# Maven Course

## To acquire:

Mature Dependency Management

Mature Project Build Lifecycle

Requirements -> Java 11, JDK

## Maven commands:

# mvn clean

# mvn install

# mvn package – create a jar file into target folder create a jar with our java classes

## Components of pom (Maven Coordinates + …)

Dependencies -> library used in our programs

ArtifactId – typically the project name

Version – refers to a specific version of project

Sample = 10.24.3-785-alpha

Major Version **10**.24.3-785-alpha

Minor 10.**24**.3-785-alpha

Incremental 10.24.**3**-785-alpha

Builder Number 10.24.3-**785**-alpha

Qualifier – string like beta, alpha 10.24.3-785-**alpha**

GroupId – typically unique to an organization

## Maven Repositories

Web: https://repo1.maven.org

Types:

Local

Central – public repository hosted by maven community

Remote – private or public -> like Google, Oracle, Android, BNP

## Maven Wagon

It’s a unified Maven API and transport abstraction for artifact and repository handling code

**If I need proxy** this topic will be very important to find the correct provider and do the right configuration

## Maven Pom

Pom -> Project Object Model

It’s a xml file with rules very well defined

## Maven Dependencies

Dependency is an artifact which your Maven project depends upon

Transitive Dependency - > dependency of artifact which your project depends on.

Dependency Mediation -> Determines what version to use multiple version of the same dependency are encountered Sample A-> B; A-> D 2.0 ; B-> D1,5; D 2.0 will be included

Excluded Dependencies

Optional Dependencies

## Dependency Scope

**Compile –** Default. Available on all class paths of project. Also, propagated to downstream projects.

**Provided –** Like Compile but expected to be provided by JDK or container at runtime.

**Runtime –** Not required for compile but needed for runtime. On runtime and test class paths, not compile.

**Test –** Only available on test class path, not transitive

**System –** like provided but JAR is added to system explicitly. (via file path)

**Import –** Imports dependency of POM

## Dependency Plugin

Dependencies are managed by Maven Dependency Plugin

Important Goals:

dependency:tree - shows the dependency tree. Useful for resolving conflicts

dependency:go-offline - Resolve all all, prepare to go offline

dependency:purge-local-repository - Clear artifacts from local repository

dependency:sources - get sources for all dependencies

## Standard Directory Layout Maven is important look at that in maven apache

## Maven Build Lifecycles

Maven is based on the concept of build lifecycles

A lifecycle is a pre-defined group of build steps called **phases**

Each phase can be bound to one or more plugin **goals**

Keep in mind all work done in Maven is done by plugins!

Lifecycles and phases provide the framework to call plugin goals in a sequence

Maven has three pre-defined lifecycles: **clean, default, site.**

**Clean**

3 Phases : pre-clean, clean and post-clean

Does a clean of the project, removes all build artifacts from working directory

Defined with plugin bindings

**Default**

Does the build and deployment of your project

Defined without plugin bindings, bindings are defined for each packaging

**Site**

Creates the a website for your project

Defined with plugin bindings

Least used in the enterprise

See any of the Maven websites for examples

All are built using the Maven site lifecycle.

## Default Lifecycle - High-level

Validate - Verify project is correct

Compile - Compile Source Code

Test - Test Compiled Source Code

Package - Package compiled files to packaging type

Verify - Run Integration Tests

Install - Install to local Maven Repository

Deploy - Deploy to shared Maven Repository

## Default Lifecycle All Phases

Validate

Initialize

Generate Sources

Process Sources

Generate Resources

Process Resources

Compile

Process Classes

Generate Test Sources

Process Test Sources

Generate Test Resources

Process Test Resources

Test Compile

Process Test Classes

Test

Prepare Package

Package

Pre-Integration Test

Integration Test

Post Integration Test

Verify

Install

Deploy

## Default Lifecycle - JAR Packaging

Phase: process-resources - Plugin: maven-resources-plugin : resources

Phase: compile - Plugin: maven-compiler-plugin : compile

Phase: process-test-resources - Plugin: maven-resources-plugin : testResources

Phase: test-compile - Plugin: maven-compiler-plugin : testCompile

Phase: test - Plugin: maven-surefire-plugin : test

Phase: package - Plugin: maven-jar-plugin : jar

Phase: install - Plugin: maven-install-plugin : install

Phase: deploy - Plugin: maven-deploy-plugin : deploy

## Site Build Lifecycle

**Site Lifecycle Phases**

Phase: Pre-site - Plugin: none

Phase: Site - Plugin: maven-site-plugin : site

Phase: Post-site - Plugin: none

Phase: Site-Deploy - Plugin: maven-site-plugin : deploy

**Conclusion**: we have many phases in each phase we can have a plugin to do what we want

## Maven Wrapper

[] optional to specify a version of maven to do wrapper

#mvn -N io.takari:maven:wrapper [-Dmaven=3.6.0]

