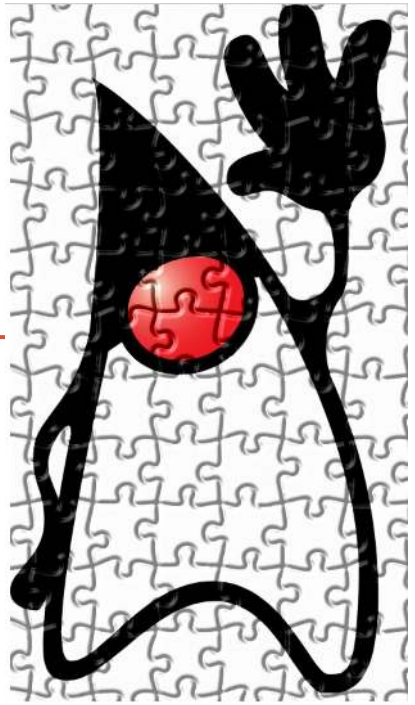
Using Java 9 Modules

Ryan Cuprak



J2D

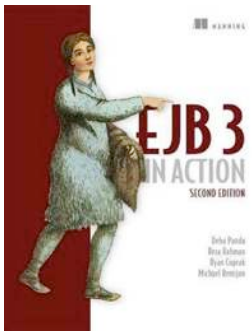
About

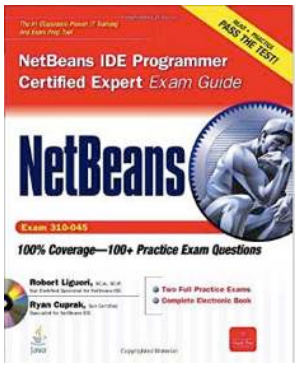
Twitter: @ctjava

Email: rcuprak@gmail.com / r5k@3ds.com

Blog: cuprak.info

Linkedin: www.linkedin.com/in/rcuprak





Project Jigsaw Goals

- Make the Java platform scalable for small computing devices.
- Improve platform security and maintainability
- Enable improved application performance
- Simplify library creation and application development

Reliable configuration
Strong Encapsulation

Why Modules?

- Java currently suffers from JAR ”hell”
- Maven tracks dependencies but no runtime enforcement
- No guarantee an application can start:
  NoClassDefFoundError
- Possible to mix library versions on classpath:
  - commons-io-2.5 and commons-io-1.6 – what happens?
- Existing module frameworks (OSGi) can’t be used to modularize the Java platform.

Project Jigsaw Pieces

- JEPs
  - 200: Modular JDK
  - 201: Modular Source Code
  - 220: Modular Run-time Images
  - 260: Encapsulate Most Internal APIs
  - 261: Module System
  - 282: jlink: Java Linker
- JSR 376 Java Platform Module System

Project Jigsaw

- Released September 21st 2017!

Java Platform, Standard Edition

Java SE 9

Java SE 9 is the latest update to the Java Platform. This release includes much awaited new features like the modularization of the Java Platform, better performance, support for new standards, and many other improvements.

Learn more

- Installation Instructions
- Release Notes
- Oracle License
- Java SE Licensing Information User Manual
- Third Party Licenses
- Certified System Configurations
- Readme

JDK DOWNLOAD

Server JRE DOWNLOAD

JRE DOWNLOAD

J2D

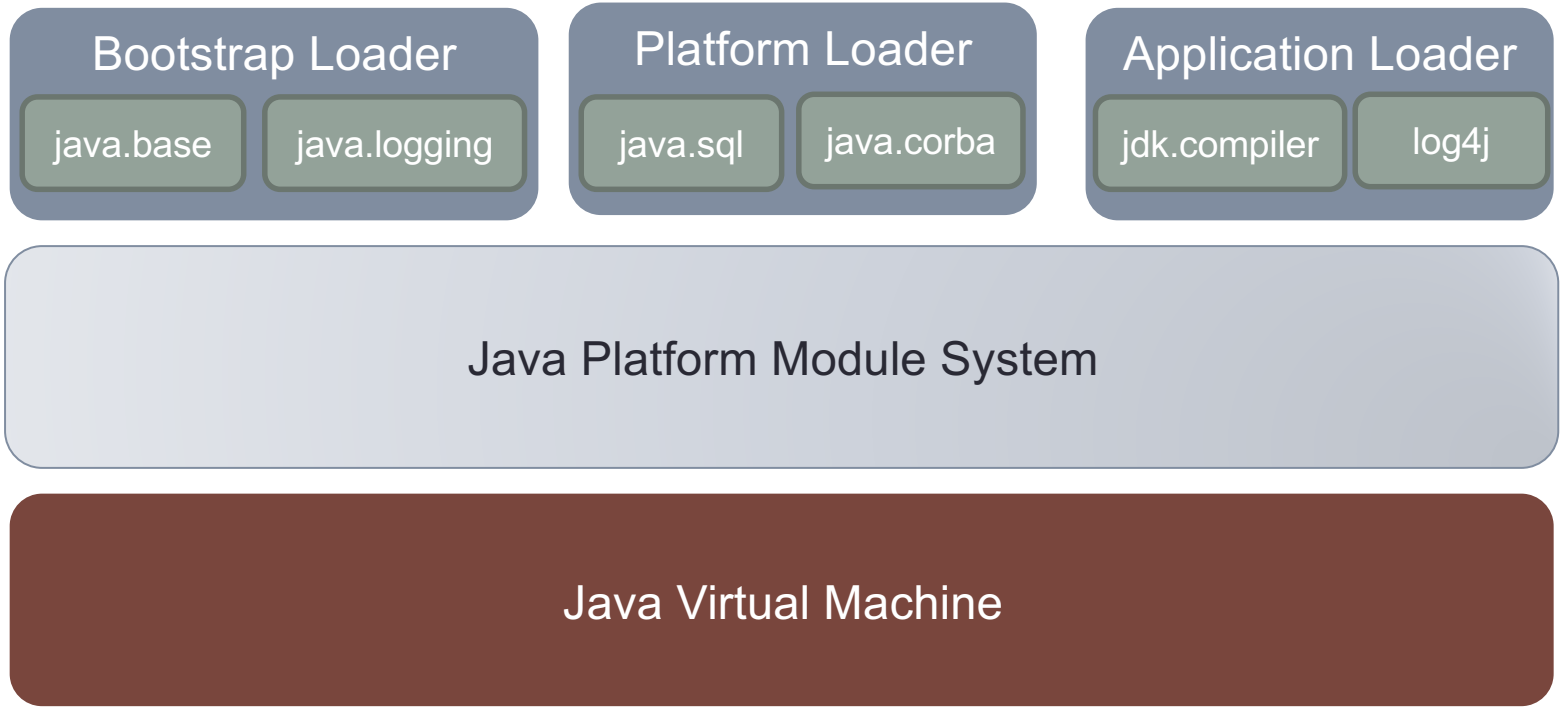
Comparing Jigsaw / OSGi / Java EE

Feature	Jigsaw	OSGi	Java EE
Allows cycles between packages in different modules	❌	✅	✅
Isolated package namespaces	❌	✅	✅
Allows lazy loading	❌	✅	✅
Allows dynamic package addition	❌	✅	✅
Unrestricted naming	❌	❌	✅
Allows multiple versions of an artifact*	❌	✅	✅
Allows split packages	❌	✅	✅
Allows textual descriptor	❌	✅	✅

Source: https://goo.gl/7RKqQx

J2D

Class Loading & Module System



J2D

Common Questions

- Do I have to modularize my application to run on Java 9?

- NO -

- Will my Java 6/7/8 application compile on Java 9?

- Depends – using private APIs? -

- Will my Java 6/7/8 application run un-modified on Java 9?

- Depends – using private APIs? -

- How do I identify problems today?

- jdeps -

Common Questions…

- Can I partially leverage modules?

- YES -

- Is tooling ready? (Maven/Gradle/IntelliJ/NetBeans/etc.)

- Mostly -

- Do application containers work on 9 today?

