Preparing for JDK 9

Rory O'Donnell
(rory.odonnell@oracle.com)
Java Platform Group
Oracle
October 2016

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Background: JDK 9 Information

https://jdk9.java.net/

JDK 9 Information

- Current proposed schedule: GA July 2017, weekly updates:
  - Early access binaries + docs: https://jdk9.java.net/
  - Early access binaries with cutting edge Jigsaw: https://jdk9.java.net/jigsaw/
- OpenJDK:
  - Project: http://openjdk.java.net/projects/jdk9/
  - Mailing list: http://mail.openjdk.java.net/mailman/listinfo/jdk9-dev
  - Source code: http://hg.openjdk.java.net/jdk9/dev/
  - Adoption: http://mail.openjdk.java.net/pipermail/adoption-discuss/
    https://wiki.openjdk.java.net/display/quality/Quality+Outreach/
  - JEPs (JDK Enhancement Proposals) used for project tracking

Quality Outreach

- Background/History
  - JDK 7 GA - 28 July 2011
  - Java 7 paralyses Lucene and Solr (29 July)
- JDK 8
  - October 2013 started Quality Outreach
  - zero to ~20 Free Open Source Software Projects (FOSS) testing EA build prior to GA
  - High priority issues were detected and addressed in the run up to JDK 8 GA
    - Apache Lucene (5 bugs)
    - Apache Maven (4 bugs)
    - Groovy (3 bugs)
    - Scala (2 bugs)

Quality Outreach Program

- ~80 FOSS projects registered (as of Oct, 2016)
  - From Apache, Eclipse, Java EE, Spring and other open source communities
  - Build tools such as Apache Ant, Apache Maven, Gradle
  - JVM languages such as Clojure, Golo, Apache Groovy, Jython, Scala
  - Development tools such as Arquillian, Byteman, Checkstyle, EasyMock, Findbugs, JaCoCo, JUnit 5 ...
  - And many more such as Apache Lucene, Apache POI, Apache Tomcat, Hibernate, Jetty, JOSM, Spring Framework, WildFly
  - Should you be on the list ???
  - To join email quality-discuss@openjdk.java.net or rory.odonnell@oracle.com

What

- We send regular EA build availability emails out every ~2-3 weeks
  - Add information **on items** of interest
    - **Integrations** of interest
    - Highlight recent discussion on mailing lists
    - Request feedback on proposed changes
    - Test results for Early Access builds
    - **Assist with bug filling and updating**
- When a fix is available
  - b130 – Hibernate (1),Tomcat (2), JOSM (1)
  - b131 – PMD(1) confirmed fix, no regressions
- We have added 20 Open Source Projects in the last 6 months
- https://wiki.openjdk.java.net/display/quality/Quality+Outreach

Total FOSS logged bugs by Priority

Two Dimensional Filter Statistics by Resolution

Priority	Unresolved	Fixed	Won't Fix	Duplicate	Incomplete	Cannot Reproduce	Not an Issue	T:
1 P1	0	11	0	0	0	0	0	11
2 P2	0	29	0	6	0	3	2	40
3 P3	11	38	2	13	1	5	5	75
4 P4	8	7	0	1	2	3	2	23
5 P5	0	1	0	0	0	0	0	1
Total Unique Issues:	**19**	**86**	**2**	**20**	**3**	**11**	**9**	**150**

FOSS logged bugs by Component

Two Dimensional Filter Statistics by Priority

Components	1 P1	2 P2	3 P3	4 P4	5 P5	T:
client-libs	0	2	11	4	0	17
core-libs	2	8	21	7	0	38
core-svc	1	0	3	0	0	4
deploy	0	0	2	2	0	4
hotspot	5	20	17	4	0	46
infrastructure	1	0	1	0	0	2
javafx	0	1	0	1	1	3
security-libs	2	0	1	3	0	6
tools	0	8	15	2	0	25
xml	0	1	4	0	0	5
Total Unique Issues:	**11**	**40**	**75**	**23**	**1**	**150**

Not just Bugs – join in the conversation - Recent Changes

- Build 135 included a fix which has introduced a behavioral change to HttpURLConnection
  - The behavior of HttpURLConnection when using a ProxySelector has been modified with this JDK release.
- FilePermission change in jdk-9+140 - If you use a security manager and file permissions then please provide feedback.
  - This code change removes pathname canonicalization from FilePermission creation, thus calculations of the equals() and implies() methods will be based on the raw path string one provides in "new FilePermission(path, action)".

[1] http://mail.openjdk.java.net/pipermail/jigsaw-dev/2016-October/009550.html

Recent proposals

- More proposals for open **Java Platform Module System** issues [1]
  - One proposal to call out is #AwkwardStrongEncapsulation as this proposes to change setAccessible(true) so that it can't be used to break into non-public types/members in exported packages.
- Proposal to Reorganize source classes in src.zip by modules [2]
  - This proposal might have a compatibility impact on IDEs or other tools that look in src.zip
  - Our proposal is to change the layout of src.zip in JDK 9 to $MODULE/$PACKAGE/*.java i.e. the source files are grouped by modules.

