Apache Maven supports ALL Java

ROBERT SCHOLTE @RFSCHOLTE

The good news

Apache Maven runs fine on JDK 9 (and 10)

Apache Maven works like heaven on JDK 11 (and 12-ea)

Possible issues are often plugin related

Upgrading to the latest version should solve the problem

Applying a module descriptor

1. Just add your module-info.java to src/main/java
2. Upgrade maven-compiler-plugin to at least 3.8.0
3. There’s no step three

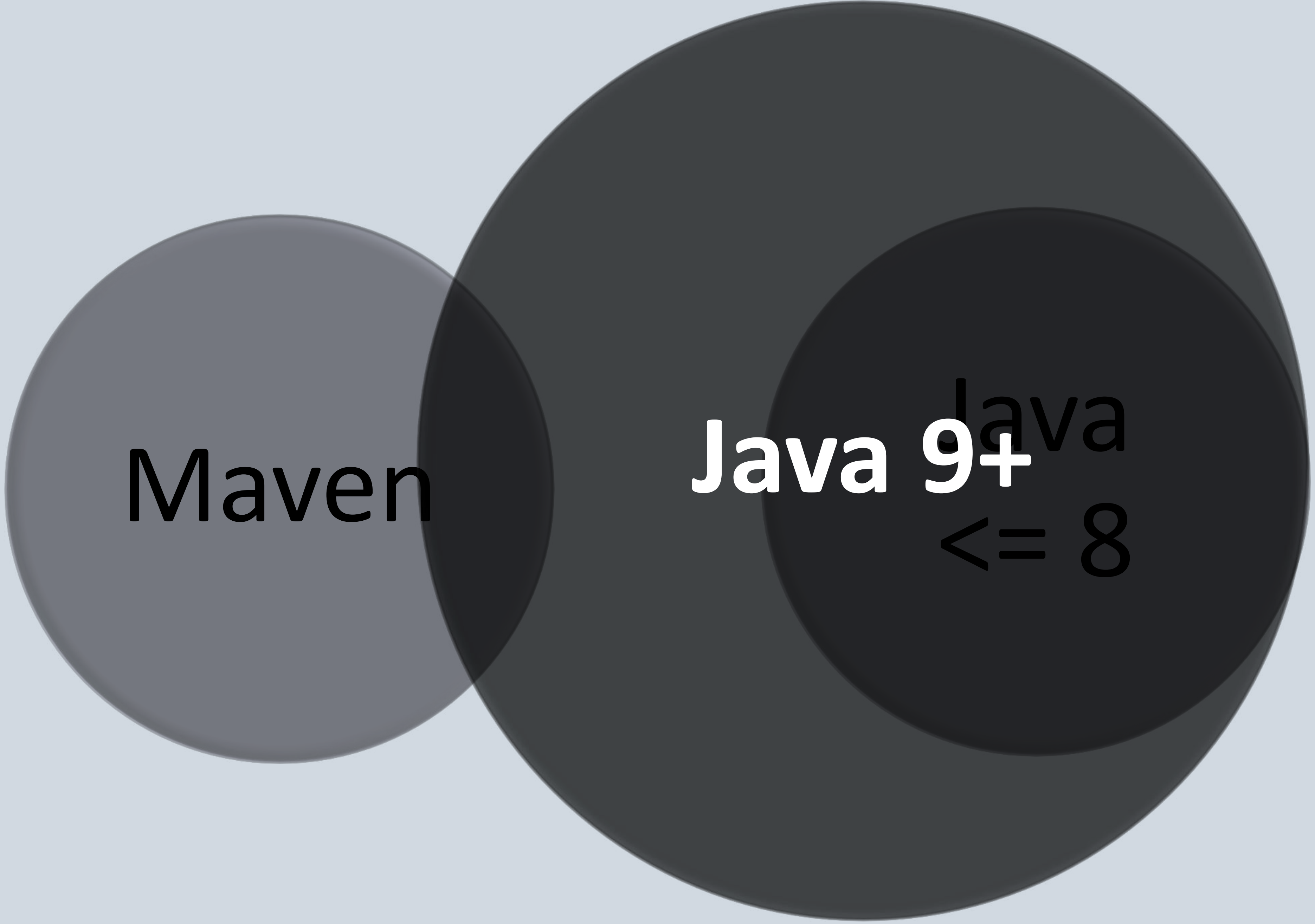
Maven will calculate which dependencies belong to the classpath and which to the module path.

