Java at 20
Where are we going?

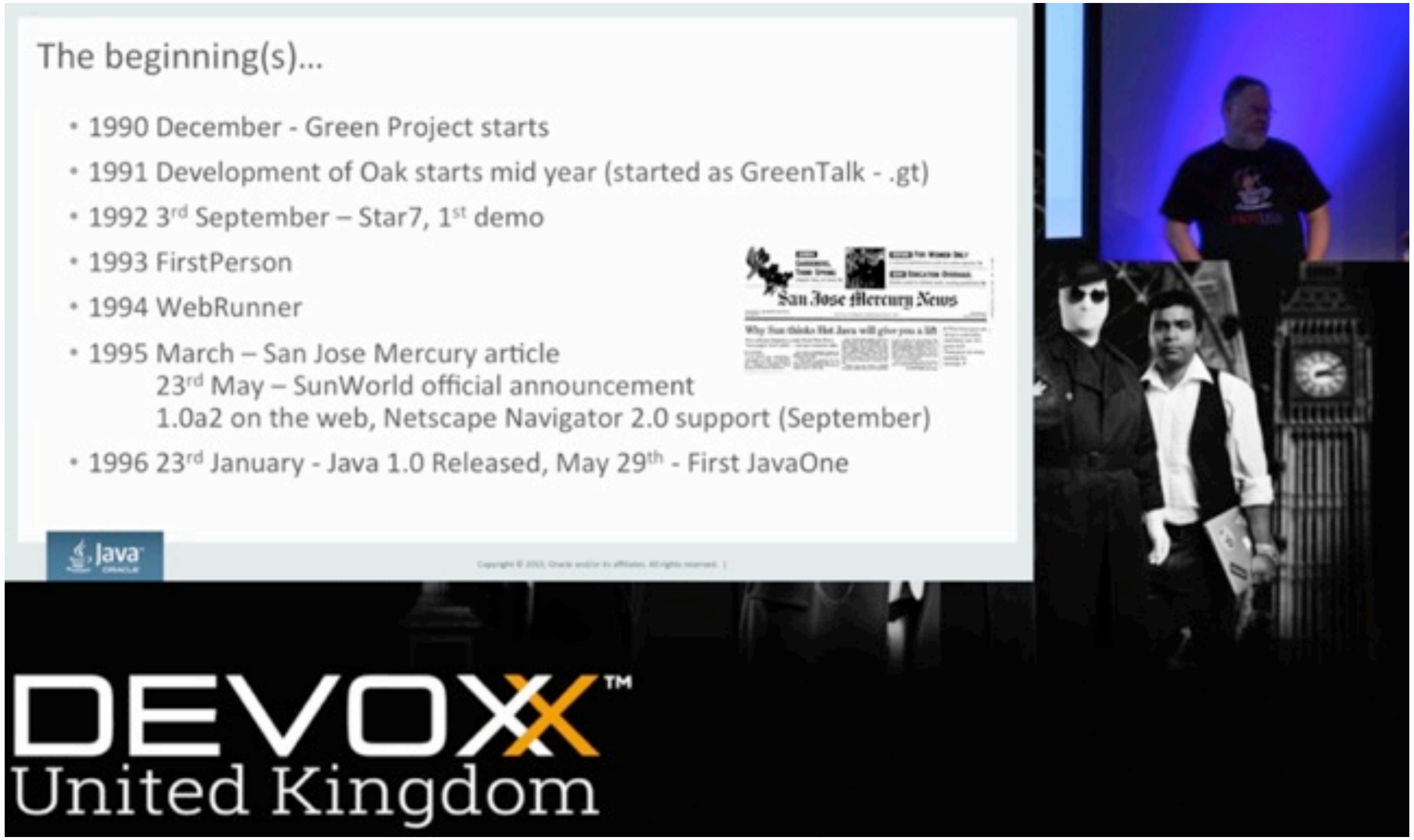
CREATE
THE
FUTURE

Steve Elliott
Java Technology Lead
Oracle UK

Docklands Java User Group
11th August 2015

Safe Harbor Statement

Celebrating Twenty Years of Innovation
#Java20
ORACLE
Java
1995–2015
20
YEARS



The beginning(s)…

- 1990 December - Green Project starts
- 1991 Development of Oak starts mid year (started as GreenTalk - .gt)
- 1992 3rd September – Star7, 1st demo
- 1993 FirstPerson
- 1994 WebRunner
- 1995 March – San Jose Mercury article
  23rd May – SunWorld official announcement
  1.0a2 on the web, Netscape Navigator 2.0 support (September)
- 1996 23rd January - Java 1.0 Released, May 29th - First JavaOne

For more history / nostalgia / trip-down-memory-lane …
https://www.parleys.com/tutorial/what-were-you-doing-1995-20-year-retrospective-java

Java Beans, RMI, JIT, Inner Classes

Swing, Plug-in, IDL, Collections, strictfp

HotSpot, RMI/CORBA, JavaSound

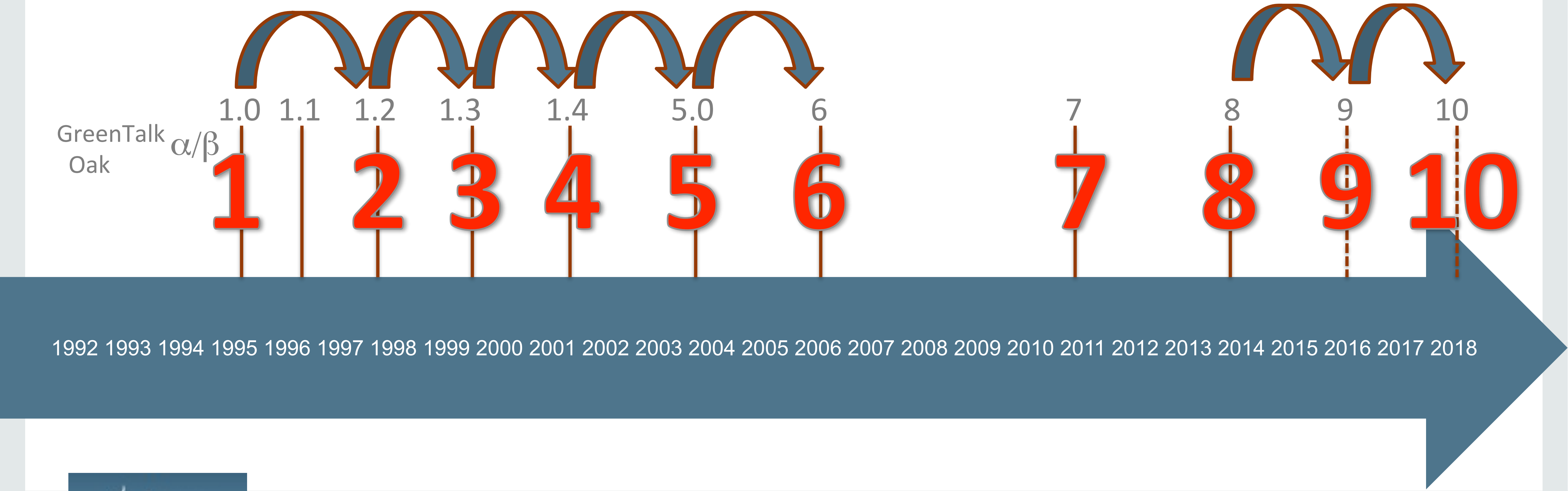
JCP, WebStart, NIO, Logging, assert

Generics, AutoBoxing, Concurrency Utilities, Annotations, varargs, static imports, enum

Scripting, XML, WebSvcs

Coin, Fork-Join, Invokedynamic, NIO-2, 166y

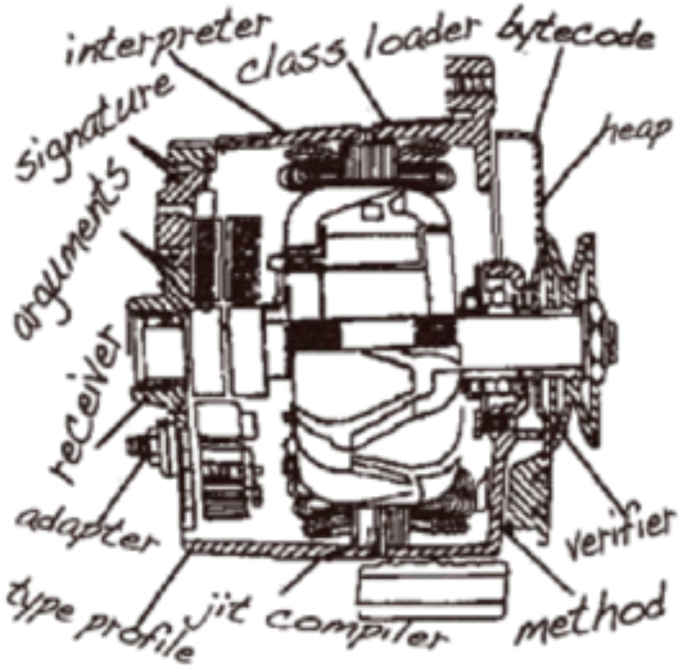
Lambdas, Streams, TypeAnn, Date-Time, Nashorn



Java Virtual Machine

HotSpot and JRockit Convergence (and CDC)

- Performance
- Remove permgen
- JIT Compilers (C1/C2 Tiered Compilation)
- GC improvements / G1 / Rationalisation
- Ergonomics
- Instrumentation / Tuning / Performance



Cloud / Multi-Tenancy / Isolation
Low Latency / Deterministic GC...

Java SE Roadmap

2014

JDK 8
- Lambda
- JSR 310: New Date and Time API
- Nashorn: JavaScript Interoperability
- JavaFX Enhancements

8u20
- G1 Performance Improvement
- JVM Performance Improvements
- Java Mission Control 5.4
- Advanced Management Console 1.0
- MSI Enterprise JRE Installer

2015

8u40
- Performance Improvements
- Density and Resource Management
- Multi-Language Support Improvements
- Accessibility Enhancements
- Continued Java SE Advanced Features

8u60
- Bug Fixes
- Continued Java SE Advanced Features

2016

JDK 9
- Modularity – Jigsaw
- HTTP 2.0
- Cloud optimized JVM
- Continued Java SE Advanced Features

2017

+ CPU updates
(Jan, Apr, Jul, Oct)

Java
End of Public Updates (EoPU)

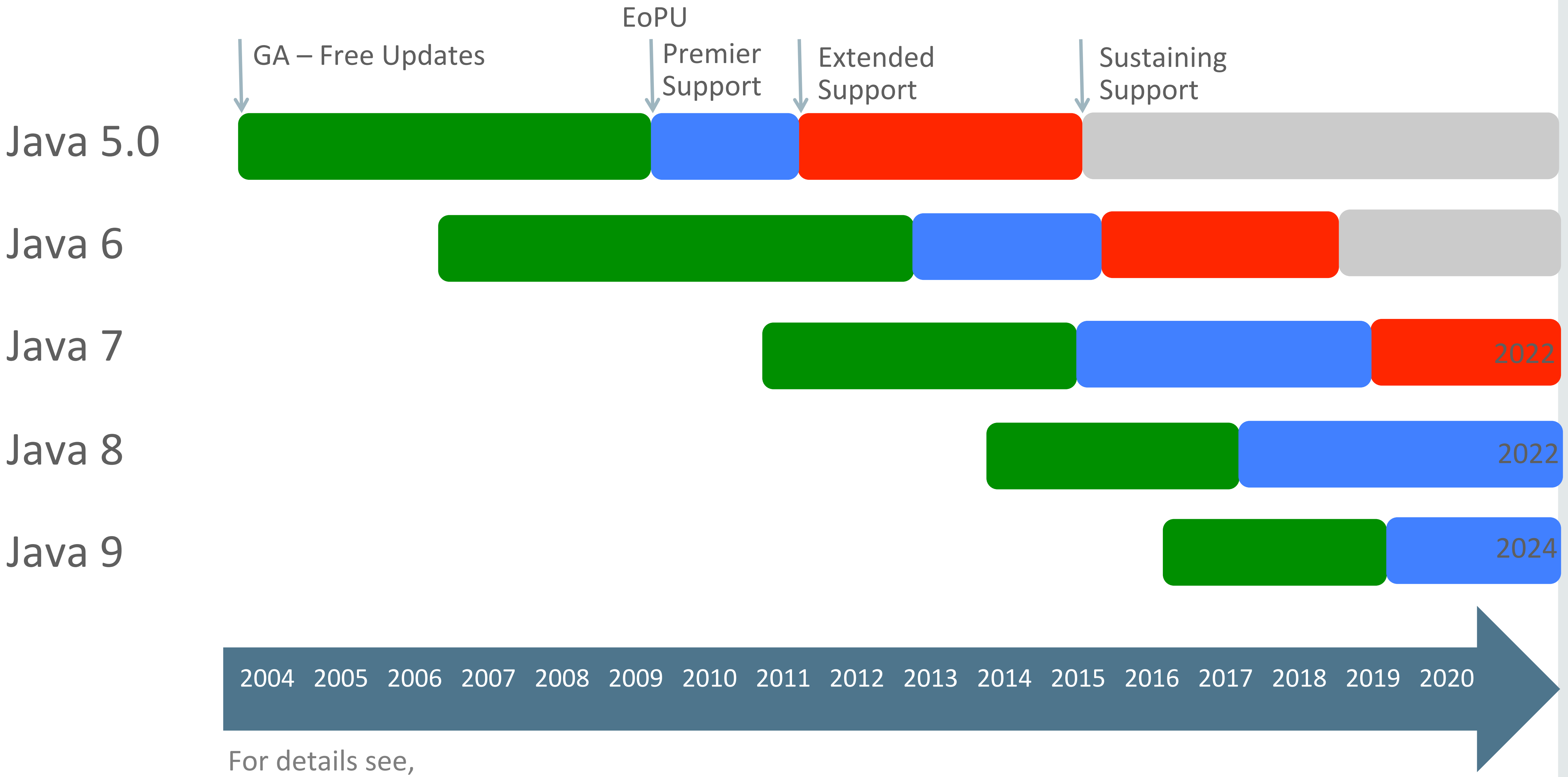
- Public Java updates are available until all three of these conditions occur
  - Three years after general availability
  - One year after being superseded by a new major release
  - Six months after the new major release is made the default on java.com
- For Java 7 this happened in April 2015
  - java.com switched to JDK 8 on Oct 2014
  - AutoUpdate from JRE 7 to JRE 8 started January 2015
  - JDK 7 updates only on MOS from Jul 2015 CPU
    (From then these patches do not go back into OpenJDK)

Java SE EOL / Lifetime Support Policy 3(+5+3+) years

	GA Date	EoPU	Premier Support	Extended Support
Java SE 1.4.2	Feb 2002	Oct 2008	Feb 2010	Feb 2013
Java SE 5	May 2004	Oct 2009	May 2011	May 2015
Java SE 6	Dec 2006	Feb 2013	~~Dec 2013~~ Dec 2015	~~Jun 2017~~ Dec 2018
Java SE 7	Jul 2011	Apr 2015	~~Jul 2016~~ Jul 2019	~~Jul 2019~~ Jul 2022
Java SE 8	Mar 2014	Mar 2017 *	Mar 2022	Mar 2025

For details see, http://www.oracle.com/technetwork/java/eol-135779.html
* Or later. Exact date TBD.

Deployment technologies (browser based) : Java 6 Premier – Jun 2017, Java 7+ Premier – 5yrs after GA, No Extended Support (moves to Sustaining)



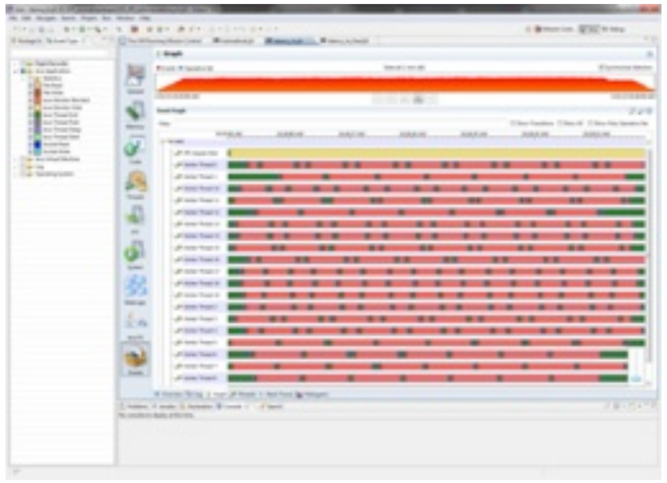
For details see,
http://www.oracle.com/technetwork/java/eol-135779.html

Java in the enterprise

Java SE Advanced

Mission Control

- Diagnose complex issues
- Low overhead ***Flight Recorder***
- Back-in-time analysis
- Plugins for many systems



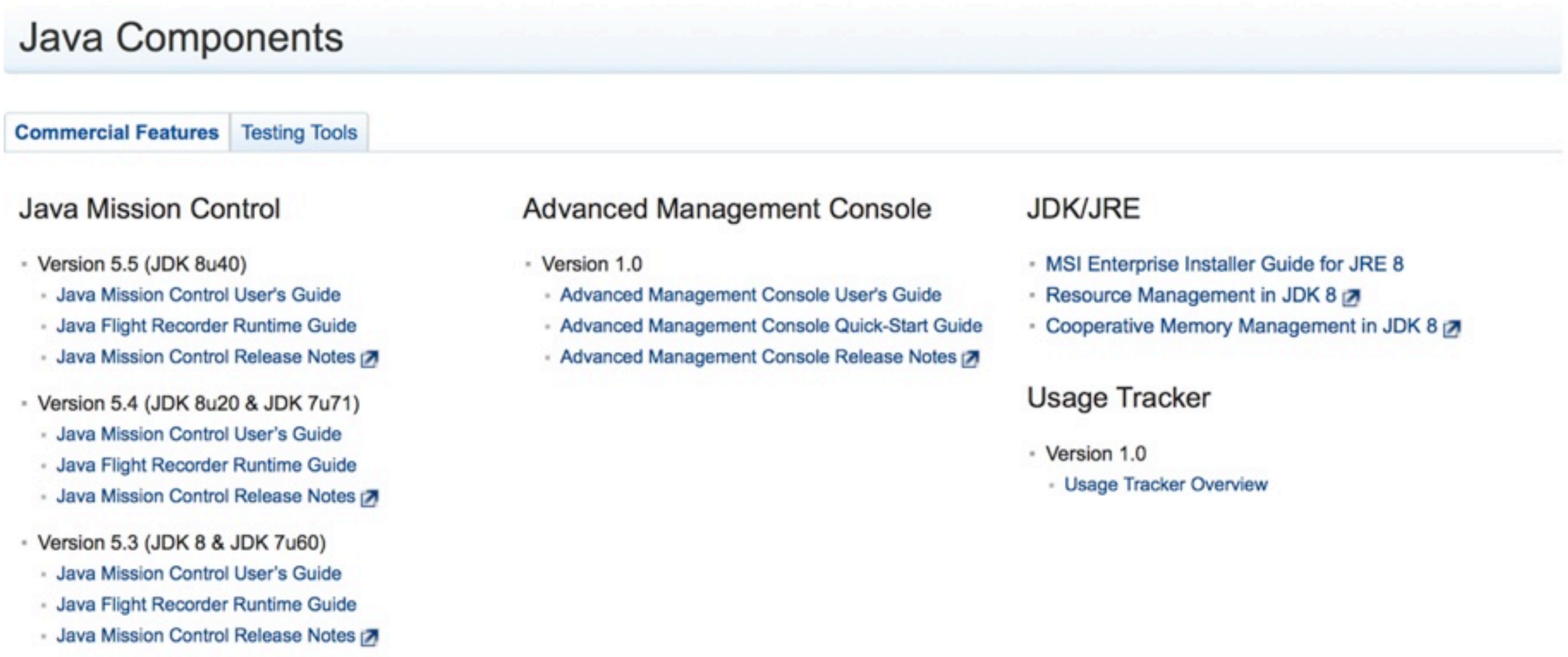
Advanced Management Console

- Track and manage clients at scale
- Usage tracking and Deployment Rule Sets

Support & Updates

- Support of complex issues & changes
- Updates to old versions like JDK 6

https://docs.oracle.com/javacomponents/



Java Components

Commercial Features | Testing Tools

Java Mission Control

- Version 5.5 (JDK 8u40)
  - Java Mission Control User's Guide
  - Java Flight Recorder Runtime Guide
  - Java Mission Control Release Notes
- Version 5.4 (JDK 8u20 & JDK 7u71)
  - Java Mission Control User's Guide
  - Java Flight Recorder Runtime Guide
  - Java Mission Control Release Notes
- Version 5.3 (JDK 8 & JDK 7u60)
  - Java Mission Control User's Guide
  - Java Flight Recorder Runtime Guide
  - Java Mission Control Release Notes

