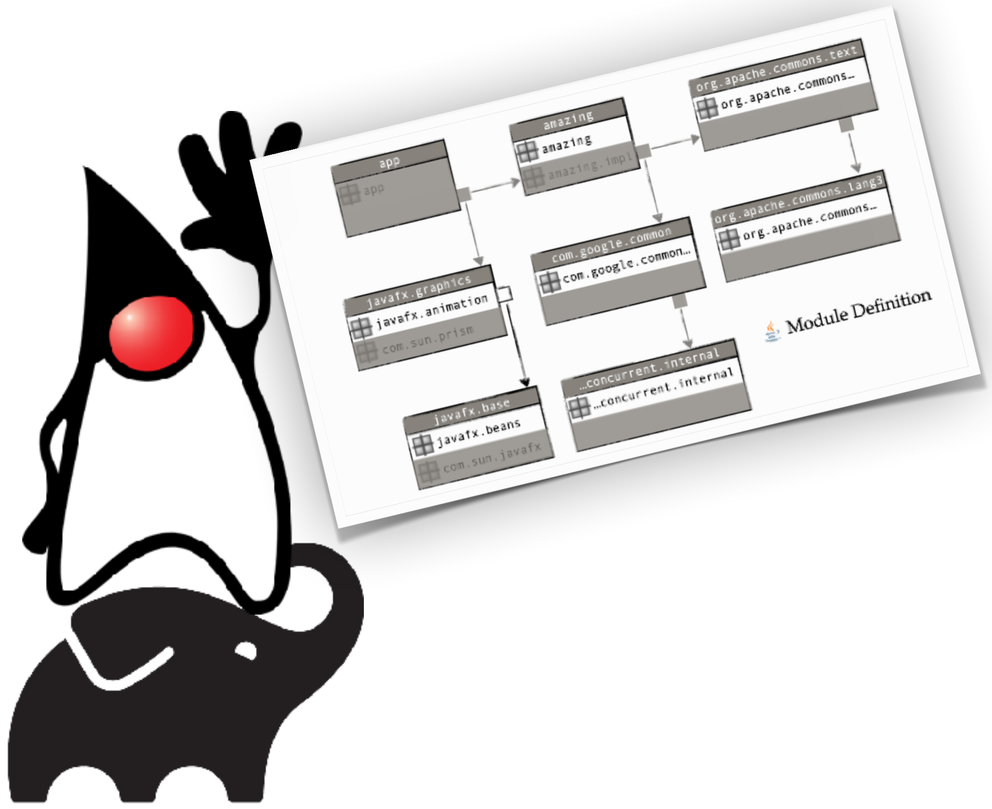
Combining Java Modules and Gradle for elegant project structures



Jendrik Johannes

github.com/jjohannes

2024

Problem

Accidental Complexity

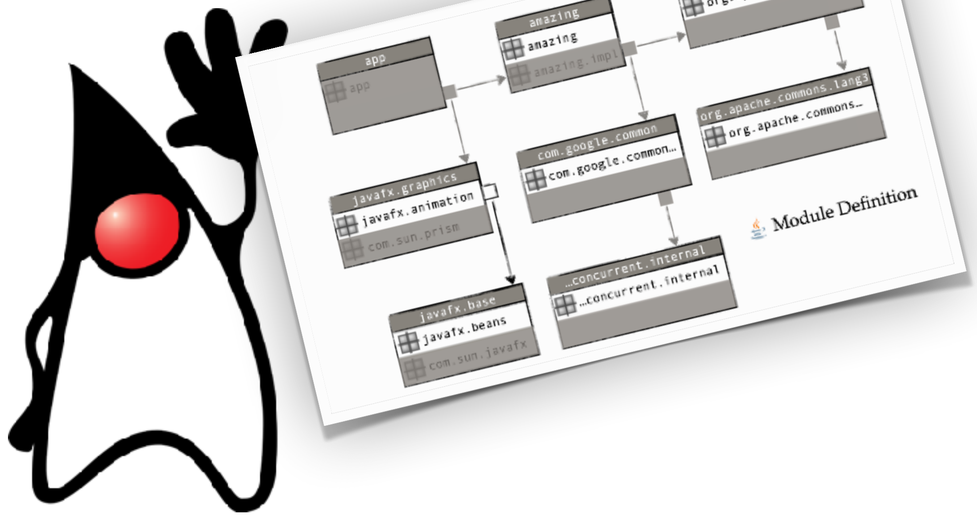
in

Build Configuration

and

Module Definition

(Architecture in Code)



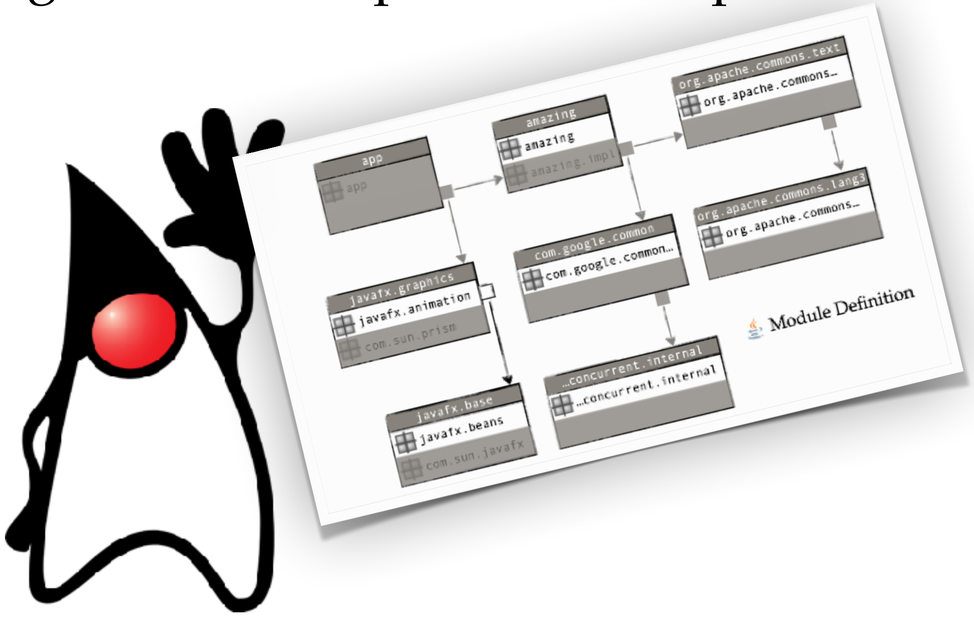
github.com/hibernate/hibernate-orm/blob/main/hibernate-core/hibernate-core.gradle

~300 Lines total
~20 Lines Module Definition

~300 Lines total
~100 Line Module Definition

```
29
30    dependencies {
31        api jakartaLibs.jpa
32        api jakartaLibs.jta
33
34        implementation libs.hcann
35        implementation libs.jandex
36        implementation libs.classmate
37        implementation libs.byteBuddy
38
39        implementation jakartaLibs.jaxbApi
40        implementation jakartaLibs.jaxb
41        implementation jakartaLibs.inject
42
43        implementation libs.antlrRuntime
44
```

github.com/apache/hadoop/blob/trunk/hadoop-common-project/hadoop-registry/pom.xml




```
29
30      <dependencies>
31
32        <dependency>
33          <groupId>org.slf4j</groupId>
34          <artifactId>slf4j-api</artifactId>
35        </dependency>
36
37        <dependency>
38          <groupId>org.apache.hadoop</groupId>
39          <artifactId>hadoop-auth</artifactId>
40        </dependency>
41
42        <dependency>
43          <groupId>org.apache.hadoop</groupId>
44          <artifactId>hadoop-annotations</artifactId>
45        </dependency>
46
```


Accidental Complexity

1. Mixing Module Definition and other concerns of the build

Essential Complexity

repo1.maven.org/maven2/org/apache/commons/commons-text/1.11.0/commons-text-1.11.0.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <scope>compile</scope>
  <version>3.13.0</version>
</dependency>
```

should be:

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <scope>runtime</scope>
  <version>3.13.0</version>
</dependency>
```


Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)

repo1.maven.org/maven2/org/apache/commons/commons-text/1.11.0/commons-text-1.11.0.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.13.0</version>
</dependency>
```

repo1.maven.org/maven2/org/apache/commons/commons-compress/1.26.1/commons-compress-1.26.1.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.14.0</version>
</dependency>
```


Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)

repo1.maven.org/maven2/org/openjfx/javafx-graphics/21.0.2/

```
org/openjfx/javafx-graphics/21.0.2/
├── javafx-graphics-21.0.2-linux.jar
├── javafx-graphics-21.0.2-mac.jar
└── javafx-graphics-21.0.2-win.jar
```

Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)
4. Parts of Java software are OS/Arch specific – at minimum the VM itself (misunderstood / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

repo1.maven.org/maven2/org/apache/commons/commons-text/1.11.0/commons-text-1.11.0.pom

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <scope>compile</scope>
  <version>3.13.0</version>
</dependency>
```

Wrong scope

repo1.maven.org/maven2/org/openjfx/javafx-graphics/21.0.2/

