Java

JDK 16 Release Notes

JDK 16 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 16 and Java SE 16. 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 16 (JSR 391) 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 16 (JSR 391) specification provides links to:

Annex 1: The complete Java SE 16 API Specification.

Annex 2: An annotated API specification showing the exact differences relative to Java SE 16. Informative background for these changes may be found in the list of approved Change Specification Requests for this release.

Annex 3: Java SE 16 Editions of The Java Language Specification and The Java Virtual Machine Specification. The Java SE 16 Editions contain all corrections and clarifications made since the Java SE 15 Editions, as well as additions for new features.

You should be aware of the content in that document as well as the items described in this page.

The descriptions on this Release Note page also identify potential compatibility issues that you might encounter when migrating to JDK 16. The Kinds of Compatibility page on the OpenJDK wiki identifies three types of potential compatibility issues for Java programs used in these descriptions:

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 16 for the list of CSRs closed in JDK 16 and the Compatibility & Specification Review (CSR) page on the OpenJDK wiki for general information about compatibility.

IANA Data 2020d

JDK 16 contains IANA time zone data version 2020d. For more information, refer to Timezone Data Versions in the JRE Software.

TOP

What's New in JDK 16 - New Features and Enhancements

This section describes some of the enhancements in Java SE 16 and JDK 16. In some cases, the descriptions provide links to additional detailed information about an issue or a change. The APIs described here are those that are provided with the Oracle JDK. It includes a complete implementation of the Java SE 16 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 16 and JDK 16 is the Java SE 16 (JSR 391) Platform Specification, which documents the changes to the specification made between Java SE 15 and Java SE 16. 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 16.

core-libs
➜ JEP 389: Foreign Linker API (Incubator)
Introduce an API that offers statically-typed, pure-Java access to native code. This API, together with the Foreign-Memory API (JEP 393), will considerably simplify the otherwise error-prone process of binding to a native library.

For further details, see JEP 389.

See JDK-8249755

core-libs
➜ JEP 396: Strongly Encapsulate JDK Internals by Default
Strongly encapsulate all internal elements of the JDK by default, except for critical internal APIs such as `sun.misc.Unsafe`. Allow end users to choose the relaxed strong encapsulation that has been the default since JDK 9.

With this change, the default value of the launcher option `--illegal-access` is now `deny` rather than `permit`. As a consequence, existing code that uses most internal classes, methods, or fields of the JDK will fail to run. Such code can be made to run on JDK 16 by specifying `--illegal-access=permit`. That option will, however, be removed in a future release.

For further details, please see JEP 396.

See JDK-8256299

core-libs
➜ JEP 393: Foreign-Memory Access API (Third Incubator)
Introduce an API to allow Java programs to safely and efficiently access foreign memory outside of the Java heap.

For further details, see JEP 393.

See JDK-8253415

core-libs
➜ JEP 390: Warnings for Value-based Classes
Users of the *value-based classes* provided by the standard libraries—notably including users of the primitive wrapper classes—should avoid relying on the identity of class instances. Programmers are strongly discouraged from calling the wrapper class constructors, which are now deprecated for removal. New `javac` warnings discourage synchronization on value-based class instances. Runtime warnings about synchronization can also be activated, using command-line option `-XX:DiagnoseSyncOnValueBasedClasses`.

For further details, see JEP 390.

See JDK-8249100

core-libs/java.lang:reflect
➜ Add InvocationHandler::invokeDefault Method for Proxy's Default Method Support
A new method, `invokeDefault`, has been added to the `java.lang.reflect.InvocationHandler` interface to allow a default method defined in a proxy interface to be invoked.

See JDK-8159746

core-libs/java.nio
➜ JEP 380: Unix domain sockets
Provides support for Unix domain sockets (AF_UNIX) in the `java.nio.channels`, `SocketChannel`, and `ServerSocketChannel` classes.

See JEP-380 for more information.

See also the following release note for information about the limitations in the support on Windows in JDK16.

See JDK-8238588

core-libs/java.time
➜ Day Period Support Added to java.time Formats
A new formatter pattern, letter 'B', and its supporting method have been added to `java.time.format.DateTimeFormatter/DateTimeFormatterBuilder` classes. The pattern and method translate `day periods` defined in Unicode Consortium's CLDR (https://unicode.org/reports/tr35/tr35-dates.html#dayPeriods). Applications can now express periods in a day, such as "in the morning" or "at night", not just am/pm. The following example demonstrates translating the day periods:

```
DateTimeFormatter.ofPattern("B").format(LocalTime.now())
```

This example produces day period text depending on the time of the day and locale.

See JDK-8247781

core-libs/java.util.stream
➜ Add Stream.toList() Method
A new method `toList` has been added to the `java.util.Stream` interface. This introduces a potential source incompatibility with classes that implement or interfaces that extend the `Stream` interface and that also statically import a `toList` method from elsewhere, for example, `Collectors.toList`. References to such methods must be changed to use a qualified name instead of a static import.

See JDK-8180352

hotspot/compiler
➜ JEP 338: Vector API (Incubator)

Provides an initial iteration of an incubator module, `jdk.incubator.vector`, to express vector computations that reliably compile at runtime to optimal vector hardware instructions on supported CPU architectures and thus achieve superior performance to equivalent scalar computations.

For further details, see JEP 338.

