

HUJAK #Java Community Keynote

After Three Decades of Java: Best Years Are Ahead

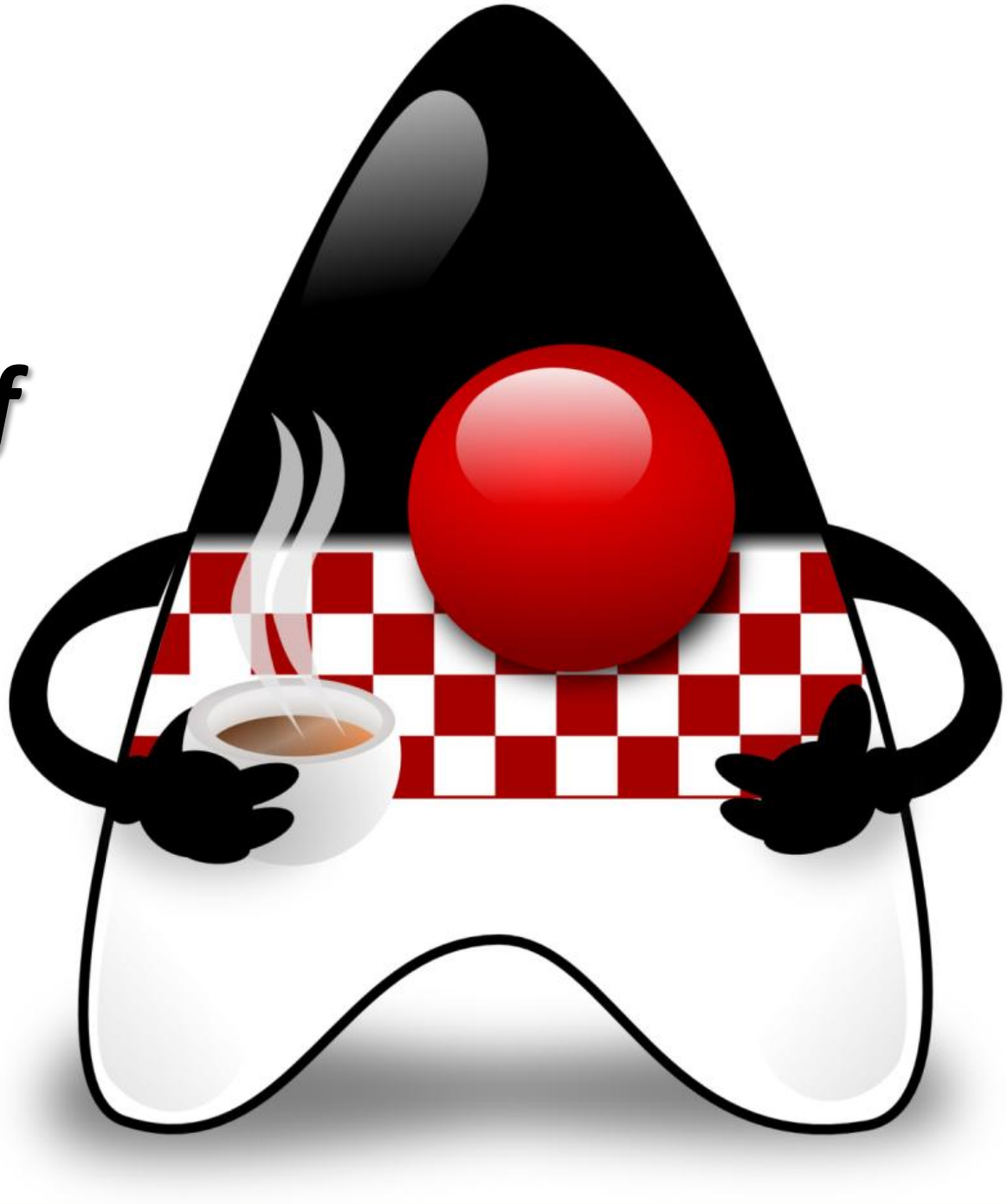
Branko Mihaljević

Stjepan Matijašević

Aleksander Radovan

Martin Žagar

HUJAK (Croatian JUG)





Java in Numbers (2026)

73B

active JVMs worldwide

51B

JVMs running in the cloud

#1

language for cloud-native development (VDC 2025)

10M+

Java developers

62%

of enterprises build AI functionality in Java (Azul 2026)

31

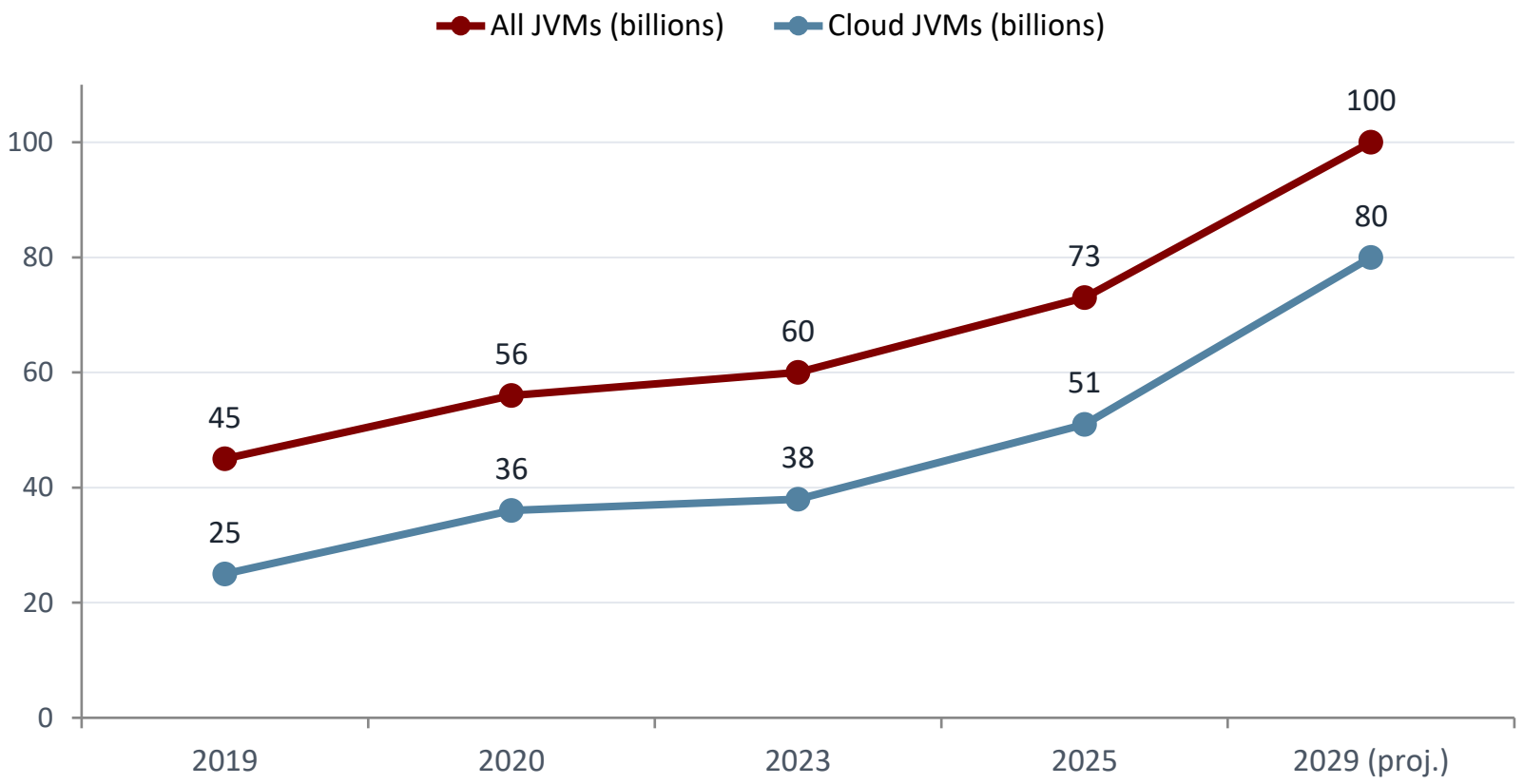
years since Java was publicly announced



Source: VDC Research 2025 for Oracle; Azul State of Java 2026; Oracle



Active JVMs



+62%

growth in active JVMs from 2019 to 2025

VDC projects 100B+ by 2029, over 80% in the cloud

Source: Oracle/VDC Research 2019, 2020, 2023 and 2025 estimates; 2029 is VDC's projection



The Next Milestone

100B+

deployed JVMs projected by 2029

Over 80% of them expected to run in the cloud (VDC Research, 2025)

#1 for **cloud-native, AI developer productivity, microservices,** and **big data**

56% of **enterprise developers use Java** (53% in 2023)

84% say Java helps build **cloud-native apps efficiently**

Source: VDC Research, Java in the Era of AI & Cloud-Native Innovation, 2025 (532 respondents)

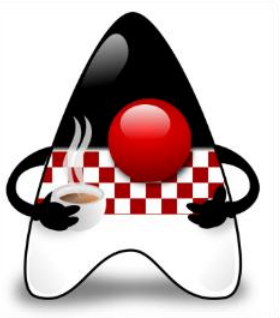


So, How Old is Java Actually?



- In 1991 – the **Stealth Project** began
- **Green Team** was formed
  after brainstorming in Aspen
- 1991/1992 – the Green team
  developed **Star 7** (*7)
  - Remote control device that could control other devices
  - MicroSPARC, 4MB RAM, LCD touchscreen, PCMCIA, IR, RF link …
- And they needed a **new programming language**





When was Java Born?



- In 1991 – the **Stealth Project** began
- In 1992 – **Oak** language and interpreter was created
- In fall 1994, the **first Java compiler** was written in Java
- **HotJava** graphical browser created
- Oak was officially **renamed to Java**
- In 1995, **Java** was **publicly announced** at **SunWorld '95** conference
- May 23rd, **1995** – **Java 1.0 alpha**
- January 23rd, **1996** – **Java 1.0**

