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

module name   main class

```
$ java -mp mods -m com.azul.app/com.azul.app.Main

Azul application initialised!
```

- -modulepath can be abbreviated to -mp

Packaging With Modular JAR Files

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

app.jar

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

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

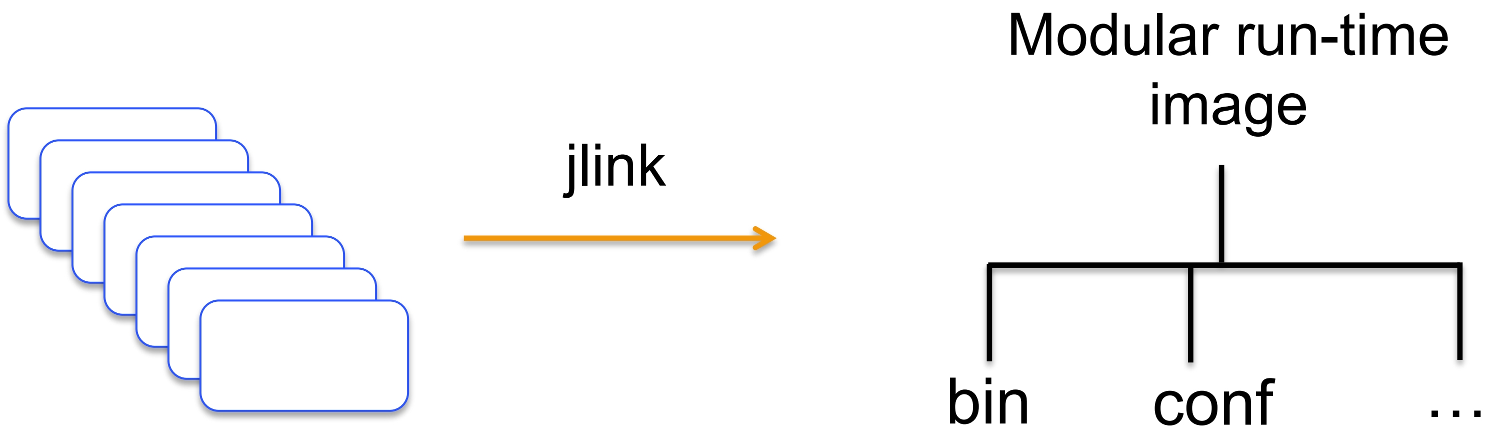
JAR Files & Module Information

```
$ 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.app

Azul application initialised!
```

