Migrating to Modules

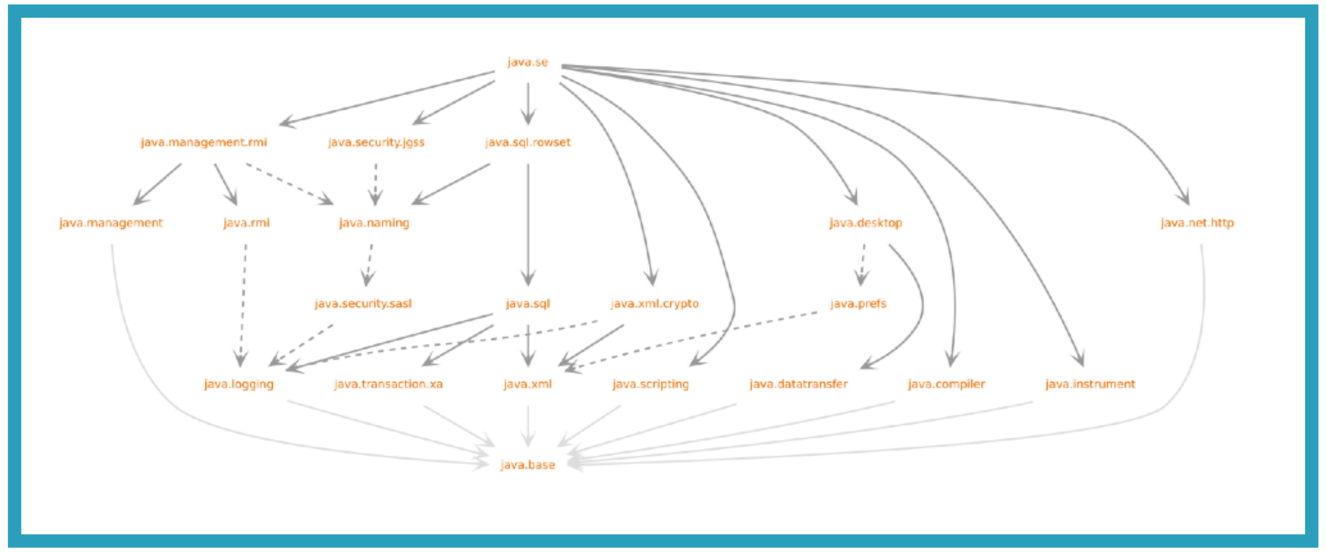
Sander Mak
Java Champion

@Sander_Mak   www.javamodularity.com

The Classpath in a Modular World

The Classpath in a Modular World

Modular JDK / Module path

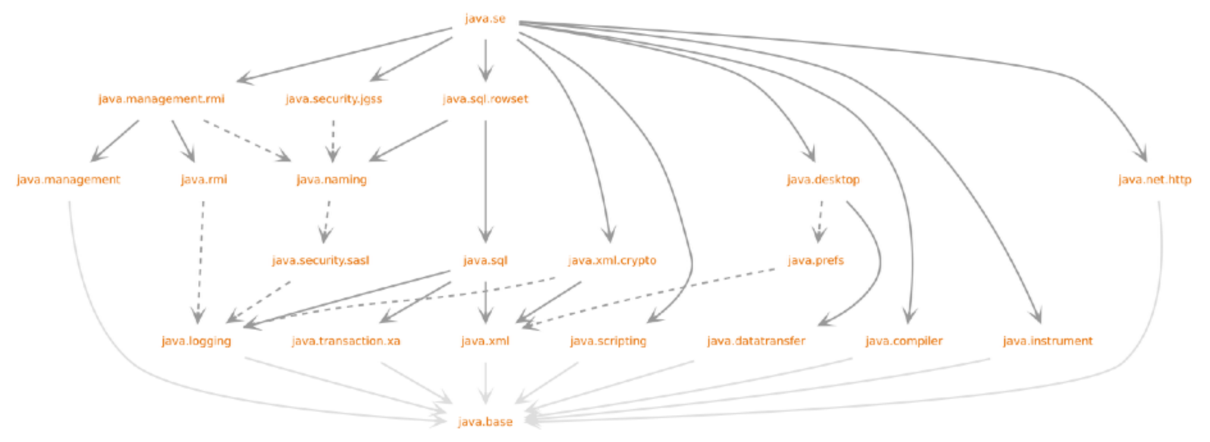


The Classpath in a Modular World

Classpath

```
my-application.jar
```

Modular JDK / Module path

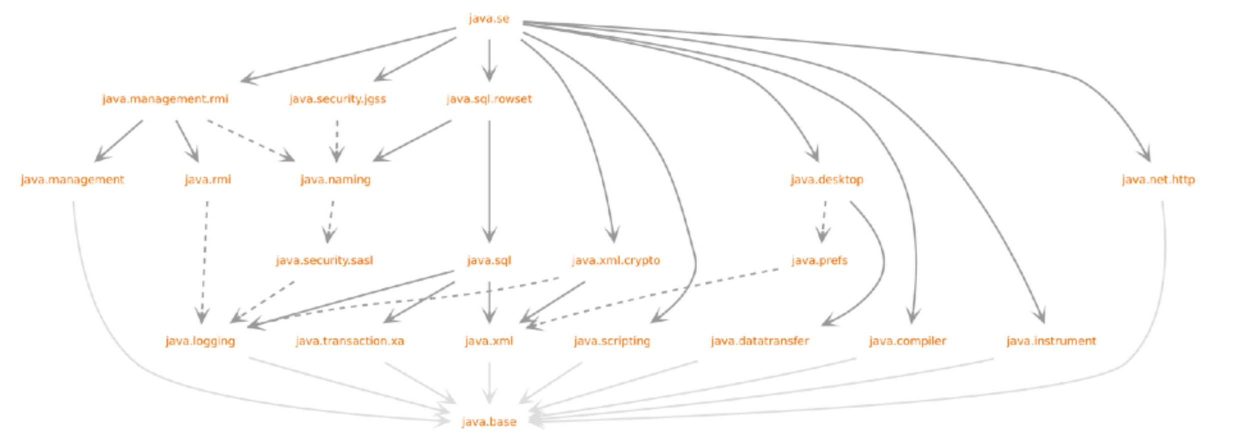


The Classpath in a Modular World

Classpath

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path

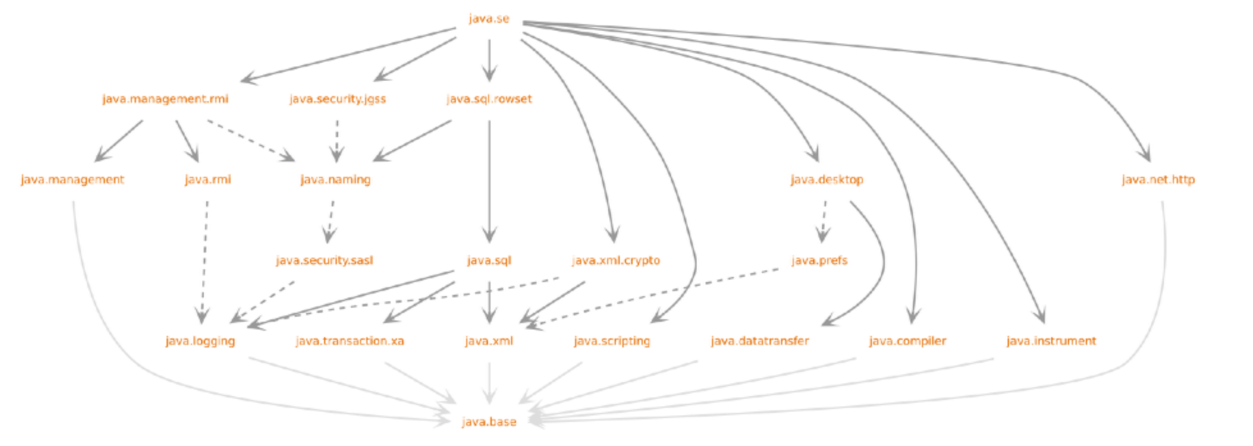


The Classpath in a Modular World

Classpath

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path



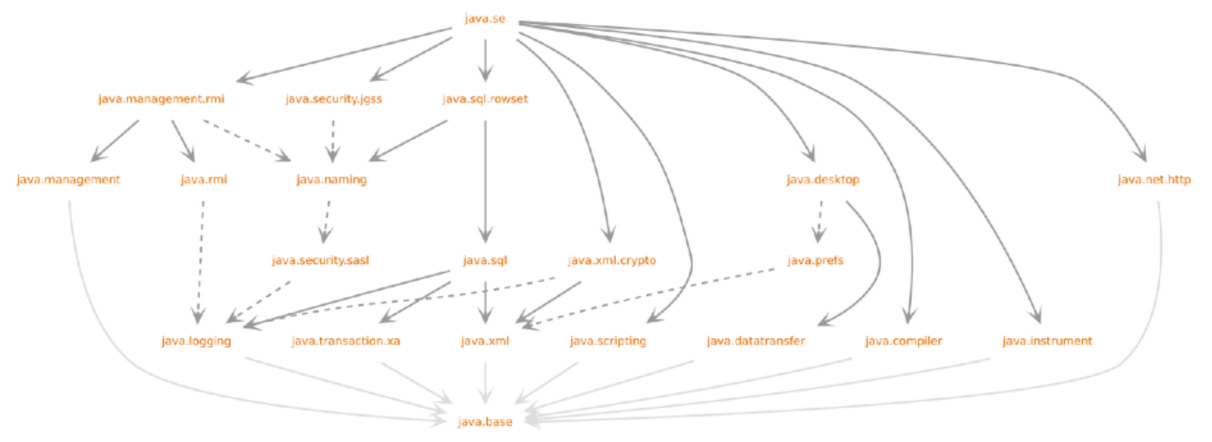
When your application and dependencies only use official Java APIs, *things should just work™*

