Maven

Shristi Technology Labs

Contents

- What is a build tool?
- Introduction to Maven
- Pom.xml
- Directory Structure of the Maven project
- Dependencies
- Repositories
- Build Lifecycle, phases, goal
- Build profiles

What is a build tool?

- A build tool automates everything related to building the project.
- It includes
  - Adding the dependencies
  - Compiling the source code
  - Packaging the compiled code into JAR or ZIP files
  - Generating documentation from the source code
- eg. Maven , ANT

Introduction to Maven

- Is a project management tool to manage project build, reporting and documentation.
- Simplifies, standardizes & automates the project build process
- Provides developers a complete build lifecycle framework.
- Uses a standard directory layout.
- Handles compilation, distribution, documentation easily(reusability)

Maven

Maven helps developers to manage

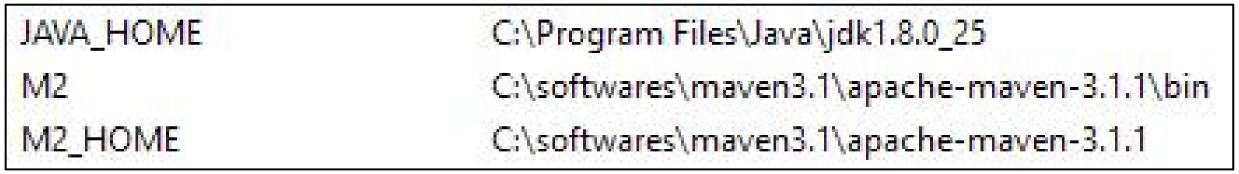
- Compilation
- Documentation
- Reporting
- Dependencies
- Distribution

- It is a tool which is reusable and maintainable.
- **Project Object Model (POM- pom.xml),** is the fundamental unit of the entire Maven system.
- **pom.xml** declares the mavens project structure

Environment setup

- Download Maven 3.1.1 archive from https://maven.apache.org/download.cgi
- Extract Maven archive
- Set up the environment for maven in Environment Variables

Variable Name	Variable Value
M2_HOME	<%maven-installation path%>
M2	% M2_HOME%/bin
JAVA_HOME	<%Java installation path%>
path	%M2_HOME%/bin

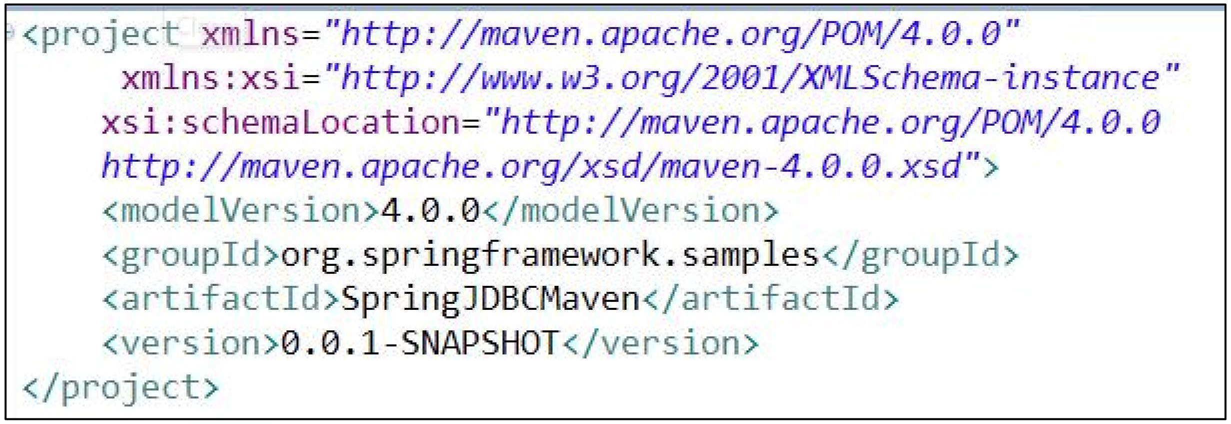


JAVA_HOME	C:\Program Files\Java\jdk1.8.0_25
M2	C:\softwares\maven3.1\apache-maven-3.1.1\bin
M2_HOME	C:\softwares\maven3.1\apache-maven-3.1.1

