Project Jigsaw in JDK 9: Modularity Comes To Java

Simon Ritter
Deputy CTO, Azul Systems

@speakjava | azul.com

Agenda

- API structure changes
- Introduction to Jigsaw
- Developing code with modules
- Application migration
- Resources

API Structure Changes

API Classification

- Supported, intended for public use
  - JCP specified: java.*, javax.*
  - JDK specific: some com.sun.*, some jdk.*
- Unsupported, not intended for public use
  - Mostly sun.*
  - Most infamous is sun.misc.Unsafe

General Java Compatability Policy

- If an application uses only supported APIs on version N of Java it *should* work on version N+1, even without recompilation
- Supported APIs can be removed, but only with advanced notice
- To date 23 classes, 18 interfaces and 379 methods have been deprecated
  - None have been removed

JDK 9: Incompatible Changes

- Encapsulate most JDK internal APIs
- Remove a small number of supported APIs
  - 6 in total, all add/remove PropertyChangeListener
  - Already flagged in JSR 337 (Java SE 8), JEP 162
- Change the binary structure of the JRE and JDK
- New version string format
- A single underscore will no longer be allowed as an identifier in source code

Removed In JDK 9

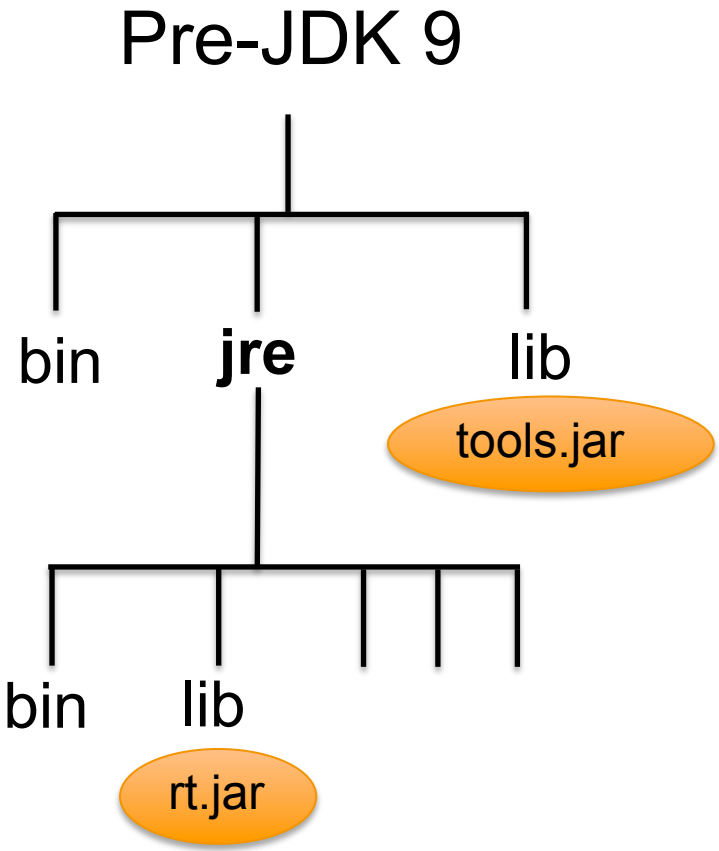
- Endorsed standard API override mechanism
- Extension mechanism

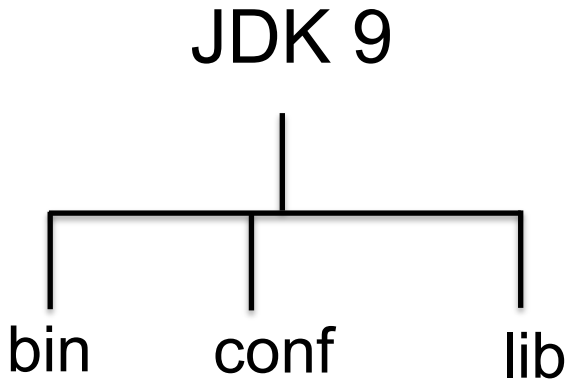
- No longer required now we have a module system

Binary Structure Of JDK/JRE

- Potentially disruptive change
  - Details in JEP 220
  - Blurs the distinction between JRE and JDK
- Implemented since late 2014
  - Allow people to get used to new organisation

JDK Structure





~~jre directory~~
~~tools.jar~~
~~rt.jar~~

Most Popular Unsupported APIs

1. sun.misc.BASE64Encoder
2. sun.misc.Unsafe
3. sun.misc.BASE64Decoder

Oracle dataset based on internal application code

JDK Internal API Classification

- Non-critical
  - Little or no use outside the JDK
  - Used only for convenience (alternatives exist)
- Critical
  - Functionality that would be difficult, if not impossible to implement outside the JDK

JEP 260 Proposal

- Encapsulate all non-critical JDK-internal APIs
- Encapsulate all critical JDK-internal APIs, for which supported replacements exist in JDK 8
- Do *not* encapsulate other critical JDK-internal APIs
  - Deprecate these in JDK 9
  - Plan to encapsulate or remove them in JDK 10
  - Provide command-line option to access encapsulated critical APIs

JEP 260 Accessible Critical APIs

- sun.misc.Unsafe
- sun.misc.Signal
- sun.misc.SignalHandler
- sun.misc.Cleaner
- sun.reflect.Reflection.getCallerClass
- sun.reflect.ReflectionFactory

Reviewing Your Own Code

- jdeps tool
  - Introduced in JDK 8, improved in JDK 9
  - Maven jdeps plugin

```
jdeps –jdkinternals path/myapp.jar
```

```
path/myapp.jar -> /opt/jdk1.8.0/jre/lib/rt.jar
   <unnamed> (myapp.jar)
      -> java.awt
      -> java.awt.event
      -> java.beans
      -> java.io
      ...
```

Introduction To Jigsaw And Modules

Goals For Project Jigsaw

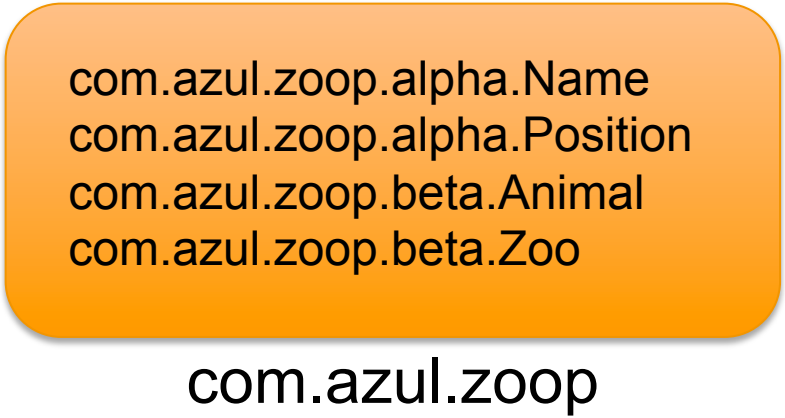
- Make Java SE more scalable and flexible
- Improve security, maintainability and performance
- Simplify construction, deployment and maintenance of large scale applications

Modularity Specifications

- Java Platform Module System
  - JSR 376: Targeted for JDK 9 (no promises)
- Java SE 9: New JSR will cover modularisation of APIs
- OpenJDK Project Jigsaw
  - Reference implementation for JSR 376
  - 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

Module Fundamentals

- Module is a grouping of code
  - For Java this is a collection of packages
- The module can contain other things
  - Native code
  - Resources
  - Configuration data



