Spring Framework Overview

Spring Website - Official

www.spring.io

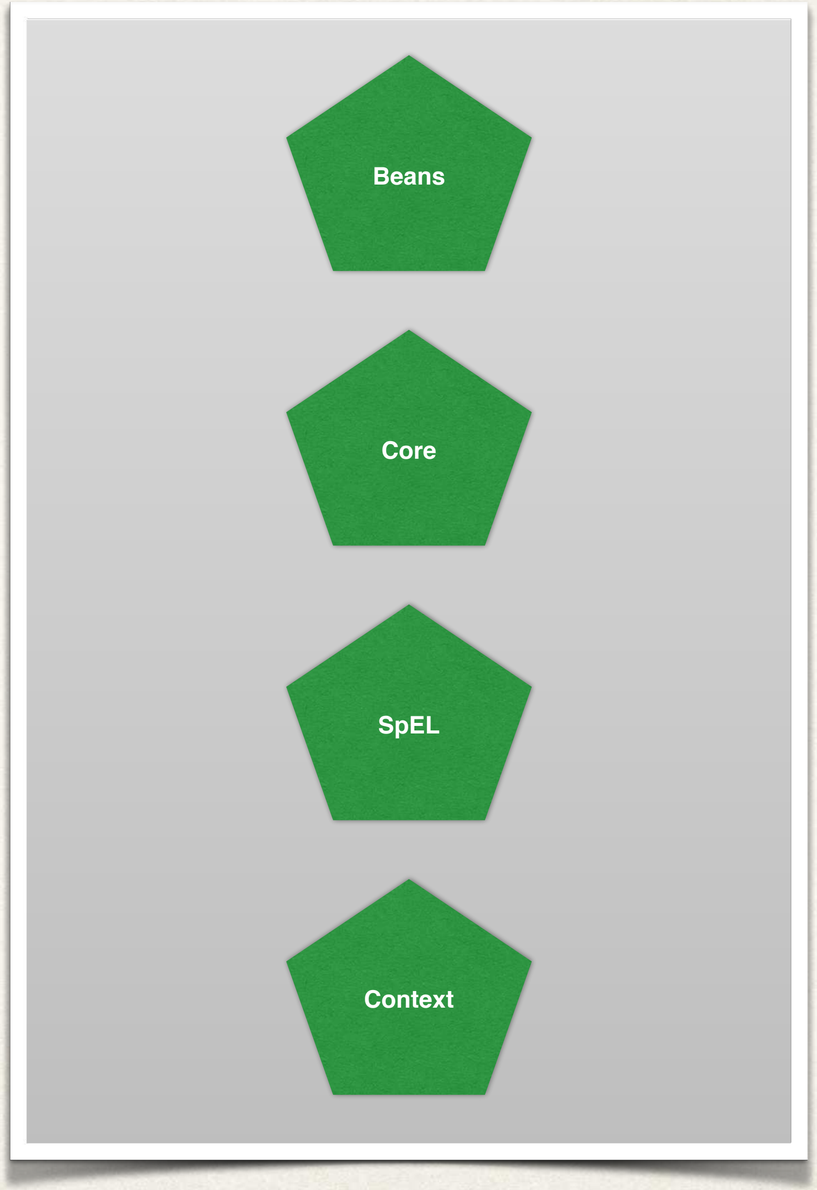
Why Spring?

Simplify Java Enterprise Development

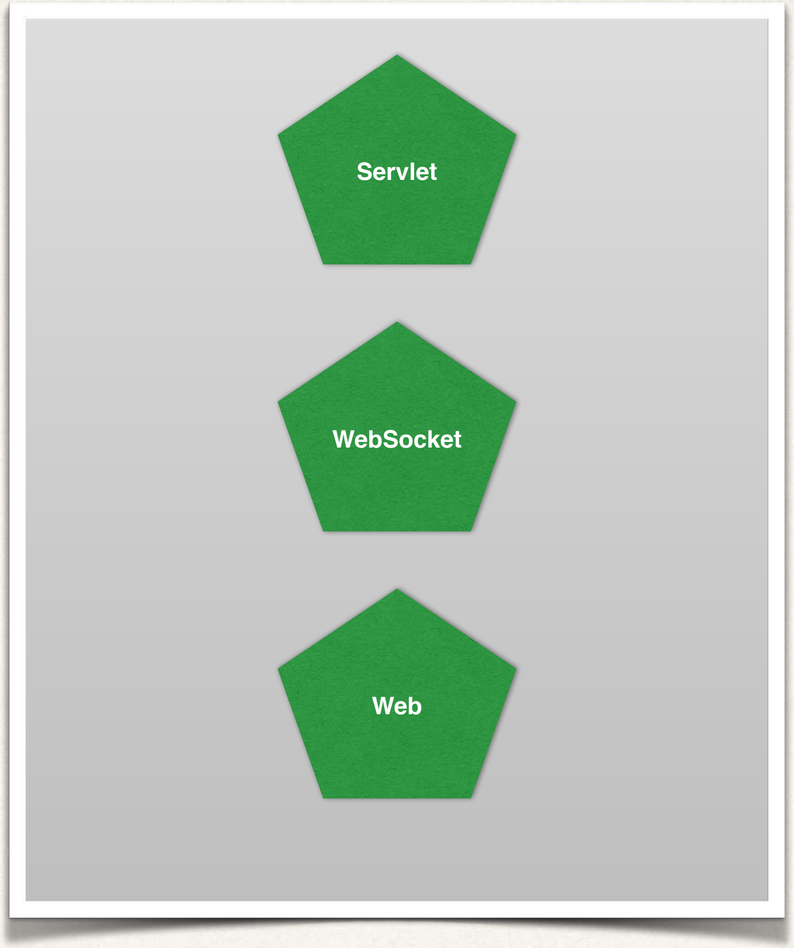
Goals of Spring

- Lightweight development with Java POJOs (Plain-Old-Java-Objects)
- Dependency injection to promote loose coupling
- Declarative programming with Aspect-Oriented-Programming (AOP)
- Minimize boilerplate Java code

