JDK 9: Mission Accomplished
What Next For Java?

Simon Ritter

Deputy CTO, Azul Systems

azul.com

@speakjava

JDK 9: Big And Small Changes

Process API Updates
HTTP 2 Client
Improve Contended Locking
Unified JVM Logging
Compiler Control
Variable Handles
Segmented Code Cache
Smart Java Compilation, Phase T
The Modular JDK
Modular Source Code
Elide Deprecation Warnings on I tatement
Resolve Lint and Doclint Warnin
Milling Project Coin
Remove GC Combinations Depre JDK 8
Tiered Attribution for javac
Process Import Statements Corr
Annotations Pipeline 2.0
Datagram Transport Layer Secur S)
Modular Run-Time Imag
Simplified Doclet API
jshell: The Java Shell (Rea nt Loop)
New Version-String Scheme
HTML5 Javadoc
Javadoc Search
UTF-8 Property Files
Unicode 7.0
Add More Diagnostic Commands
Create PKCS12 Keystores by Default
Remove Launch-Time JRE Version Selection

Improve Secure Application Performance
Generate Run-Time Compiler Tests Automatically
Test Class-File Attributes Generated by javac
Parser API for Nashorn
Linux/AArch64 Port
Multi-Release JAR Files
Remove the JVM TI hprof Agent
ve the jhat Tool
JVM Compil ace
ion-Layer Negotia nsion
Valid Comman ag Ar
Levera nstructi GHA KSA
Compil er Platt
Make G efault G
OCSP S or TLS
Store I Strings rch
Multi on Imag
Use ale Data lt
aFX UI Con SS APIs fo rization
t Strings
Merge Selected Xerces Fixes into JAX
BeanInfo Annotations
Update JavaFX/Media to Newer Version of GStreamer
HarfBuzz Font-Layout Engine
Stack-Walking API
Encapsulate Most Internal APIs
Module System
TIFF Image I/O
HiDPI Graphics on Windows and Linux

Platform Logging API and Service
Marlin Graphics Renderer
More Concurrency Updates
Unicode 8.0
XML Catalogs
Convenience Factory Methods for Collections
Reserved Stack Areas for Critical Sections
Unified
Platf eatures
DRB Securek mplementations
En Method Ha
Mo va Applicat aging
Dy king of Lar Defined Object Models
Enh
Addit us Objects in G1
Improve Test Failure T hooting
Indify String Concaten
HotSpot C++ Unit-Te ework
jlink The Java Link
Enal
New H System
Spin-Wait Hints
SHA-3 Hash Algorithms
Disable SHA-1 Certificates
Deprecate the Applet API
Filter Incoming Serialization Data
Implement Selected ECMAScript 6 Features in Nashorn
Linux/s390x Port

JDK 9

Java Platform Module System (JPMS)

- The core Java libraries are now a set of modules (JEP 220)
  - 97 modules: 28 Java SE, 8 JavaFX, 59 JDK, 2 Oracle
- Most internal APIs now encapsulated (JEP 260)
  - `sun.misc.Unsafe`
  - Some can be used with command line options
- Modular source code (JEP 201)
  - JDK source code re-organised to support modules

Migrating Applications to JPMS

- Initially, leave everything on the classpath
- Anything on the classpath is in the unnamed module
  - All packages are exported
  - The unnamed module depends on all modules
- Migrate to modules as required
  - Try automatic modules
  - Move existing jar files from classpath to modulepath

Reversing Encapsulation

- "The Big Kill Switch" to turn off encapsulation
  - `--illegal-access`
    - `permit`: Warning for first use of an encapsulated API
    - `warn`: Warning for every use of an encapsulated API
    - `debug`: Warning and stack trace for every use
    - `deny`: No access to encapsulated APIs

Reversing Encapsulation

- Allowing direct access to encapsulated APIs
  - `--add-exports`

```
--add-exports java.management/com.sun.jmx.remote.internal=mytest
--add-exports java.management/sun.management=ALL-UNNAMED
```

