Maven

Maven Basics

Lesson Objectives

- POM
- Standard Directory Structure
- Build Life Cycle
- Plug-in
- Dependency Management
- Resolving Dependency Conflicts
- Repositories

2.1 Project Object Model(POM)

- POM = Project Object Model = pom.xml
- Contains metadata about the Project
  - Location of directories, Developers/Contributors, Issue tracking system, Dependencies, Repositories to use, etc
- Example:

```
<project>
  <modelVersion>1.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <name>my maven app</name>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies/>
  <build/>
[…]
```

Minimal POM

2.1 Project Object Model(POM)

Declarative execution- POM model.
pom.xml

project-
Top level element
Of pom.xml

modelVersion-
Version of the object
Model this POM is using

groupId-
Unique identifier for the project.

artifactId-
Unique base name of the
Primary artifact being
Generated by the object.
Ex. My-app-1.0.jar

version-
Version of the generated
Artifact.
SNAPSHOT- project is
In a state of development

packaging-
Type of packaging to be
Used by this artifact.
Ex. JAR, WAR, or EAR

name-
Display name used for the
Project(Full name). Often used in
Generated Documentation,
during build
Process Of the project, or
projects that
use it As a dependency

url-
This element indicates
Where the project's site
Can be found.

description-
Basic description of the
Project.

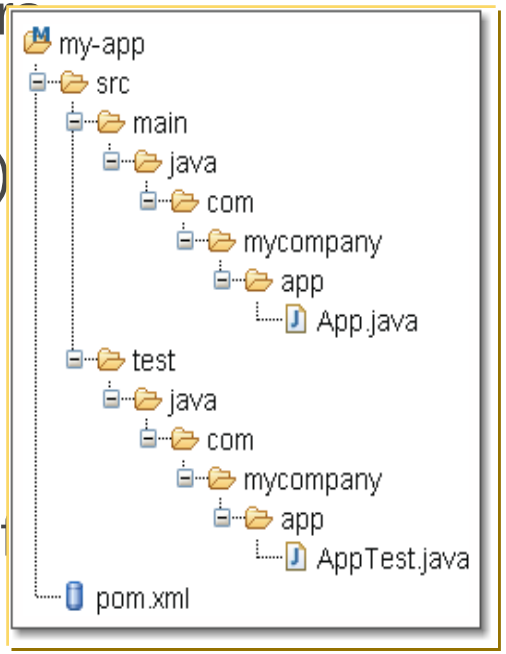
```
<project>
    <modelVersion>1.0.0</modelVersion>
     <groupId>com.example</groupId>
     <artifactId>jmy-app</artifactId>
     <packaging>jar</packaging>
     <version>1.0-SNAPSHOT</version>
     <name>my maven app</name>
     <url>http://maven.apache.org</url>
     <description></description>
    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
</project>
```

*This pom will allow you to compile, test and generate basic documentation.

2.2 Standard Directory Layout

- Having a common directory layout make the project easier to understand by other developers.
- It makes it easier to integrate plugins.
- Project home directory consists of POM(pom.xml) and two subdirectories initially:
  - src : contains all source code and
  - test: contains test source codes

- Target directory generated after the compilation of sources.



