behavioral:** Behavioral compatibility includes the semantics of the code that is executed at runtime.

See [CSRs Approved for JDK 19](#) for the list of CSRs closed in JDK 19 and the [Compatibility & Specification Review \(CSR\)](#) page on the OpenJDK wiki for general information about compatibility.

The full version string for this release is build 19+36 (where "+" means "build"). The version number is 19.

IANA Data 2022a

JDK 19 contains IANA time zone data versions 2022a. For more information, refer to [Timezone Data Versions in the Java Runtimes](#).

[TOP](#)

Major New Functionality

1. Concurrency Model Update Previews

[JEP 425 Virtual Threads \(Preview\)](#)

Introduce *virtual threads* to the Java Platform. Virtual threads are lightweight threads that dramatically reduce the effort of writing, maintaining, and observing high-throughput concurrent applications. This is a [preview API](#).

[JEP 428 Structured Concurrency \(Incubator\)](#)

Simplify multithreaded programming by introducing an API for *structured concurrency*. Structured concurrency treats multiple tasks running in different threads as a single unit of work, thereby streamlining error handling and cancellation, improving reliability, and enhancing observability. This is an [incubating API](#).

2. Language Feature Previews

[JEP 405 Record Patterns \(Preview\)](#)

Enhance the Java programming language with *record patterns* to deconstruct record values. Record patterns and type patterns can be nested to enable a powerful, declarative, and composable form of data navigation and processing. This is a [preview language feature](#).

[JEP 427 Pattern Matching for switch \(Third Preview\)](#)

Enhance the Java programming language with pattern matching for `switch` expressions and statements. Extending pattern matching to `switch` allows an expression to be tested against a number of patterns, each with a specific action, so that complex data-oriented queries can be expressed concisely and safely. This is a [preview language feature](#).

3. Libraries Preview/Incubator

[JEP 424 Foreign Function & Memory API \(Preview\)](#)

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by

the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI. This is a [preview API](#).

JEP 426 Vector API (Fourth Incubator)

Introduce an API to express vector computations that reliably compile at runtime to optimal vector instructions on supported CPU architectures, thus achieving performance superior to equivalent scalar computations.

[TOP](#)

New Features

This section describes some of the enhancements in Java SE 19 and JDK 19. In some cases, the descriptions provide links to additional detailed information about an issue or a change. The APIs described here are provided with the Oracle JDK. It includes a complete implementation of the Java SE 19 Platform and additional Java APIs to support developing, debugging, and monitoring Java applications. Another source of information about important enhancements and new features in Java SE 19 and JDK 19 is the [Java SE 19 \(JSR 394\)](#) Platform Specification, which documents the changes to the specification made between Java SE 17 and Java SE 19. This document includes descriptions of those new features and enhancements that are also changes to the specification. The descriptions also identify potential compatibility issues that you might encounter when migrating to JDK 19.

core-libs/java.lang

➔ Support Unicode 14.0 ([JDK-8268081](#))

This release upgrades Unicode support to 14.0, which includes the following:

The `java.lang.Character` class supports Unicode Character Database of 14.0 level, which adds 838 characters, for a total of 144,697 characters. These additions include 5 new scripts, for a total of 159 scripts, as well as 37 new emoji characters. The `java.text.Bidi` and `java.text.Normalizer` classes support 14.0 level of Unicode Standard Annexes, #9 and #15, respectively. The `java.util.regex` package supports Extended Grapheme Clusters based on 14.0 level of Unicode Standard Annex #29 For more detail about Unicode 14.0, refer to the [Unicode Consortium's release note](#).

core-libs/java.lang

➔ New system properties for `System.out` and `System.err` ([JDK-8283620](#))

Two new system properties, `stdout.encoding` and `stderr.encoding`, have been added. The value of these system properties is the encoding used by the standard output and standard error streams (`System.out` and `System.err`).

The default values of these system properties depend on the platform. The values take on the value of the `native.encoding` property when the platform does not provide streams for the console. The properties can be overridden on the launcher's command line option (with `-D`) to set them to `UTF-8` where required.

core-libs/java.net

➔ HTTPS Channel Binding Support for Java GSS/Kerberos ([JDK-8279842](#))

Support has been added for TLS channel binding tokens for Negotiate/Kerberos authentication over HTTPS through `javax.net.HttpURLConnection`.

Channel binding tokens are increasingly required as an enhanced form of security. They work by communicating from a client to a server the client's understanding of the binding between connection security, as represented by a TLS server cert, and higher level authentication credentials, such as a username and password. The server can then detect if the client has been fooled by a MITM and shutdown the session or connection.

The feature is controlled through a new system property `jdk.https.negotiate.cbt` which is described fully in [Networking Properties](#).

core-libs/java.time

➔ Additional Date-Time Formats ([JDK-8176706](#))

Additional date/time formats are now introduced in `java.time.format.DateTimeFormatter/DateTimeFormatterBuilder` classes. In prior releases, only 4 predefined styles, i.e., `FormatStyle.FULL/LONG/MEDIUM/SHORT` are available. Now the users can specify their own flexible style with this new `DateTimeFormatter.ofLocalizedPattern(String requestedTemplate)` method. For example,

`DateTimeFormatter.ofLocalizedPattern("yMMM")`

produces a formatter, which can format a date in a localized manner, such as "Feb 2022" in the US locale, while "2022年2月" in the Japanese locale. Supporting method `DateTimeFormatterBuilder.appendLocalized(String requestedTemplate)` is also provided.

core-libs/java.util.collections

➔ **New Methods to Create Preallocated HashMaps and HashSets** (JDK-8186958)

New static factory methods have been introduced to allow creation of `HashMap` and related instances that are preallocated to accommodate an expected number of mappings or elements. After using the `HashMap.newHashMap` method, the requested number of mappings can be added to the newly created `HashMap` without it being resized. There are similar new static factory methods for `LinkedHashMap`, `WeakHashMap`, `HashSet`, and `LinkedHashSet`. The complete set of new methods is:

- `HashMap.newHashMap`
- `LinkedHashMap.newLinkedHashMap`
- `WeakHashMap.newWeakHashMap`
- `HashSet.newHashSet`
- `LinkedHashSet.newLinkedHashSet`

The `int`-argument constructors for these classes set the "capacity" (internal table size) which is not the same as the number of elements that can be accommodated. The capacity is related to the number of elements by a simple but error-prone calculation. For this reason, programs should use these new static factory methods in preference to the `int`-argument constructors.

hotspot/compiler

➔ **Support for PAC-RET Protection on Linux/AArch64** (JDK-8277204)

Support for PAC-RET protection on the Linux/AArch64 platform has been introduced.