- Allowing reflective access to encapsulated APIs
  - `--add-opens`

```
--add-opens java.base/java.util=ALL-UNNAMED
```

Reversing Encapsulation

- Using the JAR file manifest

```
Add-Exports: java.base/sun.security.provider
```

JDK 9 And Compatibility

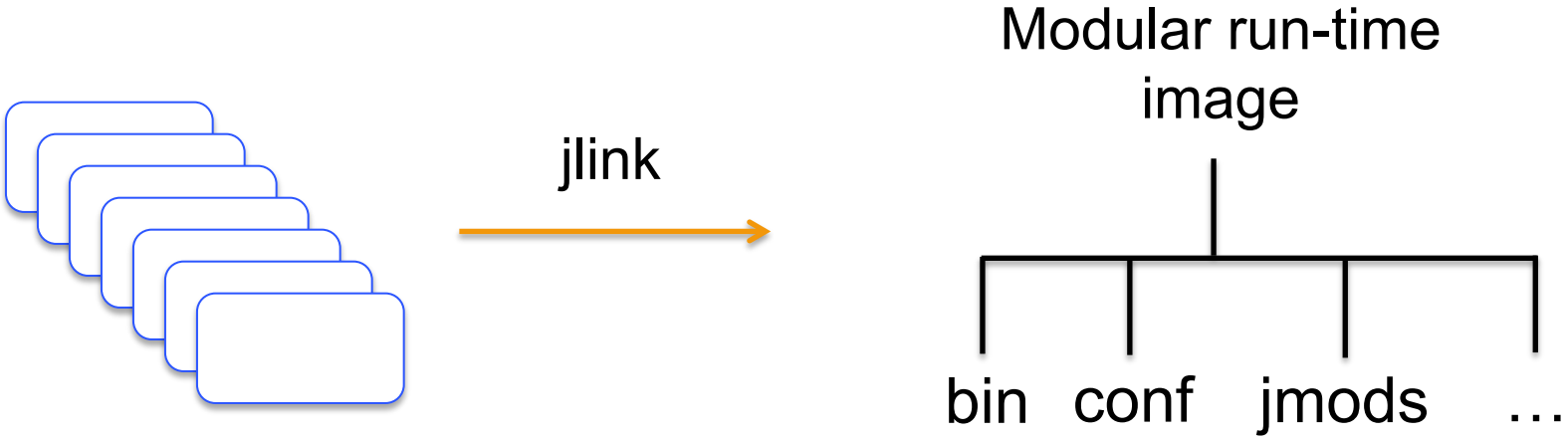
"Clean applications that just depend on java.se *should* just work" - *Oracle*

Finding Encapsulated API Use

- `jdeps`
  - Analyses dependencies on APIs
- Example: Minecraft

```
jdeps --list-deps 1.8.jar
   java.base
   java.datatransfer
   java.desktop
   java.management
   java.naming
   not found
   unnamed module: 1.8.jar
```

jlink: The Java Linker (JEP 282)




```
$ jlink --modulepath $JDKMODS \
  --addmods java.base –output myimage

$ myimage/bin/java –-list-modules
java.base@9
```

jlink: The Java Linker (JEP 282)

```
$ jlink --module-path $JDKMODS:$MYMODS \
  --addmods com.azul.app –-output myimage

$ myimage/bin/java –-list-modules
java.base@9
java.logging@9
java.sql@9
java.xml@9
com.azul.app@1.0
com.azul.zoop@1.0
com.azul.zeta@1.0
```

The Implications Of `jlink`

- "Write once, run anywhere"
  - Long term Java slogan, mainly held true
- `jlink` generated runtime may not include all Java SE modules
  - But is still a conformant Java implementation
  - To be conformant:
    - It must include the `java.base` module
    - If other modules are included, all transitive module dependencies must also be included
      - Defined as a closed implementation