Module Declaration

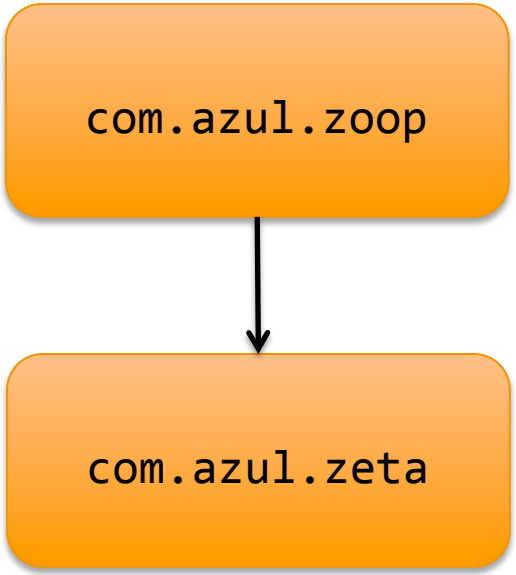
```
module com.azul.zoop {
}
```

module-info.java

```
com/azul/zoop/alpha/Name.java
com/azul/zoop/alpha/Position.java
com/azul/zoop/beta/Animal.java
com/azul/zoop/beta/Zoo.java
```

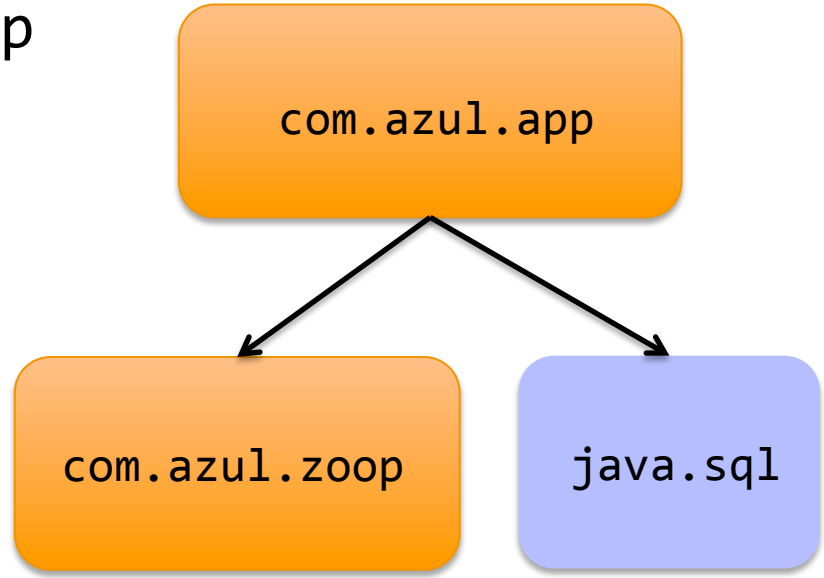
Module Dependencies

```
module com.azul.zoop {
  requires com.azul.zeta;
}
```

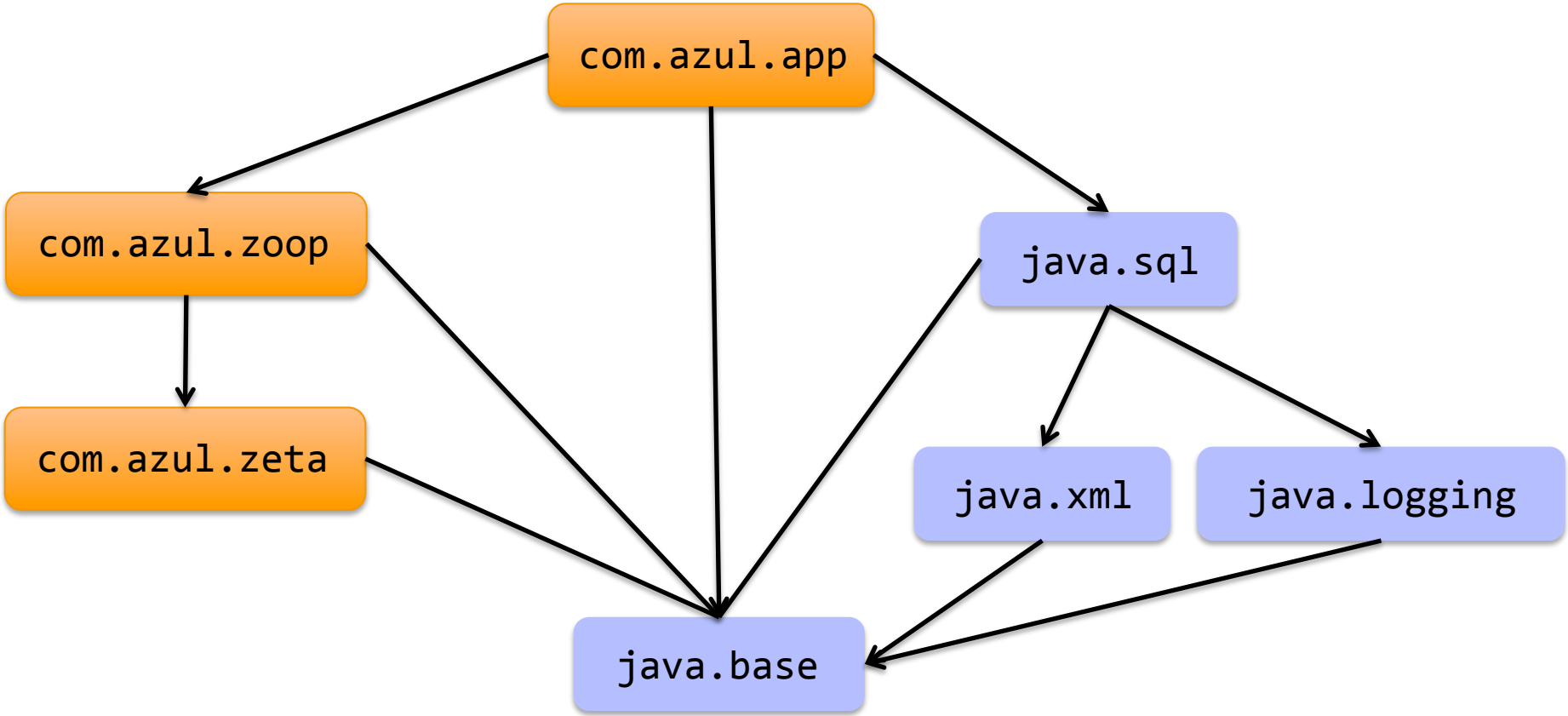


Module Dependencies

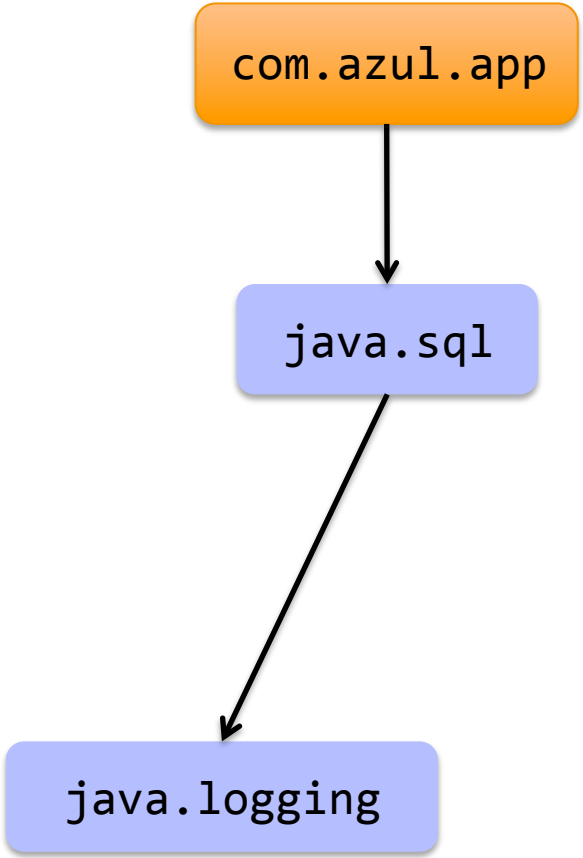
```
module com.azul.app {
  requires com.azul.zoop
  requires java.sql
}
```



