Preparing for Java 9

Sander Mak

@sander_mak www.branchandbound.net

Migrating without modules

Automatic modules

Course wrap-up

What about the classpath?

The classpath is still alive!

Modules are **optional**

Migrating a Classpath Based Application

Java 8

```
javac -cp $CLASSPATH ...

java -cp $CLASSPATH ...
```

Java 9

```
javac -cp $CLASSPATH ...

java -cp $CLASSPATH ...
```

Can it be this easy?

Yes!

Yes, unless ...

Migrating a Classpath Based Application

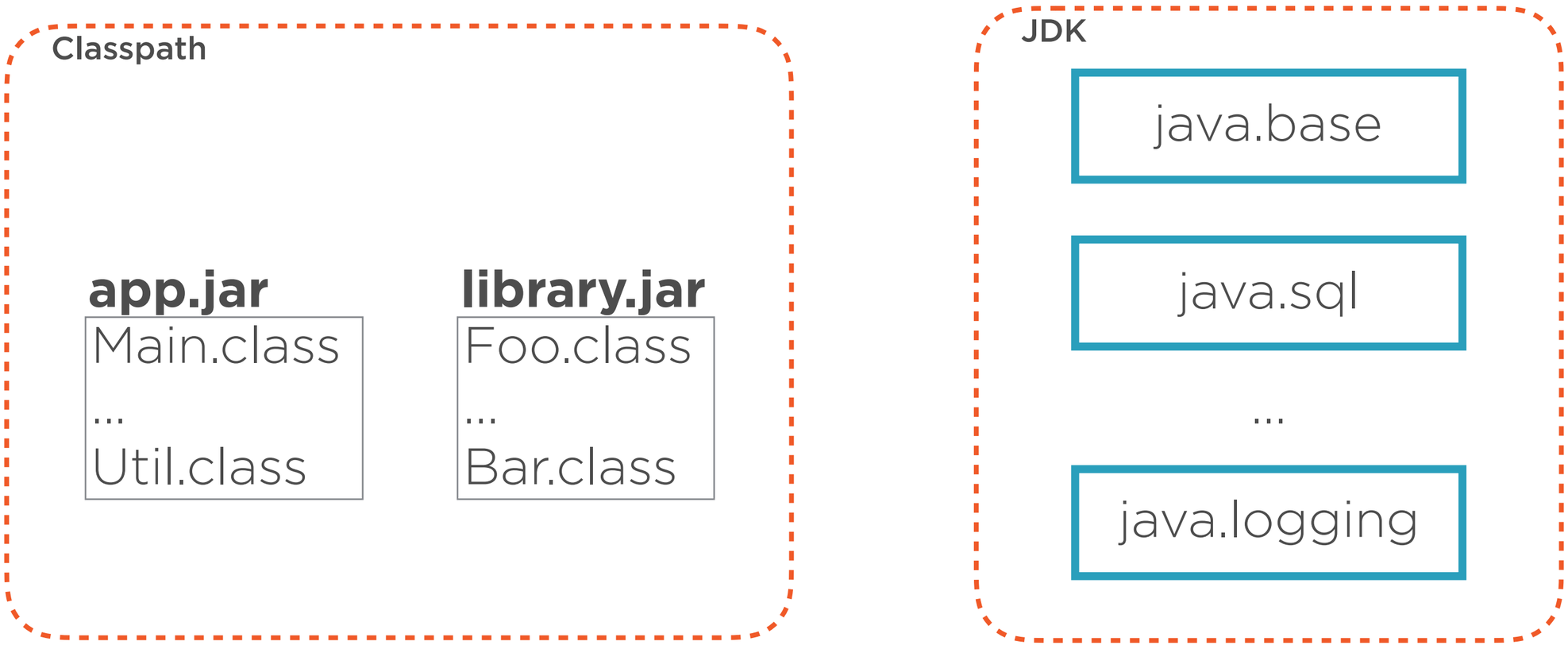
Unless ...

