

Maven Basics

“Code with Passion”



Topics

- What is Maven?
- Maven installation
- Creating “helloworld” Maven project
- POM
- Archetype
- Plugins and goals
- Lifecycles and phases
- Repositories
- Dependency management
- Maven Web Project structure
- Maven vs. Ant

Topics covered in “Maven Advanced”

- Multi-module project
- Grouping Dependencies
- POM inheritance
- Profiles
- Dependency management
- Site generation

What is Maven?

What is Maven?

- Software project management tool
 - > It is more than a build tool
- Project Object Model (POM) based - pom.xml
 - > pom.xml maintains project's build, reporting and documentation
 - > POM can be inherited between parent and child project
- Based on “Convention over configuration” principle
 - > Minimum configuration is needed
- You specify “what needs to be done”
 - > Not “how it needs to be done” (as in the case of Ant)
- Plug-in architecture
 - > Vibrant Maven Eco-system

Maven's Objectives

- Making the build process easy
 - > Why do we have to waste so much time maintaining the build?
- Providing a uniform build system
 - > Why each developer has to maintain their own build environment?
- Providing quality project information
 - > Why do we have to do extra work to get project info?
- Providing guidelines for best practices development
 - > How can we capture best practices?
- Allowing transparent migration to new features
 - > How can tool vendors to do their innovation without affecting my build?

Providing Quality Project Information

- Maven provides plenty of useful project information that is in part taken from your POM and in part generated from your project's sources.
 - > Change log document created directly from source control
 - > Cross referenced sources
 - > Mailing lists
 - > Dependency list
 - > Unit test reports including coverage
 - > Many more (through plug-ins)

Maven Features (1)

- Simple project setup that follows best practices - get a new project or module started in seconds
- Consistent usage across all projects means no ramp up time for new developers coming onto a project
- Superior dependency management including automatic updating, transitive dependencies
- Able to easily work with multiple projects at the same time

Maven Features (2)

- A large and growing repository of libraries and metadata to use out of the box, and arrangements in place with the largest Open Source projects for real-time availability of their latest releases
- Extensible, with the ability to easily write plugins in Java or scripting languages
- Instant access to new features with little or no extra configuration

Maven Installation

Installation is simple

- Download it and unzip it
 - > <http://maven.apache.org/download.html>
- Make sure JAVA_HOME environment variable is set to JDK directory
- Add it to the PATH environment variable
 - > Linux

```
export MAVEN_HOME=/home/sang/apache-maven-3.0.1
```

```
export PATH=$PATH:$MAVEN_HOME/bin
```