```
org/openjfx/javafx-graphics/21.0.2/
├── javafx-graphics-21.0.2.pom
├── javafx-graphics-21.0.2.module
├── javafx-graphics-21.0.2-linux.jar
├── javafx-graphics-21.0.2-mac.jar
└── javafx-graphics-21.0.2-win.jar
```

No rich metadata beyond POM 4.0.0


Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)
4. Parts of Java software are OS/Arch specific – at minimum the VM itself (misunderstood / bad tool support)
5. Incomplete or wrong 3rd party Module definitions (misunderstood by authors / bad tool support)

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

Accidental Complexity

1. Mixing Module Definition and other concerns of the build
2. Different Module visibility at compile time and runtime (misunderstood / bad tool support)

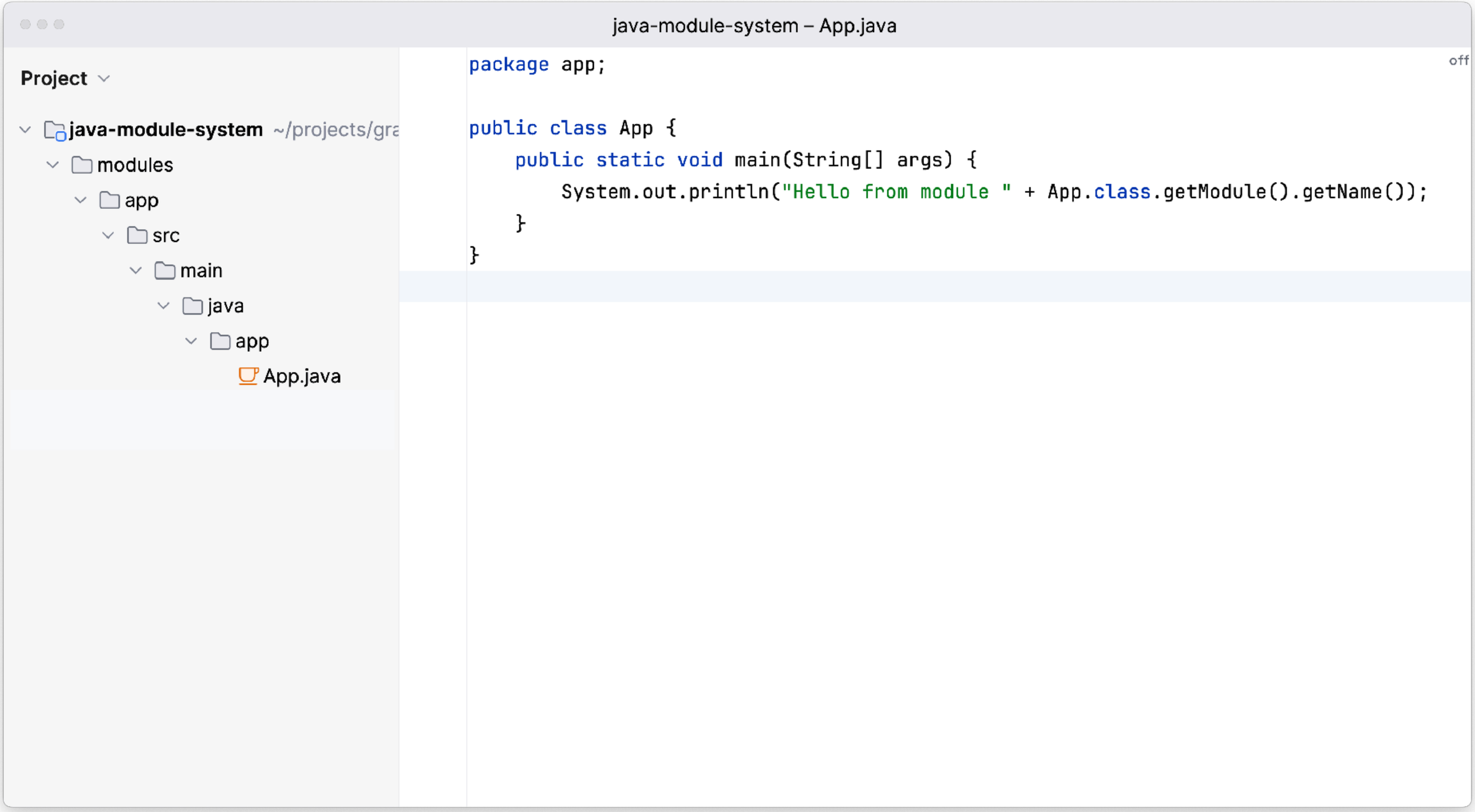
Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)

How to use the Java Module System to address this?

EXAMPLE

a pure Java Modules project


java-module-system – App.java
Project
java-module-system ~/projects/gra
modules
app
src
main
java
app
App.java
package app;
public class App {
public static void main(String[] args) {
System.out.println("Hello from module " + App.class.getModule().getName());
}
}

java-module-system – App.java

Project

java-module-system ~/projects/gra
modules
app
src
main
java
app
App.java

```java
package app;

public class App {
    public static void main(String[] args) {
        System.out.println("Hello from module " + App.class.getModule().getName());
    }
}
```

java-module-system – DefaultAmazingService.java

Project

- java-module-system ~/projects/gradle/howto/java-m
  - modules
    - amazing
      - src
        - main
          - java
            - amazing
              - impl
                - DefaultAmazingService.java
              - AmazingService.java
    - app
      - src
        - main
          - java
            - app
              - App.java

```java
package amazing.impl;

import amazing.AmazingService;

public class DefaultAmazingService implements AmazingService {
    @Override
    public void compute() {
        System.out.println("Hello from module " +
                getClass().getModule().getName());
    }
}
```

```java
package amazing;

import amazing.impl.DefaultAmazingService;

public interface AmazingService {
    AmazingService DEFAULT = new DefaultAmazingService();

    vo
}
```

java-module-system – App.java

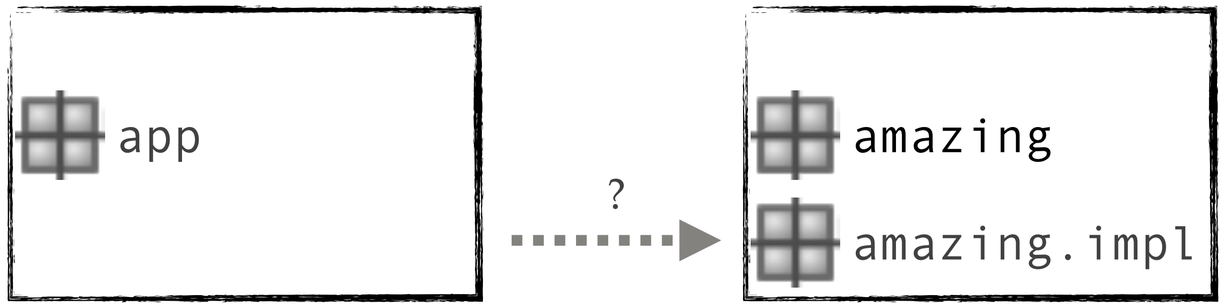
Project

- java-module-system ~/projects/gradle/howto/java-m
  - modules
    - amazing
      - src
        - main
          - java
            - amazing
              - impl
                - DefaultAmazingService.java
              - AmazingService.java
    - app
      - src
        - main
          - java
            - app
              - App.java

```java
package app;

import amazing.AmazingService;

public class App {
    public static void main(String[] args) {
        System.out.println("Hello from module " +
                App.class.getModule().getName());
        AmazingService.DEFAULT.compute();
    }
}
```



java-module-system – module-info.java (amazing)

Project

- java-module-system ~/projects/gradle/howto/java-m
  - modules
    - amazing
      - src
        - main
          - java
            - amazing
              - impl
                - DefaultAmazingService.java
              - AmazingService.java
            - module-info.java
    - app
      - src
        - main
          - java
            - app
              - App.java

```
module amazing {
    exports amazing;
}
```

off

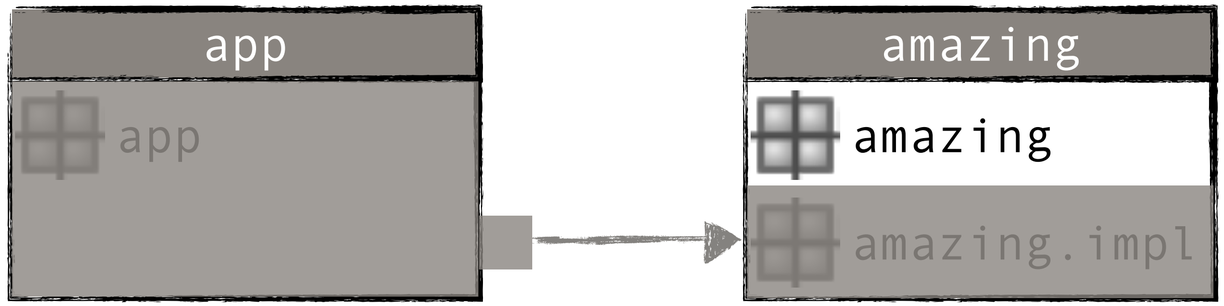
java-module-system – module-info.java (amazing)

Project

- java-module-system ~/projects/gradle/howto/java-m
  - modules
    - amazing
      - src
        - main
          - java
            - amazing
              - impl
                - DefaultAmazingService.java
              - AmazingService.java
            - module-info.java
    - app
      - src
        - main
          - java
            - app
              - App.java
            - module-info.java

```
module amazing {
    exports amazing;
}
```

```
module app {
    requires amazing;
}
```



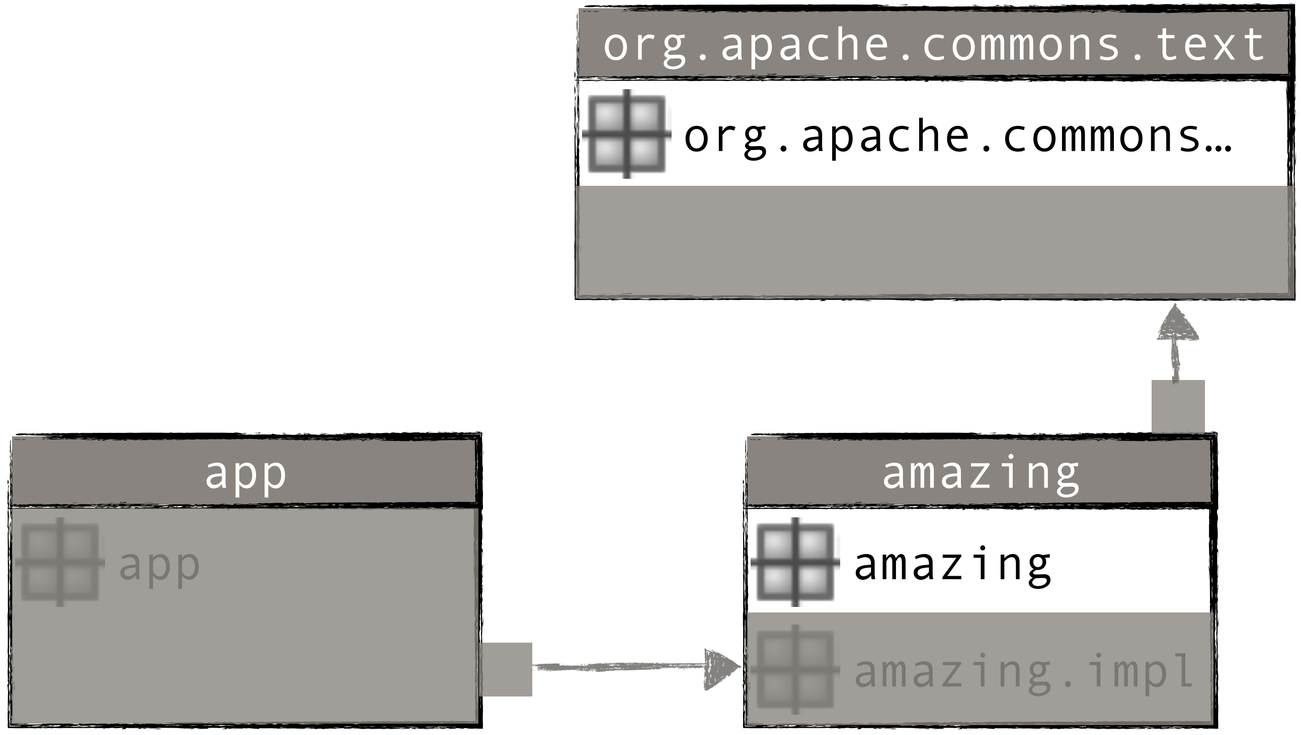
java-module-system – module-info.java (app)

Project

- java-module-system ~/projects/gradle/howto/java-m
  - modules
    - amazing
      - src
        - main
          - java
            - amazing
              - impl
                - DefaultAmazingService.java
              - AmazingService.java
            - module-info.java
    - app
      - src
        - main
          - java
            - app
              - App.java
            - module-info.java

