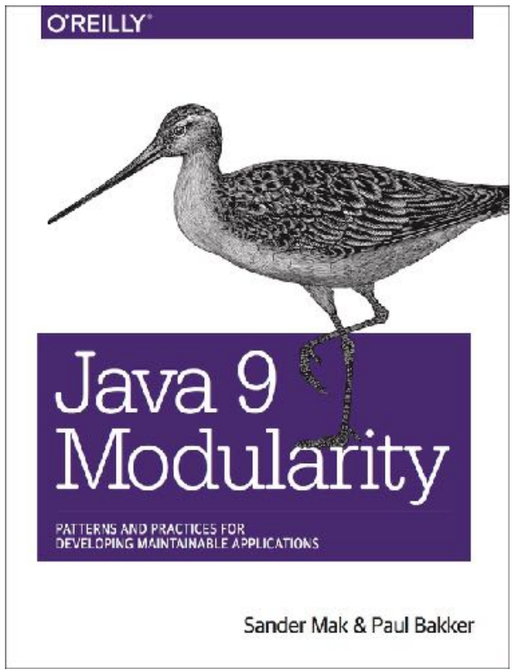
QCon LONDON

luminis
Conversing worlds



Migrating to Java 9 Modules

By Sander Mak

@Sander_Mak

Migrating to Java 9

Java 8

```
java -cp .. -jar myapp.jar
```

Java 9

```
java -cp .. -jar myapp.jar
```

Today's journey

Running on Java 9

Java 9 modules

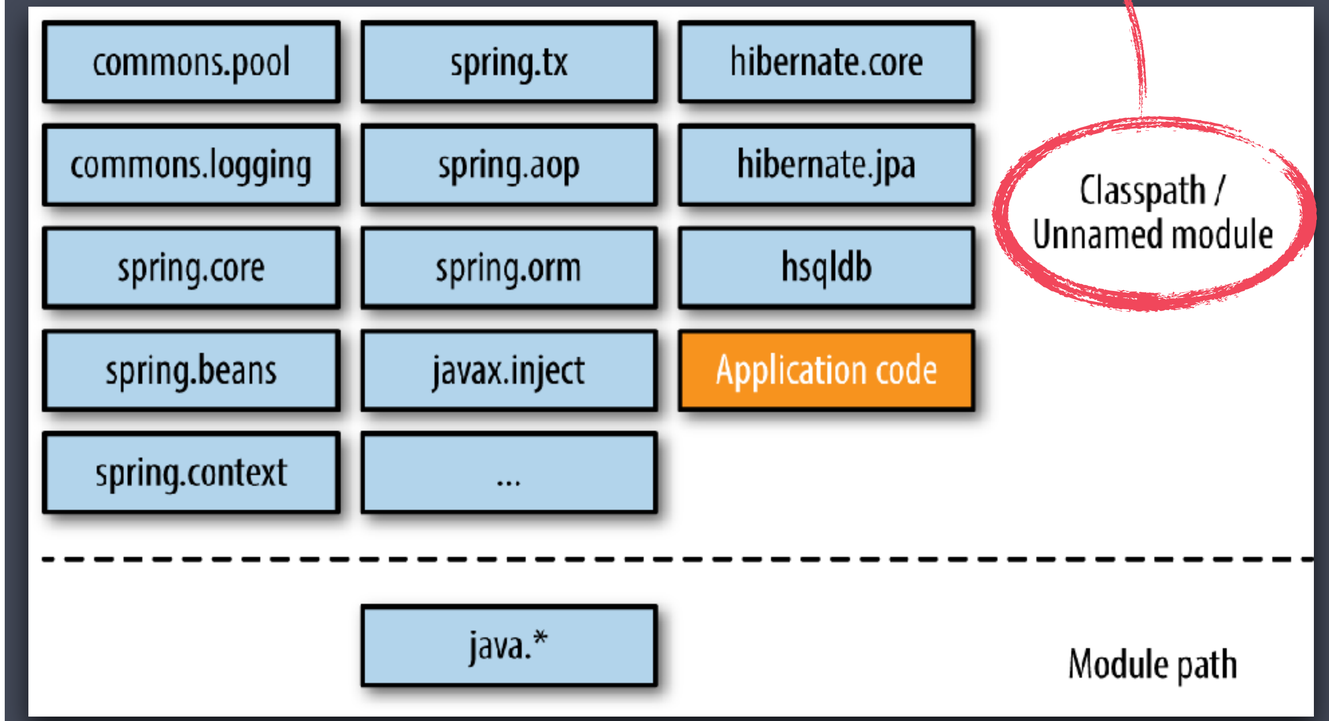
Migrating to modules

Modularising code

```
├── lib
├── run.sh
└── src
    ├── books
    │   ├── api
    │   │   ├── entities
    │   │   │   └── Book.java
    │   │   └── service
    │   │       └── BooksService.java
    │   └── impl
    │       ├── entities
    │       │   └── BookEntity.java
    │       └── service
    │           └── HibernateBooksService.java
    ├── bookstore
    │   ├── api
    │   │   └── service
    │   │       └── BookstoreService.java
    │   └── impl
    │       └── service
    │           └── BookstoreServiceImpl.java
    ├── log4j2.xml
    ├── main
    │   └── Main.java
    └── main.xml
```

Application to Migrate

Unnamed module?!



Migrating to Java 9

Java 8

```
java -cp .. -jar myapp.jar
```

Java 9

```
java -cp .. -jar myapp.jar
```

Let's try it!

Classpath migration problems

- Unresolved enterprise modules
- (Ab)use of platform internals
- Split package conflicts (e.g. jsr305 with javax.annotation)

Missing enterprise modules

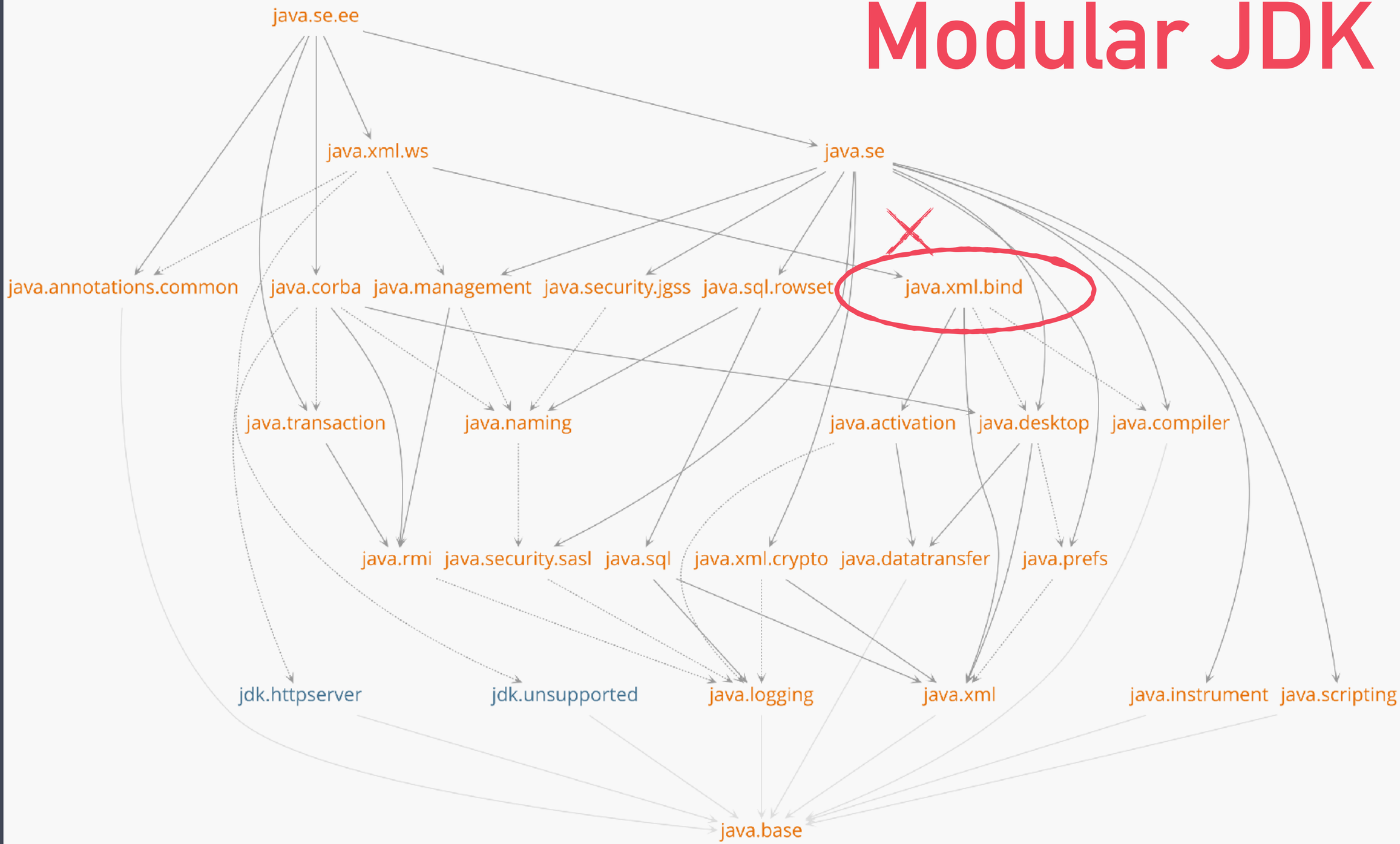
```
import javax.xml.bind.DatatypeConverter;

public class Main {

    public static void main(String... args) {
        DatatypeConverter.parseBase64Binary("SGVsbG8gd29ybGQh");
    }
}
```

