# Working with the Java Module System (Java SE 11 Developer Certification 1Z0-819)

## Introducing the Java Module System

**Sander Mak**

Java Champion

@Sander_Mak   www.javamodularity.com

## Who Is This Course For?

## Who Is This Course For?

**Experienced Java Developers**

## Who Is This Course For?

**Experienced Java Developers**

**Take the Java SE 11 Certification**

## Who Is This Course For?

**Experienced Java Developers**

**Take the Java SE 11 Certification**

**Modular Software Development**

## Course Overview

## Course Overview

Introducing the Java Module System

## Course Overview

Introducing the Java Module System

Working with Modules

## Course Overview

Introducing the Java Module System

Working with Modules

Understanding the Modular JDK

## Course Overview

Introducing the Java Module System

Working with Modules

Understanding the Modular JDK

Using Services

## Course Overview

Introducing the Java Module System

Working with Modules

Understanding the Modular JDK

Using Services

Migrating to Modules

## Follow Along

**Download JDK 11**

**adoptopenjdk.net**

## Follow Along

**Download JDK 11**

**adoptopenjdk.net**

What's New in Java 11

## What is a Module?

## Module

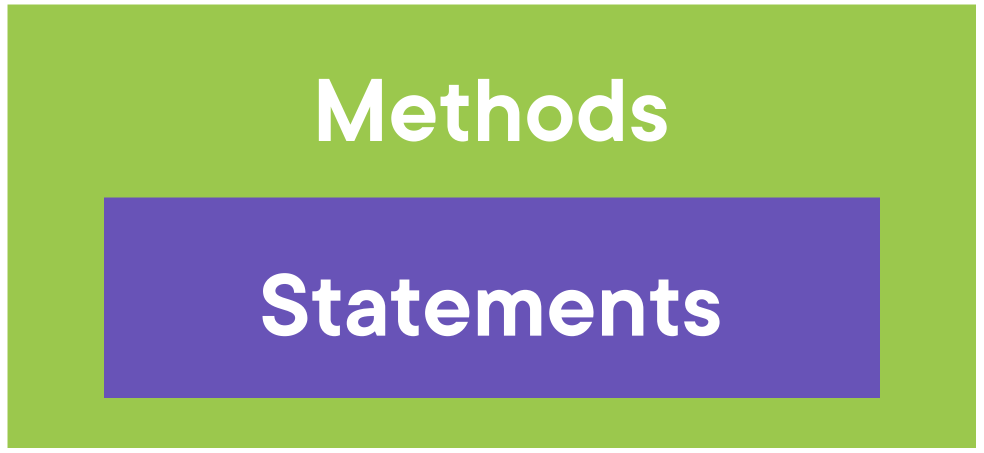
A module has a **name**, it **groups** related code and is **self-contained**

## Modules: Next Level of Abstraction for Java

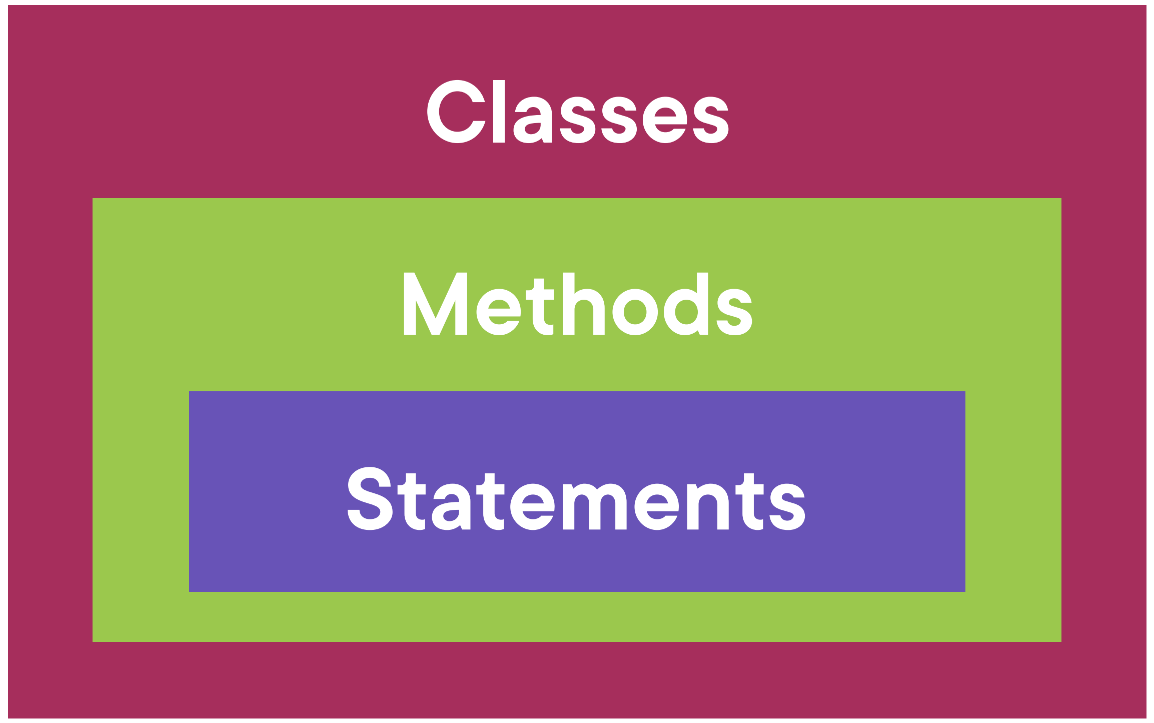
## Modules: Next Level of Abstraction for Java

Statements

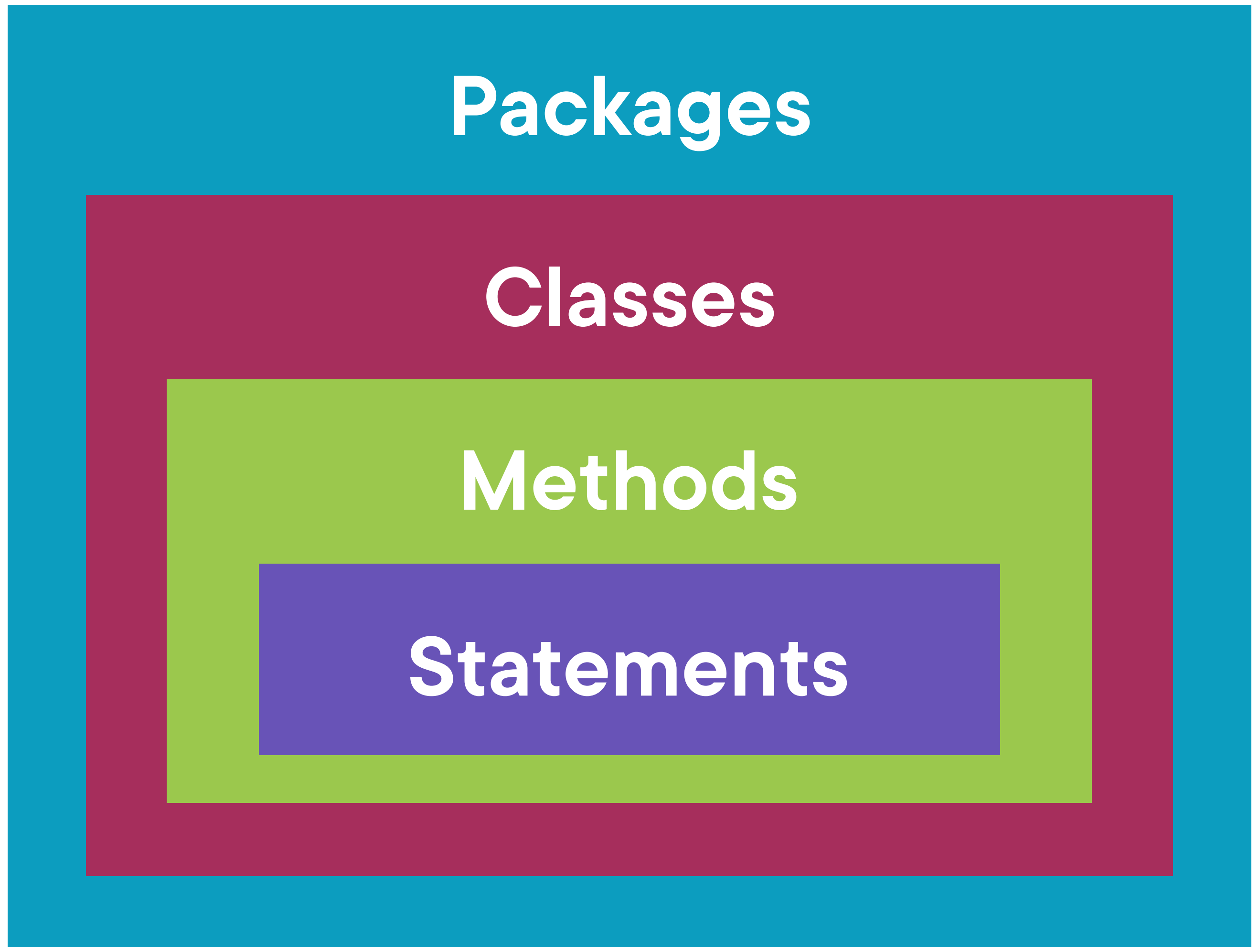
## Modules: Next Level of Abstraction for Java