"Missing" Modules

- Remember, "Clean applications that only use java.se..."
- The `java.se.ee` module not included by default
  - Compilation and runtime
- Affected modules
  - `java.corba`
  - `java.transaction`
  - `java.activation`
  - `java.xml.bind`
  - `java.xml.ws`
  - `java.xml.ws.annotation`

Using "Missing" Modules

- Use the command line option
  - `--add-modules java.corba`
- All modules (except CORBA) have standalone versions
  - Maven central
  - Relevant JSR RI
- Deploy standalone version on the upgrade module path
  - `--upgrade-module-path <path>`
- Deploy standalone version on the classpath

JDK Development Changes

OpenJDK: New Release Model

- A new version of the JDK will be released every six months
  - March and September
  - Starting this year
- OpenJDK development will be more agile
  - Previous target was a release every two years
    - Three and a half years between JDK 8 and JDK 9
- Features will be included only when ready
  - Targeted for a release when feature complete
  - Not targeted at specific release when started

OpenJDK: More Open Source

- Oracle will open-source closed-source parts of the JDK
  - Flight recorder
  - Mission control
- The goal is for there to be no functional difference between an Oracle binary and a binary built from OpenJDK source
  - Targeted for completion late 2018

JDK Version Numbering

- New scheme introduced in JDK 9 (JEP 223)
  - `JDK ${MAJOR}.${MINOR}.${SECURITY}`
  - Semantic versioning
  - Easier to understand by humans and computers

JDK Version Numbering

- New proposal for subsequent releases
  - JDK ${YEAR}.${MONTH}
    - JDK 18.3, JDK 18.9, etc.
  - Same concept as used by Ubuntu
  - Many people not happy about this

JDK Version Numbering

- New, new scheme. Just proposed JEP 322
  - $FEATURE.$INTERIM.$UPDATE.$PATCH

  - FEATURE: Was MAJOR, i.e. 10, 11, etc.
  - INTERIM: Was MINOR. Always zero, reserved for future use
  - UPDATE: Was SECURITY, but with different incrementing rule
  - PATCH: Emergency update outside planned schedule

Deprecated in JDK 9: Soon To Go

- Applets as a deployment mechanism
- CMS garbage collector
- Java policy tool, `jconsole`, Doclet API, other small things
- java.se.ee meta-module
  - `java.corba`
  - `java.transaction`
  - `java.activation`
  - `java.xml.bind`
  - `java.xml.ws`
  - `java.xml.ws.annotation`

