JSR 401: Java SE 26

Iris Clark

Specification Lead
iris.clark@oracle.com
December 9, 2025

JSR 401: Java SE 26

Specification

- Latest: https://cr.openjdk.org/~iris/se/26/latestSpec (soon!)
- Public Review to begin in January

Reference Implementation (RI) – JDK 26

- Latest: https://jdk.java.net/26 (build 27)
- Repository: https://github.com/openjdk/jdk branch ”jdk26”
- Rampdown Phase 1 (RDP1)
  - Feature set frozen
  - Development limited to selected bug fixes
- 10 Integrated JEPs (7 SE, 2 JDK, 1 Implementation)
- 107 approved SE CSRs
- General Availability (GA): Mar 2026

Technology Compatibility Took Kit (TCK) – JCK 26

- Code Freeze rapidly approaching
- Stabilization fork soon thereafter

Schedule

2025/06
Expert Group Formation

2025/01 – 2026/02
Public Review

2026/02 - 2026/03
Public Review –
Final Approval Ballot

2026/03
Final Release

SE JEPs in Java SE 26

Language

530	Primitive Types in Patterns, `instanceof`, & `switch` (Fourth Preview)

Security

524	PEM Encodings of Cryptographic Objects (Second Preview)

Libraries

517	HTTP/3 for the HTTP Client API
526	Lazy Constants (Second Preview)
525	Structured Concurrency (Sixth Preview)
500	Prepare to Make `final` Mean Final
504	Remove the Applet API

JEP 530: Primitive Types in Patterns, `instanceof`, & `switch` (Fourth Preview)

Enhance pattern matching to support primitive types in contexts previously restricted to reference and some integer types.

```
if (i instanceof byte b) {
                            // no loss
    ... b ...
}
```

History

- Previewed in Java SE 23; Unchanged in Java SE 24 and Java SE 25; Changes to identify a wider range of coding errors in Java SE 26 (source incompatible)

Why

- Eliminate restrictions on use of primitive type patterns to make the Java language more consistent and more expressive across types
- Eliminates code which may be lossy when narrowing between types

JEP 517: HTTP/3 for the HTTP Client API

Enhance the HTTP Client API to support the HTTP/3 protocol so that libraries and applications may use it with minimal code changes.

```
var client  = HttpClient.newBuilder();
                  .version(HttpClient.Version.HTTP_3)
                  .build();

var request = HttpClient.newBuilder(URI.create("https://openjdk.org/"))
                  .version(HttpClient.Version.HTTP_3)
                  .GET().build();
```

Why

- Enable applications to easily benefit from HTTP/3’s lower latency, less network congestion, and more reliable transport
- HTTP/3 is supported by most modern browers and deployed to more than one third of all websites (and growing)

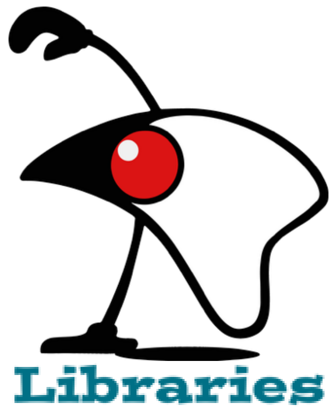
JEP 526: Lazy Constants (Second Preview)

Introduce an API for objects that hold unmodifiable data that is treated as true constants by the VM after initialization. This enables the same performance benefits of `final` fields while providing more flexible initialization timing.

```
class OrderController {
    private final LazyConstant<Logger> logger
        = LazyConstant.of(() -> Logger.create(OrderController.class));



    void submitOrder(User user, List<Product> products) {
        logger.get().info("order started");
        ...
        logger.get().info("order submitted");  }}
```