[1] http://mail.openjdk.java.net/pipermail/jigsaw-dev/2016-September/009365.html

[2] http://mail.openjdk.java.net/pipermail/jigsaw-dev/2016-October/009550.html

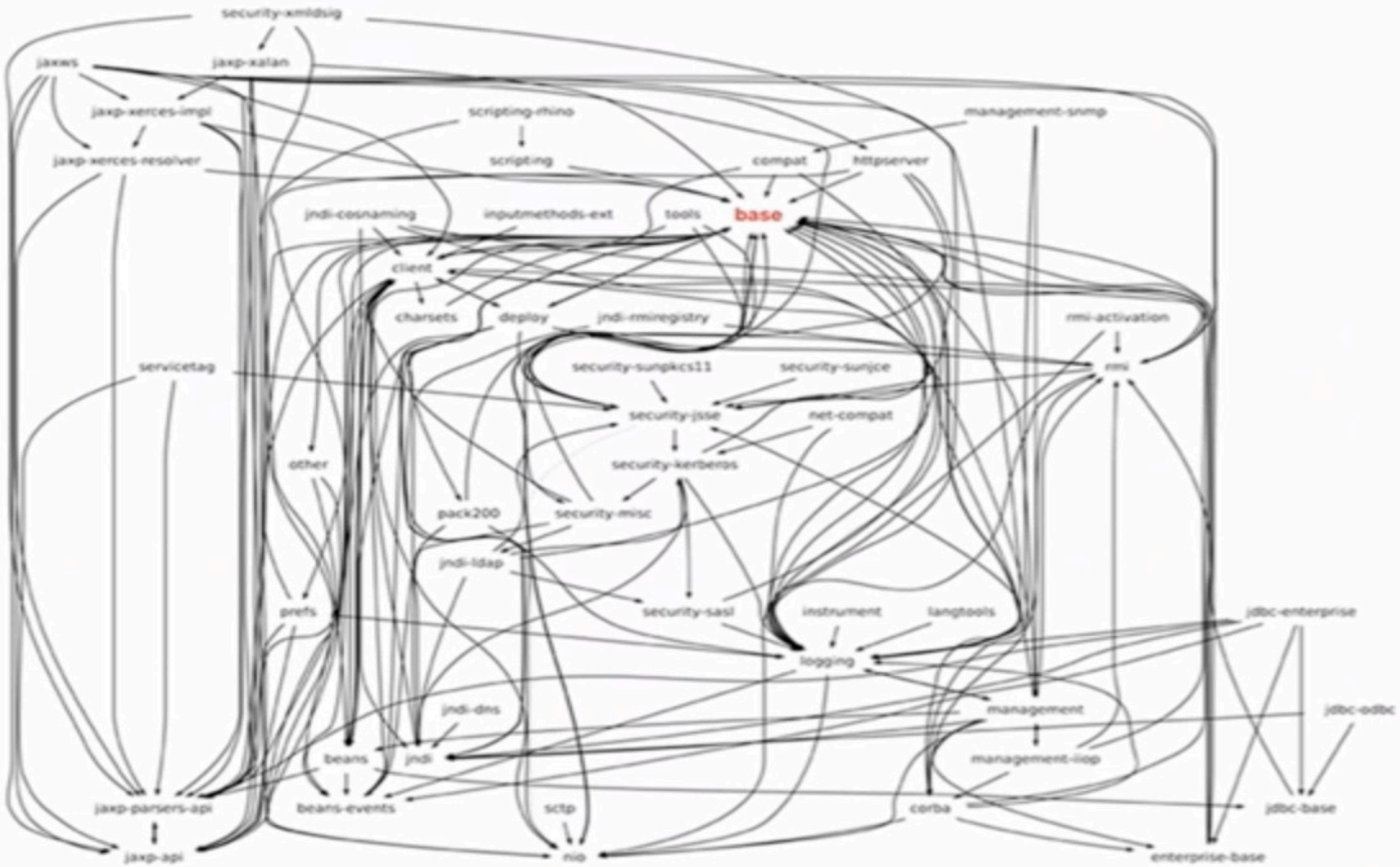
Process API Updates
HTTP 2 Client
Improve Contended Locking
Unified JVM Logging
Compiler Control
Variable Handles
Segmented Code Cache
Smart Java Compilation, Phase Two
The Modular JDK
Modular Source Code
Elide Deprecation Warnings on Import Statements
Resolve Lint and Doclint Warnings
Milling Project Coin
Remove GC Combinations Deprecated in JDK 8
Tiered Attribution for javac
Process Import Statements Correctly
Annotations Pipeline 2.0
Datagram Transport Layer Security (DTLS)
Modular Run-Time Images
Simplified Doclet API
jshell: The Java Shell (Read-Eval-Print 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

