# Java 18

Java 18 is now available for your teams! It's the ninth feature release delivered on Oracle's six-month release cadence, giving developers, end-users, and enterprises a level of predictability they can count on. Although Java 18 is not a long-term support release (LTS), it does provide additional enhancements from Java 17. Oracle also announced plans to shorten the time between LTS releases from three years to two years; Java 21 was released in September 2023. Java 18 includes nine features, which is slightly fewer than the number of features in previous releases — 14 features in Java 17, 17 features in Java 16, and 14 features in Java 15.

- Release Date: March 22, 2022
- Not a Long-Term Support Version
- End of Life Date: September 22, 2022

It is important to note that two features are incubator modules, and one is a preview language feature. Incubator modules allow non-final APIs and tools to be used by the community to gather feedback and improve the overall quality of the platform. Whereas preview features are fully implemented language or VM features but are not permanent. They are built based on use cases and made available to gain feedback before becoming a standard feature in future releases. Afterall, we all build Java, together!

## New JEPs in Java 18

As with previous releases, Java 18 continues to build on the contributions from many developers and organizations in the OpenJDK community.

### Pattern Matching for Switch (Second Preview)

The Pattern matching for switch feature entered its second preview (JEP 420). There are only minor changes from the first preview, with regards to improved exhaustiveness check and clarification of order of checks with regards to dominance checking. Overall, it looks like this feature is on track to be part of the standard in Java 19.

### Improvements to Javac Compiler

Java 18 also includes several improvements to the javac compiler itself. This could help reduce unintentional memory leaks caused by inner classes. What many developers might not be aware of is that instances of inner classes — including local and anonymous inner classes — carry an implicit reference to an instance of the enclosing class. On the language level, this is what enables the developer to access members on the enclosed instance. But often, this access is not used. So, the implicit reference is a potential memory leak that can keep the enclosing instance alive longer than needed. The improvement to javac means that now inner classes will only have the implicit reference to the outer!

### Expand Checks of Javac's Serial Lint Warning

Javac's linter has also been expanded with more serialization warnings. It not only checks for the existence of the serialVersionUID field, but now also performs checks. For example, it can check for the existence of zero-arg constructors in the class hierarchy and if the type of the fields in the class can be serialized. The new warnings can be suppressed with @SuppressWarnings("serial") like other linter warnings.

## Code Snippets in Java API Documentation

The Javadoc compiler has also seen improvements. It now offers better support for code snippets as part of the documentation (JEP 413), including support for syntax highlights. The addition of the @snippet tag also allows IDEs and other tooling to distinguish code snippets more easily from other documentation. This can provide code insight, highlight, and linting during editing.  JDK 18 also includes minor improvements to the APIs used to invoke the compiler pragmatically, including better support for logging warnings and errors from annotation processors.

## String Deduplication

String Deduplication has been a feature of the G1 garbage collector since Java 8 (JEP 192). It finds identical strings in the heap, and automatically deduplicates them. This reduces the memory footprint. While the original JEP was only intended for G1, the Shenandoah GC also supported it when it was included in OpenJDK. With the release of JDK 18, the support has been extended to three additional garbage collectors: SerialGC, ParallelGC, and ZGC. Other GC improvements in JDK 18 include the ability to increase the heap region size in the G1 GC up to 512 MB, which can help mitigate fragmentation with large objects in the heap.

## Finalization Deprecated for Removal

JDK 18 also deprecated finalization for removal (JEP 421). Finalization works as a callback that is invoked when an object is being garbage collected but has several issues. This can include the adverse effect that it can cause resurrection of the object and potential performance impact during garbage collection.

## Internet-Address Resolution SPI

The ability to control the internet address resolution is an interesting addition to the JDK. This enhancement enables developers to register a ServiceProvider that handles hostname to IP resolutions done by the InetAddress classes. A great use-case for this is unit testing. For example, you might prefer a hostname that resolves to a specific internal system for testing purposes. Previously, you would have had to edit the OS hosts file to achieve comparable results. When it comes to testing purposes, this enhancement lacks control over caching of the results, as once a hostname has been resolved it is cached. Control over the caching would enable testing of the same hostname within the lifetime of a JVM instance, but have it resolve to different IPs.

## Simple Web Server

Another interesting addition is the inclusion of a simple web server. The JDK has included a HTTP server for a very long time, but it is somewhat more complicated, allowing dynamic resolution, etc. The purpose of the web server that is part of JEP 408 is much simpler. It is to serve static data, and that’s it. For testing, ad-hoc development, and more, access to a file web server can help a lot. Especially in conjunction with the added Inernet-Address Resolution SPI, this simple web server could become a very handy tool for testing purposes.

## Final Thoughts on Java 18

Although Java 18 might not be a headliner, it continues to deliver value on a regular cadence. Following a time-based release model, rather than a feature-based release model, means that Oracle can get JDK 18 features out to the public for use. This will allow them to continuously improve features that help Java teams innovate.