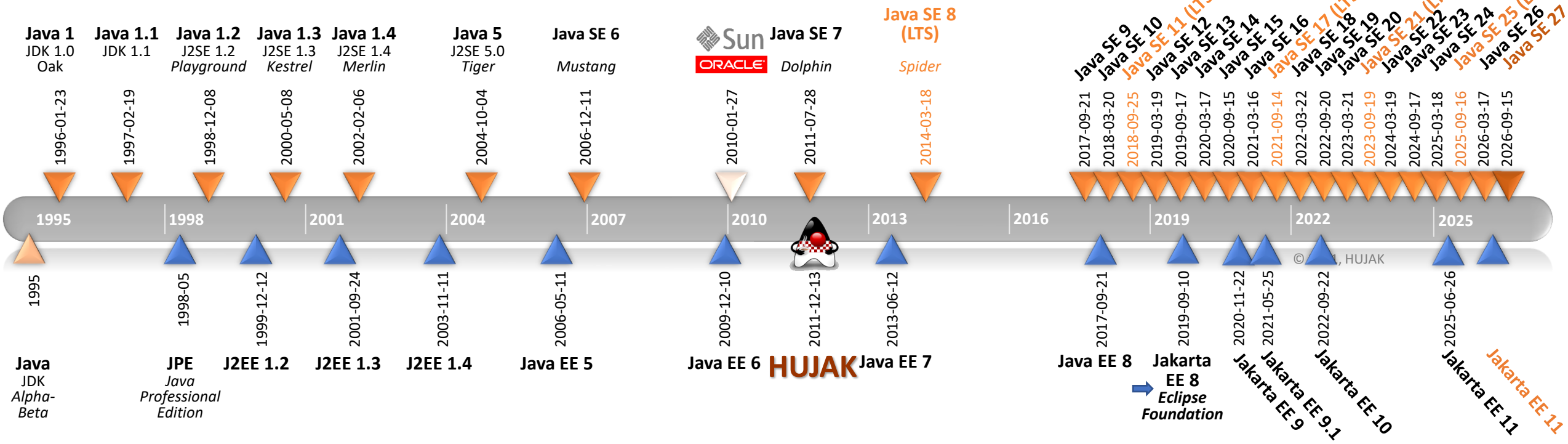


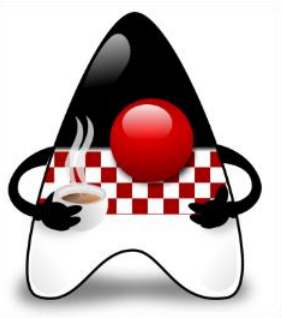




Java Timeline

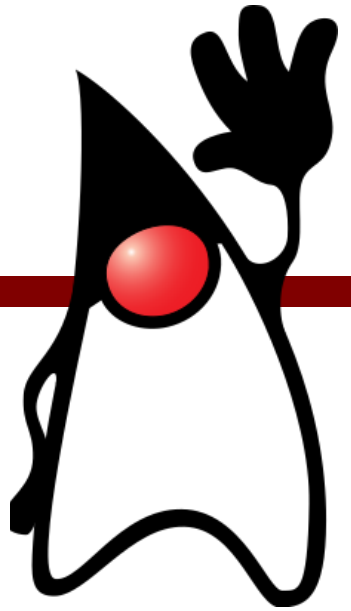
- **30+ years** of history… a legend!



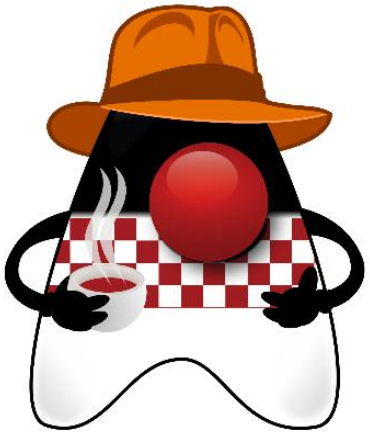


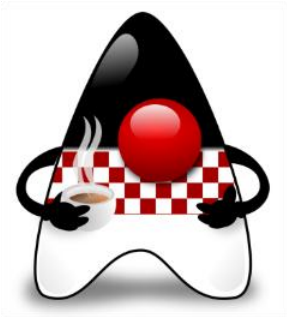
Who is (Cro)Duke?

- **Duke** – the famous **Java mascot**, on-screen animated agent on the first Java handheld device Star 7
- **CroDuke** – **Croatian version** of Duke and the **official logo of HUJAK**
- **CroDuke "Indy"** – **special version** of HUJAK's mascot designed for our other adventurous #Java conference in Zagreb, named **Javantura**









How did HUJAK Start?

- **HUJAK** was founded in late **2011**
- Firstly, we were interested in **meetups** (well, more like drinkups ☺)
- Initial goal: **World Domination (with Java)** ☺
- We had an **idea** – to **organize an international #Java conference in Croatia**
- We had a **plan**, but we needed some help to start
- And **15 years later** – this is our **15th JavaCro**!





A Kind of Sentimental Journey

- But we could not do it all alone, so we've got some **help from our friends** ☺

… the beginning of a beautiful friendship …
with HrOUG

- And so, the first **JavaCro conference** was born in **2012**!

- In the last **15 years** we (co)organized many **conferences** and **events**



HUJAK Organized 30+ Conferences (so far)

- 6 Java Days / Tracks @ HrOUG conferences (2011-2013, 2016-2019)
- 15 JavaCro conferences (2012-2026) & 3 WebCamp conferences (2012-2014)






- 7 Javantura conferences (2014-2020)








But the main question is:

Is Java Still Popular?



The One Java Ring to Rule Them All

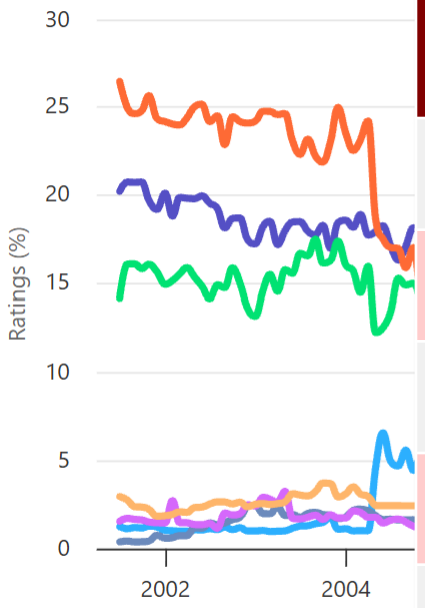
- Java Ring **runs a small JVM**
  - 134kB RAM, 32kB ROM
  - 1-wire interface
- Given to all attendees of 1998 **JavaOne** conference
- Actually, it was a packaged, rebranded **iButton** from Dallas Semiconductors, used for identification



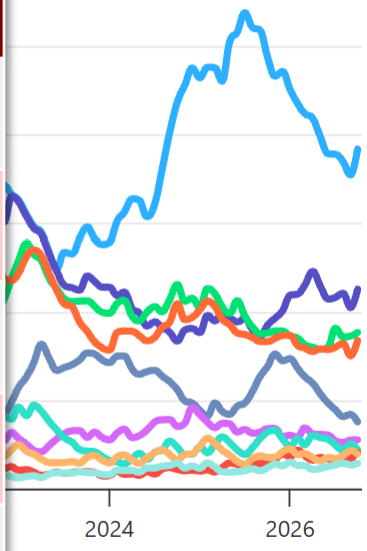


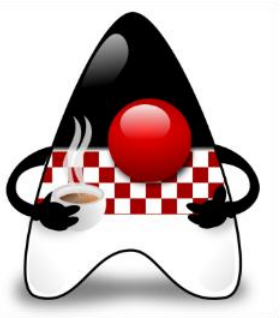
Is Java Still Popular?

- **TIOBE Index**, October 2026
  www.tiobe.com/tiobe-index/

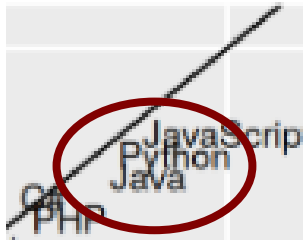


October 2026	October 2025	Programming Language	Ratings	Change
1	1	Python	19.23%	-5.23%
2	2	C	11.33%	+2.04%
3	3	C++	8.87%	+0.04%
4	**4**	**Java**	**8.41%**	**+0.06%**
5	5	C#	3.79%	-3.15%

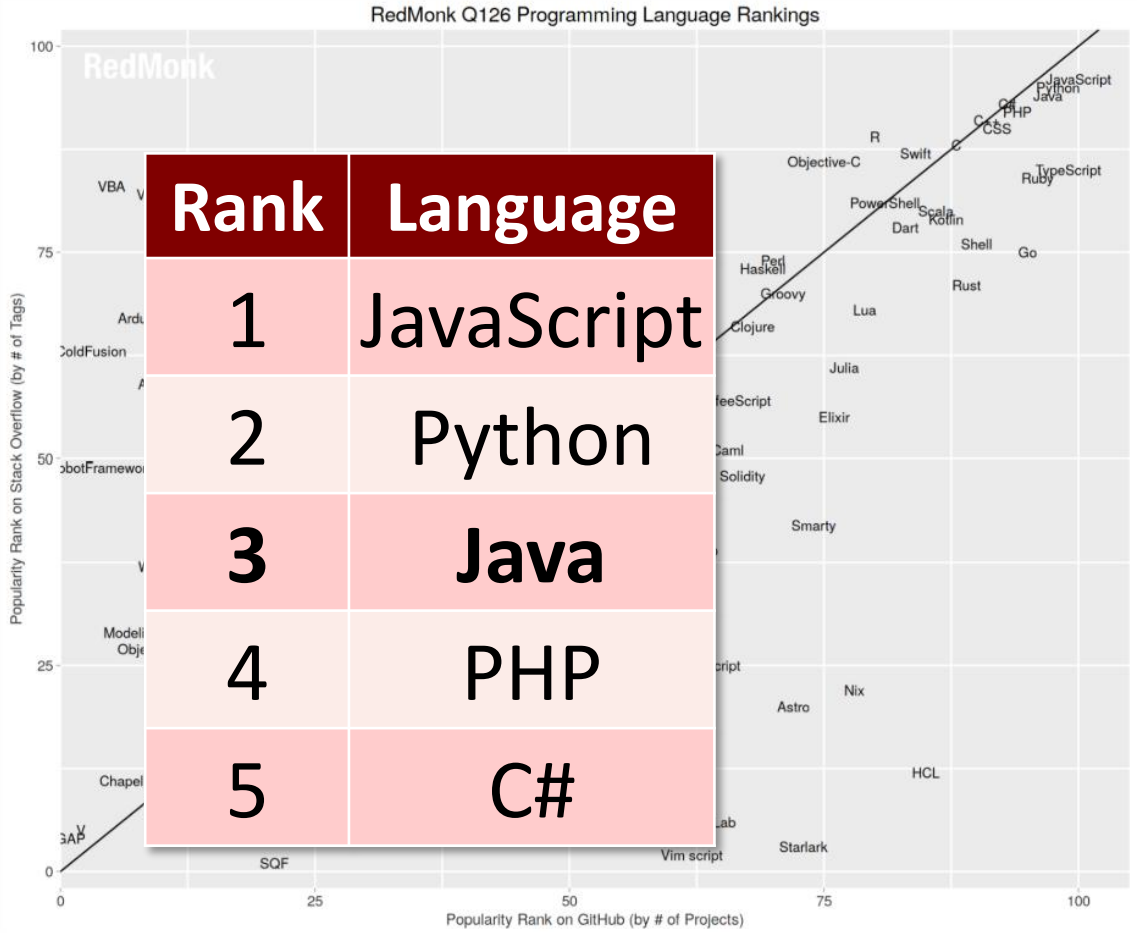




Is Java Still Popular?



