Java 17

In this tutorial, we'll talk about the news related to the new version of the Java ecosystem, Java SE 17, including the new features and the changes in its release process, LTS support and licenses.

1.. List of JEPs

First, let's talk about what can affect the everyday work in the life of Java developers.

1.1 Restore Always-Strict Floating-Point Semantics (JEP 306)

This JEP is mainly for scientific applications, and it makes floating-point operations consistently strict. The default floating-point operations are strict or strictfp, both of which guarantee the same results from the floating-point calculations on every platform. Before Java 1.2, strictfp behavior was the default one as well. However, because of hardware issues, the architects changed, and the keyword strictfp was necessary to re-enable such behavior. So, there is no need to use this keyword anymore.

1.2 Enhanced Pseudo-Random Number Generators (JEP 356)

Also related to more special use cases, JEP 356 provides new interfaces and implementations for Pseudo-Random Number Generators (PRNG). So, it's easier to use different algorithms interchangeably, and it also offers better support for stream-based programming:

```
public IntStream getPseudoInts(String algorithm, int streamSize) {
    // returns an IntStream with size @streamSize of random numbers generated using the @algorithm
    // where the lower bound is 0 and the upper is 100 (exclusive)
    return RandomGeneratorFactory.of(algorithm)
            .create()
            .ints(streamSize, 0,100);
}
```

Legacy random classes, such as **java.util.Random**, **SplittableRandom** and **SecureRandom** now extend the new **RandomGenerator** interface.

1.3 New macOS Rendering Pipeline (JEP 382)

This JEP implements a Java 2D internal rendering pipeline for macOS since Apple deprecated the OpenGL API (in macOS 10.14), used internally in Swing GUI. The new implementation uses the Apple Metal API, and apart from the internal engine, there were no changes to the existing APIs.

1.4 macOS/AArch64 Port (JEP 391)

Apple announced a long-term plan to transition its computer line from X64 to AArch64. This JEP ports the JDK to run on AArch64 in macOS platforms.

1.5 Deprecate the Applet API for Removal (JEP 398)

Although this may be sad for many Java developers who started their development career using Applet APIs, many web browsers have already removed their support for Java plugins. As the

API became irrelevant, this version marked it for removal even though it has been marked as deprecated since version 9.

1.6 Strongly Encapsulate JDK Internals (JEP 403)

JEP 403 represents one more step toward strongly encapsulating JDK internals since it removes the flag –illegal-access. The platform will ignore the flag, and if the flag is present, the console will issue a message informing the discontinuation of the flag. This feature will prevent JDK users from accessing internal APIs, except for critical ones like sun.misc.Unsafe.

1.7 Pattern Matching for Switch (Preview) (JEP 406)

This is another step toward pattern matching by enhancing pattern matching for switch expressions and statements. It reduces the boilerplate necessary to define those expressions and improves the expressiveness of the language. Let’s see two examples of the new capabilities:

```
static record Human (String name, int age, String profession) {}

public String checkObject(Object obj) {
    return switch (obj) {
        case Human h -> "Name: %s, age: %s and profession: %s".formatted(h.name(), h.age(), h.profession());
        case Circle c -> "This is a circle";
        case Shape s -> "It is just a shape";
        case null -> "It is null";
        default -> "It is an object";
    };
}

public String checkShape(Shape shape) {
    return switch (shape) {
        case Triangle t && (t.getNumberOfSides() != 3) -> "This is a weird triangle";
        case Circle c && (c.getNumberOfSides() != 0) -> "This is a weird circle";
        default -> "Just a normal shape";
    };
}
```

1.8 Remove RMI Activation (JEP 407)

Marked for removal in version 15, this JEP removed the RMI activation API from the platform in version 17.

