MIGRATING BEYOND JAVA 8

DALIA ABO SHEASHA
dalia@us.ibm.com
@DaliaShea

ABOUT ME

- Migration Tools Dev Lead @IBM
- Team Goal: painless migrations for Java apps
  - Banks, retail, insurance, etc
- Migration Tooling Development
  - Java migrations (Since Java 5)
  - On-premise to Cloud
- Application Server Development
  - Open Liberty
  - WebSphere Application Server

MIGRATING BEYOND JAVA 8

WHY MIGRATE?

WHY MIGRATE?

JDK 9

The goal of this Project was to produce an open-source reference implementation of the Java SE 9 Platform as defined by JSR 379 in the Java Community Process.

JDK 9 reached General Availability on 21 September 2017. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed a
Process, as amended by the JEP 2.0 proposal.

Features

102: Process API Updates
110: HTTP 2 Client
143: Improve Contended Locking
158: Unified JVM Logging
165: Compiler Control
193: Variable Handles
197: Segmented Code Cache
199: Smart Java Compilation, Phase Two
200: The Modular JDK
201: Modular Source Code
211: Elide Deprecation Warnings on Import Statements
212: Resolve Lint and Doclint Warnings
213: Milling Project Coin
214: Remove GC Combinations Deprecated in JDK 8
215: Tiered Attribution for javac
216: Process Import Statements Correctly
217: Annotations Pipeline 2.0
219: Datagram Transport Layer Security (DTLS)
220: Modular Run-Time Images
221: Simplified Doclet API
222: jshell: The Java Shell (Read-Eval-Print Loop)
223: New Version-String Scheme
224: HTML5 Javadoc
225: Javadoc Search
226: UTF-8 Property Files
227: Unicode 7.0
228: Add More Diagnostic Commands
229: Create PKCS12 Keystores by Default
231: Remove Launch-Time JRE Version Selection
232: Improve Secure Application Performance
233: Generate Run-Time Compiler Tests Automatically
235: Test Class-File Attributes Generated by javac
236: Parser API for Nashorn
237: Linux/AArch64 Port
238: Multi-Release JAR Files
240: Remove the JVM TI hprof Agent

JDK 10

JDK 10 is the open-source reference implementation of the Java SE 10 Platform as defined by JSR 383 in the Java Community Process.

JDK 10 reached General Availability on 20 March 2018. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal.

Features

286: Local-Variable Type Inference
296: Consolidate the JDK Forest into a Single Repository
304: Garbage-Collector Interface
307: Parallel Full GC for G1
310: Application Class-Data Sharing
312: Thread-Local Handshakes
313: Remove the Native-Header Generation
314: Additional Unicode Language-Tag Exten
316: Heap Allocation on Alternative Memory
317: Experimental Java-Based JIT Compiler
319: Root Certificates
322: Time-Based Release Versioning

JDK 11

JDK 11 is the open-source reference implementation
Platform as specified by by JSR 384 in the Java Comm

JDK 11 reached General Availability on 25 September
binaries under the GPL are available from Oracle; bin
follow shortly.

The features and schedule of this release were propo
Process, as amended by the JEP 2.0 proposal. The rele
JDK Release Process (JEP 3).

Features

181: Nest-Based Access Control
309: Dynamic Class-File Constants
315: Improve Aarch64 Intrinsics
318: Epsilon: A No-Op Garbage Collector
320: Remove the Java EE and CORBA Modules
321: HTTP Client (Standard)
323: Local-Variable Syntax for Lambda Parameters
324: Key Agreement with Curve25519 and Curve448
327: Unicode 10
328: Flight Recorder
329: ChaCha20 and Poly1305 Cryptographic Algorithms
330: Launch Single-File Source-Code Programs
331: Low-Overhead Heap Profiling
332: Transport Layer Security (TLS) 1.3
333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)
335: Deprecate the Nashorn JavaScript Engine
336: Deprecate the Pack200 Tools and API

JDK 12

JDK 12 is the open-source reference implementation of version 12 of the Java SE Platform as specified by by JSR 386 in the Java Community Process.

JDK 12 reached General Availability on 19 March 2019. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Features

189: Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)
230: Microbenchmark Suite
325: Switch Expressions (Preview)
334: JVM Constants API
340: One AArch64 Port, Not Two
341: Default CDS Archives
344: Abortable Mixed Collections for G1
346: Promptly Return Unused Committed Memory from G1

JDK 13

JDK 13 is the open-source reference implementation of version 1
Platform as specified by by JSR 388 in the Java Community Proce

JDK 13 reached General Availability on 17 September 2019. Prod
binaries under the GPL are available from Oracle; binaries from o
follow shortly.

The features and schedule of this release were proposed and tra
Process, as amended by the JEP 2.0 proposal. The release was pr
JDK Release Process (JEP 3).

Features

350: Dynamic CDS Archives
351: ZGC: Uncommit Unused Memory
353: Reimplement the Legacy Socket API
354: Switch Expressions (Preview)
355: Text Blocks (Preview)

JDK 14

JDK 14 is the open-source reference implementation of version 14 of the Java SE Platform as specified by by JSR 389 in the Java Community Process.

