Java Magazine

Java 17

The art of long-term support and what LTS means for the Java ecosystem

Donald Smith

September 8, 2021

Text Size 100%:

Here’s what Java 17 has in common with Java 11 and Java 8.

Download a PDF of this article

In June 2018, just over three years ago, Oracle and other participants in the Java ecosystem announced a change to the release cadence model for Java SE.

Rather than having a major release planned for every two to three years (which would often become three to four years), a new six-month feature-release-train model would be used: Every three years, a release would be designated as *Long-Term Support* (LTS) and receive quarterly security, stability, and performance updates only. This pattern borrowed shamelessly from the Mozilla Firefox release model but tweaked it to be more aligned with the requirements of a development platform.

The first version of Java released under that model was Java SE 11.

The release of Java SE 17, the second LTS release under the new model, is imminent, and this article will provide a refresher on how Java SE releases work. I’ll also offer some commentary on what has worked well over the past three years and what further refinements you should expect going forward.

[On September 14, 2021, Oracle proposed shifting the cadence of JDK LTS releases from every three years to every

[On September 14, 2021, Oracle proposed shifting the cadence of JDK LTS releases from every three years to every two years. If accepted, this means that the next JDK LTS release after JDK 17 will be JDK 21, rather than JDK 23. —*Ed*.]

The six-month feature-release model

Under the feature-release model, the Java platform developers can work on features and release those features within *any* six-month window—but only when the features are ready. Contrast that to the old legacy major-release model, where the Java platform developers felt enormous pressure to push features into a release; otherwise they would have to wait several years for the next cycle.

Meanwhile, application developers now enjoy a steady cadence of bite-size features on a predictable timeline. That's a lot better than having Java developers trying to consume hundreds of changes all at once every few years.

Has it worked? Three years into the new model, developer surveys show that between a quarter and a half of developers use the latest six-month Java release as their day-to-day version. Half of those said they have applications in production on the latest release.

What about the rest?

It is well understood and expected that not all developers or organizations would want to consume feature releases on a six-month cadence. More conservative organizations, especially, want to solidify a development stack around a single version and not take on risks associated with the introduction of new features. This is where Java LTS releases come into play.

LTS focuses on stability

Java LTS releases, such as Java 11 and Java 17, are similar to Firefox's Extended Support Releases. Oracle's updates to Java LTS releases provide only stability, security, and performance improvements—not new features. This reduces the risk that an update could break interaction with a tool or library. Organizations can count on Java LTS releases being available for at least eight years, providing ample time for toolchains to solidify and for developers to transition to another LTS several years later.

The LTS model allows technology providers to zero in on particular versions in the longer-term support of their products. After all, it would be impractical to expect platform providers and toolchains to provide multiple years of support on every six-month feature release. Very quickly there would be dozens of versions in need of support, as well as a fragmented user base that would be impractical to manage.

The sweet-spot timing for versions that get the LTS treatment is subjective. Historically, if you look back from Java 1.2 to Java 8, there were three to four years between major releases. The new feature-release model has three years between Java 11 and Java 17. For very conservative organizations this three-year interval is ideal, but as more developers use modern tools and techniques, there is increasing demand for Oracle to offer Java LTS releases on a shorter cycle, perhaps every two years.

Let's be clear: Each provider of Java platform binaries offers its own timelines and support offerings. The default at Oracle is that there will be eight years of support for a Java SE LTS release. For Java 8, LTS has already been extended through at least 2030, meaning that this version will have had at least 16 years of support when it's finally retired!

Meanwhile, versions such as Java 7 and Java 11 are unlikely to have support extensions. Extensions are based simply on adoption and on whether the organizations providing the respective binaries feel it's valuable to continue offering (commercial) support.

Conclusion

The current LTS versions of Java are Java 7, Java 8, Java 11, and soon, Java 17. Java 11 and Java 17 came out under the new feature-release cadence and exactly three years apart; Java 7 and Java 8 are from the legacy major-release model. The support is still the same, though, with each LTS version of Java receiving performance, stability, and security updates only.

Oracle intends to support the Java LTS releases as follows:

- Java 7 through 2022
- Java 8 through at least 2030
- Java 11 through 2026
- Java 17 through at least 2029

Oracle wants developers and organizations to have the best of both worlds: Only you can decide if it makes sense to keep up with the six-month releases and digest new features on a regular basis—or deploy to a specific LTS version for a longer haul. Organizations, of course, may mix and match their approach, for example, by keeping up with the six-month cadence during development but then locking into an LTS release once code is in production.

Dig deeper

- It's time to move your applications to Java 17. Here's why. And here's how.
- A quick summary on the new Java SE subscription
- Oracle Java SE support roadmap
- Moving the JDK to a two year LTS cadence

Donald Smith
Sr. Director of Product Management

Previous Post

Modern file input/output with Java Path API and Files helper methods

Ben Evans | 13 min read

Next Post

Java is criminally underhyped

Jackson Roberts | 7 min read