Maven Crash Course

Spring Boot and Maven

- When you generate projects using Spring Initializr: start.spring.io
  - It can generate a Maven project for you

- In this section, we will learn the basics of Maven
  - Viewing dependencies in the Maven pom.xml file
  - Spring Boot Starters for Maven

What is Maven?

- Maven is a Project Management tool

- Most popular use of Maven is for build management and dependencies

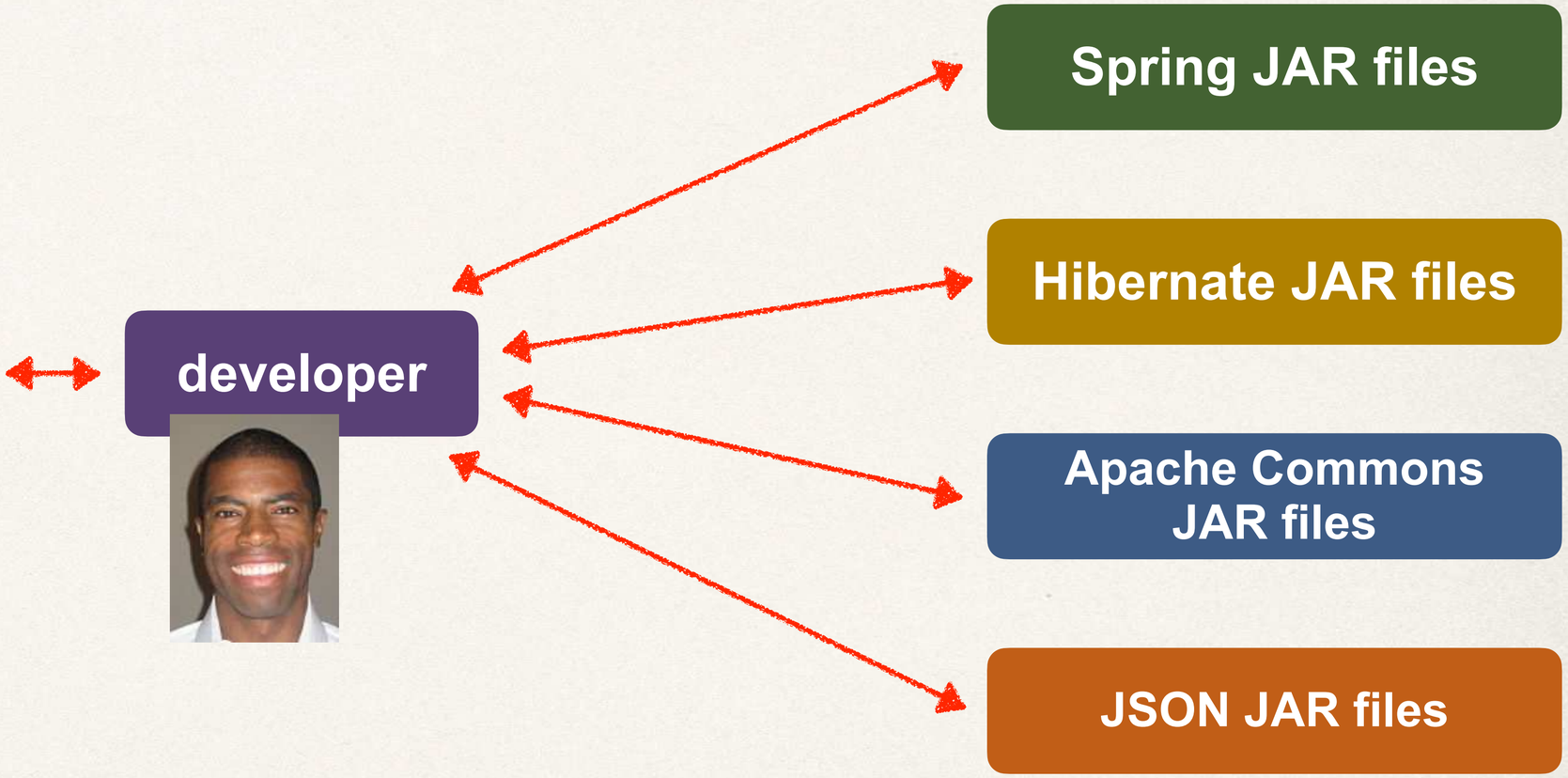
What Problems Does Maven Solve?

