InApp

Migrating from Java 8 to Java 11

A Guide Note

Prepared by InApp
March 2019

Table of Contents

Summary

Why this note?

Three significant events have happened recently in Java and Java development

1. JDK version 11 and 12 have been released
2. JDK 8 support is stopping from January 2019
3. Oracle is implementing commercial licensing for Java Development Kit

This is the first note for InApp's clients and its developers to understand the key reasons for the need to move to JDK 11 and our recommended procedure for moving to Java 11 and beyond.

Reason for move

The main reason to move on to Java 11 is the stoppage of support for Java 8 JDK and the consequent non-availability of patches and security fixes to the JDK and Java 8 libraries. The other reasons include the new features of the language and the Development Kit, which makes it useful to move.

Sticking on to the Java 8

For some time at-least we can stick on to JDK if we are supported by non-oracle vendors like IBM WebSphere and RedHat's JBoss as these vendors will release the security fixes. Another option is to use the Open Source JDK supported by communities including the Oracle Supported OpenJDK. This will remain free with community support.

Moving to Oracle Java 11

This may be the option most people will take. Moving to JDK 11 is not as difficult since most of the JDK 8 programs will run directly on JDK 11. There are tools available to run through your source code and check the same. We recommend that users move to JDK 11 with the existing code and probably minor variations. Then in the next phase try and take advantage of the new features of JDK 11.

NOTE: It may be please noted that this is NOT a universal document for all cases of Java usage. Each particular instance or project and implementation is unique in itself and have to be handled with care to its environment and priorities. This document is a high level document to make clients and our own Java team aware of the changes and InApp's generic approach.

Java - A Brief History

Java was originated under the leadership of James Gosling by the Sun Microsystems in 1995. The first version came out in Jan 1996. Its success came from it becoming a web programming language. The concepts of JSP, JDBC and others made it suitable for Enterprise Applications on the web. Java has been a prime product from Sun Microsystems until it was taken over by Oracle and it is now maintained by Oracle Corporation.

Java Editions

There were two popular streams of Java the Standard Edition and Enterprise Edition. They have a very mixed history and for more information, please look up the Wikipedia Pages on Java and J2EE.

Enterprise Edition (J2EE)

The Enterprise Edition was a popular edition and has been used in many applications and application frameworks like the IBM's Websphere, Oracle's Weblogic, RedHat's JBoss, Apache's Tomcat and many more.

Java EE was maintained by Oracle under the Java Community Process. On September 12, 2017, Oracle Corporation announced that it would submit Java EE to the Eclipse Foundation. The Eclipse Foundation was forced to change the name of Java EE because Oracle owns the trademark for the name "Java." On February 26, 2018, it was announced that the new name of Java EE will be Jakarta EE.

J2EE Users

InApp's advice will be to completely move out of J2EE and consider reengineering the application to Java SE or other programming environments like PHP, Python for back-end and HTML, JavaScript for the front end.

Standard Edition (SE)

Java Standard Edition has emerged as the popular edition post-2010 and Oracle continues to maintain it and develop it. Currently, it is on Version 12.

Java SE Releases

Java Standard Edition Release Dt.	Support Till	Major New Features
Java 8		
March, 2014 (LTS**)	Commercial Users January, 2019 Personal Users: End 2020	Lambda expressions, Method references, Functional interfaces, Stream API, Default methods
Java 9		
September, 2017 (Non LTS)	March, 2018	Platform Module System (Project Jigsaw) Interface Private Methods Try-With Resources Anonymous Classes
Java 10		
March, 2018 (Non LTS)	September, 2018	Application Data-Class Sharing Parallel Full GC for G1 Local-variable type inference
Java 11		
March, 2018 (Non LTS)	July 2019	Run Java File with single command Removed the Java EE and CORBA Modules Reactive HTTP/2 Client Epsilon GC

** LTS Versions are "major versions" where Oracle support is for a longer term.

For more information, please refer:
https://www.oracle.com/technetwork/java/java-se-support-roadmap.html

Java 11 and Further

Java SE Release Cadence

Oracle has moved from big bang major releases often at an interval of 3 years or more to a more streamlined fixed “feature releases”,

- new major release every six months (March and September)
- two minor updates for each (one and four months later)

There will not be any beta versions. All Java releases will be major releases which contain only features that are ready to be used. The new cadence makes it more manageable and predictable for third party tools vendors, as there will be a steady stream of smaller updates.