The Classpath in a Modular World

Classpath

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path

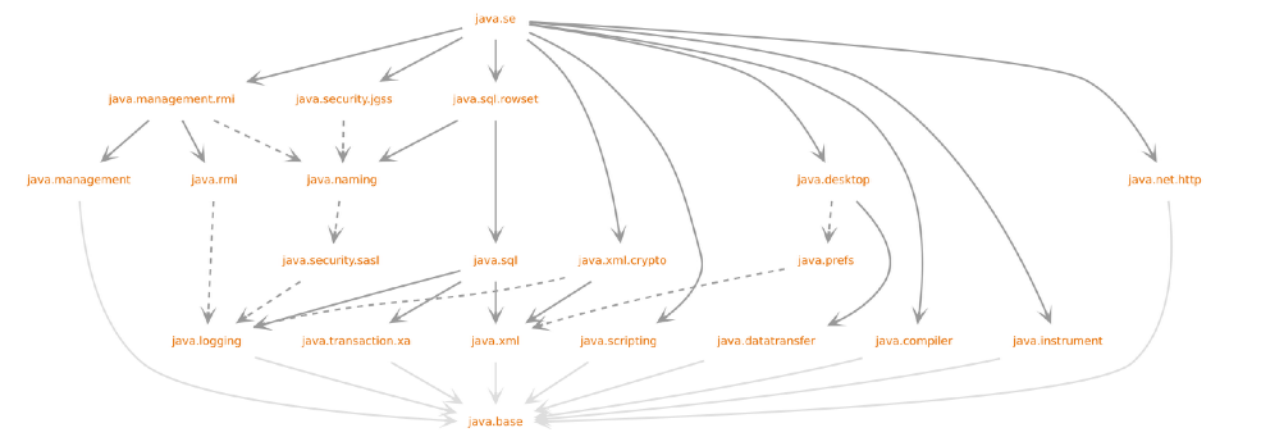


The Classpath in a Modular World

~~Classpath~~ Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path

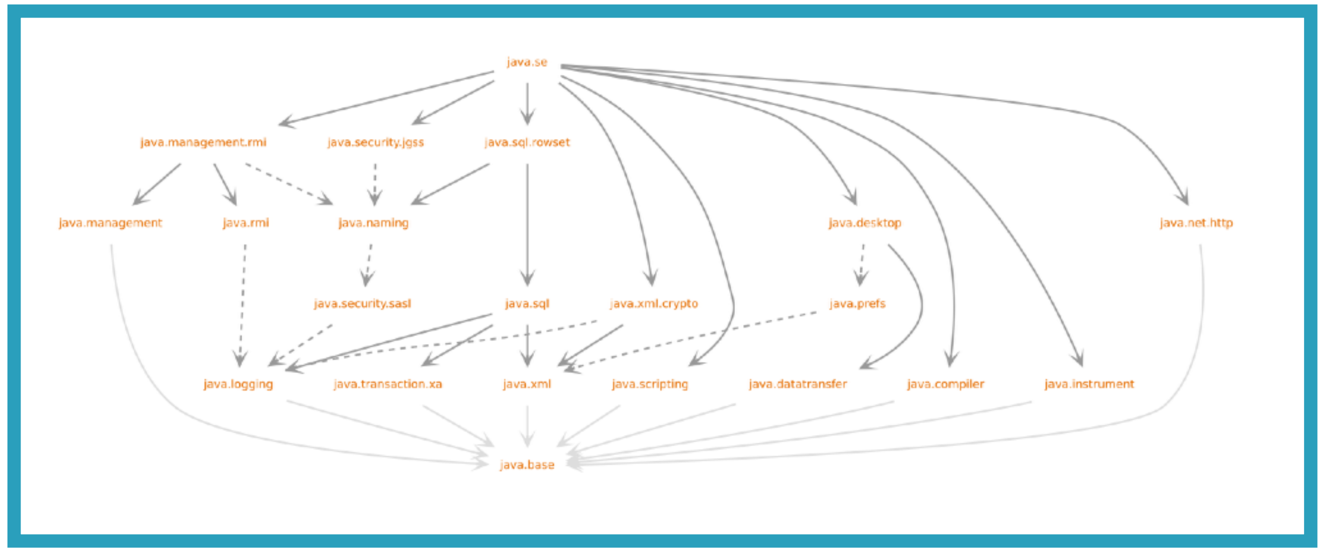


The Classpath in a Modular World

~~Classpath~~ Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path



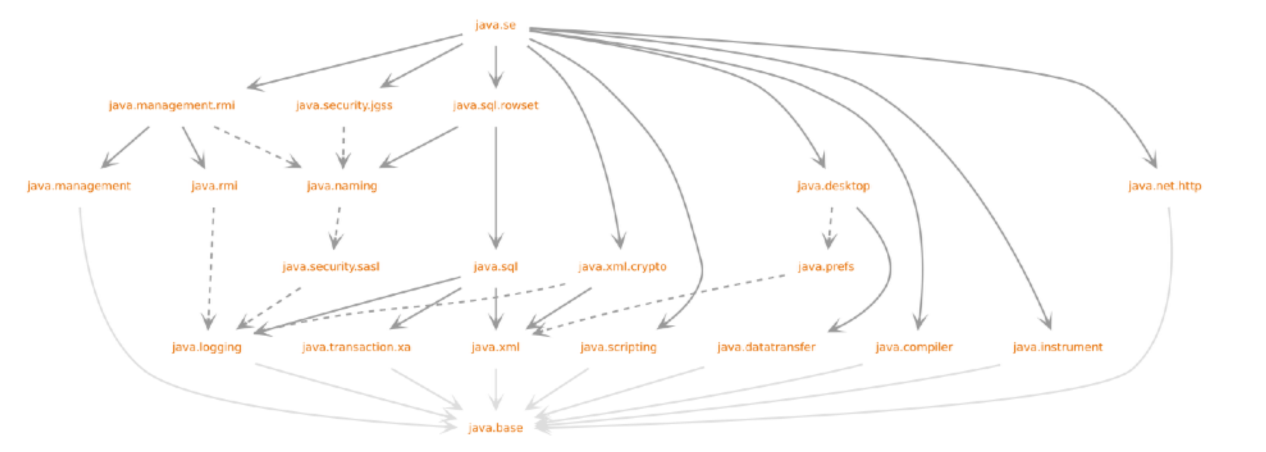
The unnamed module can read **everything** from the modular JDK **at run-time**

The Classpath in a Modular World

~~Classpath~~ Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path



The unnamed module can read **everything** from the modular JDK **at run-time**

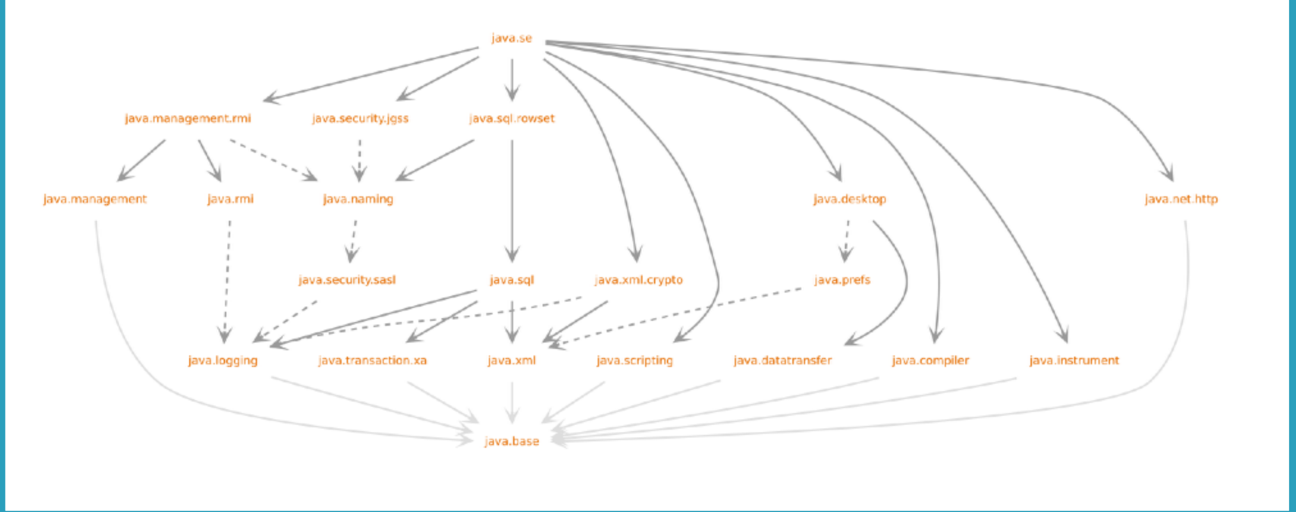
At **compile-time**, strong encapsulation is **enforced**

The Unnamed Module

Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path

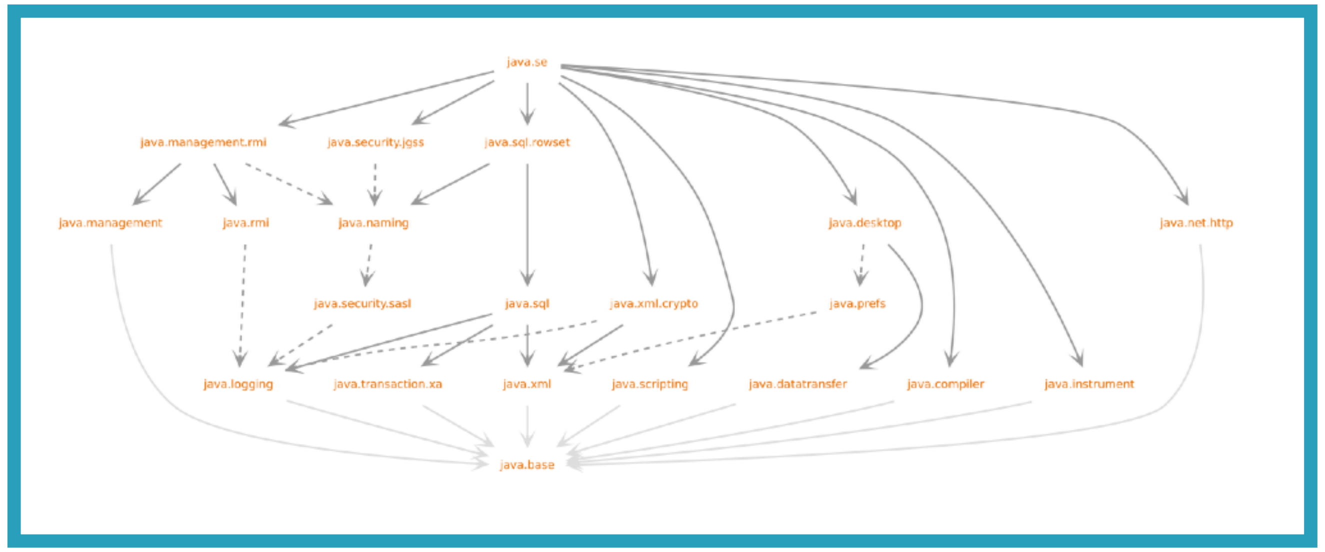


The Unnamed Module

Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path



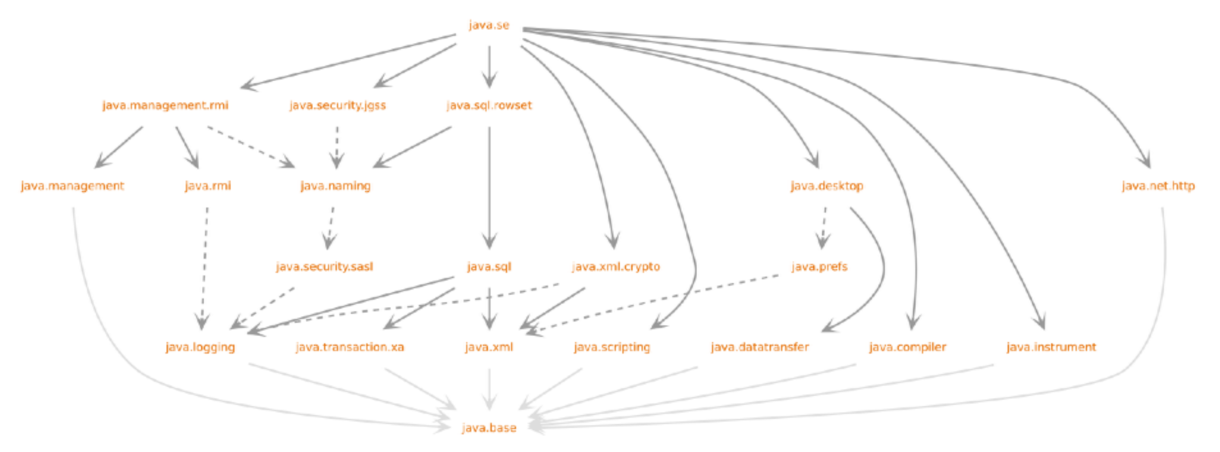
Even types in packages that are **not exported**