Module Dependency Graph



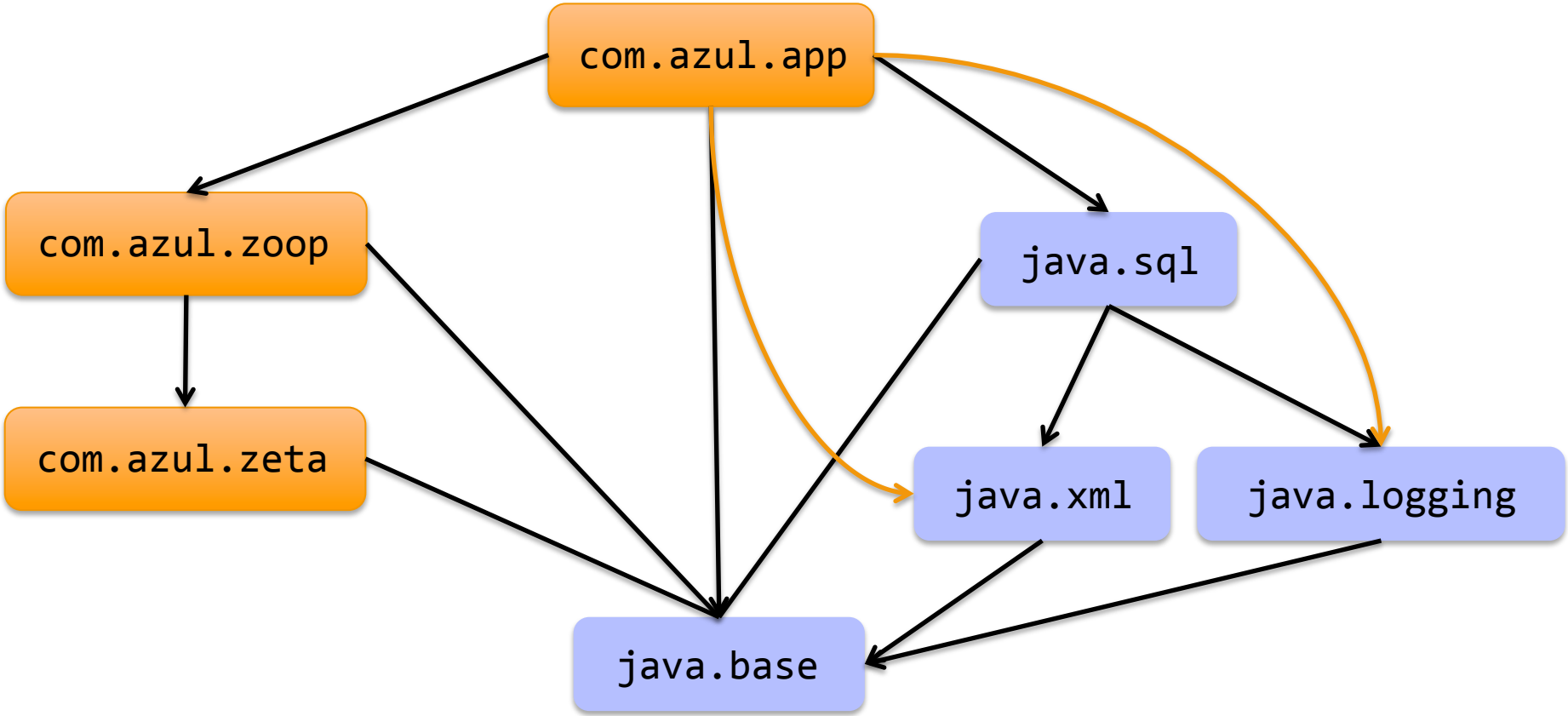
Readability v. Dependency




```
Driver d = …
Logger l = d.getParentLogger();
l.log(“azul’);
```

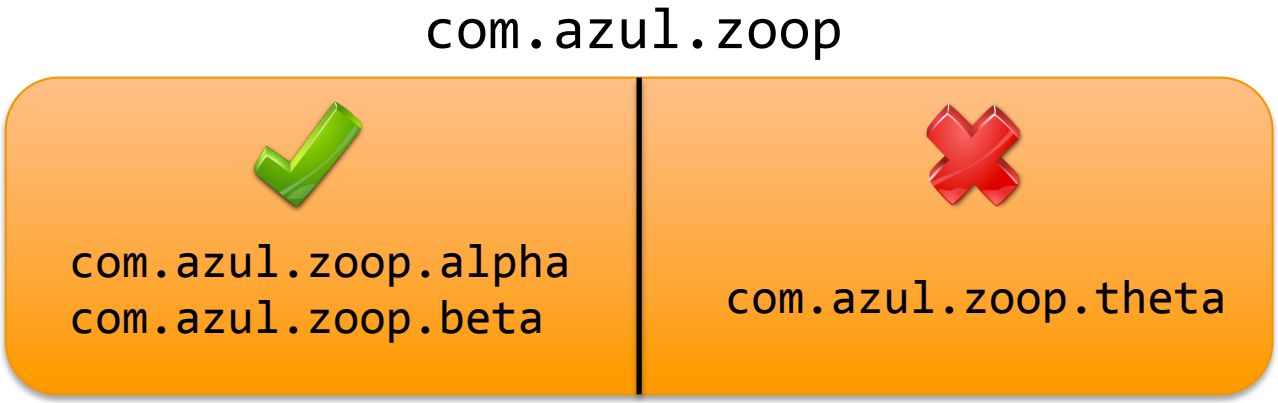
```
module java.sql {
  requires public java.logging;
}
```

Module Readability Graph



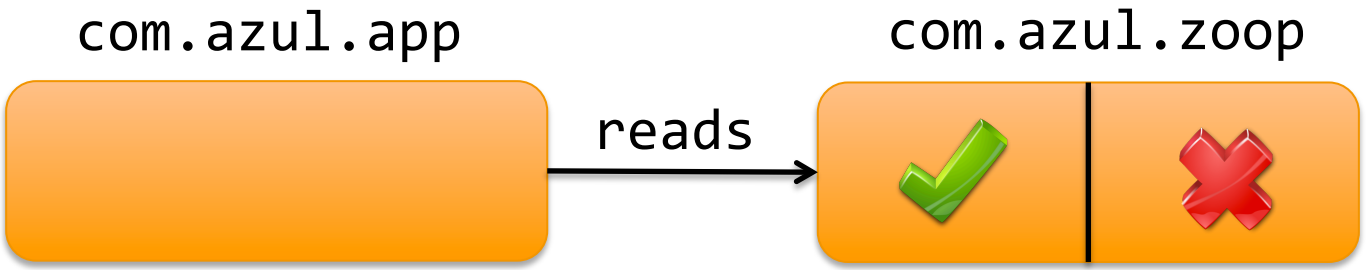
Package Visibility

```
module com.azul.zoop {
  exports com.azul.zoop.alpha;
  exports com.azul.zoop.beta;
}
```



