Advanced Java Application Development Using JavaEE

University of Colombo School of Computing

Maven 2 – setting.xml

This is main settings file in your Maven. But this is not in your .m2 folder. Create setting.xml file in m2 directory.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
          http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository/>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```

Maven 2 – setting.xml

LocalRepository – Maven stores copies of plugins and dependencies locally C:\Users\<<your_user_name>>\.m2\repository folder. This element can be used to change the path to the local repository. For example <localRepository>C:\manvenrepo</localRepository>
Here changed to repository location to mavenrepo in C drive.

InteractiveMode – When this value is set to true, the default value, Maven interacts with the user for inputs

offline – When set to true, this configuration instructs Maven to operate in an offline mode. The default value is false.

servers – Maven can interact with a variety of servers, such as Apache Subversion (SVN) servers, build servers, and remote repository servers. This element allows you to specify security credentials, such as the username and password, which you need to connect to these servers.

Maven 2 – setting.xml

mirrors – As the name suggests, mirrors allow you to specify alternate locations for your repository.

proxies – proxies contains the HTTP proxy information needed to connect to the Internet.

profiles – profiles allows you to group certain configuration elements, such as repositories and pluginRepositories.

activeProfile – The activeProfile allows you to specify a default profile to be active for Maven to use.

Maven 2 – setting.xml

Maven requires an Internet connection to download plugins and dependencies. Some companies employ HTTP proxies to restrict access to the Internet. In these scenarios, running Maven will result Unable to download artifacts errors. To address this, edit the settings.xml file and add the proxy information specific to your company.

```
<proxies>
        <proxy>
                <id>companyProxy</id>
                <active>true</active>
                <protocol>http</protocol>
                <host>proxy.company.com</host>
                        <port>8080</port>
                        <username>proxyusername</username>
                        <password>proxypassword</password>
                        <nonProxyHosts />
        </proxy>
</proxies>
```

Maven 2 – Continue…..

- Enterprise level projects typically depend on a variety of open source libraries.

  As example of your requirement to use log4J for your application logging. To accomplish your task you need to go log4J download page and download the JAR file, and put it in your lib folder or class path. ***Then you have to consider couple of problems***

- You need to check JAR files into SVN so that your project can be built on a computer other than your own.
- The JAR file you download might depends on a few other libraries. You have to then hunt down all of those dependencies and add them to your project.

Maven 2 – Continue…..

- When time comes to upgrade the JAR files you need to start the process all over again.

