JDK 17 Release Notes

All JDK Release Notes | Java Development Kit 17 Release Notes

JDK 17 Release Notes

The following sections are included in these Release Notes:

Jump to category:

Introduction

These notes describe important changes, enhancements, removed APIs and features, deprecated APIs and features, and other information about JDK 17 and Java SE 17. 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 17 (JSR 392) 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 17 (JSR 392) specification provides links to:

- **Annex 1:** The complete Java SE 17 API Specification.
- **Annex 2:** An annotated API specification showing the exact differences relative to Java SE 17. Informative background for these changes may be found in the list of approved Change Specification Requests for this release.
- **Annex 3:** Java SE 17 Editions of The Java Language Specification and The Java Virtual Machine Specification. The Java SE 17 Editions contain all corrections and clarifications made since the Java SE 16 Editions, as well as additions for new features.

You should be aware of the content in the Java SE 17 (JSR 392) 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 17. 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 17 for the list of CSRs closed in JDK 17 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 17+35 (where "+" means "build"). The version number is 17.

IANA Data 2021a

JDK 17 contains IANA time zone data version 2021a. For more information, refer to Timezone Data Versions in the JRE Software.

TOP

New Features

This section describes some of the enhancements in Java SE 17 and JDK 17. 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 17 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 17 and JDK 17 is the Java SE 17 (JSR 392) Platform Specification, which documents the changes to the specification made between Java SE 16 and Java SE 17. 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 17.

specification
➜ **JEP 409: Sealed Classes**
Sealed Classes have been added to the Java Language. Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them.

Sealed Classes were proposed by JEP 360 and delivered in JDK 15 as a preview feature. They were proposed again, with refinements, by JEP 397 and delivered in JDK 16 as a preview feature. Now in JDK 17, Sealed Classes are being finalized with no changes from JDK 16.

For further details, see JEP 409.

See JDK-8260514

specification
➜ **JEP 406: Pattern Matching for switch (Preview)**
Enhance the Java programming language with pattern matching for switch expressions and statements, along with extensions to the language of patterns. 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.

For further details, see JEP 406.

See JDK-8213076

client-libs/2d

➜ **JEP 382: New macOS Rendering Pipeline**

The Java 2D API used by the Swing APIs for rendering, can now use the new Apple Metal accelerated rendering API for macOS.

This is currently disabled by default, so rendering still uses OpenGL APIs, which are deprecated by Apple but still available and supported.

To enable Metal, an application should specify its use by setting the system property:

```
-Dsun.java2d.metal=true
```

Use of Metal or OpenGL is transparent to applications since this is a difference of internal implementation and has no effect on Java APIs. The metal pipeline requires macOS 10.14.x or later. Attempts to set it on earlier releases will be ignored.

For further details, see JEP 382.

See JDK-8238361

client-libs/javax.swing

➜ **New API for Accessing Large Icons**

A new method, `javax.swing.filechooser.FileSystemView.getSystemIcon(File, int, int)`, is available in JDK 17 that enables access to higher quality icons when possible. It is fully implemented for the Windows platform; however, results on other platforms might vary and will be enhanced later. For example, by using the following code:

```
  FileSystemView fsv = FileSystemView.getFileSystemView();

  Icon icon = fsv.getSystemIcon(new File("application.exe"), 64, 64);
  JLabel label = new JLabel(icon);
```

The user can obtain a higher quality icon for the "application.exe" file. This icon is suitable for creating a label that can be better scaled in a HighDPI environment.

See JDK-8182043

core-libs/java.net

➜ **DatagramSocket Can Be Used Directly to Join Multicast Groups**

`java.net.DatagramSocket` has been updated in this release to add support for joining multicast groups. It now defines `joinGroup` and `leaveGroup` methods to join and leave multicast groups. The class level API documentation of `java.net.DatagramSocket` has been updated to explain how a plain `DatagramSocket` can be configured and used to join and leave multicast groups.

This change means that the `DatagramSocket` API can be used for multicast applications without needing to use the legacy `java.net.MulticastSocket` API. The `MulticastSocket` API works as before, although most of its methods are deprecated.

More information on the rationale of this change can be seen in the CSR JDK-8260667.

See JDK-8237352

core-libs/java.nio

➜ **Add support for UserDefinedFileAttributeView on macOS**

The file system provider implementation on macOS has been updated in this release to support extended attributes. The `java.nio.file.attribute.UserDefinedFileAttributeView` API can now be used to obtain a view of a file's extended attributes. This (optional) view was not supported in previous JDK releases.

See JDK-8030048

core-libs/java.util

➜ **JEP 356: Enhanced Pseudo-Random Number Generators**

Provide new interface types and implementations for pseudorandom number generators (PRNGs), including jumpable PRNGs and an additional class of splittable PRNG algorithms (LXM).

For further details, see JEP 356.

See JDK-8193209

hotspot/compiler

➜ **Modernization of Ideal Graph Visualizer**

Ideal Graph Visualizer (IGV), a tool to explore visually and interactively the intermediate representation used in the HotSpot VM C2 just-in-time (JIT) compiler, has been modernized. Enhancements include:

- Support for running IGV on up to JDK 15 (the latest version supported by IGV's underlying NetBeans Platform)
- Faster, Maven-based IGV build system
- Stabilization of block formation, group removal, and node tracking
- More intuitive coloring and node categorization in default filters
- Ranked quick node search with more natural default behavior

The modernized IGV is *partially* compatible with graphs generated from earlier JDK releases. It supports basic functionality such as graph loading and visualization, but auxiliary functionality such as node clustering and coloring might be affected.