Project Object Model(POM)

- Is an xml file. It has to be in base directory
- has information about the project and various configuration details used by Maven to build the project(s).
- Maven reads the POM, gets the needed configuration information, then executes the goal.

pom.xml



```
<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/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.springframework.samples</groupId>
    <artifactId>SpringJDBCMaven</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</project>
```

groupId: Name of the company, organization, team etc., usually using the reverse URL naming convention

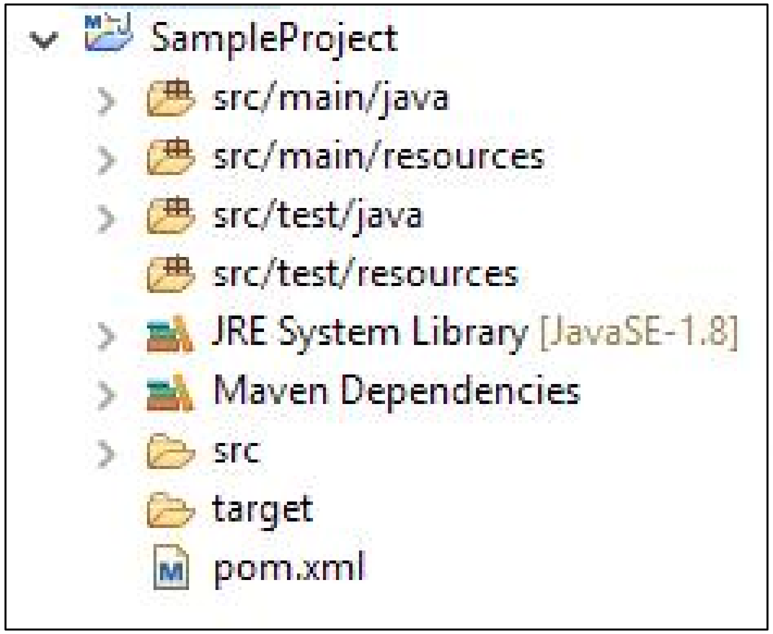
artifactId: A unique name for the project under groupId

version: The version of the project

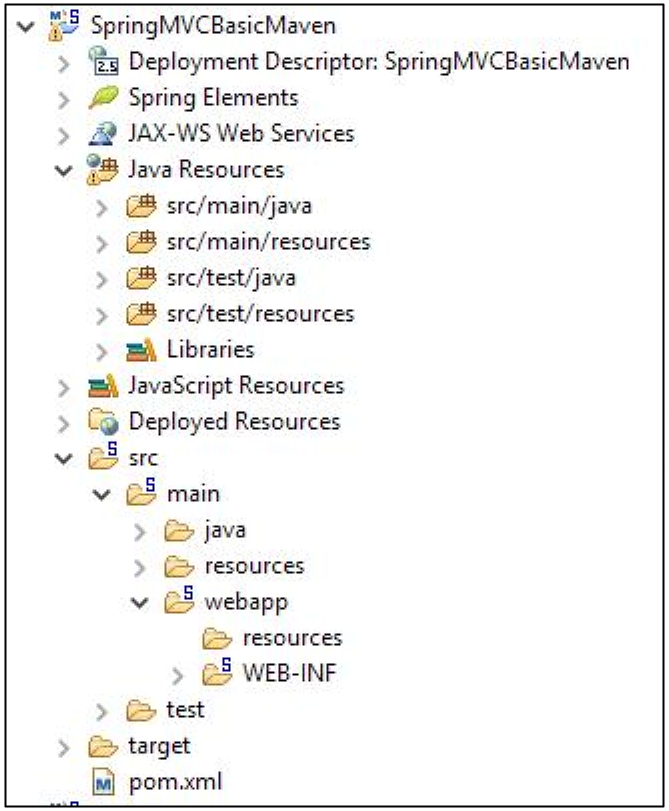
modelVersion: This sets the version of the POM model that is used. The POM version should match the Maven version. POM Version 4.0.0 matches Maven version 2 and 3