1. You use JDK types that have been encapsulated

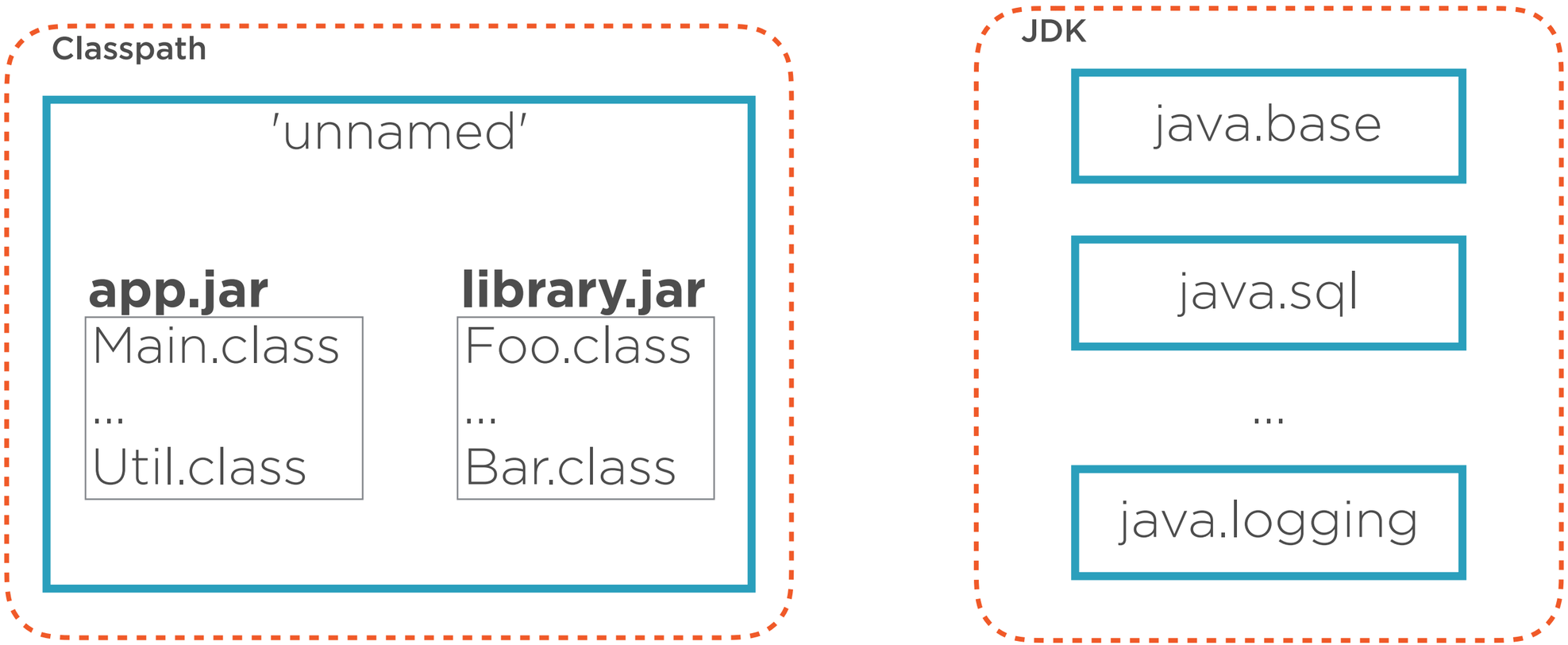
2. You use types from non-default Java SE modules