Details about building and running IGV are available in the `src/utils/IdealGraphVisualizer/README.md` file in the tool's source directory.

See JDK-8254145

tools/javadoc(tool)

➜ **Source Details in Error Messages**

When JavaDoc reports an issue in an input source file, it displays the source line for the issue, and a line containing a caret (^) pointing to the position on the line, in a manner similar to compiler (`javac`) diagnostic messages.

In addition, logging and other "info" messages are now written to the standard error stream, leaving the standard output stream to be used for output that is specifically requested by command-line options, such as command-line help.

See JDK-8267126

tools/javadoc(tool)

➜ **New Page for "New API" and Improved "Deprecated" Page**

JavaDoc can now generate a page summarizing the recent changes in an API. The list of recent releases to be included is specified with the `--since` command-line option. These values are used to find the declarations with matching `@since` tags to be included on the new page. The `--since-label` command-line option provides text to use in the heading of the "New API" page.

On the page that summarizes deprecated items, you can view items grouped by the release in which they were deprecated.

See JDK-8263468

core-libs

➜ **JEP 412: Foreign Function & Memory API (Incubator)**

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.

For further details, see JEP 412.

See JDK-8265033

core-libs
➜ **Console Charset API**
`java.io.Console` has been updated to define a new method that returns the `Charset` for the console. The returned Charset may be different from the one returned from `Charset.defaultCharset()` method. For example, it returns `IBM437` while `Charset.defaultCharset()` returns `windows-1252` on Windows (en-US). Refer to the CSR for more detail.

See JDK-8264208

core-libs/java.io:serialization
➜ **JDK Flight Recorder Event for Deserialization**
It is now possible to monitor deserialization of objects using JDK Flight Recorder (JFR). When JFR is enabled and the JFR configuration includes deserialization events, JFR will emit an event whenever the running program attempts to deserialize an object. The deserialization event is named `jdk.Deserialization`, and it is disabled by default. The deserialization event contains information that is used by the serialization filter mechanism; see the ObjectInputFilter specification. Additionally, if a filter is enabled, the JFR event indicates whether the filter accepted or rejected deserialization of the object. For further information about how to use the JFR deserialization event, see the article Monitoring Deserialization to Improve Application Security. For reference information about using and configuring JFR, see the JFR Runtime Guide and JFR Command Reference sections of the JDK Mission Control documentation.

See JDK-8261160

core-libs/java.io:serialization
➜ **JEP 415: Implement Context-Specific Deserialization Filters**
JEP 415: Context-Specific Deserialization Filters allows applications to configure context-specific and dynamically-selected deserialization filters via a JVM-wide filter factory that is invoked to select a filter for each individual deserialization operation.

The Java Core Libraries Developers Guide for Serialization Filtering describes use cases and provides examples.

See JDK-8264859

core-libs/java.lang
➜ **System Property for Native Character Encoding Name**
A new system property `native.encoding` has been introduced. This system property provides the underlying host environment's character encoding name. For example, typically it has `UTF-8` in Linux and macOS platforms, and `Cp1252` in Windows (en-US). Refer to the CSR for more detail.

See JDK-8265989

core-libs/java.time
➜ **Add java.time.InstantSource**
A new interface `java.time.InstantSource` has been introduced. This interface is an abstraction from `java.time.Clock` that only focuses on the current instant and does not refer to the time zone.

See JDK-8266846

core-libs/java.util
➜ **Hex Formatting and Parsing Utility**
`java.util.HexFormat` provides conversions to and from hexadecimal for primitive types and byte arrays. The options for delimiter, prefix, suffix, and uppercase or lowercase are provided by factory methods returning HexFormat instances.

See JDK-8251989

hotspot/compiler
➜ **Experimental Compiler Blackholes Support**
The experimental support for Compiler Blackholes is added. These are useful for low-level benchmarking, to avoid dead-code elimination on the critical paths, without affecting the benchmark performance. Current support is implemented as CompileCommand, accessible as `-XX:CompileCommand=blackhole,<method>`, with the plan to eventually graduate it to a public API.

JMH is already able to auto-detect and use this facility when instructed/available. Please consult JMH documentation for the next steps.

See JDK-8259316

hotspot/compiler
➜ **New Class Hierarchy Analysis Implementation in the HotSpot JVM**
A new Class Hierarchy Analysis implementation is introduced in the HotSpot JVM. It features enhanced handling of abstract and default methods which improves inlining decisions made by the JIT-compilers. The new implementation supersedes the original one and is turned on by default.

To help diagnose possible issues related to the new implementation, the original implementation can be turned on by specifying the `-XX:+UnlockDiagnosticVMOptions -XX:-UseVtableBasedCHA` command-line flags.

The original implementation may be removed in a future release.

See JDK-8266074

hotspot/compiler
➜ **JEP 391: macOS/AArch64 Port**
macOS 11.0 now supports the AArch64 architecture. This JEP implements support for the macos-aarch64 platform in the JDK. One of the features added is support for the W^X (write xor execute) memory. It is enabled only for macos-aarch64 and can be extended to other platforms at some point. The JDK can be either cross-compiled on an Intel machine or compiled on an Apple M1-based machine.

For further details, see JEP 391.

See JDK-8251280

hotspot/runtime
➜ **Unified Logging Supports Asynchronous Log Flushing**
To avoid undesirable delays in a thread using unified logging, the user can now request that the unified logging system operate in asynchronous mode. This is done by passing the command-line option `-Xlog:async`. In asynchronous logging mode, log sites enqueue all logging messages to a buffer. A standalone thread is responsible for flushing them to the corresponding outputs. The intermediate buffer is bounded. On buffer exhaustion, the enqueuing message is discarded. The user can control the size of the intermediate buffer by using the command-line option `-XX:AsyncLogBufferSize=<bytes>`.

