Modularization with Project Jigsaw in JDK 9

© Copyright Azul Systems 2015

Simon Ritter

Deputy CTO, Azul Systems


@speakjava | azul.com

© Copyright Azul Systems 2016

Agenda

- JDK 9 API structure overview
- Introduction to Jigsaw and modules
- Developing code with modules
- Summary and further information

© Copyright Azul Systems 2016

JDK 9 API Structure Overview

© Copyright Azul Systems 2016

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
- See how long you can keep a project going for

© Copyright Azul Systems 2016

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

© Copyright Azul Systems 2016

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

© Copyright Azul Systems 2016

JDK 9: Incompatible Changes

- Encapsulate most JDK internal APIs
- Remove a small number of supported APIs
  - 6 in total, all add/remove PropertyChangeListener
- Change the binary structure of the JRE and JDK
- Remove the endorsed-standards override and extension mechanism
- New version string format
- A single underscore will no longer be allowed as an identifier in source code

© Copyright Azul Systems 2016

Most Popular Unsupported APIs

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

Oracle dataset based on internal application code

© Copyright Azul Systems 2016

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

© Copyright Azul Systems 2016

JEP 260 Proposal

1. Encapsulate all non-critical JDK-internal APIs
2. Encapsulate all critical JDK-internal APIs, for which supported replacements exist in JDK 8
3. 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