JDK 14 reached General Availability on 17 March 2020. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Features

305: Pattern Matching for instanceof (Preview)
343: Packaging Tool (Incubator)
345: NUMA-Aware Memory Allocation for G1
349: JFR Event Streaming
352: Non-Volatile Mapped Byte Buffers
358: Helpful NullPointerExceptions
359: Records (Preview)
361: Switch Expressions (Standard)
362: Deprecate the Solaris and SPARC Ports
363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector
364: ZGC on macOS
365: ZGC on Windows
366: Deprecate the ParallelScavenge + SerialOld GC Combination
367: Remove the Pack200 Tools and API
368: Text Blocks (Second Preview)
370: Foreign-Memory Access API (Incubator)

Tons of new cool features!

Feature Lists (JDK 9, JDK 10, JDK 11, JDK 12, JDK 13, JDK 14)

WHY MIGRATE?

- New Java 11+ features
  - var for local variables (JEP 286)
  - New methods in classes (Collection, etc) (JEP 269)
  - Launch Single-File Source-Code Programs (JEP 330)

```
// Before
List<String> myList = new ArrayList<String>();
myList.add("Georgia");
myList.add("Minnesota");
myList.add("Texas");
myList = Collections.unmodifiableList(myList);

// After
var myList = List.of("Georgia", "Minnesota", "Texas");
```

WHY MIGRATE?

Security Improvements (TLS 1.3)

WHY MIGRATE?

Library/tooling developers need to keep up

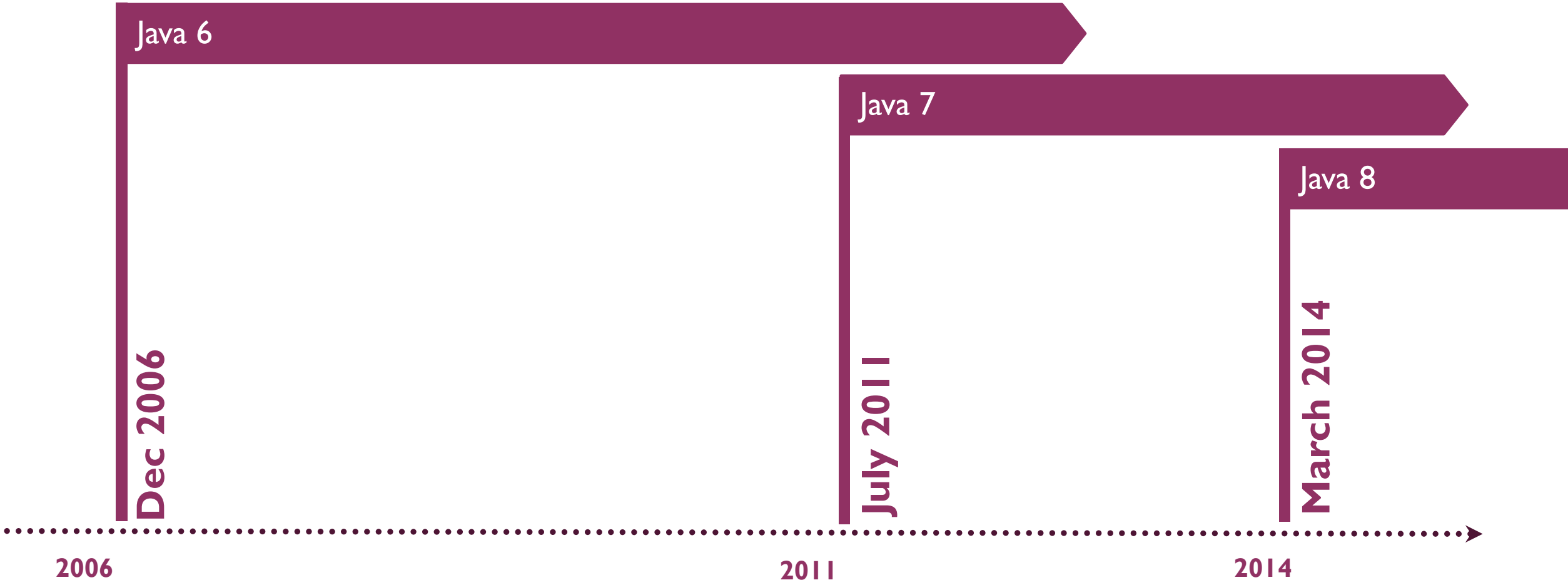
WHY MIGRATE?