- When building your Java project, you may need additional JAR files
  - For example: Spring, Hibernate, Commons Logging, JSON etc…
- One approach is to download the JAR files from each project web site
- Manually add the JAR files to your build path / classpath

My Project without Maven

My Super Cool App

Spring
Hibernate
Commons Logging
JSON
…



Maven Solution

- Tell Maven the projects you are working with (dependencies)
  - Spring, Hibernate etc ….
- Maven will go out and download the JAR files for those projects for you
- And Maven will make those JAR files available during compile/run
- Think of Maven as your friendly helper / personal shopper :-)

My Project with Maven

Maven Central Repository (remote)

My Super Cool App

Spring
Hibernate
Commons Logging
JSON
...

developer

Maven

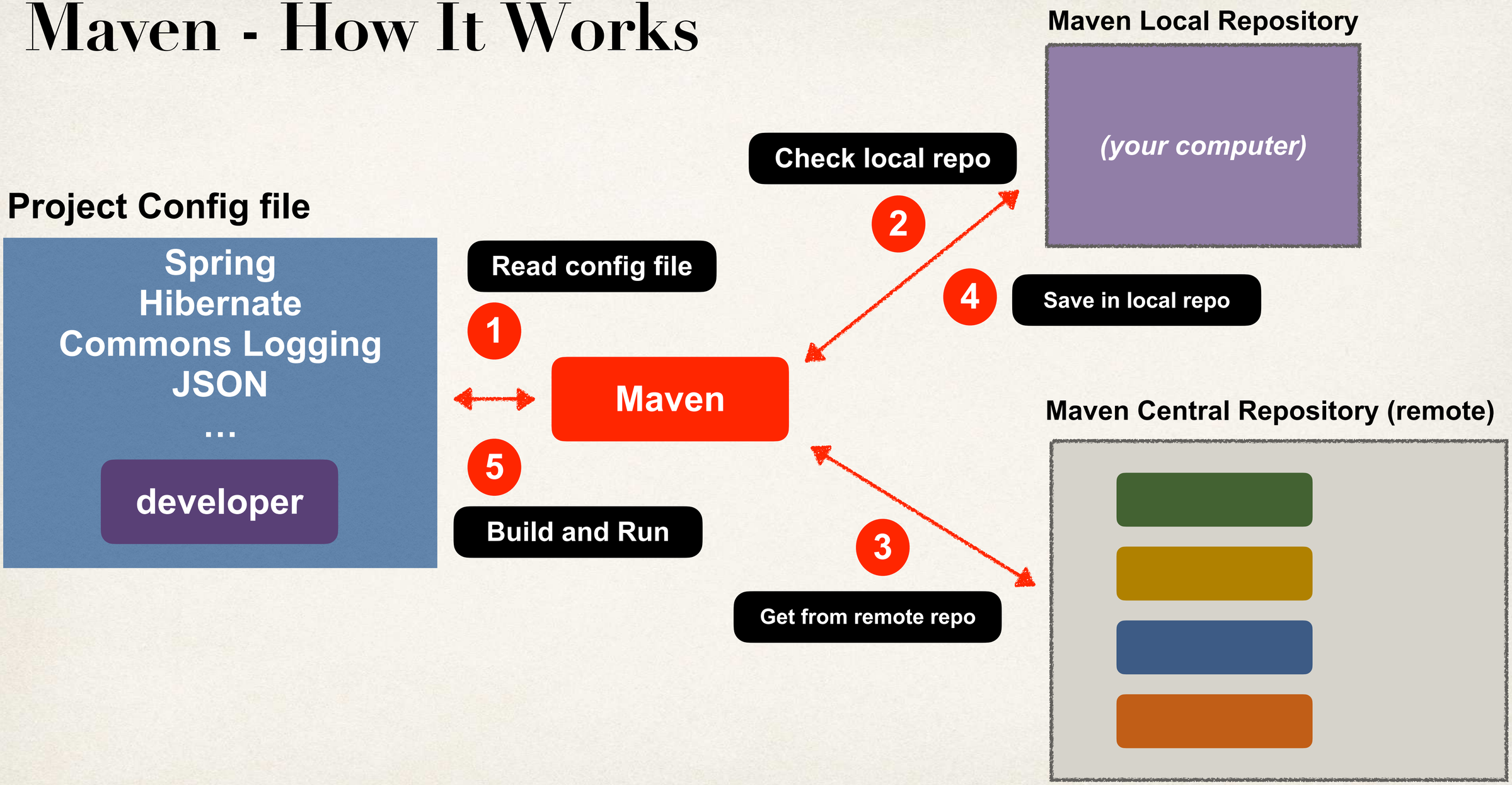
Spring JAR files

Hibernate JAR files

Apache Commons JAR files

JSON JAR files

Maven - How It Works



Handling JAR Dependencies

- When Maven retrieves a project dependency
  - It will also download supporting dependencies
  - For example: Spring depends on commons-logging …
- Maven will handle this for us automagically

Building and Running

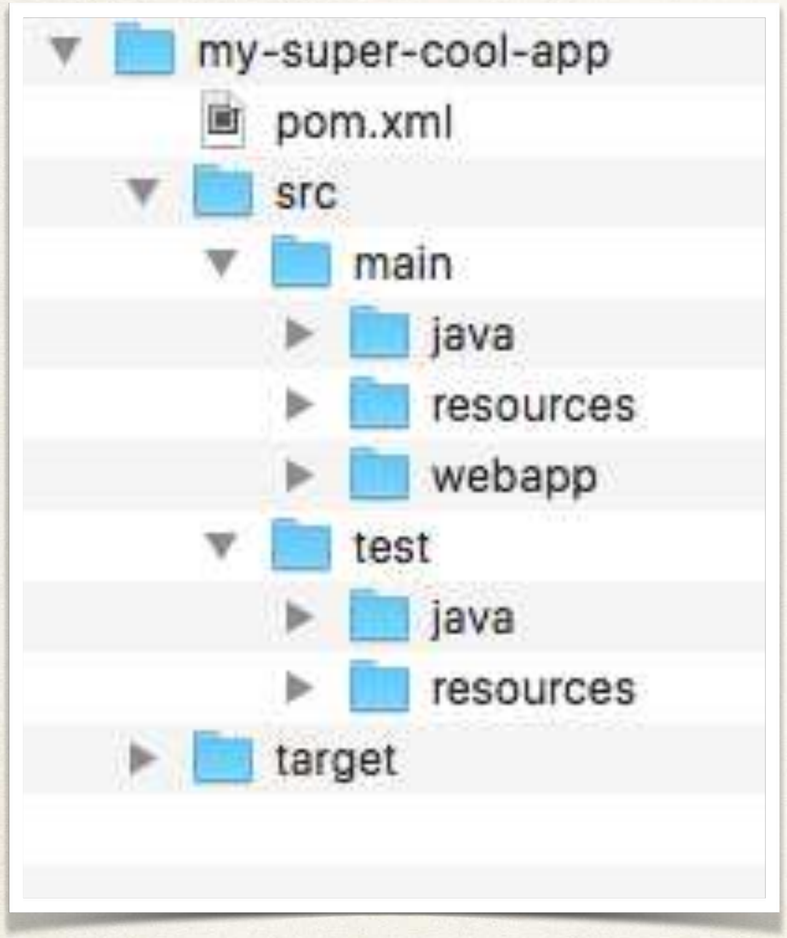
- When you build and run your app …
- Maven will handle class / build path for you
- Based on config file, Maven will add JAR files accordingly

Standard Directory Structure

- Normally when you join a new project
  - Each development team dreams up their own directory structure
  - Not ideal for new comers and not standardized