1.9 Sealed Classes (JEP 409)

Sealed classes are part of Project Amber, and this JEP officially introduces a new feature to the language, although it was available in preview mode in the JDK versions 15 and 16. The feature restricts which other classes or interfaces may extend or implement a sealed component. Showing another improvement related to pattern matching combined with the JEP 406 will allow a more sophisticated and cleaner inspection of the type, cast and act code pattern. Let’s see it in action:

```
int getNumberOfSides(Shape shape) {
    return switch (shape) {
        case WeirdTriangle t -> t.getNumberOfSides();
        case Circle c -> c.getNumberOfSides();
        case Triangle t -> t.getNumberOfSides();
        case Rectangle r -> r.getNumberOfSides();
        case Square s -> s.getNumberOfSides();
    };
}
```

1.10 Remove the Experimental AOT and JIT Compiler (JEP 410)

Introduced into JDK 9 and JDK 10, respectively, as experimental features, the Ahead-Of-Time (AOT) compilation (JEP 295) and Just-In-Time (JIT) compiler from GraalVM (JEP-317) were features with a high cost of maintenance. On the other hand, they had no significant adoption. Because of that, this JEP removed them from the platform, but developers can still leverage them using GraalVM.

1.11 Deprecate the Security Manager for Removal (JEP 411)

The security manager aimed to secure client-side Java code is yet another feature marked for removal due to not being relevant anymore.

1.12 Foreign Function and Memory API (Incubator) (JEP 412)

The Foreign Function and Memory API allow Java developers to access code from outside the JVM and manage memory out of the heap. The goal is to replace the JNI API and improve the security and performance compared to the old one. This API is another feature developed by Project Panama, and it has been evolved and predeceased by JEPs 393, 389, 383 and 370. With this feature, we can make a call to a C library from a Java class:

```java
private static final SymbolLookup libLookup;

static {
    // loads a particular C library
    var path = JEP412.class.getResource("/print_name.so").getPath();
    System.load(path);
    libLookup = SymbolLookup.loaderLookup();
}
```

First, it is necessary to load the target library we wish to invoke via the API. Next, we need to specify the signature of the target method and finally call it:

```java
public String getPrintNameFormat(String name) {

    var printMethod = libLookup.lookup("printName");

    if (printMethod.isPresent()) {
        var methodReference = CLinker.getInstance()
            .downcallHandle(
                printMethod.get(),
                MethodType.methodType(MemoryAddress.class, MemoryAddress.class),
                FunctionDescriptor.of(CLinker.C_POINTER, CLinker.C_POINTER)
            );

        try {
            var nativeString = CLinker.toCString(name, newImplicitScope());
            var invokeReturn = methodReference.invoke(nativeString.address());
            var memoryAddress = (MemoryAddress) invokeReturn;
            return CLinker.toJavaString(memoryAddress);
        } catch (Throwable throwable) {
            throw new RuntimeException(throwable);
        }
    }
    throw new RuntimeException("printName function not found.");
}
```

1.13 Vector API (Second Incubator) (JEP 414)

The Vector API deals with the SIMD (Single Instruction, Multiple Data) type of operation, meaning various sets of instructions executed in parallel. It leverages specialized CPU hardware that supports vector instructions and allows the execution of such instructions as pipelines. As a result, the new API will enable developers to implement more efficient code, leveraging the potential of the underlying hardware. Everyday use cases for this operation are scientific algebra linear applications, image processing, character processing, and any heavy arithmetic application or any application that needs to apply an operation for multiple independent operands. Let's use the API to illustrate a simple vector multiplication example:

```
public void newVectorComputation(float[] a, float[] b, float[] c) {
    for (var i = 0; i < a.length; i += SPECIES.length()) {
        var m = SPECIES.indexInRange(i, a.length);
        var va = FloatVector.fromArray(SPECIES, a, i, m);
        var vb = FloatVector.fromArray(SPECIES, b, i, m);
        var vc = va.mul(vb);
        vc.intoArray(c, i, m);
    }
}

public void commonVectorComputation(float[] a, float[] b, float[] c) {
    for (var i = 0; i < a.length; i ++) {
        c[i] = a[i] * b[i];
    }
}
```

1.14 Context-Specific Deserialization Filters (JEP 415)

JEP 290, first introduced in JDK 9, enabled us to validate incoming serialized data from untrusted sources, a common source of many security issues. That validation happens at the JVM level, giving more security and robustness. With JEP 415, applications can configure context-specific and dynamically selected deserialization filters defined at the JVM level. Each deserialization operation will invoke such filters.