```
module amazing {
    requires org.apache.commons.text;
    exports amazing;
}
```

```
module app {
    requires amazing;
}
```



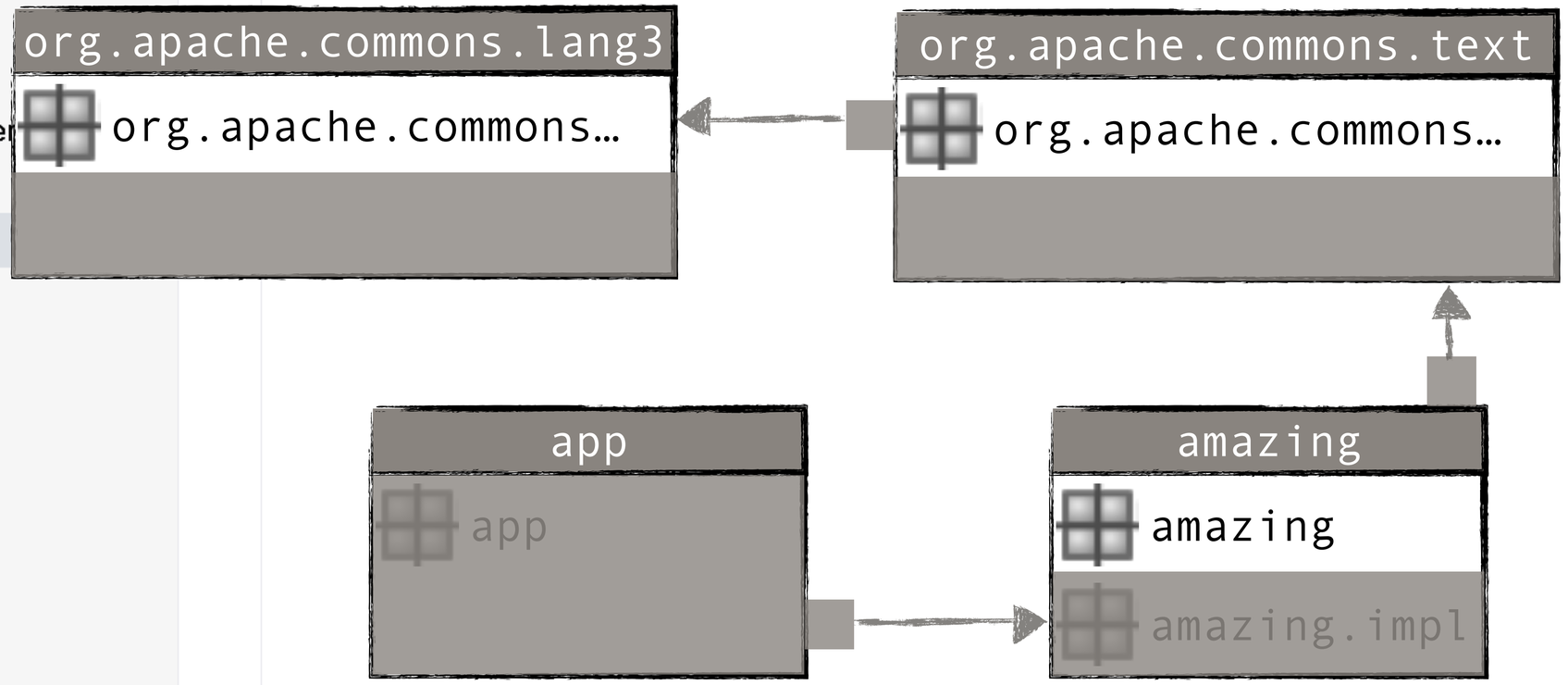
java-module-system – module-info.java (app)

Project

- java-module-system ~/projects/gradle/howto/java-m
  - modules
    - amazing
      - src
        - main
          - java
            - amazing
              - impl
                - DefaultAmazingSer
              - AmazingService.java
            - module-info.java
    - app
      - src
        - main
          - java
            - app
              - App.java
            - module-info.java

```
module amazing {
    requires org.apache.commons.text;
    exports amazing;
}
```