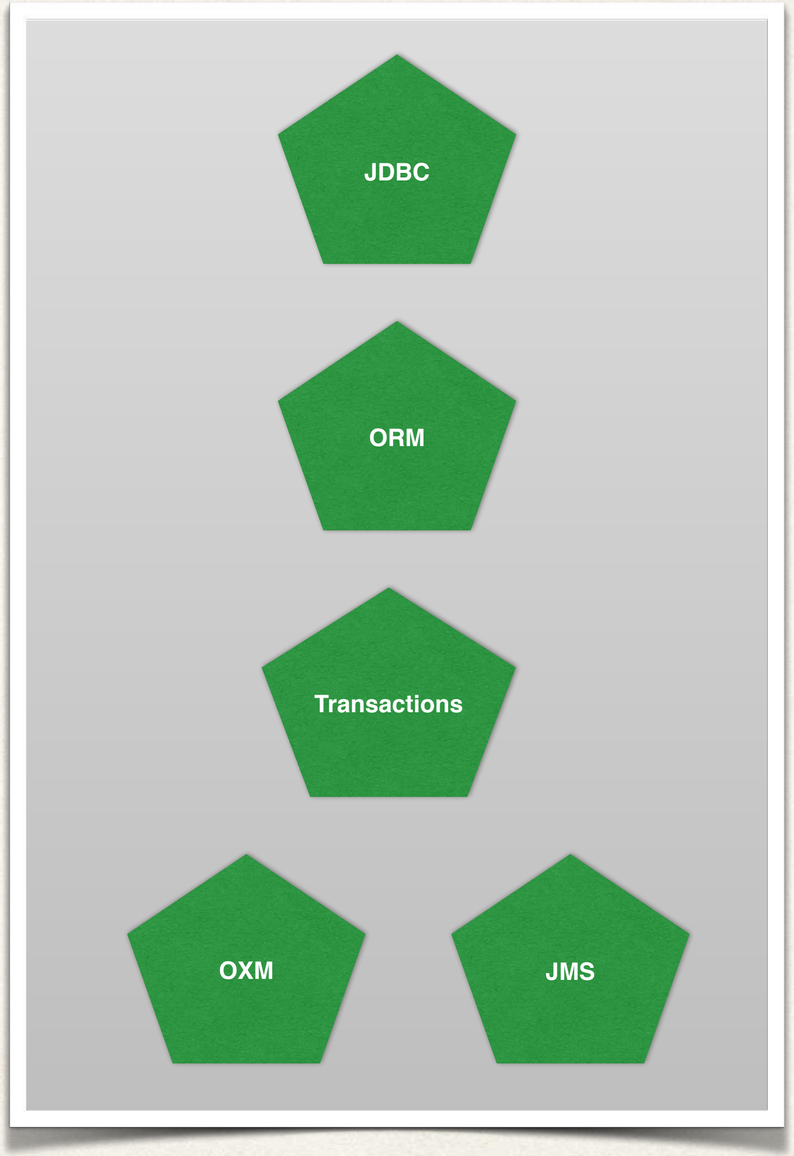
Core Container



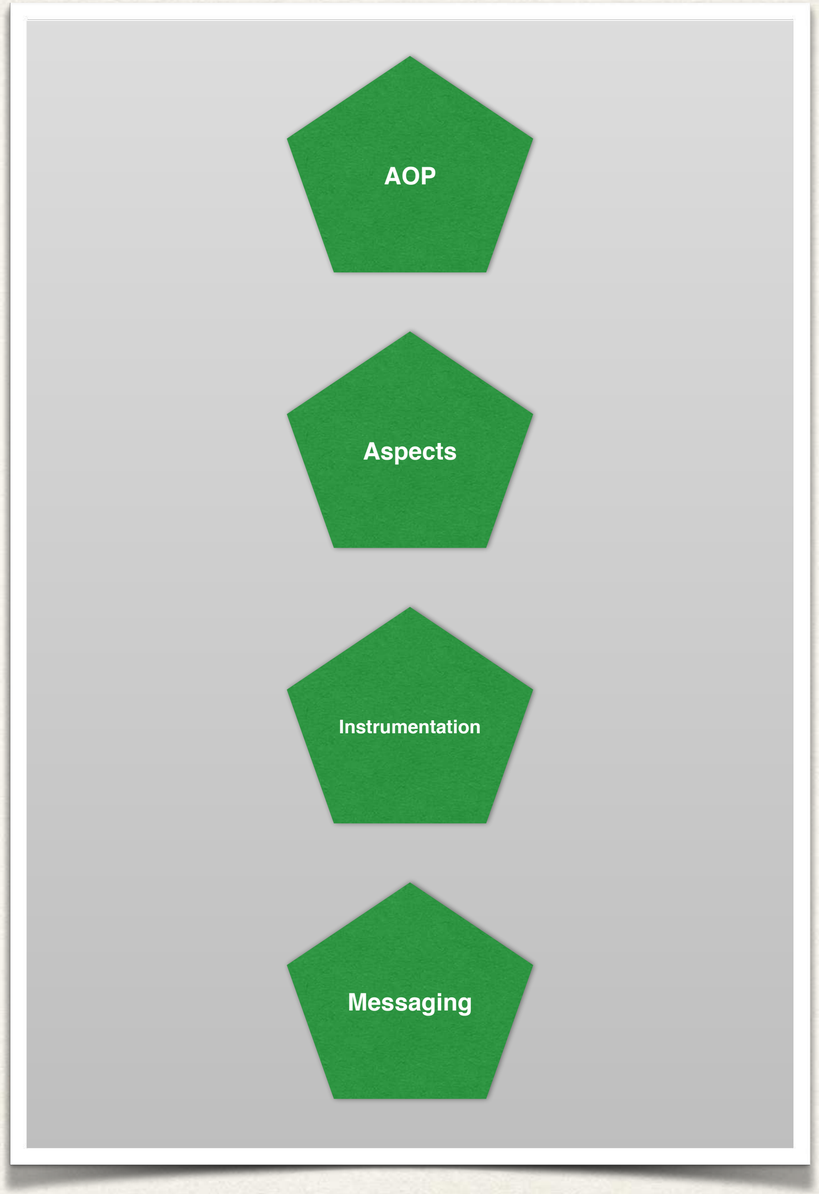
Web Layer



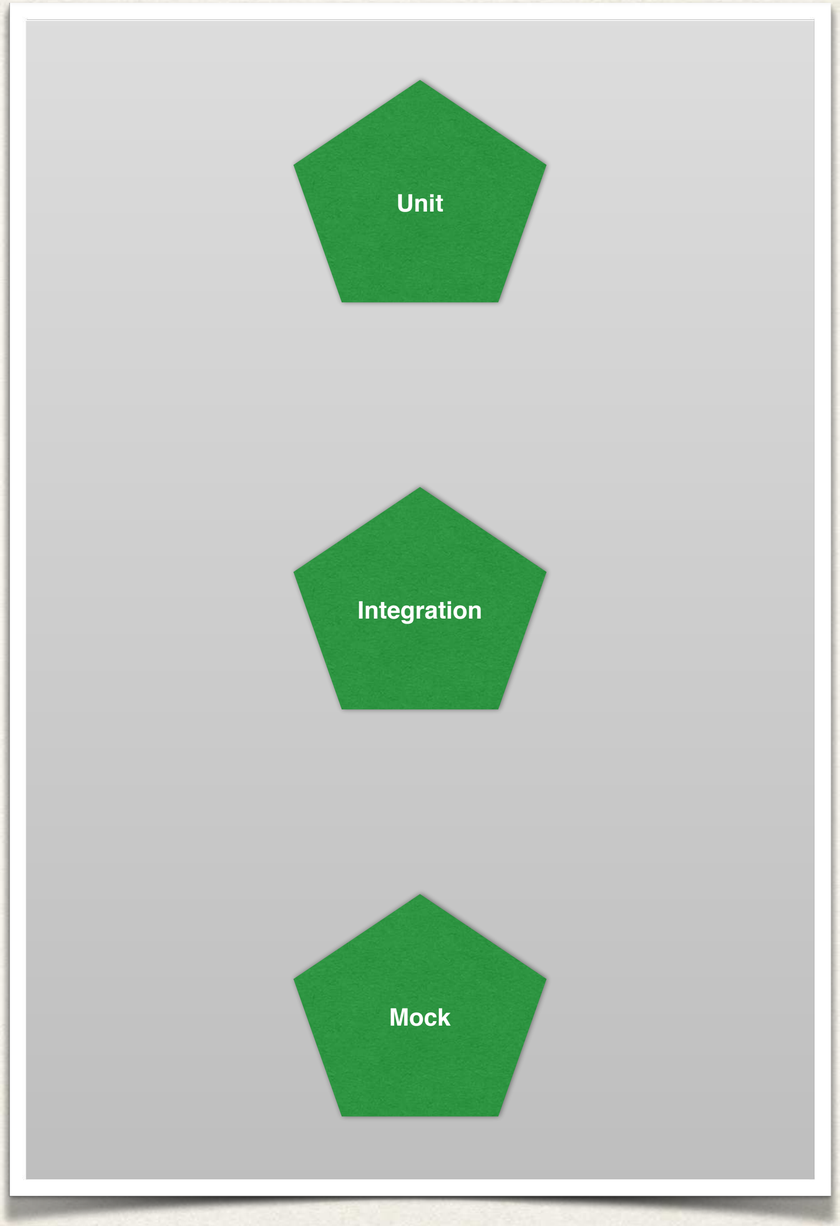
Data Access Layer



