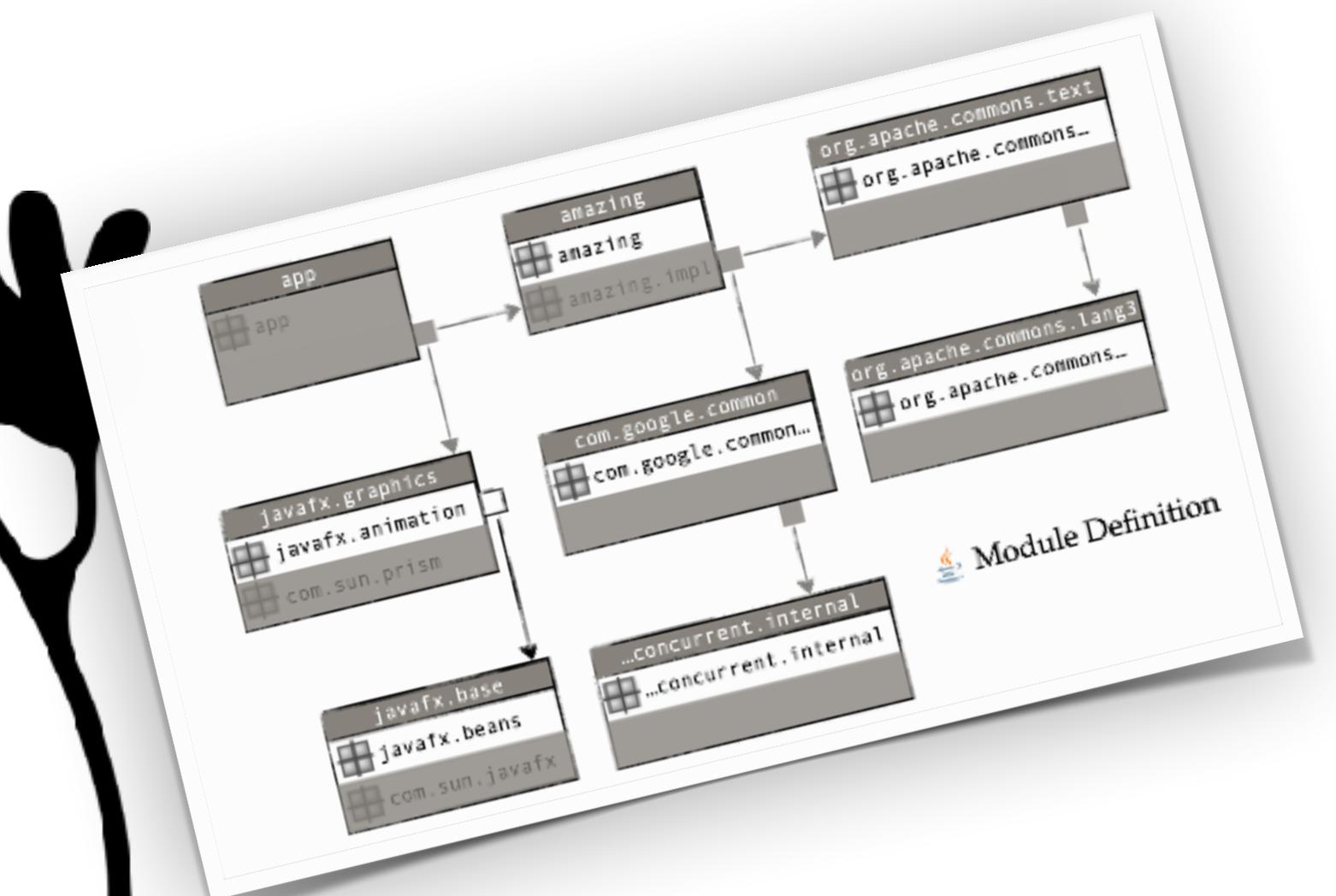
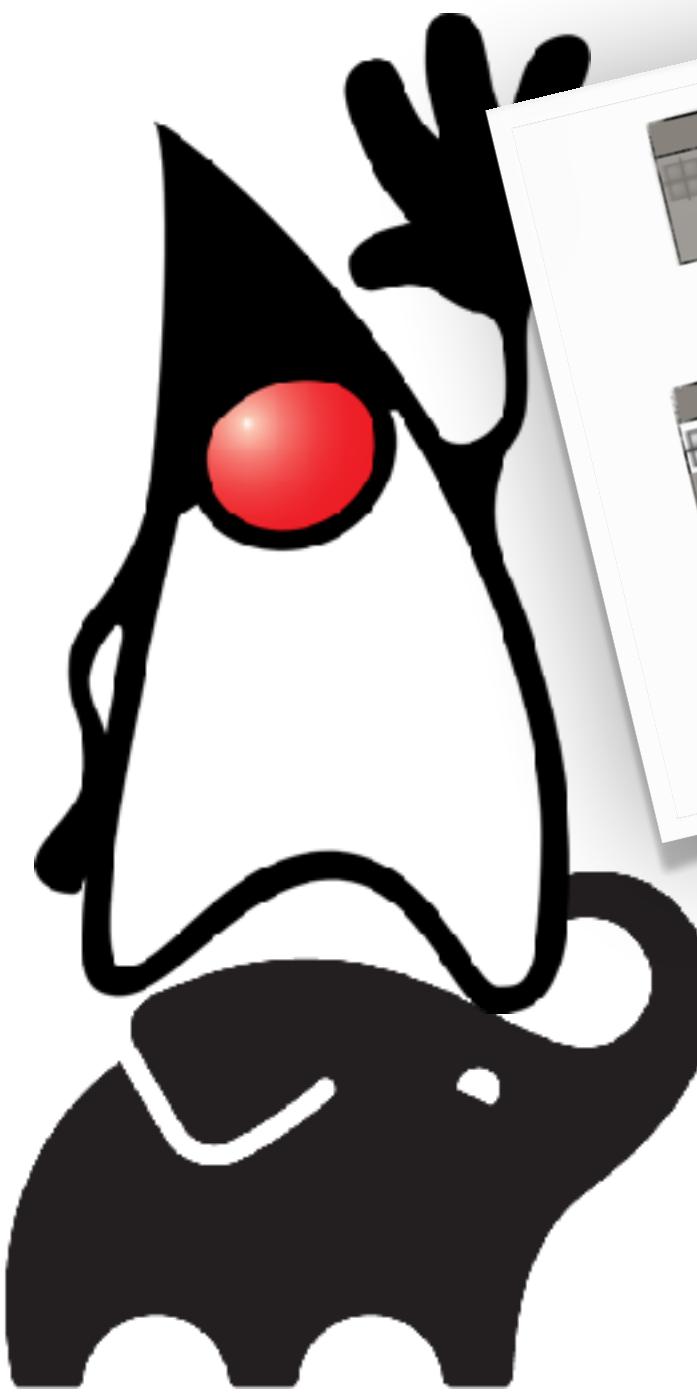