```
module app {
    requires amazing;
}
```



Accidental Complexity

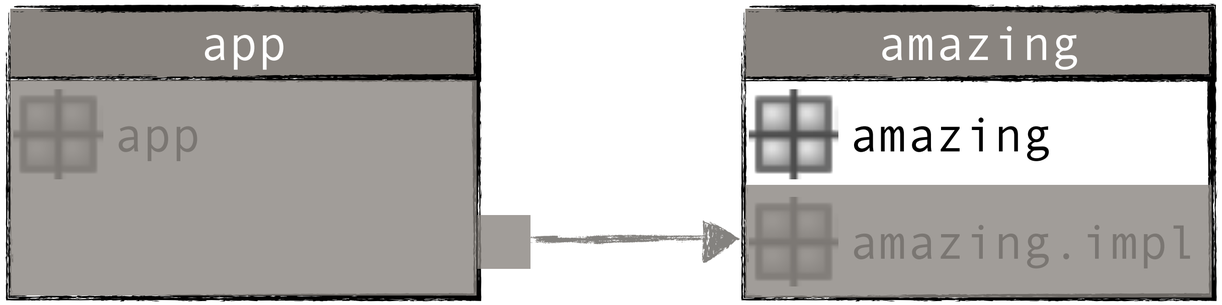
1. Mixing Module Definition and other c
2. Differe
compile time and runtime (misunderstood / bad tool support)

Module Definition

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)

Use **module-info.java** for isolated Module Definitions in standard Java syntax



Accidental Complexity

1. ~~Mixing M~~
   ~~other cor~~
2. ~~Different~~
   ~~compile t~~
   ~~(misunderstood / bad tool support)~~
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misunderstood / bad tool support)
4. Parts of Java software are OS/Arch specific – at minimum the VM itself (misunderstood / bad tool support)
5. Incomplete or wrong 3rd party Module definitions (misunderstood by authors / bad tool support)

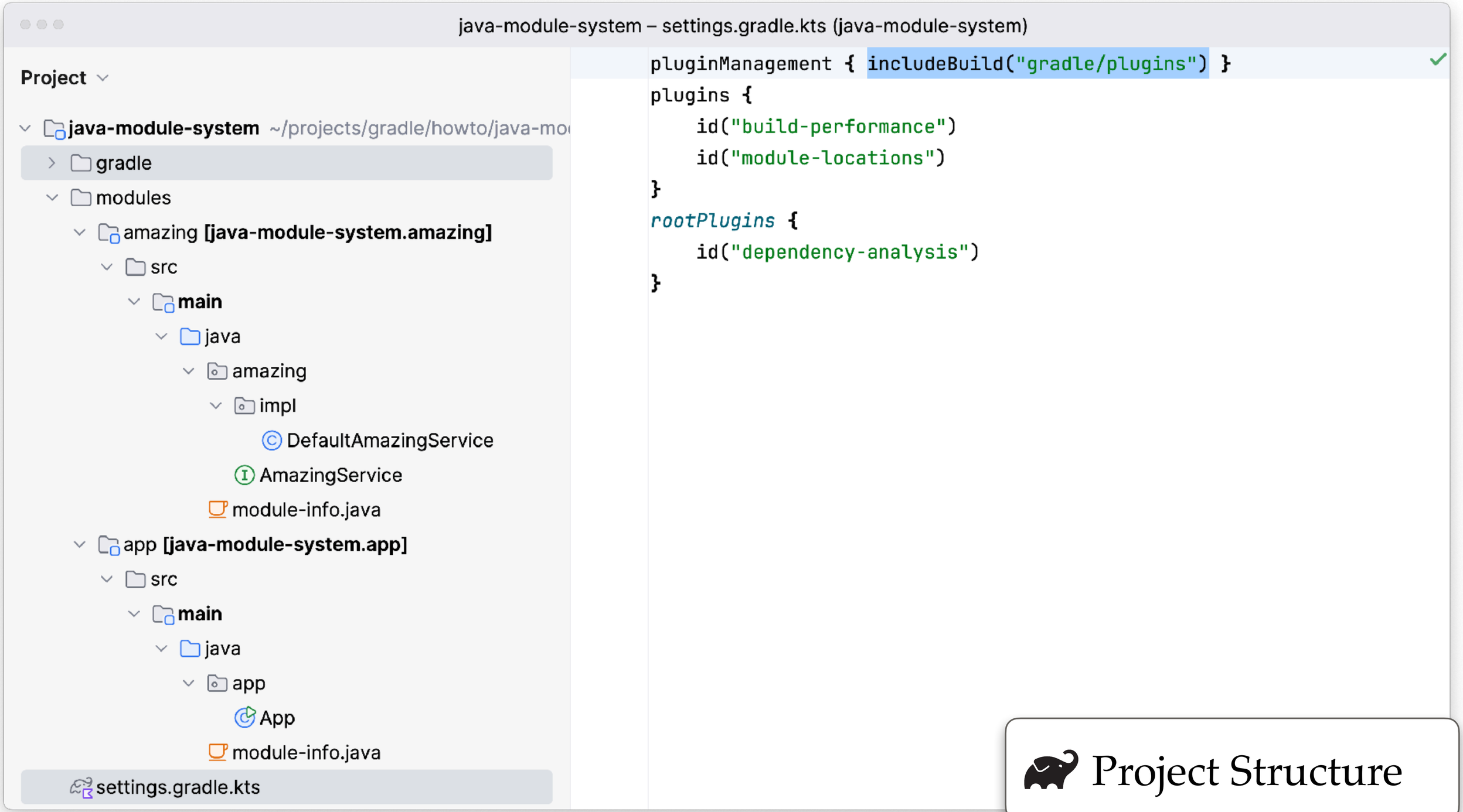
Essential Complexity

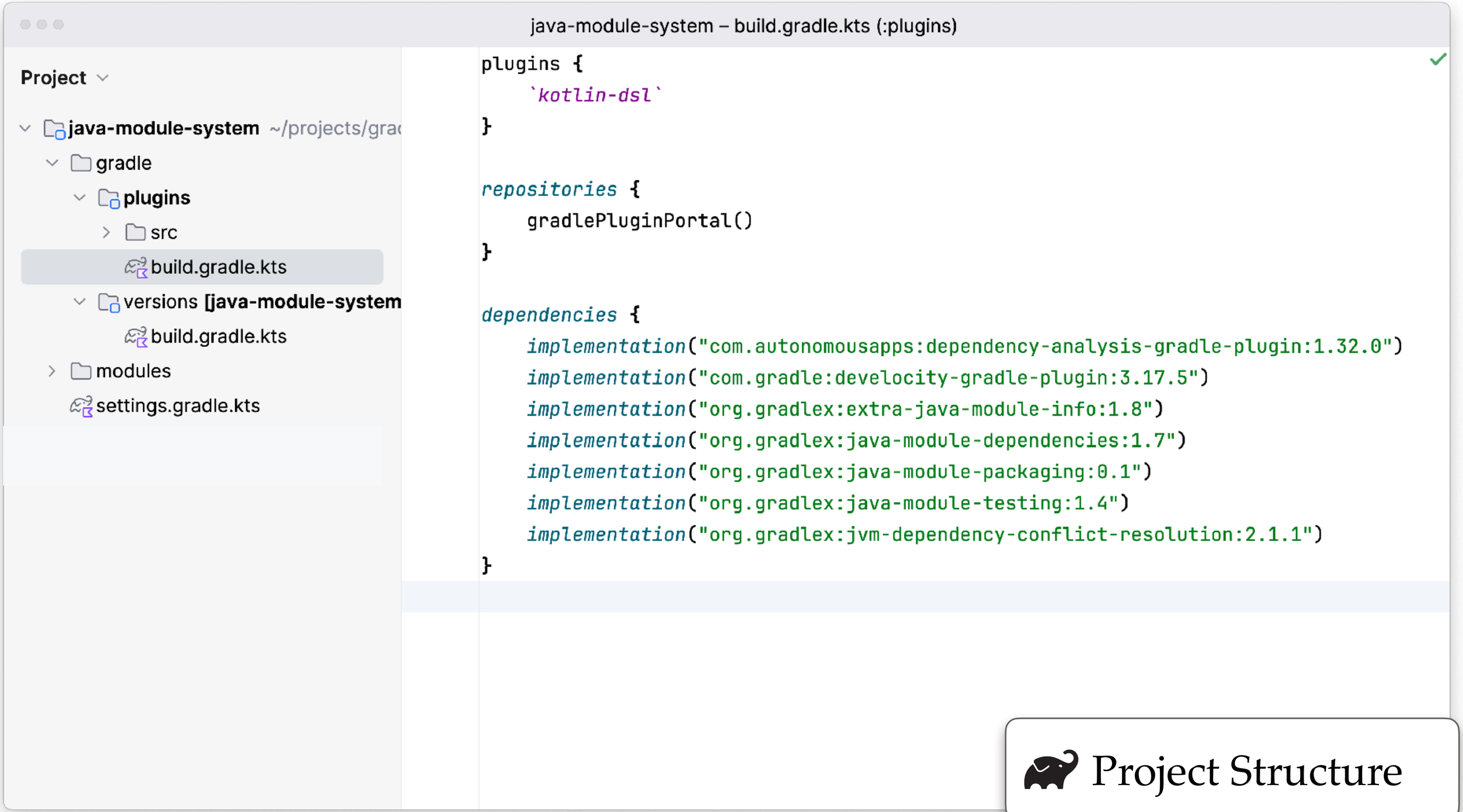
- lity at
  me (embrace
  this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept / good tool support)
- Parts of Java software are OS/Arch specific (accept / good tool support)

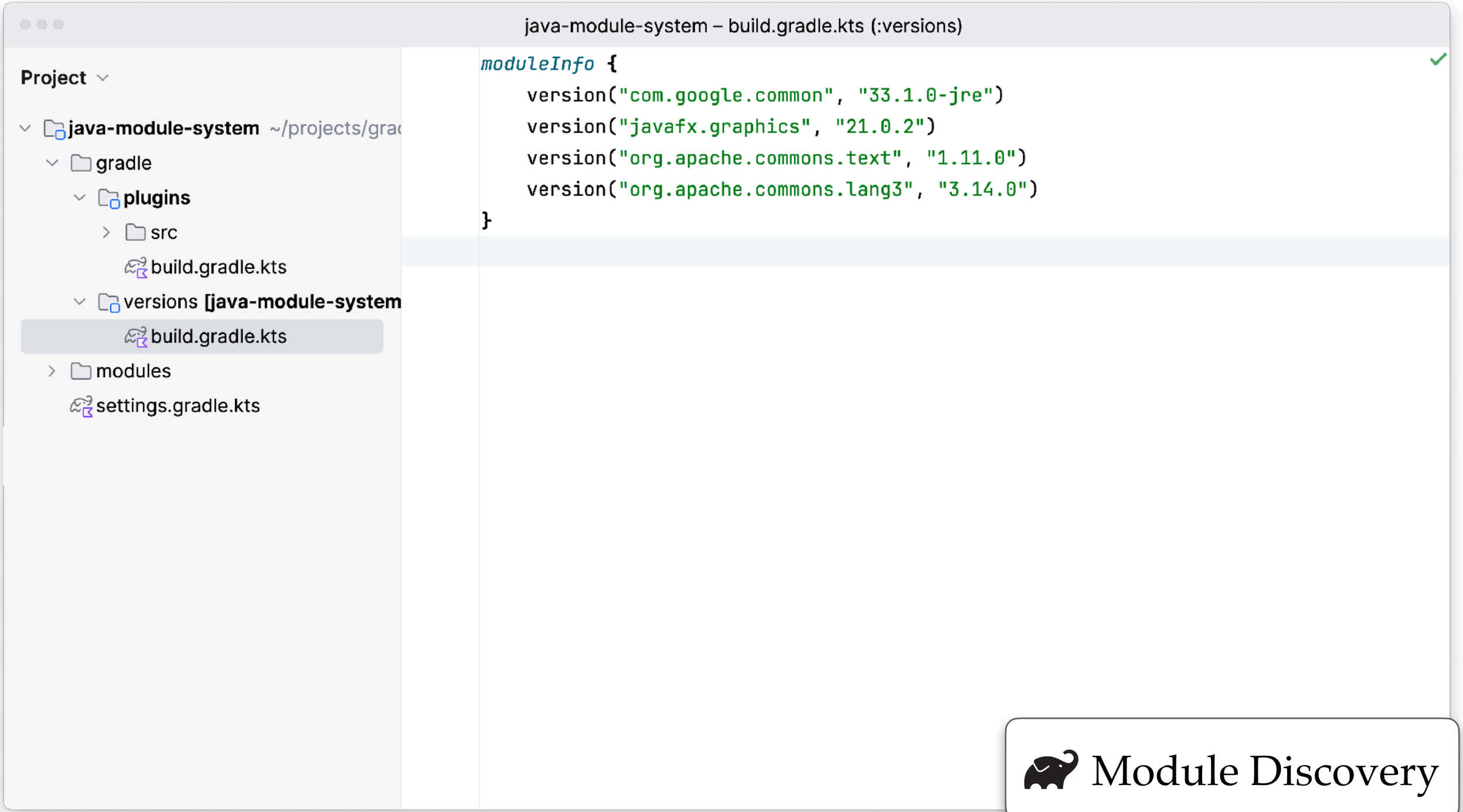
How to use Gradle's Dependency Management to address this?

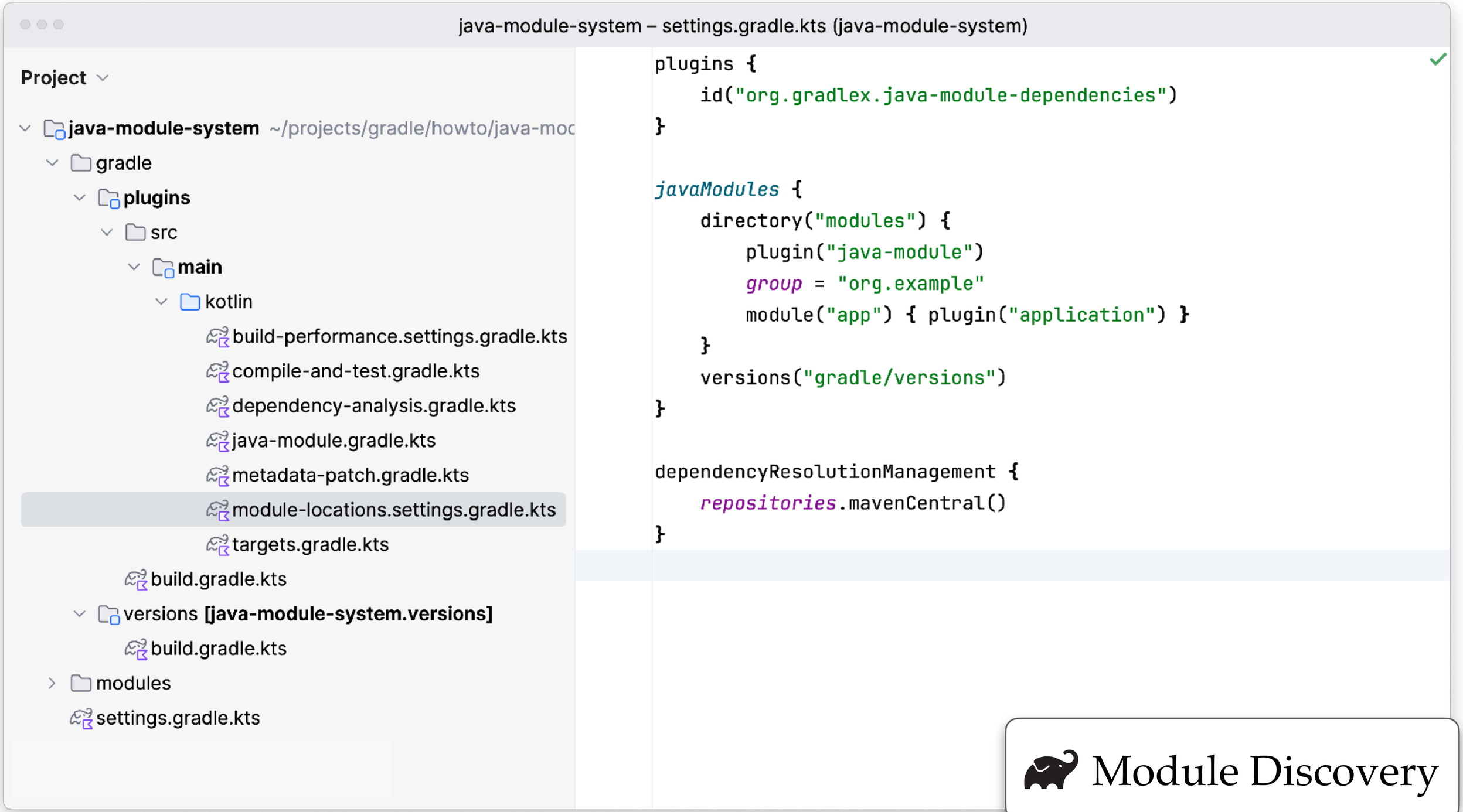
EXAMPLE

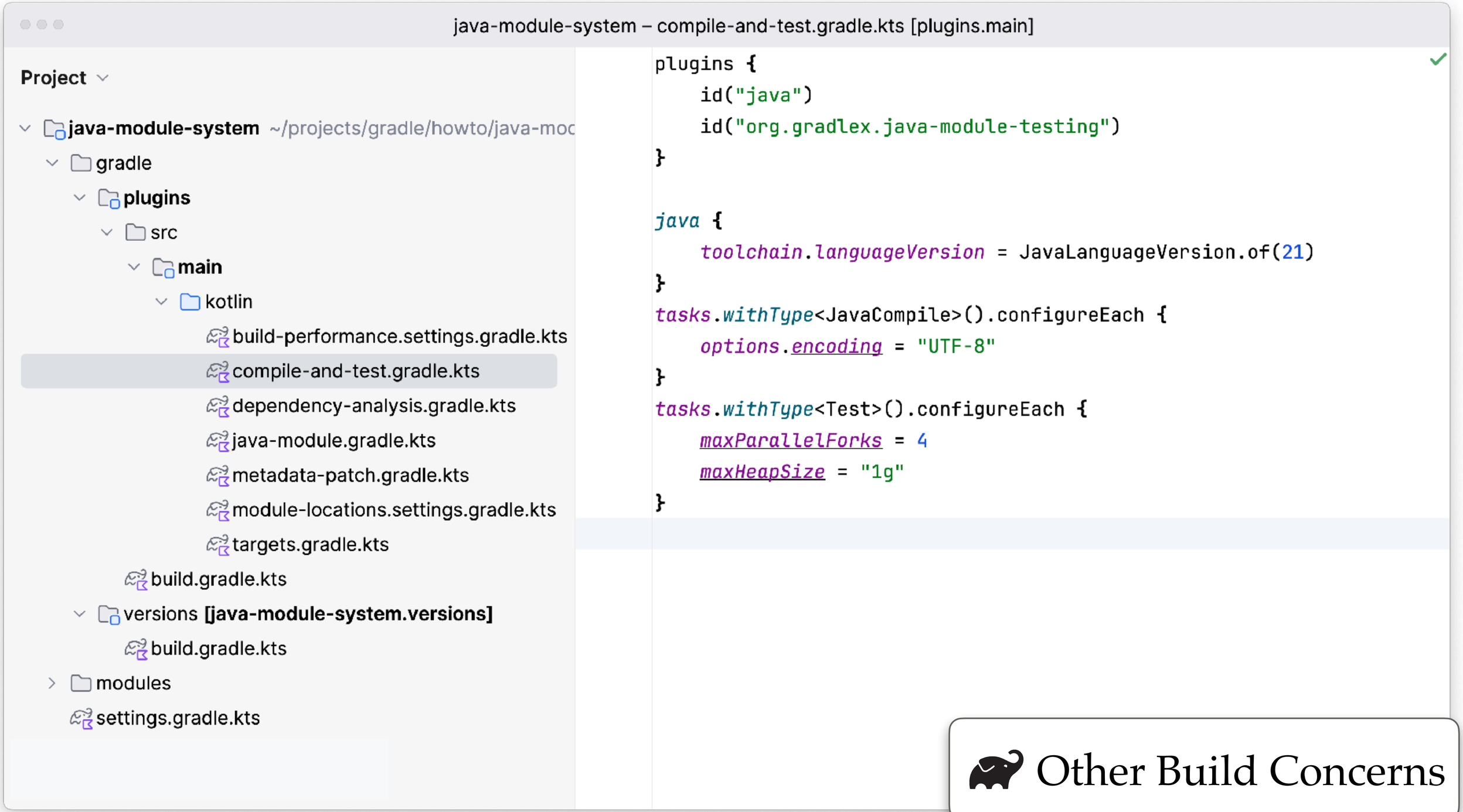
add a Gradle-based build system


java-module-system – settings.gradle.kts (java-module-system)
Project
java-module-system ~/projects/gradle/howto/java-mo
gradle
modules
amazing [java-module-system.amazing]
src
main
java
amazing
impl
DefaultAmazingService
AmazingService
module-info.java
app [java-module-system.app]
src
main
java
app
App
module-info.java
settings.gradle.kts
pluginManagement { includeBuild("gradle/plugins") }
plugins {
id("build-performance")
id("module-locations")
}
rootPlugins {
id("dependency-analysis")
}
Project Structure


java-module-system – build.gradle.kts (:plugins)
Project
java-module-system ~/projects/grad
gradle
plugins
src
build.gradle.kts
versions [java-module-system
build.gradle.kts
modules
settings.gradle.kts
plugins {
`kotlin-dsl`
}
repositories {
gradlePluginPortal()
}
dependencies {
implementation("com.autonomousapps:dependency-analysis-gradle-plugin:1.32.0")
implementation("com.gradle:develocity-gradle-plugin:3.17.5")
implementation("org.gradlex:extra-java-module-info:1.8")
implementation("org.gradlex:java-module-dependencies:1.7")
implementation("org.gradlex:java-module-packaging:0.1")
implementation("org.gradlex:java-module-testing:1.4")
implementation("org.gradlex:jvm-dependency-conflict-resolution:2.1.1")
}
Project Structure

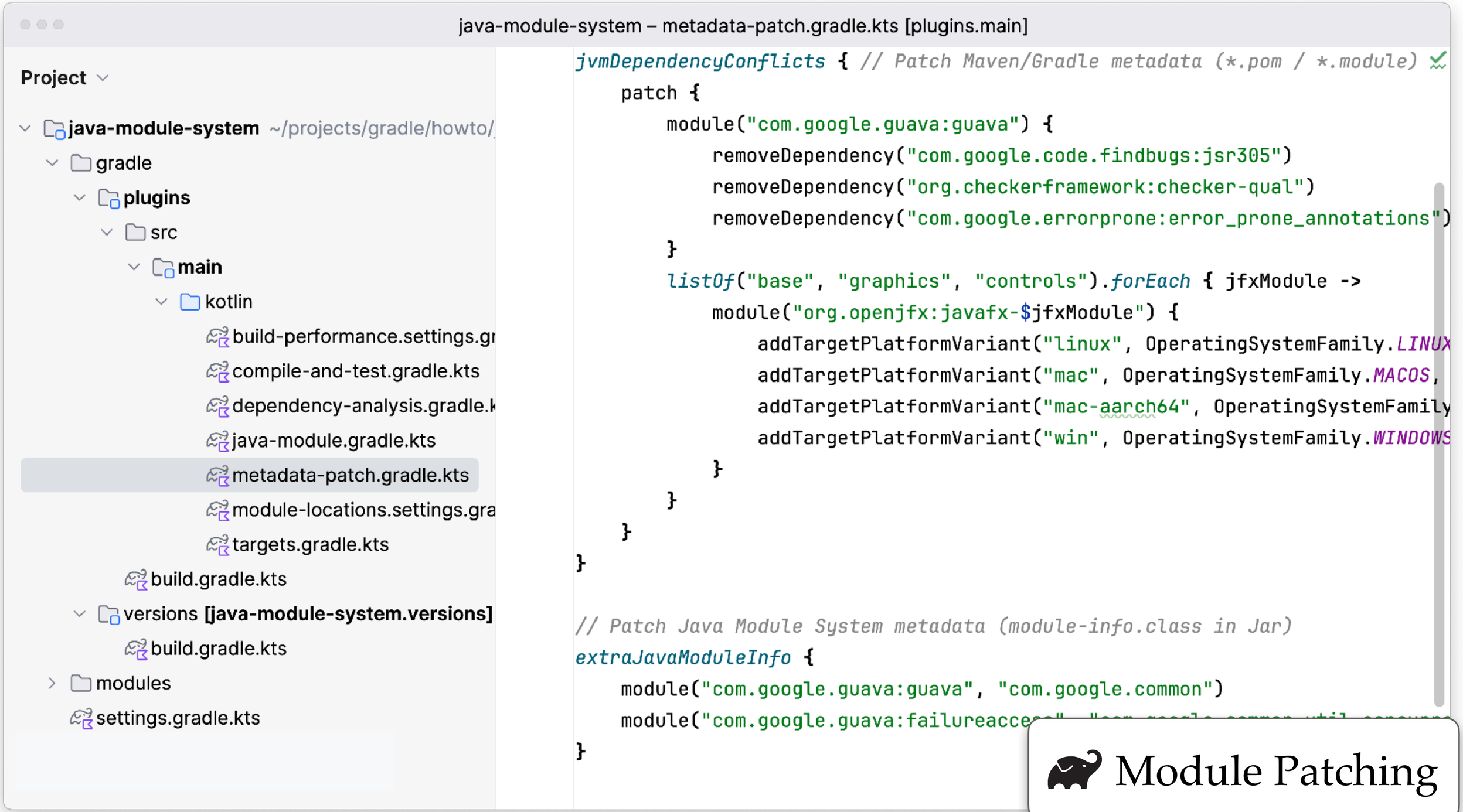

java-module-system – build.gradle.kts (:versions)
Project