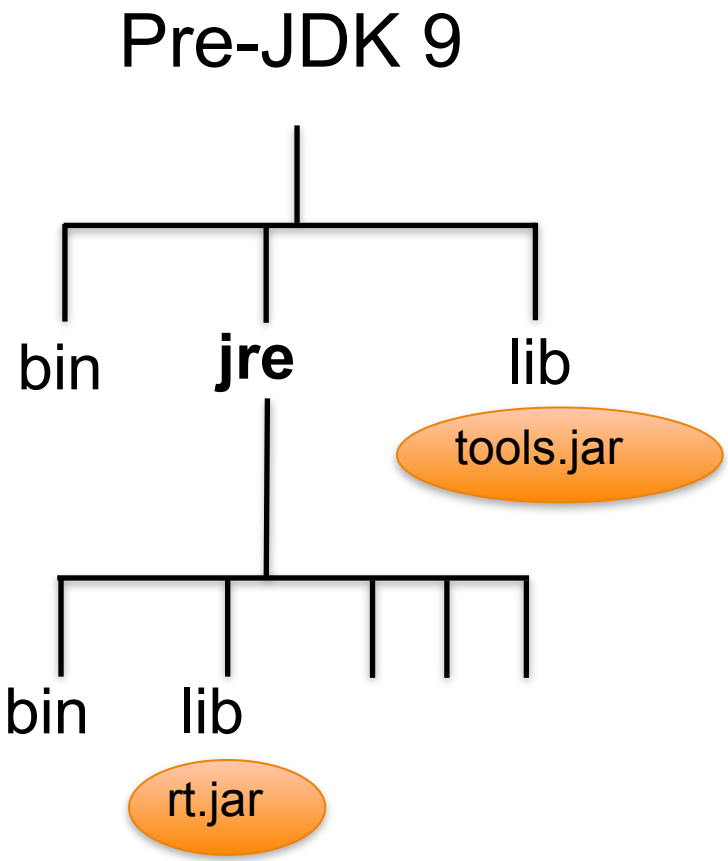
© Copyright Azul Systems 2016

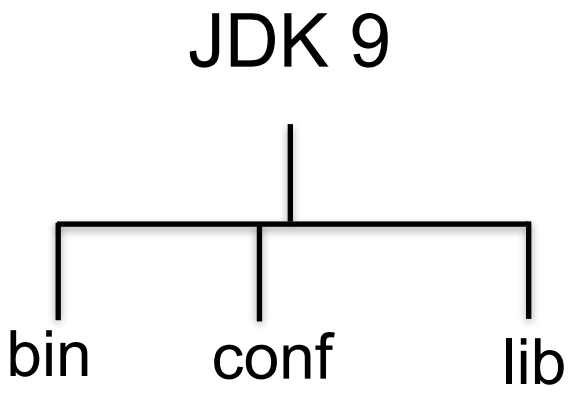
Binary Structure Of JDK/JRE

- 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

© Copyright Azul Systems 2016

JDK Structure





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

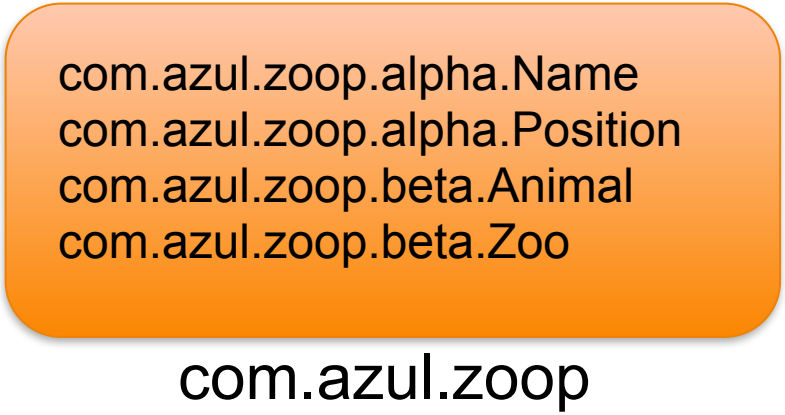
© Copyright Azul Systems 2016

Introduction to Jigsaw and Modules

© Copyright Azul Systems 2016

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



© Copyright Azul Systems 2016

Module Declaration

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

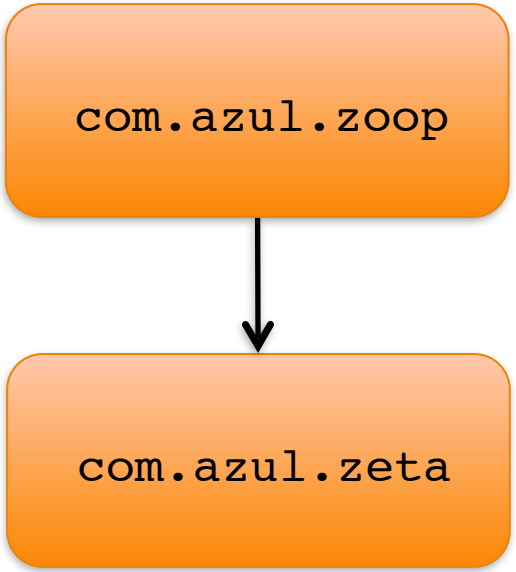
module-info.java

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

© Copyright Azul Systems 2016

Module Dependencies

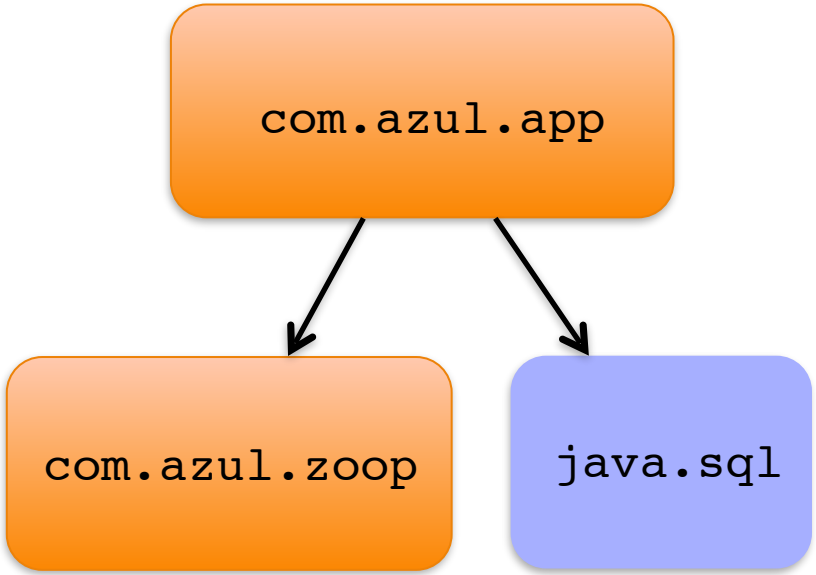
```
module com.azul.zoop {
   requires
com.azul.zeta;
}
```