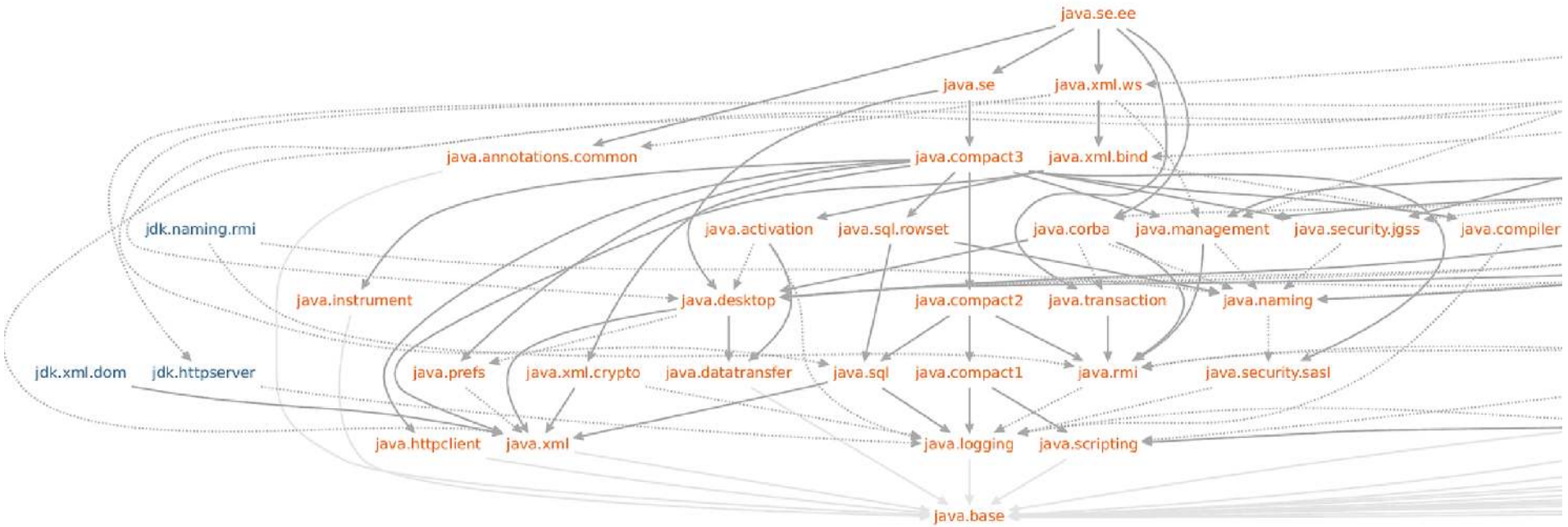
- Maybe -

- Is Java 9 being installed on client machines?

- YES -

J2D

JEP 200: Modular JDK



JDK is fully modularized!

J2D

jdeps

- Command line tool included with Java 8
- Static class dependency checker
- Key parameters
  - -jdkinternals – flags internal API usage that will break in Java 9
  - -dotoutput <dir> - dot output files
  - -cp – classpath to analyze
  - -verbose:class/package – package/class level dependencies
- Use jdeps on all generated output and dependencies

J2D

jdeps – Example

jdeps -jdkinternals jide-common.jar

```
jide-common.jar -> /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre/lib/rt.jar
   com.jidesoft.plaf.aqua.AquaJidePopupMenuUI (jide-common.jar)
      -> com.apple.laf.AquaPopupMenuUI                    JDK internal API (rt.jar)
   com.jidesoft.plaf.aqua.AquaRangeSliderUI (jide-common.jar)
      -> apple.laf.JRSUIConstants                         JDK internal API (rt.jar)
      -> apple.laf.JRSUIConstants$Orientation             JDK internal API (rt.jar)
      -> apple.laf.JRSUIConstants$State                   JDK internal API (rt.jar)
      -> com.apple.laf.AquaSliderUI                       JDK internal API (rt.jar)
   com.jidesoft.plaf.basic.BasicFolderChooserUI (jide-common.jar)
      -> sun.awt.shell.ShellFolder                        JDK internal API (rt.jar)
   com.jidesoft.plaf.basic.BasicPainter (jide-common.jar)
      -> sun.swing.plaf.synth.SynthIcon                   JDK internal API (rt.jar)
   com.jidesoft.plaf.metal.MetalUtils$GradientPainter (jide-common.jar)
      -> sun.swing.CachedPainter                          JDK internal API (rt.jar)
   com.jidesoft.plaf.windows.AnimationController (jide-common.jar)
      -> sun.awt.AppContext                               JDK internal API (rt.jar)
      -> sun.security.action.GetBooleanAction             JDK internal API (rt.jar)
…
```

JIDE: Widely used Swing component library (jidesoft.com)

jdeps – Fixing/Alternatives

Unsupported API (not for use)	Supported APIs (please use instead)	Note
core-libs		
sun.io	java.nio.charsets @since 1.4	
sun.misc.BASE64Decoder, sun.misc.BASE64Encoder, com.sun.org.apache.xml.internal.security.utils.Base64	java.util.Base64 @since 8	See http://openjdk.java.net/jeps/135
sun.misc.ClassLoaderUtil	java.net.URLClassLoader.close() @since 7	
sun.misc.Cleaner	java.lang.ref.PhantomReference @since 1.2	JDK-6417205 may help with the resource issues that can timely manner. Libraries accessing sun.misc.Cleaner hav sun.misc.Cleaner class; instead jdk.internal.misc.Cleaner. See JDK-6685587 and JDK-4724038
sun.misc.Service	java.util.ServiceLoader @since 1.6	
sun.misc.Timer	java.util.Timer @since 1.3	
sun.misc.Unsafe	sun.misc.Unsafe consists of a number of use cases. The following features are identified to provide support in the future releases: • JEP 193 Enhanced Volatile • JEP 187 Serialization 2.0 • Value types • JEP 189 Shenandoah:Low-Pause GC	In progress for JDK 9: • JEP 193 Enhanced Volatile • JDK-8044082 Efficient array comparison intrinsics • JDK-8033148 Lexicographic comparators for arrays

https://goo.gl/fxzKwP

J2D

jdeps: Build Integration

Maven JDeps Plugin: https://goo.gl/thr88W

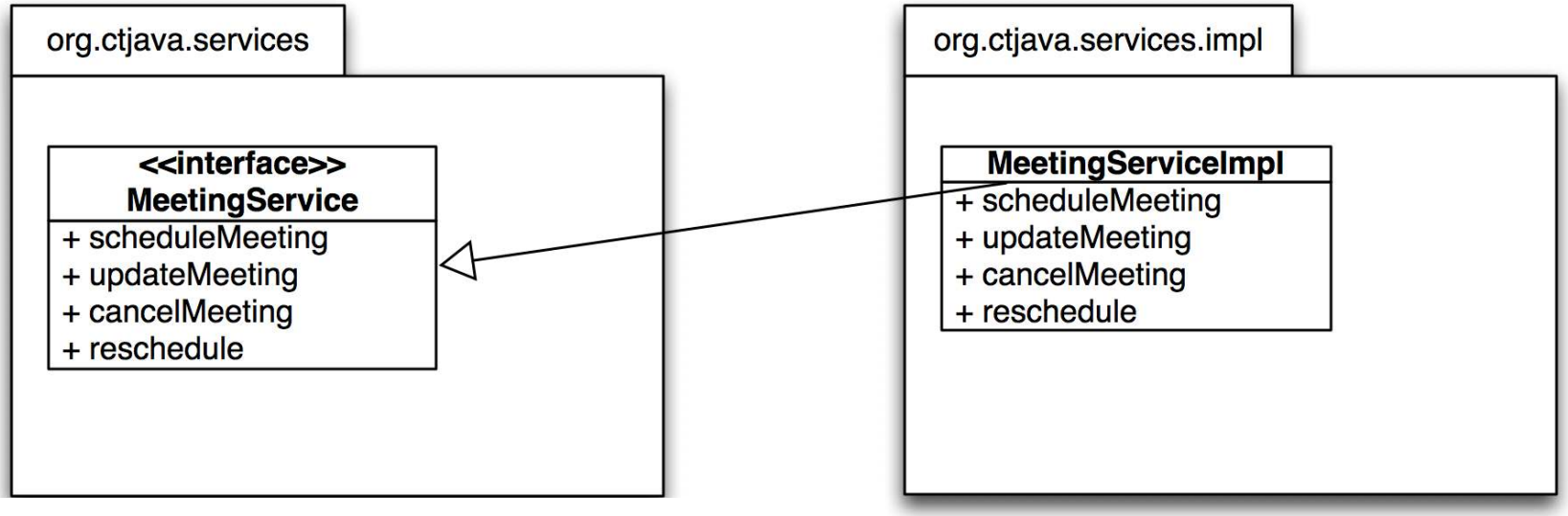
```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jdeps-plugin</artifactId>
    <version>3.0.0</version>
    <executions>
        <execution>
            <goals>
                <goal>jdkinternals</goal>
                <goal>test-jdkinternals</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Goals:

- jdeps:jdkinternals
- jdeps:test-jdkinternals

MODULE BASICS

Java Visibility Today