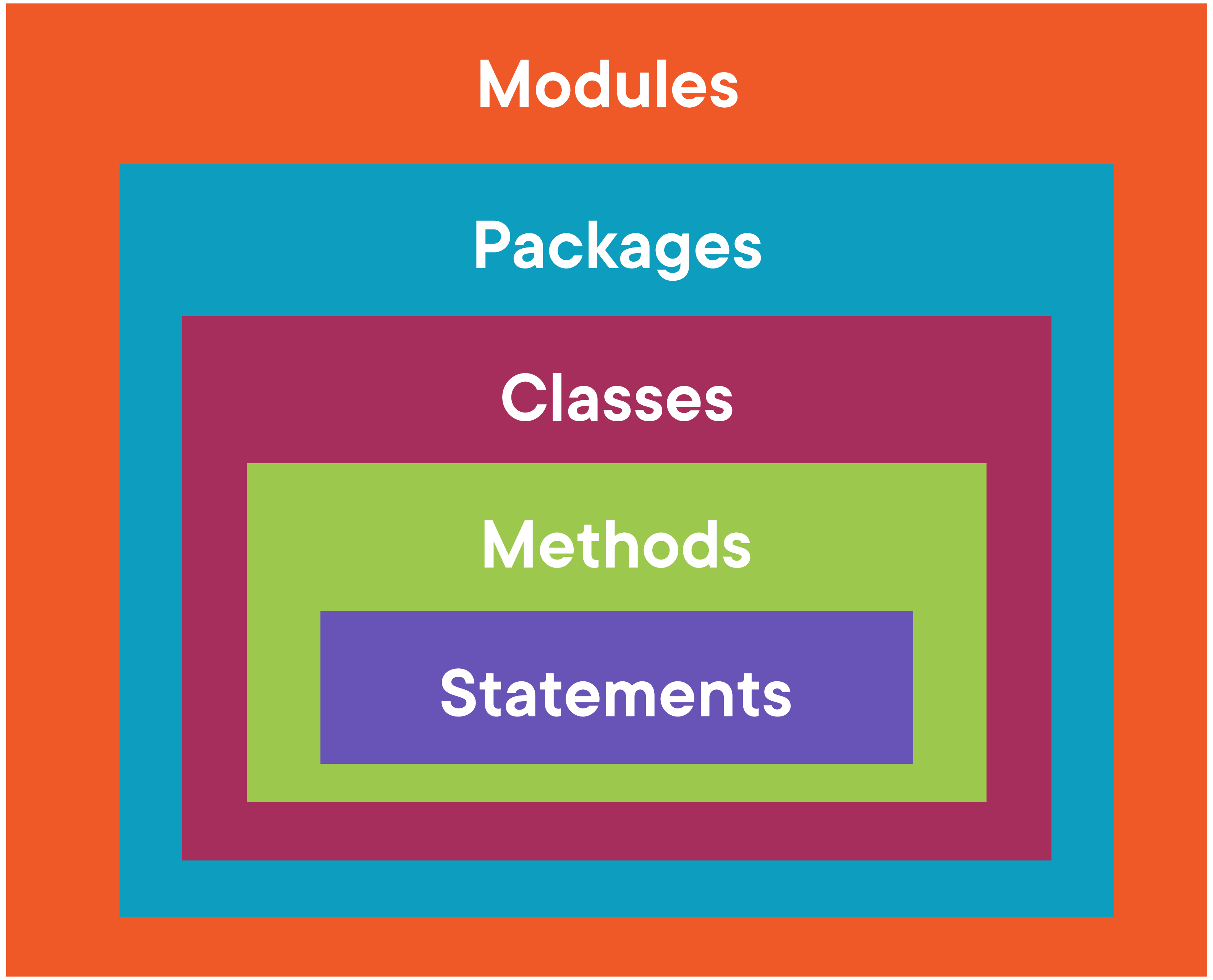
## Modules: Next Level of Abstraction for Java