The Unnamed Module

Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path

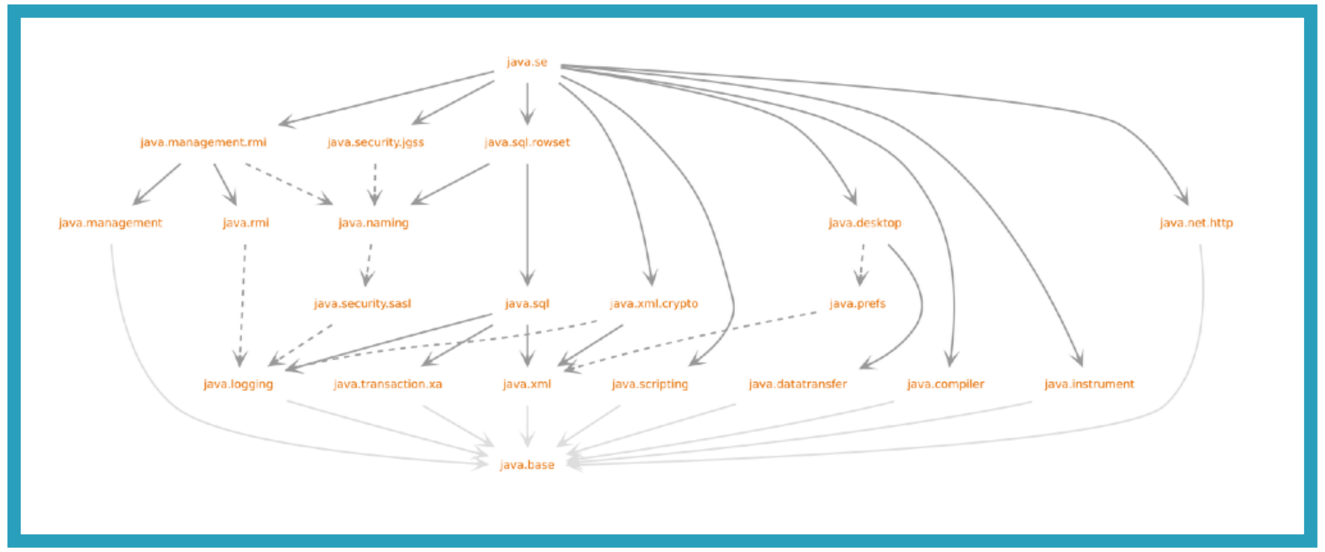


The Unnamed Module

Unnamed module

```
my-application.jar
dependency-1.0.jar
```

Modular JDK / Module path



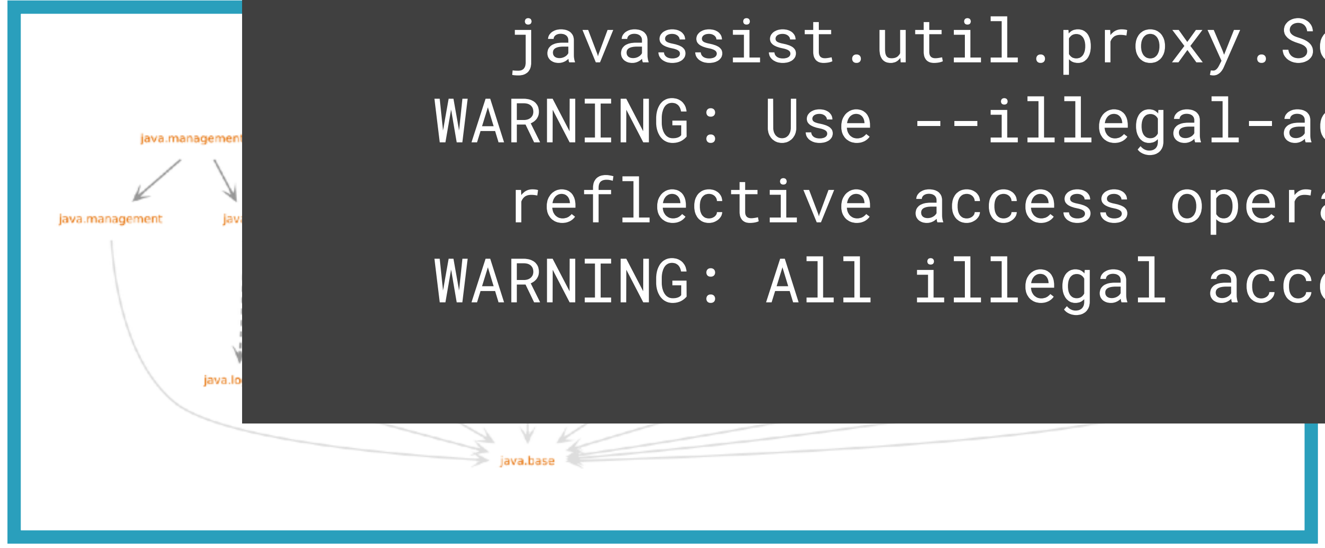
Even reflection on types in packages that are **not opened**

The Unnamed Module

Unnamed module

```
my-a
depe
```

Modul



```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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
```

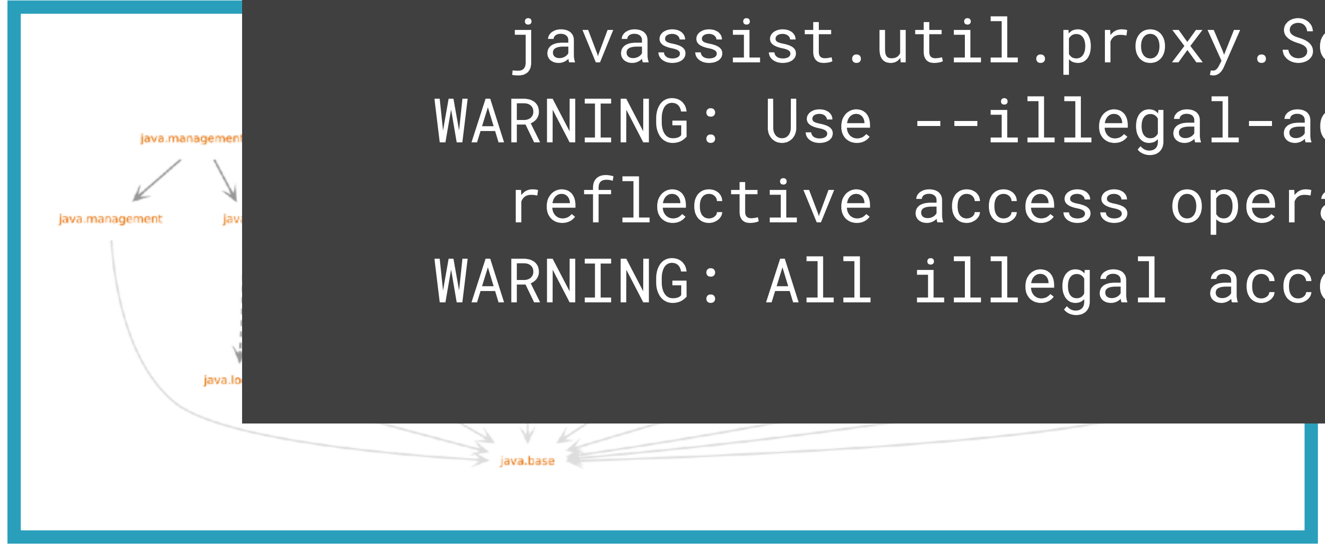
The Unnamed Module

Unnamed module

my-a

depe

Modul



```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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
```

`--illegal-access=deny`

Opening Packages from the Command Line

Opening Packages from the Command Line

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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
```

Opening Packages from the Command Line

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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
```

Upgrade to newer version

Opening Packages from the Command Line

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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
```

Opening Packages from the Command Line

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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-opens java.base/java.lang=<module_name>
```

Opening Packages from the Command Line

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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-opens java.base/java.lang=<module_name>
java --add-opens java.base/java.lang=ALL-UNNAMED
```

Opening Packages from the Command Line

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by javassist.util.proxy.SecurityActions
  (...javassist-3.20.0-GA.jar) 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-opens java.base/java.lang=<module_name>
java --add-opens java.base/java.lang=ALL-UNNAMED
--illegal-access=deny
```

Demo

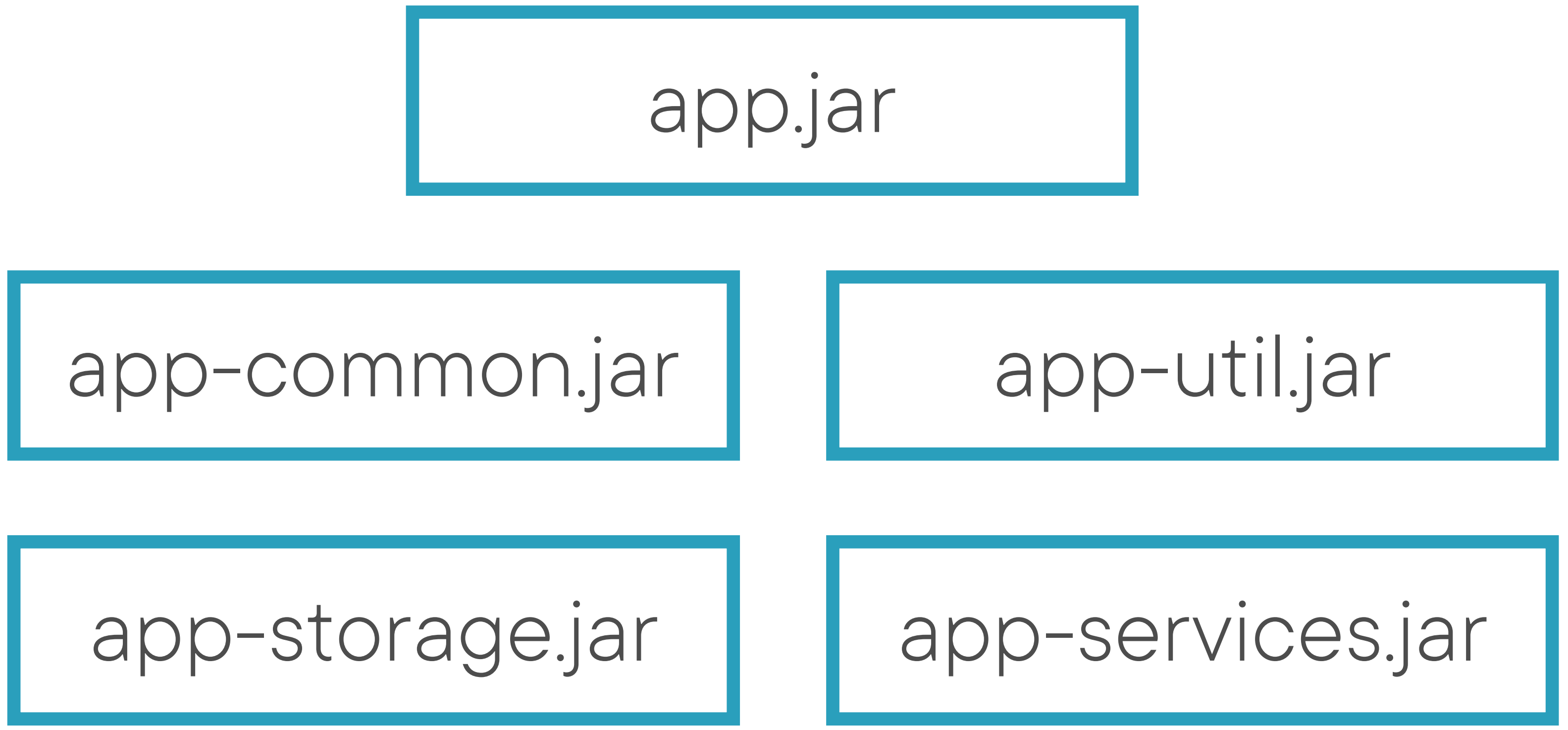
Exporting Packages from the Command Line

Demo

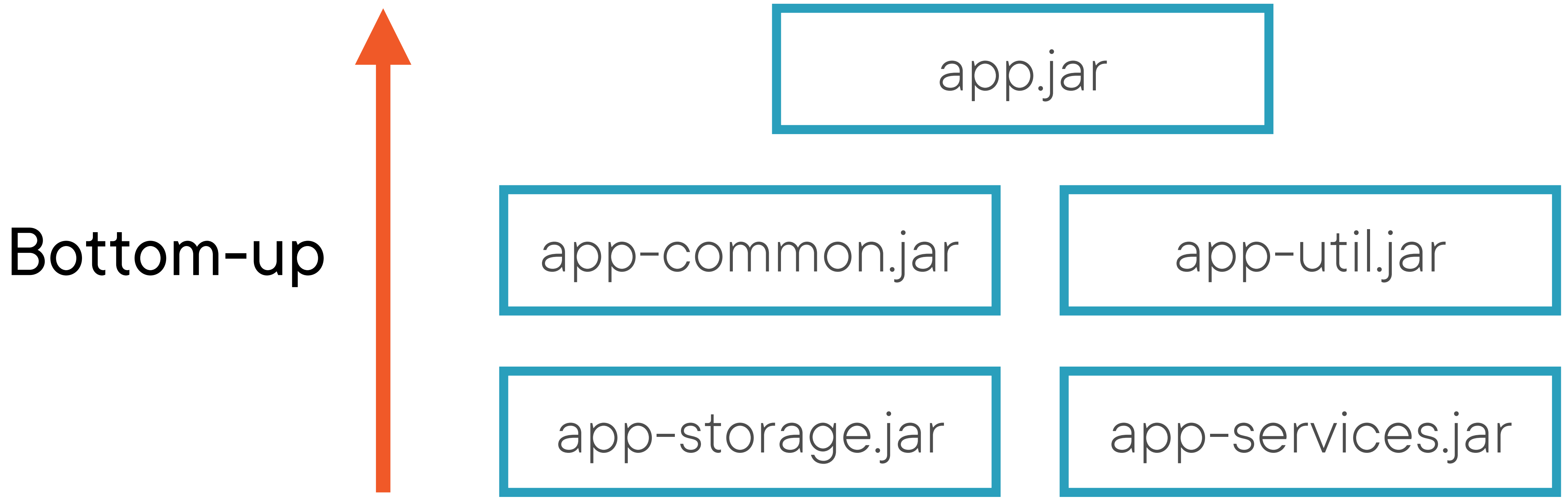
Exporting Packages from the Command Line