```
public interface MeetingService {
```

```
public class MeetingServiceImpl {
```

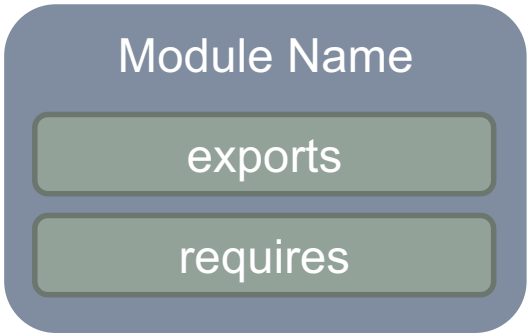
How do you hide the implementation class?

Java Accessibility

JDK 1-8	JDK 9+
public	public
protected	public to specific modules
<package>	public only within a module
private	protected
	<package>
	private

Module Definition

- *module-info.java* defines a module
- Contents define the following:
  - Unique name of the module
  - Dependencies of the module
  - Packages to be exported for use by other modules
- Placed at root of the namespace




```
module org.ctjava.services {
    exports org.ctjava.services;
}
```

J2D

Module Definition

```
<open> module <module-name> {
        [export <java package> [to <module name>]
        [requires [transitive] <module-name>]
        [opens <module name> [to <module name]]
        [provides <interface> with <implementation>]
        [uses <interface>]
}
```

Module Basics

- Module names must be unique
  - Names should follow reverse domain name pattern
- Modules are eagerly loaded
  - Dependency graph is built and checked on startup
  - Application won't start if modules are missing
- No support for versioning
  - Only one version of a module may be loaded
  - Layers maybe used to load different versions of a module

About Versioning…

- Two modules with the same package names in them are considered to be different versions of the same module.
- Two modules with the same name are considered to be two versions of the same module.
- Concealed package conflicts:
  - When two modules have the same package name in them, but the package is private in both modules, the module system cannot load both modules into the same layer

Introducing the Module Path

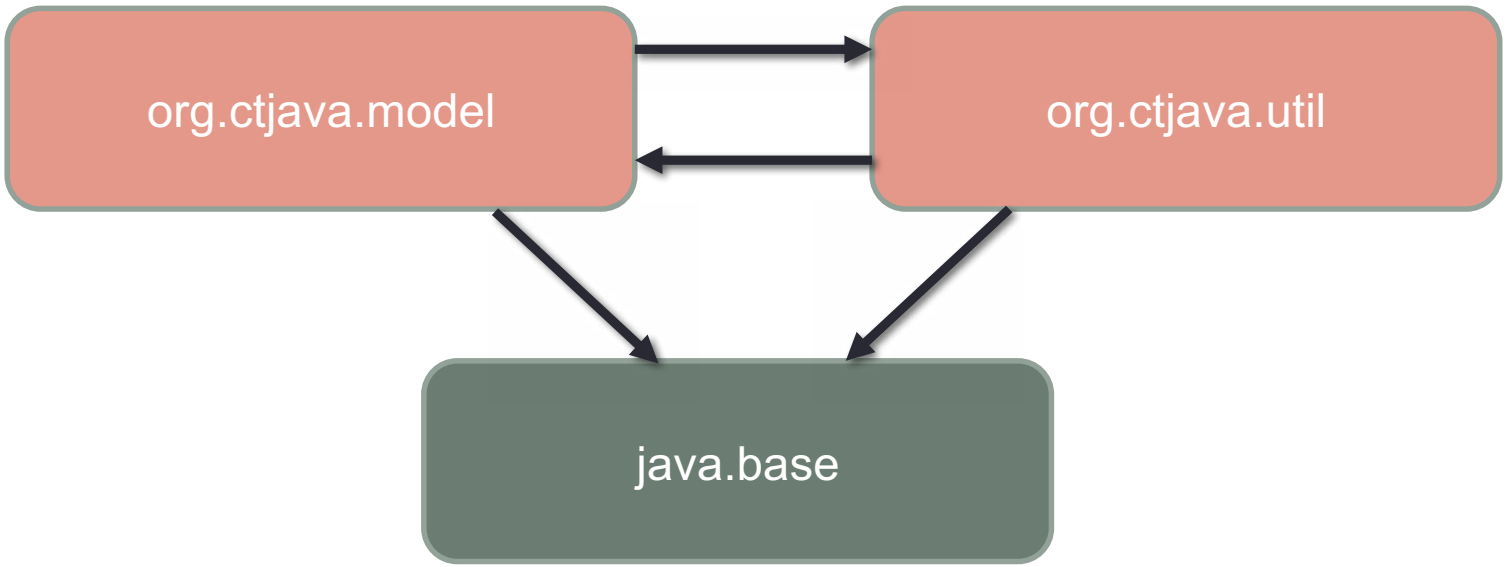
- Module path augments / extends the classpath
- All command line tools now include both
  - --module-path
  - --upgrade-module-path
  - --add-modules
  - --describe-module
  - --dry-run
  - --validate-modules

  Parameters on java command
- Use java –list-modules to see all available modules

JDK Modules

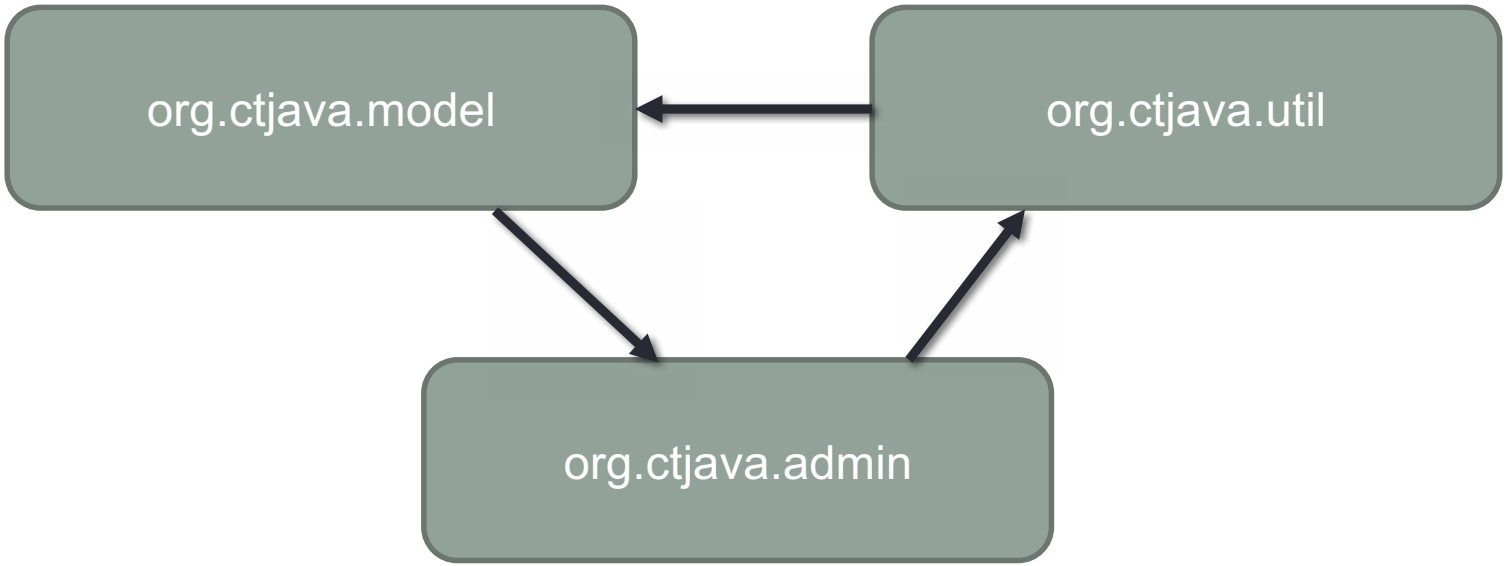
- *jdk.base* is included by default.
- *java.se* and *java.se.ee* are aggregator modules
- *java.se.ee* contains the following modules:
  - *java.corba*
  - *java.transaction*
  - *java.xml.ws.annotation*
  - *javax.xml.ws*
  - *java.xml.bind*
  - *java.activation*
- java runs with *java.se* modules – not *java.se.ee*

Dependency Cycles



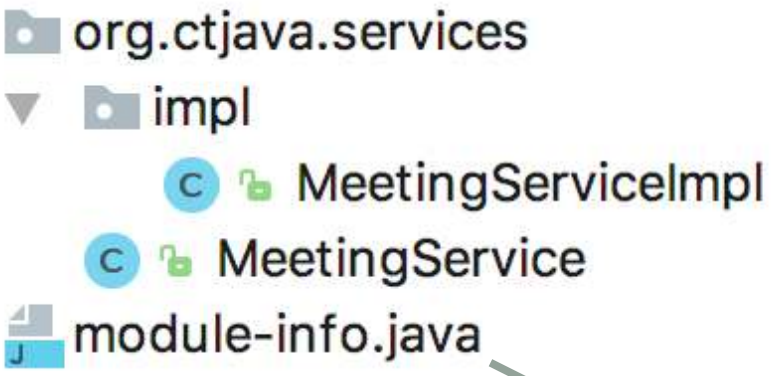
J2D

Dependency Cycles



J2D

Simple Module Example



Cannot use reflection to find MeetingServiceImpl

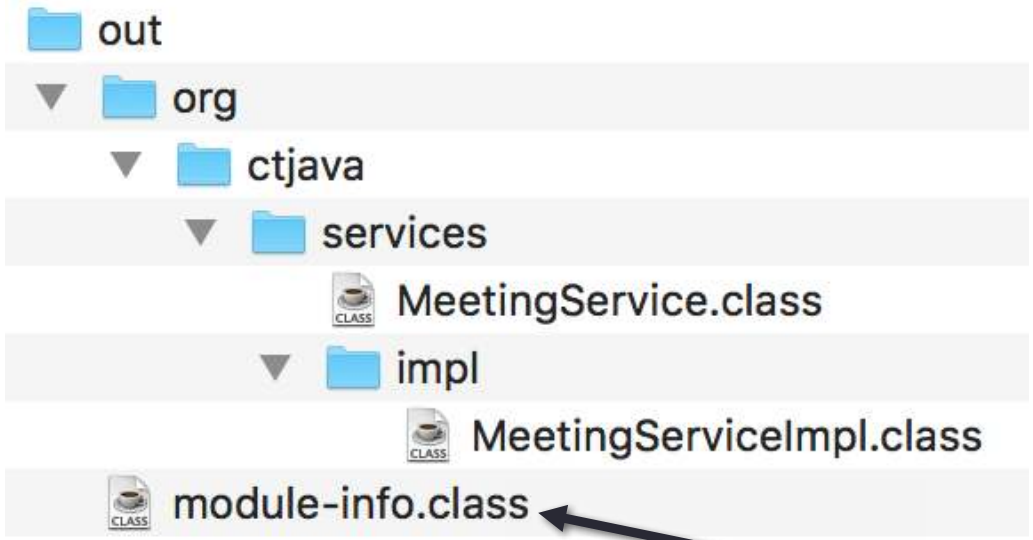
```
module org.ctjava.services {
    exports org.ctjava.services;
}
```

javac -d out src/org/ctjava/services/MeetingService.java src/org/ctjava/services/impl/MeetingServiceImpl.java src/module-info.java

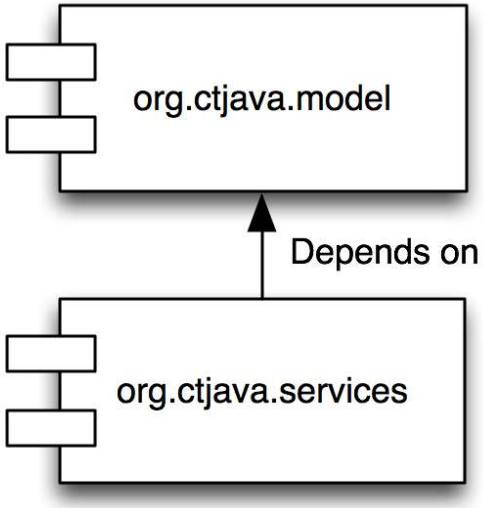
J2D

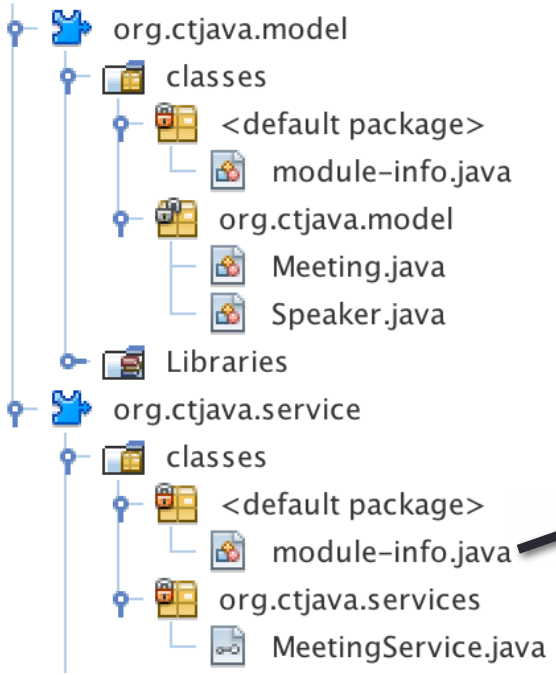
Simple Module Example…

Compiler Output:



Multiple Modules