No new dependency-element/scope

When two worlds collide…

Maven

Java
<= 8


Maven
Java 9+
Java
<= 8

Goals Java Platform Modular System

Reliable configuration

Strong encapsulation

The module-info.java

```
module com.foo.module.name
{
        // reliable configuration
        requires some.other.module;

        // strong encapsulation
        exports com.foo.package.name;
}
```

If you’re not careful with your modularization you can corrupt the whole Maven Ecosystem

Library builders should be aware of the impact of their module descriptors

Application builders should recognize these issues

Revised specifications

- Jars on modulepath
- Support ALL-MODULE-PATH
- Modulenames with numbers
- Automatic module names

Jars on modulepath

Original spec only allowed directories

Maven dependencies point to a file

Directory may contain multiple jars (javadoc,sources)

ALL-MODULE-PATH

--add-modules ALL-MODULE-PATH

In short: Make all entries on the module path see each other

Module names

Déjà Vu

What is the proper module name?

What is the proper groupId and artifactId?

Modulenames with numbers

Close to 30.000 groupId/artifactId end with a number (Central, March 2017)

Project/product names	Versioned libraries (includes versioned packages)	Bridge libraries to different versions
• AWS-EC2 • AWS-Route53 • AWS-S3 • C3P0 • DB2 • Fabric8 • H2 • JSR*nnn* • *OAuth2*	• Commons-lang2 • Commons-lang3	• Jspc-compiler-tomcat*N* • Mockwire-spring*N* • Surefire-junit*N*

Original implementation did not allow module names ending with a number.

#VersionsInModuleNames

Some have argued that library maintainers will be tempted to encode major version numbers, or even full version numbers, in module names. Is there some way we can guide people away from doing that?

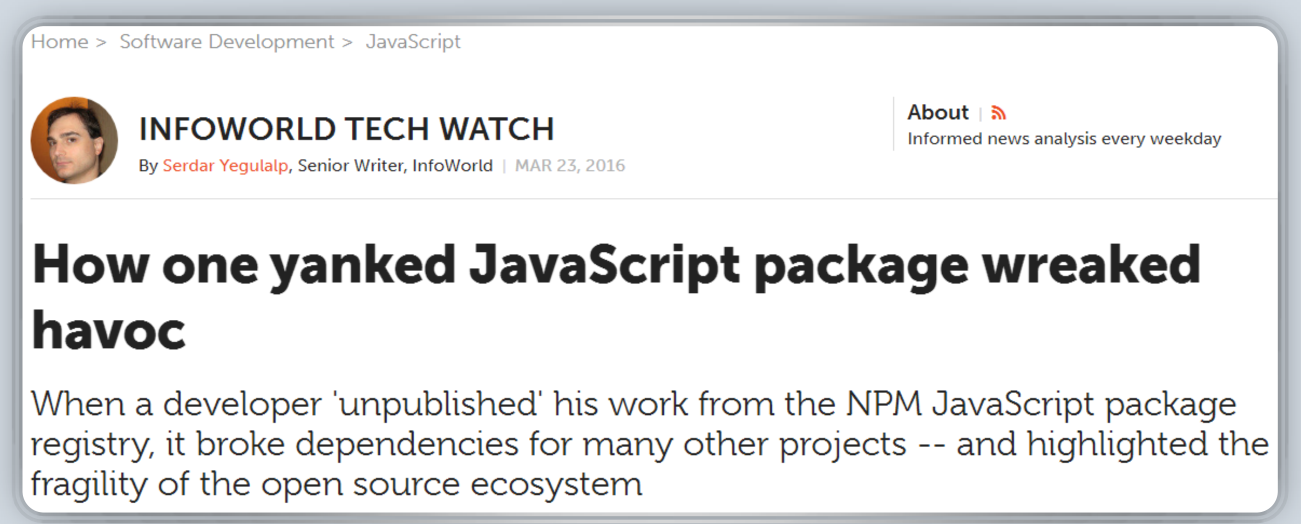
Resolution Abandon the previous proposal to mandate that module names appearing in source-form module declarations must both start and end with “Java letters”. Revise the automatic-module naming algorithm to allow digits at the end of module names.

Module names must be as unique as the coordinates of dependencies

Automatic module names

“… . The module name is otherwise derived from the name of the JAR file.”