Migrating to Modules

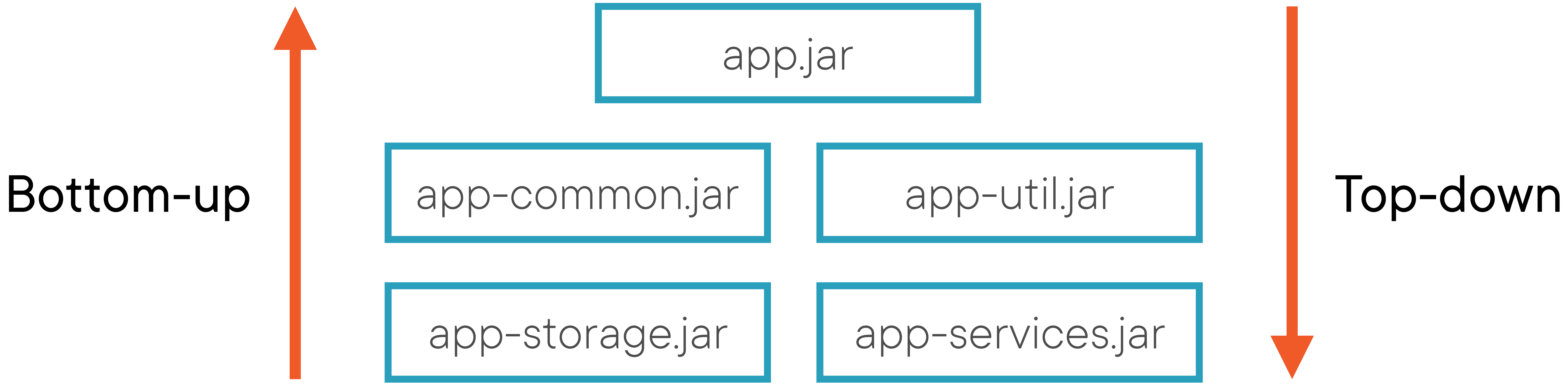
Migrating to Modules



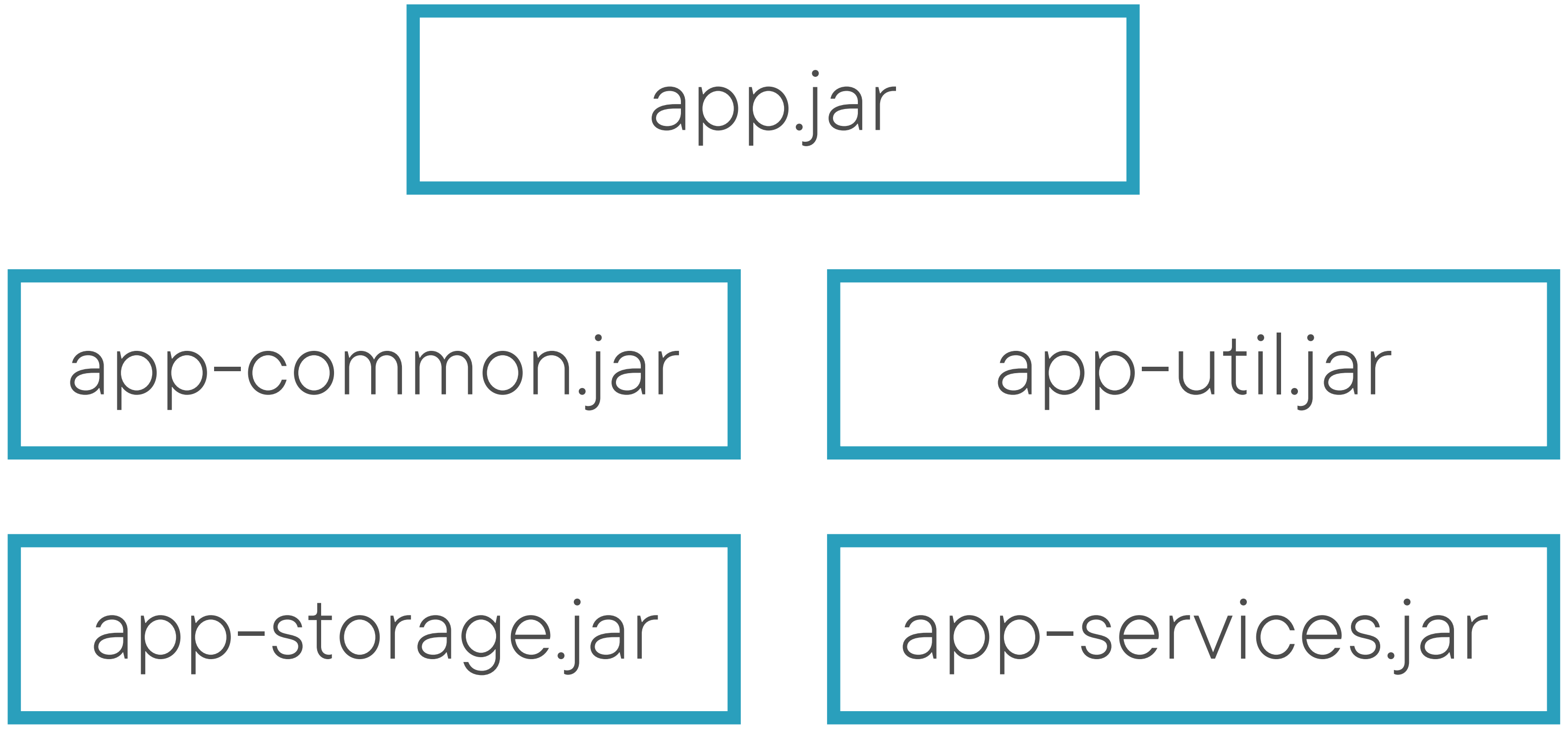
Migrating to Modules



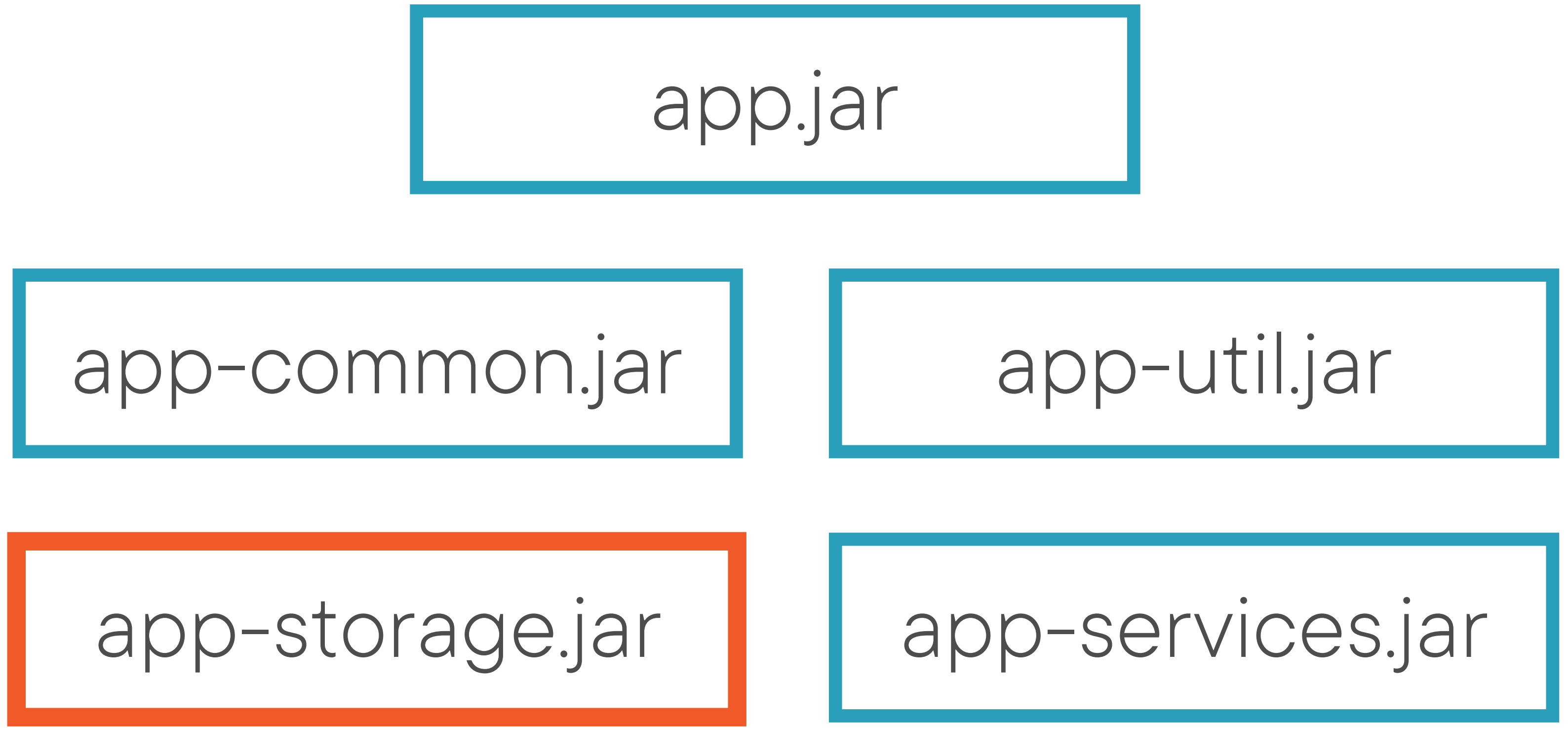
Migrating to Modules



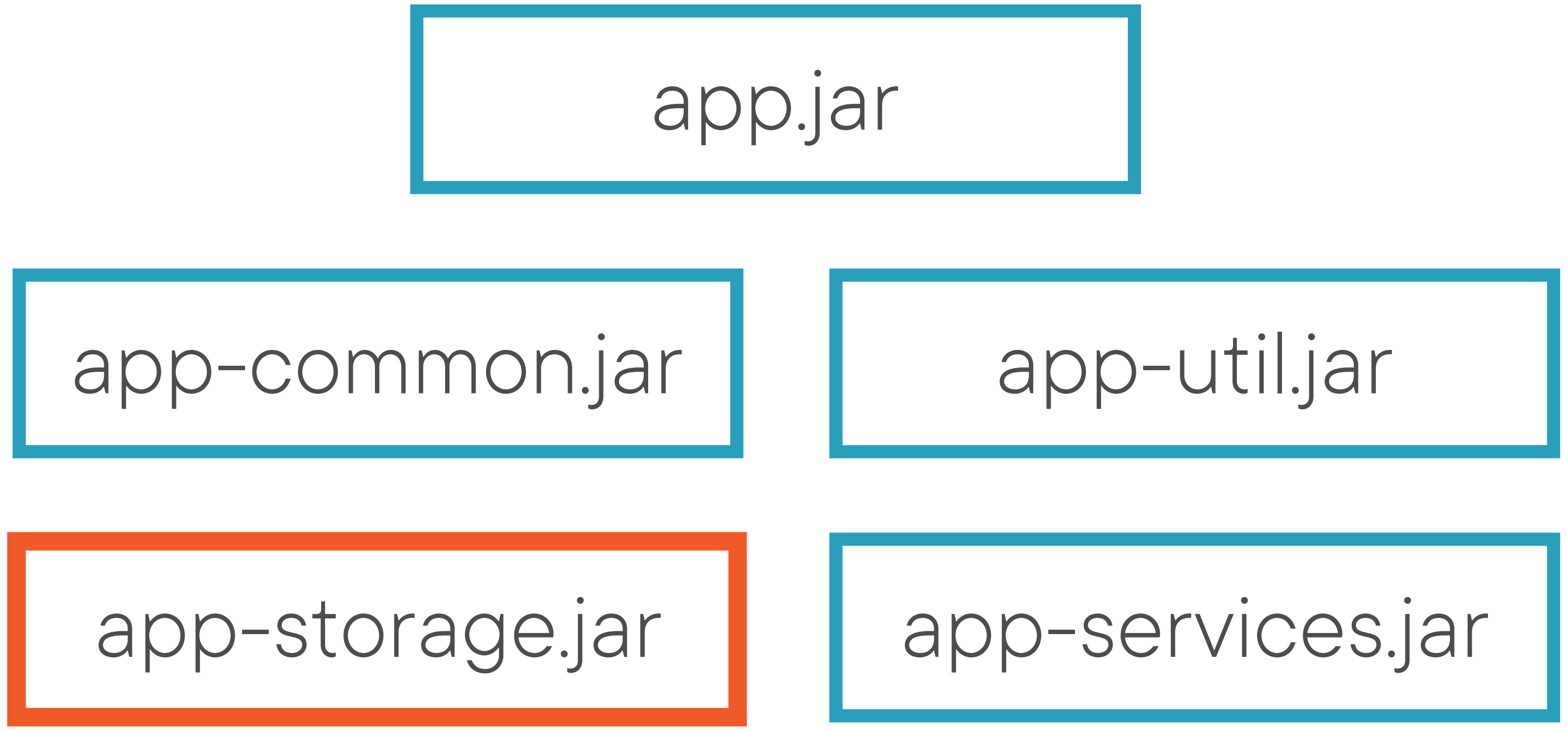
Bottom-up Migration



Bottom-up Migration



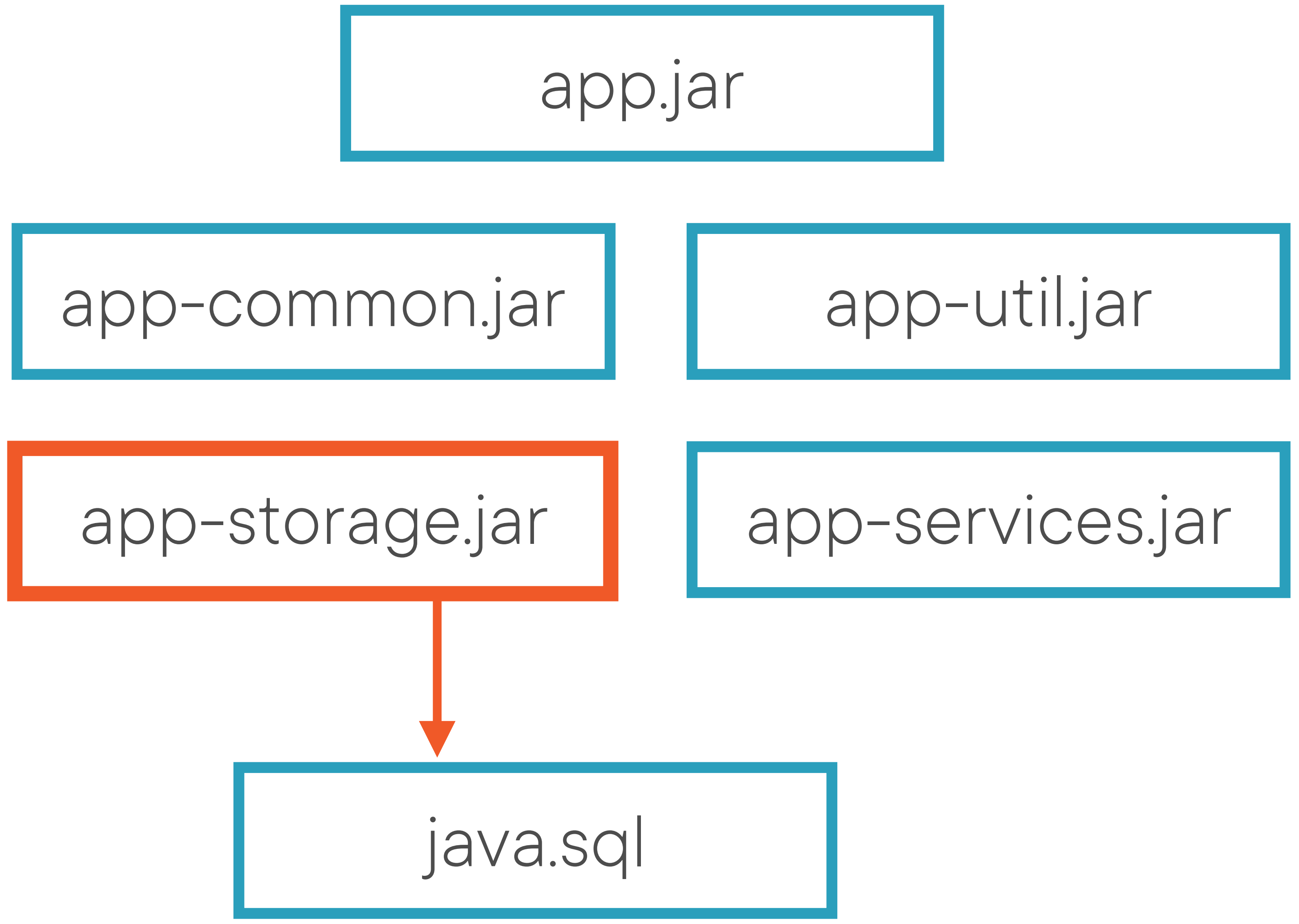
Bottom-up Migration



Starting with 'leaf' modules

- Write module-info.java
- Establish API
- Find JDK dependencies

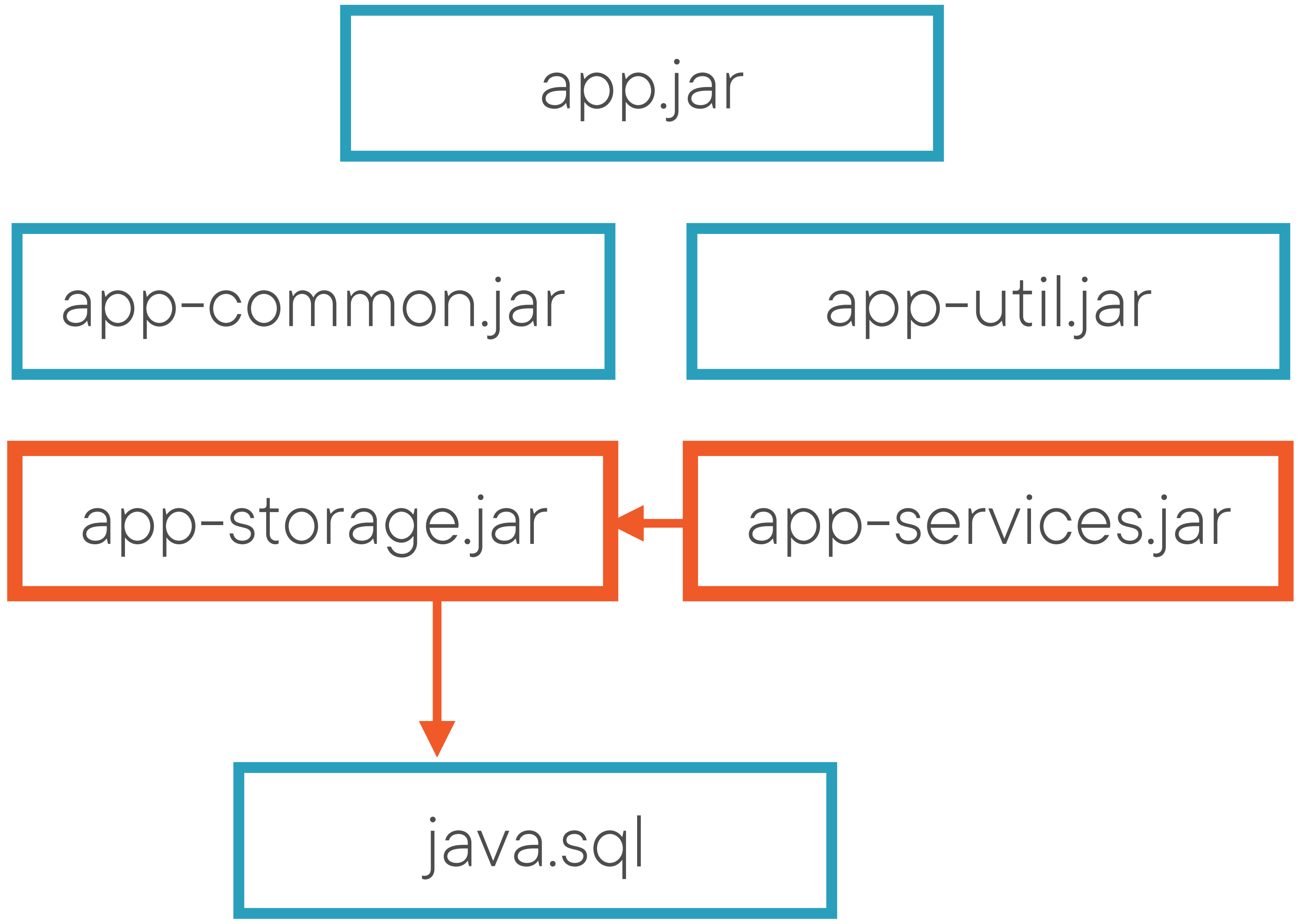
Bottom-up Migration



Starting with 'leaf' modules

- Write module-info.java
- Establish API
- Find JDK dependencies

Bottom-up Migration



Starting with 'leaf' modules

- Write module-info.java
- Establish API
- Find JDK dependencies

