

[Connected World. Connected Solutions.]



Continuous Integration

- **"continuous integration"** refers to a process that builds and tests code on a frequent basis.
- The continuous integration servers constantly monitor source code repositories and as soon as new changes/commits are detected, they initiate a new build cycle
- The build cycle actually involves code compilation and, in addition, may involve various tests and code analysis

Continuous Integration

Maven

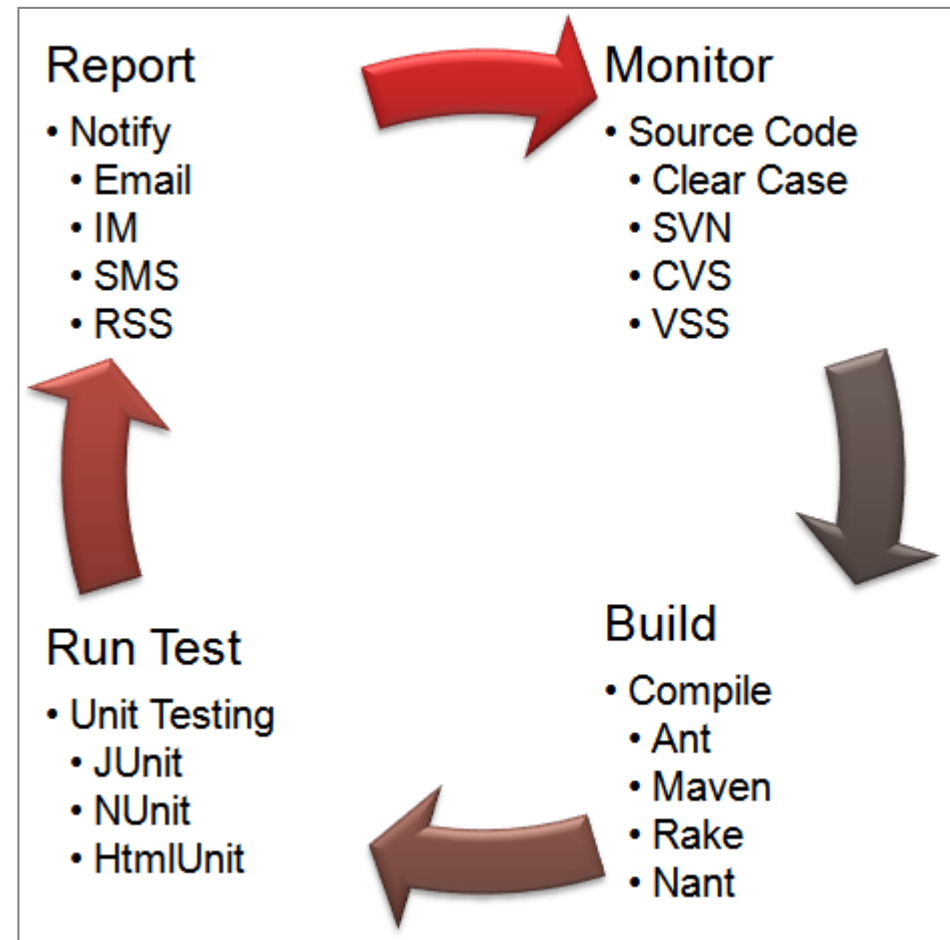
- Apache Maven is a software project management based on the concept of a project object model (POM)
- Maven can manage a project's build, reporting and documentation from a central piece of information

Jenkins

- Jenkins is a open-source continuous integration server
- Built with Java, it provides over 400 plug-ins to support building and testing virtually any project

Sonar

- Sonar is a quality measurement and reporting tool, which produces metrics on build quality such as unit test coverage and static analysis tools (Findbugs, PMD and Checkstyle)



Introduction to Maven

Agenda

- Introduce Maven
- Basic Maven POM File and Project Structure
- Dependencies
- How to create a new Maven project in Eclipse
- Maven Plugins
- Maven profiles

DAY 1

What is Apache Maven & Prerequisites ?

- Apache Maven is a software project management and comprehension tool.
- Maven provides developers a complete build lifecycle framework.
- Maven can manage a project's build, reporting and documentation from a central piece of information.