Infrastructure



Test Layer



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

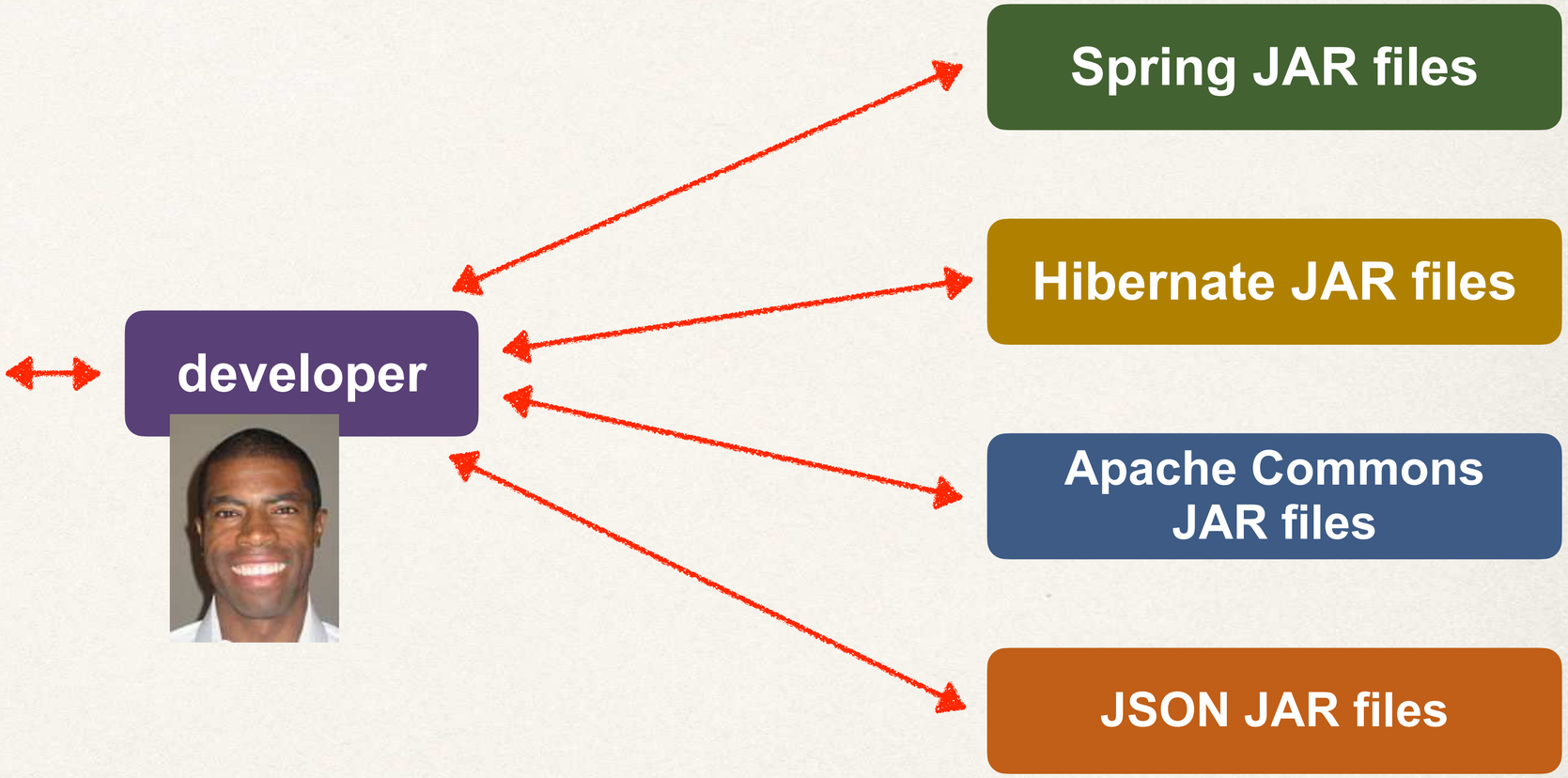
What Problems Does Maven Solve?