- **RedMonk Programming Language Rankings**: Q1 2026
  - Language rankings based on **GitHub** and **Stack Overflow**
  - Combined to reflect both code (GitHub) and discussion (Stack Overflow) traction
  - However, *"AI has had a significant and accelerating impact on Stack Overflow, which comprises one half of the data used to both plot and rank languages."*



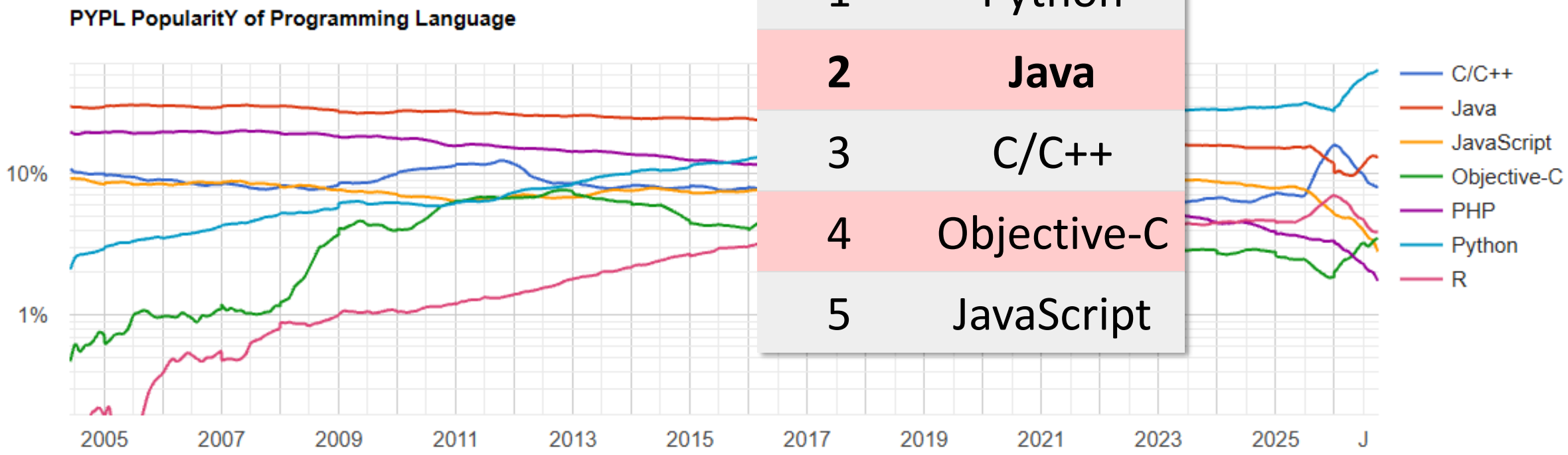
Rank	Language
1	JavaScript
2	Python
3	**Java**
4	PHP
5	C#



Is Java Still Popular?

- **PYPL Index**, October 2026
  pypl.github.io/PYPL.html

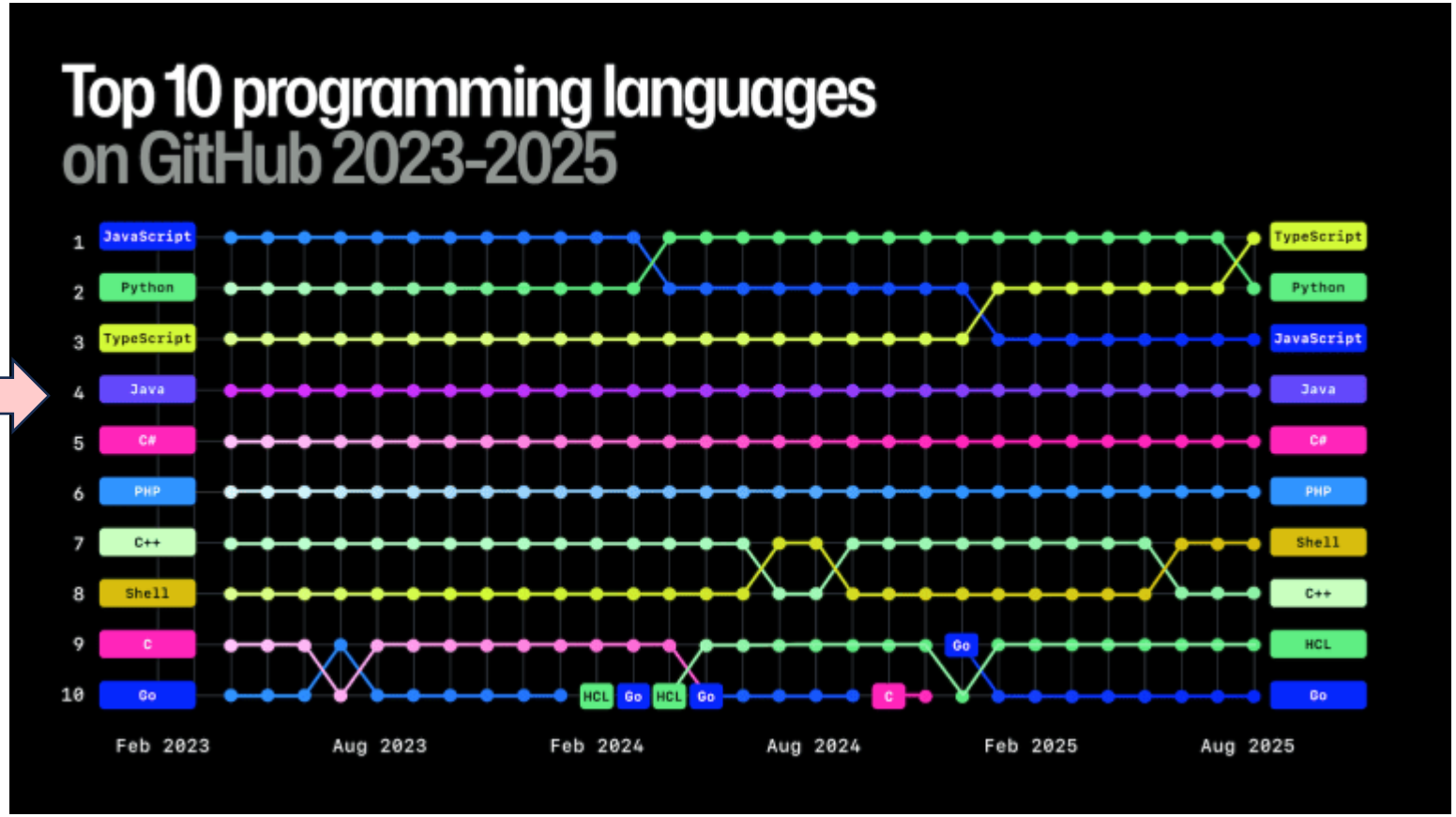
Rank	*Language*
1	Python
2	**Java**
3	C/C++
4	Objective-C
5	JavaScript





Is Java Still Popular?

- **GitHub Octoverse** 2025





Are Rankings Still Meaningful?

As AI writes more of the code,
using any one of today's popular languages may be enough,
and popularity rankings may become obsolete.

Paraphrase of IEEE Spectrum, Top Programming Languages

- *Example*: Half of RedMonk's ranking input comes from Stack Overflow, and StackOverflow questions have collapsed in 2026



Share of Mind vs Share of Production

Share of mind

- TIOBE #4
  - right behind C and C++
- PYPL #2, but …
- GitHub #4

Share of production: growing

- **73B active JVMs** (60B two years ago)
- **56% of enterprise developers** use Java (by VDC)
- **Java 21 is the most used version** (by JetBrains)
- **62% of enterprises build AI features in Java** (by Azul)



Is Java Still Moving Forward?



Java Moving Forward



- The Three Pillars

Stable evolution

- **Incremental** and **predictable**
- Careful **backward compatibility**
- **Tools, libraries, frameworks** keep up

Community acceptance

- **Open**, **transparent** development
- **OpenJDK, JCP** and **JUGs** contribute
- **Previews** collect real feedback

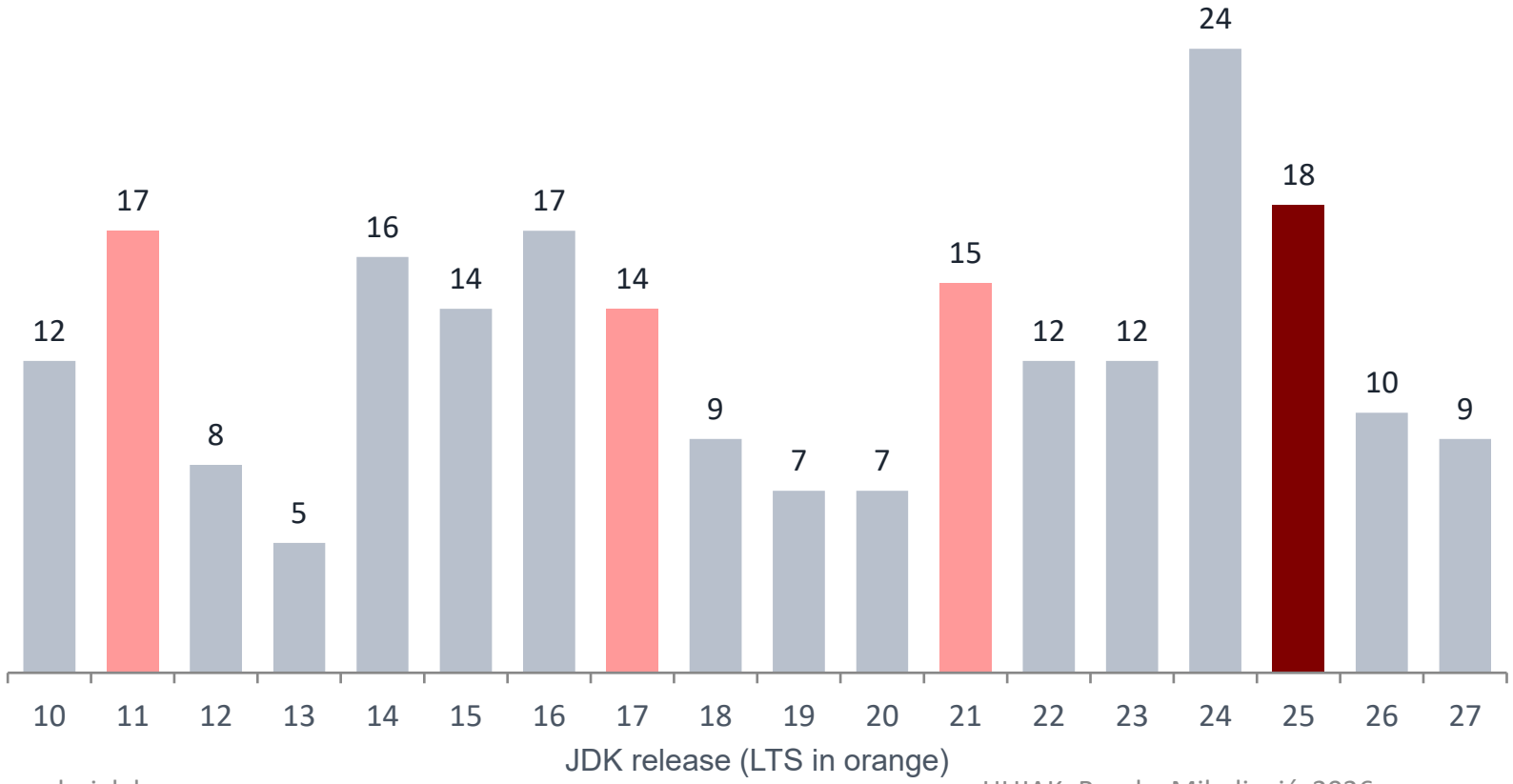
Continuous innovation

- **New features** every 6 months
- Long-running **projects** land in steps
- **Cautious**, **gradual** language **change**