2.. LTS Definition

The changes don't stay only in the code — processes are changing as well. Java platform releases have a widely known history of being long and imprecise. Despite being designed to have a three-year cadence between releases, it often became a four-year process. Moreover, given the new dynamic of the market where innovation and quick response became obligatory, the team responsible for the platform's evolution decided to change the release cadence to adapt to the new reality. As a result, a new six-month feature-release model has been adopted since Java 10 (released on March 20, 2018).

2.1 Six-Month Feature-Release Model

The new six-month feature-release model allows the platform developers to release features when they are ready. This removes the pressure of pushing the feature into the release. Otherwise, they would have to wait three to four years to make the feature available to the platform's users. The new model also improves the feedback cycle between users and the platform's architects. That's because features can be made available in an incubating mode and only released for general use after several interactions.

2.2 LTS Model

Since enterprise applications widely use Java, stability is critical. Besides, it’s costly to keep supporting and providing patch updates to all these versions. For this reason, the Long-Term Support (LTS) versions were created, offering users extended support. So, such versions naturally become more stable and secure due to bug fixes, performance improvements and security patches. In the case of Oracle, this support usually lasts for eight years.

Since the introduction of the changes in the release model, the LTS versions were Java SE 11 (released in September 2018) and Java SE 17 (released in September 2021). Nonetheless, version 17 brought something new to the model. In short, the interval between LTS versions is now two years instead of three, making Java 21 (planned for September 2023) probably the next LTS. Another point worth mentioning is that this release model is not new. It was copied shamelessly and adapted from other projects such as Mozilla Firefox, Ubuntu and others where the model proved itself.

3.. New Release Process

We based this article on the JEP 3, given that it describes all changes in the process. Please check it for further details. We’ll try to provide a concise summary of it here. Given the new model described above, combined with the continuous development of the platform and the new six-month release cadences (generally June and December), Java will move faster. The development team of the JDK will initiate the release cycle for the next feature release following the process described next. The process begins with the fork of the main-line. Then the development continues in a stabilization repository, JDK/JDK$N (for example, JDK17). There, the development continues focusing on the stabilization of the release. Before we delve deeper into the process, let’s clarify some terminology:

- Bugs: In this context, bugs means tickets or tasks:
- Current: These are either actual bugs related to the current version (the new one about to be released) or adjustments to new features already included in this version (new JEPs).
- Targeted: Related to the older versions and planned to be fixed or addressed in this new version.
- Priorities: Ranging from P1 to P5, where P1 is the most important, with the importance progressively diminishing down to P5

3.1 New Format

The stabilization process proceeds for the next three months:

The JDK/JDK$N repository works like a release branch, and at this point, no new JEPs of new JEPs go into the repository.

Next, the developments in this repository will be stabilized and ported to the main-line where other developments continue.

Ramp Down Phase 1 (RDP 1): Lasts between four and five weeks. Developers drop all currents P4-P5 and the targeted P1-P3 (depending on deferring, fix or enhancement). This means that P5+ test/docs bugs and targeted P3+ code bugs are optional.

Ramp Down Phase 2 (RDP 2): Lasts between three and four weeks. Now they postpone all currents P3-P5 and the targeted P1-P3 (depending on deferring, fix or enhancement).

Finally, the team publishes a release candidate build and makes it available to the public. This phase lasts between two and five weeks, and only current P1 fixes are addressed (using fix).

Once all those cycles finish, the new release becomes the General Availability (GA) version.

4.. What Is Next?

JDK architects continue working on many projects that aim to modernize the platform. The goal is to provide a better development experience and more robust and performant APIs. As a result, the JDK 18 should be out six months from now, although this version is not likely to contain significant or disruptive changes. We can follow the list of proposed JEPs targeted for this version in the official OpenJDK project portal. Another relevant piece of news that affects the current and future versions is the new no-fee terms and conditions license applied to the Oracle JDK distribution (or Hotspot). For most cases, Oracle offers its distribution free of cost for production and other environments, but there are a few exceptions. Again, please refer to the link. As mentioned before, the new process targets the next LTS version to be version 21, and the plan is to release it by September 2023.