Directory structure before project execution

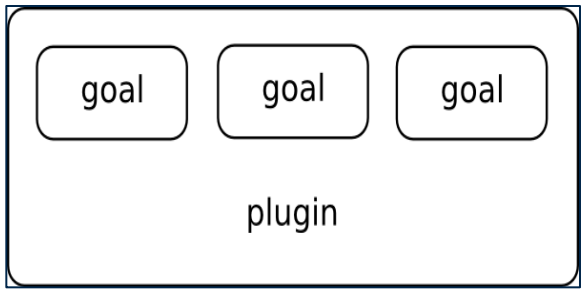
2.2 Standard Directory Layout

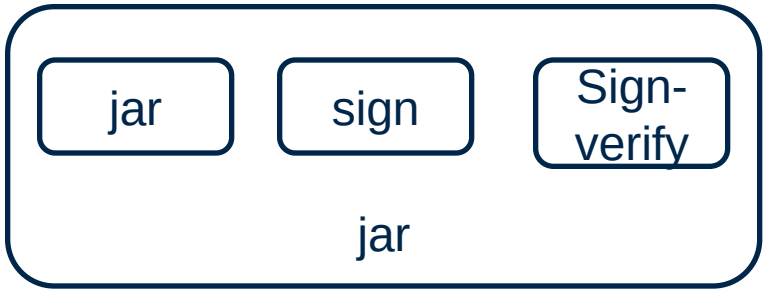
- Listing out few subdirectories in src directory

Directory name	Purpose
src/main/java	Contains the deliverable Java source code for the project.
src/main/resources	Contains the deliverable resources for the project, such as property files.
src/test/java	Contains the testing classes (JUnit or TestNG test cases, for example) for the project.
src/test/resources	Contains resources necessary for testing.
src/site	Contains files used to generate the Maven project website.

2.3 Plug-in

- Maven is built using a plugin-based architecture
- Each step in a lifecycle flow is called a phase. Zero or more plugin goals are bound to a phase.
- A plugin is a logical grouping and distribution (often a single JAR) of related goals, such as JARing.
- A goal, the most granular step in Maven, is a single executable task within a plugin.
- For example, discrete goals in the jar plugin include packaging the jar (jar:jar), signing the jar (jar:sign), and verifying the signature (jar:sign-verify).





2.3 Plug-in

- A plugin provides a set of goals that can be executed using the following syntax:
  - mvn [plugin-name]:[goal-name]
- Plugins reduces the repetitive tasks involved in the programming.
- Plugins are configured in a <plugins>-section of a pom.xml file as shown below

```
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.0</version>
<configuration>
<source>1.5</source>
<target>1.5</target>
</configuration>
</plugin>
</plugins>
```

2.3 Plug-in

Packaging

jar war ear ejb
rar pom shade

Integration

IDEA
eclipse

Core

clean
compiler
deploy
failsafe
install
resources
site
surefire
verifier

Reporting

Changelog
Changes
Checkstyle
Clover
Javadocs
PMD
Surefire -reports

Others

Cargo
jaxme
jetty
Tomcat

Tools

Ant
Antlr
Antrun
Archetype
Assembly
Help
Release

2.3 Plug-in

- Standard Plugin Configuration:
  - Build plugins will be executed during the build and they should be configured in the <build/> element from the POM.
  - All plugins should have minimal required informations: groupId, artifactId and version
- A mojo (build task) within a plug-in is executed when the Maven engine executes the corresponding phase on the build life cycle.

2.4 Build Life Cycle

- In Maven, process for building and distributing artifact is clearly defined in the form of life cycle.
- Each lifecycle contains phases in a specific order, and zero or more goals are attached to each phase.
- For example, the compile phase invokes a certain set of goals to compile a set of classes.
- Similarly phases are available for testing, installing artifacts,..
- There are three standard lifecycles in Maven
  - Clean
  - default (sometimes called build)
    - Handle project deployment
  - site