© Copyright Azul Systems 2016

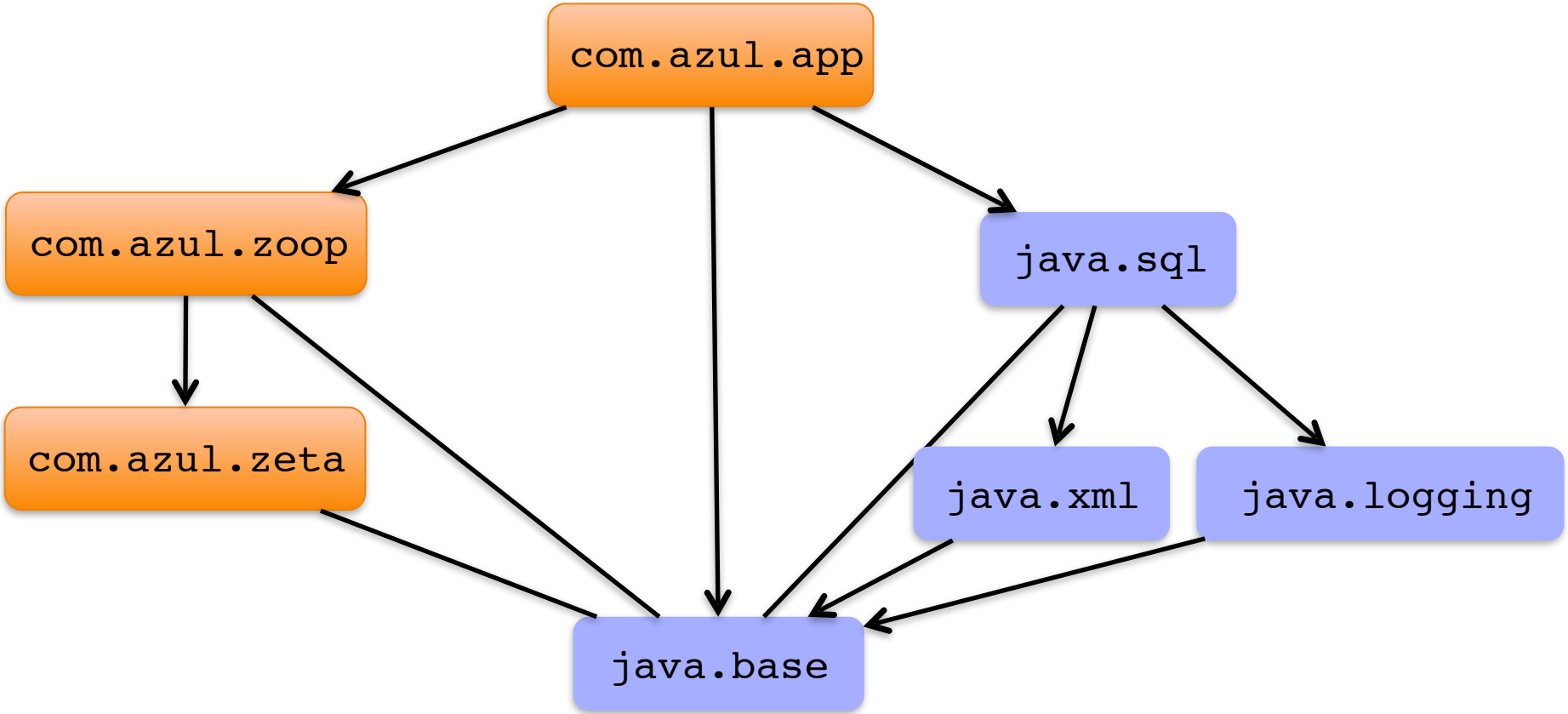
Module Dependencies

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



© Copyright Azul Systems 2016

Module Dependency Graph



© Copyright Azul Systems 2016

Package Visibility

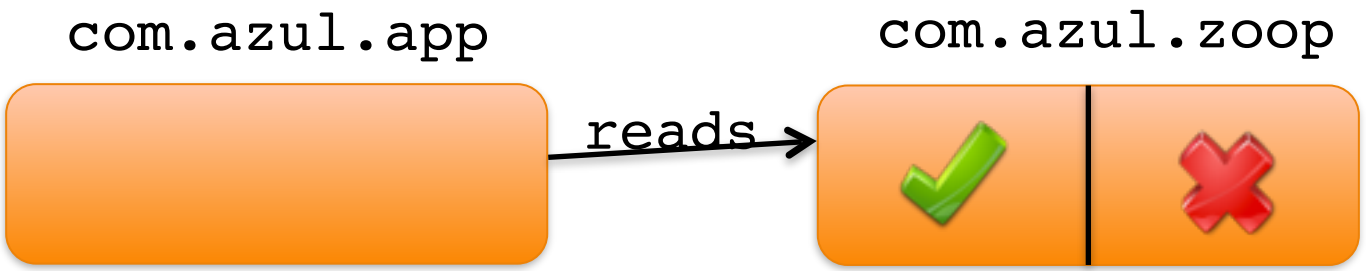
```
module com.azul.zoop {
  exports
com.azul.zoop.alpha;
  exports com.azul.zoop.beta;
}
```