Combining Java Modules and Gradle for elegant project structures

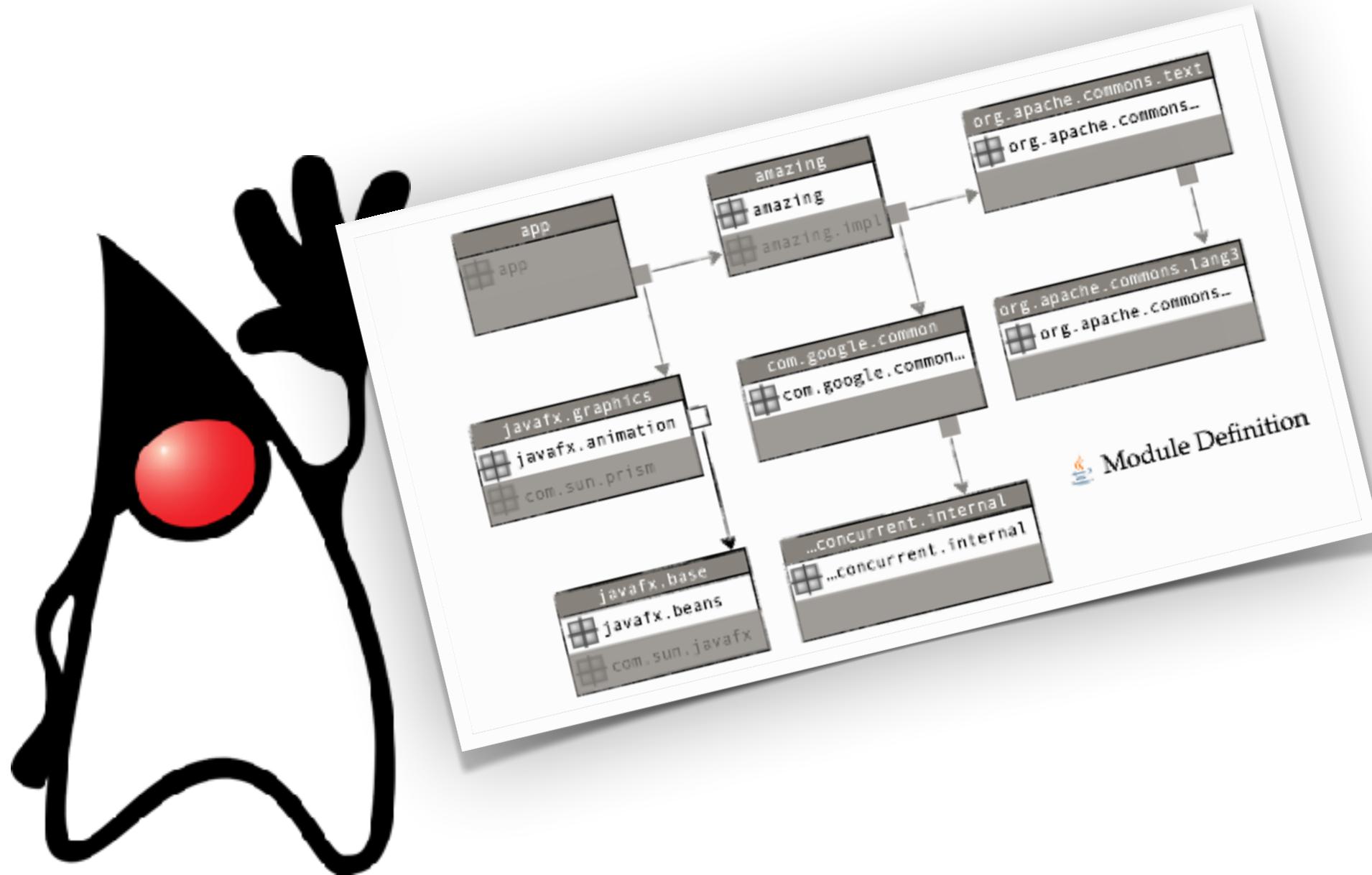


Jendrik Johannes
github.com/jjohannes

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

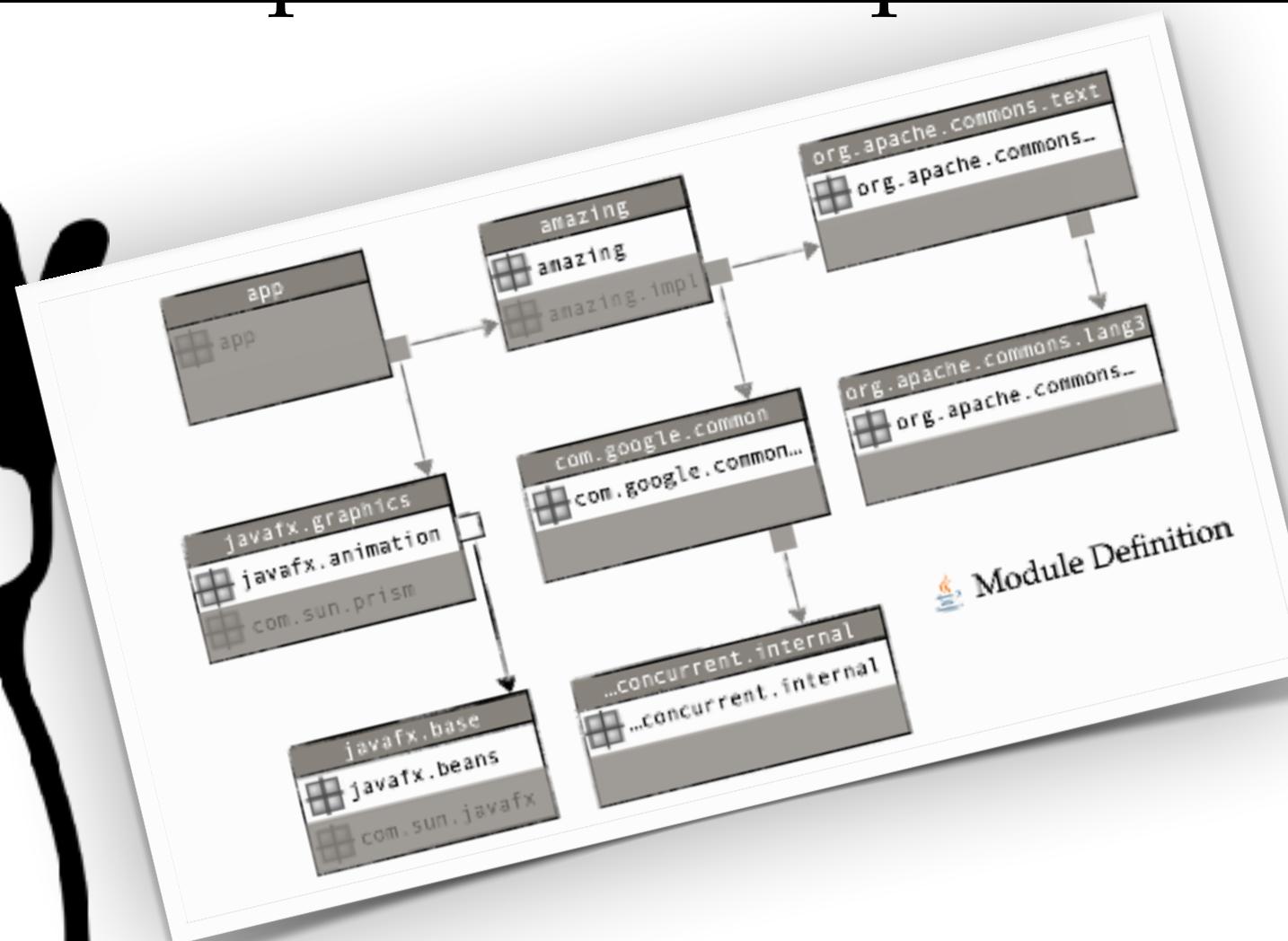
~20 Lines Module Definition

~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)

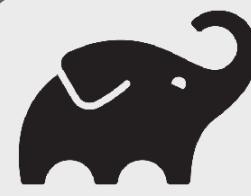


Essential Complexity

- 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/gran...

