
Java™
ORACLE

Java Update and Roadmap

November 2014

Tomas Nilsson
Senior Principal Product Manager
Java SE

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.

Agenda

1. Oracle and Java
2. Java SE 8 Overview
3. Java SE 9 and Beyond

1 Oracle and Java

2 Java SE 8 Overview

3 Roadmap

Oracle and Java

- Oracle has used Java since the beginning of time (eg 1990s)
- Acquired JAVA (Sun Microsystems) in 2010, including Java IP, trademarks
- Embraced OpenJDK, open community, open JCP
  - Welcomed IBM, Apple, SAP, ARM, AMD, Intel, Twitter, Goldman Sachs, Microsoft and many others
  - Made OpenJDK official Java SE reference implementation
  - Ongoing move towards open development, governance, transparency
- JDK development: Oracle and community
  - Oracle focus on modernization, security, big ticket R&D and commercial value to Oracle
  - Community contributes based on interest and ability, examples:
    - Doug Lea (concurrency, memory model), Stephen Colebourne (date/time), Michael Ernst (type annotations)
    - IBM and SAP (PPC port), CPU manufacturers (optimizations), OS vendors (ports, integration and optimizations)
    - JUGs: adopt-OpenJDK, adopt-a-JSR

1 Oracle and Java

2 Java SE 8 Overview

3 Roadmap

Java 8

"One of the biggest updates ever to a major language"

Andrew Binstock, Editor in Chief, Dr.Dobbs

Java 8 Features

Innovation

- Lambda aka Closures
- Language Interop
- Nashorn

Core Libraries

- Parallel operations for core collections APIs
- Improvements in functionality
- Improved type inference

Security

- Limited doPrivilege
- NSA Suite B algorithm support
- SNI Server Side support
- DSA updated to FIPS186-3
- AEAD JSSE CipherSuites

Java for Everyone

- Profiles for constrained devices
- JSR 310-Date & Time APIs
- Non-Gregorian calendars
- Unicode 6.2
- ResourceBundle
- BCP47 locale matching
- Globalization & Accessibility

Client

- Deployment enhancements
- JavaFX 8
- Public UI Control API
- Java SE Embedded support
- Enhanced HTML5 support
- 3D shapes and attributes
- Printing

Tools

- JSR 308-Annotations on Java Type
- Repeating Annotations
- Java Dependency Analysis Tool
- App Store packaging tool (8u20)

General Goodness

- JVM enhancements
- No PermGen limitations
- Performance improvements

Enterprise

- Mission Control 5.3
- Flight Recorder
- Usage Tracker
- Advanced Mgmt Console (8u20)
- MSI JRE Installer (8u20)

Lambda Expressions

(Anonymous inner classes done right)

- A lambda expression (closure) is an anonymous method
  - Has an argument list, a return type, and a body

```
(Object o) -> o.toString()
```

  - Can refer to values from the enclosing lexical scope

```
(Person p) -> p.getName().equals(name)
```

- A method reference is a reference to an existing method

```
Object::toString
```

- Allow you to *treat code as data*
  - Behavior can be stored in variables and passed to methods

Streams

java.util.stream

- A sequence of elements supporting sequential and parallel aggregate operations.
- Concerned with declaratively describing
  a. their source, and
  b. The operations which will be performed on that source
- To perform a computation, stream operations are composed into a stream pipeline
- …which may execute either sequentially or in parallel

Example: Lambda Functions and Streams

```
Set<Seller> sellers = new HashSet<>();

for (Txn t : txns) {
    if (t.getBuyer().getAge() >= 65)
        sellers.add(t.getSeller());
}

List<Seller> sorted = new ArrayList<>(sellers);

Collections.sort(sorted, new Comparator<Group>() {
    public int compare(Seller a, Seller b) {
        return a.getName().compareTo(b.getName());
    }
});

for (Seller s : sorted)
    System.out.println(s.getName());
```

Example: Lambda Functions and Streams

```
txns.stream()
    .filter(t -> t.getBuyer().getAge() >= 65)
    .map(Txn::getSeller)
    .distinct()
    .sort(comparing(Seller::getName))
    .forEach(s -> System.out.println(s.getName());
```