Bottom-up Migration: Using Jdeps

Bottom-up Migration: Using Jdeps

```
$ jdeps -s app-storage.jar
app-storage.jar -> java.base
app-storage.jar -> java.sql
```

Bottom-up Migration: Using Jdeps

```
$ jdeps -s app-storage.jar
app-storage.jar -> java.base
app-storage.jar -> java.sql
```

```
$ jdeps --generate-module-info . app-storage.jar
writing to ./app.storage/module-info.java
```

Bottom-up Migration: Limitations

Bottom-up Migration: Limitations

What if we don't control a dependency?

external-library.jar

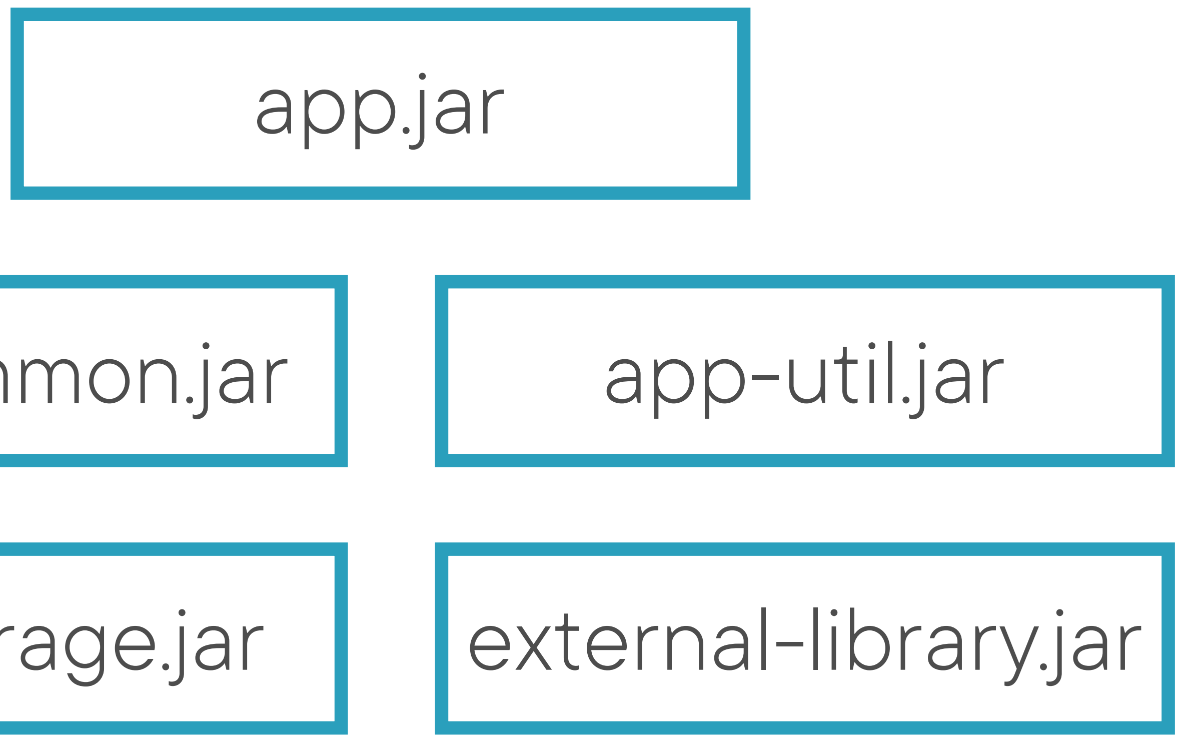
Bottom-up Migration: Limitations

What if we don't control a dependency?

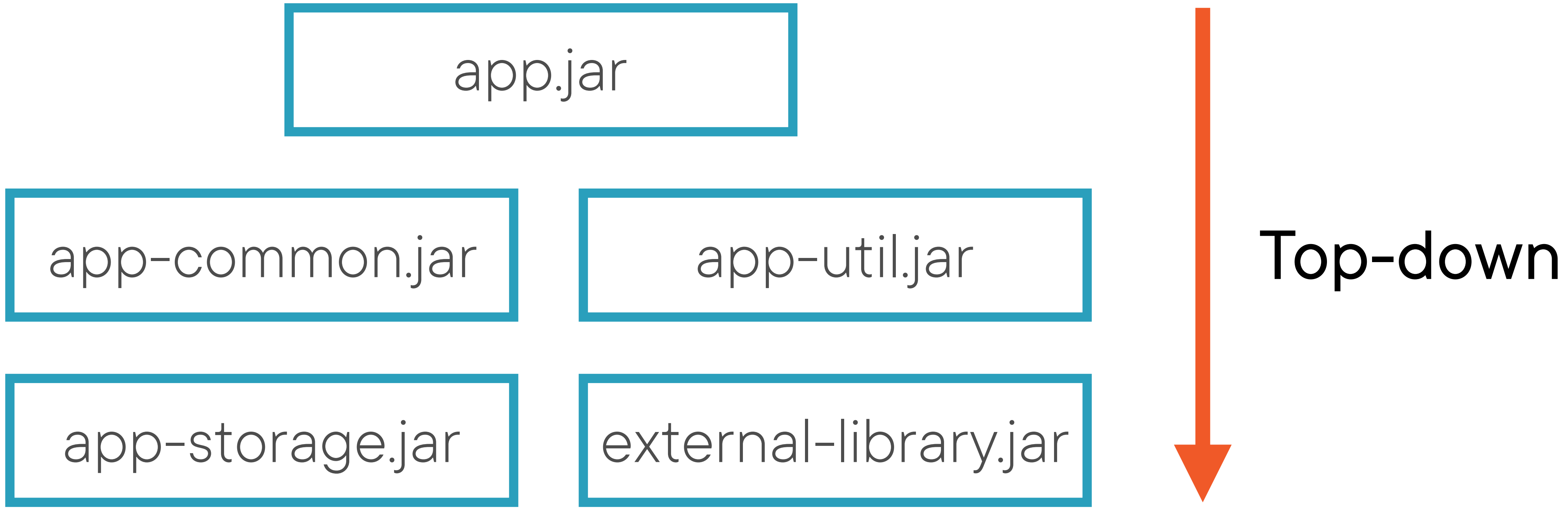
external-library.jar

- Wait for it to be modularized?
- Patch with our own module descriptor?

