JDT embraces Java™ 9
—
An insider's view

Stephan Herrmann

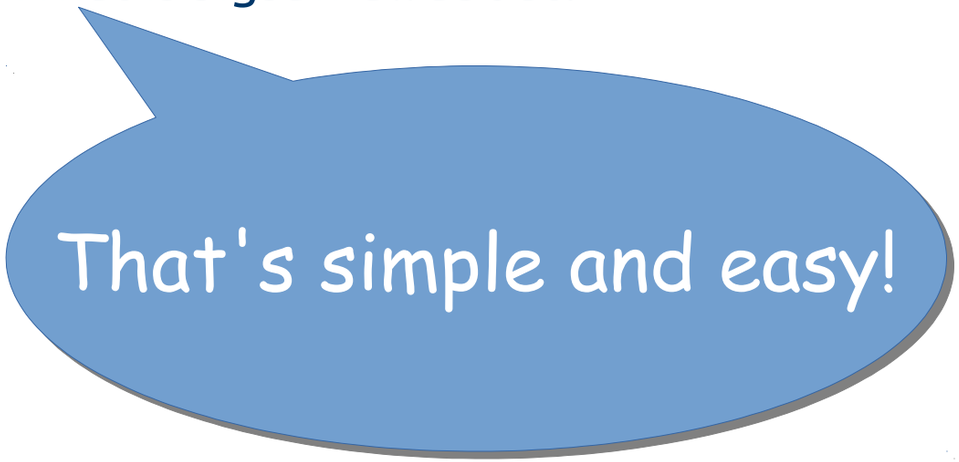
GK SOFTWARE

Simply Retail.

JDT embraces Java™ 9

A Play in 4 Acts

- What is Jigsaw all about?



Simple: 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
(YourProject > Configure > Create module-info.java)

A closer look at module-info
–
from a tool smith's perspective

JDT API (1):

- org.eclipse.jdt.core.ToolFactory.createScanner(..)
- org.eclipse.jdt.core.compiler.IScanner
  - void resetTo(int startPosition, int endPosition)
  - int getNextToken()
- org.eclipse.jdt.core.compiler.ITerminalSymbols.TokenNameIdentifier

In Java 9 this API is beyond repair

- restricted keywords:
  - “keywords that are keywords when they are keywords, else identifiers” [my words]
  - classification keyword / identifier happens after parsing

JDT API (1):

- org.eclipse.jdt.core.ToolFactory.createS
- org.e
  -
  -
- ~~org.e~~

- In Java
  - rest
    -
    -

```
import to.uses;
import to.module;
import to.to;

module module {
   requires requires; exports
   to to exports; uses module;
   provides uses with to;
   requires transitive;
}
```

technically clean solution exists:
^
but has been rejected

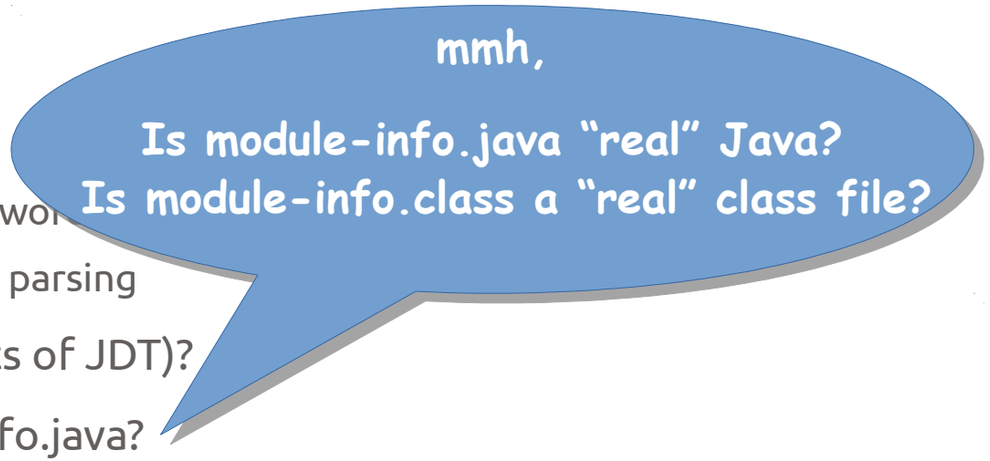
" [my words]

JDT API (1):

- org.eclipse.jdt.core.ToolFactory.createScanner(..)
- org.eclipse.jdt.core.compiler.IScanner
  - void resetTo(int startPosition, int endPosition)
  - int getNextToken()
- ~~org.eclipse.jdt.core.compiler.ITerminalSymbols.TokenNameIdentifier~~

In Java 9 this API is beyond repair

- restricted keywords:
  - "keywords that are keywords when they are keywo
  - classification keyword / identifier happens after parsing
- should we remove this API (and break all clients of JDT)?
- just hope that nobody will use it for module-info.java?



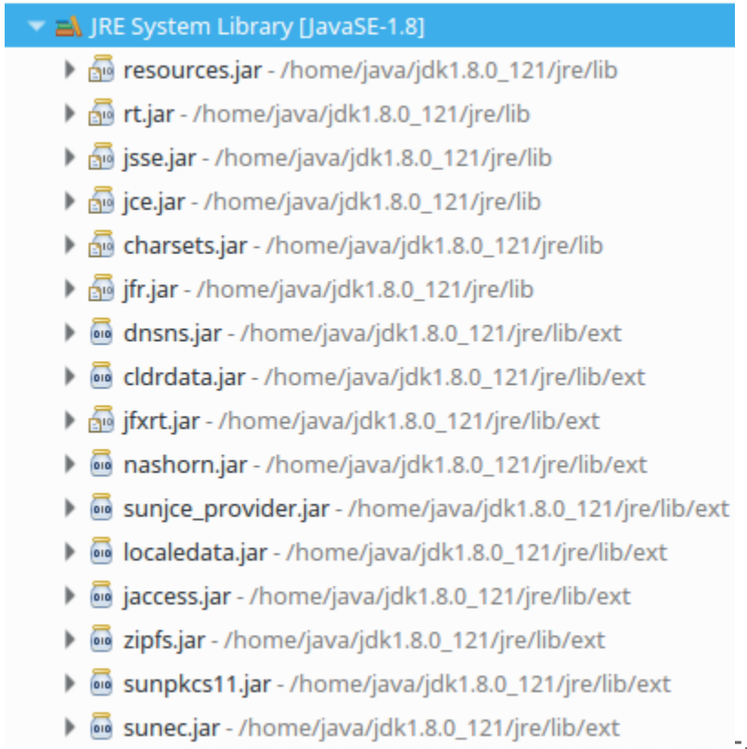
JDT API (2):

- **@Deprecated**
  ~~org.eclipse.jdt.core.IClassFile.getType()~~
  - will never answer null
  - unless the class file is module-info.class!
- clients need to know the difference:
  - org.eclipse.jdt.core.IOrdinaryClassFile
  - org.eclipse.jdt.core.IModularClassFile

Another damaged API

JDT API (3):

- org.eclipse.jdt.core.**IClasspathContainer**
  - e.g., JRE System Library
  - resolves to *n* **IClasspathEntry**;s
  - each resolved entry representing a library or project, here: jar