- 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

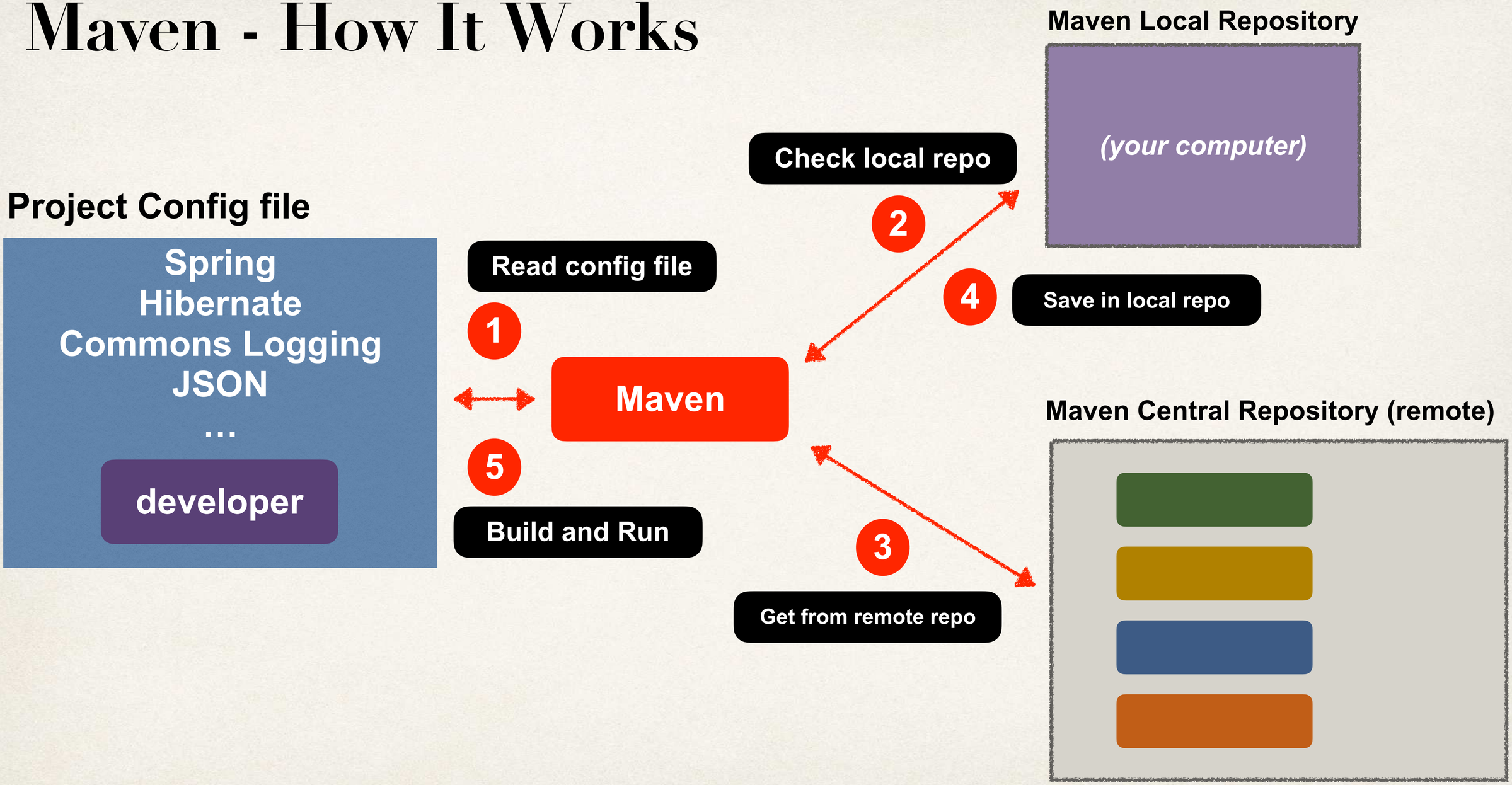
My Project without Maven

My Super Cool App

Spring
Hibernate
Commons Logging
JSON
…



Maven - How It Works