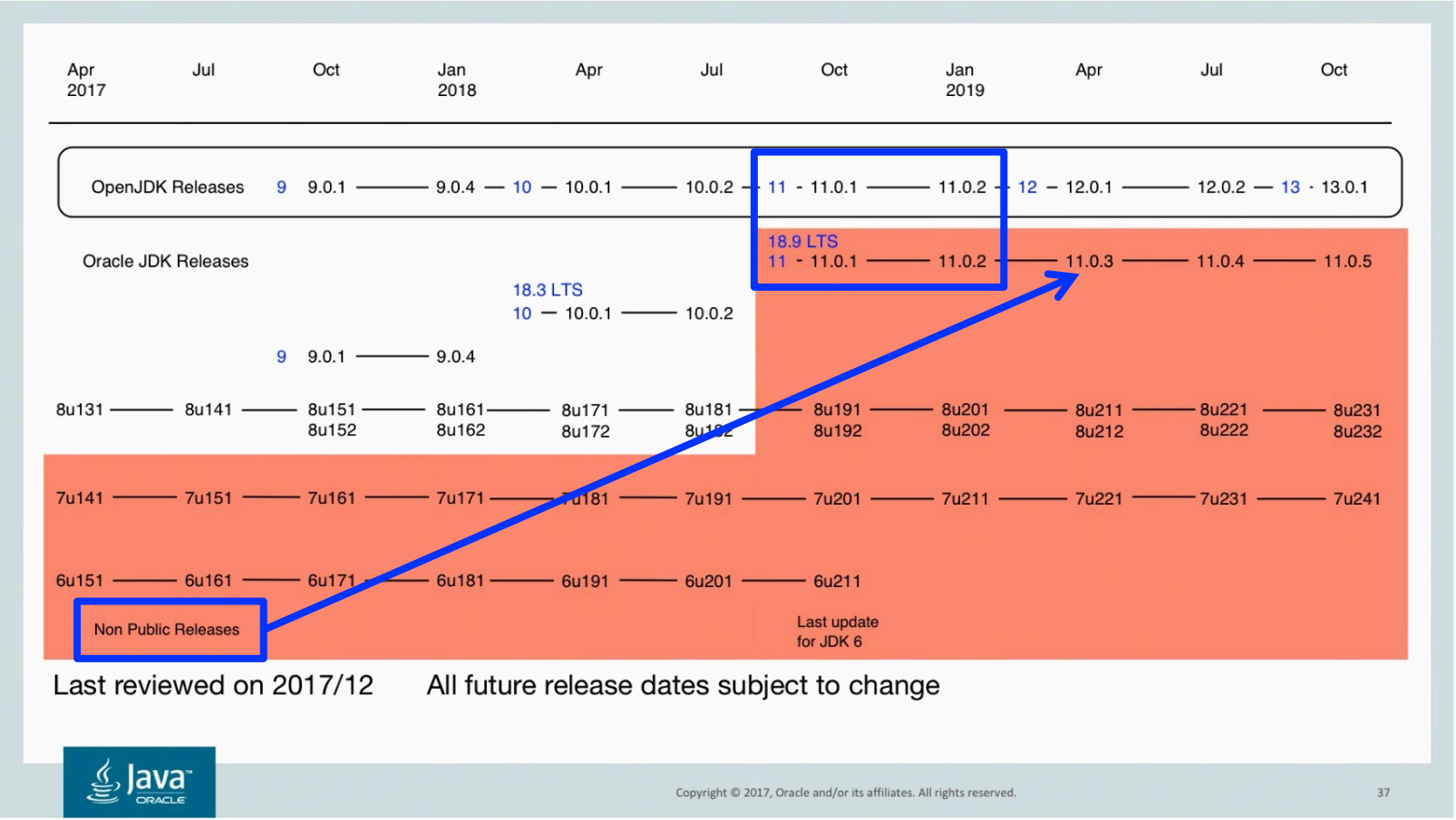
Availability Of JDK Updates

- Oracle is switching to a long term support (LTS) model
  - **ONLY** for customers of commercial support
- There will be 3 years between LTS releases
  - JDK 8 has been classified as an LTS release
    - It will have updates for more than 3 years
  - Next LTS release will be September, 2018 (JDK 11)
- Non-LTS releases are "Feature Releases"
  - JDK 9 is a feature release
- **No overlap of support**

Oracle Binaries

- Until now released under Oracle Binary Code License
  - Have to accept to download
  - Classic "field-of-use" restriction
- Moving forward
  - Binaries available under GPLv2 with CPE
  - No more 32-bit binaries
  - No more ARM binaries
  - Windows, Linux, Mac and Solaris SPARC only
    - All 64-bit

JDK 11: All Change



JDK.${NEXT}

JDK 10 (JSR 383)

- JSR 383 submitted and expert group formed
  - Lead by Brian Goetz
  - Oracle, IBM, Red Hat, SAP and Azul (Me)
- Schedule approved
  - General Availability: 20/3/18
- Release candidate now available

JDK 10

- JEP 286: Local variable type inference

```
var list = new ArrayList<String>(); // infers ArrayList<String>
var stream = list.stream();         // infers Stream<String>
```

- JEP 307: Parallel Full GC for G1
  - Still a full GC with potentially big pauses
- JEP 310: Application Class-Data Sharing
- JEP 317: Experimental Java-based JIT compiler (Graal)
- JEP 319: Root Certificates
- JEP 296: Consolidate JDK forests into single repo