Prerequisites

- Introduction to the Java Stack
- Basic Java and XML skill

Maven Objective and Mindset

- Objective
 - Making the build process easy.
 - Providing a uniform build system.
 - Providing quality project information
 - Reduce the development effort time to make builds
- All build systems are essentially the same:
 - Compile Source code
 - Copy Resource
 - Compile and Run Tests
 - Package Project
 - Deploy Project
 - Cleanup
- Describe the project and configure the build
 - You don't script a build
 - Maven has no concept of a condition
 - Plugins are configured

Build tools

➤ ANT

- Ant's first official version was released in 2000.
- Multiple targets can be chained to combine single units of work into full workflows.
- Ant doesn't give any guidance on how to structure your project.
- External libraries required by your project are usually checked into version control, because there is no automated mechanism to pull them from a central location.

➤ Maven

- Maven 2, released in October 2005.
- It provided a standardized project and directory structure, as well as dependency management.
- Projects consisting of multiple modules could define their dependencies on each other

➤ Gradle

- Gradle build scripts are declarative, readable, and clearly express their intention.
- Writing code in Groovy instead of XML, significantly cuts down the size of a build script and is far more readable.

Problems Without Maven

- **Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.
- **Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.
- **Building and Deploying the project:** We must have to build and deploy the project so that it may work.

Advantages and Disadvantages of Maven

➤ Advantages

- It maintains the repository of the archives. So does not required to maintain jar archives in local.
- All compile/build/dependency info is bundled with your maven spec.
- It simplifies the way of your build will work, because every build follows standard procedure.
- All dependencies are downloaded automatically.
- Every member of your team will by building/deploying in the same way with every compile

➤ Disadvantages

- Your project pretty much has to be laid out the correct way.
- Its verbose and complex
- If you have dependent jar is not mavenized, it is very difficult to integrate those jar in your application.
- Learning curve should require
- Developer should know maven command line or use IDE to integrate maven.

Maven Features

- Dependency System
- Multi-module builds
- Consistent project structure
- Consistent build model
- Plugin oriented
- Project generated sites
- Profiling

Maven POM

- Stands for Project Object Model
- Describes a project
 - Name and Version
 - Artifact Type
 - Source Code Locations
 - Dependencies
 - Plugins
 - Profiles (Alternate build configurations)
- Uses XML by Default
 - Not the way Ant uses XML

Maven Project Name (GAV)

- Maven uniquely identifies a project using:
 - ❑ groupId: Arbitrary project grouping identifier (no spaces or colons)
 - Usually loosely based on Java package
 - ❑ artifactId: Arbitrary name of project (no spaces or colons)
 - ❑ version: Version of project
 - Format {Major}.{Minor}.{Maintenance}
 - Add '-SNAPSHOT' to identify in development
- GAV Syntax: groupId:artifactId:version

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.techm.training</groupId>
  <artifactId>maven-training</artifactId>
  <version>1.0</version>
</project>
```

Packaging

- Build type identified using the “packaging” element
- Tells Maven how to build the project
- Example packaging types:
 - pom, jar, war, ear, custom
 - Default is jar

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.techm.training</groupId>
  <artifactId>maven-training</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
</project>
```

Maven Folder Structure

- Maven is maintained its own project structure
- src: All project source files go in this directory
- src/main: All sources that go into primary artifact
- src/main/java: All java source files
- src/main/webapp: All web source files
- src/main/resources: All non compiled source files
- src/test: All sources contributing to testing project
- src/test/java: All java test source files
- src/test/resources: All non compiled test source files
- target: Default work directory

Maven Build Lifecycle

- A Maven build follow a lifecycle
- Default lifecycle
 - generate-sources/generate-resources
 - compile
 - test
 - package
 - integration-test (pre and post)
 - Install
 - deploy
- There is also a Clean lifecycle