…and of course: Java 8 will EOL

MIGRATING BEYOND JAVA 8

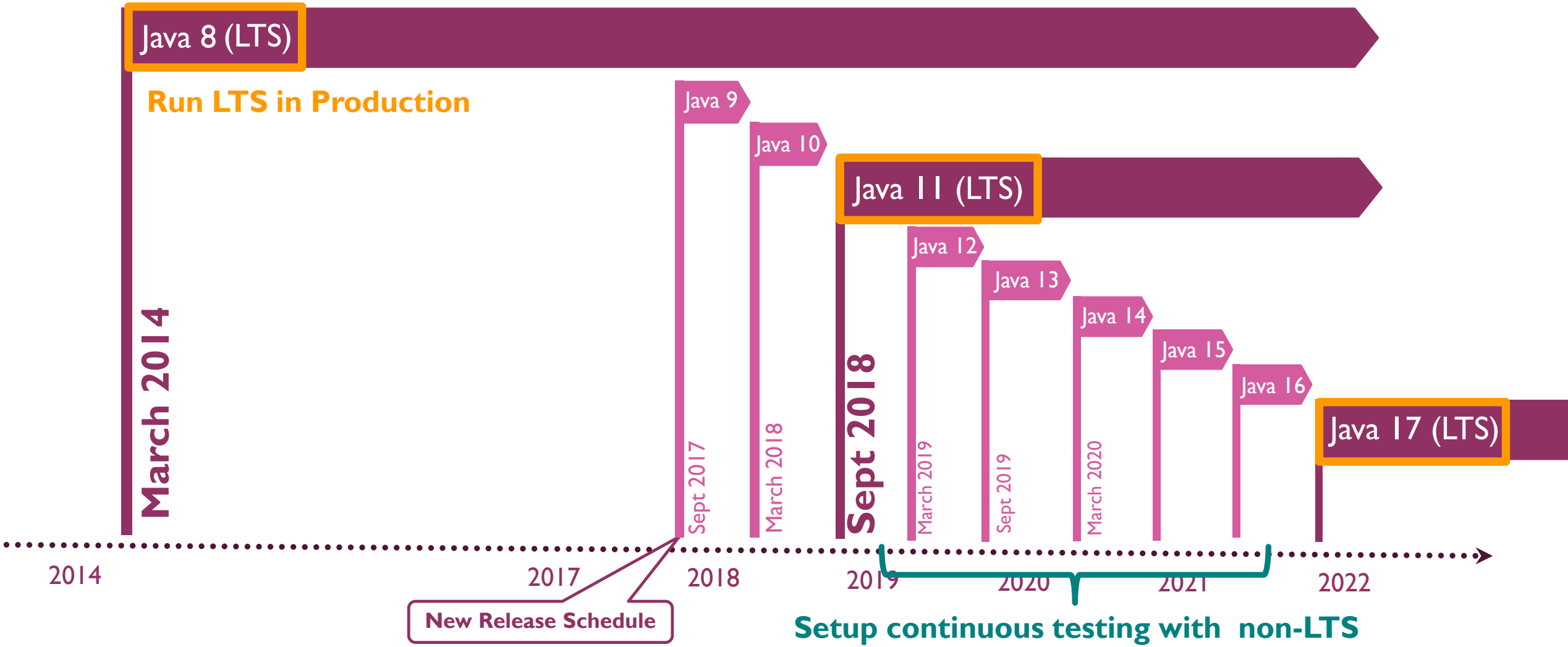
WHAT IS MY TARGET JAVA VERSION?

ONCE UPON A TIME



CURRENT TIMELINE

LTS: Long Term Support



AdoptOpenJDK

Prebuilt OpenJDK Binaries for Free!

Java™ is the world's leading programming language and platform. AdoptOpenJDK uses infrastructure, build and test scripts to produce prebuilt binaries from OpenJDK™ class libraries and a choice of either the OpenJDK HotSpot or Eclipse OpenJ9 VM.
All AdoptOpenJDK binaries and scripts are open source licensed and available for free.

Download for Windows x64

1. Choose a Version
   - ○ OpenJDK 8 (LTS)
   - ◉ OpenJDK 11 (LTS)
   - ○ OpenJDK 14 (Latest)

2. Choose a JVM Help Me Choose
   - ○ HotSpot
   - ◉ OpenJ9

Latest release
jdk-11.0.7+10.1_openj9-0.20.0

Other platforms

Release Archive & Nightly Builds

AdoptOpenJDK now also distributes OpenJDK upstream builds! (Built by Red Hat)

LOOKING FOR A FREE JAVA 11 BUILD?

Checkout AdoptOpenJDK!

