

Prepare for JDK 9!

Alan Bateman
Java Platform Group, Oracle
February 2016



#Jfokus #Jigsaw #JDK9

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Background: JDK 9 and Project Jigsaw goals

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

Background: Modularity Landscape

- Java Platform Module System
 - JSR 376, targeted for Java SE 9
- Java SE 9
 - JSR XXX, will own the modularization of the Java SE APIs
- OpenJDK Project Jigsaw
 - Reference implementation for JSR 376
 - JEP 200, 201, 220, 260, 261

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

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 `ClassNotFoundException`, 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.

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 advanced notice

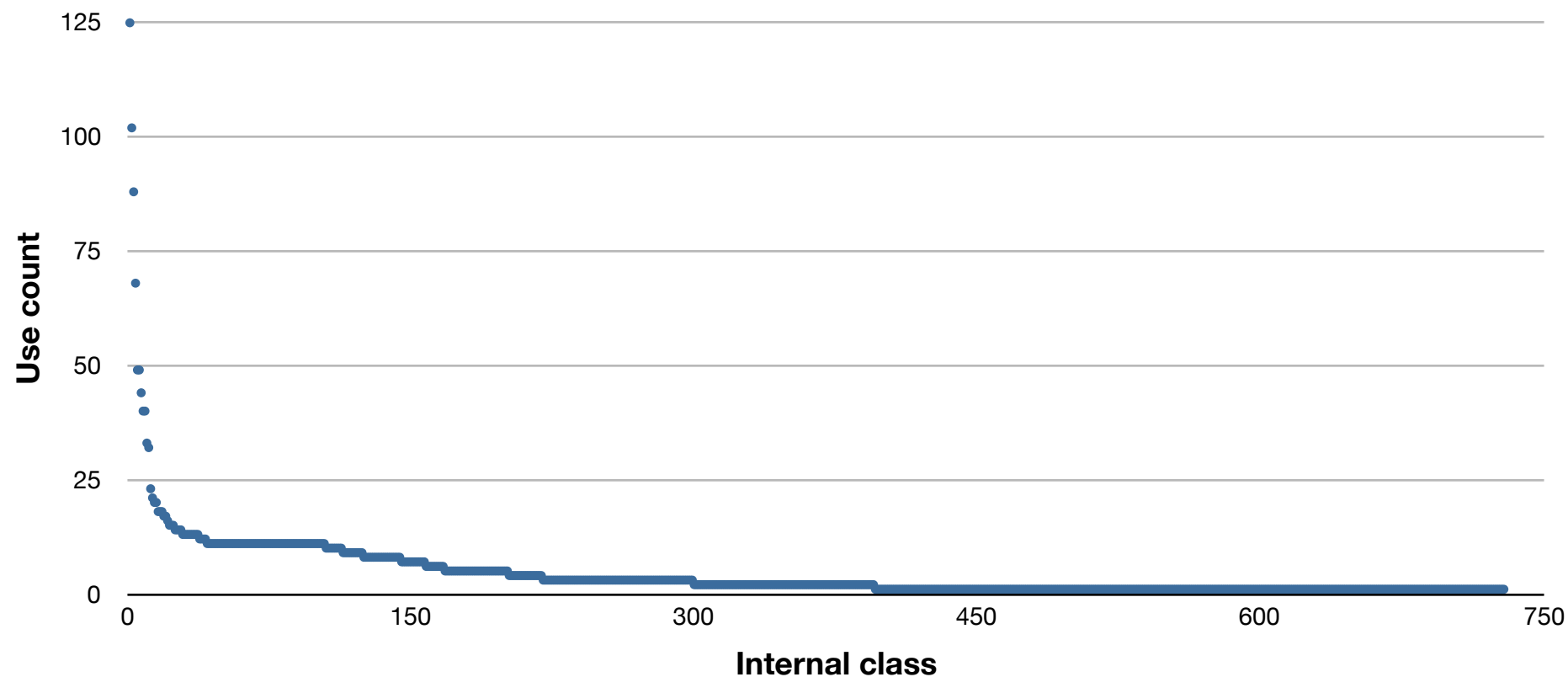
Managing incompatibilities

- 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 make sense
- Provide workarounds where possible

