CADEC 2016 - JAVA 9 MODULE SYSTEM "JIGSAW"

HÅKAN DAHL

2016-01-27 | CALLISTAENTERPRISE.SE

CALLISTA
— ENTERPRISE —

AGENDA

- Introduction
- Module system mechanics
- Ecosystem

CALLISTA
— ENTERPRISE —

INTRODUCTION - JAVA 9 FEATURES

102: Process API Updates
110: HTTP 2 Client
143: Improve Contended Locking
158: Unified JVM Logging
165: Compiler Control
193: Variable Handles
197: Segmented Code Cache
199: Smart Java Compilation, Phase Two
201: Modular Source Code
211: Elide Deprecation Warnings on Import Statements
212: Resolve Lint and Doclint Warnings
213: Milling Project Coin
214: Remove GC Combinations Deprecated in JDK 8
215: Tiered Attribution for javac
216: Process Import Statements Correctly
217: Annotations Pipeline 2.0
219: Datagram Transport Layer Security (DTLS)
220: Modular Run-Time Images
221: Simplified Doclet API
222: jshell: The Java Shell (Read-Eval-Print Loop)
223: New Version-String Scheme
224: HTML5 Javadoc
225: Javadoc Search
226: UTF-8 Property Files
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
232: Improve Secure Application Performance
233: Generate Run-Time Compiler Tests Automatically
235: Test Class-File Attributes Generated by javac
236: Parser API for Nashorn
237: Linux/AArch64 Port
238: Multi-Release JAR Files
240: Remove the JVM TI hprof Agent
241: Remove the jhat Tool
243: Java-Level JVM Compiler Interface
244: TLS Application-Layer Protocol Negotiation Extension
245: Validate JVM Command-Line Flag Arguments
246: Leverage CPU Instructions for GHASH and RSA
247: Compile for Older Platform Versions
248: Make G1 the Default Garbage Collector
249: OCSP Stapling for TLS
250: Store Interned Strings in CDS Archives
251: Multi-Resolution Images
252: Use CLDR Locale Data by Default
253: Prepare JavaFX UI Controls & CSS APIs for Modularization
254: Compact Strings
255: Merge Selected Xerces 2.11.0 Updates into JAXP
256: BeanInfo Annotations
257: Update JavaFX/Media to Newer Version of GStreamer
258: HarfBuzz Font-Layout Engine
259: Stack-Walking API
262: TIFF Image I/O
263: HiDPI Graphics on Windows and Linux
264: Platform Logging API and Service
265: Marlin Graphics Renderer
266: More Concurrency Updates
267: Unicode 8.0
268: XML Catalogs
269: Convenience Factory Methods for Collections
270: Reserved Stack Areas for Critical Sections
272: Platform-Specific Desktop Features
273: DRBG-Based SecureRandom Implementations
274: Enhanced Method Handles
276: Dynamic Linking of Language-Defined Object Models

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

CALLISTA
— ENTERPRISE —

INTRODUCTION - JAVA 9 FEATURES

Main feature: the **module system**, aka project **Jigsaw**

JEP 200: The Modular JDK
JEP 201: Modular Source Code
JEP 220: Modular Run-Time Images
JEP 260: Encapsulate Most Internal APIs
JEP 261: Module System

JSR 376: Java Platform Module System

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

CALLISTA
— ENTERPRISE —

Why should I care about the module system?

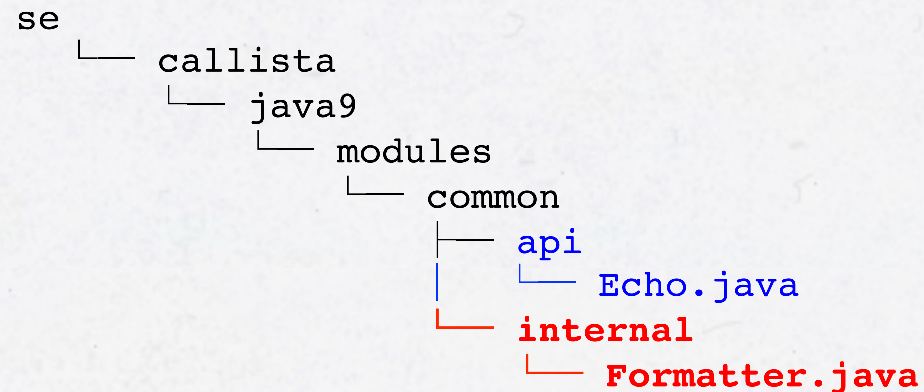
- Targets some long-time painpoints
- Platform change - affects tooling and ecosystem
- Partly disruptive - no opt-in for some parts