See JDK-8229517

infrastructure/build
➜ **macOS on ARM Early Access Available**
A new macOS is now available for ARM systems. The ARM port should behave similarly to the Intel port. There are no known feature differences. When reporting issues on macOS, please specify if using ARM or x64.

See JDK-8266858

security-libs/java.security
➜ **Provide Support for Specifying a Signer in Keytool -genkeypair Command**
The `-signer` and `-signerkeypass` options have been added to the `-genkeypair` command of the `keytool` utility. The `-signer` option specifies the keystore alias of a `PrivateKeyEntry` for the signer and the `-signerkeypass` option specifies the password used to protect the signer's private key. These options allow `keytool -genkeypair` to sign the certificate by using the signer's private key. This is especially useful for generating a certificate with a key agreement algorithm as its public key algorithm.

See JDK-8260693

security-libs/javax.crypto
➜ **SunJCE Provider Supports KW and KWP Modes With AES Cipher**

The SunJCE provider has been enhanced to support the AES Key Wrap Algorithm (RFC 3394) and the AES Key Wrap with Padding Algorithm (RFC 5649). In earlier releases, the SunJCE provider supported RFC 3394 under the "AESWrap" cipher algorithm that could only be used to wrap and unwrap keys. With this enhancement, two block cipher modes, KW and KWP, have been added that support data encryption/decryption and key wrap/unwrap by using AES. Please check the "SunJCE provider" section of the "JDK Providers Documentation" guide for more details.

See JDK-8248268

security-libs/javax.crypto:pkcs11
➜ **New SunPKCS11 configuration properties**
SunPKCS11 provider adds new provider configuration attributes to better control native resources usage. The SunPKCS11 provider consumes native resources in order to work with native PKCS11 libraries. To manage and better control the native resources, additional configuration attributes are added to control the frequency of clearing native references as well as whether to destroy underlying PKCS11 Token after logout.

The 3 new attributes for SunPKCS11 provider configuration file are:

1. `destroyTokenAfterLogout` (boolean, defaults to false) If set to true, when `java.security.AuthProvider.logout()` is called upon the SunPKCS11 provider instance, the underlying Token object will be destroyed and resources will be freed. This essentially renders the SunPKCS11 provider instance unusable after logout() calls. Note that a PKCS11 provider with this attribute set to true should not be added to the system provider list since the provider object is not usable after a logout() method call.

2. `cleaner.shortInterval` (integer, defaults to 2000, in milliseconds) This defines the frequency for clearing native references during busy period, i.e. how often should the cleaner thread processes the no-longer-needed native references in the queue to free up native memory. Note that the cleaner thread will switch to the 'longInterval' frequency after 200 failed tries, i.e. when no references are found in the queue.

3. `cleaner.longInterval` (integer, defaults to 60000, in milliseconds) This defines the frequency for checking native reference during non-busy period, i.e. how often should the cleaner thread check the queue for native references. Note that the cleaner thread will switch back to the 'shortInterval' value if native PKCS11 references for cleaning are detected.

See JDK-8240256

security-libs/javax.crypto:pkcs11
➜ **SunPKCS11 Provider Supports ChaCha20-Poly1305 Cipher and ChaCha20 KeyGenerator if Supported by PKCS11 Library**
SunPKCS11 provider is enhanced to support the following crypto services and algorithms when the underlying PKCS11 library supports the corresponding PKCS#11 mechanisms:

- ChaCha20 KeyGenerator <=> CKM_CHACHA20_KEY_GEN mechanism
- CHACHA20-POLY1305 Cipher <=> CKM_CHACHA20_POLY1305 mechanism
- CHACHA20-POLY1305 AlgorithmParameters <=> CKM_CHACHA20_POLY1305 mechanism
- CHACHA20 SecretKeyFactory <=> CKM_CHACHA20_POLY1305 mechanism

See JDK-8255410

security-libs/javax.net.ssl
➜ **Configurable Extensions With System Properties**
Two new system properties have been added. The system property, `jdk.tls.client.disableExtensions`, is used to disable TLS extensions used in the client. The system property, `jdk.tls.server.disableExtensions`, is used to disable TLS extensions used in the server. If an extension is disabled, it will be neither produced nor processed in the handshake messages.

The property string is a list of comma separated standard TLS extension names, as registered in the IANA documentation (for example, server_name, status_request, and signature_algorithms_cert). Note that the extension names are case sensitive. Unknown, unsupported, misspelled and duplicated TLS extension name tokens will be ignored.

Please note that the impact of blocking TLS extensions is complicated. For example, a TLS connection may not be able to be established if a mandatory extension is disabled. Please do not disable mandatory extensions, and do not use this feature unless you clearly understand the impact.

See JDK-8217633

security-libs/org.ietf.jgss:krb5
➜ **Use permitted_enctypes if default_tkt_enctypes or default_tgs_enctypes is not present**
Use permitted_enctypes as the default value of default_tkt_enctypes or default_tgs_enctypes if any of the them are not defined in krb5.conf.

See JDK-8262389

tools/javadoc(tool)
➜ **"Related Packages" on a Package Summary Page**
The summary page for a package now includes a section listing any "related packages". The set of related packages is determined heuristically on common naming conventions, and may include the following:

- The "parent" package (that is, the package for which a package is a subpackage)
- Sibling packages (that is, other packages with the same parent package)
- Any subpackages

The related packages need not all be in the same module.