Advanced Management Console

- Version 1.0
  - Advanced Management Console User's Guide
  - Advanced Management Console Quick-Start Guide
  - Advanced Management Console Release Notes

JDK/JRE

- MSI Enterprise Installer Guide for JRE 8
- Resource Management in JDK 8
- Cooperative Memory Management in JDK 8

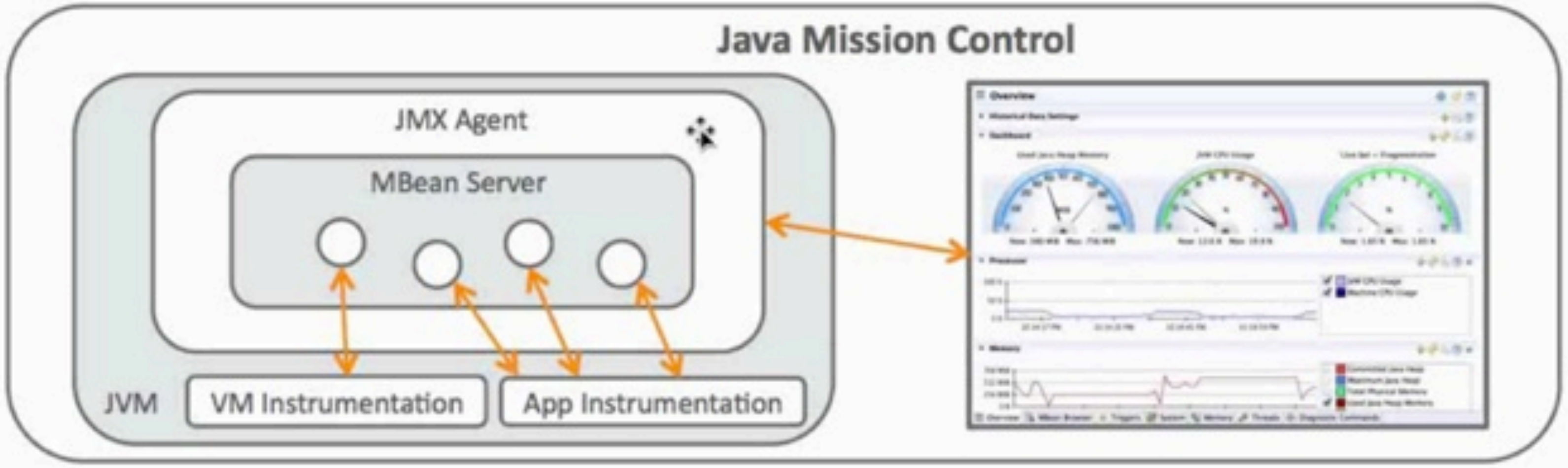
Usage Tracker

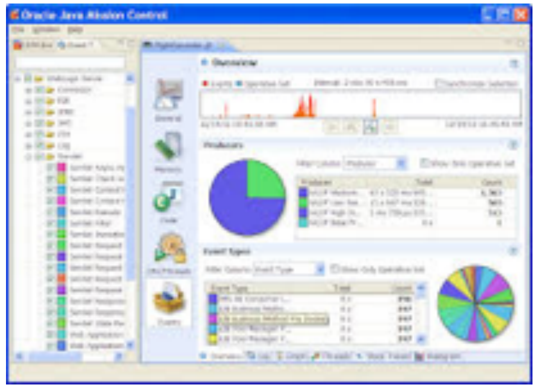
- Version 1.0
  - Usage Tracker Overview

Java Mission Control / Java Flight Recorder (free for development)

Built for monitoring, profiling and troubleshooting Java applications, **Java Mission Control** consists of:

- **JMX Console** for monitoring JVM and application in real time
- **Java Flight Recorder** for collecting data about JVM and application
- Optional tools via plug-ins (e.g. heap dump analysis, DTrace recording)



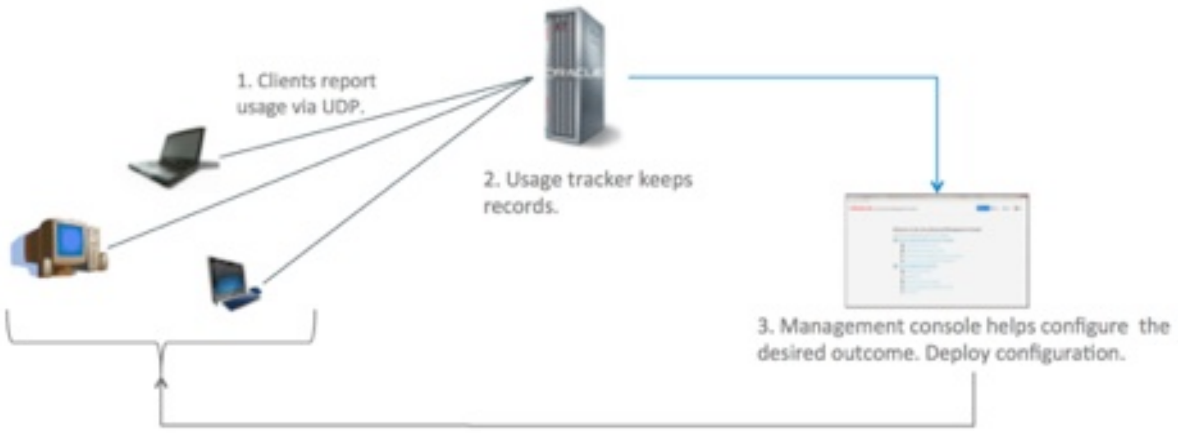


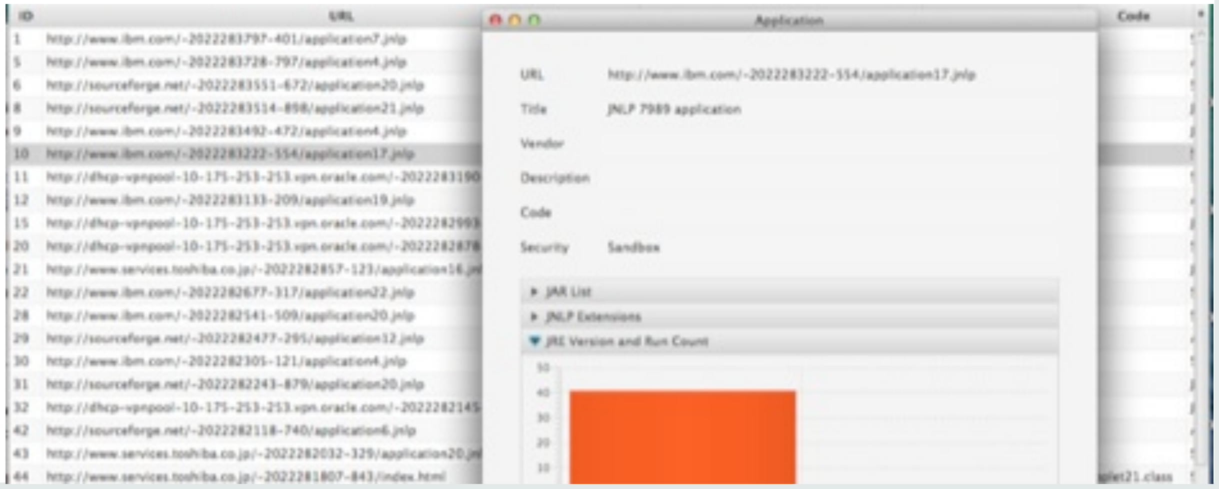
Tutorial -- http://hirt.se/blog/?p=611

Advanced Management Console

Java Monitoring & Management

- Usage tracking across installations.
  - Tracks applications run & Java versions used
  - Properly identifies application
- Deployment Rule Set tool
  - Control prompts: run or block
  - Automatically run with a specific Java version.
  - Guided from usage tracker data
  - Verifiable against tracker data


1. Clients report usage via UDP.
2. Usage tracker keeps records.
3. Management console helps configure the desired outcome. Deploy configuration.



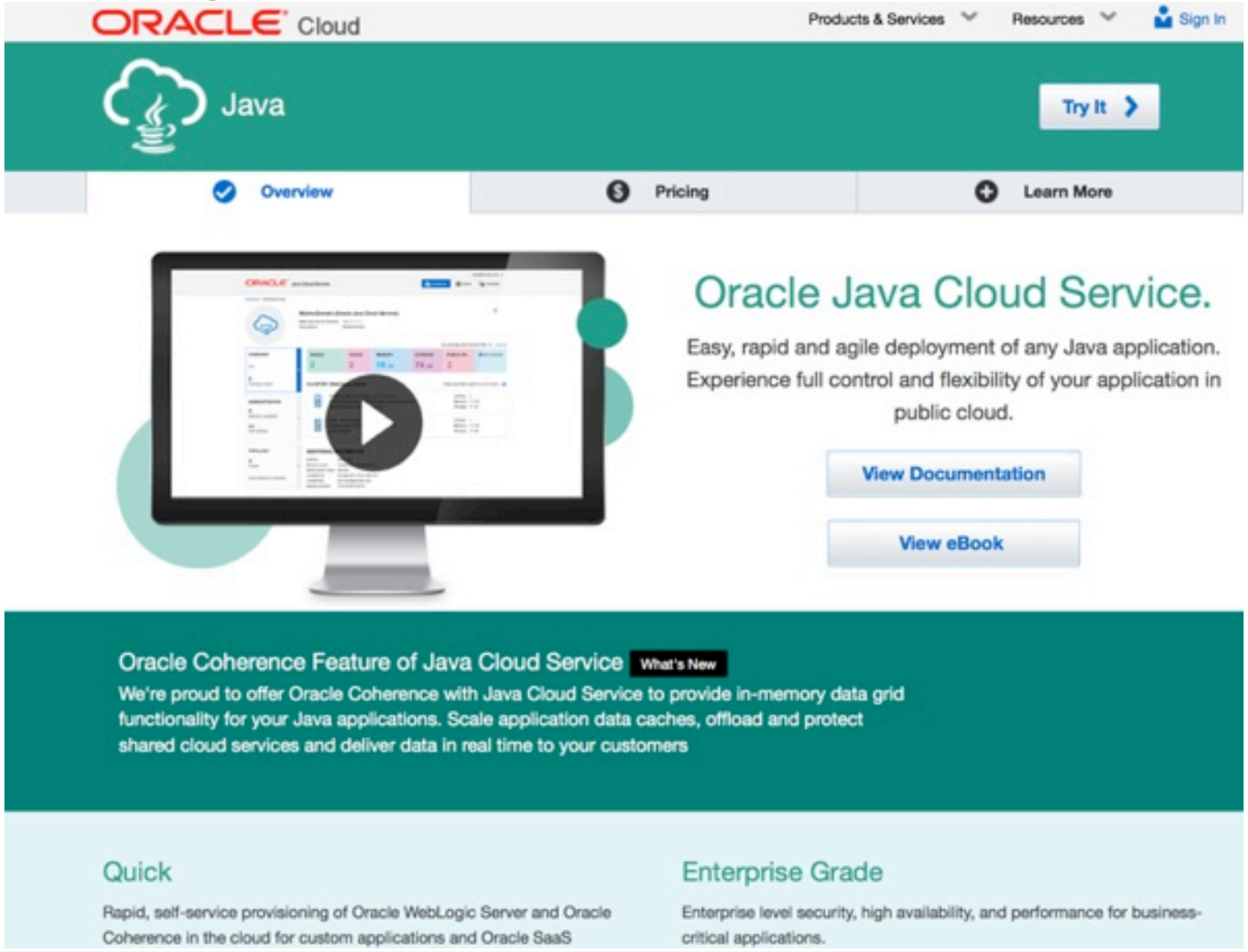
ID	URL
1	http://www.ibm.com/-2022283797-401/application7.jnlp
5	http://www.ibm.com/-2022283728-797/application4.jnlp
6	http://sourceforge.net/-2022283551-672/application20.jnlp
8	http://sourceforge.net/-2022283514-898/application21.jnlp
9	http://www.ibm.com/-2022283492-472/application4.jnlp
10	http://www.ibm.com/-2022283222-554/application17.jnlp
11	http://dhcp-vpnpool-10-175-253-253.vpn.oracle.com/-2022283190
12	http://www.ibm.com/-2022283133-209/application19.jnlp
15	http://dhcp-vpnpool-10-175-253-253.vpn.oracle.com/-2022282993
20	http://dhcp-vpnpool-10-175-253-253.vpn.oracle.com/-2022282878
21	http://www.services.toshiba.co.jp/-2022282857-123/application16.jn
22	http://www.ibm.com/-2022282677-317/application22.jnlp
28	http://www.ibm.com/-2022282541-509/application20.jnlp
29	http://sourceforge.net/-2022282477-295/application12.jnlp
30	http://www.ibm.com/-2022282305-121/application4.jnlp
31	http://sourceforge.net/-2022282243-879/application20.jnlp
32	http://dhcp-vpnpool-10-175-253-253.vpn.oracle.com/-2022282145
42	http://sourceforge.net/-2022282118-740/application6.jnlp
43	http://www.services.toshiba.co.jp/-2022282032-329/application20.jn
44	http://www.services.toshiba.co.jp/-2022281807-843/index.html

Application

URL	http://www.ibm.com/-2022283222-554/application17.jnlp
Title	JNLP 7989 application
Vendor	
Description	
Code	
Security	Sandbox

ORACLE
Java
1995–2015
20
YEARS
#1 Development Platform,
Now in the
Cloud
#Java20

cloud.oracle.com/java



Oracle Java Cloud Service.

Easy, rapid and agile deployment of any Java application. Experience full control and flexibility of your application in public cloud.

Oracle Coherence Feature of Java Cloud Service What's New

We're proud to offer Oracle Coherence with Java Cloud Service to provide in-memory data grid functionality for your Java applications. Scale application data caches, offload and protect shared cloud services and deliver data in real time to your customers

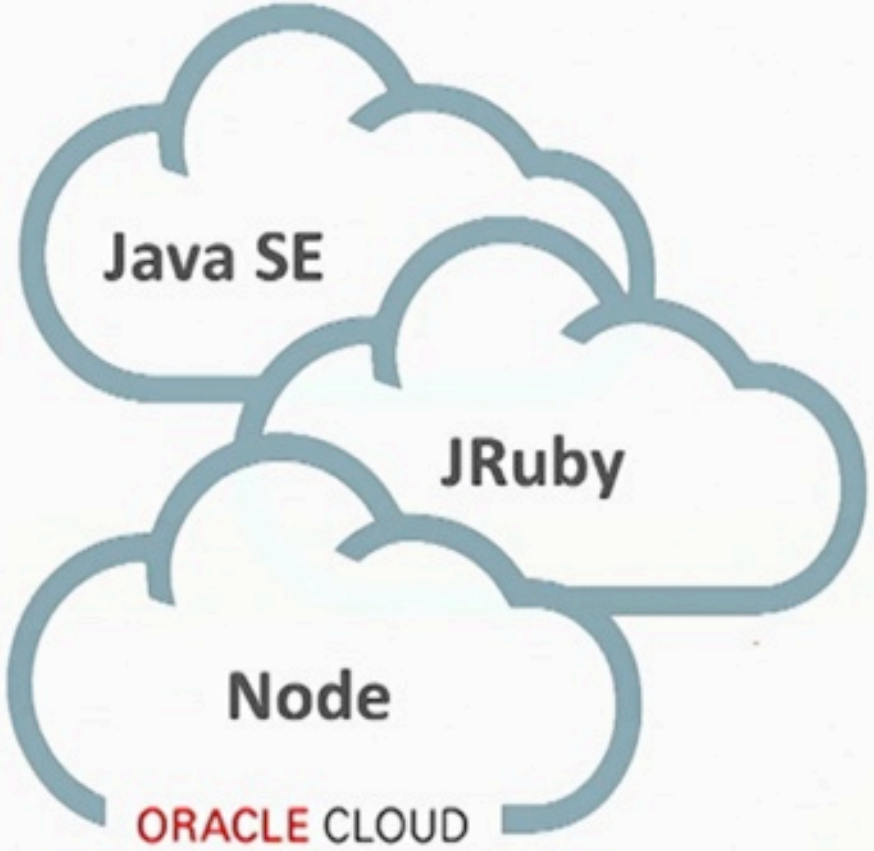
Quick

Rapid, self-service provisioning of Oracle WebLogic Server and Oracle Coherence in the cloud for custom applications and Oracle SaaS

Enterprise Grade

Enterprise level security, high availability, and performance for business-critical applications.