```
module org.ctjava.model {
    exports org.ctjava.model;
}
```

```
module org.ctjava.service {
    exports org.ctjava.services;
    requires org.ctjava.model;
}
```

J2D

Compiling

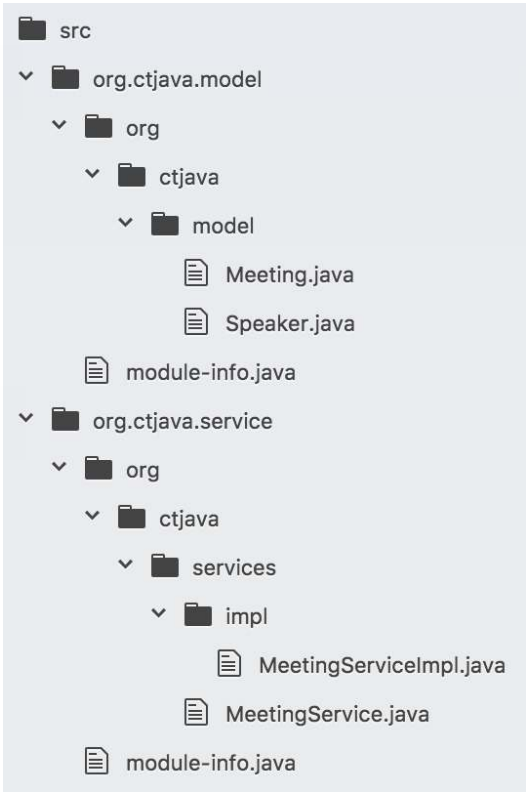
```
javac -d out --module-source-path src
  $(find . -name "*.java")
```