- 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());
    }
}
```

off

java-module-system – App.java

Project ▾

java-module-system ~/projects/gran...

- 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());
    }
}
```

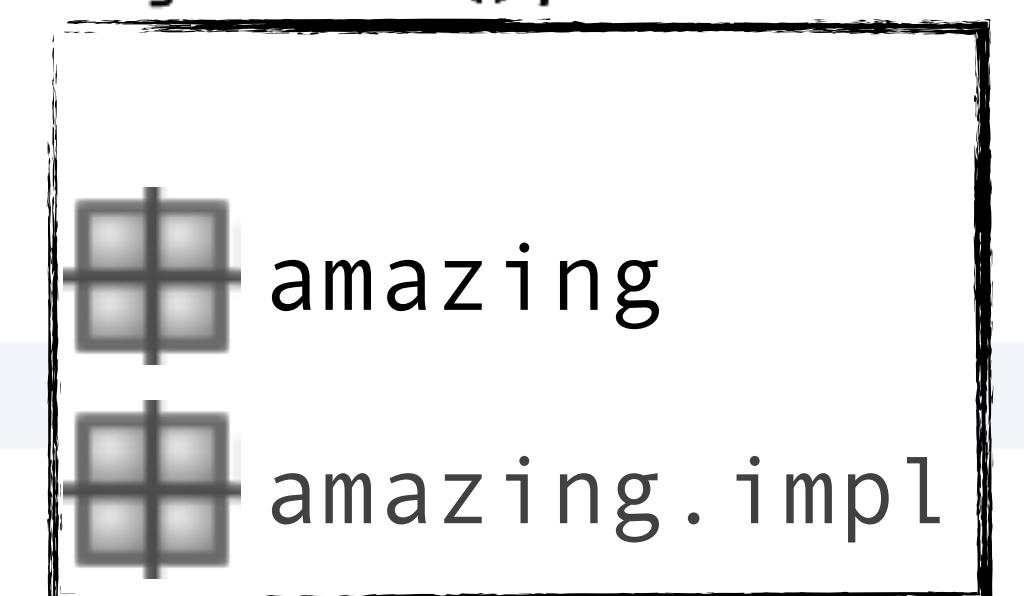
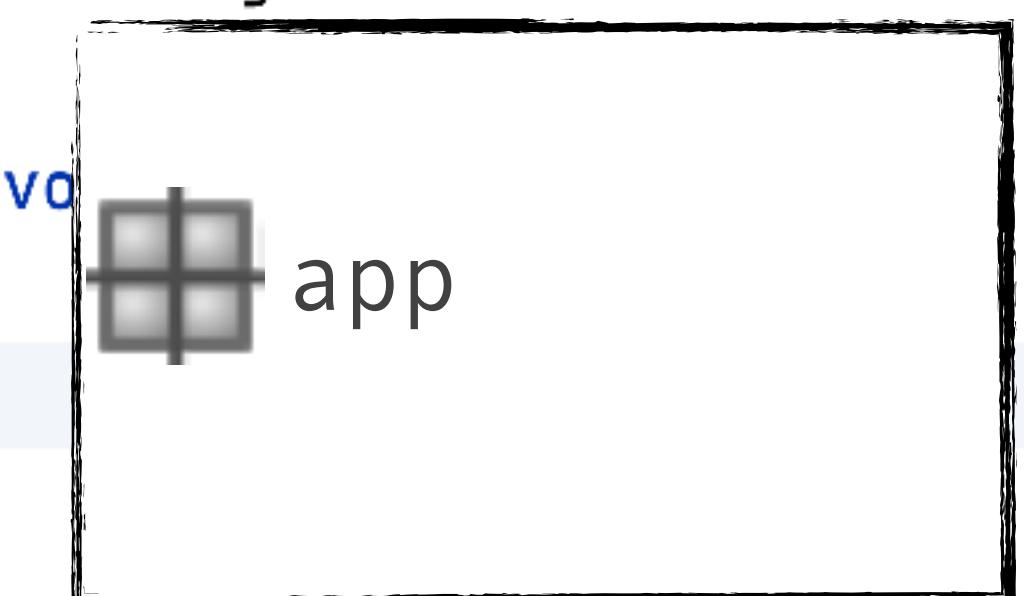


java-module-system – DefaultAmazingService.java

Project ▾

java-module-system ~/projects/gradle/howto/java-mo
modules
 amazing
 src
 main
 java
 amazing
 impl
 DefaultAmazingService.java
 AmazingService.java
 app
 src
 main
 java
 app
 App.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());  
    }  
}  
  
package amazing;  
  
import amazing.impl.DefaultAmazingService;  
  
public interface AmazingService {  
    AmazingService DEFAULT = new DefaultAmazingService();  
    void app();  
}
```



java-module-system – App.java

Project ▾

java-module-system ~/projects/gradle/howto/java-mo

modules

amazing

src

main

java

amazing

impl

DefaultAmazingService.java

AmazingService.java

app

src

main

java

app

App.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();
    }
}
```

The diagram illustrates the module dependency relationship. It consists of three rectangular boxes with black borders. The leftmost box contains the word 'app'. A grey dotted arrow points from it to the middle box, which contains the word 'amazing'. Another grey dotted arrow points from the 'amazing' box to the rightmost box, which contains the words 'amazing' and 'impl' on separate lines.

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

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

Project

java-module-system ~/projects/gradle/howto/java-mo

modules

amazing

src

main

java

amazing

impl

DefaultAmazingService.java

AmazingService.java

module-info.java

app

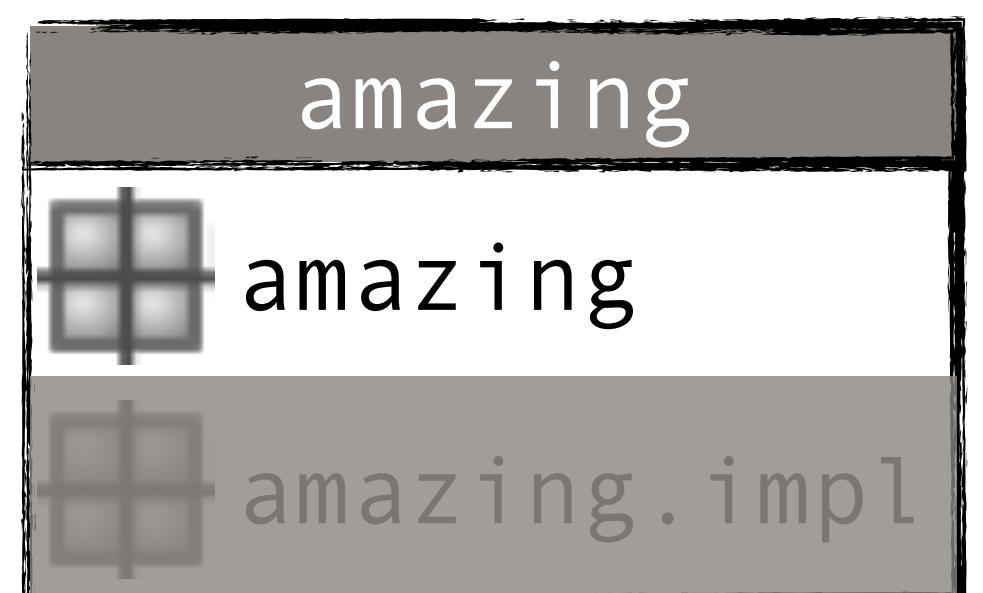
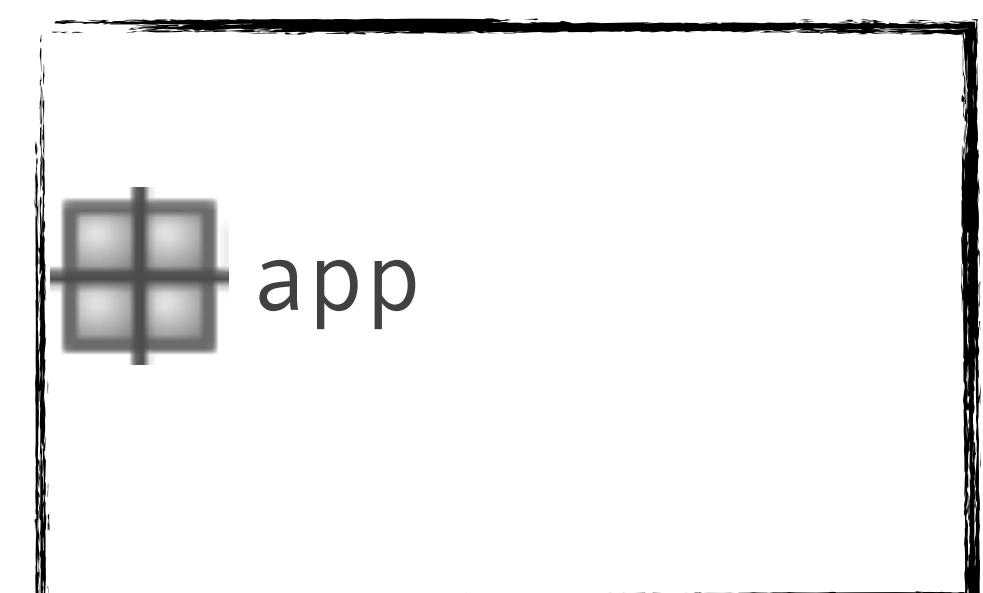
src