New: Java SE, JRuby And Node Cloud Services



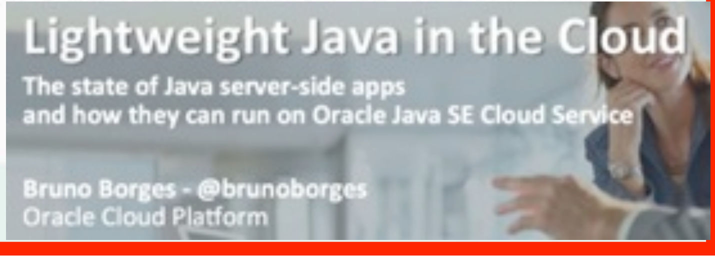
Key Features

- Useful for any Java, Node.js or Ruby Framework
- Java SE advanced and 1000s of Node Libraries on Oracle Cloud
- IDE Choice - JDeveloper, Eclipse, NetBeans - and API access
- Continuous integration with Developer Cloud
- Cloud tooling for lifecycle management

Benefits

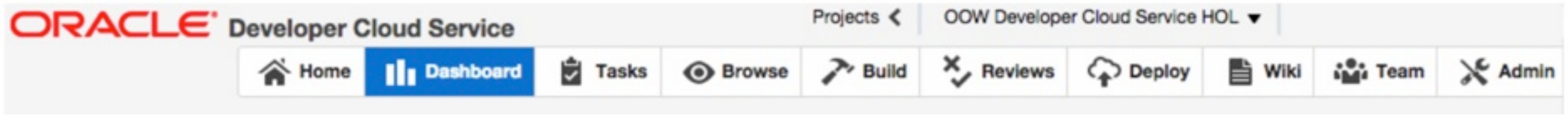
- Self-service application platform with advanced cloud tools
- Secure, Highly Available with Clustering
- Fully automated provisioning, patching, backup, and recovery

http://www.slideshare.net/brunoborges/lightweight-java-in-the-cloud


Lightweight Java in the Cloud
The state of Java server-side apps
and how they can run on Oracle Java SE Cloud Service
Bruno Borges - @brunoborges
Oracle Cloud Platform

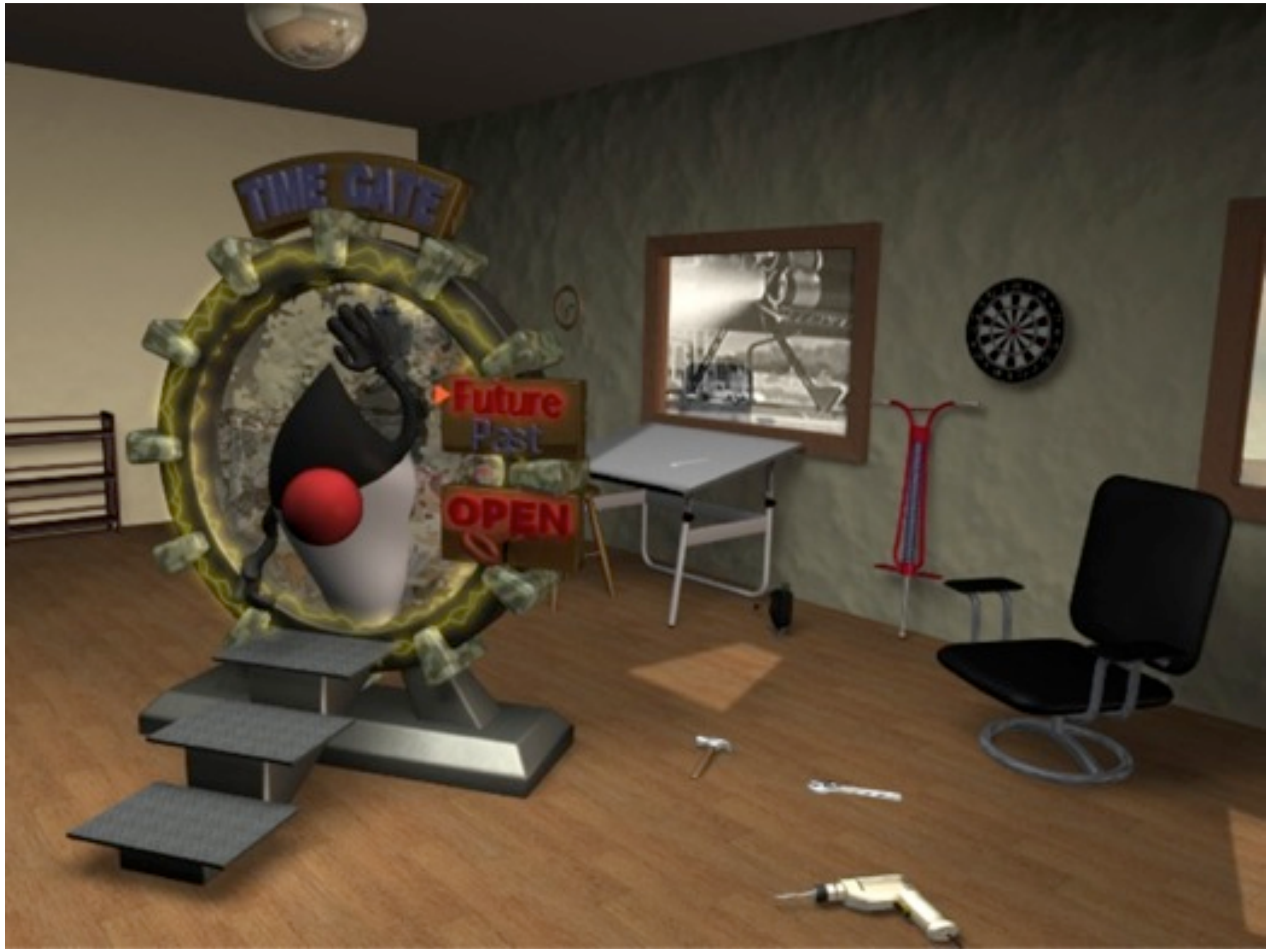
Oracle Developer Cloud Service

https://cloud.oracle.com/developer



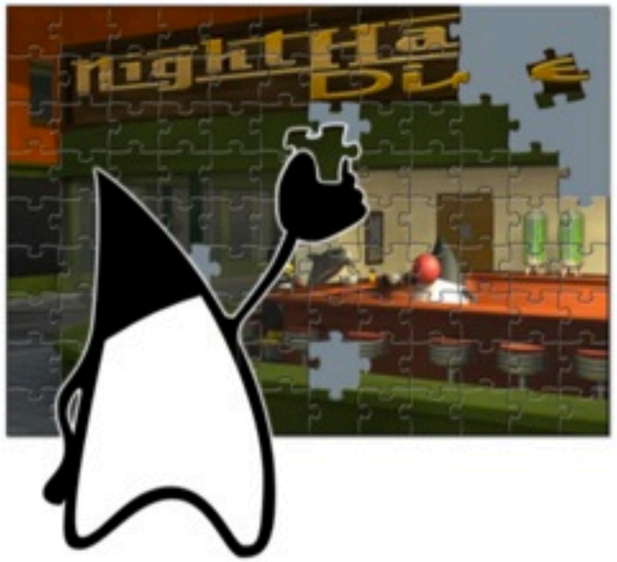
- Free with Java Cloud Service or Messaging Cloud Service
- Already used by 21 different product development organizations within Oracle
- Features include:
  - Project based, multi-tenant
  - Integrated wiki server
  - Integrated task/defect service
  - IDE integration
  - Code review
  - Flexible source repository
  - Maven integration
  - Continuous integration

The Future…



Java 9

Project Jigsaw



JEP 200: The Modular JDK

JEP 201: Modular Source Code

JEP 220: Modular Run-Time Images

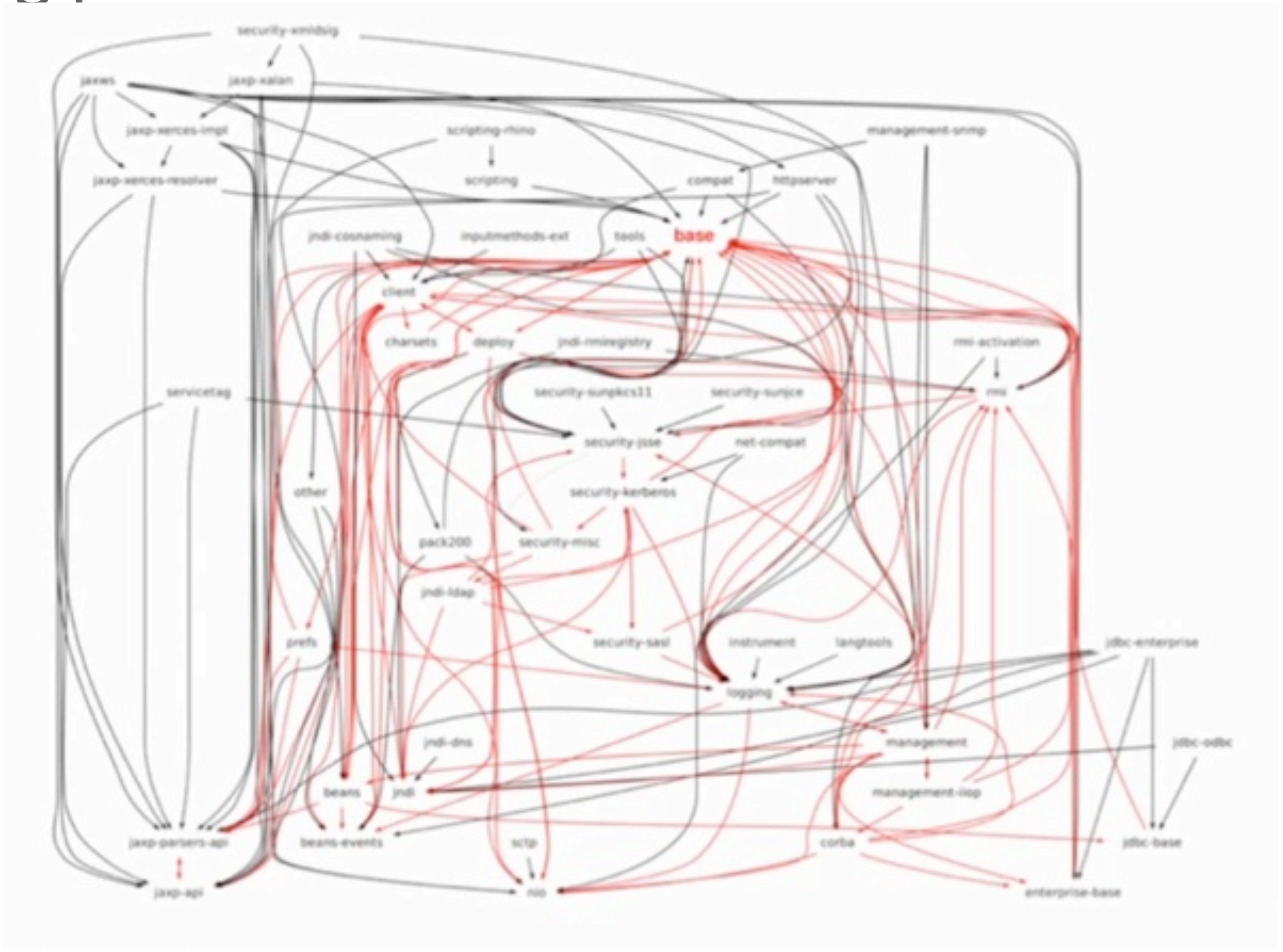
JEP TBD / JSR 376: Java Platform Module System

OpenJDK *http://openjdk.java.net/projects/jigsaw*

```
java.io
java.lang
java.lang.annotation
java.lang.invoke
java.lang.ref
java.lang.reflect
java.math
java.net
java.nio
java.nio.channels
java.nio.channels.spi
java.nio.charset
java.nio.charset.spi
java.nio.file
java.nio.file.attribute
java.nio.file.spi
java.security
java.security.cert
java.security.interfaces
java.security.spec
java.text
java.text.spi
java.time
java.time.chrono
java.time.format
java.time.temporal
java.time.zone
java.util
java.util.concurrent
java.util.concurrent.atomic
java.util.concurrent.locks
java.util.function
java.util.jar
java.util.regex
java.util.spi
java.util.stream
java.util.zip
javax.crypto
javax.crypto.interfaces
javax.crypto.spec
javax.security.auth
javax.security.auth.callback
javax.security.auth.login
javax.security.auth.spi
javax.security.auth.x500
jdk
jdk.internal.org.objectweb.asm
jdk.internal.org.xml.sax
jdk.internal.util.xml
jdk.internal.util.xml.impl
jdk.jigsaw.module
jdk.jigsaw.tools.jlink
jdk.joptsimple
jdk.joptsimple.internal
jdk.joptsimple.util
sun.invoke
sun.invoke.anon
sun.invoke.empty
sun.invoke.util
sun.launcher
sun.launcher.resources
sun.misc
sun.misc.resources
sun.net
sun.net.ftp
sun.net.ftp.impl
sun.net.idn
sun.net.sdp
sun.net.smtp
sun.net.spi
sun.net.spi.nameservice
sun.net.util
sun.net.www
sun.net.www.content.text
sun.net.www.http
sun.net.www.protocol.file
sun.net.www.protocol.ftp
sun.net.www.protocol.http
sun.net.www.protocol.jar
sun.net.www.protocol.mailto
sun.net.www.protocol.netdoc
sun.nio
sun.nio.ch
sun.nio.cs
sun.nio.fs
sun.reflect
sun.reflect.annotation
sun.reflect.generics.factory
sun.reflect.generics.parser
sun.reflect.generics.scope
sun.reflect.generics.tree
sun.reflect.generics.visitor
sun.reflect.misc
sun.security.action
sun.security.jca
sun.security.pkcs
sun.security.pkcs12
sun.security.provider
sun.security.provider.certpath
sun.security.rsa
sun.security.timestamp
sun.security.util
sun.security.validator
sun.security.x509
sun.text
sun.text.bidi
sun.text.normalizer
sun.text.resources
sun.text.resources.cldr
sun.text.resources.cldr.en
sun.text.resources.en
sun.util
sun.util.calendar
sun.util.cldr
sun.util.locale
sun.util.locale.provider
sun.util.logging
sun.util.logging.resources
sun.util.resources
sun.util.resources.cldr
sun.util.resources.cldr.en
sun.util.resources.en
sun.util.spi
```

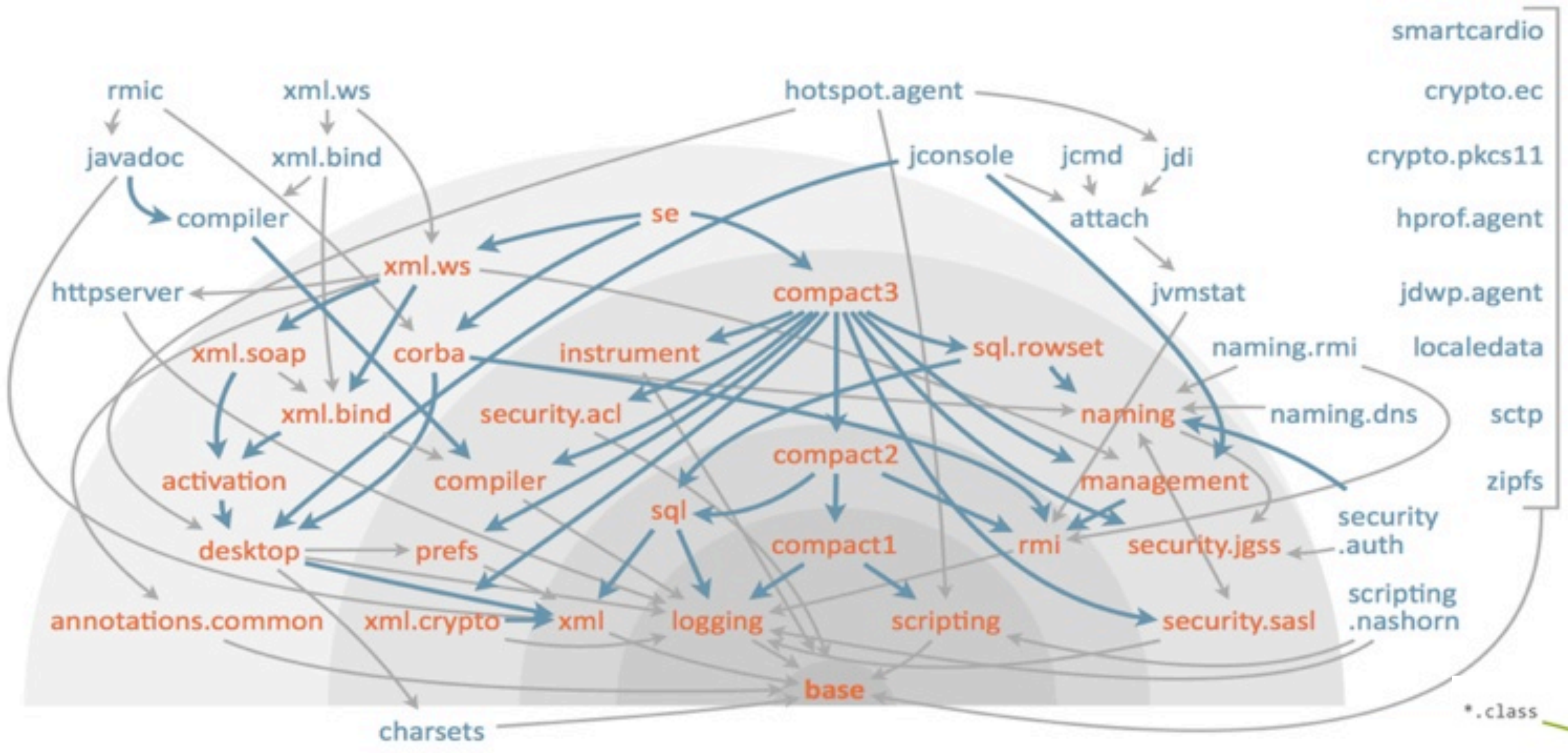
sun.*
.internal.