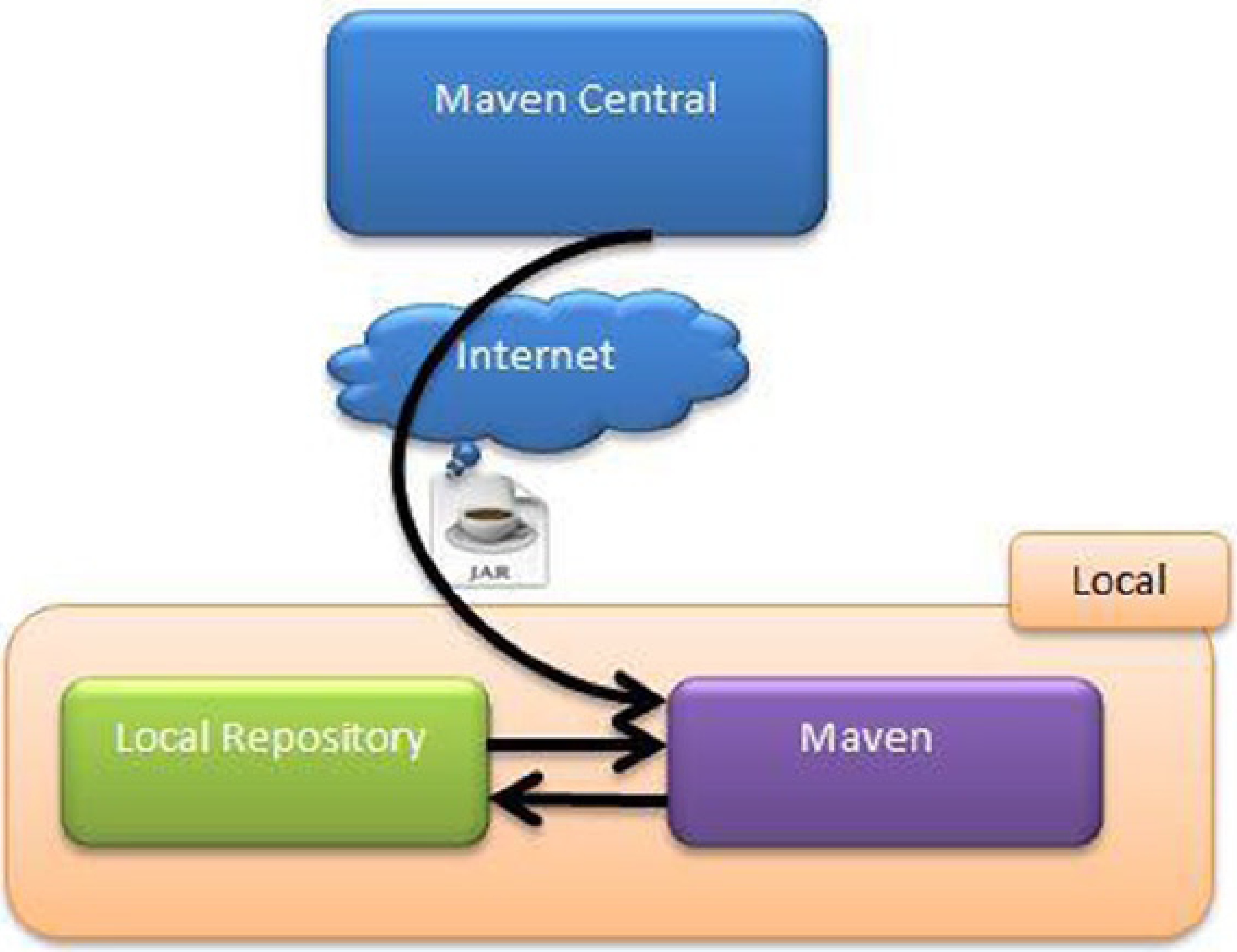
- It becomes difficult to share JAR files across teams within your organization

  To address these problems, Maven provides declarative dependency management. With this approach, you declare your projects dependencies in an external file called ***pom.xml***.

  Maven will automatically download those dependencies and had them over to your project for the purpose of building, testing, or packaging.
- Default remote repository with which Maven interacts is called Maven Central, and it is located at repo.maven.apache.org and uk.maven.org. Maven places the downloaded artifacts in the local repository

UCSC

Maven 2 – Continue…..