java-module-system ~/projects/grad
gradle
plugins
src
build.gradle.kts
versions [java-module-system
build.gradle.kts
modules
settings.gradle.kts
moduleInfo {
version("com.google.common", "33.1.0-jre")
version("javafx.graphics", "21.0.2")
version("org.apache.commons.text", "1.11.0")
version("org.apache.commons.lang3", "3.14.0")
}
Module Discovery


java-module-system – settings.gradle.kts (java-module-system)
Project
java-module-system ~/projects/gradle/howto/java-mod
gradle
plugins
src
main
kotlin
build-performance.settings.gradle.kts
compile-and-test.gradle.kts
dependency-analysis.gradle.kts
java-module.gradle.kts
metadata-patch.gradle.kts
module-locations.settings.gradle.kts
targets.gradle.kts
build.gradle.kts
versions [java-module-system.versions]
build.gradle.kts
modules
settings.gradle.kts
plugins {
id("org.gradlex.java-module-dependencies")
}
javaModules {
directory("modules") {
plugin("java-module")
group = "org.example"
module("app") { plugin("application") }
}
versions("gradle/versions")
}
dependencyResolutionManagement {
repositories.mavenCentral()
}
Module Discovery



Other Build Concerns


java-module-system – targets.gradle.kts [plugins.main]
Project
java-module-system ~/projects/gradle/howto/java-mod
gradle
plugins
src
main
kotlin
build-performance.settings.gradle.kts
compile-and-test.gradle.kts
dependency-analysis.gradle.kts
java-module.gradle.kts
metadata-patch.gradle.kts
module-locations.settings.gradle.kts
targets.gradle.kts
build.gradle.kts
versions [java-module-system.versions]
build.gradle.kts
modules
settings.gradle.kts
plugins {
id("org.gradlex.java-module-packaging")
}
javaModulePackaging {
target("ubuntu-22.04") {
operatingSystem = OperatingSystemFamily.LINUX
architecture = MachineArchitecture.X86_64
packageTypes = listOf("deb")
}
target("macos-14") {
operatingSystem = OperatingSystemFamily.MACOS
architecture = MachineArchitecture.ARM64
packageTypes = listOf("dmg")
}
target("windows-2022") {
operatingSystem = OperatingSystemFamily.WINDOWS
architecture = MachineArchitecture.X86_64
packageTypes = listOf("exe")
}
primaryTarget(target("macos
}
OS / Arch Specifics


java-module-system – metadata-patch.gradle.kts [plugins.main]
Project
java-module-system ~/projects/gradle/howto/
gradle
plugins
src
main
kotlin
build-performance.settings.gr
compile-and-test.gradle.kts
dependency-analysis.gradle.k
java-module.gradle.kts
metadata-patch.gradle.kts
module-locations.settings.gra
targets.gradle.kts
build.gradle.kts
versions [java-module-system.versions]
build.gradle.kts
modules
settings.gradle.kts
jvmDependencyConflicts { // Patch Maven/Gradle metadata (*.pom / *.module)
patch {
module("com.google.guava:guava") {