## Modules: Next Level of Abstraction for Java

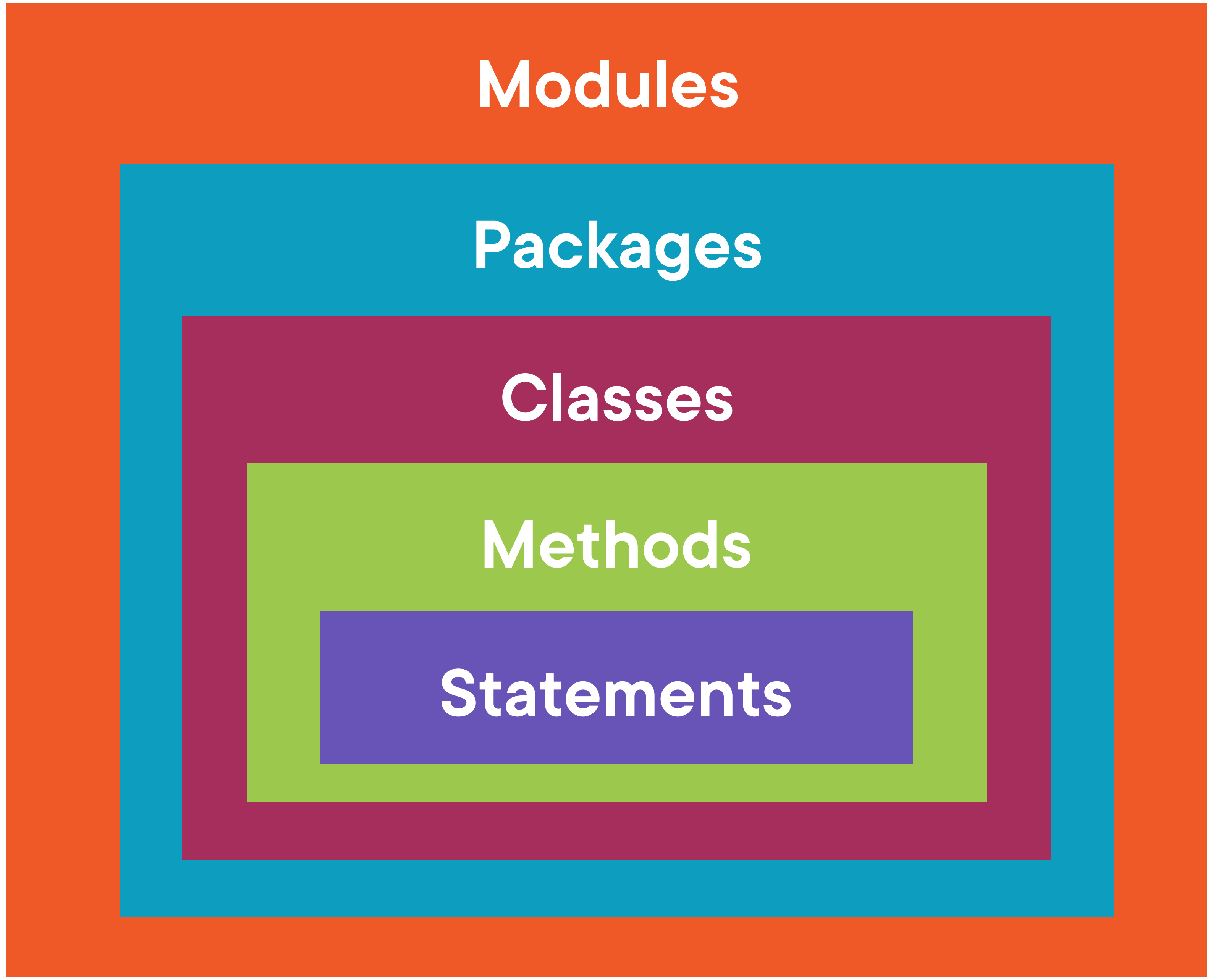


# Modules: Next Level of Abstraction for Java



## Modules: Next Level of Abstraction for Java

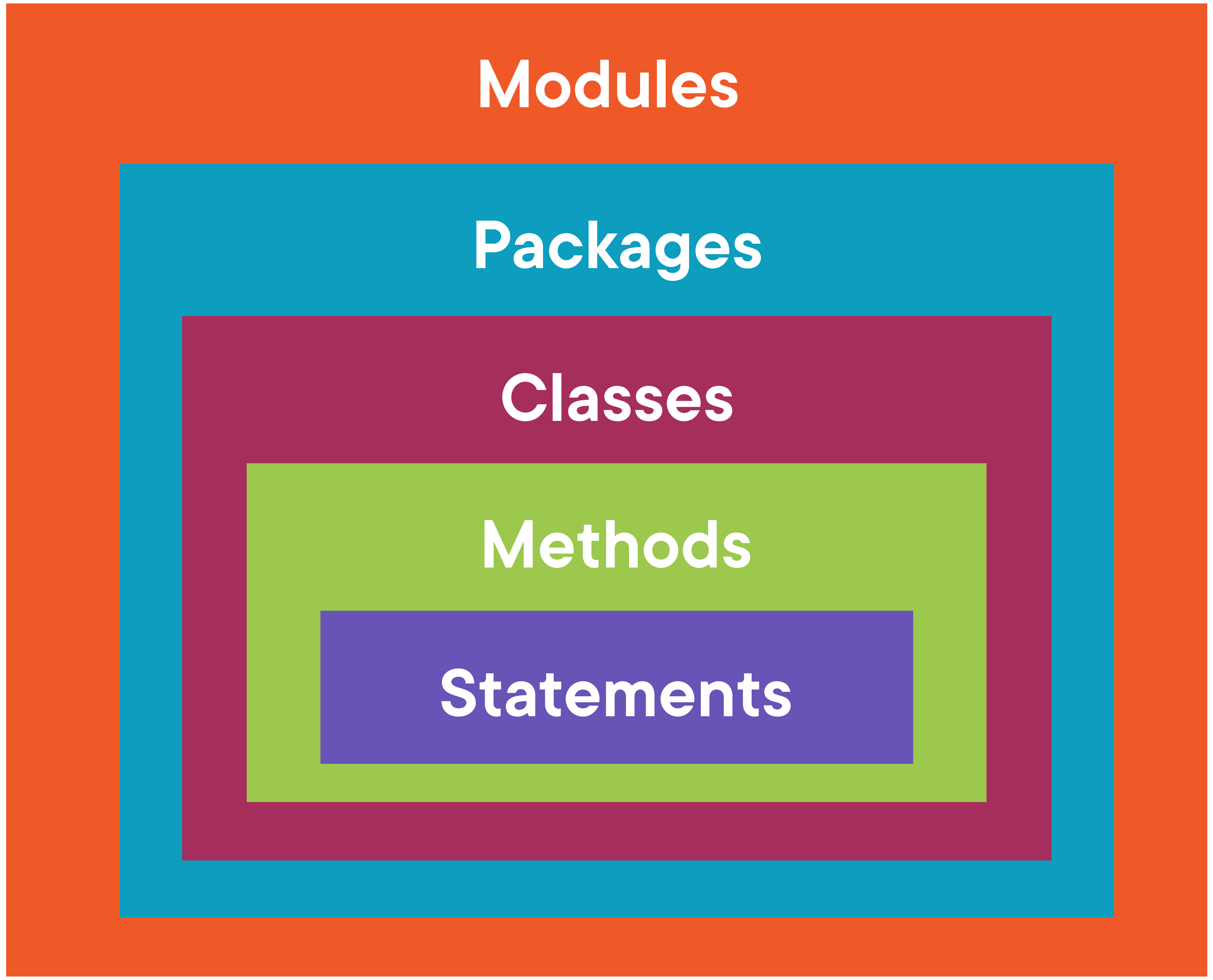
**A module:**



# Modules: Next Level of Abstraction for