Resources – Introducing Maven Balaji Varanasi and Sudha Belida

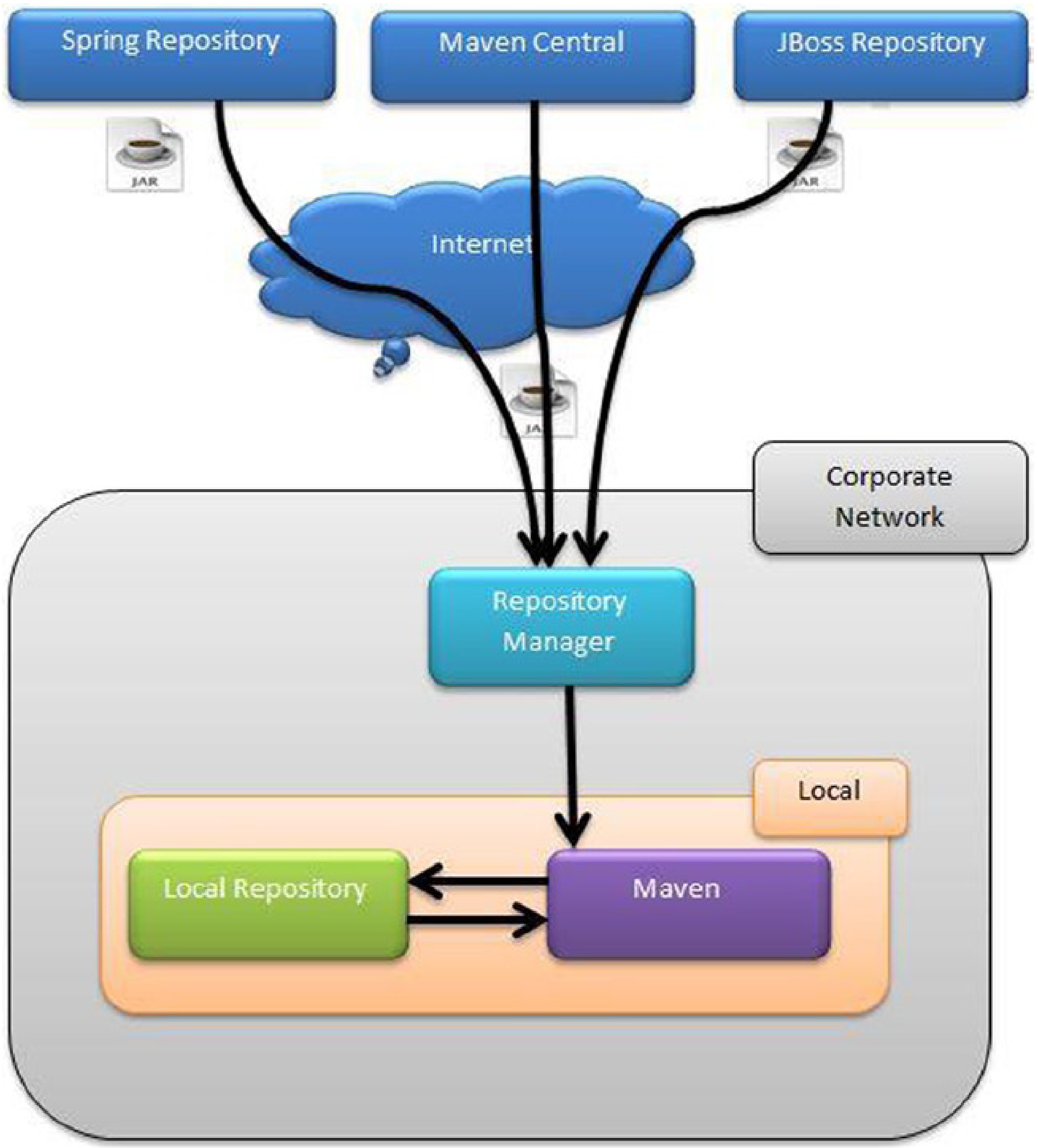
Maven 2 – Continue…..

Using this architecture possess its own drawbacks.

- First one is that sharing company related artifacts between teams is not possible. Because of security and intellectual properties concerns. You wouldn't want to publish your enterprise's artifacts on Maven Central.
- Another one is legal and licensing issues. Your company might want the terms only to use officially approved open source softwares, and this architecture would not fit in that model.
- Final issue is bandwidth and download speeds. In times of heavy load on Maven Central, the download speeds of Maven artifacts are reduced, and this might have a negative impact on your build.

Alternative solution is Enterprise Maven repository architecture.

Maven 2 – Continue…..



Resources – Introducing Maven Balaji Varanasi and Sudha Belida

Maven 2 – Continue…..

In this Enterprise model your company can alleviate above issues. In here it has introduced internal repository manager. This internal repository manager acts as a proxy to remote repositories. Because you have full control over the internal repository, you can regulate the types of artifacts allowed in your company.

Additionally, you can also push your organizations artifacts in the server, thereby enabling collaboration. To do this there are several open source repository managers.

Sonatype Nexus :- www.sonatype.com/nexus
Apache Archiva :- http://archiva.apache.org/
Artifactory :- www.jfrog.com/open_source/

Maven 2 – Continue…..

Information regarding repositories can be provided in the settings.xml or the pom.xml file. There are pros and cons to each approach. Putting repository information in the pom.xml file can make your builds portable. It enables developers to download projects and simply build them without any further modifications to their local settings.xml file. The problem with this approach is that when artifacts are released, the corresponding pom.xml files will have the repository information hard coded in them. If the repository URLs were ever to change, consumers of these artifacts will run into errors due to broken repository paths. Putting repository information in the settings.xml file addresses this problem, and because of the flexibility it provides, the settings.xml approach is typically recommended in an enterprise setting.

Maven 2 – Continue…..

In order to add repositories – add them into setting.xml

```
<profiles>
      <profile>
             <id>your_company</id>
                 <repositories>
                      <repository>
                           <id>spring_repo</id>
                           <url>http://repo.spring.io/release/</url>
                      </repository>
                      <repository>
                           <id>jboss_repo</id>
                           <url>https://repository.jboss.org/</url>
                      </repository>
                 </repositories>
       </profile>
</profiles>
<activeProfiles>
         <activeProfile>your_company</activeProfile>
</activeProfiles>
```

Maven 2 – Continue…..