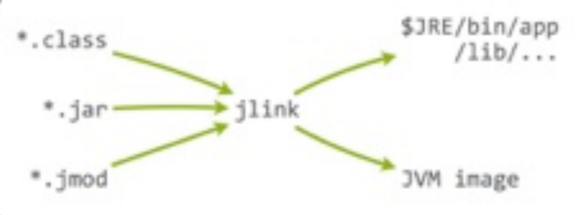
The starting point…



Jigsaw

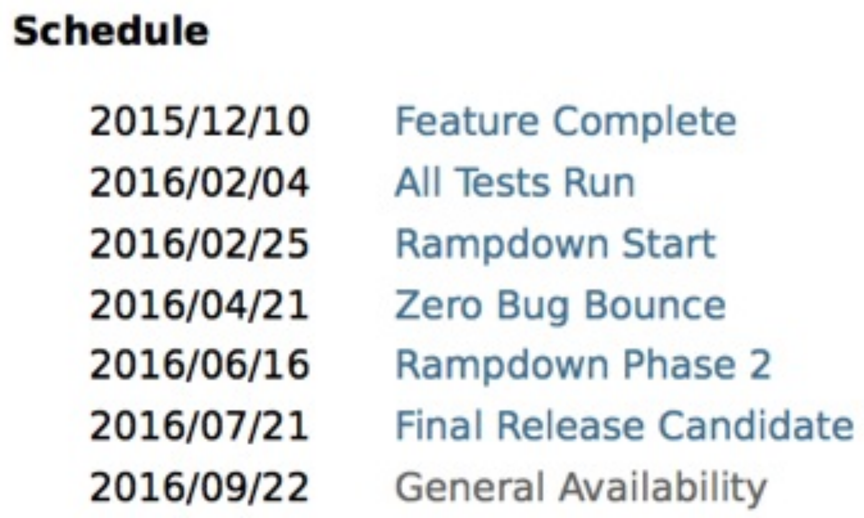
(see MR https://www.parleys.com/tutorial/java-9-make-way-modules)





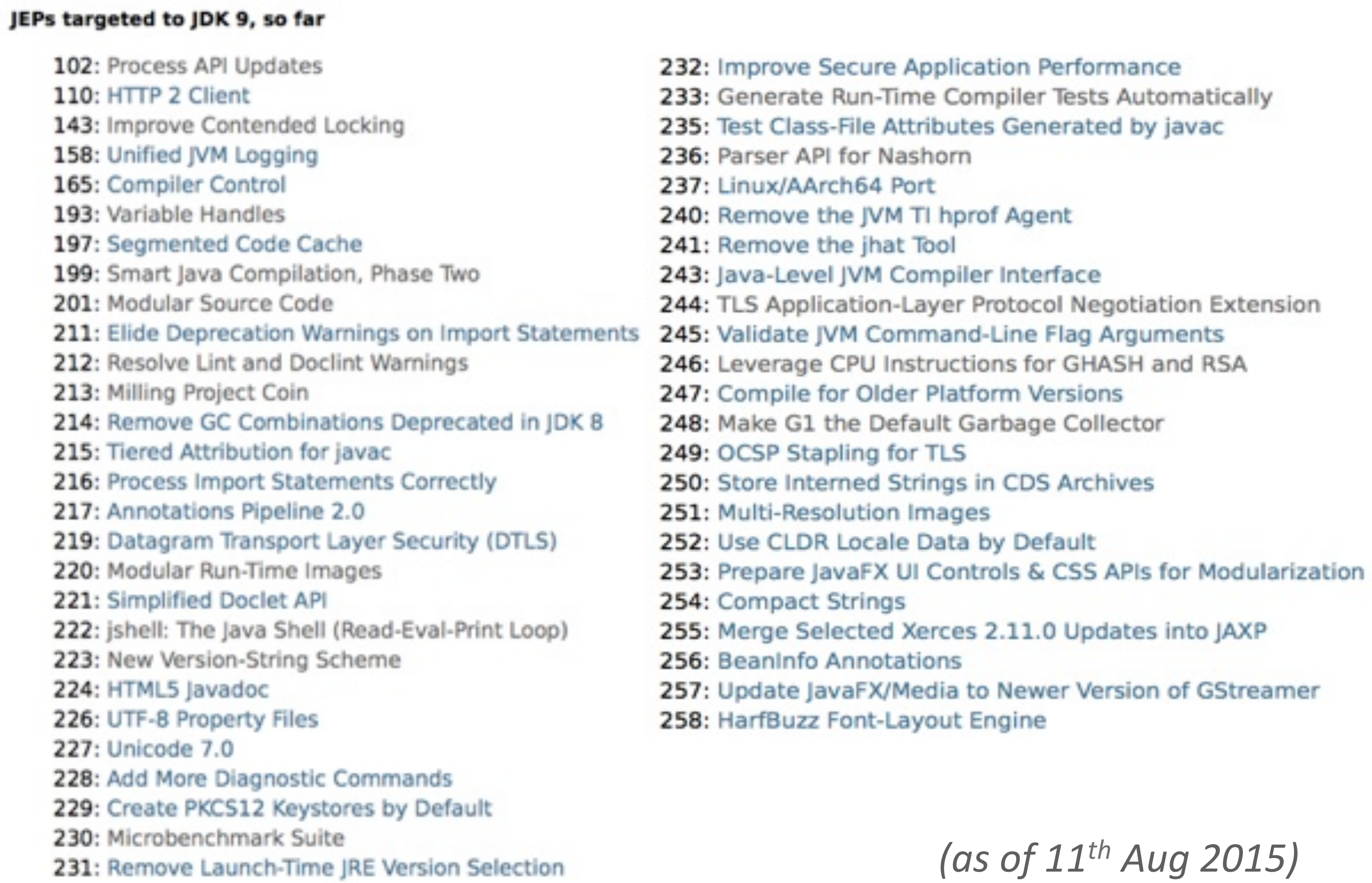
Java 9

http://openjdk.java.net/projects/jdk9



Schedule

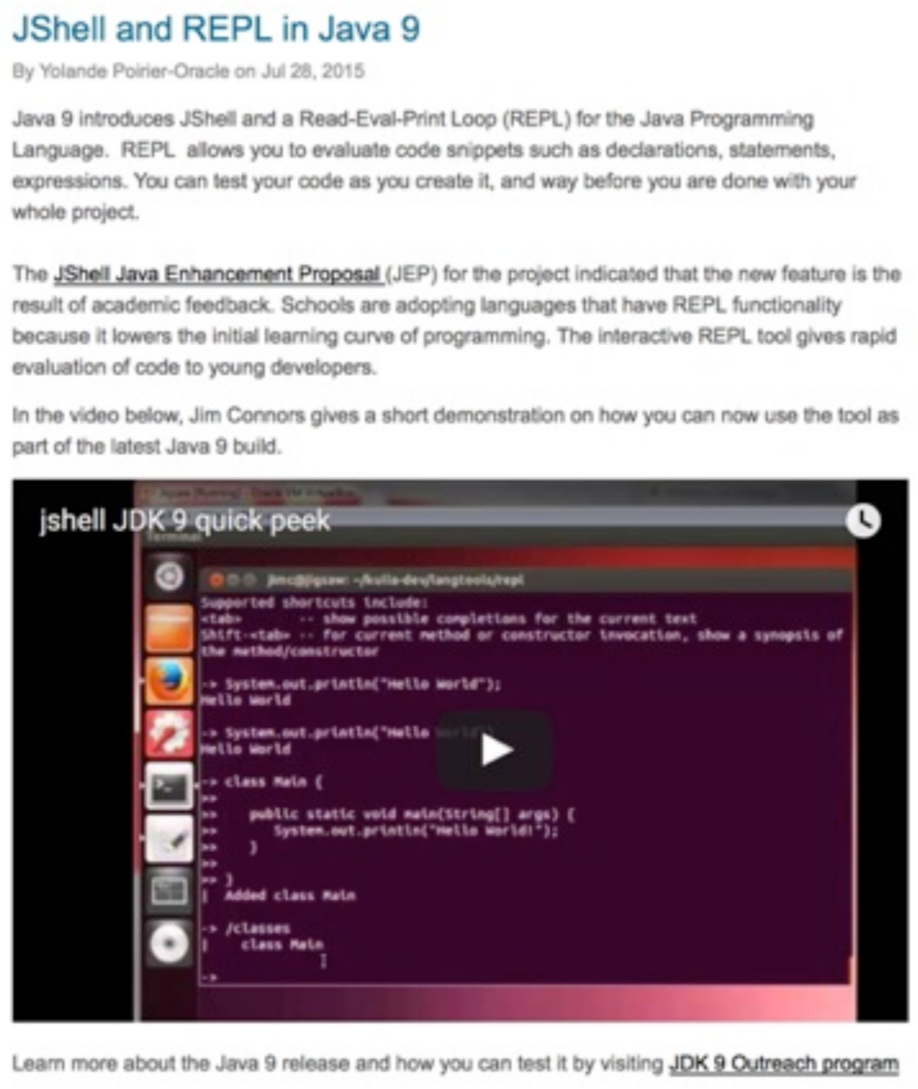
2015/12/10	Feature Complete
2016/02/04	All Tests Run
2016/02/25	Rampdown Start
2016/04/21	Zero Bug Bounce
2016/06/16	Rampdown Phase 2
2016/07/21	Final Release Candidate
2016/09/22	General Availability



JEPs targeted to JDK 9, so far

102: Process API Updates	**232**: Improve Secure Application Performance
110: HTTP 2 Client	**233**: Generate Run-Time Compiler Tests Automatically
143: Improve Contended Locking	**235**: Test Class-File Attributes Generated by javac
158: Unified JVM Logging	**236**: Parser API for Nashorn
165: Compiler Control	**237**: Linux/AArch64 Port
193: Variable Handles	**240**: Remove the JVM TI hprof Agent
197: Segmented Code Cache	**241**: Remove the jhat Tool
199: Smart Java Compilation, Phase Two	**243**: Java-Level JVM Compiler Interface
201: Modular Source Code	**244**: TLS Application-Layer Protocol Negotiation Extension
211: Elide Deprecation Warnings on Import Statements	**245**: Validate JVM Command-Line Flag Arguments
212: Resolve Lint and Doclint Warnings	**246**: Leverage CPU Instructions for GHASH and RSA
213: Milling Project Coin	**247**: Compile for Older Platform Versions
214: Remove GC Combinations Deprecated in JDK 8	**248**: Make G1 the Default Garbage Collector
215: Tiered Attribution for javac	**249**: OCSP Stapling for TLS
216: Process Import Statements Correctly	**250**: Store Interned Strings in CDS Archives
217: Annotations Pipeline 2.0	**251**: Multi-Resolution Images
219: Datagram Transport Layer Security (DTLS)	**252**: Use CLDR Locale Data by Default
220: Modular Run-Time Images	**253**: Prepare JavaFX UI Controls & CSS APIs for Modularization
221: Simplified Doclet API	**254**: Compact Strings
222: jshell: The Java Shell (Read-Eval-Print Loop)	**255**: Merge Selected Xerces 2.11.0 Updates into JAXP
223: New Version-String Scheme	**256**: BeanInfo Annotations
224: HTML5 Javadoc	**257**: Update JavaFX/Media to Newer Version of GStreamer
226: UTF-8 Property Files	**258**: HarfBuzz Font-Layout Engine
227: Unicode 7.0	
228: Add More Diagnostic Commands	
229: Create PKCS12 Keystores by Default	
230: Microbenchmark Suite	
231: Remove Launch-Time JRE Version Selection	

(as of 11th Aug 2015)



JShell and REPL in Java 9

By Yolande Poirier-Oracle on Jul 28, 2015

Java 9 introduces JShell and a Read-Eval-Print Loop (REPL) for the Java Programming Language. REPL allows you to evaluate code snippets such as declarations, statements, expressions. You can test your code as you create it, and way before you are done with your whole project.

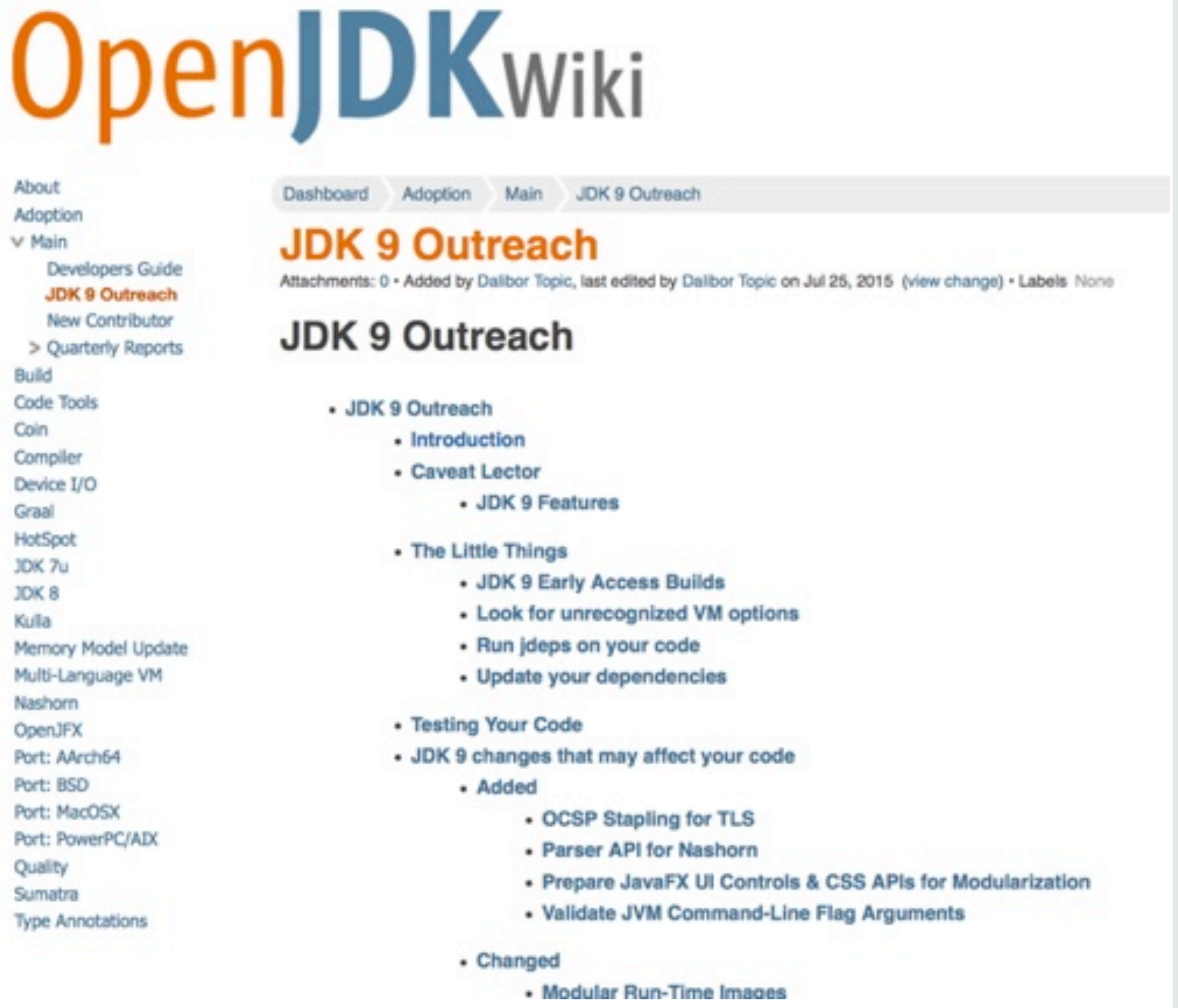
The JShell Java Enhancement Proposal (JEP) for the project indicated that the new feature is the result of academic feedback. Schools are adopting languages that have REPL functionality because it lowers the initial learning curve of programming. The interactive REPL tool gives rapid evaluation of code to young developers.

In the video below, Jim Connors gives a short demonstration on how you can now use the tool as part of the latest Java 9 build.

```
Supported shortcuts include:
<tab>        -- show possible completions for the current text
Shift-<tab> -- for current method or constructor invocation, show a synopsis of
the method/constructor

-> System.out.println("Hello World");
Hello World

-> System.out.println("Hello World");
Hello World

-> class Main {
>>
>>    public static void main(String[] args) {
>>       System.out.println("Hello World!");
>>    }
>>
>> }
|  Added class Main

-> /classes
|    class Main

->
```

Learn more about the Java 9 release and how you can test it by visiting JDK 9 Outreach program

https://blogs.oracle.com/java/entry/jshell_and_relp_in_java
http://www.infoq.com/articles/Java9-New-HTTP-2-and-REPL



OpenJDK Wiki

- About
- Adoption
- Main
  - Developers Guide
  - JDK 9 Outreach
  - New Contributor
  - Quarterly Reports
- Build
- Code Tools
- Coin
- Compiler
- Device I/O
- Graal
- HotSpot
- JDK 7u
- JDK 8
- Kulla
- Memory Model Update
- Multi-Language VM
- Nashorn
- OpenJFX
- Port: AArch64
- Port: BSD
- Port: MacOSX
- Port: PowerPC/AIX
- Quality
- Sumatra
- Type Annotations

Dashboard > Adoption > Main > JDK 9 Outreach

JDK 9 Outreach

Attachments: 0 • Added by Dalibor Topic, last edited by Dalibor Topic on Jul 25, 2015 (view change) • Labels None

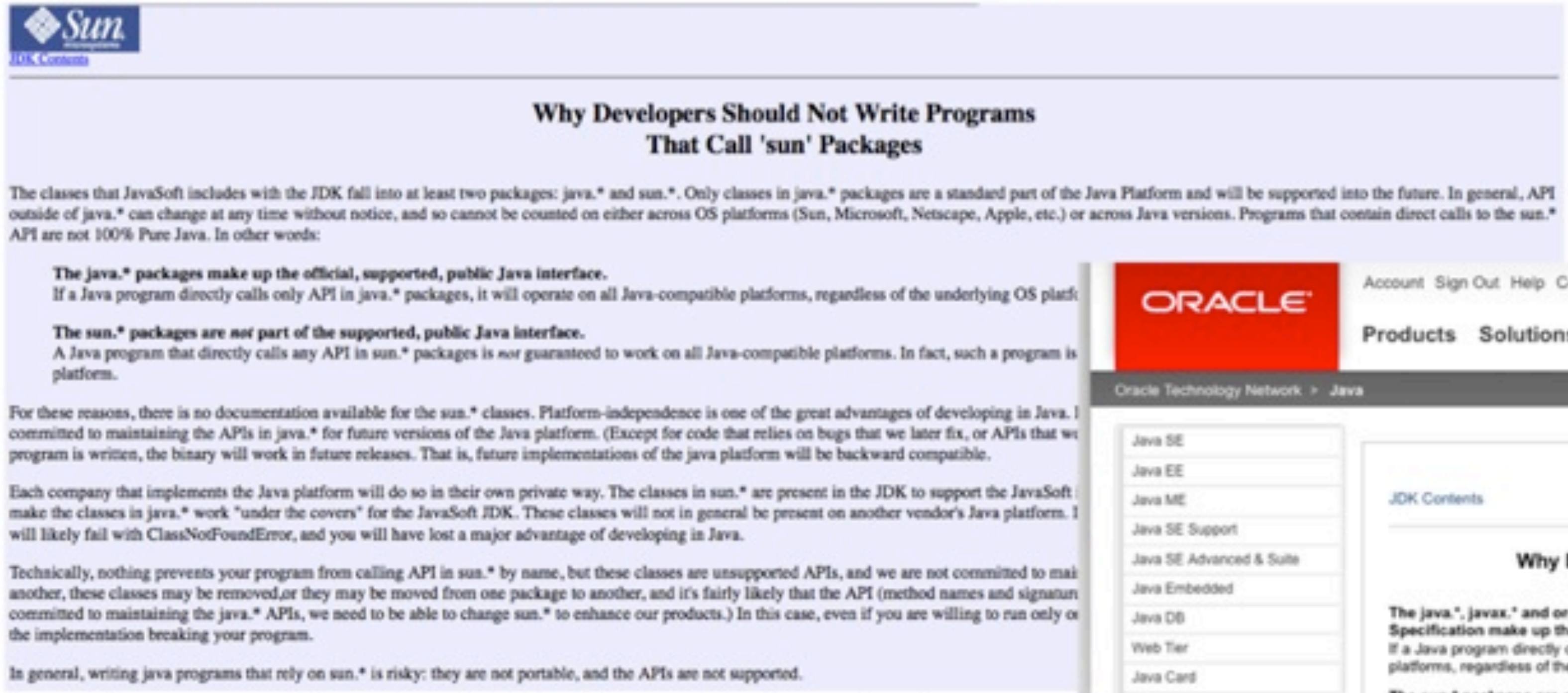
JDK 9 Outreach

- JDK 9 Outreach
  - Introduction
  - Caveat Lector
    - JDK 9 Features
  - The Little Things
    - JDK 9 Early Access Builds
    - Look for unrecognized VM options
    - Run jdeps on your code
    - Update your dependencies
  - Testing Your Code
  - JDK 9 changes that may affect your code
    - Added
      - OCSP Stapling for TLS
      - Parser API for Nashorn
      - Prepare JavaFX UI Controls & CSS APIs for Modularization
      - Validate JVM Command-Line Flag Arguments
    - Changed
      - Modular Run-Time Images

https://wiki.openjdk.java.net/display/Adoption/JDK+9+Outreach

Some APIs were never supposed to be used…

Warnings posted from Feb 1998 to today



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 platf
>
> **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
> platform.