Java

**A module:**

**Groups related packages**

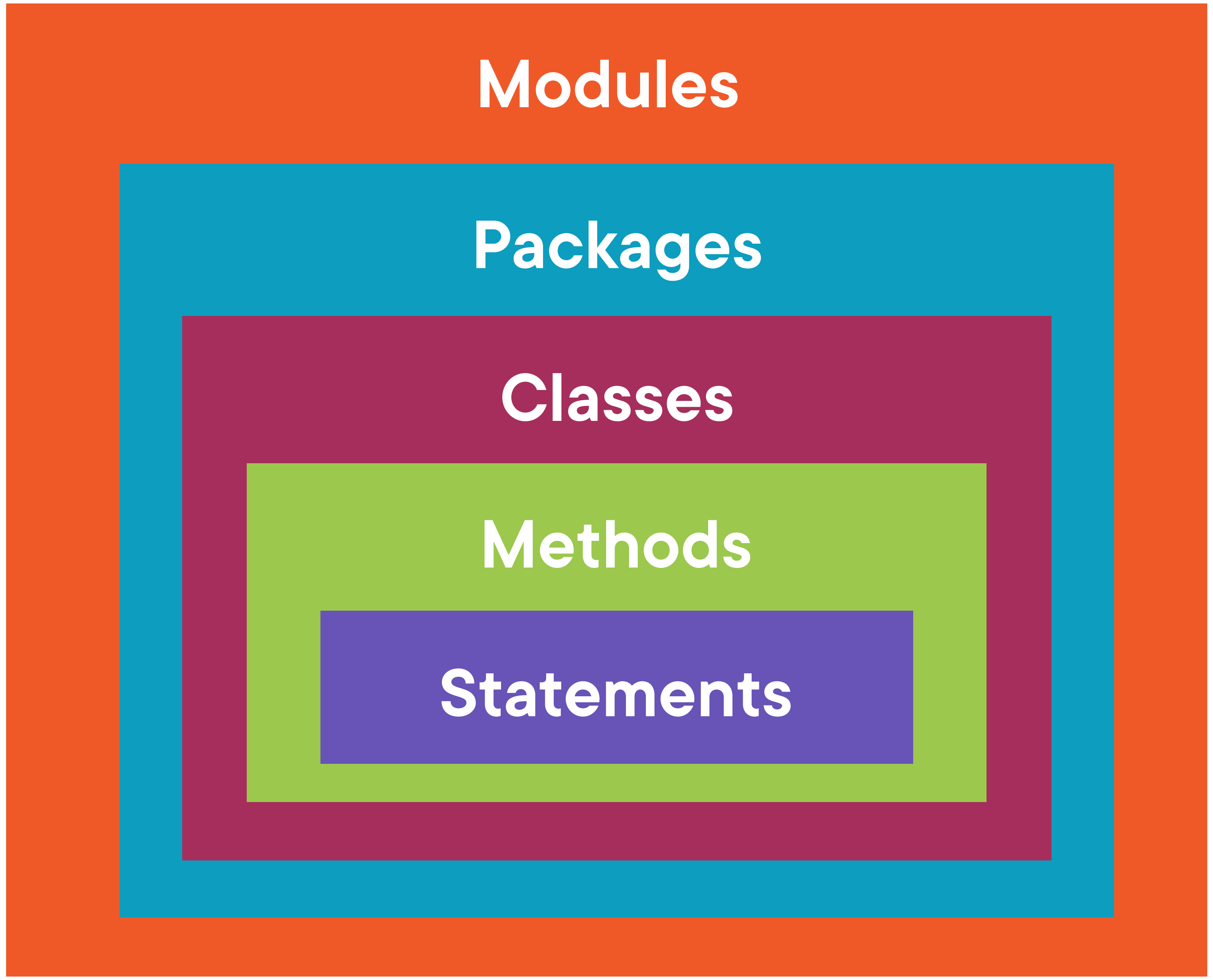


## Modules: Next Level of Abstraction for Java

**A module:**

Groups related packages

**Has a name**



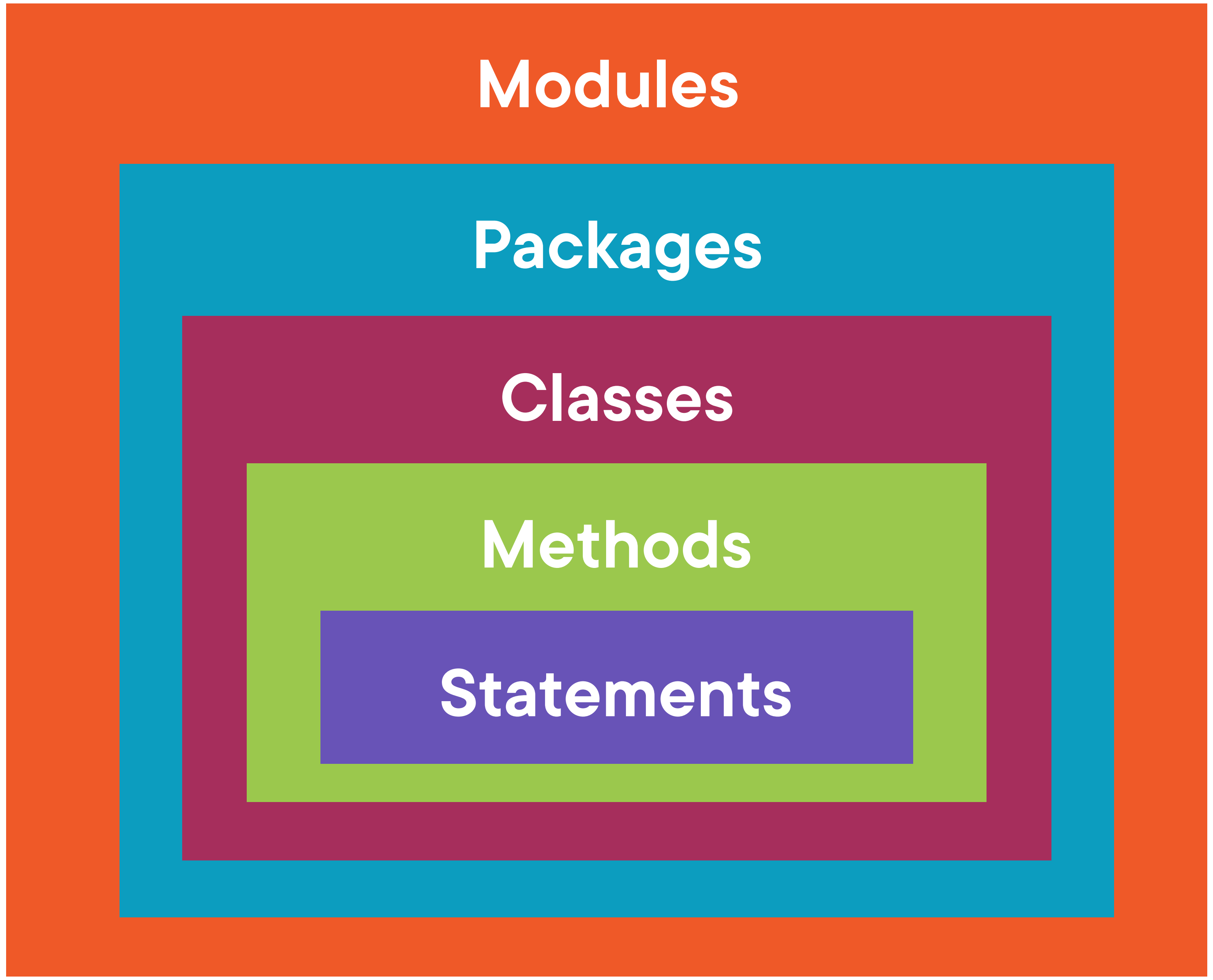
## Modules: Next Level of Abstraction for Java

