**46.1. Maven Overview**

**What is Maven**:

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.

In short terms we can tell maven is a tool that can be used for building and managing any Java-based project. maven make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

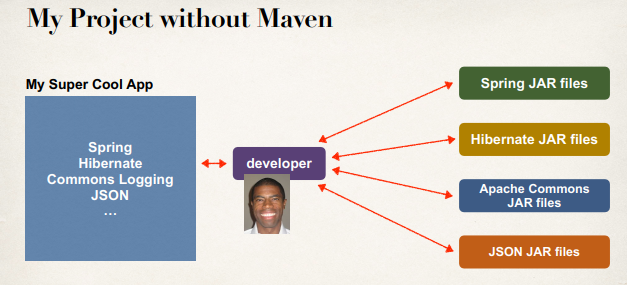
On a word maven is a project management tool for our application. The most popular use of maven is for build management and dependencies.

**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 / class-path

**Java project without Maven**:

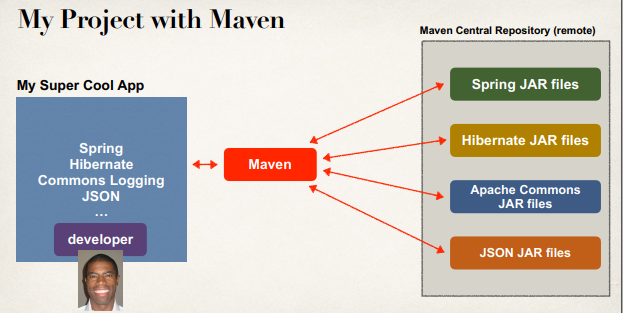


Maven can actually help us with this process and do a lot of works for us.

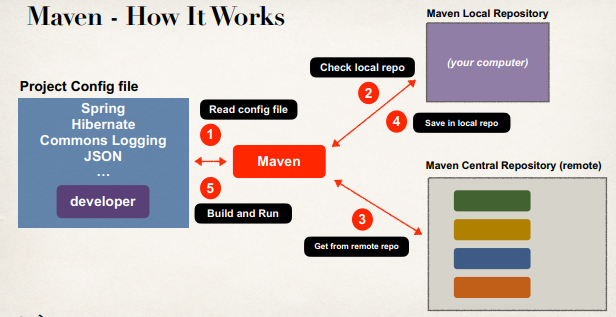
**Maven Solution**:

* Tell Maven the projects we 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/run
* Think of Maven as our friendly helper / personal shopper :-)

**My Project with Maven**:



**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

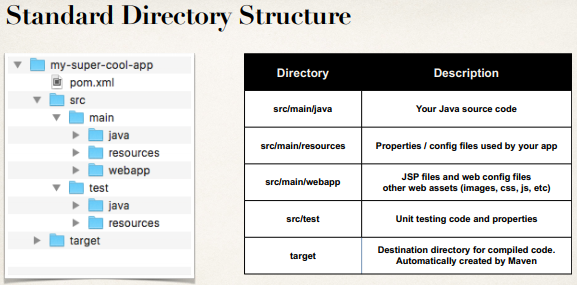
**Building and Running**:

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

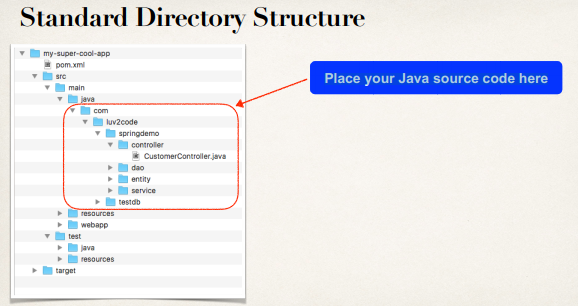
**Standard Directory Structure**:

* Normally when we 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

**Maven Directory Structure**:



**Place Java source code – Maven project**:



**Standard Directory Structure Benefits**:

* For new developers joining a project
  + They can easily find code, properties files, unit tests, web files etc.
* 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!

**Advantages of Maven**:

* Dependency Management
  + Maven will find JAR files for us
  + No more missing JARs
* Building and Running our Project
  + No more build path / class-path issues
  + Standard directory structure

**Best Maven Benefit(s)**:

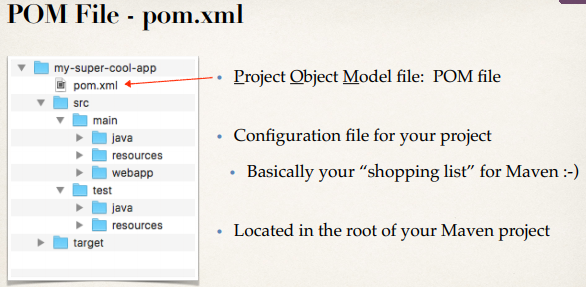
* Once we learn Maven, we can join a new project and be productive
* We can build and run a project with minimal local configuration

**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 our Maven project



**POM File Structure**:

* Project name, version etc.