CALLISTA
— ENTERPRISE —

INTRODUCTION - MOTIVATION FOR A MODULE SYSTEM

Painpoints

- JAR-hell / classpath mess
- monolithic JDK
- encapsulation (public accessor, no means to hide internals)



CALLISTA
— ENTERPRISE —

INTRODUCTION - MOTIVATION FOR A MODULE SYSTEM

```
|  |  \- com.fasterxml.jackson.core:jackson-databind:jar:2.4.3:com
|  |     +- com.fasterxml.jackson.core:jackson-annotations:jar:2.4
|  |     \- com.fasterxml.jackson.core:jackson-core:jar:2.4.3:comp
|  +- org.jasypt:jasypt:jar:1.8:compile
|  +- org.apache.activemq:activemq-core:jar:5.6.0:compile
|  |  +- org.apache.geronimo.specs:geronimo-jms_1.1_spec:jar:1.1.1
|  |  +- org.apache.activemq:kahadb:jar:5.6.0:compile
|  |  +- org.apache.activemq.protobuf:activemq-protobuf:jar:1.1:co
|  |  +- org.osgi:org.osgi.core:jar:4.1.0:compile
|  |  +- org.apache.geronimo.specs:geronimo-j2ee-management_1.1_sp
mpile
|  |  \- commons-net:commons-net:jar:2.2:compile
|  +- org.apache.xbean:xbean-spring:jar:3.9:compile
|  |  \- commons-logging:commons-logging:jar:1.0.3:compile
|  +- commons-dbcp:commons-dbcp:jar:1.4:compile
|  +- net.sourceforge.jtds:jtds:jar:1.2.4:compile
|  +- org.slf4j:slf4j-api:jar:1.7.7:compile
|  +- org.slf4j:jcl-over-slf4j:jar:1.7.7:compile
|  +- org.apache.logging.log4j:log4j-1.2-api:jar:2.1:compile
|  +- org.apache.logging.log4j:log4j-api:jar:2.1:compile
|  +- org.apache.logging.log4j:log4j-core:jar:2.1:compile
|  +- org.apache.logging.log4j:log4j-jcl:jar:2.1:compile
|  +- org.apache.logging.log4j:log4j-jul:jar:2.1:compile
|  +- org.apache.logging.log4j:log4j-slf4j-impl:jar:2.1:compile
|  +- com.lmax:disruptor:jar:3.3.0:compile
|  +- org.mule.distributions:mule-standalone:pom:3.7.0:compile
|  |  +- org.mule:mule-core:jar:3.7.0:compile
|  |  |  +- org.mule.extensions:mule-extensions-api:jar:1.0.0-alph
|  |  |  +- com.github.stephenc.eaio-uuid:uuid:jar:3.4.0:compile
|  |  |  |  \- com.github.stephenc.eaio-grabbag:grabbag:jar:1.8.1:
|  |  |  +- commons-cli:commons-cli:jar:1.2:compile
|  |  |  +- commons-pool:commons-pool:jar:1.6:compile
|  |  |  +- org.apache.geronimo.specs:geronimo-jta_1.1_spec:jar:1.
|  |  |  +- org.apache.geronimo.specs:geronimo-j2ee-
tor_1.5_spec:jar:1.1:compile
|  |  |  +- javax.inject:javax.inject:jar:1:compile
```

... and 6 more pages ...

JAR-hell / classpath mess

- Is everything I need there?
- Are there any split-packages?

```
+- org.codehaus.groovy:groovy-all:jar:indy:2.3.7:com

\- org.codehaus.groovy:groovy-all:jar:2.2.2:compile
```


CALLISTA
— ENTERPRISE —

INTRODUCTION - MOTIVATION FOR A MODULE SYSTEM

Module system requirements

- **Reliable configuration**, *to replace the brittle, error-prone class-path mechanism with a means for program components to declare explicit dependences upon one another;*
- **Strong encapsulation**, *to allow a component to declare which of its APIs are accessible by other components, and which are not;*
- **A scalable Java SE Platform**, *whose components can be assembled by developers into custom configurations that contain only the functionality actually required by an application;*
- **Greater platform integrity**, *to ensure that code that is internal to a platform implementation is not accessible from outside the implementation; and*
- **Improved performance**, *by applying whole-program optimization techniques to complete configurations of platform, library, and application components.*

http://openjdk.java.net/projects/jigsaw/spec/reqs/

CALLISTA
— ENTERPRISE —

INTRODUCTION - HISTORY

- Java 7 (2011) --> Java 8 (2014) --> Java 9
- vs OSGi ?