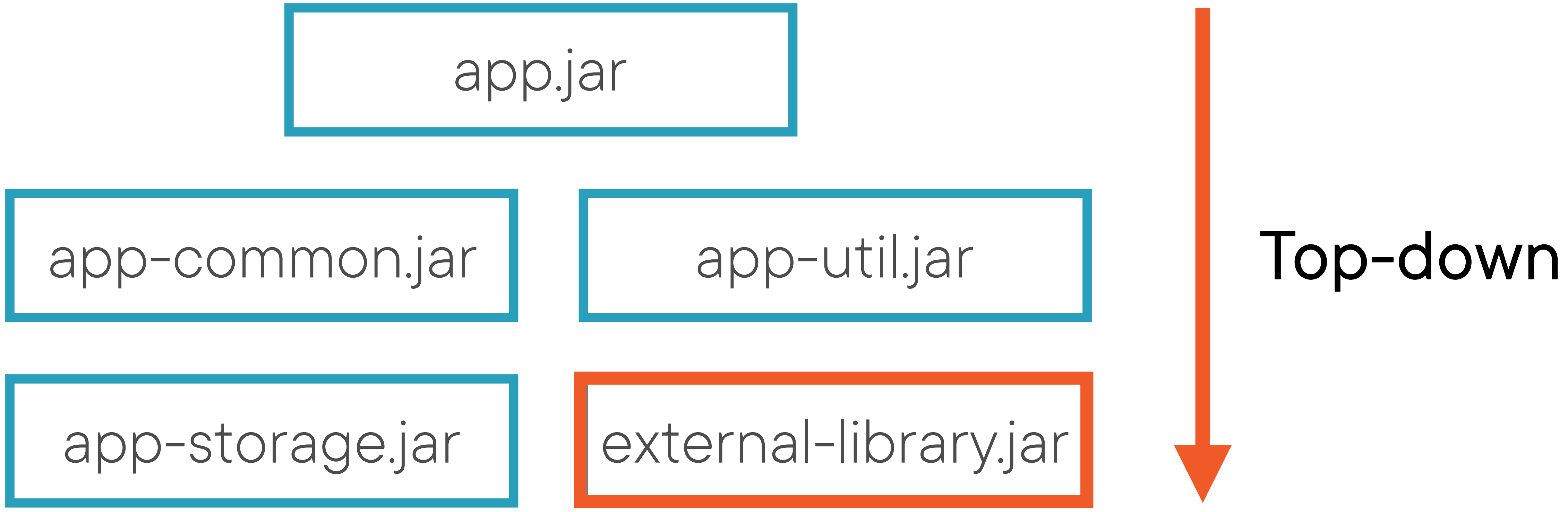
Top-down Migration



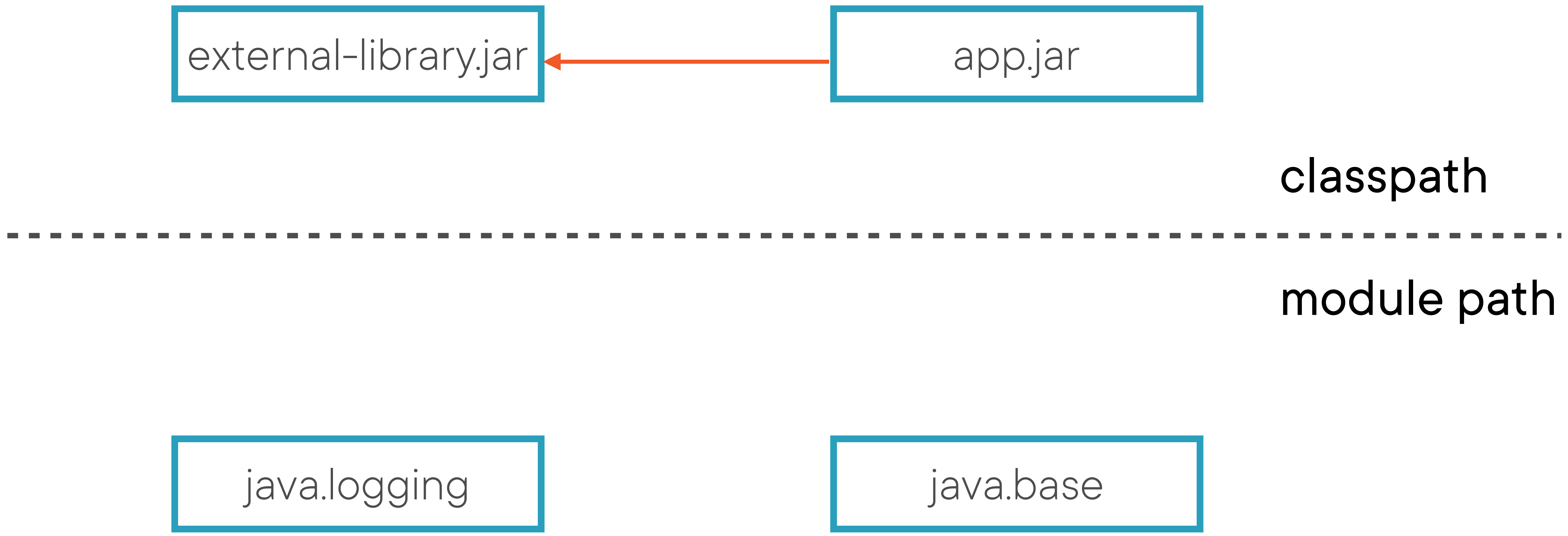
Top-down Migration



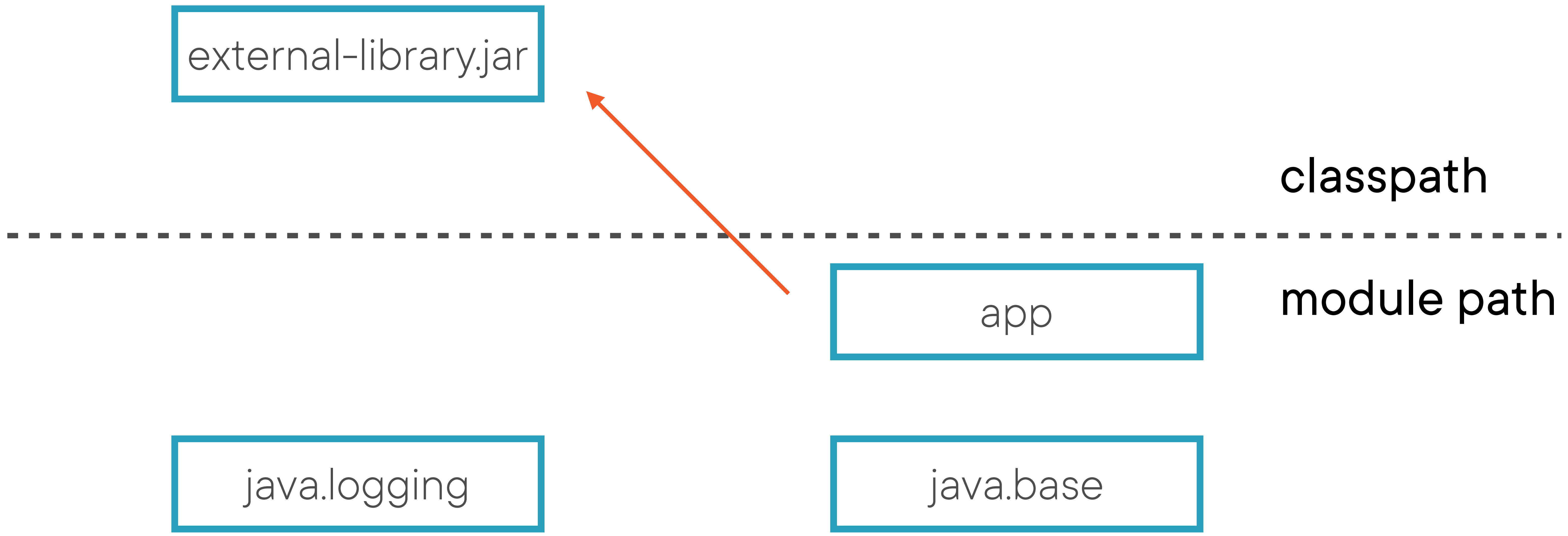
Top-down Migration



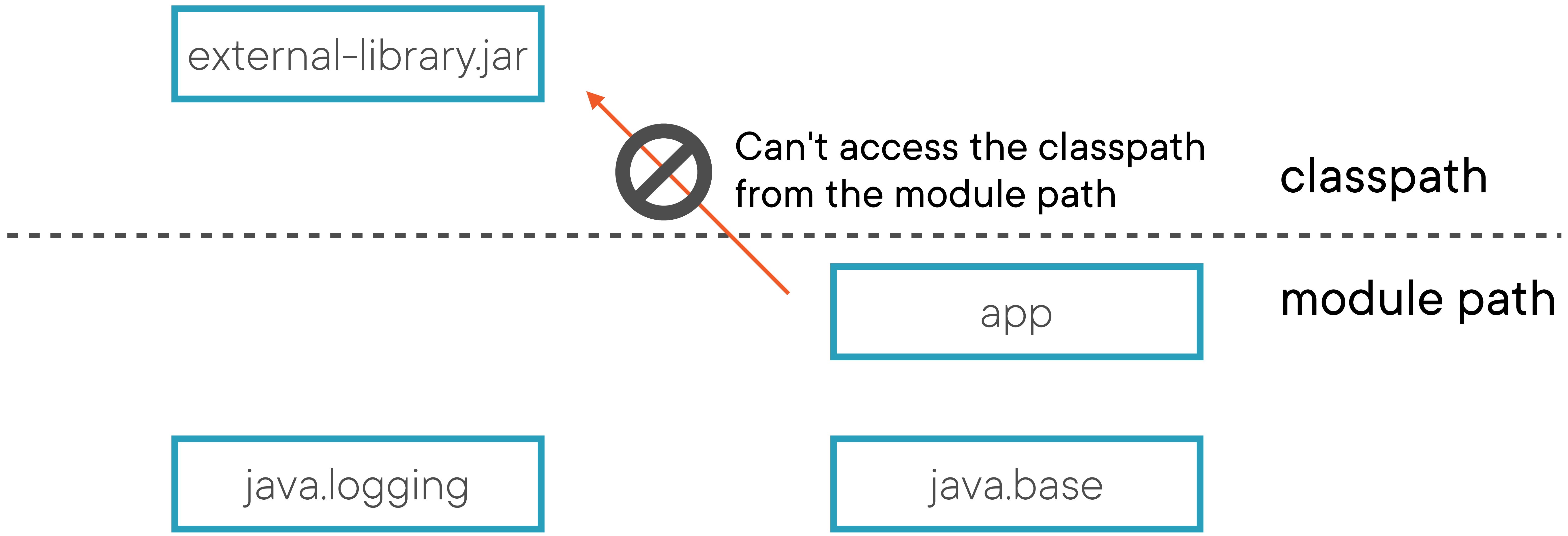
Automatic Modules



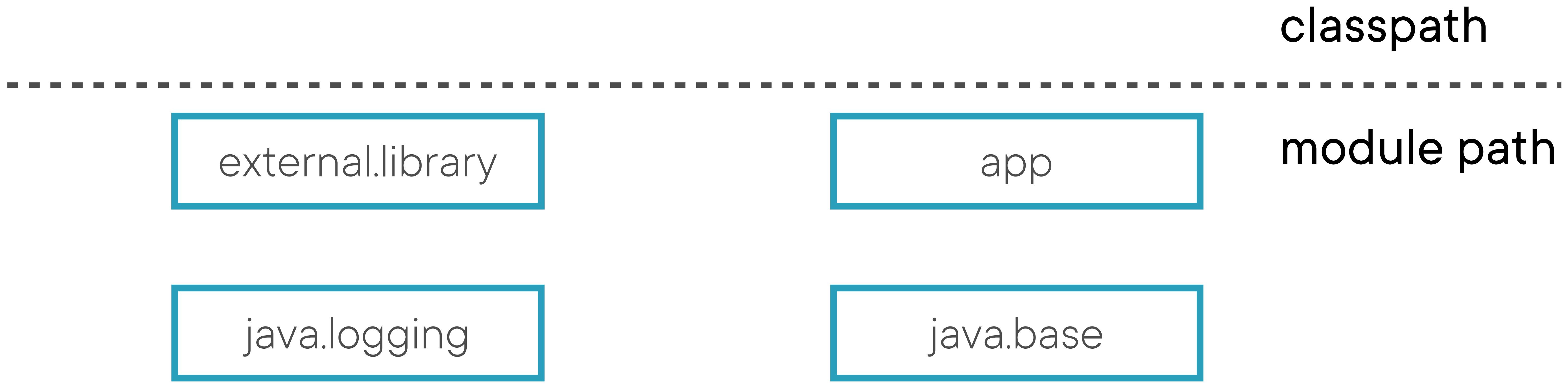
Automatic Modules



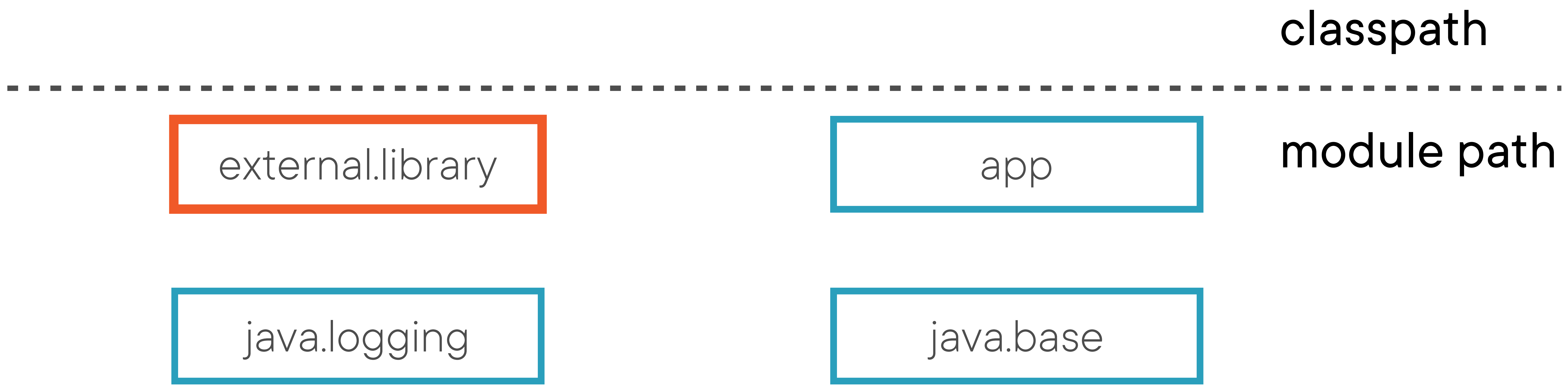
Automatic Modules



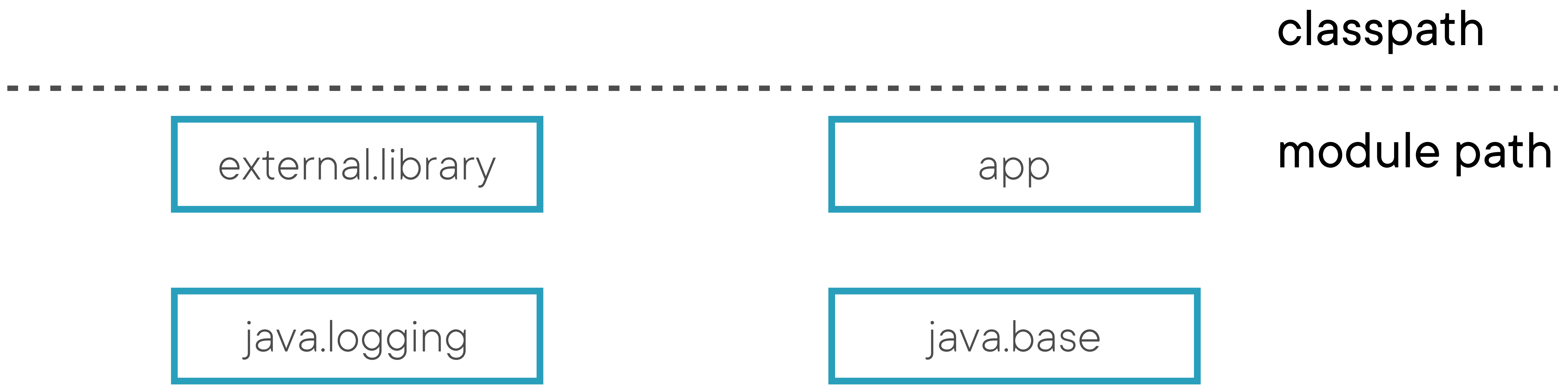
Automatic Modules



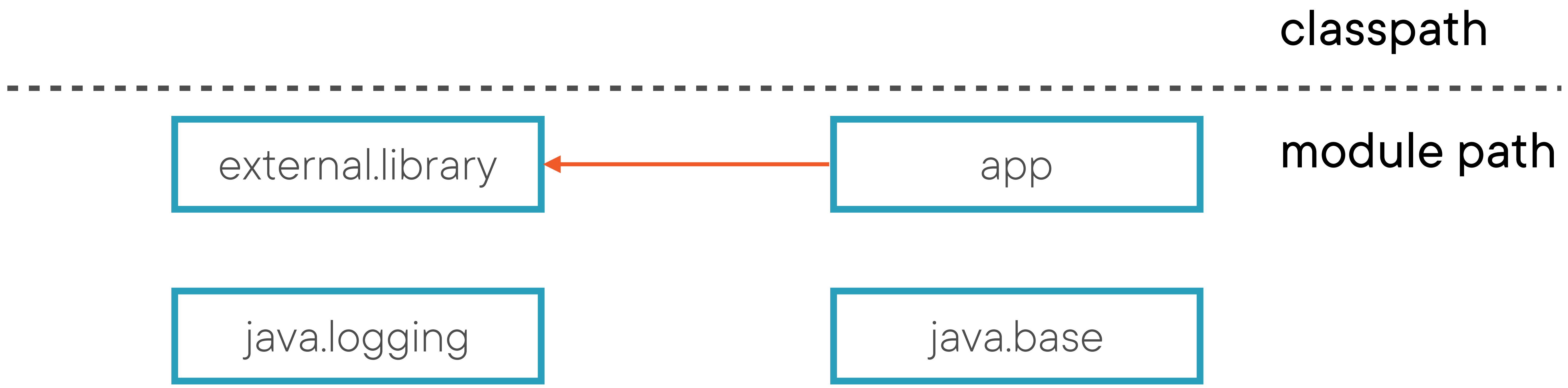
Automatic Modules



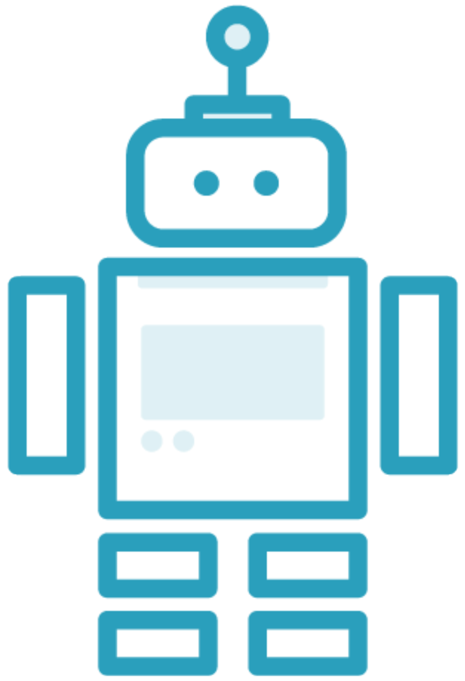
Automatic Modules



Automatic Modules

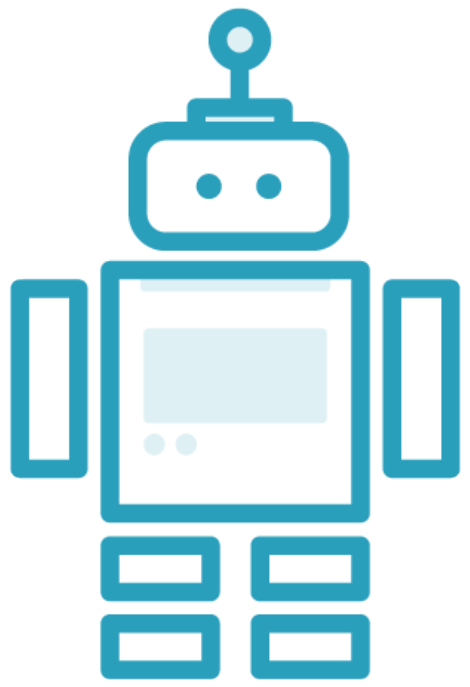


Automatic Modules



Put a non-modular JAR on the module path