See JDK-8201271

hotspot/compiler

➜ Improved CompileCommand Flag

The CompileCommand flag has an option type that has been used for a collection of sub commands. These commands weren't verified for validity so spelling mistakes lead to the command being ignored. They had the form:

```
-XX:CompileCommand=option,<method pattern>,<option name>,<value type>,<value>
```

All option commands now exist as ordinary commands with this form:

```
-XX:CompileCommand=<option name>,<method pattern>,<value>
```

The option name is verified and the type is inferred. Helpful error messages are given if the command name doesn't exist, or if the value doesn't match the type of the command. All command names are case insensitive.

The old syntax for option commands can still be used. Verification that the option name, value type, and value is consistent has been added.

All available options can be listed with:

```
-XX:CompileCommand=help
```

See JDK-8256508

hotspot/gc

➜ JEP 376: ZGC Concurrent Stack Processing

The Z Garbage Collector now processes thread stacks concurrently. This allows all roots in the JVM to be processed by ZGC in a concurrent phase instead of stop-the-world pauses. The amount of work done in ZGC pauses has now become constant and typically not exceeding a few hundred microseconds.

For further details, see JEP 376.

See JDK-8239600

hotspot/gc

➜ Concurrently Uncommit Memory in G1

This new feature is always enabled and changes the time when G1 returns Java heap memory to the operating system. G1 still makes sizing decisions during the GC pause, but offloads the expensive work to a thread running concurrently with the Java application.

See JDK-8236926

hotspot/jfr

➜ New jdk.ObjectAllocationSample Event Enabled by Default

A new JFR event, `jdk.ObjectAllocationSample`, is introduced to allow always-on, low-overhead allocation profiling. The event is enabled in both the `default` and `profile` configurations, with a maximum rate of 150 and 300 events/s, respectively. Events `jdk.ObjectAllocationInNewTLAB` and `jdk.ObjectAllocationOutsideTLAB` also detail memory allocations, but have comparatively higher overhead. They were previously disabled in the `default` configuration but are now also disabled in the `profile` configuration to reduce the overhead further.

The impact of this change can be seen in JDK Mission Control (JMC) 8.0, or earlier releases, where the 'TLAB Allocations' page will be missing data. The recommendation is to upgrade to a later version of JMC when the `jdk.ObjectAllocationSample` support is available. Meanwhile, it is possible to enable the `jdk.ObjectAllocationInNewTLAB` and `jdk.ObjectAllocationOutsideTLAB` events in the JMC recording wizard, or edit the .jfc file manually.

See JDK-8257602

hotspot/runtime

➜ JEP 387: Elastic Metaspace

JEP 387 "Elastic Metaspace" overhauls the VM-internal metaspace- and class-space-implementation. Less memory is used for class metadata. The savings effect is mostly noticeable in scenarios involving lots of small grained class loaders. Upon class unloading, memory is timely returned to the operating system.

A switch is added to fine-tune metaspace reclamation: `-XX:MetaspaceReclaimPolicy=(balanced|aggressive|none)`. `balanced`, the default, causes the VM to reclaim memory while keeping computational overhead minimal; `aggressive` moderately increases the reclaim rate at the cost of somewhat more expensive bookkeeping; `none` switches reclamation off altogether.

The switches `InitialBootClassLoaderMetaspaceSize` and `UseLargePagesInMetaspace` have been deprecated.

See JDK-8251158

security-libs/java.security

➜ Signed JAR Support for RSASSA-PSS and EdDSA

This enhancement includes two main changes:

The JarSigner API and the `jarsigner` tool now support signing a JAR file with an RSASSA-PSS or EdDSA key.

Instead of signing the `.SF` file directly, `jarsigner` creates a SignerInfo signedAttributes field which contains ContentType, MessageDigest, SigningTime, and CMSAlgorithmProtection. The field will not be generated if an alternative signing mechanism is specified by the `jarsigner -altsigner` option. Please note that although this field was not generated by `jarsigner` before this code change, it has always been supported when parsing the signature. This means newly signed JAR files with the field can be verified by earlier JDK releases.

See JDK-8242068

security-libs/java.security

➜ SUN, SunRsaSign, and SunEC Providers Supports SHA-3 Based Signature Algorithms
SUN, SunRsaSign, and SunEC provider have been enhanced to support SHA-3 based signature algorithms. DSA signatures, RSA, and ECDSA signature implementations with SHA-3 family of digests are now supported through these providers. In addition, RSASSA-PSS signature implementation from SunRsaSign provider can recognize SHA-3 family of digests when specified in signature parameters.

See JDK-8172366

security-libs/java.security
➜ jarsigner Preserves POSIX File Permission and symlink Attributes
When signing a file that contains POSIX file permission or symlink attributes, `jarsigner` now preserves these attributes in the newly signed file but warns that these attributes are unsigned and not protected by the signature. The same warning is printed during the `jarsigner -verify` operation for such files.

Note that the `jar` tool does not read/write these attributes. This change is more visible to tools like `unzip` where these attributes are preserved.

See JDK-8218021

security-libs/java.security
➜ Added -trustcacerts and -keystore Options to keytool -printcert and -printcrl Commands
The `-trustcacerts` and `-keystore` options have been added to the `-printcert` and `-printcrl` commands of the `keytool` utility. The `-printcert` command does not check for the weakness of a certificate's signature algorithm if it is a trusted certificate in the user's keystore or in the `cacerts` keystore. The `-printcrl` command verifies the CRL using a certificate from the user's keystore or the `cacerts` keystore, and will print out a warning if it cannot be verified.