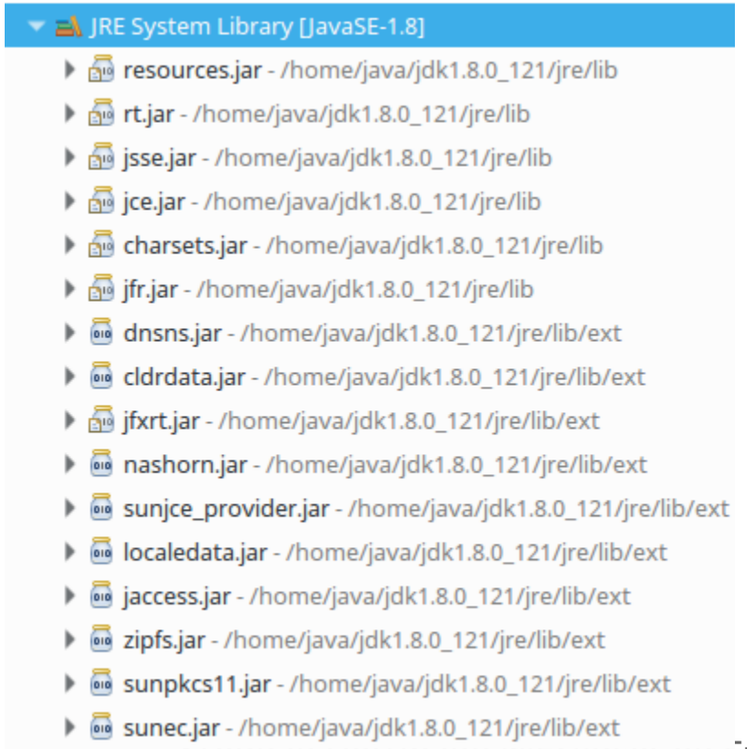
Act 2.5: Something's weird

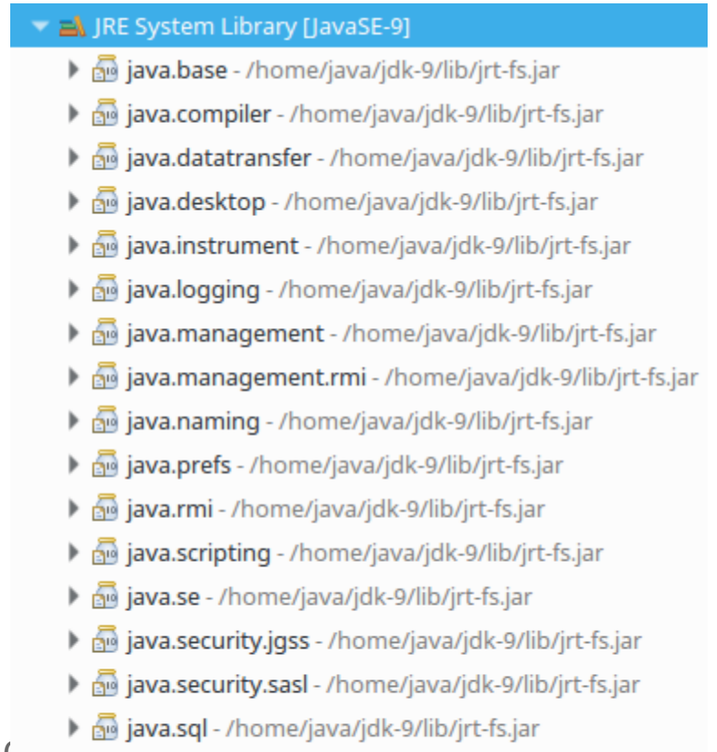
JDT API (3):

- org.eclipse.jdt.core.**IClasspathContainer**
  - e.g., JRE System Library
  - resolves to *n* **IClasspathEntry**;s
  - ~~each resolved entry representing a library or project, here: jar~~

Format changed:

no longer uses jars





Act 2.5: Something's weird

JDT API (3):

- org.eclipse.jdt.core.**IClasspathContainer**
  - e.g., JRE System Library
  - resolves to *n* **IClasspathEntry**;s
  - ~~each resolved entry representing a library or project, here: jar~~
- org.eclipse.jdt.core.IJavaProject.findPackageFragmentRoots(IClasspathEntry)
  - still works ☺
  - returns **IPackageFragmentRoot[]** even for single resolved JRE entry
  - each IPackageFragmentRoot corresponds to one module

Format changed:

no longer uses jars

End of Act 2

There is complexity

—

Questions:

What is the meaning of **a.b.C1**?

Is the reference **a.b.C1** legal?

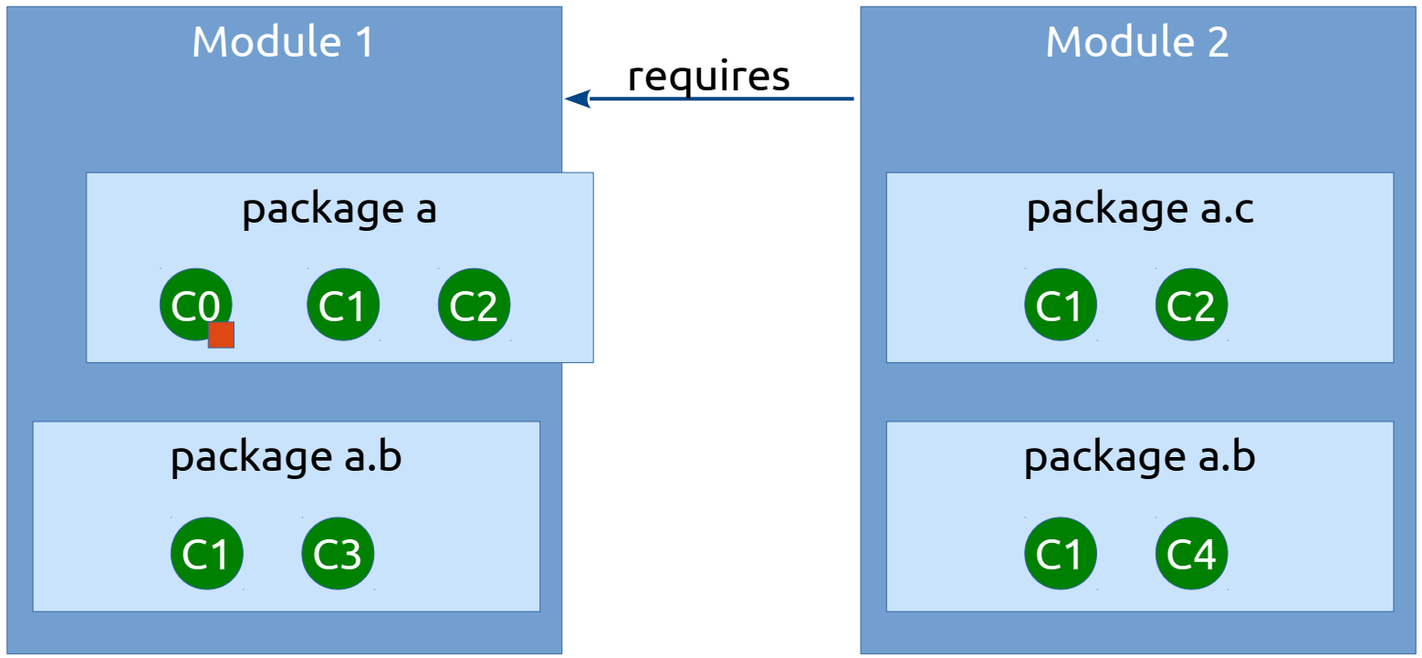
Does Java 9 support **split packages**?