When enabled, OpenJDK will use hardware features from the ARMv8.3 Pointer Authentication Code (PAC) extension to protect against Return Orientated Programming (ROP) attacks. For more information on the PAC extension see "[Providing protection for complex software](#)" or the "Pointer authentication in AArch64 state" section in the [Arm ARM](#).

To take advantage of this feature, first OpenJDK must be built with the configuration flag `--enable-branch-protection` using GCC 9.1.0+ or LLVM 10+ . Then, the runtime flag `-XX:UseBranchProtection=standard` will enable PAC-RET protection if the system supports it and the `java` binary was compiled with branch-protection enabled; otherwise the flag is silently ignored. Alternatively, `-XX:UseBranchProtection=pac-ret` will also enable PAC-RET protection, but in this case if the system does not support it or the `java` binary was not compiled with branch-protection enabled, then a warning will be printed.

hotspot/runtime

➔ **Automatic Generation of the CDS Archive** (JDK-8261455)

The JVM option `-XX:+AutoCreateSharedArchive` can be used to automatically create or update a CDS archive for an application. For example:

```
java -XX:+AutoCreateSharedArchive -XX:SharedArchiveFile=app.jsa -cp app.jar App
```

The specified archive will be written if it does not exist, or if it was generated by a different version of the JDK

security-libs/java.security

➔ **Windows KeyStore Updated to Include Access to the Local Machine Location** (JDK-6782021)

The Windows KeyStore support in the SunMSCAPI provider has been expanded to include access to the local machine location. The new keystore types are:

- "Windows-MY-LOCALMACHINE"
- "Windows-ROOT-LOCALMACHINE"

The following keystore types were also added, allowing developers to make it clear they map to the current user:

- "Windows-MY-CURRENTUSER" (same as "Windows-MY")
- "Windows-ROOT-CURRENTUSER" (same as "Windows-ROOT")

security-libs/java.security

➔ **Break Up SEQUENCE in X509Certificate::getSubjectAlternativeNames and X509Certificate::getIssuerAlternativeNames in otherName** (JDK-8277976)

The JDK implementation of `X509Certificate::getSubjectAlternativeNames` and `X509Certificate::getIssuerAlternativeNames` has been enhanced to additionally return the `type-id` and `value` fields of an `otherName`. The `value` field is returned as a `String` if it is encoded as a character string or otherwise as a byte array, which is helpful as it avoids having to parse the ASN.1 DER encoded form of the name.

security-libs/javax.net.ssl
➔ **(D)TLS Signature Schemes** ([JDK-8280494](#))

New Java SE APIs, `javax.net.ssl.SSLParameters.getSignatureSchemes()` and `javax.net.ssl.SSLParameters.setSignatureSchemes()`, have been added to allow applications to customize the signature schemes used in individual TLS or DTLS connections.

Note that the underlying provider may define the default signature schemes for each TLS or DTLS connection. Applications may also use the existing "jdk.tls.client.SignatureSchemes" and/or "jdk.tls.server.SignatureSchemes" system properties to customize the provider-specific default signature schemes. If not `null`, the signature schemes passed to the `setSignatureSchemes()` method will override the default signature schemes for the specified TLS or DTLS connections.

Note that a provider may not have been updated to support the new APIs and in that case may ignore the signature schemes that are set. The JDK `SunJSSE` provider supports this method. It is recommended that 3rd party providers add support for these methods when they add support for JDK 19 or later releases.

security-libs/jdk.security
➔ **Add a -providerPath Option to jarsigner** ([JDK-8281175](#))

A new option `-providerPath` has been added to `jarsigner`. One can use this option to specify the class path of an alternate keystore implementation. It can be used together with the `-providerClass` option.

security-libs/org.ietf.jgss:krb5
➔ **New Options for ktab to Provide Non-default Salt** ([JDK-8279064](#))

Two new options are added to the `ktab` command when adding new keytab entries. When `ktab -a username password -salt salt` is called, `altSalt` is used instead of the default salt. When `ktab -a username password -f` is called, the tool will contact the KDC to fetch the actual salt used.

xml/jaxp
➔ **New XML Processing Limits** ([JDK-8270504](#) (not public))

Three processing limits have been added to the XML libraries. These are:

- `jdk.xml.xpathExprGrpLimit`

Description: Limits the number of groups an XPath expression can contain.

Type: integer

Value: A positive integer. A value less than or equal to 0 indicates no limit. If the value is not an integer, a `NumberFormatException` is thrown. Default 10.

- `jdk.xml.xpathExprOpLimit`

Description: Limits the number of operators an XPath expression can contain.

Type: integer

Value: A positive integer. A value less than or equal to 0 indicates no limit. If the value is not an integer, a `NumberFormatException` is thrown. Default 100.

- `jdk.xml.xpathTotalOpLimit`

Description: Limits the total number of XPath operators in an XSL Stylesheet.

Type: integer

Value: A positive integer. A value less than or equal to 0 indicates no limit. If the value is not an integer, a `NumberFormatException` is thrown. Default 10000.

Supported processors

- `jdk.xml.xpathExprGrpLimit` and `jdk.xml.xpathExprOpLimit` are supported by the XPath processor.
- All three limits are supported by the XSLT processor.

Setting properties

For the XSLT processor, the properties can be changed through the `TransformerFactory`. For example,

```
TransformerFactory factory = TransformerFactory.newInstance();

factory.setAttribute("jdk.xml.xpathTotalOpLimit", "1000");
```

For the XPath processor, the properties can be changed through the `XPathFactory`. For example,

```
XPathFactory xf = XPathFactory.newInstance();

xf.setProperty("jdk.xml.xpathExprGrpLimit", "20");
```

For both the XPath and XSLT processors, the properties can be set through the system property and `jaxp.properties` configuration file located in the `conf` directory of the Java installation. For example,

```
System.setProperty("jdk.xml.xpathExprGrpLimit", "20");
```

or in the `jaxp.properties` file,

```
jdk.xml.xpathExprGrpLimit=20
```

There are two known issues:

An XPath expression that contains a short form of the parent axis `.."` can return incorrect results. See [JDK-8284920](#) for details.