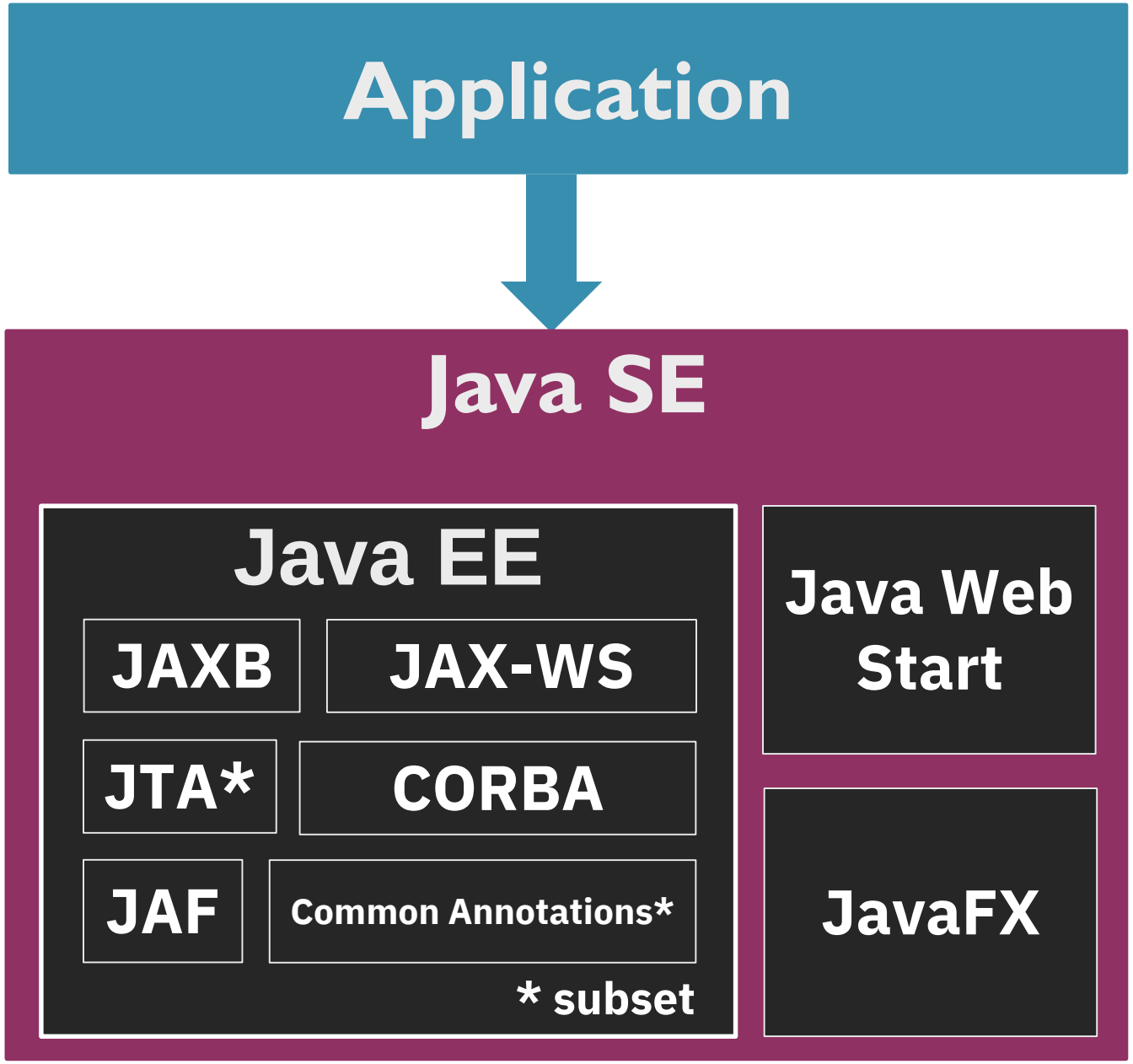
MIGRATING BEYOND JAVA 8

WHAT ARE THE TOP MIGRATION ISSUES FOR JAVA 8 → JAVA 11+?