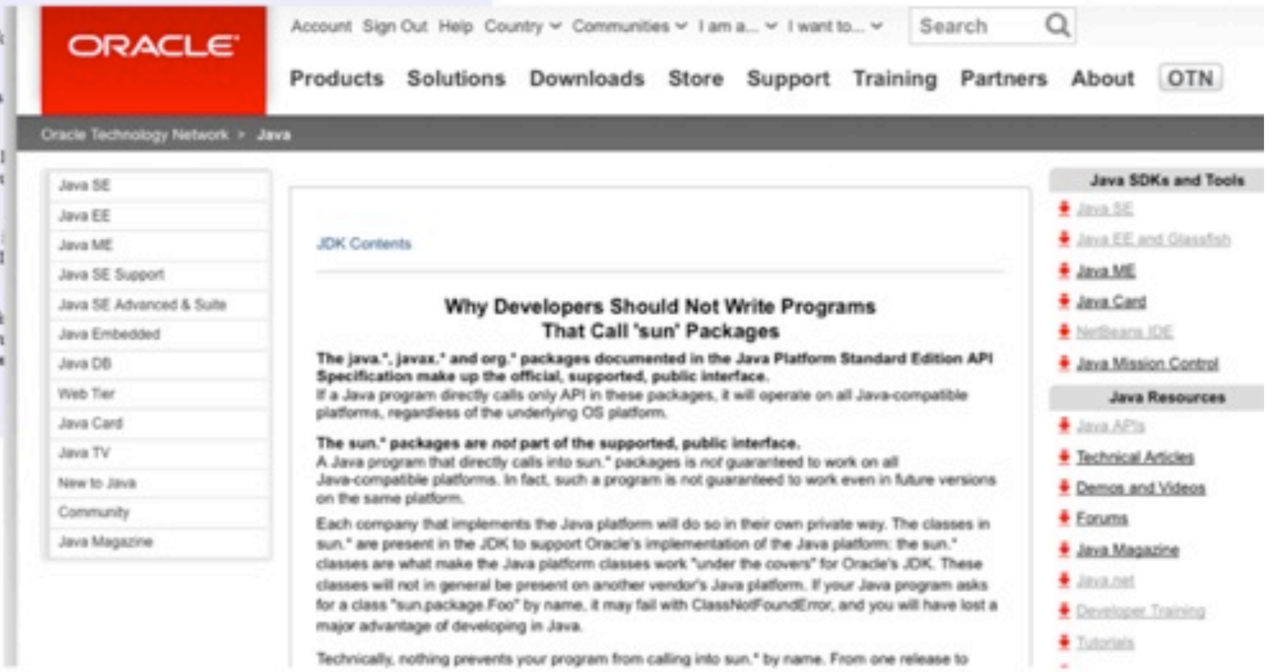
For these reasons, there is no documentation available for the sun.* classes. Platform-independence is one of the great advantages of developing in Java. I
committed to maintaining the APIs in java.* for future versions of the Java platform. (Except for code that relies on bugs that we later fix, or APIs that w
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
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. I
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 mai
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 signatur
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 o
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.

1998 Coolest phone
Nokia 5110



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

The java.*, javax.* and org.* packages documented in the Java Platform Standard Edition API Specification make up the official, supported, public interface.
If a Java program directly calls only API in these 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 interface.
A Java program that directly calls into 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.

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 Oracle's implementation of the Java platform: the sun.* classes are what make the Java platform classes work "under the covers" for Oracle's 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 may fail with ClassNotFoundError, and you will have lost a major advantage of developing in Java.

Technically, nothing prevents your program from calling into sun.* by name. From one release to

http://java.sun.com/products/jdk/faq/faq-sun-packages.html
(no longer valid, but available on some archive sites)

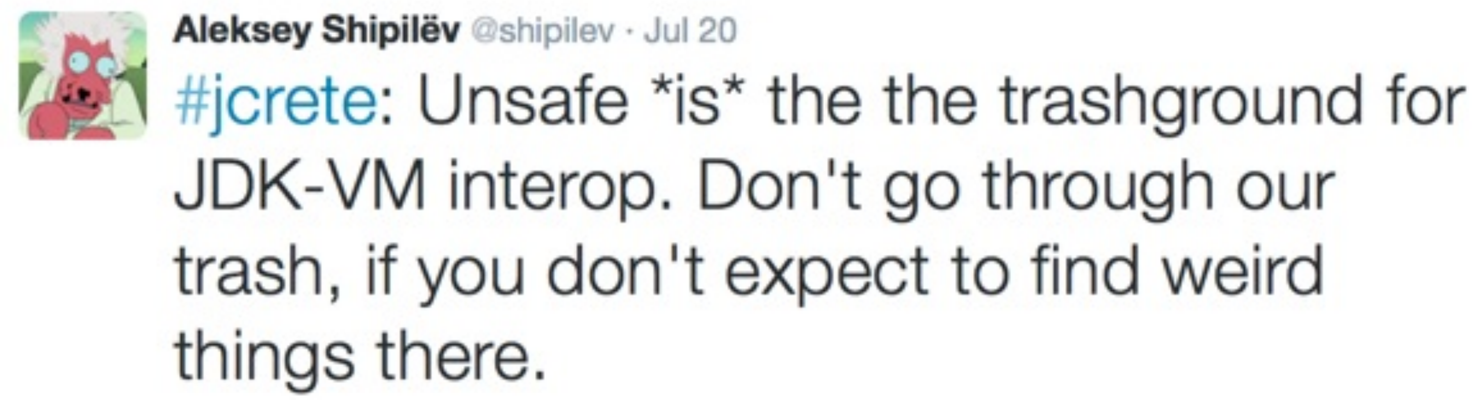
http://www.oracle.com/technetwork/java/faq-sun-packages-142232.html

What to do if you think you are using internal APIs

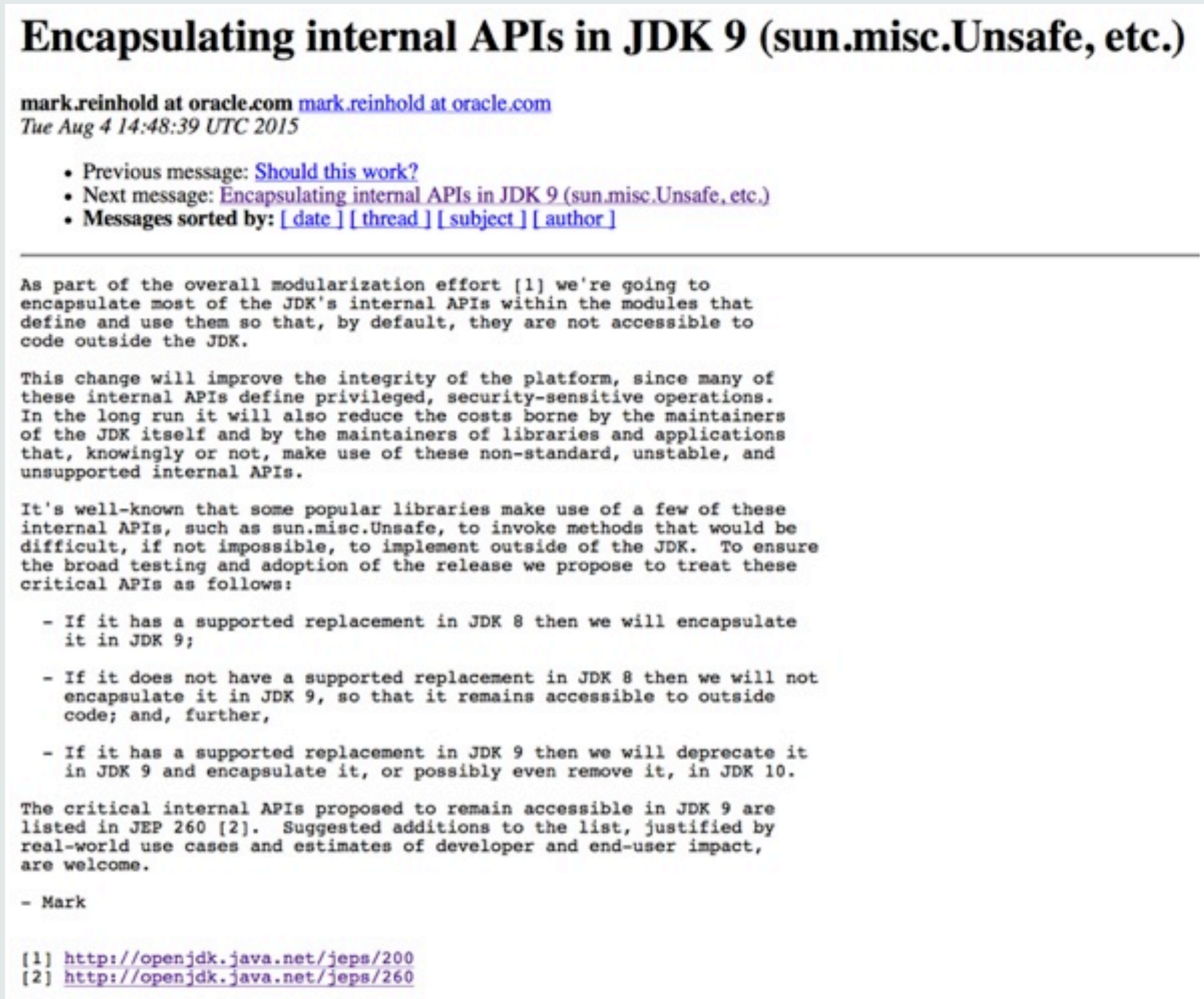
- For your own code
  - Use JDeps, available on JDK 8, to scan your programs/libraries for problems
    - When possible JDeps will propose alternative APIs
- For Third Party Programs and Libraries
  - You can run JDeps on the bytecode so you don't need the source code
  - Point the vendor to the many articles warning of the need to remove dependencies on this; ask your vendor to confirm if they are ready for JDK 9
  - Search for alternative programs / libraries
- If unable to move off private APIs
  - Plan to keep JDK/JRE 8 for those programs until you can find a replacement

For JDeps introduction and explanation search for:
Closing the closed APIs

```
$ jdeps –jdkinternals app.jar
```


Aleksey Shipilëv @shipilev · Jul 20

#jcrete: Unsafe *is* the the trashground for JDK-VM interop. Don't go through our trash, if you don't expect to find weird things there.

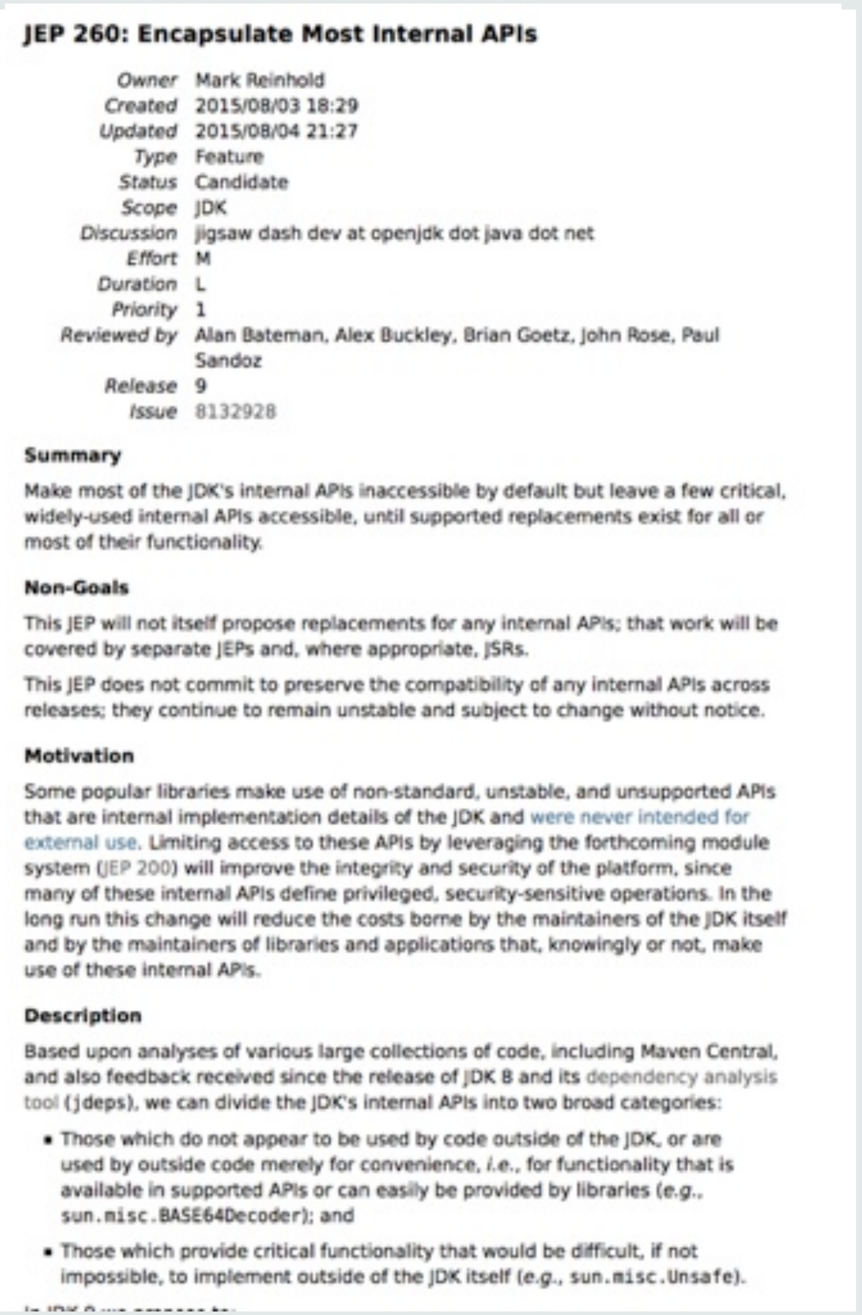


Encapsulating internal APIs in JDK 9 (sun.misc.Unsafe, etc.)

mark.reinhold at oracle.com mark.reinhold at oracle.com
Tue Aug 4 14:48:39 UTC 2015

- Previous message: Should this work?
- Next message: Encapsulating internal APIs in JDK 9 (sun.misc.Unsafe, etc.)
- **Messages sorted by:** [date] [thread] [subject] [author]

```
As part of the overall modularization effort [1] we're going to
encapsulate most of the JDK's internal APIs within the modules that
define and use them so that, by default, they are not accessible to
code outside the JDK.

This change will improve the integrity of the platform, since many of
these internal APIs define privileged, security-sensitive operations.
In the long run it will also reduce the costs borne by the maintainers
of the JDK itself and by the maintainers of libraries and applications
that, knowingly or not, make use of these non-standard, unstable, and
unsupported internal APIs.

It's well-known that some popular libraries make use of a few of these
internal APIs, such as sun.misc.Unsafe, to invoke methods that would be
difficult, if not impossible, to implement outside of the JDK.  To ensure
the broad testing and adoption of the release we propose to treat these
critical APIs as follows:

  - If it has a supported replacement in JDK 8 then we will encapsulate
    it in JDK 9;

  - If it does not have a supported replacement in JDK 8 then we will not
    encapsulate it in JDK 9, so that it remains accessible to outside
    code; and, further,

  - If it has a supported replacement in JDK 9 then we will deprecate it
    in JDK 9 and encapsulate it, or possibly even remove it, in JDK 10.

The critical internal APIs proposed to remain accessible in JDK 9 are
listed in JEP 260 [2].  Suggested additions to the list, justified by
real-world use cases and estimates of developer and end-user impact,
are welcome.

- Mark

[1] http://openjdk.java.net/jeps/200
[2] http://openjdk.java.net/jeps/260
```



JEP 260: Encapsulate Most Internal APIs

Owner	Mark Reinhold
Created	2015/08/03 18:29
Updated	2015/08/04 21:27
Type	Feature
Status	Candidate
Scope	JDK
Discussion	jigsaw dash dev at openjdk dot java dot net
Effort	M
Duration	L
Priority	1
Reviewed by	Alan Bateman, Alex Buckley, Brian Goetz, John Rose, Paul Sandoz
Release	9
Issue	8132928

Summary

Make most of the JDK's internal APIs inaccessible by default but leave a few critical, widely-used internal APIs accessible, until supported replacements exist for all or most of their functionality.

Non-Goals

This JEP will not itself propose replacements for any internal APIs; that work will be covered by separate JEPs and, where appropriate, JSRs.

This JEP does not commit to preserve the compatibility of any internal APIs across releases; they continue to remain unstable and subject to change without notice.

Motivation

Some popular libraries make use of non-standard, unstable, and unsupported APIs that are internal implementation details of the JDK and were never intended for external use. Limiting access to these APIs by leveraging the forthcoming module system (JEP 200) will improve the integrity and security of the platform, since many of these internal APIs define privileged, security-sensitive operations. In the long run this change will reduce the costs borne by the maintainers of the JDK itself and by the maintainers of libraries and applications that, knowingly or not, make use of these internal APIs.

Description

Based upon analyses of various large collections of code, including Maven Central, and also feedback received since the release of JDK 8 and its dependency analysis tool (`jdeps`), we can divide the JDK's internal APIs into two broad categories:

- Those which do not appear to be used by code outside of the JDK, or are used by outside code merely for convenience, *i.e.*, for functionality that is available in supported APIs or can easily be provided by libraries (*e.g.*, `sun.misc.BASE64Decoder`); and
- Those which provide critical functionality that would be difficult, if not impossible, to implement outside of the JDK itself (*e.g.*, `sun.misc.Unsafe`).