Accessibility

- For a package to be visible
  - The package must be exported by the containing module
  - The containing module must be read by the using module
- Public types from those packages can then be used



Java Accessibility (pre-JDK 9)

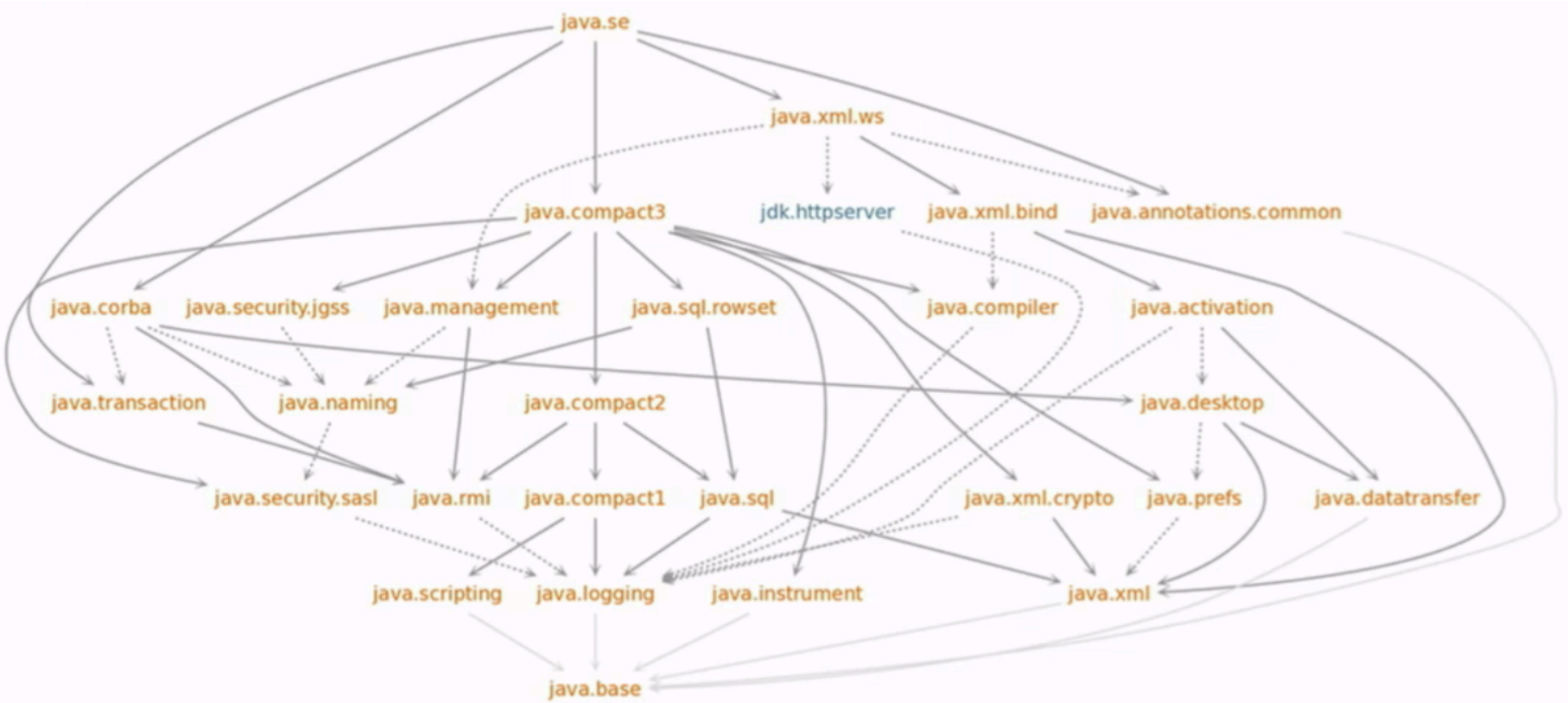
```
public
protected
<package>
private
```

Java Accessibility (JDK 9)

public to everyone
public, but only to specific modules
public only within a module
protected
<package>
private

public ≠ accessible (fundamental change to Java)

JDK Platform Modules

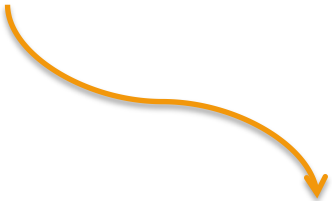


Developing Code With Modules

Compilation

```
$ javac -d mods \
  src/zeta/module-info.java \
  src/zeta/com/azul/zeta/Vehicle.java
```

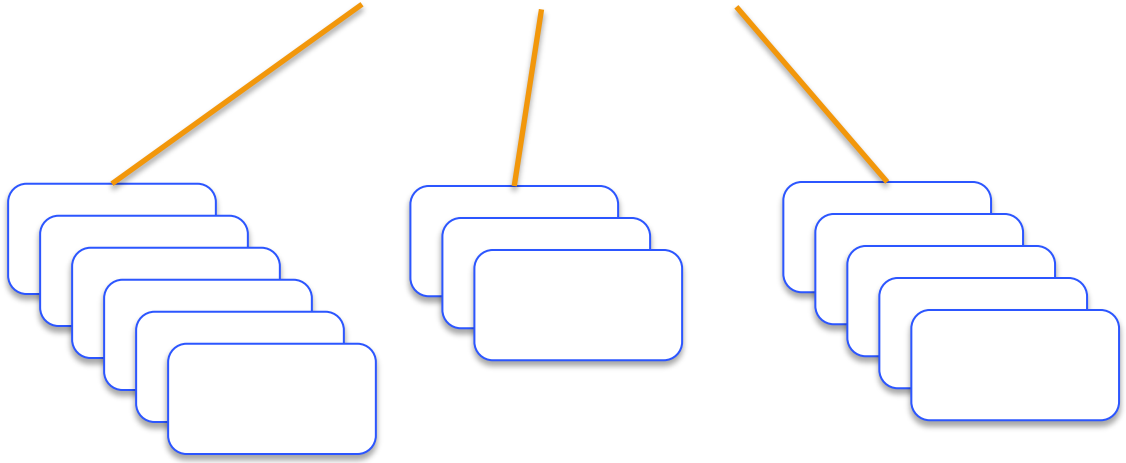
```
src/zeta/module-info.java
src/zeta/com/azul/zeta/Vehicle.java
```



```
mods/zeta/module-info.class
mods/zeta/com/azul/zeta/Vehicle.class
```

Module Path

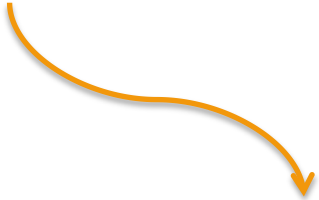
```
$ javac –modulepath dir1:dir2:dir3
```



Compilation With Module Path

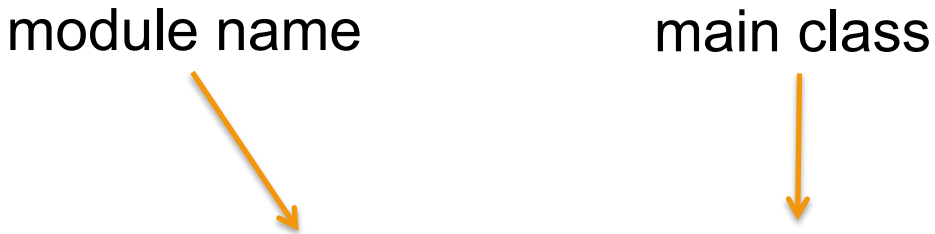
```
$ javac -modulepath mods -d mods \
  src/zoop/module-info.java \
  src/zoop/com/azul/zoop/alpha/Name.java
```

```
src/zoop/module-info.java
src/zoop/com/azul/zoop/alpha/Name.java
```



```
mods/zoop/module-info.class
mods/zoop/com/azul/zoop/alpha/Name.class
```