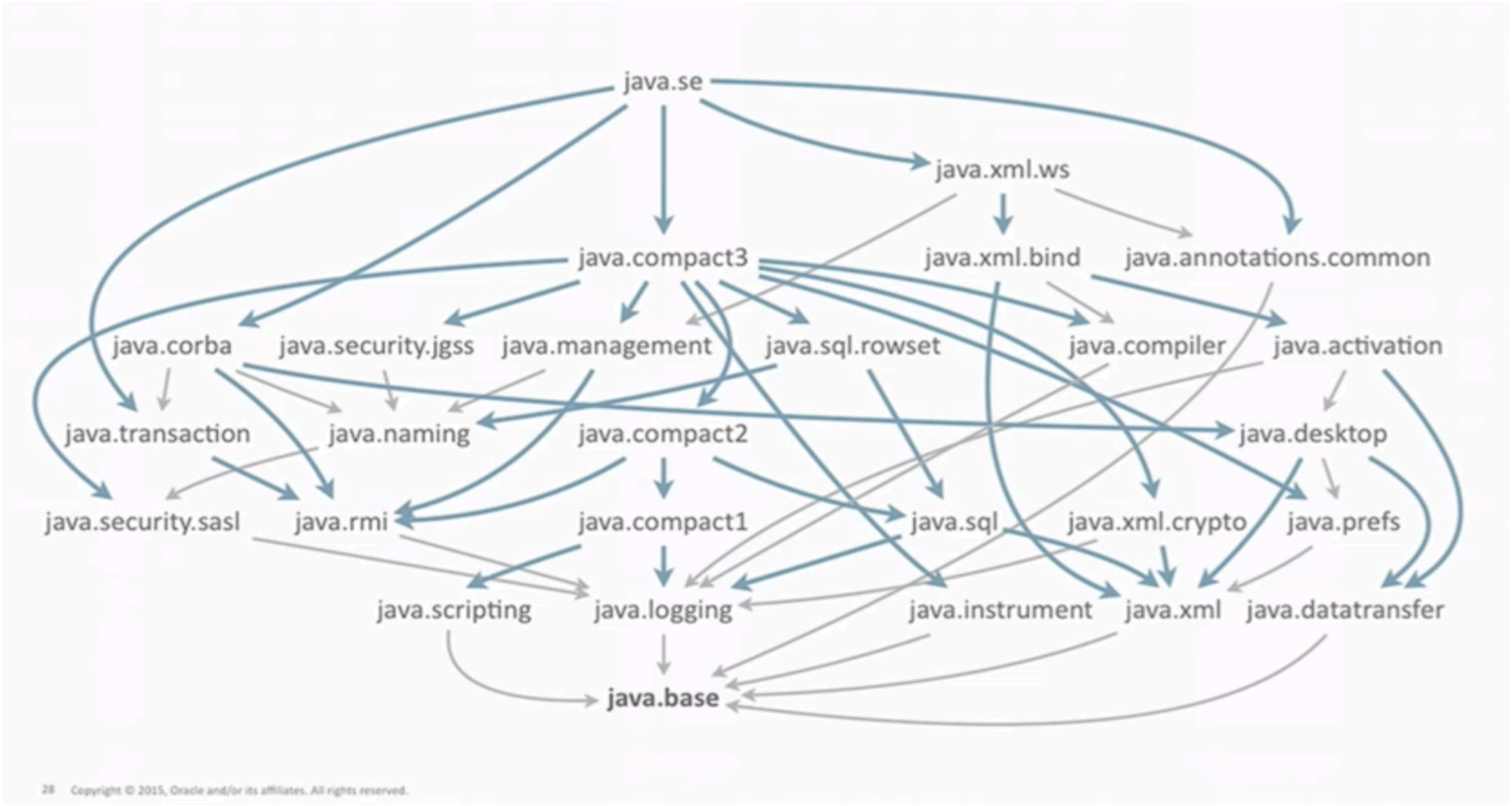
Generate Run-Time Compiler Tests Automatically
Test Class-File Attributes Generated by javac
Parser API for Nashorn
Linux/AArch64 Port
Multi-Release JAR Files
Remove the JVM TI hprof Agent
Remove the jhat Tool
Java-Level JVM Compiler Interface
TLS Application-Layer Protocol Negotiation Extension
Validate JVM Command-Line Flag Arguments
Leverage CPU Instructions for GHASH and RSA
Compile for Older Platform Versions
Make G1 the Default Garbage Collector
OCSP Stapling for TLS
Store Interned Strings in CDS Archives
Multi-Resolution Images
Use CLDR Locale Data by Default
Prepare JavaFX UI Controls & CSS APIs for Modularization
Compact Strings
Merge Selected Xerces 2.11.0 Fixes into JAXP
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 GC Logging
Platform-Specific Desktop Features
DRBG-Based SecureRandom Implementations
Enhanced Method Handles
Modular Java Application Packaging
Dynamic Linking of Language-Defined Object Models
Enhanced Deprecation
Additional Tests for Humongous Objects in G1
Improve Test-Failure Troubleshooting
Indify String Concatenation
HotSpot C++ Unit-Test Framework
jlink: The Java Linker
Enable GTK 3 on Linux
New HotSpot Build 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

Background : JDK 9 and Project Jigsaw goals

- Make Java SE more flexible and scalable
- Improve security and maintainability
  - Encapsulate within module boundaries
- Make it easier to construct, maintain, deploy and upgrade large applications
- Enable improved performance