NPM Javascript package registry



Home > Software Development > JavaScript

INFOWORLD TECH WATCH

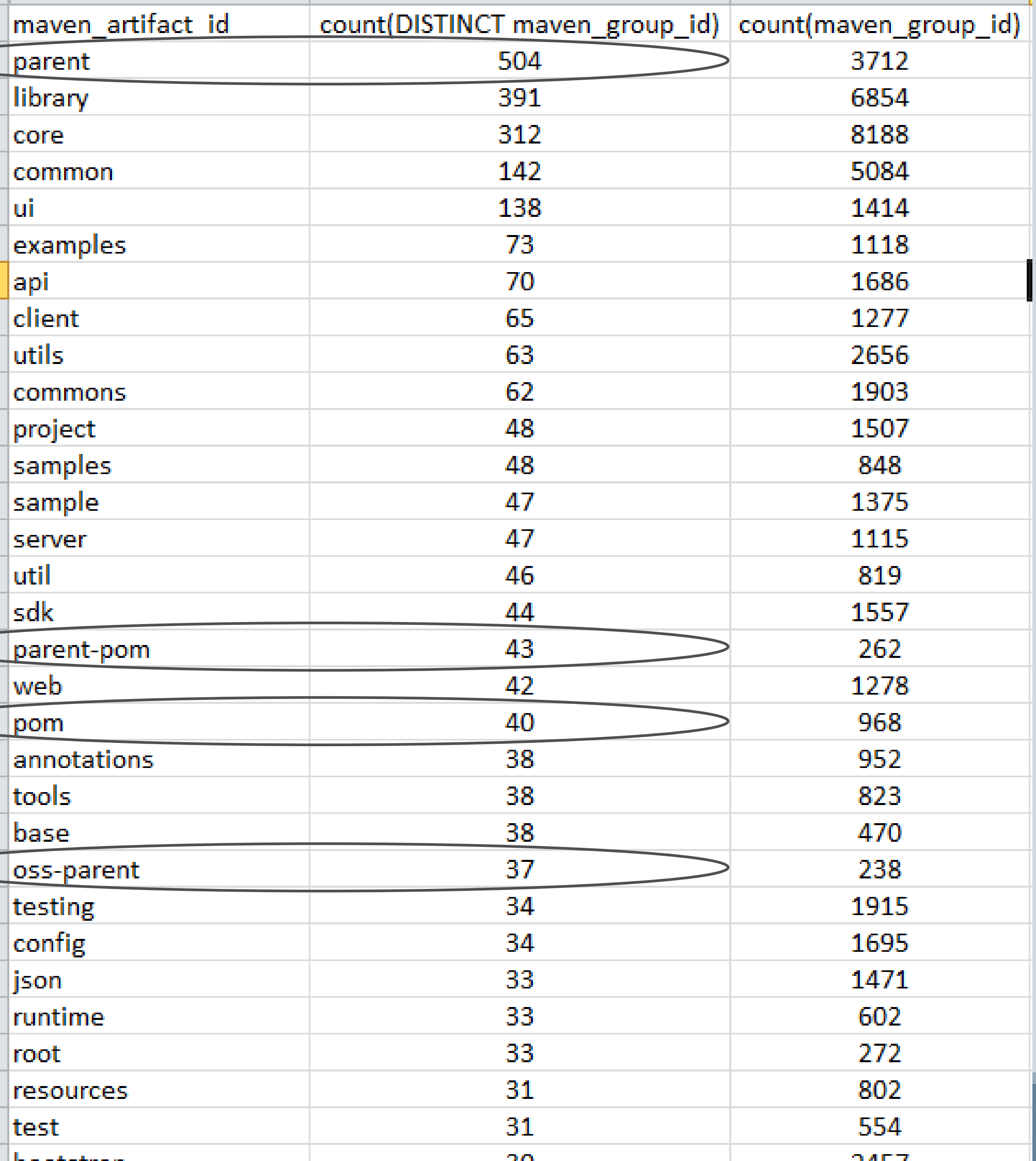
By Serdar Yegulalp, Senior Writer, InfoWorld | MAR 23, 2016

About |

Informed news analysis every weekday

How one yanked JavaScript package wreaked havoc

When a developer 'unpublished' his work from the NPM JavaScript package registry, it broke dependencies for many other projects -- and highlighted the fragility of the open source ecosystem



maven_artifact_id	count(DISTINCT maven_group_id)	count(maven_group_id)
parent	504	3712
library	391	6854
core	312	8188
common	142	5084
ui	138	1414
examples	73	1118
api	70	1686
client	65	1277
utils	63	2656
commons	62	1903
project	48	1507
samples	48	848
sample	47	1375
server	47	1115
util	46	819
sdk	44	1557
parent-pom	43	262
web	42	1278
pom	40	968
annotations	38	952
tools	38	823
base	38	470
oss-parent	37	238
testing	34	1915
config	34	1695
json	33	1471
runtime	33	602
root	33	272
resources	31	802
test	31	554

Evidence (OCT 2016)

Over 13500 ‘rows’ of collisions

JIGSAW

Automatic modules are required for top-down adoption

MAVEN

References to automatic module names will cause collisions sooner or later

Library builders should never refer to automatic modules* and deploy to a public repository.
Application builders can choose to refer to automatic modules.