2.4 Build Life Cycle

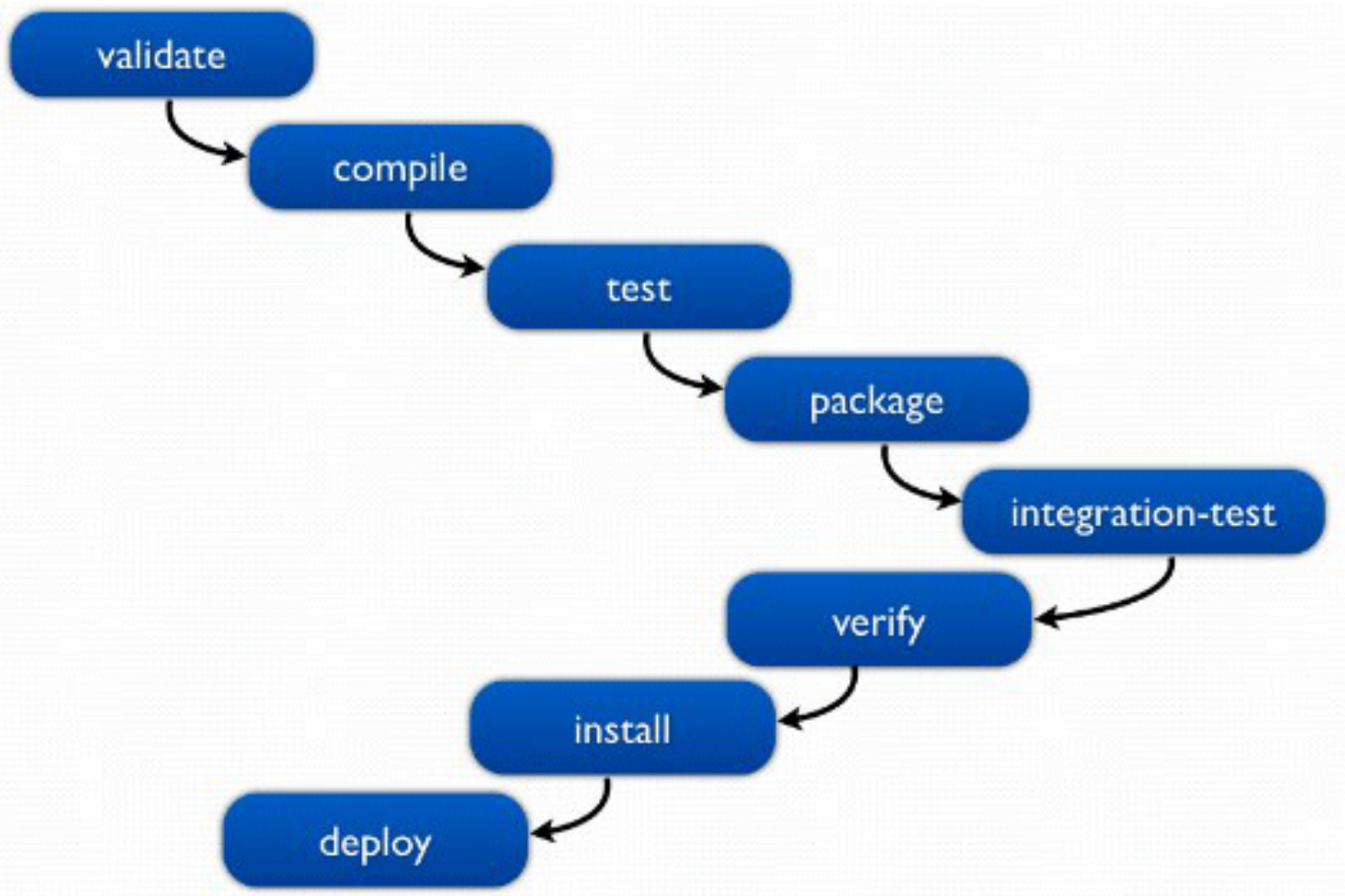
- clean lifecycle handles the cleaning of all project files generated by a previous build.
- Running mvn clean invokes the clean lifecycle

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**

2.4 Build Life Cycle

- The default lifecycle handles your project deployment.
- Some Key Phases in default life cycle are:
  - validate
  - compile
  - Test
  - Package
  - integration-test
  - Install
  - deploy