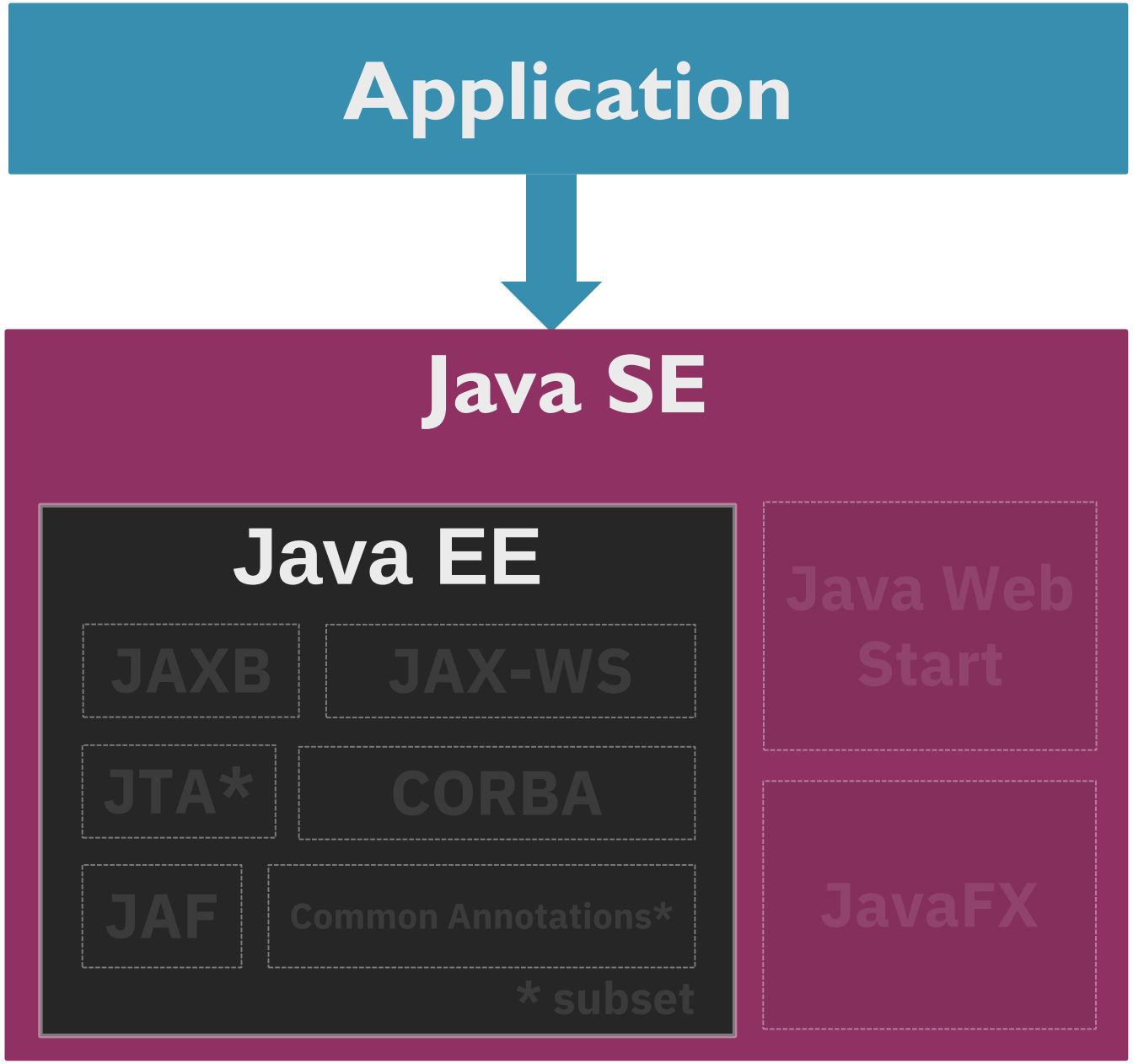
Top Migration Issues

- Missing Libraries
- Removed APIS
- Out-of-date Dependencies

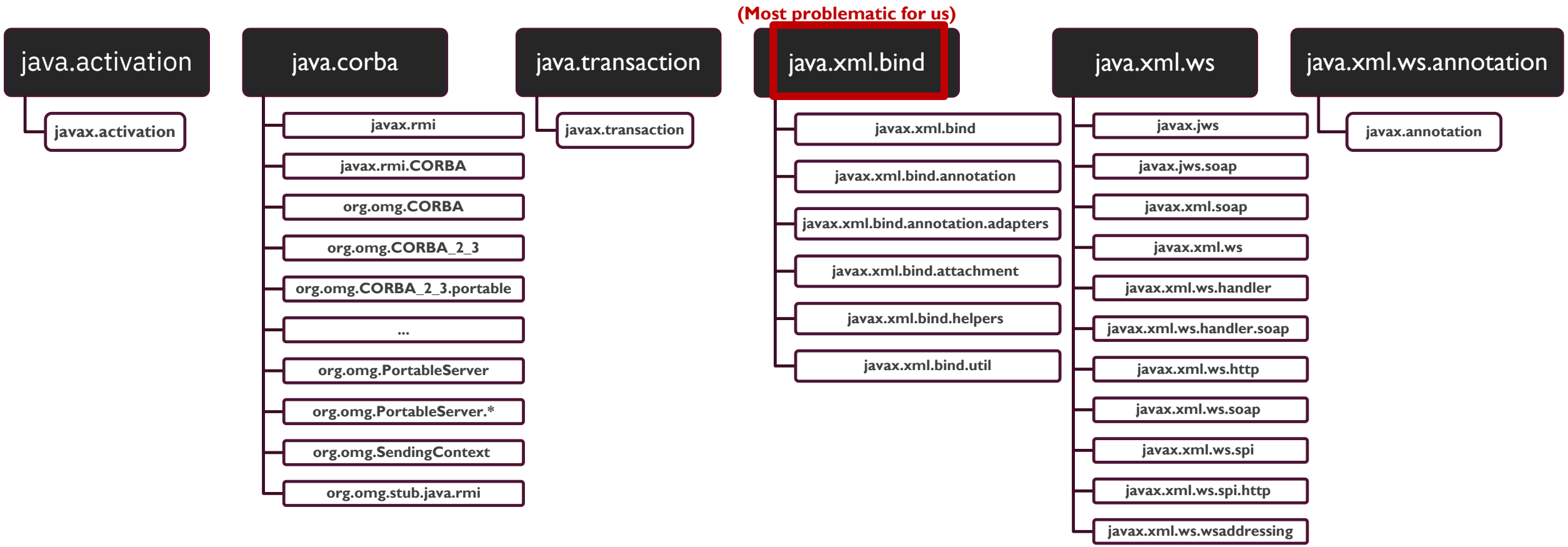
JAVA 8


Application
Java SE
Java EE
JAXB
JAX-WS
JTA*
CORBA
JAF
Common Annotations*
* subset
Java Web Start
JavaFX

JAVA 11+


Application
Java SE
Java EE
JAXB
JAX-WS
JTA*
CORBA
JAF
Common Annotations*
* subset
Java Web Start
JavaFX

