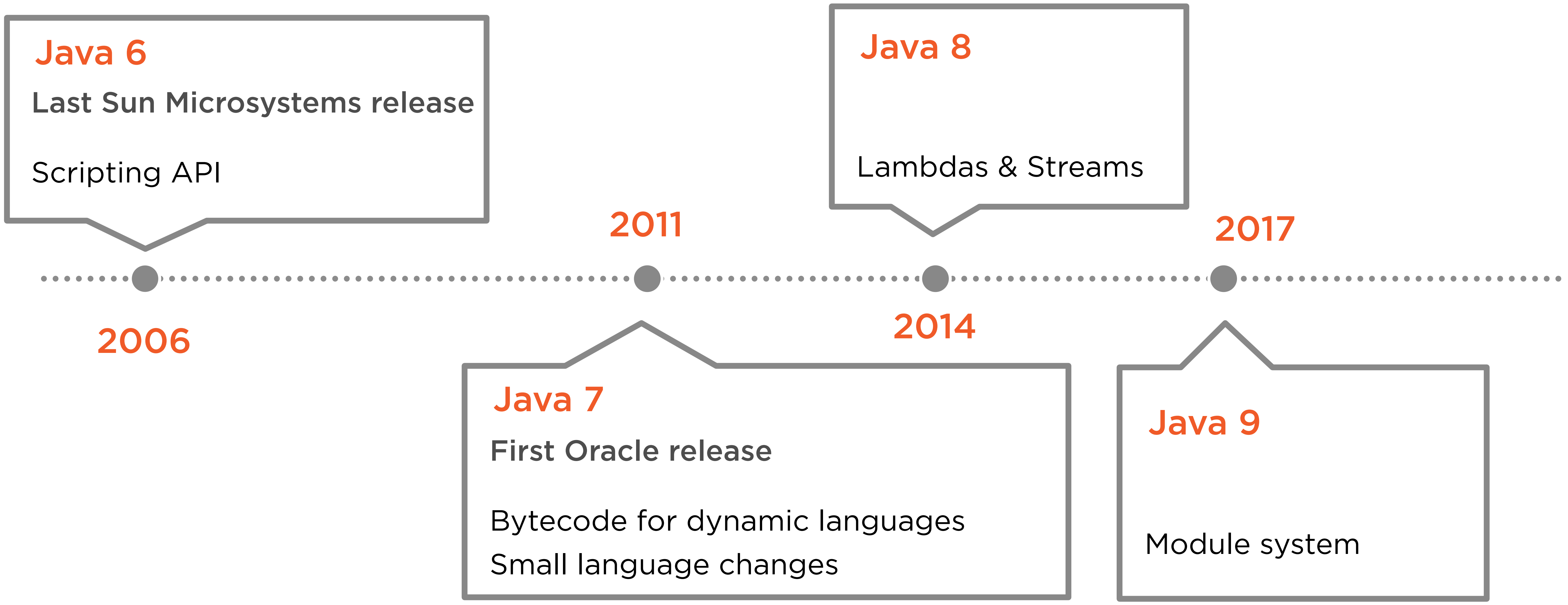
What's New in Java 9

THE JAVA PLATFORM MODULE SYSTEM

Sander Mak
FELLOW & SOFTWARE ARCHITECT
@Sander_Mak

Java 9 in Perspective



Course Overview

Modules

JShell

Library & Language Improvements

Course Overview

New APIs

Desktop Java Enhancements

Performance & Security

Follow Along

Download JDK 9

jdk.java.net/9/

IDE: IntelliJ Community Edition

jetbrains.com/idea/download/

The Java Platform Module System

One of the biggest changes in Java, ever

- Language
- Compiler
- Virtual Machine
- Tooling

The Java Platform Module System

Modularize the JDK

Modularize applications

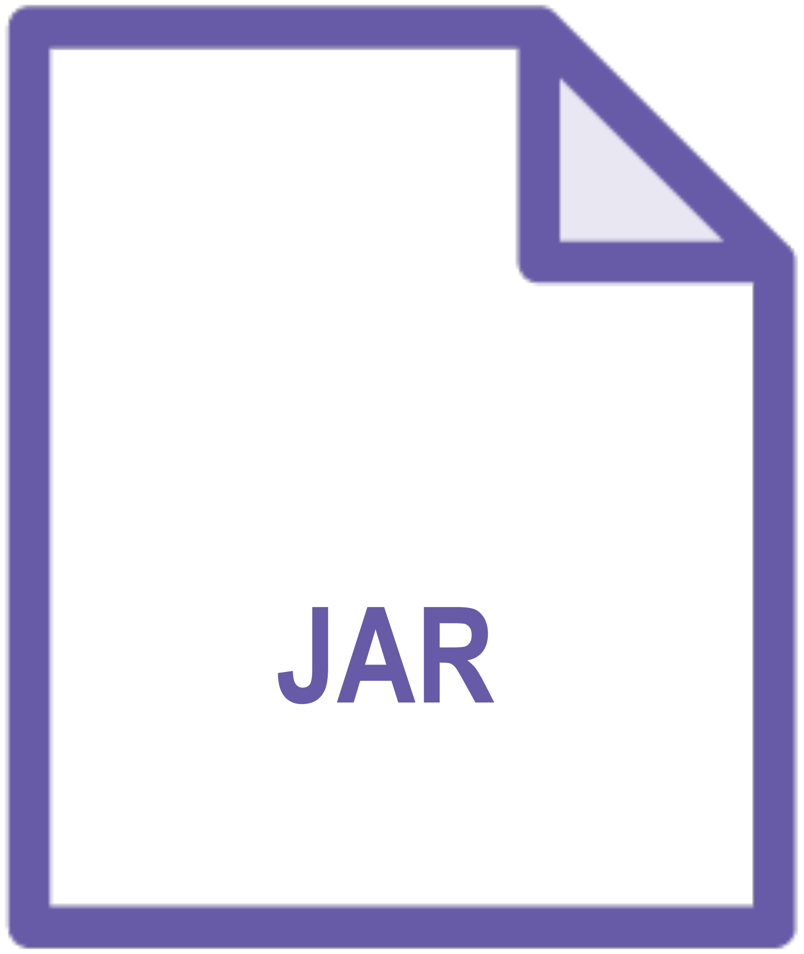
Using the module system is optional!