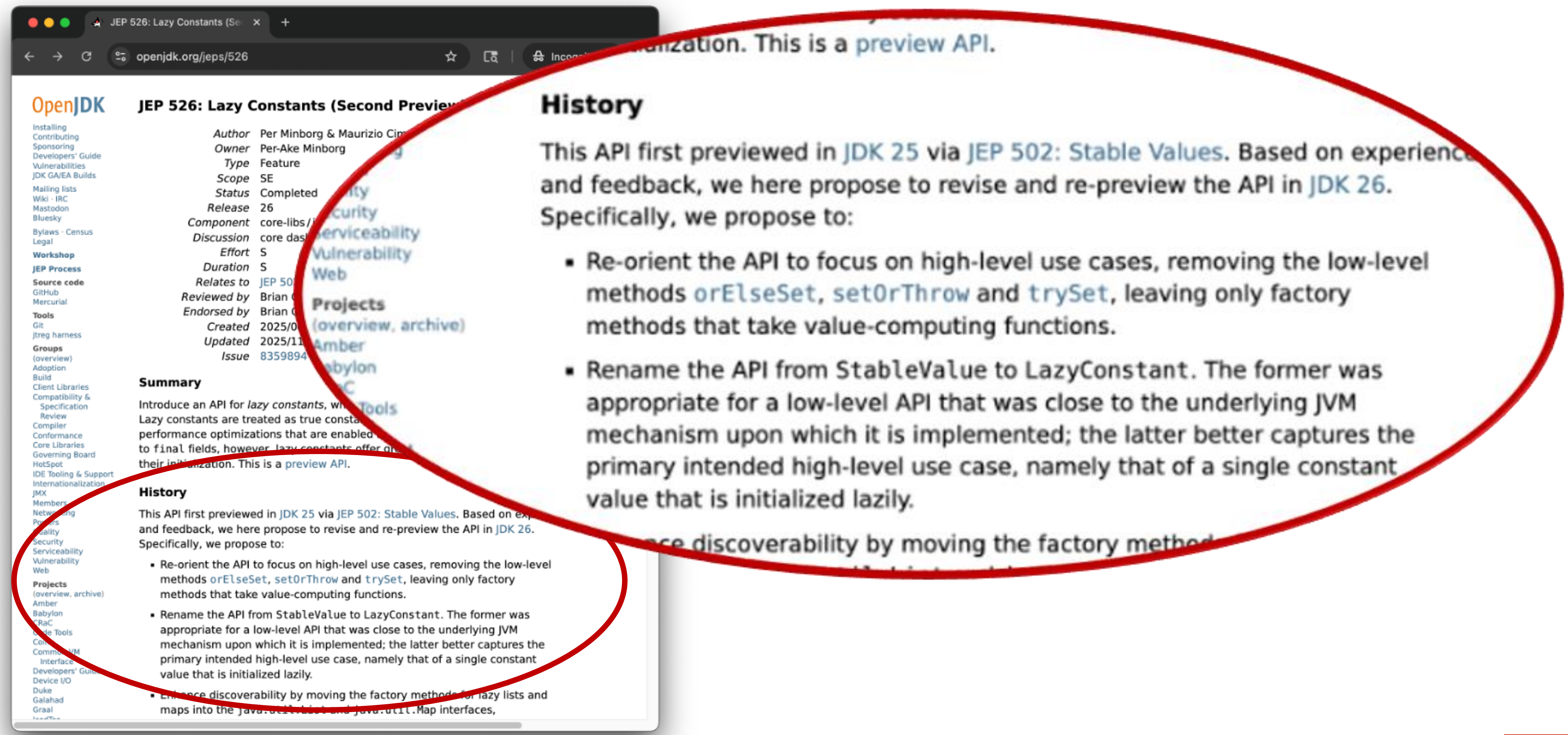
History

- Previewed in Java SE 25 as “Stable Values”; New title and significant changes in Java SE 26

Why

- Optimize Java application start-up by delaying initialization until first use
- Guarantee immutability, even in multi-threaded programs

An Aside: Preview Feature “History”



JEP 526: Lazy Constants (Second Previe

Author	Per Minborg & Maurizio Cim
Owner	Per-Ake Minborg
Type	Feature
Scope	SE
Status	Completed
Release	26
Component	core-libs /
Discussion	core das
Effort	S
Duration	S
Relates to	JEP 50
Reviewed by	Brian
Endorsed by	Brian
Created	2025/0
Updated	2025/11
Issue	8359894

Summary

Introduce an API for *lazy constants*, wh
Lazy constants are treated as true consta
performance optimizations that are enabled
to `final` fields, however lazy constants offer gr
their initialization. This is a preview API.