com.azul.zoop

© Copyright Azul Systems 2016

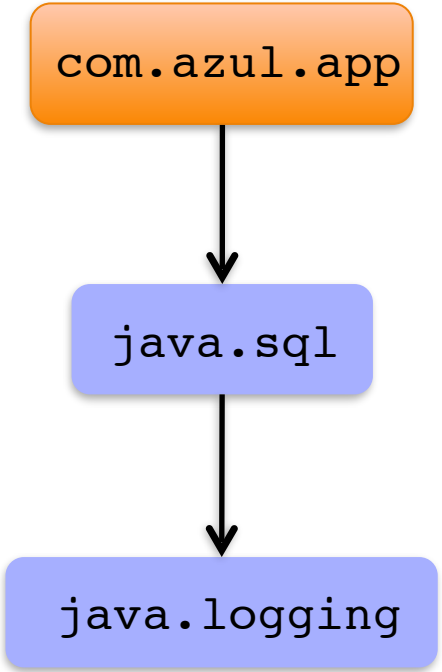
Accessibility

- 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



© Copyright Azul Systems 2016

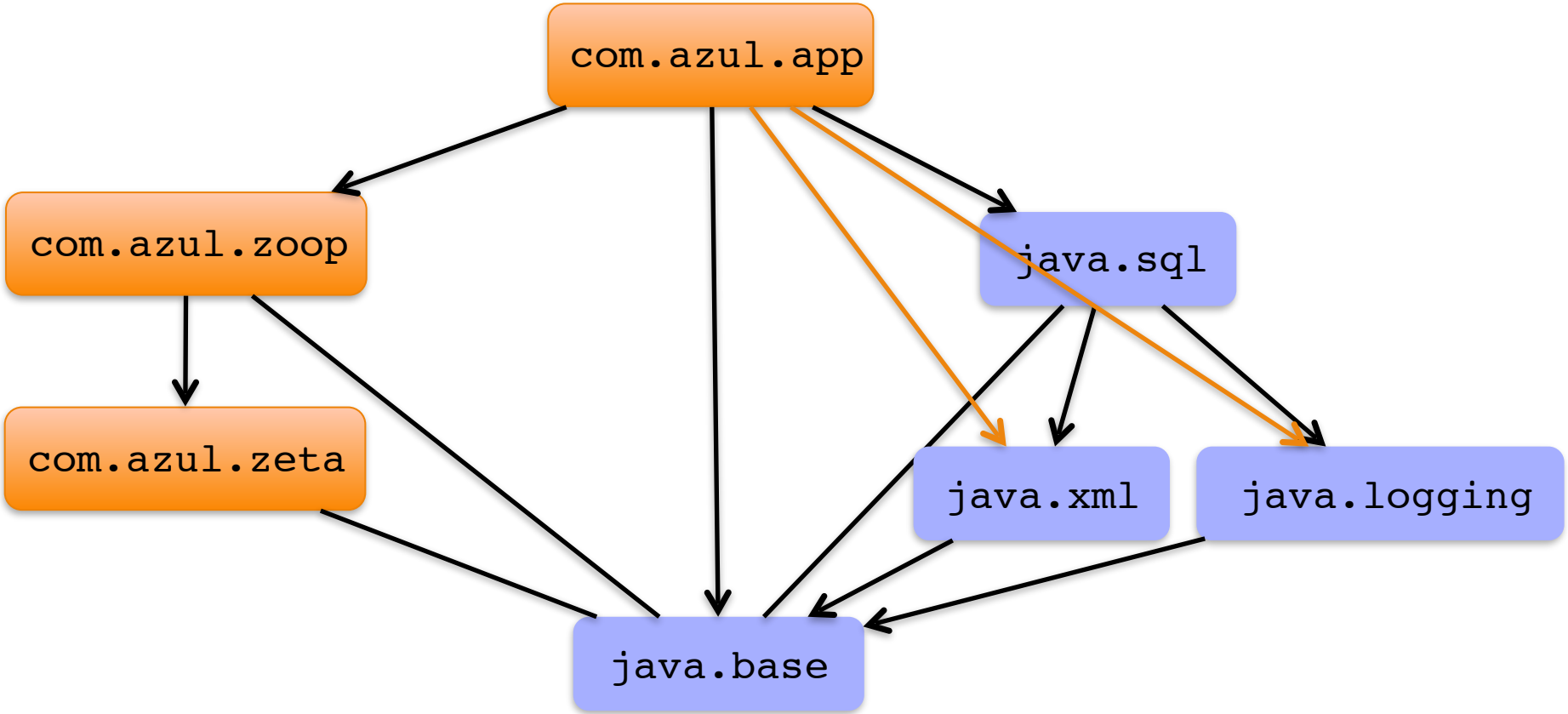
Readability v. Dependency




```
    Driver d = …
    Logger l =
    d.getParentLogger();
    l.log(“azul’);
module java.sql {
  requires public
java.logging;
}
```