The Java Platform Module System

Modularize applications

Course:
Java 9 Modularity: First Look

Before the Modular JDK



rt.jar

One huge library

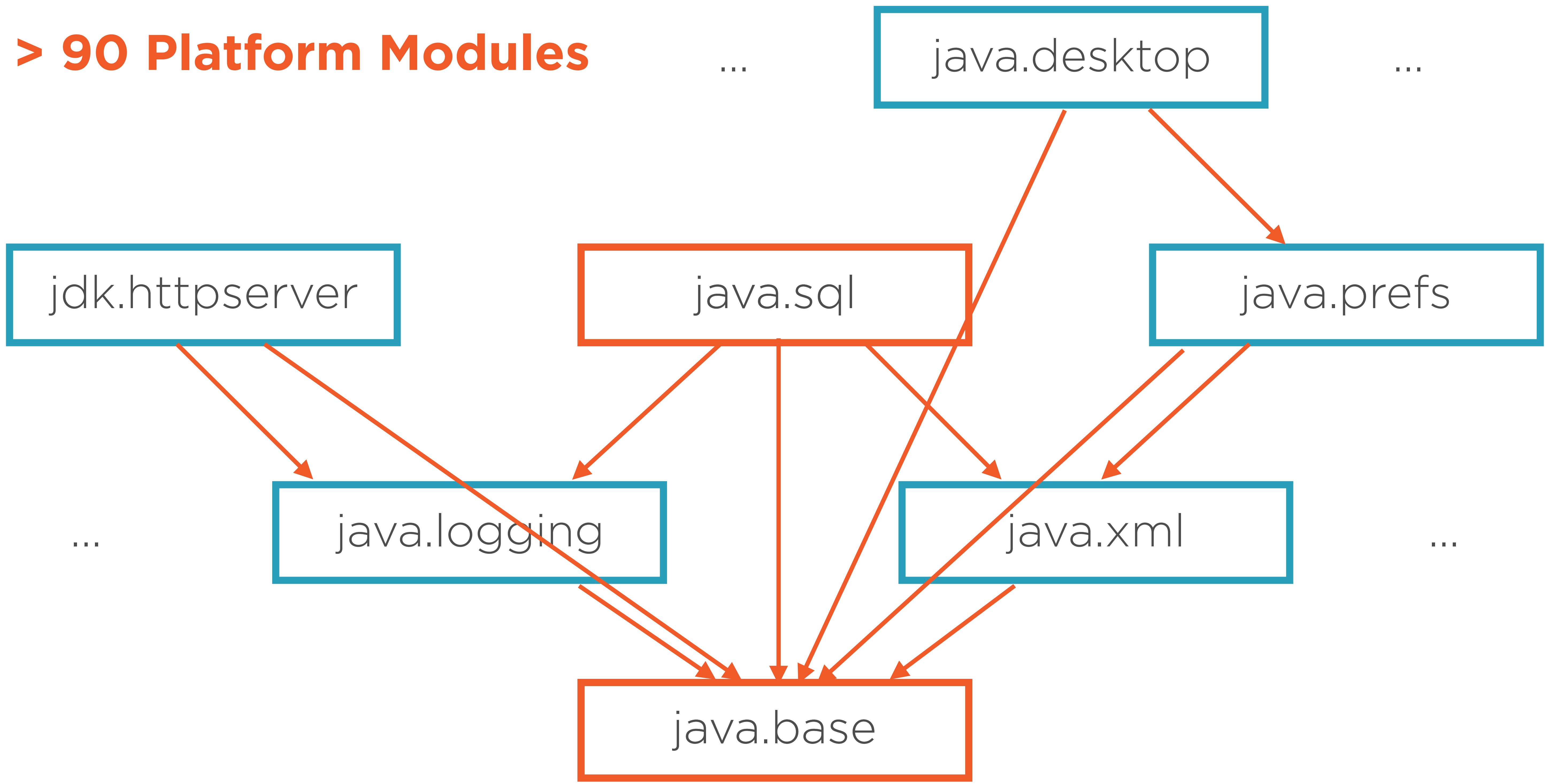
Many entangled classes

Hard to evolve

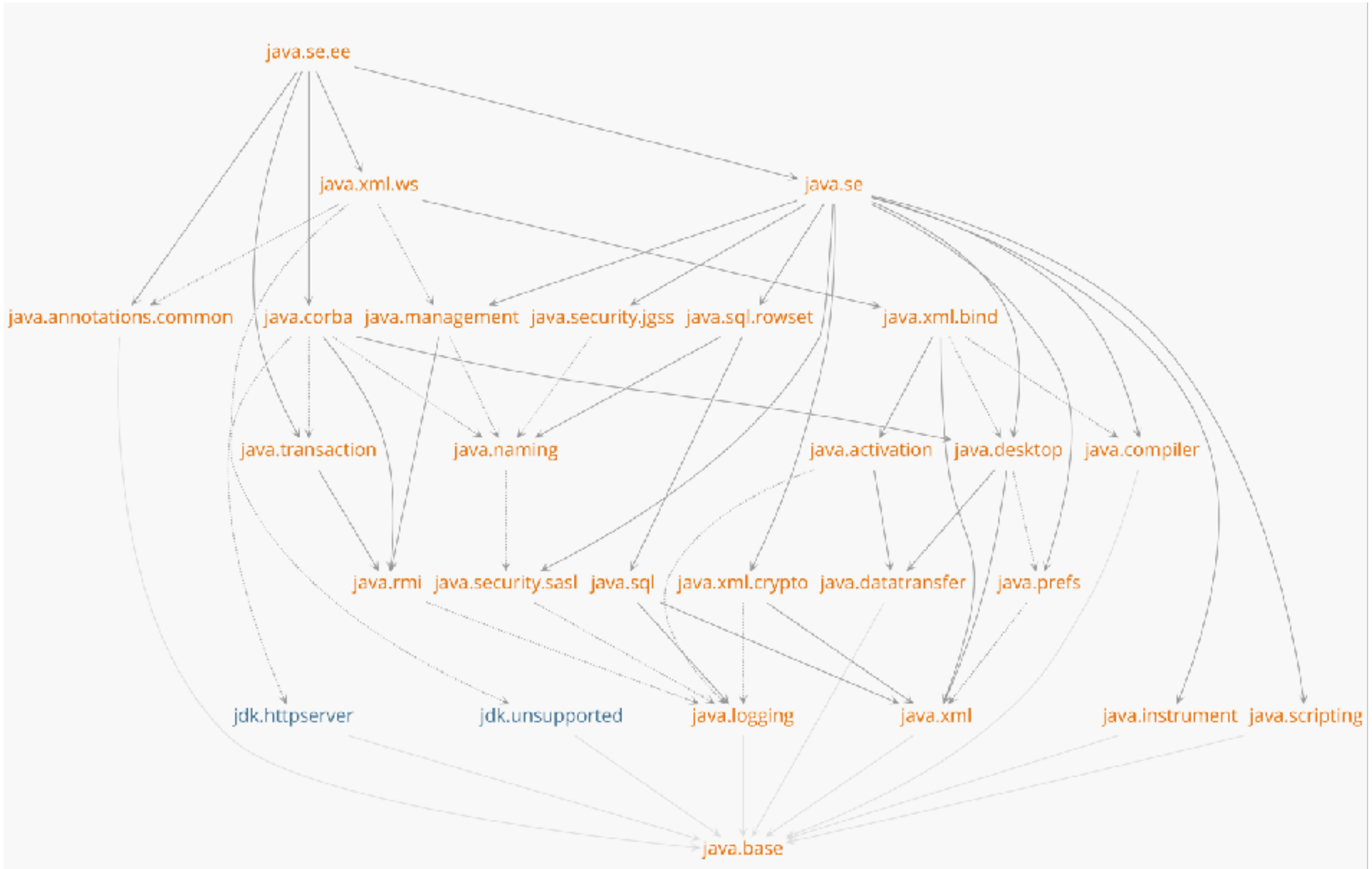
Restricted by backward-compatibility