Act 3: Characters

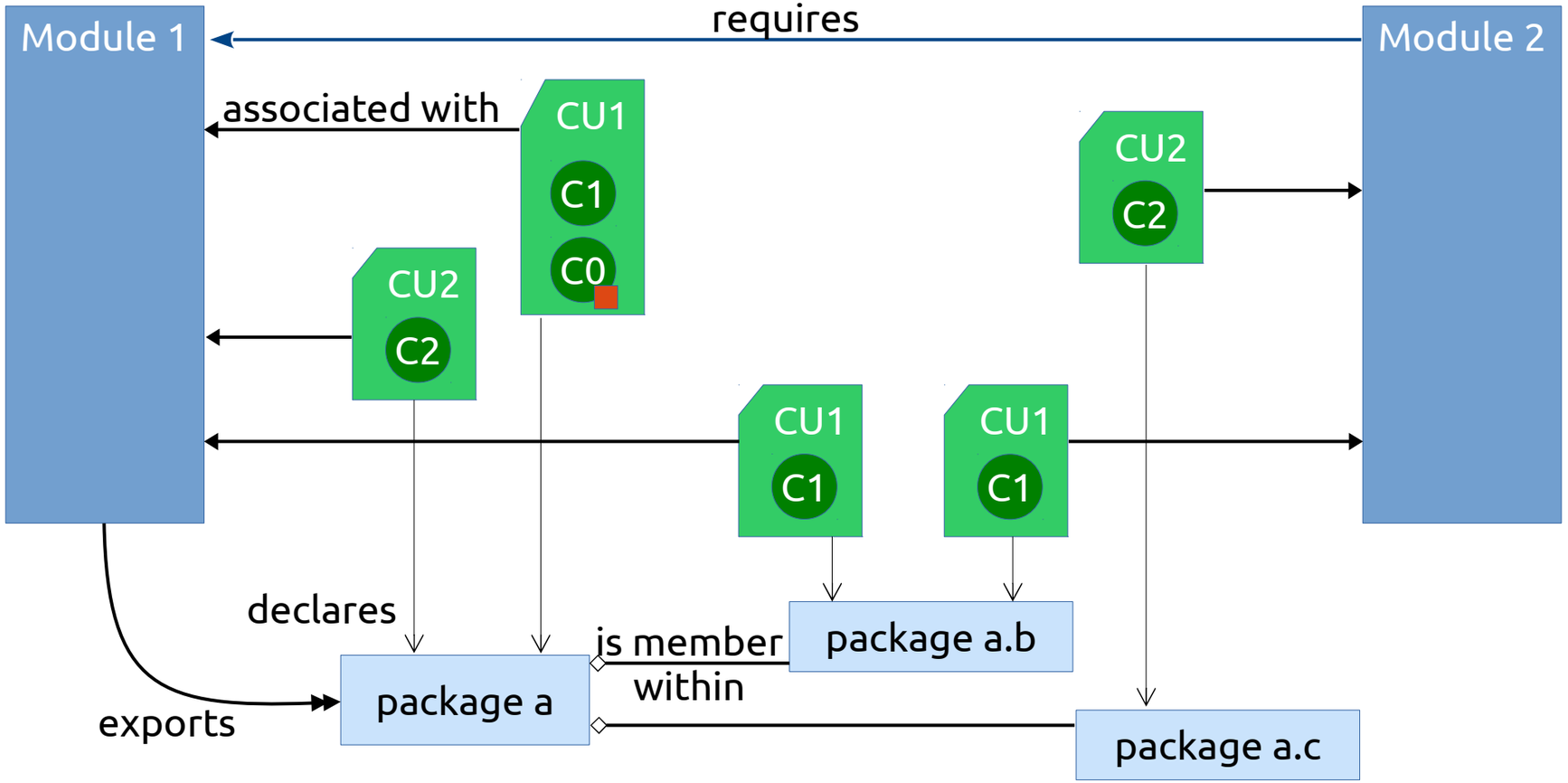
- Java Language Specification (JLS)
  - everything that a compiler engineer needs to know (?)
- Java Virtual Machine Specification (JVMS)
  - the fine print (class file format)
- JEP 261 (“Module System”)
  - see here for command line options
- JDK API Documentation
  - find here the platform aspect of Jigsaw

JPMS

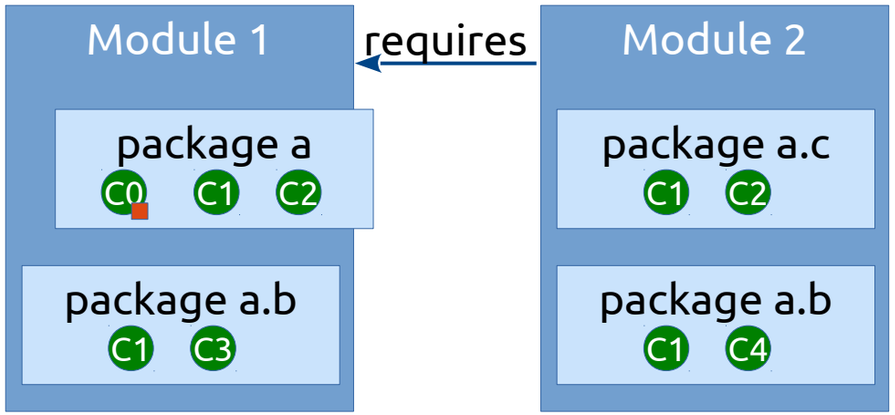
- How we want to see the world:

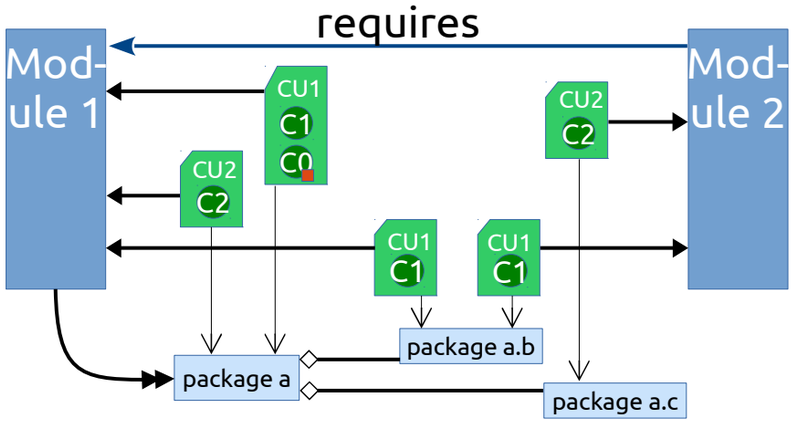


- This is the world according to JLS:



Act 3: unnecessary complexity

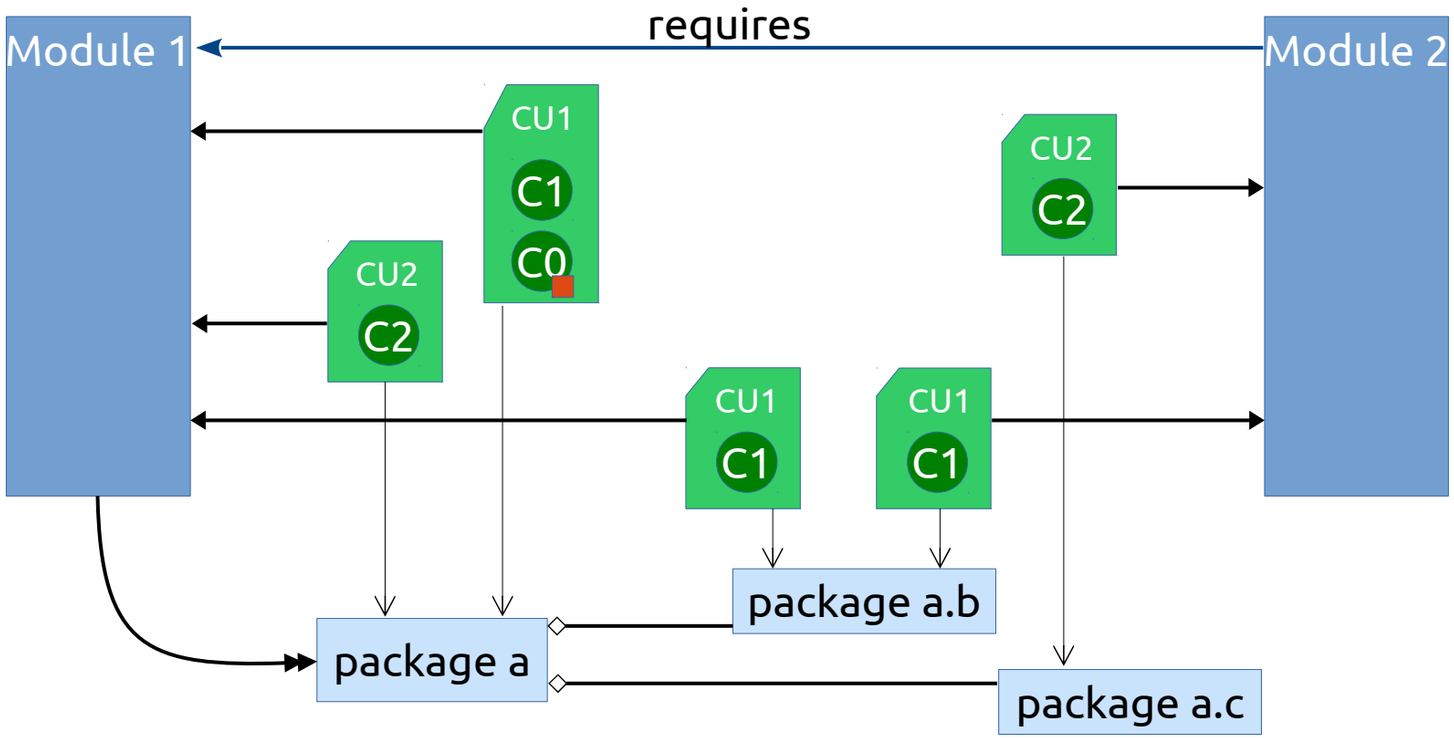




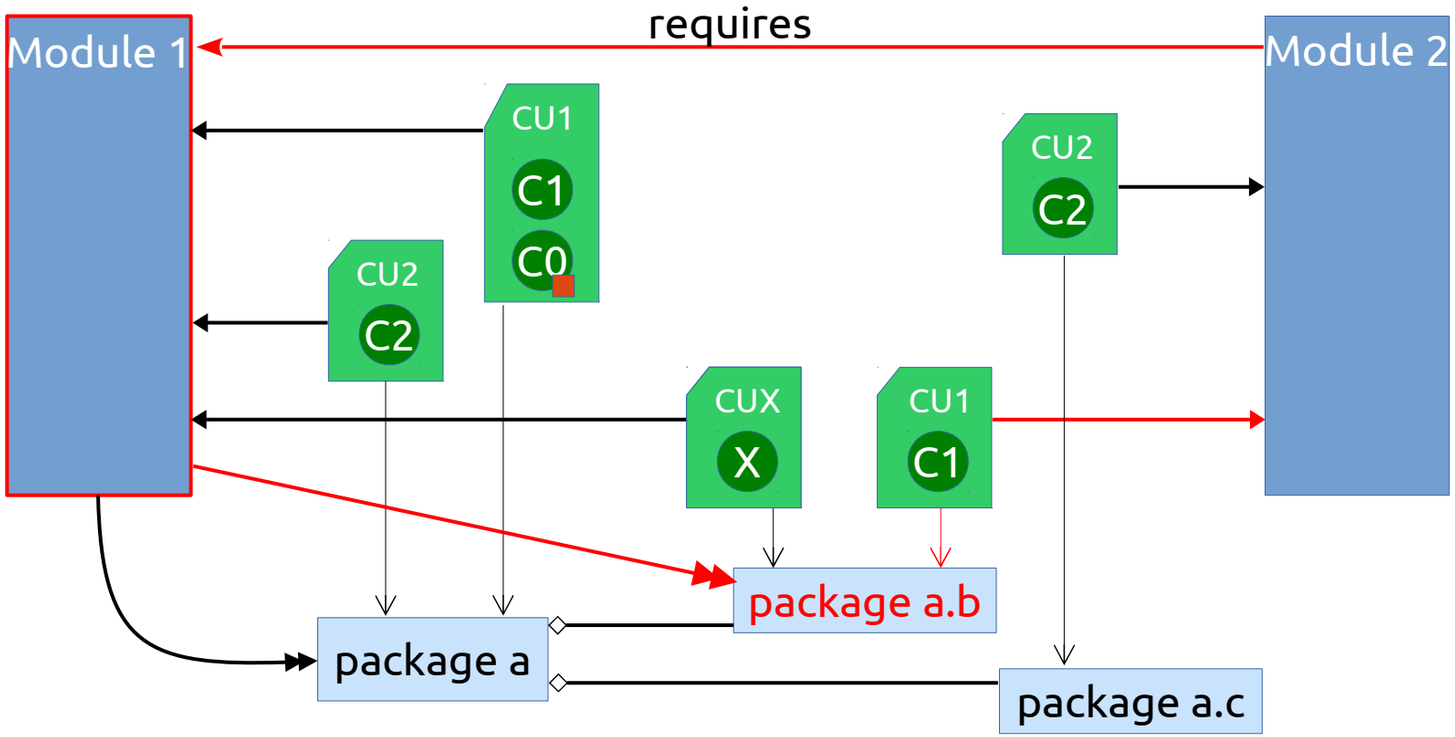
- Containment
  - module
  - package
  - class
- Independent
  - packages among each other

- Emulation by indirect relations
  - cu is associated to module
  - cu declares package
  - cu contains class
- Containment
  - package
  - sub-package

- Inside Module2, what does a.b.C1 mean?
  - a.b.C1 is found in the CU1 associated with Module2 ✔
  - package a.b is **uniquely visible** from Module2 ✔