Using Encapsulated Types

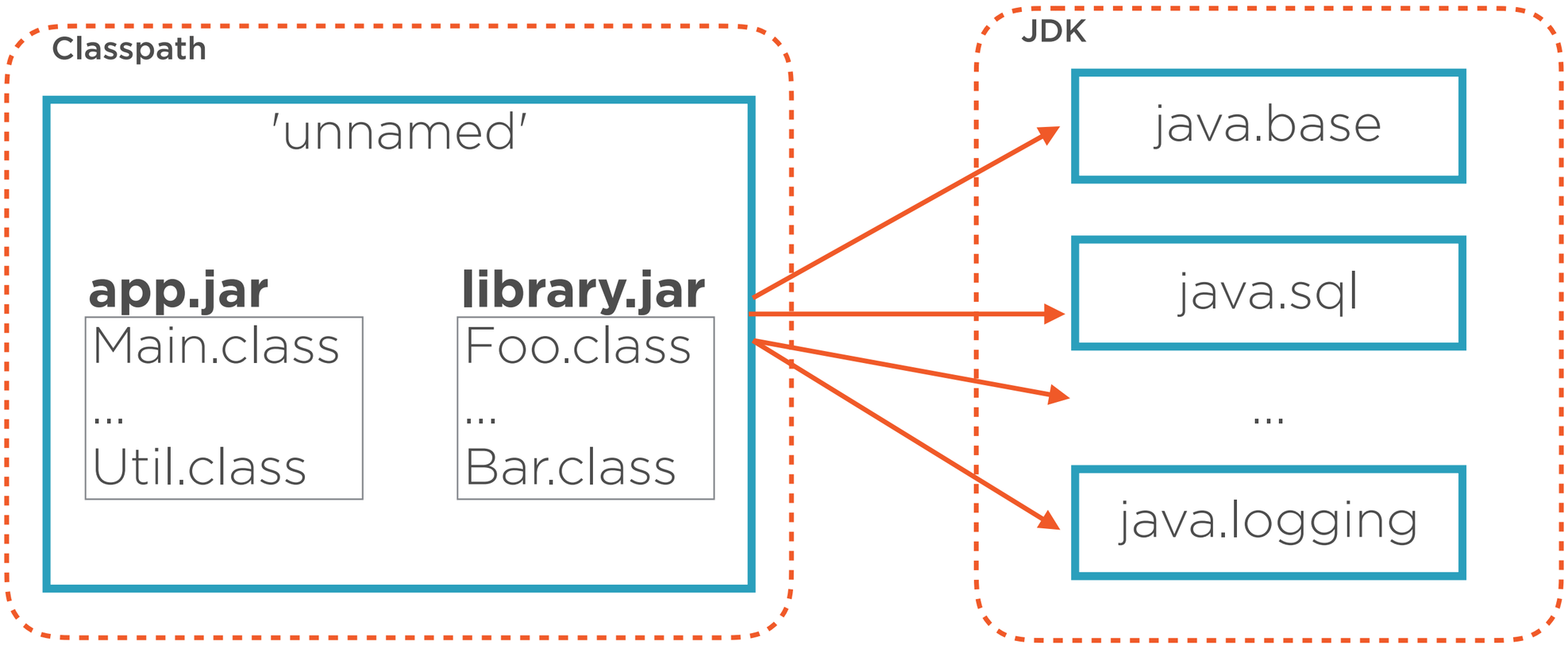
Using Encapsulated Types



Using Encapsulated Types

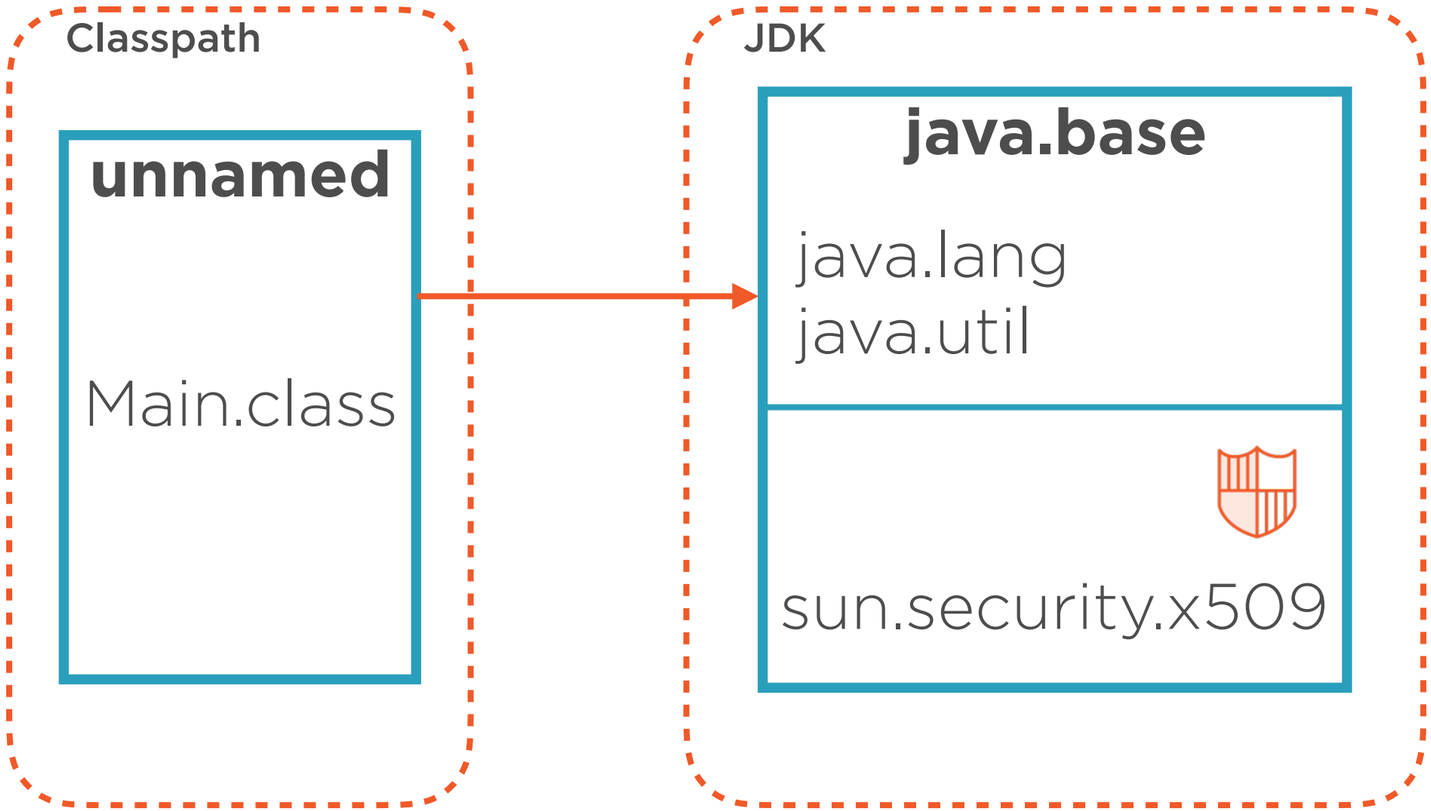


Using Encapsulated Types



Demo

Using encapsulated types



Use jdeps to detect these issues

Using Encapsulated Types

What if I can't change the code?

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

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

Migrating a Classpath Based Application

Unless ...