© Copyright Azul Systems 2016

Module Readability Graph



© Copyright Azul Systems 2016

Java Accessibility (pre-JDK 9)

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

© Copyright Azul Systems 2016

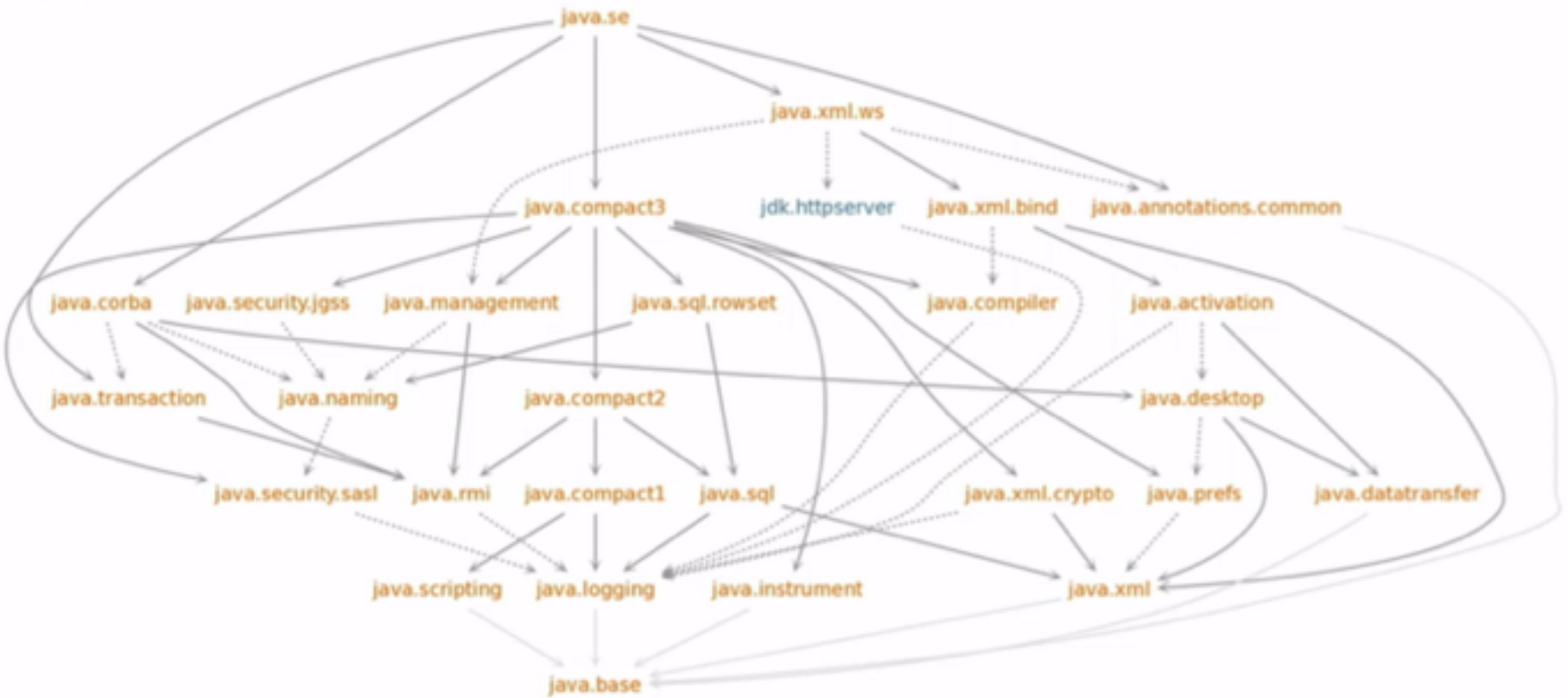
Java Accessibility (JDK 9)