```
module org.ctjava.model {
    exports org.ctjava.model;
}
```

```
module org.ctjava.service {
    exports org.ctjava.services;
    requires org.ctjava.model;
}
```

Module →

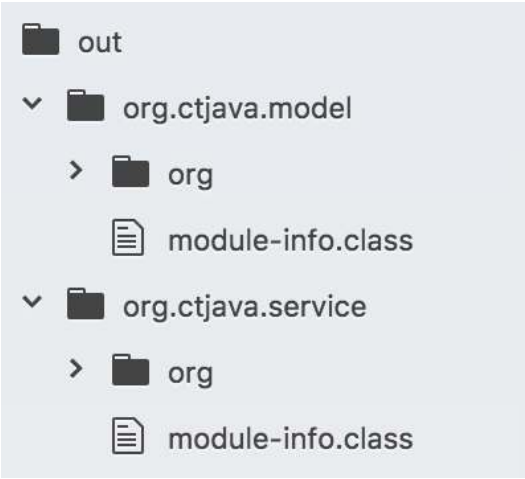
Module →



Directory containing module uses module name.
Names must match or compiler error.

J2D

Packaging



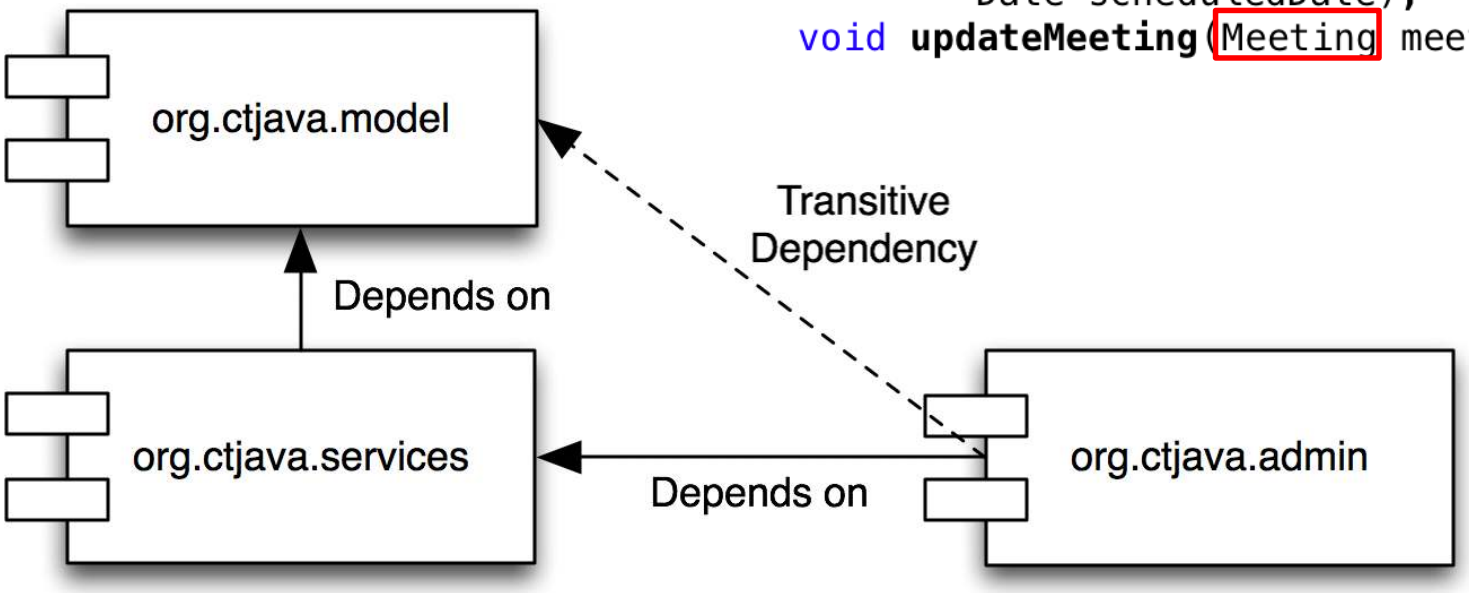

```
jar --create --file=org.ctjava.model@1.0.jar
  --module-version=1.0 -C out/org.ctjava.model .

jar --create --file=org.ctjava.service@1.0.jar
  --module-version=1.0 -C out/org.ctjava.service .
```

Note: Module version isn't used for module resolution. It is recorded in module-info.class.

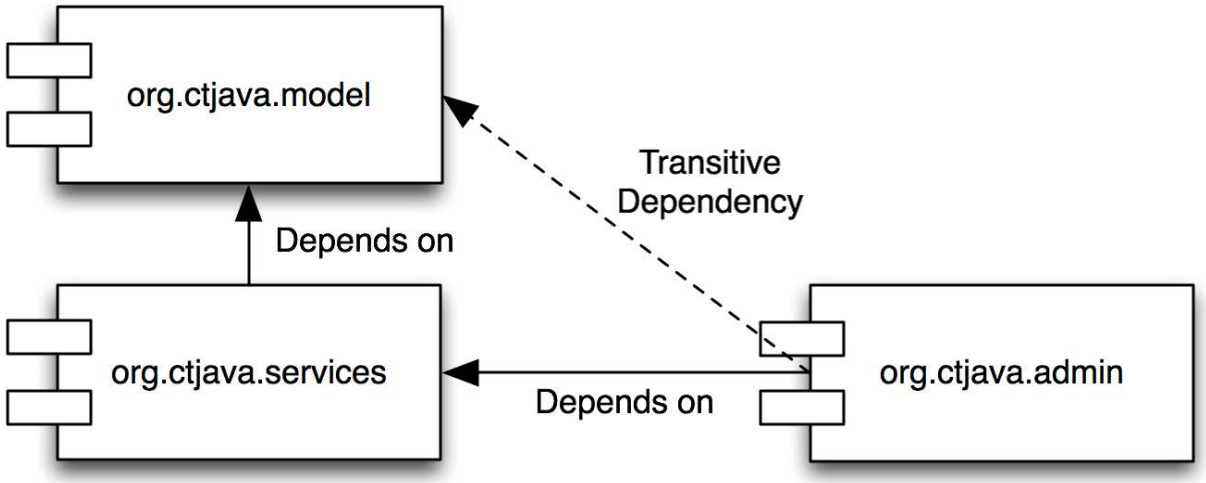
Transitive Dependencies

```
public interface MeetingService {
    void scheduleMeeting(Meeting meeting,
            Date scheduledDate);
    void updateMeeting(Meeting meeting);
```



J2D

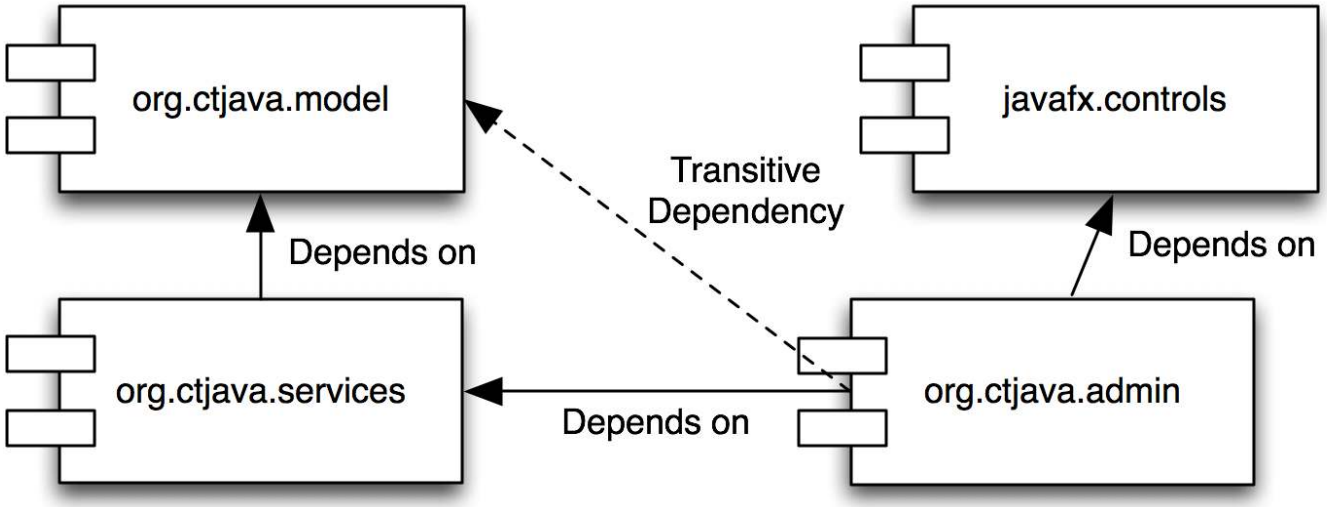
Transitive Dependencies…