removeDependency("com.google.code.findbugs:jsr305")
removeDependency("org.checkerframework:checker-qual")
removeDependency("com.google.errorprone:error_prone_annotations")
}
listOf("base", "graphics", "controls").forEach { jfxModule ->
module("org.openjfx:javafx-$jfxModule") {
addTargetPlatformVariant("linux", OperatingSystemFamily.LINUX
addTargetPlatformVariant("mac", OperatingSystemFamily.MACOS,
addTargetPlatformVariant("mac-aarch64", OperatingSystemFamily
addTargetPlatformVariant("win", OperatingSystemFamily.WINDOWS
}
}
}
}
// Patch Java Module System metadata (module-info.class in Jar)
extraJavaModuleInfo {
module("com.google.guava:guava", "com.google.common")
module("com.google.guava:failureacces
}
Module Patching

Accidental Complexity

1. Mixing Module Definition and other c
2. Differe compile time and runtime (misunderstood / bad tool support)
3. Multiple Versions and Variants of a 3rd party Module are in conflict (misu
4. Parts c specific – at minimum the VM itself (misunderstood / bad tool support)
5. Incom Modu by author / bad tool support)

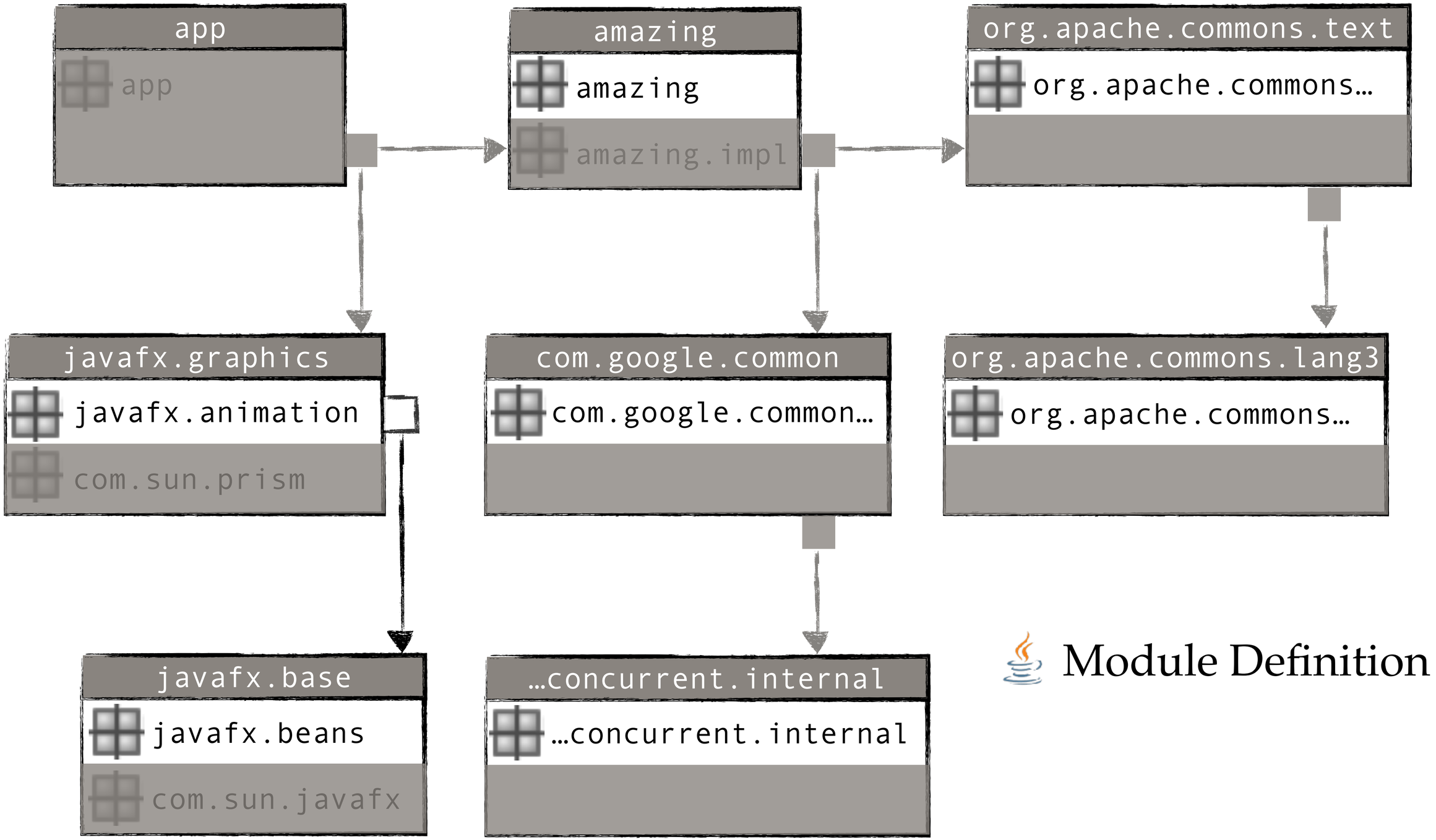
Module Definition

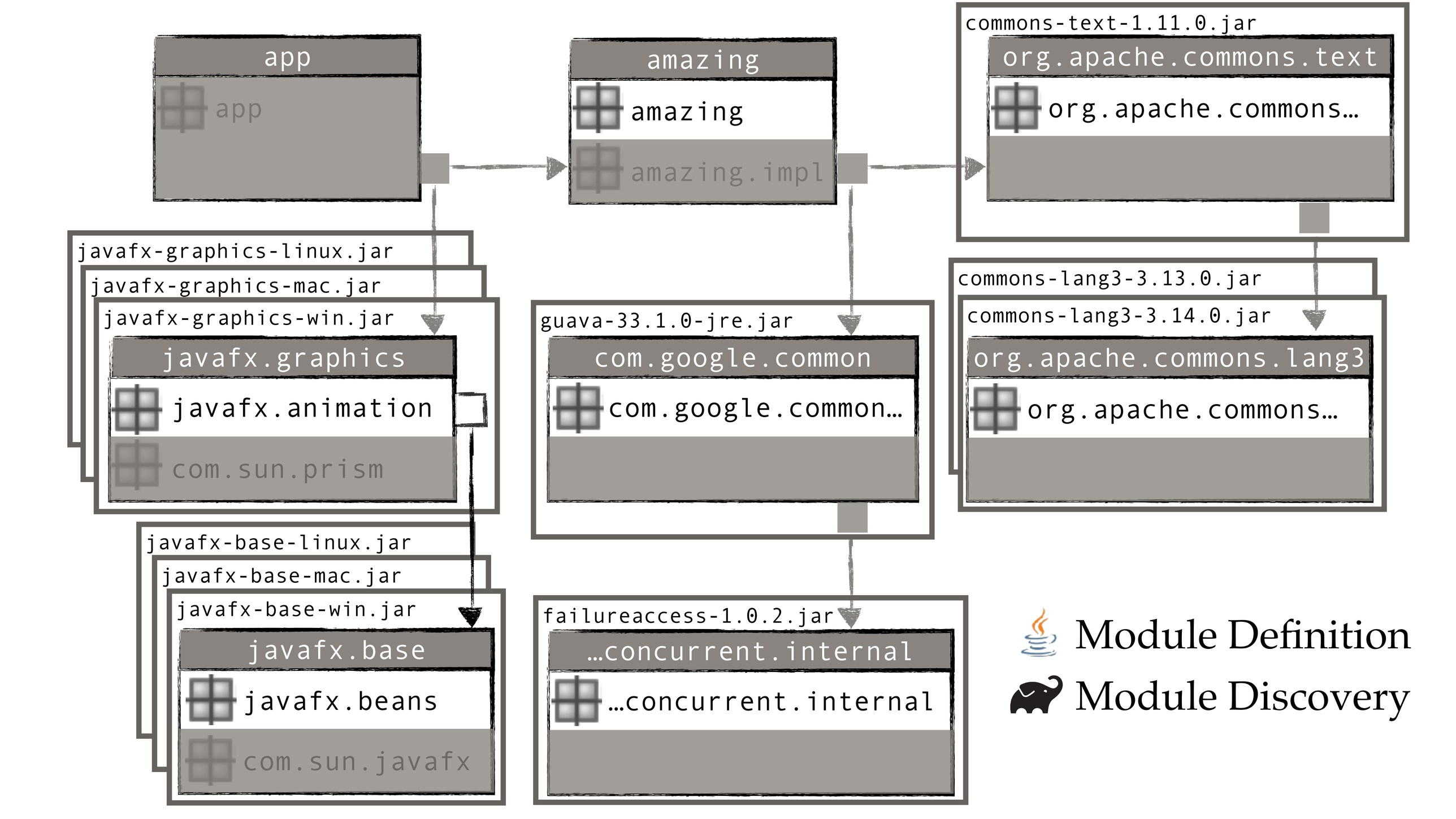
Module Discovery

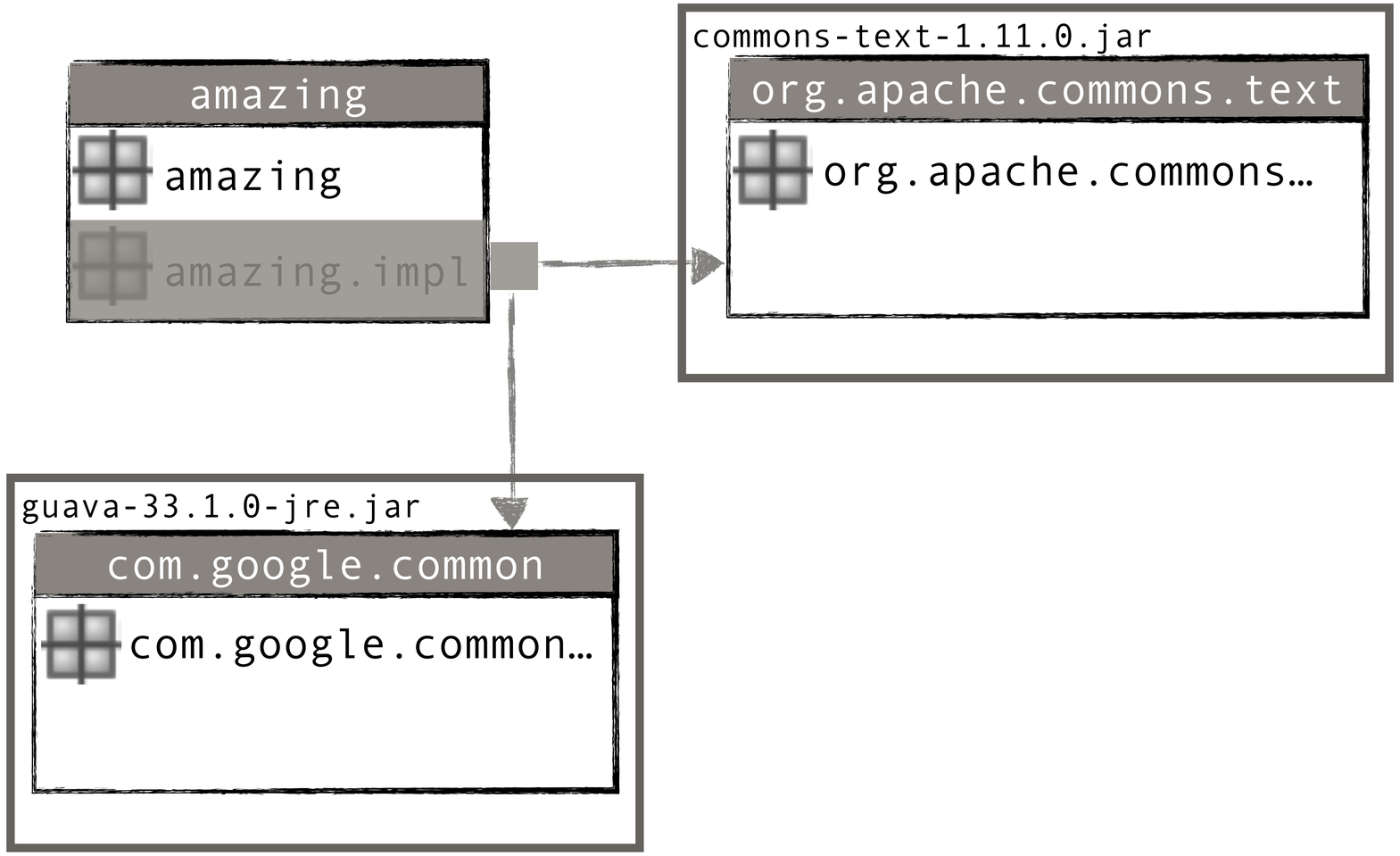
Module Patching

Essential Complexity

- Different Module visibility at compile time and runtime (embrace this feature / good tool support)
- Multiple Versions and Variants of a 3rd party Module are in conflict (accept reality / good tool support)
- Parts of Java software are OS/Arch specific (accept reality / good tool support)


app
app
amazing
amazing
amazing.impl
org.apache.commons.text
org.apache.commons…
javafx.graphics
javafx.animation
com.sun.prism
com.google.common
com.google.common…
org.apache.commons.lang3
org.apache.commons…
javafx.base
javafx.beans
com.sun.javafx
…concurrent.internal
…concurrent.internal
Module Definition

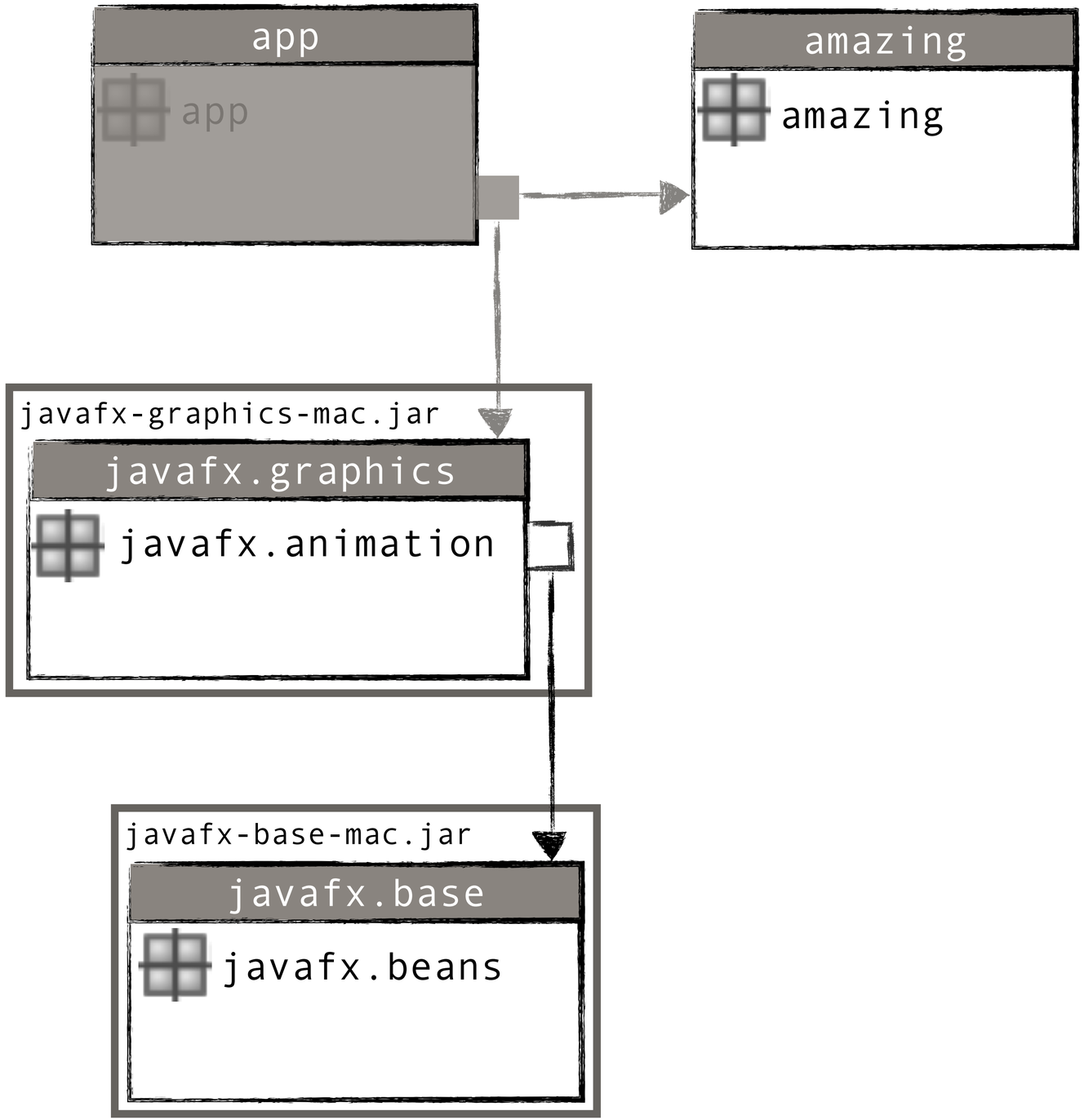

app
app
amazing
amazing
amazing.impl
commons-text-1.11.0.jar
org.apache.commons.text
org.apache.commons…
javafx-graphics-linux.jar
javafx-graphics-mac.jar
javafx-graphics-win.jar
javafx.graphics
javafx.animation
com.sun.prism
guava-33.1.0-jre.jar
com.google.common
com.google.common…
commons-lang3-3.13.0.jar
commons-lang3-3.14.0.jar
org.apache.commons.lang3
org.apache.commons…
javafx-base-linux.jar
javafx-base-mac.jar
javafx-base-win.jar
javafx.base
javafx.beans
com.sun.javafx
failureaccess-1.0.2.jar
…concurrent.internal
…concurrent.internal
Module Definition
Module Discovery



Module Definition

Module Discovery

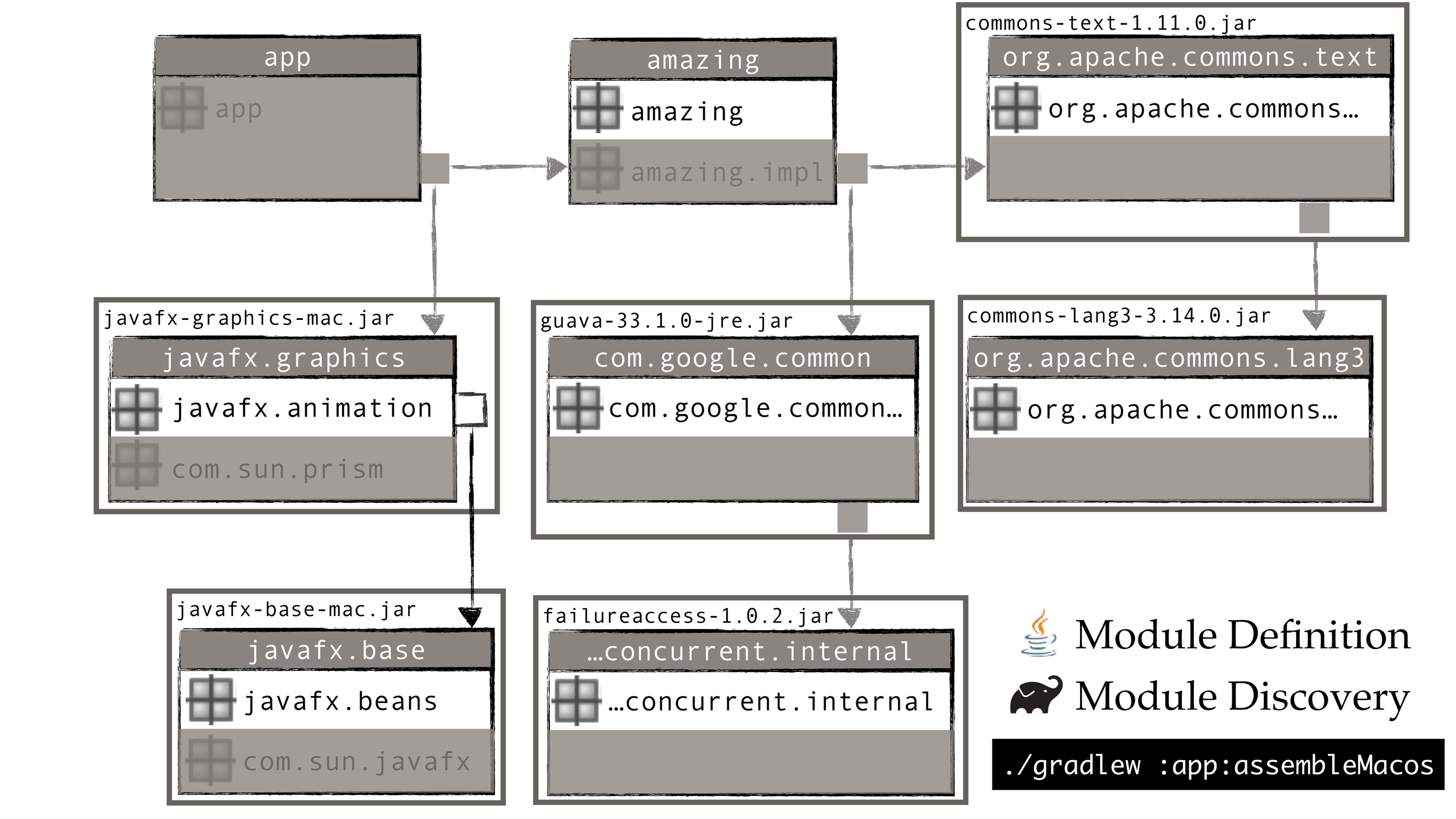
```
./gradlew :amazing:compileJava
```

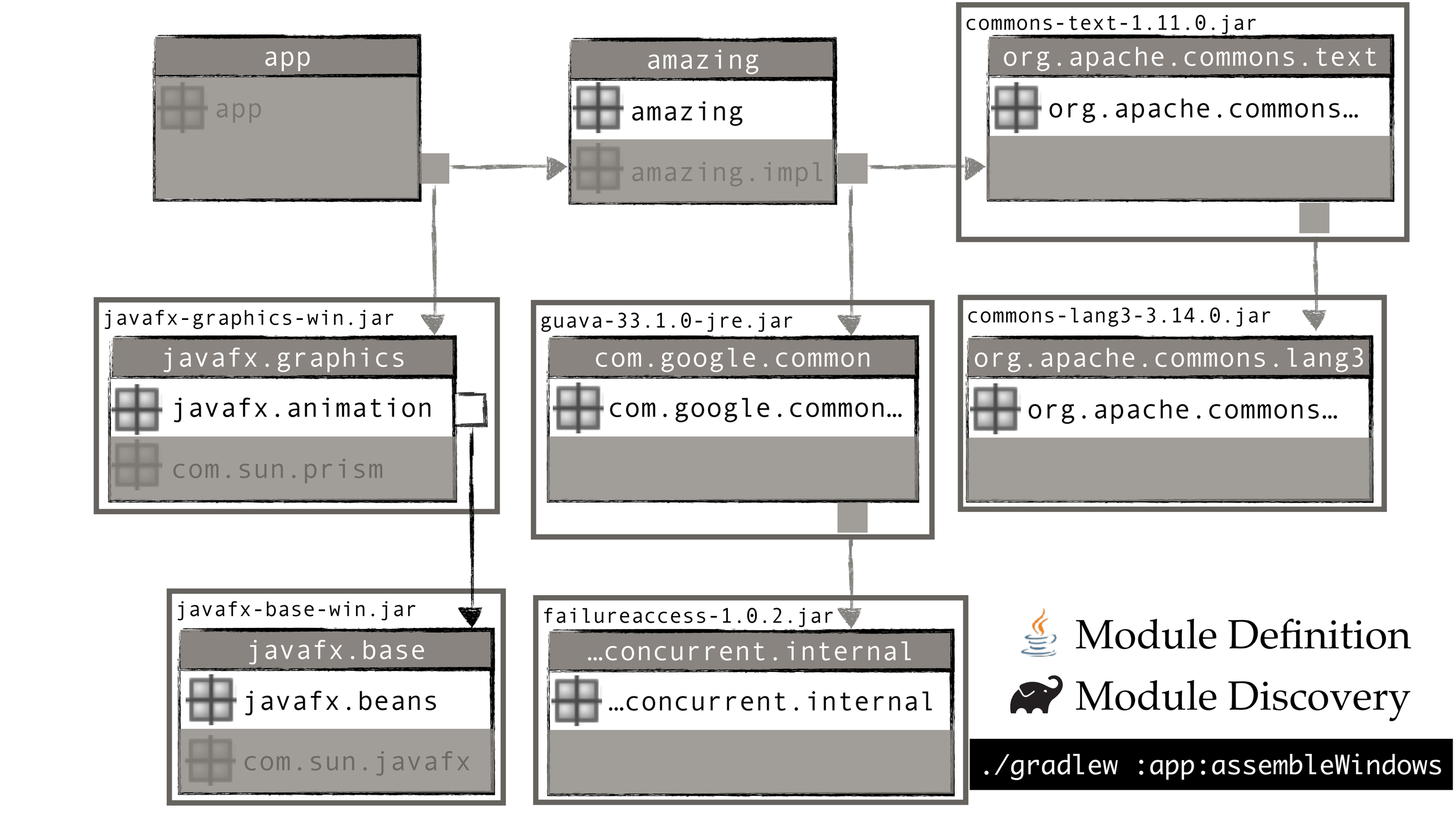


Module Definition

Module Discovery