1. You use JDK types that have been encapsulated

2. You use types from non-default Java SE modules

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

public class Main {

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

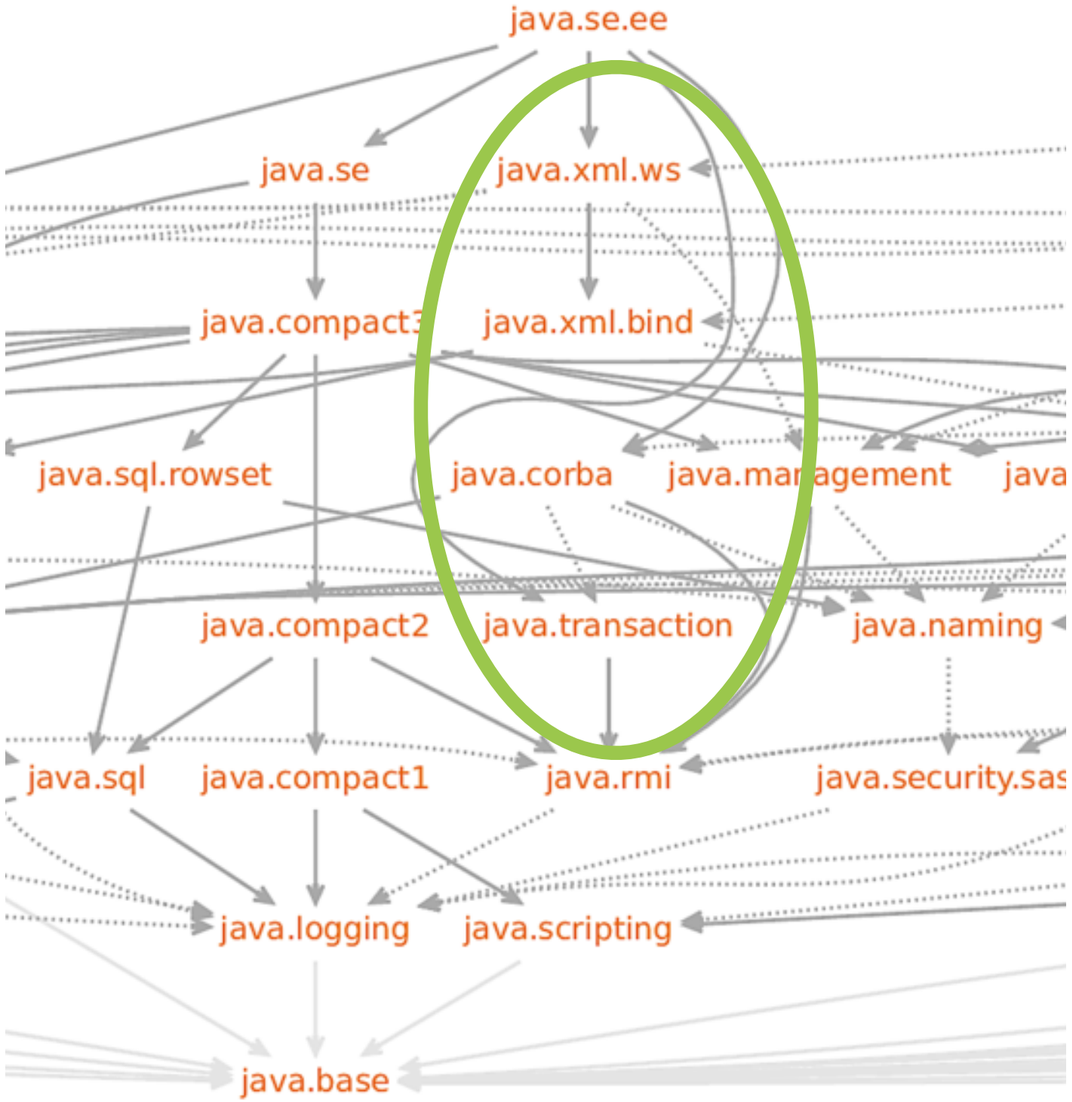
$ javac src/Main.java

src/Main.java:1: error: package javax.xml.bind does not exist

src/Main.java:6: error: cannot find symbol: variable DatatypeConverter

2 errors

Using Non-default Modules



- java.transaction
- java.annotations.common
- java.xml.bind
- java.xml.ws
- java.corba
- java.activation

Using Non-default Modules

```
javac --add-modules java.xml.bind ...
```

```
java --add-modules java.xml.bind ...
```

```
java --add-modules java.se.ee ...
```