Example: Lambda Functions and Streams

```
txns.stream()
    .parallel()
    .filter(t -> t.getBuyer().getAge() >= 65)
    .map(Txn::getSeller)
    .distinct()
    .sort(comparing(Seller::getName))
    .forEach(s -> System.out.println(s.getName());
```

Example: Static and Default Methods in Interfaces

```
public interface TimeClient {
    void setTime(int hour, int minute, int second);
    void setDate(int day, int month, int year);
    void setDateAndTime(int day, int month, int year,
                                int hour, int minute, int second);
    LocalDateTime getLocalDateTime();

    static ZoneId getZoneId (String zoneString) {
        try {
            return ZoneId.of(zoneString);
        } catch (DateTimeException e) {
            System.err.println("Invalid time zone: " + zoneString +
                "; using default time zone instead.");
            return ZoneId.systemDefault();
        }
    }

    default ZonedDateTime getZonedDateTime (String zoneString) {
        return ZonedDateTime.of(getLocalDateTime(), getZoneId(zoneString));
    }
}
```

Nashorn JavaScript Engine

- “Pure Java” Java Script Engine, replaces Rhino
- ECMAScript 5.1 compliant
- Optimized for and with Invokedynamic (JSR-292)
- Invoke from Java code or command line (jjs)
- Simple access to Java classes and objects from JS code
- Integrates with JavaFX 8

Example: Invoking JS from Java

```
public class EvalScript {
    public static void main(String[] args) throws Exception {

        // create a script engine manager
        ScriptEngineManager factory = new ScriptEngineManager();

        // create a Nashorn script engine
        ScriptEngine engine = factory.getEngineByName("nashorn");

        // evaluate JavaScript statement
        try {
            engine.eval("print('Hello, World!');");
        } catch (final ScriptException se) { se.printStackTrace(); }
    }
}
```

Example: Accessing Java from JS

```
var Run = Java.type("java.lang.Runnable");
var MyRun = Java.extend(Run, {
    run: function() {
        print("Run in separate thread");
    }
});
var Thread = Java.type("java.lang.Thread");
var th = new Thread(new MyRun());
```

Java Date/Time Features – JSR 310

- Fluent, Immutable, Thread Safe, Easy to use
- Strong typing with fit for purpose types
- Easy to use formatting and parsing
- Interoperable with java.util.Calendar/Date
- Extensible Units, Fields, and Chronologies
- Supports Regional Calendars
- DatePicker support in JavaFX
- Supported by JDBC, java.sql.Date/Time/Timestamp
- The essential ISO Calendar for global business

Other Internationalization Enhancements

Feature	Benefit
Unicode 6.2	Improved support for languages/scripts used in Asia, Africa and the Middle East
Custom resource bundles	Enables easy extension with custom/specialized locales
Common Locale Data Repository (CLDR) support	Enables use of CLDR locale service providers
BCP 47 locale matching	Enables matching of application locale to user preferences

JavaFX 8

- New Modern Theme: Modena
- Enhancements to Collections, Bindings, Tasks and Services
- Full screen improvements, new unified stage style, rich text and printing
- Embed Swing in JavaFX applications
- DatePicker and TreeTableViews
- Public API for CSS structure
- WebView enhancements (Web Sockets, Web Workers, Web Fonts)
- JavaFX 3D
- Multi-touch (introduced in 7, relevant in 8)
- Hi-DPI (Retina) display support

Annotations & Developer Tools

- Annotations on Java Types (JSR-308)
  - Enabled through type checking frameworks such as http://types.cs.washington.edu/checker-framework/

```
@NonNull String str;
```

- Repeating Annotations

```
@Schedule(dayOfMonth="last")
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }
```

- Java Dependency Analysis Tool (jdeps)