```
./gradlew :app:compileJava
```


app
app
amazing
amazing
amazing.impl
commons-text-1.11.0.jar
org.apache.commons.text
org.apache.commons…
javafx-graphics-mac.jar
javafx.graphics
javafx.animation
com.sun.prism
guava-33.1.0-jre.jar
com.google.common
com.google.common…
commons-lang3-3.14.0.jar
org.apache.commons.lang3
org.apache.commons…
javafx-base-mac.jar
javafx.base
javafx.beans
com.sun.javafx
failureaccess-1.0.2.jar
…concurrent.internal
…concurrent.internal
Module Definition
Module Discovery
./gradlew :app:assembleMacos


app
app
amazing
amazing
amazing.impl
commons-text-1.11.0.jar
org.apache.commons.text
org.apache.commons…
javafx-graphics-win.jar
javafx.graphics
javafx.animation
com.sun.prism
guava-33.1.0-jre.jar
com.google.common
com.google.common…
commons-lang3-3.14.0.jar
org.apache.commons.lang3
org.apache.commons…
javafx-base-win.jar
javafx.base
javafx.beans
com.sun.javafx
failureaccess-1.0.2.jar
…concurrent.internal
…concurrent.internal
Module Definition
Module Discovery
./gradlew :app:assembleWindows

But in the real world … ?

Real World example: Hedera Services

github.com/hashgraph/hedera-services/blob/develop/hedera-node/hedera-consensus-service-impl/src/main/java/module-info.java

a module-info.java

github.com/hashgraph/hedera-services/blob/develop/hedera-node/hedera-consensus-service-impl/build.gradle.kts

a minimalistic build.gradle

github.com/hashgraph/hedera-services/blob/develop/hedera-dependency-versions/build.gradle.kts

versions

github.com/hashgraph/hedera-services/blob/develop/gradle/plugins/src/main/kotlin/com.hedera.gradle.jpms-modules.gradle.kts

3rd party patching

github.com/hashgraph/hedera-services/tree/develop/gradle/plugins/src/main/kotlin

other build concerns

What if I can’t use the Java Module System?

Module Definition

Java's module-info

- Module naming
- Module dependencies
  - Module private (requires)
  - Re-exported (req transitive)
  - Compile-only (req static)
- Module private packages

Module Discovery

Gradle Dependency Management

- Discover and build Modules from source on demand
- Discover Modules in Repositories by Name and Version
- Detect and handle Version and Variant conflicts
- Patch Module Metadata before handing it to Java tools

Module Definition

Java's module-info (build time only)

- Module naming
- Module dependencies
  - Module private (requires)
  - Re-exported (req transitive)
  - Compile-only (req static)
- ~~Module private packages~~