The Modular JDK: Explicit Dependencies

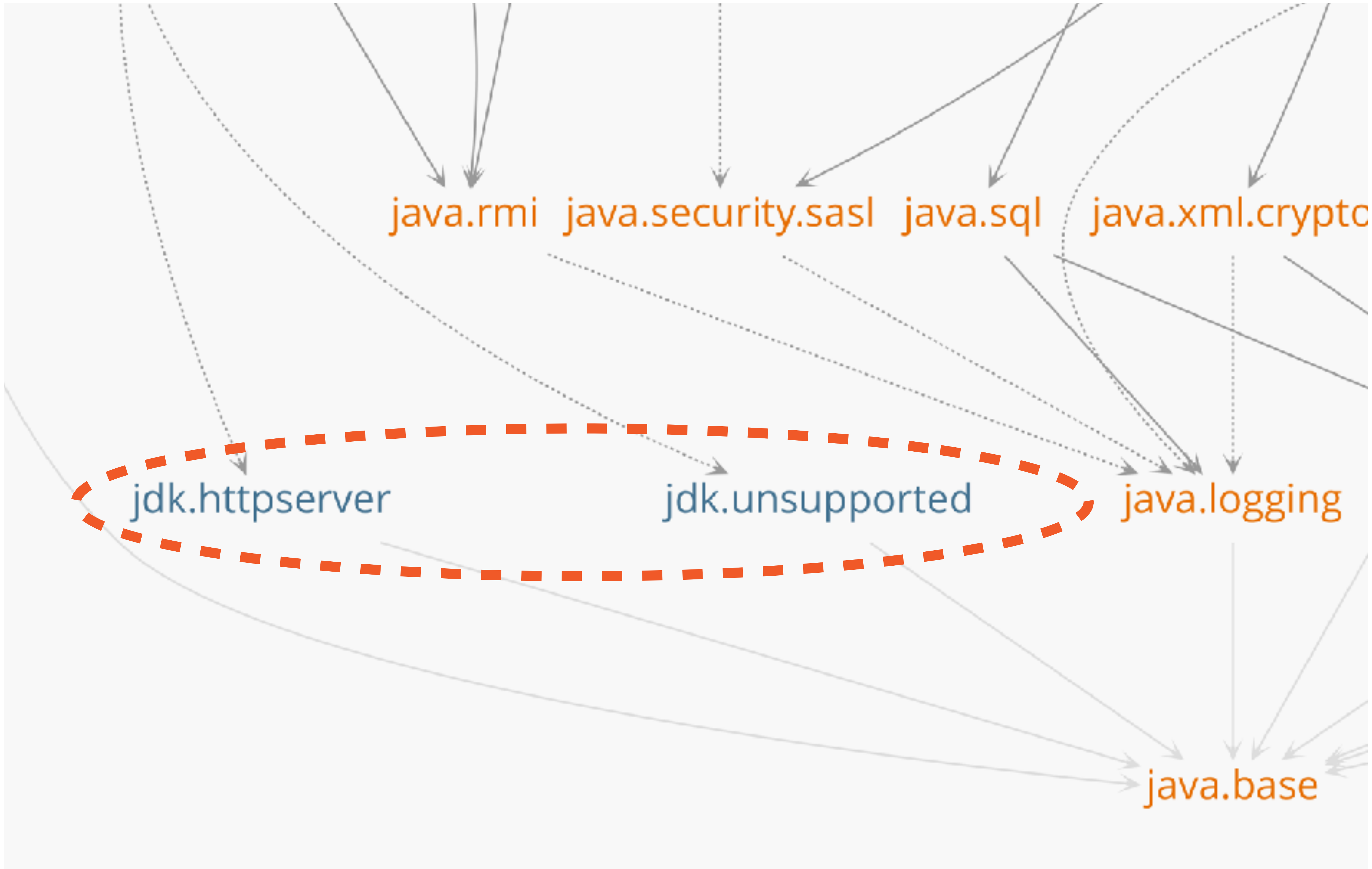
> 90 Platform Modules



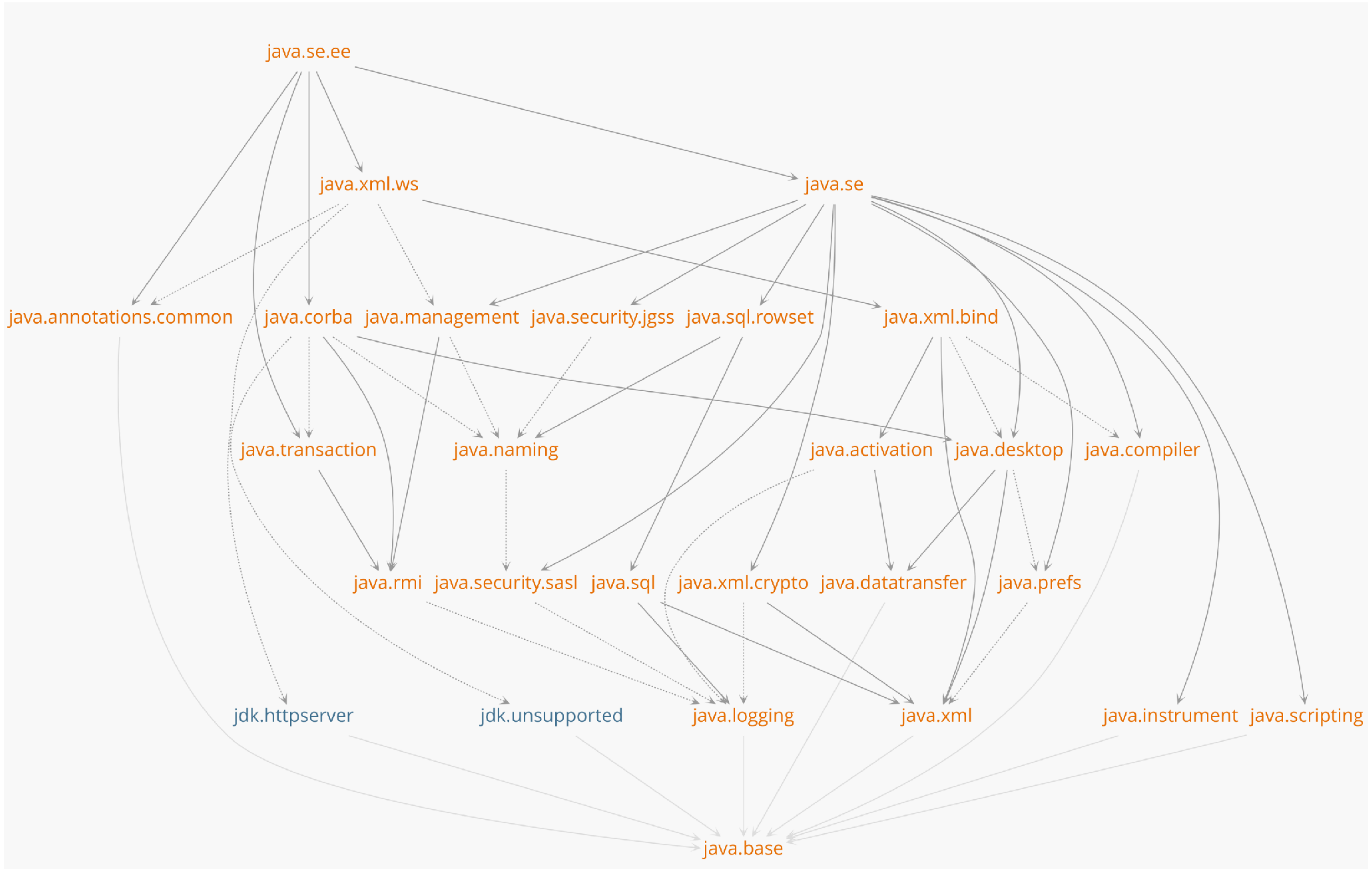
The Modular JDK: Explicit Dependencies



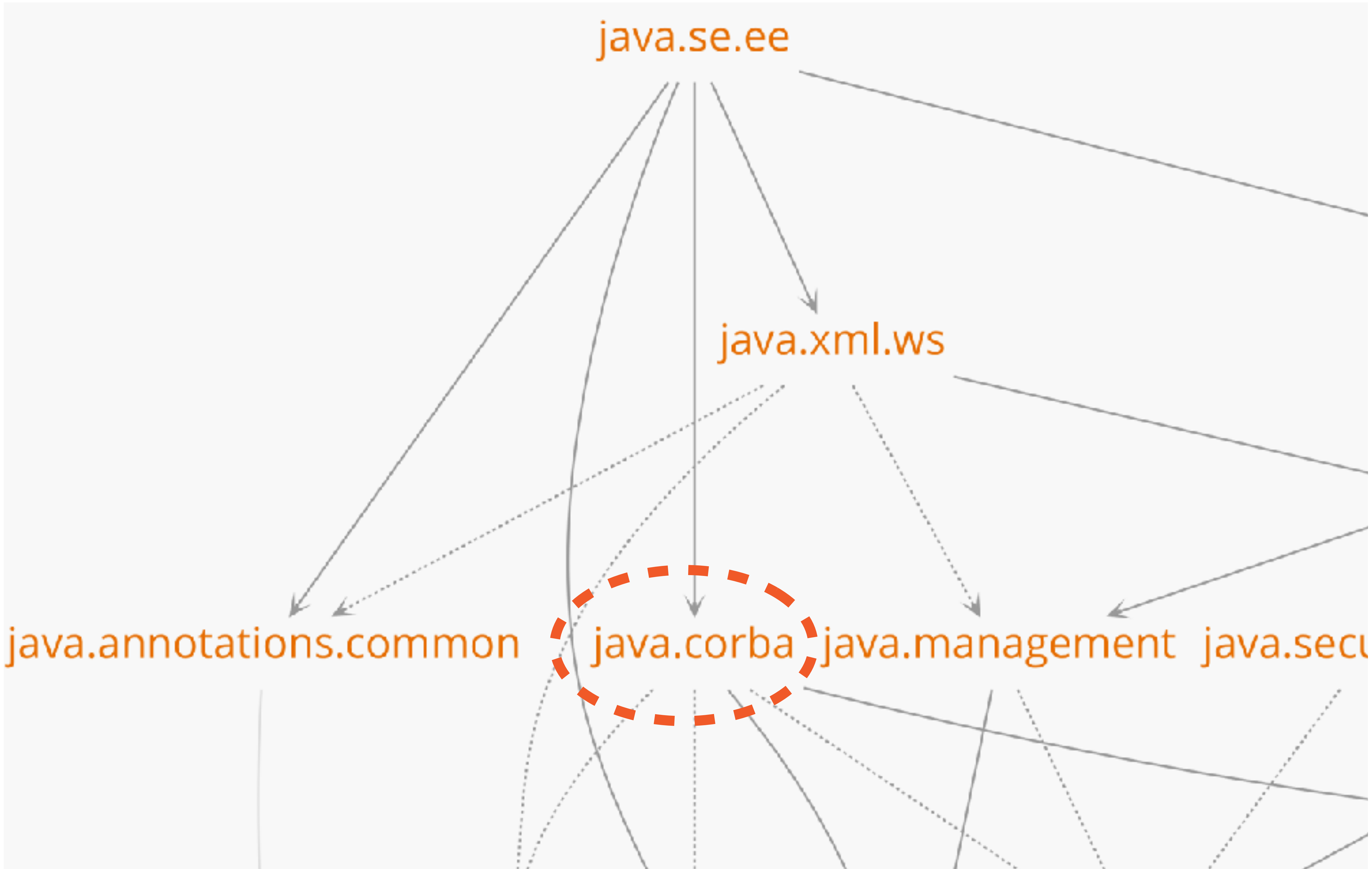
The Modular JDK: Explicit Dependencies