Archives of Maven repositories JAR, WAR, EAR and ZIP. Each Maven dependency is uniquely identified using the following groups, artifact and version (GAV) coordinates.

groupId – Identifier of the organization or group that is responible for this project. Ex. Include org.hibernate.log4j and org.springframework.boot.

artifactId – Identifier or the artifact being generated by the project. This must be unique among the projects using same groupId. Ex. Includes hibernate-tools, log4j, spring-core, and so on.

version – Includes the version number of the project. Ex. Include 1.0.0, 2.3.1-SNAPSHOT, and 4.3.6.Final.

Type – indicates the packing of the generated artifact. Ex. Includes JAR, WAR and EAR.

Maven 2 – Dependencies

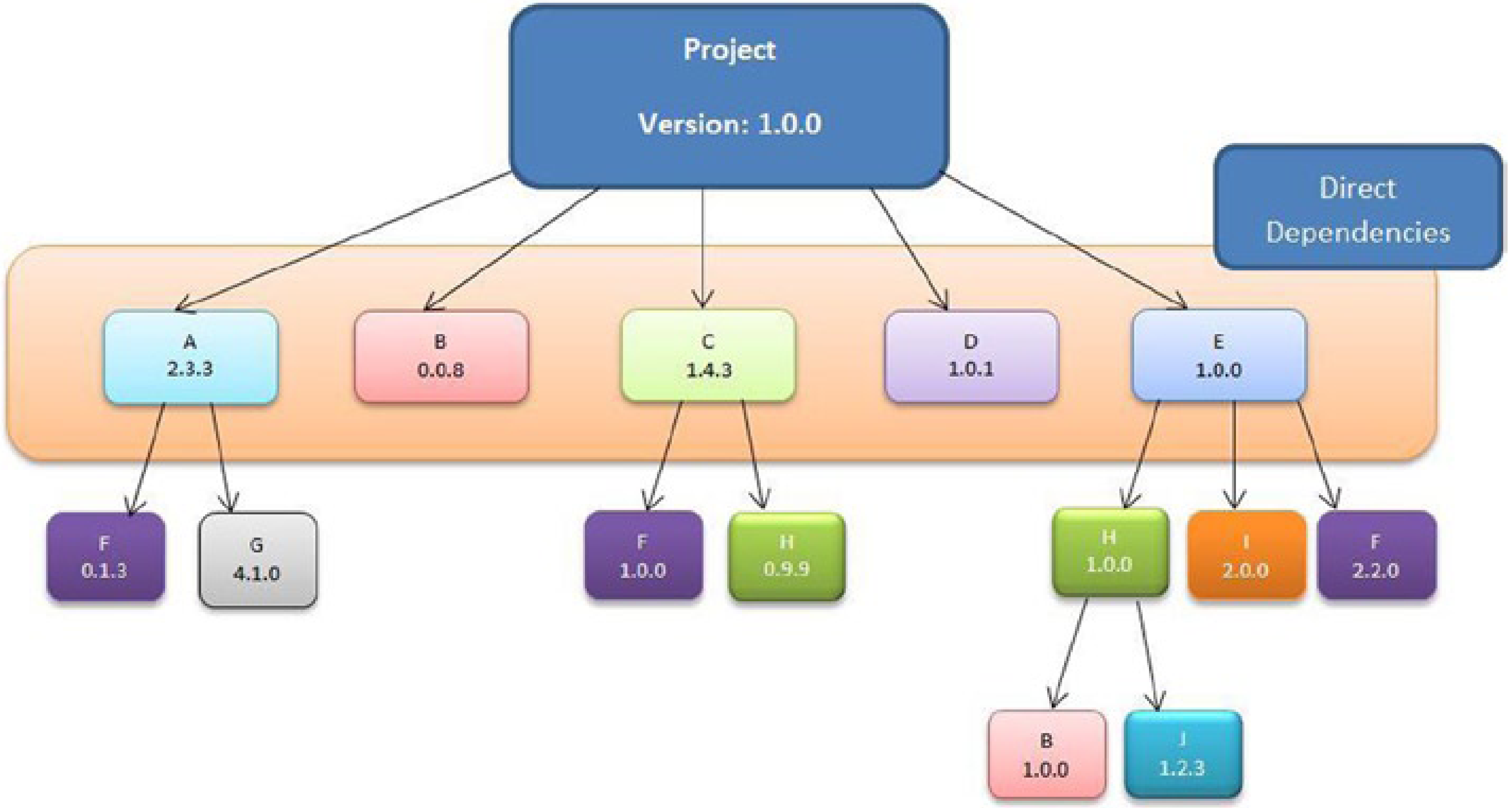
Transitive Dependency

Dependencies declared in your pom.xml file often have their own dependencies. Such dependencies are called transitive dependencies.