See JDK-8244148

security-libs/javax.crypto
➜ SunPKCS11 Provider Supports SHA-3 Related Algorithms
The SunPKCS11 provider has been updated with SHA-3 algorithm support. Additional KeyGenerator support for Hmac using message digests other than SHA-3 has also been added. When the corresponding PKCS11 mechanisms are supported by the underlying PKCS11 library, the SunPKCS11 provider now supports the following additional algorithms:

MessageDigest: SHA3-224, SHA3-256, SHA3-384, SHA3-512
Mac: HmacSHA3-224, HmacSHA3-256, HmacSHA3-384, HmacSHA3-512
Signature: SHA3-224withDSA, SHA3-256withDSA, SHA3-384withDSA, SHA3-512withDSA, SHA3-224withDSAinP1363Format, SHA3-256withDSAinP1363Format, SHA3-384withDSAinP1363Format, SHA3-512withDSAinP1363Format, SHA3-224withECDSA, SHA3-256withECDSA, SHA3-384withECDSA, SHA3-512withECDSA, SHA3-224withECDSAinP1363Format, SHA3-256withECDSAinP1363Format, SHA3-384withECDSAinP1363Format, SHA3-512withECDSAinP1363Format, SHA3-224withRSA, SHA3-256withRSA, SHA3-384withRSA, SHA3-512withRSA, SHA3-224withRSASSA-PSS, SHA3-256withRSASSA-PSS, SHA3-384withRSASSA-PSS, SHA3-512withRSASSA-PSS.
KeyGenerator: HmacMD5, HmacSHA1, HmacSHA224, HmacSHA256, HmacSHA384, HmacSHA512, HmacSHA512/224, HmacSHA512/256, HmacSHA3-224, HmacSHA3-256, HmacSHA3-384, HmacSHA3-512.

See JDK-8242332

security-libs/javax.net.ssl
➜ Improve Certificate Chain Handling
A new system property, `jdk.tls.maxHandshakeMessageSize`, has been added to set the maximum allowed size for the handshake message in TLS/DTLS handshaking. The default value of the system property is 32768 (32 kilobytes).

A new system property, `jdk.tls.maxCertificateChainLength`, has been added to set the maximum allowed length of the certificate chain in TLS/DTLS handshaking. The default value of the system property is 10.

JDK-8245417 (not public)

security-libs/javax.net.ssl
➜ Improve Encoding of TLS Application-Layer Protocol Negotiation (ALPN) Values
Certain TLS ALPN values couldn't be properly read or written by the SunJSSE provider. This is due to the choice of Strings as the API interface and the undocumented internal use of the UTF-8 Character Set which converts characters larger than U+00007F (7-bit ASCII) into multi-byte arrays that may not be expected by a peer.

ALPN values are now represented using the network byte representation expected by the peer, which should require no modification for standard 7-bit ASCII-based character Strings. However, SunJSSE now encodes/decodes String characters as 8-bit ISO_8859_1/LATIN-1 characters. This means applications that used characters above U+000007F that were previously encoded using UTF-8 may need to either be modified to perform the UTF-8 conversion, or set the Java security property *`jdk.tls.alpnCharset`* to "UTF-8" revert the behavior.

See JDK-8254631

security-libs/javax.net.ssl
➜ TLS Support for the EdDSA Signature Algorithm
The SunJSSE provider now supports the use of the EdDSA signature algorithm. Specifically SunJSSE may use certificates containing EdDSA keys for server side and client side authentication and may use certificates signed with the EdDSA algorithm. Additionally, EdDSA signatures are supported for TLS handshake messages that require digital signatures.

See JDK-8166596

tools/javac
➜ JEP 397: Sealed Classes (Second Preview)
Sealed classes and interfaces have been previewed again in JDK 16, initially added to the Java language in JDK 15. Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them.

For further details, see JEP 397.

See JDK-8246775

tools/javac

➜ JEP 395: Records

Records have been added to the Java language. Records are a new kind of class in the Java language. They act as transparent carriers for immutable data with less ceremony than normal classes.

Since nested classes were first introduced to Java, with the exception of static final fields initialized by constant expressions, nested class declarations that are inner have been prohibited from declaring static members. This restriction applies to non-static member classes, local classes, and anonymous classes.

JEP 384: Records (Second Preview) added support for local interfaces, enum classes, and record classes, all of which are static definitions. This was a well-received enhancement, permitting coding styles that reduce the scope of certain declarations to local contexts.

While JEP 384 allowed for static local classes and interfaces, it did not relax the restriction on static member classes and interfaces of inner classes. An inner class could declare a static interface inside one of its method bodies, but not as a class member.

As a natural next step, JEP 395 further relaxes nesting restrictions, and permits static classes, methods, fields, etc., to be declared within inner classes.

For further details, see JEP 395.

See JDK-8246771

tools/javac
➜ JEP 394: Pattern Matching for instanceof
Pattern matching for the `instanceof` operator has been made a final and permanent feature of the Java language in JDK 16. Pattern matching allows common logic in a Java program to be expressed more concisely and safely, namely the conditional extraction of components from objects.

For further details, see JEP 394.

See JDK-8250623

tools/jpackage
➜ JEP 392: Packaging Tool
Provides the `jpackage` tool, for packaging self-contained Java applications. The `jpackage tool` was introduced as an incubating tool in JDK 14 by JEP 343. It remained an incubating tool in JDK 15, to allow time for additional feedback. It has been promoted in JDK 16 from incubation to a production-ready feature. As a consequence of this transition, the name of the `jpackage` module has changed from `jdk.incubator.jpackage` to `jdk.jpackage`.