Directory Structure

- The directory structure of Maven project



```
SampleProject
  src/main/java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

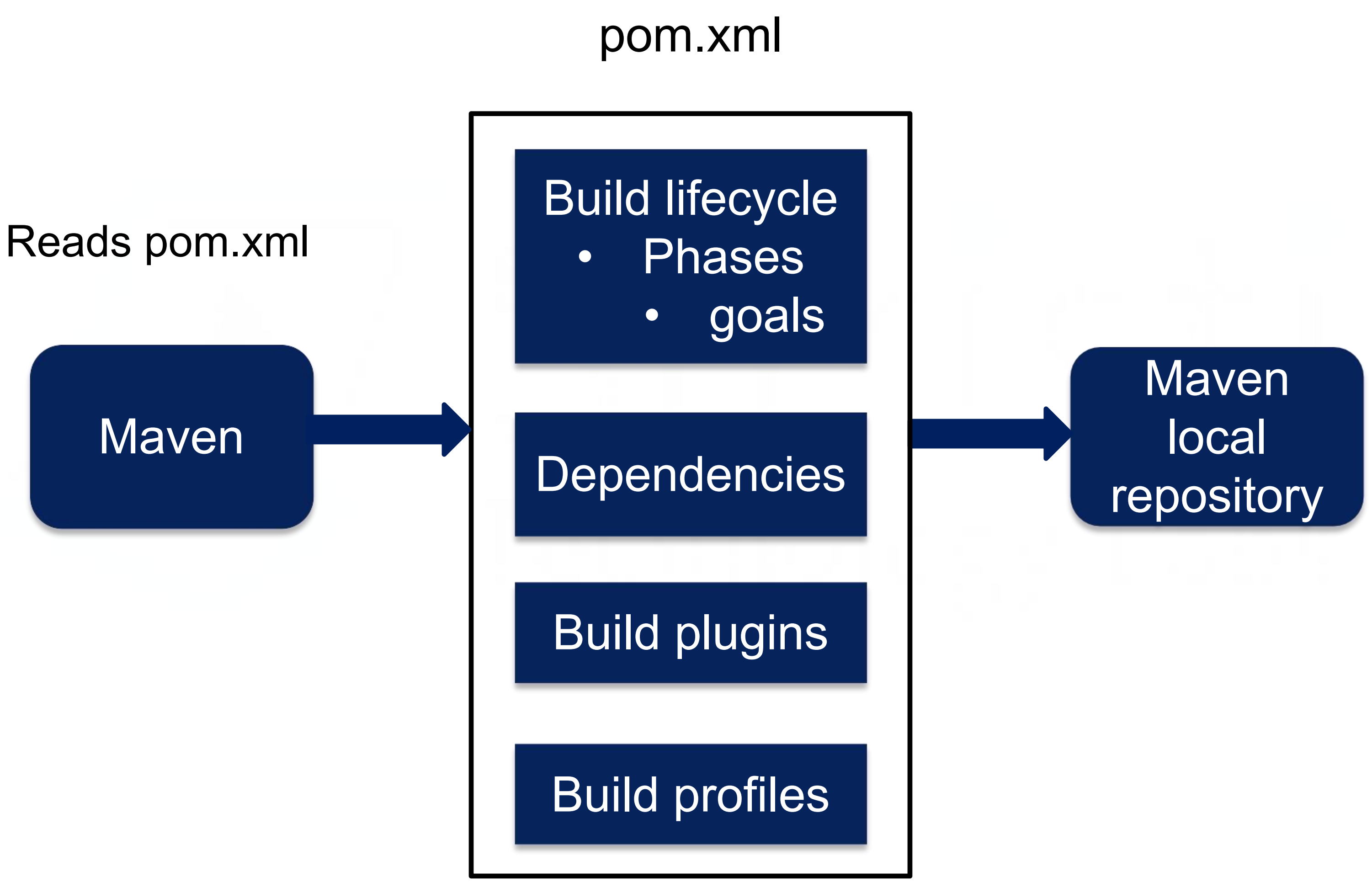