CALLISTA
— ENTERPRISE —

INTRODUCTION - RELEASE SCHEDULE JDK 9

2015-12-01

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

2015-12-09

Schedule

2016/05/26	Feature Complete
2016/08/11	All Tests Run
2016/09/01	Rampdown Start
2016/10/20	Zero Bug Bounce
2016/12/01	Rampdown Phase 2
2017/01/26	Final Release Candidate
2017/03/23	General Availability

"It would be best to
use the additional time to stabilize, polish, and fine-tune the features
that we already have rather than add a bunch of new ones."

Mark Reinhold

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

http://mail.openjdk.java.net/pipermail/jdk9-dev/2015-December/003149.html

When will Java 8 "end of public updates" occur?

"One year after the GA of a subsequent major release"

→ April 2018

Oracle Java SE Support Roadmap
http://www.oracle.com/technetwork/java/eol-135779.html

CALLISTA
— ENTERPRISE —

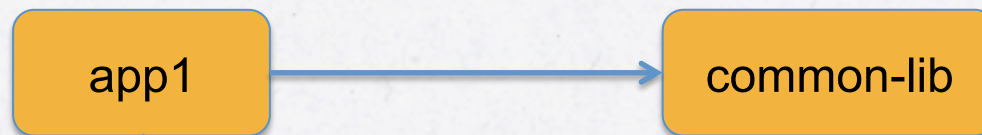
MODULE SYSTEM MECHANICS

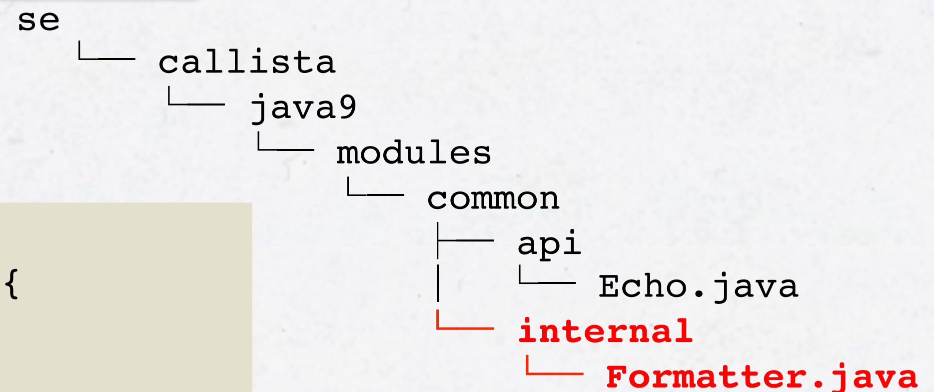
- Exported API
- Modularised JDK
- Modules vs classpath
- Build system interaction

CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

Example: dependency with internal API usage

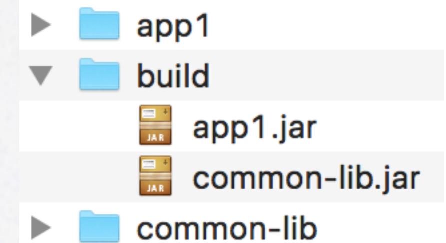





```
public class App1 {
  public static void main(String[] args) {
    Echo e = new Echo();
    e.echo("hello : "
        + new Formatter().formatDate(new Date()));
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API



Java 8 - using the classpath to compile app1 ...

```
javac -classpath build/common-lib.jar $(find app1/src -name "*.java")
```

... and run

```
java -classpath build/app1.jar:build/common-lib.jar se.callista.java9.modules.app1.App1

hello : 2015-12-30 18:05:07
```

If a dependency is missing from classpath (in runtime) we get

```
java -classpath build/app1.jar se.callista.java9.modules.app1.App1

Exception in thread "main" java.lang.NoClassDefFoundError: se/callista/java9/modules/common/api/Echo
	at se.callista.java9.modules.app1.App1.main(App1.java:11)
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

Java 9 – declare modules using **module-info.java**

```
module se.callista.java9.modules.app1 {
    requires se.callista.java9.modules.common;
}
```

*requires **module***

```
module se.callista.java9.modules.common {
    exports se.callista.java9.modules.common.api;
}
```

*exports **package***

CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

module-info.java lives in the source root

```
common-lib
 └──── src
        └──── main
               └──── java
                      ├──── module-info.java
                      └──── se
                             └──── callista
                                    └──── java9
                                           └──── modules
                                                  └──── common
                                                         ├──── api
                                                         │      └──── Echo.java
                                                         └──── internal
                                                                └──── Formatter.java
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

Java 9 – using the modulepath to compile app1 ...