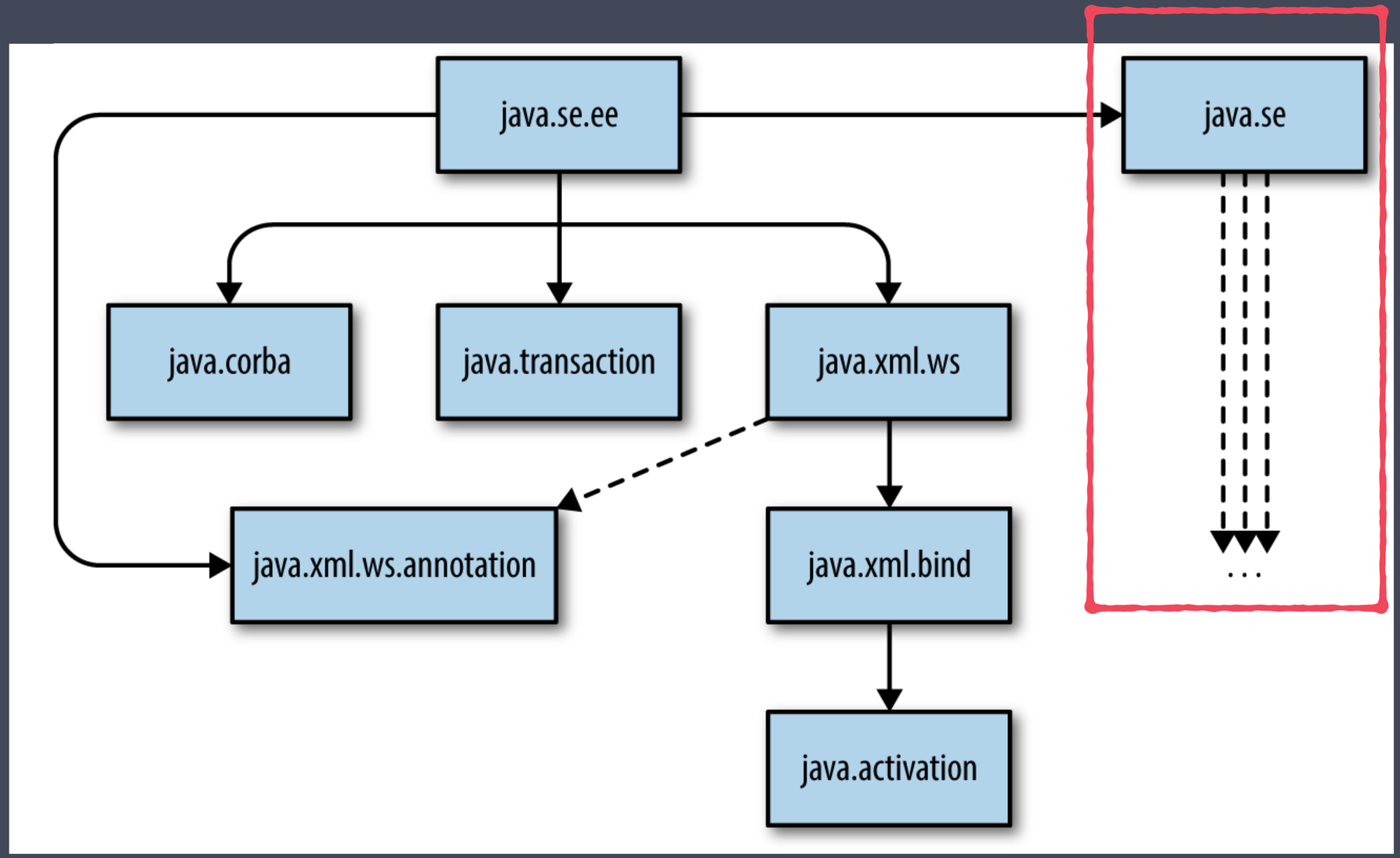
```
error: package javax.xml.bind does not exist
```

Modular JDK


java.se.ee
java.xml.ws
java.se
java.annotations.common
java.corba
java.management
java.security.jgss
java.sql.rowset
java.xml.bind
java.transaction
java.naming
java.activation
java.desktop
java.compiler
java.rmi
java.security.sasl
java.sql
java.xml.crypto
java.datatransfer
java.prefs
jdk.httpserver
jdk.unsupported
java.logging
java.xml
java.instrument
java.scripting
java.base

Missing enterprise modules

Classpath
compile-time
“java.se”
module is used



Add unresolved enterprise modules

Find

```
jdeps demo/Main.class

Main.class -> java.base
Main.class -> not found
   <unnamed>           -> java.lang        java.base
   <unnamed>           -> javax.xml.bind   not found
```

Fix

```
javac --add-modules java.xml.bind demo/Main.java
```

```
java --add-modules java.xml.bind demo/Main.java
```

Better yet, add an implementation to classpath
(enterprise modules will be gone in Java 11!)

Add unresolved enterprise modules

```
javac --add-modules java.xml.bind demo/Main.java
```

```
java --add-modules java.xml.bind demo/Main.java
```

Let's try it!

JDK Warning in Production, What?!

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions ...)
         to method java.lang.ClassLoader.defineClass(..)
WARNING: Please consider reporting this to the maintainers of
         javassist.util.proxy.SecurityActions
WARNING: Use --illegal-access=warn to enable warnings of further illegal
         reflective access operations
WARNING: All illegal access operations will be denied in a future release
```

```
java --add-modules java.xml.bind
     --add-opens java.base/java.lang=ALL-UNNAMED
```

Using encapsulated APIs

```
public class Main {

  public static void main(String... args) throws Exception {
    sun.security.x509.X500Name name =
      new sun.security.x509.X500Name("CN=user");
  }
}
```

```
Main.java:4: error: package
sun.security.x509 does not exist

    sun.security.x509.X500Name name =
                     ^
```

Use jdeps to find problems

```
jdeps -jdkinternals Main.class

Main.class -> java.base
     Main       -> sun.security.x509.X500Name      JDK internal API (java.base)

JDK Internal API                        Suggested Replacement
----------------                        ---------------------
sun.security.x509.X500Name              Use javax.security.auth.x500.X500Principal
                                        @since 1.4
```

Using encapsulated APIs

Compile with 1.8, run with 9?