An invalid XPath expression that ends with a relational operator such as `<` `>` and `=` will cause the processor to erroneously throw `StringIndexOutOfBoundsException` instead of `XPathExpressionException`. See [JDK-8284548](#) for details.

[TOP](#)

Removed Features and Options

This section describes the APIs, features, and options that were removed in Java SE 19 and JDK 19. The APIs described here are those that are provided with the Oracle JDK. It includes a complete implementation of the Java SE 19 Platform and additional Java APIs to support developing, debugging, and monitoring Java applications. Another source of information about important enhancements and new features in Java SE 19 and JDK 19 is the [Java SE 19 \(JSR 394\)](#) Platform Specification, which documents changes to the specification made between Java SE 17 and Java SE 19. This document includes the identification of removed APIs and features not described here. The descriptions below might also identify potential compatibility issues that you could encounter when migrating to JDK 19. See [CSRs Approved for JDK 19](#) for the list of CSRs closed in JDK 19.

hotspot/gc

➔ Removal of Diagnostic Flag `GCPParallelVerificationEnabled` ([JDK-8286304](#))

The diagnostic flag `GCPParallelVerificationEnabled` has been removed.

There are no known advantages of disabling parallel heap verification, so this flag has never been used except with its default value. This default value enabled multi-threaded verification for a very long time with no issues. Single-threaded heap verification would even be much slower than verification already is.

security-libs/javax.net.ssl

➔ **Remove Finalizer Implementation in SSLSocketImpl** (JDK-8212136)

The finalizer implementation in SSLSocket has been removed, with the underlying native resource releases now done by the Socket implementation. With this update, the TLS close_notify messages will no longer be emitted if SSLSocket is not explicitly closed.

Not closing Sockets properly is an error condition that should be avoided. Applications should always close sockets and not rely on garbage collection.

security-libs/javax.security

➔ **Remove the Alternate ThreadLocal Implementation of the Subject::current and Subject::callAs APIs** (JDK-8282676 (not public))

The `jdk.security.auth.subject.useTL` system property and the alternate `ThreadLocal` implementation of the `Subject::current` and `Subject::callAs` APIs have been removed. The default implementation of these APIs is still supported.

[TOP](#)

Deprecated Features and Options

Additional sources of information about the APIs, features, and options deprecated in Java SE 19 and JDK 19 include:

- The [Deprecated API](#) page identifies all deprecated APIs including those deprecated in Java SE 19.
- The [Java SE 19 \(JSR 394\)](#) specification documents changes to the specification made between Java SE 17 and Java SE 19 that include the identification of deprecated APIs and features not described here.
- [JEP 277: Enhanced Deprecation](#) provides a detailed description of the deprecation policy. You should be aware of the updated policy described in this document.

You should be aware of the contents in those documents as well as the items described in this release notes page.

The descriptions of deprecated APIs might include references to the deprecation warnings of `forRemoval=true` and `forRemoval=false`. The `forRemoval=true` text indicates that a deprecated API might be removed from the next major release. The `forRemoval=false` text indicates that a deprecated API is not expected to be removed from the next major release but might be removed in some later release.

The descriptions below also identify potential compatibility issues that you might encounter when migrating to JDK 19. See [CSRs Approved for JDK 19](#) for the list of CSRs closed in JDK 19.

core-libs/java.lang

➔ **java.lang.ThreadGroup Is Degraded** (JDK-8284161)

Legacy `java.lang.ThreadGroup` has been degraded in this release. It is no longer possible to explicitly destroy a thread group. In its place, `ThreadGroup` is changed to no longer keep a strong reference to subgroups. A thread group is thus eligible to be GC'ed when there are no live threads in the group and nothing else is keeping the thread group alive.

The behavior of several methods, deprecated for removal in prior releases, are changed as follows:

- The `destroy` method does nothing.
- The `isDestroyed` method returns false.
- The `setDaemon` and `isDaemon` methods set/get a daemon status that is not used for anything.
- The `suspend`, `resume`, and `stop` methods throw `UnsupportedOperationException`.

For further details, see the [JEP 425, section java.lang.ThreadGroup](#).

core-libs/java.util.i18n

➔ **Deprecation of Locale Class Constructors** (JDK-8282819)

New `Locale.of()` factory methods replace deprecated `Locale` constructors. The factory methods are efficient and reuse existing `Locale` instances. `Locales` are also provided by `Locale.forLanguageTag()` and `Locale.Builder`.

security-libs/java.security

➔ **PSSParameterSpec(int) Constructor and DEFAULT Static Constant Are Deprecated** ([JDK-8254935](#))

It is recommended to construct PSSParameterSpec explicitly with all desired values instead of using the DEFAULT static constant or the single argument constructor which takes the salt length. Both use the default values in the initial version of the PKCS#1 standard and some of these values are no longer recommended due to advances in cryptanalysis.

security-libs/javax.crypto

➔ **OAEPParameterSpec.DEFAULT Static Constant Is Deprecated** ([JDK-8284553](#))

It is recommended to construct OAEPParameterSpec explicitly with desired values instead of using the DEFAULT static constant. The DEFAULT static constant uses the default values in the initial version of the PKCS#1 standard and some of these values are no longer recommended due to advances in cryptanalysis.

[TOP](#)

Known Issues

The following notes describe known issues or limitations in this release.

xml/jaxp

➔ **JDK XSLT Transformer Limitations** ([JDK-8290347](#))

Applications using the JDK XSLT transformer to convert stylesheets to Java objects can encounter the following exception:

```
com.sun.org.apache.xalan.internal.xsltc.compiler.util.InternalError: Internal XSLTC error: a
method in the translet exceeds the Java Virtual Machine limitation on the length of a method of
64 kilobytes. This is usually caused by templates in a stylesheet that are very large. Try
restructuring your stylesheet to use smaller templates.
```

Applications will encounter the above exception if the size of the XSL template is too large. It is recommended to split the XSL template into smaller templates. Alternatively, applications can override the JDK XSLT Transformer by providing third-party implementation JAR files in the class path.

install/install

➔ **Installation of Oracle Linux Specific x64 JDK RPMs Pulls in i686 Dependencies** ([JDK-8297475](#) (Not Public))

This issue prevents yum from automatically installing the correct packages required by Oracle Linux specific x86_64 headless and headful JDK packages. Instead of x86_64 packages, it will install i686 packages. To workaround the issue, you may manually install packages with the same names as indicated by yum but with the x86_64 architecture.