Application Execution



- -modulepath can be abbreviated to -mp

Module Diagnostics

```
$ java -Xdiag:resolver -mp mods -m com.azul.zoop/com.azul.zoop.Main
```

Packaging With Modular Jars

```
mods/zoop/module-info.class
mods/zoop/com/azul/app/Main.class
```

app.jar

```
module-info.class
com/azul/zoop/Main.class
```

```
$ jar --create --file myLib/app.jar \
    --main-class com.azul.zoop.Main \
    -C mods .
```

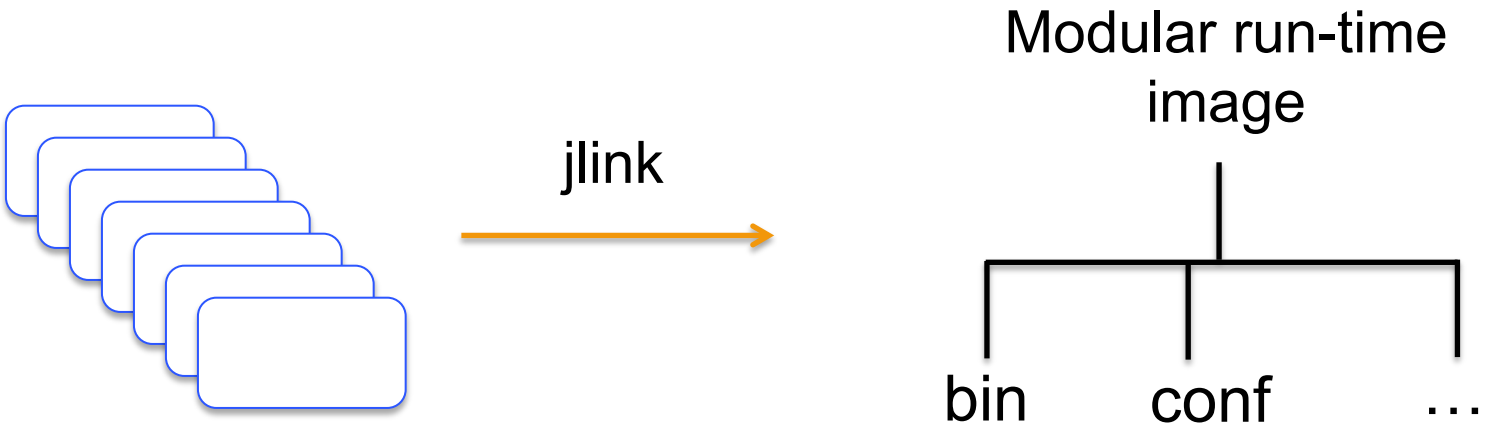
Jar Files & Module Information

```
$ jar --file myLib/app.jar -p
Name:
  com.azul.zoop
Requires:
  com.azul.zeta
  java.base [MANDATED]
  java.sql
Main class:
  com.azul.zoop.Main
```

Application Execution (JAR)

```
$ java -mp myLib:mods -m com.azul.zoop.Main

Azul application initialised!
```

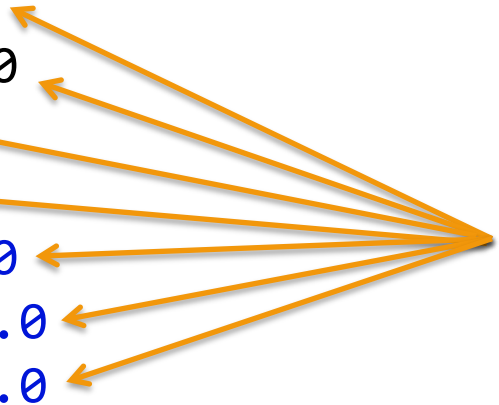
Linking




```
$ jlink --modulepath $JDKMODS \
  --addmods java.base –output myimage

$ myimage/bin/java –listmods
java.base@9.0
```

Linking An Application

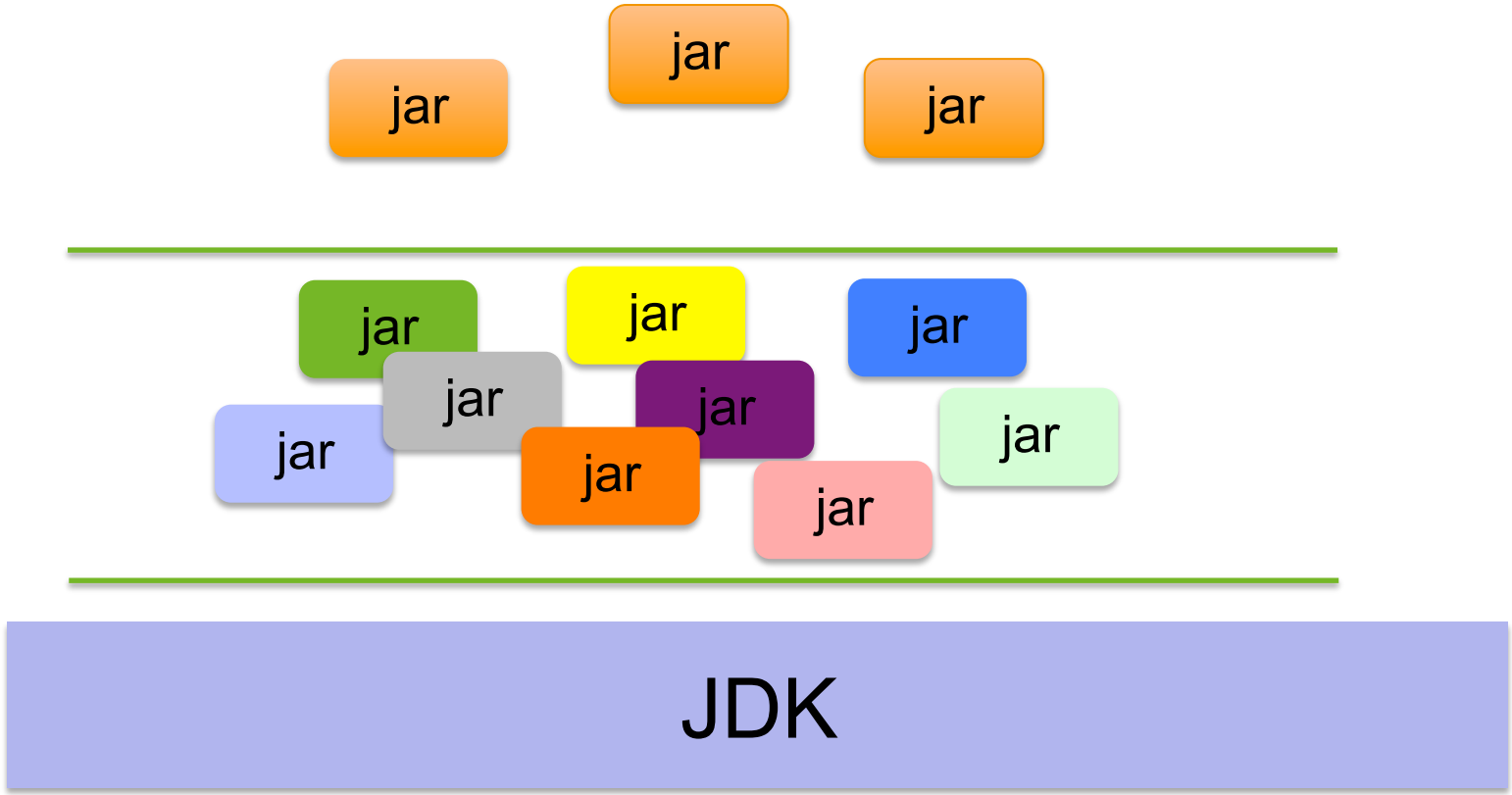
```
$ jlink --modulepath $JDKMODS:$MYMODS \
  --addmods com.azul.app –output myimage

$ myimage/bin/java –listmods
java.base@9.0
java.logging@9.0
java.sql@9.0
java.xml@9.0
com.azul.app@1.0
com.azul.zoop@1.0
com.azul.zeta@1.0
```