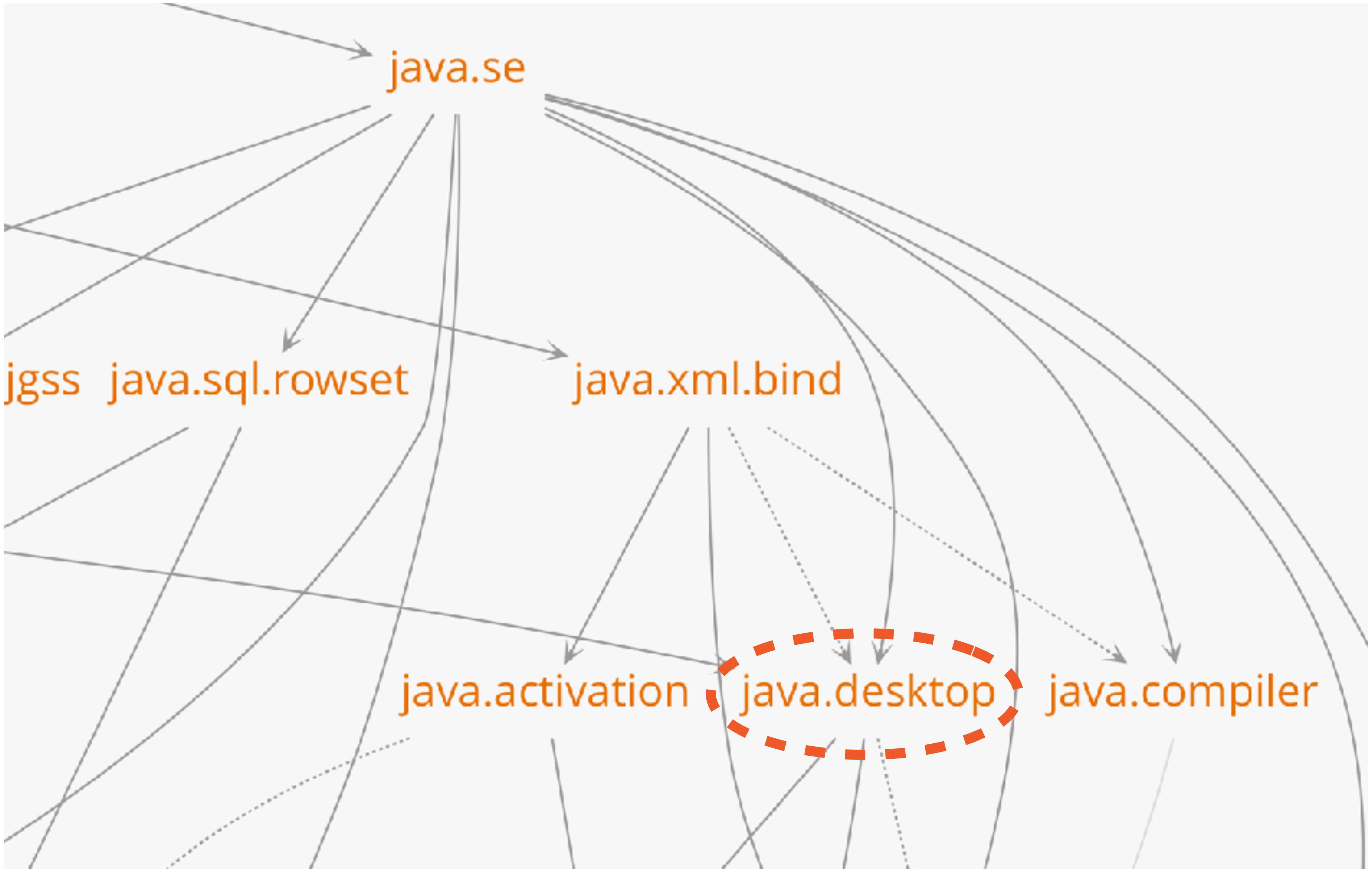
The Modular JDK: Explicit Dependencies



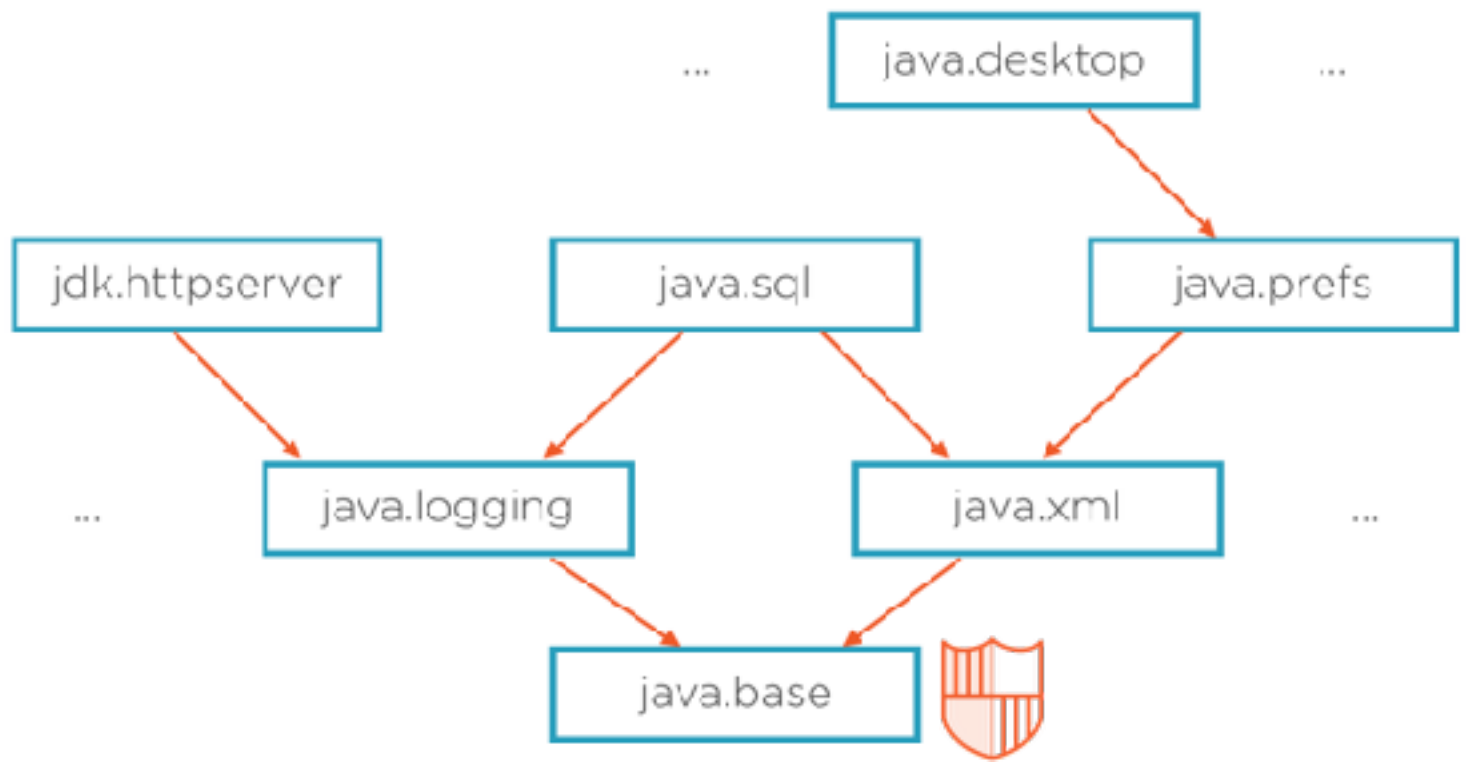
The Modular JDK: Explicit Dependencies



The Modular JDK: Explicit Dependencies

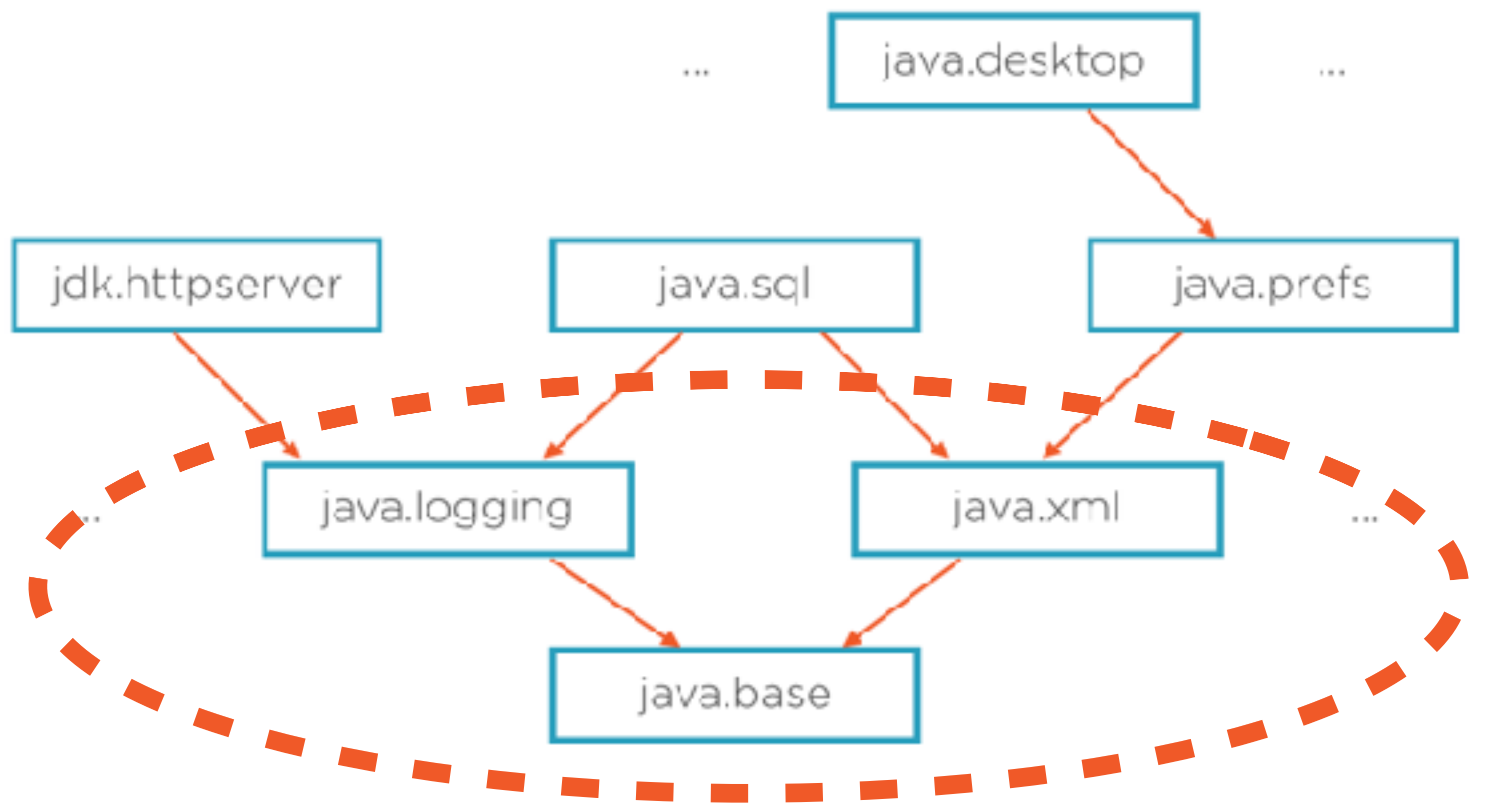


The Modular JDK: Advantages