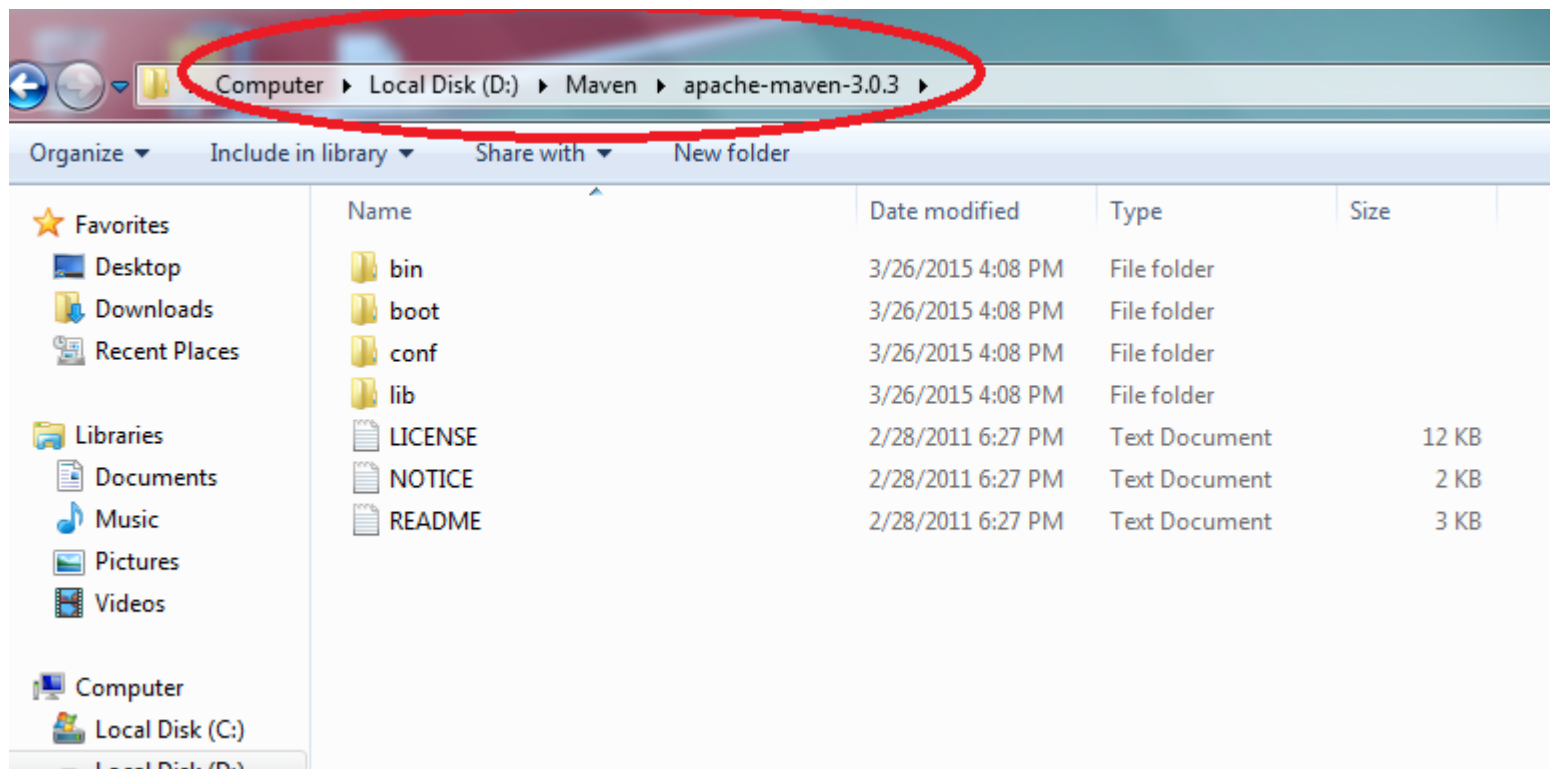
Example Maven Goals

To invoke a Maven build you set a lifecycle “goal”

- mvn install
 - Invokes generate* and compile, test, package, integration-test, install
- mvn clean
 - Invokes just clean
- mvn clean compile
 - Clean old builds and execute generate*, compile
- mvn compile install
 - Invokes generate*, compile, test, integration-test, package, install
- mvn test clean
 - Invokes generate*, compile, test then cleans