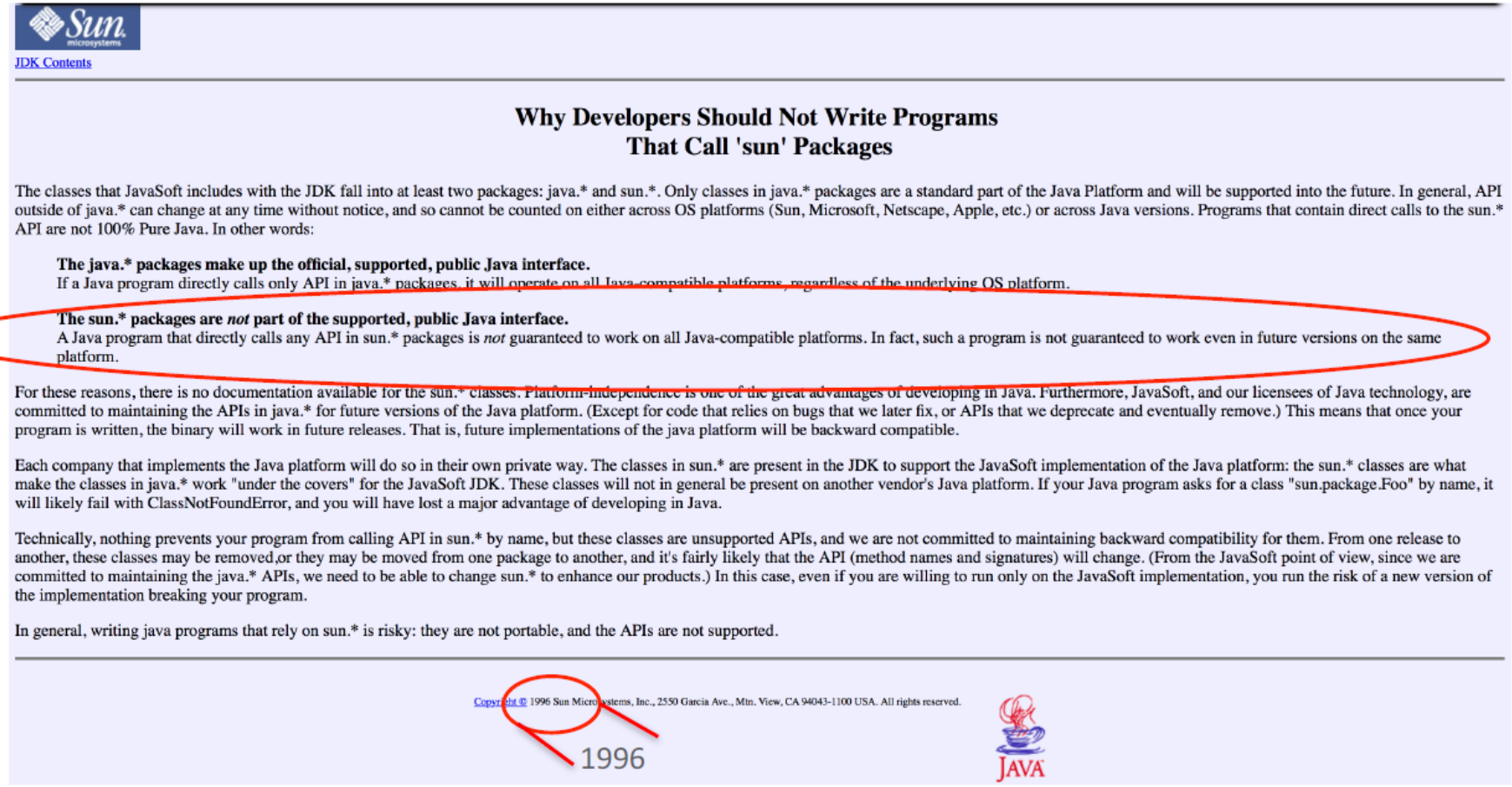




Background: Categories of APIs in the JDK

- Supported, intended for external use
  - JCP standard, java.*, javax.*
  - JDK-specific API, some com.sun.*, some jdk.*
- Unsupported, JDK-internal, not intended for external use
  - sun.* mostly

Internal API Related Changes



JDK Contents

Why Developers Should Not Write Programs That Call 'sun' Packages

The classes that JavaSoft includes with the JDK fall into at least two packages: java.* and sun.*. Only classes in java.* packages are a standard part of the Java Platform and will be supported into the future. In general, API outside of java.* can change at any time without notice, and so cannot be counted on either across OS platforms (Sun, Microsoft, Netscape, Apple, etc.) or across Java versions. Programs that contain direct calls to the sun.* API are not 100% Pure Java. In other words:

The java.* packages make up the official, supported, public Java interface.
If a Java program directly calls only API in java.* packages, it will operate on all Java-compatible platforms, regardless of the underlying OS platform.

The sun.* packages are *not* part of the supported, public Java interface.
A Java program that directly calls any API in sun.* packages is *not* guaranteed to work on all Java-compatible platforms. In fact, such a program is not guaranteed to work even in future versions on the same platform.

For these reasons, there is no documentation available for the sun.* classes. Platform-independence is one of the great advantages of developing in Java. Furthermore, JavaSoft, and our licensees of Java technology, are committed to maintaining the APIs in java.* for future versions of the Java platform. (Except for code that relies on bugs that we later fix, or APIs that we deprecate and eventually remove.) This means that once your program is written, the binary will work in future releases. That is, future implementations of the java platform will be backward compatible.

Each company that implements the Java platform will do so in their own private way. The classes in sun.* are present in the JDK to support the JavaSoft implementation of the Java platform: the sun.* classes are what make the classes in java.* work "under the covers" for the JavaSoft JDK. These classes will not in general be present on another vendor's Java platform. If your Java program asks for a class "sun.package.Foo" by name, it will likely fail with ClassNotFoundError, and you will have lost a major advantage of developing in Java.

Technically, nothing prevents your program from calling API in sun.* by name, but these classes are unsupported APIs, and we are not committed to maintaining backward compatibility for them. From one release to another, these classes may be removed,or they may be moved from one package to another, and it's fairly likely that the API (method names and signatures) will change. (From the JavaSoft point of view, since we are committed to maintaining the java.* APIs, we need to be able to change sun.* to enhance our products.) In this case, even if you are willing to run only on the JavaSoft implementation, you run the risk of a new version of the implementation breaking your program.

In general, writing java programs that rely on sun.* is risky: they are not portable, and the APIs are not supported.

Copyright © 1996 Sun Microsystems, Inc., 2550 Garcia Ave., Mtn. View, CA 94043-1100 USA. All rights reserved.

1996

JAVA

General compatibility policies

- If an application uses only supported APIs and works on release N then it should work on N+1, even without recompilation
- Supported APIs can be removed but only with advance notice