Automatic Modules

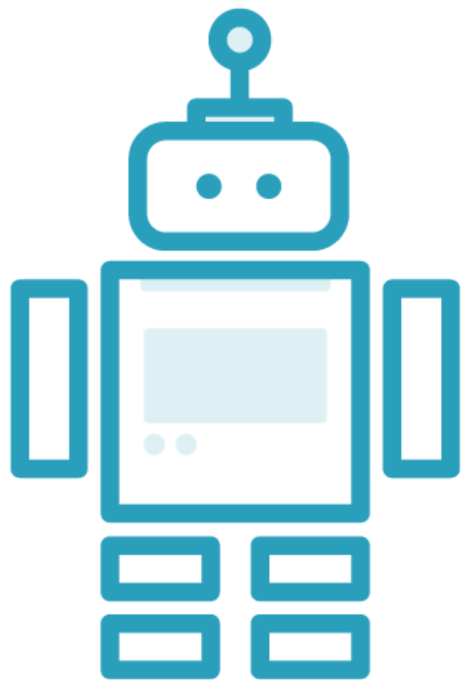


Put a non-modular JAR on the module path

– Name derived from JAR filename

See JavaDoc of `ModuleFinder`

Automatic Modules

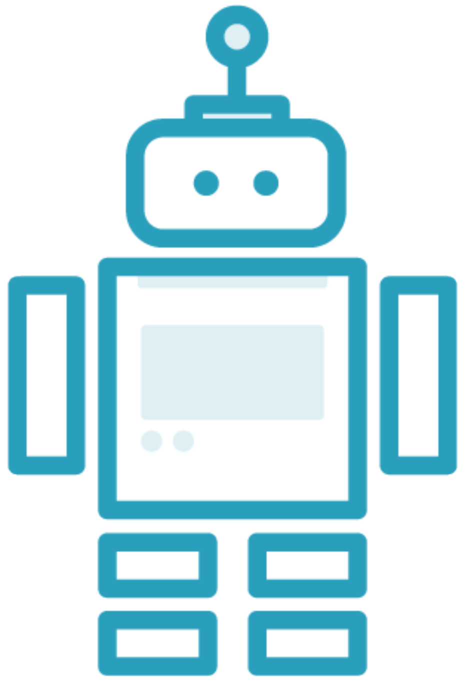


Put a non-modular JAR on the module path

- Name derived from JAR filename
- Or from `Automatic-Module-Name` header

In the JAR file's `META-INF/MANIFEST.MF`

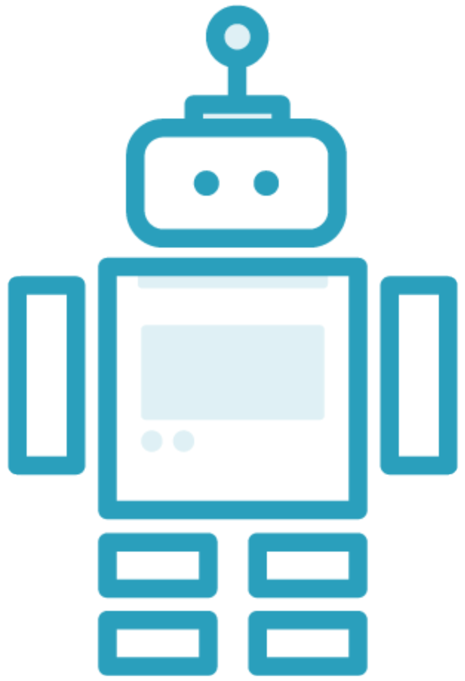
Automatic Modules



Put a non-modular JAR on the module path

- Name derived from JAR filename
- Or from `Automatic-Module-Name` header
- Exports & opens all packages

Automatic Modules



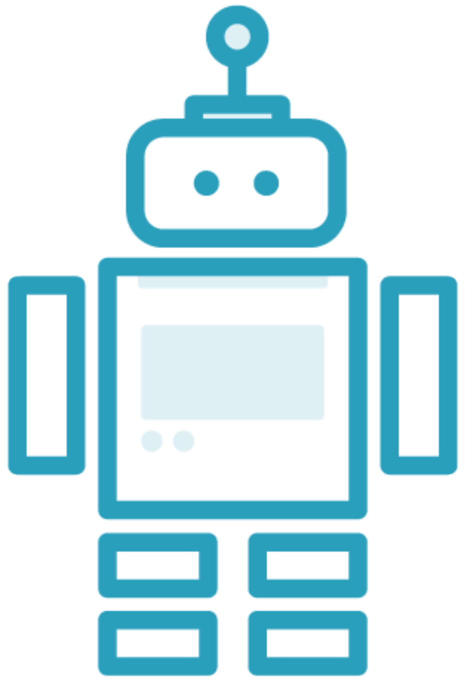
Put a non-modular JAR on the module path