Linking



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

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

Linking An Application

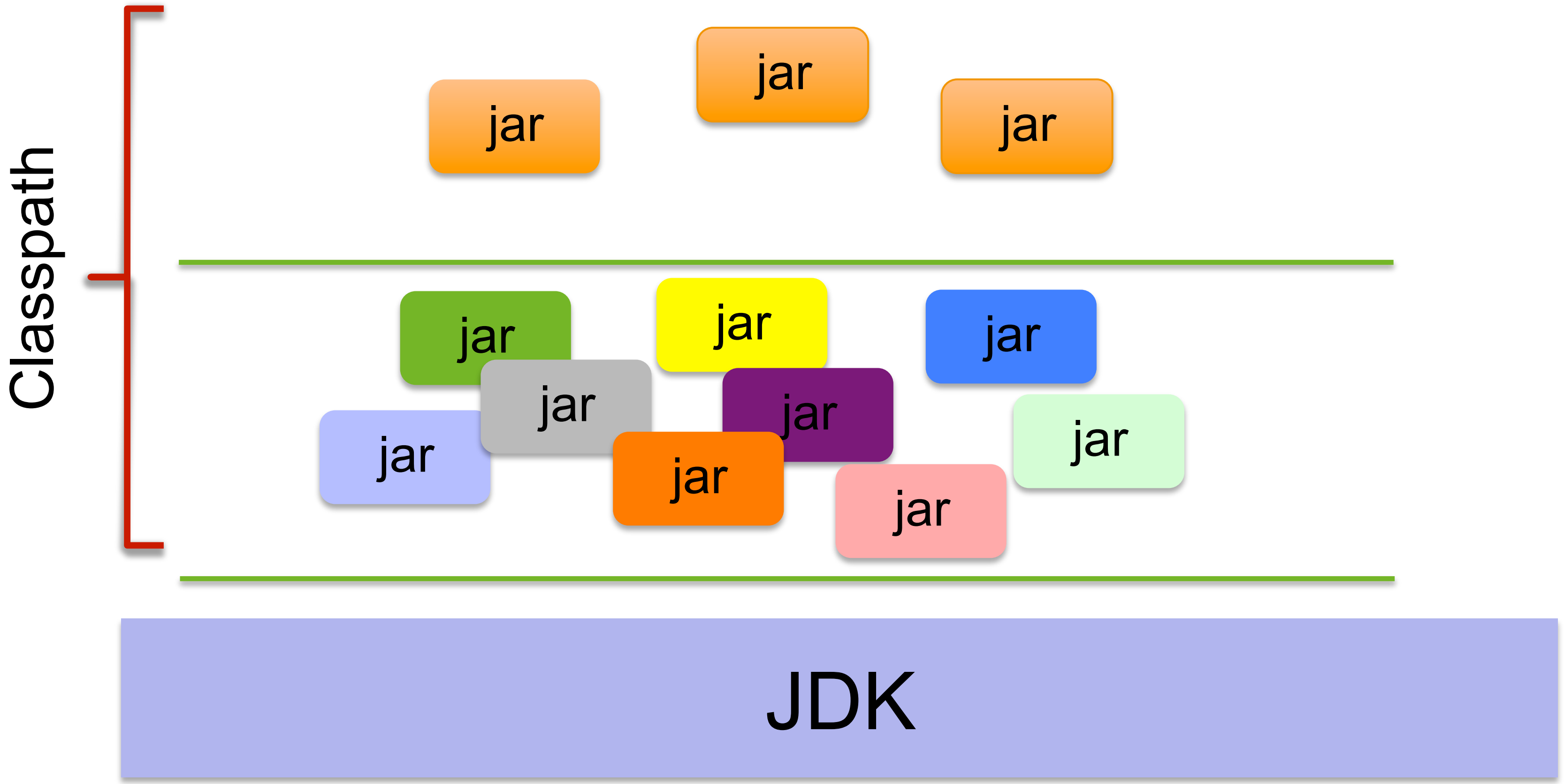
```
$ jlink --modulepath $JDKMODS:$MYMODS \
