
Java
25
BLING
BLING

🔐 JEP 470 — PEM Encodings of Cryptographic Objects (Preview)

- Encode/decode keys, certs, CRLs to/from PEM via simple encoder/decoder APIs.
- Supports PKCS#8, X.509; encryption/decryption of private keys; reusable, thread-safe.

```
var enc = PEMEncoder.of();
String pubText = enc.encodeToString( publicKey );

var dec = PEMDecoder.of();
ECPublicKey key = dec.decode( pubText, ECPublicKey.class );

// Encrypted private key
String pem = enc.withEncryption( password ).encodeToString( privateKey
);
ECPrivateKey pk = dec.withDecryption( password ).decode( pem,
ECPrivateKey.class );
```

💎 JEP 502 — Stable Values (Preview)

- Defer initialization safely; JVM treats set values as constants for folding.
- Thread-safe at-most-once initialization via StableValue and suppliers.
- Enable with --enable-preview (JDK 25).

```
private final StableValue<Logger> logger = StableValue.of();
Logger log() { return logger.orElseSet(() ->
Logger.create(getClass())); }

// Or at declaration site
private final Supplier<Logger> LOG = StableValue.supplier( () ->
Logger.create( getClass() ) );
void use() { LOG.get().info("hello"); }
```

🧹 JEP 503 — Remove the 32-bit x86 Port

- Drops source/build support for 32-bit x86 to simplify HotSpot and infra.
- Unblocks new features lacking 32-bit fallbacks; does not affect other 32-bit ports.
- Migration: use 64-bit JDKs; older releases retain 32-bit x86 where present.

```
// No code changes required.
// Build/CI environments should target 64-bit x86 or other supported arches.
// Example: docker image switch to amd64, aarch64, riscv64 as needed.
```

🧵 JEP 505 — Structured Concurrency (Fifth Preview)

- Treat related tasks as one unit: simpler cancellation, error handling, observability.
- Static factory open(); policies via Joiner; integrates with virtual threads.
- Enable with --enable-preview (JDK 25).

```
try (var scope = StructuredTaskScope.open()) {
   var u = scope.fork(() -> findUser());
   var o = scope.fork(() -> fetchOrder());
   scope.join(); // cancels sibling on failure
   return new Response( u.get(), o.get() );
}
```

🎯 JEP 506 — Scoped Values (Final)

- Share immutable data down-call and to child threads safely and efficiently.
- Cleaner than ThreadLocal; works great with virtual threads and structured concurrency.

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

where(CTX, ctx).run(() -> {
    service.handle();                    // CTX.get() available here
});
```

🔢 JEP 507 — Primitive Types in Patterns, instanceof, and switch (Third Preview)

- Use primitive type patterns in instanceof/switch; safe, non-lossy matching.
- switch now supports long, float, double, boolean; instanceof works with primitives.
- Enable with --enable-preview (JDK 25).

```
if (i instanceof byte b) { use(b); }      // only if exact, no data loss
switch (v) {
   case 0f -> out(5f);
   case float x when x == 1f -> out(6f + x);
   case float x -> out(7f + x);
}
if (json instanceof JsonNumber(int age)) { … }
// safe narrowing inside record pattern
```

🧬 JEP 508 — Vector API (Tenth Incubator)

- Express vector computations that compile to optimal SIMD on x64/AArch64.
- Links math funcs via FFM; adds Float16 autovectorization; shuffles with MemorySegment.

```
static final VectorSpecies<Float> S = FloatVector.SPECIES_PREFERRED;
for (int i = 0; i < S.loopBound(n); i += S.length()) {
   var a = FloatVector.fromArray(S, A, i);
   var b = FloatVector.fromArray(S, B, i);
   a.mul(a).add(b.mul(b)).neg().intoArray(C, i);
}
```

⏱️ JEP 509 — JFR CPU-Time Profiling (Experimental)

- JFR samples per CPU time on Linux for accurate CPU profiles, incl. native code.
- New jdk.CPUTimeSample event; throttle via time or rate; stream or record.