After you have the x86_64 headless and/or headful jdk packages installed, you can get the list of required x86_64 packages by running the following script:

```
rpm -qa | grep -E -e '^jdk-.*-headful-.*\.x86_64$' -e '^jdk-.*-headless-.*\.x86_64$' | xargs -r
rpm -q --requires | sort -u | cut -d ' ' -f 1 | grep -v '^rpmlib' | xargs -r rpm -q --
whatprovides | sort -u | grep -e '.i[3456]86$' | xargs -r rpm -q --queryformat '%
{name}.x86_64\n' | xargs -r echo
```

It will output a space-separated list of names of required x86_64 packages to stdout. You can pass this list to a `sudo yum install` command to ensure the installation of the required packages.

[TOP](#)

Other Notes

The following notes describe additional changes and information about this release. In some cases, the following descriptions provide links to additional detailed information about an issue or a change.

client-libs/2d

➔ **Metal Is Now the Default Java 2D Rendering Pipeline on macOS** ([JDK-8284378](#))

Previously JDK desktop applications using Swing and Java2D (tm) would render using OpenGL on macOS. As of this release of JDK, they now are rendered using Apple's new Metal accelerated graphics API. This has been available since JDK 17 (JEP 382), but was not automatically enabled. Now it is enabled by default. Applications will not need to take any action, as they will automatically benefit from faster graphics with lower power consumption, and the use of a more modern stable graphics API which will be able to work better on current and future Apple Mac systems. Any user who would prefer to continue to use OpenGL whilst it is still supported can disable rendering with Metal by starting their application with either "java -Dsun.java2d.metal=false" or "java -Dsun.java2d.opengl=true" and it will run with OpenGL as it used to in JDK 17.

core-libs/java.io

➔ **New System Property to Disable Windows Alternate Data Stream Support in java.io.File** (JDK-8285445)

The Windows implementation of `java.io.File` allows access to NTFS Alternate Data Streams (ADS) by default. Such streams have a structure like “filename:streamname”. A system property `jdk.io.File.enableADS` has been added to control this behavior. To disable ADS support in `java.io.File`, the system property `jdk.io.File.enableADS` should be set to `false` (case ignored). Stricter path checking however prevents the use of special devices such as `NUL`:

core-libs/java.lang

➔ **User's Home Directory Is Set to \$HOME if Invalid** (JDK-8280357)

On Linux and macOS, the system property `user.home` is set to the home directory provided by the operating system. If the directory name provided is empty or only a single character, the value of the environment variable `HOME` is used instead.

The directory name and the value of `$HOME` are usually the same and valid. The fallback to `$HOME` is unusual and unlikely to occur except in environments such as `systemd` on Linux or when running in a container such as Docker.

core-libs/java.lang

➔ **Thread Context ClassLoader Changed to be a Special Inheritable Thread-local** (JDK-8284161)

The thread context `ClassLoader` is specified in this release to be a special inheritable thread-local. This change should be transparent to existing code with the exception of code that uses the 5-arg `Thread` constructor (added in Java 9) to create a `Thread` that does not inherit the initial values of inheritable thread-locals from the constructing thread. With this release, invoking the 5-arg `Thread` constructor with the parameter `inheritInheritableThreadLocals` set to `false` will create a `Thread` that does not inherit the initial value of the context `ClassLoader` from the constructing thread. The `Thread.setContextClassLoader` method may be used to change the context `ClassLoader` of the new thread if needed.

For further details, see the [JEP 425, section Thread-local variables](#).

core-libs/java.lang

➔ **Source and Binary Incompatible Changes to java.lang.Thread** (JDK-8284161)

The are a few source and binary incompatible changes that may impact code that extends `java.lang.Thread`.

- `Thread` defines several new methods in this release. If code in an existing source file extends `Thread` and a method in the subclass conflicts with any of the new `Thread` methods then the file will not compile without change.
- `Thread.Builder` is added as a nested interface. If code in an existing source file extends `Thread`, imports a class named `Builder`, and code in the subclass references "Builder" as a simple name, then the file will not compile without change.
- `Thread.isVirtual()`, `Thread.threadId()` and `Thread.join(Duration)` are added as final methods. If there is existing compiled code that extends `Thread` and the subclass declares a method with the same name, parameters, and return type as any of these methods then `IncompatibleClassChangeError` will be thrown at run-time if the subclass is loaded.

For further details, see the [JEP 425, section java.lang.Thread](#).

core-libs/java.lang

➔ **Incorrect Handling of Quoted Arguments in ProcessBuilder** (JDK-8282008)

`ProcessBuilder` on Windows is restored to address a regression caused by [JDK-8250568](#). Previously, an argument to `ProcessBuilder` that started with a double-quote and ended with a backslash followed by a double-quote was passed to a command incorrectly and may cause the command to fail. For example the argument "C:\\Program Files\\", would be seen by the command with extra double-quotes. This update restores the long standing behavior that does not treat the backslash before the final double-quote specially.

core-libs/java.lang

➔ **Double.toString(double) and Float.toString(float) May Return Slightly Different Results** (JDK-4511638)

The specification of these methods is now tighter than in earlier releases and the new implementation fully adheres to it.

As a consequence, some returned strings are now shorter than when using earlier releases, and for inputs at the extremes of the subnormal ranges near zero, might look differently. However, the number of cases where there's a difference in output is quite small compared to the sheer number of possible `double` and `float` inputs.