main

java

app

App.java



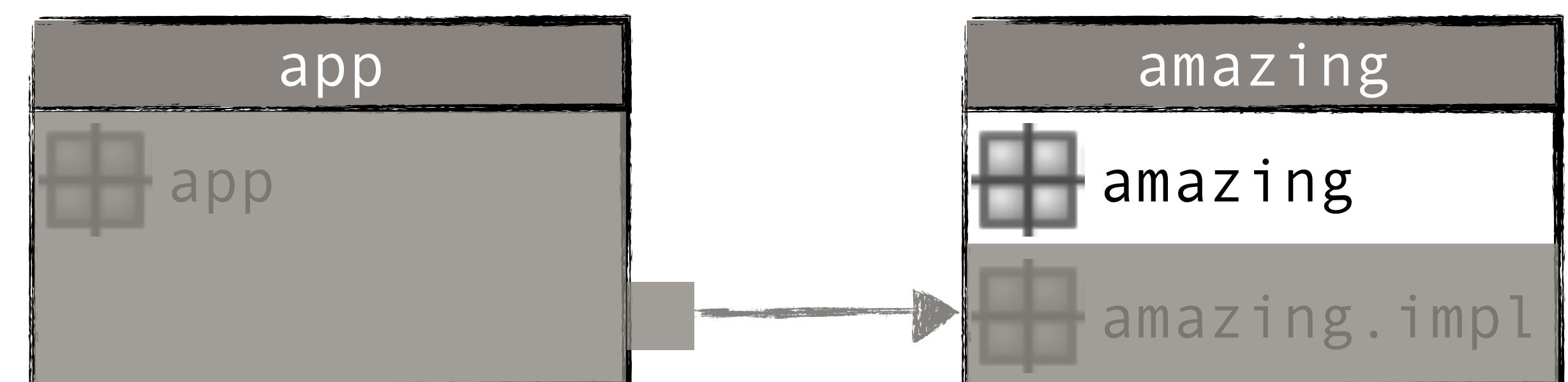
java-module-system – module-info.java (amazing)

Project

```
java-module-system ~/projects/gradle/howto/java-mo  
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-mo

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;  
}
```

module app {

 requires amazing;

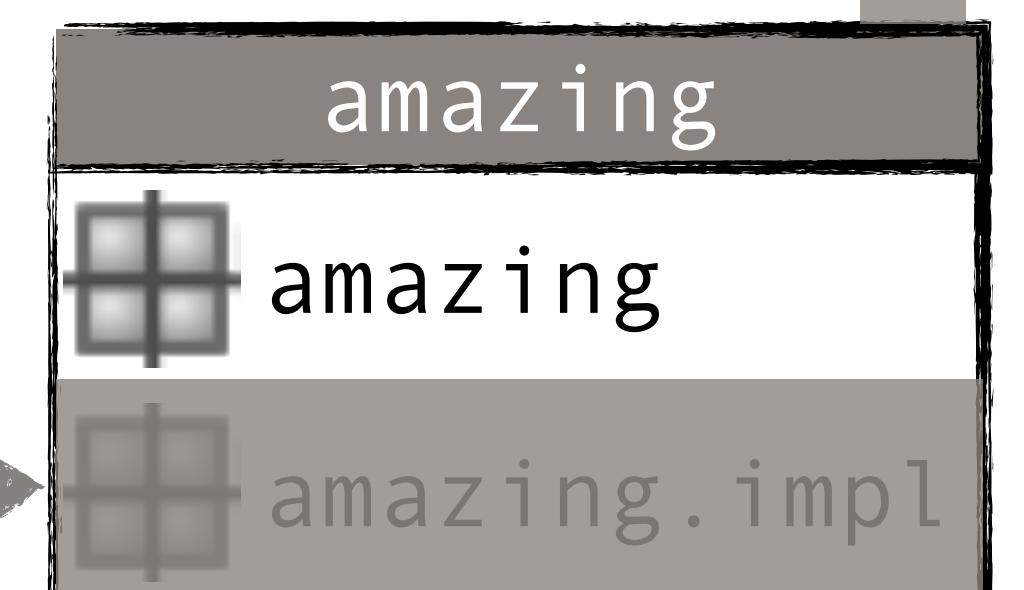
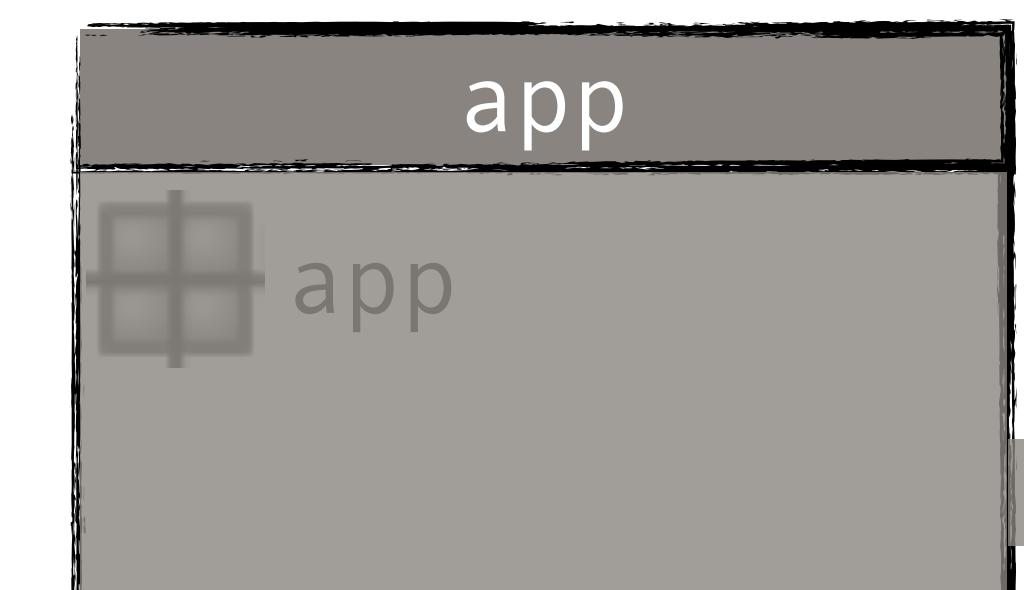
}

}

org.apache.commons.text



org.apache.common...



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

Project

java-module-system ~/projects/gradle/howto/java-mo

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;  
}
```

org.apache.commons.lang3

org.apache.common...

org.apache.commons.text

org.apache.common...

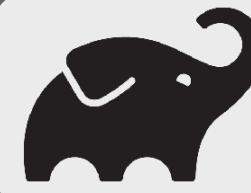
app

app

amazing

amazing

amazing.impl



Accidental Complexity

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



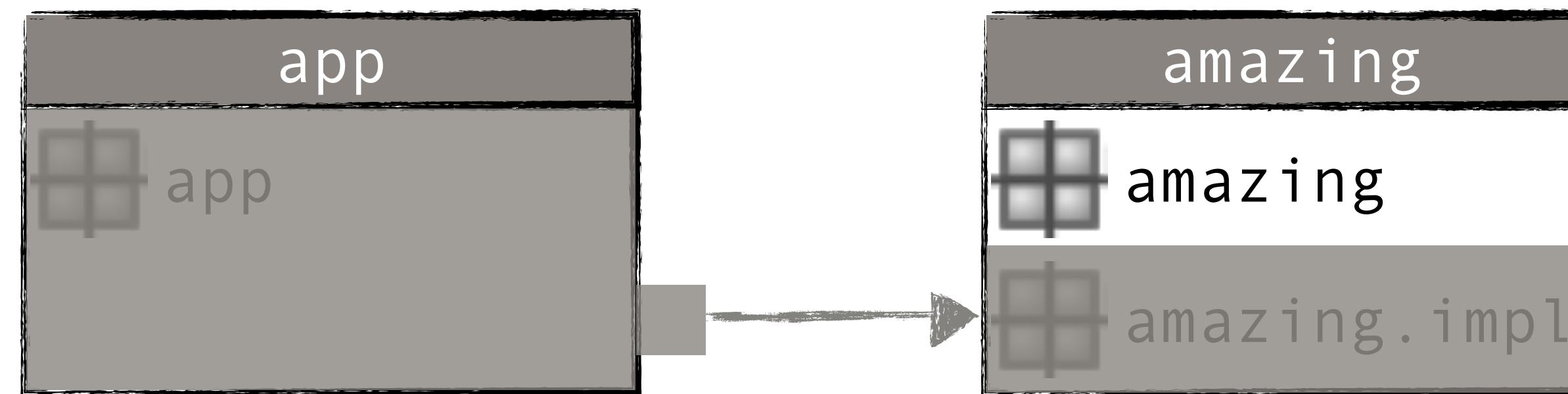
Module Definition



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





1. ~~Mixing Modules from different ecosystems~~ (misunderstood / bad tool support)
2. ~~Different Java compilers~~ (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)

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

lity at
ne (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)

EXAMPLE

add a Gradle-based build system

java-module-system – settings.gradle.kts (java-module-system)

Project

java-module-system ~/projects/gradle/howto/java-module-system

- 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)