- Name derived from JAR filename
- Or from `Automatic-Module-Name` header
- Exports & opens all packages
- Implicitly requires all resolved modules

Automatic Modules

Put a non-modular JAR on the module path

- Name derived from JAR filename
- Or from `Automatic-Module-Name` header
- Exports & opens all packages
- Implicitly requires all resolved modules
- Can access the classpath



Automatic Modules

Automatic modules are only meant to be used for migration

Automatic Modules

Course Wrap-up

Course Wrap-up

Introducing the Java Module System

Course Wrap-up

Introducing the Java Module System

Working with Modules

Course Wrap-up

Introducing the Java Module System

Working with Modules

Understanding the Modular JDK

Course Wrap-up

Introducing the Java Module System

Working with Modules

Understanding the Modular JDK

Using Services

Course Wrap-up

Introducing the Java Module System

Working with Modules

Understanding the Modular JDK

Using Services

Migrating to Modules

Topics Not Discussed

Topics Not Discussed

Advanced module system topics

Topics Not Discussed

Advanced module system topics

– Module Layer API

Topics Not Discussed

Advanced module system topics

- Module Layer API
- **Custom run-time images with jlink**

Topics Not Discussed

Advanced module system topics

- Module Layer API
- Custom run-time images with jlink
- **Resource loading with modules**

Topics Not Discussed

Advanced module system topics

– Module Layer API

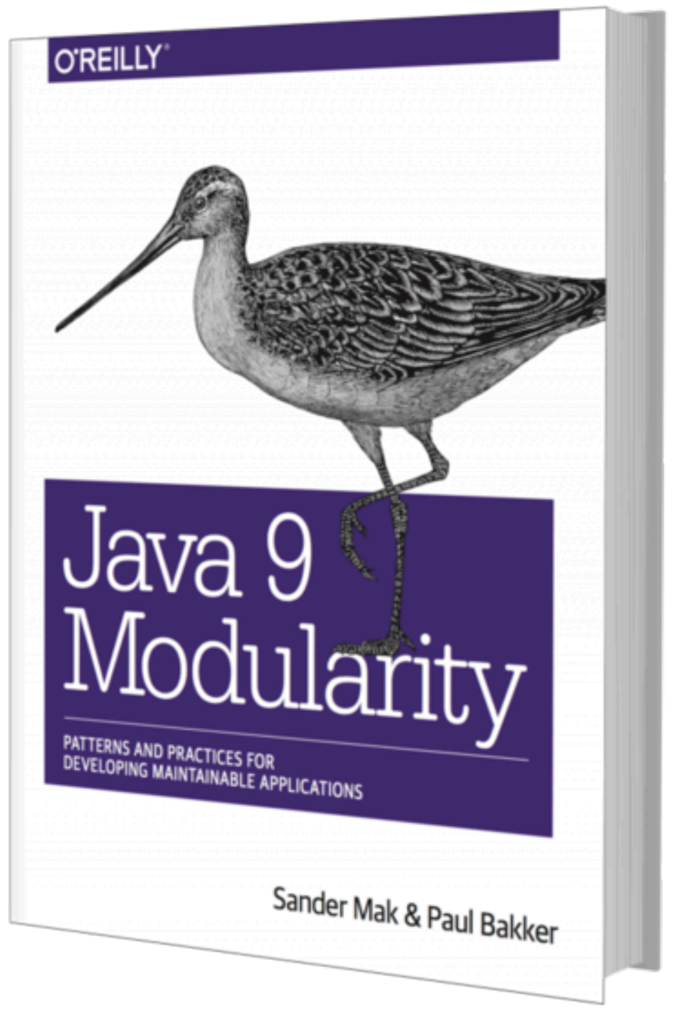
– Custom run-time images with jlink

– Resource loading with modules

– **Integration with build tools**

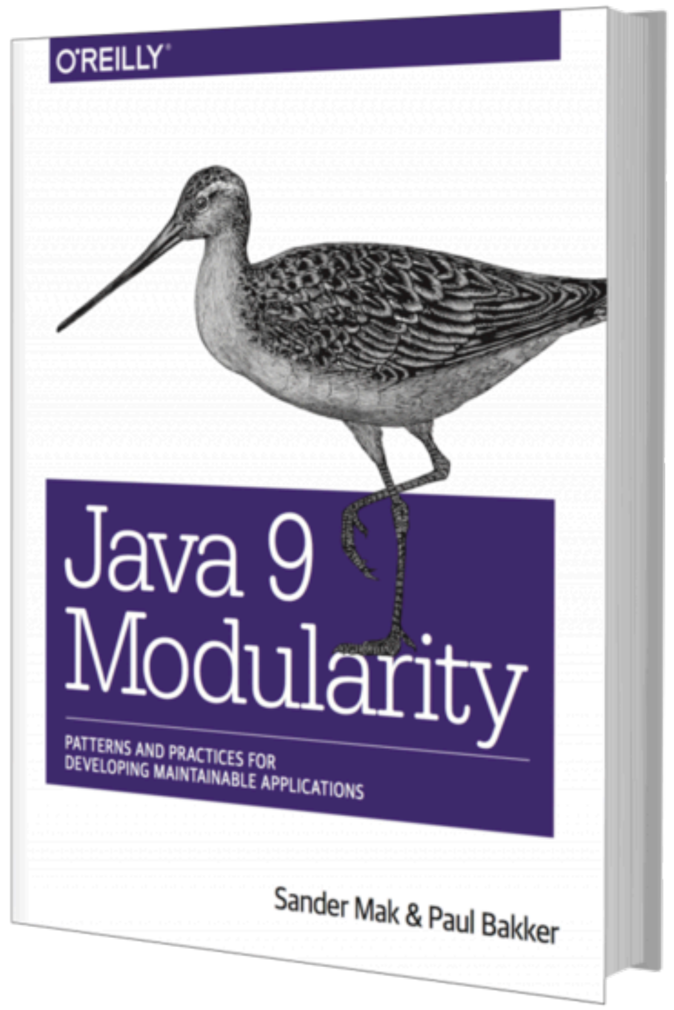
Additional Resources

Additional Resources

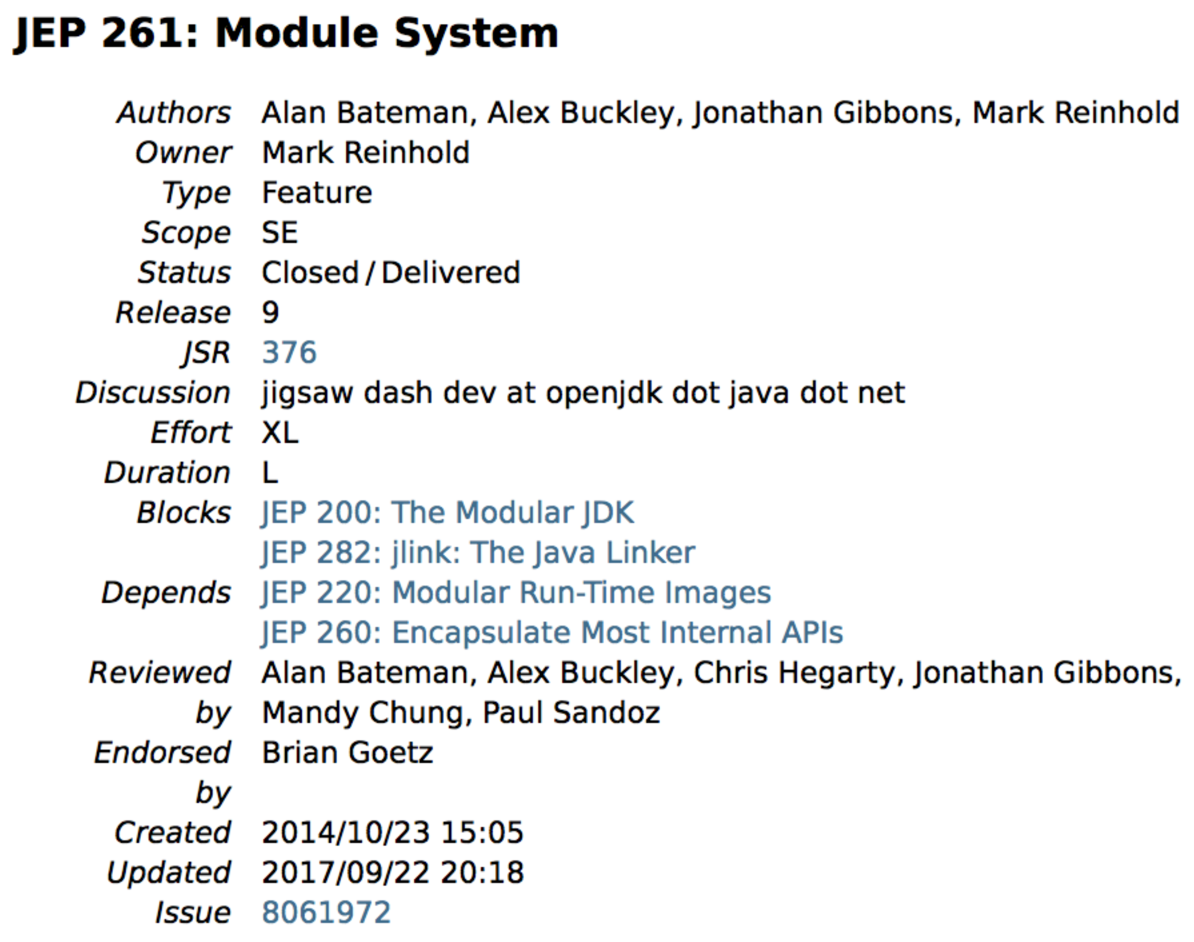

O'REILLY
Java 9 Modularity
PATTERNS AND PRACTICES FOR DEVELOPING MAINTAINABLE APPLICATIONS
Sander Mak & Paul Bakker

javamodularity.com

Additional Resources


O'REILLY
Java 9 Modularity
PATTERNS AND PRACTICES FOR DEVELOPING MAINTAINABLE APPLICATIONS
Sander Mak & Paul Bakker

javamodularity.com



JEP 261: Module System

Authors	Alan Bateman, Alex Buckley, Jonathan Gibbons, Mark Reinhold
Owner	Mark Reinhold
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	9
JSR	376
Discussion	jigsaw dash dev at openjdk dot java dot net
Effort	XL
Duration	L
Blocks	JEP 200: The Modular JDK JEP 282: jlink: The Java Linker
Depends	JEP 220: Modular Run-Time Images JEP 260: Encapsulate Most Internal APIs
Reviewed by	Alan Bateman, Alex Buckley, Chris Hegarty, Jonathan Gibbons, Mandy Chung, Paul Sandoz
Endorsed by	Brian Goetz
Created	2014/10/23 15:05
Updated	2017/09/22 20:18
Issue	8061972

openjdk.java.net/jeps/261

Additional Resources

What's New in Java {9,10,11}

Additional Resources

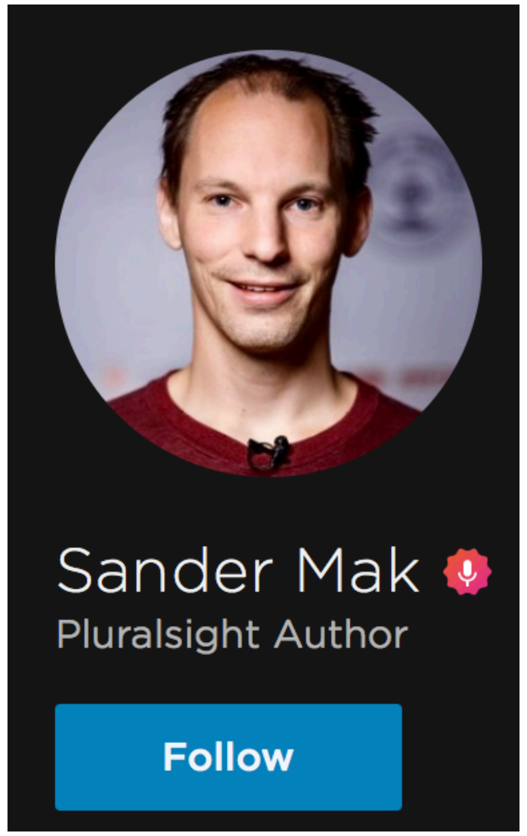
Additional Resources

@Sander_Mak

Additional Resources

@Sander_Mak

Follow for updates



bit.ly/ps-sander