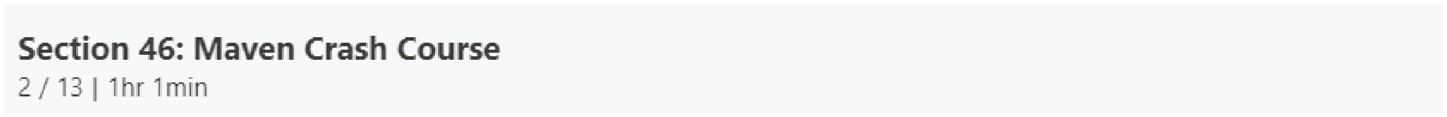
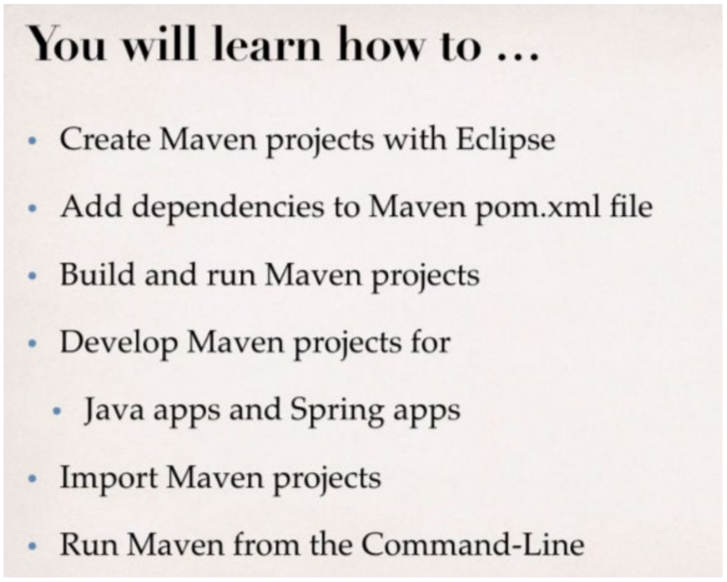

Section 46: Maven Crash Course
2 / 13 | 1hr 1min

<u>Maven Overview:</u>


You will learn how to …

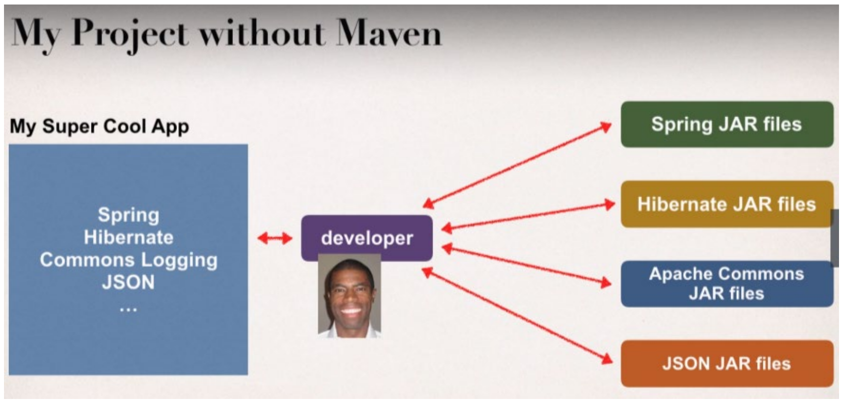
- Create Maven projects with Eclipse
- Add dependencies to Maven pom.xml file
- Build and run Maven projects
- Develop Maven projects for
  - Java apps and Spring apps
- Import Maven projects
- Run Maven from the Command-Line

Maven is a project management tool for our application.

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

<u>What Problems does Maven solve?</u>

- When building our java project, wwe 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.
- Then manually add those JAR files to your build path/classpath.

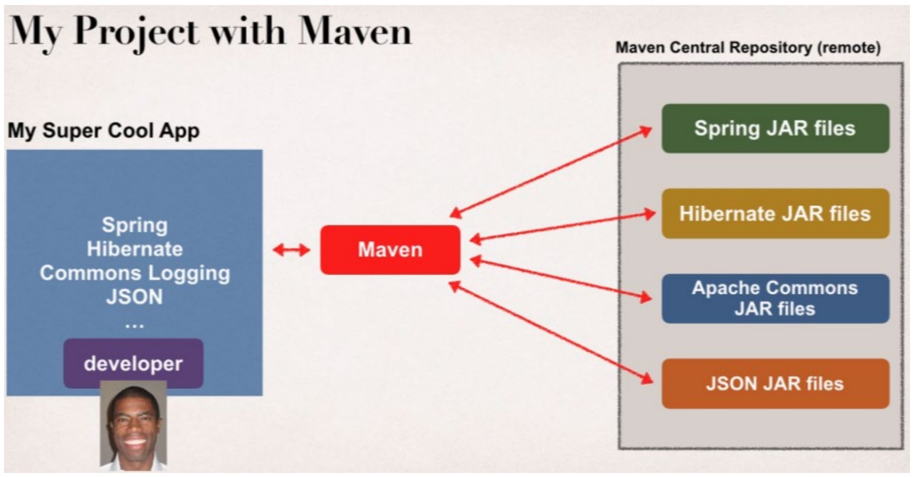


Maven Solution

- Tell Maven the projects that you are working with (Dependencies).
  - Spring, Hibernate etc.
- Maven will go out and download the JAR Files for those projects for us.
- And Maven will make those JAR Files available during compile and runtime.
- So, think of Maven as your friendly helper/personal shopper.