Ex. If you need to use Hibernate Core, function it properly it needs Jboss Logging, dom4j, and javaassist and so forth.

The hibernate core declared in your pom.xml file is considered direct dependency and dependencies such as dom4j, and javaassist are considered your project's transitive dependencies.

Maven 2 – Dependencies



Resources – Introducing Maven Balaji Varanasi and Sudha Belida

Maven 2 – Dependencies

Maven uses a technique known as dependency mediation to resolve version conflicts.
Simply stated, dependency mediation allows Maven to pull the dependency that is closest to the project in the tree. There are two versions of dependency B: 0.0.8 and 1.0.0. In this scenario, version 0.0.8 of dependency B is included in the project, because it is a direct dependency and closest to the tree. Now look at the three versions of dependency F: 0.1.3, 1.0.0, and 2.2.0. All three dependencies are at the same depth. In this scenario, Maven will use the first-found dependency, which would be 0.1.3, and not the latest 2.2.0 version.

Resources – Introducing Maven Balaji Varanasi and Sudha Belida

Maven 2 – Scope

Maven is provided you with scope to integrate dependencies when you need it. To test you need Junit JAR. It should be in your project during testing. In this scenario you not need to bundle the Junit in your final production archive Similarly consider MySQL database driver. You need dependency when you are running the application inside a container such as Tomcat. Concept of Scope facilitate to specify dependencies when and where you need a particular dependency. Maven provided six types of scope in this case.