2.4 Build Life Cycle



2.4 Build Life Cycle

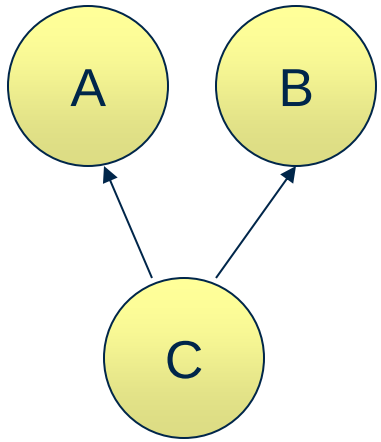
- Site lifecycle handles the creation of your project's site documentation.
- You can generate a site from a Maven project by running the following command:
-

pre-site	executes processes needed prior to the actual project site generation
site	generates the project's site documentation
post-site	executes processes needed to finalize the site generation, and to prepare for site deployment
site-deploy	deploys the generated site documentation to the specified web server

2.5 Dependency Management

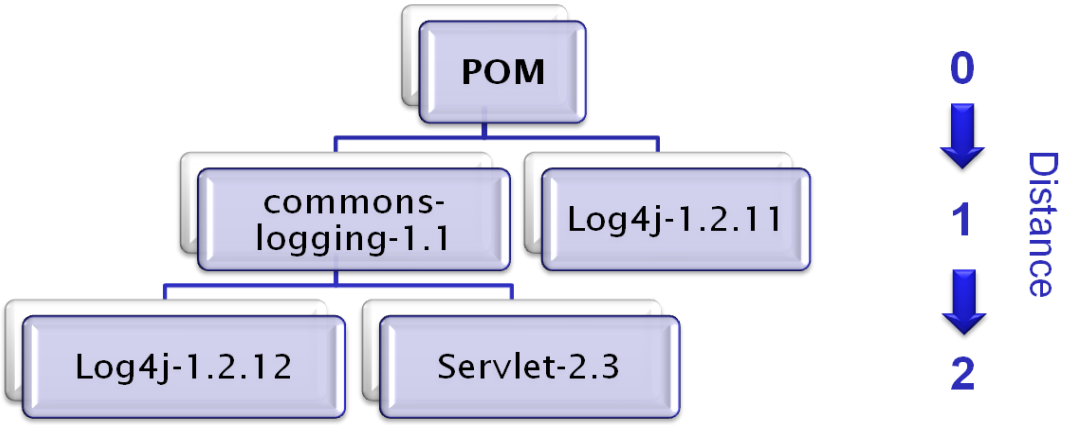
- The dependency management is a mechanism for centralizing dependency information.
- In Maven, Dependencies are defined in the POM.