```
javac -modulepath build $(find app1/src -name "*.java")
```

... now gives an error

```
checking se/callista/java9/modules/common/module-info
app1/src/main/java/se/callista/java9/modules/app1/App1.java:6: error: package
se.callista.java9.modules.common.internal does not exist
import se.callista.java9.modules.common.internal.Formatter;
                                               ^
app1/src/main/java/se/callista/java9/modules/app1/App1.java:12: error: cannot find symbol
                    e.echo("hello : " + new Formatter().formatDate(new Date()));
```

Strong encapsulation!

MODULE SYSTEM MECHANICS - EXPORTED API

Remove usage of common internal class from App1 ...

```
package se.callista.java9.modules.app1;

import java.util.Date;
import se.callista.java9.modules.common.api.Echo;
//import se.callista.java9.modules.common.internal.Formatter;

public class App1 {
    public static void main(String[] args) {
        Echo e = new Echo();
        //e.echo("hello : " + new Formatter().formatDate(new Date()));
        e.echo("hello : " + new Date());
```

... and it compiles using

```
javac -modulepath build $(find app1/src -name "*.java”)
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

Trying to run without all required modules ...

```
rm build/se.callista.java9.modules.common.jar
java -modulepath build -m se.callista.java9.modules.app1/se.callista.java9.modules.app1.App1
```

... gives an error

```
Error occurred during initialization of VM
java.lang.module.ResolutionException: Module se.callista.java9.modules.common not found, required by se.callista.java9.modules.app1
        at java.lang.module.Resolver.fail(java.base@9-ea/Resolver.java:860)
```

Reliable configuration!

CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

Split packages: trying to compile with two modules exporting the same package

```
javac -modulepath build $(find app1/src -name "*.java")
```

... gives an error

```
Error occurred during initialization of VM
java.lang.module.ResolutionException: Modules se.callista.java9.modules.common and
se.callista.java9.modules.common.splitpackage export package
se.callista.java9.modules.common.api to module se.callista.java9.modules.app1
        at java.lang.module.Resolver.fail(java.base@9-ea/Resolver.java:860)
```

Reliable configuration!

CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - MODULARISED JDK

But hey, we used the java.util package without requireing it ...

```
module se.callista.java9.modules.app1 {
    requires se.callista.java9.modules.common;
}
```

```
import java.util.Date;

public class App1 {
    ...
        e.echo("hello : " + new Date());
```

All modules requires **java.base** (implicitly)

```
module se.callista.java9.modules.app1 {
    requires java.base;
    requires se.callista.java9.modules.common;
}
```

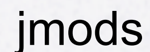
```
module java.base {
    exports
        java.lang
        java.net
        java.text
        java.util
        ...
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - MODULARISED JDK

JDK 9 has been restructured for modules

jdk1.8.0_66
- bin
- jre x
- lib not the same

jdk-9-ea-96-jigsaw
- bin
- conf moved here from jre/...
- jmods 
- lib mostly native libs

Note: JDK internal API's no longer accessible:
- com.sun.*
- ...

java.activation.jmod
java.annotations.common.jmod
java.base.jmod
java.compact1.jmod
java.compact2.jmod
java.compact3.jmod
java.compiler.jmod
java.corba.jmod
java.datatransfer.jmod
java.desktop.jmod
java.instrument.jmod
java.logging.jmod
...

A scalable Java SE Platform!

CALLISTA ENTERPRISE

MODULE SYSTEM MECHANICS - EXPORTED API

Transitive module dependencies

- or how to avoid re-declaring module dependencies for API-dependencies

Example: Add a new method taking a java.sql.Timestamp method to API

common-lib

```
package se.callista.java9.modules.common.api;

import java.sql.Timestamp;

public class Echo {
  ...

  public void echo(String s, Timestamp ts) {
   ...
  }
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - EXPORTED API

Declaring a transitive dependency

```
module se.callista.java9.modules.app1 {
    requires se.callista.java9.modules.common;
}
```

```
public class App1 {
  ...
    e.echo("hello", new java.sql.Timestamp(1));
```

```
module se.callista.java9.modules.common {
    exports se.callista.java9.modules.common.api;
    requires public java.sql;
}
```


CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - ADVANCED

More advanced module relationships

- ”provides” (service provider)
- ”export <package> to <module>”
  - compare with ”friend”, ”buddy-classloading”
  - think hibernate ...