```
java --illegal-access=deny Main
```

```
Exception in thread "main" java.lang.IllegalAccessError: class Main
(in unnamed module @0x4cdbe50f) cannot access class
sun.security.x509.X500Name (in module java.base) because module
java.base does not export sun.security.x509 to unnamed module
@0x4cdbe50f
   at Main.main(Main.java:4)
```

```
java --add-exports java.base/sun.security.x509=ALL-UNNAMED Main
```

Classpath Migration Recap

Upgrade to latest library/tool versions

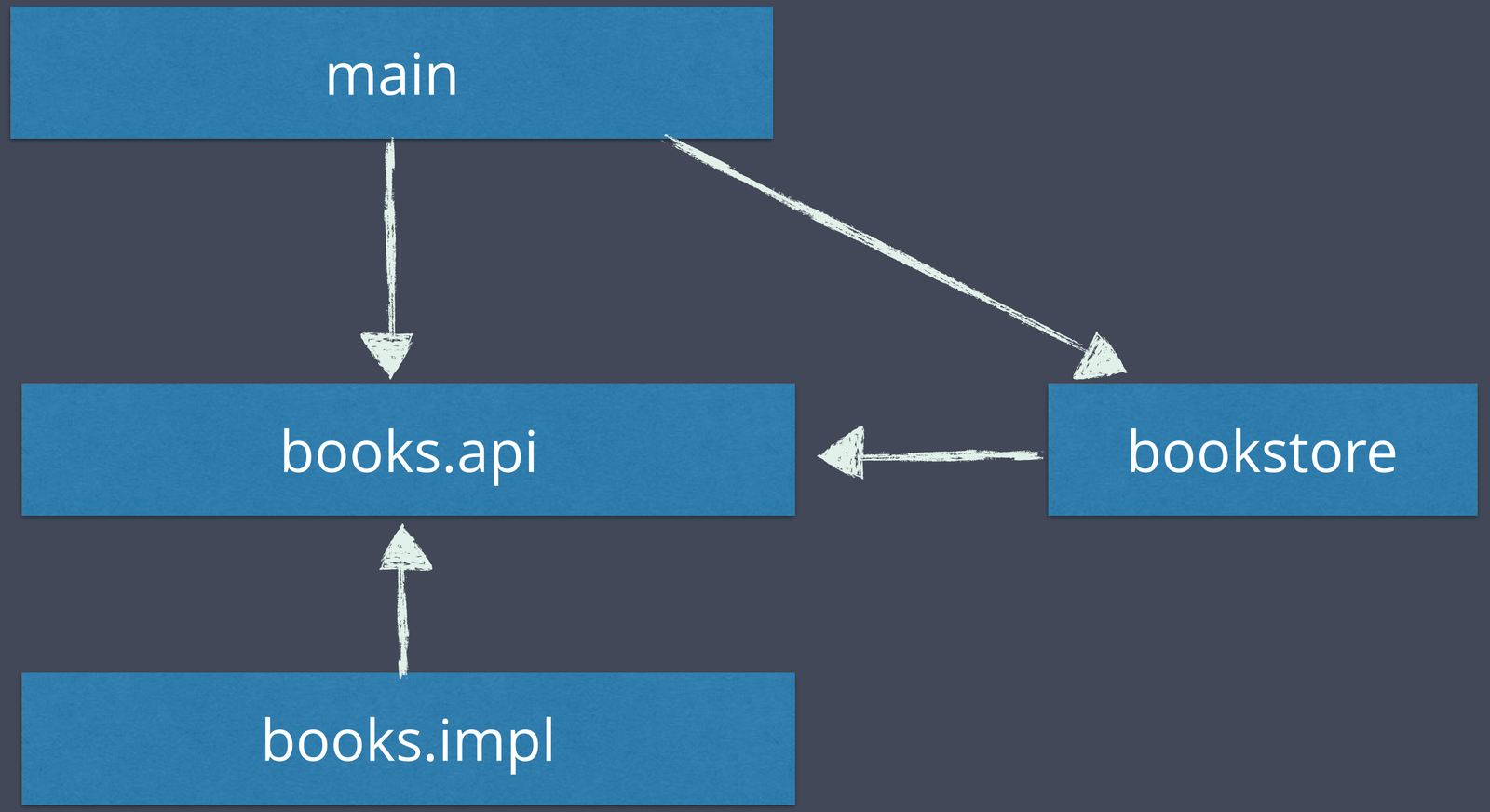
Use Jdeps to find potential issues:

- Dependencies on EE modules
- Use of JDK internal APIs
- Split packages

Fix illegal access warnings

Back to Modules...

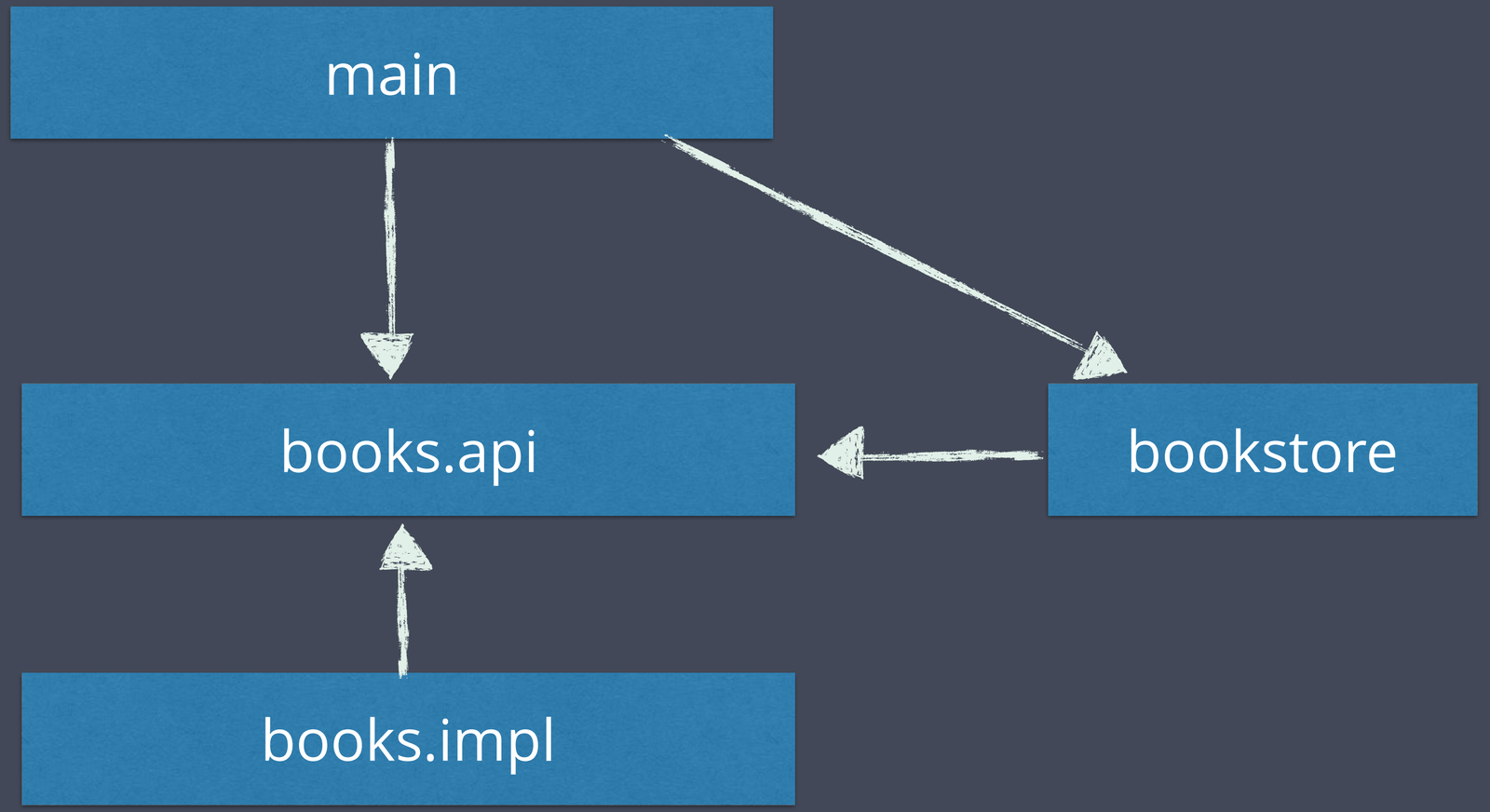
Mental picture of your app



Actual view of your app