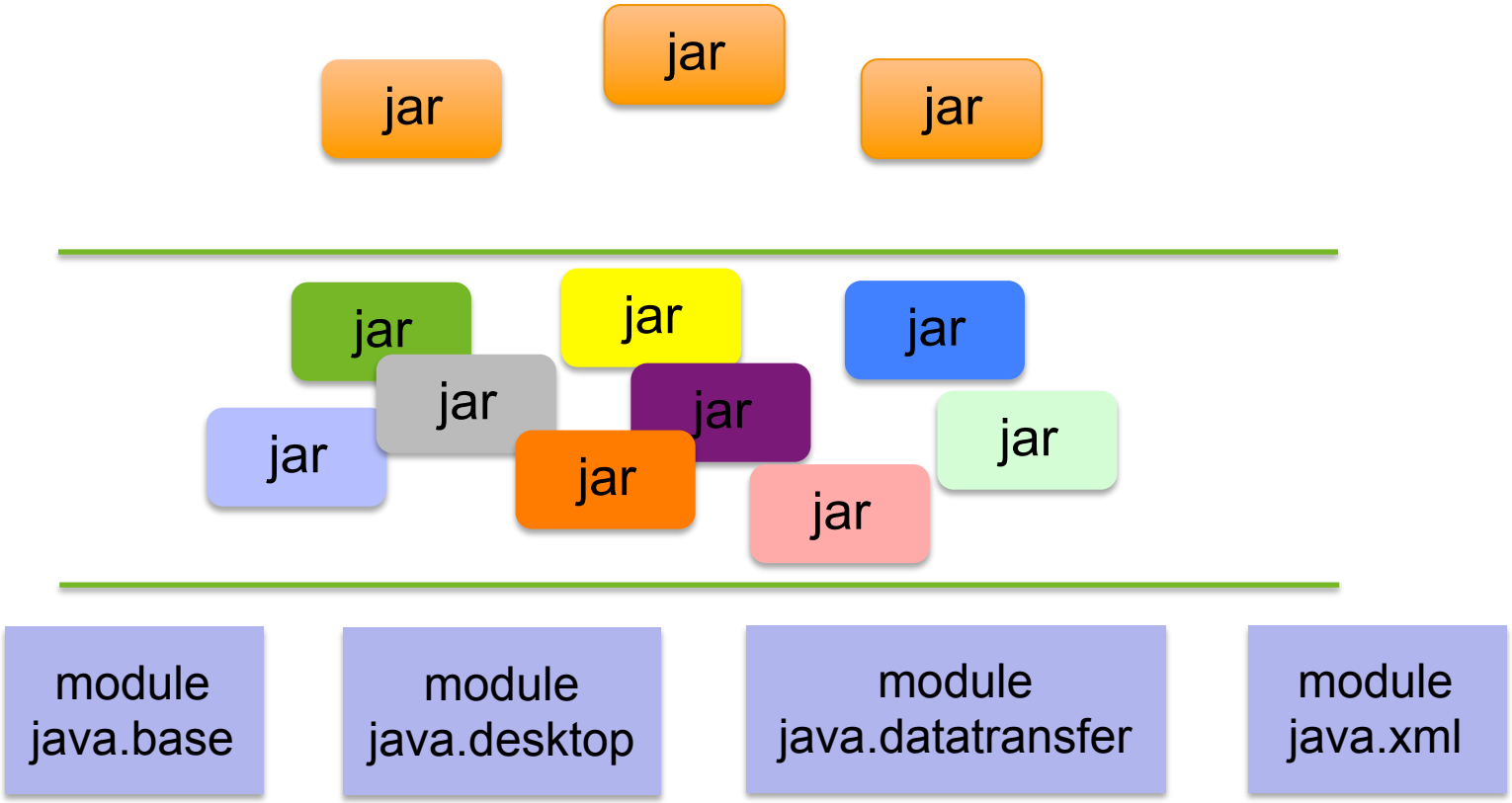
Version numbering for information purposes only

Application Migration

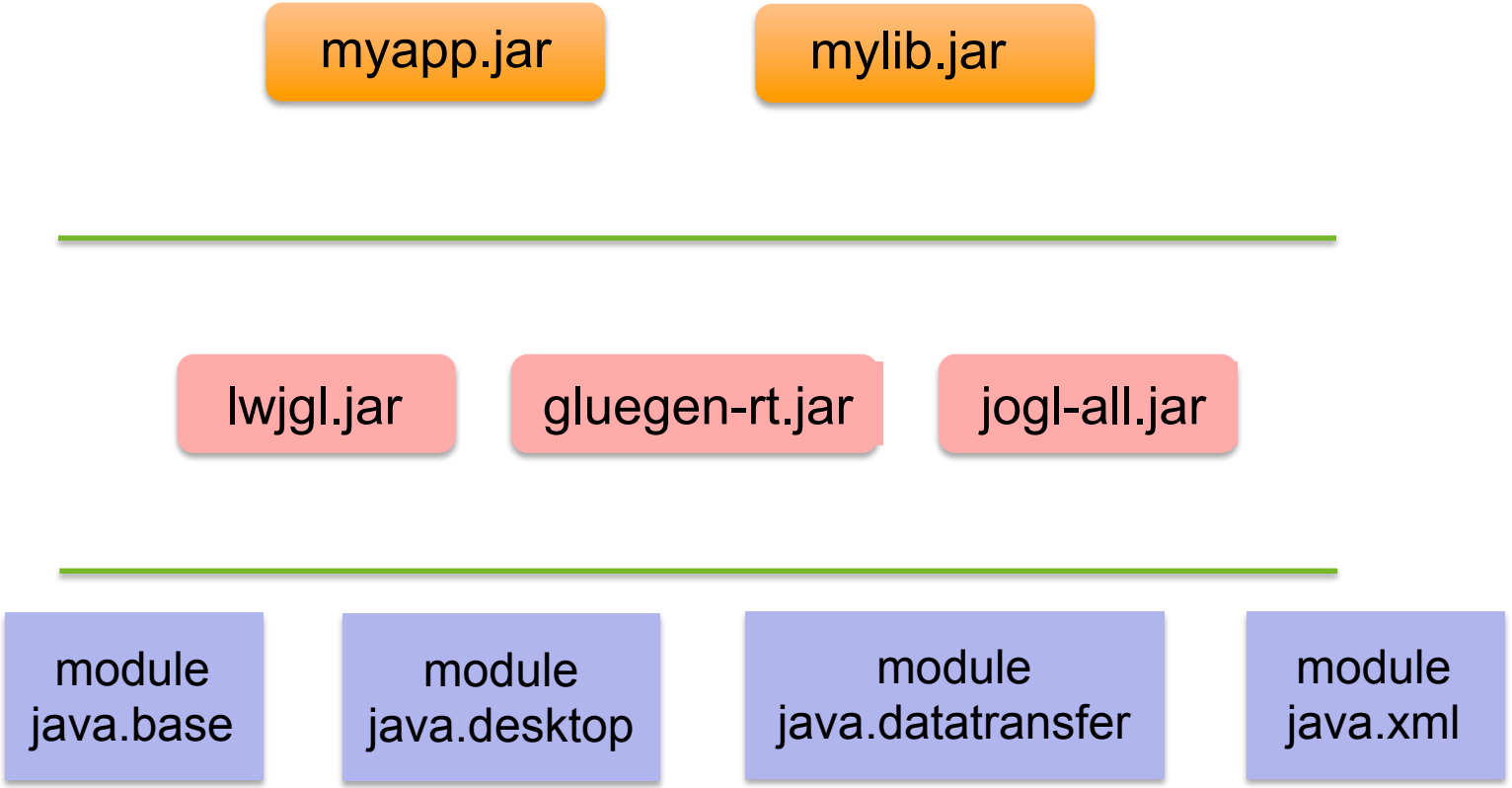
Typical Application (JDK 8)