One example is `Double.toString(2e23)`, which now returns `"2.0E23"`, whereas in earlier releases it returns `"1.9999999999999998E23"`. Another example, in the `double` subnormal range, is `Double.toString(1e-323)` which now returns `"9.9E-324"`, as mandated by the new specification.

core-libs/java.lang:reflect

➔ **Make Annotation toString Output for Enum Constants Usable for Source Input** (JDK-8281462)

The exact `toString` format for annotations is not specified; however, the `toString` output is intended to be usable for source input. The `toString` output of enum constants was changed to two ways so that it would be usable as source input:

- The name of the constant is used (rather than the output is its `toString` method)
- For the name of the enum class, its canonical name rather than its binary name is used.

core-libs/java.net

➔ **MD5 and SHA-1 Are Disabled by Default for HTTP Digest Authentication** (JDK-8281561)

The MD5 and SHA-1 message digest algorithms have been disabled by default for HTTP Digest authentication. MD5 and SHA-1 are considered insecure and are deprecated generally. Accordingly, they have both been disabled by default for some usages of HTTP Digest authentication with `java.net.HttpURLConnection`. They can re-enabled on an opt-in basis by setting a new system property. More information about them can be found in [Networking Properites](#).

core-libs/java.net

➔ **Improved HTTP Proxy Detection on Windows** (JDK-8262442)

When multiple Windows proxy configuration options are available, proxy selector now attempts all options in sequence until a proxy is selected or all options have been tried. Previously, only the first option was tried. For example, if automatic proxy detection was enabled, manual proxy setup was never used.

core-libs/java.net

➔ **java.net.InetAddress Updated to Reject Ambiguous IPv4 Address Literals** (JDK-8277608 (not public))

The `java.net.InetAddress` class has been updated to strictly accept IPv4 address literals in decimal quad notation. The `InetAddress` class methods are updated to throw a `java.net.UnknownHostException` for invalid IPv4 address literals. To disable this check, the new `jdk.net.allowAmbiguousIPAddressLiterals` system property can be set to `"true"`.

core-libs/java.net

➔ **Make HttpURLConnection Default Keep Alive Timeout Configurable** (JDK-8278067)

Two system properties have been added which control the keep alive behavior of `HttpURLConnection` in the case where the server does not specify a keep alive time. Two properties are defined for controlling connections to servers and proxies separately. They are `http.keepAlive.time.server` and `http.keepAlive.time.proxy` respectively. More information about them can be found in [Networking Properites](#).

core-libs/java.nio

➔ **FileChannel.transferFrom May Transfer Fewer Bytes than Expected** (JDK-8286763)

The performance of `FileChannel.transferFrom()` has been improved significantly on Linux kernel version 4.5 and later for the case where the method is used to transfer bytes from one `FileChannel` to another. This change adds to the preexisting set of scenarios in which the number of bytes actually transferred might be less than the number requested to be transferred. That is to say, the value returned by `transferFrom()` can be less than the value of the `count` parameter: a “short transfer.” This is permitted by the specification, but might impact broken code that ignores the returned count and assumes it is always equal to `count`.

core-libs/java.nio

➔ **The mark and set Methods of InputStream and FilterInputStream Are No Longer Synchronized** (JDK-8284930)

The `synchronized` keyword is removed from the `mark` and `reset` methods of `java.io.InputStream` and `java.io.FilterInputStream`. This keyword serves no purpose as the other methods in these classes do not synchronize.

core-libs/java.nio

➔ **Files.copy Copies POSIX Attributes to Target on Foreign File System** (JDK-8267820)

The method `java.nio.file.Files.copy(Path, Path)` has been changed to copy POSIX file attributes from the source file to the destination file when the two files are associated with different file system providers, for example copying a file from the

default file system to a zip file system. Both the source and target file systems must support the POSIX file attribute view. The POSIX attributes copied are constrained to the file access permissions; owner and group owner of the file are not copied.

core-libs/java.nio

➔ **FileChannel.lock/tryLock Changed to Treat Size 0 to Mean the Locked Region Goes to End of File** (JDK-5041655)

The method `java.nio.channels.FileChannel.lock(long position, long size, boolean shared)` has been changed such that a size value of zero means to lock all bytes from the specified starting position to the end of the file, regardless of whether the file is subsequently extended or truncated.

core-libs/java.time

➔ **java.time.DateTimeFormatter: Wrong Definition of Symbol F** (JDK-8282081)

The definition and its implementation of the pattern symbol `F` in `java.time.format.DateTimeFormatter/Builder` has been modified. It was tied with `ChronoField.ALIGNED_DAY_OF_WEEK_IN_MONTH` field, which did not agree with `java.text.SimpleDateFormat` and Unicode Consortium's LDML. With this release, it represents `ChronoField.ALIGNED_WEEK_OF_MONTH` field. For example, the number 2 means "the 2nd Wednesday in July."

core-libs/java.time

➔ **Support for IsoFields in JapaneseDate/MinguoDate/ThaiBuddhistDate** (JDK-8279185)

Three chronologies in `java.time.chrono` package, namely `JapaneseChronology`, `MinguoChronology`, and `ThaiBuddhistChronology` now support ISO-based fields, such as `IsoFields.QUARTER_OF_YEAR`. These chronologies implement the new method, `isIsoBased()` which has been added in the `java.time.chrono.Chronology` interface. The boolean returned from this method indicates if the implementing chronology is ISO chronology based, which means it has the same year/month structure as `IsoChronology`.

Here is an example:

```
JapaneseDate.now().getLong(IsoFields.QUARTER_OF_YEAR)
```

will return the correct quarter-of-year value, which used to be throwing an `UnsupportedTemporalTypeException` with prior JDK releases.

core-libs/java.util.concurrent

➔ **ForkJoinPool and ThreadPoolExecutor Do Not Use Thread::start to Start Worker Threads** (JDK-8284161)

`java.util.concurrent.ForkJoinPool` and `java.util.concurrent.ThreadPoolExecutor` have changed in this release to not start worker threads with the `Thread.start` method. This may impact code that constructs a `ForkJoinPool` or `ThreadPoolExecutor` with a thread factory that creates worker `Threads` that override the no-arg `start` method. The overridden `start` method will not be invoked when worker threads are started. The change in behavior does not impact code that overrides the `ForkJoinWorkerThread.onStart()` method. The `onStart()` method will continue to be invoked by fork join worker threads when they start. A future release will re-examine the issue of thread factories creating threads that override the `start` method.

core-libs/java.util.jar

➔ **InflaterInputStream.read Throws EOFException** (JDK-8292327)

A change to `java.util.zip.InflaterInputStream` in this release means it is possible that reading uncompressed bytes with this API can fail with an unexpected `java.io.EOFException`.

The issue arises when reading uncompressed bytes with a byte array that isn't large enough to fit all bytes that have been uncompressed. In that case, the additional uncompressed bytes are buffered by the implementation to be consumed by the next call to the `read` method. If the compressed stream is at end of stream then a subsequent read of the uncompressed data will fail incorrectly with `EOFException`.