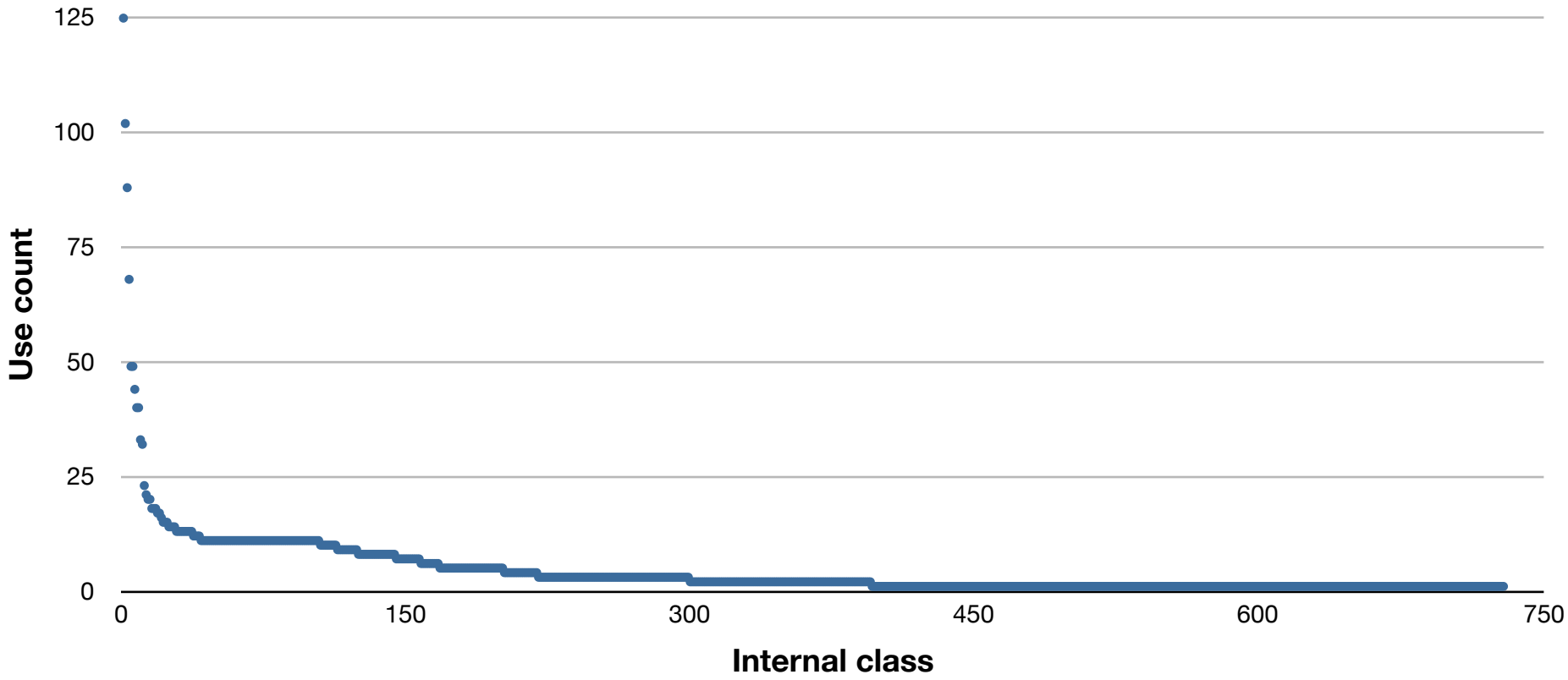
Managing incompatibilities

- Judge risk and impact based on actual data (where possible)
- Communicate early and vigorously
- Make it easy to understand how existing code will be affected
- When removing unsupported APIs, provide replacements where it makes sense
- Provide workarounds where possible

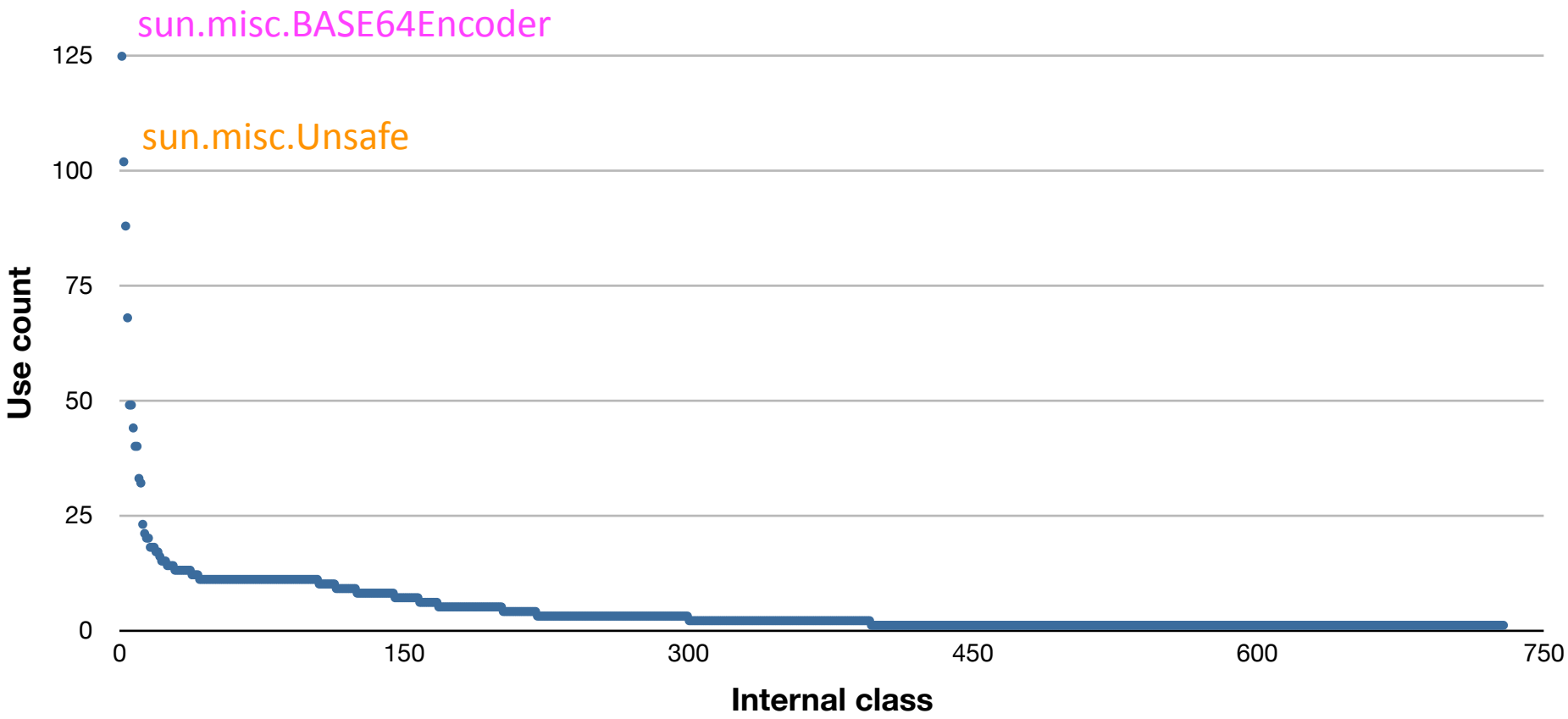
Incompatible changes in JDK 9

- Encapsulate most JDK-internal APIs
- Change the binary structure of the JRE and JDK
- Remove a small number of supported , but deprecated , JCP-standard APIs
- Removed the endorsed-standards override and extension mechanisms
- New version-string format