JDK 10

- JEP 316: Heap allocation on alternative devices
  - NV-RAM with same semantics as DRAM
- JEP 313: Remove `javah` tool
  - Same functionality through `javac`
- JEP 304: Garbage Collector Interface (Internal JVM)
  - Easier to add new algorithms
- JEP 312: Thread-Local Handshakes

JDK 10

- 73 New APIs
  - List, Set, Map.copyOf(Collection)
  - Optional.orElseThrow()
  - Collectors
    - toUnmodifiableList
    - toUnmodifiableMap
    - toUnmodifiableSet

JDK 10

- Miscellaneous
  - XMLInputFactory.newFactory() de-deprecated
  - com.sun.security.auth package
    - Six deprecated classes removed
  - java.lang.SecurityManager
    - One deprecated field and seven methods removed
  - JVM now more Docker container aware
    - Uses container CPU count and memory size

Longer Term JDK Futures

JDK 11 (JSR 384)

- Expert group same as JSR 383 (and JSR 379)
- More of a rolling JSR
- Features proposed so far
  - Epsilon GC (JEP 318)
  - Dynamic class file constants (JEP 309)
  - Remove the Java EE and CORBA modules (JEP 320)
  - Local variable syntax for Lambda parameters (JEP 323)

OpenJDK Projects

- Amber
  - Simplifying syntax
- Loom
  - Continuations and fibres
- Metropolis
  - The JVM re-written in Java
- Panama
  - FFI replacement for JNI
- Valhalla
  - Value types and specialised generics

Project Amber: Pattern Matching

- JEP 305: Pattern matching
  - Type test and switch statement support to start

```
String formatted;
switch (obj) {
    case Integer i: formatted = String.format("int %d", i); break;
    case Byte b:    formatted = String.format("byte %d", b); break;
    case Long l:    formatted = String.format("long %d", l); break;
    case Double d:  formatted = String.format(“double %f", d); break;
    case String s:  formatted = String.format("String %s", s); break
    default:        formatted = obj.toString();
}
```

Project Valhalla

- Java has:
  - Primitives: for performance
  - Objects: for encapsulation, polymorphism, inheritance, OO
- Problem is where we want to use primitives but can't
  - `ArrayList<int>` won't work
  - `ArrayList<Integer>` requires boxing and unboxing, object creation, heap overhead, indirection reference

Project Valhalla

- Value types
- "Codes like a class, works like a primitive"
  - Can have methods and fields
  - Can implement interfaces
  - Can use encapsulation
  - Can be generic

Project Loom

- Further work on making concurrent programming simpler
- Threads are too heavyweight
  - Too much interaction with operating system
- Loom will introduce fibres
  - JVM level threads (remember green threads?)
  - Add continuations to the JVM
  - Use the `ForkJoinPool` scheduler
  - Much lighter weight than threads
    - Less memory
    - Close to zero overhead for task switching

Project Metropolis

- Run Java on Java
  - Rewrite most of the JVM in Java
- Use the Graal compiler project as significant input
- Easier ports to new platforms
  - Less native code to modify and compile
- Faster new features on front-end
  - Easier to write Java than C++
- Performance is an issue to be explored and resolved
  - AOT compiler in JDK 9 is the start of this