See JDK-8260388

TOP

Removed Features and Options

This section describes the APIs, features, and options that were removed in Java SE 17 and JDK 17. The APIs described here are those that are provided with the Oracle JDK. It includes a complete implementation of the Java SE 17 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 17 and JDK 17 is the Java SE 17 (JSR 392) Platform Specification, which documents changes to the specification made between Java SE 16 and Java SE 17. 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 17.See CSRs Approved for JDK 17 for the list of CSRs closed in JDK 17.

core-libs
➜ **JEP 403: Strongly Encapsulate JDK Internals**
Strongly encapsulate all internal elements of the JDK, except for critical internal APIs such as `sun.misc.Unsafe`.

With this change, the `java` launcher option `--illegal-access` is obsolete. If used on the command line it causes a warning message to be issued, and otherwise has no effect. Existing code that must use internal classes, methods, or fields of the JDK can still be made to work by using the `--add-opens` launcher option, or the `Add-Opens` JAR-file manifest attribute, to open specific packages.

For further details, please see JEP 403.

See JDK-8266851

security-libs/java.security
➜ **Removed Telia Company's Sonera Class2 CA Certificate**
The following root certificate has been removed from the cacerts truststore:

```
+ Telia Company

  + soneraclass2ca
    DN: CN=Sonera Class2 CA, O=Sonera, C=FI
```

See JDK-8225081

core-libs
➜ **Removal of sun.misc.Unsafe::defineAnonymousClass**
`sun.misc.Unsafe::defineAnonymousClass` API has been removed in JDK 17. The API replacement is `java.lang.invoke.MethodHandles.Lookup::defineHiddenClass` and `java.lang.invoke.MethodHandles.Lookup::defineHiddenClassWithClassData`.

See JDK-8243287

core-libs/java.rmi
➜ **JEP 407: Remove RMI Activation**
The Remote Method Invocation (RMI) Activation mechanism has been removed. RMI Activation was an obsolete part of RMI that has been optional since Java SE 8. RMI Activation was deprecated for removal by JEP 385 in Java SE 15, and it was removed from this release by JEP 407. The `rmid` tool has also been removed. See JEP 385 for background, rationale, risks, and alternatives. The rest of RMI remains unchanged.

See JDK-8263550

hotspot/compiler
➜ **JEP 410: Remove the Experimental AOT and JIT Compiler**
AOT Compiler related code in HotSpot VM has been removed. Using HotSpot VM options defined by JEP295 produce "Unrecognized VM option" error on VM initialization.

For further details, see JEP 410.

See JDK-8263327

TOP

Deprecated Features and Options

Additional sources of information about the APIs, features, and options deprecated in Java SE 17 and JDK 17 include:

- The Deprecated API page identifies all deprecated APIs including those deprecated in Java SE 17.
- The Java SE 17 (JSR 392) specification documents changes to the specification made between Java SE 16 and Java SE 17 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 17. See CSRs Approved for JDK 17 for the list of CSRs closed in JDK 17.

client-libs/java.awt
➜ **JEP 398: Deprecate the Applet API for Removal**
JEP 398: Deprecate the Applet API for Removal. It is essentially irrelevant since all web-browser vendors have either removed support for Java browser plug-ins or announced plans to do so.

The Applet API was previously deprecated, though not for removal, by JEP 289 in Java 9.

See JDK-8256145

security-libs/java.security
➜ **JEP 411: Deprecate the Security Manager for Removal**
The Security Manager and APIs related to it have been deprecated and will be removed in a future release. To ensure that developers and users are aware that the Security Manager is deprecated for removal, the Java runtime issues a warning at startup if the Security Manager is enabled on the command line via `java -Djava.security.manager`. The Java runtime also issues a warning at run time if the Security Manager is enabled dynamically via the `System::setSecurityManager` API. These warnings cannot be disabled.

See JEP 411 for more information and a list of APIs that have been deprecated for removal.

See JDK-8264713

security-libs/org.ietf.jgss:krb5
➜ **Deprecate 3DES and RC4 in Kerberos**
The `des3-hmac-sha1` and `rc4-hmac` Kerberos encryption types (etypes) are now deprecated and disabled by default. Users can set `allow_weak_crypto = true` in the `krb5.conf` configuration file to re-enable them (along with other weak etypes including `des-cbc-crc` and `des-cbc-md5`) at their own risk. To disable a subset of the weak etypes, users can list preferred etypes explicitly in any of the `default_tkt_enctypes`, `default_tgs_enctypes`, or `permitted_enctypes` settings.

See JDK-8139348

core-libs/java.net
➜ **Deprecate the Socket Implementation Factory Mechanism**
The following static methods used to set the system-wide socket implementation factories have been deprecated:

- `static void ServerSocket.setSocketFactory(SocketImplFactory fac)`
- `static void Socket.setSocketImplFactory(SocketImplFactory fac)`
- `static void DatagramSocket.setDatagramSocketImplFactory(DatagramSocketImplFactory fac)`

These API points were used to statically configure a system-wide factory for the corresponding socket types in the `java.net` package. These methods have mostly been obsolete since Java 1.4.

See JDK-8235139

hotspot/jvmti
➜ **Deprecate JVM TI Heap functions 1.0**
The following JVM TI functions have been deprecated in this release:

- `IterateOverObjectsReachableFromObject`
- `IterateOverReachableObjects`
- `IterateOverHeap`
- `IterateOverInstancesOfClass`

These functions were superseded in JVM TI version 1.2 (Java SE 6) by more powerful and flexible versions. These functions will be changed to return an error in a future release to indicate that they are no longer implemented/supported. The VM flags `-Xlog:jvmti=trace and -XX:TraceJVMTI=<function_name>` can be used to identify any residual usages of these functions. For example, `-Xlog:jvmti=trace -XX:TraceJVMTI=IterateOverHeap` is one way to get trace output when IterateOverHeap is used.