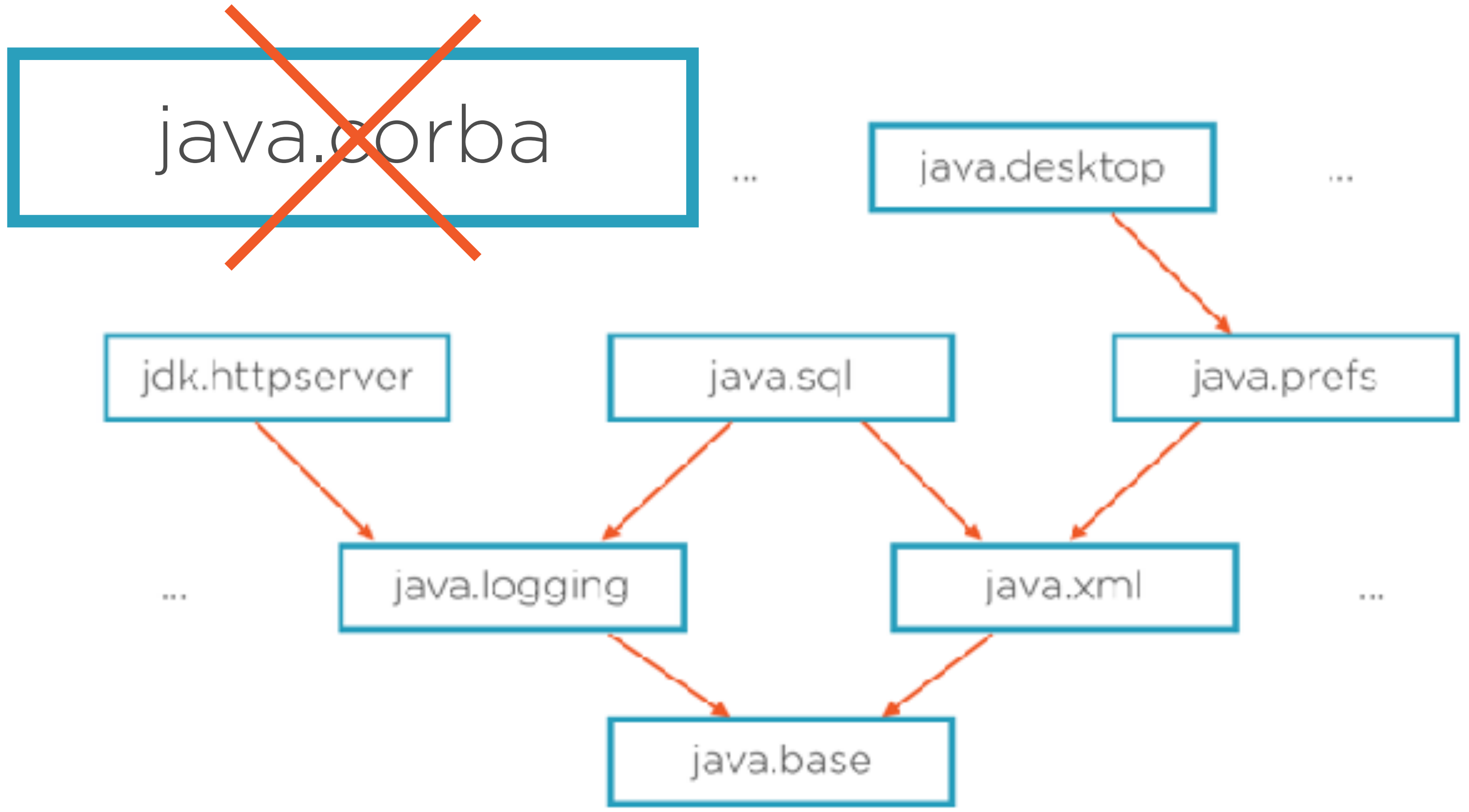
Increased security

The Modular JDK: Advantages



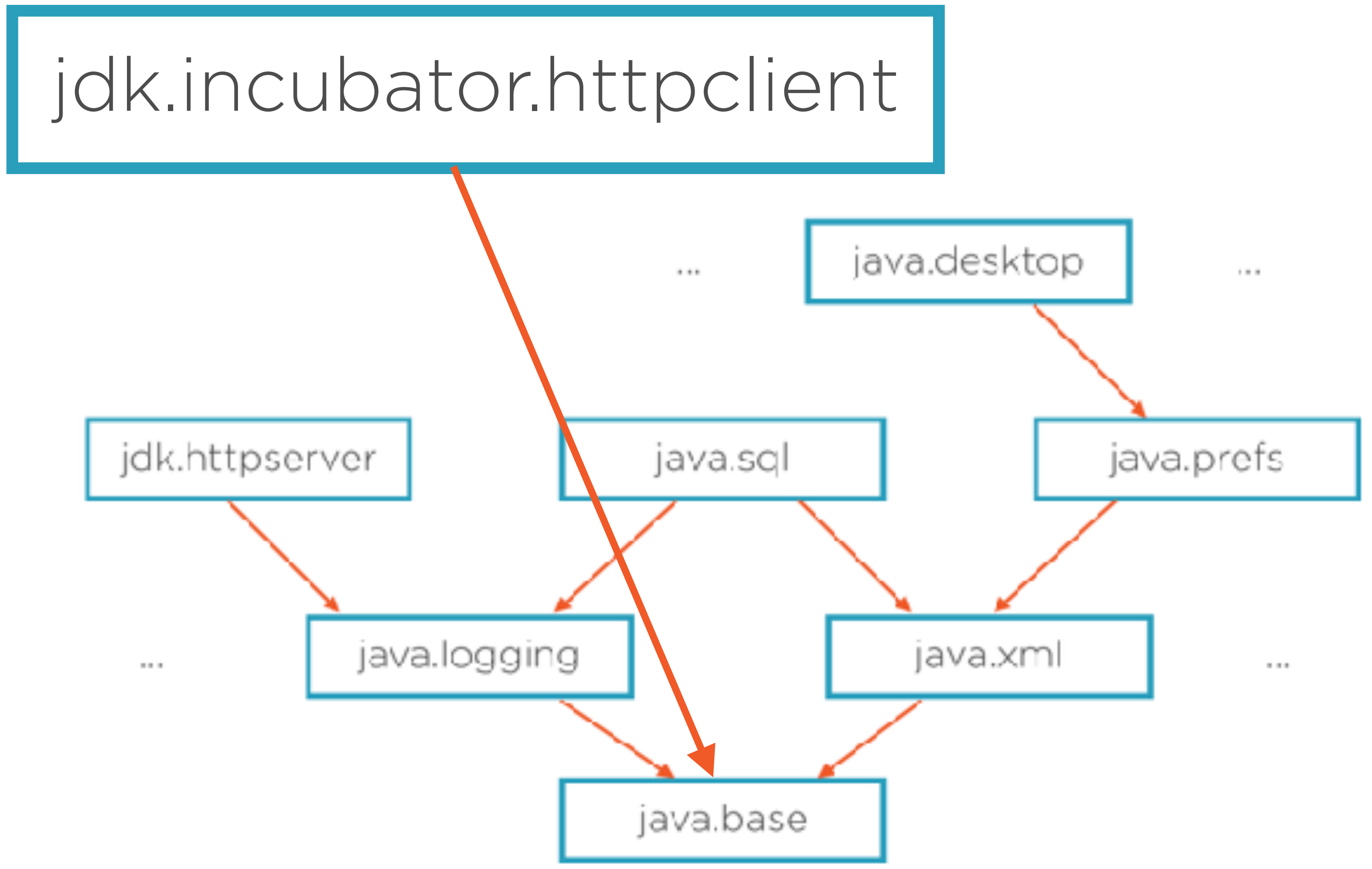
Reduced footprint

The Modular JDK: Advantages



Easy deprecation

The Modular JDK: Advantages

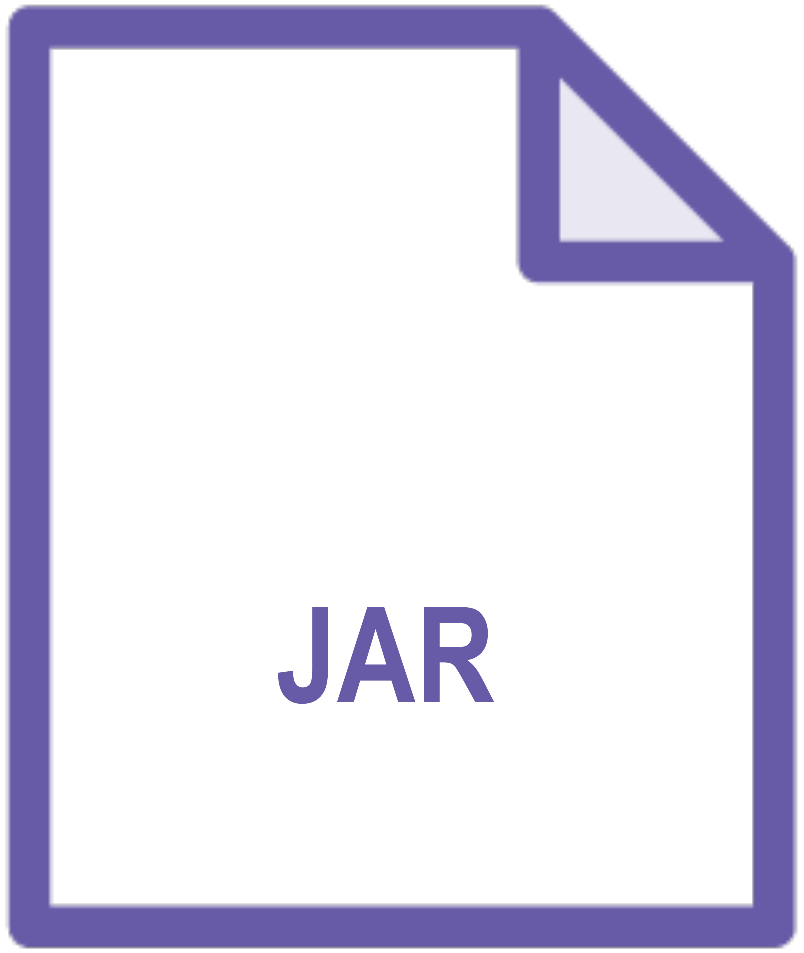


Future-proof

What Is a Module?