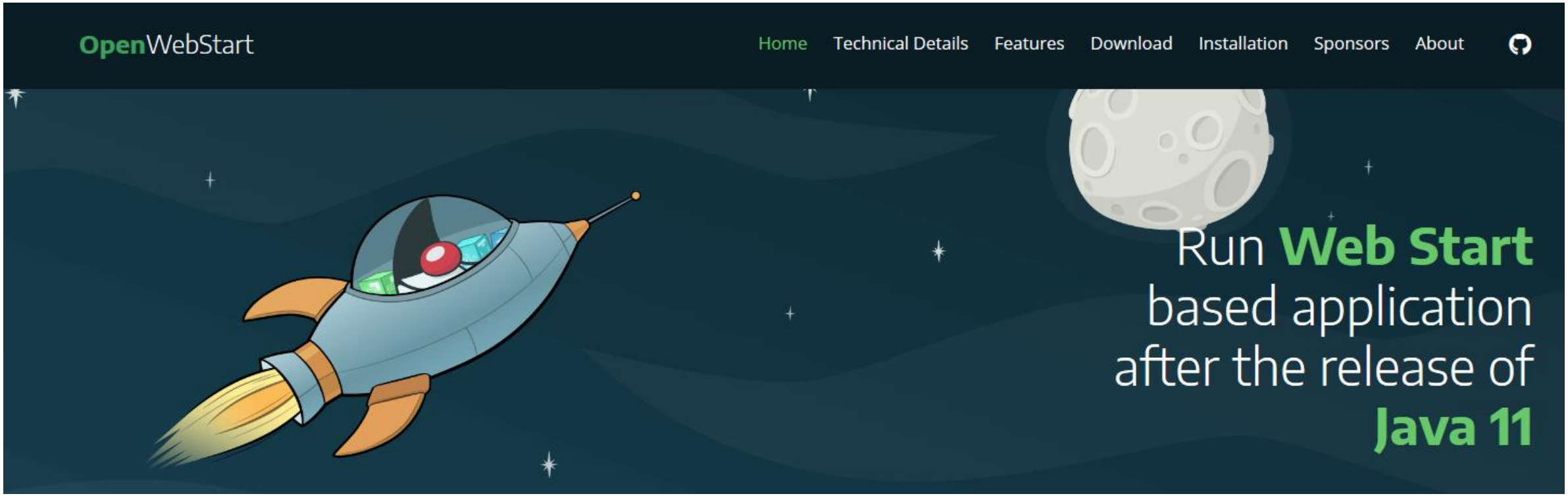
Java 11+ (Java EE)



Option 1: package your own dependencies
Option 2: rely on the app server to provide them (OpenLiberty with Java 11+)

Java 11+: Java EE

(Checkout Jakarta.ee)



Java 11+: Java Web Start

(Checkout OpenWebStart.com)

Java 11+: JavaFX

(Checkout Openjfx.io)

Top Migration Issues

- Missing Libraries
- Removed APIS
- Out-of-date Dependencies

JAVA 11+

Removed **Packages/Classes**

com.sun.awt.AWTUtilities
com.sun.image.codec.jpeg.*
com.sun.java.browser.plugin2.DOM
com.sun.security.auth.callback.DialogCallbackHandler
com.sun.security.auth.module.SolarisLoginModule
com.sun.security.auth.module.SolarisSystem
com.sun.security.auth.PolicyFile
com.sun.security.auth.SolarisNumericGroupPrincipal
com.sun.security.auth.SolarisNumericUserPrincipal
com.sun.security.auth.SolarisPrincipal
com.sun.security.auth.X500Principal

Java 8

```
com.sun.security.auth.SolarisNumericGroupPrincipal principal
```

Java 11

```
com.sun.security.auth.SolarisNumericGroupPrincipal principal
```

```
com.sun.security.auth.UnixNumericGroupPrincipal principal
```

JAVA 11+

Removed **Packages/Classes**

com.sun.xml.internal.bind.*
java.awt.dnd.peer.*
java.awt.peer.*
javax.security.auth.Policy
sun.misc.BASE64Decoder
sun.misc.BASE64Encoder
sun.misc.Unsafe.defineClass
sun.plugin.dom.DOMObject

Java 8

```
if( button.getPeer() != null) {
```

Java 11

```
if( button.getPeer() != null) {
```

```
if(button.isDisplayable()) {
```

JAVA 11+

Removed **Methods**

java.lang.Runtime.getLocalizedInputStream(..)
java.lang.Runtime.getLocalizedOutputStream(..)
java.lang.Runtime.runFinalizersOnExit(..)
java.lang.SecurityManager.checkAwtEventQueueAccess()
java.lang.SecurityManager.checkMemberAccess(..)
java.lang.SecurityManager.checkSystemClipboardAccess()
java.lang.SecurityManager.checkTopLevelWindow(..)
java.lang.SecurityManager.classDepth(..)
java.lang.SecurityManager.classLoaderDepth()
java.lang.SecurityManager.currentClassLoader()
java.lang.SecurityManager.currentLoadedClass()
java.lang.SecurityManager.getInCheck()
java.lang.SecurityManager.inClass(..)
java.lang.SecurityManager.inClassLoader()