```
plugins {
    `kotlin-dsl`
}

repositories {
    gradlePluginPortal()
}

dependencies {
    implementation("com.autonomousapps:dependency-analysis-gradle-plugin:1.32.0")
    implementation("com.gradle:velocity-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

java-module-system ~/projects/gradle
gradle
plugins
src
build.gradle.kts
versions [java-module-system]
build.gradle.kts
modules
settings.gradle.kts



Project Structure

java-module-system – build.gradle.kts (:versions)

```
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")  
}
```

Project ▾

java-module-system ~/projects/gradle
 gradle
 plugins
 src
 build.gradle.kts
 versions [java-module-system]
 build.gradle.kts
 modules
 settings.gradle.kts



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

java-module-system – compile-and-test.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("java")  
    id("org.gradlex.java-module-testing")  
}  
  
java {  
    toolchain.languageVersion = JavaLanguageVersion.of(21)  
}  
  
tasks.withType<JavaCompile>().configureEach {  
    options.encoding = "UTF-8"  
}  
  
tasks.withType<Test>().configureEach {  
    maxParallelForks = 4  
    maxHeapSize = "1g"  
}
```



Other Build Concerns

java-module-system – targets.gradle.kts [plugins.main]

Project

java-module-system ~/projects/gradle/howto/java-mod
 └── gradle
 └── plugins
 └── main
 └── kotlin
 └── 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-14"))
```