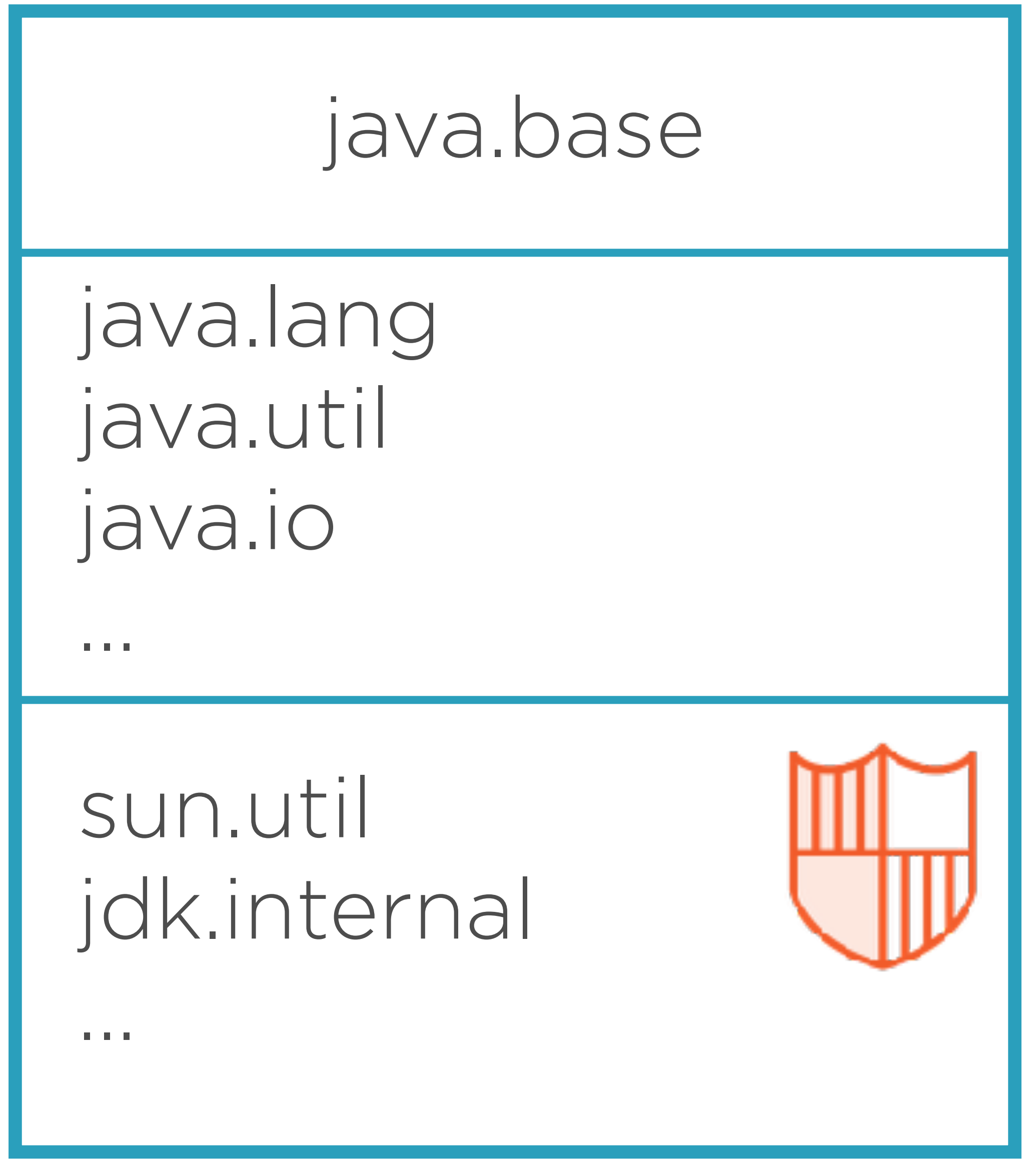
A module has a **name**, it **groups** related code and is **self-contained**

Are JAR Files Modules?



- ✓ They have a (file)name
- ✓ JARs group related code
- ⊘ No explicit dependencies
- ⊘ Weak boundaries

The Modular JDK: Encapsulation



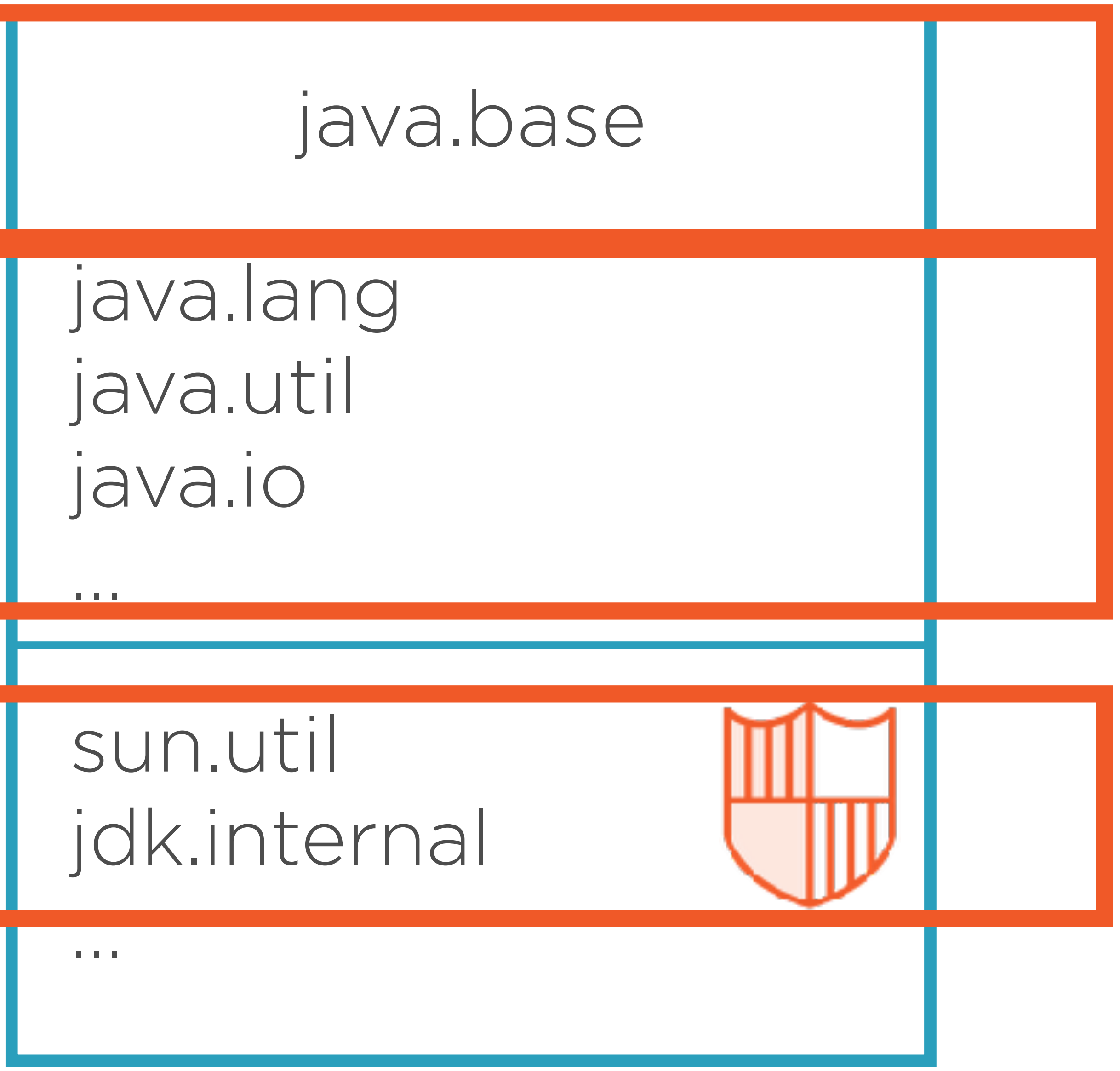
module-info.java

```
module java.base {

  exports java.lang;
  exports java.util;
  exports java.io;
  // and more

}
```

Module Descriptors



module-info.java

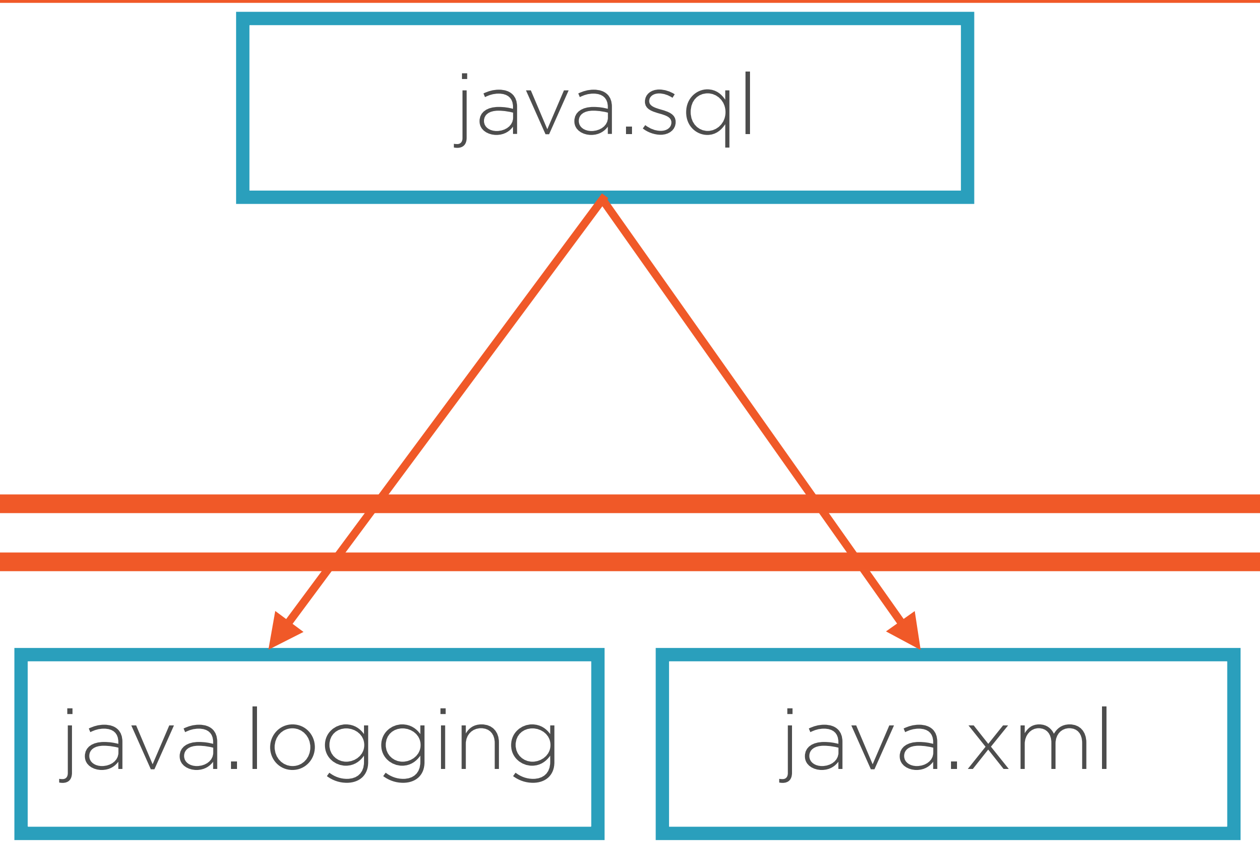
Module Descriptors

```
module java.sql {

  exports java.sql;
  exports javax.sql;
  exports
    javax.transaction.xa;

  requires java.logging;
  requires java.xml;

}
```