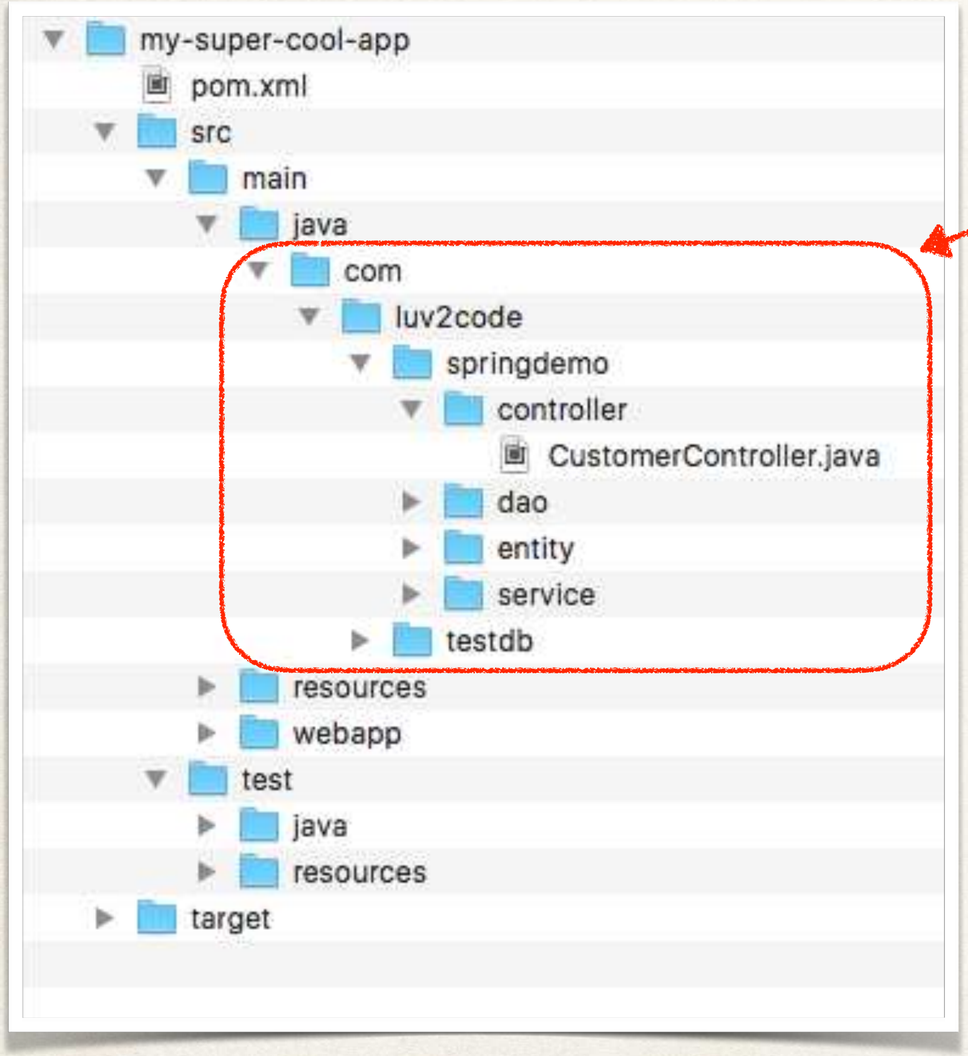
- Maven solves this problem by providing a standard directory structure