For further details, see JEP 392.

See JDK-8247768

TOP

Removed Features and Options

This section describes the APIs, features, and options that were removed in Java SE 16 and JDK 16. The APIs described here are those that are provided with the Oracle JDK. It includes a complete implementation of the Java SE 16 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 16 and JDK 16 is the Java SE 16 (JSR 391) Platform Specification, which documents changes to the specification made between Java SE 15 and Java SE 16. 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 16. See CSRs Approved for JDK 16 for the list of CSRs closed in JDK 16.

client-libs/java.awt
➜ Removal of java.awt.PeerFixer
The non-public class `java.awt.PeerFixer` has been removed in this release. This class was used to provide deserialization support of ScrollPane objects created prior JDK 1.1.1.

See JDK-8253965

hotspot/compiler
➜ Removal of Experimental Features AOT and Graal JIT
The Java Ahead-of-Time compilation experimental tool `jaotc` has been removed. Using HotSpot VM options defined by JEP295 produce a not supported option warning but will otherwise be ignored.

The experimental Java-based JIT compiler, Graal JEP317, has been removed. Attempting to use it produces a JVMCI error: `JVMCI compiler 'graal' not found`.

See JDK-8255616

hotspot/runtime
➜ Deprecated Tracing Flags Are Obsolete and Must Be Replaced With Unified Logging Equivalents
When Unified Logging was added in Java 9, a number of tracing flags were deprecated and mapped to their unified logging equivalent. These flags are now obsolete and will no longer be converted automatically to enable unified logging. To continue getting the same logging output, you must explicitly replace the use of these flags with their unified logging equivalent.

Obsoleted Option	Unified Logging Replacement
-XX:+TraceClassLoading	-Xlog:class+load=info
-XX:+TraceClassUnloading	-Xlog:class+unload=info
-XX:+TraceExceptions	-Xlog:exceptions=info

See JDK-8256718

security-libs/java.security

➜ Removed Root Certificates with 1024-bit Keys

The following root certificates with weak 1024-bit RSA public keys have been removed from the `cacerts` keystore:

```
+ alias name "thawtepremiumserverca [jdk]"

  Distinguished Name: EMAILADDRESS=premium-server@thawte.com,
CN=Thawte Premium Server CA, OU=Certification Services Division,
O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA

+ alias name "verisignclass2g2ca [jdk]"
  Distinguished Name: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only",
OU=Class 2 Public Primary Certification Authority - G2,
O="VeriSign, Inc.", C=US

+ alias name "verisignclass3ca [jdk]"
  Distinguished Name: OU=Class 3 Public Primary Certification Authority, O="VeriSign, Inc.", C=US

+ alias name "verisignclass3g2ca [jdk]"
  Distinguished Name: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only",
OU=Class 3 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US

+ alias name "verisigntsaca [jdk]"
  Distinguished Name: CN=Thawte Timestamping CA, OU=Thawte Certification, O=Thawte, L=Durbanville, ST=Western Cape, C=ZA
```

See JDK-8243559

security-libs/javax.crypto

➜ Removal of Legacy Elliptic Curves

The SunEC provider no longer supports the following elliptic curves that are either obsolete or not implemented using modern formulas and techniques:

```
secp112r1, secp112r2, secp128r1, secp128r2, secp160k1, secp160r1, secp160r2,

secp192k1, secp192r1, secp224k1, secp224r1, secp256k1, sect113r1, sect113r2, sect131r1,
sect131r2, sect163k1, sect163r1, sect163r2, sect193r1, sect193r2, sect233k1, sect233r1, sect239k1,
sect283k1, sect283r1, sect409k1, sect409r1, sect571k1, sect571r1, X9.62 c2tnb191v1,
X9.62 c2tnb191v2, X9.62 c2tnb191v3, X9.62 c2tnb239v1, X9.62 c2tnb239v2, X9.62 c2tnb239v3,
X9.62 c2tnb359v1, X9.62 c2tnb431r1, X9.62 prime192v2, X9.62 prime192v3, X9.62 prime239v1,
X9.62 prime239v2, X9.62 prime239v3, brainpoolP256r1 brainpoolP320r1, brainpoolP384r1,
brainpoolP512r1
```

To continue using any of these curves, users must find third-party alternatives.

See JDK-8235710

TOP

Deprecated Features and Options

Additional sources of information about the APIs, features, and options deprecated in Java SE 16 and JDK 16 include:

The Deprecated API page identifies all deprecated APIs including those deprecated in Java SE 16.
The Java SE 16 (JSR 391) specification documents changes to the specification made between Java SE 15 and Java SE 16 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 16. See CSRs Approved for JDK 16 for the list of CSRs closed in JDK 16.

core-libs/java.lang

➜ Terminally Deprecated ThreadGroup stop, destroy, isDestroyed, setDaemon and isDaemon

The `stop`, `destroy`, `isDestroyed`, `setDaemon` and `isDaemon` methods defined by `java.lang.ThreadGroup` have been terminally deprecated in this release.

`ThreadGroup::stop` is inherently unsafe and has been deprecated since Java 1.2. The method is now terminally deprecated and will be removed in a future release.