- > Windows

```
set M2_HOME=:\Program Files\apache-maven-3.0.1
```

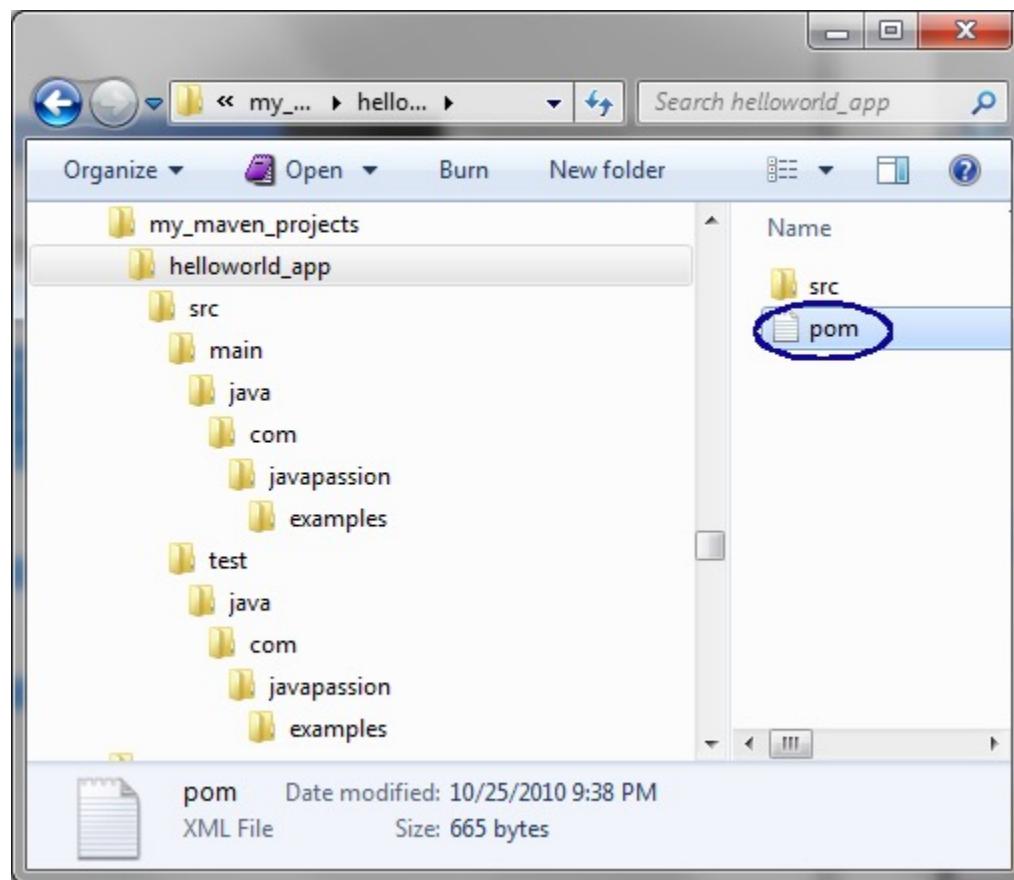
```
set PATH=%PATH%;%M2_HOME%\bin
```

Creating “Helloworld” Maven Project

Steps to create a simple project

- *mvn archetype:generate*
 - > Asking Maven to generate a Maven project
- You will be prompted to provide the following info.
 - > Archetype (project type)
 - > Group Id
 - > Artifact Id
 - > Version
 - > Package
- End result
 - > Project directory structure (for a chosen archetype)
 - > *pom.xml*

Maven Created pom.xml



Maven Created Directory Structure (for maven-archetype-quickstart)

- <name of the project>
 - > src/
 - > main/java/
 - com/javapassion/examples
 - > test/java/
 - com/javapassion/examples

Lab:

Exercise 1: Create a simple
Maven Project at the commandline

Step 1-3

[5072_tools_maven.zip](#)



**pom.xml
(POM file)**

What is POM file?

- Contains project information such as
 - > What type of project?
 - > What is the project's name?
 - > What is the project's identity (coordinates)?
 - > What are the build customizations?
 - > What are the dependencies?
 - > What plug-ins are used?

POM file (Simplest version)

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javapassion.examples</groupId>
  <artifactId>helloworld_app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld_app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Project Identity (Artifact Identity)

- Everything (your projects, plug-in's, dependencies) in Maven world is a project and every project has a unique identity
- Project identity is specified by project “coordinates”
 - > Consider it as an address for a specific point in “space”
 - > Uniquely identifies a project in repositories
- Dependencies and parent references are described with their own project coordinates
- Created with the combination of the following

```
<groupId>com.javapassion.examples</groupId>
<artifactId>helloworld_app</artifactId>
<version>1.0-SNAPSHOT</version>
```

Project Identity

- groupId
 - > Typically represents an organization
 - > Convention is using reverse domain name
 - > Example: com.jpassion.examples
- artifactId
 - > A unique identifier under groupId
- version
 - > A specific version of a project under artifactId

Archetype

What is Archetype?