Uses of JDK-internal APIs



Uses of JDK-internal APIs



Categories of JDK-internal APIs

- Non-critical
  - No evidence of use outside of JDK
  - or used only for convenience
- Critical
  - Functionality that would be difficult, if not impossible, to implement outside of the JDK

JEP 260

- Encapsulate all non-critical internal APIs by default
- Encapsulate all critical internal APIs for which supported replacements exist in JDK 8
- Do not encapsulate critical internal APIs
  - Deprecate them in JDK 9
  - Plan to remove in JDK 10
  - Provide a workaround via command-line flag

JEP 260

- Propose as critical internal APIs
  - sun.misc.Unsafe
  - sun.misc.{Signal,SignalHandler}
  - sun.reflect.Reflection::getCallerClass
  - sun.reflect.ReflectionFactory

Finding uses of JDK-internal APIs

- jdeps tool in JDK 8, improved in JDK 9
- Maven JDeps Plugin

```
$ jdeps −jdkinternals glassfish/modules/security.jar
```

```
$ jdeps -jdkinternals glassfish/modules/security.jar
security.jar -> java.base
   com.sun.enterprise.common.iiop.security.GSSUPName (security.jar)
      -> sun.security.util.ObjectIdentifier                JDK internal API (java.base)
   com.sun.enterprise.common.iiop.security.GSSUtilsContract (security.jar)
      -> sun.security.util.ObjectIdentifier                JDK internal API (java.base)
   com.sun.enterprise.security.auth.login.LoginContextDriver (security.jar)
      -> sun.security.x509.X500Name                        JDK internal API (java.base)
   com.sun.enterprise.security.auth.login.LoginContextDriver$4 (security.jar)
      -> sun.security.x509.X500Name                        JDK internal API (java.base)
   com.sun.enterprise.security.auth.realm.certificate.CertificateRealm (security.jar)
      -> sun.security.x509.X500Name                        JDK internal API (java.base)
   com.sun.enterprise.security.auth.realm.ldap.LDAPRealm (security.jar)
      -> sun.security.x509.X500Name                        JDK internal API (java.base)
   com.sun.enterprise.security.ssl.JarSigner (security.jar)
      -> sun.security.pkcs.ContentInfo                     JDK internal API (java.base)
      -> sun.security.pkcs.PKCS7                           JDK internal API (java.base)
      -> sun.security.pkcs.SignerInfo                      JDK internal API (java.base)
      -> sun.security.x509.AlgorithmId                     JDK internal API (java.base)
      -> sun.security.x509.X500Name                        JDK internal API (java.base)

Warning: JDK internal APIs are unsupported and private to JDK implementation that are
subject to be removed or changed incompatibly and could break your application.
Please modify your code to eliminate dependency on any JDK internal APIs.
For the most recent update on JDK internal API replacements, please check:
https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool

JDK Internal API                        Suggested Replacement
----------------                        ---------------------
sun.security.x509.X500Name              Use javax.security.auth.x500.X500Principal @since 1.4
```

Example: Glassfish 4.1

java.lang.IllegalAccessError: class com.sun.enterprise.security.provider.PolicyWrapper (in unnamed module @0x7cdbc5d3) cannot access class sun.security.provider.PolicyFile (in module java.base), because module java.base does not export sun.security.provider to unnamed module @0x7cdbc5d3

```
com.sun.enterprise.security.provider.PolicyWrapper.getNewPolicy(PolicyWrapper.java:75)
```
at com.sun.enterprise.security.provider.BasePolicyWrapper.<init>(BasePolicyWrapper.java:148)
at com.sun.enterprise.security.provider.PolicyWrapper.<init>(PolicyWrapper.java:67)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(java.base@9.0/Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance(java.base@9.0/NativeConstructorAccessorImpl.java:62)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(java.base@9.0/DelegatingConstructorAccessorImpl.java:45)
at java.lang.reflect.Constructor.newInstance(java.base@9.0/Constructor.java:443)
at java.lang.Class.newInstance(java.base@9.0/Class.java:525)
at com.sun.enterprise.security.PolicyLoader.loadPolicy(PolicyLoader.java:155)
at com.sun.enterprise.security.SecurityLifecycle.onInitialization(SecurityLifecycle.java:163)
at com.sun.enterprise.security.SecurityLifecycle.postConstruct(SecurityLifecycle.java:208)
:

Example: Gradle 2.7

```
:compileJava FAILED

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':compileJava'.
> Could not create an instance of type com.sun.tools.javac.api.JavacTool.

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get
more log output.

BUILD FAILED
```

Example: Gradle 2.7

```
$ gradle --stacktrace myjar

:

Caused by: java.lang.IllegalAccessException: class
org.gradle.internal.reflect.DirectInstantiator cannot access class
com.sun.tools.javac.api.JavacTool (in module jdk.compiler) because module jdk.compiler
does not export package com.sun.tools.javac.api to unnamed module @2f490758
    at org.gradle.internal.reflect.DirectInstantiator.newInstance(DirectInstantiator.java:49)
    ... 82 more
```

Don’t panic

```
--add-exports java.base/sun.security.provider=ALL-UNNAMED
--add-exports java.base/sun.security.pkcs=ALL-UNNAMED
--add-exports java.base/sun.security.util=ALL-UNNAMED
--add-exports java.base/sun.security.x509=ALL-UNNAMED
:
```