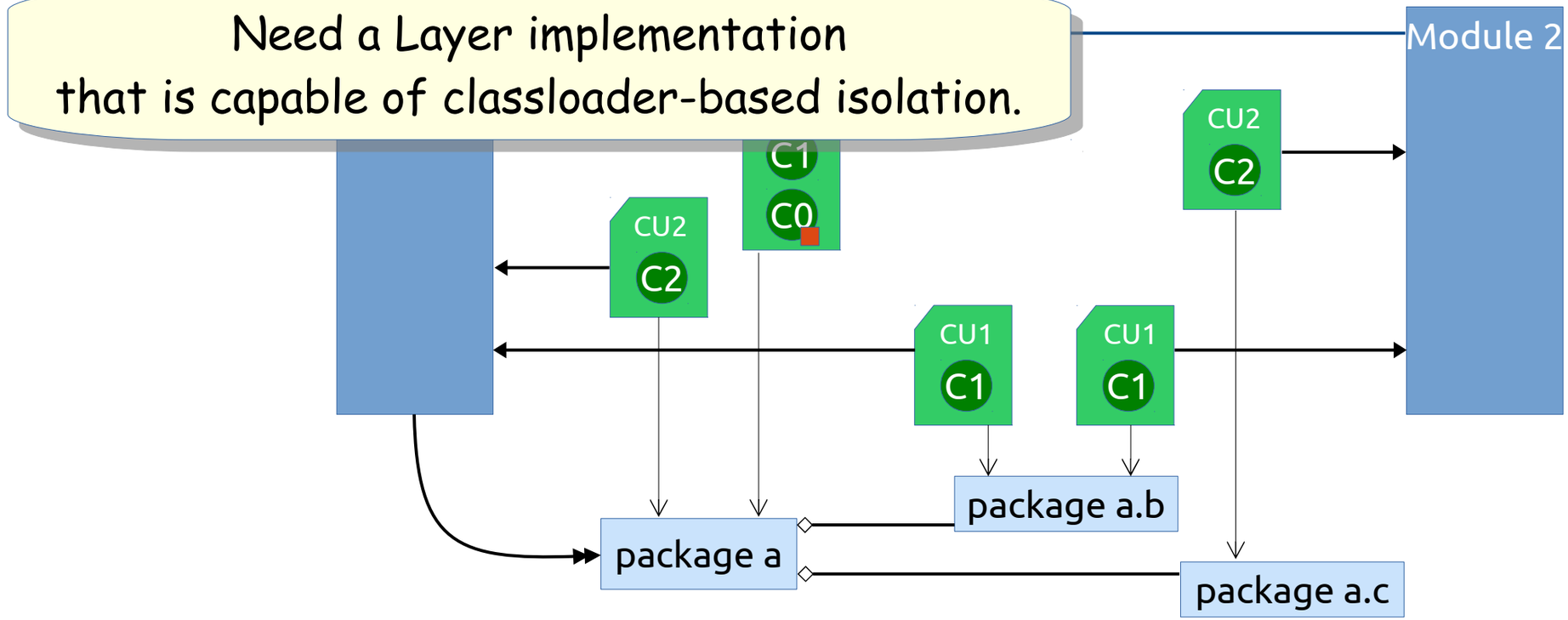


- Inside Module2, what does a.b.C1 mean?
  - package a.b is not **uniquely visible,** could be from Module1 or from Module2
  - this configuration is illegal



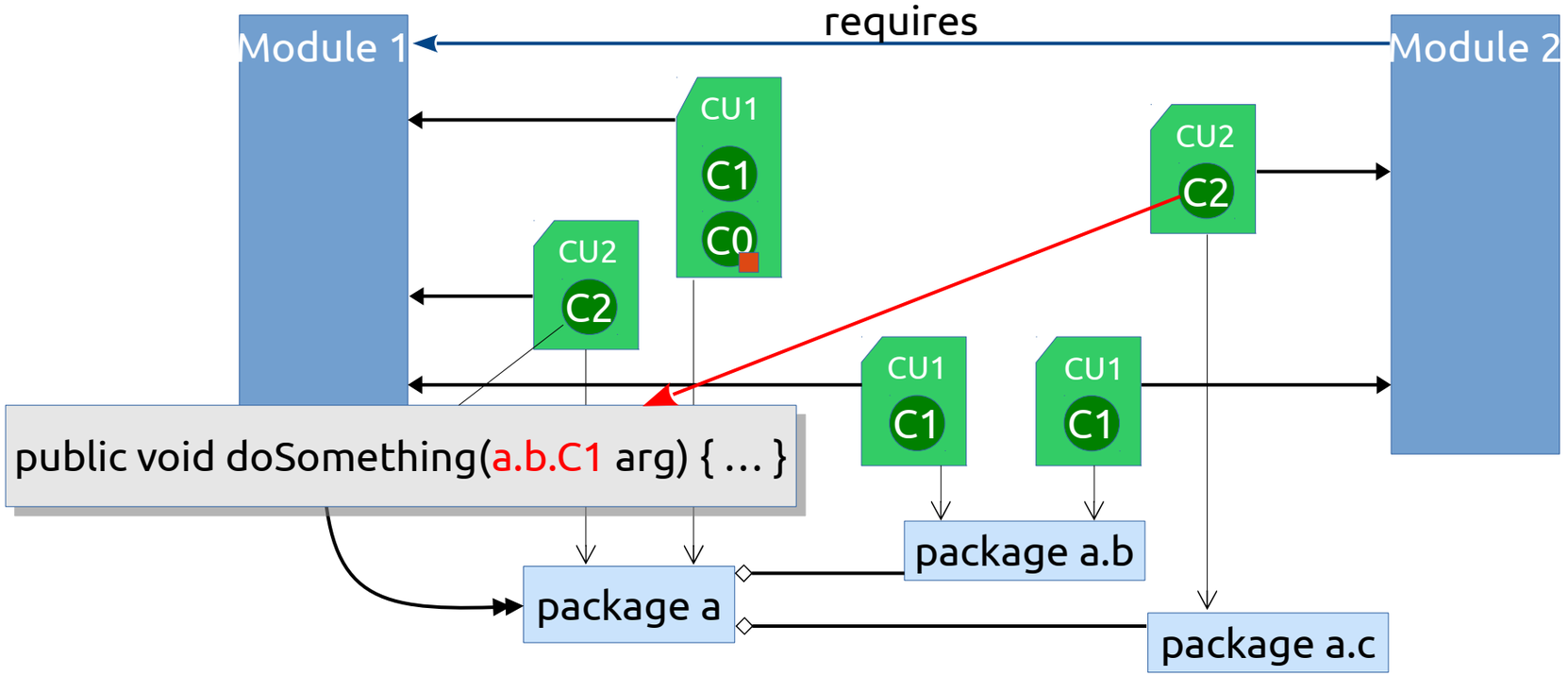
```
$ java -m Module2
Error occurred during initialization of boot layer
java.lang.LayerInstantiationException: Package a.b in both module Module1 and module Module2
$ _
```

Need a Layer implementation
that is capable of classloader-based isolation.



- API Leak
  - API mentions a type, which the client cannot mention
  - legal
  - new complexity in the implementation
  - put overloading and inheritance in the mix: infinite supply of new Java puzzlers

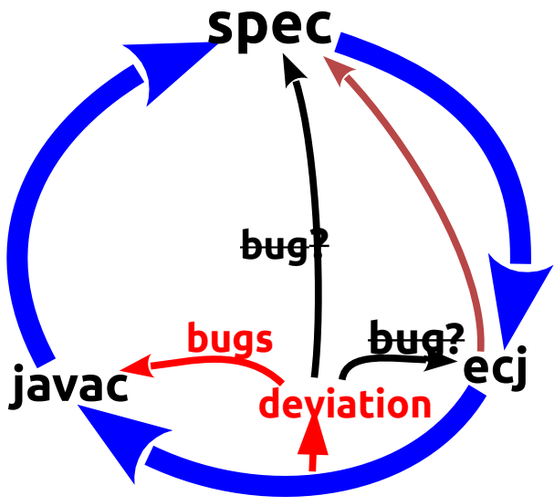
Missed Opportunity
- **API leaks should be illegal**
- **Or method should be unmentionable for clients**



Act 3: summary encapsulation

- Split packages
  - concealed: legal
    - but no promise that the runtime can handle it
    - need a capable Layer implementation
  - merging a package (like fragment & host bundle) is illegal
    - even if no class-level conflict is involved
- API leaks
  - JLS is silent
  - compilers issue warnings
    - attempts to coordinate warnings had little effect

- 2 Implementations & 1 Specification
  - JLS should be “boss”
  - Work on ecj is QA for Java
  - Put JLS to the center of every bug discussion
- JLS 9
  - Quality
  - Time