JVM and Performance Improvements

10-30% improvement common on multi-core HW

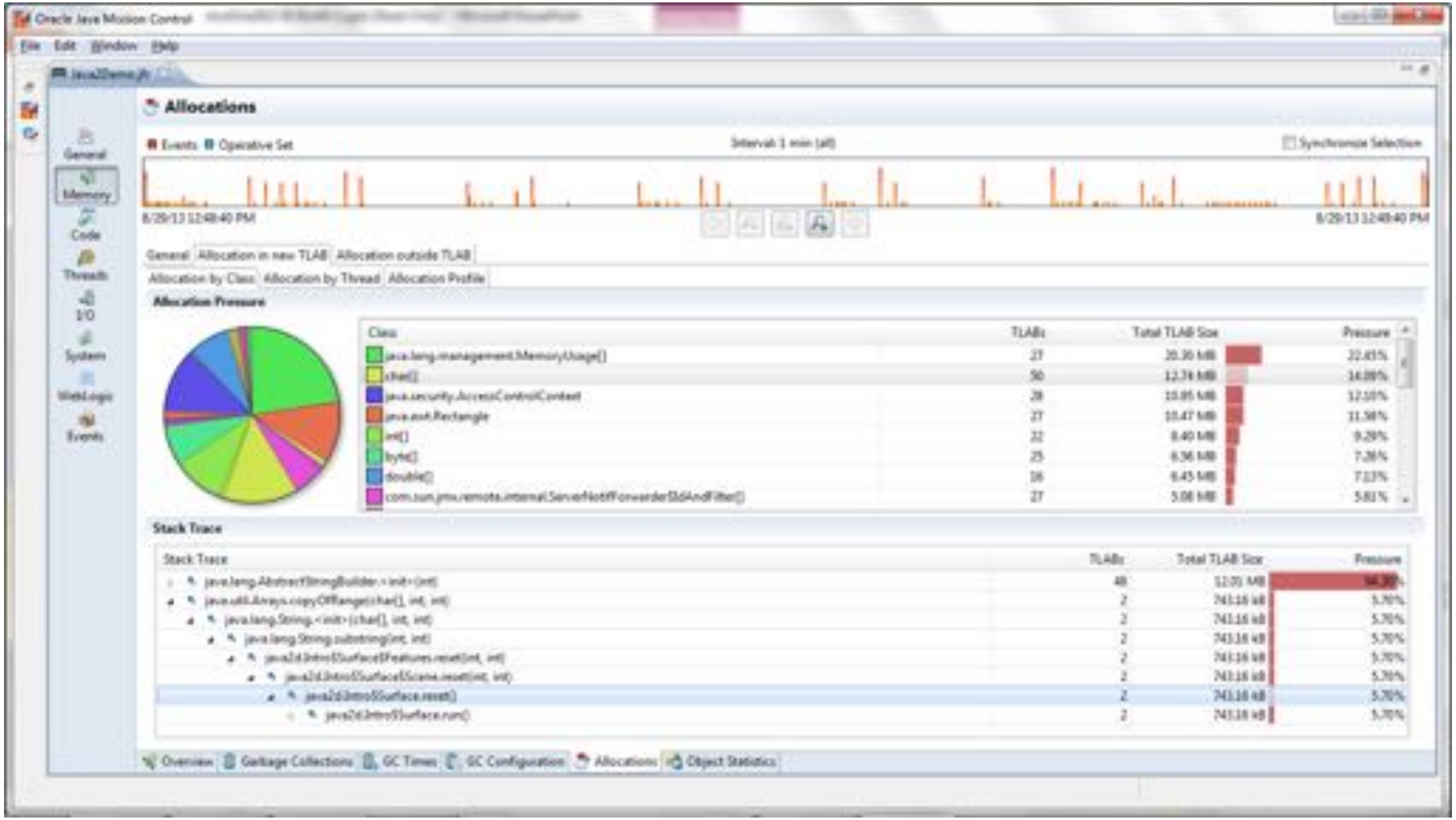
- Concurrency JSR 166
  - Improved performance of existing APIs and addition of new APIs
- Reduced false sharing
  - Move frequently updated memory words to separate cache lines
- Tiered Compilation enabled by default
- Faster JSR-292 implementation
- Faster crypto performance on modern hardware
- Permgen removal
- Continued G1 GC improvements

Java Mission Control and Flight Recorder

From JRockit

- In-depth profiling and analysis in production and development
- Always on
  - Virtually no performance overhead
- JDK and application data
  - Information logged in all layers
- Analyze root cause
  - Detailed data collected leading up to the event (time machine)