Constant Evolution

- JEPs per release, JDK 10 to 27

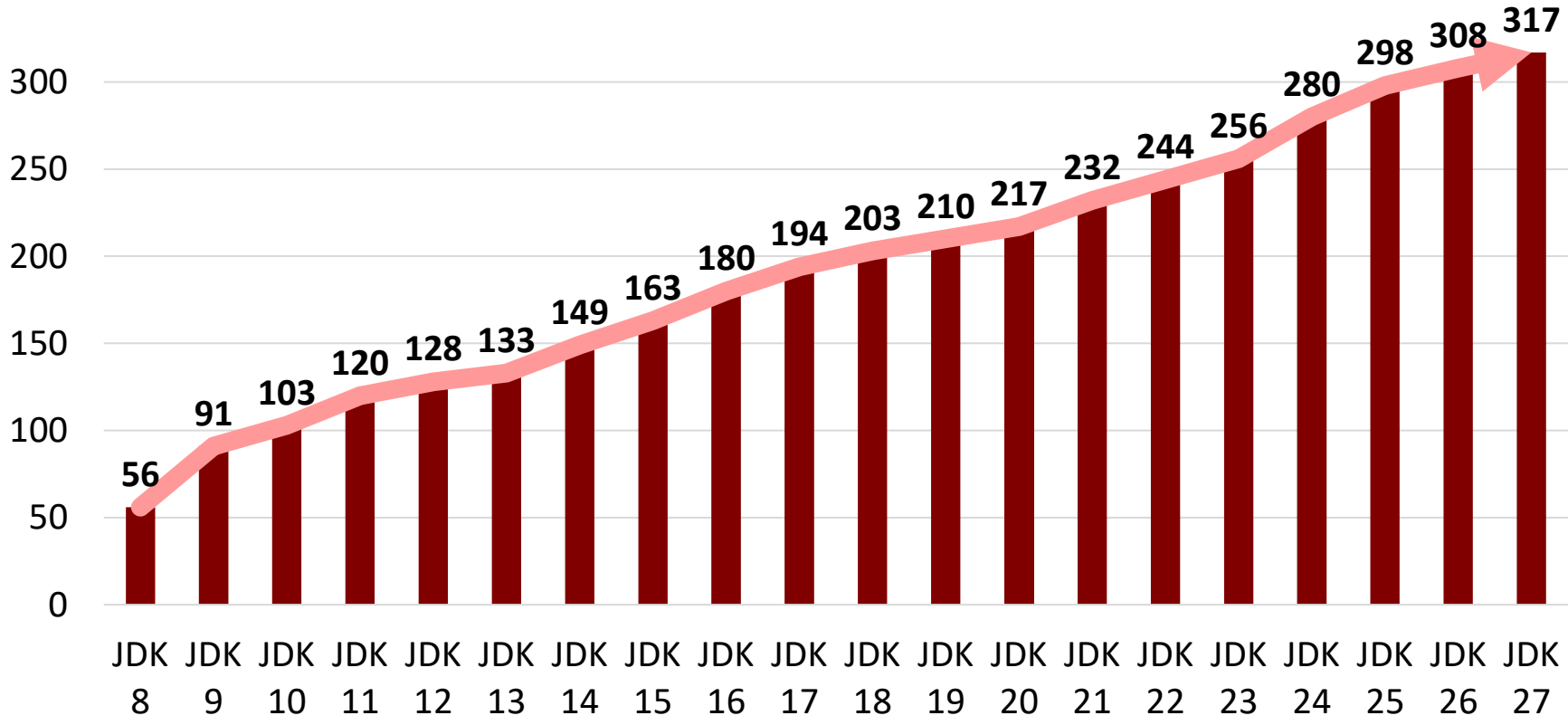


- JDK 24 shipped **24** JEPs
  - The **record** in the 6-month cadence
- JDK 25 LTS shipped **18**
- JDK 26 shipped **10**
- JDK 27 shipped **9**



Constant Evolution with JEPs

The number of JEPs in JDK 8 to JDK 27



- The number of JEPs is on the **constant rise**
- Continuous flow of **new features**
- What about Long Term Support (LTS)?

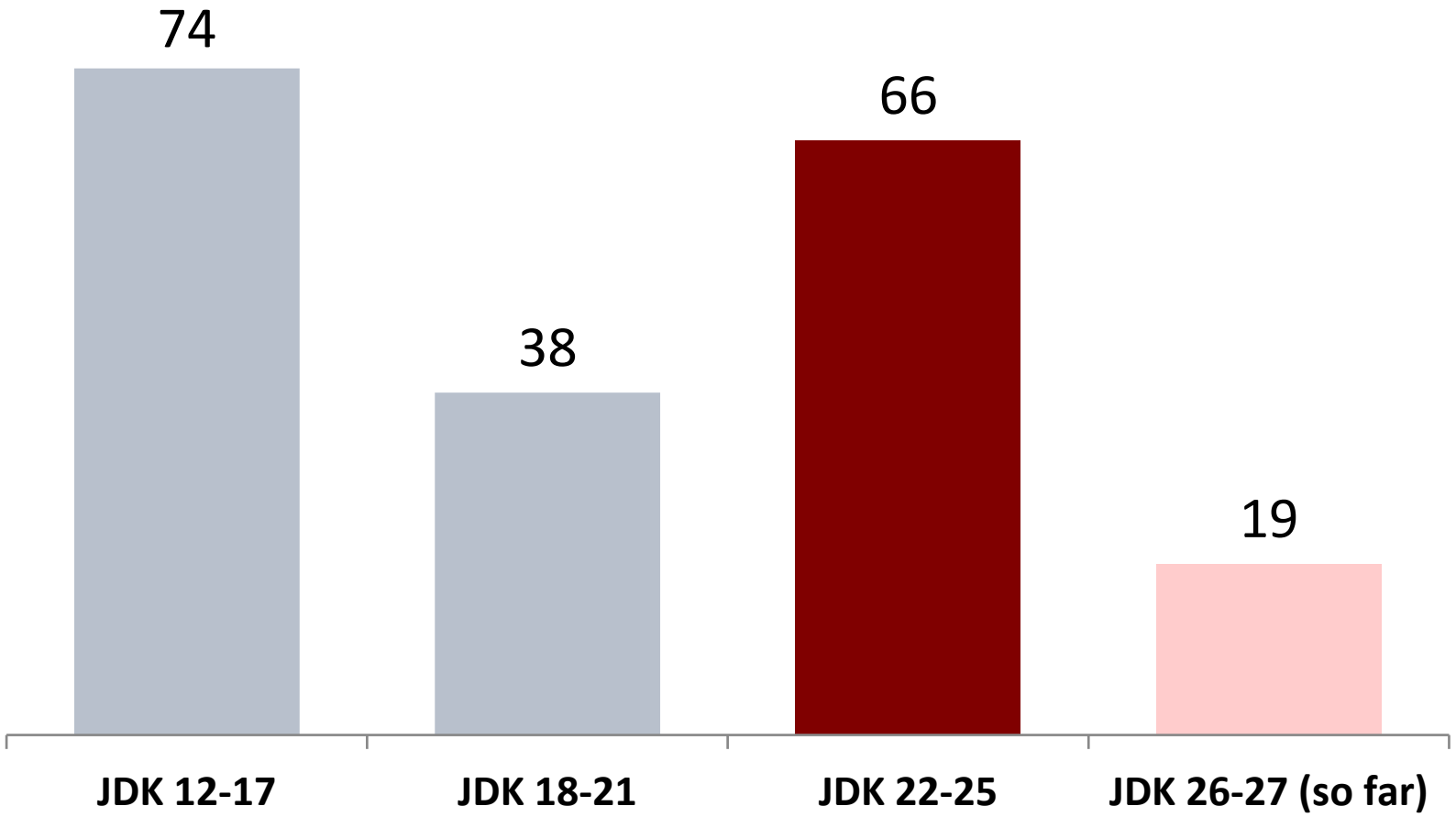


JEPs per LTS Cycle

66 JEPs

between JDK 21 and 25

- The second-richest LTS cycle
- JDK 26-29 is halfway, with **19** so far





The Releases

- The release train runs on time…

Sep 2025	Mar 2026	Sep 2026	Mar 2027	Sep 2027
JDK 25 LTS (18 JEPs)	JDK 26 (10 JEPs), launched at JavaOne	JDK 27 (9 JEPs), GA 15 September	JDK 28: value classes preview, JSON API incubator	JDK 29: next LTS (two-year cadence)



How New Features Arrive

Experimental

Test-bed for VM features.

Enable with -XX:+UnlockExperimentalVMOptions.

Incubator

New APIs and tools in jdk.incubator.* modules.

Likely to become permanent.

Preview

Fully specified and implemented, not yet permanent.

Enable with --enable-preview.

Final

Permanent part of Java SE.

Backward compatibility applies from here on.



Available OpenJDKs?

- Oracle OpenJDK
- Eclipse Temurin (AdoptOpenJDK/Adoptium)
- Amazon Corretto
- Azul Zulu
- Red Hat OpenJDK
- IBM Semeru (OpenJ9)
- Microsoft OpenJDK
- Bellsoft Liberica JDK
- SAP SapMachine
- Linux distribution's OpenJDK
- Alibaba Dragonwell
- Tencent Kona JDK
- Huawei BiSheng JDK

JDKs in other VMs and packages:

- Oracle GraalVM (CE / EE)
- Azul Platform Prime (PrimeVM)
- BellSoft Liberica NIK (Native Image Kit)
- Red Hat Mandrel
- Liberica NIK
- Amazon Corretto with SnapStart
- Microsoft OpenJDK (Azure-Optimized)
- Gluon Substrate
- Tencent Kona JDK (Enhanced Edition)

- and, of course, the Oracle JDK



Is Java "Free"? (in Oracle JDK)



Version	Oracle JDK	Licence
Any OpenJDK build	**Any version, any vendor**	**GPLv2 + Classpath Exception, always free**
Java 27	All releases	**No-Fee Terms and Conditions (NFTC)**
Java 25 LTS	Releases through September 2028	**NFTC, free for all users**
Java 21 LTS	Free NFTC updates ended in 2026	**OTN: paid subscription in production**
Java 17 LTS	17.0.13 and later	**OTN: paid subscription in production**
Java 11 & Java 8	All current updates	**OTN: paid subscription in production**

- September 2026 is **end of free Oracle JDK 21 updates** under NFTC, so later Oracle JDK 21 updates are under OTN: paid Java SE Universal Subscription in production
- Options: **move** to **Oracle JDK 25** (free until Sep 2028) or **any free OpenJDK 21-25** build (Temurin, Corretto, Zulu...) or **pay** for Oracle support



The Latest JDKs and JEPs

JDK 25 LTS, JDK 26, JDK 27, and a first look at JDK 28



JDK 25 LTS (18 JEPs)

- Released 16 September 2025 · openjdk.org/projects/jdk/25

470	PEM Encodings of Cryptographic Objects	Preview	511	Module Import Declarations	Final
502	Stable Values	Preview	512	Compact Source Files & Instance Main	Final
503	Remove the 32-bit x86 Port	Removal	513	Flexible Constructor Bodies	Final
505	Structured Concurrency (5th)	Preview	514	AOT Command-Line Ergonomics	Final
506	Scoped Values	Final	515	AOT Method Profiling	Final
507	Primitive Types in Patterns (3rd)	Preview	518	JFR Cooperative Sampling	Final
508	Vector API (10th)	Incubator	519	Compact Object Headers	Final
509	JFR CPU-Time Profiling	Experimental	520	JFR Method Timing & Tracing	Final
510	Key Derivation Function API	Final	521	Generational Shenandoah	Final