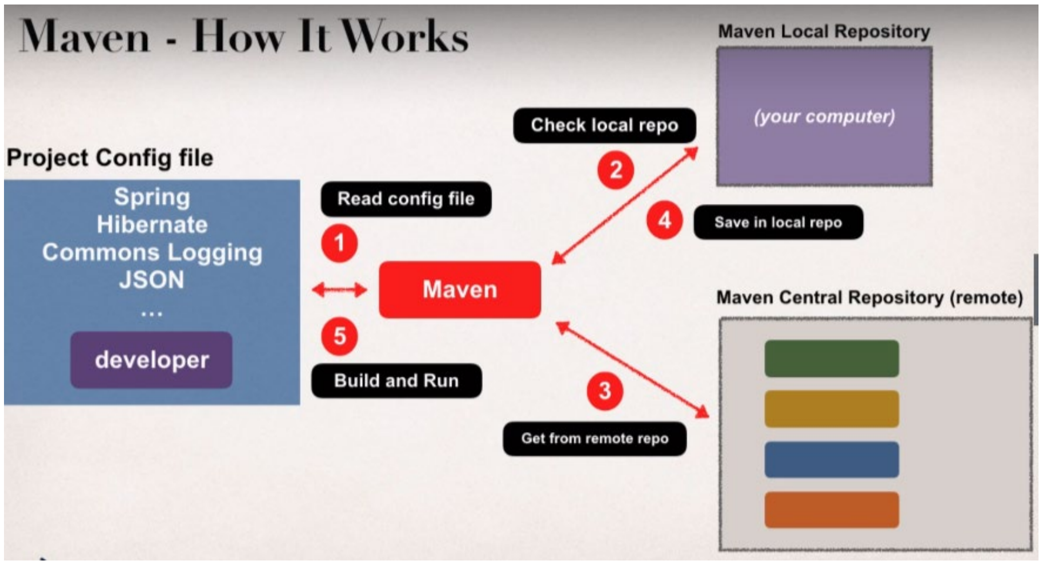
My Project with Maven

There is Maven Central Repository which is on internet, then in our app we simply tell Maven that here is our shopping list then Maven will go off and download those spring Jar files, Hibernate Jar files commons and JSON etc. and make these available to use into our app.



Maven – How It Works?

Using Maven, we will have a project config file which Maven will read then Maven will check in our Maven Local Repository which resides in our computer, and if we don't have files in our local repo then Maven will actually go out to the internet at the Maven Central Repository and it will pull those Jar files from there and then it saves them into our local repo. Later maven will use that to build and run the application.



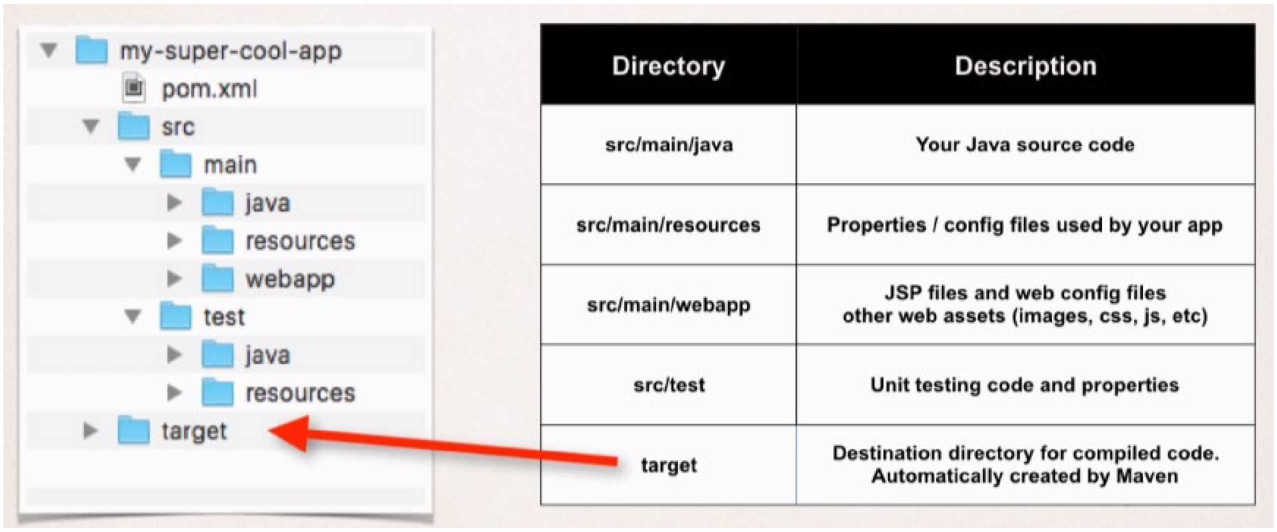
Handling Jar Dependencies:

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

Building and Running

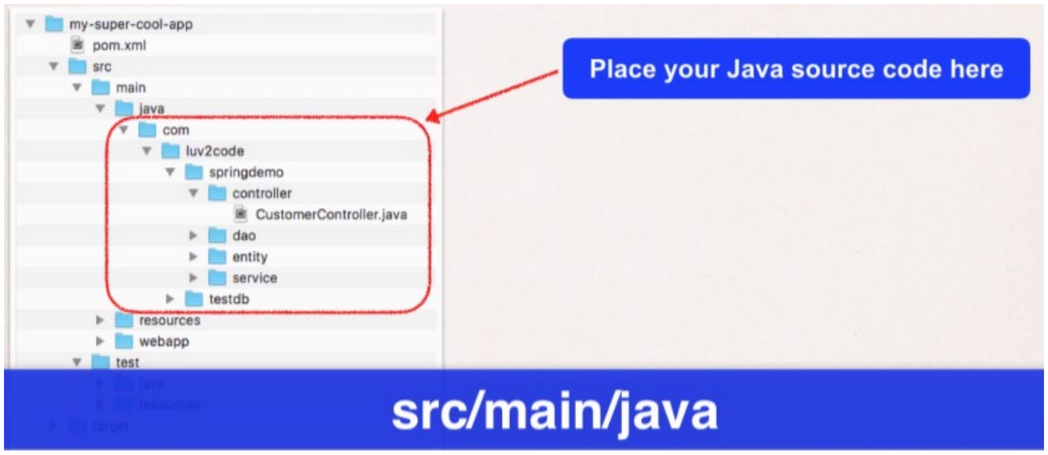
- When we build and run our app
- Maven will handle class/build path for us
- Based on config file, maven will add JAR files accordingly

Maven Provides a standard directory structure for a project.

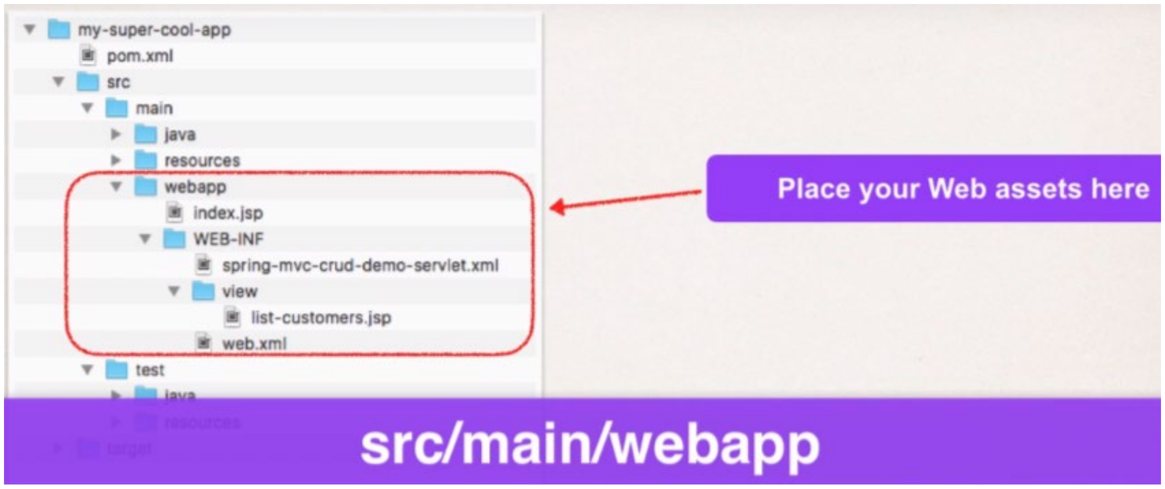


```
my-super-cool-app
    pom.xml
    src
        main
            java
            resources
            webapp
        test
            java
            resources
    target
```