Typical Application (JDK 9)



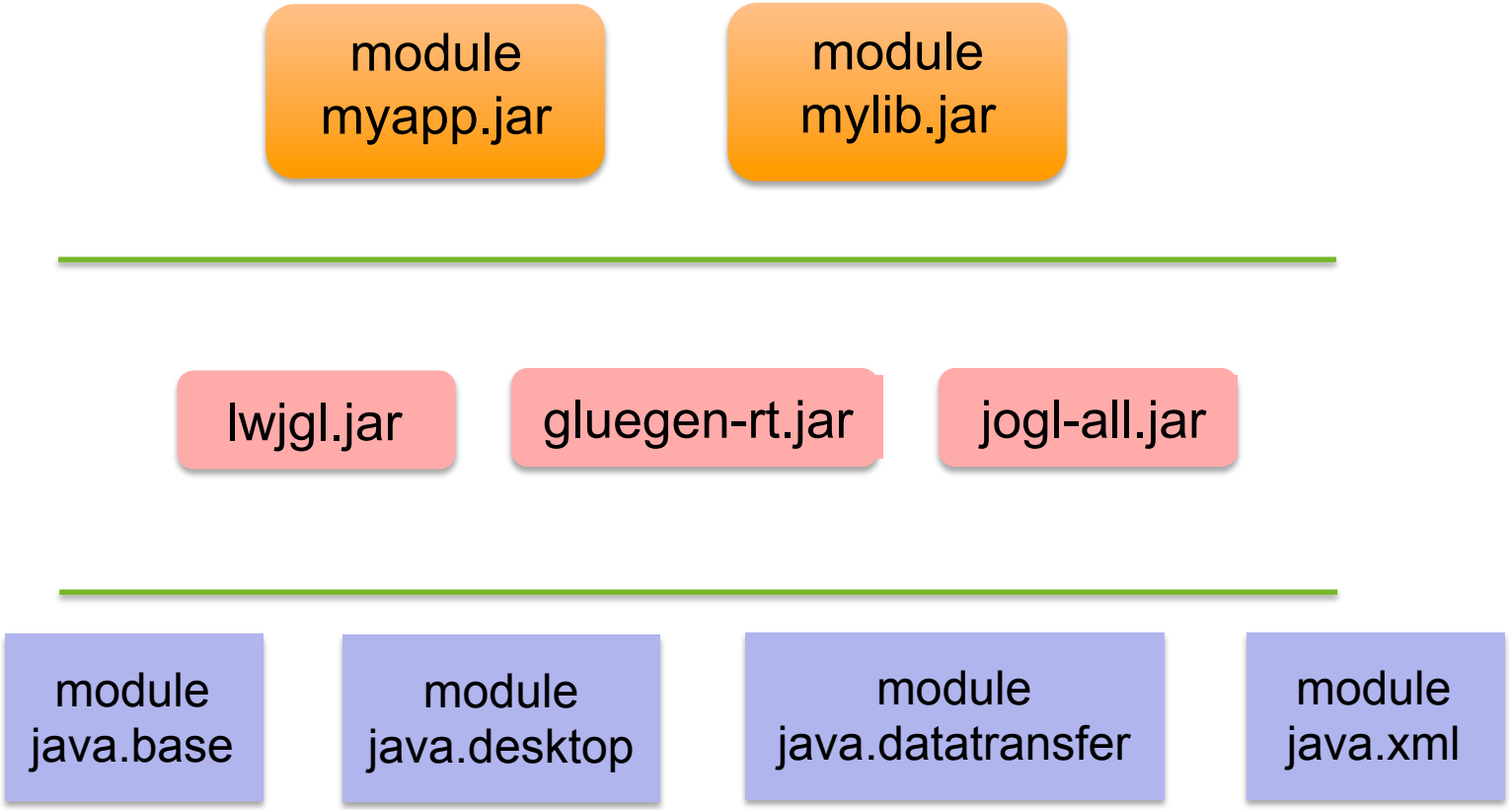
Sample Application



Run Application With Classpath

```
$ java –classpath \
  lib/myapp.jar: \
  lib/mylib.jar: \
  lib/liblwjgl.jar: \
  lib/gluegen-rt.jar: \
  lib/jogl-all.jar: \
  myapp.Main
```