```
module org.ctjava.services {
    requires transitive org.ctjava.model;
    exports org.ctjava.services;
}
```

Qualified Exports

Admin utility uses a JavaFX UI




```
module org.ctjava.admin {
    requires javafx.controls;
    requires org.ctjava.services;
}
```

J2D

Qualified Exports…

```
Exception in thread "main" java.lang.reflect.InvocationTargetException
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:564)
        at java.base/sun.launcher.LauncherHelper$FXHelper.main(LauncherHelper.java:946)
Caused by: java.lang.RuntimeException: Unable to construct Application instance: class org.ctjava.admin.AdminConsole
        at javafx.graphics/com.sun.javafx.application.LauncherImpl.launchApplication1(LauncherImpl.java:963)
        at javafx.graphics/com.sun.javafx.application.LauncherImpl.lambda$launchApplication$2(LauncherImpl.java:198)
        at java.base/java.lang.Thread.run(Thread.java:844)
Caused by: java.lang.IllegalAccessException:
        class com.sun.javafx.application.LauncherImpl (in module javafx.graphics) cannot access class
        org.ctjava.admin.AdminConsole (in module org.ctjava.admin) because module org.ctjava.admin does not export
        org.ctjava.admin to module javafx.graphics
```

- JavaFX LauncherImpl cannot access AdminConsole
- Class in javafx.graphics tried to instantiate AdminConsole in org.ctjava.admin

Qualified Exports

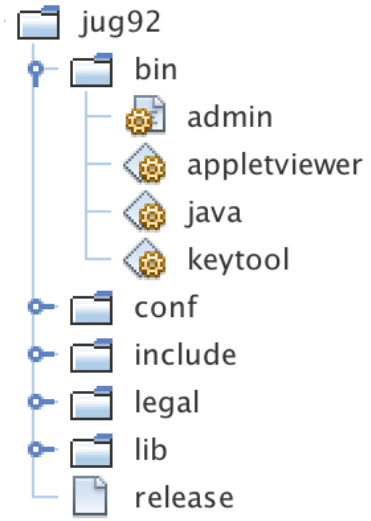
```
module org.ctjava.admin {
    exports org.ctjava.admin to javafx.graphics;
    requires javafx.controls;
    requires org.ctjava.services;
}
```

J2D

jlink

- Generates a runtime-image including the JVM.
- Only required modules are included
- Basic invocation:

```
jlink --module-path <modulepath>
      --add-modules <modules>
      --limit-modules <modules>
      --output <path>
```



Demo used
JavaFX: 95 mb
Full JDK: 454 mb

J2D

DIGGING DEEPER

JDKIO
0xCAFEBABE

Deprecated Modules

Code compiled using Java 8:

```
@WebService
@SOAPBinding(style = Style.RPC)
public interface AdminService {
    @WebMethod String exportMeetingAnalytics();
}
```

```
@WebService(endpointInterface = "org.ctjava.admin.AdminS
public class AdminServiceImpl implements AdminService {
    @Override
    public String exportMeetingAnalytics() {
        return "analytics";
    }
}
```

```
// publish the service
Endpoint.publish("http://localhost:9999/ws/analytics", new AdminServiceImpl());
```

Deprecated Modules…

java -jar WebEndpointTest.jar

```
Exception in thread "main" java.lang.NoClassDefFoundError: javax/xml/ws/Endpoint
        at org.ctjava.web.WebservicePublisher.main(WebservicePublisher.java:13)
Caused by: java.lang.ClassNotFoundException: javax.xml.ws.Endpoint
        at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:553)
        at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:185)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:486)
        ... 1 more
```

Worked on Java 8 – what happened?

Fix:

java --add-modules java.xml.ws -jar WebEndpointTest.jar

J2D

Module Types

Automatic Modules	• Mechanism for using JARs which do not include a module-info.java configuration • Export all of the JAR packages • Reads all other modules • Module name is dynamically generated
Unamed Module	• Alls JARs and classes on the classpath will be contained un the Unamed Module. • Similar to Automatic Modules • Does not possess a name
Platform Modules	• JDK modules

JARs & Module System

	--module-path	-classpath
Modular JAR	Application Module	Unnamed Module
Non-modular JAR	Automatic Module	Unnamed Module

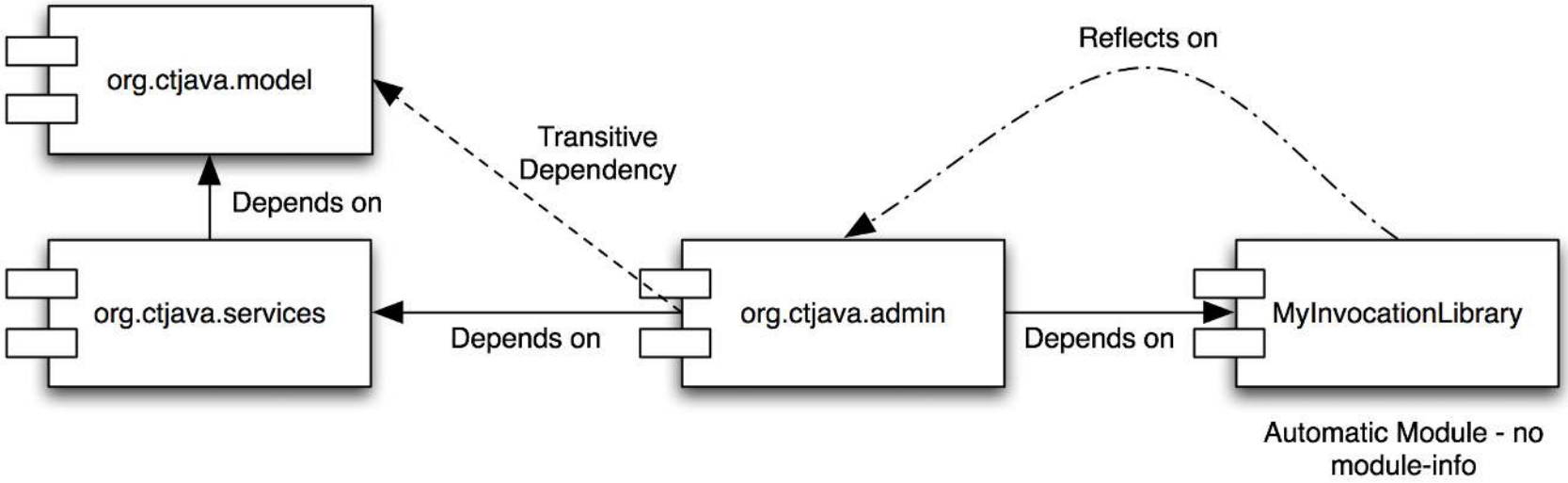
Unnamed module name: ALL-UNNAMED

J2D

Readability

Module Type	Origin	Export Packages	Read Modules
Named Platform	Provided by platform	Explicitly	
Named Application	All JARS containing module-info on the module path	Explicitly	• Platform • Application • Automatic
Automatic	All JARs without module-info but on module path	All	• Platform • Application • Automatic • Unnamed
Unnamed	Classpath	All	• Platform • Automatic • Application

Automatic Modules



Automatic Modules…

Code in MyInvocationLibrary:

```
public class InstantiateUtility {

    public Object create(String className) {
        try {
            Class clazz = Class.forName(className);
            return clazz.newInstance();
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException ex) {
            Logger.getLogger(InstantiateUtility.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }

}
```

Code in org.ctjava.admin – module:

```
InstantiateUtility utility = new InstantiateUtility();
utility.create("org.ctjava.admin.AdminCallback");
```

Automatic Modules…