History

This API first previewed in JDK 25 via JEP 502: Stable Values. Based on exp and feedback, we here propose to revise and re-preview the API in JDK 26. Specifically, we propose to:

- Re-orient the API to focus on high-level use cases, removing the low-level methods `orElseSet`, `setOrThrow` and `trySet`, leaving only factory methods that take value-computing functions.
- Rename the API from `StableValue` to `LazyConstant`. The former was appropriate for a low-level API that was close to the underlying JVM mechanism upon which it is implemented; the latter better captures the primary intended high-level use case, namely that of a single constant value that is initialized lazily.
- Enhance discoverability by moving the factory methods for lazy lists and maps into the `java.util.List` and `java.util.Map` interfaces,

ization. This is a preview API.

History

This API first previewed in JDK 25 via JEP 502: Stable Values. Based on experienc and feedback, we here propose to revise and re-preview the API in JDK 26. Specifically, we propose to:

- Re-orient the API to focus on high-level use cases, removing the low-level methods `orElseSet`, `setOrThrow` and `trySet`, leaving only factory methods that take value-computing functions.
- Rename the API from `StableValue` to `LazyConstant`. The former was appropriate for a low-level API that was close to the underlying JVM mechanism upon which it is implemented; the latter better captures the primary intended high-level use case, namely that of a single constant value that is initialized lazily.
- nce discoverability by moving the factory metho

JEP 525: Structured Concurrency (Sixth Preview)

Introduce APIs to structure a task as a family of concurrent subtasks, and to coordinate them as a unit.

```
try (var scope = new StructuredTaskScope.open()) {
    Subtask<String>  subtask1 = scope.fork() -> query(left);
    Subtask<Integer> subtask2 = scope.fork() -> query(right);

    // join subtasks, propagating exceptions
    scope.join();
    // Both subtasks have succeeded, compose their results
    return new Response(subtask1.get(), subtask2.get());
} // close
```

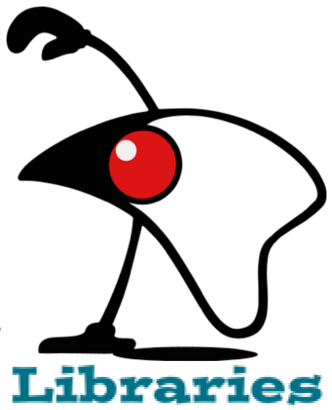
History

- Incubated in Java SE 19 and Java SE 20; Previewed in Java SE 21; Unchanged in Java SE 22, Java SE 23, and Java SE 24; Several API changes in Java SE 25; Minor changes in Java SE 26

Why

- Provide structure for large numbers of virtual threads
- Streamline error handling, improving reliability and enhancing observability

JEP 500: Prepare to Make `final` Mean Final



Issues warnings when deep reflection is used to mutate `final` fields. This change prepares for a future release when such mutations will fail by default.

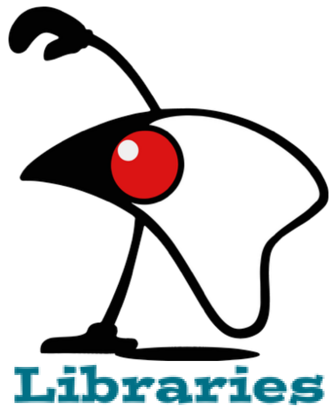
```
WARNING: Final field f in p.C has been [mutated/unreflected for mutation] by class com.foo.Bar.caller
         in module N (file:/path/to/foo.jar)
WARNING: Use --enable-final-field-mutation=N to avoid a warning
WARNING: Mutating final fields will be blocked in a future release unless final field mutation is enabled
```

- Allow code in the specified module to mutate final fields
  `--enable-final-field-mutation` *`M`*
  where *`M`* is a comma-separated list of modules on the module path or `ALL-UNNAMED` to indicate code on the class path
- Temporary option to specify how illegal final field mutation is handled
  `--illegal-final-field-mutation=X`
  where *X* is `allow`, `warn`, `debug`, or `deny`

Why

- Another step towards integrity by default as described by JEP DRAFT 8305968: Integrity by Default [https://openjdk.org/jeps/8305968]

JEP 504: Remove the Applet API



Removes the entire `java.applet` package, class `javax.swing.Japplet` and interface `java.beans.AppletInitializer`, and any API elements that reference those classes including methods in `java.beans.Beans` and `javax.swing.RepaintManager`.