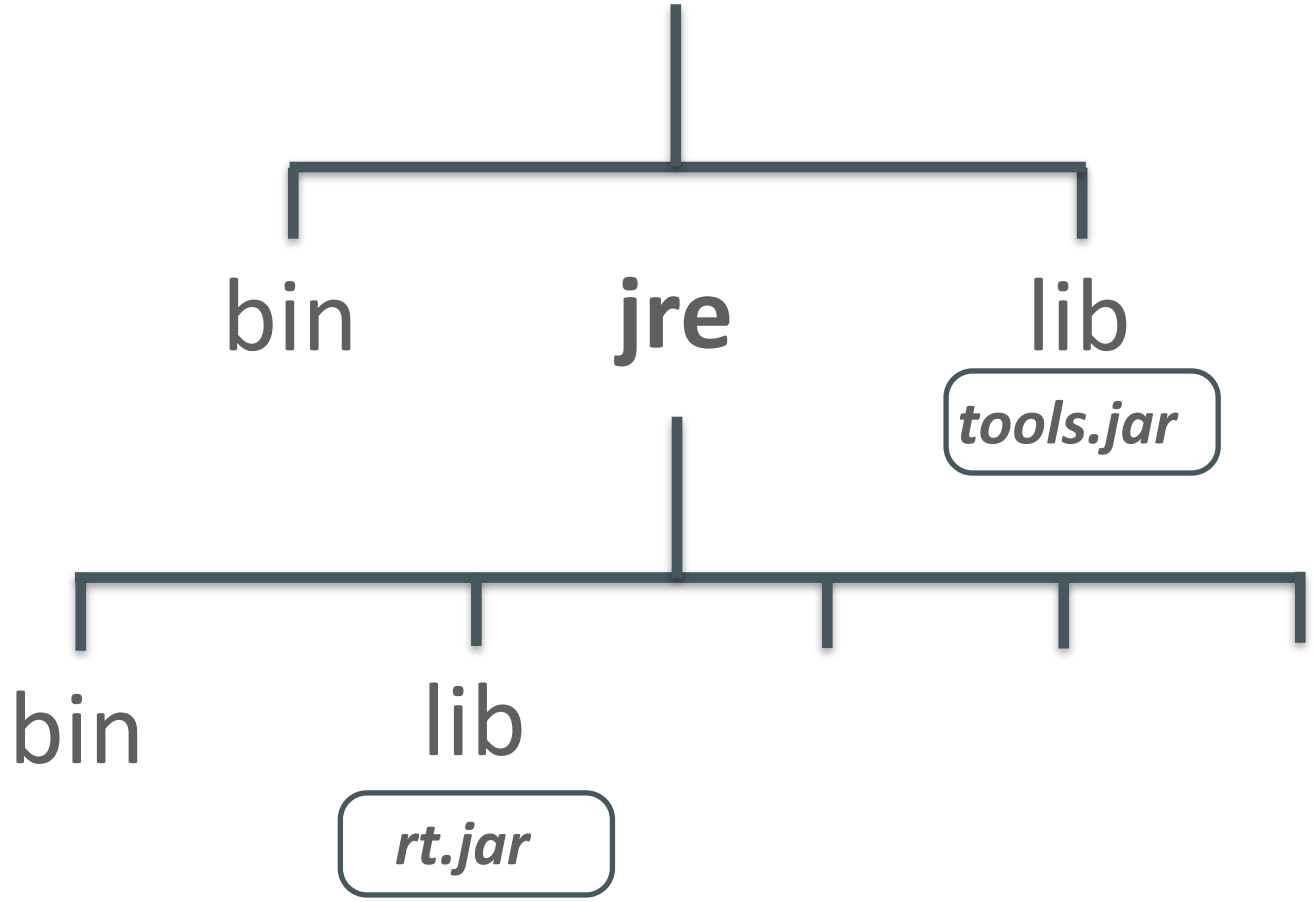
Breaking encapsulation using the command line

```
--add-exports java.base/sun.security.provider=ALL-UNNAMED
```

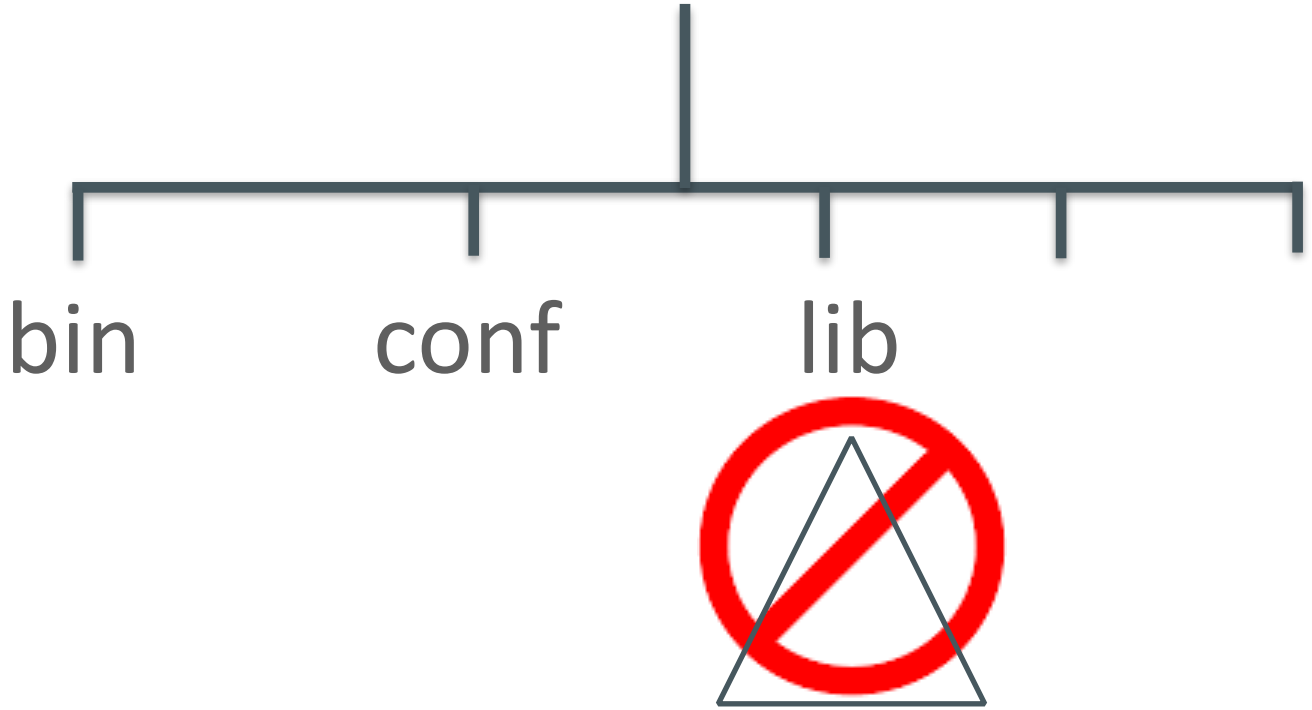
Change the binary structure of the JRE and JDK