```
SpringMVCBasicMaven
  Deployment Descriptor: SpringMVCBasicMaven
  Spring Elements
  JAX-WS Web Services
  Java Resources
    src/main/java
    src/main/resources
    src/test/java
    src/test/resources
    Libraries
  JavaScript Resources
  Deployed Resources
  src
    main
      java
      resources
      webapp
        resources
        WEB-INF
    test
  target
  pom.xml
```

Directory Structure of Maven project

Directory	Stores
src/main/java	Application related java files
src/main/resources	Resource files (xml, property files)
src/main/webapp	Files related to web application
src/main/webapp/resources	Resource files (css, image files)
src/test/java	Test source files
src/test/resources	Test resource files
target	The directory to store the output of the build

Maven – work flow



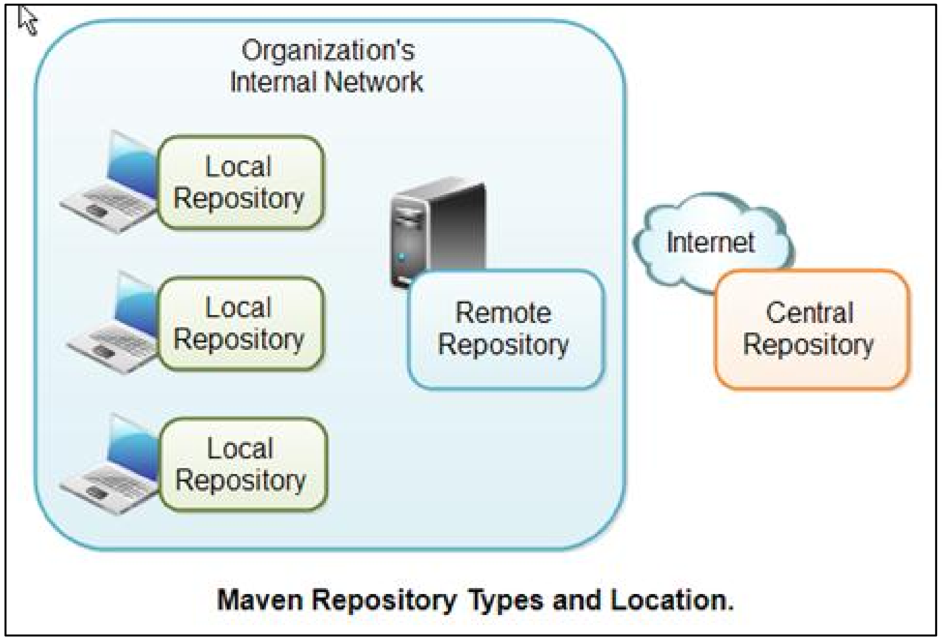
Repository

Maven Repository

Has the build artifacts, tools and dependencies

Types

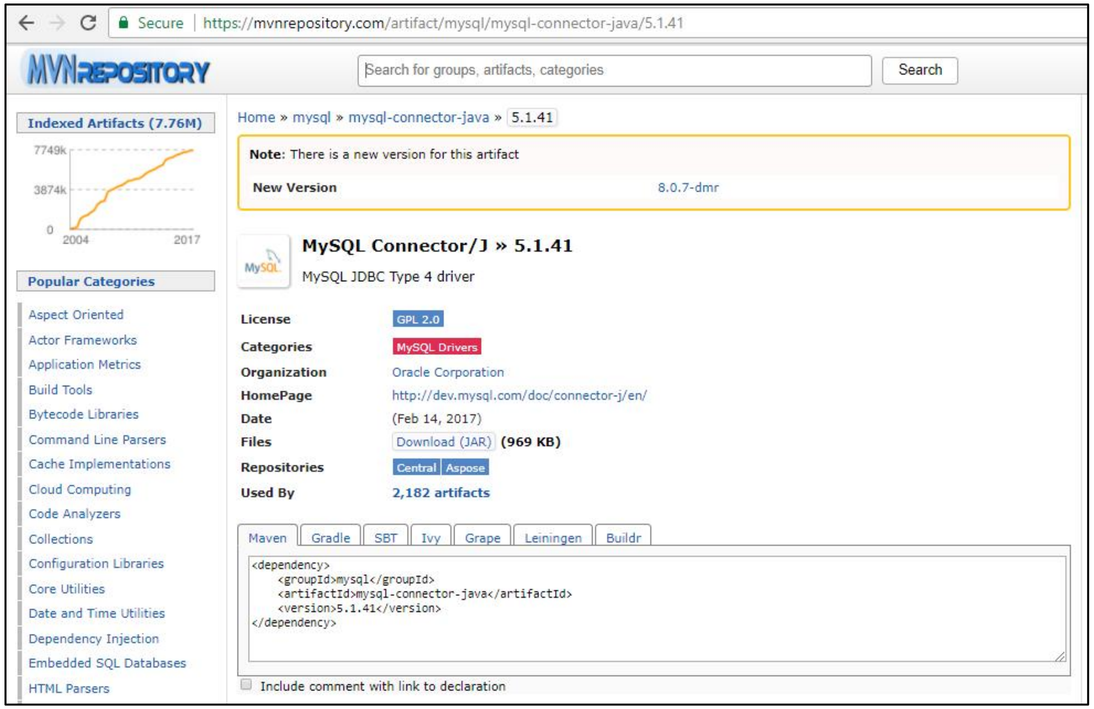
- Central
- Local
- Remote



Maven Repository Types and Location.

Central Repository

- It is provided by the Maven community.
- No special configuration needed to access the central repository
- Access the maven repository from the browser using
  https://mvnrepository.com/
- Use the search box to search for a dependency /library



Secure | https://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.41

MVNREPOSITORY — Search for groups, artifacts, categories — Search

Indexed Artifacts (7.76M)

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers

Home » mysql » mysql-connector-java » 5.1.41

Note: There is a new version for this artifact

New Version 8.0.7-dmr

MySQL Connector/J » 5.1.41

MySQL JDBC Type 4 driver

License	GPL 2.0
Categories	MySQL Drivers
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	(Feb 14, 2017)
Files	Download (JAR) (969 KB)
Repositories	Central Aspose
Used By	2,182 artifacts

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

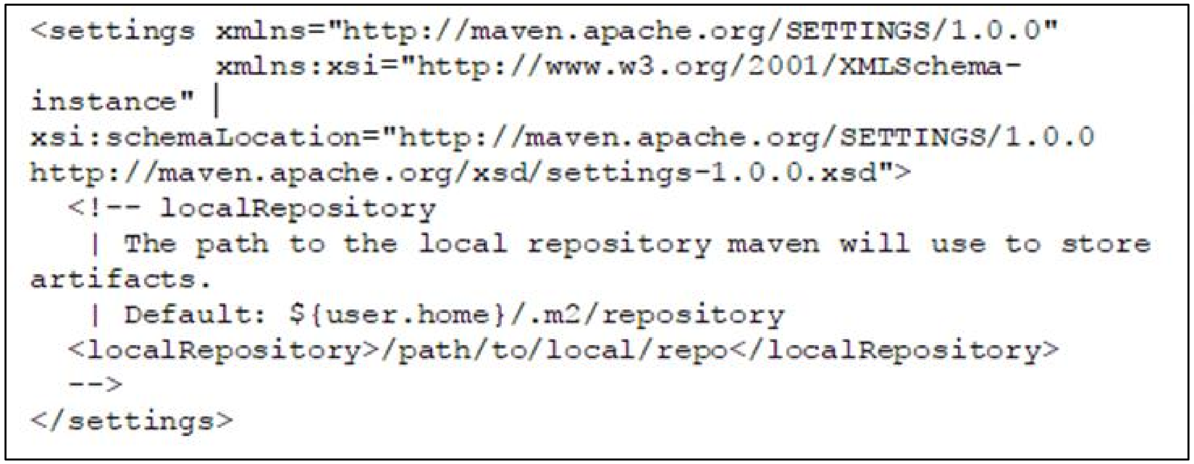
```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.41</version>
</dependency>
```

☐ Include comment with link to declaration

Local Repository

- Is a directory in the developer's computer.
- Contains all the dependencies Maven downloads
- By Default, Maven creates a local repository inside the user home directory
- To choose a different directory, open

  `<maven installation path>/conf/settings.xml`
- Specify the location in <settings>

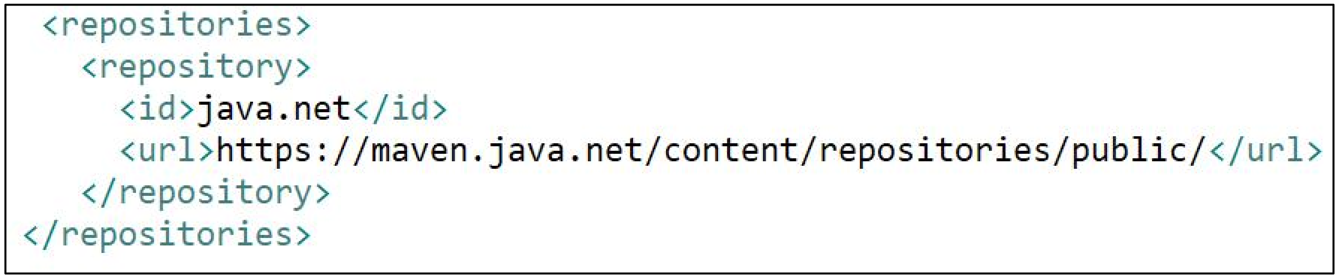