In JDK 9 we propose to:

JEP 260 *(excerpt)*

In JDK 9 we propose to:

- Encapsulate all non-critical internal APIs by default:
  The modules that define them will not export their packages for outside use.
  (Access to such APIs will be available, as a last resort, via a command-line flag at both compile time and run time, unless those APIs are revised or removed for other reasons.)
- Encapsulate critical internal APIs for which supported replacements exist in JDK 8, in the same manner and with the same last-resort workaround.
  (A supported replacement is one that is either part of the Java SE 8 standard (i.e., in a java.* or javax.* package) or else JDK-specific and annotated with @jdk.Exported (typically in a com.sun.* or jdk.* package).)
- Not encapsulate critical internal APIs for which supported replacements do not exist in JDK 8 and, further, deprecate those which have supported replacements in JDK 9 with the intent to encapsulate them, or possibly even remove them, in JDK 10.

The critical internal APIs proposed to remain accessible in JDK 9 are:

sun.misc.Cleaner
sun.misc.{Signal,SignalHandler}
sun.misc.Unsafe (The functionality of many of the methods in this class is now available via variable handles (JEP 193).)
sun.reflect.Reflection::getCallerClass (The functionality of this method may be provided in a standard form via JEP 259.)
sun.reflect.ReflectionFactory

Suggested additions to this list, justified by real-world use cases and estimates of developer and end-user impact, are welcome.

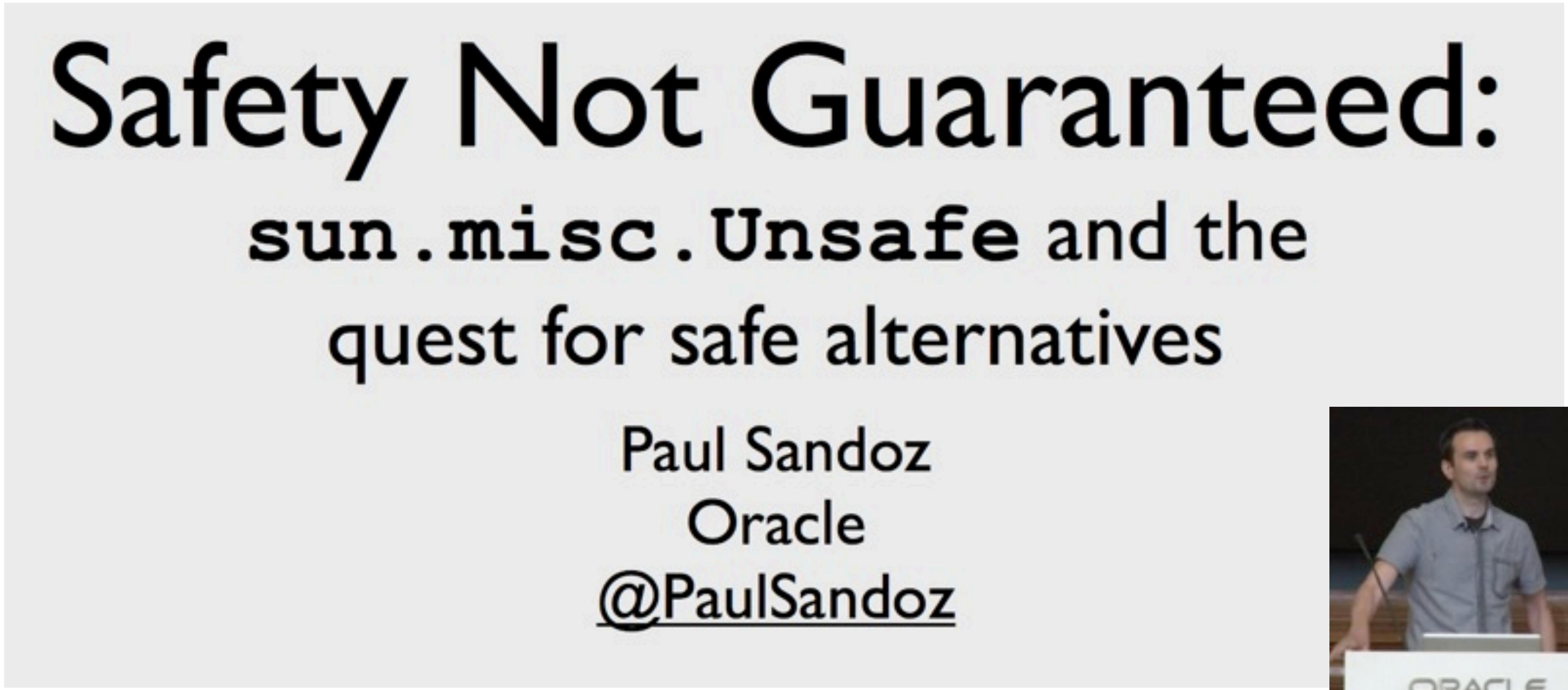
** Update from JVM Language Summit 2015

(video from August 11th 2015)
https://www.youtube.com/watch?v=4HG0YQVy8UM

The Secret History & Tragic Fate of sun.misc.Unsafe

Mark Reinhold (@mreinhold)
Chief Architect, Java Platform Group
Oracle

JVMLS
2015/8/11



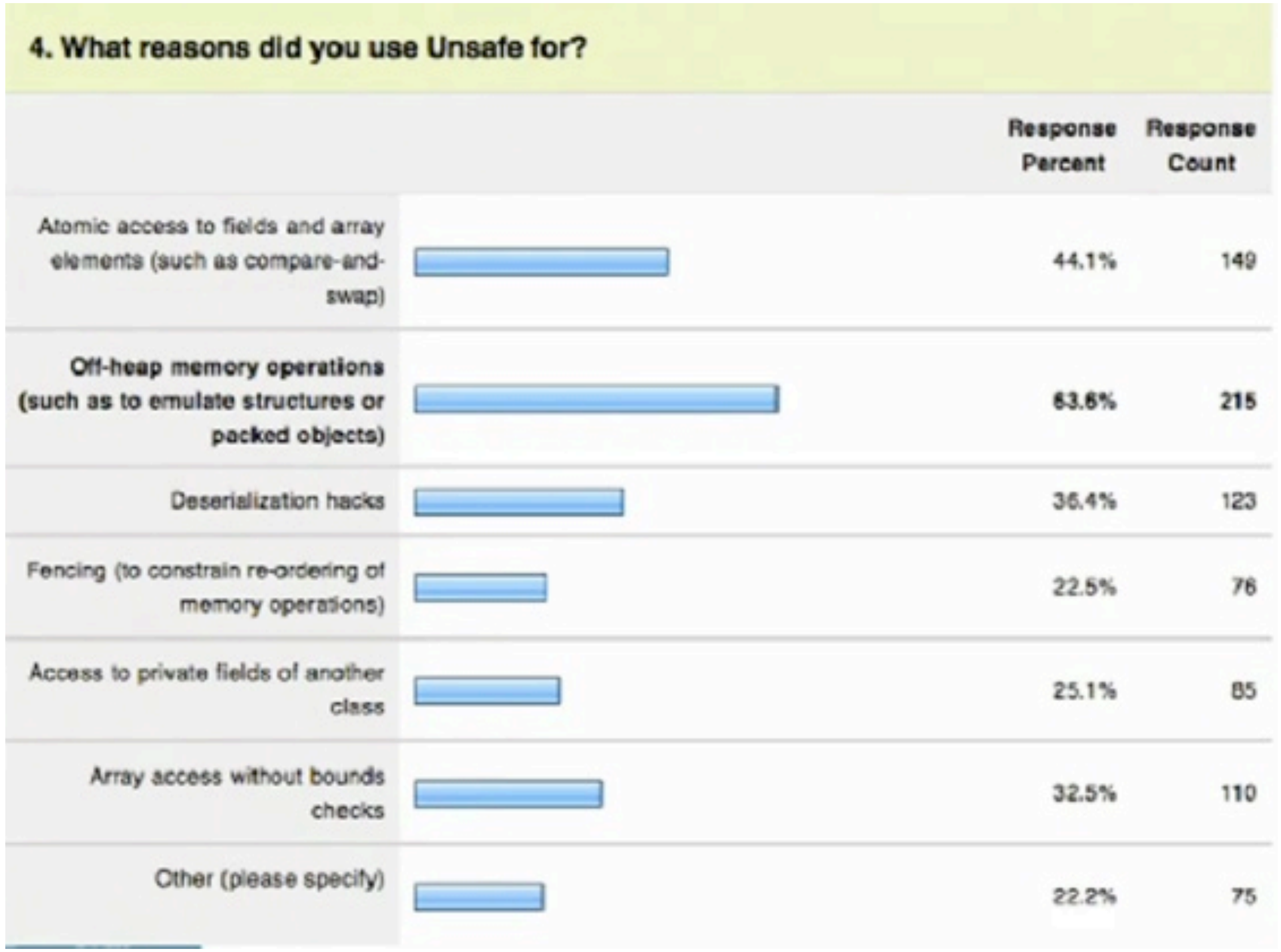
Safety Not Guaranteed:

`sun.misc.Unsafe` and the quest for safe alternatives

Paul Sandoz
Oracle
@PaulSandoz

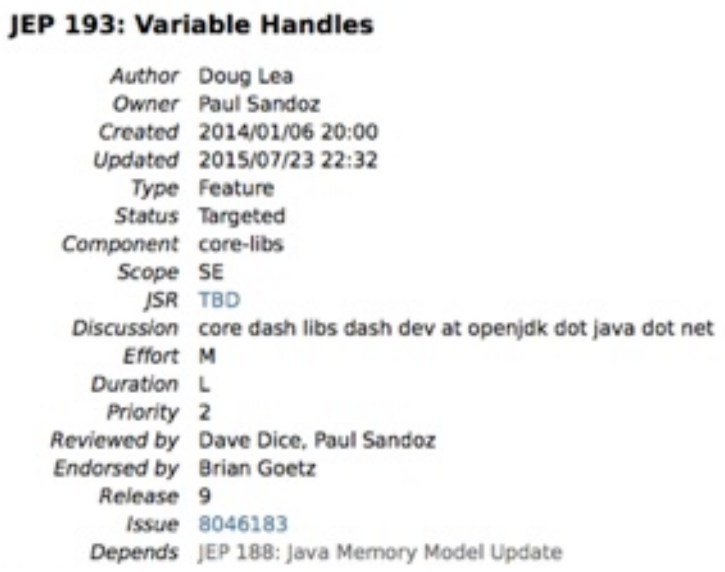
https://oracleus.activeevents.com/2014/connect/sessionDetail.ww?SESSION_ID=5150
And
http://download.oracle.com/technetwork/java/javase/community/JVMLS2014/JVMLS2014-10-Sandoz_H.264.mov

Survey from January 2014



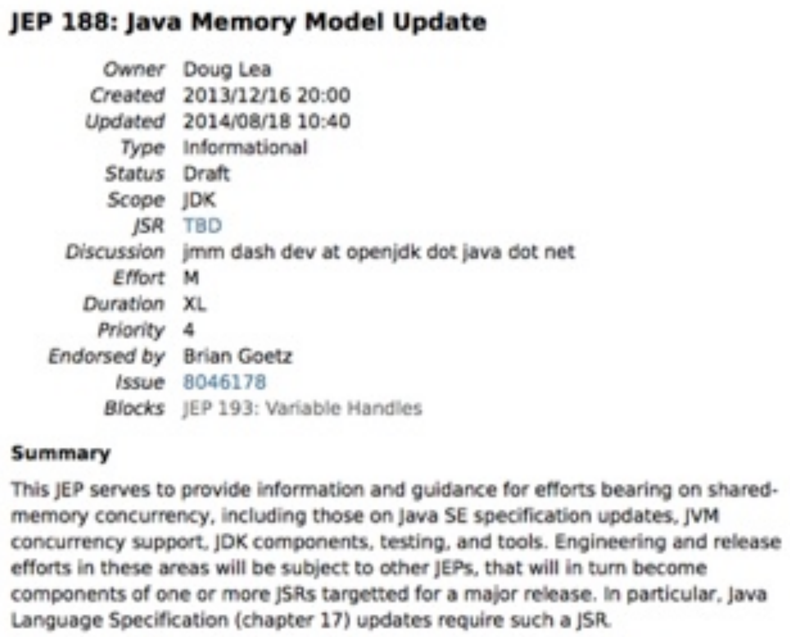
4. What reasons did you use Unsafe for?

	Response Percent	Response Count
Atomic access to fields and array elements (such as compare-and-swap)	44.1%	149
Off-heap memory operations (such as to emulate structures or packed objects)	**63.6%**	**215**
Deserialization hacks	36.4%	123
Fencing (to constrain re-ordering of memory operations)	22.5%	76
Access to private fields of another class	25.1%	85
Array access without bounds checks	32.5%	110
Other (please specify)	22.2%	75



JEP 193: Variable Handles

Author	Doug Lea
Owner	Paul Sandoz
Created	2014/01/06 20:00
Updated	2015/07/23 22:32
Type	Feature
Status	Targeted
Component	core-libs
Scope	SE
JSR	TBD
Discussion	core dash libs dash dev at openjdk dot java dot net
Effort	M
Duration	L
Priority	2
Reviewed by	Dave Dice, Paul Sandoz
Endorsed by	Brian Goetz
Release	9
Issue	8046183
Depends	JEP 188: Java Memory Model Update

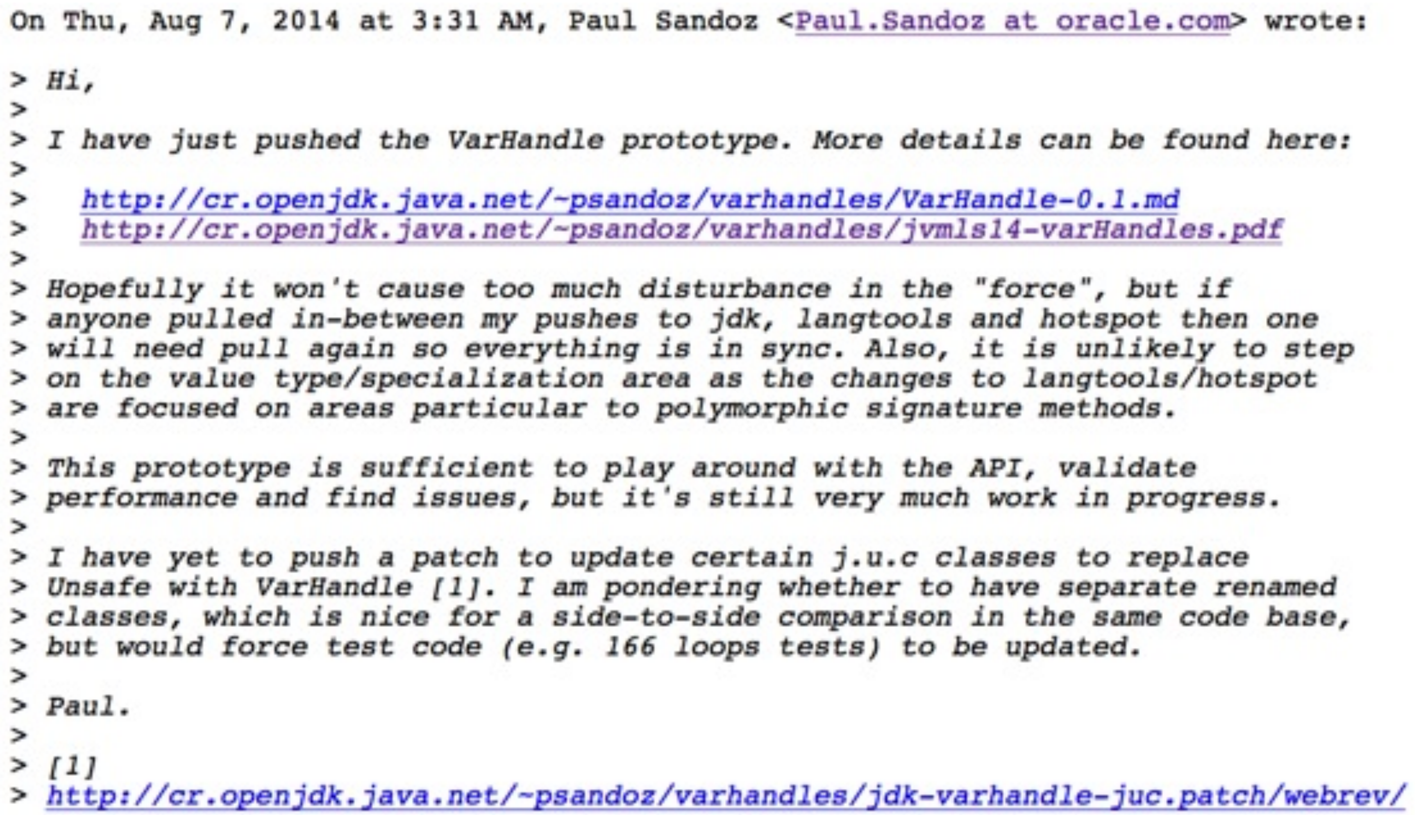


JEP 188: Java Memory Model Update

Owner	Doug Lea
Created	2013/12/16 20:00
Updated	2014/08/18 10:40
Type	Informational
Status	Draft
Scope	JDK
JSR	TBD
Discussion	jmm dash dev at openjdk dot java dot net
Effort	M
Duration	XL
Priority	4
Endorsed by	Brian Goetz
Issue	8046178
Blocks	JEP 193: Variable Handles