Java 11 Licensing

As of Java 11 Oracle now maintains two different JDK builds:

- Oracle’s JDK is fully commercial and applications can be developed and tested on the development environment. However we will have to pay if we need to use the application to run in production.

- Oracle’s OpenJDK (open source) – We can use this for free in any environment, like any open source library

All Java/JDK development is done in the public OpenJDK repository and then they are propagated to the Oracle JDK. Hence, Oracle JDK builds and OpenJDK builds will be essentially identical but with some cosmetic and packaging differences (mainly in the license and support model).

Since Java 11, as Oracle’s commercial JDK and Oracle’s OpenJDK builds are functionally the same, we should be able to run our applications on either without having to make any changes or losing any features.

Option: Staying with Java 8

The code on Java SE 8 will run on Java SE 11 without any changes. Hence there is no immediate need for reengineering. However, it is highly recommended to step to Java 11 platform as it has many features and makes the system easier to use.

Oracle JDK 8 is undergoing the “End of Public Updates” process, and there will be no more free updates for commercial use after January 2019.

Staying with Oracle's Java SE 8 has the following implications:

- **Security:** Theoretically, Oracle JDK 8 can be used indefinitely without updates. But your application is open to vulnerabilities and hacking as there are no further updates.
- **OS Updates:**The Linux OSes will update Java 8 using OpenJDK or other sources. On platforms such as Red Hat, Debian, Fedora, Arch, etc updates to the JDK are delivered via the operating system vendor. Companies like Red Hat have promised Java 8 updates until June 2023 in Red Hat Enterprise Linux - but they also have an "upstream first" policy, meaning they prefer to push fixes back to the "upstream" OpenJDK project.
- Users of other platforms like IBM WebSphere, RedHat JBoss, TomCat can depend upon the upgrades from these vendors and can stay on Java 8.
- Use the non-commercial build in a commercial setting - Oracle will provide builds of Oracle JDK 8 for non-commercial use until December 2020, so we could use this loophole but it is essentially illegal to use Java 8 for commercial purposes.

Companies can either go onto a paid support plan or use a Java SE 8 / OpenJDK 8 binary distribution from another provider or continue to use Oracle JDK 8 indefinitely without updates. If you are not using Oracle JDK 8, then your current Java SE 8 / OpenJDK 8 provider will provide updates and/or paid support plans to choose from.

Some options available include:

- Paying for support - A number of companies, including Azul, IBM, Oracle and Red Hat, offer ongoing support for Java. By paying the vendors, you get access to the stream of security patches and update releases with certain guarantees (as opposed to volunteer-led approaches).
- Companies build OpenJDK themselves - The stream of security patches * is published to a public Mercurial repository under the GPL license. The company will have to build OpenJDK themselves keeping track of commits to the repository.
- Use the builds from AdoptOpenJDK - AdoptOpenJDK community takes the stream of security patches * and turn them into releases. Their plan is to produce Java 8 builds until September 2023 or later (two years after Java 17 comes out). As it is a community build farm project there won't be any warranty or organised support.

Option: Moving to Java 11

Java 11 did not have any major language changes but has taken the advantage of features released in the earlier versions including Java 8, 9 and 10. Java 8 code will run under Java 11, 12 and further as they will be backward compatible. However it will be very useful to take advantage of the enhancements provided by Java 11.

Some of the core enhancements include:

1. Full support for containers

Containerizing applications in Docker makes it easy to abstract key application elements from the infrastructure and thus makes it easier to scale services with changing demand.

The JVM now recognizes constraints set by container control groups. Both memory and cpu constraints can be used to manage Java applications directly in containers. These include:

- Adhering to memory limits set in the container
- Setting available cpus in the container
- Setting cpu constraints in the container

2. Java Platform module system

The Java Platform module system introduces a new kind of Java programing component, the module, which is a named, self-describing collection of code and data. Now we can pack a customized subset of JRE, based on the individual needs of the applications that run on it and increase their efficiency manifold with a drastically reduced footprint and increased performance.

3. Supports HTTP/2 Requests And Responses

The new API makes a clean break with the past, by abandoning any attempt to maintain protocol independence. Instead, the API focuses solely on HTTP.

- It provides support for the new framing and connection handling parts of the protocol.
- It supports HTTP/1.1 and HTTP/2, both synchronous and asynchronous programming models,
- It handles request and response bodies in a reactive manner which gives you full control over the bytes going over the wire:

- You can throttle, you can stream (to conserve memory), and you can expose a result as soon as you found it (instead of waiting for the entire body to arrive).