  --addmods com.azul.app –output myimage

$ 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

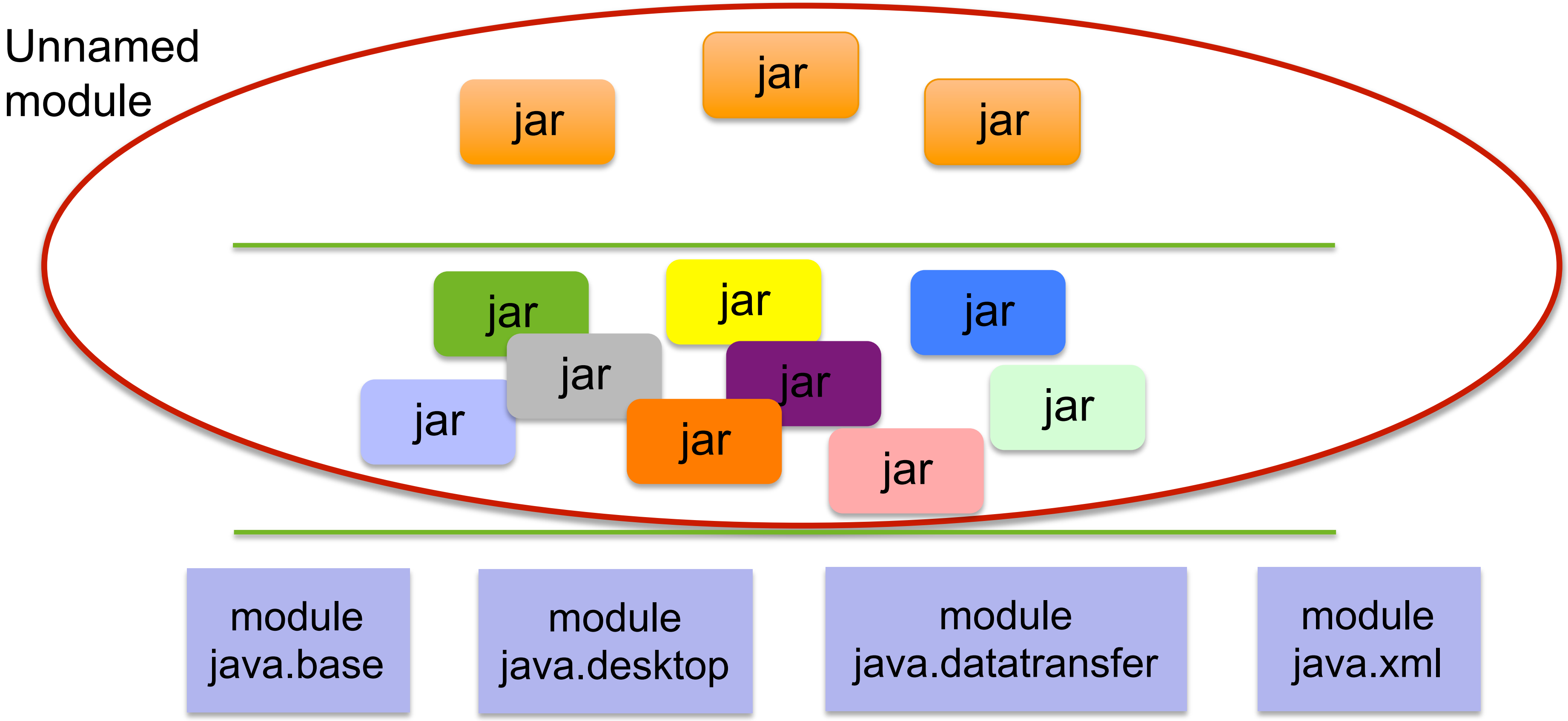
"It is not a goal of the module system to solve the version-selection problem"

Application Migration