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

public ≠ accessible (fundamental change to Java)

© Copyright Azul Systems 2016

JDK Platform Modules



© Copyright Azul Systems 2016

Developing Code With Modules

© Copyright Azul Systems 2016

Compilation

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

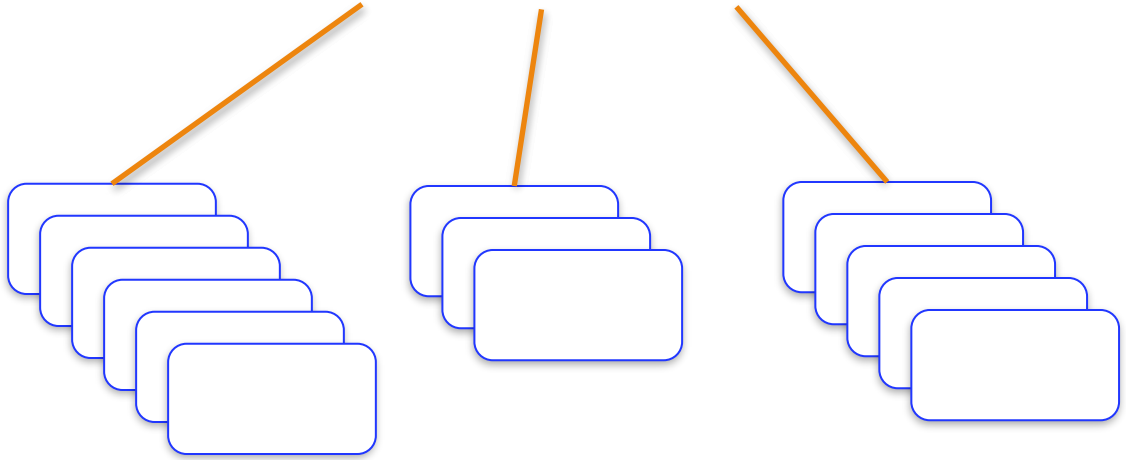
```
src/zeta/mod-info.java
src/zeta/com/azul/zeta/
Vehicle.java
```

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

© Copyright Azul Systems 2016

Module Path

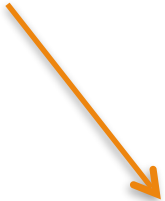
```
$ javac –modulepath dir1:dir2:dir3
```



© Copyright Azul Systems 2016

Compilation With Module Path

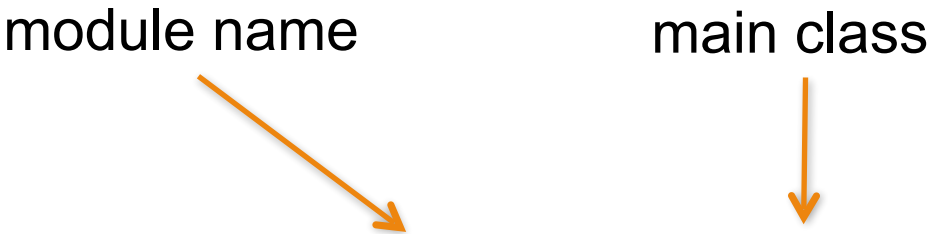
```
$ javac –modulepath mlib –d mods \
  src/zoop/module-info.java \
  src/zoop/com/azul/zoop/alpha/
Name.java
 src/zoop/mod-info.java
 src/zoop/com/azul/zoop/alpha/Name.java
```