Using Non-default Modules

Find usage with jdeps

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

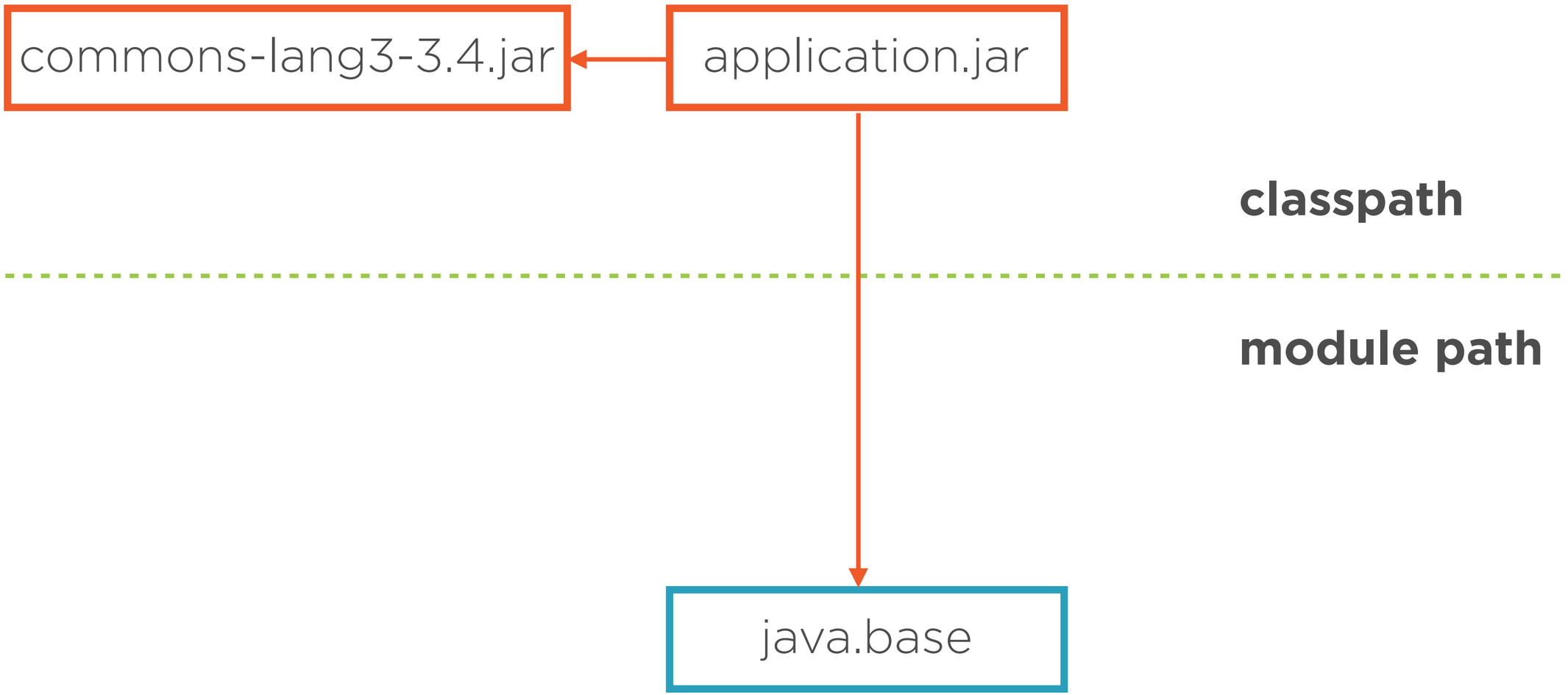
Modularizing your own code

What if my dependencies aren't modules yet?