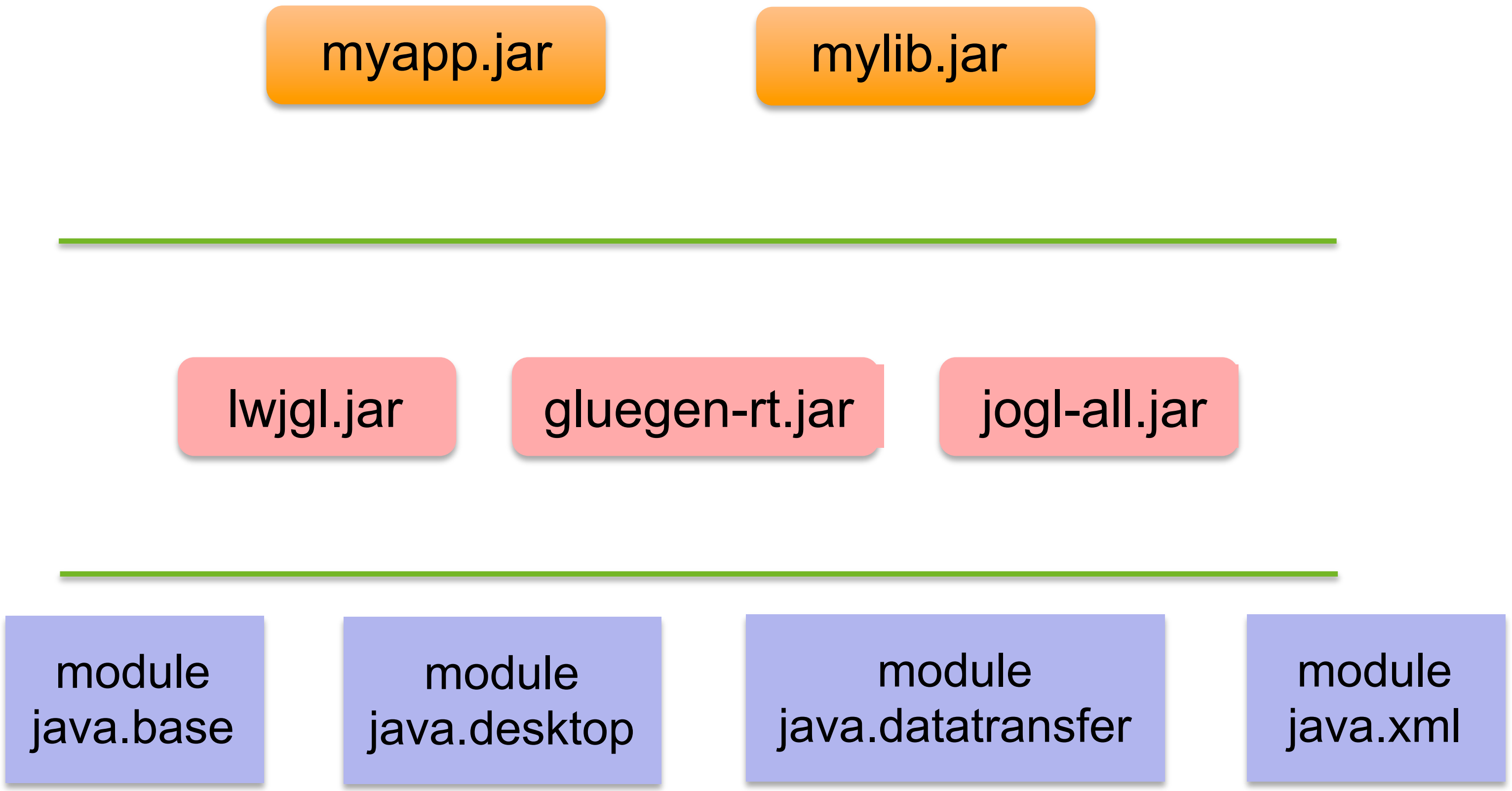
Typical Application (JDK 8)



Typical Application (JDK 9)



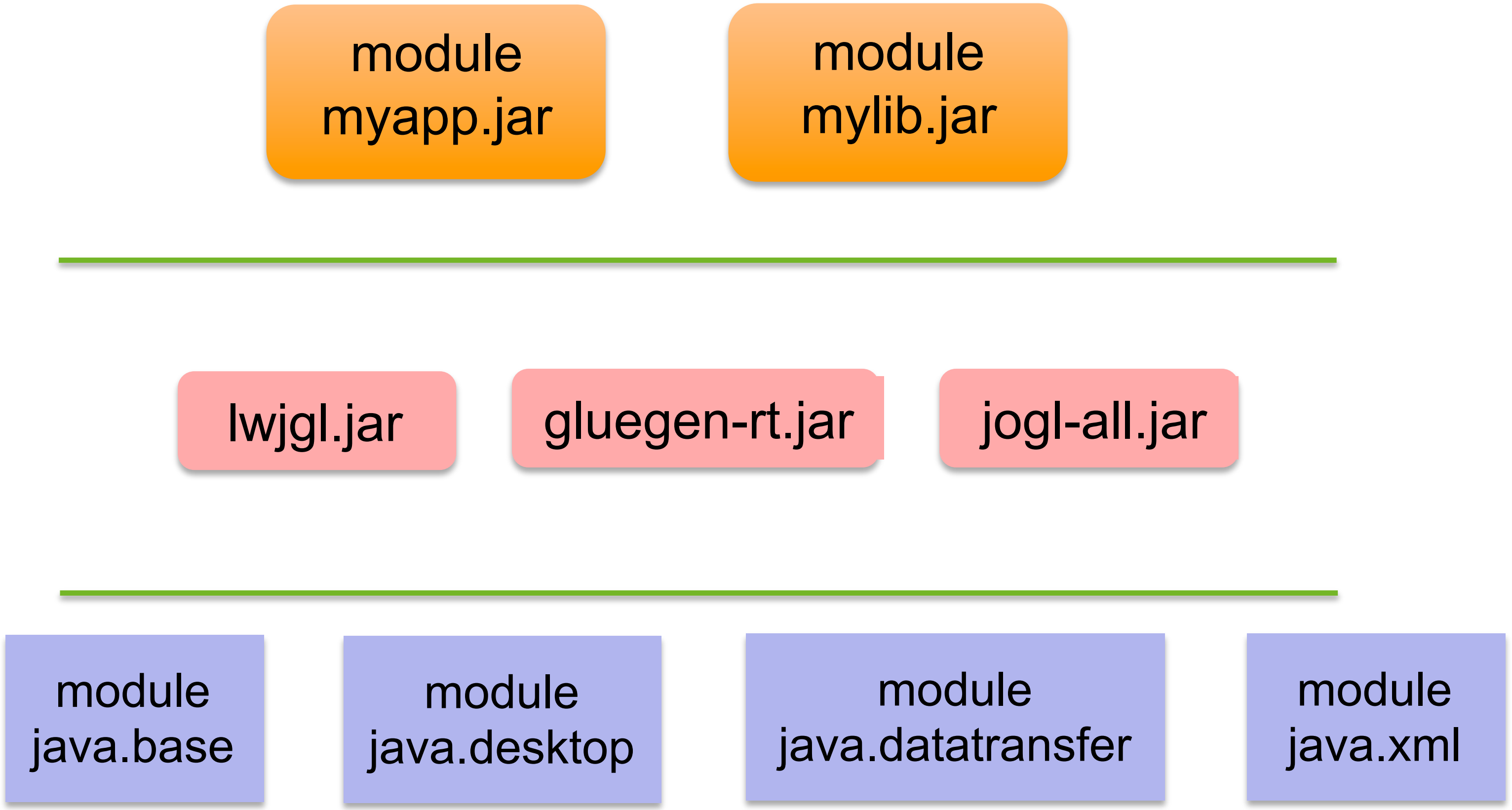
Sample Application



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
```