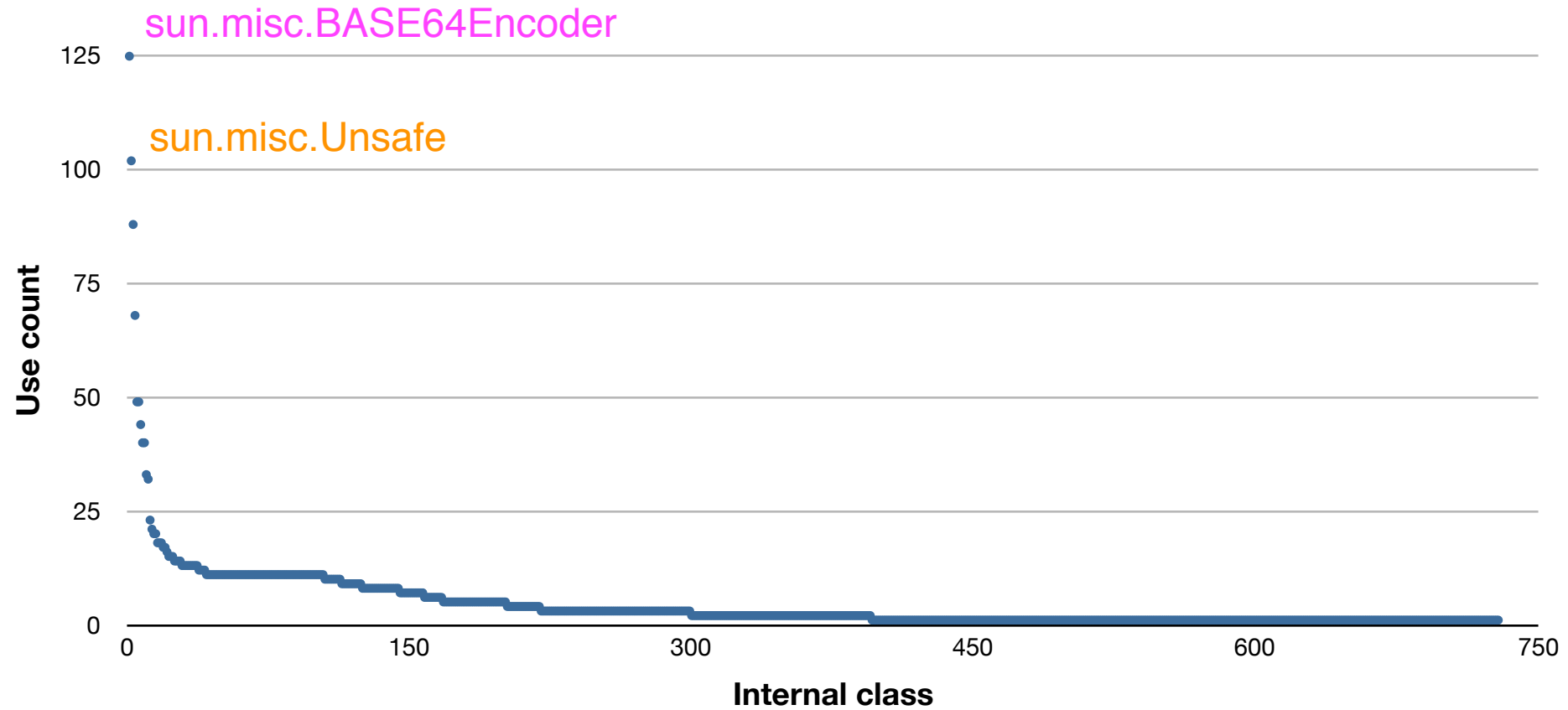
Incompatible changes in JDK 9

- Encapsulate most JDK-internal APIs
- Remove a small number of supported, JCP-standard APIs
- Change the binary structure of the JRE and JDK
- Remove the endorsed-standards override and extension mechanisms
- New version-string format
- Underscore not allowed as a one-character identifier in source code