Summary

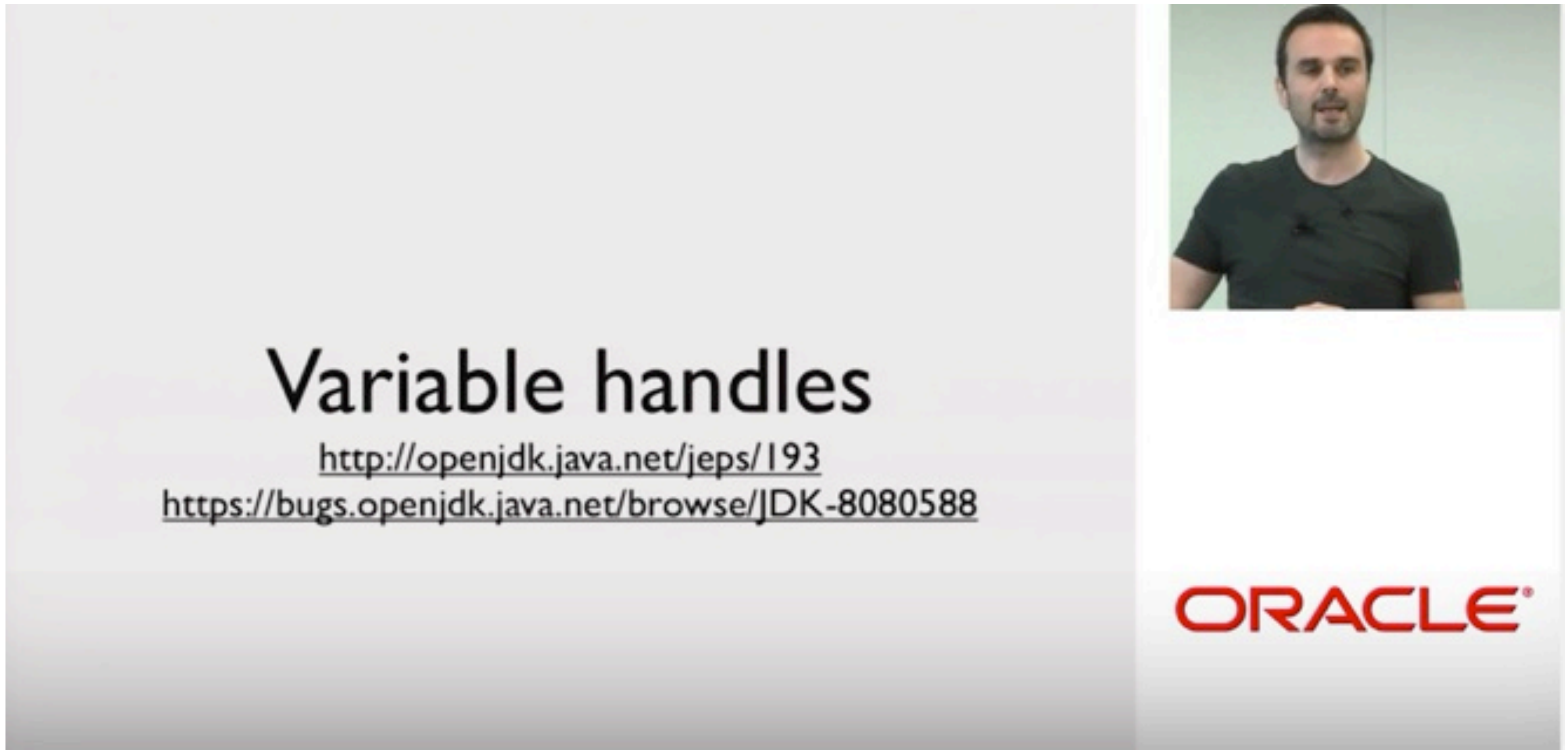
This JEP serves to provide information and guidance for efforts bearing on shared-memory concurrency, including those on Java SE specification updates, JVM concurrency support, JDK components, testing, and tools. Engineering and release efforts in these areas will be subject to other JEPs, that will in turn become components of one or more JSRs targetted for a major release. In particular, Java Language Specification (chapter 17) updates require such a JSR.



```
On Thu, Aug 7, 2014 at 3:31 AM, Paul Sandoz <Paul.Sandoz at oracle.com> wrote:

> Hi,
>
> I have just pushed the VarHandle prototype. More details can be found here:
>
>   http://cr.openjdk.java.net/~psandoz/varhandles/VarHandle-0.1.md
>   http://cr.openjdk.java.net/~psandoz/varhandles/jvmls14-varHandles.pdf
>
> Hopefully it won't cause too much disturbance in the "force", but if
> anyone pulled in-between my pushes to jdk, langtools and hotspot then one
> will need pull again so everything is in sync. Also, it is unlikely to step
> on the value type/specialization area as the changes to langtools/hotspot
> are focused on areas particular to polymorphic signature methods.
>
> This prototype is sufficient to play around with the API, validate
> performance and find issues, but it's still very much work in progress.
>
> I have yet to push a patch to update certain j.u.c classes to replace
> Unsafe with VarHandle [1]. I am pondering whether to have separate renamed
> classes, which is nice for a side-to-side comparison in the same code base,
> but would force test code (e.g. 166 loops tests) to be updated.
>
> Paul.
>
> [1]
> http://cr.openjdk.java.net/~psandoz/varhandles/jdk-varhandle-juc.patch/webrev/
```

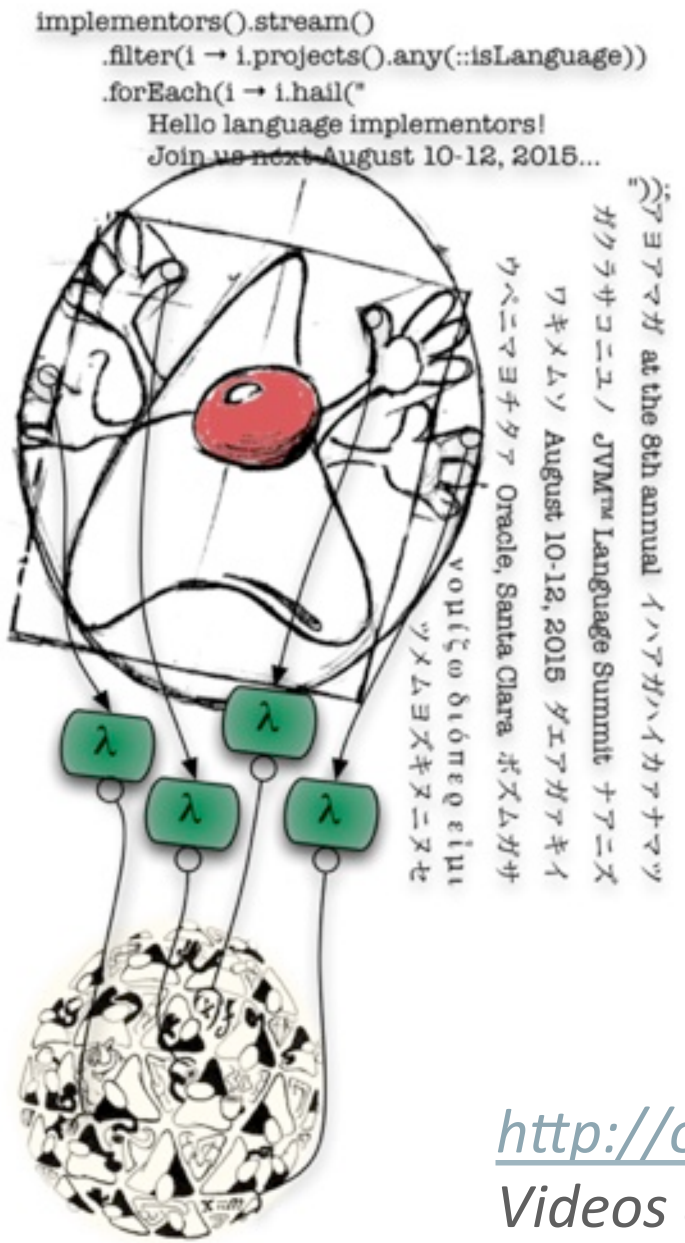
** Update from JVM Language Summit 2015

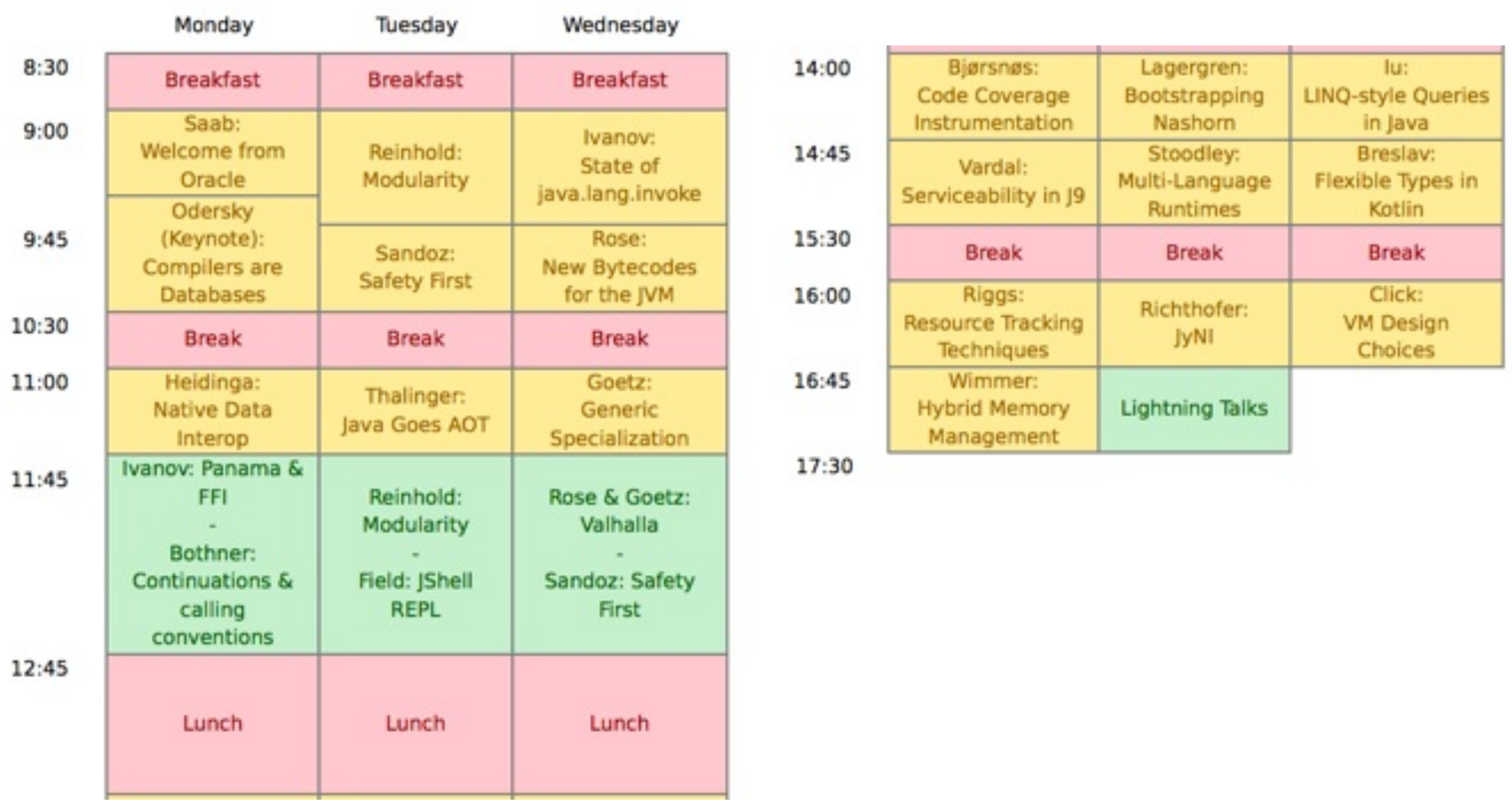
(video from August 11th 2015)
https://www.youtube.com/watch?v=ycKn18LtNtk



Variable handles

http://openjdk.java.net/jeps/193
https://bugs.openjdk.java.net/browse/JDK-8080588


implementors().stream()
.filter(i → i.projects().any(::isLanguage))
.forEach(i → i.hail("
Hello language implementors!
Join us next August 10-12, 2015...
");
アヨアマガ at the 8th annual イハアガハイカァナマツ
ガクラサコニユノ JVM™ Language Summit ナアニズ
ワキメムソ August 10-12, 2015 ダエアガァキイ
ウペニマヨチタア Oracle, Santa Clara ボズムガサ
νομίζω διόπερ εἰμι
ツメムヨズキヌニヌセ
λ



	Monday	Tuesday	Wednesday
8:30	Breakfast	Breakfast	Breakfast
9:00	Saab: Welcome from Oracle	Reinhold: Modularity	Ivanov: State of java.lang.invoke
9:45	Odersky (Keynote): Compilers are Databases	Sandoz: Safety First	Rose: New Bytecodes for the JVM
10:30	Break	Break	Break
11:00	Heidinga: Native Data Interop	Thalinger: Java Goes AOT	Goetz: Generic Specialization
11:45	Ivanov: Panama & FFI - Bothner: Continuations & calling conventions	Reinhold: Modularity - Field: JShell REPL	Rose & Goetz: Valhalla - Sandoz: Safety First
12:45	Lunch	Lunch	Lunch
14:00	Bjørsnøs: Code Coverage Instrumentation	Lagergren: Bootstrapping Nashorn	Iu: LINQ-style Queries in Java
14:45	Vardal: Serviceability in J9	Stoodley: Multi-Language Runtimes	Breslav: Flexible Types in Kotlin
15:30	Break	Break	Break
16:00	Riggs: Resource Tracking Techniques	Richthofer: JyNI	Click: VM Design Choices
16:45	Wimmer: Hybrid Memory Management	Lightning Talks	
17:30			

http://openjdk.java.net/projects/mlvm/jvmlangsummit

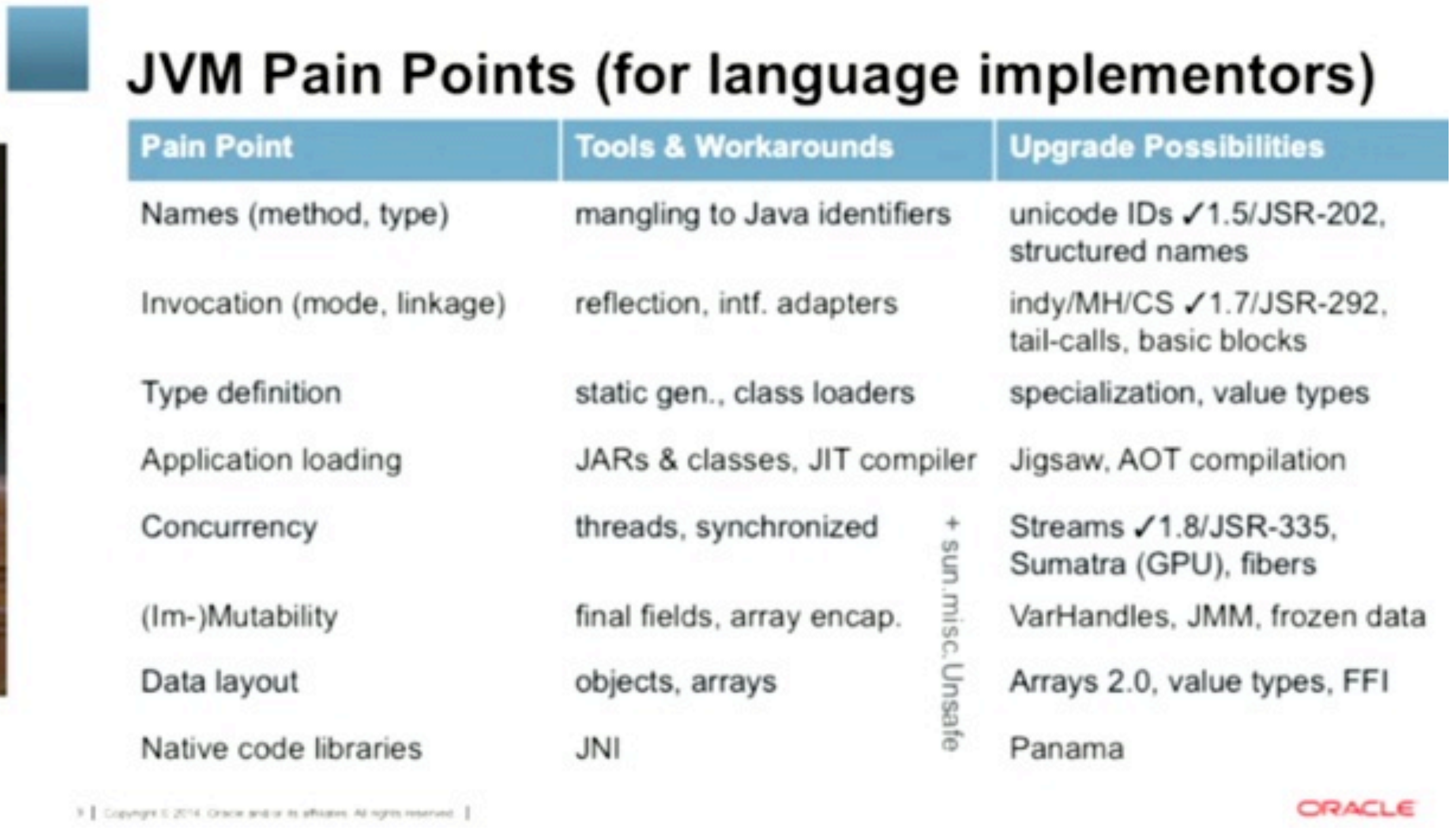
Videos are appearing now at https://www.youtube.com/user/java/videos

Some (possible) things to address going forwards

- Startup & Warmup time
- Memory overhead
- Optimizations for more specialized hardware
- Unpredictable latency due to GC
- Big Data (eg, the Hadoop ecosystem)
- Cloud & large multi-tenant deployments
- (More) JVM improvements for non-Java languages
- …

JVM language summit July 2014

http://www.oracle.com/technetwork/java/javase/community/jlssessions-2255337.html



JVM Pain Points (for language implementors)

Pain Point	Tools & Workarounds	Upgrade Possibilities
Names (method, type)	mangling to Java identifiers	unicode IDs ✓1.5/JSR-202, structured names
Invocation (mode, linkage)	reflection, intf. adapters	indy/MH/CS ✓1.7/JSR-292, tail-calls, basic blocks
Type definition	static gen., class loaders	specialization, value types
Application loading	JARs & classes, JIT compiler	Jigsaw, AOT compilation
Concurrency	threads, synchronized	Streams ✓1.8/JSR-335, Sumatra (GPU), fibers
(Im-)Mutability	final fields, array encap.	VarHandles, JMM, frozen data
Data layout	objects, arrays	Arrays 2.0, value types, FFI
Native code libraries	JNI	Panama

+ sun.misc.Unsafe

ORACLE

Object Identity / Pointers

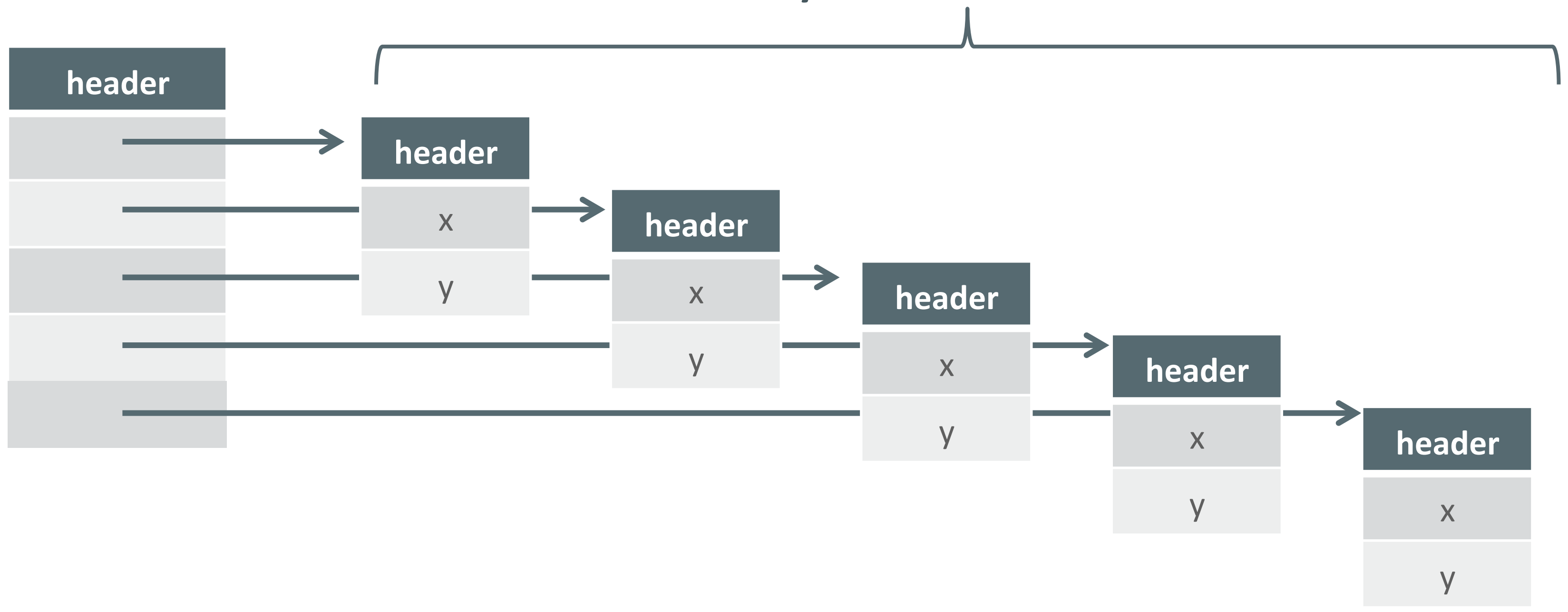
- Java's type system gives us:
  - Primitives (fixed set of primitive value types)
  - Arrays (homogeneous aggregation, <u>with identity</u>)
  - Objects (heterogeneous aggregation, <u>with identity</u>)
- Nice thing about primitive types
  - No identity
  - No Object Header
  - No indirection
  - Can store in registers
  - Can push on stack
- But… we can't make new ones