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



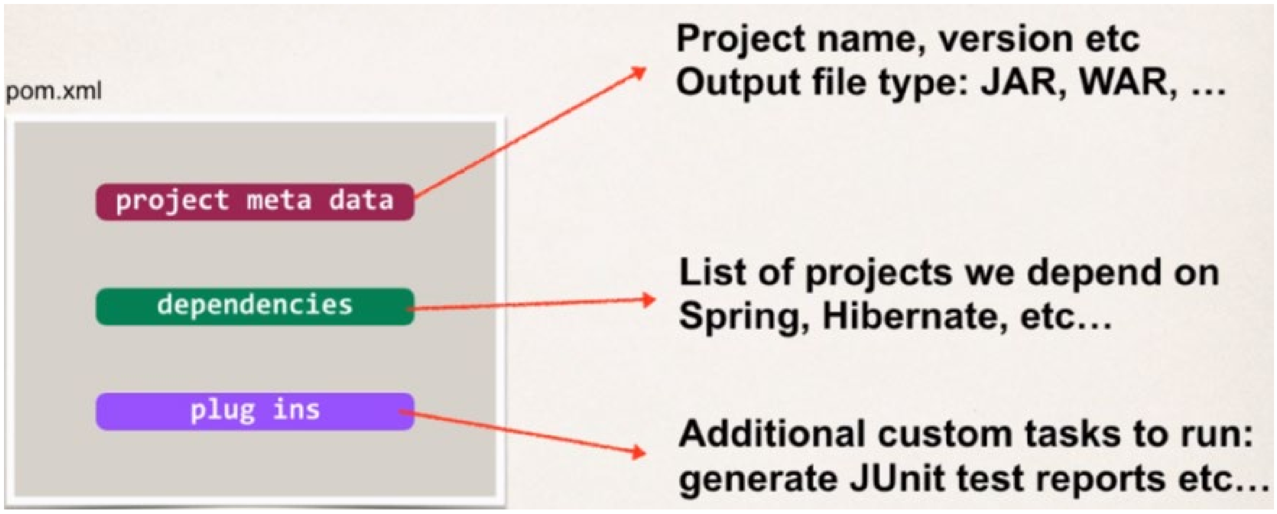
And if we are working on a web project then we will keep our web assets (like JSP files, configurations files, CSS, images and so on) in:



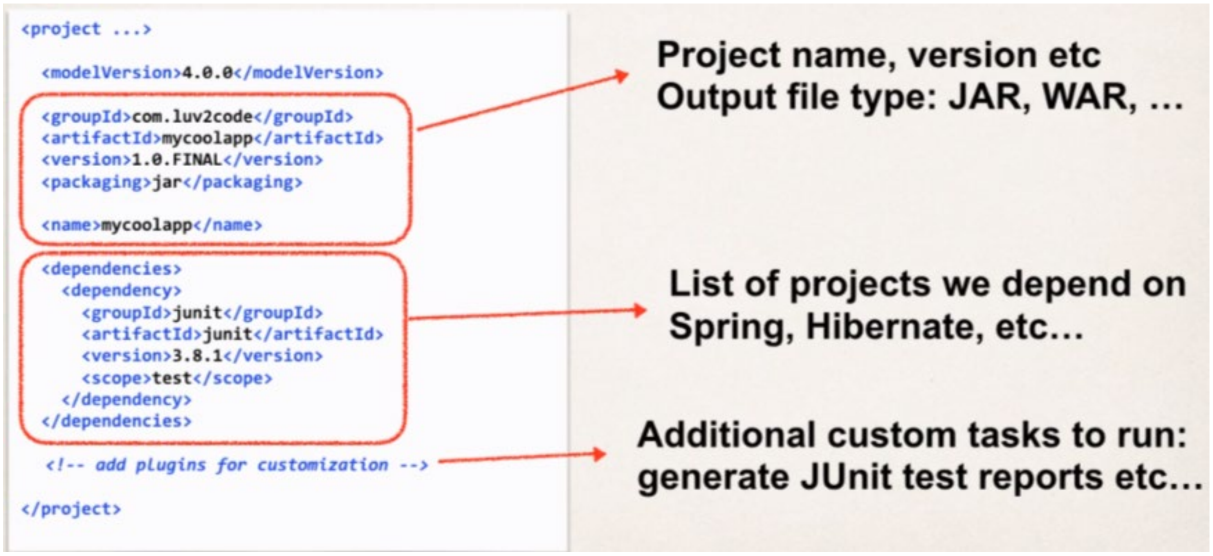
Maven Key Concepts:

❖ <u>POM File – pom.xml</u>
  - POM File which stands for **P**roject **O**bject **M**odel File.
  - This is our configuration file
  - Basically our ''Shopping List" for Maven.
  - This POM file is always located in root of our maven project.

POM File Structure:


pom.xml
project meta data
dependencies
plug ins
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...

Sample POM File:

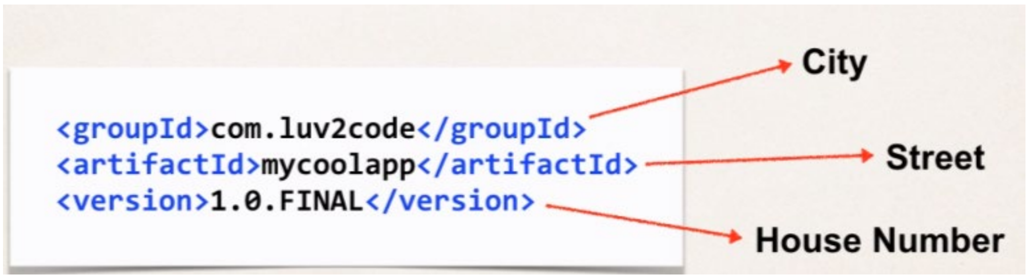


```
<project ...>

  <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>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <!-- add plugins for customization -->

</project>
```