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 proposal

- 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 proposal

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

Uses of JDK-internal APIs

State	Count	Type
R8	125	sun.misc.BASE64Encoder
C	102	sun.misc.Unsafe
R8	88	sun.misc.BASE64Decoder
R2	68	java.awt.peer.ComponentPeer
R4	49	com.sun.image.codec.jpeg.JPEGImageEncoder
R4	49	com.sun.image.codec.jpeg.JPEGCodec
R3	44	com.sun.net.ssl.internal.ssl.Provider
R1	40	sun.security.action.GetPropertyAction
R4	40	com.sun.image.codec.jpeg.JPEGEncodeParam
C	33	sun.reflect.ReflectionFactory
X	32	sun.security.util.DerValue
X	23	com.sun.org.apache.xml.internal.serialize.XMLSerializer
R4	21	com.sun.image.codec.jpeg.JPEGImageDecoder
C	20	sun.reflect.ReflectionFactory\$GetReflectionFactoryAction
X	20	com.sun.org.apache.xml.internal.serialize.OutputFormat
X	18	sun.net.www.ParseUtil
R4	18	sun.security.x509.X500Name
X	18	sun.security.util.ObjectIdentifier
R9	17	sun.security.krb5.EncryptionKey
R9	17	com.sun.org.apache.xml.internal.resolver.tools.CatalogResolver
R0	16	java.awt.peer.LightweightPeer
X	15	sun.net.www.protocol.http.HttpURLConnection
R6	15	sun.misc.Service
R9	15	sun.awt.CausedFocusEvent\$Cause
X	14	sun.nio.cs.Surrogate\$Parser

State	Count	Type
X	14	sun.nio.cs.Surrogate
R7	14	com.sun.rowset.CachedRowSetImpl
X	14	sun.security.x509.X509CertImpl
X	13	sun.java2d.pipe.Region
S9	13	org.w3c.dom.xpath.XPathResult
S9	13	org.w3c.dom.xpath.XPathNSResolver
S9	13	org.w3c.dom.xpath.XPathEvaluator
X	13	com.sun.org.apache.xpath.internal.objects.XObject
X	13	com.sun.org.apache.xpath.internal.objects.XNodeSet
R9	13	com.sun.org.apache.xml.internal.resolver.CatalogManager
R9	13	com.sun.org.apache.xml.internal.resolver.Catalog
X	13	com.sun.org.apache.xerces.internal.jaxp.DocumentBuilder...
C	12	sun.reflect.Reflection
X	12	sun.misc.CharacterEncoder
X	12	sun.security.util.DerInputStream

- C = Critical, no supported replacement in 8, will remain in 9, gone in 10
- S9 = Non-critical, but now supported in 9
- RN = Non-critical, supported replacement added in JDK N (N < 9), encapsulated in 9
- X = Non-critical, no replacement planned, encapsulated in 9

Finding uses of JDK-internal APIs

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

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



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

```
security.jar -> java.base
```

```
com.sun.enterprise.common.iioop.security.GSSUPName (security.jar)
  -> sun.security.util.ObjectIdentifier          JDK internal API (java.base)
com.sun.enterprise.common.iioop.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



#Jfokus #Jigsaw #JDK9

Example: Glassfish 4.1

java.lang.IllegalAccessError: class com.sun.enterprise.security.provider.PolicyWrapper (unnamed module) cannot access class sun.security.provider.PolicyFile (module java.base), sun.security.provider is not exported