```
module org.ctjava.admin {
    exports org.ctjava.admin to javafx.graphics;
    requires javafx.controls;
    requires org.ctjava.services;
    requires MyInvocationLibrary;
}
}
```

java.lang.IllegalAccessException: class org.ctjava.util.InstantiateUtility (in module MyInvocationLibrary) cannot access class org.ctjava.admin.AdminCallback (in module org.ctjava.admin) because module org.ctjava.admin does not export org.ctjava.admin to module MyInvocationLibrary

Escape & Encapsulation Kill Switch

Important javac command line options:

--add-opens – opens a package

--permit-illegal-access - All code in the unnamed module can access other types regardless of any limitations.

Will result in a WARNING: to be removed in Java 10.

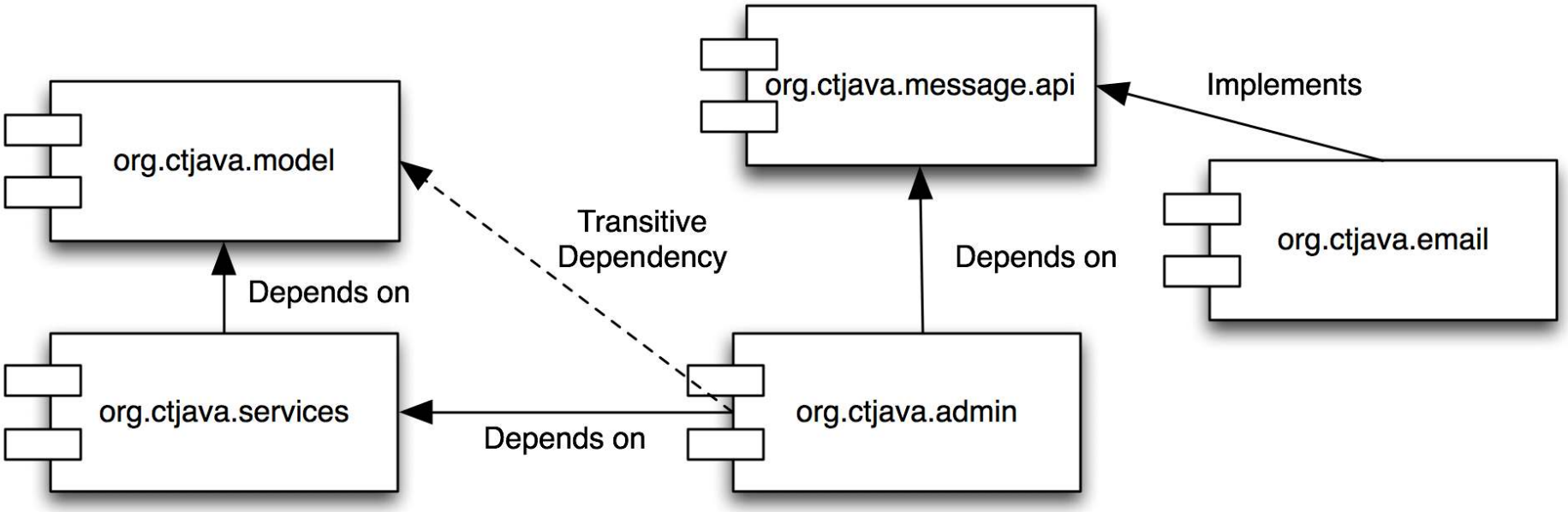
J2D

SERVICES

Services

- API first added in Java 6
- Rarely used outside of JDK
- Enhanced/altered for Jigsaw

Services Example



Defining Service API

```
package org.ctjava.message.api;

public interface MessageService {
    void sendMessage(Member memeber, String message);
}
```

```
module org.ctjava.message.api {
    requires org.ctjava.model;
    exports org.ctjava.message.api;
}
```

J2D

Service Implementation

```
module org.ctjava.email {
    requires org.ctjava.model;
    requires org.ctjava.message.api;
    provides org.ctjava.message.api.MessageService
            with org.ctjava.email.EmailMessageService;
}
```

```
package org.ctjava.email;

import org.ctjava.message.api.MessageService;
import org.ctjava.model.Member;

public class EmailMessageService implements MessageService {
    @Override
    public void sendMessage(Member memeber, String message) {
        // send message
    }
}
```

Service in Action

```
module org.ctjava.admin {
    exports org.ctjava.admin to javafx.graphics;
    requires javafx.controls;
    requires org.ctjava.services;
    requires org.ctjava.message.api;
    uses org.ctjava.message.api.MessageService;
}
```

