


Java Update

Java 23




Georges Saab

Senior Vice President, Development, Java | Chair, OpenJDK Governing Board

 @gsaab

Chad Arimura

Vice President, Java Developer Relations

 @chadarimura

Oct 16, 2024



JDK 23

12 JDK Enhancement Proposals (JEPs) delivered

Plus thousands of performance, stability and security updates

Language Improvements

- 455: Primitive Types in Patterns, instanceof, and switch Preview
- 476: Module Import Declarations Preview
- 477: Implicitly Declared Classes and Instance Main Methods Third Preview
- 482: Flexible Constructor Bodies Second Preview

Performance and Runtime

- 474: ZGC: Generational Mode by Default

Tools

- 467: Markdown Document Comments

Libraries

- 466: Class-File API Second Preview
- 469: Vector API Eighth Incubator
- 473: Stream Gatherers Second Preview
- 480: Structured Concurrency Third Preview
- 481: Scoped Values Third Preview

Stewardship

- 471: Deprecate the Memory-Access Methods in `sun.misc.unsafe` for Removal

openjdk.org/projects/jdk/23

Thoughtful evolution and stewardship

String Templates Intended Goals

- ...for expressions that combine strings with run-time values:
 - Simplify and improve readability
 - Improve security
 - Enable transformation for non-strings
 - Allow extension outside of JDK

Summary

- **The process is working;** end-user feedback allows Java to evolve through thoughtful ecosystem involvement and evaluation.
- Collective end-user feedback helped determine to end and explore new approaches to achieve String Template intentions in future feature(s)
- First introduced as a preview feature in Java 21 (JEP 430)
- Introduced as a second preview in Java 22 (JEP 459)

Details:

<https://inside.java/2024/06/20/newscast-71/>

Experimental Features

A test-bed mechanism used to gather feedback on nontrivial HotSpot enhancements.

Incubator Modules

Enables the inclusion of JDK APIs and JDK tools that might one day, after improvements and stabilizations, be included and supported in the Java SE platform or in the JDK.

Preview Features

A new feature of the Java language, Java Virtual Machine, or Java SE API that is fully specified, fully implemented, and yet impermanent. It is available in a JDK feature release to provoke developer feedback based on real world use; this may lead to it becoming permanent in a future Java SE Platform.

Java and AI

Oracle Java AI Three-Pronged Strategy



Integration with
Enterprise Data and
Cloud Services



Making the Java
Platform even better
for Native AI



Connecting Business
Logic to Native AI
Libraries



Oracle Java AI triple-play advantage



OCI AI Services and
OCI SDK For Java



Native Java
Frameworks such as
Tribuo, LangChain4j,
CoreNLP



Integrate services with
business logic using
Panama & GraalPy





We continuously
evolve Java to meet
your **future app
development** needs

Java has **30 years** of
experience evolving with the
latest tech trends

Data-centric World

Amber

Continuously **improve developer productivity** through evolution of the Java language.

ZGC

Create a **scalable low-latency** garbage collector capable of handling terabyte heaps.

Cloud-powered World

Loom

Massively scale lightweight threads, **making concurrency simple again.**

Leyden

Improve the start-up time and time to peak performance of cloud applications.

AI-driven World

Babylon

Extend the reach of Java including to machine learning models and GPUs

Panama

Safe and efficient interoperation with native libraries and native Java.

Valhalla

Unify primitives and classes to **improve productivity and performance.**





March 17-20, 2025 | Redwood Shores, CA

javaone.com

Some of the prep: Integrity by default

OpenJDK

- Installing
- Contributing
- Sponsoring
- Developers' Guide
- Vulnerabilities
- JDK GA/EA Builds
- Mailing lists
- Wiki · IRC
- Bylaws · Census
- Legal
- Workshop**
- JEP Process**
- Source code**
- Mercurial
- GitHub
- Tools**
- Git
- jreg 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

JEP draft: Integrity by Default

Authors	Ron Pressler, Alex Buckley, & Mark Reinhold
Owner	Ron Pressler
Type	Informational
Scope	SE
Status	Draft
Relates to	JEP 261: Module System JEP 260: Encapsulate Most Internal APIs JEP 396: Strongly Encapsulate JDK Internals by Default JEP 403: Strongly Encapsulate JDK Internals JEP 451: Prepare to Disallow the Dynamic Loading of Agents JEP 471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal JEP 472: Prepare to Restrict the Use of JNI
Created	2023/04/13 16:06
Updated	2024/08/23 10:25
Issue	8305968


Summary

Developers expect that their code and data is protected against use that is unwanted or unwise. The Java Platform, however, contains unsafe APIs that can undermine this expectation, thereby damaging the correctness, maintainability, scalability, security, and performance of applications. Going forward, we will restrict the unsafe APIs so that, by default, libraries, frameworks, and tools cannot use them. Application authors will have the ability to override this default.