```
at 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
```



#Jfokus #Jigsaw #JDK9

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!

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

Don't panic!

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

The screenshot shows a JIRA issue page for a bug. The title is "JDK9 - REFERENCES TO JDK INTERNAL API IN main/appserver/security/core-ee/src/main/java/com/sun/enterprise/security/provider/PolicyWrapper.java". The issue is categorized as a "Bug" with a "Major" priority. It affects version 4.1 and is associated with the "security" component. The status is "Open" and the resolution is "Unresolved". The description details a runtime error: "java.lang.IllegalAccessError: class com.sun.enterprise.security.provider.PolicyWrapper (in module: Unnamed Module) cannot access class sun.security.provider.PolicyFile (in module: java.base), sun.security.provider is not exported to Unnamed Module". The stack trace includes: "at 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(Native Method)", "at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)", "at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:46)".

-XaddExports = breaking encapsulation

`-XaddExports:java.base/sun.security.provider=ALL-UNNAMED`



Incompatible changes in JDK 9

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

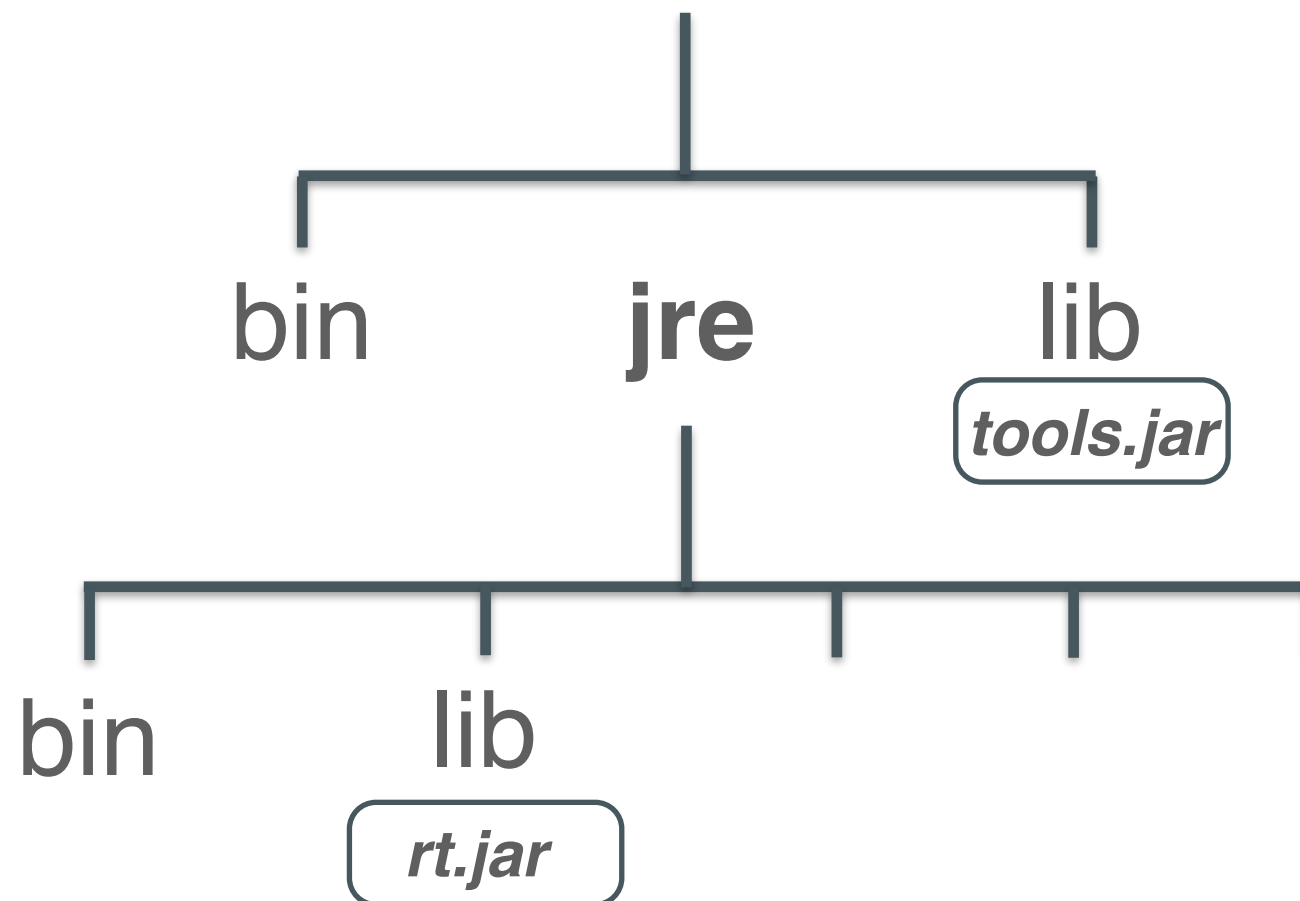
Removed 6 deprecated methods