```
CLASSPATH=lib/antlr-2.7.7.jar:lib/cdi-api-1.1.jar:lib/classmat
e-1.3.0.jar:lib/commons-dbcp-1.4.jar:lib/commons-logging-1.2.j
ar:lib/commons-pool-1.5.4.jar:lib/dom4j-1.6.1.jar:lib/el-api-2
.2.jar:lib/geronimo-jta_1.-1.1.1.jar:lib/hibernate-commons-ann
otations-5.0.1.Final.jar:lib/hibernate-core-5.2.2.Final.jar:li
b/hibernate-jpa-2.1-api-1.0.0.Final.jar:lib/hsqldb-2.3.4.jar:l
ib/jandex-2.0.0.Final.jar:lib/javassist-3.20.0-GA.jar:lib/java
x.inject-1.jar:lib/jboss-interceptors-api_1.1_spec-1.0.0.Beta1
.jar:lib/jboss-logging-3.3.0.Final.jar:lib/jcl-over-slf4j-1.7.
21.jar:lib/jsr250-api-1.0.jar:lib/log4j-api-2.6.2.jar:lib/log4
j-core-2.6.2.jar:lib/slf4j-api-1.7.21.jar:lib/slf4j-simple-1.7
.21.jar:lib/spring-aop-4.3.2.RELEASE.jar:lib/spring-beans-4.3.
2.RELEASE.jar:lib/spring-context-4.3.2.RELEASE.jar:lib/spring-
core-4.3.2.RELEASE.jar:lib/spring-expression-4.3.2.RELEASE.jar
:lib/spring-jdbc-4.3.2.RELEASE.jar:lib/spring-orm-4.3.2.RELEAS
E.jar:lib/spring-tx-4.3.2.RELEASE.jar
```

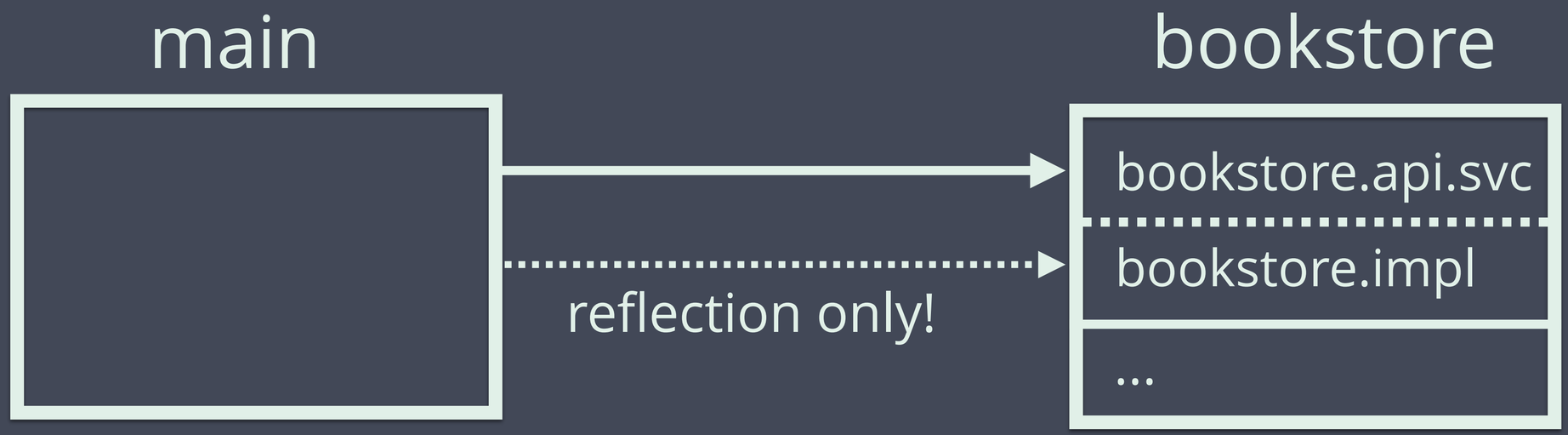
Java 9 modules make this possible!



A Module Primer

```
module main {
  requires bookstore;
}
```

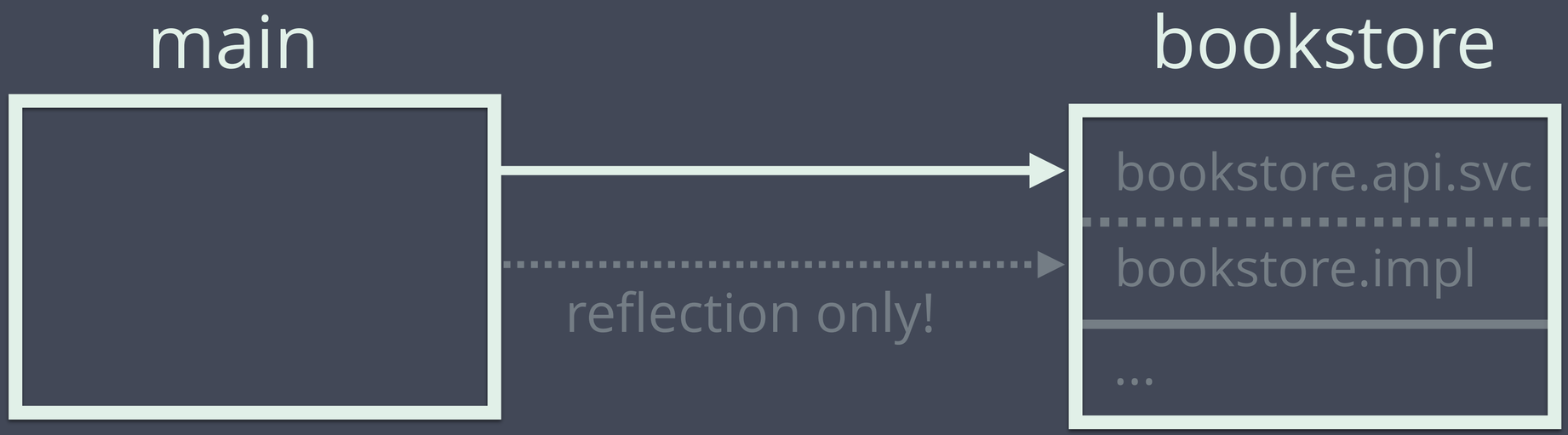
```
module bookstore {
  exports bookstore.api.svc;
  opens bookstore.impl;
}
```



A Module Primer

```
module main {
  requires bookstore;
}
```

Modules define dependencies explicitly

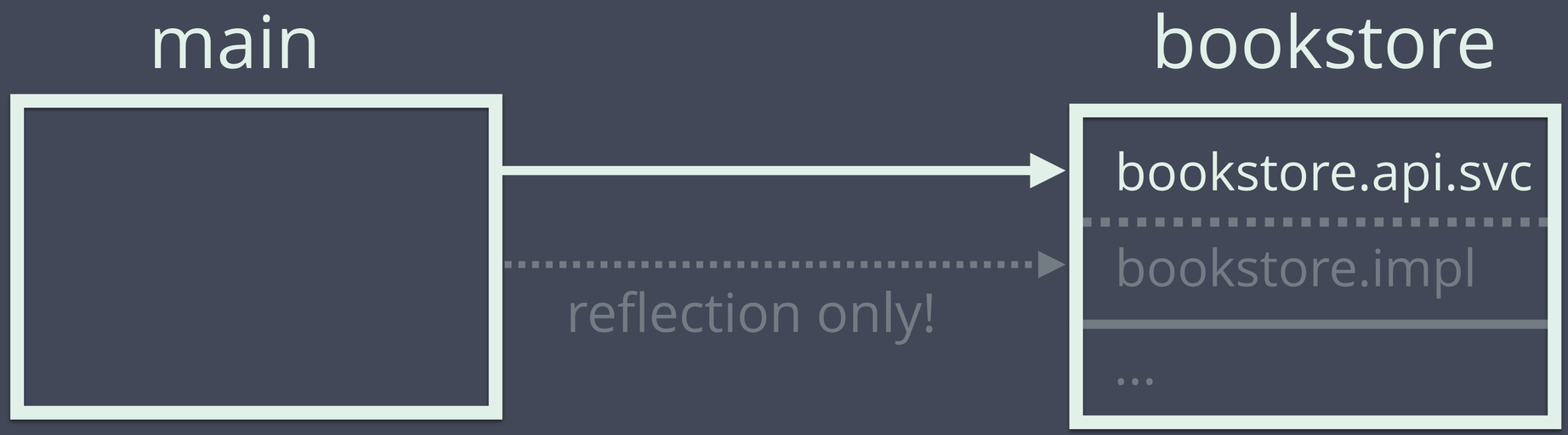


A Module Primer