See JDK-8268241

TOP

Known Issues

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

xml/jaxp
➜ **JDK XSLT Transformer Limitations**
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.

See JDK-8290347

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.

core-libs/javax.naming
➜ **New System and Security Properties to Control Reconstruction of Remote Objects by JDK's Built-in JNDI RMI and LDAP Implementations**
`jdk.jndi.object.factoriesFilter`: This system and security property allows a serial filter to be specified that controls the set of object factory classes permitted to instantiate objects from object references returned by naming/directory systems. The factory class named by the reference instance is matched against this filter during remote reference reconstruction. The filter property supports pattern-based filter syntax with the format specified by JEP 290. This property applies both to the JNDI/RMI and the JNDI/LDAP built-in provider implementations. The default value allows any object factory class specified in the reference to recreate the referenced object.

`com.sun.jndi.ldap.object.trustSerialData`: This system property allows control of the deserialization of java objects from the `javaSerializedData` LDAP attribute. To prevent deserialization of java objects from the attribute, the system property can be set to `false` value. By default, deserialization of java objects from the `javaSerializedData` attribute is allowed.

JDK-8244473 (not public)

core-libs/java.util:collections
➜ **TreeMap.computeIfAbsent Mishandles Existing Entries Whose Values Are null**
Enhancement JDK-8176894 inadvertently introduced erroneous behavior in the `TreeMap.computeIfAbsent` method. The other `TreeMap` methods that were modified by this enhancement are unaffected. The erroneous behavior is that, if the map contains an existing mapping whose value is null, the `computeIfAbsent` method immediately returns null. To conform with the specification, `computeIfAbsent` should instead call the mapping function and update the map with the function's result.

See JDK-8259622

install
➜ **Change to Package Names in Linux RPM/DEB Installers**
On the Linux platform, the names of JDK packages provided by Java RPM and DEB installers have been changed. Names of JDK packages follow the `jdk-<feature_release_version>` pattern instead of the `jdk-<update_release_version>` pattern that was previously used. For example, the new names of JDK 11, 16, and 17 packages are `jdk-11`, `jdk-16`, and `jdk-17` respectively.

The change to package names disables side-by-side installation of multiple JDKs of the same release family. Only one JDK per release family can be installed on a system with RPM and DEB installers.

If a user wants to have multiple update releases from the same family, the user must download the `tar.gz` bundles.

JDK-8266653 (not public)

install/install
➜ **Updated List of Capabilities Provided by JDK RPMs**
The following capabilities have been removed from the list of what OracleJDK/OracleJRE RPMs provide: `xml-commons-api`, `jaxp_parser_impl`, and `java-fonts`. This clean-up of the list resolves existing and potential conflicts with modular RPMs.

There are other RPMs providing these capabilities, so there should be no impact on packages that depend on them. Package managers can use other rpms to satisfy the dependencies provided by the OracleJDK/OracleJRE RPMs before this change.

JDK-8263575 (not public)

install/install
➜ **ADDLOCAL=ToolsFeature,SourceFeature Argument No Longer Needed For Windows JDK Installer**
The `ADDLOCAL=ToolsFeature,SourceFeature` argument is no longer needed for the JDK installer silent mode. All required files are now installed by default.

JDK-8262043 (not public)

security-libs/java.security
➜ **Added 2 HARICA Root CA Certificates**
The following root certificates have been added to the cacerts truststore:

```
+ HARICA

  + haricarootca2015
    DN: CN=Hellenic Academic and Research Institutions RootCA 2015, O=Hellenic Academic and Research Institutions Cert. Authority, L=Athens, C=GR

  + haricaeccrootca2015
    DN: CN=Hellenic Academic and Research Institutions ECC RootCA 2015, O=Hellenic Academic and Research Institutions Cert. Authority, L=Athens, C=GR
```

See JDK-8256421

security-libs/java.security
➜ **Disable SHA-1 JARs**
JARs signed with SHA-1 algorithms are now restricted by default and treated as if they were unsigned. This applies to the algorithms used to digest, sign, and optionally timestamp the JAR. It also applies to the signature and digest algorithms of the certificates in the certificate chain of the code signer and the Timestamp Authority, and any CRLs or OCSP responses that are used to verify if those certificates have been revoked.

In order to reduce the compatibility risk for applications that have been previously timestamped or use private CAs, there are two exceptions to this policy:

Any JAR signed with SHA-1 algorithms and timestamped prior to January 01, 2019 will not be restricted.
Any JAR signed with a SHA-1 certificate that does not chain back to a Root CA included by default in the JDK `cacerts` keystore will not be restricted.
These exceptions may be removed in a future JDK release.

Users can, at their own risk, remove these restrictions by modifying the `java.security` configuration file (or overriding it using the `java.security.properties` system property) and removing "SHA1 jdkCA & usage SignedJAR & denyAfter 2019-01-01" from the `jdk.certpath.disabledAlgorithms` security property and "SHA1 jdkCA & denyAfter 2019-01-01" from the

`jdk.jar.disabledAlgorithms` security property.

See JDK-8196415

security-libs/javax.xml.crypto
➜ **Enable XML Signature Secure Validation Mode by Default**
The XML Signature secure validation mode has been enabled by default (previously it was not enabled by default unless running with a security manager). When enabled, validation of XML signatures are subject to stricter checking of algorithms and other constraints as specified by the `jdk.xml.dsig.secureValidationPolicy` security property.