What is integrity?

The Oxford English Dictionary defines “integrity” as “the state of being whole and undivided; the condition of being sound in construction.”

And now, JEP 14: Tip and Tail



- Installing
- Contributing
- Sponsoring
- Developers' Guide
- Vulnerabilities
- JDK GA/EA Builds
- Mailing lists
- Wiki · IRC
- Bylaws · Census
- Legal
- Workshop
- JEP Process
- Source code
 - Mercurial
 - GitHub
- Tools
 - Git
 - 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)

JEP 14: The Tip & Tail Model of Library Development

<i>Authors</i>	Alex Buckley, Brian Goetz, & Ron Pressler
<i>Owner</i>	Alex Buckley
<i>Type</i>	Informational
<i>Scope</i>	JDK
<i>Status</i>	Active
<i>Discussion</i>	jdk dash dev at openjdk dot org
<i>Reviewed by</i>	Alan Bateman, Mark Reinhold, Paul Sandoz
<i>Created</i>	2024/09/30 23:14
<i>Updated</i>	2024/10/07 17:47
<i>Issue</i>	8341287

Summary

Tip & tail is a release model for software libraries that gives application developers a better experience while helping library developers innovate faster. The *tip* release of a library contains new features and bug fixes, while *tail* releases contain only critical bug fixes. As little as possible is backported from the tip to the tails. The JDK has used tip & tail since 2018 to deliver new features at a faster pace, as well as to provide reliable and predictable updates for users focused on stability.

Goals

- Help the Java ecosystem maintain the balance between innovating rapidly for new development and ensuring stability for long-term deployments.
- Recognize that application developers have diverse views about the kinds of changes that make it necessary to update libraries and the JDK.
- Ensure that library developers do not have to choose between supporting users of older JDKs and embracing new features (virtual threads, patterns, etc.) that excite users of newer JDKs.
- Do not constrain library release cycles, version schemes, or tool choices.



Thank you

<https://openjdk.org>

<https://dev.java>

<https://inside.java>

<https://youtube.com/java>

<https://github.com/java>



@Java | @OpenJDK

Appendix

A look back at the “bad old days”

- Previously had a coarse-grained, feature-boxed release model
 - 2-4 years between releases, frequent delays, no predictability
 - Expensive, heavyweight release management process
 - Irresistible temptation to integrate features “under the wire”
(And also to backport many improvements)
- This wasn’t working for anyone
 - Developers were frustrated by latency and delays to get new features
 - Late integrations reduced stability of GA release
 - Excessive backports risked stability of older trains
(and perversely, discouraged adoption of newer releases)

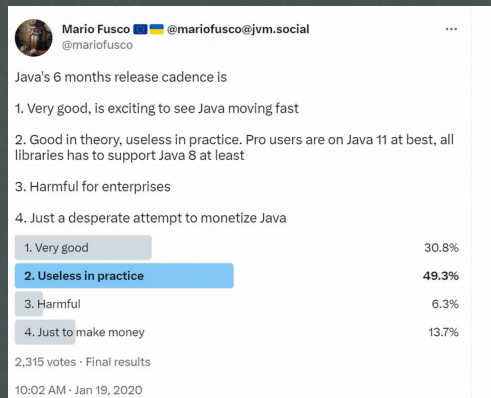
Turning the ship

- We first asked: what does the ecosystem really need?
 - Developers: want access to the latest features
 - Enterprises: want stability for deployment
- Old model made no one happy
- New model: “tip and tail”, with six-month tip release cadence
 - Features are integrated into the tip only when ready
 - If you miss the train, no problem, another is coming soon
 - Offer long-term commercial support (LTS) on select older releases (the “tail”)
 - These get security updates and critical fixes *only*, ensuring stability



Skepticism at first...