#./mvnw –version

They are two files mvnw and mvnw.cmd and this functionality is very useful when we have different version and we want run with other

## Maven Archetypes

Maven Archetypes are project templates

Web: https://maven.apache.org/archetypes

Archetype - “An original pattern or model from with all other things of the same kind are made.”

Apache Maven provides a variety of standard archetypes to serve as starters for common Java projects

Maven Archetypes are also available from a variety of 3rd parties

Some of the archetypes are getting rather dated.

ie - J2EE

## Maven Clean Plugin

Build Lifecycle - CLEAN

Has only one goal - ‘clean’

Purpose is to remove files generated during build process.

By default removes /target directory project root and submodule root folders

## Maven Compiler Plugin

Build Lifecycle - DEFAULT

Has two goals - compiler:compile, compiler:testCompile

By Default uses the compiler ‘javax.tools.JavaCompiler

Can be configured to use javac if needed

Default source and target language levels are Java 1.6

Apache team encourages these values to be set

## Maven Resources Plugin

Build Lifecycle - DEFAULT

Web: https://maven.apache.org/plugins/maven-resources-plugin/examples/encoding.html

Has 3 goals - resources:resources, resources:testResources, resources:copy-resources

Purpose is to copy project resources to output directory (target dir)

Can be configured for encoding, source and target directories

Rather versatile configuration options for copying files during build processing

## Maven Surefire Plugin

Build Lifecycle - DEFAULT

Web: https://maven.apache.org/surefire/maven-surefire-plugin/

Has one goal: surefire:test

The Surefire plugin is used to execute unit test of the project.

By default supports JUnit 3, JUnit 4, JUnit 5, and TestNG

Cucumber runs under JUnit, Spock compiles to JUnit byte code.

By default includes classes named:

\*\*/Test\*.java; \*\*/\*Test.java; \*\*/\*Tests.java; \*\*/\*TestCase.java

## Maven jar plugin

Build Lifecycle - DEFAULT

Web: https://maven.apache.org/plugins/maven-jar-plugin/

Has two goals: jar:jar, jar:test-jar

Purpose is to build jars from complied artifacts and project resources

Can be configured for custom manifests, and to make executable jars.

## Maven Deploy Plugin

Build Lifecycle - DEFAULT

Web: https://maven.apache.org/plugins/maven-deploy-plugin/

Has two goals - deploy:deploy, deploy:deploy-file

Purpose is to deploy project artifacts to remote Maven repositories

Often done in CI

Configuration is typically part of the Maven POM

## Maven Site Plugin

Build Lifecycle - SITE

Web: https://maven.apache.org/plugins/maven-site-plugin/

Has 7 goals:

site:site - Generate site for project

site:deploy - Deploy site via Wagon

site:run - Run Site locally using Jetty as web server

site:stage - generate site to a local staging directory

site:stage-deploy - Deploy site to remote staging location

## Maven Site Plugin

Site Plugin Goals (continued):

site:attach-descriptor - adds site.xml (site map file used by search engines) to files for deployment

site:jar - bundles site into a jar for deployment to a repository

site:effective-site - generates the site.xml file

## Maven cheat sheet



## Testing with Maven

Unit Testing is completed by the Maven Surefire Plugin

Integration Testing is completed by the Maven Failsafe Plugin

Surefire / Failsafe support:

**POJO tests**

**JUnit versions 3 -5**

**TestNG**

## WebSites

* <https://kotlinlang.org/docs/reference/using-maven.html>
* <https://maven.apache.org/surefire/maven-surefire-plugin/examples/testng.html>
* <https://www.mojohaus.org/flatten-maven-plugin/usage.html>
* <https://maven.apache.org/enforcer/enforcer-rules/index.html>
* <https://maven.apache.org/guides/mini/guide-encryption.html>
* <https://docs.oracle.com/middleware/1213/core/MAVEN/config_maven_repo.htm#MAVEN9010>