```
module amazing {
  requires org.apache.commons.text;
}
```

Module Discovery

Gradle Dependency Management

- Discover and build Modules from source on demand
- Discover Modules in Repositories by Name and Version
- Detect and handle Version and Variant conflicts
- Patch Module Metadata before handing it to Java tools

Module Definition

Gradle's build.gradle `dependencies`

- Module naming
- Module dependencies
  - Module private (implementation)
  - Re-exported (api)
  - Compile-only (compileOnly)
- ~~Module private packages~~

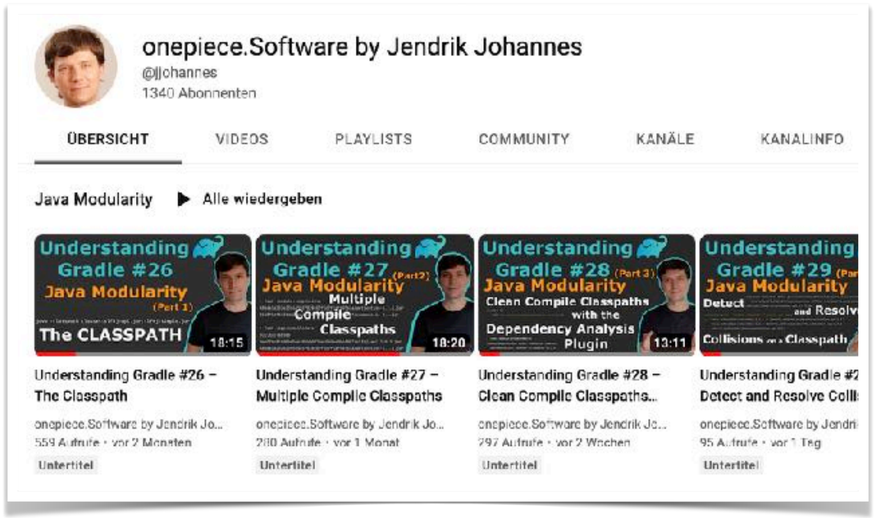
Module Discovery

Gradle Dependency Management

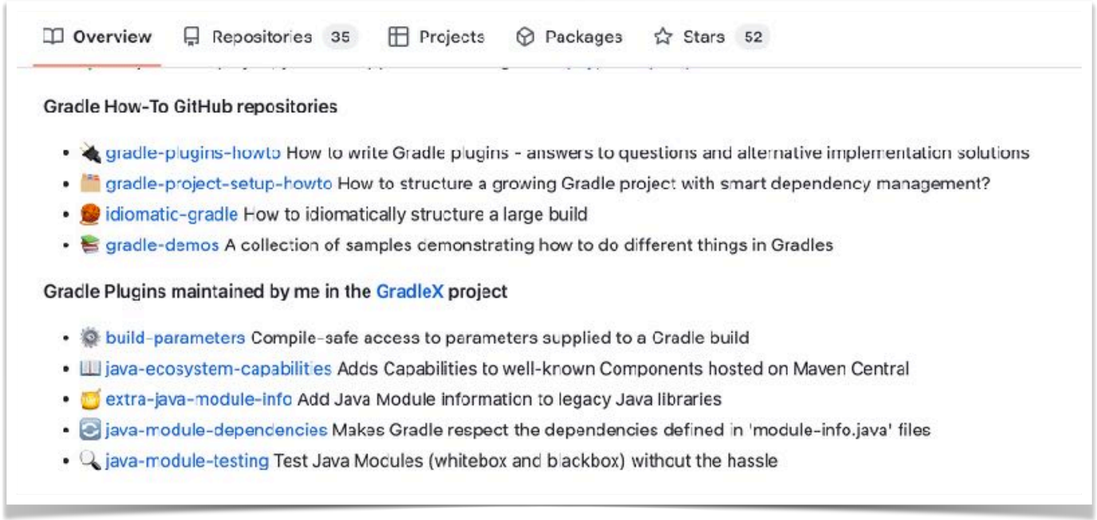
- Discover and build Modules from source on demand
- Discover Modules in Repositories by Name and Version
- Detect and handle Version and Variant conflicts
- Patch Module Metadata before handing it to Java tools

```
dependencies {
  implementation("org.apache.commons:commons-text")
}
```

github.com/jjohannes/java-module-system (slides / example)
github.com/jjohannes/gradle-project-setup-howto
github.com/hashgraph/hedera-services



youtube.com/@jjohannes



github.com/jjohannes

stay in touch
mastodon.social/@jendrik
linkedin.com/in/jendrikjohannes
jendrik@onepiece.software

training and consulting
onepiece.software

GradleX Project
gradlex.org