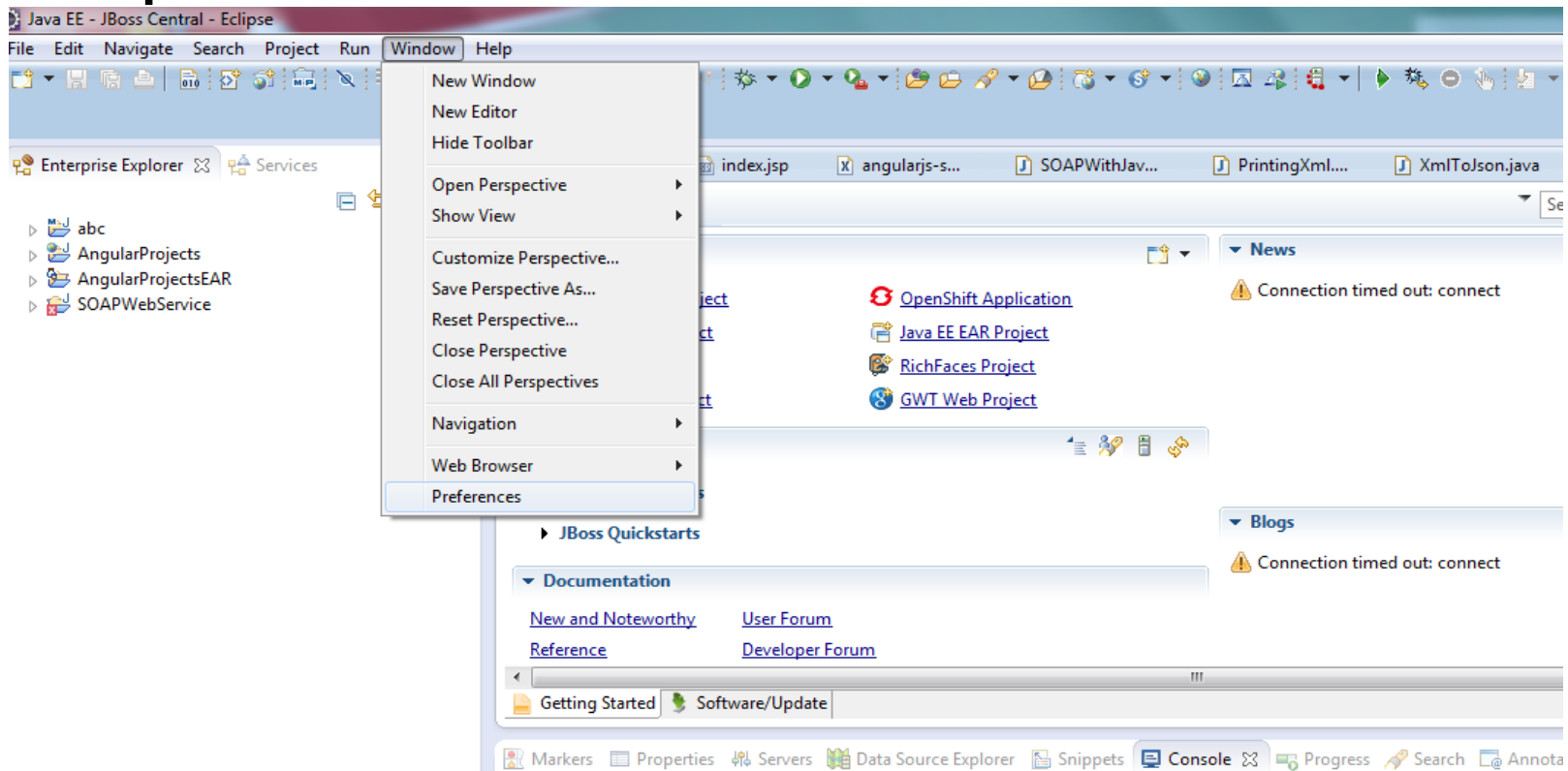
How to Install Maven

- **Step 1 : Download maven and extract it
apache-maven-3.1.1-bin.zip**



