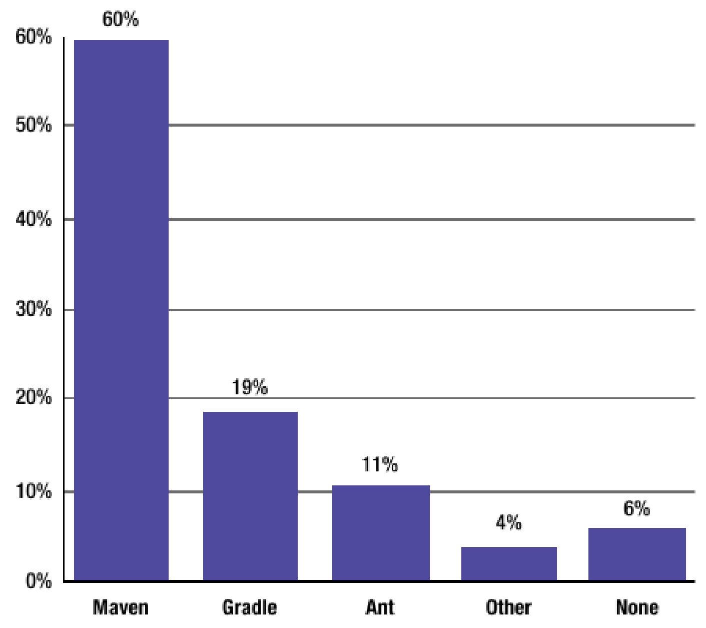
Full Stack Development

Containers, Microservices and UI

7a. Build Management with Maven

Build Tools

- Build task automation
  - Compile, clean, test, package and deploy
- Variety of build tools available in the Java world
  - Apache Ant is the oldest and crankiest to use
  - Gradle is an up to date version of Ant using Kotlin or Groovy scripting instead of XML
  - The most common is Maven, derived for the Apache Jakarta project



Maven

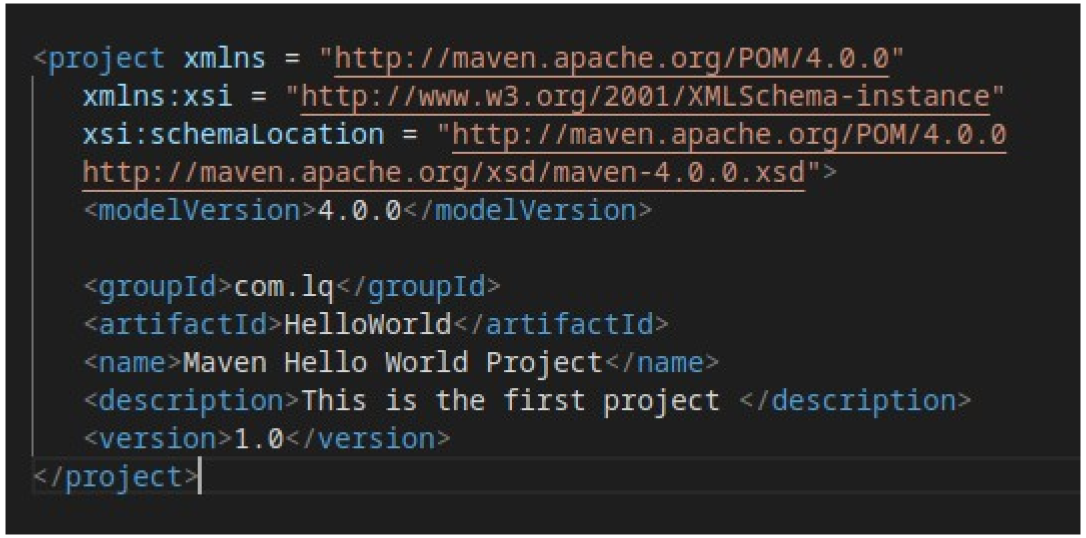
- Convention over configuration
  - Standard directory structure for modular projects that can be customized
  - Standard build lifecycle steps that can be customized
  - Mature automatic dependency management from various repositories
  - Uses what are considered “sensible defaults”
- The pom.xml file defines the project structure
  - Generated from Maven archetype
  - Multiple templates available for different kinds of projects
- Uniform build abstraction
  - Same set of commands are used across different maven projects
  - Plugins customize what is done at each step
  - Large and mature plugin community