MODULE SYSTEM MECHANICS - MODULES VS CLASSPATH

How do modules and classpath interact?

- Explicit modules
  - modular JAR on module path
- Automatic modules
  - classic JAR on module path (named)
  - exports all / requires all
- Unnamed module
  - everything on the classpath
  - requires all

MODULE SYSTEM MECHANICS - BUILD SYSTEM INTERACTION

- Module dependencies are **not versioned**
  - out-of-scope for Jigsaw due to complexity

- We now have two places to declare dependencies …
  - module-info.java (**no version**)
  - build system dependencies (**with version**)

- Most obvious solution:
  - try to generate one set of declarations …

MODULE SYSTEM MECHANICS - BUILD SYSTEM INTERACTION

Gradle demo

- new software model
- how dependencies are handled
- support for Java 7 / 8 / 9

https://docs.gradle.org/current/userguide/java_software.html

Java Components: Solving the Puzzle with Jigsaw and Gradle
https://www.youtube.com/watch?v=0-EP7TzpxAI

CALLISTA
— ENTERPRISE —

MODULE SYSTEM MECHANICS - BUILD SYSTEM INTERACTION

```
plugins {
  id 'jvm-component'
  id 'java-lang'
}

model {
  components {
    'common-lib'(JvmLibrarySpec) {
      api {
        exports 'se.callista.java9.modules.common.api'
      }

      targetPlatform 'java9'
    }
  }
}
```

generate

module-info.java

```
module se.callista.java9.modules.common {
  exports se.callista.java9.modules.common.api;
}
```


CALLISTA
— ENTERPRISE —

ECOSYSTEM

- The missing pieces
  - build tools (Gradle, Maven, ...)
  - IDE's
  - binary repositories
    - » needs java 9 modular jars
    - » metadata
    - » and who starts ...
  - optional dependencies?
  - Spring 5 targets Java 9
  - Java EE

CALLISTA
— ENTERPRISE —

ECOSYSTEM - JAVA EE

- Java EE packaging not defined for modules yet
  - does it matter?
  - java-containerless is the new black …

CALLISTA
— ENTERPRISE —

SUMMARY

- You can start modularizing your builds now using Java 7/8 with Gradle
  - sharpen your API design
  - sanitize your dependencies
  - Note: limited to compile time dependency checking
- Run jdeps to find code that needs to be migrated
- The Java ecosystem will need some time to adapt before Java 9 modules can be fully used
- **Simple migration to Java 9 – use the classpath**

CALLISTA
— ENTERPRISE —

QUESTIONS ?

EXTRA MATERIAL

APPLICATIONS ON JAVA 9

- Existing app on Java 9?
- How to apply modules?
  - automatic module vs explicit module vs unnamed module
  - app as module(s)
  - libs as modules

MODULE SYSTEM MECHANICS - EXPORTED API

Running an app using the module path

```
java -modulepath build
    -m se.callista.java9.modules.app1/se.callista.java9.modules.app1.App1
```

No need for classpath scan!

RESOURCE LOADING

Resource loading across modules

```
<spring:import resource="classpath*:context-in-jarfile.xml"/>
```

Works differently with modules!

None of the below works across modules:

```
InputStream is =
    Thread.currentThread().getContextClassLoader().getResourceAsStream(name)

Enumeration<URL> urls =
    Thread.currentThread().getContextClassLoader().getResources(name);
```

MODULE SYSTEM MECHANICS - JDEPS TOOL

How do I know in which module a certain class lives?

```
$ jdeps -s -M -include-system-modules java.sql.Timestamp
java.sql -> java.base
```

Dependencies for the application module

```
$ jdeps -s -mp build build/se.callista.java9.modules.app1.jar
se.callista.java9.modules.app1.jar -> java.base
se.callista.java9.modules.app1.jar -> java.sql
se.callista.java9.modules.app1.jar -> not found
java.sql -> java.base
java.sql -> java.logging
java.sql -> java.xml
```

← se.callista.java9.modules.common.jar