**A module:**

Groups related packages

Has a name

Controls access



## Modules: Next Level of Abstraction for Java

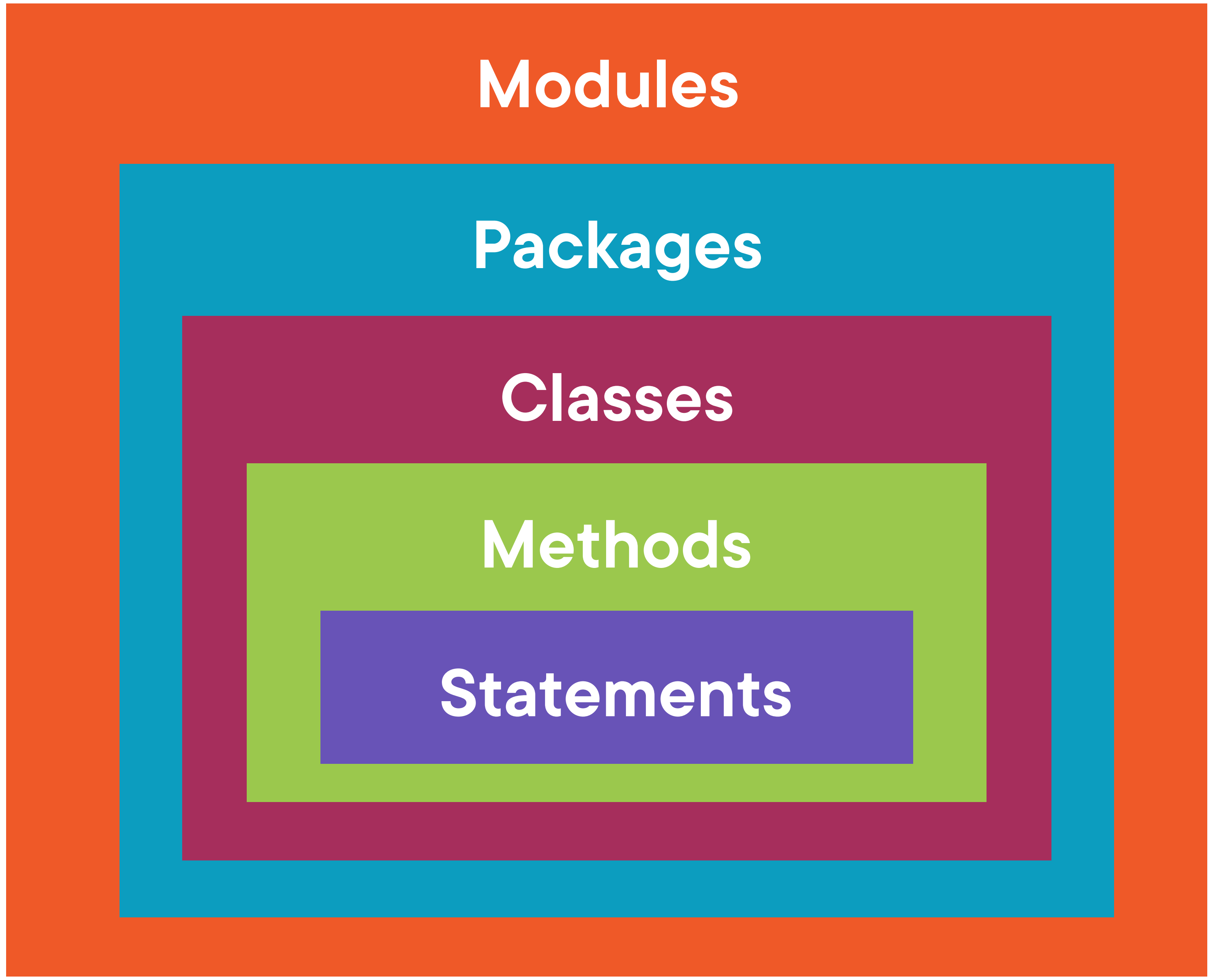
**A module:**

**Groups related packages**

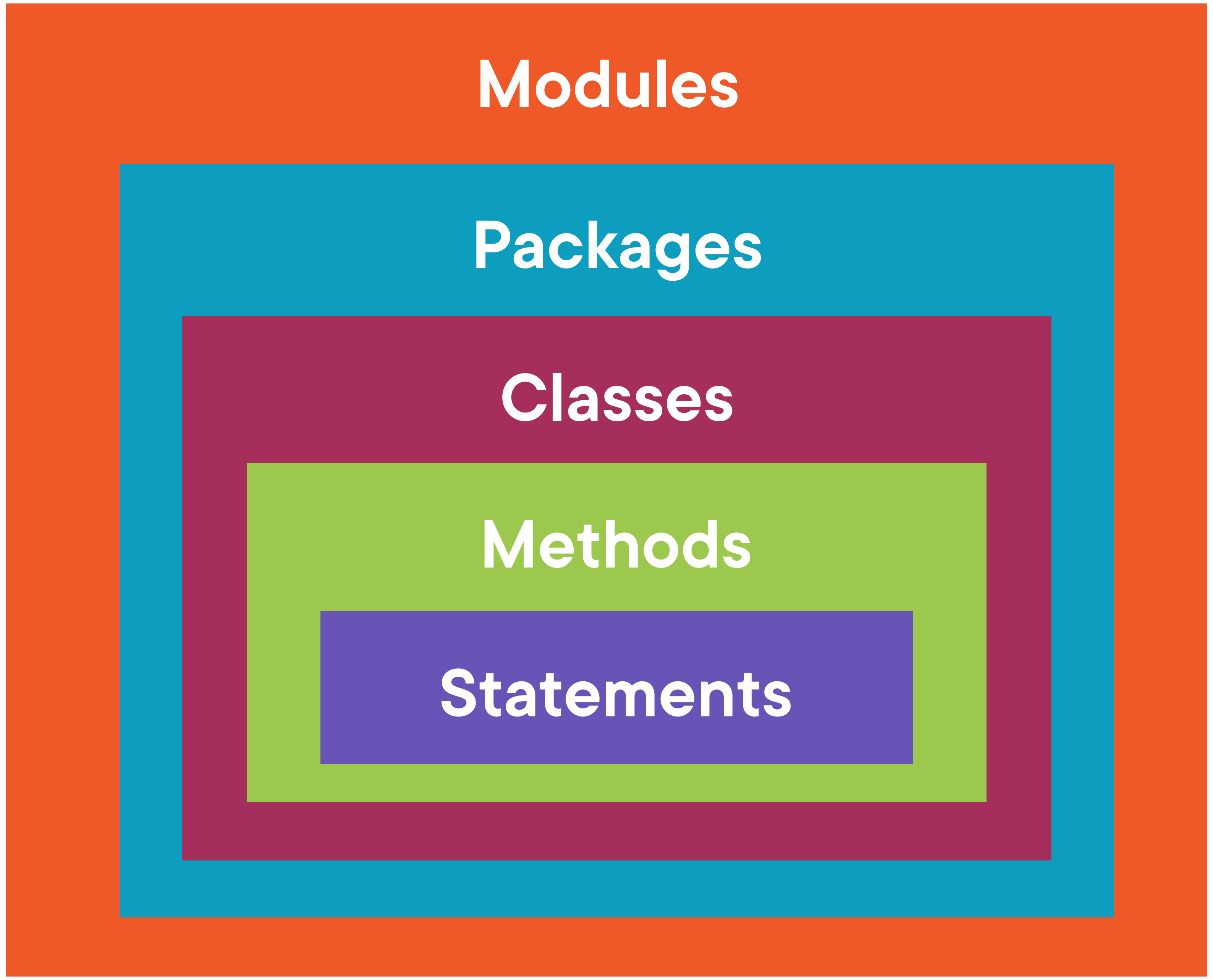
**Has a name**

**Controls access**

**Describes dependencies**

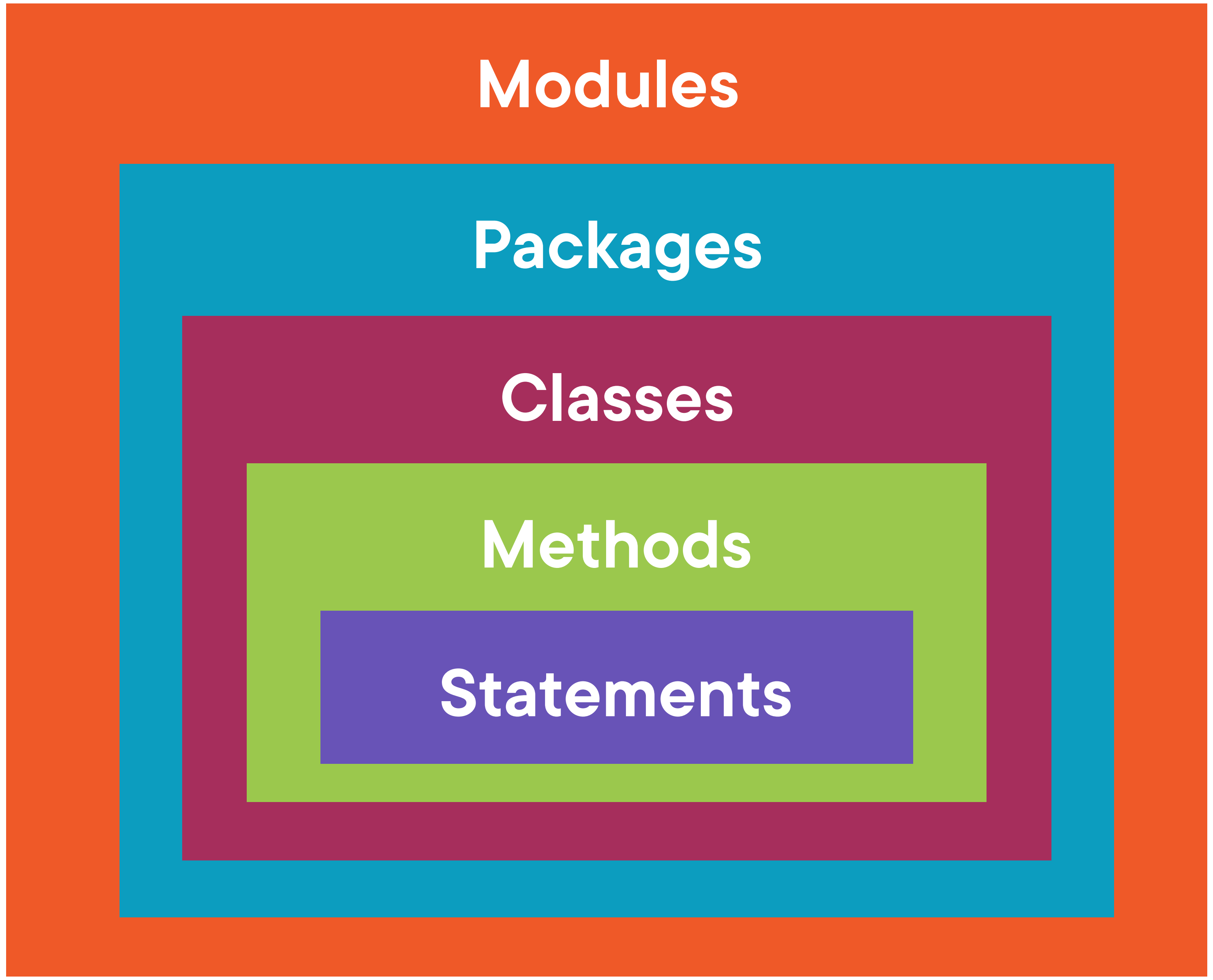


# Modules: Next Level of Abstraction for Java



## Modules: Next Level of Abstraction for Java

**Modules are optional**



## JAR Files and the Java Classpath

## JAR Files and the Java Classpath

A JAR file is **not a module**:

## JAR Files and the Java Classpath

A JAR file is **not a module**:

.. it has a name, which disappears at run-time

## JAR Files and the Java Classpath

A JAR file is **not a module**:

.. it has a name, which disappears at run-time

.. it groups code, but without access control

## JAR Files and the Java Classpath

A JAR file is **not a module**:

.. it has a name, which disappears at run-time

.. it groups code, but without access control

.. it does not describe its dependencies

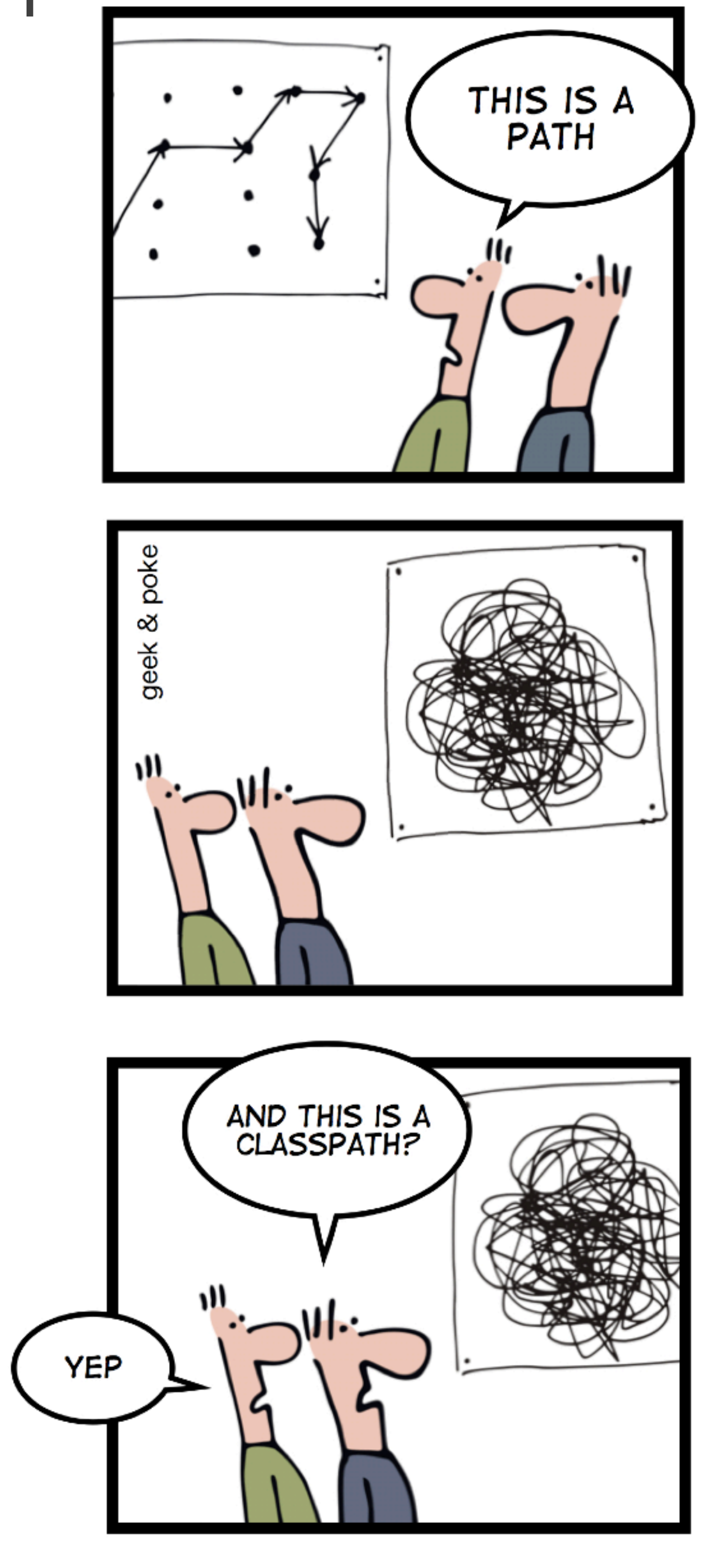
# JAR Files and the Java Classpath

A JAR file is **not a module**:

.. it has a name, which disappears at run-time

.. it groups code, but without access control

.. it does not describe its dependencies



CC BY 3.0, source: bit.ly/gp-classpath

## Why Modules?

## Why Modules?

**Understandability: explicit boundaries and dependencies**

## Why Modules?

**Understandability: explicit boundaries and dependencies**

**Maintainability: hiding implementation details**

## Why Modules?

**Understandability: explicit boundaries and dependencies**

**Maintainability: hiding implementation details**

**Flexibility: decoupling of parts of your system**

## The Three Tenets of Modularity

# The Three Tenets of Modularity

**Strong Encapsulation**

Hide your internals, be strict about what is public API

## The Three Tenets of Modularity

**Strong Encapsulation**

Hide your internals, be strict about what is public API

**Well-defined Interfaces**

When modules interact, use stable and well-defined interfaces

# The Three Tenets of Modularity

**Strong Encapsulation**

Hide your internals, be strict about what is public API

**Well-defined Interfaces**

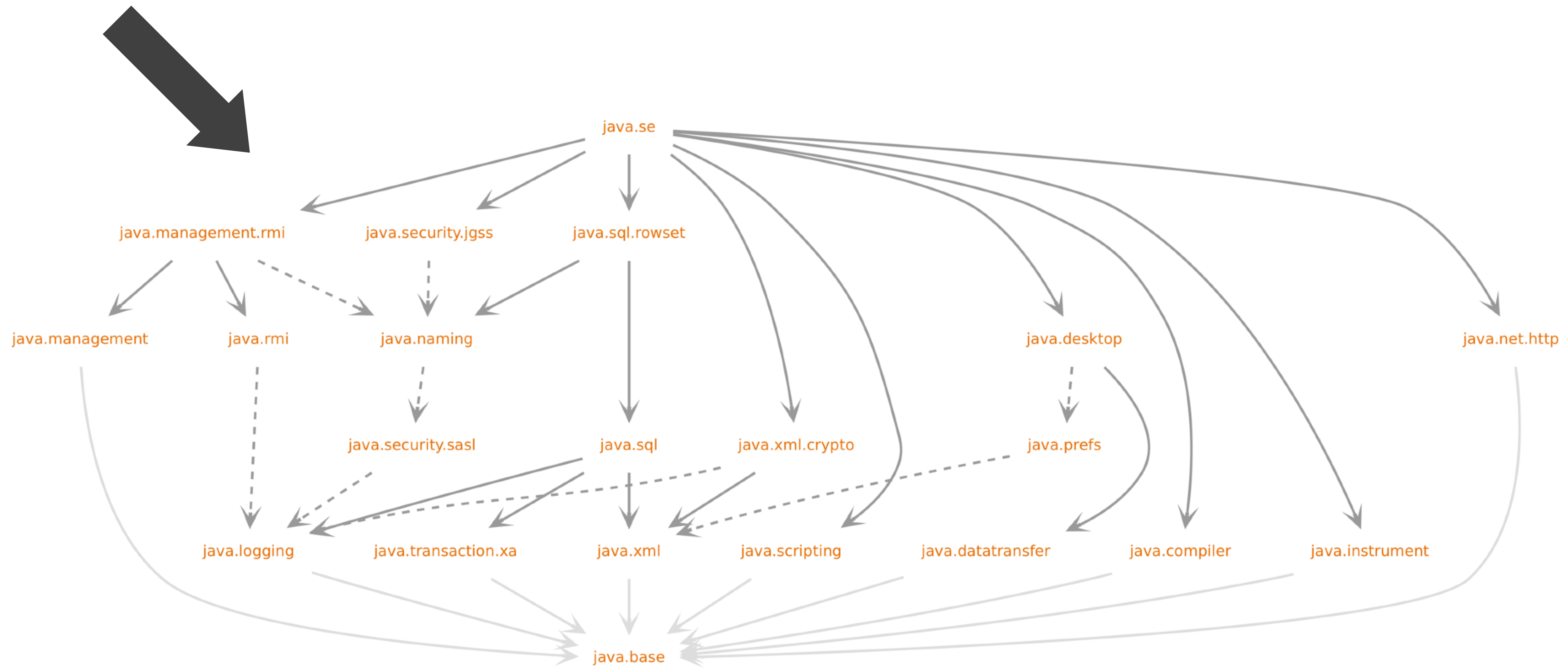
When modules interact, use stable and well-defined interfaces