* Filename based

Application versus library

Application

Module descriptor without exports

maven-compiler-plugin logs info message in case of automatic module usage

Library

Module descriptor with exports

maven-compiler-plugin logs **WARNING** message in case of automatic module usage

Tip: Use Maven 3.5.0+ for colour support

Automatic modules

Ease of top-down migration for application builders

But what about “in the middle” *library* builders?

Java Platform. It will be approachable, *i.e.*, easy to learn and easy to use, so that developers can use it to construct and maintain libraries and large applications for both the Java SE and Java EE Platforms.

JSR 376: Java™ Platform Module System

Original Java Specification Request (JSR)

Section 2.1

Conference example

Application	**my-app**			
Libraries	jackson-core	jackson-databind	jackson-annotations	my-lib
java.base				

More realworld example

Application	**my-app**			
Libraries (direct deps)	…	…	…	my-lib
Libraries (transitive deps) …	…	…	…	
Libraries (independent deps)	jackson-core	jackson-databind	jackson-annotations	
java.base				

currency-1.0.jar

1

```
buy-service-1.0.jar

module org.moneylibs.buy
{
  requires currency;
}
```

2

3

```
currence-2.0.jar

module org.moneylibs.currency
{

}
```

```
sell-service-1.0.jar

module org.moneylibs.sell
{
  requires org.moneylibs.currency;
}
```