```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" |
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store
artifacts.
   | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->
</settings>
```

Remote Repository

- Can be located anywhere on the internet, or inside a local network
  eg. **java.net, JBoss**

- Configure the remote repository in the POM file.

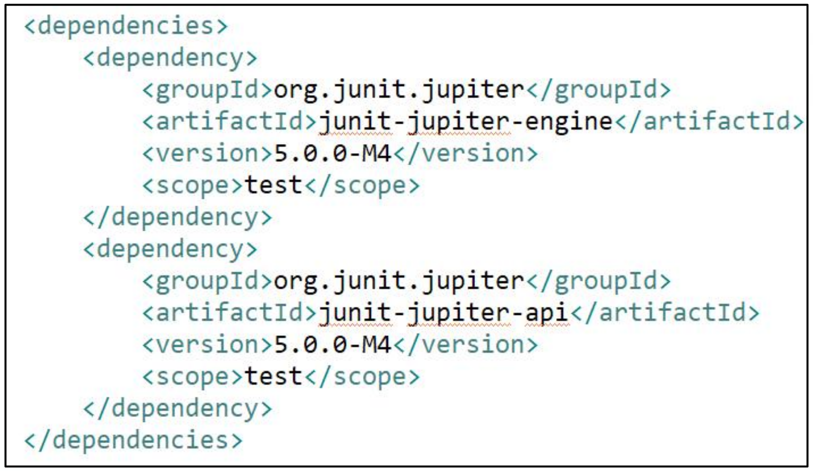