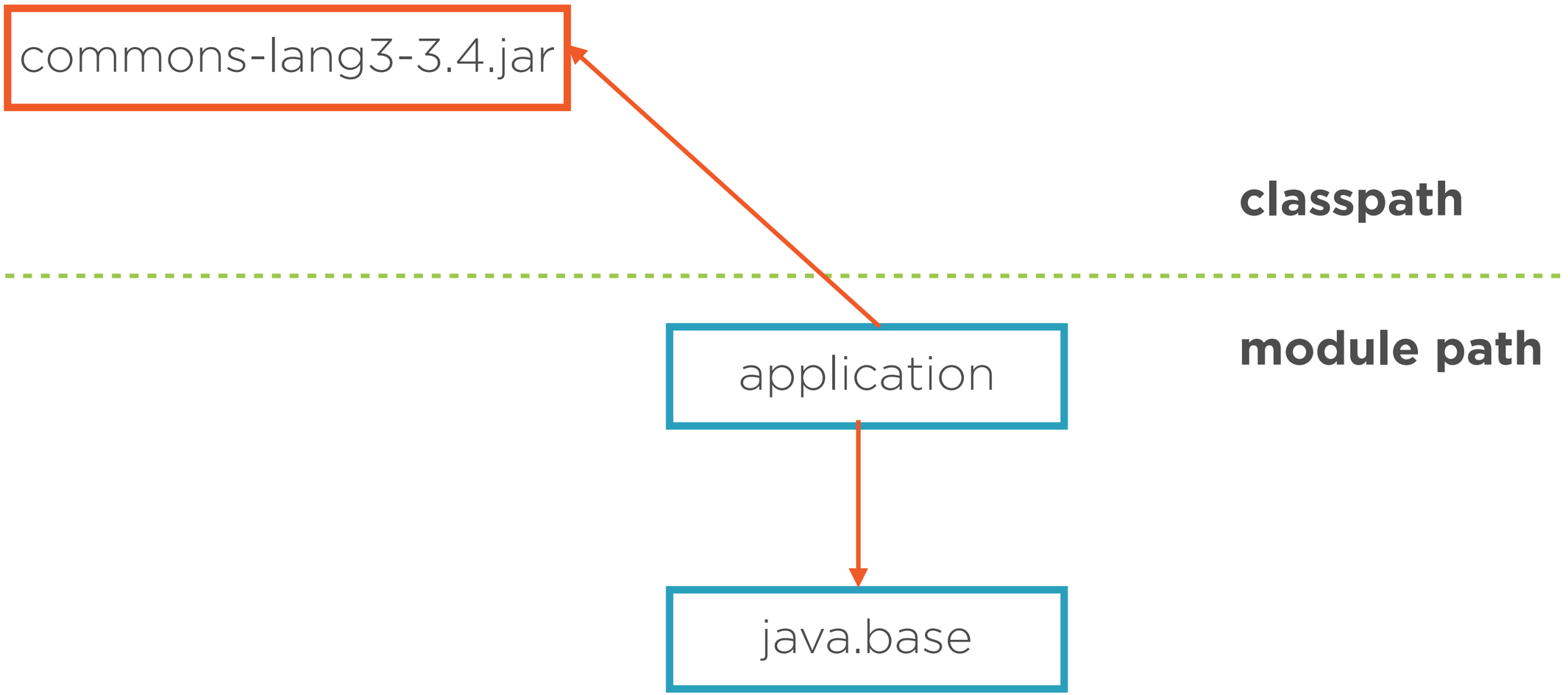
```
package javamodularity.automaticmods;

import org.apache.commons.lang3.StringUtils;

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

javac -cp lib/commons-lang3-3.4.jar -d out ...

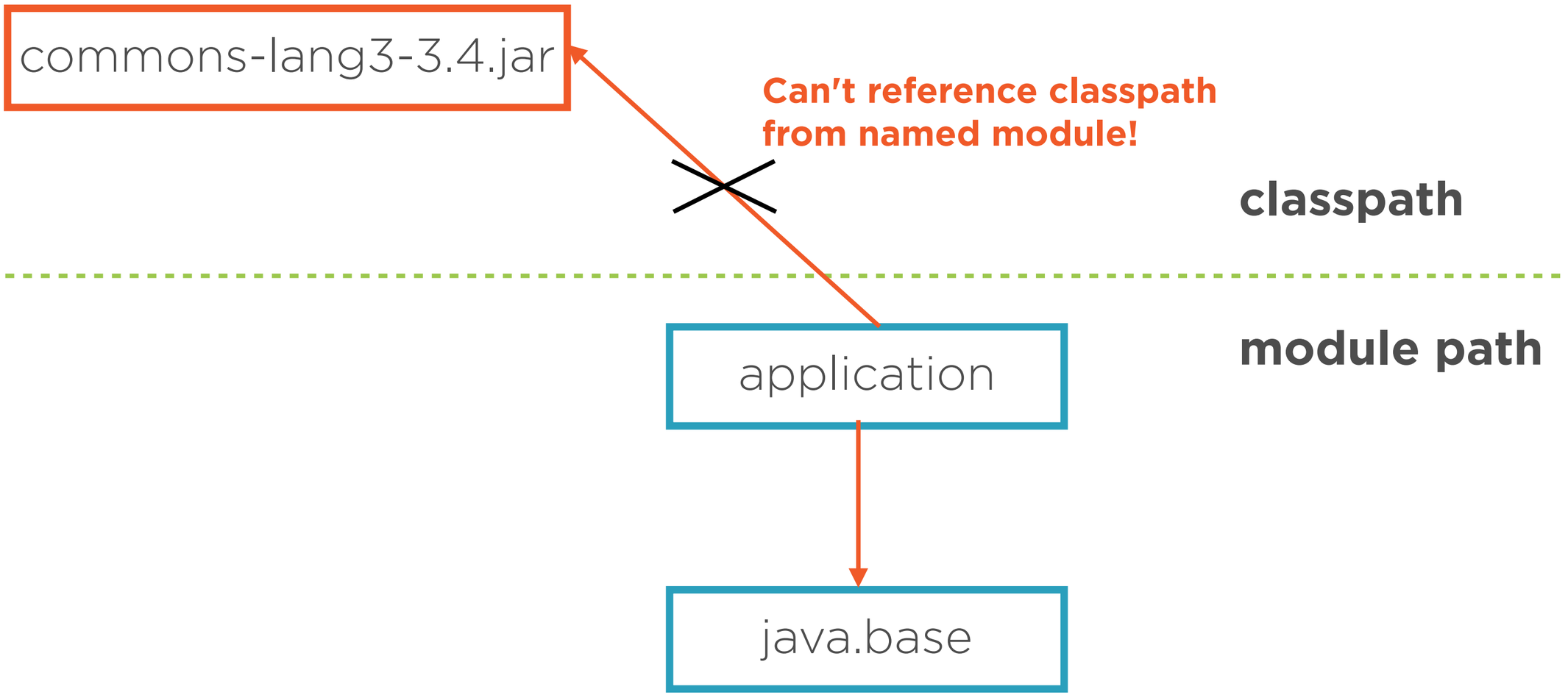
Automatic Modules



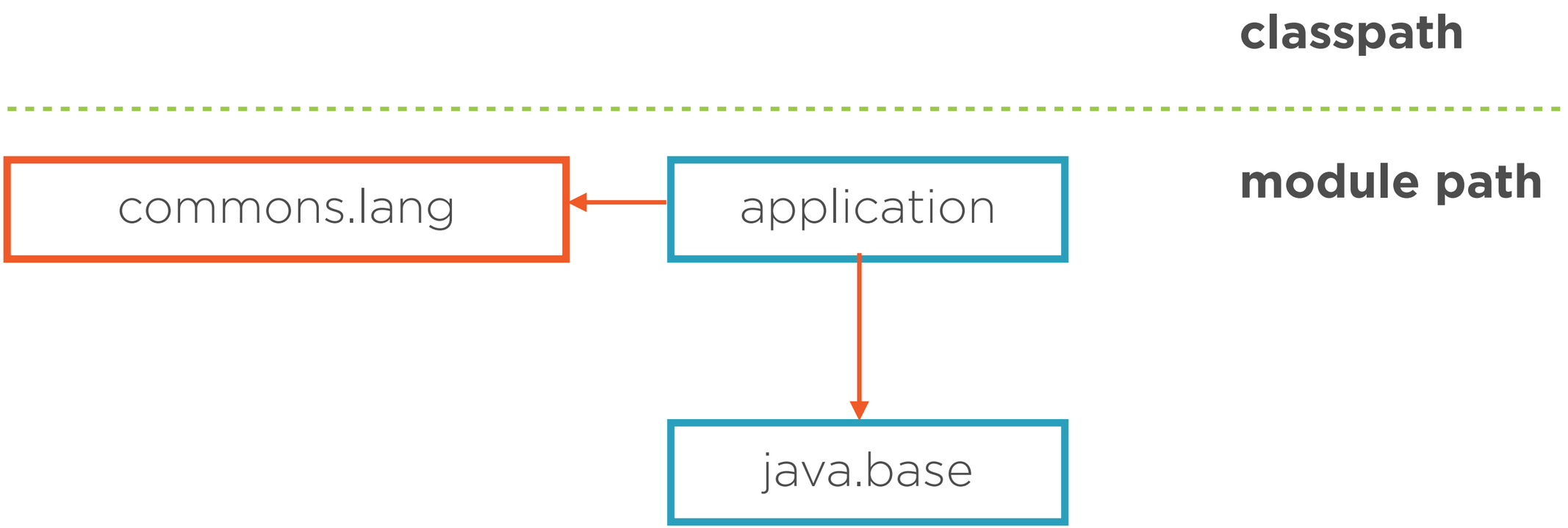
Automatic Modules