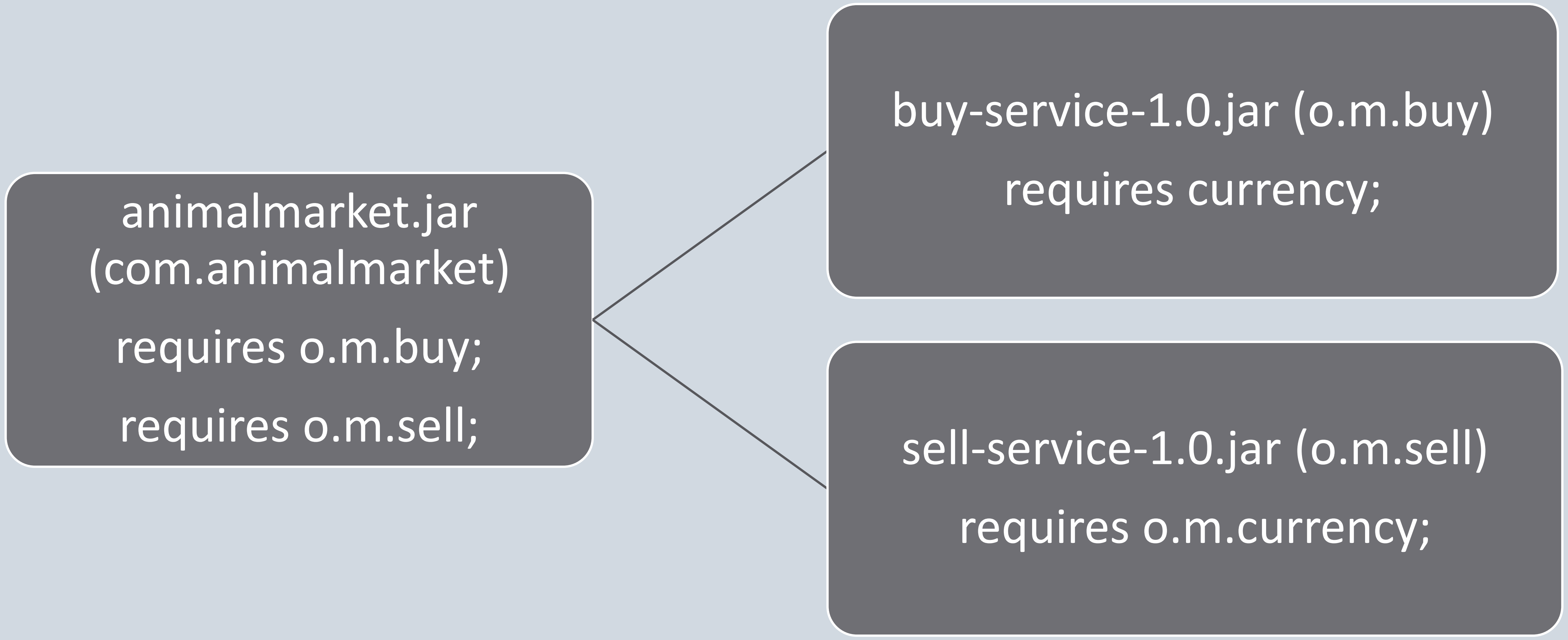
4

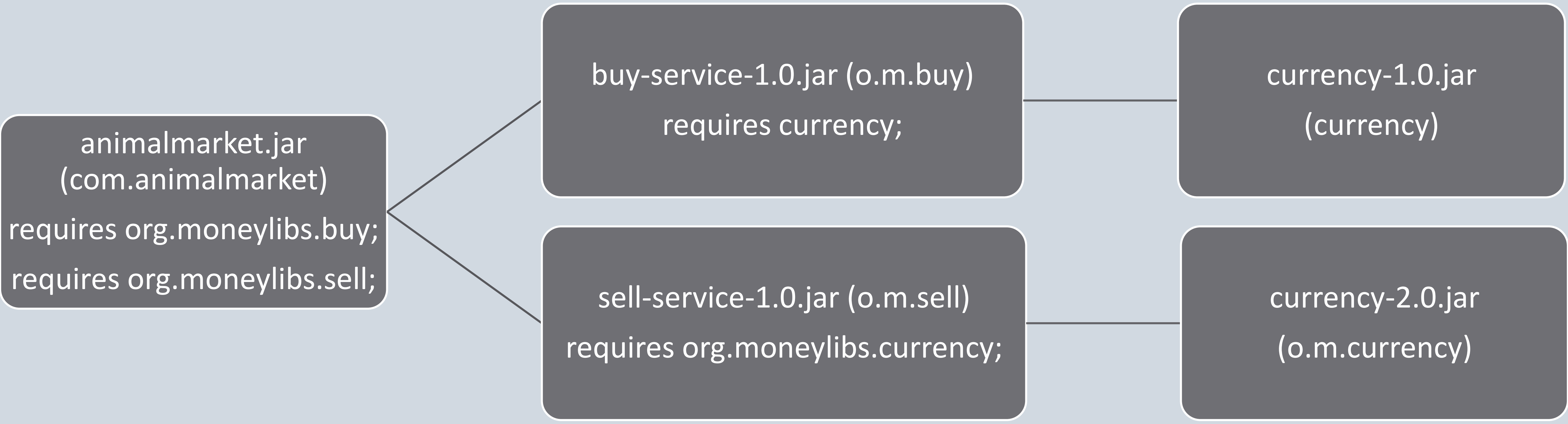
```
animalmarket-1.0.jar

module com.animalmarket
{
  requires org.moneylibs.buy;
  requires org.moneylibs.sell;
}
```

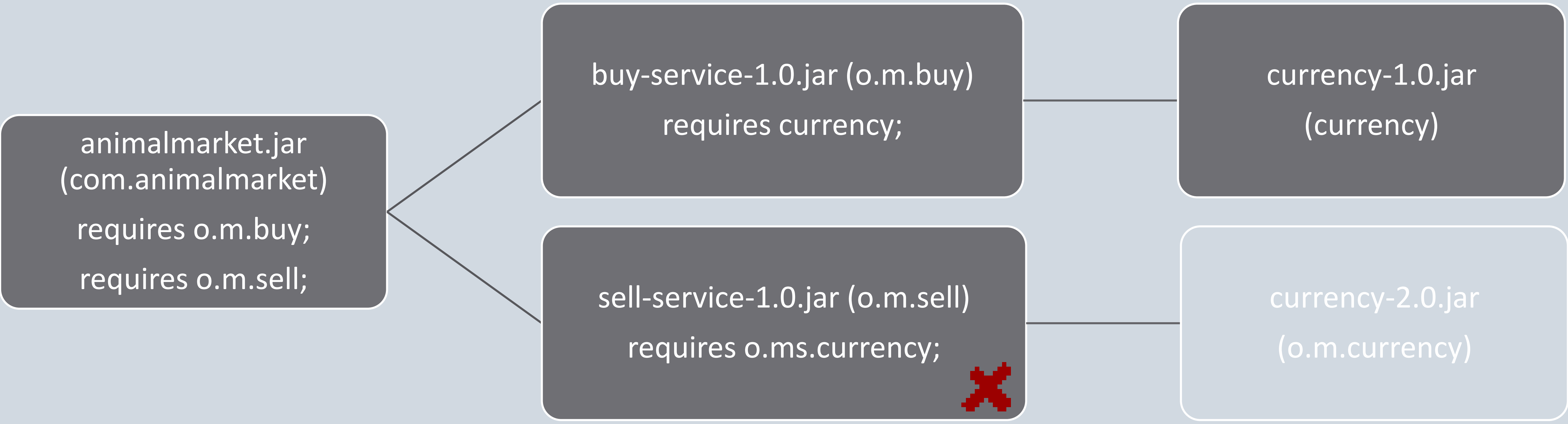
5

```xml
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.animalmarket</groupId>
  <artifactId>animalmarket</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <dependencies>
        <!-- the dependencies -->
  </dependencies>
</project>
```

