JSR 400: Java SE 25

Iris Clark

Specification Lead
iris.clark@oracle.com
August 12, 2025

JSR 400: Java SE 25

Specification

- Latest: https://cr.openjdk.org/~iris/se/25/latestSpec
- Public Review: 15 Jul – 18 Aug

Reference Implementation (RI) – JDK 25

- Latest: https://jdk.java.net/25 (build 35)
- Repository: https://github.com/openjdk/jdk
- Development complete
  - Release Candidate Phase (RC)
- 18 Closed, Delivered JEPs (9 SE, 5 JDK, 4 Implementation)
- 109 approved SE CSRs
- General Availability (GA): Sep 2025

Technology Compatibility Took Kit (TCK) – JCK 25

- Code Freeze recently; first GAC soon

Schedule

2024/12
Expert Group Formation

2025/07/15 – 2025/08/18
Public Review

2025/08/19 – 2025/08/25
Public Review –
Final Approval Ballot

2025/09
Final Release

SE JEPs in Java SE 25

Language

- 511 Module Import Declarations
- 513 Flexible Constructor Bodies
- 512 Compact Source Files & Instance `main` Methods
- 507 Primitive Types in Patterns, `instanceof`, & `switch` (Third Preview)

Libraries

- 505 Structured Concurrency (Fifth Preview)
- 506 Scoped Values
- 502 Stable Values (Preview)

Security

- 510 Key Derivation Function API
- 470 PEM Encodings of Cryptographic Objects (Preview)

JEP 511: Module Import Declarations

Provide a simple means to import all public types in all packages of a module.

```
import module java.base;


public class A {
    public static void main(String... args) {
        List<Path> l = new ArrayList<>[];
        System.out.println(l.getClass().getName());
    }
}
```

History

- First previewed in Java SE 23; Refined in Java SE 24 with two additions; Unchanged in Java SE 25

Why

- Eliminate the need for multiple import-on-demand declarations when using diverse parts of an API exported by a module
- Simplify the re-use of modular libraries without requiring code to be in a module itself

JEP 513: Flexible Constructor Bodies

In Java language constructors, allow selected statements that do not reference
The instance being created to appear before invoking super(…) or this(…).

```
Class A { ... A(int i) { ... throw new IllegalArgumentException(); ... } }
Class B extends A {
    int save;
    B(int i) {
        if (i < 0) throw new IllegalArgumentException("Bad number: " + i);
        save = i;              // x belongs to “this”
        super(i);   }}
```

History

- First previewed in Java SE 22; New title and significant changes in Java SE 23; Unchanged in Java SE 24 and Java SE 25

Why

- Allows argument validation, computation, and field initialization before delegation to another constructor

JEP 512: Compact Source Files & Instance `main` Methods

Reduce syntactic complexity of simple programs for novice users.

```
void main() {
    IO.println("Hello, World!");
}
```

History

- First previewed in Java SE 21; New title and significant changes in Java SE 22; Two additions for automatic imports for implicitly declared classes in Java SE 23; New terminology and title but otherwise unchanged in Java SE 24; New title and minor improvements based in Java SE 25

Why

- “Hello, World” exposes too many concepts that may intimidate beginning programmers
- Reduce ceremony for simple programs such as scripts and command-line utilities

JEP 507: Primitive Types in Patterns, `instanceof`, & `switch` (Third 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

- Re-previewed without change in Java SE 24 and Java SE 25

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 505: Structured Concurrency (Fifth 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

Why

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

JEP 506: Scoped Values

Introduce scoped values, which enable safe and efficient sharing of immutable data within and across threads. Preferred over thread-local

```
final static ScopedValue<...> NAME = ScopedValue.newInstance();

// In some method
ScopedValue.runWhere(NAME, "duke", () -> { ... NAME.get() ... call methods ... });

// In a method called directly or indirectly from the lambda expression
... NAME.get() ...
```