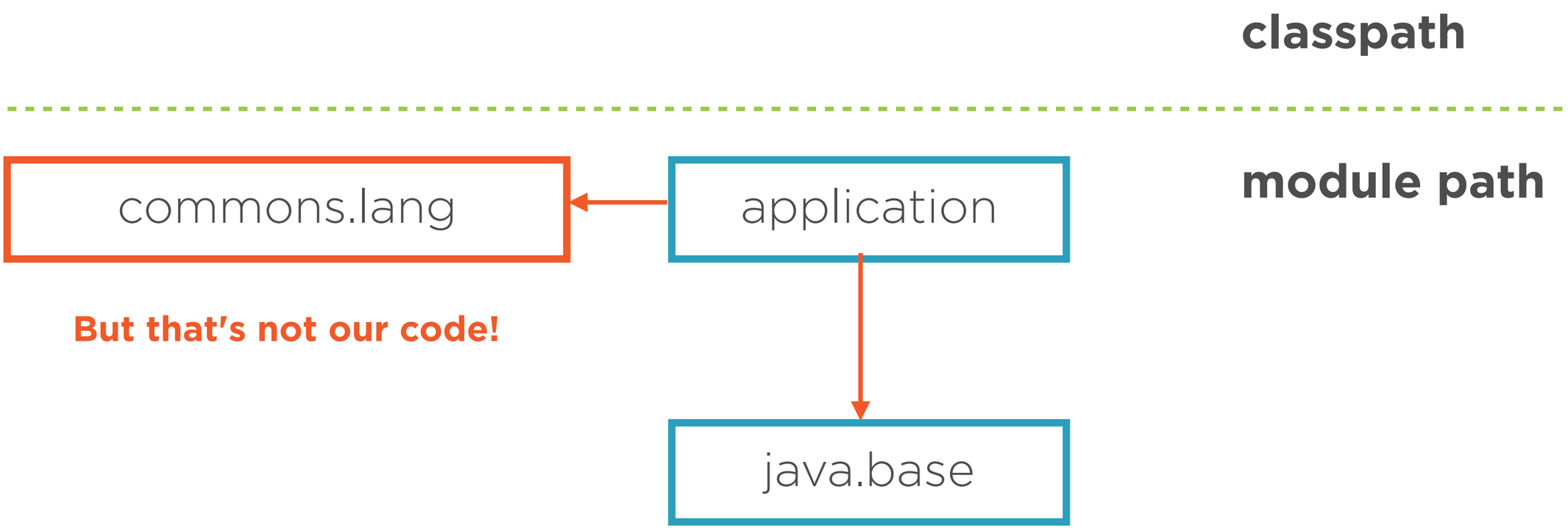
Automatic Modules



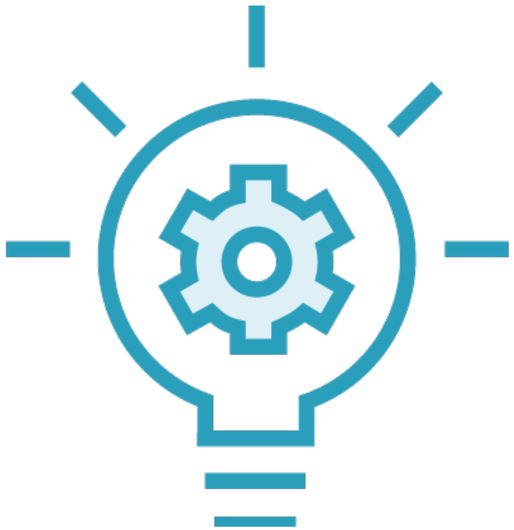
Automatic Modules



Automatic Modules



Course Wrap-up: Migration



Add non-modular JAR to module path

Name derived from JAR filename

Exports all packages

Reads all other modules

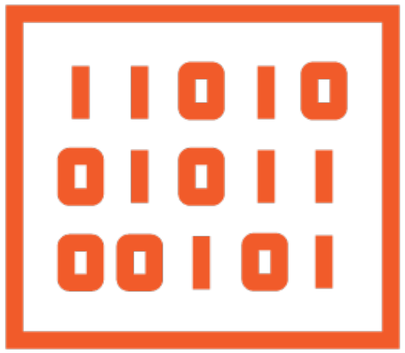
```
module application {