```
module m
  require
}
```

Packages are encapsulated by default

```
module bookstore {
  exports bookstore.api.svc;
  opens bookstore.impl;
}
```

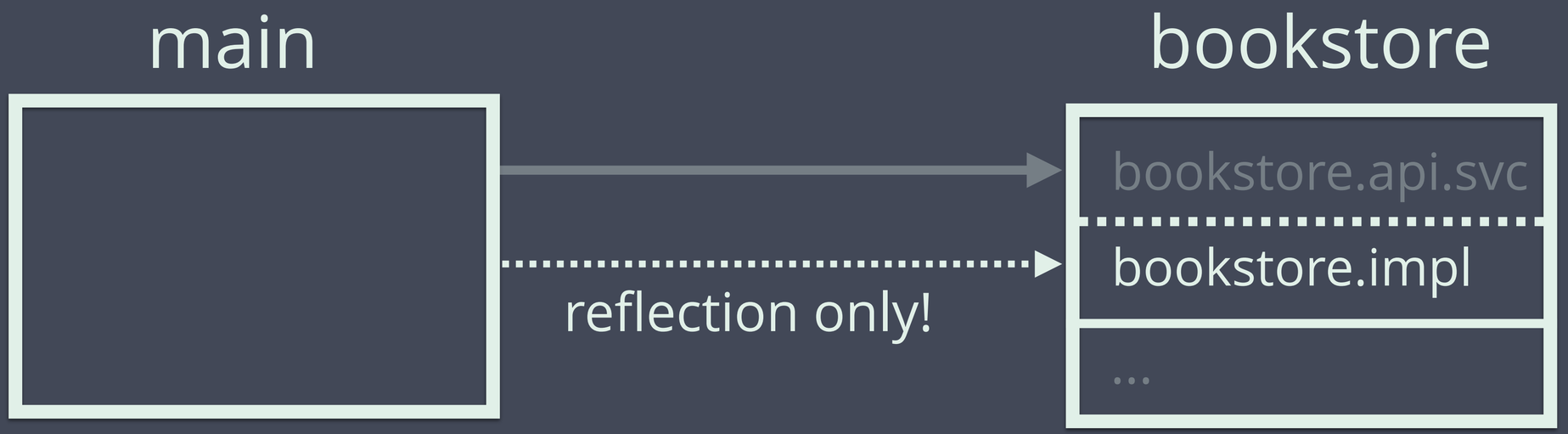


A Module Primer

```
module main {
  require
}
```

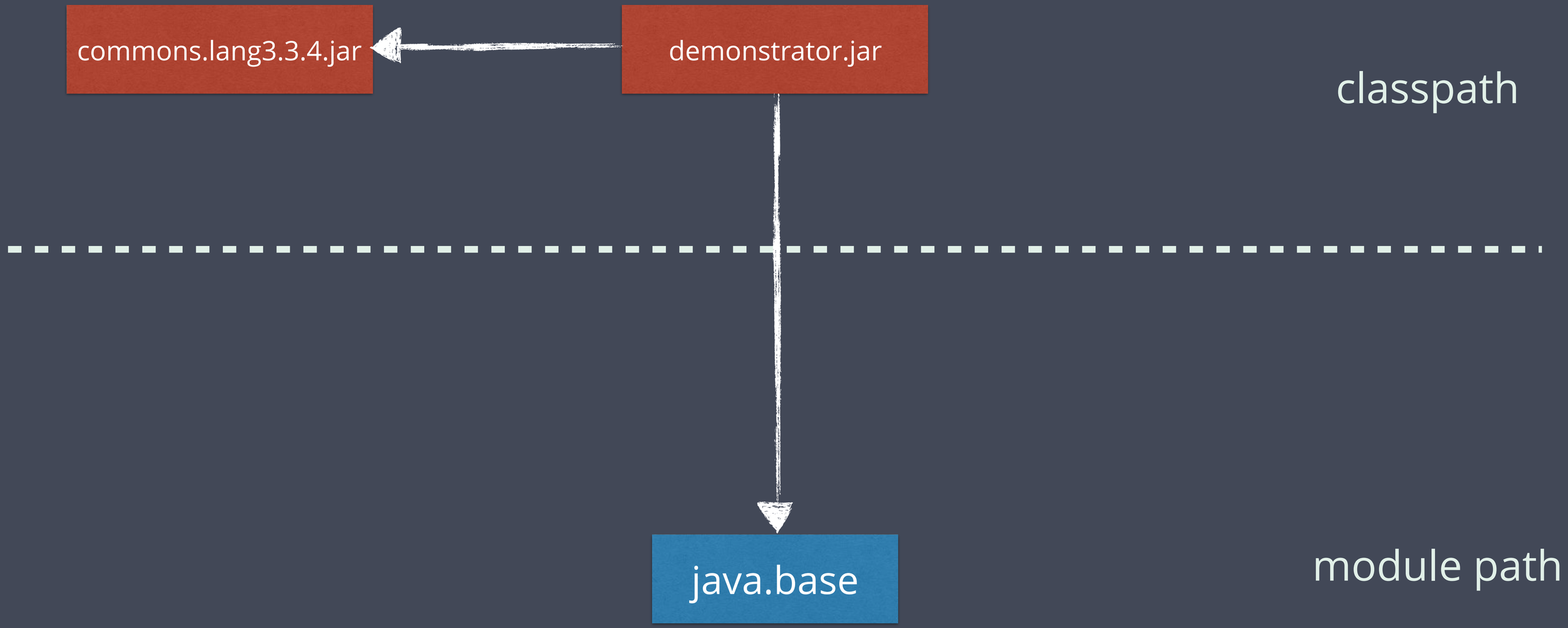
Packages can be “opened” for deep reflection at run-time

```
module bookstore {
  exports bookstore.api.svc;
  opens bookstore.impl;
}
```



Migrating to modules

Top down migration



Classic classpath