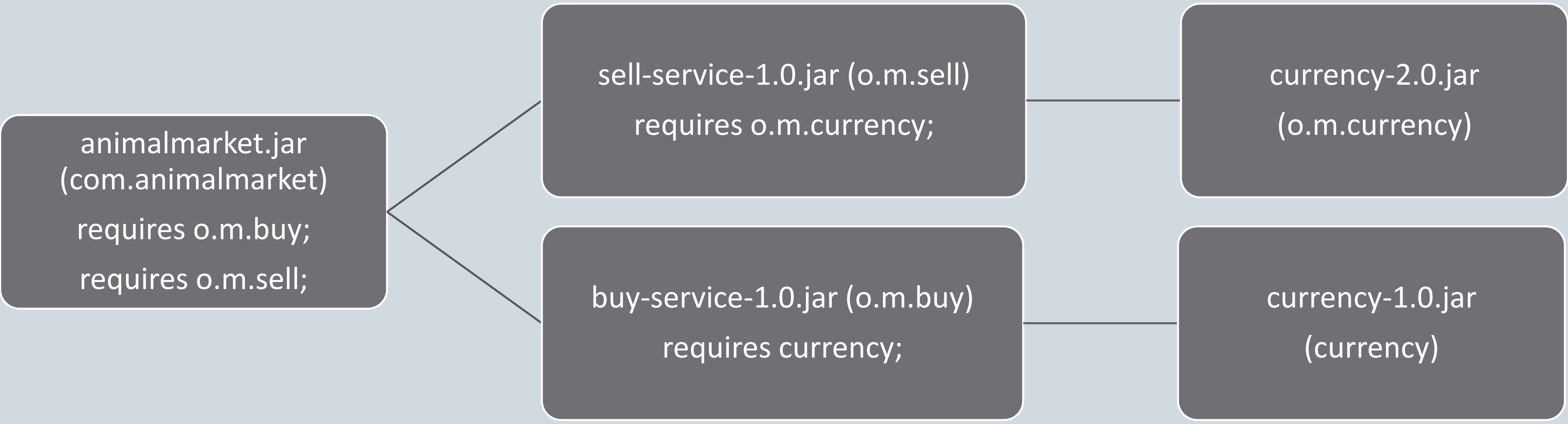
```xml
<dependency>
  <groupId>org.moneylibs</groupId>
  <artifactId>buy-service</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>org.moneylibs</groupId>
  <artifactId>sell-service</artifactId>
  <version>1.0</version>
</dependency>
```

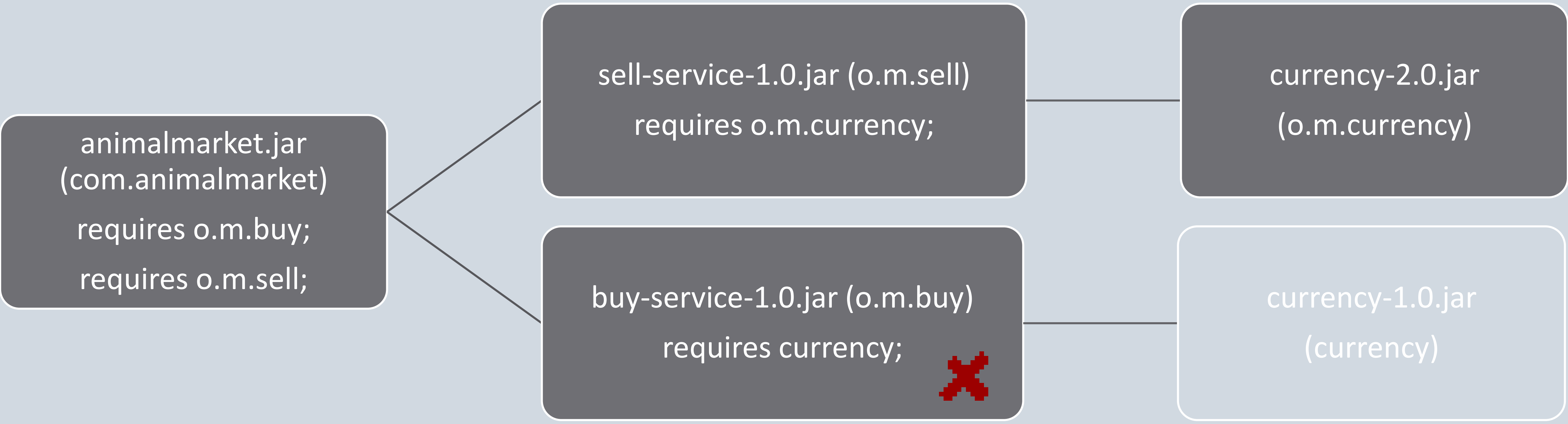

animalmarket.jar (com.animalmarket)
requires o.m.buy;
requires o.m.sell;
buy-service-1.0.jar (o.m.buy)
requires currency;
sell-service-1.0.jar (o.m.sell)
requires o.m.currency;