If necessary, and at their own risk, applications can disable the mode by setting the `org.jcp.xml.dsig.secureValidation` property to `Boolean.FALSE` with the `DOMValidateContext.setProperty()` API.

See JDK-8259801

security-libs/javax.xml.crypto
➜ **Disable SHA-1 XML Signatures**
XML signatures that use SHA-1 based digest or signature algorithms have been disabled by default. SHA-1 is no longer a recommended algorithm for digital signatures. If necessary, and at their own risk, applications can workaround this policy by modifying the `jdk.xml.dsig.secureValidationPolicy` security property and re-enabling the SHA-1 algorithms.

See JDK-8259709

core-libs
➜ **RegEx Pattern Matching Loses Character Class After Intersection (&&) Operator**
This release fixes a buggy behavior in regular expression pattern intersection. In prior releases, if a nested character class were included in some intersections after the intersection (`&&`) operator, it would be ignored and not included in the generated matcher from the pattern. This change brings the behavior in line with the intersection regex patterns seen in Ruby.

See JDK-8037397

core-libs/java.lang:reflect
➜ **Remove Vestiges of Intermediate JSR 175 Annotation Format**
When annotations were added to the platform in Java SE 5.0, early builds used a different representation of annotations in the class file than the final format. Support for this intermediate format has now been removed. Reading an annotation from a class file using the intermediate format which differs from the final format yields an exception similar to:

```
java.lang.reflect.GenericSignatureFormatError: Signature Parse error: Expected Field Type Signature
```

Recompiling the sources or otherwise regenerating the class file to conform to the proper format will resolve the issue.

See JDK-8265591

core-libs/java.net
➜ **URL FTP Protocol Handler: IPv4 Address Validation in Passive Mode**
Client-side FTP support in the Java platform is available through the FTP URL stream protocol handler, now referred to as the FTP Client.

The following system property has been added for validation of server addresses in `FTP` passive mode.

- `jdk.net.ftp.trustPasvAddress`.

In this release, the FTP Client has been enhanced to reject an address sent by a server, in response to a `PASV` command from the FTP Client, when that address differs from the address which the FTP Client initially connected.

To revert to the prior behavior, the `jdk.net.ftp.trustPasvAddress` system property can be set to `true`. The affect of setting this property is that the FTP Client accepts and uses the address value returned in reply to a `PASV` command

JDK-8258432 (not public)

core-libs/java.nio
➜ **New Implementation of java.nio.channels.Selector on Microsoft Windows**
The Windows implementation of the `java.nio.channels.Selector` API has been replaced in this release to use a new more scalable implementation. No behavior or compatibility issues were observed during testing of the new implementation. The old implementation has not been removed and the JDK can be configured to use the old implementation, if needed, by running with `–Djava.nio.channels.spi.SelectorProvider=sun.nio.ch.WindowsSelectorProvider` on the command line.

See JDK-8266369

core-libs/java.util
➜ **Extra '0' in java.util.Formatter for '%012a' Conversion With a Sign Character**
In previous releases, formatter conversions with a `%a` conversion that used the `0` padding flag and a width specifier would produce paddings containing too many zeros if a leading sign or space character was also specified by their respective flags. This has been fixed so that paddings no longer include too many leading zeros.

See JDK-8262351

core-libs/java.util:collections
➜ **Collections.unmodifiable* Methods Are Idempotent for Their Corresponding Collection**
The `unmodifiable*` methods in `java.util.Collections` will no longer re-wrap a given collection with an unmodifiable view if that collection has already been wrapped by same method.

See JDK-6323374

core-libs/java.util:i18n
➜ **Support for CLDR Version 39**
Locale data based on Unicode Consortium's CLDR has been upgraded to version 39. For the detailed locale data changes, please refer to the Unicode Consortium's CLDR release notes:

- http://cldr.unicode.org/index/downloads/cldr-39

See JDK-8258794

core-libs/java.util:i18n
➜ **ISO 639 Language Codes for Hebrew/Indonesian/Yiddish**
Historically, Java has used old/obsolete ISO 639 language codes for Hebrew/Indonesian/Yiddish languages to maintain compatibility. From Java 17, the default codes are the current codes. For example, "he" is now the language code for "Hebrew" instead of "iw". A new system property has also been introduced to revert to the legacy behavior. If `–Djava.locale.useOldISOCodes=true` is specified on the command line, it behaves the same way as the prior releases.

See JDK-8263202

core-svc/java.lang.instrument
➜ **Requirements of an Agent's premain Method Changed to Conform to the Specification**
The `java.lang.instrument` implementation has been changed in this release to require that agent `premain` and `agentmain` methods are public. The specification has always required this, but it was not enforced. Attempting to run with an agent where these methods are not public will fail with an exception such as:

```
java.lang.IllegalAccessException: method <fully-qualified-class-name>.premain must be declared public.
```

A related change in this release is that the `premain` and `agentmain` methods must be defined in the agent class. The implementation no longer searches for these methods in superclasses.

See JDK-8165276

docs/release_notes
➜ **XML Implementation Specific Features and Properties**

Documentation for Implementation Specific Features and Properties has been added to the `java.xml` module summary. Along with the existing properties, two new properties are introduced in JDK 17. The following section describes the changes in more detail:

1. Added javadoc for the XML processing limits.
XML processing limits were introduced in JDK 7u45 and JDK 8. They were previously documented in the Java Tutorial Processing Limits section.