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.gradle
 compile-and-test.gradle.kts
 dependency-analysis.gradle.kts
 java-module.gradle.kts
 metadata-patch.gradle.kts
 module-locations.settings.gradle
 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.MACOS_ARM)
                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:failureaccess", "com.google.errorprone")
}
```





Accidental Complexity

1. Mixing Module Definition and other code (misunderstood / bad tool support)
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. Incompletely specified API by Module authors (misunderstood / bad tool support)



Module Definition



Module Discovery

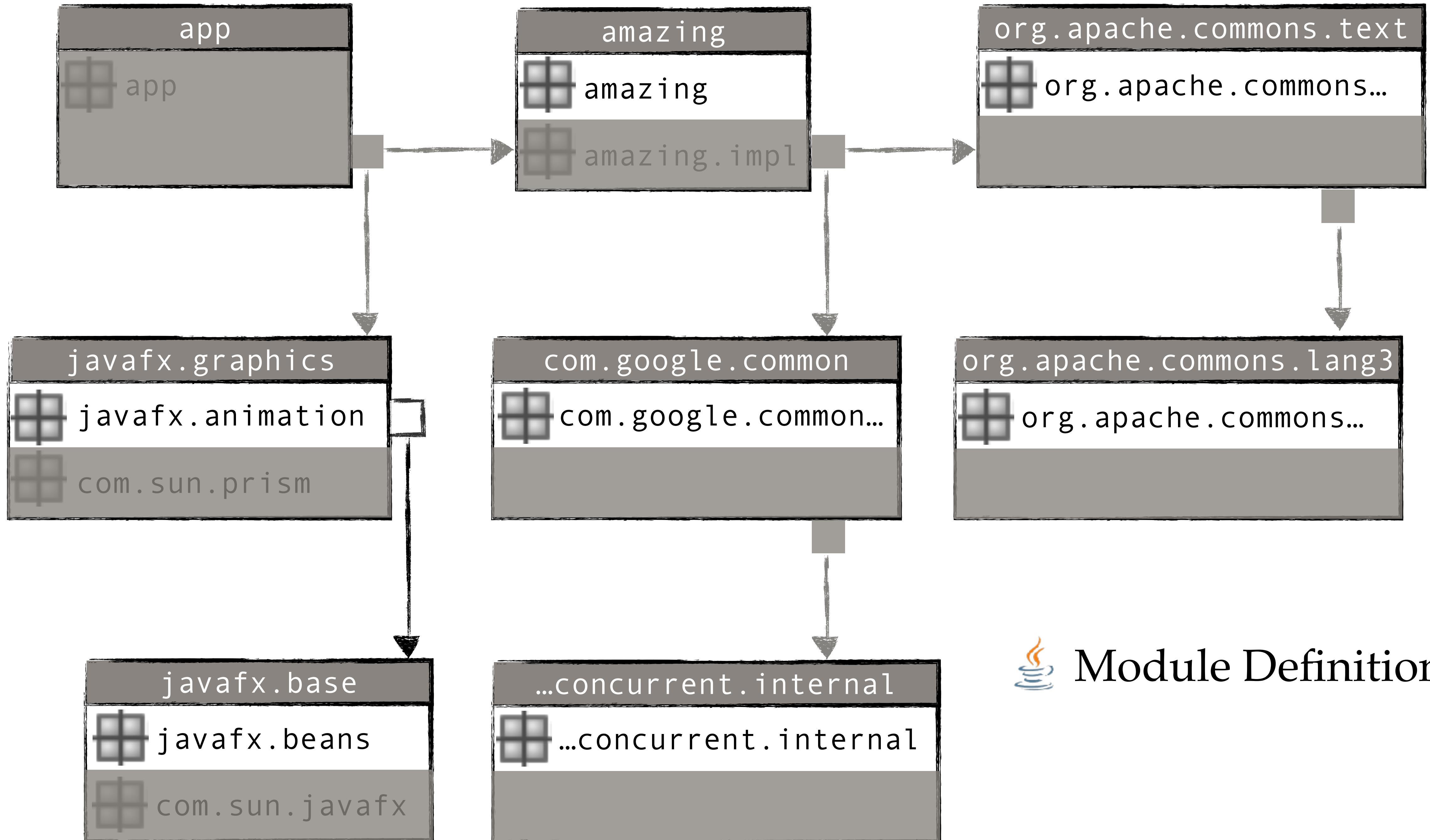


Module Patching

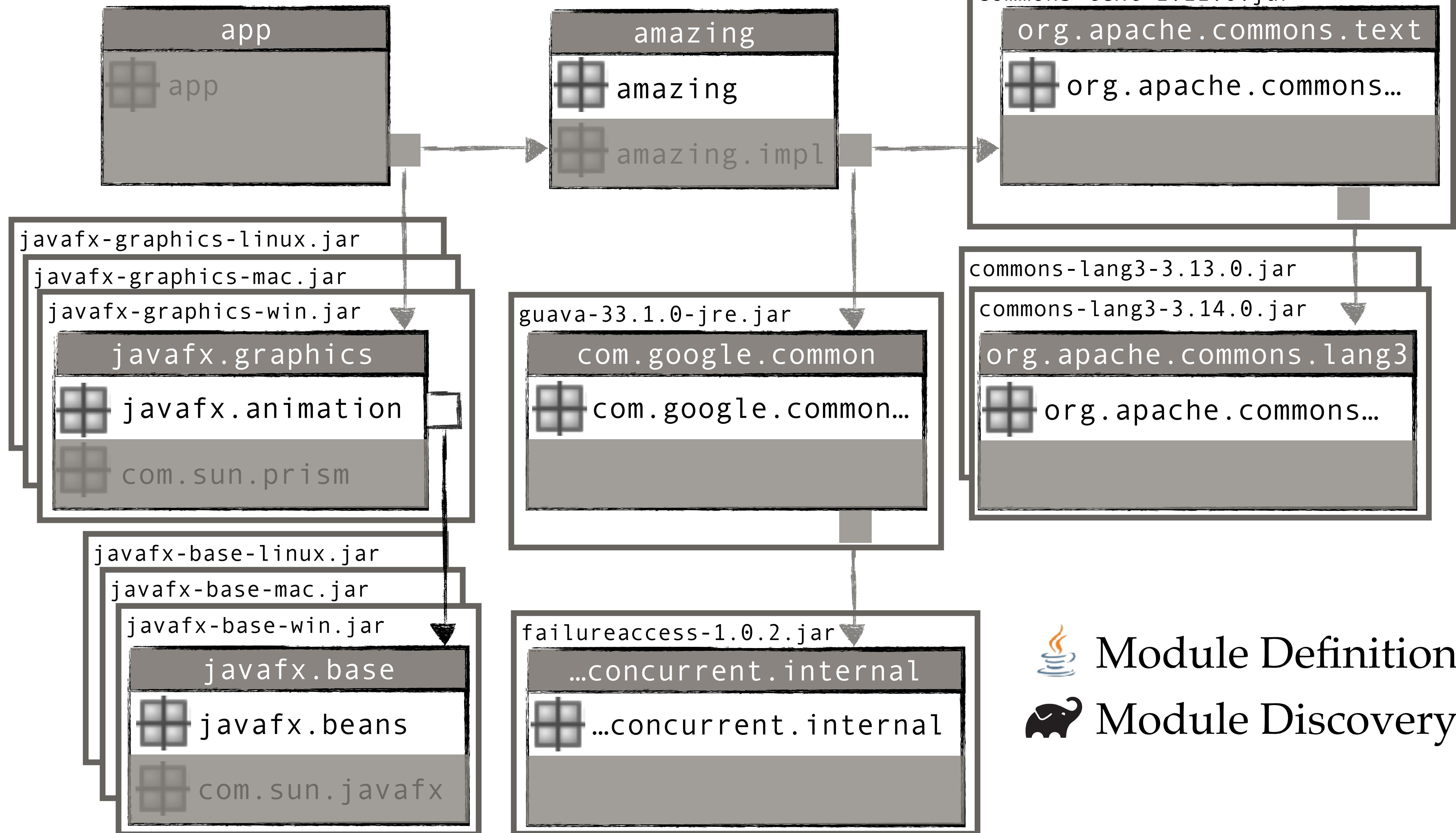


Essential Complexity

- 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)



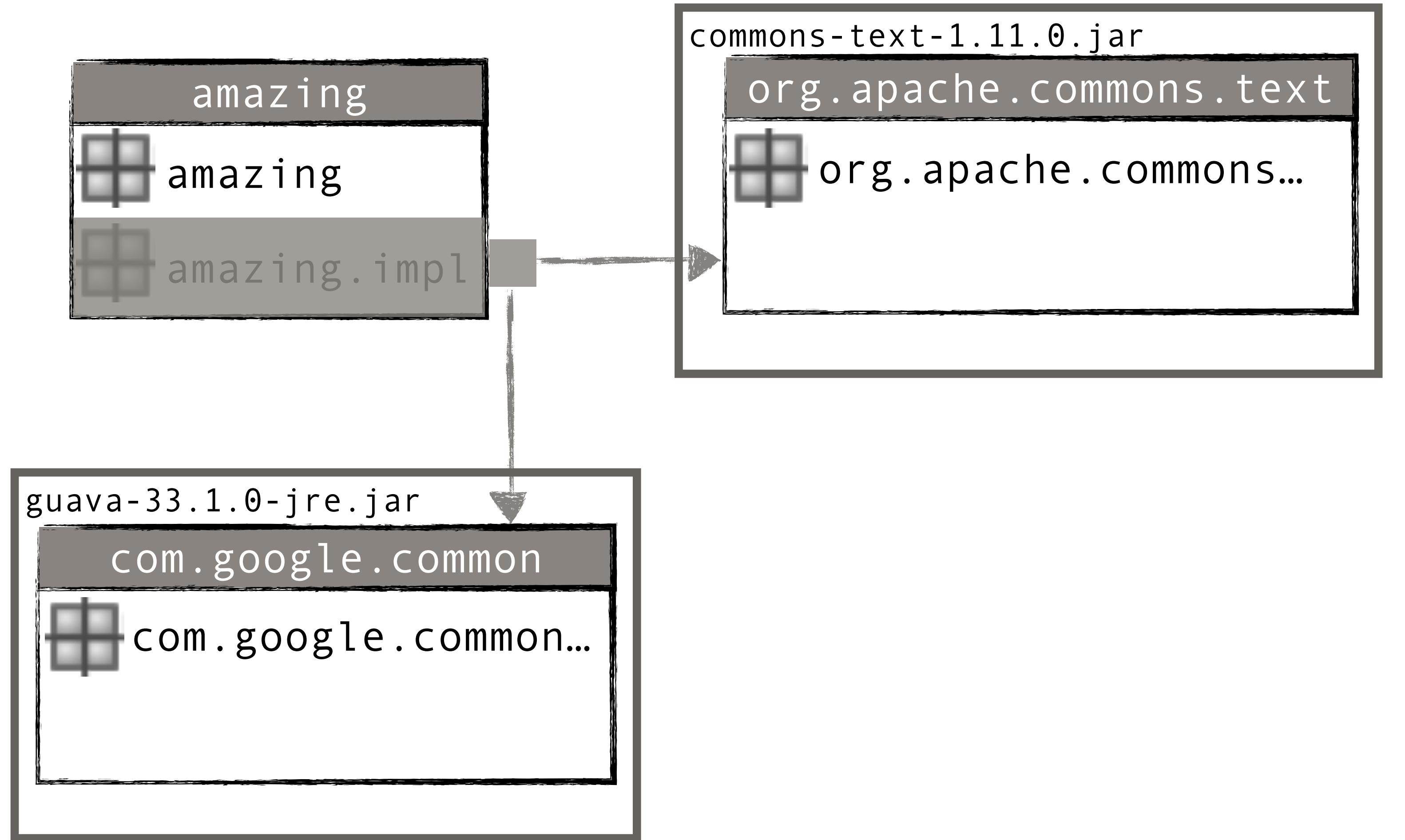
Module Definition



Module Definition

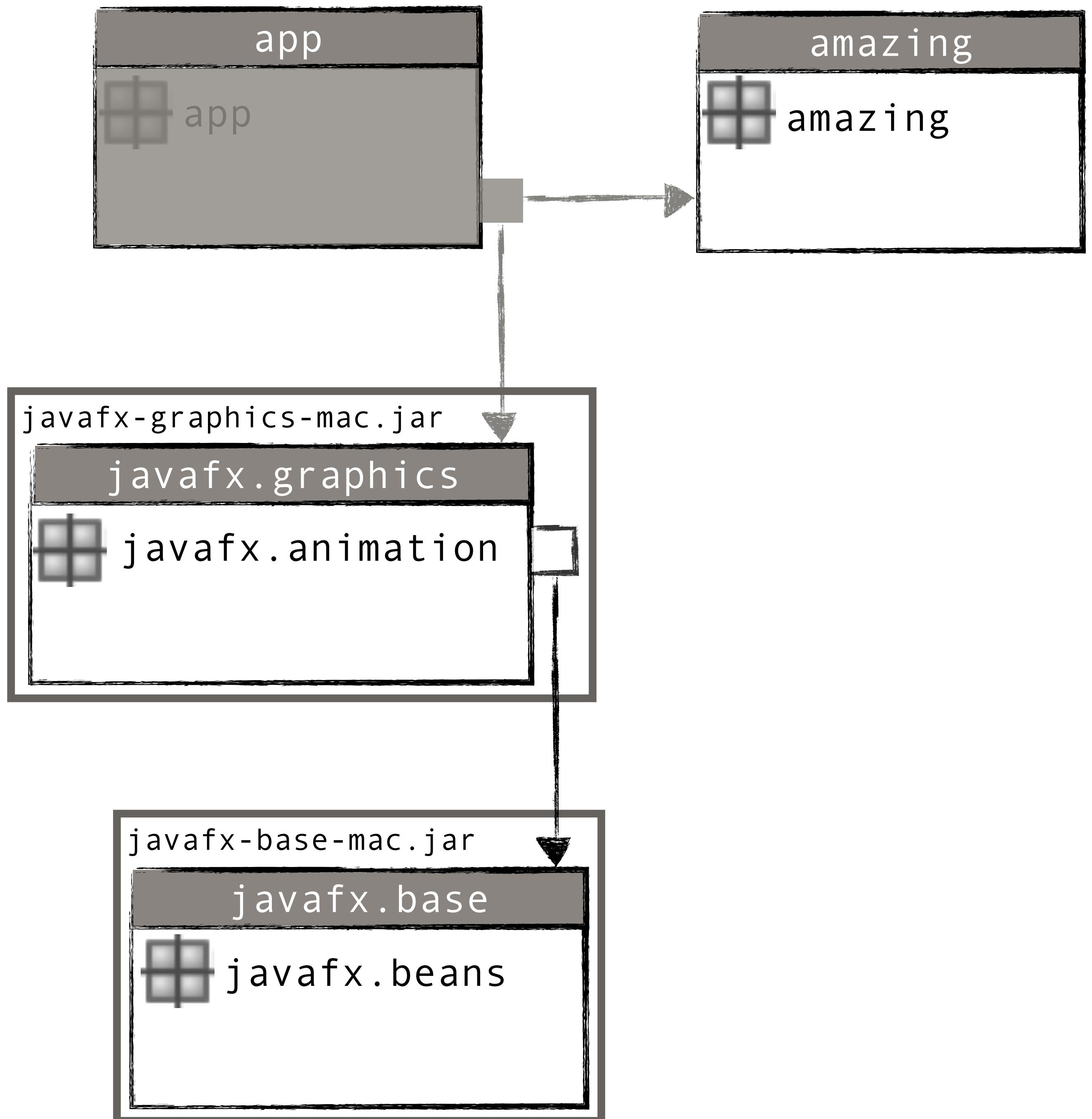


Module Discovery

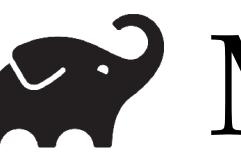


Module Definition
 Module Discovery

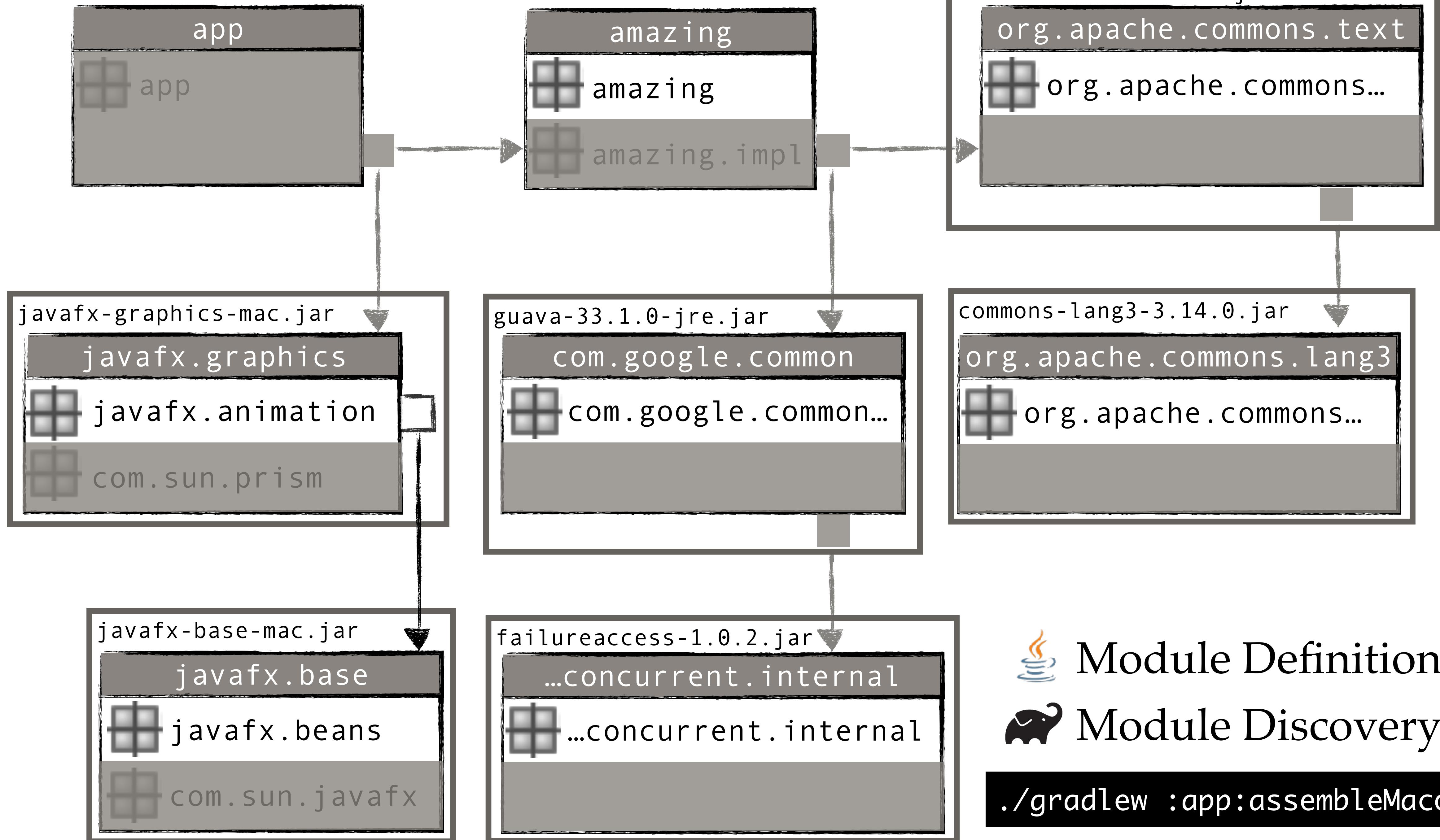
```
./gradlew :amazing:compileJava
```



 Module Definition

 Module Discovery

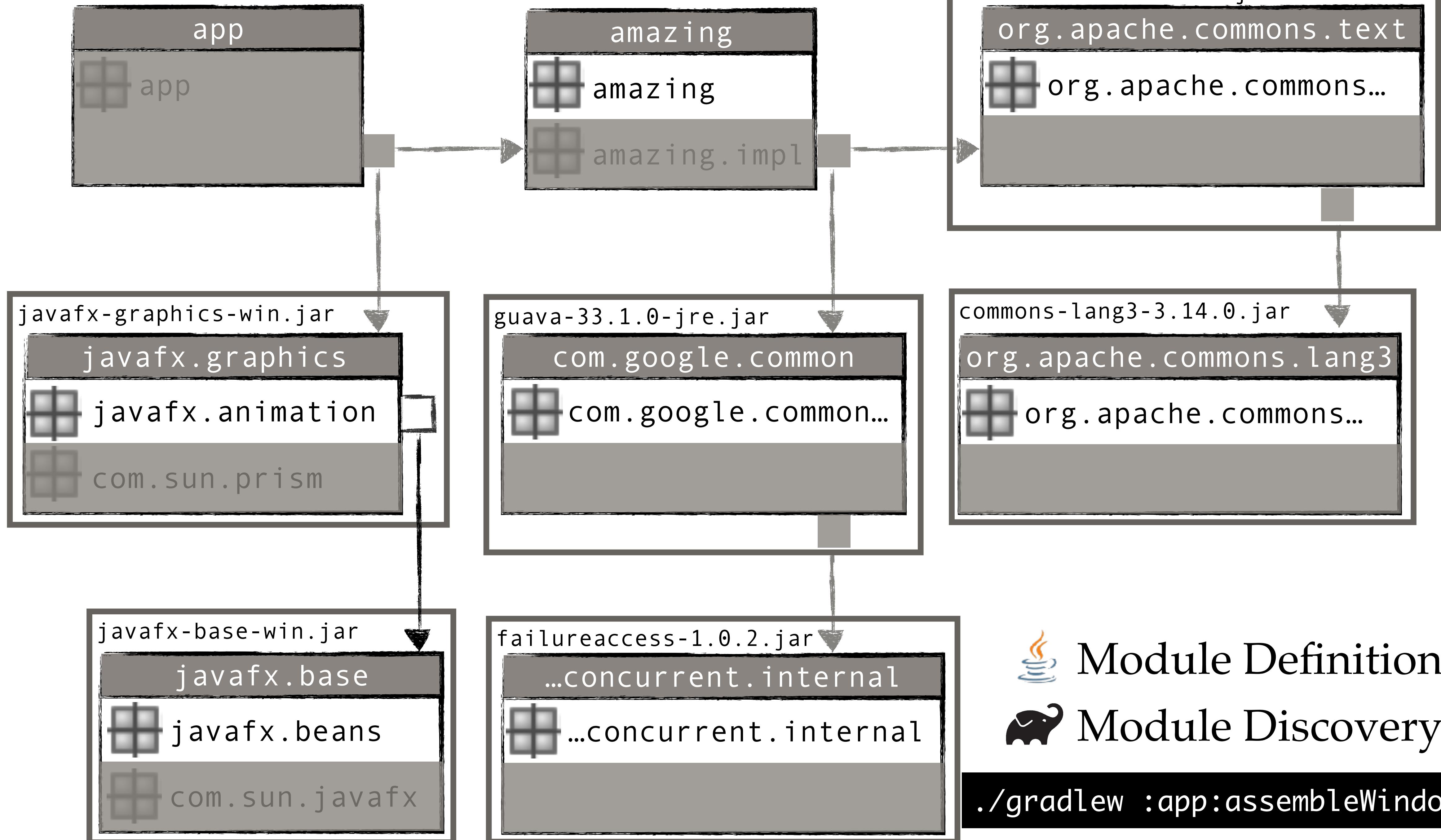
```
./gradlew :app:compileJava
```



Module Definition