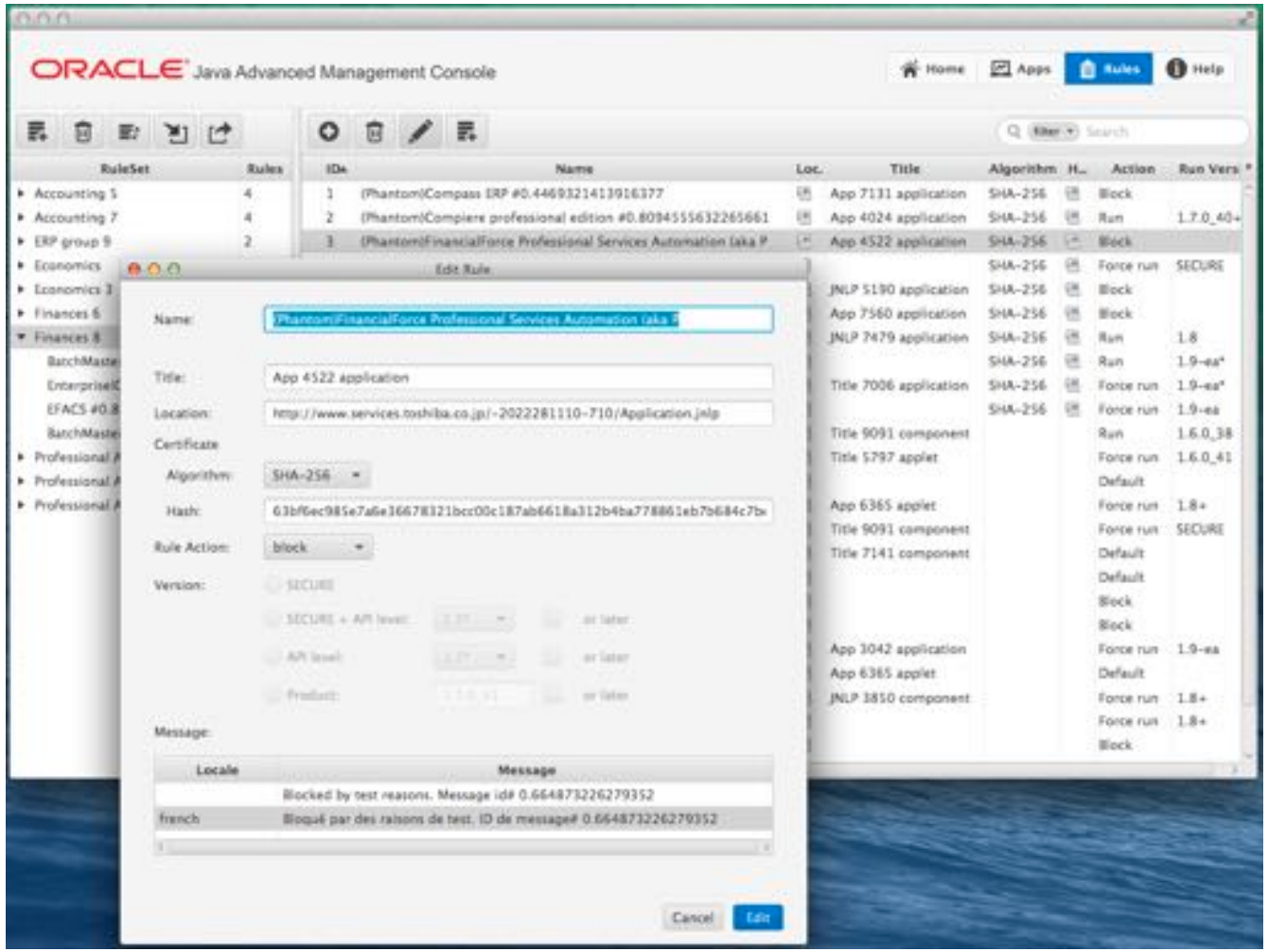
Java Mission Control and Flight Recorder



Advanced Management Console

Introduced in 8u20

- Track and manage clients at scale
- Usage tracking analysis, Client security configuration management