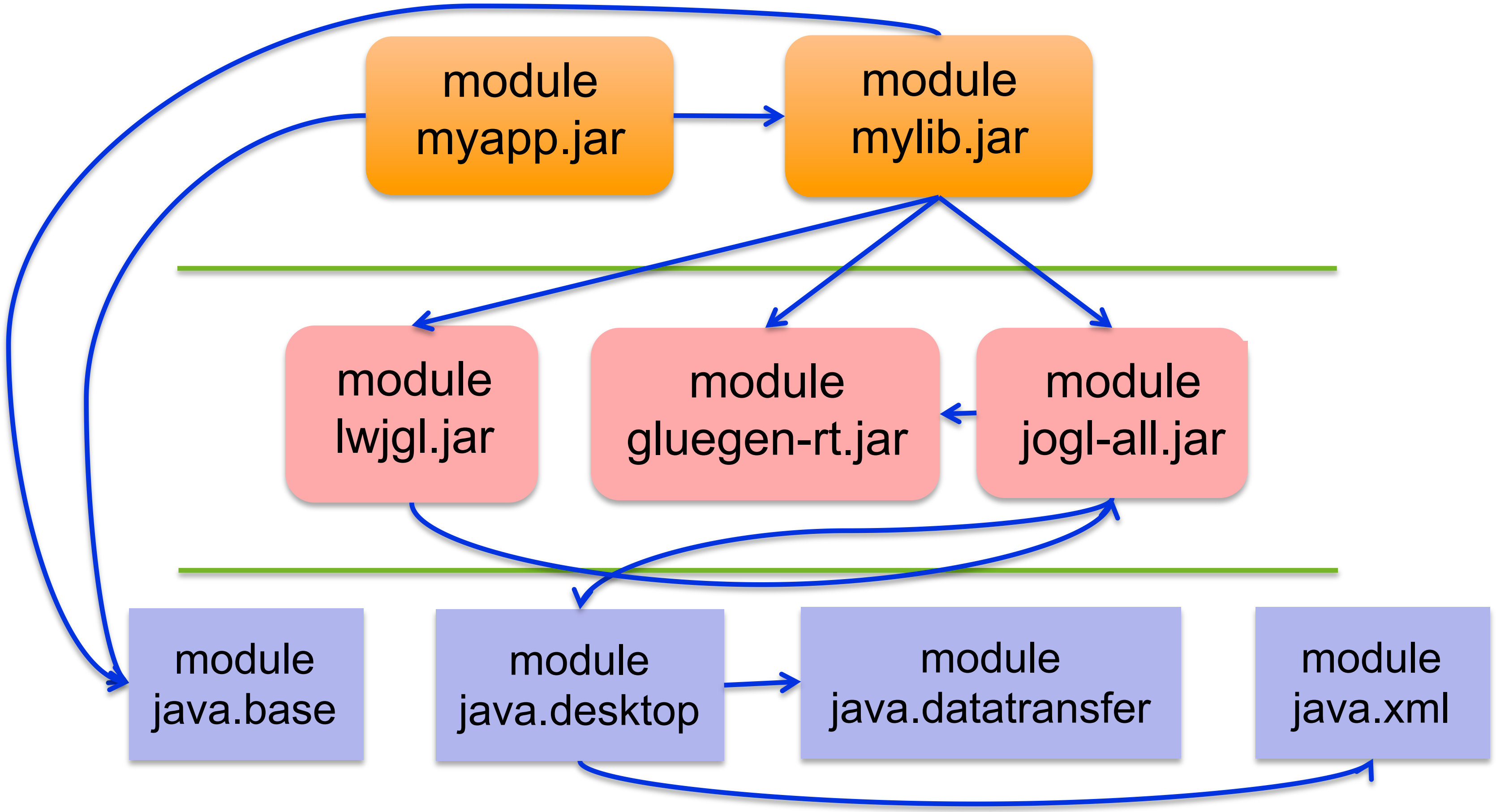
Sample Application



Application module-info.java

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

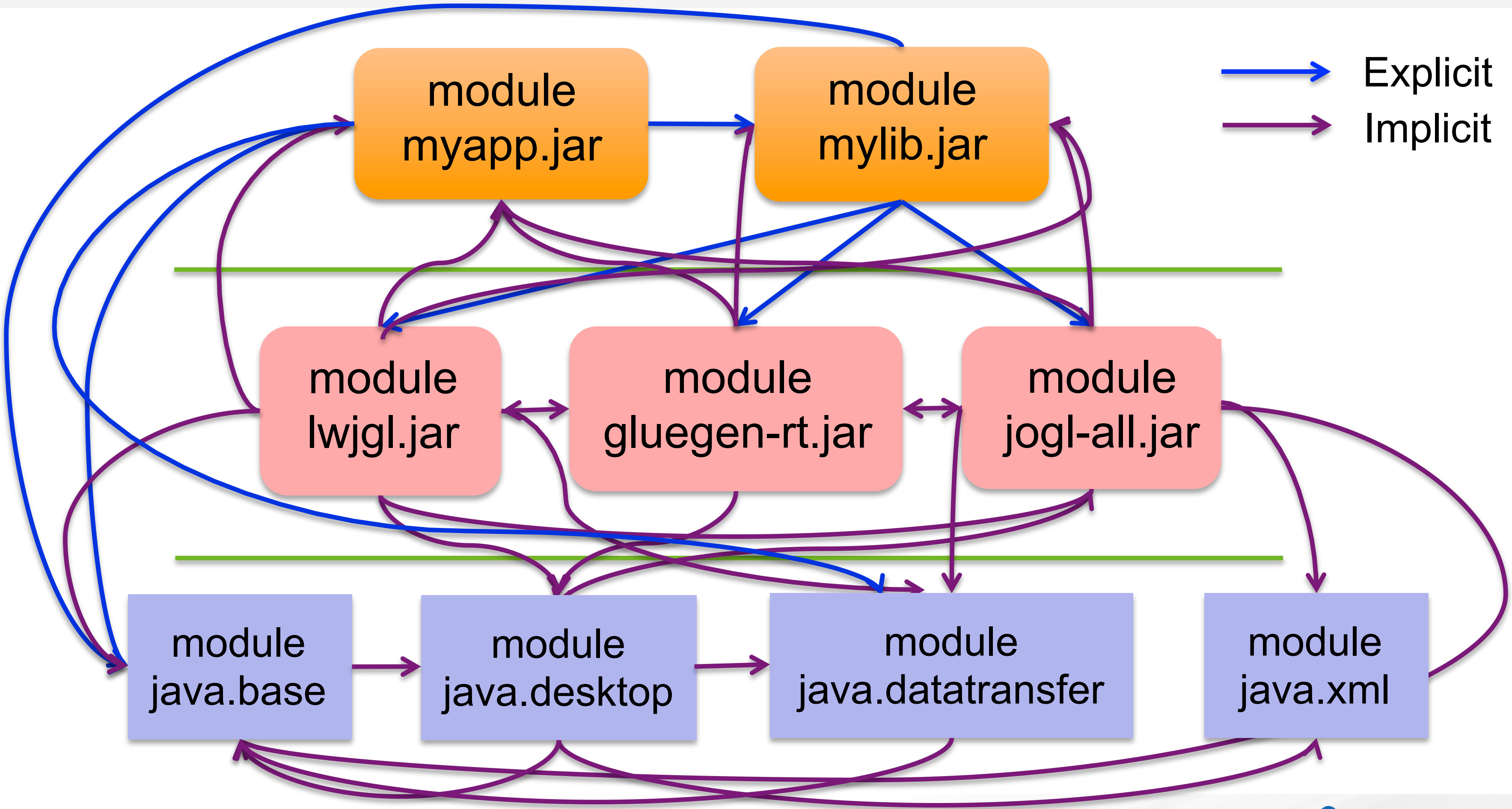
Sample Application



Automatic Modules

- Real modules
- Simply place unmodified jar file on module path
  - Rather than classpath
- No changes to JAR file
- Module name derived from JAR file name
- Exports all its packages
  - No selectivity
- Automatically requires all modules on the module path