The API and mechanism for destroying a ThreadGroup is inherently flawed. The methods that support explicitly or automatically destroying a thread group have been terminally deprecated and will be removed in a future release.

See JDK-8256643

hotspot/runtime

➜ Parts of the Signal-Chaining API Are Deprecated

The signal-chaining facility was introduced in JDK 1.4 and supported three different Linux signal-handling API's: `sigset`, `signal` and `sigaction`. Only `sigaction` is a cross-platform, supported, API for multi-threaded processes. Both `signal` and `sigset` are considered obsolete on those platforms that still define them. Consequently, the use of `signal` and `sigset` with the signal-chaining facility are now deprecated, and support for their use will be removed in a future release.

See JDK-8257572

security-libs/java.security
➜ Deprecated the java.security.cert APIs That Represent DNs as Principal or String Objects
The following APIs have been deprecated:

```
java.security.cert.X509Certificate.getIssuerDN()

java.security.cert.X509Certificate.getSubjectDN()
java.security.cert.X509CRL.getIssuerDN()
java.security.cert.X509CertSelector.setIssuer(String)
java.security.cert.X509CertSelector.setSubject(String)
java.security.cert.X509CertSelector.getIssuerAsString()
java.security.cert.X509CertSelector.getSubjectAsString()
java.security.cert.X509CRLSelector.addIssuerName(String)
```
These APIs either take or return Distinguished Names as `Principal` or `String` objects and can cause issues due to loss of encoding information or differences when comparing names across different Principal implementations. All of them have alternative APIs that use `X500Principal` objects instead.

See JDK-8241003

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/java.io
➜ Line Terminator Definition Changed in java.io.LineNumberReader
The definition of line terminator has been extended to include end of stream, or one of the previously defined line terminators `'\n'`, `'\r'`, or `'\r\n'` followed immediately by end of stream. This changes behavior in certain cases. For example, a file containing the lines

```
line 1\n

line 2\n
line 3
```
would, before this change, have been considered to contain two lines, each terminated by '\n'. After this change, it is considered to contain three lines, the third being terminated by end of stream.

See JDK-8235792

core-libs/java.io:serialization
➜ Enhanced Support of Proxy Class
The deserialization of `java.lang.reflect.Proxy` objects can be limited by setting the system property `jdk.serialProxyInterfaceLimit`. The limit is the maximum number of interfaces allowed per Proxy in the stream. Setting the limit to zero prevents any Proxies from being deserialized including Annotations, a limit of less than 2 might interfere with RMI operations.

JDK-8236862 (not public)

core-libs/java.lang
➜ Support Supplementary Characters in String Case Insensitive Operations
Case insensitive operations in `java.lang.String` class now correctly do case insensitive comparisons for supplementary characters (characters which have code point values over `U+FFFF`). For details, see the updates to the methods:

```
   - compareToIgnoreCase(String other)

   - equalsIgnoreCase(String other)
   - regionMatches(boolean ignoreCase, ...)
```
For example,

```
"\ud801\udc00".equalsIgnoreCase("\ud801\udc28")
```

returns `true`, because '𐐀' ("\ud801\udc00") and '𐐨' ("\ud801\udc28") are equal to each other character in case insensitive comparison.

See JDK-8248655

core-libs/java.lang
➜ Module::getPackages Returns the Set of Package Names in This Module
`Module::getPackages` returns the set of package names for the packages in this module.

For unnamed modules, the specification and implementation have been corrected to return the names of the packages defined in the unnamed module. Prior to Java SE 16, `Module::getPackages` returned the set of package names for the packages defined in the module's class loader if the module was an unnamed module and the result included packages of other named modules defined in the module's class loader, if any.

See JDK-8256063

core-libs/java.lang:reflect
➜ Proxy Classes Are Not Open for Reflective Access

All proxy classes are not open for reflective access in Java SE 16.

Prior to Java SE 16, if `java.lang.reflect.Proxy` was used to implement only public exported proxy interfaces, the proxy class was generated in an unnamed module which was open for reflective access. In Java SE 16, the proxy class is generated in an exported package in a named module. Programs that assume the private members of a proxy class can be made accessible via `setAccessible(true)` will fail with `InaccessibleObjectException`. Proxy classes are already defined in dynamic modules in other cases since Java SE 9. Such programs would already fail when it calls `setAccessible(true)` to those proxy classes prior to this change.

See JDK-8159746

core-libs/java.net
➜ The Default HttpClient Implementation Returns Cancelable Futures
In this release, the default `HttpClient` returns *cancelable* futures.

The default `HttpClient` is created by a call to `HttpClient.newHttpClient()`, or by invoking the `build` method on builders returned by `HttpClient.newBuilder()`. The implementation of the `sendAsync` methods in the default `HttpClient` now return `CompletableFuture` objects that are *cancelable*. Invoking `cancel(true)` on a *cancelable* future that is not completed, attempts to cancel the HTTP exchange in an effort to release underlying resources as soon as possible. More information can be obtained by reading the API documentation of the `HttpClient::sendAsync` methods.

See JDK-8245462

core-libs/java.net
➜ HttpClient.newHttpClient and HttpClient.Builder.build Might Throw UncheckedIOException
Creation of an instance of `java.net.http.HttpClient` may fail with `UncheckedIOException` if the underlying resources required by the implementation cannot be allocated.