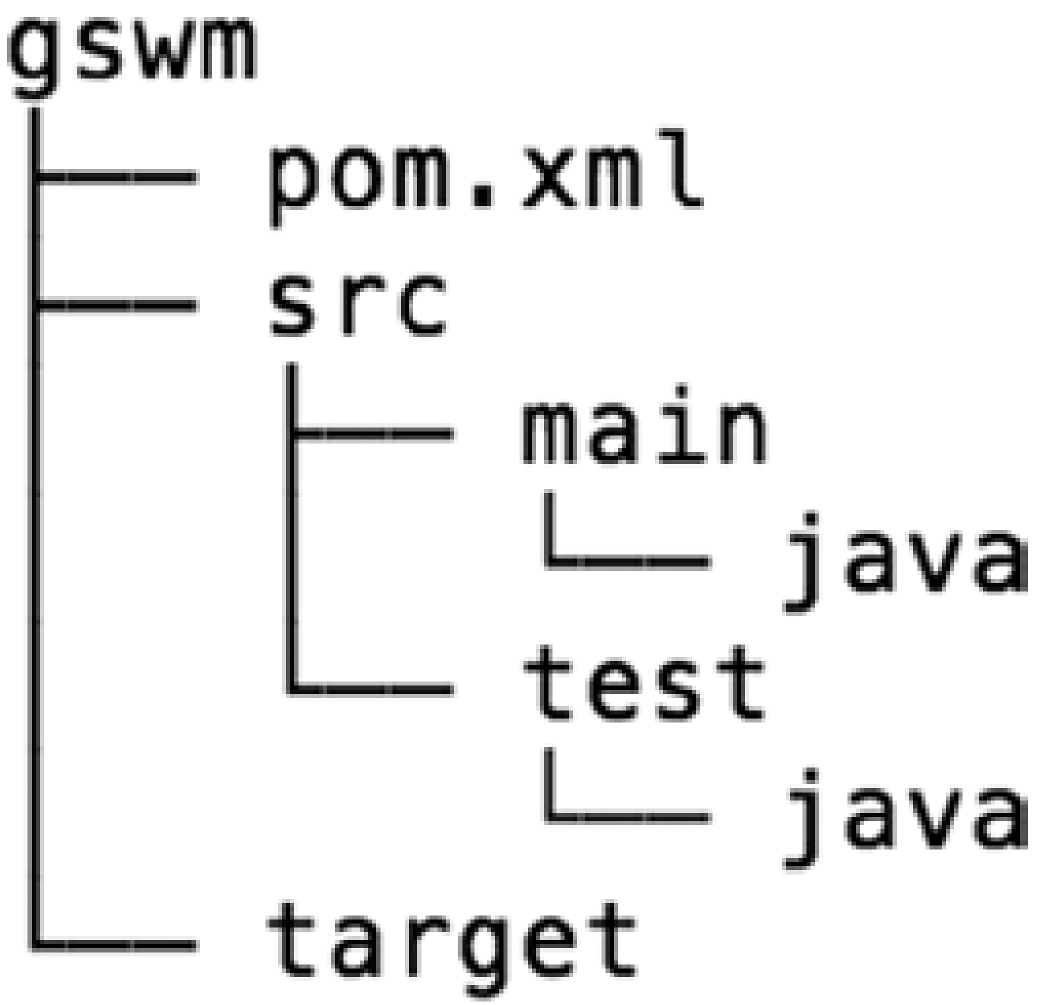
Maven 2 – Scope

- **compile:** Dependencies with the compile scope are available in the class path in all phases on a project build, test, and run. This is the default scope.
- **provided:** Dependencies with the provided scope are available in the class path during the build and test phases. They don’t get bundled within the generated artifact. Examples of dependencies that use this scope include Servlet api, JSP api, and so on.
- **runtime:** Dependencies with the runtime scope are not available in the class path during the build phase. Instead they get bundled in the generated artifact and are available during runtime.

Maven 2 – Scope

- **test:** Dependencies with the test scope are available during the test phase. JUnit and TestNG are good examples of dependencies with the test scope.
- **system:** Dependencies with the system scope are similar to dependencies with the provided scope, except that these dependencies are not retrieved from the repository. Instead, a hard-coded path to the file system is specified from which the dependencies are used.
- **import:** The import scope is applicable for .pom file dependencies only. It allows you to include dependency management information from a remote .pom file. The import scope is available only in Maven 2.0.9 or later.

Maven 2 – Basic Project Organization



```
gswm
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   └── test
│       └── java
└── target
```

Resources – Introducing Maven Balaji Varanasi and Sudha Belida

Maven 2 – Basic Project Organization

Other folders added in Maven project according to Source Control Management Systems (SCM) like SVN or Git.

- The src/main/java folder contains the Java source code.
- The src/test/java folder contains the Java unit test code.
- The target folder holds generated artifacts, such as .class files. Generated artifacts are typically not stored in SCM, so you don’t commit the target folder and its contents in to SCM.
- Every Maven project has a pom.xml file at the root of the project. It holds project and configuration information, such as dependencies and plug-ins.

Resources – Introducing Maven Balaji Varanasi and Sudha Belida

Maven 2 – Basic Project Directories

- ***src/main/resources*** - Holds resources, such as Spring configuration files and velocity templates, that need to end up in the generated artifact.
- ***src/main/config* -** Holds configuration files, such as Tomcat context files, James Server configuration files, and so on. These files will not end up in the generated artifact.
- ***src/main/scripts* -** Holds any scripts that system administrators and developers need for the application.
- ***src/test/resources* -** Holds configuration files needed for testing.
- ***src/main/webapp* -** Holds web assets such as .jsp files, style sheets, and images.
- ***src/it* -** Holds integration tests for the application.
- ***src/main/db* -** Holds database files, such as SQL scripts.
- ***src/site* -** Holds files required during the generation of the project site.

Maven 2 – Life Cycle and Phases

Maven follows a well-defined build life cycle when it builds, tests, and distributes an artifact. The life cycle constitutes a series of stages or steps that get executed in the same order, independent of the artifact being produced. Maven refers to the steps in a life cycle as phases. Maven has the following three built-in life cycles:

