JAVA IN 2025: NAVIGATING MIGRATION, SECURITY, AND LONG-TERM RISK

A White Paper for CIOs, CISOs, and Engineering Leaders

@herodevs

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Java continues to serve as the backbone of enterprise systems in 2025, powering mission-critical workloads in banking, government, healthcare, and e-commerce. Although the language's syntax remains familiar, the surrounding ecosystem has changed dramatically. Enterprises must now rethink how they approach migration, security, and governance.
Oracle's two-year Long-Term Support (LTS) cadence now dictates enterprise planning horizons. Migration projects once handled once a decade are now recurring every four to six years. Each new LTS release introduces breaking changes in JDK internals, module removals, and architectural shifts.

Security events such as Log4Shell and Spring4Shell revealed how a single open-source dependency can destabilize thousands of applications within hours. A 2024 Synopsys study found that 74 percent of codebases contained high-risk open-source vulnerabilities, up from 48 percent just two years earlier. At the same time, Sonatype reported a 156 percent year-over-year surge in malicious open-source packages.

The question for CIOs, CISOs, and engineering leaders is no longer whether to continue relying on Java. It is how to migrate safely between LTS versions, reduce exposure in legacy environments, and implement governance frameworks that withstand regulatory scrutiny.
This white paper provides detailed analysis of migration realities, real-world breach lessons, supply-chain risk, and the economic, regulatory, and vendor dynamics shaping enterprise decisions in 2025.

JAVA'S ENDURING ROLE IN THE ENTERPRISE

Despite the rise of cloud-native languages such as Go and Rust, Java remains deeply embedded in enterprise IT. It consistently ranks among the top three most-used programming languages worldwide and boasts more than 12 million active developers.

Mission-critical systems — financial clearinghouses, electronic health record systems, e-commerce platforms, and government applications — depend on the JVM. Its longevity has been both a blessing and a liability. While stability reassures technology leaders, it also encourages delay. The "if it isn't broken, don't fix it" mindset has left countless organizations with unsupported runtimes, widening attack surfaces, and spiraling compliance risks.

THE NEW JAVA RELEASE AND SUPPORT PARADIGM

Since 2018, Oracle and the OpenJDK community have adopted a six-month feature release cadence with LTS versions every two years. The current LTS releases are Java 8, 11, 17, and 21, with Java 25 expected in September 2025.

Most enterprises adopt a “skip two releases” approach, targeting migrations every four to six years. This reduces the number of upgrades but compounds the complexity of each. For example:

- **Migrating from Java 8 to 11 requires replacing deprecated modules such as CORBA and Java EE.**
- **Migrating from 11 to 17 enforces strong encapsulation of internal APIs and deprecates the Security Manager.**
- **Migrating from 17 to 21 introduces new garbage collection strategies and virtual threads that impact concurrency models.**
- **Migrating directly to Java 25 will likely require addressing both long-standing deprecations and the complete disablement of the Security Manager.**

Licensing changes add further disruption. Oracle’s shift to per-employee subscriptions, combined with shorter free-support windows, has pushed many organizations to adopt alternatives such as Adoptium, Amazon Corretto, or Azul.

MIGRATION REALITIES AND PITFALLS

Migration is rarely a drop-in replacement. Each LTS shift can break frameworks, libraries, and deployment models. Legacy systems relying on internal reflection, SOAP bindings, or WAR deployments often require major rewrites. Performance profiles shift with new garbage collectors, demanding full benchmarking.

Enterprises that delay migrations face amplified risk. Leaping directly from Java 8 or 11 to Java 25 will likely require multi-stage upgrades, multiplying costs and elongating timelines. Organizations that embed migration planning into their IT roadmaps avoid the high costs of emergency programs.

LESSONS FROM REAL-WORLD VULNERABILITIES

The last decade has demonstrated how quickly vulnerabilities in Java ecosystems can translate into catastrophic consequences.

- **Equifax (2017):** An unpatched Apache Struts vulnerability exposed 147 million consumer records. Patch SLAs must be measured in days, not weeks.
- **Log4Shell (2021):** Exploitation began within hours of disclosure, proving that SBOM visibility and dependency governance are essential.
- **Spring4Shell (2022):** Deployment topology determined exposure, with WAR deployments on Tomcat particularly vulnerable. Security requires both patching and architectural hygiene.

These incidents illustrate that delaying modernization is no longer a viable option. Vulnerabilities cascade quickly, exploit campaigns move faster, and regulators expect proactive governance.

THE EXPANDING SUPPLY-CHAIN THREAT

Modern Java risk extends beyond the runtime itself.

- **Synopsys OSSRA 2024:** 74 percent of audited codebases contained high-risk vulnerabilities, up from 48 percent in 2022.
- **Sonatype 2024:** Over 500,000 malicious open-source packages were documented in a single year, a 156 percent increase.
- **Q2 2025:** 16,279 malicious packages were identified, many designed for credential theft and data exfiltration.
- **Targeted campaigns:** Lazarus Group exploited typosquatted packages to compromise over 36,000 developers.