Application Module Dependencies



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
```

Advanced Stuff

Modular Jar Files And JMODs

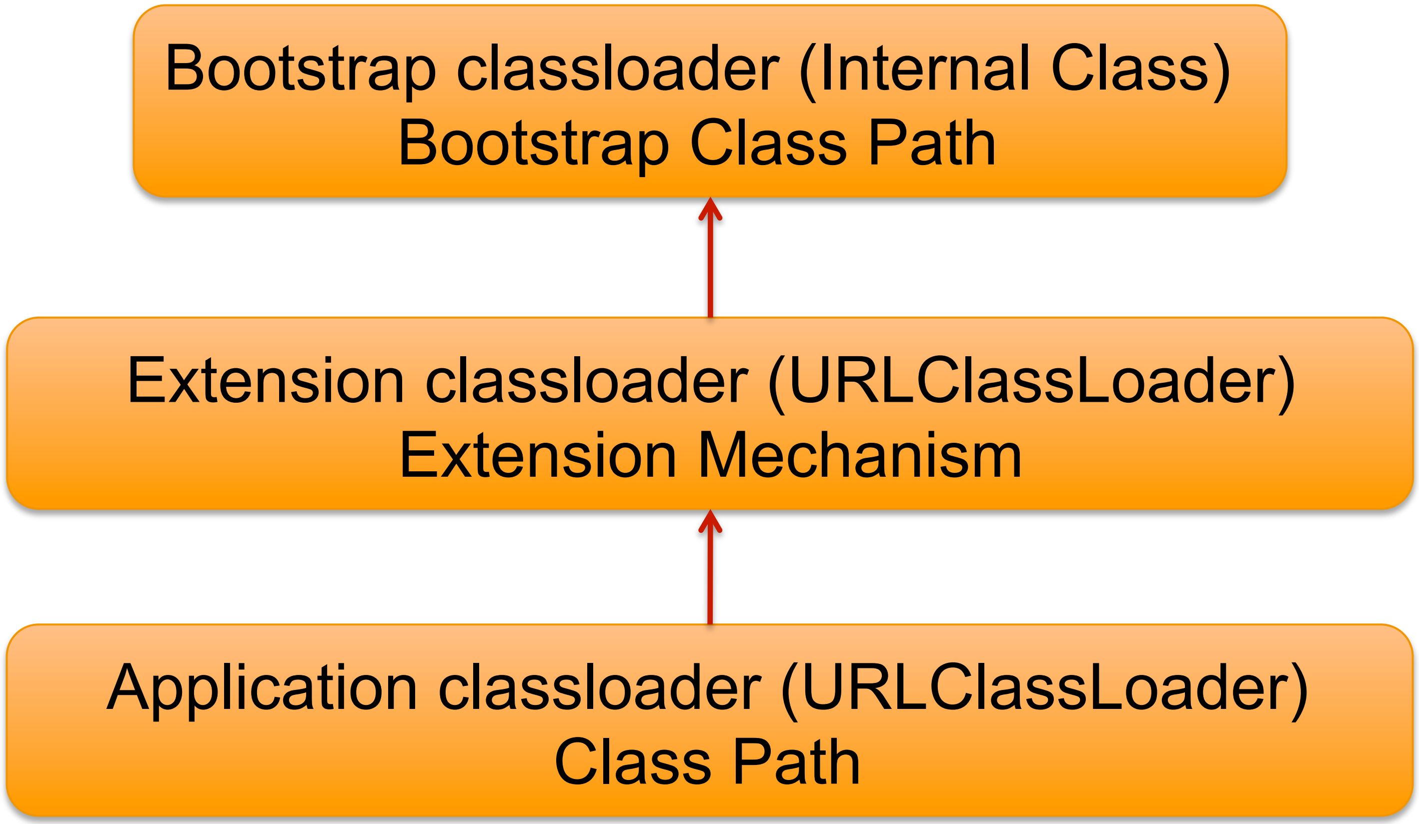
- Modular jar files are simple
  - Standard jar file possibly with module-info.class file
  - Can use existing (unmodified) jar files
- JMOD files
  - More complex module files
  - Used for modules in the JDK
  - Can include native files (JNI), configuration files and other data
  - Based on zip file format (pending final details - JEP 261)

jmod Command

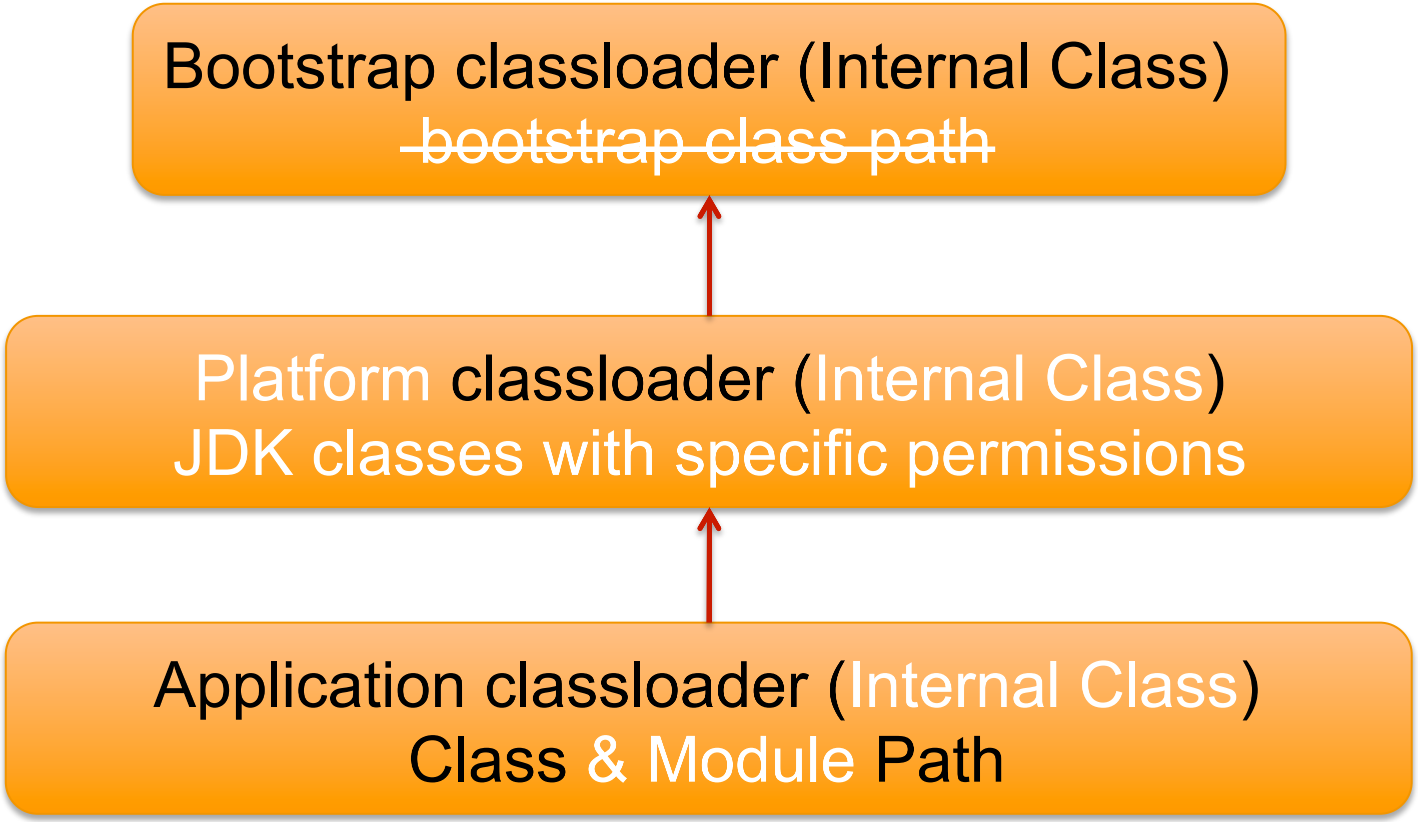
```
jmod (create | list | describe) <options> <jmod-file>
```

- Create can specify several details:
  - Main class
  - Native libraries and commands
  - Configuration data
  - OS name, version and machine architecture
- Details included in module-info.class file

Classloaders (Since JDK 1.2)