Standard Directory Structure



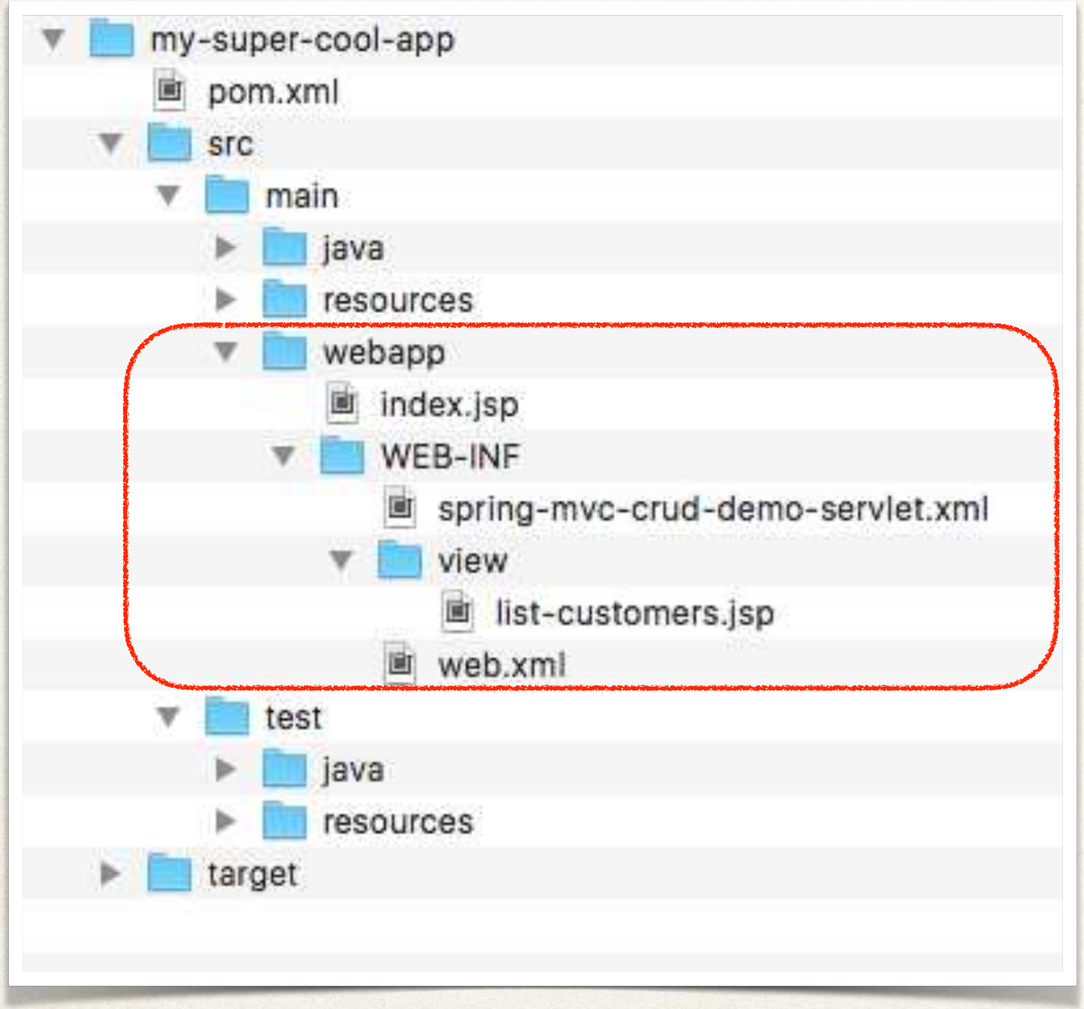
Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/main/webapp	JSP files and web config files other web assets (images, css, js, etc)
src/test	Unit testing code and properties
target	Destination directory for compiled code. Automatically created by Maven

Standard Directory Structure



Place your Java source code here

Standard Directory Structure



Place your Web assets here

Standard Directory Structure Benefits

- For new developers joining a project
  - They can easily find code, properties files, unit tests, web files etc …

Standard Directory Structure Benefits

- Most major IDEs have built-in support for Maven
  - Eclipse, IntelliJ, NetBeans etc
  - IDEs can easily read/import Maven projects
- Maven projects are portable
  - Developers can easily share projects between IDEs
  - No need to fight about which IDE is the best LOL!

Advantages of Maven

- Dependency Management
  - Maven will find JAR files for you
  - No more missing JARs
- Building and Running your Project
  - No more build path / classpath issues
- Standard directory structure

My Personal Best Maven Benefit(s)