```
 <repositories>
   <repository>
     <id>java.net</id>
     <url>https://maven.java.net/content/repositories/public/</url>
   </repository>
</repositories>
```

Dependencies

Project Dependencies

- A project needs external Java APIs that are packaged as JAR files
- The jar files should be added to the classpath of the project
- Maven has built-in dependency management
- Specify the external libraries that the project depends in the POM file.
- Maven downloads the jars and adds them to the local repository



```xml
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.0.0-M4</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.0.0-M4</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

External Dependencies

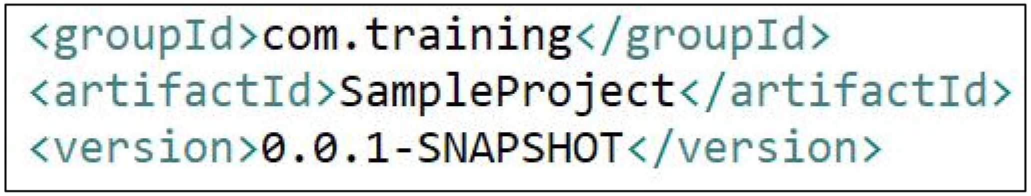
- This is a dependency (JAR file) which is not located in a Maven repository (local, central or remote repository).
- It is located in the local hard disk, (eg. In lib folder or downloads folder)
- External means - external to the Maven repository system
- Configure an external dependency as

```
<dependency>
    <groupId>ojdbc6</groupId>
    <artifactId>ojdbc6</artifactId>
    <scope>system</scope>
    <version>11.2.0.3</version>
    <systemPath>${basedir}/lib/ojdbc6-11.2.0.3.jar</systemPath>
</dependency>
```

Snapshot Dependencies

- These are dependency JAR files which are under development.
- Use snapshot dependencies rather than updating the version.
- Snapshot versions are downloaded into the local repository for every build, even if a matching snapshot version is already located in the local repository.
- Downloading the snapshot dependencies assures that the project has the latest version of the jar always for every build.