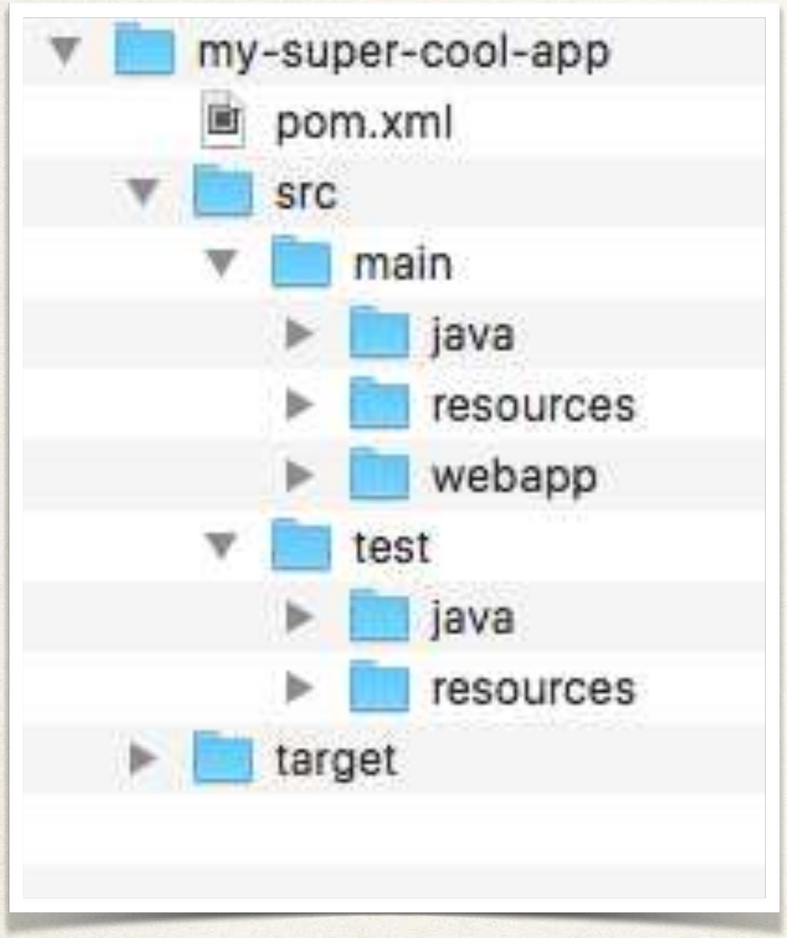


Standard Directory Structure

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

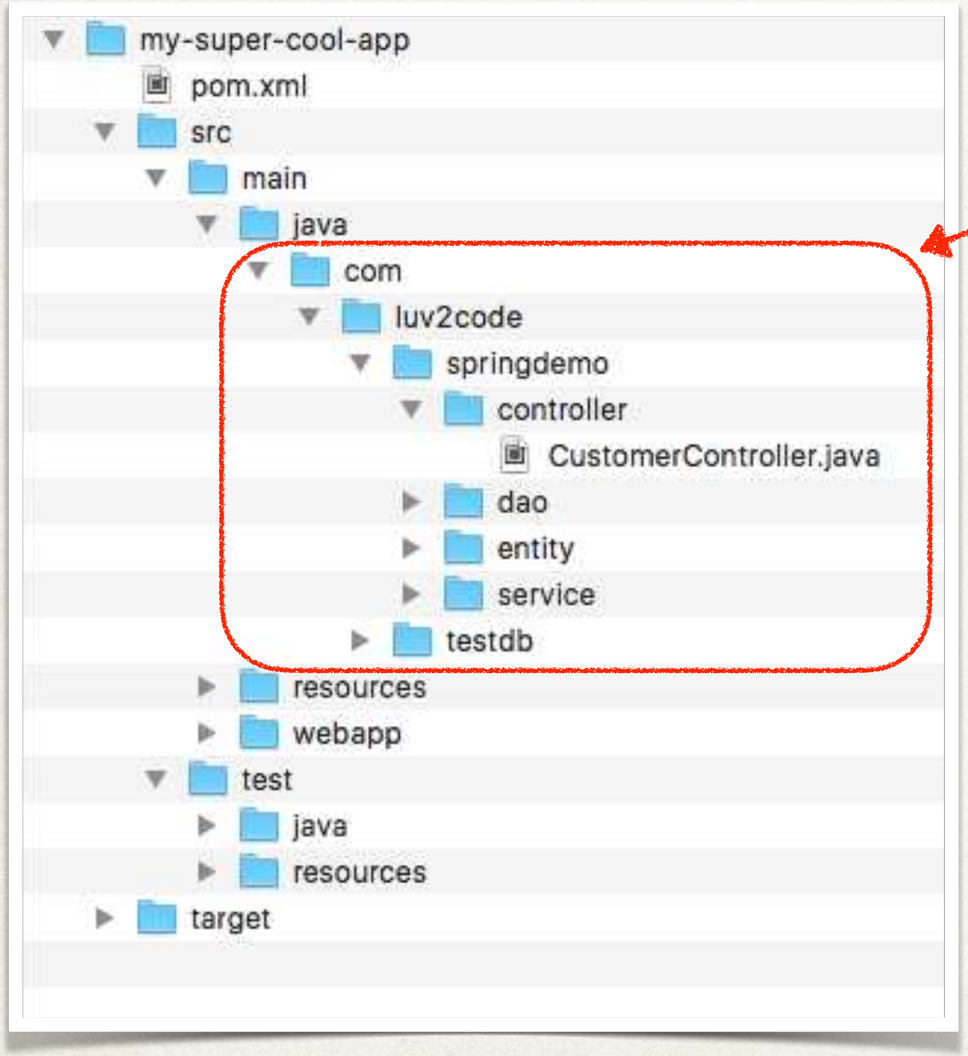
- Maven solves this problem by providing a standard directory structure