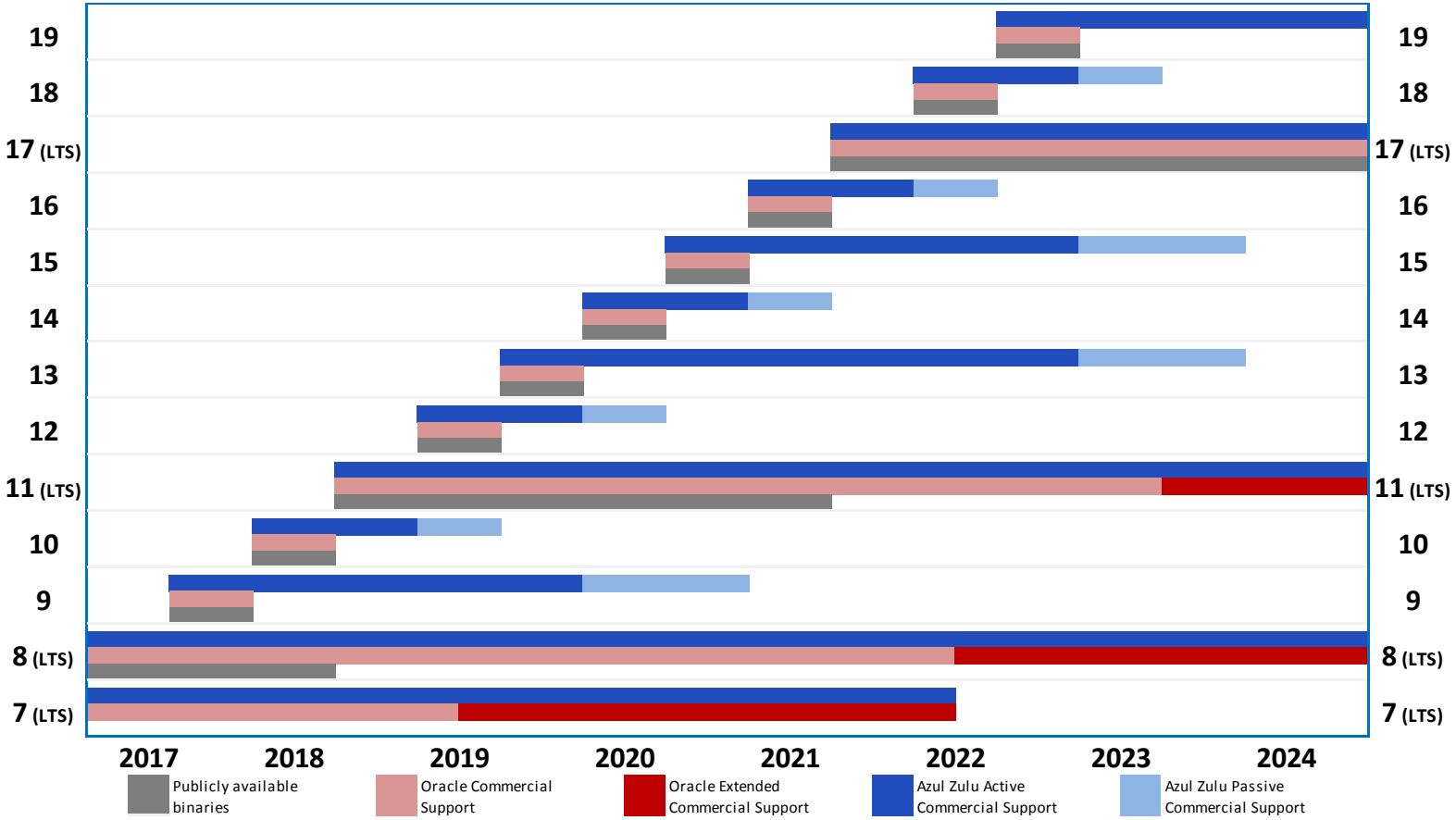
Azul's Zulu Java

Zulu Java

- Azul's binary distribution of OpenJDK
  - Passes all TCK tests
  - Multi-platform (Windows, Linux, Mac)
- JDK 6, 7, 8 and 9 available
- Wider platform support:
  - Intel 32-bit Windows and Linux
  - ARM 32 and 64-bit
  - PowerPC

www.zulu.org/download

Azul Support Timeline



Summary

Java Continues Moving Forward

- JDK 9 is out
  - But not an LTS
  - JPMS may require changes to existing applications
- Faster Java releases
  - Feature release every 6 months, LTS every 3 years
  - Support is a major consideration
- Lots of ideas to improve Java
  - Value types, better JNI, better type inference
  - Many smaller things

Thank you!

Simon Ritter

Deputy CTO, Azul Systems

azul.com

@speakjava