- Once you learn Maven, you can join a new project and be productive

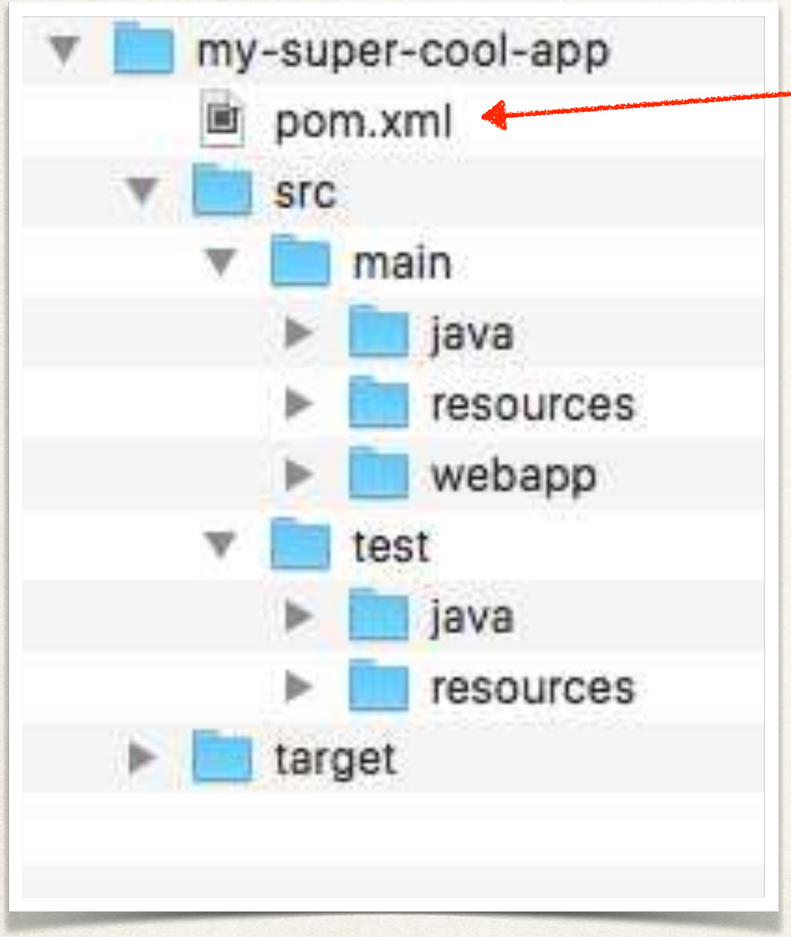
- You can build and run a project with minimal local configuration

Maven Key Concepts

Maven Key Concepts

- POM File - pom.xml
- Project Coordinates

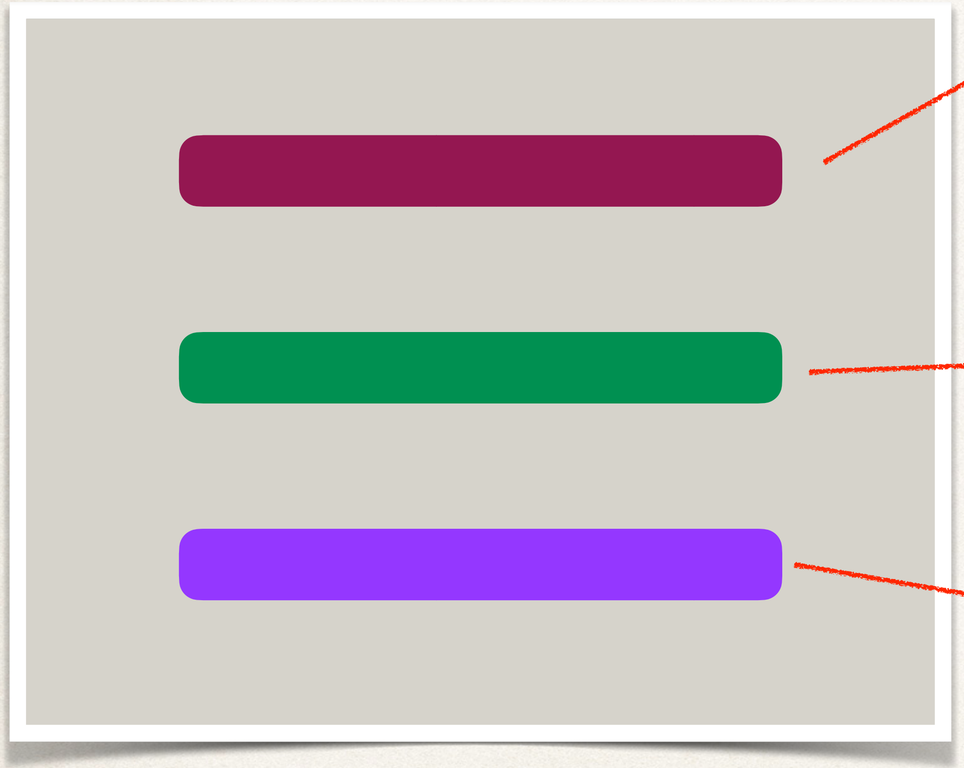
POM File - pom.xml



- Project Object Model file: POM file
- Configuration file for your project
  - Basically your “shopping list” for Maven :-)
- Located in the root of your Maven project