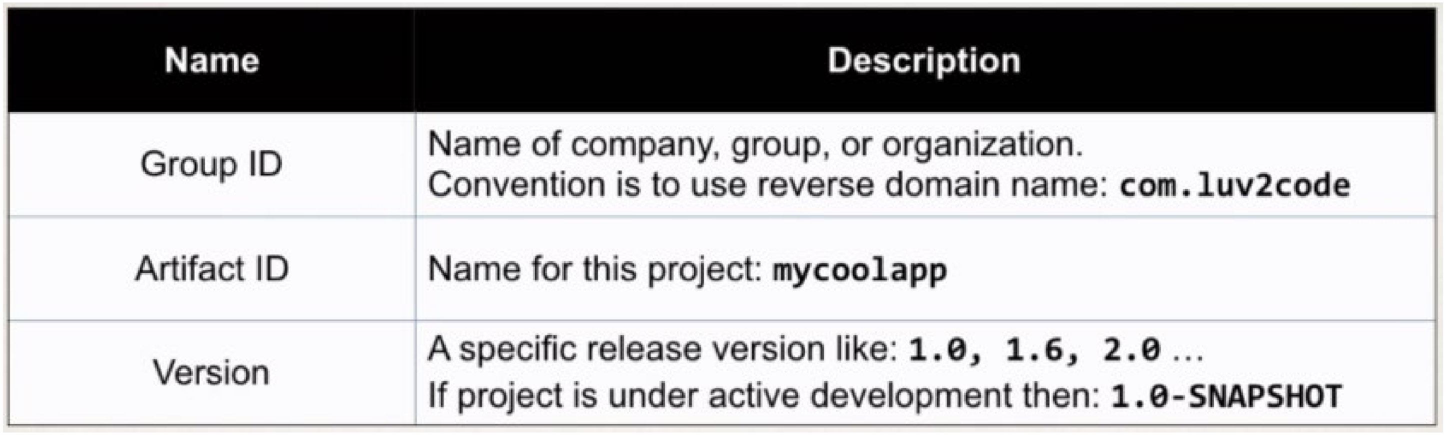
Project name, version etc
Output file type: JAR, WAR, ...
List of projects we depend on
Spring, Hibernate, etc...
Additional custom tasks to run:
generate JUnit test reports etc...

Project Coordinates:

- Project coordinates uniquely identify a project.
- It similar to GPS coordinates for your house: latitude/longitude
- It precise on information for how to find our house (city, street, house number)



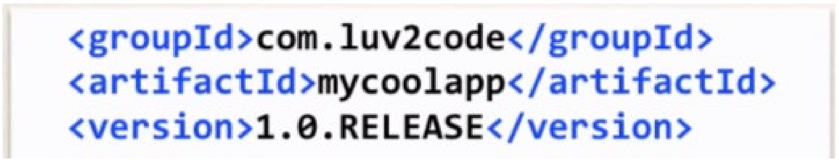
```xml
<groupId>com.luv2code</groupId>
<artifactId>mycoolapp</artifactId>
<version>1.0.FINAL</version>
```



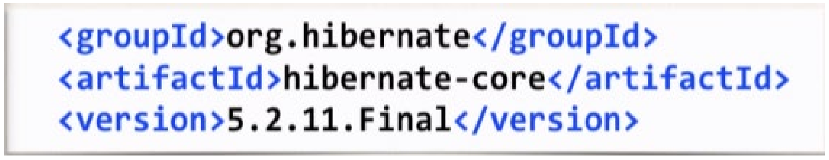
Name	Description
Group ID	Name of company, group, or organization. Convention is to use reverse domain name: **`com.luv2code`**
Artifact ID	Name for this project: **`mycoolapp`**
Version	A specific release version like: **`1.0, 1.6, 2.0`** … If project is under active development then: **`1.0-SNAPSHOT`**

If the project is under active development, then we can use 1.0-SNAPSHOT, which means that this work is still under active development.

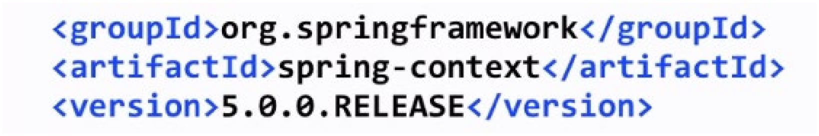
Example of Project coordinates:



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

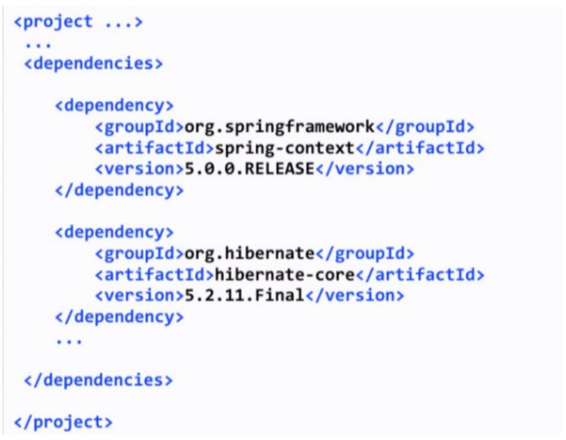


```xml
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.2.11.Final</version>
```



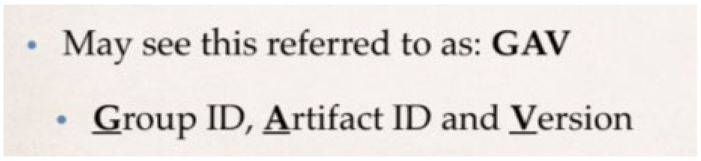
```xml
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>5.0.0.RELEASE</version>
```

Adding Dependencies:



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

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.0.0.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.11.Final</version>
    </dependency>
    ...