This issue will be fixed in a future update. It may be possible to workaround the issue in some cases by calling `read` with a larger byte array.

core-libs/java.util.regex

➔ **Regex \b Character Class Now Matches ASCII Characters only by Default** (JDK-8264160)

The `\b` metacharacter now matches ASCII word characters by default in the same way that the `\w` metacharacter does. For `\b` to match Unicode characters, the `UNICODE_CHARACTER_CLASS` must be set in the same way that it would need to be set for `\w` to match Unicode characters.

core-libs/java.util.i18n

➔ **Support for CLDR Version 41** (JDK-8265315)

Locale data based on Unicode Consortium's CLDR has been upgraded to version 41. For the detailed locale data changes, please refer to the Unicode Consortium's [CLDR release notes](#).

core-libs/javax.naming

➔ **Parsing of URL Strings in Built-in JNDI Providers Is More Strict** (JDK-8278972 (not public))

The parsing of URLs in the LDAP, DNS, and RMI built-in JNDI providers has been made more strict. The strength of the parsing can be controlled by system properties:

```
-Dcom.sun.jndi.ldapURLParsing="legacy" | "compat" | "strict"      (to control "ldap:" URLs)

-Dcom.sun.jndi.dnsURLParsing="legacy" | "compat" | "strict"      (to control "dns:" URLs)
-Dcom.sun.jndi.rmiURLParsing="legacy" | "compat" | "strict"      (to control "rmi:" URLs)
```

The default value is "compat" for all of the three providers.

- The "legacy" mode turns the new validation off.
- The "compat" mode limits incompatibilities.
- The "strict" mode is stricter and may cause regression by rejecting URLs that an application might consider as valid.

In "compat" and "strict" mode, more validation is performed. As an example, in the URL authority component, the new parsing only accepts brackets around IPv6 literal addresses. Developers are encouraged to use `java.net.URI` constructors or its factory method to build URLs rather than handcrafting URL strings.

If an illegal URL string is found, a `java.lang.IllegalArgumentException` or a `javax.naming.NamingException` (or a subclass of it) is raised.

core-svc/tools

➔ **jstatd No Longer Requires a SecurityManager** (JDK-8272317)

`jstatd` no longer requires a Security Manager and policy file. Running with `-Djava.security.policy=` to set a policy has no effect.

Internally to `jstatd`, an `ObjectInputFilter` is used to allow only essential classes to be deserialized over the RMI connection.

hotspot/jvmti

➔ **JVM TI Changes to Support Virtual Threads** (JDK-8284161)

The JVM Tool Interface (JVM TI) has been updated in this release to support virtual threads. Maintainers of agents that use JVM TI are strongly recommended to read [JEP 425](#) and the JVM TI 19.0 specification. The following is a summary of the JVM TI support for virtual threads:

- Most JVM TI functions that are called with a `jthread`, such as a JNI reference to a `Thread` object, can be called with a reference to a virtual thread. The functions that are not supported on virtual threads are `PopFrame`, `ForceEarlyReturn`, `StopThread`, `AgentStartFunction`, and `GetThreadCpuTime`. The `SetLocal*` functions support setting local variables in the top-most frame of virtual threads that are suspended at a breakpoint or single step event but are allowed to fail with `JVMTI_ERROR_OPAQUE_FRAME` in other scenarios.
- All JVM TI events, with the exception of those posted during early VM startup or during heap iteration, can have event callbacks invoked in the context of a virtual thread.
- The `GetAllThreads` and `GetAllStackTraces` functions are specified to return all platform threads rather than all threads.
- New functions `SuspendAllVirtualThreads` and `ResumeAllVirtualThreads` are added to support bulk suspend and resume of virtual threads. New events `VirtualThreadStart` and `VirtualThreadEnd` are added to support tracking of virtual threads. A new capability, `can_support_virtual_threads` is used to enable the use of the new functions and events.

Existing JVM TI agents will mostly work as before, but may encounter errors if they invoke functions that are not supported on virtual threads. These will arise when an agent that is unaware of virtual threads is used with an application that uses virtual threads. The change to `GetAllThreads` to return an array containing only the platform threads may be an issue for some agents. Existing agents that enable the `ThreadStart` and `ThreadEnd` events for all threads may encounter performance issues until they are upgraded to have finer control of these events.

hotspot/runtime

➔ **JNI GetVersion Returns JNI_VERSION_19** (JDK-8286176)

The Java Native Interface function `GetVersion` has been changed in this release to return `JNI_VERSION_19` (value `0x00130000`).

hotspot/runtime

➔ **CPU Shares Ignored When Computing Active Processor Count** ([JDK-8281181](#))

Previous JDK releases used an incorrect interpretation of the Linux cgroups parameter "cpu.shares". This might cause the JVM to use fewer CPUs than available, leading to an under utilization of CPU resources when the JVM is used inside a container.

Starting from this JDK release, by default, the JVM no longer considers "cpu.shares" when deciding the number of threads to be used by the various thread pools. The `-XX:+UseContainerCpuShares` command-line option can be used to revert to the previous behavior. This option is deprecated and may be removed in a future JDK release.

install/install

➔ **RPM JDK Installer Changes** ([JDK-8275446](#))

Installation directory name of Oracle JDK in RPM package has changed from `/usr/java/jdk-${VERSION}` to `/usr/lib/jvm/jdk-${FEATURE}-oracle-${ARCH}`. Thus the 19.0.1 and 19.0.2 releases for x64 will both be installed in `/usr/lib/jvm/jdk-19-oracle-x64` directory. RPM package will create `/usr/java/jdk-${FEATURE}` link pointing to the installation directory for backward compatibility.