Module Discovery

`./gradlew :app:assembleMacos`



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

The screenshot shows a YouTube channel page for 'onepiece.Software by Jendrik Johannes'. The channel has 1340 subscribers. The main navigation bar includes 'ÜBERSICHT', 'VIDEOS', 'PLAYLISTS', 'COMMUNITY', 'KANÄLE', and 'KANALINFO'. Below the navigation, a section titled 'Java Modularity' lists four video thumbnails:

- 'Understanding Gradle #26 Java Modularity (Part 1)' - 18:15
- 'Understanding Gradle #27 Java Modularity Multiple Compile Classpaths' - 18:20
- 'Understanding Gradle #28 (Part 3) Java Modularity Clean Compile Classpaths with the Dependency Analysis Plugin' - 13:11
- 'Understanding Gradle #29 (Part 4) Java Modularity Detect and Resolve Collisions in Classpath' - 13:11

Below the thumbnails, there are two rows of video cards:
Row 1:

- 'Understanding Gradle #26 – The Classpath' - 18:15
- 'Understanding Gradle #27 – Multiple Compile Classpaths' - 18:20
- 'Understanding Gradle #28 – Clean Compile Classpaths...' - 13:11
- 'Understanding Gradle #29 – Detect and Resolve Coll...' - 13:11

Row 2:

- 'onepiece.Software by Jendrik Jo...' - 559 Aufrufe • vor 2 Monaten
- 'onepiece.Software by Jendrik Jo...' - 280 Aufrufe • vor 1 Monat
- 'onepiece.Software by Jendrik Jo...' - 297 Aufrufe • vor 2 Wochen