JDK 26 (10 JEPs)

- Released 17 March 2026 (launched on stage at JavaOne 2026)

500	Prepare to Make Final Mean Final	Final	524	PEM Encodings of Cryptographic Objects (2nd)	Preview
504	Remove the Applet API	Removal	525	Structured Concurrency (6th)	Preview
516	AOT Object Caching with Any GC	Final	526	Lazy Constants (2nd)	Preview
517	HTTP/3 for the HTTP Client API	Final	529	Vector API (11th)	Incubator
522	G1 GC: Improve Throughput by Reducing Synchronization	Final	530	Primitive Types in Patterns (4th)	Preview



JDK 27 (9 JEPs)

- Released 15 September 2026 · post-quantum security and better defaults

523	Make G1 the Default GC in All Environments	Final
527	Post-Quantum Hybrid Key Exchange for TLS 1.3	Final
534	Compact Object Headers by Default	Final
536	JFR In-Process Data Redaction	Final
531	Lazy Constants (3rd)	Preview
532	Primitive Types in Patterns (5th)	Preview
533	Structured Concurrency (7th)	Preview
538	PEM Encodings (3rd)	Preview
537	Vector API (12th)	Incubator



JDK 28 (Targeted So Far)

- GA planned March 2027 · 6 JEPs targeted, 3 proposed to target

401	Value Objects (Preview) – Project Valhalla	Preview
539	Strict Field Initialization in the JVM (Preview)	Preview
543	Structured Concurrency – final, after 7 previews	Proposed
542	PEM Encodings of Cryptographic Objects	Final
535	Shenandoah GC: Generational Mode by Default	Final
540	Simple JSON API (Incubator)	Incubator
545	Faster Startup and Warmup with ZGC	Proposed
546	Adaptive Heap Sizing for ZGC	Proposed
541	Deprecate the macOS/x64 Port for Removal	Removal



Four Themes Across JDK 25, 26 and 27

Concurrency

- Virtual Threads, Scoped Values (final)
- Structured Concurrency (7th preview)

Performance

- Compact Object Headers by default, G1 everywhere
- AOT cache, Lazy Constants, Vector API

Security and integrity

- HTTP/3, post-quantum TLS, PEM, KDF
- Final means final, no Security Manager

Simplicity

- Compact source files, module imports
- Flexible constructors, primitive patterns



Virtual Threads (JDK 21) Are Here to Stay

- JEP 444: **lightweight JVM threads**, final since JDK 21
- Thread-per-request style scales with near-optimal hardware use
- Existing java.lang.Thread code adopts with minimal change
- VTs from JDK 24 (JEP 491) are now **synchronized, no longer pins the carrier thread**
  - The biggest production blocker is gone
- Debugging and profiling with the usual JDK tools and JFR

1M+

virtual threads in one JVM are routine; a platform thread costs about 1 MB of stack



Scoped Values (JDK 25)

- JEP 506: **immutable, thread-safe alternative to ThreadLocal**
- Pass context down a call chain without method parameters
- Bound for a well-defined scope, then gone: no leaks
- Cheap to inherit by many virtual threads
- Designed to work with **structured concurrency**
- Five rounds of incubation and preview, JDK 20 to 24



***ThreadLocal*, done right**



Structured Concurrency (Preview)

- **Treat related tasks in different threads as one unit of work**
- Errors and cancellation propagate automatically
- Clear entry and exit points; thread dumps show the hierarchy
- New API shape since JDK 25: StructuredTaskScope.open() and Joiners
- Still preview in JDK 27, the last big piece of Project Loom
- Production-ready? Final proposed for JDK 28 (JEP 543), **review ends today**

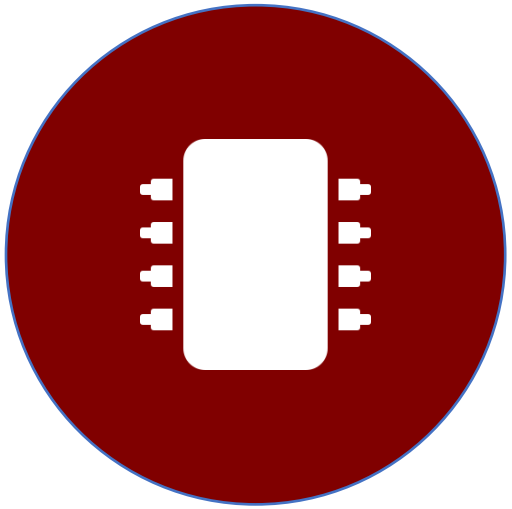
7th

preview round in JDK 27
(incubated in JDK 19-20,
preview since JDK 21)



Foreign Function & Memory API (JDK 22)

- JEP 454: **call native libraries from pure Java**, no JNI glue code
- Safe access to off-heap memory with Arenas and MemorySegments
- jextract generates Java bindings from C headers
- JNI is now restricted with warnings (JEP 472, JDK 24)
- The foundation for AI inference libraries, GPU and SIMD work



Java meets native, safely



Vector API (Incubator, JDK 27)

- **Express SIMD computations** that compile to optimal CPU vector instructions
- x64 (AVX) and AArch64 (NEON, SVE) supported
- Key for **AI inference, ML kernels, image processing, search**
- Waiting for Valhalla: once value classes ship it will move to a java.* package
- Usable today with --add-modules jdk.incubator.vector

12th

Incubator
Since JDK 16 (2021)
Final after Valhalla



Lazy Constants (Preview, JDK 27)

- Formerly **Stable Values** (JEP 502, JDK 25), renamed in JEP 526
- A **value set at most once**, then treated by the JIT like a **final** field
- Initialise expensive state on first use, not at startup
- Thread-safe without double-checked locking
- Lazy lists and maps of constants, too

3rd

preview in JDK 27,
4th preview is drafted
for JDK 28



Garbage Collectors Keep Getting Better

JDK 21	JDK 23-24	JDK 25	JDK 26	JDK 27	JDK 28
Generational ZGC	ZGC generational by default; non-generational removed	Generational Shenandoah (product)	G1 throughput: less sync with app threads (JEP 522)	G1 default in all environments (JEP 523)	Shenandoah generational by default (JEP 535)



Compact Object Headers

- **Object headers shrink** from 96-128 bits to 64 bits
- Product in JDK 25 (JEP 519), on by default in JDK 27 (JEP 534)

Free Memory, No Code Changes

-22%

heap used in SPECjbb2015

-8%

CPU time in SPECjbb2015

-15%

garbage collections (G1 and Parallel)

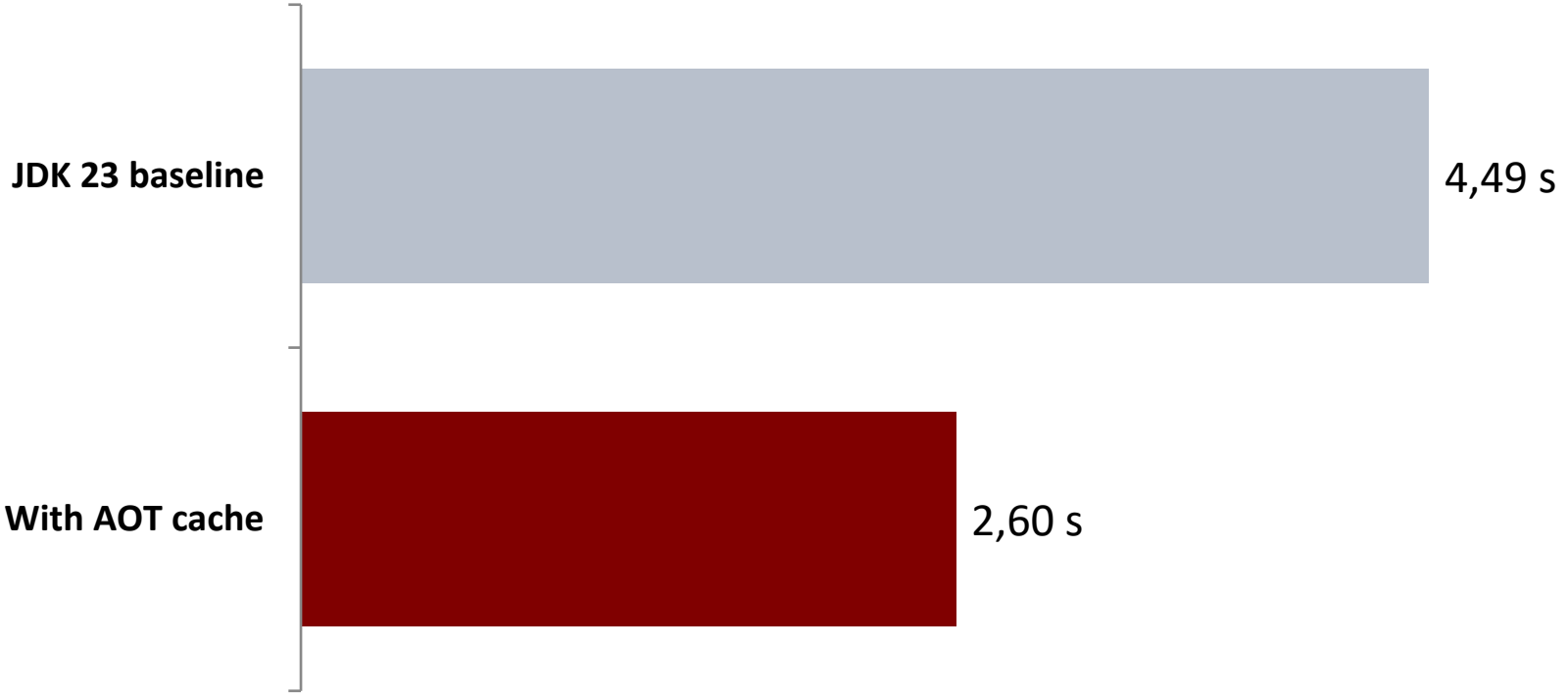
+10%

faster: highly parallel JSON parser benchmark

Source: JEP 519: Compact Object Headers



Project Leyden Makes Startup Faster



Source: JEP 483: Ahead-of-Time Class Loading & Linking

-42%

startup time for Spring PetClinic 3.2 with the AOT cache (21,000 classes loaded at startup)

JDK 25 and 26 add method profiles and cached objects



JDK Flight Recorder Keeps Growing

CPU-time profiling (JDK 25)

Accurate CPU-time samples on Linux (JEP 509, experimental)

Cooperative sampling (JDK 25)

Stack walks at safepoints, more stable sampling (JEP 518)

Method timing and tracing (JDK 25)

Time and trace chosen methods via bytecode instrumentation (JEP 520)

In-process data redaction (JDK 27)

Remove sensitive values before recordings leave the process (JEP 536)



HTTP/3 for the HTTP Client (JDK 26)

- JEP 517: **QUIC-based HTTP/3** in java.net.http
- Faster connection setup, no head-of-line blocking
- Minimal code change for existing clients
- HTTP/2 remains the default