Standard Directory Structure



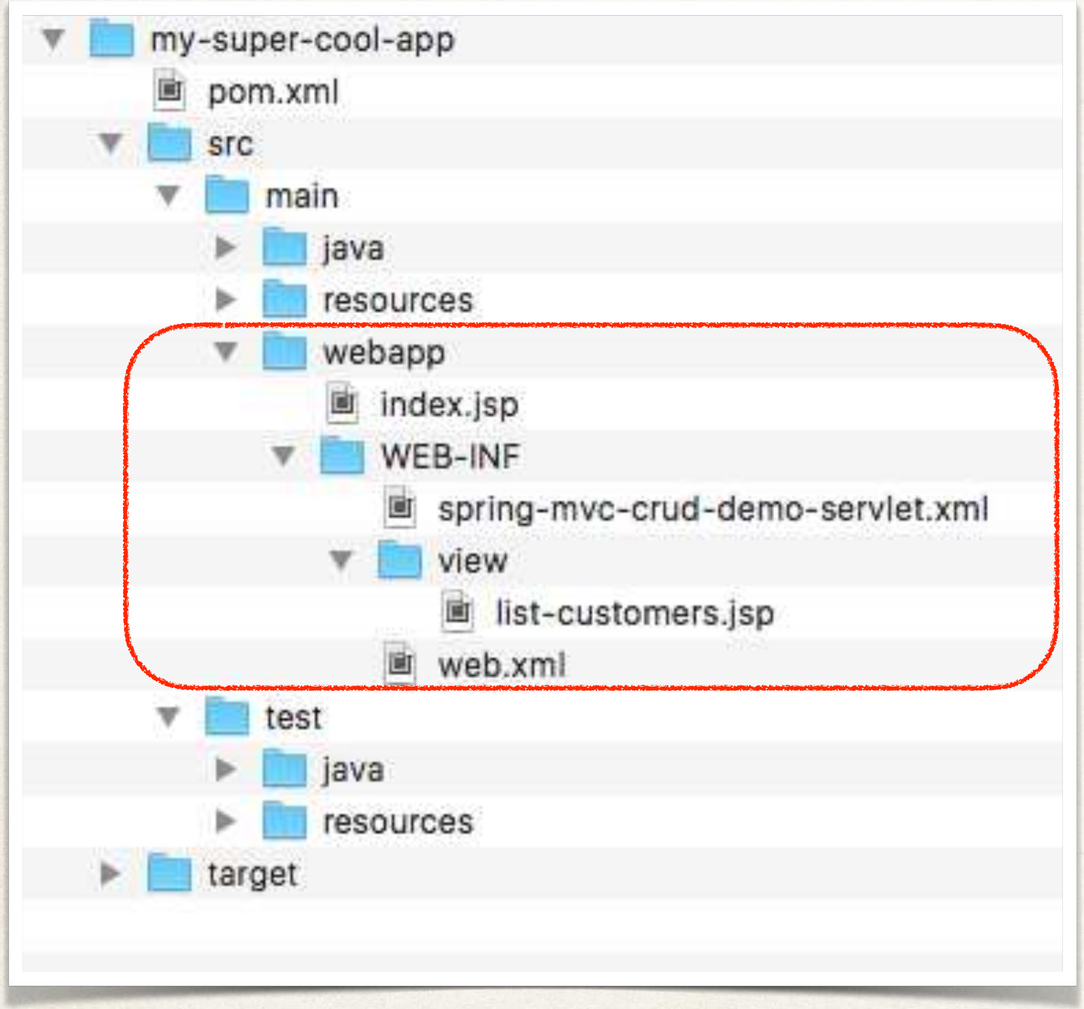
Directory	Description
src/main/java	Your Java source code
src/main/resources	Properties / config files used by your app
src/main/webapp	JSP files and web config files other web assets (images, css, js, etc)
src/test	Unit testing code and properties
target	Destination directory for compiled code. Automatically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