```
Iterable<MessageService> mservice = ServiceLoader.load(MessageService.class);
for(MessageService ms : mservice) {
    System.out.println("Message service: " + ms);
}
```

J2D

JAVA 9 TOOLING

June 2017 Status

Tooling Support

- Use jenv to swich between Java 8 & 9

  http://www.jenv.io/

  Examples:

  jenv local 9 (locally switch to Java 9)

  jevn global 1.8 (globally use Java 8)

Note: rt.jar and tools.jar were removed.

J2D

Tooling Support

Tool	Status
Apache NetBeans 9	🔶 Developer Preview
IntelliJ 2017.1	✅
Eclipse	🔶
Maven	🔶
Gradle	🔶

- Maven & Gradle do not generate module-info.java.
- Only most recent releases of Maven / Gradle work on 9
- Eclipse Neon won't start with Java 9 edit ini file.

Maven

Maven status: https://goo.gl/PxuqZF

groupId	artifactId	affected goal	Plugin Name	Minimum Compatible Version	Related Issues
org.apache.maven.plugins	maven-compiler-plugin	compile, testCompile	Maven Compiler Plugin	3.6.1	required to compile module-info files requires plexus-compiler 2.8.1
org.apache.maven.plugins	maven-javadoc-plugin	jar, javadoc, aggregate	Maven Javadoc Plugin	2.10.4	MJAVADOC-441, MJAVADOC-442, MJAVADOC-449
org.apache.maven.plugins	maven-plugin-plugin	descriptor	Maven Plugin Plugin	3.5	MPLUGIN-300, MPLUGIN-304
org.apache.maven,plugins	maven-war-plugin	war	Maven War Plugin	3.0.0-SNAPSHOT (not yet released)	MWAR-405
org.apache.maven.plugins	maven-ejb-plugin	ejb	Maven EJB Plugin	3.0.0-SNAPSHOT (not yet released)	? Need to be created
org.apache.maven.plugins	maven-ear-plugin	ear	Maven EAR Plugin	3.0.0-SNAPSHOT (not yet released)	
org.codehaus.plexus	plexus-component-metadata	generate-metadata	Plexus :: Component Metadata	1.7	#1

J2D

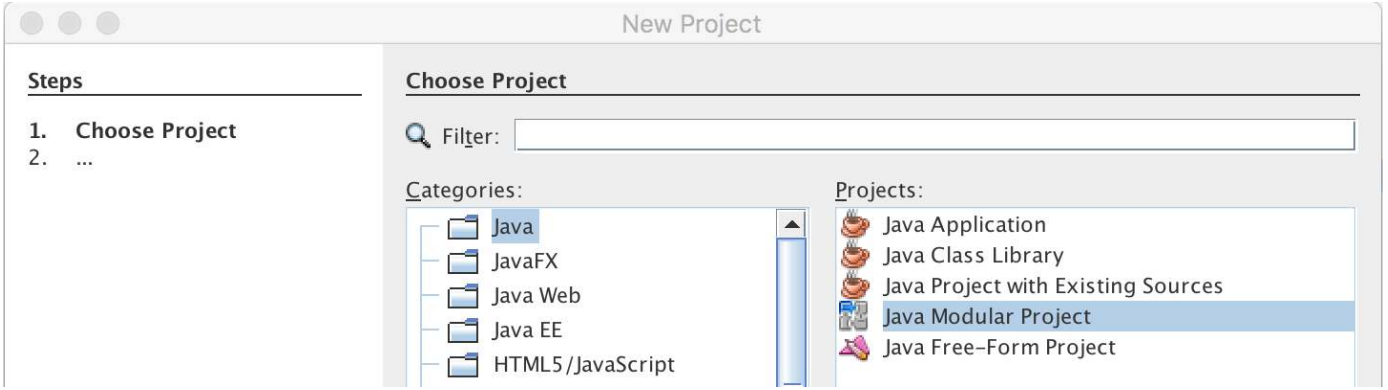
Maven

- Maven Shade Plugin and Jigsaw don’t mix (uber jar)
- Example project with Maven and Java 9/Jigsaw:
- https://goo.gl/rmzKBa

J2D

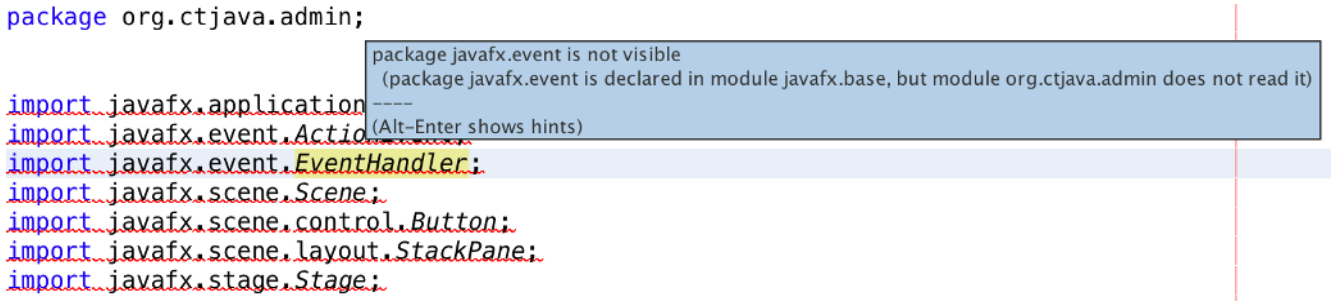
NetBeans & Java 9

- NetBeans Developer Preview required for Java 9.
- New project template for modular projects.
- Creates Ant build files



NetBeans & Java 9

- Module structure not compatible with javac:
  - <module>/classes/<source code>
- Detects imports where the module is not imported.
  - Does not auto-import yet.



IntelliJ

- Support added in 2017.1 (release March 2017)
- For multi-module project:
  1. Start with Empty Project
  2. Add new module for each Jigsaw module
- IntelliJ detects when an import requires a module definition

Demo

BEST PRACTICES

Java 9 Preparation

- Analyze dependencies using *jdeps*
  - Immediately fix code using internal APIs
  - Check transitive dependencies for internal JDK APIs
- Integrate jdeps into continuous integration
  - Use Maven JDeps Plugin
- Analyze reflection code: *setAccessible(true)*
- Refactor code to remove duplicate packages across JARs
  - common.jar: com.foo.utilities
  - Server.jar com.foo.utilitie

Legacy Code

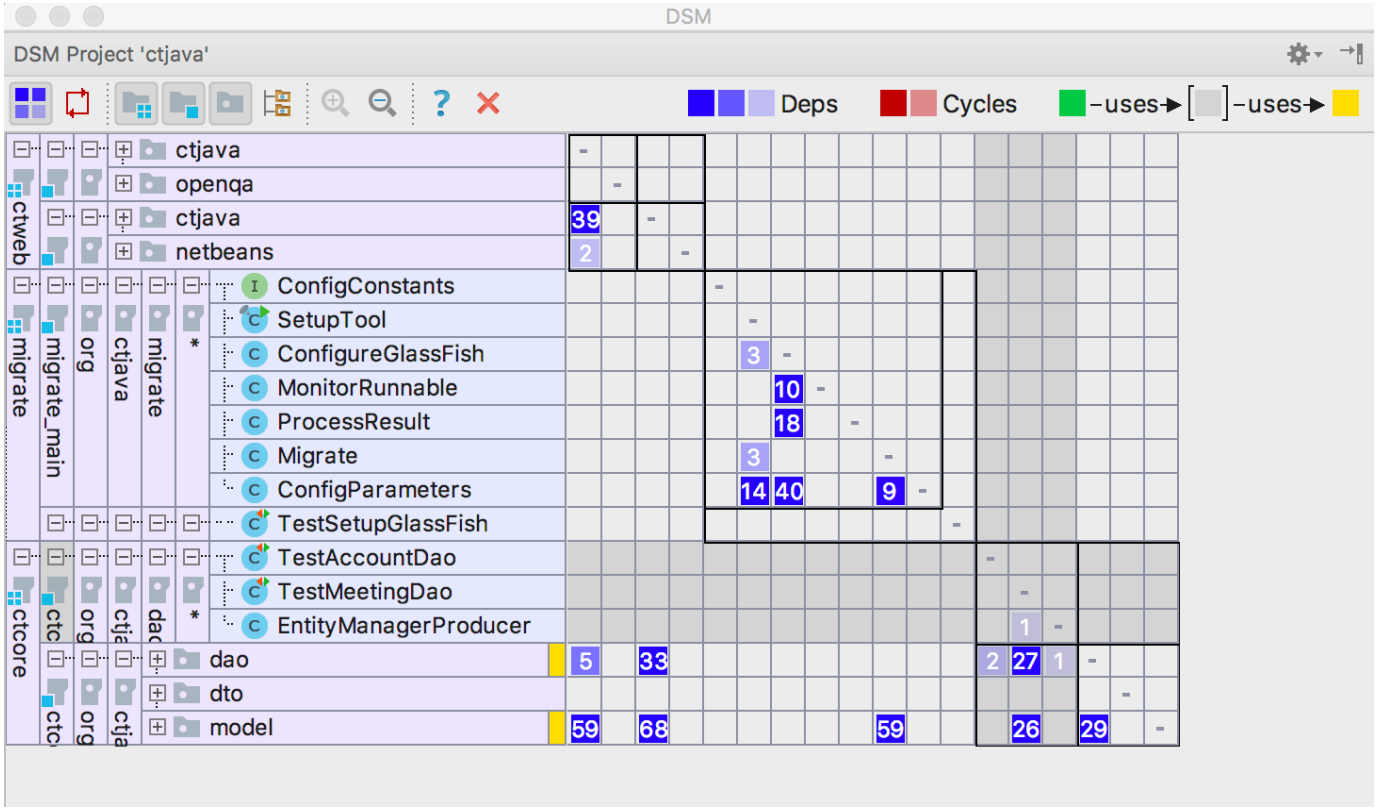
- If using internal JDK API, fix or use flag:

  --add-exports, --add-opens, --permit-illegal-access

- If using removed JDK module: (ex. Activation):

  --add-modules <module>

Analyze Project Structure



IntelliJ DSM Analyzer

https://goo.gl/FLzCL7

Remove cycles!

J2D

Java EE & Spring

- Support for Jigsaw unknown
- Major effort required for containers to support Jigsaw
- Concerns raised in JCP voting over Jigsaw and EE
- Regardless: Dependencies on JDK internal APIs must be resolved.

Desktop App Impact

- Impacts of JDK internal dependencies maybe felt more on Java desktop applications
- Many hacks were required to work around limitations/bugs in Swing
- Older applications may need to be ported to JavaFX

Technical Debt Bill is NOW DUE.

Best Practice

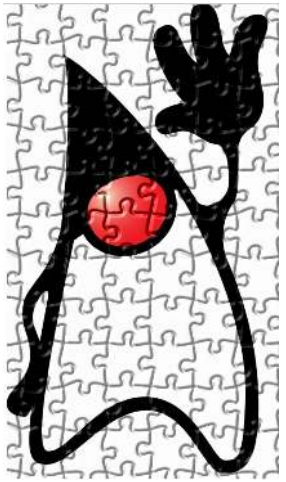
- Don’t depend upon java.se
- Avoid using qualified exports
- Don’t remove exports from a module in future releases
- Keep modules-info clean:
  - requires javafx.controls
  - ~~requires javafx.base~~; -- controls already includes base

Resources

- Books
  - The Java 9 Module System (https://goo.gl/QM1LS1)
  - Java 9 Modularity (https://goo.gl/zJeYsd)
- Links
  - Issues: https://goo.gl/xK4ymJ
  - Oracle Talks: http://openjdk.java.net/projects/jigsaw/talks/
  - Tutorial: https://goo.gl/ET9Hn1
  - Tutorial: https://goo.gl/GzPXTw

Focus on newer Jigsaw material (2016 or later)

Q&A



Twitter: @ctjava

Email: rcuprak@gmail.com / r5k@3ds.com

Blog: cuprak.info

Linkedin: www.linkedin.com/in/rcuprak

Slides: www.slideshare.net/rcuprak/presentations

J2D