End of Act 3

A simple surface
realized by complexity

There's a lot more

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 ✔ **#AwkwardStrongEncapsulation** flectionWithoutReadability ✔ · #ReadabilityAddedByLayerCreator ✔ · #In veModuleAndLayerClasses ✔
- Class loaders: #AvoidConcealedPackageConflicts ✔ · #PlatformClassLoader ✔ · #ClassLoaderNames ✔
- Versioning: #StaticLayerConfiguration ✔ · #MultipleModuleVersions ✔ · #VersionsInModuleNames ✔ · #VersionedDependences ✔ · #VersionSyntax ✔ · #ModuleIdentifiers ✔
- Layers: #NonHierarchicalLayers ✔ · #DiscardableModules ✔ · #LayerPrimitives ✔
- Tooling: #BootstrapClassLoaderSearchInJVMTI ✔ · #ReflectiveAccessByInstrumentationAgents ✔

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 ✔ · #ResourceEncaps… …dSize ✔ · #ReflectiveAccessToNonExportedTypes ✔ **#Awkw…** …ctionWithoutReadability ✔ · #ReadabilityAddedByLayerCreator ✔ · #In… …veModuleAndLayerClasses ✔
- Class loaders: #AvoidConcealedPa… …latformClassLoader ✔ · #ClassLoaderNames ✔
- Versioning: #StaticLayerConfigurati… · #MultipleModuleVersions ✔ · #VersionsInModuleNames ✔ · #VersionedDependences ✔ · #VersionSyntax ✔ · #ModuleIdentifiers ✔
- Layers: #NonHierarchicalLayers ✔ · #DiscardableModules ✔ · #LayerPrimitives ✔
- Tooling: #BootstrapClassLoaderSearchInJVMTI ✔ · #ReflectiveAccessByInstrumentationAgents ✔

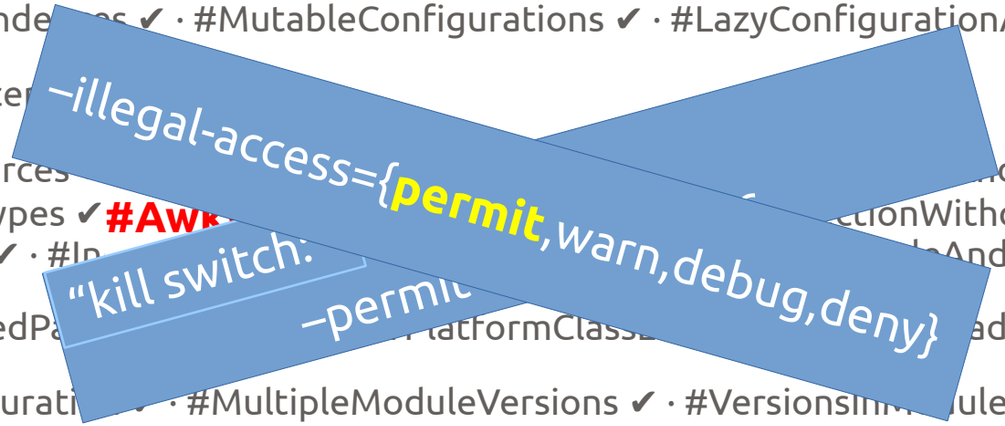
"kill switch:"

–permit-illegal-access

Sun Shine & Roses?

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

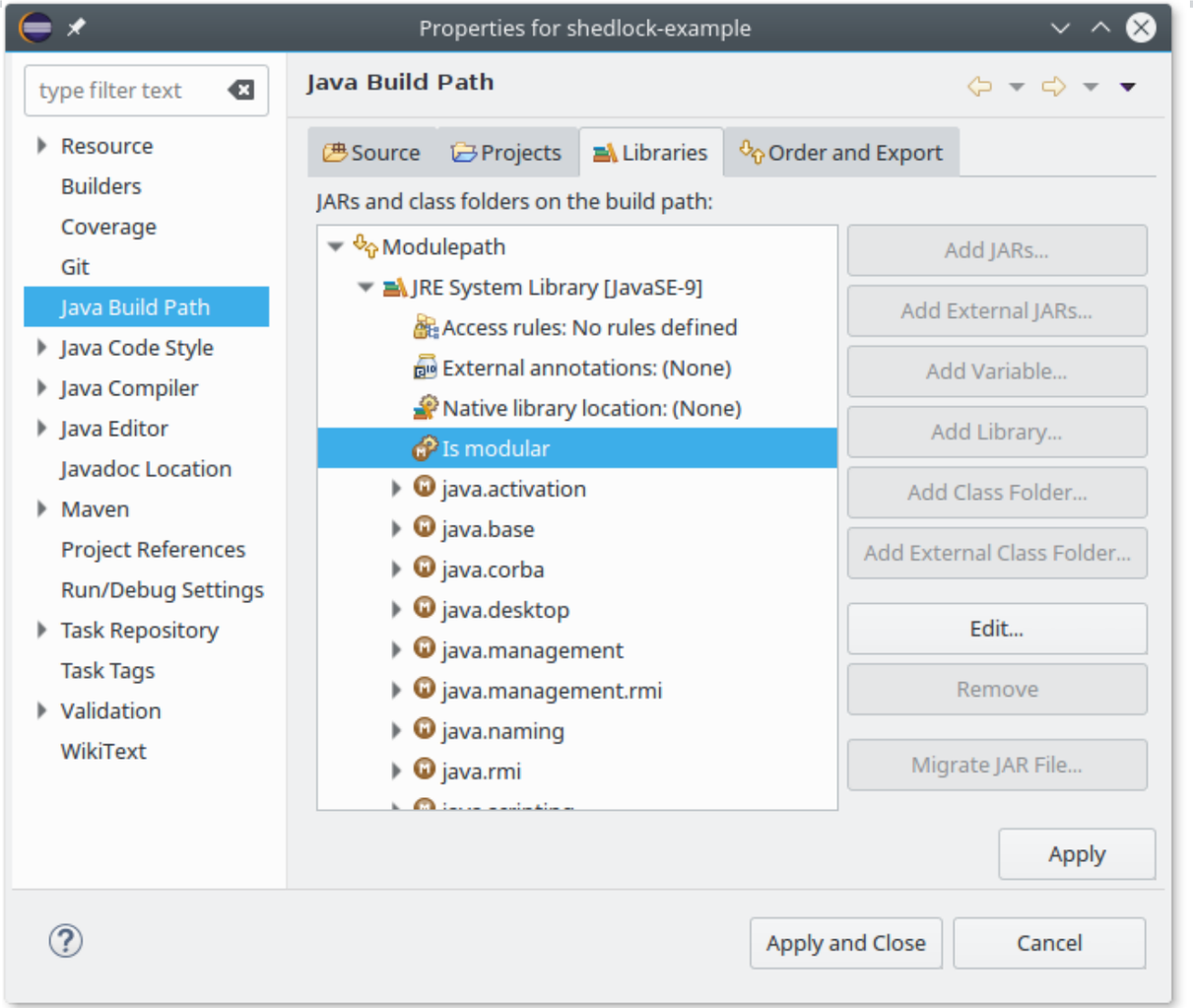
- Module declarations: #ModuleNameSyntax ✔ · #ModuleNameCharacters ✔ · #CompileTimeDependences ✔ · #ModuleAnnotations ✔ · #ModuleDeprecation ✔ · #ExportAnnotation ✔ · #**CompilationWithConcealedPackages** ? · #**ResolutionAtCompileTime** ? · #**RestrictedKeywords** ?
- Module artifacts: #MultiModuleExecutableJARs ✔ · #MultiModuleJARs ✔ · #ReifiedModuleGraphs ✔ · #AddExportsInManifest ✔
- Module descriptors: #ClassFileModuleName ✔ · #ClassFileAccPublic ✔ · #ClassFileAccModule ✔ · #StandardModuleAttributes ✔
- Automatic modules: #CustomizableAutomaticModuleNameMapping ✔ · #ModuleNameInManifest ✔ · #AutomaticModuleNames ✔
- Module graphs: #CyclicDepende…s ✔ · #MutableConfigurations ✔ · #LazyConfigurationAndInstantiation ✔
- Services: #ServiceLoaderEnhancer…
- Reflection: #ClassFilesAsResources…dSize ✔ · #ReflectiveAccessToNonExportedTypes ✔ **#Aw…** …tionWithoutReadability ✔ · #ReadabilityAddedByLayerCreator ✔ · #In… …AndLayerClasses ✔
- Class loaders: #AvoidConcealedPa… …latformClass… …aderNames ✔
- Versioning: #StaticLayerConfigurati… · #MultipleModuleVersions ✔ · #VersionsIn… …uleNames ✔ · #VersionedDependences ✔ · #VersionSyntax ✔ · #ModuleIdentifiers ✔
- Layers: #NonHierarchicalLayers ✔ · #DiscardableModules ✔ · #LayerPrimitives ✔
- Tooling: #BootstrapClassLoaderSearchInJVMTI ✔ · #ReflectiveAccessByInstrumentationAgents ✔