- In your project append ***-SNAPSHOT*** to the version number in the beginning of the POM



```xml
<groupId>com.training</groupId>
<artifactId>SampleProject</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

Build Lifecycle

Build Lifecycle

- Build lifecycle is a sequence of phases which define the order in which the goals should be executed
- A phase in a stage in the lifecycle.
- Has three default life cycles
  - **default** – handles compiling and packaging (deployment)
  - **clean** – removes temporary files like .class files, jar files(cleaning)
  - **site** – handles creation of project documentation

- Each life cycle is made of different build phases
- Each build phase is made of plugin goals
- The build lifecycle can be executed directly or any of the build phase can be executed

default lifecycle

- The default life cycle cannot be executed directly.
- Execute the build phases in default lifecycle
- The main build phases in the default lifecycle
  - **`validate, compile, test, package, install, deploy`**
- To execute the build phase pass the phase name in mvn command

  `mvn install`
- While executing a build phase, all build phases before that build phase in this standard phase sequence are executed.

eg. **`mvn install`**

- This command executes each default life cycle phase in order (validate, compile, package, etc.), before executing install
- The build phase is made up of plugin goals

Shristi Technology Labs

Build phase in default lifecycle

Build Phase	Description
validate	To validate if the project has all the necessary information. check if the dependencies are downloaded.
compile	To compile the source code of the project.
test	To runs the tests against the compiled source code using a suitable unit testing framework.
package	To package the compiled code in its distributable format, like a JAR.
install	To Install the package into the local repository, for use as a dependency in other projects locally.
deploy	To copy the final package to the remote repository for sharing with other developers and projects.

clean lifecycle

- Used to remove the temporary files like .class files and .jar files
- The main build phases in the default lifecycle
  - `pre-clean, clean, post-clean`

eg. **`mvn clean`**

- This command will run through pre-clean and clean

Build Phase	Description
pre-clean	execute processes needed prior to the actual project cleaning
clean	remove all files generated by the previous build
post-clean	execute processes needed to finalize the project cleaning

site lifecycle

- Used to create fresh documentation
- Execute the lifecycle using `mvn site`
- Typical usage is `mvn clean site-deploy` to clean a project and generate fresh documentation and deploy in the server

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

Plugin Goals

- A plugin goal represents a specific task (finer than a build phase) which contributes to the building and managing of a project
- A plugin provides a set of goals that can be executed using the syntax

```
mvn [plugin-name]:[goal-name]
```

- A goal may be bound to zero or more build phases
- The goal bound to a build phase will be executed by default while executing the build phase.

```
mvn compiler:compile
```

- The goal not bound to a build phase can be called directly

```
mvn  clean dependency:copy-dependencies package
```

Build Profiles

- Profiles enables to build the project using different configurations.
- Rather than creating two separate POM files, specify a profile with the different build configuration, and build the project with the build profile when needed
- A profile can be triggered/activated in several ways:
  - Explicitly
  - Through Maven settings
  - Based on environment variables
  - OS settings

```
<profile>
  <id>jdk-1.4</id>

  <activation>
    <jdk>1.4</jdk>
  </activation>

  <repositories>
    <repository>
      <id>jdk14</id>
      <name>Repository for JDK 1.4 builds</name>
      <url>http://www.myhost.com/maven/jdk14</url>
      <layout>default</layout>
      <snapshotPolicy>always</snapshotPolicy>
    </repository>
  </repositories>
</profile>
```

```
<settings>
    <activeProfiles>
    <activeProfile>development</activeProfile>
  </activeProfiles>
  ...
</settings>
```

Summary

- Introduction to Maven
- pom.xml
- Directory Structure of the Maven project
- Maven workflow
- Build Lifecycle
- Build phase & goals
- Build profiles