```
// At launch
java
-XX:StartFlightRecording=jdk.CPUTimeSample#enabled=true,filename=profil
e.jfr …

// Later, with jcmd
jcmd <pid> JFR.start settings=/tmp/cpu_profile.jfc duration=4m
jfr view cpu-time-hot-methods profile.jfr
```

🔑 JEP 510 — Key Derivation Function API (Final)

- Standard javax.crypto.KDF for HKDF (and future KDFs like Argon2).
- Derive keys/data deterministically; provider-extensible; supports PKCS#11 HKDF.

```
KDF hkdf = KDF.getInstance( "HKDF-SHA256" );
var params = HKDFParameterSpec.ofExtract()

.addIKM(ikm).addSalt(salt).thenExpand(info, 32);
SecretKey key = hkdf.deriveKey( "AES", params );
byte[] bytes = hkdf.deriveData(params);
```

📦 JEP 511 — Module Import Declarations (Final)

- import module M; brings all exported packages of a module into scope.
- Coexists with normal imports; resolves via specificity; great for prototyping.

```
import module java.base;    // List, Path, Stream, etc.
import module java.sql;     // java.sql.*, javax.sql.* and indirect exports

// Disambiguate when needed
import java.sql.Date;
Date d = new Date(System.currentTimeMillis());
```

📝 JEP 512 — Compact Source Files & Instance Main Methods (Final)

- Write tiny programs without boilerplate; instance main(); implicit class in compact files.
- java.lang.IO for simple console I/O; auto-import of java.base in compact files.

```
// Compact source file
void main() {
    String name = IO.readln("Name: ");
    IO.println("Hello, " + name + "!");
    var nums = List.of(1,2,3);
    nums.forEach(n -> IO.println(n));
}

// Regular class with instance main
class App { void main() { IO.println("Hi"); } }
```

🏗️ JEP 513 — Flexible Constructor Bodies (Final)

- Statements may appear before super()/this(); no reference to uninitialized this.
- Validate args early; init subclass fields before superclass sees them; safer construction.

```
class Employee extends Person {
    String officeID;
    Employee(int age, String officeID) {
        if (age < 18 || age > 67) throw new IllegalArgumentException();
        this.officeID = officeID;  // allowed in prologue
        super(age);                 // call after safe setup
        // epilogue …
    }
}
```

JEP 514 — AOT Command-Line Ergonomics 🚀

- One-step training **and** cache creation:
  `-XX:AOTCacheOutput=app.aot`
- Temp AOT config is handled for you (no leftover file)
- Keep 2-step when needed; pass creation-only flags via
  `JDK_AOT_VM_OPTIONS`
- Takeaway: simpler workflow → easier, faster startup with AOT caches

JEP 515 — Ahead-of-Time Method Profiling 📈

- Saves **method execution profiles** from training runs **into the AOT cache**
- JIT compiles hot methods **immediately at startup** → shorter warmup
- Still adapts at runtime if behavior changes
- No code changes; tiny cache growth for a notable warmup win

JEP 518 — JFR Cooperative Sampling 

- Samples stacks **only at safepoints** via cooperative requests (no risky heuristics)
- **More stable & scalable** JFR stack walking; fewer crashes
- **Minimizes safepoint bias**; falls back when in native code
- Net: better profiling fidelity with low risk

JEP 519 — Compact Object Headers 📦

- Promoted to **product feature** (no `-XX:+UnlockExperimentalVMOptions`)
- Enable with `-XX:+UseCompactObjectHeaders`
- Wins: **smaller heap**, **fewer GCs**, **lower CPU** in many workloads
- Not the default header layout

JEP 520 — JFR Method Timing & Tracing ⏱️

- **Bytecode instrumentation** to time/trace selected methods exactly
- Filter by method/class/annotation (e.g., `HashMap::resize`, `@jakarta.ws.rs.GET`)
- Configure via `StartFlightRecording` (`method-timing`, `method-trace`) or JMX
- Use surgically—**higher overhead** than sampling; great for hotspots

JEP 521 — Generational Shenandoah 🌱

- **Generational mode** is now a **product feature**
- Enable: `-XX:+UseShenandoahGC -XX:ShenandoahGCMode=generational`
- Benefits: better young/old separation → improved pauses/throughput
- Not the default; single-gen remains default