Let's map command line options to UI

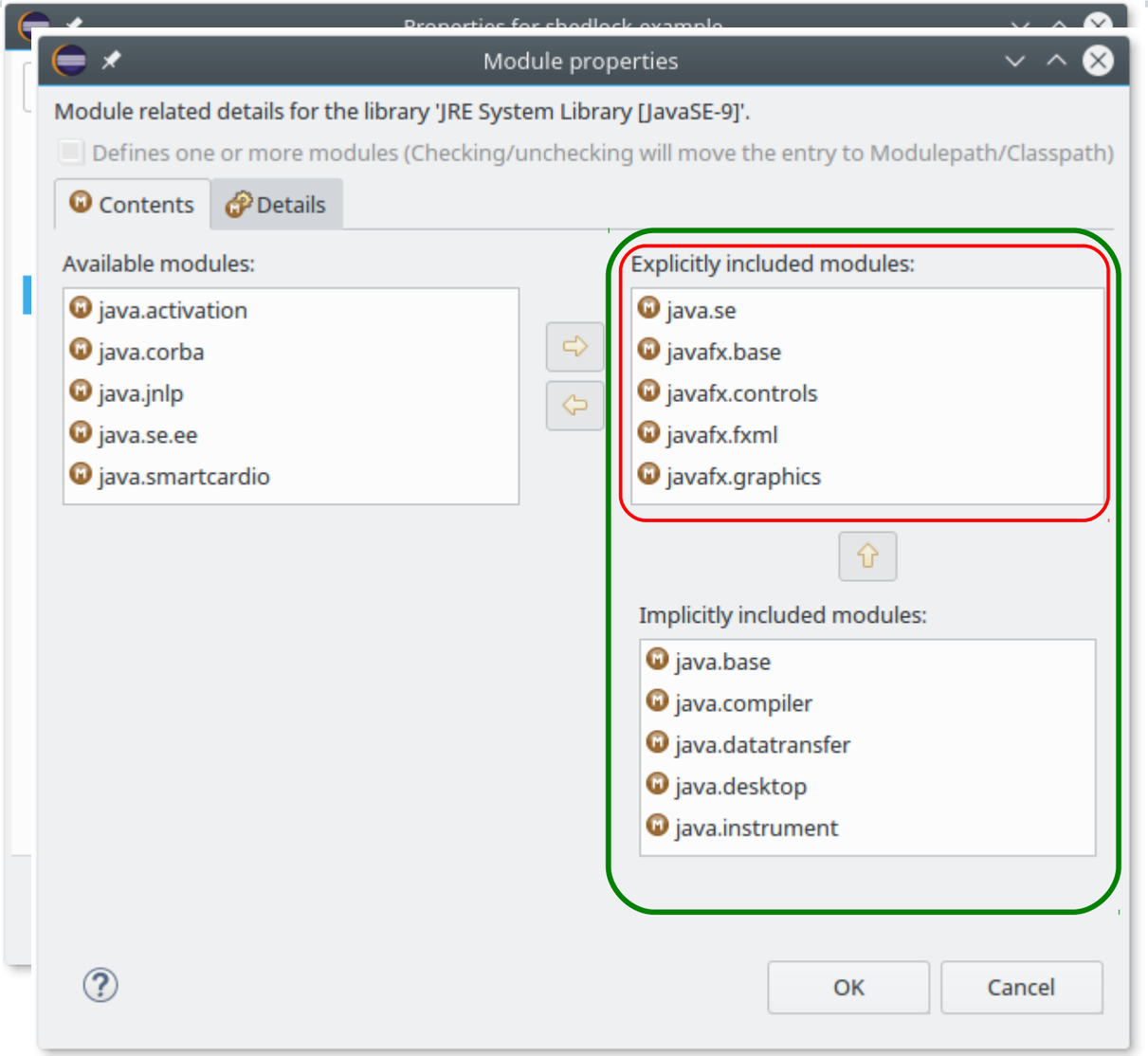
Act 4: Options

- “Is modular”
  - Edit: enter the rabbit hole



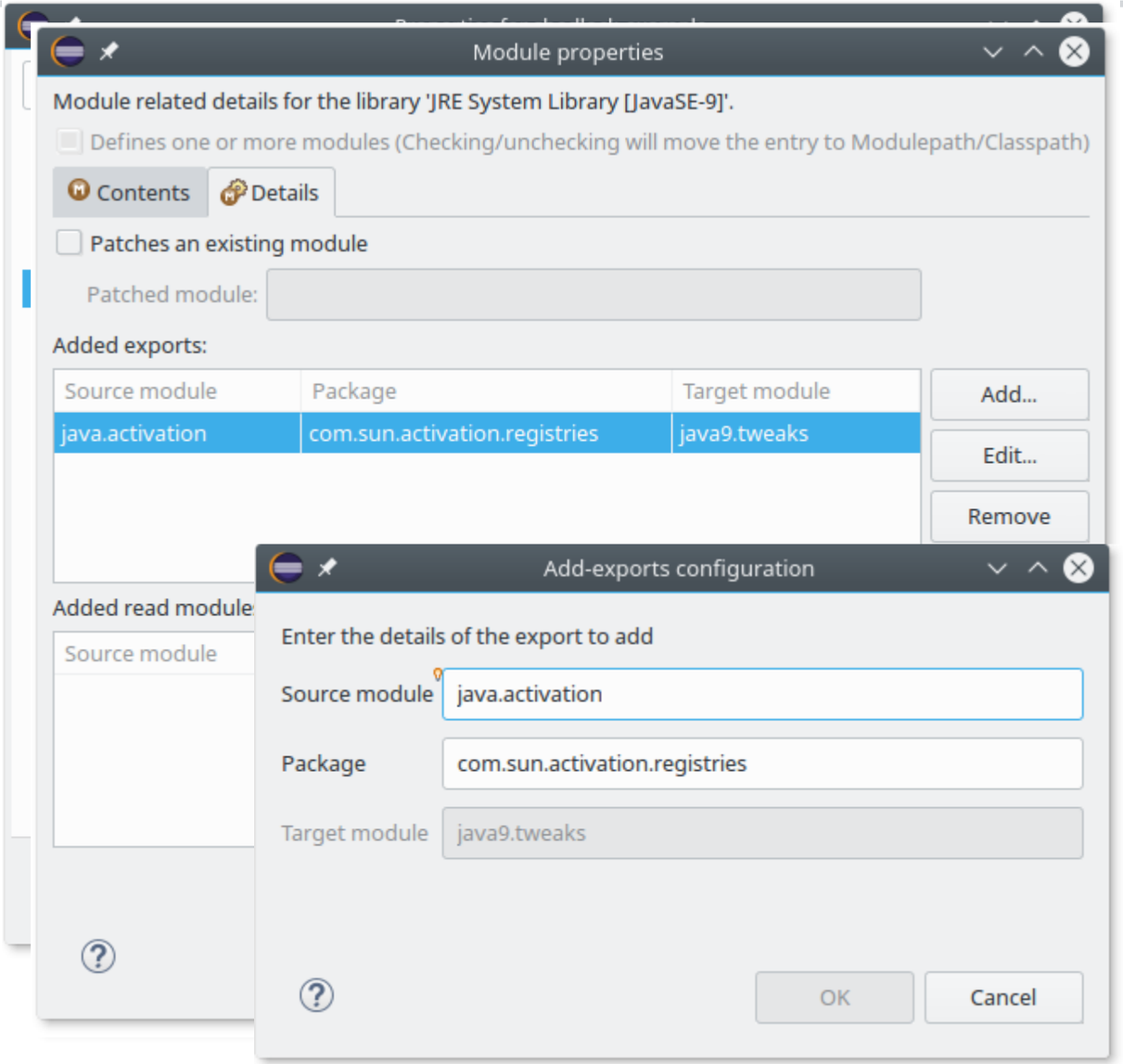
Act 4: Options

- “Is modular”
  - Edit: enter the rabbit hole
- Contents
  - what you ask for
  - what you get
  - --limit-modules
  - default as per JEP 261



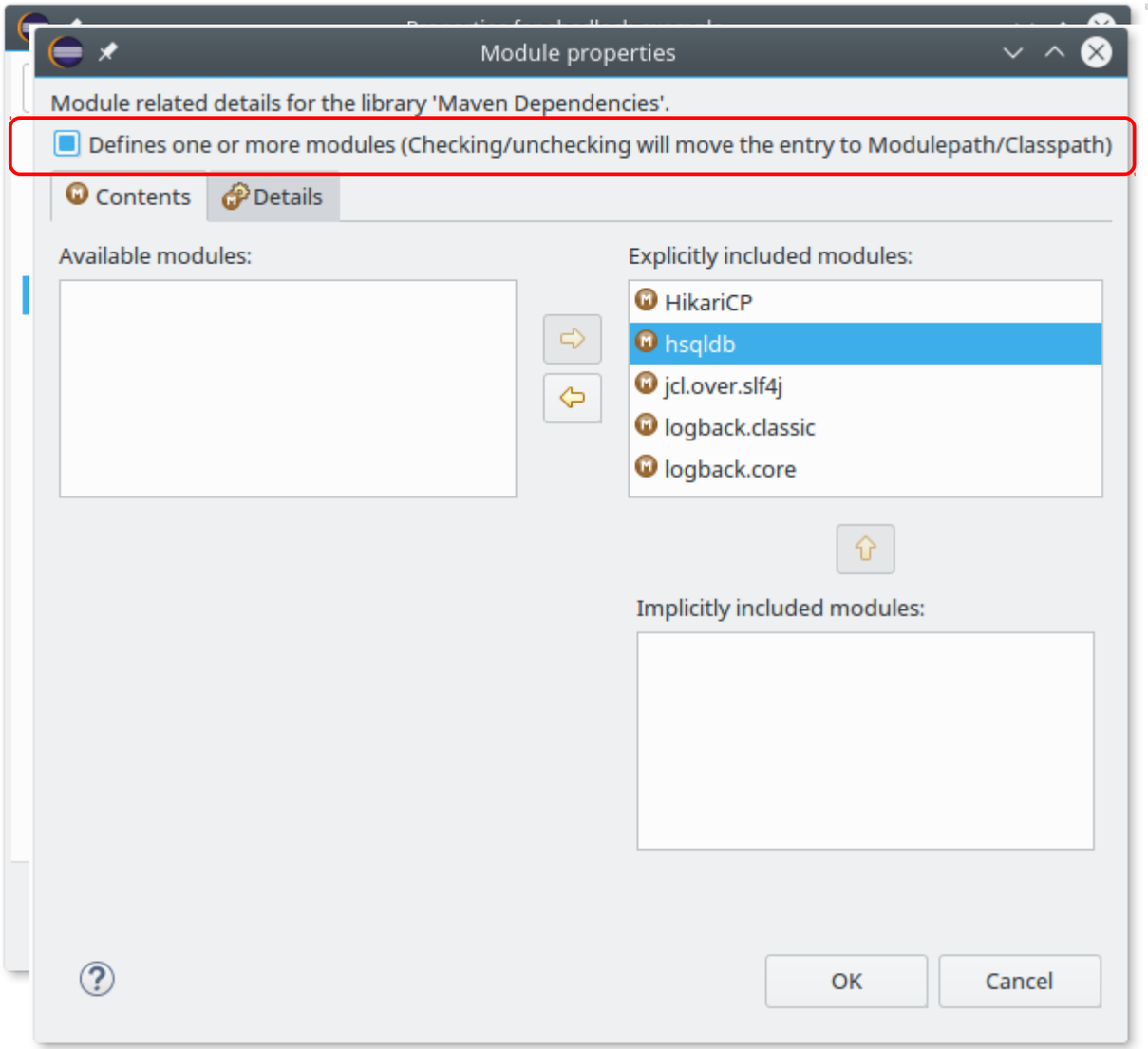
Act 4: Options

- “Is modular”
  - Edit: enter the rabbit hole
- Contents
  - what you ask for
  - what you get
  - --limit-modules
  - default as per JEP 261
- Details
  - --add-exports
  - --add-reads (rarely needed)
  - --patch-module (careful!)



Act 4: Options

- “Is modular”
  - Edit: enter the rabbit hole
- Contents
  - what you ask for
  - what you get
  - --limit-modules
  - default as per JEP 261
- Details
  - --add-exports
  - --add-reads (rarely needed)
  - --patch-module (careful!)
- Power switch



Epilogue

- Act 4: Modifying Encapsulation?
  - “Gradual Encapsulation” [Herrmann 2008]
    - encapsulation should be your friend
    - sometimes it's your enemy
    - technology vs. stakeholders
    - enforcement & negotiation
- Act 3: Isolation?
  - Separate namespace per module?
    - partly achieved / emulated
    - several opportunities missed
- Act 2: Compatibility?
  - Java 9 is disruptive
    - against frameworks and applications
    - against tools

- Act 4: Modifying Encapsulation?
  - “Gradual Encapsulation” [Herrmann 2008]
    - encapsulation shou
    - sometimes
    - technology
    - enforceme
- Act 3: Isolation?
  - Separate nam
    - partly achieved / emulated
    - several opportunities missed
- Act 2: Compatibility?
  - Java 9 is disruptive
    - against frameworks and applications
    - against tools

coming soon:

```
@NonNullByDefault
module org.eclipse.mymodule {
        …
}
```