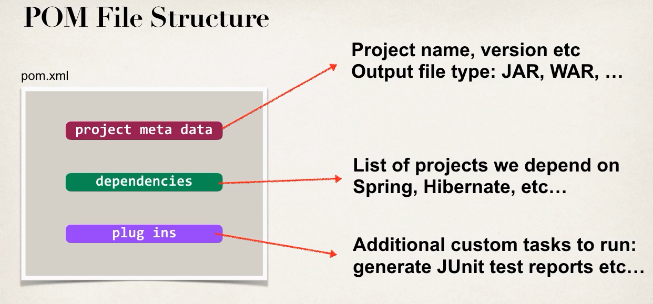
Output file type: JAR, WAR, …

* List of projects we depend on

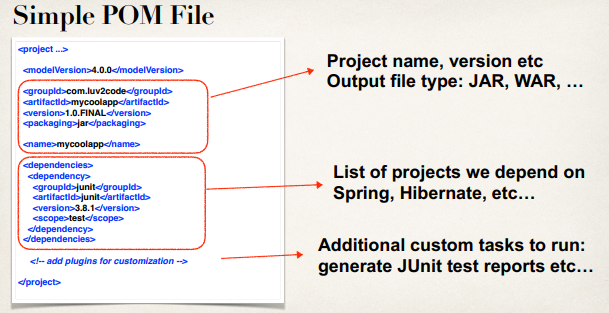
Spring, Hibernate, etc.…

* Additional custom tasks to run:

generate JUnit test reports etc.…



**Simple POM File**:



**Project Coordinates**:

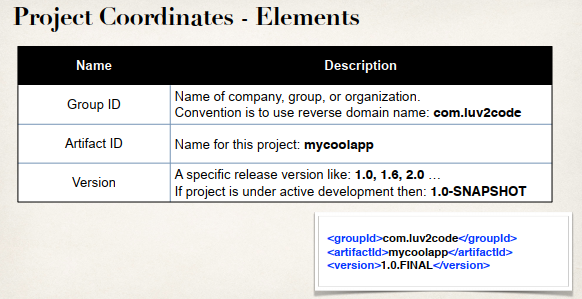
* Project Coordinates uniquely identify a project
  + Similar to GPS coordinates for our house: latitude / longitude
  + Precise information for finding our house (city, street, house #)



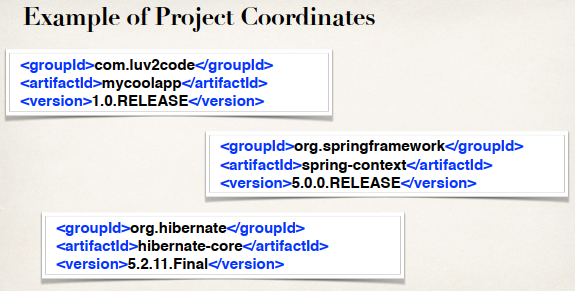
**Project Coordinates – Elements**:

|  |  |
| --- | --- |
| 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** |

**Project Coordinates – Elements – Example**:



**Example of Project Coordinates**:



**Adding Dependencies**:



**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
  + Group ID, Artifact ID and Version

**How to Find Dependency Coordinates**:

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

**Option 2**: Visit <http://search.maven.org> (easiest approach)

**Maven Archetypes**:

* Archetypes can be used to create new Maven projects
* Contains template files for a given Maven project
* Think of it as a collection of "starter files" for a project
  + Java Project, Web Project, etc.

**Common Archetypes**:

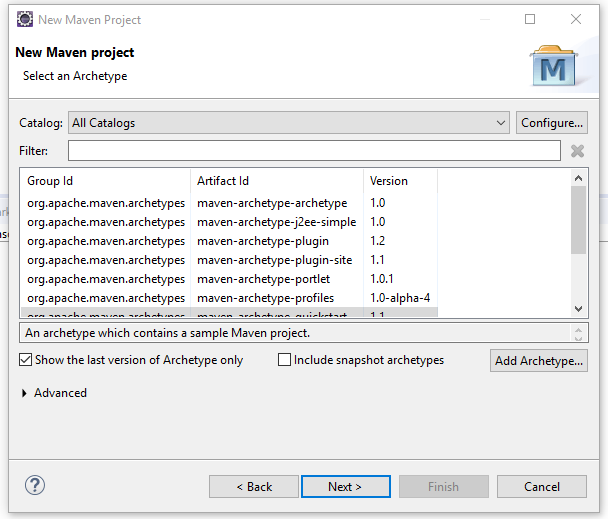
|  |  |
| --- | --- |
| **Archetype Artifact ID** | **Description** |
| maven-archetype-quickstart | An archetype to generate a sample Maven project. |
| maven-archetype-webapp | An archetype to generate a sample Maven Webapp project. |
| … others … | … |

There is another archetype out there. <http://maven.apache.org/archetypes>

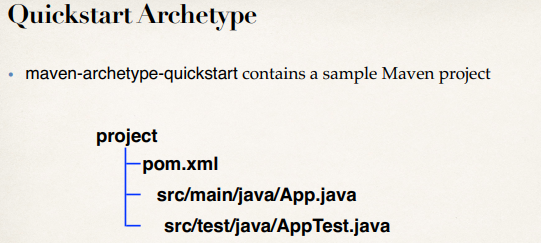
**Archetypes**:

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

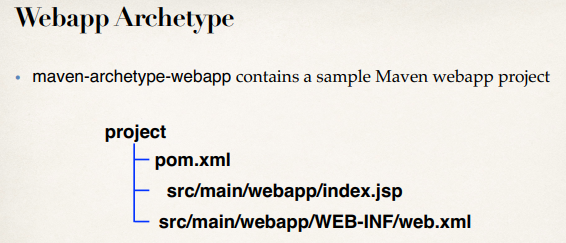
**List of Archetypes – Eclipse**:



**Quickstart Archetype**:



**Webapp Archetype**:



46.1. Maven Overview