Communication with the alternatives framework of JDK RPM package has changed. JDK RPM packages of prior versions registered a single `java` group of commands with the alternatives framework. The JDK 19 RPM package registers `java` and `javac` groups with the alternatives framework. `java` group is for commands used to run applications: `java`, `keytool`, and `rmiregistry`. `javac` group is used for all other commands. The set of commands registered by the package has not changed.

install/install

➔ **All JDK Update Releases Are Installed into the Same Directory on macOS** ([JDK-8281010](#))

The Oracle JDK installation directory name will be changed from `/Library/Java/JavaVirtualMachines/jdk-${VERSION}.jdk` to `/Library/Java/JavaVirtualMachines/jdk-${FEATURE}.jdk`. Thus the 19.0.1 and 19.0.2 releases will both install into the `/Library/Java/JavaVirtualMachines/jdk-19.jdk` installation directory. Installing an older JDK update release will log an error, and not install the JDK, if a newer version of the same feature release already exists. An error dialog will be shown except in the case of a silent installation.

install/install

➔ **JDK-8278370: [win] Disable Side-by-Side Installations of Multiple JDK Updates in Windows JDK Installers** ([JDK-8278370](#))

Windows JDK installers must install the Oracle JDK in `%Program Files%\Java\jdk-%FEATURE%` instead of `%Program Files%\Java\jdk-%VNUM%`. I.e. all updates of the same release must share one installation directory.

Thus the 19.0.1 and 19.0.2 releases will both install into `%Program Files%\Java\jdk-19` by default, and they both cannot be installed at the same time.

If the JDK19.0.2 installer is launched when JDK19.0.1 is already installed, it will auto-upgrade them to JDK19.0.2. There may be a Files In Use dialog shown if the older version was running and locking JDK files.

If the JDK19.0.1 installer is launched when JDK19.0.2 is already installed, it will show an error that a newer version of this JDK family is already installed.

security-libs/java.security

➔ **Only Expose Certificates With Proper Trust Settings as Trusted Certificate Entries in macOS KeychainStore** ([JDK-8278449](#) (not public))

On macOS, only certificates with proper trust settings in the user keychain will be exposed as trusted certificate entries in the KeychainStore type of keystore. Also, calling the `KeyStore::setCertificateEntry` method or the `keytool -importcert` command on a KeychainStore keystore now fails with a `KeyStoreException`. Instead, call the macOS "security add-trusted-cert" command to add a trusted certificate into the user keychain.

security-libs/java.security

➔ **RC2 and ARCFOUR Algorithms Added to jdk.security.legacyAlgorithms Security Property** ([JDK-8286090](#))

The RC2 and ARCFOUR (RC4) algorithms have been added to the `jdk.security.legacyAlgorithms` security property in the `java.security` configuration file. The `keytool` tool issues warnings when a weak RC2 or ARCFOUR algorithm is used for its commands associated with secret key entries in the keystore.

security-libs/java.security

➔ **Use Larger Default Key Sizes if not Explicitly Specified** ([JDK-8267319](#))

JDK providers use provider-specific default values if the caller does not specify a key size when using a `KeyPairGenerator` or `KeyGenerator` object to generate a key pair or secret key. With this enhancement, the default key sizes for various crypto algorithms have been increased as follows:

- RSA, RSASSA-PSS, DH: from 2048 to 3072
- EC: from 256 to 384
- AES: from 128 to 256 (if permitted by crypto policy), falls back to 128 otherwise.

In addition, the `jarsigner` tool will now use SHA-384 instead of SHA-256 as the default digest algorithm. The default signature algorithm for the `jarsigner` tool has also been adjusted accordingly. SHA-384 is used instead of SHA-256 except for longer key sizes whose security strength matches SHA-512. Note that for DSA keys, `jarsigner` will continue using `SHA256withDSA` as the default signature algorithm. This ensures maximum interoperability with older JDK releases. For more details, please refer to the `keytool` and `jarsigner` documentation.

security-libs/java.security

➔ **getParameters of ECDSA Signature Objects Always Return Null** ([JDK-8286908](#))

In order to be compliant to [RFC 5758 Section 3.2](#), the `Signature::getParameters` method on an ECDSA signature object from the SunEC security provider will always return `null`, even if an earlier `setParameter` method has been called on this object.

security-libs/java.security

➔ **DES, DESede, and MD5 Algorithms Added to jdk.security.legacyAlgorithms Security Property** ([JDK-8255552](#))

The DES, DESede and MD5 algorithms have been added to the `jdk.security.legacyAlgorithms` security property in the `java.security` configuration file. The `keytool` tool issues warnings when a weak DES or DESede algorithm is used for its commands associated with secret key entries in the keystore.

security-libs/javax.net.ssl

➔ **Fully Support Endpoint Identification Algorithm in RFC 6125** ([JDK-7192189](#))

The JDK `SunJSSE` provider implementation has been enhanced to be fully compliant with RFC 6125. Prior to this release, the implementation was compliant except for one case, which has now been addressed: the implementation will not attempt to match wildcard domains in TLS certificates where the wildcard character comprises a label other than the left-most label.

If necessary, applications can workaround this restriction by implementing their own `HostnameVerifier` or `TrustManager`.

security-libs/javax.net.ssl

➔ **TLS Cipher Suites using 3DES Removed from the Default Enabled List** ([JDK-8163327](#))

The following TLS cipher suites that use the obsolete 3DES algorithm have been removed from the default list of enabled cipher suites:

- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA

Note that cipher suites using 3DES are already disabled by default in the `jdk.tls.disabledAlgorithms` security property. You may use these suites at your own risk by removing `3DES_EDE_CBC` from the `jdk.tls.disabledAlgorithms` security property and re-enabling the suites via the `setEnabledCipherSuites()` method of the `SSLSocket`, `SSLServerSocket`, or `SSLEngine` classes. Alternatively, if an application is using the `HttpsURLConnection` class, the `https.cipherSuites` system property can be used to re-enable the suites.

tools/javac

➔ **Indy String Concat Changes Order of Operations** ([JDK-8273914](#))

String concatenation has been changed to evaluate each argument and eagerly convert it to a string, in left-to-right order. This fixes a bug in the invokedynamic-based string concatenation strategies introduced in [JEP 280](#).