- The internal reaction was ... skeptical
 - “We can’t possibly run a release every six months, we’d get eaten by process overhead”
 - “We might not have anything to ship”
 - “We’re going to get overwhelmed with backports”
 - “We fear change”
- The ecosystem was even more skeptical...
 - “Java 8 is all we’ll ever need”
 - “We’ll get overwhelmed with releases”
 - “You can’t possibly maintain quality at that pace”
 - “We fear change”



There were some technical challenges

- The repo structure designed in the late nineties made changes across components take weeks
- The tests were managed separately from implementation code
- Lots of code complexity from trying to support multiple Java versions with the JDK held back progress
- Build system needed an overhaul
- Dependency management was too ad hoc (pets rather than cattle)
- Improvements to infrastructure to support self service automated build and test

Cultural Change was required

- More than switching methodologies, it was switching mindsets
 - Unlearning decades of experience and muscle memory
 - Learning to NOT panic about missing the train
- Don't backport what doesn't need to be backported
 - The tail is for stability, not for shiny features
- Learn to break features down into smaller deliverables
- Right size the release management processes

The payoff

- This worked out better than we could have hoped!
 - Release process could be slashed because release risk was so reduced
 - Backports could be slashed because many fewer were needed
 - Reduced backports meant more stability for LTS releases
 - All this meant more time for development
 - More features, faster!
- Everyone is happy
 - Developers get features delivered earlier, more rapid progress, predictability
 - Enterprises get commercially supported stable releases
 - JDK developers spend more time developing, less in meetings
 - Predictable, repeatable process reduced stress for everyone



Amazing-Cicada5536 · vor 21 Tagen

The new release model is probably the best thing that happened to Java on the management side — instead of the usual churn of “this feature has to be done till this deadline”, which might work for that boring CRUD feature but is absolutely terrible for such a complex program like the JVM, it allows for each new development to take as much time as necessary. If it didn't get ready for release N, just continue to work on it and 6 months later it can be delivered. It is especially important as many of these features have non-trivial interactions with each others.



StoneOfTriumph · vor 21 Tagen

What some people forget is that the JDK is advancing at a faster pace with smaller deliverables more frequency than before, so while it seems that they released a buttload of versions, the releases themselves include a smaller delta, so the jump to 8 to 17 isn't that bad in terms of breaking changes, not near as difficult to perform as 6 to 8.

When you take the time to actually look at the changelog, when you take the time to actually upgrade your project's java version source/targets to a higher version as part of a proof of concept/spike/timebox, there's much less than you may think.



Joram2 · vor 21 Tagen

Java 20 is basically a patch release on Java 19. But it's important + necessary.



henk53 · vor 21 Tagen

Why don't we backport *everything* to the supported LTS releases? Then those supported LTS releases will be virtually identical to Java 20 each ;)



1



Antworten Teilen ...



JakeWharton · vor 21 Tagen

That's just called updating to the latest. The best LTS release remains and will remain simply tracking the latest version.



MyFavouriteNick · vor 20 Tagen

For all the in-house projects that I maintain the upgrade from OpenJDK 17 to 19 (and now 20) has been a simple version bump too.


In my opinion, 12/13/14/15/16 can and should be used in production. But you don't use just 12 or 16 and leave it there, you always depend on the latest release, and plan accordingly. 6 months of support is absolutely fine if an adequately set up CI is in place and you have good end-to-end test coverage.



TheCountRushmore · vor 21 Tagen

This contains lots of other bug fixes and improvements. It's a full production quality release.

And now, JEP 14: Tip and Tail



- Installing
- Contributing
- Sponsoring
- Developers' Guide
- Vulnerabilities
- JDK GA/EA Builds
- Mailing lists
- Wiki · IRC
- Bylaws · Census
- Legal
- Workshop
- JEP Process
- Source code
 - Mercurial
 - GitHub
- Tools
 - Git
 - 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)

JEP 14: The Tip & Tail Model of Library Development

<i>Authors</i>	Alex Buckley, Brian Goetz, & Ron Pressler
<i>Owner</i>	Alex Buckley
<i>Type</i>	Informational
<i>Scope</i>	JDK
<i>Status</i>	Active
<i>Discussion</i>	jdk dash dev at openjdk dot org
<i>Reviewed by</i>	Alan Bateman, Mark Reinhold, Paul Sandoz
<i>Created</i>	2024/09/30 23:14
<i>Updated</i>	2024/10/07 17:47
<i>Issue</i>	8341287

Summary

Tip & tail is a release model for software libraries that gives application developers a better experience while helping library developers innovate faster. The *tip* release of a library contains new features and bug fixes, while *tail* releases contain only critical bug fixes. As little as possible is backported from the tip to the tails. The JDK has used tip & tail since 2018 to deliver new features at a faster pace, as well as to provide reliable and predictable updates for users focused on stability.

Goals

- Help the Java ecosystem maintain the balance between innovating rapidly for new development and ensuring stability for long-term deployments.
- Recognize that application developers have diverse views about the kinds of changes that make it necessary to update libraries and the JDK.
- Ensure that library developers do not have to choose between supporting users of older JDKs and embracing new features (virtual threads, patterns, etc.) that excite users of newer JDKs.
- Do not constrain library release cycles, version schemes, or tool choices.