```
package com.javamodularity.demonstrator;

import org.apache.commons.lang3.StringUtils;

public class Demo {

  public static void main(String args[]) {
      String output = StringUtils.leftPad("Leftpad FTW!", 20);
      System.out.println(output);
  }
}
```

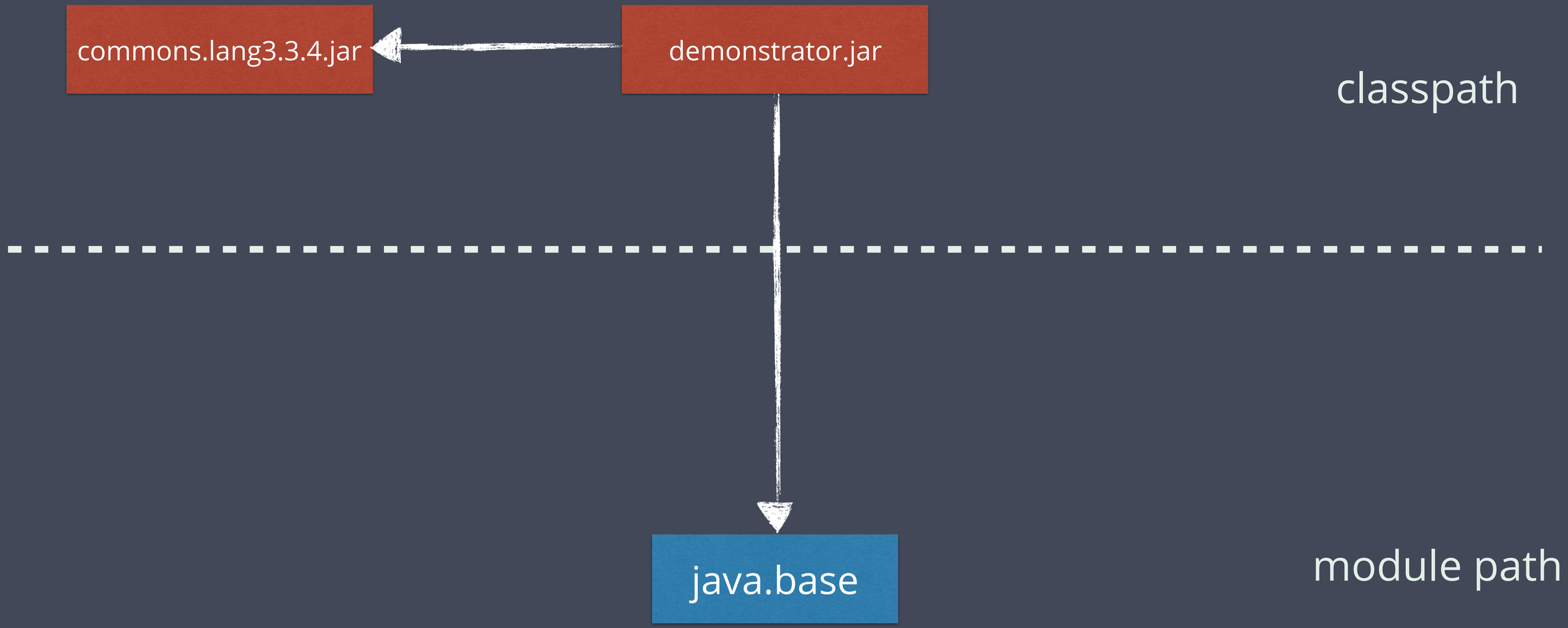
Compile

```
javac -cp lib/commons-lang3-3.4.jar
      -d out $(find src -name '*.java')
```

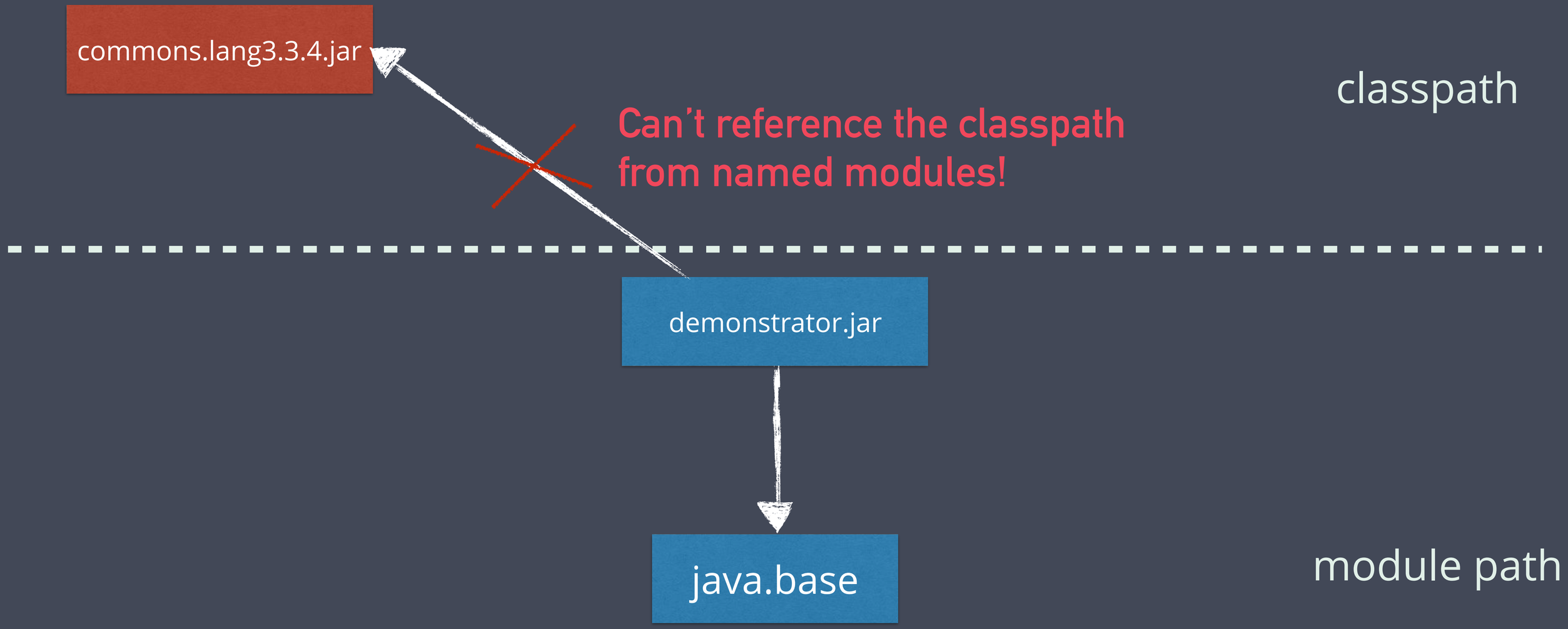
Run

```
java -cp out:lib/commons-lang3-3.4.jar
      com.javamodularity.demonstrator.Demo
```

Top down migration

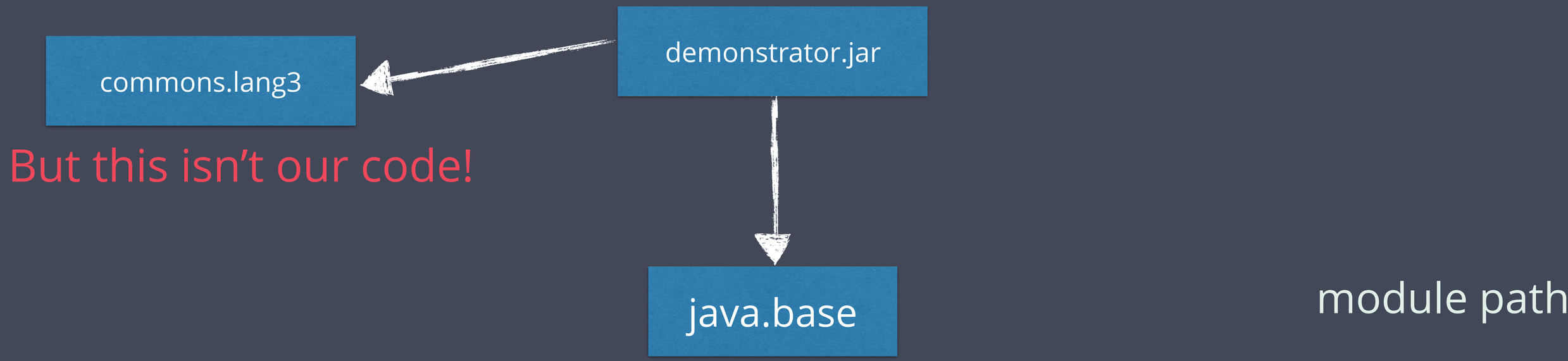


Top down migration



Top down migration

classpath



But this isn’t our code!

module path

Automatic Modules

- A plain JAR on the module path becomes an **Automatic Module**
- Module name derived from JAR name (or Automatic-Module-Name in manifest)
- Exports everything
- Reads all other modules **and** the classpath

Modularize **your code** without waiting on libraries

Using Automatic Modules