History

- Deprecated in Java SE 9 via JEP 289; Deprecated `forRemoval=true` in Java SE 17 via JEP 398

Why

- Applets are obsolete, unsupported by modern browsers, and insecure without the `SecurityManager` which was permanently disabled in Java SE 24 via JEP 486

JEP 524: PEM Encodings of Cryptographic Objects (Second Preview)

Introduce an API for encoding and decoding objects that represent cryptographic keys, certificates, and certificate revocation lists into the widely-used Privacy-Enhanced Mail (PEM) transport format.

- DEREncodable – marker interface indicating that implementations support conversion to/from byte arrays in the Distinguished Encoding Rules (DER) format
- PEMEncoder and PEMDecoder – immutable, thread-safe, and reusable classes for converting to and from the PEM format

```
PEMEncoder pe = PEMEncoder.of();
byte[] pemData = pe.encode(privateKey);
```

- PEM – implementation of DEREncodable which represents PEM data by its type and Base64 form

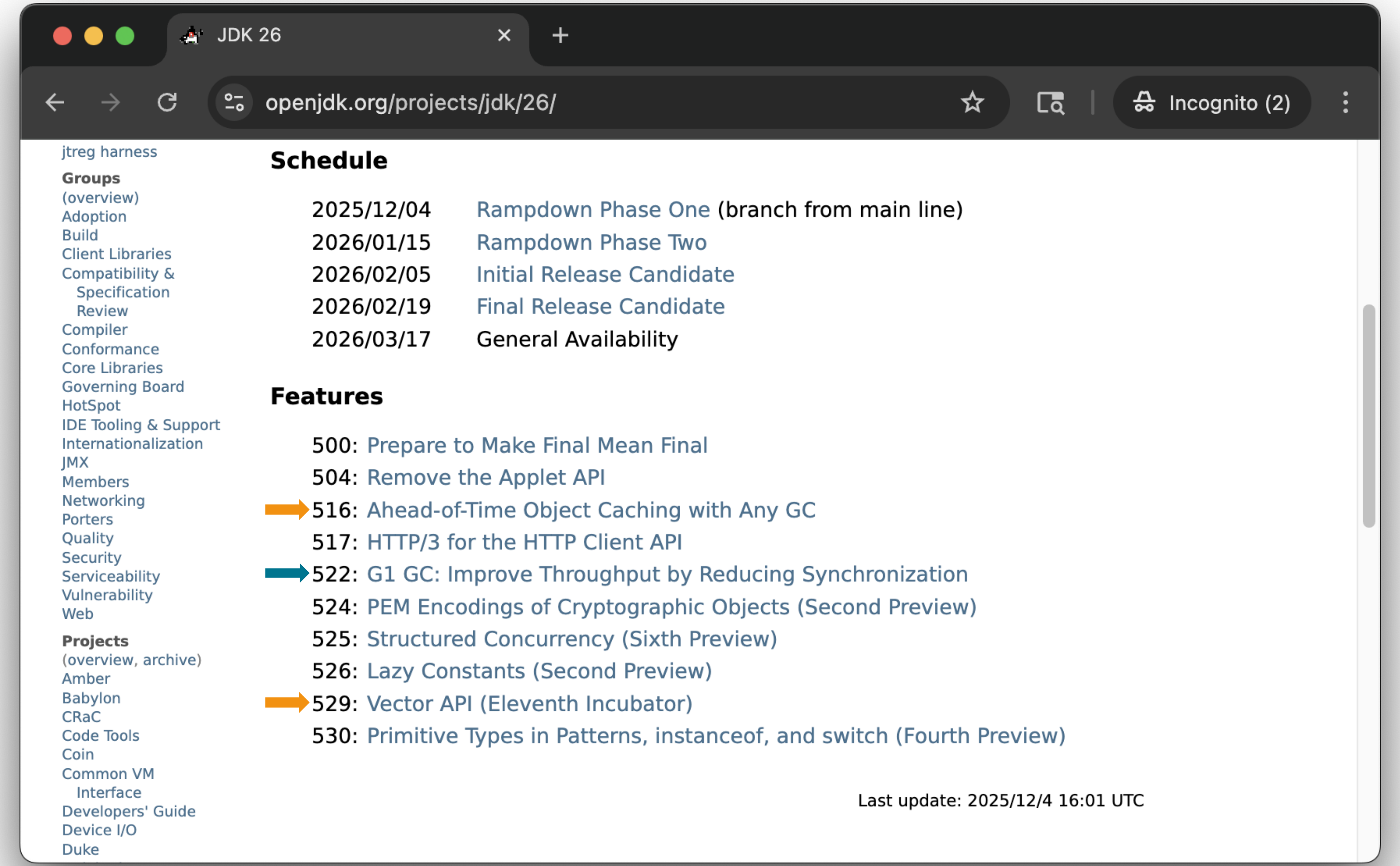
History

- Re-previewed in Java SE 26 with several API changes

Why

- Provides a concise API to convert between PEM text and cryptographic objects
- Support conversions between PEM text and cryptographic objects that have standard representations in binary formats such as PKCS#8 v1.2, PKCS#8 v2.0, and X.509

openjdk.org/projects/jdk/26



jtreg harness

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality
Security
Serviceability
Vulnerability
Web

Projects
(overview, archive)
Amber
Babylon
CRaC
Code Tools
Coin
Common VM Interface
Developers' Guide
Device I/O
Duke

Schedule

2025/12/04	Rampdown Phase One (branch from main line)
2026/01/15	Rampdown Phase Two
2026/02/05	Initial Release Candidate
2026/02/19	Final Release Candidate
2026/03/17	General Availability

Features

500: Prepare to Make Final Mean Final
504: Remove the Applet API
🟧➡ 516: Ahead-of-Time Object Caching with Any GC
517: HTTP/3 for the HTTP Client API
🟦➡ 522: G1 GC: Improve Throughput by Reducing Synchronization
524: PEM Encodings of Cryptographic Objects (Second Preview)