 </dependencies>

</project>
```

Dependency Coordinates:



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

To add given dependency project, we need

- Group ID, Artifact ID
- Version is optional

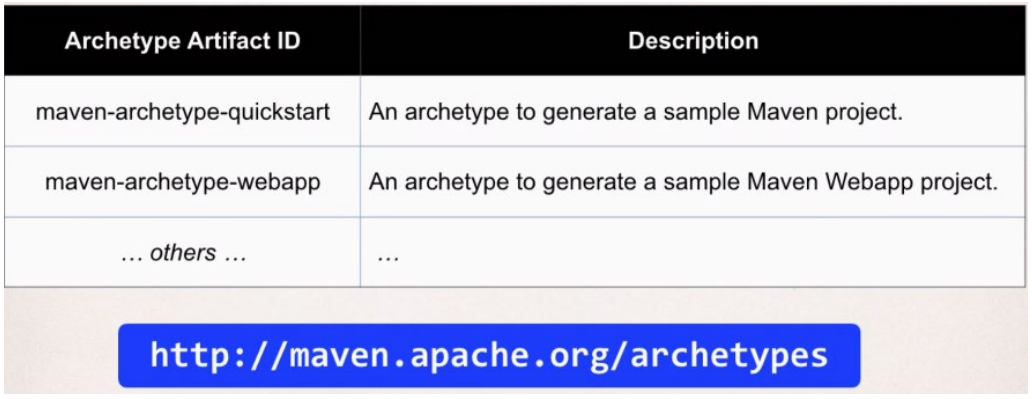
Best practice is to include the version (for repeatable builds). In this way we can say our project works with version xyz of a given project. It's very important in **DevOps** world.

How to find Dependency Coordinates:

1. Visit the project page (spring.io, hibernate.org etc)
2. Visit http://search.maven.org (easiest approach)

Maven Archetypes:

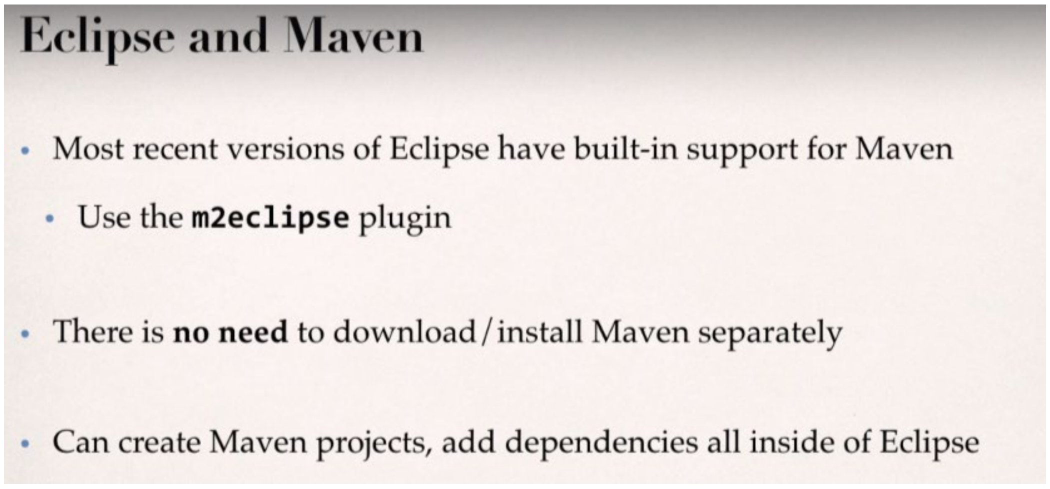
- Archetypes can be used to create a new Maven Projects.
- Contains template files for a given Maven project.
- Think of it as a collection of "Starter files" for a project
  - Either we are starting a Java Project or a Web Project, etc then maven will give us some starter files for that project.



Archetype Artifact ID	Description
maven-archetype-quickstart	An archetype to generate a sample Maven project.
maven-archetype-webapp	An archetype to generate a sample Maven Webapp project.
... others ...	...

http://maven.apache.org/archetypes

- We can create new projects using these Maven Archetypes (starter project)
  - Form the command line with Maven
  - From an IDE ex. Eclipse, IntelliJ, NetBeans, etc.



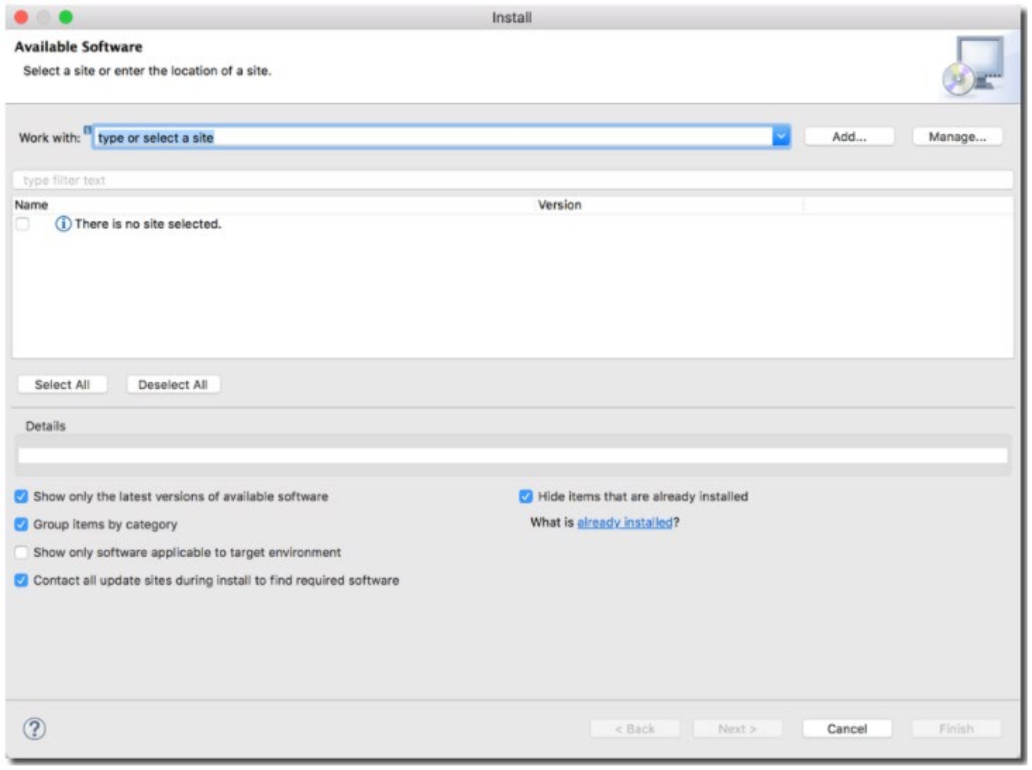
Eclipse and Maven

- Most recent versions of Eclipse have built-in support for Maven
  - Use the **`m2eclipse`** plugin
- There is **no need** to download/install Maven separately
- Can create Maven projects, add dependencies all inside of Eclipse

How to Install the m2eclipse Plugin

How to Install Maven m2eclipse Plugin?
Note: Recent releases of Eclipse (Mar, Neon, Oxygen) already have the m2eclipse plugin installed. Only follow these steps if you don't have the plugin installed.

1. In Eclipse, select **Help > Install New Software**
This will display the Install dialog



Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

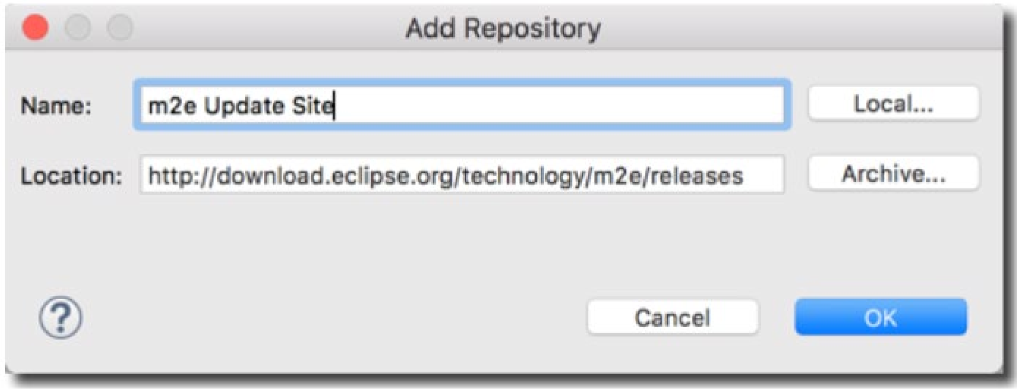
Name | Version

There is no site selected.