The implication is clear. Even a fully patched JVM can be compromised by vulnerable frameworks, HTTP clients, or logging libraries. Governance must extend into dependency chains, with automated SBOMs, SCA scans, and repository-level defenses against typosquatting and malicious injections.

A SECURE MIGRATION BLUEPRINT

Enterprises that succeed in modernization treat migration as a structured program:

1. **Inventory:** Generate SBOMs and map dependencies.
2. **Remediation:** Replace deprecated APIs and unsupported modules.
3. **Testing:** Validate functional, performance, and security baselines.
4. **Deployment:** Roll out via canary releases, monitoring for anomalies.
5. **Governance:** Enforce patch SLAs, automate scans, and retire temporary runtime workarounds.

When immediate migration is not feasible, commercial long-term support and compensating controls such as WAF rules, runtime monitoring, and restricted egress policies allow organizations to maintain compliance while planning sustainable upgrades.

INDUSTRY EXAMPLES

- **Finance:** A major bank delayed migrating from Java 8 to 11, only to be forced into an emergency program when vendor support ended, costing millions.
- **Government:** A federal agency running Java 7 relied on extended support to maintain FedRAMP and NIST compliance while planning a phased migration.
- **Healthcare:** A hospital system incurred HIPAA penalties after unpatched Java applications failed compliance audits. Migration to Java 17 combined with backport patching resolved the issue.

THE ECONOMICS OF MIGRATION VS. SUPPORT

Direct Costs of Migration

Every migration carries significant direct costs. Developer hours must be allocated to code rewrites, dependency updates, and system testing. External consultants are often engaged at premium rates, particularly when migrations occur under time pressure. Licensing expenses for new distributions add further burden.

Hidden Costs and Opportunity Loss

Migrations also create hidden expenses. Retraining engineers on new language features, pausing innovation projects, and managing downtime during phased rollouts all translate into lost productivity. These indirect costs, often unbudgeted, can rival or exceed direct costs.

Long-Term Support as a Financial Strategy

Commercial long-term support provides stability. Instead of incurring a multi-million-dollar migration every four years, organizations can spread costs across predictable annual subscriptions. This model appeals to CFOs and CIOs seeking to stabilize budgets, avoid emergency spending, and maintain compliance without sacrificing strategic agility.

REGULATORY AND COMPLIANCE DIMENSIONS

Compliance Expectations

Frameworks such as PCI DSS, HIPAA, GDPR, and FedRAMP require timely patching of known vulnerabilities. Unsupported Java runtimes violate these expectations, triggering audit findings and potential penalties.

SBOM and Supply Chain Regulations

The U.S. Executive Order 14028, CISA's Known Exploited Vulnerabilities catalog, and the EU's Cyber Resilience Act all demand greater transparency into software supply chains. SBOMs are now expected artifacts for regulated organizations. Unsupported runtimes undermine compliance with these mandates.

Enforcement Trends

Auditors expect documentation of patching SLAs, SBOMs, and remediation processes. Organizations unable to produce evidence face failed audits, reputational damage, and regulatory penalties. In sectors such as healthcare and finance, noncompliance can result in blocked operations or fines in the millions.

THE ROLE OF AI IN JAVA SECURITY

AI as an Attack Tool

Threat actors use AI to scan dependency graphs, generate exploit proof-of-concepts, and automate reconnaissance. Nation-state actors have already deployed AI in campaigns targeting developer ecosystems.

AI in Defensive Security

Defensive tools leverage AI to accelerate SCA scans, identify vulnerable transitive dependencies, and prioritize remediation based on exploit likelihood. This improves visibility and response times.

Human-Led Governance

AI cannot decide when to migrate, when to backport, or how to apply compensating controls. Governance remains a human responsibility, ensuring compliance and strategic alignment while AI provides speed and scale.

EMBEDDING GOVERNANCE IN JAVA PROGRAMS

Governance must be a continuous process, not a one-time initiative.

- Align enterprise roadmaps with Oracle's two-year LTS cadence.
- Enforce SLAs that patch KEV-listed vulnerabilities within days.
- Automate SBOM generation at every build and release.
- Harden deployment strategies by favoring executable JARs over WARs.
- Track and remediate temporary runtime exceptions such as `--add-opens`.

Embedding governance in the Java program itself reduces exposure, strengthens audit resilience, and provides long-term operational stability.

CONCLUSION

Java's stability has been both its greatest strength and its most dangerous weakness. Organizations that fail to modernize in step with LTS cadences face regulatory penalties, technical debt, and catastrophic breaches.

The path forward requires a balance of proactive migration planning, comprehensive supply-chain governance, and commercial long-term support. HeroDevs believes enterprises should not be forced into insecure or rushed migrations. With proactive planning and Never-Ending Support, organizations can ensure their Java estates remain secure, compliant, and future-ready.

REFERENCES

1. Synopsys. Open Source Security and Risk Analysis (OSSRA) 2024.
2. Sonatype. State of the Software Supply Chain Report 2024.
3. Sonatype. Q2 2025 Malicious Package Activity Report.
4. ITPro. “North Korean hackers targeting developers via open source malware.” 2025.