- “Archetype” is “an original model or type after which other similar things can be patterned or prototyped”
 - > Can be thought of a project template
- Captures the best practices
 - > Directory structure, dependencies, plugin's needed
- There are many archetypes already provided by Maven community
 - > Simple Java SE app
 - > Spring app
 - > Hibernate app
 - > JSF app
 - > Java EE 6 app
 - > Many more

User-specific configuration & Local repository

User specific configuration and Local Repository

- <Home_directory>/.m2/settings.xml
 - > Contains user specific configuration for authentication, repositories, proxy server and port, and other information to customize the behavior of Maven
- <Home_directory>/.m2/repository
 - > Local Maven repository
 - > Stores locally generated artifacts (jar files, war files, etc.)
 - > Stores copies of dependencies and plug-in's downloaded from remote repositories

<Home_directory>/.m2/repository

```
Administrator: Command Prompt
Volume Serial Number is F090-5679
Directory of C:\Users\sang\.m2\repository

10/26/2010  03:31 PM <DIR> .
10/26/2010  03:31 PM <DIR> ..
10/21/2010  06:10 PM <DIR> .cache
10/26/2010  03:05 PM <DIR> ant
10/08/2010  01:59 PM <DIR> aopalliance
10/26/2010  11:38 AM <DIR> asm
10/25/2010  11:10 PM <DIR> biz
12/20/2009  11:48 PM <DIR> classworlds
10/21/2010  06:16 PM <DIR> com
10/26/2010  11:39 AM <DIR> commons-beanutils
12/20/2009  11:49 PM <DIR> commons-cli
10/21/2010  12:05 PM <DIR> commons-collections
10/26/2010  11:39 AM <DIR> commons-digester
10/08/2010  02:00 PM <DIR> commons-fileupload
10/08/2010  02:00 PM <DIR> commons-io
10/21/2010  12:05 PM <DIR> commons-lang
10/26/2010  11:39 AM <DIR> commons-logging
10/08/2010  02:06 PM <DIR> commons-validator
10/21/2010  12:05 PM <DIR> dom4j
10/08/2010  02:06 PM <DIR> doxia
10/26/2010  03:05 PM <DIR> geronimo-spec
10/26/2010  03:05 PM <DIR> javax
10/08/2010  02:00 PM <DIR> jdom
10/25/2010  11:10 PM <DIR> jline
10/08/2010  07:00 PM <DIR> joda-time
10/21/2010  06:14 PM <DIR> jtidy
12/20/2009  11:48 PM <DIR> junit
10/08/2010  02:00 PM <DIR> log4j
10/21/2010  12:05 PM <DIR> net
10/26/2010  10:24 AM <DIR> org
10/08/2010  02:06 PM <DIR> oro
10/21/2010  06:14 PM <DIR> plexus
10/08/2010  02:00 PM <DIR> rome
10/26/2010  03:31 PM <DIR> servletapi
10/26/2010  11:39 AM <DIR> velocity
10/21/2010  12:05 PM <DIR> xml-apis
12/20/2009  11:50 PM <DIR> xpp3
               0 File(s)          0 bytes
               37 Dir(s) 21,234,976,768 bytes free

C:\Users\sang\.m2\repository>_
```

Maven Lifecycle & Phases

Build Lifecycle Basics

- Maven is based around the central concept of a build lifecycle
 - > What this means is that the process for building and distributing a particular artifact (project) is clearly defined.
 - > For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired
- There are three built-in build lifecycles:
 - > *default* - handles your project build/test/deployment
 - > *clean* – handles project cleaning
 - > *site* - handles the creation of your project's site documentation.

Phases of Build Lifecycle

- Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

“Default” Lifecycle's Build Phases

- *validate* - validate the project is correct and all necessary information is available
- *compile* - compile the source code of the project
- *test* - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- *package* - take the compiled code and package it in its distributable format, such as a JAR or a WAR file
- *integration-test* - process and deploy the package if necessary into an environment where integration tests can be run
- *verify* - run any checks to verify the package is valid and meets quality criteria
- *install* - install the package into the local repository, for use as a dependency in other projects locally
- *deploy* - done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Build Lifecycle Basics

- These build phases are executed sequentially to complete the lifecycle
- *mvn compile*
 - > All the phases up to compile phase will be executed in sequence
- *mvn test*
 - > All the phases up to test phase will be executed in sequence
- *mvn install*
 - > All the phases up to *install* phase will be executed in sequence
- *mvn integration-test*
 - > All the phases up to *integration-test* phase (validate, compile, package, etc.) will be executed in sequence

“Clean” Lifecycle's Phases

- *pre-clean* - executes processes needed prior to the actual project cleaning
- *clean* - remove all files generated by the previous build
- *post-clean* - executes processes needed to finalize the project cleaning

Phases and Goals

- Plugin goals can be attached to a lifecycle phase
- As Maven moves through the phases in a lifecycle, it will execute the goals attached to each particular phase
 - > Each phase may have zero or more goals (of various plugins) bound to it

Lab:

Exercise 1: Create a simple
Maven Project at the commandline

Step 4-6

[5072_tools_maven.zip](#)



Maven Repositories

What is a Repository?