Maven

- CICD integration
  - Maven integrates with various CICD pipeline tools like Jenkins
  - Supported by almost all IDEs
- Archetypes
  - Pre-configured project templates used to generate new projects
  - Archetypes generate all of the folders and files needed to start the project
  - Customized as needed
- Maven has created de facto standards
  - Standard directory layout: now used by other tools like Gradle
  - Artifact naming: Using a set of specific “coordinates”
  - Java dependency repositories: did not exist prior to Maven - now standarized

Maven Coordinates

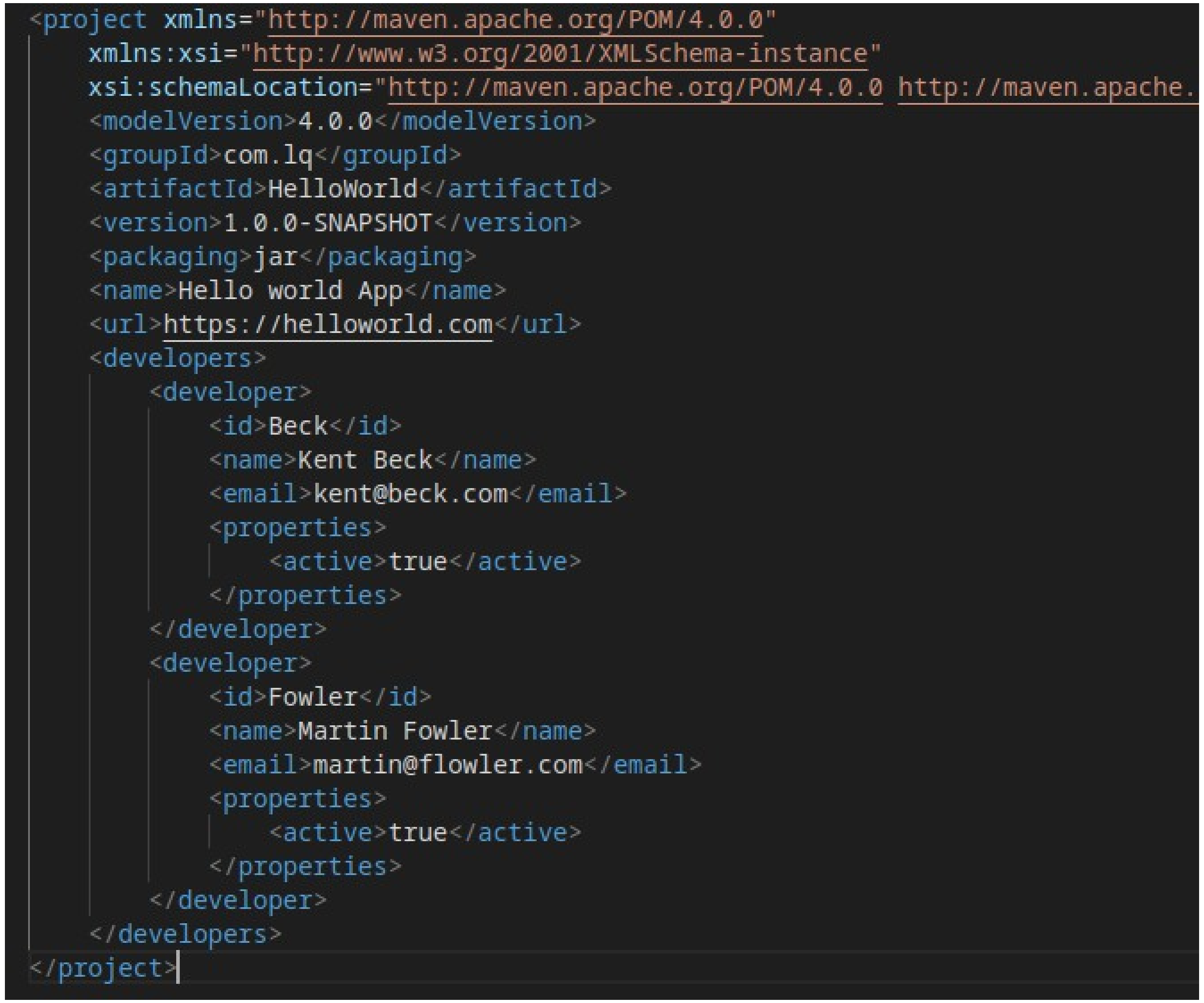
- Define the properties of the project
  - * indicates required fields
  - *Group ID - the organization name
  - *Artifact ID - the name of the app
  - Name - the display name of app
  - Description - Doc string
  - *Version - the version of this app



```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.lq</groupId>
   <artifactId>HelloWorld</artifactId>
   <name>Maven Hello World Project</name>
   <description>This is the first project </description>
   <version>1.0</version>
</project>
```