**Explicit Dependencies**

A module lists what it needs from other modules

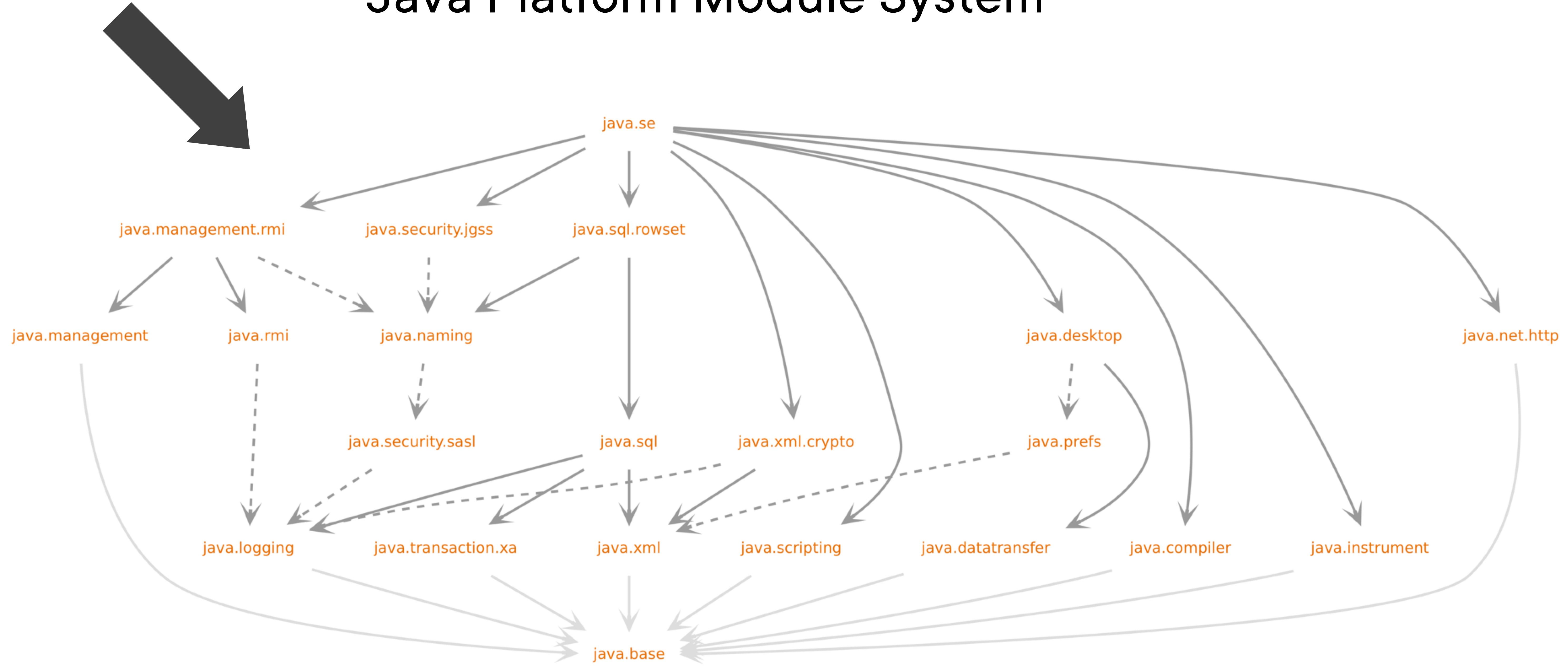
## The Modular JDK

# The Modular JDK



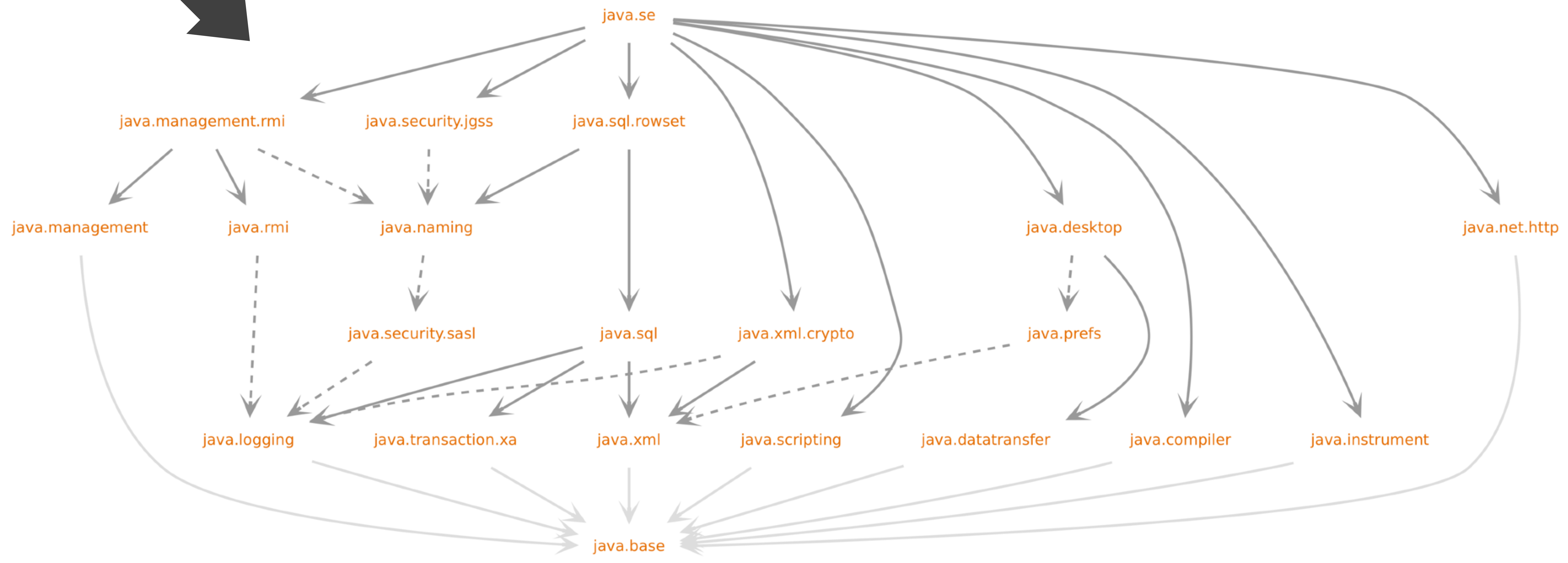
# The Modular JDK

## Java Platform Module System



# The Modular JDK

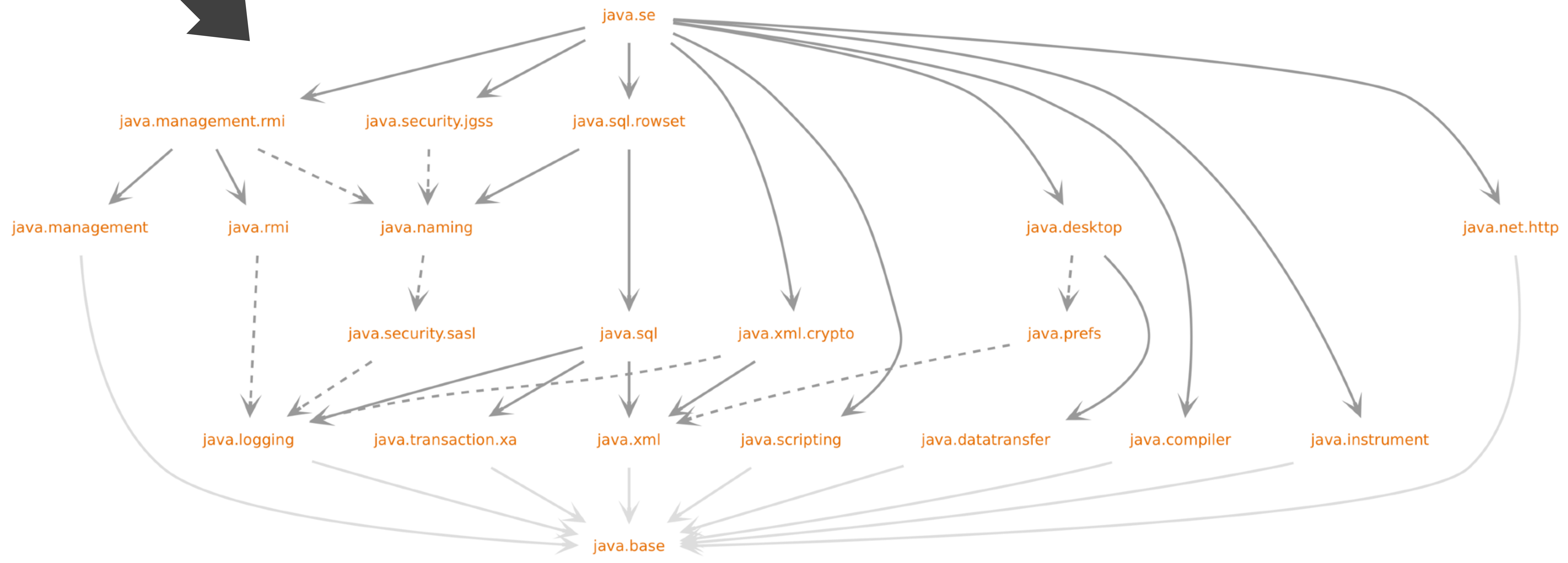
~~Java Platform Module System~~ Java Modules



rt.jar

## The Modular JDK

~~Java Platform Module System~~ Java Modules



## Creating a Module in Java

## Creating a Module in Java

module-info.java

```
module myfirstmodule {
}
```

## Creating a Module in Java

module-info.java

```
module myfirstmodule {
}
```

```
src/
   com/pluralsight/A.java
   com/pluralsight/B.java
   com/pluralsight/util/C.java
   module-info.java
```

## Module Naming

module-info.java

```
module myfirstmodule {
}
```

## Module Naming

**Separate namespace**

module-info.java

```
module myfirstmodule {
}
```

## Module Naming

module-info.java

```
module my.first.module {
}
```

Separate namespace

**One or more Java identifiers separated by `.`**

## Module Naming

module-info.java

```
module myfirstmodule2 {
}
```

Separate namespace

One or more Java identifiers separated by `.`

**Avoid terminal digits**

## Module Naming

module-info.java

```
module com.pluralsight {
}
```

Separate namespace

One or more Java identifiers separated by `.`

Avoid terminal digits

**Common practice: root package as module name**

## Compiled Module

module-info.java

```
module myfirstmodule {
}
```

```
src/
  com/pluralsight/A.java
  com/pluralsight/B.java
  com/pluralsight/util/C.java
  module-info.java
```

# Compiled Module

module-info.java

```
module myfirstmodule {
}
```

```
com/pluralsight/A.class
com/pluralsight/B.class
com/pluralsight/util/C.class
module-info.class
```

## Compiled Module

module-info.java

```
module myfirstmodule {
}
```

myfirstmodule.jar

```
META-INF/
  MANIFEST.MF
com/pluralsight/A.class
com/pluralsight/B.class
com/pluralsight/util/C.class
module-info.class
```

Demo

## Creating and Running a Module

Demo

## Creating and Running a Module

Demo

## Creating and Running a Module

- No IDE!
- Using regular compilation
- Using module-specific compilation flag

## Recap: Compilation Flags

# Recap: Compilation Flags

Single module

## Recap: Compilation Flags

**Single module**

```
javac -d {dir}
```

# Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

# Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

```
javac -d out \
src/com/javamodularity/greeter/Main.java \
src/module-info.java
```

# Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

**Multiple modules**

# Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

**Multiple modules**

```
javac -d {dir} \
```

# Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

**Multiple modules**

```
javac -d {dir} \
  --module-source-path {src_dir} \
```

## Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

**Multiple modules**

```
javac -d {dir} \
  --module-source-path {src_dir} \
  -m {module_name}
```

## Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

**Multiple modules**

```
javac -d {dir} \
  --module-source-path {src_dir} \
  -m {module_name},{module_name}
```

# Recap: Compilation Flags

**Single module**

```
javac -d {dir} {all source files, including module-info.java}
```

**Multiple modules**

```
javac -d {dir} \
  --module-source-path {src_dir} \
  -m {module_name},{module_name}

javac -d out \
  --module-source-path src \
  --module greeter
```

## Recap: JVM Flags

## Recap: JVM Flags

Running a module

# Recap: JVM Flags

**Running a module**

```
java -p {module_path} \
     -m {module}/{fully qualified main class}
```

## Recap: JVM Flags

**Running a module**

```
java -p {module_path} \
     -m {module}/{fully qualified main class}

java --module-path {module_path} \
     --module {module}/{fully qualified main class}
```

## Recap: JVM Flags

**Running a module**

```
java -p {module_path} \
     -m {module}/{fully qualified main class}

java --module-path {module_path} \
     --module {module}/{fully qualified main class}
```

```
java --module-path out \
     --module greeter/com.javamodularity.greeter.Main
```

## Summary

## Summary

## Summary

Why modularity?

## Summary

Why modularity?

Module declarations

## Summary

Why modularity?

Module declarations

Compile and run a module