Typically, this may happen if the underlying resources required to opening a `java.nio.channels.Selector` are not available. In this case `Selector.open()` will throw an `IOException` which the default implementation of `java.net.http.HttpClient` will wrap in an `UncheckedIOException`. The API documentation of `HttpClient.newHttpClient()` and `HttpClient.newBuilder().build()` have been updated to specify that `UncheckedIOException` may be thrown if the underlying resources required by the implementation cannot be allocated. Prior to this change, an undocumented `InternalError` would have been thrown.

See JDK-8248006

core-libs/java.net
➜ `HttpPrincipal::getName` Returned Incorrect Name
The behavior of the `HttpPrincipal::getName` method has been updated to return the name in the correct format as outlined in its specification, i.e. `"realm:username"`. Previously, the method incorrectly returned `"username"` only.

See JDK-8255584

core-libs/java.nio
➜ Incomplete Support for Unix Domain Sockets in Windows 2019 Server
JEP 380: Unix-Domain Socket Channels has been integrated into JDK 16, but due to a problem with the underlying implementation in Windows 2019 Server, it has been disabled on that platform in this release. The functionality is supported on Linux, macOS and on Windows operating systems whose build number is 18362 or greater. This includes Windows 10 (Version 1903) and newer. There are no known issues with Windows 10 (Version 1903). However, some newer versions of Windows 10 may also show the problem, which affects the ability of some native tools (such as cygwin) to access and delete socket files. Other Java APIs are also affected by the problem, but most native Windows tools are not affected by it. More information can be seen in bugs JDK-8259014 and JDK-8252971. This issue will be resolved in JDK 17 and in a future update to JDK 16.

See JDK-8259014

core-libs/java.nio
➜ (fs) NullPointerException Not Thrown When First Argument to Path.of or Paths.get Is null
The var args form of `Path.of()` and `Paths.get()` method are changed in this release to throw `NullPointerException` consistently when the first parameter is null. Historically these methods missed a null check on the first parameter when invoked with more than one parameter.

See JDK-8254876

core-libs/java.nio
➜ DataInputStream:readFully now correctly throws specified exceptions
`DataInputStream.readFully(byte[] b, int off, int len)` now correctly throws specified `NullPointerException` and `IndexOutOfBoundsException`.

`DataInputStream.readFully(byte[] b, int off, int len)` is specified to throw a `NullPointerException` if `b` is `null`, and an `IndexOutOfBoundsException` if `off` is negative, `len` is negative, or `len` is greater than `b.length - off`. For example if `b` was `null` and `len` was zero, then the method would fail silently without throwing a `NullPointerException`. Also if `b` was non-`null`, `off` was negative and `len` was zero, then the method would fail silently without throwing an `IndexOutOfBoundsException`. The implementation has been corrected to throw the respective exceptions as specified.

See JDK-8245036

core-libs/java.time
➜ US/Pacific-New Zone Name Removed as Part of tzdata2020b
Following the JDK's update to tzdata2020b, the long-obsolete files named `pacificnew` and `systemv` have been removed. As a result, the "US/Pacific-New" Zone name declared in the `pacificnew` data file is no longer available for use.

Information regarding this update can be viewed at https://mm.icann.org/pipermail/tz-announce/2020-October/000059.html

See JDK-8254177

core-libs/java.util

➜ Argument Index of Zero or Unrepresentable by int Throws IllegalFormatException.
Format string specifiers now throw exceptions when given values outside of valid ranges of values.

The Formatter class in java.util defines format specifiers such as argument indexes, argument widths, and argument precisions. Numeric values that are invalid (zero for argument index) or too large (beyond the size of an int) could create unexpected results with undefined behavior. This update gives explicit value ranges for these format specifiers. Widths and indexes will be valid from [1, Integer.MAX_VALUE] and precision fields will be valid from [0, Integer.MAX_VALUE]. Values outside of these ranges will result in IllegalFormatException or one of its subclasses being thrown. Note that argument indexes of zero prior to this change did not throw an exception and produced behavior that was undefined, but had the appearance of correctness despite causing side effects that were not obvious.

See JDK-8253459

core-libs/java.util.jar
➜ Refine ZipOutputStream.putNextEntry() to Recalculate ZipEntry's Compressed Size
Prior to JDK 16, `ZipOutputStream.putnextEntry()` would not recalculate the compressed size for a compressed (DEFLATED) entry. This could result in the ZipException, "invalid entry compressed size", being thrown if the current ZLIB implementation being used when `ZipOutputStream.putNextEntry()` was called differed from the implementation at the time when the entry was added to the original ZIP file.

Starting with JDK 16, if the compressed size has not been explicitly set with the `ZipEntry.setCompressedSize(long)` method when writing a compressed (DEFLATED) entry, then the compressed size is set to the actual compressed size after deflation.

See JDK-8253952

core-libs/java.util.jar
➜ GZIPOutputStream Sets the GZIP OS Header Field to the Correct Default Value
Prior to JDK 16, GZIPOutputStream set the OS field in the GZIP header to 0 (meaning FAT filesystem), which does not match the default value specified in section 2.3.1.2 of the GZIP file format specification version 4.3 (RFC 1952).

As of JDK 16, the GZIP OS Header Field is set to 255, which is the default value as defined in RFC 1952.

See JDK-8244706

core-libs/java.util.logging
➜ java.util.logging.LogRecord Updated to Support Long Thread IDs
In this release, `java.util.logging.LogRecord` has been updated to support long thread ids.