```
module demonstrator {
    requires commons.lang3;
}
```

Compile

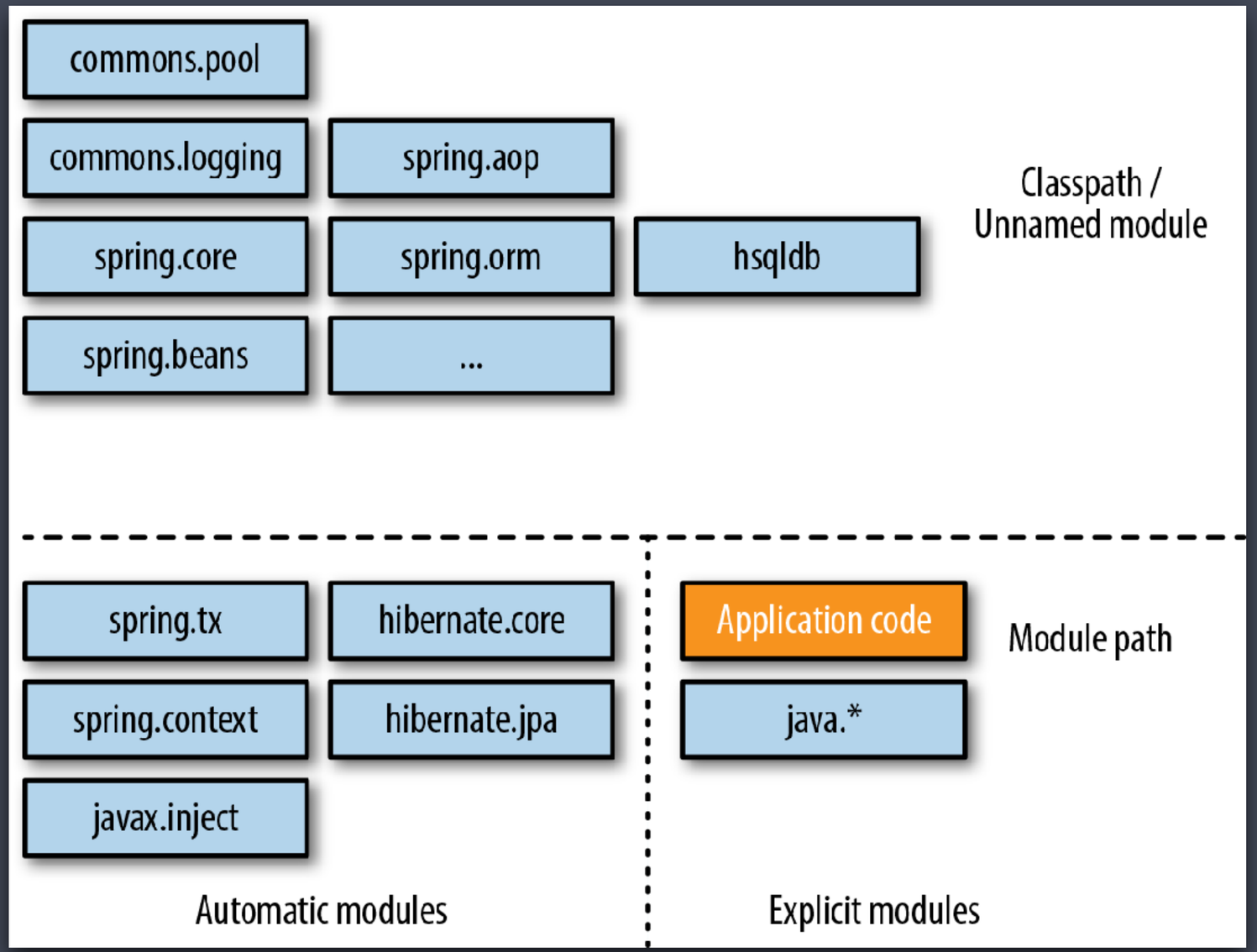
```
javac --module-path lib
      --module-source-path src
      -d mods $(find src -name '*.java')
```

Run

```
java --module-path mods:lib
     -m demonstrator/com.javamodularity.demonstrator.Demo
```

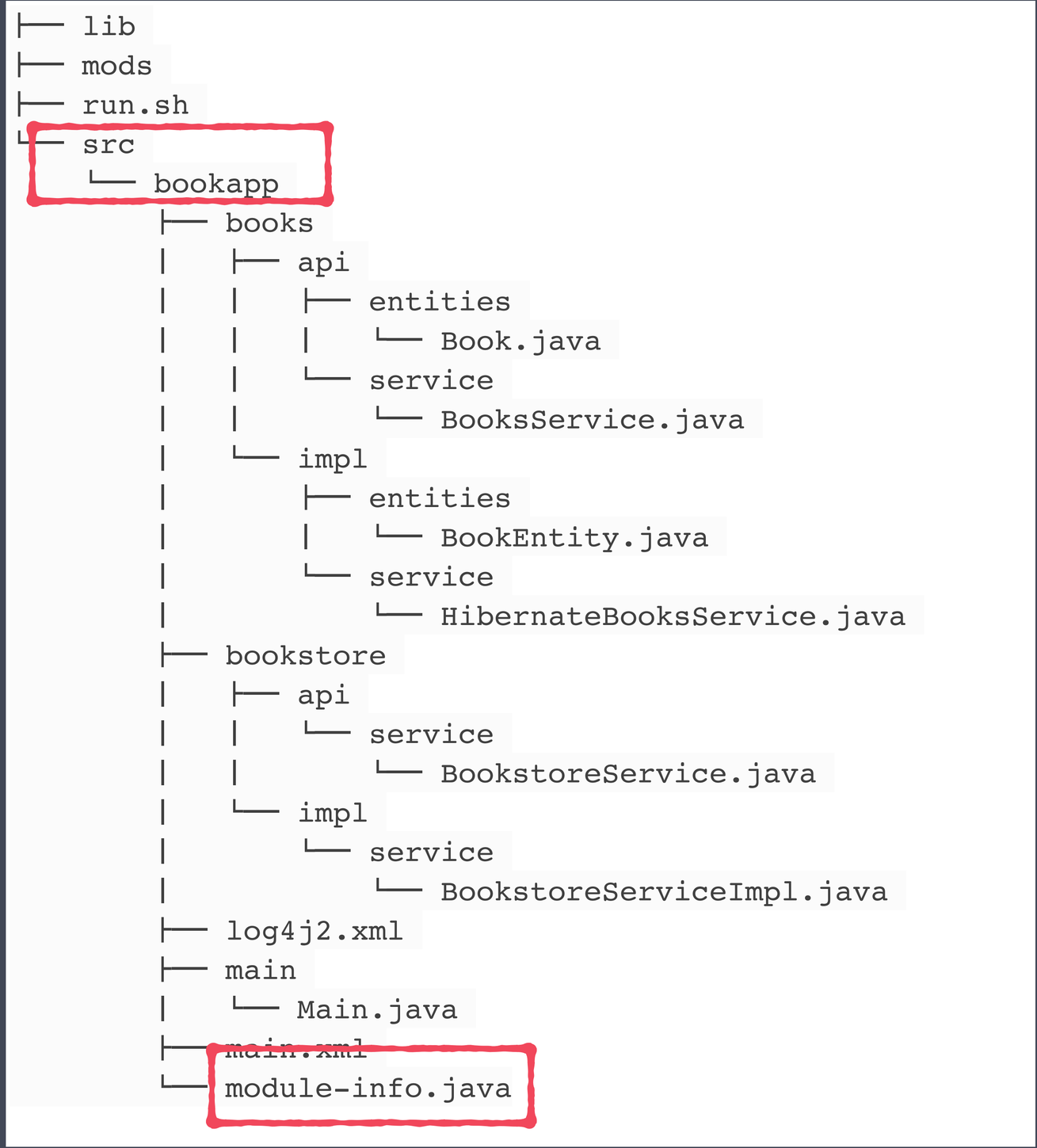
Migrating the Spring app

The (Intermediate) Goal



Step 1

- module-info.java
- compile with --module-source-path



Step 2

- Compensate for Hibernate being an automatic module
- Hibernate *should* `'requires transitive java.naming'`