Details

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☑ Hide items that are already installed
What is already installed?

2. Click the **"Add..."** button.
3. In the dialog, enter the following from the screenshot below:



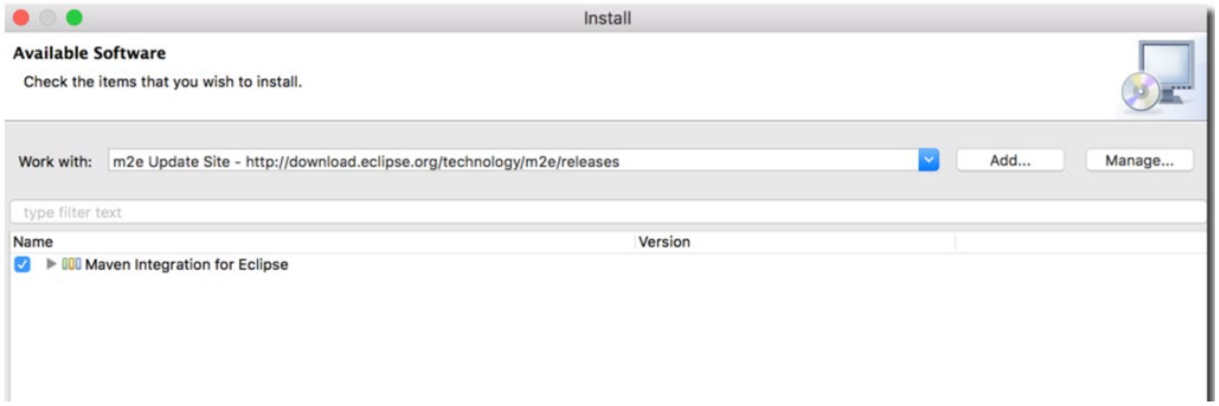
Add Repository

Name: m2e Update Site

Location: http://download.eclipse.org/technology/m2e/releases

4. Wait for a couple of minutes while Eclipse fetches the files

5. In the dialog, check the box for **"Maven Integration for Eclipse"**



m2e Update Site - http://download.eclipse.org/technology/m2e/releases

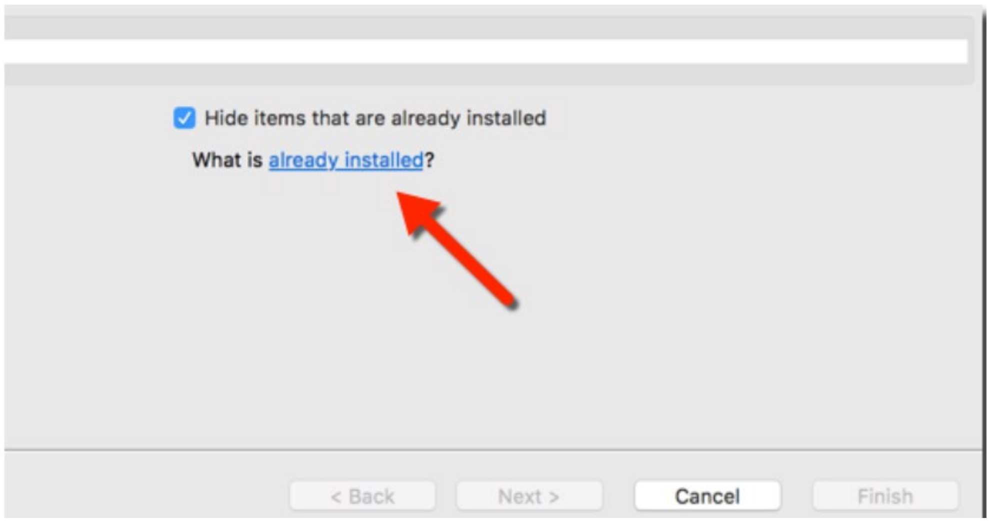
☑ Maven Integration for Eclipse

6. Click Next

7. It will show a dialog to review components, Click **Next**
8. Accept the License Agreement, Click **Finish**
9. Once installation is complete, restart Eclipse.

Verify Installation

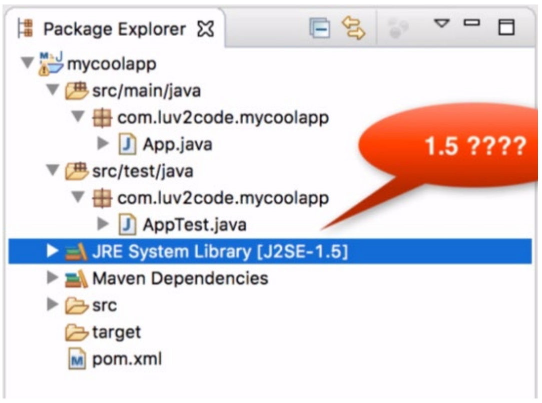
10. In Eclipse, select **Help > Install New Software**
11. In bottom right of dialog, you will see a blue link: **What is already installed?**


Hide items that are already installed
What is already installed?
< Back
Next >
Cancel
Finish

Create Maven Project – Quickstart archetypes

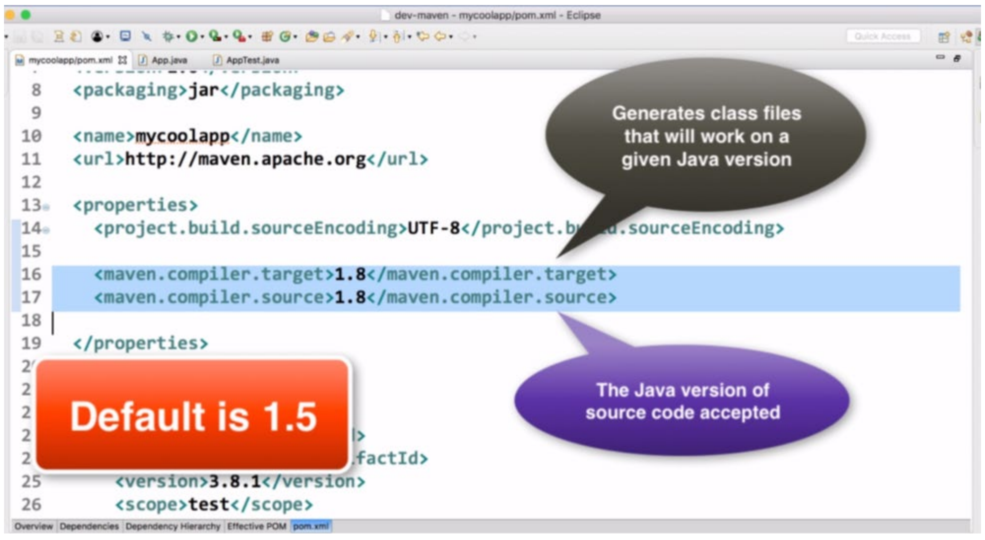
When we create the Maven Project then we get to see that our project is using Java 5. So, by Default Maven will use java 5 but since we are using higher version of Java so we need to update our pom.xml file.

I need to tell the Maven that the compiler version that it should use is 1.8 since currently we are using java 8 in our system.



"source" xml tag is the actual version of the source code that is accepted and then the "target" is actually the generated class files that will run on a given Java version.

If we don't specify these values then default will be 1.5 Java version.

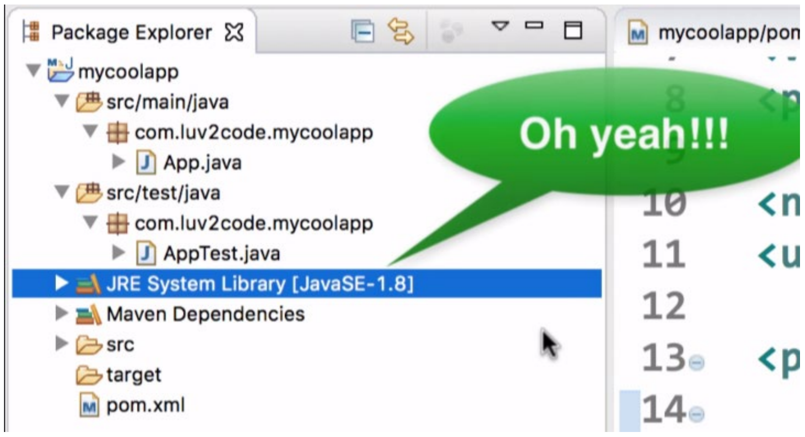

dev-maven - mycoolapp/pom.xml - Eclipse
mycoolapp/pom.xml
App.java
AppTest.java
Generates class files that will work on a given Java version
The Java version of source code accepted
Default is 1.5
Overview
Dependencies
Dependency Hierarchy
Effective POM
pom.xml