Opt in with one line

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

var request = HttpRequest.newBuilder(
        URI.create("https://openjdk.org/"))
        .GET()
        .build();

var response = client.send(request,
        HttpResponse.BodyHandlers.ofString());
```



Java Goes Post-Quantum

- Harvest now, decrypt later: attackers record encrypted traffic today and decrypt it once quantum computers arrive

JDK 21	JDK 24	JDK 25	JDK 26	JDK 27
Key Encapsulation Mechanism API (JEP 452)	ML-KEM and ML-DSA (JEP 496, JEP 497)	Key Derivation Function API final (JEP 510)	PQ-ready JAR signing, HPKE	Hybrid PQ key exchange for TLS 1.3 (JEP 527)



PEM Encodings of Cryptographic Objects

- Read and write keys, certificates and CRLs in **PEM** text format
- No more Base64 juggling or Bouncy Castle just for PEM
- Supports PKCS#8, X.509 and encrypted private keys
- Preview in JDK 25-27; final targeted for JDK 28 (JEP 542)

3rd

preview in JDK 27, final expected in JDK 28



Integrity by Default, Step by Step

- Step by step, Java closes the back doors: code can trust what it declares, and every exception needs an explicit opt-in.

JDK 9	JDK 16-17	JDK 21	JDK 24	JDK 26
Module system	JDK internals strongly encapsulated	Warn on dynamically loaded agents	Security Manager disabled; JNI and Unsafe warnings	Prepare to make final mean final (JEP 500)



Final Means Final (JEP 500, JDK 26)

- Mutating final fields via reflection triggers a warning in JDK 26
- A future release will deny it by default
- The JIT can trust final, so code gets faster
- Libraries (DI, mocking, serialisation) must adapt

Deep reflection on final fields now warns

```
class Config {
    private final int port;
    Config(int port) { this.port = port; }
}

Field f = Config.class.getDeclaredField("port");
f.setAccessible(true);
f.set(config, 9090);  // JDK 26: WARNING
                      // later: exception

// explicit opt-in while libraries migrate:
// --enable-final-field-mutation=ALL-UNNAMED
```



Goodbye, Old Friends

Applet API

Removed in JDK 26 (JEP 504), deprecated since JDK 9

Security Manager

Permanently disabled in JDK 24 (JEP 486)

32-bit x86 port

Removed in JDK 25 (JEP 503)

macOS/x64 port

Deprecation targeted for JDK 28 (JEP 541)



Hello, World: 1996 vs 2026

JDK 1.0 (1996)

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

JDK 25 (JEP 512, final)

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



Module Import Declarations (JDK 25)

- JEP 511: **import every package a module exports**
- Your code does not need to be a module
- Compact source files import java.base implicitly
- Fewer import lines in scripts and teaching

One import for a whole module

```
import module java.base;  // 54 packages

void main() {
    var names = List.of("Ana", "Ivan", "Marko");
    Map<String, Integer> lengths = names.stream()
        .collect(Collectors.toMap(n -> n,
                                  String::length));
    IO.println(lengths);
}
```



Flexible Constructor Bodies (JDK 25)

- JEP 513: statements before super(...) or this(...)
- Validate and compute arguments first
- Cannot touch the object being built
- Fewer static helper methods

Validate before calling super(...)

```
public class PositiveBigInteger extends BigInteger {

    public PositiveBigInteger(long value) {
        if (value <= 0)
            throw new IllegalArgumentException(
                "non-positive");
        super(Long.toString(value));
    }
}
```



Primitive Types in Patterns (Preview)

- **switch on any primitive type**
- instanceof checks safe conversions
- Record patterns extract primitives without casts
- Preview in JDK 23-27

Patterns now work for int, long, double...

```
String describe(int status) {
    return switch (status) {
        case 0       -> "okay";
        case 1       -> "warning";
        case int i   -> "unknown status: " + i;
    };
}

if (value instanceof byte b) {  // if it fits
    buffer.put(b);
}
```



The Point Class Across Three Decades

1996: a class

```
class Point {
  final int x, y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  int getX() {return x;}
  int getY() {return y;}
  // equals, hashCode,
  // toString ...
}
```

2021: a record (JDK 16)

```
record Point(
    int x, int y) {}
```

2023: a pattern (JDK 21)

```
if (o instanceof
      Point(int x, int y)) {
  IO.println(x + y);
}
```

2027: a value (JDK 28)

```
value record Point(
    int x, int y) {}

// no identity, flat
// in memory, == by state
```



The Long-Running Projects: Where Are They Now?

Amber, Valhalla, Loom, Panama, Leyden, Babylon and Lilliput



Project Status Board, October 2026

Amber
In progress

Records, patterns, compact sources done; primitive patterns in preview

Valhalla
New

Value classes (JEP 401) integrated for JDK 28 preview, after 12 years

Loom
Delivered

Virtual threads and scoped values final; structured concurrency in 7th preview

Panama
Delivered

FFM final since JDK 22; Vector API waits for Valhalla

Leyden
In progress

AOT cache: classes (24), profiles (25), objects (26); AOT code next

Babylon
Research

Code reflection and HAT: Java on GPUs via CUDA and OpenCL

Lilliput
Delivered

Compact object headers: product in 25, default in 27

Galahad
Research

Bringing Graal JIT ideas into OpenJDK



Project Valhalla

- JEP 401, **Value Classes and Objects**: preview in JDK 28 (March 2027)
- Already in JDK 28 early-access builds
- Objects without identity: == compares state, not references
- The JVM may flatten them in memory and in arrays
- About 30 JDK classes become value classes: **Integer, Long, Double, Optional...**
- Next: null-restricted types and specialised generics

12

years from Project Valhalla's start (2014) to a preview JEP



Value Classes in Practice (JDK 28 Preview)

- **value modifier** on classes and records
- Fields are implicitly final
- No synchronisation on value objects
- Integer and friends become value classes

Codes like a class, works like an int

```
value record Point(int x, int y) {}

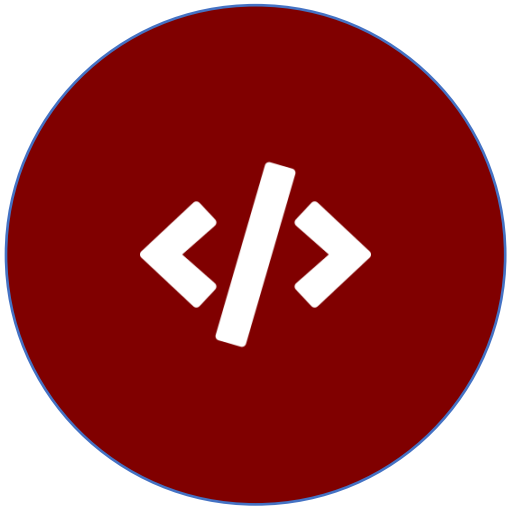
value class Money {
    private final long cents;
    private final String currency;
    Money(long cents, String currency) {
        this.cents = cents;
        this.currency = currency;
    }
}