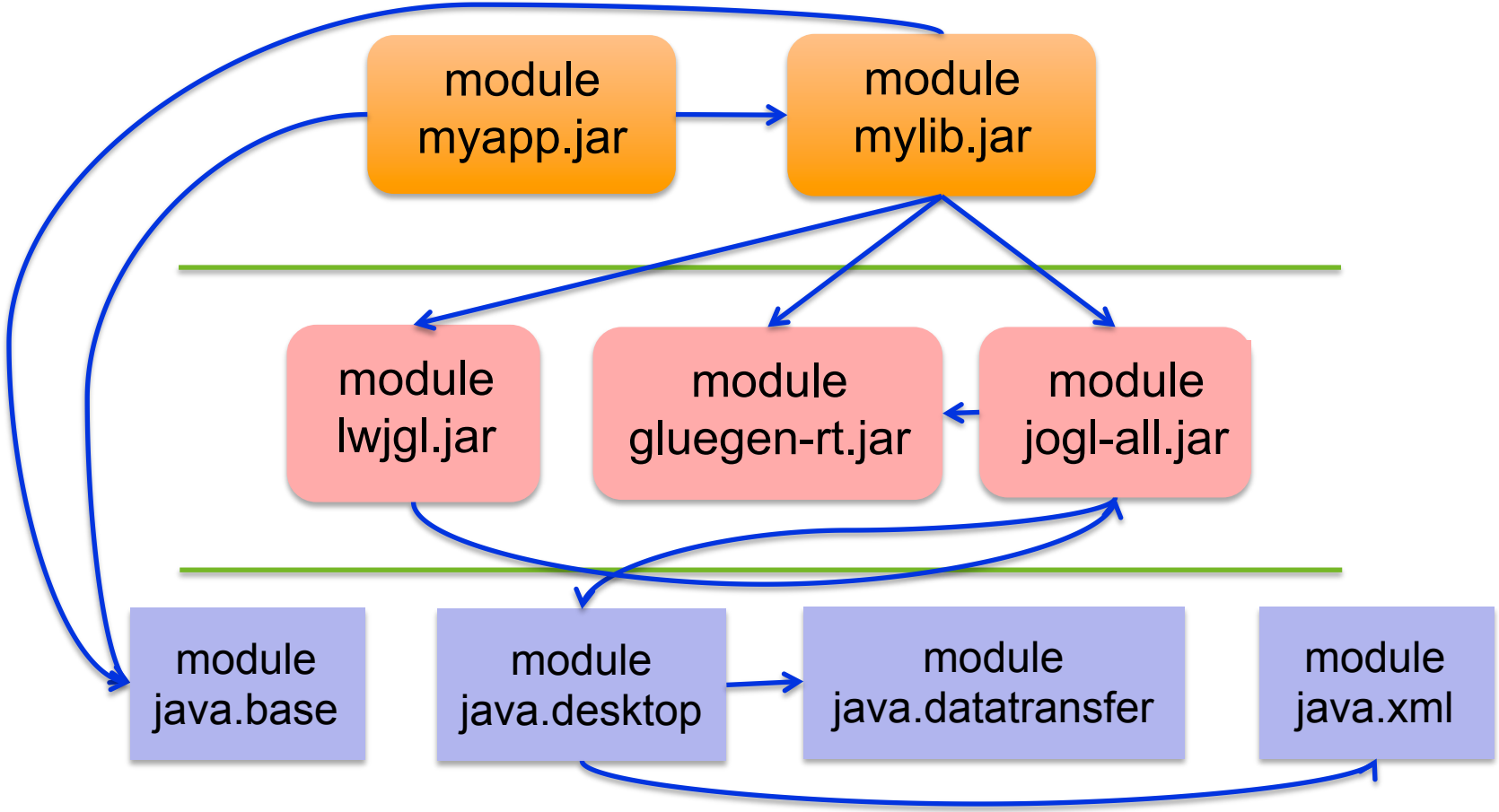
Sample Application



Application module-info.java

```
module myapp {
  requires mylib;
  requires java.base;
  requires java.sql;
  requires lwjgl;         ????
  requires gluegen-rt;    ????
  requires jogl-all;      ????
}
```

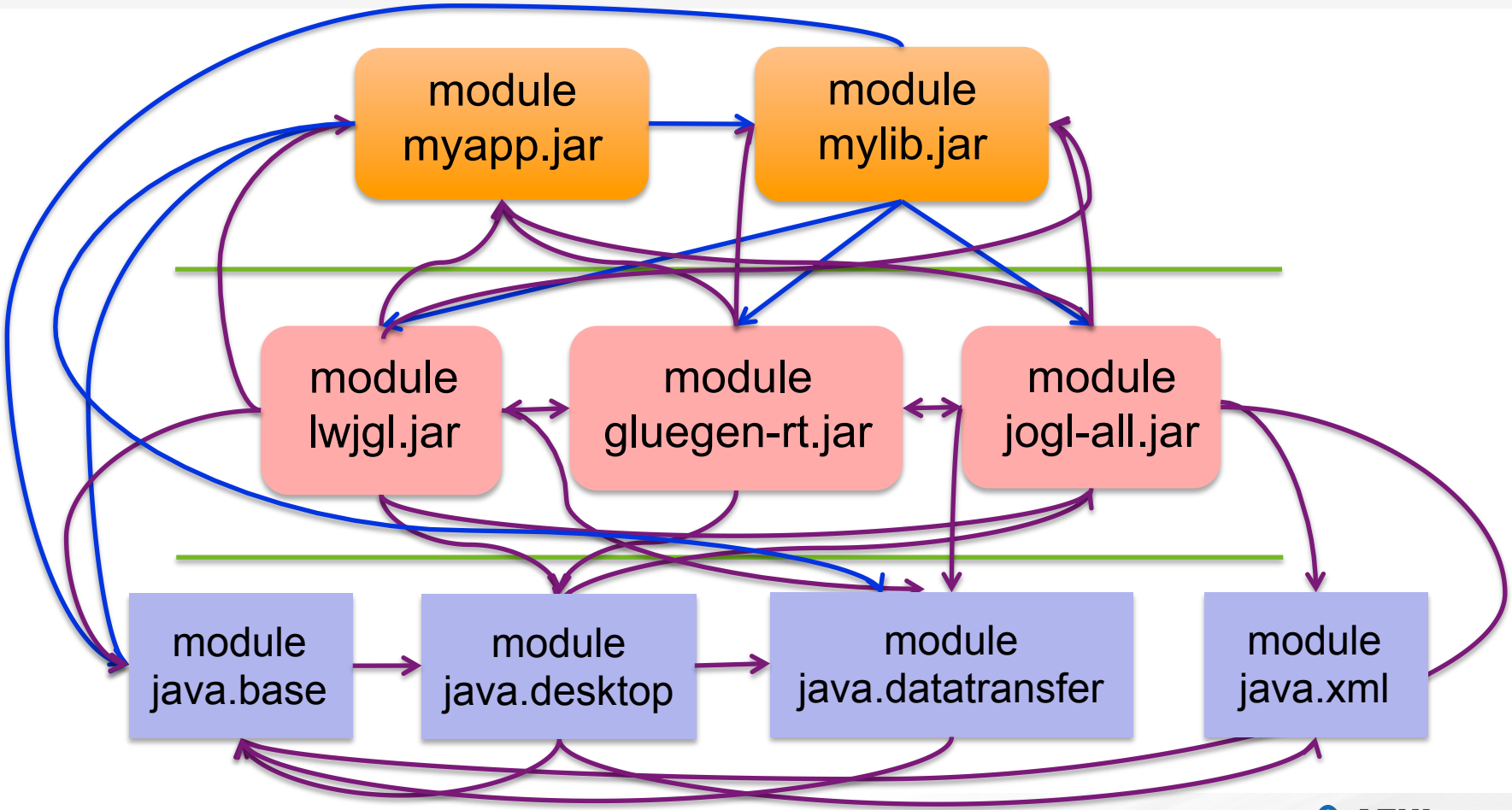
Sample Application



Automatic Modules

- Real modules
- Reuse an existing JAR without change
- Module name derived from JAR file name
- Exports all its packages
  - No selectivity
- Requires all modules accessible from the module path

Sample Application



Run Application With Modules

```
$ java -classpath \
  lib/myapp.jar: \
  lib/mylib.jar: \
  lib/liblwjgl.jar: \
  lib/gluegen-rt.jar: \
  lib/jogl-all.jar: \
  myapp.Main

$ java –mp mylib:lib –m myapp
```

Summary & Further Information

Summary

- Modularisation is a big change for Java
  - JVM/JRE rather than language/APIs
- Potentially disruptive changes to exposure of non-public APIs
  - Is it safe?
- Developing modular code will require some learning
  - Not a huge change, though

Zulu.org

- Azul binary distribution of OpenJDK source code
- Passed all TCK tests
- Completely FREE!
  - Pay us if you'd like enterprise level support
- Just announced: Embedded Zulu support for ARM 32-bit
  - Intel already supported
  - "Requires no licensing fee"

Further Information

- openjdk.java.net
- openjdk.java.net/jeps
- openjdk.java.net/projects/jigsaw
- jcp.org

Questions

Simon Ritter
Deputy CTO, Azul Systems

@speakjava | azul.com