Java 8

```
securityManager.checkAwtEventQueueAccess();
```

Java 11

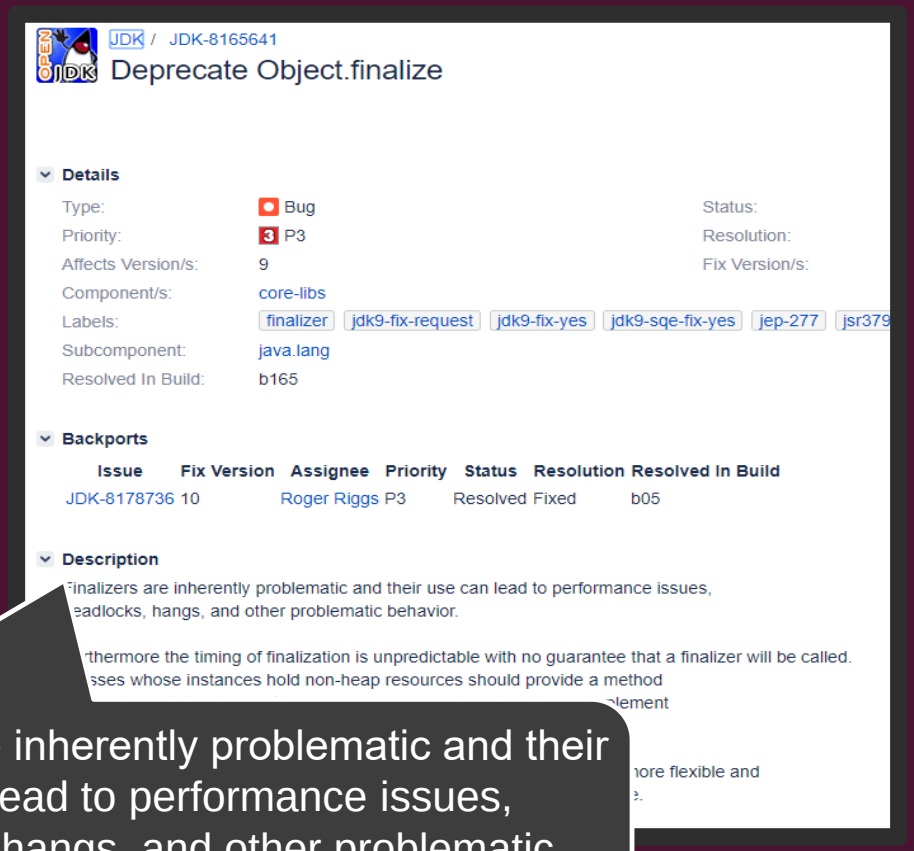
```
securityManager.checkAwtEventQueueAccess();
```

```
securityManager.checkPermission(permission);
```

JAVA 11+

Removed **Methods**

java.lang.System.runFinalizersOnExit(..)
java.lang.Thread.destroy()
java.lang.Thread.stop(java.lang.Throwable)
java.util.jar.Pack200.Packer.addPropertyChangeListener(..)
java.util.jar.Pack200.Packer.removePropertyChangeListener(..)
java.util.jar.Pack200.Unpacker.addPropertyChangeListener(..)
java.util.jar.Pack200.Unpacker.removePropertyChangeListener(..)
java.util.logging.LogManager.addPropertyChangeListener(..)
java.util.logging.LogManager.removePropertyChangeListener(..)



"Finalizers are inherently problematic and their use can lead to performance issues, deadlocks, hangs, and other problematic behavior."

JAVA 14 (NON-LTS)

Removed **Packages/Classes**

- com.sun.awt.SecurityWarning
- java.security.acl.*
- java.util.jar.Pack200

Removed **Methods**

- java.io.FileInputStream.finalize()
- java.io.FileOutputStream.finalize()
- java.lang.Runtime.traceInstructions(boolean)
- java.lang.Runtime.traceMethodCalls(boolean)
- java.util.zip.ZipFile.finalize()
- java.util.zip.Inflater.finalize()
- java.util.zip.Deflater.finalize()

Top Migration Issues

- Missing Libraries
- Removed APIS
- Out-of-date Dependencies

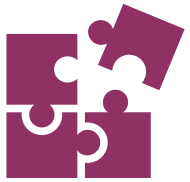
OUT-OF-DATE DEPENDENCIES

- Libraries (ASM, Mockito, Spring, etc)
- Build Tools (Gradle, Maven, etc)
- IDE (Eclipse, IntelliJ, VS Code, etc)
- Application servers (OpenLiberty, JBoss, WebLogic, etc)
- **My advice:** take this time to upgrade your dependencies to the latest supported version. Setup automation tools.

MIGRATING BEYOND JAVA 8

WHAT ABOUT MODULARITY?

MODULARITY



Java Platform Module System (JPMS)