- 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

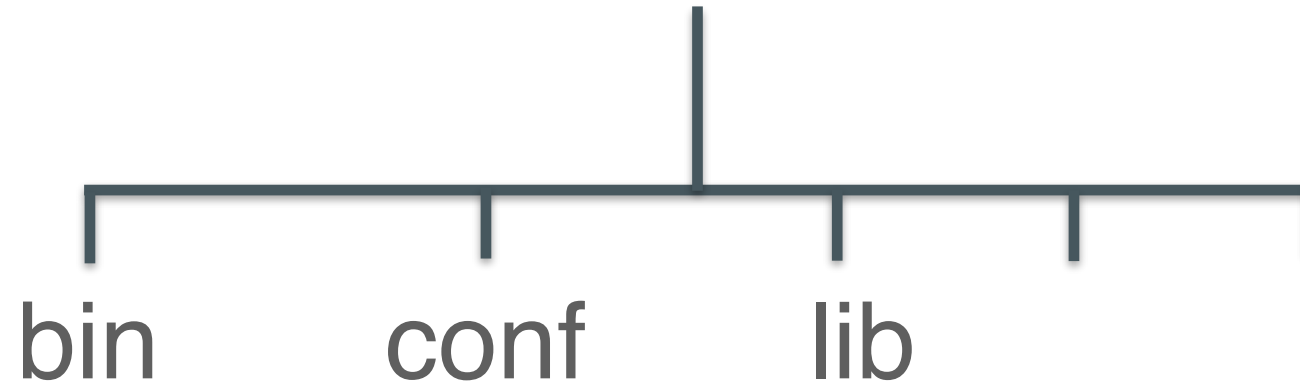
Change the binary structure of the JRE and JDK

- 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

Legacy JDK image



Modular run-time image



~~jre directory~~

~~rt.jar~~

~~tools.jar~~



Removed

- Endorsed standards override mechanism
- Extension mechanism

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

Release Type	Old		New	
	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

_ no longer allowed as a one-character identifier

- Source incompatibility

```
MyClass.java:2: error: as of release 9, '_' is a keyword, and may not be  
used as an identifier
```

```
    static Object _ = new Object();  
                ^
```

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
- Check code for uses of underscore as an identifier
- 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

More Information

OpenJDK Project Jigsaw page, this has links to all the JEPs

<http://openjdk.java.net/projects/jigsaw/>
<mailto:jigsaw-dev@openjdk.java.net>

Early Access Builds

<https://jdk9.java.net/download>
<https://jdk9.java.net/jigsaw/>

Java Dependency Analysis Tool

<https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool>

JEP 223: New Version-String Scheme

<http://openjdk.java.net/jeps/223>



#Jfokus #Jigsaw #JDK9

Safe Harbor Statement

The preceding 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.

ORACLE®