History

- Incubated in Java SE 20; Previewed in Java SE 21; Unchanged in Java SE 22; One concern addresses in Java SE 23; Remove 2 methods in `ScopedValue` rendering it completely fluent in Java SE 24; Disallow `null` argument in `ScopedValue.orElse` in Java SE 25

Why

- Alternative to thread-local variables and method arguments for sharing data across components

JEP 502: Stable Values (Preview)

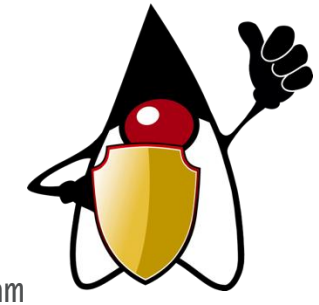
Introduce an API for objects that hold immutable data but are treated as constants by the VM after initialization, enabling the same performance optimizations that are enabled by declaring a field final but providing more flexibility for initialization timing.

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

    Logger getLogger() {
        return logger.orElseSet() -> logger.create(OrderController.class));  }


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

Why

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

JEP 510: Key Derivation Function API



Introduce an API for Key Derivation Functions (KDFs), which are cryptographic algorithms for deriving additional keys from a secret key and other data.

```
KDF hkdf = KDF.getInstance("HKDF-SHA256");       // Create a KDF for the specified algorithm

AlgorithmParameterSpec params =                  // Create an ExtractExpand parameter specification
    HKDFParamenterSpec.ofExtract()
                      .addIKM(initialKeyMaterial)
                      .addSalt(salt)
                      .thenExpand(info, 32);

SecretKey key = hkdf.deriveKey("AES", params);   // Derive a 32-byte AES key
```

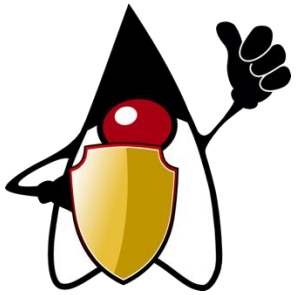
History

- Previewed in Java SE 24; Unchanged in Java SE 25.

Why

- Second step towards support of Hybrid Public Key Encryption (HPKE)
- Protects against ”harvest now, decrypt later” quantum attacks

JEP 470: PEM Encodings of Cryptographic Objects (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