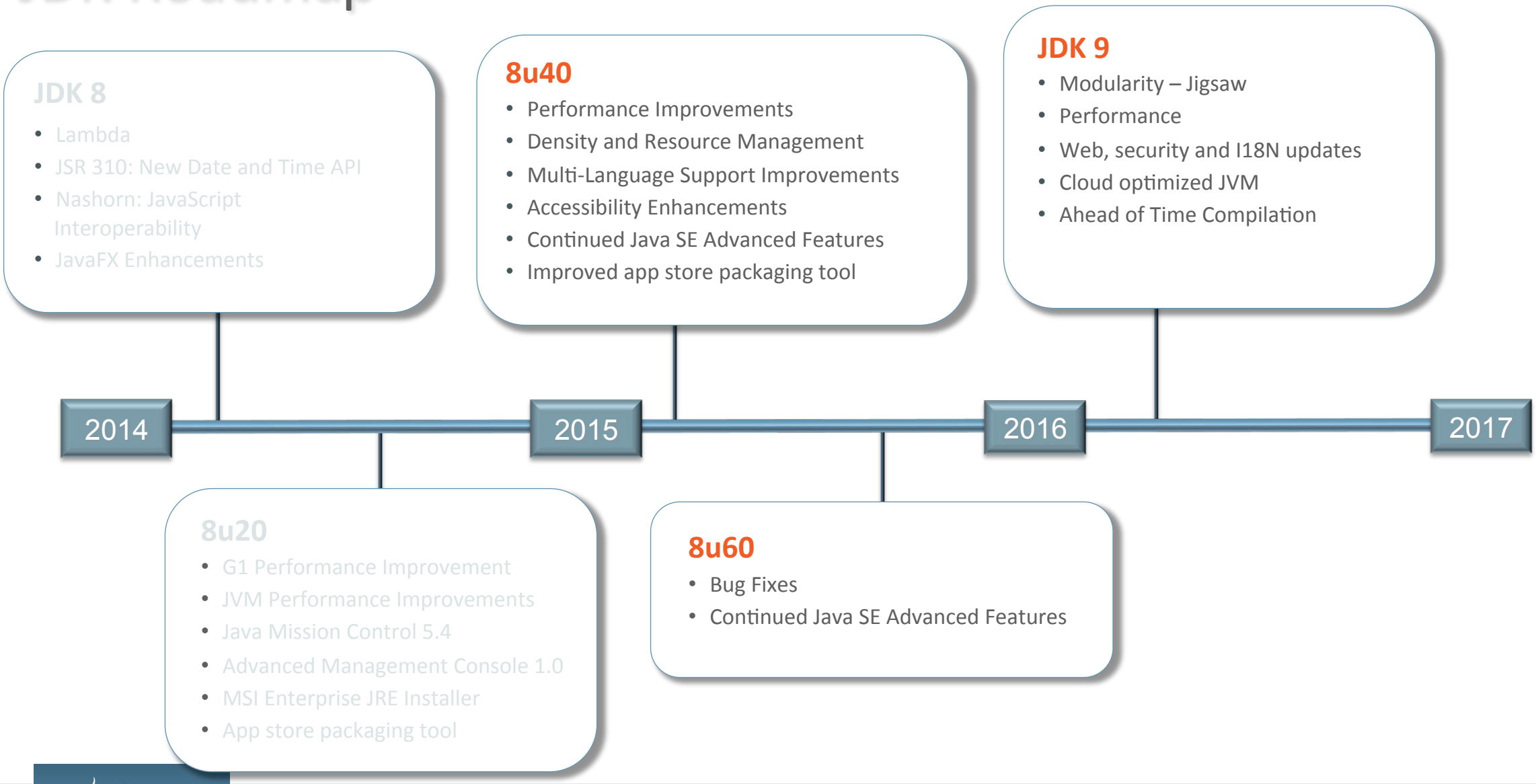


1 Oracle and Java

2 Java SE 8 Overview

3 Roadmap

JDK Roadmap



JDK 8u40 Release Content

- Performance
  - Memory footprint optimizations
  - Reduce need for full GCs in G1
  - Crypto improvements on SPARC
- Small feature enhancements (highlights)
  - Nashorn: Improved integration with Java security model, ECMAScript 6 preparations, performance
  - JavaFX: Modernize media stack on OS X, Accessibility
  - Install/config: Improved packaging tool for app stores, etc (*javapackager*)
  - JavaDB and Mission Control updates