`LogRecord::getThreadID` and `LogRecord::setThreadID` are deprecated. New accessors, `LogRecord::getLongThreadID` and `LogRecord::setLongThreadID`, are provided and should be used instead.

The serial field `threadID` has been deprecated and a new serial field `longThreadID` that can support long values has been introduced. The deprecated `threadID` field is kept in the serial form for backward compatibility. Long thread ids that are less than `Integer.MAX_VALUE` are directly mapped to `threadID` and `longThreadID` has the same value. For `longThreadID` greater than Integer.MAX_VALUE, a new negative value for the `int threadID` is synthesized by using a deterministic algorithm based on the `longThreadID` hash. The resulting value is guaranteed to be a negative value.

See JDK-8245302

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

core-libs/java.util:i18n
➜ Support for CLDR Version 38
Locale data based on Unicode Consortium's CLDR has been upgraded to their version 38. For the detailed locale data changes, please refer to the Unicode Consortium's CLDR release notes:

- http://cldr.unicode.org/index/downloads/cldr-38

See JDK-8251317

core-libs/javax.naming
➜ Added Property to Control LDAP Authentication Mechanisms Allowed to Authenticate Over Clear Connections
A new environment property, `jdk.jndi.ldap.mechsAllowedToSendCredentials`, has been added to control which LDAP authentication mechanisms are allowed to send credentials over `clear` LDAP connections - a connection not secured with TLS. An `encrypted` LDAP connection is a connection opened by using `ldaps` scheme, or a connection opened by using `ldap` scheme and then upgraded to TLS with a STARTTLS extended operation.

The value of the property, which is by default not set, is a comma separated list of the mechanism names that are permitted to authenticate over a `clear` connection. If a value is not specified for the property, then all mechanisms are allowed. If the specified value is an empty list, then no mechanisms are allowed (except for `none` and `anonymous`). The default value for this property is 'null' (i.e. `System.getProperty("jdk.jndi.ldap.mechsAllowedToSendCredentials")` returns 'null'). To explicitly permit all mechanisms to authenticate over a `clear` connection, the property value can be set to `"all"`. If a connection is downgraded from `encrypted` to `clear`, then only the mechanisms that are explicitly permitted are allowed.

The property can be supplied to the LDAP context environment map, or set globally as a system property. When both are supplied, the environment map takes precedence.

Note: `none` and `anonymous` authentication mechanisms are exempted from these rules and are always allowed regardless of the property value.

JDK-8237990 (not public)

core-libs/javax.naming

➜ LDAP Channel Binding Support for Java GSS/Kerberos

A new JNDI environment property `"com.sun.jndi.ldap.tls.cbtype"` has been added to enable TLS Channel Binding data in LDAP authentication over SSL/TLS protocol to the Windows AD server. Possible value is `"tls-server-end-point"` - Channel Binding data is created on the base of the TLS server certificate. See the module description of the `java.naming` module.

See JDK-8245527

hotspot/gc

➜ Shenandoah: Concurrent weak reference processing

The Shenandoah GC now processes all soft, weak and phantom references as well as finalizers concurrently with the running Java application, instead of at a GC pause. This reduces GC induced latency, especially with workloads that churn many such references or finalizers.

See JDK-8254315

hotspot/jvmti

➜ Make JVMTI Table Concurrent

For improved performance, JVM/TI ObjectFree events are no longer posted within GC pauses. The events are still posted as requested, and will be posted before ObjectFree events are enabled or disabled with SetNotificationMode. SetNotificationMode can be used to explicitly flush ObjectFree events, if needed.

See JDK-8212879

hotspot/runtime

➜ IncompatibleClassChangeError Exceptions Are Thrown For Failing 'final' Checks When Defining a Class

As a result of changes to section 5.3.5 of the Java Virtual Machine Specification, a `java.lang.IncompatibleClassChangeError` exception is thrown instead of a `java.lang.VerifyError` exception when defining a class whose super class is final and when defining a class that tries to override a final method.

See JDK-8243583

hotspot/runtime

➜ Object Monitors No Longer Keep Strong References to Their Associated Object

When synchronization is performed on an object, an association is established between the object and the object monitor that implements the synchronization. In the past, the reference from a monitor to its associated object was a strong reference. These strong references would be observable through JVM TI functions that walk the heap (reported as `JVMTI_HEAP_ROOT_MONITOR` or `JVMTI_HEAP_REFERENCE_MONITOR`) and in heap dumps (reported as `HPROF_GC_ROOT_MONITOR_USED`). As of this release, a weak reference is used. These are not observable to JVM TI or heap dumps. Consequently, `JVMTI_HEAP_ROOT_MONITOR`, `JVMTI_HEAP_REFERENCE_MONITOR` and `HPROF_GC_ROOT_MONITOR_USED` are longer reported.