- 'onepiece.Software by Jendrik Jo...' - 95 Aufrufe • vor 1 Tag

The screenshot shows the GitHub profile for 'jjohannes'. The top navigation bar includes 'Overview', 'Repositories 35', 'Projects', 'Packages', and 'Stars 52'. The 'Overview' tab is selected. Below the navigation, there are two sections:

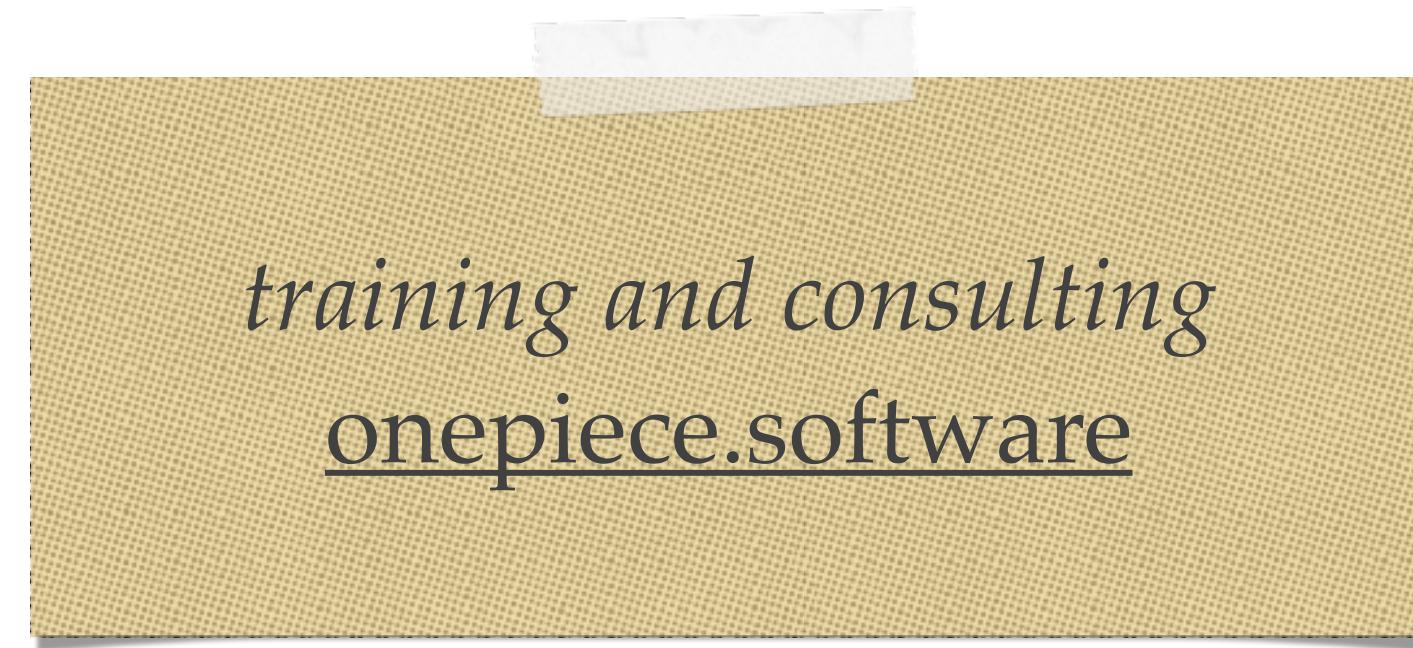
Gradle How-To GitHub repositories

- gradle-plugins-howto How to write Gradle plugins - answers to questions and alternative implementation solutions
- gradle-project-setup-howto How to structure a growing Gradle project with smart dependency management?
- idiomatic-gradle How to idiomatically structure a large build
- gradle-demos A collection of samples demonstrating how to do different things in Gradles

Gradle Plugins maintained by me in the GradleX project

- build-parameters Compile-safe access to parameters supplied to a Cradle build
- java-ecosystem-capabilities Adds Capabilities to well-known Components hosted on Maven Central
- extra-java-module-info Add Java Module information to legacy Java libraries
- java-module-dependencies Makes Gradle respect the dependencies defined in 'module-info.java' files
- java-module-testing Test Java Modules (whitebox and blackbox) without the hassle

youtube.com/@jjohannes



github.com/jjohannes