The definitions for these limits have been added to the `java.xml` module summary. See JDK-8261670.

2. Moved the javadoc for `JAXP Lookup Mechanism` to the `java.xml` module summary.
The javadoc for `JAXP Lookup Mechanism` has been moved to the module summary. The original javadoc in JAXP factories are replaced with a link to that section in the module summary.

See JDK-8261673.

3. Added a property to control the newline after the XML header for DOM LSSerializer.
The DOM Load and Save `LSSerializer` did not have an explicit control for whether or not the XML Declaration ends with a newline. In this release, a JDK implementation specific property, `jdk.xml.isStandalone`, and its corresponding System property, `jdk.xml.isStandalone`, have been added to control the addition of a newline and acts independently without having to set the pretty-print property. This property can be used to reverse the incompatible change introduced in Java SE 7 Update 4 with an update of Xalan 2.7.1 in which a newline is omitted after the XML header.

Usage:

```
// to set the property, get an instance of LSSerializer

LSSerializer ser = impl.createLSSerializer();
// the isStandalone property is effective whether or not pretty-print is set
ser.getDomConfig().setParameter("format-pretty-print", pretty ? true : false);
ser.getDomConfig().setParameter("jdk.xml.isStandalone", standalone ? true : false);

// to use the System property, set it before initializing a LSSerializer
System.setProperty("jdk.xml.isStandalone", standalone ? “true” : "false");

// to clear the property, place the line anywhere after the LSSerializer is initialized
System.clearProperty("jdk.xml.isStandalone");
```
See JDK-8249867.

4. Added a property to control the newline after the XML header for XSLTC Serializer`java.xml`.
The XSLTC Serializer supported a property, `http://www.oracle.com/xml/is-standalone`, introduced through JDK-7150637, to control whether or not the XML Declaration ends with a newline. It is, however, not compliant with the new specification for Implementation Specific Features and Properties. In order to maintain compatibility, the legacy property is preserved, and a new property, `jdk.xml.xsltcIsStandalone`, along with its corresponding System property, `jdk.xml.xsltcIsStandalone`, have been created to perform the same function for the XSLTC Serializer as the `isStandalone` property for DOMLS LSSerializer. Note that the former has an extra prefix `xsltc` to avoid conflict with the later in case it is set through the System property.

Usage:

```
// to set the property, get an instance of the Transformer

Transformer transformer = getTransformer(…);
// the isStandalone property is effective whether or not pretty-print is set
transformer.setOutputProperty(OutputKeys.INDENT, pretty ? "yes" : "no");
transformer.setOutputProperty("jdk.xml.xsltcIsStandalone", standalone ? "yes" : "no");

// to use the System property, set it before initializing a Transformer
System.setProperty("jdk.xml.xsltcIsStandalone", standalone ? "yes" : "no");

// to clear the property, place the line anywhere after the Transformer is initialized
System.clearProperty("jdk.xml.xsltcIsStandalone");
```
See JDK-8260858.

5. Added existing features and properties and standardizing the prefix to `jdk.xml`.
Existing features and properties have been added to the `Implementation Specific Features and Properties` tables in the `java.xml` module summary. All of the features and properties, existing and new, now have a prefix of `jdk.xml` as redefined in the `Naming Convention` section. System properties are searchable in the Java API documentation by the full name, such as `jdk.xml.entityExpansionLimit`.

See JDK-8265252.

See JDK-8261856

hotspot/gc
➜ **Segmentation Fault Error on 9th and 10th Generation Intel® Core™ Processors**
When running Java on 9th and 10th Gen Intel® Core™ processors, a segmentation fault indicating invalid permissions for a mapped object may be observed. A workaround is included that reduces the frequency of the occurrences.

See JDK-8263710

hotspot/gc
➜ **Parallel GC Enables Adaptive Parallel Reference Processing by Default**
Parallel GC now ergonomically determines the optimal number of threads to use for processing `java.lang.ref.Reference` instances during garbage collection. The option `–XX:ParallelRefProcEnabled` is now `true` (enabled) by default.

The change improves this phase of the garbage collection pause significantly on machines with more than one thread available for garbage collection.

If you experience increased garbage collection pauses, you can revert to the original behavior by specifying `–XX:–ParallelRefProcEnabled` on the command line.

The ergonomics of `java.lang.ref.Reference` processing can be tuned by using the experimental option `–XX:ReferencesPerThread` (default value: 1000).

See JDK-8204686

security-libs/java.security
➜ **New System Property Added to Enable the OCSP Nonce Extension**
A new system property, `jdk.security.certpath.ocspNonce`, has been added to enable the OCSP Nonce Extension. This system property is disabled by default, and can be enabled by setting it to the value `true`. If set to `true`, the JDK implementation of `PKIXRevocationChecker` includes a nonce extension containing a 16 byte nonce with each OCSP request. See RFC 8954 for more details on the OCSP Nonce Extension.

See JDK-8256895

security-libs/java.security
➜ **Updated keytool to Create AKID From SKID of Issuing Certificate as Specified by RFC 5280**
The `gencert` command of the `keytool` utility has been updated to create AKID from the SKID of the issuing certificate as specified by RFC 5280.

See JDK-8257497

security-libs/java.security
➜ **Updated Specifications of KeyStoreSpi.engineStore(KeyStore.LoadStoreParameter) and KeyStore.store(KeyStore.LoadStoreParameter) Methods**

The specifications of the `KeyStoreSpi.engineStore(KeyStore.LoadStoreParameter param)` and `KeyStore.store(KeyStore.LoadStoreParameter param)` methods have been updated to specify that an `UnsupportedOperationException` is thrown if the implementation does not support the `engineStore()` operation. This change adjusts the specification to match the existing behavior.

