Approaching: The Delayed Train to Modularity "Java 9" Victory at Last?

https://marketplace.eclipse.org/content/java-9-support-beta-oxygen

Stephan Herrmann
Preface: Time Frame

**Release Date**

- February 13\textsuperscript{th} 2015 — “Java 8.5”
- September 22\textsuperscript{nd} 2016
- March 23\textsuperscript{rd} 2017
- July 27\textsuperscript{th} 2017
- September 21\textsuperscript{st} 2017

**JDT**

- Started work on Java 9 in **early 2015**
  - Disclaimer: I've long stood at the side line

**Specification**

- First draft: December 2016
- Acceptable draft: May 2017
JPMS – “Jigsaw”

Levels of impact

▸ New syntax to define modules
▸ Semantics
  ▸ Compile time
  ▸ Runtime
▸ New formats and tools to create and work with modules
▸ The JDK will be modularized
Module Syntax vs. OSGi Manifest

**module-info.java**

- `module org.m1 { ...`
- `requires org.m2;`
- `requires transitive org.m3;`
- `exports org.pack1;`
- `exports org.pack2 to org.m2;`
- `provides org.Ifc with org.Impl;`
- `uses org.Ifc;`

**MANIFEST.MF**

- `Bundle-SymbolicName: org.m1`
- `Require-Bundle: org.m2`
- `org.m3;visibility:=reexport`
- `Export-Package: org.pack1 org.pack2; x-friends="org.m2"`

**service.xml**

```
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="IMyService">
  <implementation class="org.Impl"/>
  <service>
    <provide interface="org.Ifc"/>
  </service>
</scr:component>
```
DEMO: Modules in Eclipse
Sun Shine & Roses?

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

- Module declarations: #ModuleNameSyntax ✓ · #ModuleNameCharacters ✓ · #CompileTimeDependences ✓ · #ModuleAnnotations ✓ · #ModuleDeprecation ✓ · #ExportAnnotation ✓ · #CompilationWithConcealedPackages ? · #ResolutionAtCompileTime ? · #RestrictedKeywords ?

- Module artifacts: #MultiModuleExecutableJARs ✓ · #MultiModuleJARs ✓ · #ReifiedModuleGraphs ✓ · #AddExportsInManifest ✓

- Module descriptors: #ClassFileModuleName ✓ · #ClassFileAccPublic ✓ · #ClassFileAccModule ✓ · #StandardModuleAttributes ✓

- Automatic modules: #CustomizableAutomaticModuleNameMapping ✓ · #ModuleNameInManifest ✓ · #AutomaticModuleNames ✓

- Module graphs: #CyclicDependences ✓ · #MutableConfigurations ✓ · #LazyConfigurationAndInstantiation ✓

- Services: #ServiceLoaderEnhancements ✓

- Reflection: #ClassFilesAsResources ✓ · #ResourceEncapsulation ✓ · #ResourceExistenceAndSize ✓ · #ReflectiveAccessToNonExportedTypes ✓ · #Readability ✓ · #ReadabilityAddedByLayerCreator ✓ · #ReifiedReflectiveAccess ✓

- Class loaders: #AvoidConcealedPackageConflicts ✓ · #PlatformClassLoader ✓ · #ClassLoaderNames ✓

- Versioning: #StaticLayerConfiguration ✓ · #MultipleModuleVersions ✓ · #VersionsInModuleName ✓ · #VersionedDependences ✓ · #VersionSyntax ✓ · #ModuleIdentifiers ✓

- Layers: #NonHierarchicalLayers ✓ · #DiscardableModules ✓ · #LayerPrimitives ✓

- Tooling: #BootstrapClassLoaderSearchInJVMTI ✓ · #ReflectiveAccessByInstrumentationAgents ✓

Stephan Herrmann @Oxygen Demo Camp München – published under the EPL
jlink

-create a modular image out of class files, modular jars & jmods

URLs: what’s the result of: ClassLoader.getSystemResource("java/lang/Class.class")

Java 8: jar:file:/usr/local/jdk8/jre/lib/rt.jar!/java/lang/Class.class
Java 9: jrt:/java.base/java/lang/Class.class

- accessible only using java.nio.file.FileSystem
- internal, undocumented format
- jmod: single module
- jimage: aggregated runtime image
Possible to deploy only some modules
❯ smaller footprint for devices with limited resources

JDK internals will be protected by module boundary
❯ Infamous example: `sun.misc.Unsafe`
❯ *should not use* → *cannot use* (eventually)

Java 9: new deprecated module `jdk.unsupported`

Java 10
❯ some parts will become public API
❯ other parts will be fully encapsulated / hidden

Some language changes are in fact motivated by JRE concerns.

```
jdeps -jdkinternals ...
```
Jigsaw vs. OSGi

**OSGi but not Jigsaw:**
- **Each** bundle has its own class loader, always
- Cycles are allowed
- Versioning!!

**Jigsaw but not OSGi:**
- *Layers optionally map modules to class loaders*
- Module boundaries enforced also for reflective access
- **requires static** for compile time dependency
- *Missed the chance to prohibit API leaks*

Enable better optimization.
JEP 277: Enhanced Deprecation
JEP 226: UTF-8 Property Files
JEP 213: Milling Project Coin
  ❯ Allow @SafeVargs on private instance methods.
  ❯ Allow effectively-final variables to be used as resources in the try-with-resources statement.
  ❯ Allow diamond with anonymous classes if the argument type of the inferred type is denotable.
  ❯ Complete the removal, begun in Java SE 8, of underscore from the set of legal identifier names.
  ❯ – see next slide –

JEPs to improve implementation and test coverage of javac...
In Java 9, one of the following will become a legal modifier for methods in an interface.

Which one?

- static
- private
- synchronized
- optional
Multi Release Jars (JEP 238)

“Summary
Extend the JAR file format to allow multiple, Java-release-specific versions of class files to coexist in a single archive.”

```
jar root
- A.class
- B.class
- C.class
- D.class
- META-INF
  - versions
    - 9
      - A.class
      - B.class
    - 10
      - A.class
```
Project Valhalla

- Value Types
  - accessible
  - “codes like a class, works like an int”
  - provide value performance for user-defined structures
  - example: array-of-int vs. array of Integer: page misses!

- Specializing Generics
  - support value types as generic type arguments
  - erasure-based implementation no longer works, because value types <: java.lang.Object
  - borrow from C++ template expansion
  - let VM perform the expansion / specialization on demand

Project Amber

- JEP 286 Local-Variable Type Inference
- JEP 301 Enhanced Enums
- JEP 302 Lambda Leftovers