## PackageCloud

* <https://packagecloud.io/users/new?plan=free_usage_plan>
* Use a maven to deploy to package cloud
* <https://github.com/computology/maven-packagecloud-wagon>

## Nexus Repository Manager Overview

Official name is Nexus Repository OSS (aka Nexus)

Open Source application written primarily in Java, by Sonatype

Used by 10 million developers around the world

Nexus is free to use, and commonly used by companies for internal Maven artifacts

Nexus Repository Pro is a commercially supported version of Nexus Repository OSS

Sonatype started as core contributors to Apache Maven and they host Maven Central

Maven central is the default repository used by Maven to locate open source artifacts

### Proxying Maven Repositories

Nexus has the ability to proxy other Maven Repositories.

Commonly used by companies to setup an internal repository which mirrors external repositories

Can be faster if corporate network is more performant than internet access

May be needed for compliance in some industries requiring auditing or approval of 3rd party libraries

May also be helpful if repository requires registration

<https://help.sonatype.com/repomanager3/system-requirements>

<https://help.sonatype.com/repomanager3/download>

<https://help.sonatype.com/repomanager3/installation/installation-methods>

<https://hub.docker.com/r/sonatype/nexus3/>

<https://help.sonatype.com/repomanager3/installation/accessing-the-user-interface>

Docker instructions

#docker run -d -p 8081:8081 –name nexus sonatype/nexus3

#docker exec -it <container name> /bin/bash

#cd nexus-data/admin.password

#change to admin123

Default user id is: admin

Default password is: admin123

### Create a snapshot repository nexus:

* Repositories->create Repository ->maven2(hosted)->Fill Name; version policy; layout policy(strict) -> save -> copy URL

### Nexus Repository Group

* Repositories->create Repository ->maven2(group)->Fill Name; version policy; layout policy(strict) -> select nexus release, snapshot and maven central-> save -> copy url
* Refactor to use a nexus group

## Maven Build Profiles

Maven Build Profiles allow you to specify a set of build configuration values

Values you can specify are largely the same data elements found in the POM

Values set in build profiles:

Can be in addition to what is in the project POM

OR used to override POM values

More than one profile can be set to active at time

**Caution**: No priority is available for multiple active profiles.

Duplicate Property resolution is random

### Declaring Build Profiles

Per Project

Defined in pom.xml

Command Line - mvn package -S <path to settings file>

Per User

Defined in <user home>/.m2/settings.xml

Global

Defined in <Maven Home>/conf/settings.xml

### Activating Build Profiles

In the Profile configuration under activation attribute:

setting activeByDefault property to true

default activation for OS, JDK versions

existence of system properties

specific values of system properties

missing files (ie build artifact not in target dir)

Command line: mvn package -P <profile-1>,<profile-2>

In settings.xml file - activeProfiles section

### Deactivating Profiles

Profiles can be deactivated from the command line:

mvn package -P !<profile-1>,!<profile-2>

mvn package -P -<profile-1>,-<profile-2>

### POM Elements in Profiles

repositories

pluginRespositories

dependencies

plugins

properties

modules

reporting

dependencyManagement

distributionManagement

Build Element: (Only)

defaultGoal

resources

testResources

finalName

### View Active Profiles

Command: mvn help:active-profiles

Note should be used with relevant CLI options

mvn help:active-profiles -P <profile-1>

would show profile-1 active

## Maven Release Plugin

<https://maven.apache.org/maven-release/maven-release-plugin/>

### Maven Release Plugin Goals

#### Prepare Release

Check no uncommitted changes

Check no snapshots

Update to release version

Run Tests

Tag in SCM

Next Snapshot version

Commit

#### Perform Release

Checkout of SCM with SCM tag

Run release goals

default deploy site deploy

Remove release files

Checkout of SCM master

#### Release Rollback

Typically used if errors have occurred in release:prepare goal

Will revert POMs back to pre-release state

SCM Tag may be removed

Tag removal is not fully implemented (depends on SCM)

SourceTree is a handy tool for GIT

Will not work if goal release:clean has been run

#### Clean Release

Cleans up release files

removes release.properties

removes any backup POMs

Normally removed with release:perform

#### Update POM Versions

Goal can be used to update POM versions in multi-model projects

Handy tool if default release process does not meet your needs

Not shown in course

command: mvn release:update-versions -DautoVersionSubmodules=true

### Maven SCM Plugin

<https://maven.apache.org/scm/maven-scm-plugin/>