- Maintains plugins and artifacts
 - > The plugins and artifacts are retrieved from the remote repository as needed basis
- Default remote repositories maintain public plugins and artifacts
 - > <http://repo1.maven.org/maven2>
 - > Called “Maven Central”
- Custom repositories can be set up to maintain non-public plugins and artifacts
 - > The default remote repositories can be replaced or augmented with references to custom repositories

Repository Structure

- Each artifact is maintained in a directory structure that matches a project coordinates
 - > /<groupId>/<artifactId>/<version>/<artifactId>-<version>.<packaging>
- Example
 - > “org.apache.commons:commons-email:1.1” (artifact)
 - > “/org/apache/commons/commons-email/1.1/commons-email-1.1.jar” (directory path)
 - > junit:junit:3.8.1 is available as /junit/junit/3.8.1/junit-3.8.1.jar
- Maven can easily locate the artifact in a repository (local and remote) based on artifact coordinates

Maven Central

The screenshot shows a Mozilla Firefox browser window with the title "Maven Central Search Engine - Mozilla Firefox". The address bar displays the URL <http://search.maven.org/#browse|-2021159614>. The page content is the "Maven Central Repository Browser" for the "junit" package. The search bar contains "junit" and there is a "SEARCH" button. Below the search bar, there are links for "Search Help" and "Advanced Search". A "Feedback" link is located on the right side of the page. The main table lists various versions of the junit dependency, all last modified on 07-Dec-2010. The table has columns for "Name" and "Last Modified". At the bottom of the page, there is a "Find:" input field containing "junit" and navigation buttons for "Next", "Previous", "Highlight all", and "Match case". The status bar at the bottom right shows the temperature as 68.0°F and some weather icons.

Name	Last Modified
3.7/	07-Dec-2010
3.8.1/	07-Dec-2010
3.8.2/	07-Dec-2010
3.8/	07-Dec-2010
4.0/	07-Dec-2010
4.1/	07-Dec-2010
4.2/	07-Dec-2010
4.3.1/	07-Dec-2010
4.3/	07-Dec-2010

Lab:

Maven Central Demo

<http://search.maven.org/#browse|47>



Plugins and Goals

Plugin Architecture

- Maven is based on Plugin architecture
 - > All Maven tasks are performed through plugins
- Maven core is basically a shell
 - > It parses a POM file and figures out which plugins are needed and then download them
- Plugins are downloaded, like dependencies are downloaded, from remote repositories as needed basis and updated periodically
 - > A plugin is a Maven project and has its own identity (coordinates)
 - > A downloaded plugin is then maintained in the local repository

What is Maven Plugin?

- A Maven plugin is a collection of one or more goals
- Examples of “ready-to-use” plugins
 - > Archetype plugin - contains goals for creating Maven projects
 - > Jar plugin - contains goals for creating JAR files
 - > Compiler plugin - contains goals for compiling source code and unit tests
 - > Hibernate3 plugin - contains goals for integration with the Hibernate library
 - > JRuby plugin - contains goals for executing JRuby as part of Maven build

Custom Plugin

- You can create a custom plugin
- A custom plugin can be written in many languages
 - > Java, Groovy, Ant, Ruby, etc

Benefits of Plugin Architecture

- Common plugin used by everyone to every project
 - > Everyone understands what the plugin does - no need to relearn
- Plugin can evolve/improve without breaking other parts of the build
- Change/improvement in a plugin (by community) benefit everyone

What is Maven Goal?