Bootstrap classloader (Internal Class)
Bootstrap Class Path
Extension classloader (URLClassLoader)
Extension Mechanism
Application classloader (URLClassLoader)
Class Path

Classloaders (JDK 9)



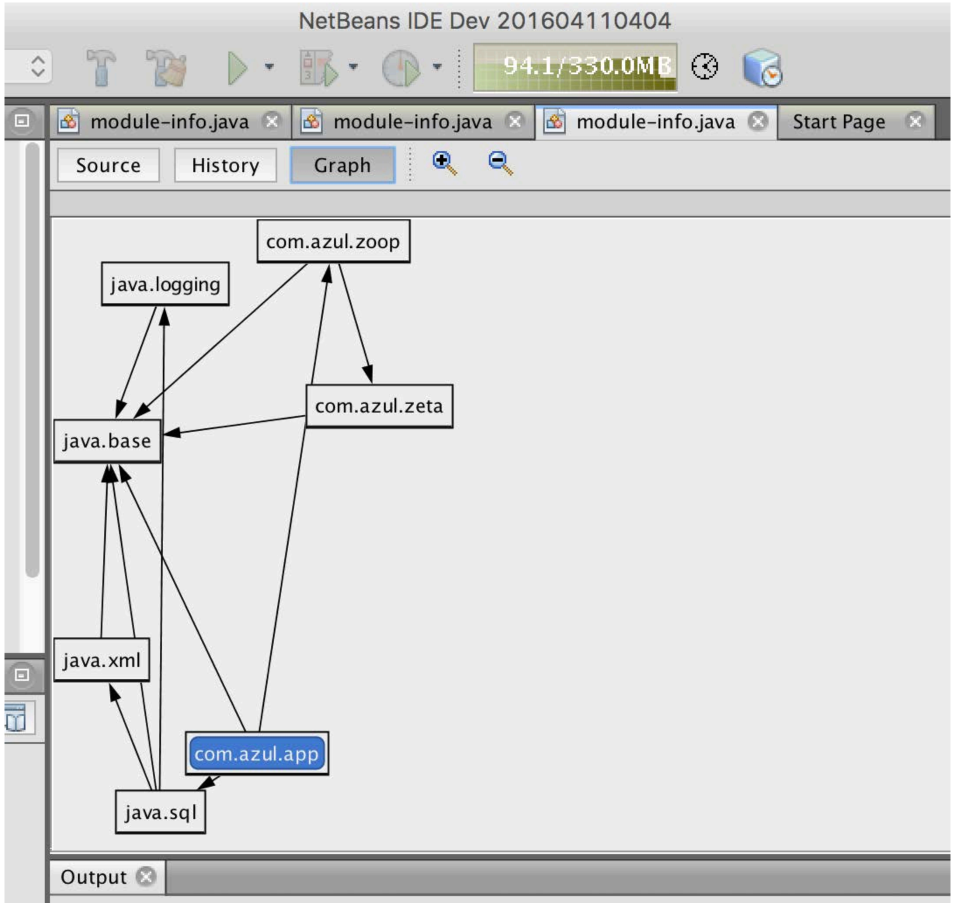
Bootstrap classloader (Internal Class)
~~bootstrap class path~~
Platform classloader (Internal Class)
JDK classes with specific permissions
Application classloader (Internal Class)
Class & Module Path

Summary & Further Information

Tooling Support

- NetBeans leads the way
- Early Access NetBeans 9 available
- Support for module-info.java file
  - Graphing of dependencies

NetBeans Tooling



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

Further Information

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

- www.zulu.org

Questions

Simon Ritter
Deputy CTO, Azul Systems

@speakjava | azul.com