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

© Copyright Azul Systems 2016

Application Execution




```
$ java –modulepath mods –m com.azul.app/
com.azul.app.Main

Azul application initialised!
```

- -modulepath can be abbreviated to -mp

© Copyright Azul Systems 2016

Packaging With Modular Jars

```
mods/app/mod-info.class
mods/app/com/azul/app/Main.class
```

app.jar

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

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

© Copyright Azul Systems 2016

Jar Files & Module Information

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

© Copyright Azul Systems 2016

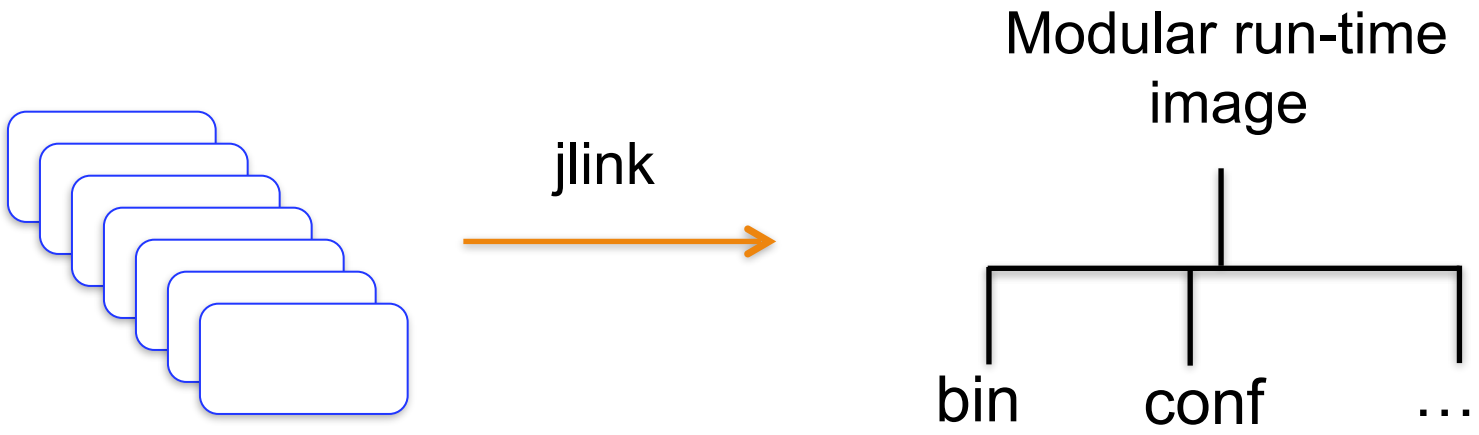
Application Execution (JAR)

```
$ java –mp mlib:mods –m com.azul.app.Main

Azul application initialised!
```

© Copyright Azul Systems 2016

Linking




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

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

© Copyright Azul Systems 2016

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
```

© Copyright Azul Systems 2016

Summary & Further Information

© Copyright Azul Systems 2016

Summary

- Modularisation is a big change for Java
  - JVM/JRE rather than language/APIs
  - Public access isn't necessarily the same
- Flexibility to define what is exported
- New linking capability to generate runtime image
- More to learn about converting existing code
- Will make all applications simpler to deploy and manage

© Copyright Azul Systems 2016

Further Information

- openjdk.java.net
- openjdk.java.net/jeps
  - 200, 201, 220, 260, 261
- openjdk.java.net/projects/jigsaw
- jcp.org
  - JSR 376

- www.zulu.org (OpenJDK supported builds JDK 6, 7, 8, 9)
- www.azul.com (Zing JVM for low latency)

© Copyright Azul Systems 2016

Modularization with Project Jigsaw in JDK 9

© Copyright Azul Systems 2015

Simon Ritter

Deputy CTO, Azul Systems


@speakjava | azul.com

© Copyright Azul Systems 2016