See JDK-8247281

security-libs/java.security

➜ Added Entrust Root Certification Authority - G4 certificate

The following root certificate has been added to the cacerts truststore:

```
+ Entrust

  + entrustrootcag4
    DN: CN=Entrust Root Certification Authority - G4, OU="(c) 2015 Entrust, Inc. - for authorized use only",
           OU=See www.entrust.net/legal-terms, O="Entrust, Inc.", C=US
```

See JDK-8243321

security-libs/java.security

➜ Added 3 SSL Corporation Root CA Certificates

The following root certificates have been added to the cacerts truststore:

```
+ SSL Corporation

  + sslrootrsaca
    DN: CN=SSL.com Root Certification Authority RSA, O=SSL Corporation, L=Houston, ST=Texas, C=US

  + sslrootevrsaca
    DN: CN=SSL.com EV Root Certification Authority RSA R2, O=SSL Corporation, L=Houston, ST=Texas, C=US

  + sslrooteccca
    DN: CN=SSL.com Root Certification Authority ECC, O=SSL Corporation, L=Houston, ST=Texas, C=US
```

See JDK-8243320

security-libs/java.security

➜ Upgraded the Default PKCS12 Encryption and MAC Algorithms

The default encryption and MAC algorithms used in a PKCS #12 keystore have been updated. The new algorithms are based on AES-256 and SHA-256 and are stronger than the old algorithms that were based on RC2, DESede, and SHA-1. See the security properties starting with `keystore.pkcs12` in the `java.security` file for detailed information.

For compatibility, a new system property named `keystore.pkcs12.legacy` is defined that will revert the algorithms to use the older, weaker algorithms. There is no value defined for this property.

See JDK-8153005

security-libs/javax.net.ssl

➜ Disable TLS 1.0 and 1.1

TLS 1.0 and 1.1 are versions of the TLS protocol that are no longer considered secure and have been superseded by more secure and modern versions (TLS 1.2 and 1.3).

These versions have now been disabled by default. If you encounter issues, you can, at your own risk, re-enable the versions by removing "TLSv1" and/or "TLSv1.1" from the `jdk.tls.disabledAlgorithms` security property in the `java.security` configuration file.

See JDK-8202343

tools/javac

➜ Annotation Interfaces May Not Be Declared As Local Interfaces

Prior to JDK 16, the `javac` compiler accepted annotations declared as local interfaces. Now the JLS 16 at section 14.3 forbids it. For example, the `javac` compiler had accepted code such as:

```
    class C {

        void m() {
            @interface A {}
        }
    }
```

This code is no longer acceptable according to Section [14.3] of the JLS 16: "A local interface may be a normal interface (§9.1), but not an annotation interface (§9.6)."

To fix this issue, the `javac` compiler implementation has been synchronized with the JLS 16, so that annotation interfaces cannot be declared as local interfaces.

See JDK-8250741

tools/javac

➜ C-Style Array Declarations Are Not Allowed in Record Components

Prior to JDK 16, the `javac` compiler accepted C-style array declarations in record components. The JLS 16 now forbids it. In particular, the compiler had accepted code such as:

```
    record R(int i[]) {}
```

This code is no longer acceptable according to the specification for JDK 16. Section 8.10.1 of the JLS 16 defines the syntax of a record component as:

```
RecordComponent:

    {RecordComponentModifier} UnannType Identifier
    VariableArityRecordComponent
VariableArityRecordComponent:
    {RecordComponentModifier} UnannType { Annotation } ... Identifier
RecordComponentModifier:
    Annotation
```

which clearly forbids C-style array declarations in record components. To fix this issue, the compiler has been synchronized with the JLS 16, so that C-style array declarations are no longer allowed in record components.

See JDK-8250629

tools/javac

➜ DocLint Support Moved to jdk.javadoc Module

The code that provides the DocLint feature in *javac* has been moved from the `jdk.compiler` module to the `jdk.javadoc` module. For most users, there will be no impact. However, any user wanting to build a minimal image for running *javac* with DocLint enabled will need to ensure that the `jdk.javadoc` module is available when linking the image.

See JDK-8252712

tools/javadoc(tool)

➜ Improvements for JavaDoc Search

The interactive search feature in API documentation generated by the JavaDoc Standard Doclet has been improved to find additional matches that differ only in case. This is useful when the exact spelling or capitalization of a name is not known.

For full details on JavaDoc Search, see the link to the JavaDoc Search Specification that is found on at the bottom of the HELP page in any API documentation.

See JDK-8244535

tools/javadoc(tool)

➜ API Documentation Links to Platform Documentation

API documentation generated by the JavaDoc Standard Doclet will now automatically link to the appropriate set of platform classes, based on the value of the `--source` or `--release` option.

The behavior can be disabled by using the new `--no-platform-links` option.

The URLs for the different versions of the platform documentation can be configured by the new `--link-platform-properties` option.

See JDK-8216497

tools/javadoc(tool)

➜ Viewing API Documentation on Small Devices

API documentation generated by the JavaDoc Standard Doclet has been updated so that it can be viewed more easily on small devices.

See JDK-8248566

tools/javadoc(tool)

➜ Eliminating Duplication in Simple Documentation Comments

For many simple methods, the first sentence is often a near duplicate of the text provided for the `@return` tag. To help avoid such duplication, the `@return` tag has been enhanced so that it can be used as an inline tag at the beginning of the comment, to provide the text for both the first sentence, and the description of the return value.

See JDK-8075778

tools/javadoc(tool)

➜ "Type" Terminology in Generated Documentation

In line with guidelines described elsewhere 1, the Standard Doclet uses the following terms when generating documentation for JDK 16 or later. For consistency, the older terms will continue to be used when generating documentation for older releases.

- *Classes and Interfaces* (instead of *Types*)
- *Annotation Interface* (instead of *Annotation Type*)
- *Enum Class* (instead of *Enum*)
- *Record Class* (instead of *Record*)

See JDK-8258002

Differences Between Oracle JDK and Oracle's OpenJDK

Although we have stated the goal to have OpenJDK and Oracle JDK 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 is released under the OTN 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.