Maven pom.xml



```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.lq</groupId>
    <artifactId>HelloWorld</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Hello world App</name>
    <url>https://helloworld.com</url>
    <developers>
        <developer>
            <id>Beck</id>
            <name>Kent Beck</name>
            <email>kent@beck.com</email>
            <properties>
                <active>true</active>
            </properties>
        </developer>
        <developer>
            <id>Fowler</id>
            <name>Martin Fowler</name>
            <email>martin@flowler.com</email>
            <properties>
                <active>true</active>
            </properties>
        </developer>
    </developers>
</project>
```

Hello World Maven

Demo

Hello World

Lab Maven 1

Maven Lifecycles

- Maven has a default life lifecycle made up of phases
  - **validate**: check if all information necessary for the build is available
  - **compile**: compile the source code
  - **test-compile:** compile the test source code
  - **test: run unit tests**
  - **package:** package compiled source code into the distributable format (jar, war, …)
  - **integration-test**: process and deploy the package if needed to run integration tests
  - **install:** install the package to a local repository
  - **deploy**: copy the package to the remote repository
- If any phase is run (maven package) for example then all of the phases prior to that are also run
  - When executing a Maven command, the “target” is one of more of the phases above
  - **maven compile** executes the Maven target “compile”

Maven Plugins

- Customized behavior in each phase can be added via plug-ins
  - This will be done in the lab
- The default project properties and tools used can be overridden by specific property statements
  - Maven assumes a “reasonable” set of project defaults
  - Convention over configuration
- Plugins are maintained by a large plugin development community
  - Collection curated by the Apache Maven project
  - https://maven.apache.org/plugins/

Maven Archetypes

- An archetype is a reusable project type
- Consists of a project structure and related dependencies
- An archetype is often used as a starting point for a project
  - Takes care of writing all the initial boilerplate code
- Apache maintains an archetype project
  - https://maven.apache.org/archetype/index.html
- Like plugins, there is a large community of archetype developers
  - https://github.com/tbroyer/gwt-maven-archetypes