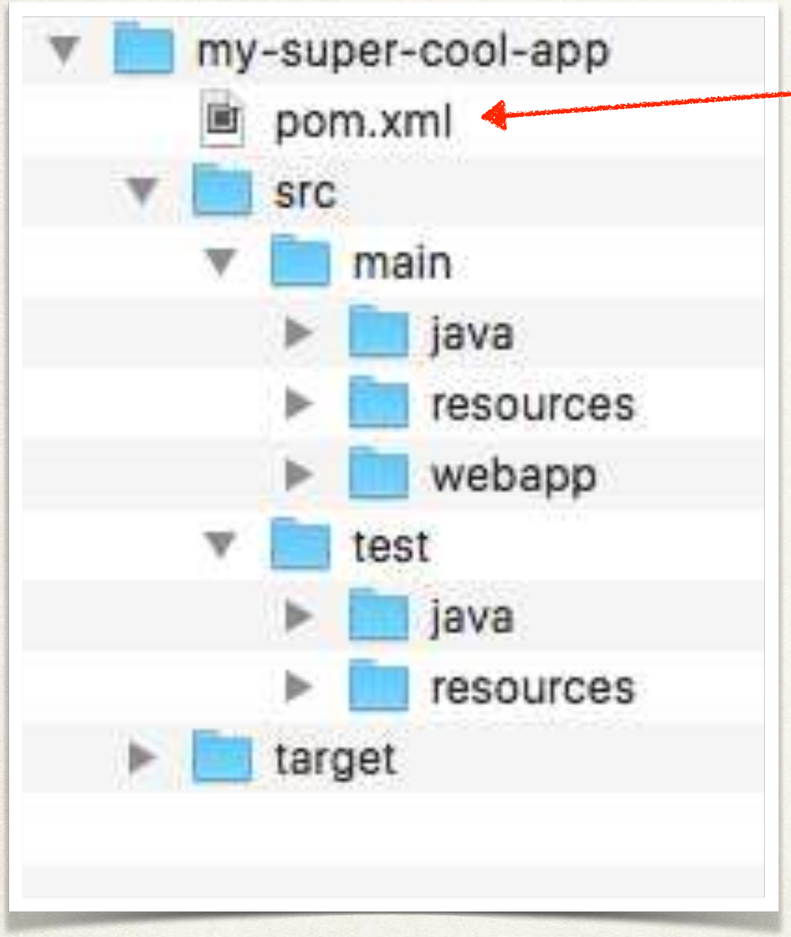
Standard Directory Structure Benefits

- 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

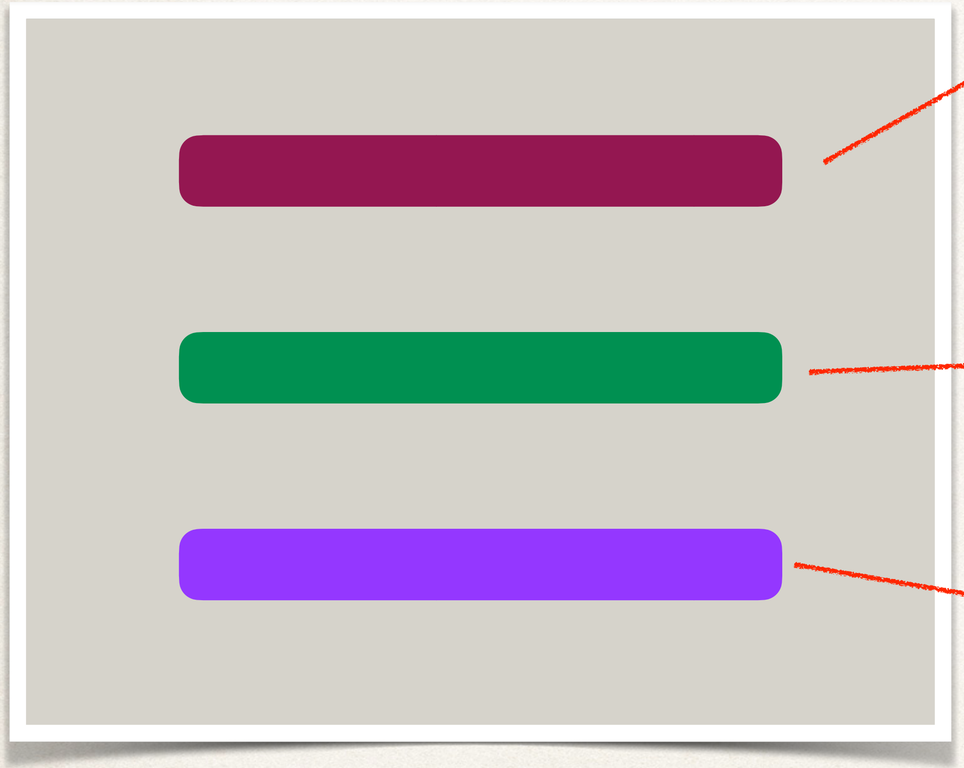
POM File - pom.xml



- 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…

Spring Boot: Run from Command-Line

Running from the Command-Line

- When running from the command-line
  - No need to have IDE open/running



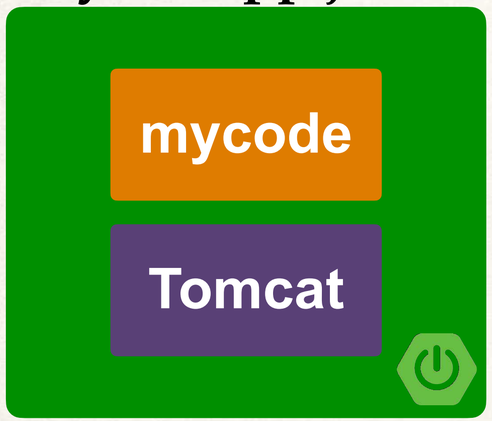
- Since we using Spring Boot, the server is embedded in our JAR file
  - No need to have separate server installed/running

- Spring Boot apps are self-contained

Running from the Command-Line

- Spring Boot apps are self-contained

mycoolapp.jar



Self-contained unit
Nothing else to install

JAR file
includes your application code
AND
includes the server

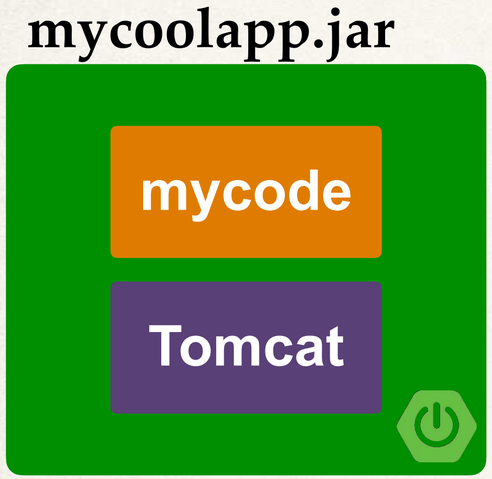
Running from the Command-Line

- Two options for running the app

- Option 1: Use **`java -jar`**

- Option 2: Use Spring Boot Maven plugin
  - **`mvnw spring-boot:run`**

Option 1: Use `java -jar`

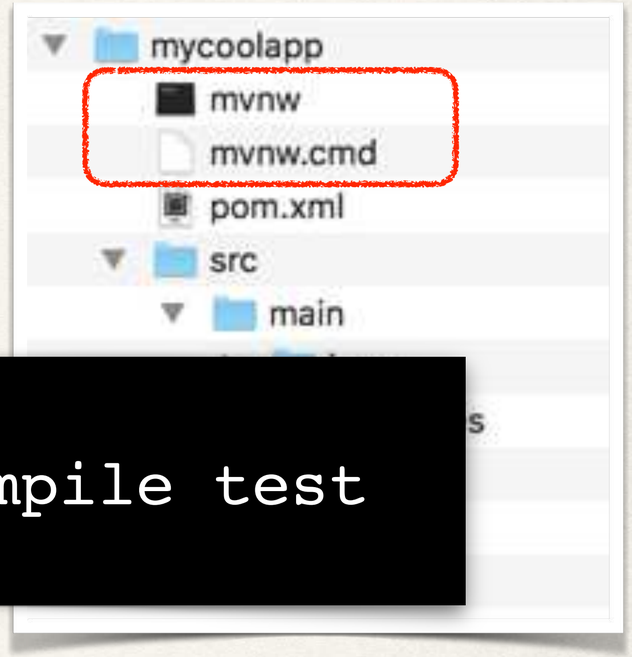



```
> java -jar mycoolapp.jar
```

Name of our JAR file

Option 2: Use Spring Boot Maven plugin

- **`mvnw`** allows you to run a Maven project
  - No need to have Maven installed or present on your path
  - If correct version of Maven is NOT found on your computer
    - **Automatically downloads** correct ve
      and runs Maven
- Two files are provided
  - **`mvnw.cmd`** for MS Windows
  - **`mvnw.sh`** for Linux/Mac