Breaks code into modules containing packages

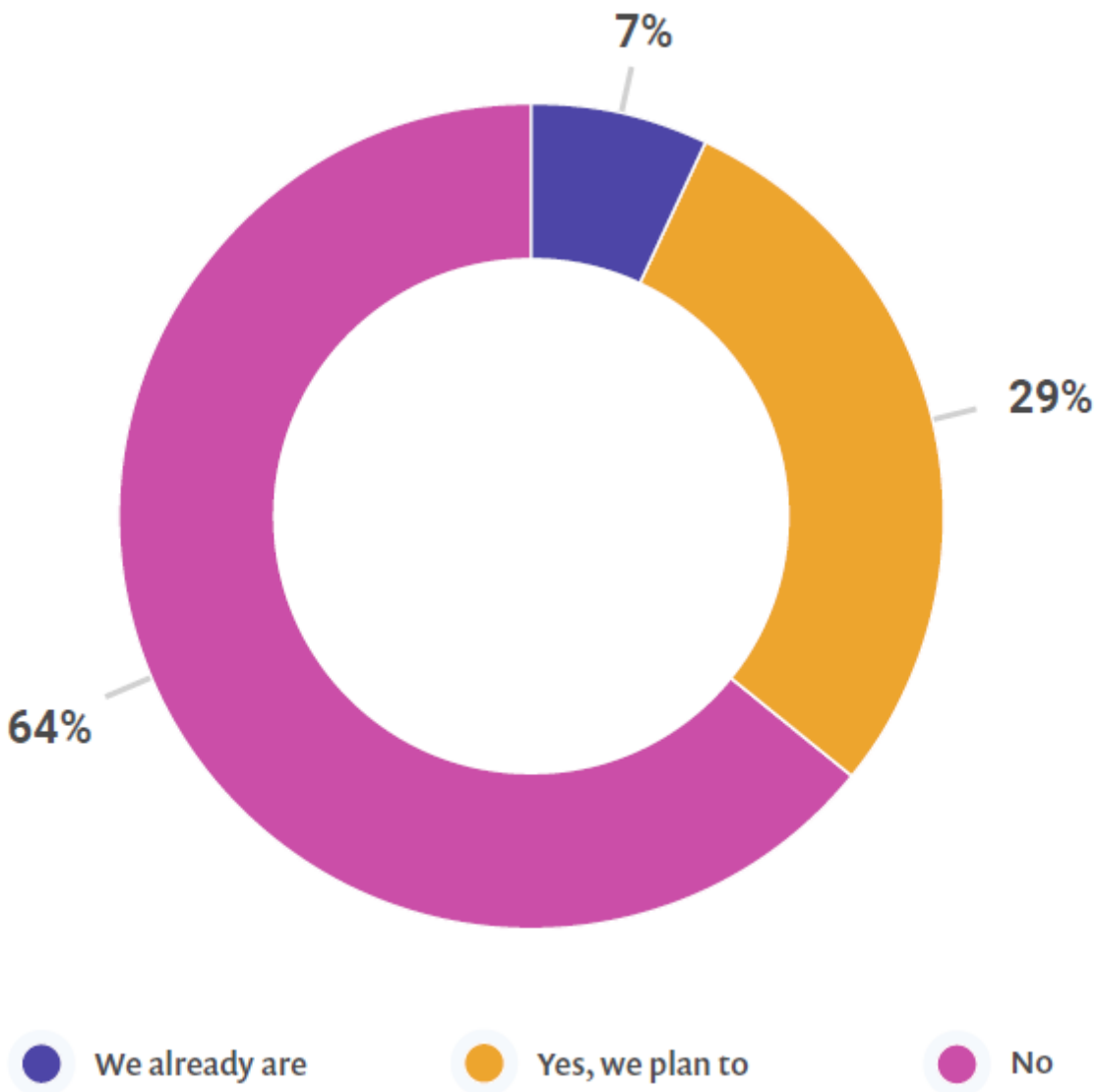
Applications must declare dependency on modules for access

Disruptive for Java SE application migrations

“Kill switch” on by default (currently)

--illegal-access=permit

My Advice: when first migrating to Java 11+, keep the “kill switch” on



ARE YOU USING, OR ARE YOU PLANNING TO USE, JAVA MODULES IN YOUR JAVA APPLICATIONS?

Source: JVM Ecosystem Report 2020
https://snyk.io/blog/jvm-ecosystem-report-2020/

MIGRATING BEYOND JAVA 8

WHAT TOOLS CAN I USE TO MAKE MY MIGRATION EASIER?

TOOLING

Application Binary Scanner Tool

- Download: http://ibm.biz/WAMT4AppBinaries
- Command line tool (free personal and commercial use)
- Scans an application binary for migration issues and produces a report with identified problems, solutions and resource links
- Documentation

JDeps

- Command utility shipped with JDK (run with the target Java version – Java 11+)
- Migration Relevant Option: -jdkinternals
- Binary scanner will recommend running JDeps if it identifies internal APIs
- Documentation

Tools to automate dependency updates (various options)

- Use an auto-upgrade dependency tool to keep your dependencies up-to-date
- Configure the tools to update based on various settings (frequency, etc)
- Example: dependabot

DEMO

NEXT STEPS

Step 1

Try running your app on Java 11+. If your app runs successfully, you can skip the next two steps.

Step 2

Debug your Java 11+ migration issues or run the application binary scanner tool.

java -jar binaryAppScanner.jar C:\Apps\MyApplication.jar --analyzeJavaSE --sourceJava=oracle8 --targetJava=java11

(Help command: java -jar binaryAppScanner.jar --help --analyzeJavaSE)

Step 3

Fix your application code or build files. Repeat steps 1-3.

Step 4

Update your dependencies and tools as needed.

Step 5

Run all your application tests.

Step 6

Celebrate!

Let me know if the tools helped….or didn't! (@DaliaShea)

HELPFUL LINKS

- My Blog on adding the Java 11+ feature to the migration tools (written instructions on running binary scanner): https://developer.ibm.com/tutorials/migration-to-java-11-made-easy/
- Binary Scanner: https://developer.ibm.com/wasdev/downloads/#asset/tools-Migration_Toolkit_for_Application_Binaries
  - Tool documentation and videos - see the « additional information » section
- Eclipse IDE Plugin Source Scanner: https://developer.ibm.com/wasdev/downloads/#asset/tools-WebSphere_Application_Server_Migration_Toolkit
  - Tool documentation and videos - see the « additional information » section
  - Video of how to migrate to Java 11 using the Eclipse IDE source scanner: https://youtu.be/m-l9eu4AAq4
- Link to the java11demo GitHub repo: https://github.com/daliasheasha/java11demo
- JVM Ecosystem Report 2020: https://snyk.io/blog/jvm-ecosystem-report-2020/
- Oracle Java SE Support Roadmap: https://www.oracle.com/technetwork/java/java-se-support-roadmap.html
- Migrating from Oracle Java to AdoptOpenJDK: https://adoptopenjdk.net/MigratingtoAdoptOpenJDKfromOracleJava.pdf
- Andy Guibert's blog on OpenLiberty Java 11+ support: https://openliberty.io/blog/2019/02/06/java-11.html

QUESTIONS?

dalia@us.ibm.com
@DaliaShea