```
module bookapp {

    requires spring.context;
    requires spring.tx;
    requires javax.inject;
    requires hibernate.core;
    requires hibernate.jpa;

}
```

Not an issue when Hibernate ships as **explicit modules**

Step 3

- Compensate for Spring being an automatic module
- Spring *should* require `java.sql` and `java.xml.bind`

```
module bookapp {

    requires spring.context;
    requires spring.tx;
    requires javax.inject;
    requires hibernate.core;
    requires hibernate.jpa;

}
```

Not an issue when Spring ships as **explicit modules**

Step 4

- Spring/Hibernate use reflection to load our code
- Default encapsulation breaks our application

This is something you'll **always** need to address!

Step 4: Open Modules/Packages

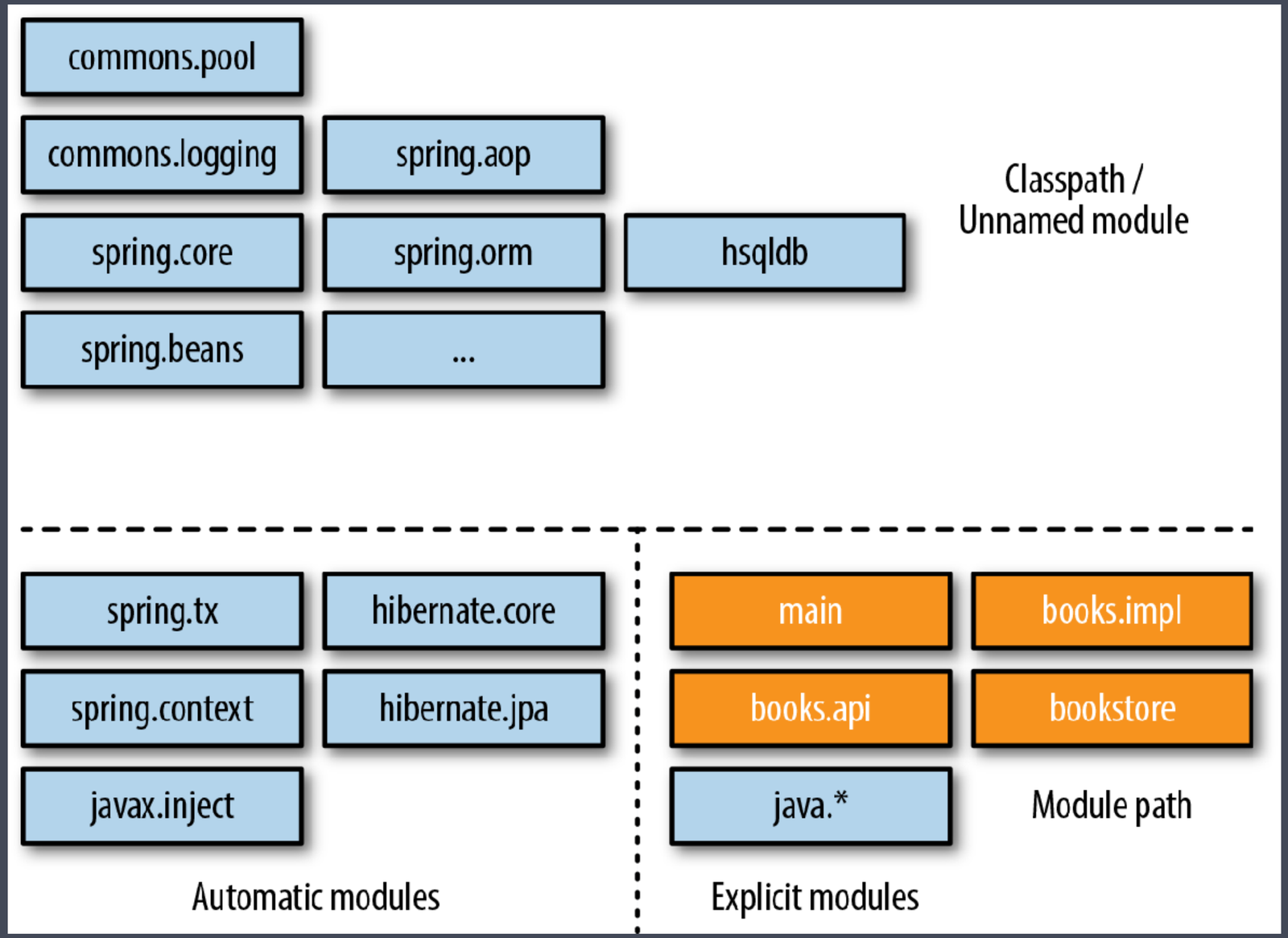
- Open package allows **deep** reflection at run-time
  - No compile-time dependency possible
  - Ideal for frameworks like Spring/Hibernate
- Open module is a module where all packages are opened

Step 4: Open Modules/Packages

Type	Compile time	Reflection on public	Deep reflection
Exports	✓	✓	✗
Open	✗	✓	✓
Exports + Open	✓	✓	✓

Migrating the Spring app

The End Goal



Modules: Linking

- Use a linking tool (jlink) to create a custom 'runtime image' with only the modules you need
- Uses explicit dependencies from module-info.class
- Allows for whole-program optimization

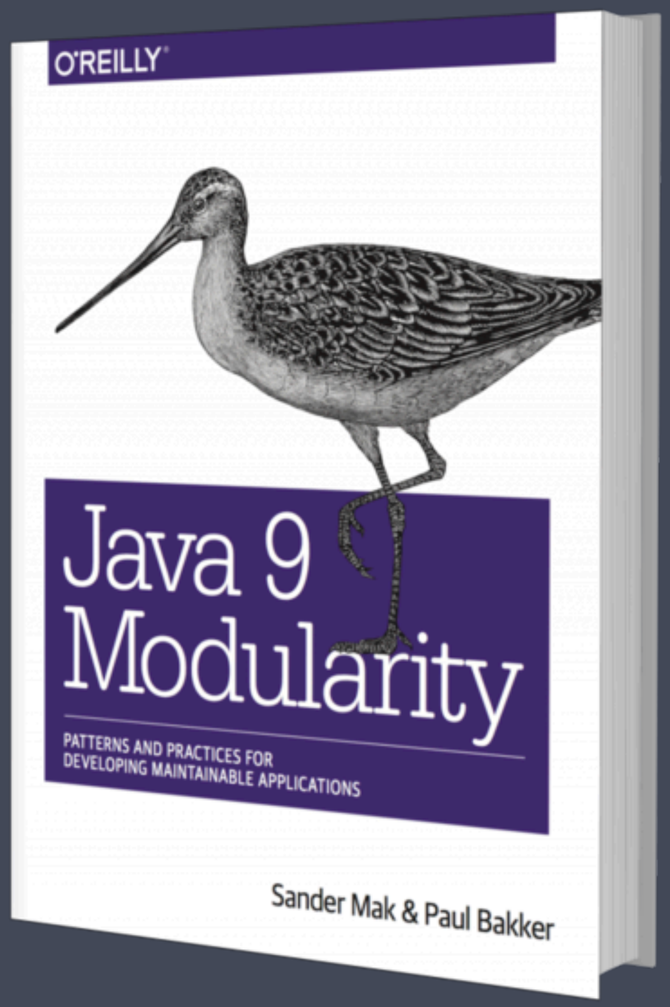
Caveat: **jlink** doesn't accept **automatic modules**

Migration Steps - Recap

- Migrate to Java 9 using classpath only (run with `--illegal-acces=deny`)
- Create a module around your whole app
- Modularize your application!

Urge library maintainers to produce Java 9 modules

Should I adopt Java 9 or wait?



Thank you. luminis

Conversing worlds

PLURALSIGHT

javamodularity.com

bit.ly/sander-ps