- Goal is a unit of task
 - > Same as “target” in Ant
- Example goals
 - > “generate” goal of the “archetype” plugin
 - > “compile” goal of the “compiler” plugin
 - > “test” goal of the “surefire” plugin

Plugins Can be Configured

- Plugins can be configured via configuration properties
- Example - Use JDK 1.6 for Compiler plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Example: Make Executable Jar Plug-in

```
<build>
  <plugins>
    <plugin>

      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <mainClass>com.javapassion.examples.App</mainClass>
          </manifest>
        </archive>
      </configuration>

    </plugin>
  </plugins>
</build>
```

Lab:

Exercise 2: Use a “Make Executable Jar”
Plugin
[5072_tools_maven.zip](#)



Dependency Management

How Dependency is Specified

- Each dependency is specified using the coordinates

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javapassion.examples</groupId>
  <artifactId>helloworld_app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld_app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Transitive Dependencies

- Usage scenario
 - > Your project depends on a library A
 - > Library A depends on 5 other libraries - B,C,D,E,F
- Your project need to specify dependency only on A
 - > Maven will handle the fact that A depends on B,C,D,E,F
- Maven also handles the conflict between dependencies
- You can see the dependency tree through Maven command
 - > *mvn dependency:tree*

mvn dependency:tree

```
Administrator: Command Prompt
[INFO] -----
[WARNING] The POM for joda-time:joda-time-jsptags:jar:1.0.2 is invalid, transitive dependencies may be missing or incomplete. See the documentation for more details
[INFO]
[INFO] --- maven-dependency-plugin:2.1:tree (default-cli) @ mvc_basics_Validator ---
[WARNING] Invalid POM for joda-time:joda-time-jsptags:jar:1.0.2, transitive dependency details omitted
[INFO] org.springframework.samples:mvc_basics_Validator:war:1.0.0-SNAPSHOT
[INFO] +- org.springframework:spring-context:jar:3.0.4.RELEASE:compile
[INFO] |   +- org.springframework:spring-aop:jar:3.0.4.RELEASE:compile
[INFO] |   |   \- aopalliance:aopalliance:jar:1.0:compile
[INFO] |   +- org.springframework:spring-beans:jar:3.0.4.RELEASE:compile
[INFO] |   +- org.springframework:spring-core:jar:3.0.4.RELEASE:compile
[INFO] |   +- org.springframework:spring-expression:jar:3.0.4.RELEASE:compile
[INFO] |   \- org.springframework:spring-asrn:jar:3.0.4.RELEASE:compile
[INFO] +- org.springframework:spring-webmvc:jar:3.0.4.RELEASE:compile
[INFO] |   +- org.springframework:spring-context-support:jar:3.0.4.RELEASE:compile
[INFO] |   \- org.springframework:spring-web:jar:3.0.4.RELEASE:compile
[INFO] +- org.slf4j:slf4j-api:jar:1.5.10:compile
[INFO] +- org.slf4j:jcl-over-slf4j:jar:1.5.10:runtime
[INFO] +- org.slf4j:slf4j-log4j12:jar:1.5.10:runtime
[INFO] +- log4j:log4j:jar:1.2.16:runtime
[INFO] +- javax.validation:validation-api:jar:1.0.0.GA:compile
[INFO] +- org.hibernate:hibernate-validator:jar:4.0.2.GA:compile
[INFO] |   +- javax.xml.bind:jaxb-api:jar:2.1:compile
[INFO] |   |   +- javax.xml.stream:stax-api:jar:1.0-2:compile
[INFO] |   |   \- javax.activation:activation:jar:1.1:compile
[INFO] |   \- com.sun.xml.bind:jaxb-impl:jar:2.1.3:compile
[INFO] +- joda-time:joda-time:jar:1.6:runtime
[INFO] +- joda-time:joda-time-jsptags:jar:1.0.2:runtime
[INFO] +- org.tuckey:urlrewritefilter:jar:3.1.0:compile
[INFO] +- javax.servlet:servlet-api:jar:2.5:provided
[INFO] +- javax.servlet.jsp:jsp-api:jar:2.1:provided
[INFO] +- javax.servlet:jstl:jar:1.2:compile
[INFO] \- junit:junit:jar:4.7:test
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.716s
[INFO] Finished at: Tue Mar 22 21:01:09 EDT 2011
[INFO] Final Memory: 4M/7M
[INFO] -----
C:\1passion.labs\handson_spring\spring3_mvc_form\samples\mvc_form_Validator>
```