- Motivation
- Not an API but still a disruptive change
- Details in JEP 220
- In JDK 9 since late 2014 to give lots of time for the tools to catch up

JDK 8 runtime image



Modular run-time image



~~jre directory~~

~~rt.jar~~

~~tools.jar~~

Removed 6 deprecated methods

- Removed
  - java.util.logging.LogManager::addPropertyChangeListener
  - java.util.logging.LogManager::removePropertyChangeListener
  - java.util.jar.Pack200.Packer::addPropertyChangeListener
  - java.util.jar.Pack200.Packer::removePropertyChangeListener
  - java.util.jar.Pack200.Unpacker::addPropertyChangeListener
  - java.util.jar.Pack200.Unpacker::removePropertyChangeListener
- Flagged for removal in JSR 337, and JEP 162

Removed

- Endorsed standards override mechanism
- Extension mechanism
- Preparation:
  - JDK 8u40 introduced -XX:+CheckEndorsedAndExtDirs to print a message if you are using the extension or standard mechanisms

Other changes

- Application and extension class loaders are no longer instances of java.net.URLClassLoader
- Removed: -Xbootclasspath and -Xbootclasspath/p are removed
- Removed: system property sun.boot.class.path
- JEP 261 has the full list of the issues that we know about

New version-string scheme, JEP 223

- Old versioning format is difficult to understand
- New format addresses these problems
- Impacts java -version and related properties

New version-string format

	Old		New	
Release Type	long	short	long	short
Early Access	1.9.0-ea-b19	9-ea	9-ea+19	9-ea
Major	1.9.0-b100	9	9+100	9
Security #1	1.9.0_5-b20	9u5	9.0.1+20	9.0.1
Security #2	1.9.0_11-b12	9u11	9.0.2+12	9.0.2
Minor #1	1.9.0_20-b62	9u20	9.1.2+62	9.1.2
Security #3	1.9.0_25-b15	9u25	9.1.3+15	9.1.3
Security #4	1.9.0_31-b08	9u31	9.1.4+8	9.1.4
Minor #2	1.9.0_40-b45	9u40	9.2.4+45	9.2.4

System properties

System Property	Existing	Proposed
Major (GA)		
java.version	1.9.0	9
java.runtime.version	1.9.0-b100	9+100
java.vm.version	1.9.0-b100	9+100
java.specification.version	1.9	9
java.vm.specification.version	1.9	9
Minor #1 (GA)		
java.version	1.9.0_20	9.1.2
java.runtime.version	1.9.0_20-b62	9.1.2+62
java.vm.version	1.9.0_20-b62	9.1.2+62
java.specification.version	1.9	9
java.vm.specification.version	1.9	9

General rule: Look out for unrecognized VM options

- Launching JRE with unrecognized VM options fails
- Using deprecated options in JDK 8 triggers warning messages

```
$ java -XX:MaxPermSize=1G -version
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize; support was removed in 8.0
```

- Previously deprecated options, removed in JDK 9, will fail to start
  - Perm generation was removed in JDK 8 (JEP 122)
    - Remove the permanent generation from the Hotspot JVM and thus the need to tune the size of the permanent generation. Part of the Hotspot & JRockit convergence effort
  - Expect many programs to run into this problem

What can you do to prepare?

- Check code for usages of JDK-internal APIs with jdeps
- Check code that might be sensitive to the version change
- If you develop tools then check code for a dependency on rt.jar or tools.jar or the runtime-image layout
- Test the JDK 9 EA builds and Project Jigsaw EA builds – weekly!
- Get familiar with new features like Multi Release JARs
- Join the Early Access mailing list - email quality-discuss@openjdk.java.net or rory.odonnell@oracle.com

Two JEPs to get familiar with.

- JEP 238 : Multi-Release JAR Files
  - Extend the JAR file format to allow multiple, Java-release-specific versions of class files to coexist in a single archive
- JEP 247 : Compile for Older Platforms Versions
  - Enhance javac so that it can compile Java programs to run on selected older versions of the platform.

More information

- OpenJDK Project Jigsaw Page , this has links to the JEPs
  - http://openjdk.java.net/projects/jigsaw
  - mailto:jigsaw-dev@openjdk.java.net
- Early Access builds
  - http://jdk9.java.net/downloads
- Java Dependency Analysis Tool
  - https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool
- JEP 223: New Version string
  - http://openjdk.java.net/jeps/223
- JEP 238: Multi-Release JAR files
  - http://openjdk.java.net/jeps/238

Even more information

- Prepare for JDK 9: by Alan Bateman
- Introduction to Module Development by Alan Bateman
- Modules and Services by Alex Buckley
- Advanced Modular Development by Alan Bateman and Alex Buckley
- Project Jigsaw: Under The Hood by Alex Buckley


Java™
ORACLE®