525: Structured Concurrency (Sixth Preview)
526: Lazy Constants (Second Preview)
🟧➡ 529: Vector API (Eleventh Incubator)
530: Primitive Types in Patterns, instanceof, and switch (Fourth Preview)

Last update: 2025/12/4 16:01 UTC

JEP Scope =
🟧➡ JDK
🟦➡ Implementation

Other notable changes in Java SE 26

107 Approved Compatibility & Specification Review (CSR) Requests
https://bugs.openjdk.org/issues/?filter=47281

2 JSR Maintenance Releases
221: JDBC API Specification [MR 5]
269: Pluggable Annotations Processing API [MR 20]

20 Removed APIs

```
java.applet package containing 4 classes (17)
java.beans.AppletInitializer (17)
java.beans.Beans.instantiate(ClassLoader,String,BeanContext,
    AppletInitializer) (17)
java.lang.Thread.stop (18)
java.net.DatagramSocketImpl.setTTL (23)
java.net.DatagramSocketImpl.getTTL (23)
java.net.MulticastSocket.setTTL (23)
java.net.MulticastSocket.getTTL (23)
java.net.MulticastSocket.send (23)
javax.imageio.spi.ServiceRegistry.finalize (18)
javax.imageio.stream.FileCacheImageInputStream.finalize (18)
javax.imageio.stream.FileImageInputStream.finalize (18)
javax.imageio.stream.FileImageOutputStream.finalize (18)
javax.imageio.stream.ImageInputStreamImpl.finalize (18)
javax.imageio.stream.MemoryCacheImageInputStream.finalize (18)
javax.management.modelmbean.DescriptorSupport.toXMLString (25)
javax.management.modelmbean.DescriptorSupport <init> (25)
javax.imageio.spi.ServiceRegistry.finalize (18)
javax.swing.Japplet (17)
javax.swing.RepaintManager.addDirtyRegion (17)
```

2 Terminally Deprecated APIs Added

```
java.net.Socket.setPerformancePreferences (26)
java.net.ServerSocket.setPerformancePreferences (26)
```

Resources

- https://openjdk.org/projects/jdk/26/spec/
    - https://jcp.org/en/jsr/detail?id=401
    - JEPs: https://bugs.openjdk.org/secure/Dashboard.jspa?selectPageId=23506
    - CSRs: https://bugs.openjdk.org/secure/Dashboard.jspa?selectPageId=23507
    - https://mail.openjdk.org/mailman/listinfo/java-se-spec-experts
    - https://jdk.java.net/26/
- https://mail.openjdk.org
- https://github.com/openjdk/jdk