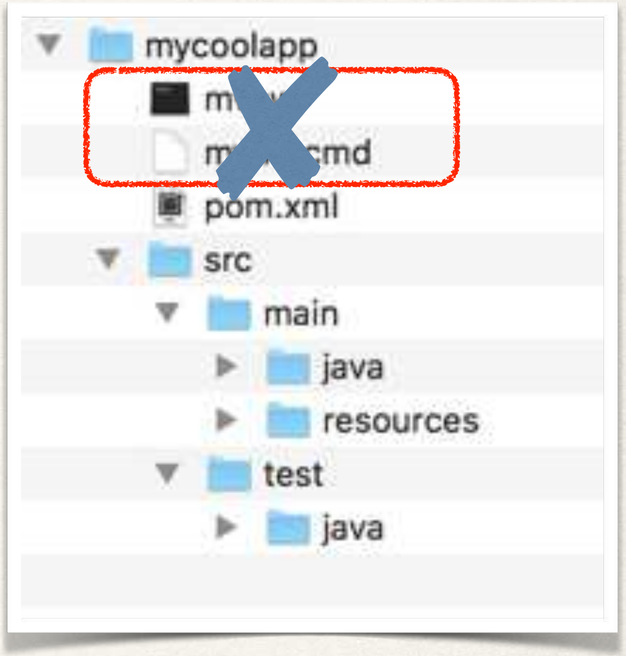

```
> mvnw clean compile test
```

```
$ ./mvnw clean compile test
```

Maven Wrapper files

- If you already have Maven installed previously
  - Then you can ignore/delete the **`mvnw`** files
- Just use Maven as you normally would

```
$ mvn clean compile test
```



Option 2: Use Spring Boot Maven plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

To package executable jar or war archive

Can also easily run the app

```
$ ./mvnw package

$ ./mvnw spring-boot:run
```

Can also just use:

mvn package
mvn spring-boot:run

Development Process

1. Exit the IDE

2. Package the app using **`mvnw package`**

3. Run app using **`java -jar`**

4. Run app using Spring Boot Maven plugin, **`mvnw spring-boot:run`**

Spring Boot - Custom Application Properties

Problem

- You need for your app to be configurable … no hard-coding of values

- You need to read app configuration from a properties file

Solution: Application Properties file

- By default, Spring Boot reads information from a standard properties file
  - Located at: **`src/main/resources/application.properties`**
- You can define ANY custom properties in this file
- Your Spring Boot app can access properties using **`@Value`**

Standard Spring Boot file name

No additional coding or configuration required

Step 1: Define custom application properties

File: src/main/resources/application.properties

```
#
# Define custom properties
#
coach.name=Mickey Mouse
team.name=The Mouse Club
```

You can give ANY custom property names

Step 2: Inject Properties into Spring Boot app

No additional coding or configuration required

```
@RestController
public class FunRestController {

  // inject properties for: coach.name and team.name

  @Value("${coach.name}")
  private String coachName;

  @Value("${team.name}")
  private String teamName;

  ...
}
```

File: src/main/resources/application.properties

```
#
# Define custom properties
#
coach.name=Mickey Mouse
team.name=The Mouse Club
```

Spring Boot Properties

Spring Boot Properties

- Spring Boot can be configured in the **`application.properties`** file
- Server port, context path, actuator, security etc …
- Spring Boot has 1,000+ properties … wowzers!

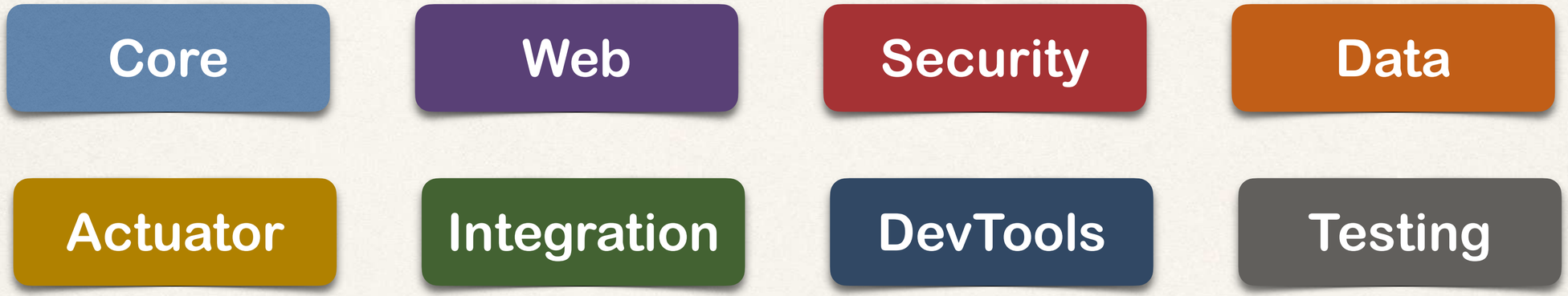
Spring Boot Properties

List of Common Properties

www.luv2code.com/spring-boot-props

Spring Boot Properties

- Don’t let the 1,000+ properties overwhelm you
- The properties are roughly grouped into the following categories



Spring Boot Properties

We'll review some of the properties …

Core Properties

Core

File: src/main/resources/application.properties

```
# Log levels severity mapping
logging.level.org.springframework=DEBUG
logging.level.org.hibernate=TRACE
logging.level.com.luv2code=INFO

# Log file name
logging.file.name=my-crazy-stuff.log
logging.file.path=c:/myapps/demo
...
```

Logging Levels

TRACE
DEBUG
INFO
WARN
ERROR
FATAL
OFF

Spring Boot Logging
www.luv2code.com/spring-boot-logging

Web Properties

Web

File: src/main/resources/application.properties

```
# HTTP server port
server.port=7070

# Context path of the application
server.servlet.context-path=/my-silly-app

# Default HTTP session time out
server.servlet.session.timeout=15m
...
```

http://localhost:7070/my-silly-app/fortune

15 minutes

Actuator Properties

Actuator

File: src/main/resources/application.properties

```
# Endpoints to include by name or wildcard
management.endpoints.web.exposure.include=*

# Endpoints to exclude by name or wildcard
management.endpoints.web.exposure.exclude=beans,mapping

# Base path for actuator endpoints
management.endpoints.web.base-path=/actuator
...
```

http://localhost:7070/actuator/health

Security Properties

Security

File: src/main/resources/application.properties

```
# Default user name
spring.security.user.name=admin

# Password for default user
spring.security.user.password=topsecret
...
```

Data Properties

Data

File: src/main/resources/application.properties

```
# JDBC URL of the database
spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce

# Login username of the database
spring.datasource.username=scott

# Login password of the database
spring.datasource.password=tiger
...
```

More on this in later videos

Spring Boot Properties

List of Common Properties

www.luv2code.com/spring-boot-props

Development Process

1. Configure the server port

2. Configure the application context path