animalmarket.jar
(com.animalmarket)
requires org.moneylibs.buy;
requires org.moneylibs.sell;
buy-service-1.0.jar (o.m.buy)
requires currency;
sell-service-1.0.jar (o.m.sell)
requires org.moneylibs.currency;
currency-1.0.jar
(currency)
currency-2.0.jar
(o.m.currency)


animalmarket.jar (com.animalmarket)
requires o.m.buy;
requires o.m.sell;
buy-service-1.0.jar (o.m.buy)
requires currency;
sell-service-1.0.jar (o.m.sell)
requires o.ms.currency;
currency-1.0.jar (currency)
currency-2.0.jar (o.m.currency)

```xml
<dependency>
  <groupId>org.moneylibs</groupId>
  <artifactId>buy-service</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>org.moneylibs</groupId>
  <artifactId>sell-service</artifactId>
  <version>1.0</version>
</dependency>
```


animalmarket.jar (com.animalmarket)
requires o.m.buy;
requires o.m.sell;
sell-service-1.0.jar (o.m.sell)
requires o.m.currency;
currency-2.0.jar
(o.m.currency)
buy-service-1.0.jar (o.m.buy)
requires currency;
currency-1.0.jar
(currency)


animalmarket.jar (com.animalmarket)
requires o.m.buy;
requires o.m.sell;
sell-service-1.0.jar (o.m.sell)
requires o.m.currency;
currency-2.0.jar (o.m.currency)
buy-service-1.0.jar (o.m.buy)
requires currency;
currency-1.0.jar (currency)