var p = new Point(1, 2);
var q = new Point(1, 2);
IO.println(p == q);   // true: same state
```



Project Amber

- Delivered: **var, switch expressions, text blocks, records, sealed types, patterns**
- Delivered in JDK 25: **compact source files, module imports, flexible constructors**
- In preview: **primitive types in patterns**
- Next: **constant patterns and pattern assignment** (deconstruct without if)
- Under discussion: withers (derived record creation, JEP 468)
- Withdrawn on purpose: String Templates (2024), to be redesigned



Right-sizing language ceremony



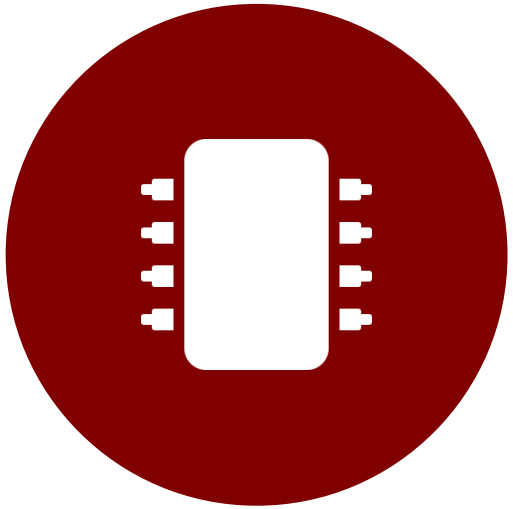
Project Loom: From Fibers to Production

2017	JDK 19	JDK 21	JDK 24	JDK 25	JDK 27
Project Loom starts (fibers, continuations)	Virtual threads preview	Virtual threads final	No more pinning on synchronized	Scoped values final	Structured concurrency, 7th preview



Project Babylon: Java on the GPU

- Code reflection: Java code as a model frameworks can analyse and transform
- **HAT (Heterogeneous Accelerator Toolkit)**: Java kernels to CUDA and OpenCL
- Same idea can target SQL, ONNX models, other foreign runtimes
- Showcased at JavaOne 2026 and Devoxx Belgium 2025 (GPU-ready AI models in pure Java)
- Experimental, no product timeline yet



Pure Java,
GPU speed



Coming: a Simple JSON API in the JDK

- JEP 540: **Simple JSON API**, incubator targeted to JDK 28
- Parse and generate JSON without a third-party library
- Aimed at simple tasks, scripts and teaching, not at replacing Jackson
- Fits the on-ramp story with compact source files

JDK 28

first incubation, planned for March 2027

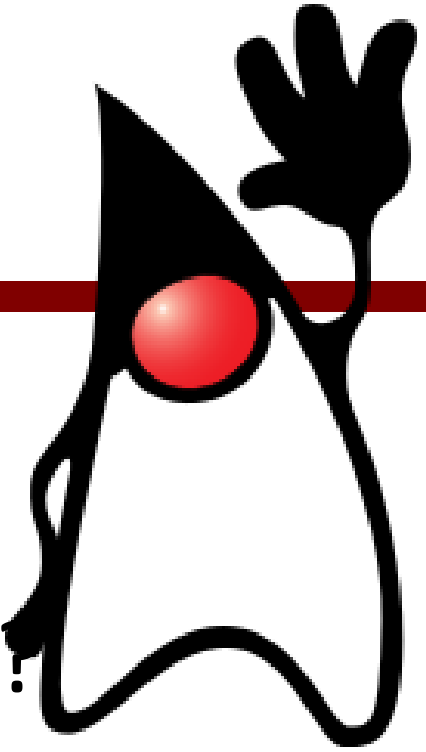


Current State of Java?

What surveys of tens of thousands of developers say



Some Questions for All of Us



- Still using **Java 8** (2014)?
- Maybe **Java 11 LTS** (2018)?
- Upgraded to the "**older**" **Java 17 LTS** (2021)?
- On "**newer**" **Java 21 LTS** (2023)?
- Already on the "**new**" **Java 25 LTS** (2025)?
- Trying **Java 26/27** (2026)?

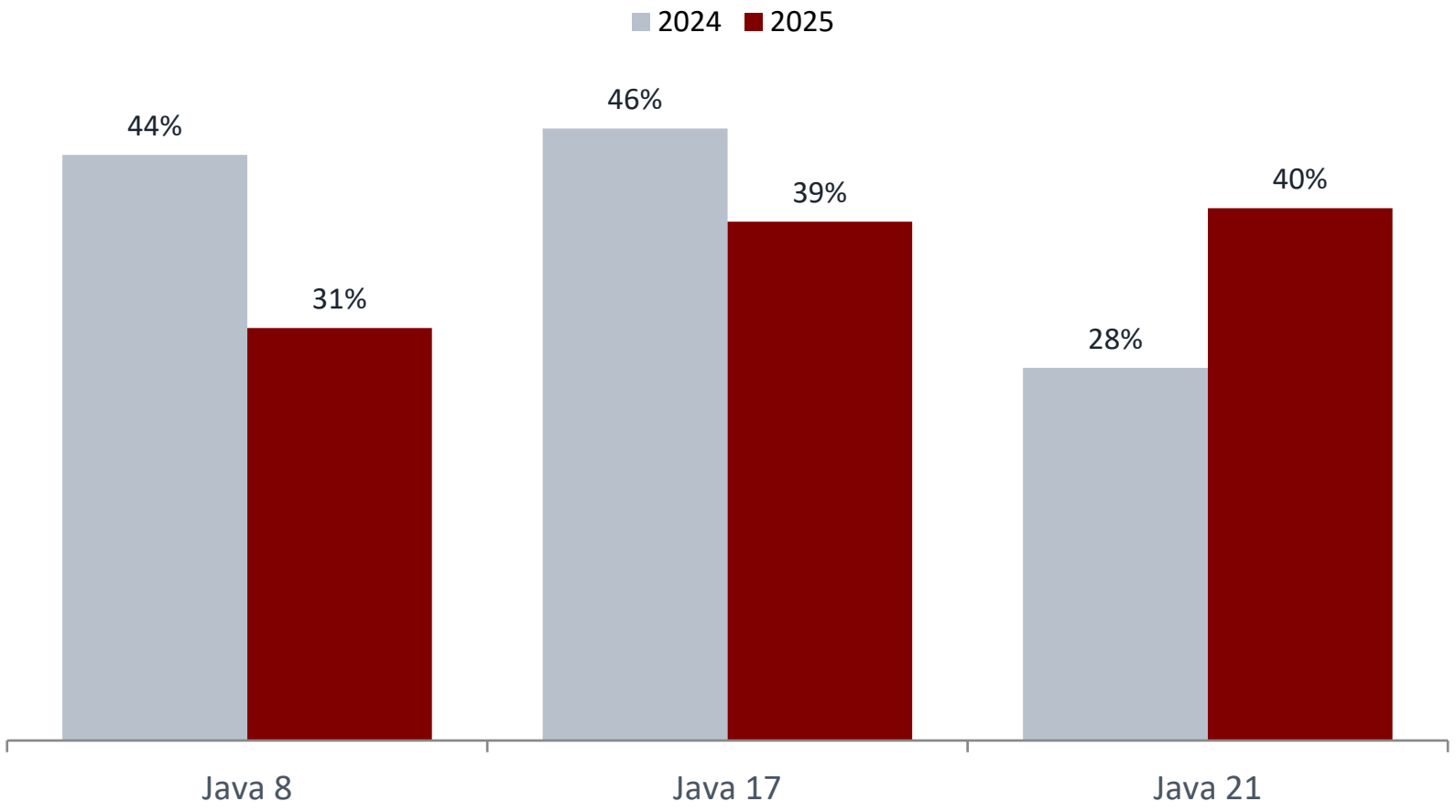


The Surveys

Survey	Published	Respondents	Focus
Azul State of Java 2026	Feb 2026	2,039	Enterprise Java, AI, cloud, Oracle
JetBrains State of Java 2025	2025	5,000+	Java developers
JetBrains Developer Ecosystem 2025	Oct 2025	24,534	All developers, AI
Stack Overflow Developer Survey 2025	Jul 2025	49,009	All developers, AI
Jakarta EE Developer Survey 2025	Sep 2025	1,700+	Enterprise Java
JRebel Java Productivity Report 2025	Mar 2025	731	Java teams, tools
VDC Research for Oracle	2025	532	Enterprise, cloud, AI



Java 21 Is Now the Most Used Version



Source: JetBrains, State of Java 2025 (5,000+ developers)

40%

of Java developers use Java 21 (28% a year earlier)

Java 8 fell from 44% to 31%

14% already used Java 24



Same Trend in the Enterprise

43%

of Jakarta EE developers on Java 21 (30% in 2024)

Java 11 still at 37%. Java 8 and 17 declining.
18% already on Jakarta EE 11.

- 1,700+ respondents, March-June 2025
- Jakarta EE (58%) overtook Spring (56%) among respondents
- Cloud: 22% lift-and-shift, 14% already fully cloud

Source: Eclipse Foundation, Jakarta EE Developer Survey 2025 (via ADTmag)



Tools of the Trade

72%

use IntelliJ IDEA

67%

build with Maven

65%

use the Spring framework

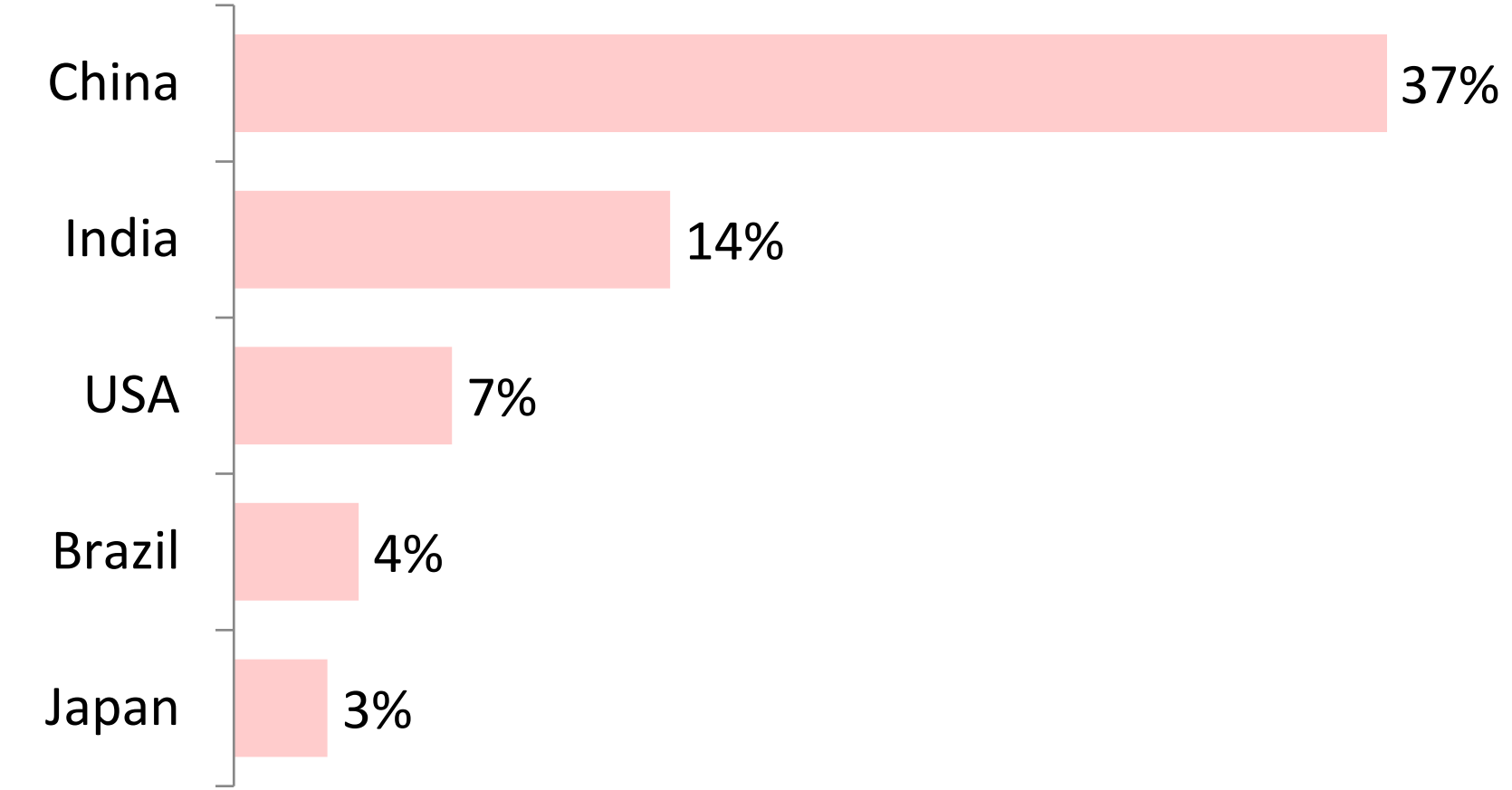
78%

work on back-end development

Source: JetBrains, State of Java 2025



Where Java Developers Come From



Source: JetBrains, State of Java 2025

37%

of JetBrains' Java respondents are in China.

Survey samples shape the story: always ask who answered.

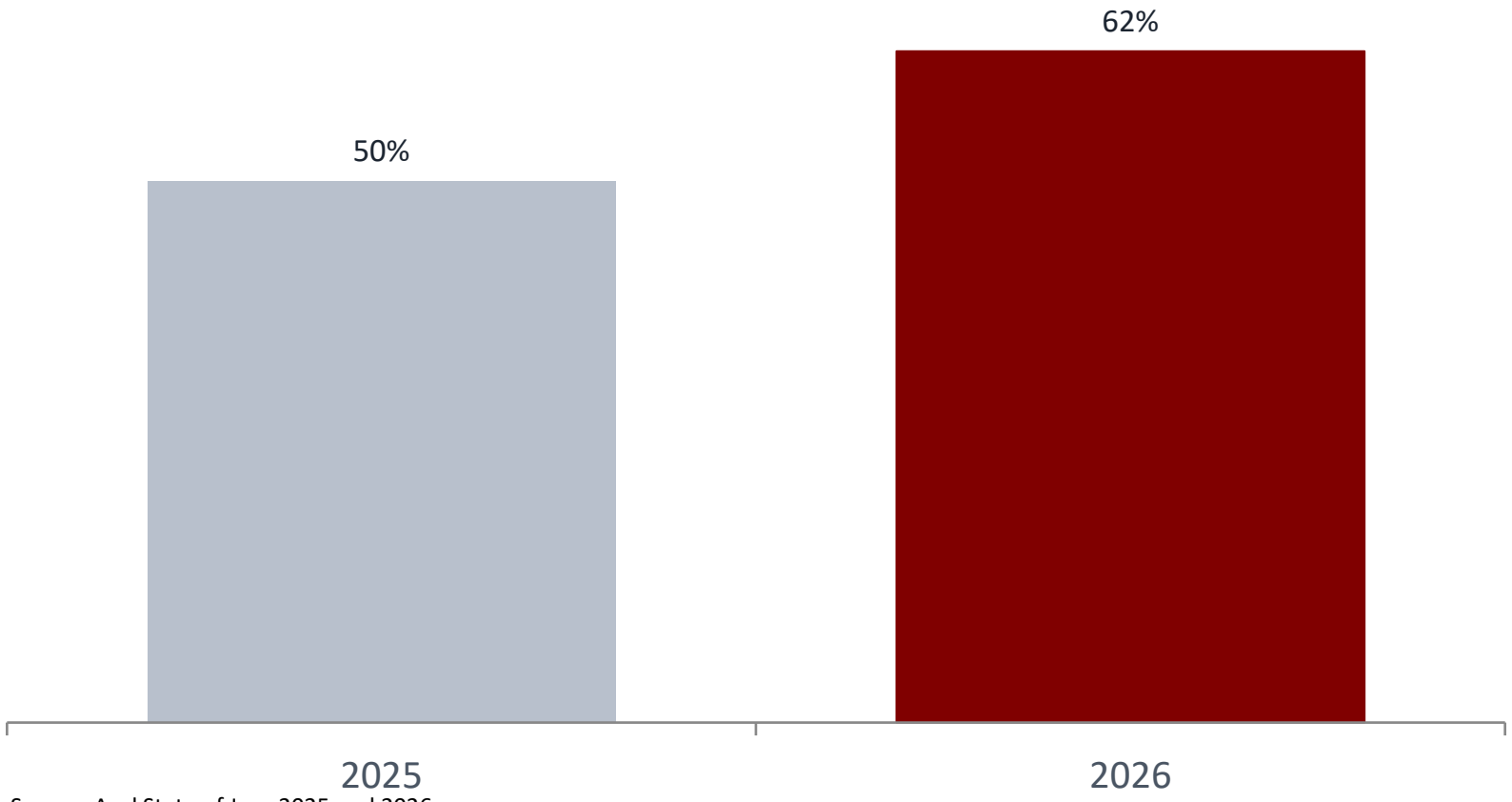


Java and AI:
Where Enterprise AI Will Run

From hype to production plumbing



Java Is Becoming an AI Language



Source: Azul State of Java 2025 and 2026

31%

say more than half of their Java applications already contain AI functionality



AI in Every Developer's Day

85%

of developers regularly use AI for coding

62%

rely on at least one AI assistant, agent or AI editor

20%

save 8 or more hours a week with AI

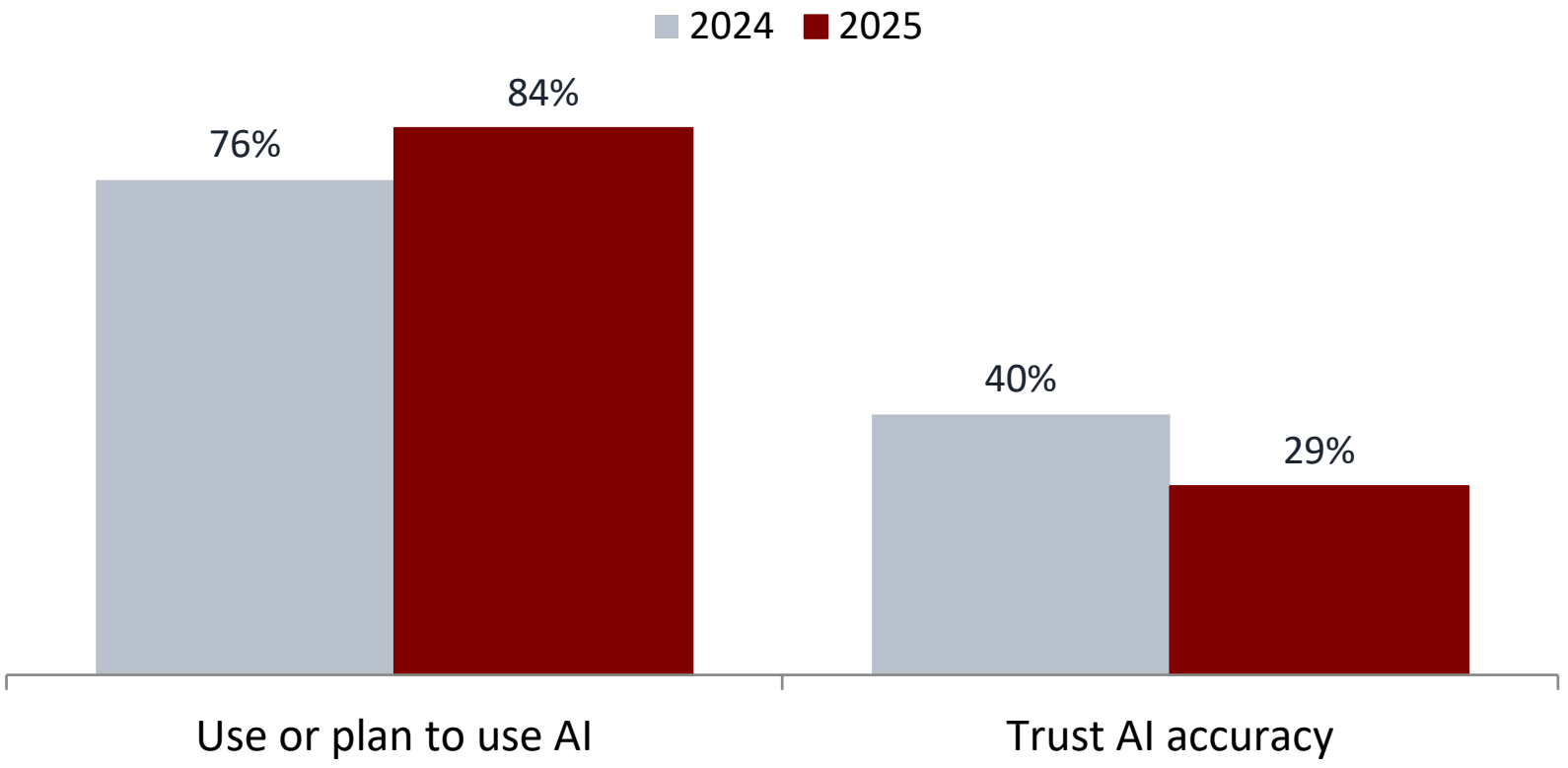
68%

expect employers to require AI skills soon

Source: JetBrains, State of Developer Ecosystem 2025 (24,534 developers, 194 countries)



AI Usage Up, Trust Down



Source: Stack Overflow Developer Survey 2025 (49,009 respondents)

66%

struggle with AI answers that are "almost right, but not quite“

46%

actively distrust AI accuracy



Three Layers of Java for AI (by Oracle)

1. Agentic development

- Build agents in Java