Demo: Modules and Descriptors in the JDK

List all modules

Demo: Modules and Descriptors in the JDK

Inspect module definitions

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 …

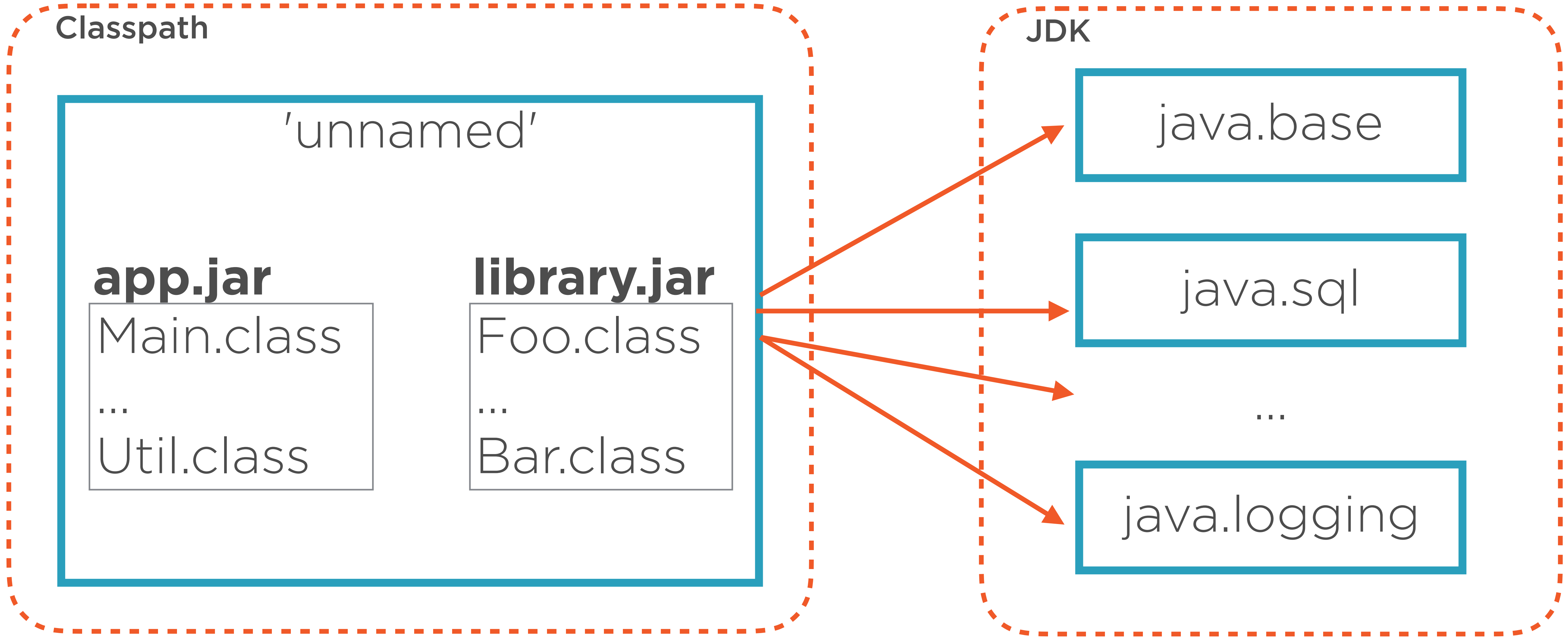
Migrating a Classpath-based Application

Unless ...

1. **You use JDK types that have been encapsulated**

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

Using Encapsulated Types



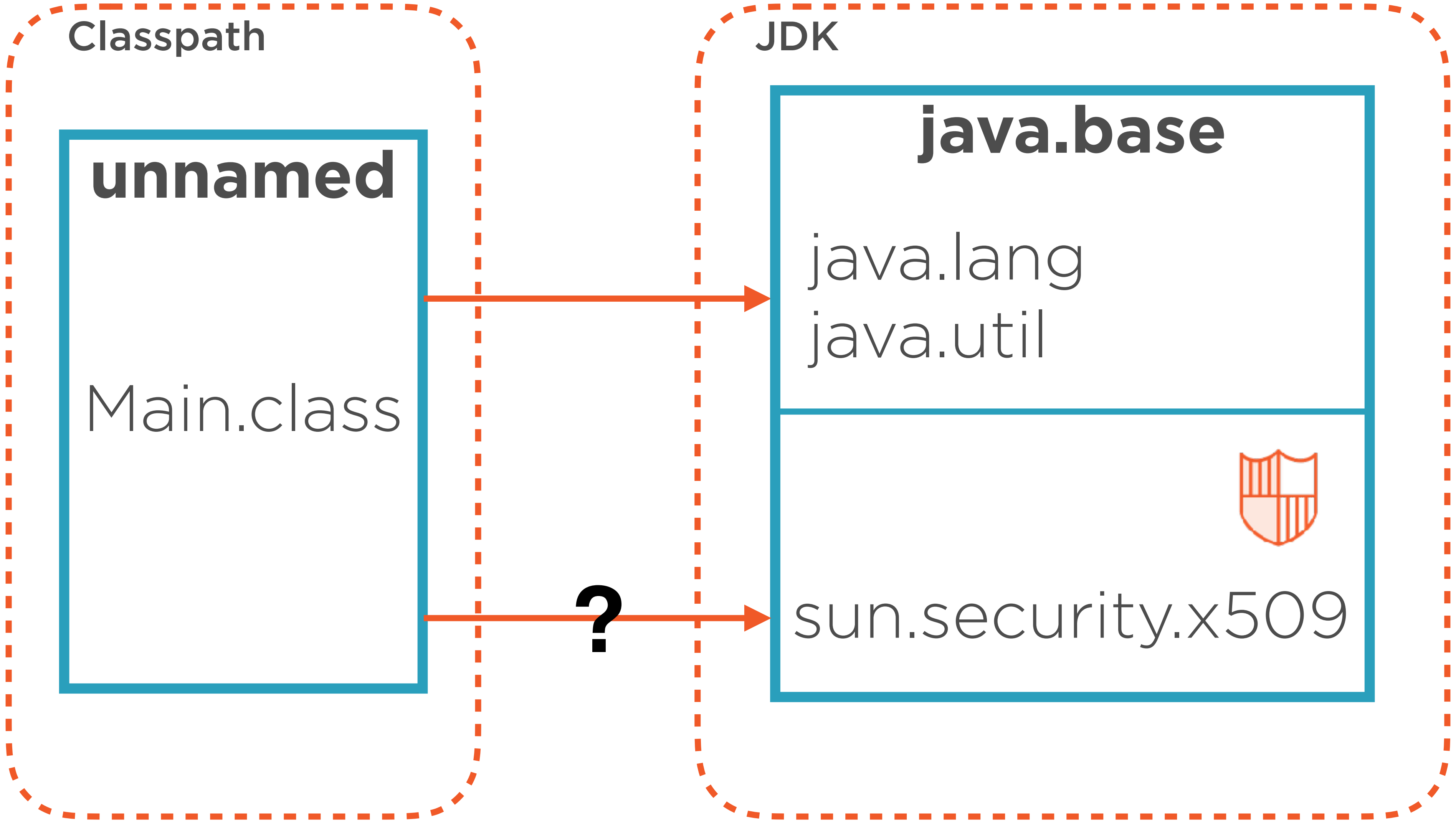
```
import sun.security.x509.X500Name;

public class Main {

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

}
```

Using Encapsulated Types



```java
import sun.security.x509.X500Name;

public class Main {

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

}
```

```
$ /java1.8/bin/javac Main.java
  Main.java:1: warning: X500Name is internal proprietary API and may be
  removed in a future release
  import sun.security.x509.X500Name;
                ^
$ /java9/bin/java Main
```

```java
import sun.security.x509.X500Name;

public class Main {

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

}
```

```
$ /java9/bin/javac Main.java

Main.java:1: error: package sun.security.x509 is not visible
import sun.security.x509.X500Name;
                  ^
  (package sun.security.x509 is declared in module java.base, which
    does not export it to the unnamed module)
1 error
```

```java
import sun.security.x509.X500Name;

public class Main {

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

}
```

```
$ /java1.8/bin/javac Main.java
$ /java9/bin/java --illegal-access=deny Main
```

```
Exception in thread "main" java.lang.IllegalAccessError: class Main (in
unnamed module @0x7b3300e5) cannot access class
sun.security.x509.X500Name (in module java.base) because module
java.base does not export sun.security.x509 to unnamed module
@0x7b3300e5
```

Using Encapsulated Types

What if I <u>can't</u> change the code?

```
java --illegal-access=permit Main
```

Breaks all strong encapsulation guarantees!