POM File Structure

pom.xml



Project name, version etc
Output file type: JAR, WAR, …

List of projects we depend on
Spring, Hibernate, etc…

Additional custom tasks to run:
generate JUnit test reports etc…

Simple POM File

```
<project ...>

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.luv2code</groupId>
  <artifactId>mycoolapp</artifactId>
  <version>1.0.FINAL</version>
  <packaging>jar</packaging>

  <name>mycoolapp</name>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.9.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

    <!-- add plugins for customization -->

</project>
```

Project name, version etc
Output file type: JAR, WAR, …

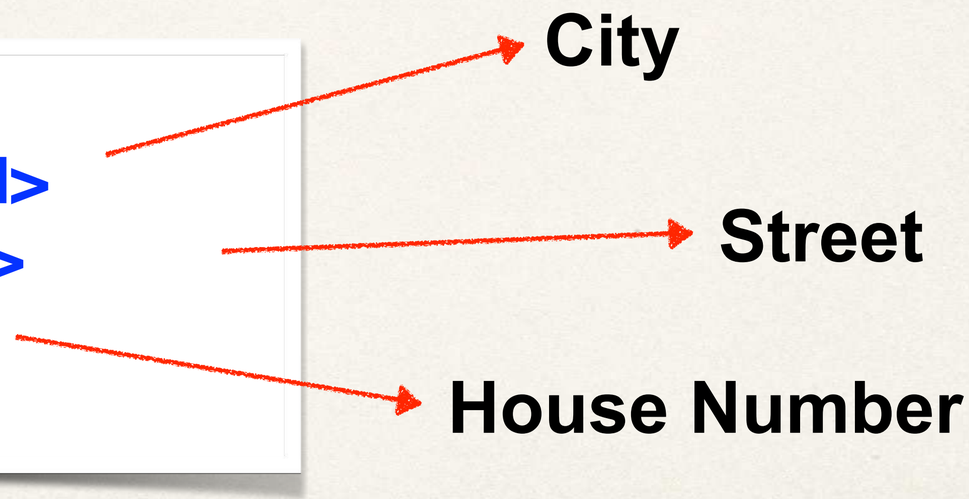
List of projects we depend on
Spring, Hibernate, etc…

Additional custom tasks to run:
generate JUnit test reports etc…

Project Coordinates

- Project Coordinates uniquely identify a project
  - Similar to GPS coordinates for your house: latitude / longitude
  - Precise information for finding your house (city, street, house #)

```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.FINAL</version>
```



Project Coordinates - Elements

Name	Description
Group ID	Name of company, group, or organization. Convention is to use reverse domain name: **com.luv2code**
Artifact ID	Name for this project: **mycoolapp**
Version	A specific release version like: **1.0, 1.6, 2.0** … If project is under active development then: **1.0-SNAPSHOT**

```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.FINAL</version>
```

Example of Project Coordinates

```
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.RELEASE</version>
```

```
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>6.0.0</version>
```

```
<groupId>org.hibernate.orm</groupId>
<artifactId>hibernate-core</artifactId>
<version>6.1.4.Final</version>
```

Adding Dependencies

```
<project ...>
 ...
 <dependencies>

      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-context</artifactId>
         <version>6.0.0</version>
      </dependency>

      <dependency>
         <groupId>org.hibernate.orm</groupId>
         <artifactId>hibernate-core</artifactId>
         <version>6.1.4.Final</version>
      </dependency>
      ...

 </dependencies>

</project>
```

Dependency Coordinates

- To add given dependency project, we need
  - **Group ID**, **Artifact ID**
  - **Version** is optional …
    - Best practice is to include the version (repeatable builds)

- May see this referred to as: **GAV**
  - **G**roup ID, **A**rtifact ID and **V**ersion

How to Find Dependency Coordinates

- Option 1: Visit the project page (spring.io, hibernate.org etc)

- Option 2: Visit **`https://central.sonatype.com`** (easiest approach)