Java SE Advanced Upcoming Features

Desktop Management, Multi-Tenancy Optimizations

- Advanced Management Console 2.0 – Desktop JRE management
  - Improved usage tracking/analytics
  - Auditing of installed versions, configuration
  - Centralized configuration management
  - Tool support for building customized JRE installers
  - Improved scalability (100,000s of JREs)
- Resource Management APIs
  - Define resource contexts, tracking of CPU/memory/IO usage per context
  - OOTB WebLogic Server integration, open API for 3rd party integration
- Other Future Additions
  - Cooperative Lifecycle support for virtualized/cloud environments
  - Low-latency GC (vastly enhanced variant of JRockit Deterministic GC)

JDK 9 Release Content (1/2)

Under review for inclusion in 9*

- Jigsaw
  - Modularized JDK, Java module system
- Performance
  - JIT, locking, memory footprint enhancements
  - Crypto acceleration enhancements
- Web, Security, Internationalization updates
  - HTTP 2.0, JAXP update
  - JCE updates, TLS updates, DTLS support, Keystore enhancements
  - Unicode 7, UTF-8 properties support
- Serviceability
  - Improved JVM logging
  - More diagnostics commands

* As of October 31, 2014. For up to date information, see openjdk.java.net.

JDK 9 Release Content (2/2)

Under review for inclusion in 9*

- Other features
  - Ahead of Time compilation (improved startup time)
  - Read Eval Print Loop (REPL) support in JDK
  - Enhanced volatiles
  - OS Process control API
- Cleanup
  - Remove old GC combinations (deprecated in JDK 8)
  - Remove old/less frequently used tools from JDK
  - Revisit version numbering scheme
  - Block access to JDK internal APIs by default
- Tooling
  - javac enhancements
  - JIT, locking, memory footprint enhancements

* As of October 31, 2014. For up to date information, see openjdk.java.net.

Avoid private (internal, non-spec) APIs

Private APIs are (finally) going away

Some APIs were never supposed to be used…

Warnings posted from Feb 1998 to today

1998 Coolest phone
Nokia 5110

Sun

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. 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. Those classes will not in general be present on another vendor's Java platform. 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.

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

ORACLE

Account Sign Out Help Country Communities I am a... I want to... Search

Products Solutions Downloads Store Support Training Partners About OTN

Oracle Technology Network > Java

Java SE
Java EE
Java ME
Java SE Support
Java SE Advanced & Suite
Java Embedded
Java DB
Web Tier
Java Card
Java TV
New to Java
Community
Java Magazine

JDK Contents

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

Java SDKs and Tools
Java SE
Java EE and Glassfish
Java ME
Java Card
NetBeans IDE
Java Mission Control

Java Resources
Java APIs
Technical Articles
Demos and Videos
Forums
Java Magazine
Java.net
Developer Training
Tutorials

Java™ ORACLE

Exec summary of previous page

- Private packages like sun.* are “JDK internal workings”
  - Not part of the specification
  - Are not supported
  - Any one might change/disappear in any release without advanced warning
- Smart and informed developers do not use private JDK packages
- Smart and informed developers do not let friends use private JDK packages

Exec summary of previous page (Updated)

- Private packages like sun.* are “JDK internal workings”
  - Not part of the specification
  - Are not supported
  - All of them will “disappear” in JDK 9 with plenty of advanced warning
- Smart and informed developers do not use private JDK packages
- Smart and informed developers do not let friends use private JDK packages

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

Future Direction

Addressing Big Problems and Interesting Areas

- Known “Weaknesses”
  - Startup & Warmup time
  - Memory overhead
  - Optimizations for more specialized hardware (vector instructions, GPUs, zero-copy I/O, transactional synchronization/memory)
  - Unpredictable latency due to GC
  - Code verbosity
- “New” Opportunities
  - Big Data (eg, the Hadoop ecosystem)
  - Cloud & large multi-tenant deployments
  - (More) JVM improvements for non-Java languages

More Information, Actions

- Join a user group
  - https://www.java.net/jugs/java-user-groups
- Prepare for change
  - https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool
  - Move to 64-bit
  - Sign your applets
- Follow JDK development, help with early access testing
  - http://openjdk.java.net/projects/jdk8u/
  - http://openjdk.java.net/projects/jdk9/
  - http://wiki.openjdk.java.net/
- Contribute!
  - http://openjdk.java.net/contribute/
  - https://java.net/projects/adoptajsr/pages/Home
  - https://java.net/projects/adoptopenjdk


Java™
ORACLE


ORACLE®