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



2.6 Resolving Dependency Conflicts

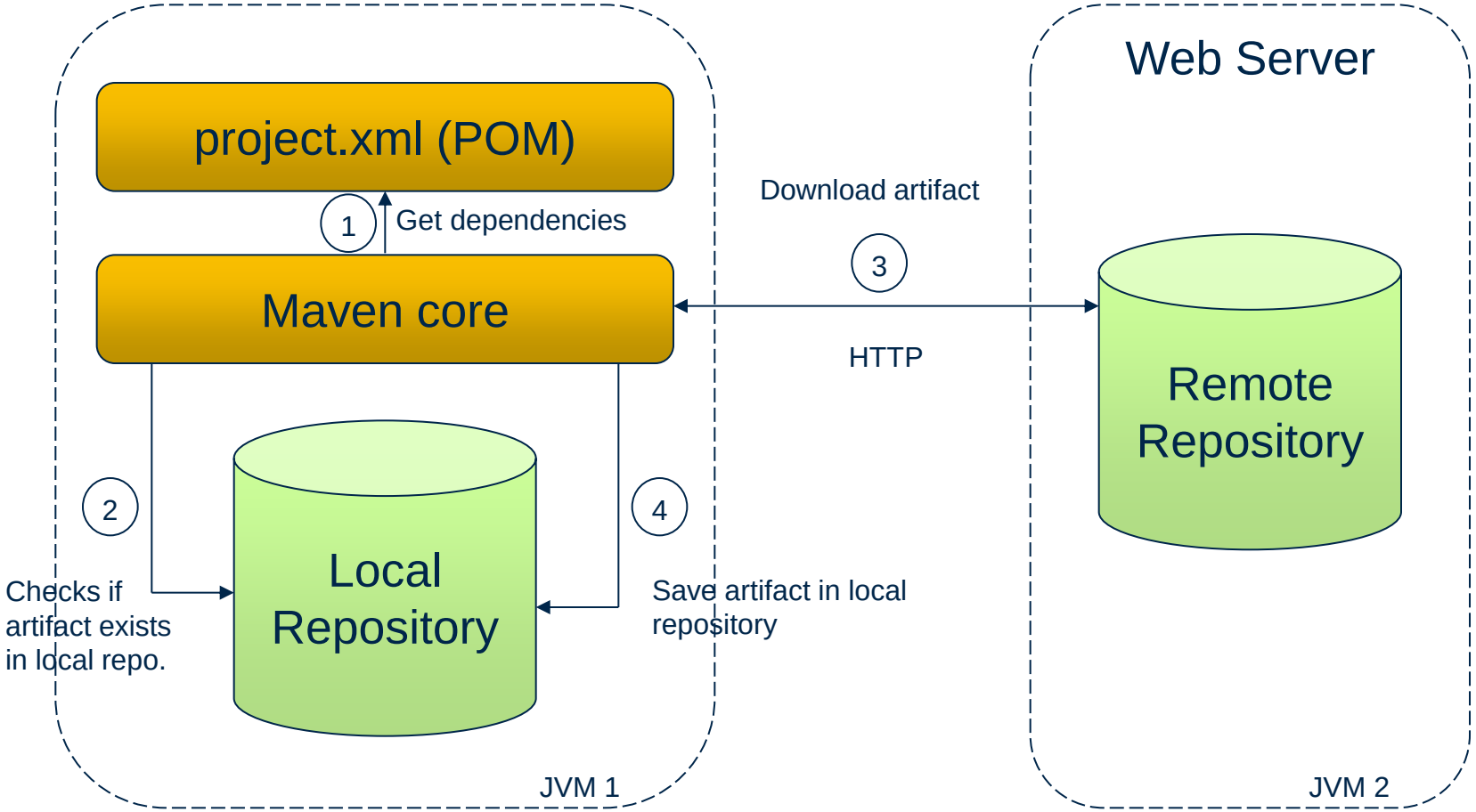
- Conflicts arise in Maven when the same dependency (Ex. Log4j) of different version is identified in dependency graph.
- While resolving such conflicts Maven traverses the dependency in a top down manner and selects the version “nearest” to the top of the tree.
- For an Example, looking for Log4j-1.2.12 dependency in a dependency graph as shown below.
- In this image Log4j-1.2.11 is selected as it is closer to the root of the tree.



2.7 Repositories

- Repositories store a collection of artifacts used by Maven during dependency resolution for a project.
- An artifact is a resource generated by maven project usually bundled as a JAR, WAR, EAR, or other code-bundling type.
- For an example, junit.jar is an artifact.
- An artifact in repositories can be uniquely identified using coordinates:
  - The group ID
  - The artifact ID
  - The version
- Maven has two types of repositories:
  - Local
  - Remote

2.7 Repositories



Summary

- POM
- Standard Directory Structure
- Build Life Cycle
- Plug-in
- Dependency Management
- Resolving Dependency Conflicts
- Repositories

Review Question

- Question1: Clean Life cycle phases are _____, _____, _____
- Question 2: Which command generates default Site for a Maven Project?
- Question 3: Plugin used for running JUnit tests _____.
- Question 4: For identifying artifact in repositories, coordinates required are _____, _____ and _____.

Review Question

- Question 5: Invoking the deploy phase deploys the application in which environment?
  - Option 1: Local repository
  - Option 2: Release environment
  - Option 3: External Repository