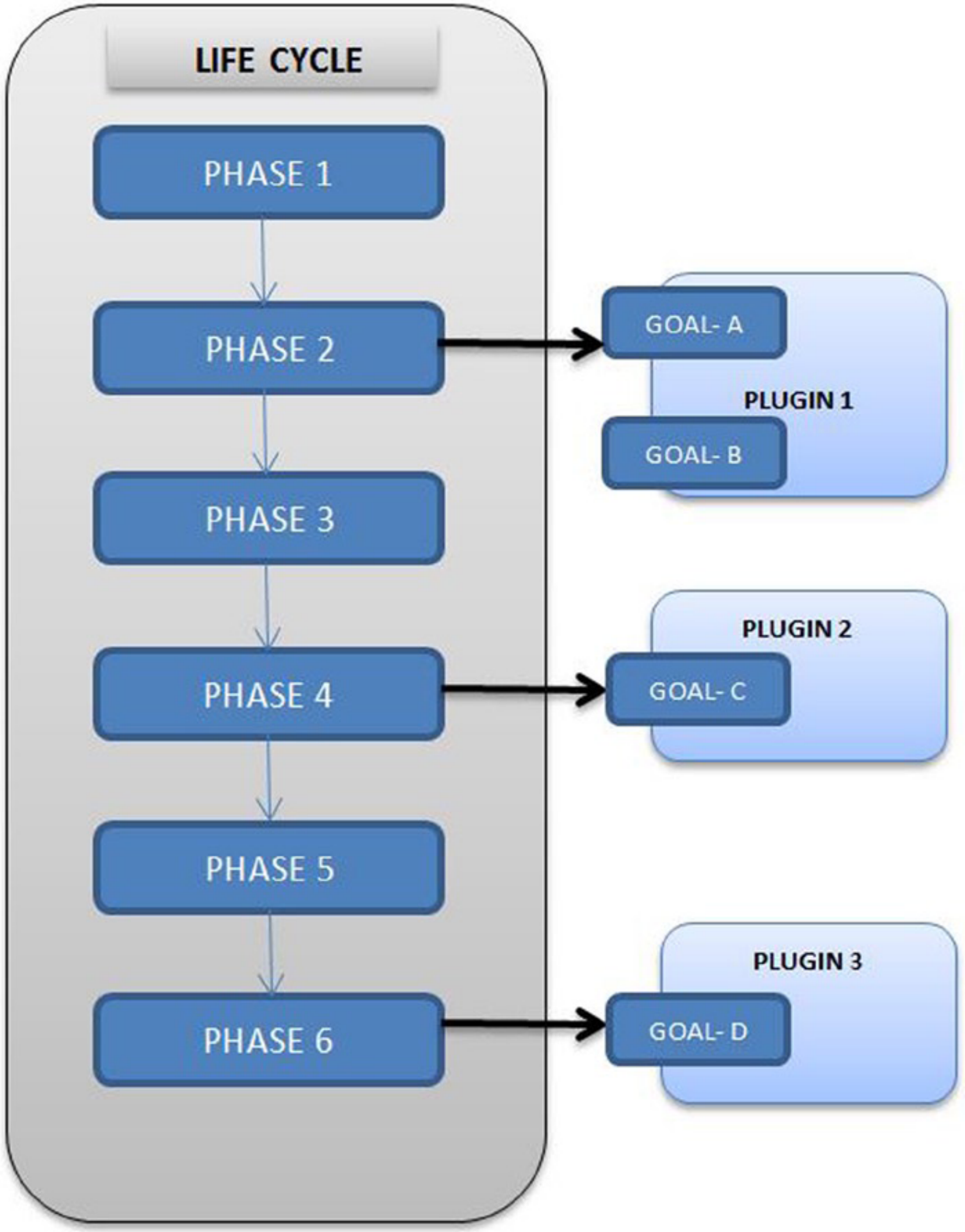
- **Default:** This life cycle handles the compiling, packaging, and deployment of a Maven project.
- **Clean:** This life cycle handles the deletion of temporary files and generated artifacts from the target directory.
- **Site:** This life cycle handles the generation of documentation and site generation.

Maven 2 – Life Cycle and Phases

To better understand the build life cycle and its phases, let’s look at the some of the phases associated with the default life cycle:

- **Validate:** Runs checks to ensure that the project is correct and that all dependencies are downloaded and available.
- **Compile:** Compiles the source code.
- **Test:** Runs unit tests using frameworks. This step doesn’t require that the application be packaged.
- **Package:** Assembles compiled code into a distributable format, such as JAR or WAR.
- **Install:** Installs the packaged archive into a local repository. The archive is now available for use by any project running on that machine.
- **Deploy:** Pushes the built archive into a remote repository for use by other teams and team members.

Maven 2 – Life Cycle and Phases



Resources – Introducing Maven Balaji Varanasi and Sudha Belida

Maven 2 – Life Cycle and Phases