JDT embraces Java™ 9
—
An insider's view

Stephan Herrmann
JDT embraces Java™ 9

A Play in 4 Acts
What is Jigsaw all about?

That's simple and easy!
**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

```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>
```
Act 1: That's simple and easy!

DEMO: Modules in Eclipse
(YourProject > Configure > Create module-info.java)
Expect a slight sense of surprise

A closer look at module-info – from a tool smith's perspective
Act 2: Something's weird

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
Act 2: Something's weird

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

Is module-info.class a "real" class file?

- 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
Act 2: Something's weird

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"
  - 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?

mmh,

Is module-info.java “real” Java?
Is module-info.class a “real” class file?
Act 2: Something's weird

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
Act 2.5: Something's weird

JDT API (3):

- org.eclipse.jdt.core.IClasspathContainer
  - e.g., JRE System Library
  - resolves to \( n \) IClasspathEntry;
  - 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;
    - 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):

```java
go.org.eclipse.jdt.core.\*IClasspathContainer
  e.g., JRE System Library
  resolves to \*IClasspathEntry\*
    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
```
Act 2: Something's weird

End of Act 2
There is complexity

Questions:

What is the meaning of \texttt{a.b.C1}?

Is the reference \texttt{a.b.C1} legal?

Does Java 9 support \texttt{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
Act 3: Concepts

How we want to see the world:

Module 1

- Package a
  - C0
  - C1
  - C2

- Package a.b
  - C1
  - C3

Module 2

- Package a.c
  - C1
  - C2

- Package a.b
  - C1
  - C4

requires
Act 3: Realization

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

Stephan Herrmann @EclipseCon Europe 2017 – published under the EPL
Inside Module 2, 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
Act 3: Legal but broken

$ 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.
Act 3: Legal but asking for trouble

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

```java
public void doSomething(a.b.C1 arg) { ... }
```
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
Act 3: There is Complexity

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 ✔ · #AutomaticModuleName ✔

Module graphs: #CyclicDependences ✔ · #MutableConfigurations ✔ · #LazyConfigurationAndInstantiation ✔

Services: #ServiceLoaderEnhancements ✔

Reflection: #ClassFilesAsResources ✔ · #ResourceEncapsulation ✔ · #ResourceExistenceAndSize ✔ · #ReflectiveAccessToNonExportedTypes ✔ · #AwkwardStrongEncapsulation ✔ · #ReflectionWithoutReadability ✔ · #ReadabilityAddedByLayerCreator ✔ · #MoveModuleAndLayerClasses ✔

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 ✔ · #AutomaticModuleName ✔
- Module graphs: #CyclicDependences ✔ · #MutableConfigurations ✔ · #LazyConfigurationAndInstantiation ✔
- Services: #ServiceLoaderEnhancements ✔
- Reflection: #ClassFilesAsResources ✔ · #ResourceEncapsulation ✔ · #ResourceInfo ✔ · #ResourceFactor ✔ · #JARsAsResources ✔ · #JarAsResource ✔ · #ResolveJarAsResource ✔ · #ResourceWithNonDefaultJarAsResource ✔ · #ResourceWithoutReadability ✔ · #ReadabilityAddedByLayerCreator ✔ · #LayerAwareResource ✔ · #PerLayerResource ✔ · #PerResourceClass ✔ · #ResourceClass ✔ · #ResourceLayer ✔ · #ResourceJar ✔ · #ResourceJarLayer ✔ · #ResourceJarLayerClass ✔ · #ResourceJarLayerClassClass ✔ · #ResourceJarLayerClassClassClass ✔
- 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 ✓ · #ReflectiveAccessToNonExportedTypes ✓ · #ReadabilityAddedByLayerCreator ✓ · #LayerPrimitives ✓

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

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

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

- Tooling: 
  - #BootstrapClassLoaderSearchInJVMTI ✓ · #ReflectiveAccessByInstrumentationAgents ✓

Stephan Herrmann @EclipseCon Europe 2017 – published under the EPL
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

Stephan Herrmann @EclipseCon Europe 2017 – published under the EPL
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!)

Stephan Herrmann @EclipseCon Europe 2017 – published under the EPL
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
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 name space
- partly achieved / emulated
- several opportunities missed

Act 2: Compatibility?

Java 9 is disruptive
- against frameworks and applications
- against tools

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

Stephan Herrmann @EclipseCon Europe 2017 – published under the EPL