   requires commons.lang;

}
```

javac --module-path lib --module-source-path ...

java --module-path lib:out -m application/...

Course Wrap-up

Course Wrap-up: Three Tenets of Modularity



Strong Encapsulation

Hide your internals, be strict about what is public API

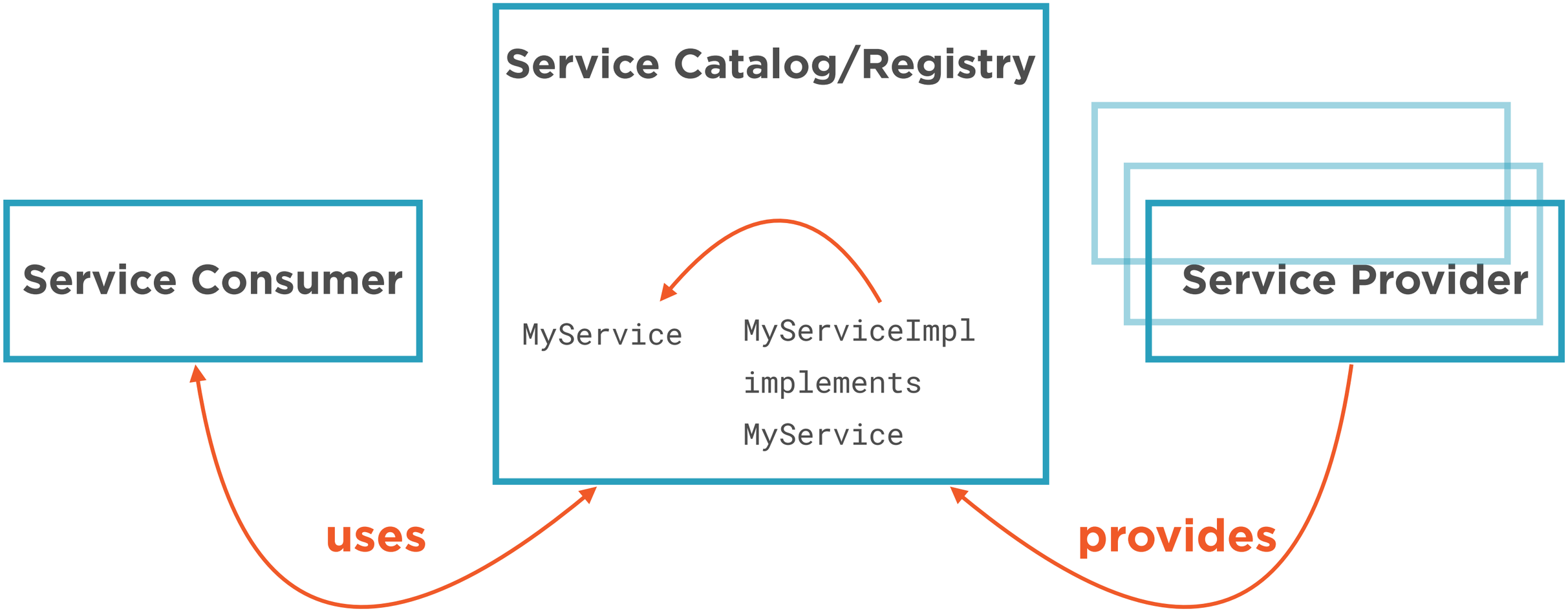
Well-defined Interfaces

When modules interact, use stable and well-defined interfaces

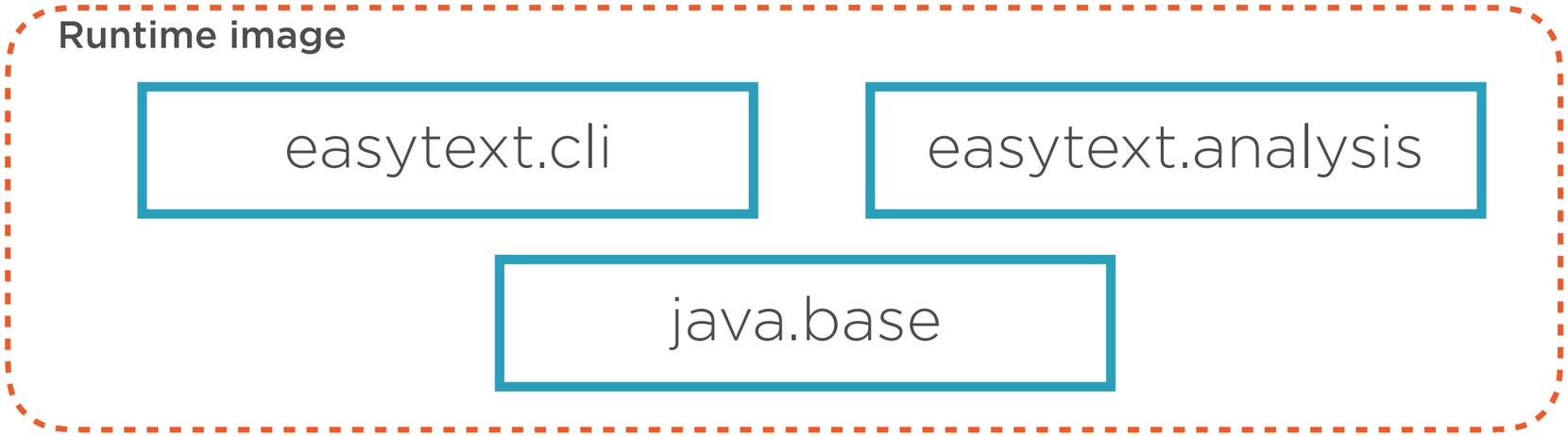
Explicit Dependencies

A module lists what it needs from other modules

Course Wrap-up: Services



Course Wrap-up: Linking



Stand-alone custom run-time image

Small footprint

Link-time optimizations

Course Wrap-up: Migration

The classpath is still alive!

With some cases to be aware of...

Use jdeps to find problems early

Automatic modules enable seamless migration

Next Steps

https://jdk9.java.net/jigsaw/

Java 9 Modularity (O'Reilly)

bit.ly/java9book

Design modular applications!