```xml
<packaging>jar</packaging>

<name>mycoolapp</name>
<url>http://maven.apache.org</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.b   .sourceEncoding>

  <maven.compiler.target>1.8</maven.compiler.target>
  <maven.compiler.source>1.8</maven.compiler.source>

</properties>
                                   >
                                   .factId>
    <version>3.8.1</version>
    <scope>test</scope>
```

After making the modification onto Pom.xml file

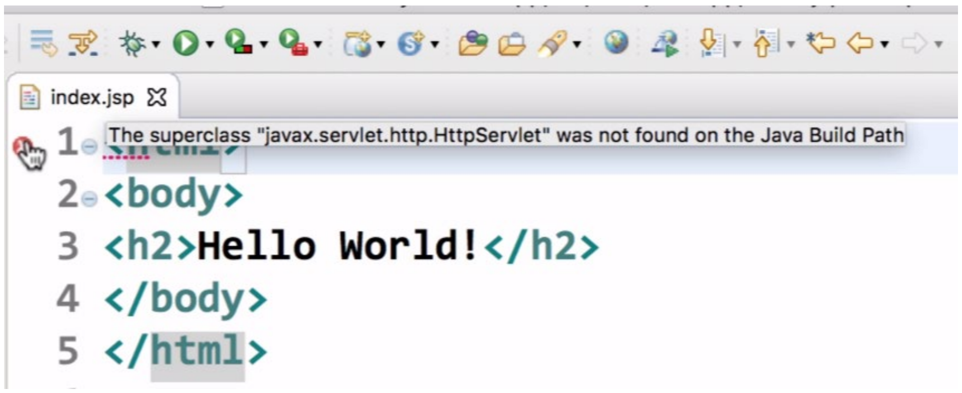
- Right click on **project** > **maven** > select **update project**
- **Now make sure to select last three checkboxes i.e, update, refresh, clean project**
- Then click on **OK**

Then our project will get updated as per our java version.



Creating Maven Project – maven-archetype-webapp

On creating the project, we will get to see the error in our **`index.jsp`** page.



The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

Error: The superclass “javax.servlet.http.HttpServlet” was not found on the Java build path

Since we are using Maven, which means that we have a dependency which is missing.

So, we need to update our pom.xml file since Maven is handling all the building and handling classpath issues.

In the pom.xml, we can see that there is nothing regarding **<u>`javax.servlet`</u>**

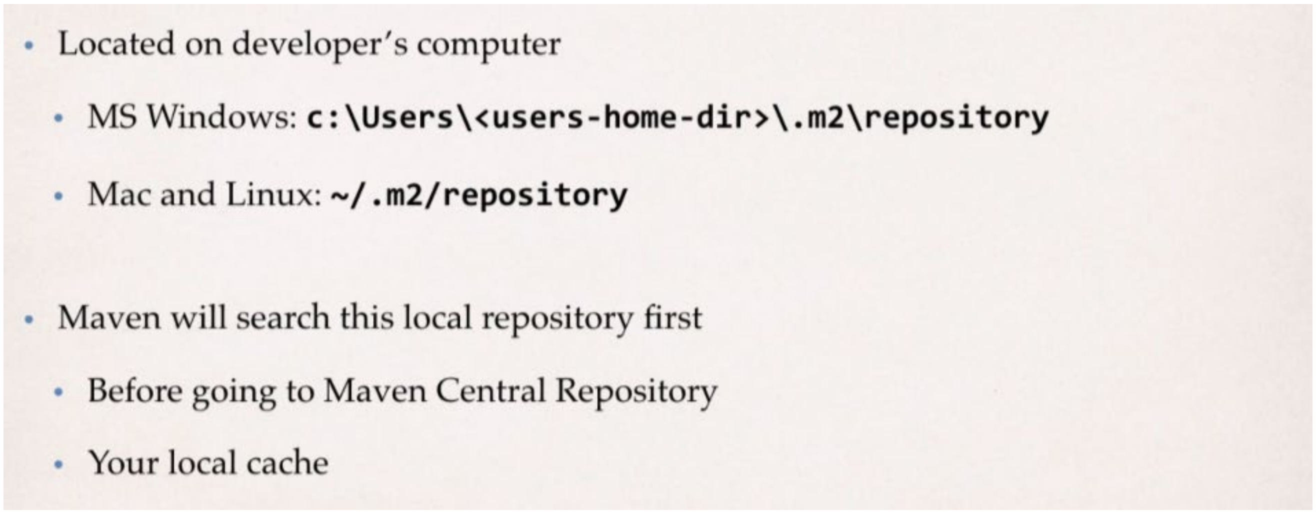
```
</dependencies>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
       <version>4.11</version>
       <scope>test</scope>
    </dependency>
  </dependencies>
```

So, we need to add servlet dependency as well:

```
<!-- we need to add the servlet API dependency: and the artifact id is: javax.servlet-api -->
      <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
      </dependency>
```

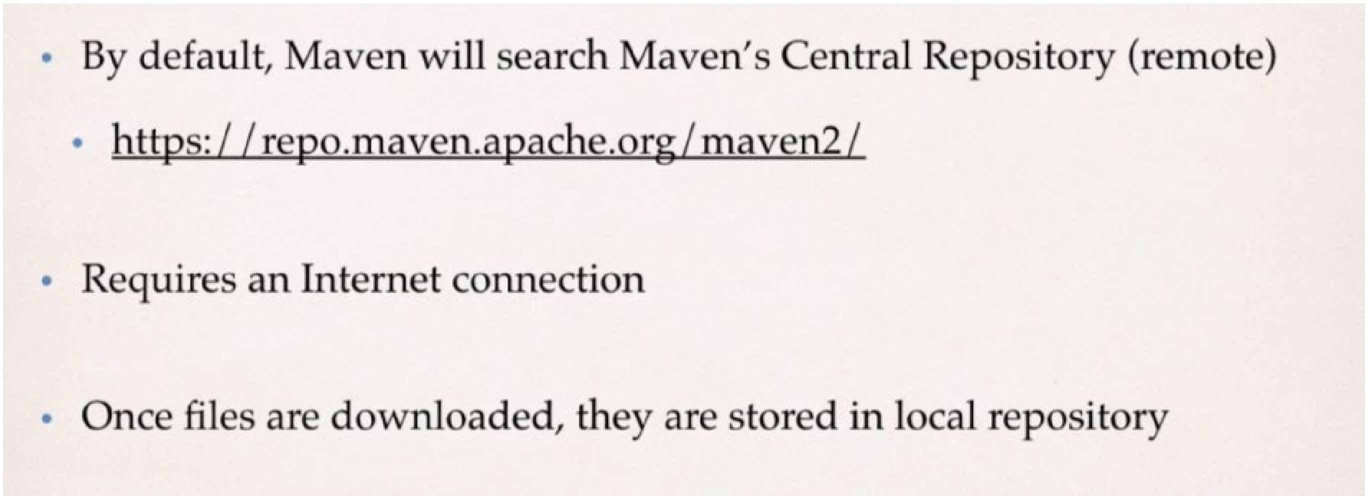
Now, save the pom.xml file and incase the error still persist then go and update the Maven Project and then the error will be gone!

- **Maven Local Repository:**



- Located on developer's computer
  - MS Windows: `c:\Users\<users-home-dir>\.m2\repository`
  - Mac and Linux: `~/.m2/repository`
- Maven will search this local repository first
  - Before going to Maven Central Repository
  - Your local cache

- **Maven Central Repository**



- By default, Maven will search Maven's Central Repository (remote)
  - https://repo.maven.apache.org/maven2/
- Requires an Internet connection
- Once files are downloaded, they are stored in local repository

- **Private Repositories:**

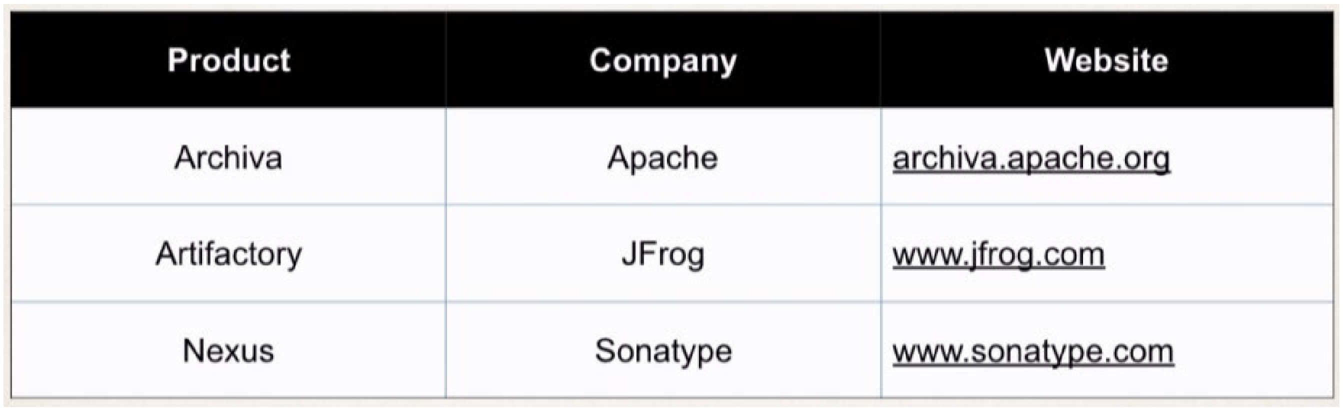
Use Case:

- Let say the organization has created super-top-secret code modules.
- Would like to share with other development teams at the company.
- But like to keep it private and it NOT AVAILABLE TO THE PUBLC.

So, we will set up a server in house.

- We can set up our own private Maven Repository.
  - Secure it with credentials: id/password.
- Then we can Create those super-top-secret projects and publish on private repository.
- And then our development team can access the private repo using their credentials.

So, we need to setup the Maven Server at our company; and we can make use of any one of the server products listed below:



Product	Company	Website
Archiva	Apache	archiva.apache.org
Artifactory	JFrog	www.jfrog.com
Nexus	Sonatype	www.sonatype.com

✓ And if don't want to do self-host internally at the company then we can make use of cloud:

i.e., cloud hosted solutions are available

Check for www.packagecloud.io, www.mymavenrepo.com