See JDK-8246005

security-libs/java.security
➜ **jarsigner Tool Warns if Weak Algorithms Are Used in Signer's Certificate Chain**
The `jarsigner` tool has been updated to warn users when weak keys or cryptographic algorithms are used in certificates of the signer's certificate chain.

See JDK-8259401

security-libs/javax.net.ssl
➜ **SocketExceptions Are Not Wrapped Into SSLExceptions in SSLSocketImpl**
This release reverts the behavior of SSLSocketImpl and SSLTransport introduced by JDK-8196584. SocketException will now be thrown as is instead of being suppressed into an SSLException.

See JDK-8259662

specification/language
➜ **JEP 306: Restore Always-Strict Floating-Point Semantics**
Floating-point operations are now consistently *strict*, rather than having both "strict" floating-point semantics (`strictfp`) and subtly different "default" floating-point semantics. This restores the original floating-point semantics of the language and VM, matching the semantics before the introduction of "strict" and "default" floating-point modes in Java SE 1.2.

For further details, see JEP 306.

See JDK-8175916

tools/javadoc(tool)
➜ **Improved Nested Class Summary**
When a class or interface has nested classes or interfaces, the list is improved to show the kind of class or interface, such as enum class, record class, annotation interface, as appropriate.

See JDK-8266044

tools/javadoc(tool)
➜ **Improved Package Summary Pages**
The summary page for a package has been restructured to display the different kinds of classes and interfaces in a single tabbed table, instead of a series of separate tables. Additional links have been provided in the navigation bar at the top of the page, to aid in faster navigation to different parts of the page.

See JDK-8263507

tools/javadoc(tool)
➜ **Improved Output for @see Tags**
When a declaration has a series of `@see` tags, the output is generated in the form of an HTML `<ul>` list, instead of a simple comma-separated list of links. The style of the list depends on the number and kind of the links.

See JDK-8262992

tools/javadoc(tool)
➜ **Ids Used by the Standard Doclet**
"Multi-word" ids in the HTML generated by the Standard Doclet have been converted to a uniform style of lowercase words separated by hyphens. This primarily affects the ids used to navigate within the generated documentation and does not affect the ids used for field and method declarations, and which may be used in external pages to reference such declarations within the documentation.

See JDK-8261976

tools/javadoc(tool)
➜ **Improved "Help" Page**
The content of the "Help" page generated by the Standard Doclet has been revised, improved, and new information added.

- There is a new "Navigation" section that provides general information on how to navigate around the documentation.
- Information about the different kinds of pages has been gathered into a new section, along with new information about pages that were not previously documented.
- There is a brief "table of contents" at the top of the page that provides links to all of the sections and subsections on the page.

In addition, the HELP link in the navigation bar for each kind of page now links directly to the section on the Help page for that kind of page.

See JDK-8263198

tools/javadoc(tool)
➜ **Legal Headers for Generated Files**
The set of files generated by the Standard Doclet typically includes some files with associated licensing requirements. The Standard Doclet now provides support for including the associated legal files, with default behavior for the common case and a new command-line option (`--legal-notices`) to override that behavior when appropriate.

See JDK-8259530

tools/javadoc(tool)
➜ **Check for Empty Paragraphs**
DocLint (invoked from `javac` and `javadoc` with the `-Xdoclint` option) now checks for constructs that lead to empty paragraphs in the generated documentation, which might be flagged by an HTML validator. The most common cause is the redundant use of `<p>` at the end of a block of text.

See JDK-8258957

tools/javadoc(tool)
➜ **DocLint Reports Missing "descriptions"**
DocLint detects and reports documentation comments that do not have any description about the associate declaration, before any block tags that may be present. (DocLint is a feature of the `javac` and `javadoc` tools, to detect and report issues in documentation comments.)

See JDK-8272374

core-libs/java.lang:class_loading
➜ **URLClassLoader No Longer Throws Undocumented IllegalArgumentException From getResources and findResources**
In the event that there is a problem getting a resource, `URLClassLoader.getResource()` and `findResource()` now return `null` instead of throwing an undocumented `IllegalArgumentException`. The same is true of `Enumerations` obtained from `URLClassLoader.getResources()` and `URLClassLoader.findResources()`.

This behavior conforms with the long-standing specification. The situation would typically occur on Windows, due to the use of a Windows-style path (`"c:/windows"`).

See JDK-8262277

core-libs/java.lang
➜ **Less Ambiguous Processing of ProcessBuilder Quotes on Windows**

In the `java.lang.ProcessBuilder` implementation on Windows, the system property `jdk.lang.Process.allowAmbiguousCommands=false` ensures, for each argument, that double-quotes are properly encoded in the command string passed to Windows `CreateProcess`. An argument with a final trailing double-quote preceded by a backslash is encoded as a literal double-quote; previously, the argument including the double-quote would be joined with the next argument. An empty argument is encoded as a pair of double-quotes ("") resulting in a zero length string passed for the argument to the process; previously, it was silently ignored. An argument containing double-quotes, other than first and last, is encoded to preserve the double-quotes when

passed to the process; previously, the embedded double-quotes would be dropped and not passed to the process. If a security manager is set, such as in WebStart applications, double-quotes are encoded as described. When there is no security manager, there is no change to existing behavior; the `jdk.lang.Process.allowAmbiguousCommands` property can be set to `true`: `jdk.lang.Process.allowAmbiguousCommands=true` or `false`. If left unset, it is the same as setting it to `true`.

JDK-8250568 (not public)