Data layout

What we have today

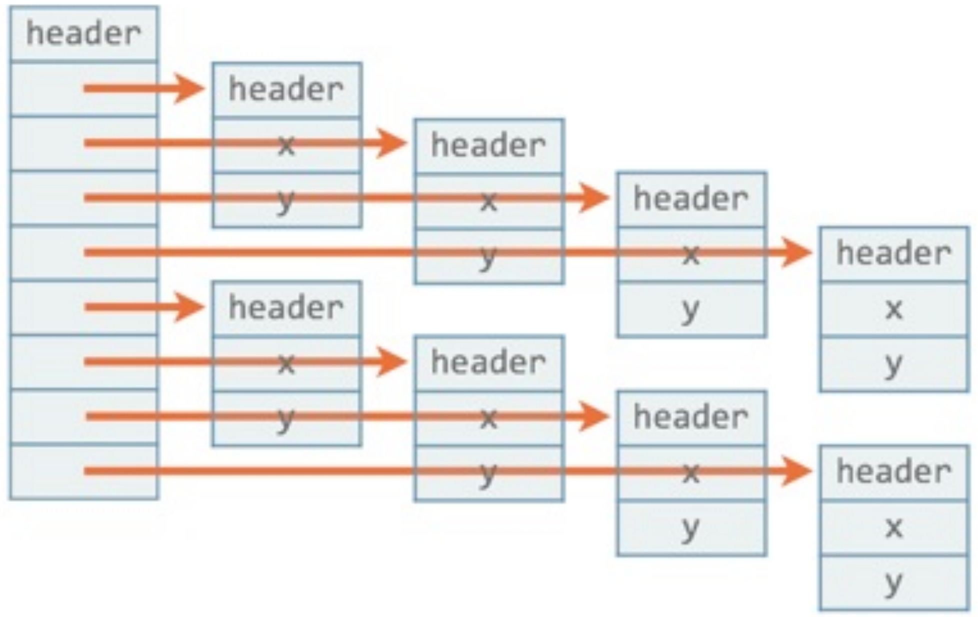
```
final class Point {
    final int x;
    final int y;
}

Point[] pts =
```

Layout of these in memory is effectively random!



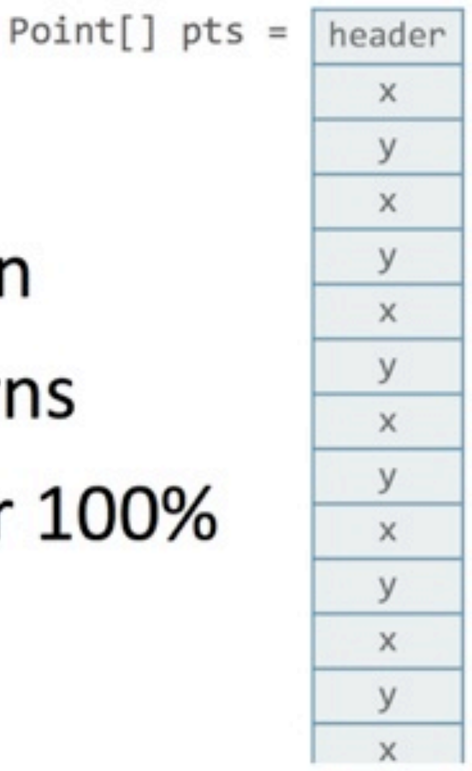
Each pointer is a gamble

- 20 years ago, a memory fetch and an add cost about the same
- Today, a cache miss can cost 1000 instruction cycles
- Hardware tries to cover latency with prefetch
- Prefetch works best with flat, regular layouts



Flat data: The better way

- Simple to write
- Simple to read
- Java class-based abstraction
- Predictable memory patterns
- Payload per cache line near 100%
  ⇒ better density



Value Types

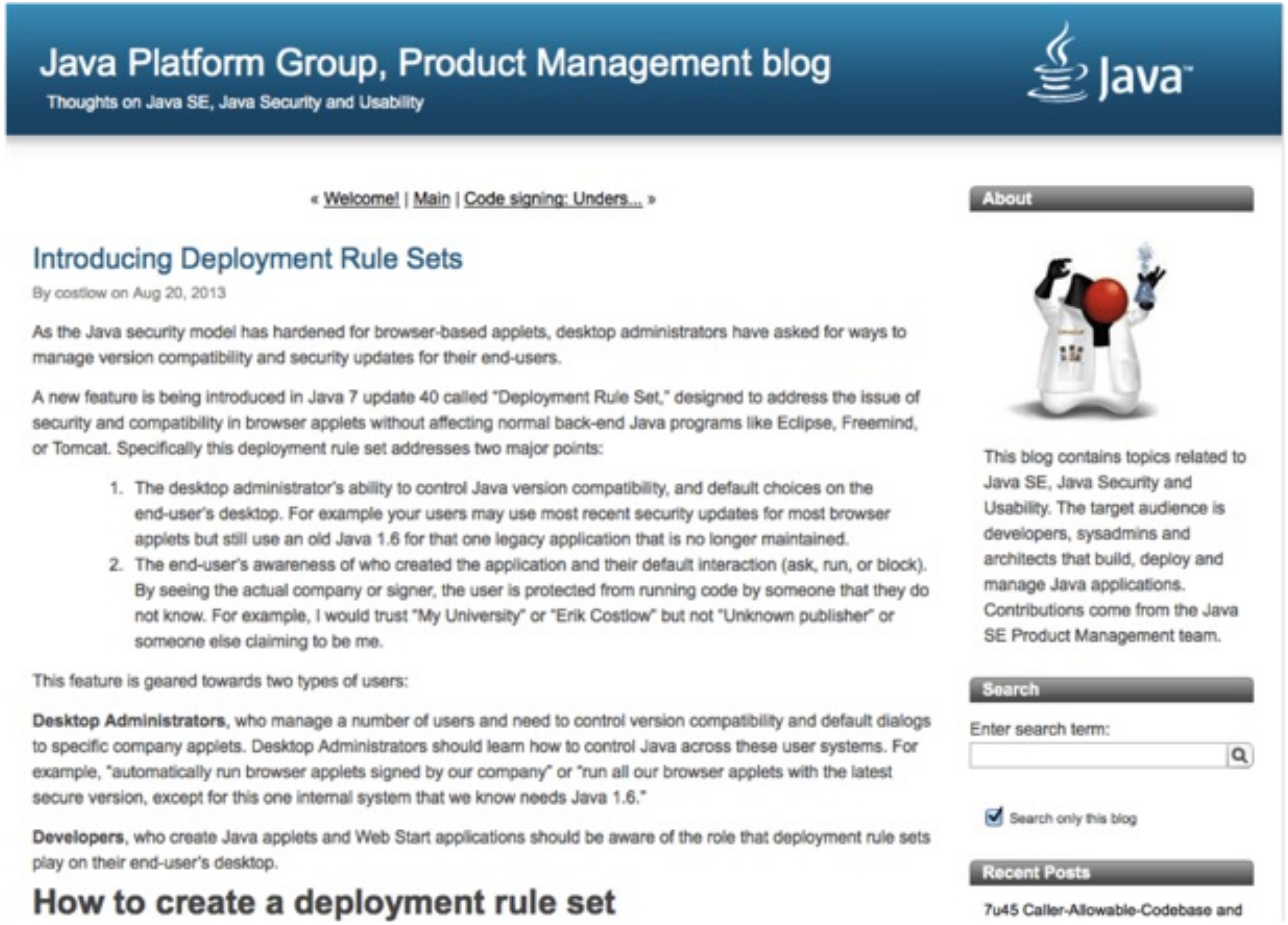
“Codes like a class, works like an int.”

Post Java 9

- **Project Valhalla** http://openjdk.java.net/projects/valhalla
  - Value Types – aggregates without identity
    http://cr.openjdk.java.net/~jrose/values/values-0.html
  - Specialization – templated types on demand
    http://cr.openjdk.java.net/~briangoetz/valhalla/specialization.html
  - JMM Update – VarHandles
- **Project Panama** http://openjdk.java.net/projects/panama
  - Arrays 2.0 – flexible array implementation and organization
  - Layouts – flexible object layout
  - FFI (JEP 191) – better native code interop

http://mail.openjdk.java.net/pipermail/valhalla-dev
http://mail.openjdk.java.net/pipermail/panama-dev

https://blogs.oracle.com/java-platform-group



Java Platform Group, Product Management blog

Thoughts on Java SE, Java Security and Usability

« Welcome! | Main | Code signing: Unders... »

Introducing Deployment Rule Sets

By costlow on Aug 20, 2013

As the Java security model has hardened for browser-based applets, desktop administrators have asked for ways to manage version compatibility and security updates for their end-users.

A new feature is being introduced in Java 7 update 40 called "Deployment Rule Set," designed to address the issue of security and compatibility in browser applets without affecting normal back-end Java programs like Eclipse, Freemind, or Tomcat. Specifically this deployment rule set addresses two major points:

1. The desktop administrator's ability to control Java version compatibility, and default choices on the end-user's desktop. For example your users may use most recent security updates for most browser applets but still use an old Java 1.6 for that one legacy application that is no longer maintained.
2. The end-user's awareness of who created the application and their default interaction (ask, run, or block). By seeing the actual company or signer, the user is protected from running code by someone that they do not know. For example, I would trust "My University" or "Erik Costlow" but not "Unknown publisher" or someone else claiming to be me.

This feature is geared towards two types of users:

Desktop Administrators, who manage a number of users and need to control version compatibility and default dialogs to specific company applets. Desktop Administrators should learn how to control Java across these user systems. For example, "automatically run browser applets signed by our company" or "run all our browser applets with the latest secure version, except for this one internal system that we know needs Java 1.6."

Developers, who create Java applets and Web Start applications should be aware of the role that deployment rule sets play on their end-user's desktop.

How to create a deployment rule set

About

This blog contains topics related to Java SE, Java Security and Usability. The target audience is developers, sysadmins and architects that build, deploy and manage Java applications. Contributions come from the Java SE Product Management team.

Search

Enter search term:

[✓] Search only this blog

Recent Posts

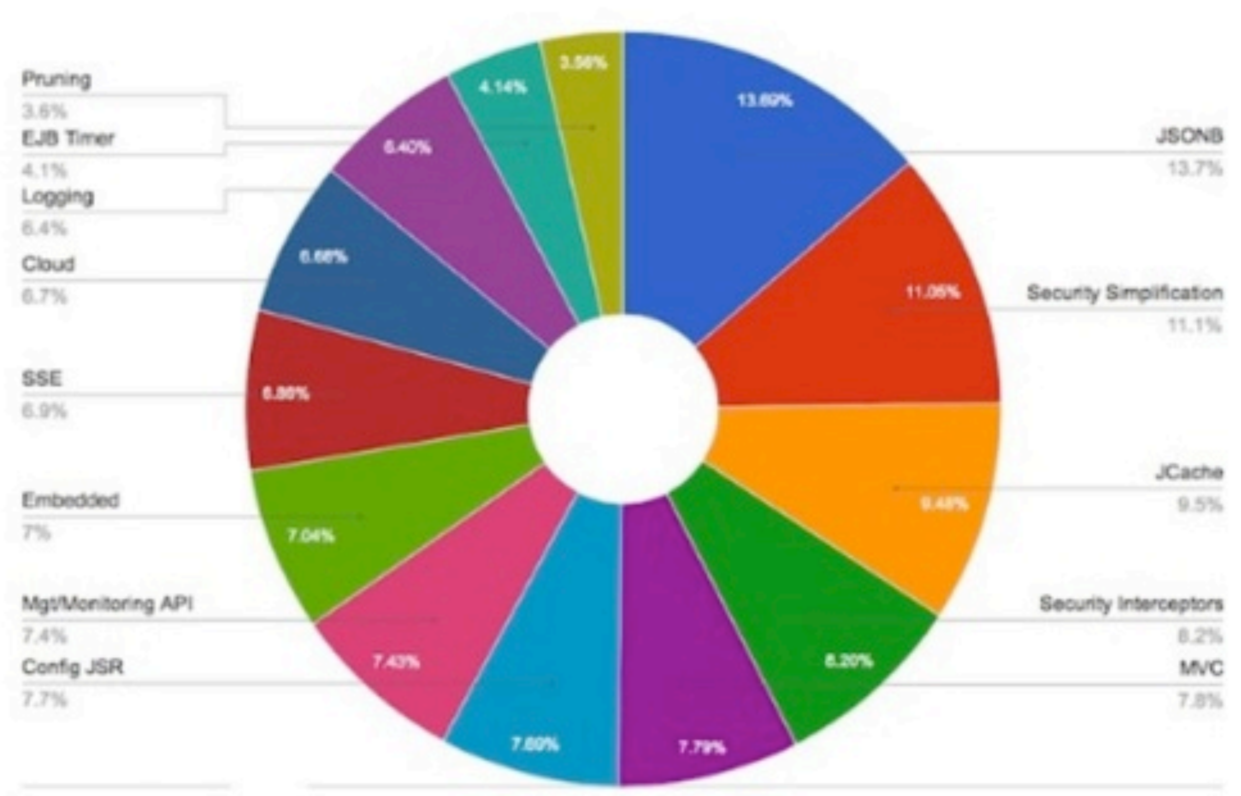
7u45 Caller-Allowable-Codebase and

Q & A

The road to Java EE 8

- **New Specifications**
  - MVC 1.0 (JSR 371)
  - JSON-B 1.0 (JSR 367)
  - Java EE Security 1.0 (JSR 375)
  - JCache (JSR 107)
- **Updated Specifications**
  - CDI 2.0 (JSR 365)
  - JAX-RS 2.1 (JSR 370)
  - Servlet 4.0 (JSR 369)
  - JSON-P 1.1 (JSR 374)
  - JMS 2.1 (JSR 368)
  - Java EE Management 2.0 (JSR 373)
  - JSF 2.3 (JSR 372)

Java EE 8 Community Survey

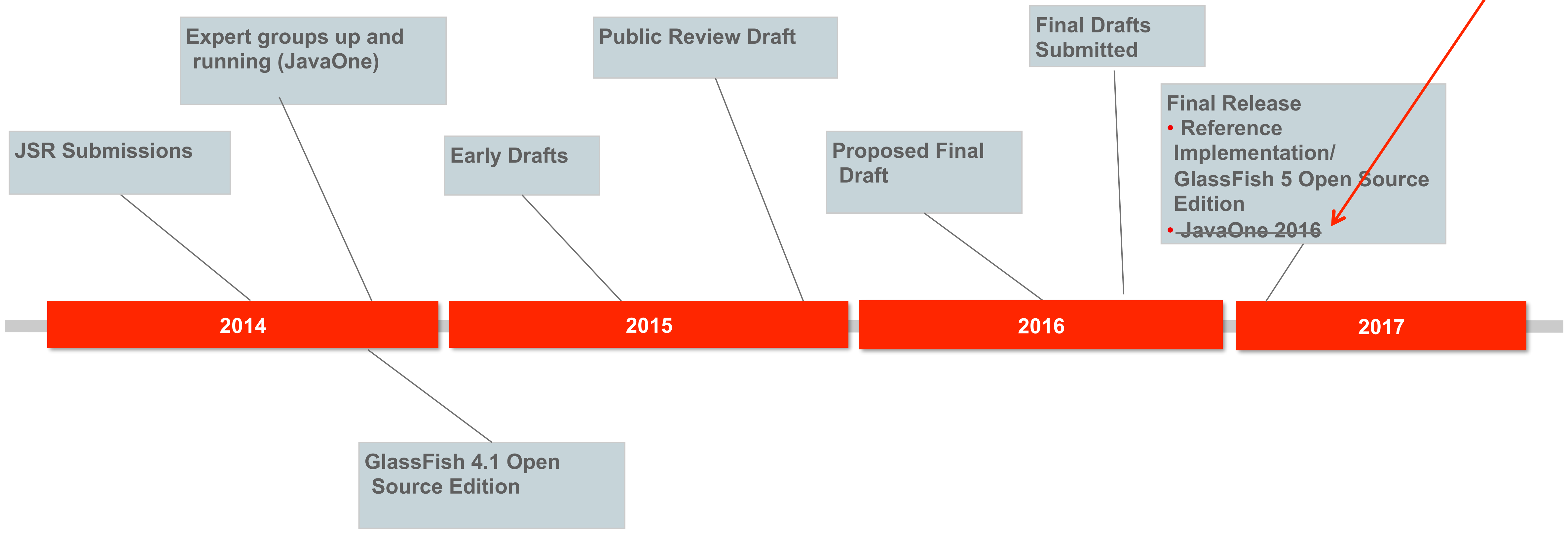


https://blogs.oracle.com/ldemichiel/entry/results_from_the_java_ee

https://java.net/downloads/javaee-spec/JavaEE8_Community_Survey_Results.pdf

Java EE/GlassFish Roadmap

** See Update at
https://blogs.oracle.com/theaquarium/entry/java_ee_8_roadmap_update
Now 1HCY2017



Thanks…

- Slides/materials from many and varied sources:
  JavaOne, JVM Language Summit, Devoxx, OpenJDK wiki / mailing lists etc

- In particular thanks to
  - Brian Goetz
  - Mark Reinhold
  - John Rose
  - Paul Sandoz
  - Simon Ritter, Doug Lea, Marcus Hirt, Aleksey Shipilëv, Bruno Borges
    and anyone I have forgotten…