Dependency Scope

- Each dependency is specified with a scope

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javapassion.examples</groupId>
  <artifactId>helloworld_app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld_app</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Dependency Scope Example

- When a dependency has a scope of “test”, it will not be available to the “compile” goal of the Compiler plugin
- It will be added to the classpath for only the “*compiler:testCompile*” and “*surefire:test*” goals

Dependency Scope (1)

- compile
 - > Default scope, used if none is specified.
- provided
 - > Much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime.
 - > For example, when building a web application, you would set the dependency on the Servlet API and related Java EE APIs to scope *provided* because the web container provides those classes.
 - > This scope is only available on the compilation and test classpath, and is not transitive.
- runtime
 - > The dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.

Dependency Scopes (2)

- test
 - > Dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.
- system
 - > Similar to *provided* except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
- import (only available in Maven 2.0.9 or later)
 - > Only used on a dependency of type pom in the <dependencyManagement> section

Packaging of Dependencies

- When you create a JAR for a project
 - > Dependencies are not bundled with the generated artifact - they are used only for compilation
- When you create a WAR/EAR file
 - > You can configure POM so that dependencies are bundled with the generated artifact
 - > You can also configure to exclude certain dependencies using “provided” scope - a dependency is needed for compilation but should not be bundled

Lab:

Exercise 3: Add Log4J Dependency
[5072_tools_maven.zip](#)



Maven Support in IDE's

Maven Support

- Each IDE uses its own proprietary project metadata
 - > Results in IDE lock-in
- Maven standardized project metadata
 - > Developer can use whatever IDE of his/her choice on any Maven projects
- All major IDE's (Eclipse, IntelliJ IDEA, etc) support Maven
 - > Create a Maven project, Import a Maven project
 - > Extras: form-based POM editor
- Tight integration with other IDE build tools
 - > Version management
 - > Task management

Lab:

Exercise 5: Import a Maven Project into Eclipse
[5072_tools_maven.zip](#)



Maven Web Project Structure

Maven Web Project Structure

- /src/main/java - source files for the dynamic content of the application
- /src/test/java - source files for unit tests
- /src/main/webapp - files for creating a valid web application, e.g. “web.xml“, view pages, etc
- /target - compiled and packaged deliverable
- pom.xml

Lab:

Exercise 7: Create a simple Maven
Web Project using Eclipse
[5072_tools_maven.zip](#)



Comparison to Ant

Example: Ant

```
<project name="my-project" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src/main/java"/>
  <property name="build" location="target/classes"/>
  <property name="dist" location="target"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}" />
  </target>

  <target name="compile" depends="init"
         description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}" />
  </target>

  <target name="dist" depends="compile"
         description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib" />

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}" />
  </target>

  <target name="clean"
         description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}" />
    <delete dir="${dist}" />
  </target>
</project>
```

Ant vs. Maven

- Ant
 - > It doesn't have formal conventions like a common project directory structure or default behavior. You have to tell Ant exactly where to find the source and where to put the output.
 - > It is procedural. You have to tell Ant exactly what to do and when to do it. You have to tell it to compile, then copy, then compress.
 - > It doesn't have a lifecycle. You have to define tasks and task dependencies.

Example: Maven

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sonatype.mavenbook</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

Ant vs. Maven

- Maven
 - > It has conventions. It knows where your source code is because you followed the convention. Maven's Compiler plugin puts the bytecode in target/classes, and it produces a JAR file in target.
 - > It is declarative. All you had to do is to create a pom.xml file and put your source in the default directory. Maven takes care of the rest.
 - > It has a lifecycle which gets invoked when you executes mvn install. This command told Maven to execute a series of sequential lifecycle phases until it reaches the install lifecycle phase. As a side-effect of this journey through the lifecycle, Maven executes a number of default plugin goals which did things like compile and create a JAR.

Code with Passion!