- Allows clients to indicate cancellation of requests that a server has already started working on.

4. Support parallel full garbage collection on G1 - A scalable, low-latency garbage collector, ZGC or Z Garbage Collector, is added along with Epsilon GC, an experimental No-op Garbage Collector

5. Flight Recorder, Flight Recorder which earlier used to be a commercial add-on in Oracle JDK is now open sourced. JFR is a profiling tool used to gather diagnostics and profiling data from a running Java application. Its performance overhead is negligible and that's usually below 1%. Hence it can be used in production applications.

6. Heap allocation on alternative memory devices - Accessible via JVMTI, a low-overhead heap profiling is now available

7. Ahead-of-time compilation and GraalVM.

8. Transport Layer Security (TLS) 1.3.

9. JShell.

10. Support for "shebang" Java files! #!/bin/java

Migration from Java 8 to Java 11

Run Current Application on JDK11

The major incremental steps for migration are:

1. Run an existing Java application with JDK 11.
2. Compile the application with Java 11.
3. Modularize the application to use Module System (you are not required to create modules to have your code run on Java 11)

Run the application

First of all, download and install the latest JDK release. The application (jars) created with earlier Java versions can run on JDK 11 without major issues, Some exceptions are:

- If the code depends on Java EE or CORBA modules which were removed from JDK.
- Some libraries that need to be upgraded.
- Missing classes will be needed to add explicitly in case of class file errors please update the Java bytecode enhancement libraries like ASM, bytebuddy, javassist or cglib.

If your application starts successfully, look carefully at your tests and ensure that the behaviour is the same as on the JDK version you have been using. For example, a few early adopters have noticed that their dates and currencies are formatted differently. To ensure that the code works on the latest JDK release, understand the new features and changes in each of the JDK releases.

The following steps are iterative -

1. Update Third-Party Libraries

Everything needs to be updated, including the IDE, build tool, its plugins, and, most importantly, the dependencies. Some dependencies that we need to monitor (and versions that are known to work on Java 11):

- Build tools like Maven or Gradle
- IDE's like NetBeans, Eclipse, and IntelliJ IDEs all have versions available that include support for the latest JDK.
- Anything that operates on bytecode like ASM (7.0), Byte Buddy (1.9.0), cglib (3.2.8), or Javassist (3.23.1-GA). Since Java 9, the bytecode level is increased every six months, so we will have to update libraries like these pretty regularly.
- Anything that uses something that operates on bytecode like Spring (5.1), Hibernate, Mockito (2.20.0), etc.

Here are the recommended minimum versions for a few tools:

- IntelliJ IDEA: 2018.2
- Eclipse: Photon 4.9RC2 with Java 11 plugin
- Maven: 3.5.0
- Compiler plugin: 3.8.0
- Surefire and failsafe: 2.22.0
- Gradle: 5.0

2. Compile Your Application with Java 11 if needed

Compiling your code with the latest JDK compiler will ease migration to future releases since the code may depend on APIs and features, which have been identified as problematic.

3. Run jdeps on Your Code.

Run the jdeps tool on your application to see what packages and classes your applications and libraries depend on. If you use internal APIs, then jdeps may suggest replacements to help you to update your code.

Next Steps

Once the application is working on JDK 11, some suggestions that can help you get the most from the Java SE Platform:

1. If needed, cross-compile to an older release of the platform using the new --release flag in the javac tool.
2. Take advantage of your IDE's suggestions for updating your code with the latest features.
3. Find out if your code is using deprecated APIs by running the static analysis tool jdeprscan. These APIs can be removed from the JDK, but only with advance notice.

4. Get familiar with new features like multi-release JAR files (see jar)
5. Stick to supported apis
6. Use standardized behaviour
7. Use well-maintained projects
8. Keep dependencies and tools up to date
9. Consider using jlink

Start taking advantages of JDK 11 features

- The first initiative for larger organisations would be to move to Modular Structure of programs
- Use of the 'var' in variable declarations makes writing code simple and error free.
- JShell: A shell for trying out simple programs like in Python or Ruby and other scripting languages.
- The HTTP/2 features
- JFR (Java Flight Recorder) profiling tool included in the release
- Many more...

References

http://www.oracle.com/technetwork/java/javase/11-relnote-issues-5012449.html#NewFeature

https://wiki.openjdk.java.net/display/quality/Quality+Outreach