- Embabel, Spring AI, Oracle Code Assist

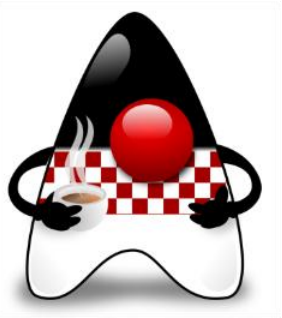
2. Model integration

- Add AI to existing business apps
- Helidon MCP, LangChain4j, OCI SDK

3. Training and deployment

- Run models on the JVM, on GPUs
- Panama, Valhalla, Babylon (future)

Source: Oracle JDK 26 launch, via DevOps.com (March 2026)



The Java AI Stack

Frameworks

- Spring AI 2.0
- LangChain4j
- Quarkus LangChain4j

Agents

- Embabel
- Koog (Kotlin)
- Google ADK for Java

Protocols

- MCP Java SDK 2.0
- A2A Java SDK
- Jakarta Agentic AI

Runtime & GPU

- Vector API, FFM
- Babylon / HAT
- TornadoVM, DJL

May 2025 — Spring AI 1.0, LangChain4j 1.0, Embabel launched

Dec 2025 — MCP Java SDK and Spring AI reach early adopters

Mar 2026 — JavaOne: 'Java for an AI World' keynote

May 2026 — Koog stable at KotlinConf; ADK for Java 1.0

Jun 2026 — Spring AI 2.0 GA with MCP annotations

Jul 2026 — Jakarta Agentic AI 1.0, first milestone



Views from the Community

- *The LLM becomes part of the application, not just an endpoint.*

  Rod Johnson, creator of Spring and Embabel (The New Stack, April 2026)

- *Python may start the story, but Java is where many of these AI applications truly come to life.*

  Richard Fichtner, InfoQ Java Trends Report 2025



Predictions for Java

Java as enterprise AI glue

The layer between business data and LLMs

Compute demand jumps

AI features in CRM and ERP need much more processing power

Vibe coding has limits

Mission-critical systems still need experienced developers

Modernisation shifts

From cloud migration to Java upgrades driven by cost, risk and AI

Source: Simon Ritter (Azul), A Java Champion explains his Java predictions for 2026



Free Places to Learn Modern Java

dev.java

Official tutorials and learning paths

inside.java

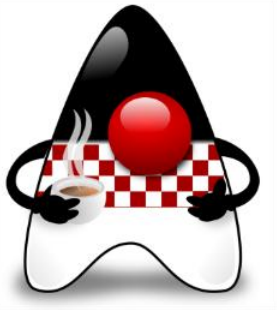
News, Newscast, JEP Café from Oracle's Java team

JavaOne 2026 videos

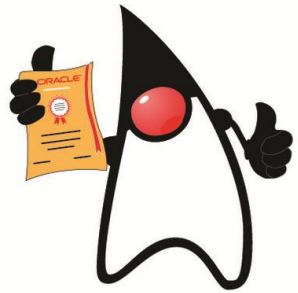
Keynotes and talks on the Java YouTube channel

HUJAK & JUGs

Meetups, JavaCro, Javantura, the hallway track



Oracle Java Certification



- **Oracle Certified Professional (OCP): Java SE 25 Developer**
  - Exam: Java SE 25 Developer
    1Z0-831
- There are also (but why?):
  - Oracle Certified Professional (OCP): Java SE 21 Developer
    - Exam: Java SE 21 Developer 1Z0-830

↑ Exam 1Z0-831: Java SE 25 Developer Professional

	Delivery	Format	Duration	Number of Questions	Passing Score
Exam	Proctored Online	Multiple Choice	120 Minutes	50	68%



Why Are the Best Years Ahead?

What is coming, and what we should do about it



The Best Years Are Ahead

31 → 100B

31 years old, on the way to 100 billion JVMs

- Three decades of compatibility, community, and disciplined evolution
- The next decade runs on the same platform, with AI built in



What Is Coming Next

Mar 2027

JDK 28: value classes preview, JSON API incubator, strict field init

2027

Structured concurrency and lazy constants heading to final

Sep 2027

JDK 29: the next LTS

Beyond

Null-restricted types, specialised generics, Vector API final, AOT code



Is Java Really Progressing?

- A **new release** every 6 months, on time, for 9 years
- An **LTS** every 2 years for production
- Long-running **projects** landing: Loom done, Valhalla in preview
- Free **performance** by default: compact headers, G1, AOT cache
- **Post-quantum** ready and **integrity** by default
- A **growing AI stack**: Spring AI, LangChain4j, Embabel, MCP
- Java is (mostly) **free**: pick any OpenJDK



Instead of the Conclusion

- Use **Java 25** LTS
- At least **Java 21** LTS
  - on Oracle JDK 21, check your licence
- Any **OpenJDK** build, it is up to you
- Leave **Java 8, 11 and 17** behind
- **Migrate** with every LTS (every 2 years) if possible
- Try the **previews**: structured concurrency, lazy constants, value classes
- Get involved with **HUJAK** and come to more **conferences**



See you on Java 29 LTS



Conferences and Events

- **JavaCro'27** next year, also in October, in Rovinj – **see you again!** ☺

Other #Java conferences or events HUJAK organizes or supports, such as:

- **JCON Europe**, April 26-29, 2027, Cologne, Germany 2027.europe.jcon.one – **FREE TICKETS FOR HUJAK MEMBERS!!!**
- **JCON Slovenia**, May 2027, Portorož, Slovenia
- "New" **AI&Java-dventure**, Spring 2027 Zagreb, Croatia
- **Contribute** to **HUJAK's** activities! ☺



Thank you & greetings from HUJAK!

- Web page **hujak.hr**

LinkedIn group **HUJAK**

Facebook group page **HUJAK.hr**

X/Twitter profile **@HUJAK_hr**