For example, the following now prints "foofoobar", not "foobarfoobar":

```
Stringbuilder builder = new StringBuilder("foo");

System.err.println("" + builder + builder.append("bar"));
```

tools/javac

➔ **Lambda Deserialization Fails for Object Method References on Interfaces** ([JDK-8282080](#))

Deserialization of serialized method references to Object methods, which was using an interface as the type on which the method is invoked, can now be deserialized again. Note the classfiles need to be recompiled to allow the deserialization.

tools/javadoc(tool)

➔ **JavaDoc Search Enhancements** ([JDK-8248863](#))

API documentation generated by Javadoc now provides a standalone search page and the search syntax has been enhanced to allow for multiple search terms.

tools/jpackage

➔ **Allow Per-User and System Wide Configuration of a jpackaged App** ([JDK-8250950](#))

jpackaged applications support both system-wide and per-user configuration.

jpackage application launcher will look up the corresponding .cfg file not only in the application installation directory (the system-wide installation location) but also in user-specific locations. User-specific directories for .cfg file look up are:

Linux

```
~/.local/${PACKAGE_NAME}
```

```
~/.${PACKAGE_NAME}
```

macOS

```
~/Library/Application Support/${PACKAGE_NAME}
```

Windows

```
%LocalAppData%\%PACKAGE_NAME%
```

```
%AppData%\%PACKAGE_NAME%
```

where \${PACKAGE_NAME} and %PACKAGE_NAME% refer to jpackaged application name.

tools/jshell

➔ **JShell Highlights Deprecated Elements, Variables, and Keywords** ([JDK-8274148](#))

JShell now marks deprecated elements and highlights variables and keywords in the console.

tools/launcher

➔ **-Xss May Be Rounded up to a Multiple of the System Page Size** ([JDK-8236569](#))

The actual java thread stack size may differ from the value specified by -Xss command line option; it may be rounded up to a multiple of the system page size when required by the operating system.

tools/launcher

➔ **Use Larger Default key Sizes if not Explicitly Specified** ([JDK-8267319](#))

JDK providers use provider-specific default values if the caller does not specify a key size when using a KeyPairGenerator or KeyGenerator object to generate a key pair or secret key. With this enhancement, the default key sizes for various crypto algorithms have been increased as follows:

- RSA, RSASSA-PSS, DH: from 2048 to 3072
- EC: from 256 to 384
- AES: from 128 to 256 (if permitted by crypto policy), falls back to 128 otherwise

In addition, the jarsigner tool will now use SHA-384 instead of SHA-256 as the default digest algorithm. The default signature algorithm for the jarsigner tool has also been adjusted accordingly. SHA-384 is used instead of SHA-256 except for longer key sizes whose security strength matches SHA-512. Note that for DSA keys, jarsigner will continue using SHA256withDSA as the

default signature algorithm. This ensures maximum interoperability with older JDK releases. For more details, please refer to the `keytool` and `jarsigner` documentation.

infrastructure

➔ **Toolchain Upgrade to Visual Studio 2022** ([JDK-8283723](#))

As part of ongoing maintenance, the JDK for Windows is built using the Microsoft Visual Studio 2022 toolchain starting with this release.

If you have issues with a Java application and if you have native or JNI libraries that are compiled with a different release of the compiler, then you must consider compatibility issues between the runtimes. Specifically, your environment is supported only if you follow the Microsoft guidelines when dealing with multiple runtimes.

core-libs/java.lang

➔ **System Property for Java SE Specification Maintenance Version** ([JDK-8285497](#))

The `java.specification.maintenance.version` system property is defined to indicate the maintenance release number of the Java SE specification being implemented by the JDK. If the implemented specification has not undergone a maintenance release, the value of system property is not set.

security-libs/javax.net.ssl

➔ **Change in `SSLEngine.closeInbound()` Behavior** ([JDK-8273553](#))

The `SunJSSE` close notification checks for `SSLEngine` to have been made less strict to conform to changes in the Transport Layer Security (TLS) RFCs. See also [JDK-8253368](#).

Specifically, if an application tries to close its `SSLEngine` inbound side using `SSLEngine.closeInbound()` without having received a close notification message from its peer, the `SSLEngine` will no longer:

1. trigger the transmission of a TLS fatal-level alert to the peer, and
2. invalidate the current TLS session

The new behavior will still consider this condition an error and will throw a local `javax.net.ssl.SSLException`. But a fatal-level alert will no longer be generated to be sent to the peer, and the underlying session will remain valid.

In addition, the internal transport context for the `SSLEngine` will also now be closed. This may result in a different `SSLEngineResult.HandshakeStatus` value on the `SSLEngine`. Any outstanding outbound data must still be obtained (`SSLEngine.wrap()`) and sent in order to gracefully close the connection.

[TOP](#)

Differences Between Oracle JDK and OpenJDK

Although we have stated the goal to have Oracle JDK and OpenJDK binaries be as close to each other as possible, there remain several differences between the two options.

The current differences are:

- Oracle JDK offers "installers" (`msi`, `rpm`, `deb`, etc.) which not only place the JDK binaries in your system but also contain update rules and in some cases handle some common configurations like set common environmental variables (such as, `JAVA_HOME` in Windows) and establish file associations (such as, use `java` to launch `.jar` files). OpenJDK is offered only as compressed archive (`tar.gz` or `.zip`).
- Usage Logging is only available in Oracle JDK.
- Oracle JDK requires that third-party cryptographic providers be signed with a Java Cryptography Extension (JCE) Code Signing Certificate. OpenJDK continues allowing the use of unsigned third-party crypto providers.
- The output of `java -version` is different. Oracle JDK returns `java` and includes the Oracle-specific identifier. OpenJDK returns `OpenJDK` and does not include the Oracle-specific identifier.
- Oracle JDK 17 and later are released under the [Oracle No-Fee Terms and Conditions License](#). OpenJDK is released under [GPLv2wCP](#). License files included with each will therefore be different.
- Oracle JDK distributes FreeType under the FreeType license and OpenJDK does so under GPLv2. The contents of `\legal\java.desktop\freetype.md` is therefore different.
- Oracle JDK has Java cup and steam icons and OpenJDK has Duke icons.
- Oracle JDK source code includes "ORACLE PROPRIETARY/CONFIDENTIAL. Use is subject to license terms." Source code distributed with OpenJDK refers to the GPL license terms instead.

Resources for

Careers

Developers

Investors

Partners

Researchers

Students and Educators

Why Oracle

Analyst Reports

Best cloud-based ERP

Cloud Economics

Corporate Responsibility

Diversity and Inclusion

Security Practices

Learn

What is cloud computing?

What is CRM?

What is Docker?

What is Kubernetes?

What is Python?

What is SaaS?

News and Events

News

Oracle CloudWorld

Oracle CloudWorld Tour

Oracle Health Conference

DevLive Level Up

Search all events

Contact Us

US Sales: +1.800.633.0738

How can we help?

Subscribe to emails

Integrity Helpline


© 2023 Oracle


Privacy / Do Not Sell My Info


Cookie Preferences


Ad Choices


Careers









 Country/Region