Migrate

META-INF/MANIFEST.MF

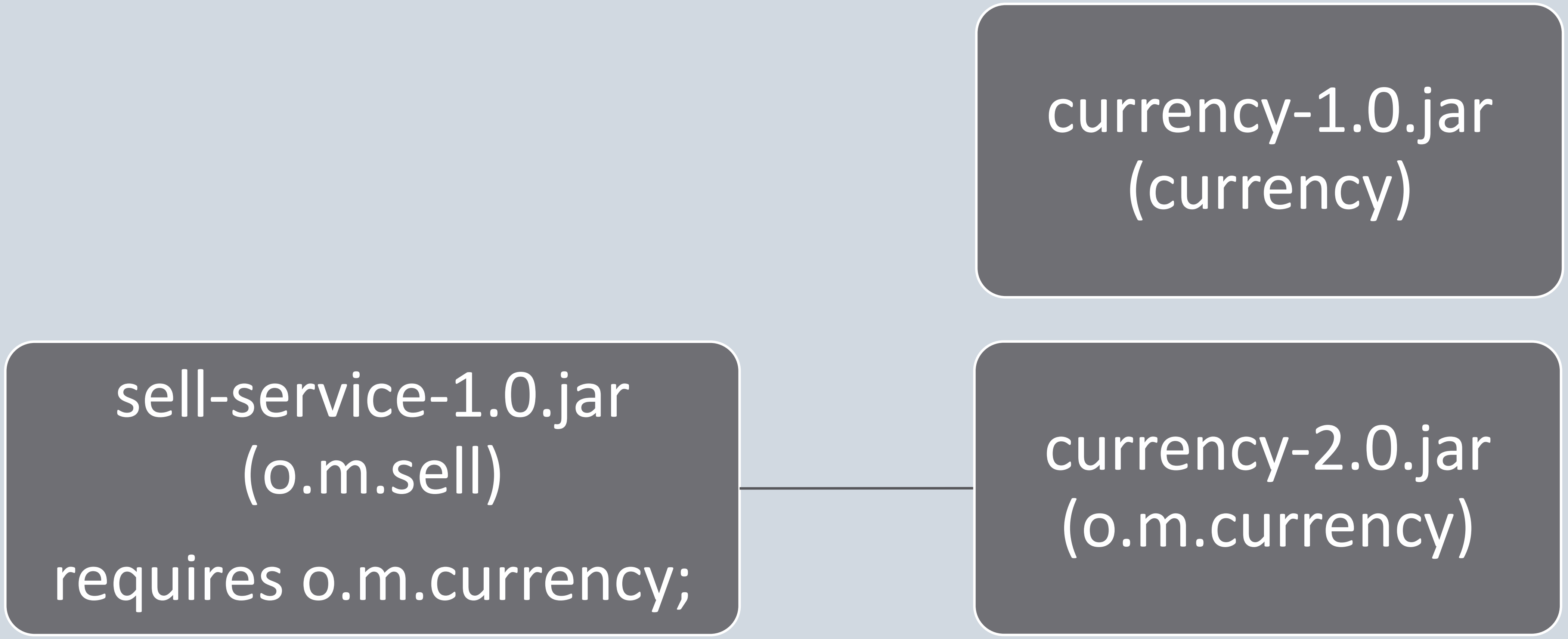
- Automatic-Module-Name

2

```
buy-service-1.0.jar

META-INF/MANIFEST.MF
Automatic-Module-Name:
org.moneylibs.buy
```

Point of no return

“If you refer to dependency X:N as a module and dependency X:N-1 has no module name, then you cannot downgrade this dependency anymore”


currency-1.0.jar
(currency)
sell-service-1.0.jar
(o.m.sell)
requires o.m.currency;
currency-2.0.jar
(o.m.currency)

Strong advices

- Project must be Java 9 ready!!
  - No split packages
  - No root classes
- For libraries that depends on at least one filename based automodule:
  - Help depending projects by providing intended module name via MANIFEST
- Pick your modulename with care, e.g. the shared package

Mistakes will happen

In fact, first invalid modules are already available at Maven Central

asm-6.0_BETA (org.ow2.asm)

asm-all-6.0_BETA (org.ow2.asm.all) [1]

asm-debug-all-6.0_BETA (org.ow2.asm.debug.all) [1]

Application developer cannot fix these mistakes (dependency-exclude doesn’t work)

[1] Fixed with asm-6.0 by dropping *-all artifacts (asm includes debug information by default)

Same packages must have the same modulename (otherwise potential split package issue)

Tips for using JAVA 9 with MAVEN

DON’T change your folder structure (no need for extra folder with module name)

Modulepath or classpath? No change to dependencies, just add module-info.java (Plexus-java can help plugins to build up the path)

Understanding plexus-java

Maven independent library for general Java features

- LocationManager
- Version

Used by

- maven-compiler-plugin
- maven-failsafe-plugin
- maven-javadoc-plugin
- maven-jlink-plugin
- maven-jmod-plugin
- maven-surefire-plugin
- …

Plexus Java :: LocationManager

If there’s a module descriptor, all its direct and indirect required modules will be put on the module path, the rest on the classpath

Most plugins show the paths as debug logging (-X / --debug)

Learn the JPMS specifications

JLINK

“You can use the jlink tool to assemble and optimize a set of modules and their dependencies into a custom runtime image”

Enhanced solution for fat executable jar

However… it is overrated

Only works with explicit modules!

Source / target 1.9

<release>9<release>

source/target <= 1.8 : animal-sniffer

Issues?

1) Stackoverflow
2) Apache Maven mailinglists
3) Apache Maven Jira in case of bugs / improvements

Frequently Asked Questions

Can every project become modular?

NO

-Java 9 is a gamechanger, it introduces new rules

-The older the project, the more likely it cannot follow these rules

-No worries, the classpath is still there and will stay!

The boomerang question

Or “the ever returning Maven/JavaNEXT/Conference question”

Will Maven generate the module descriptor?

No

Different purpose

- Pom is used to download jars and make them available
- Module descriptor is used to specify required modules

Not all modules are dependencies

- (e.g. java.logging, jdk.compiler)

Module descriptor elements not covered:

- Module name
- Open module
- Exported packages
- Uses / provides services

Pom 4.0.0 has no space for new elements

Pom hygiene

dependency:analyse

- Analyzes the dependencies of this project and determines which are:
  - used and declared (good)
  - used and undeclared (via transitive dependency)
  - unused and declared (ballast!)

Dependencies can be excluded,
required modules cannot

…BUT JDEPS can do it, right?

Can create a rough module descriptor

Intended to help with an initial descriptor

Uses binary classes, i.e. AFTER compile-phase

Some open source project will…

https://github.com/moditect/moditect

[RESULT] Apache Maven supports ALL Java

Up-for-grabs

~60-80% of Java Project/Developers use Maven

The Apache Maven Project holds ~95 (sub)projects

Maintained by ~5-10 active volunteers (No Company!)

Let’s restore the balance!

https://s.apache.org/up-for-grabs_maven

https://maven.apache.org/guides/development/guide-committer-school.html

Thank you

@ASFMAVENPROJECT