Maven Archetypes

Demo

Maven Archetypes

Lab Maven 2

Maven and Eclipse

Demo

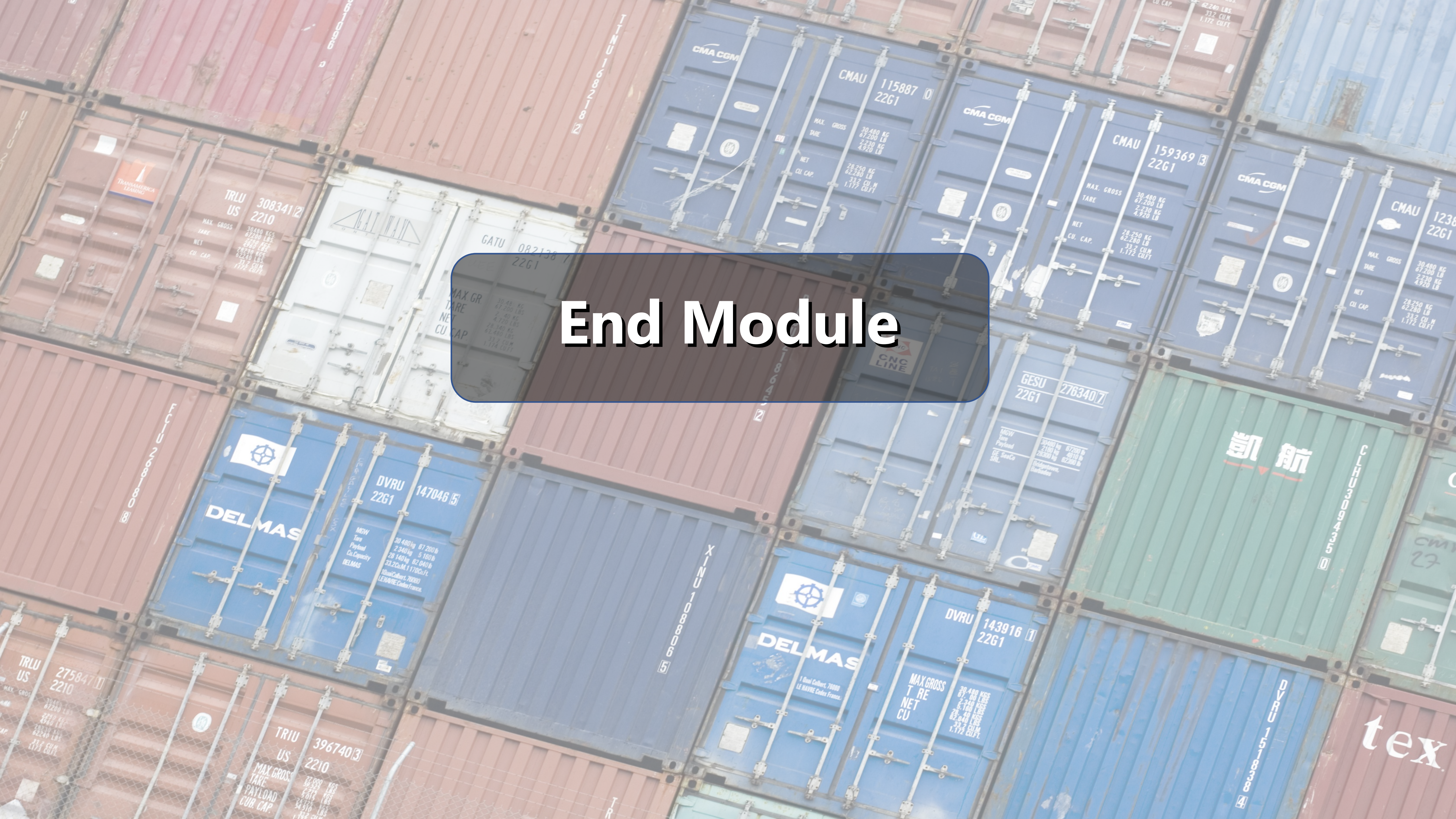
Maven and Eclipse

Lab Maven 3

Maven Dependency Management

- Most projects have some sort of dependency
- For example, JUnit for testing
- Maven does automatic dependency management
  - Common dependencies are found in the Maven repository
  - Other repositories are also available
- Maven does transitive dependency management
  - If a dependency specified has a dependency, Maven resolves both of them
- Maven can be configured to use different repositories
  - Important when security is a concern
  - Private repositories keep library artifacts private
- We will see dependency management in the Spring section

Questions?



End Module