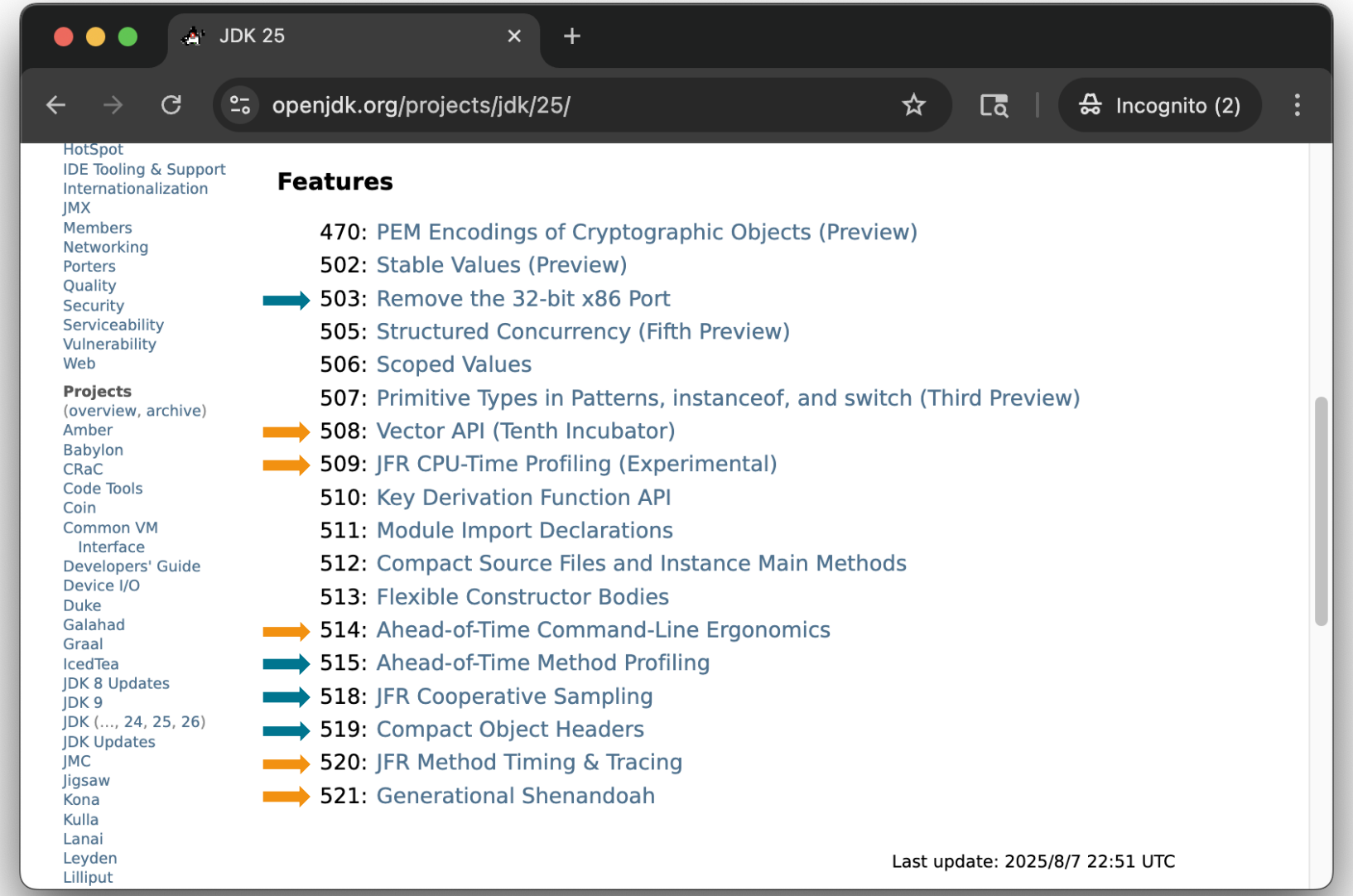
```
PEMEncoder pe = PEMEncoder.of();
byte[] pemData = pe.encode(privateKey);
```

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

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/25



JEP Scope =
- JDK
- Implementation

Other notable changes in Java SE 25

109 Approved Compatibility & Specification Review (CSR) Requests
https://bugs.openjdk.org/issues/?filter=46522

1 JSR Maintenance Release
269: Pluggable Annotations Processing API [MR 19]

2 Removed APIs
`javax.swing.platform.synth.SynthLookAndFeel.load(URL)` (21)
`javax.swing.plaf.basic.BasicSliderUI.BasicSliderUI.<init>` (23)

26 Terminally Deprecated APIs Added
- 21 `*Permission` classes which directly or indirectly extend `java.security.Permission`
- 2 classes which retrieve `Permissions`
  `java.net.HttpURLConnection.getPermission` (25)
  `java.net.URLConnection.getPermission` (25)
- 3 APIs related to parsing `ModelMBean` XML formats
  `javax.management.modelmbean.XMLParseException` (25)
  `javax.management.modelmbean.DescriptorSupport<init>` (25)
  `javax.management.modelmbean.DescriptorSupport.toXMLString` (25)

Resources

- https://openjdk.org/projects/jdk/25/spec/
  - https://jcp.org/en/jsr/detail?id=400
  - JEPs: https://bugs.openjdk.org/secure/Dashboard.jspa?selectPageId=23201
  - CSRs: https://bugs.openjdk.org/secure/Dashboard.jspa?selectPageId=23200
  - https://mail.openjdk.org/mailman/listinfo/java-se-spec-experts
  - https://jdk.java.net/25/
- https://mail.openjdk.org
- https://github.com/openjdk/jdk