Logs warnings for each reflective illegal access

To be removed in future Java release

Using Encapsulated Types

What if I <u>don't want</u> 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
```

No warnings

Using Jdeps

```
$ jdeps -jdkinternals Main.class

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

Warning: JDK internal APIs are unsupported and private to JDK implementation that are
subject to be removed or changed incompatibly and could break your application.
Please modify your code to eliminate dependence on any JDK internal APIs.
For the most recent update on JDK internal API replacements, please check:
https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool

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

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**

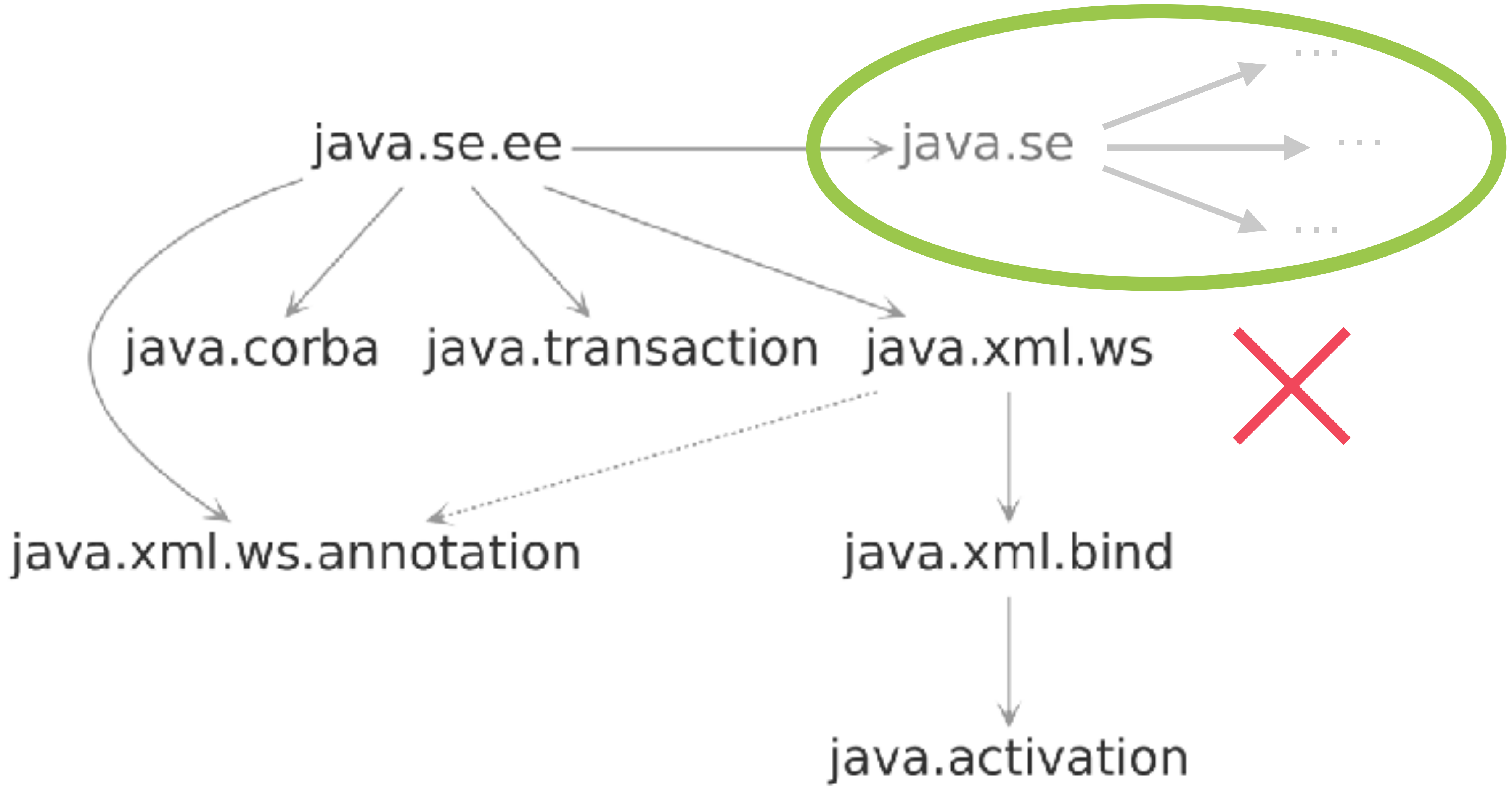
```java
import javax.xml.bind.DatatypeConverter;

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.xml.bind

java.xml.ws

java.xml.ws.annotation

java.corba

java.activation

Default:

Only modules reachable from java.se accessible

Using Non-default Modules

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

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

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

Using Jdeps

Find usage with jdeps

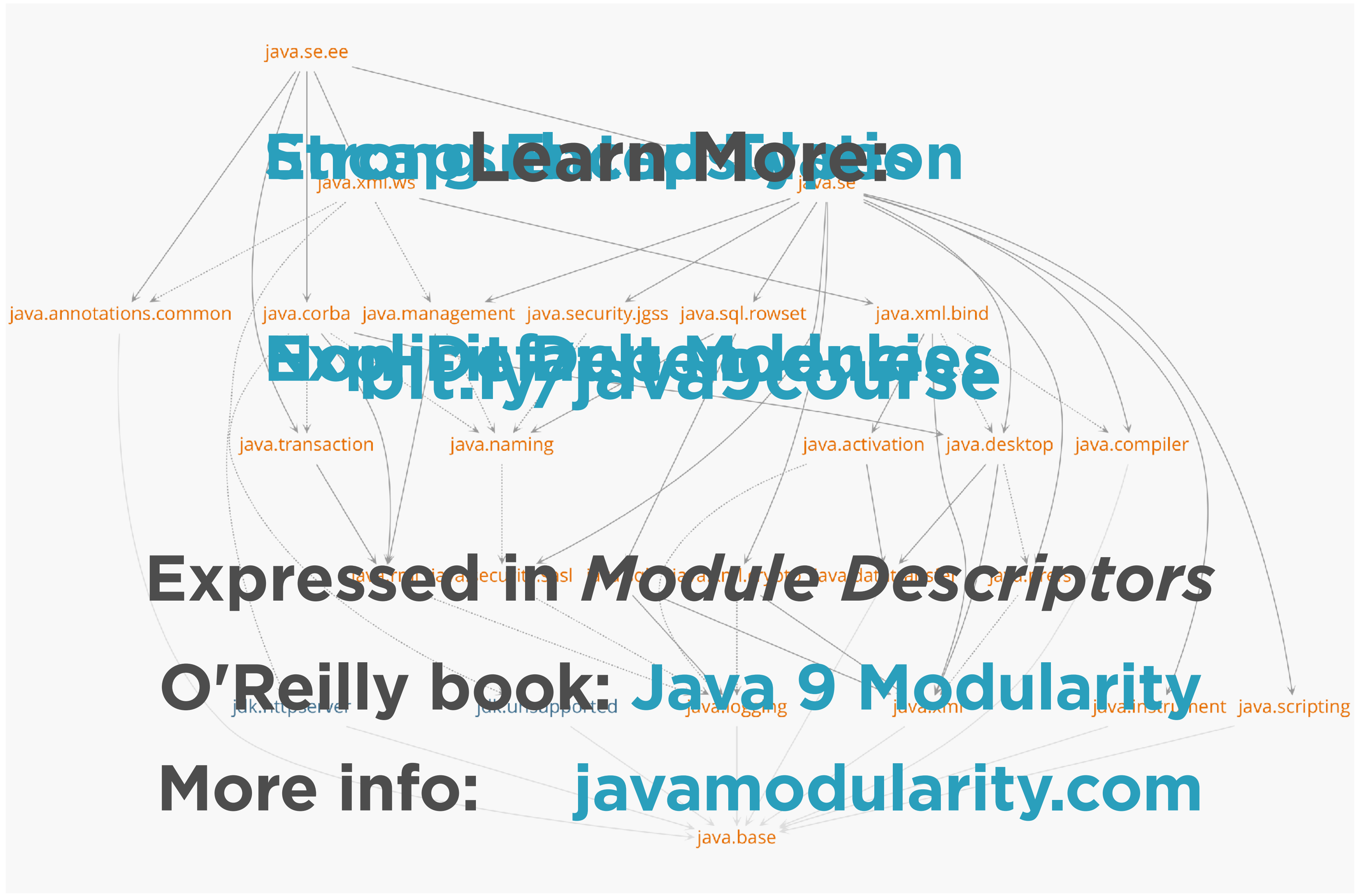
```
$ jdeps Main.class
Main.class -> java.base
Main.class -> not found
  <unnamed>       -> java.lang        java.base
  <unnamed>       -> javax.xml.bind   not found
```

Summary

Java Platform Module System

Migrating Classpath-based Applications

Modular JDK



Strong Encapsulation

Learn More:

Explicit Dependencies

bit.ly/java9course

Expressed in *Module Descriptors*

O'Reilly book: Java 9 Modularity

More info: javamodularity.com