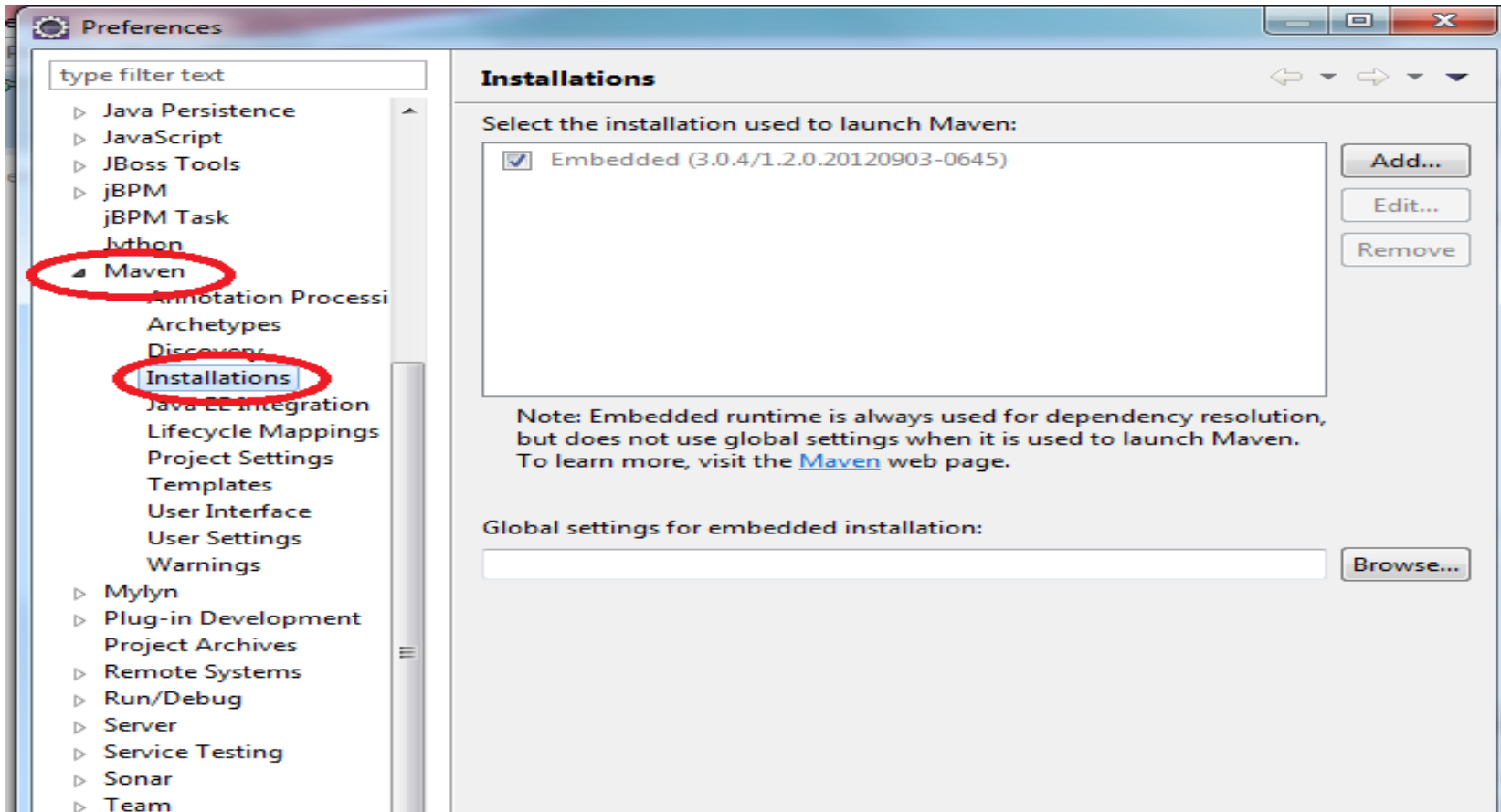
Configuring Maven to Workspace

- **Step 2 : Open Eclipse IDE**
- **Step 3 : Select Windows -> Preferences**



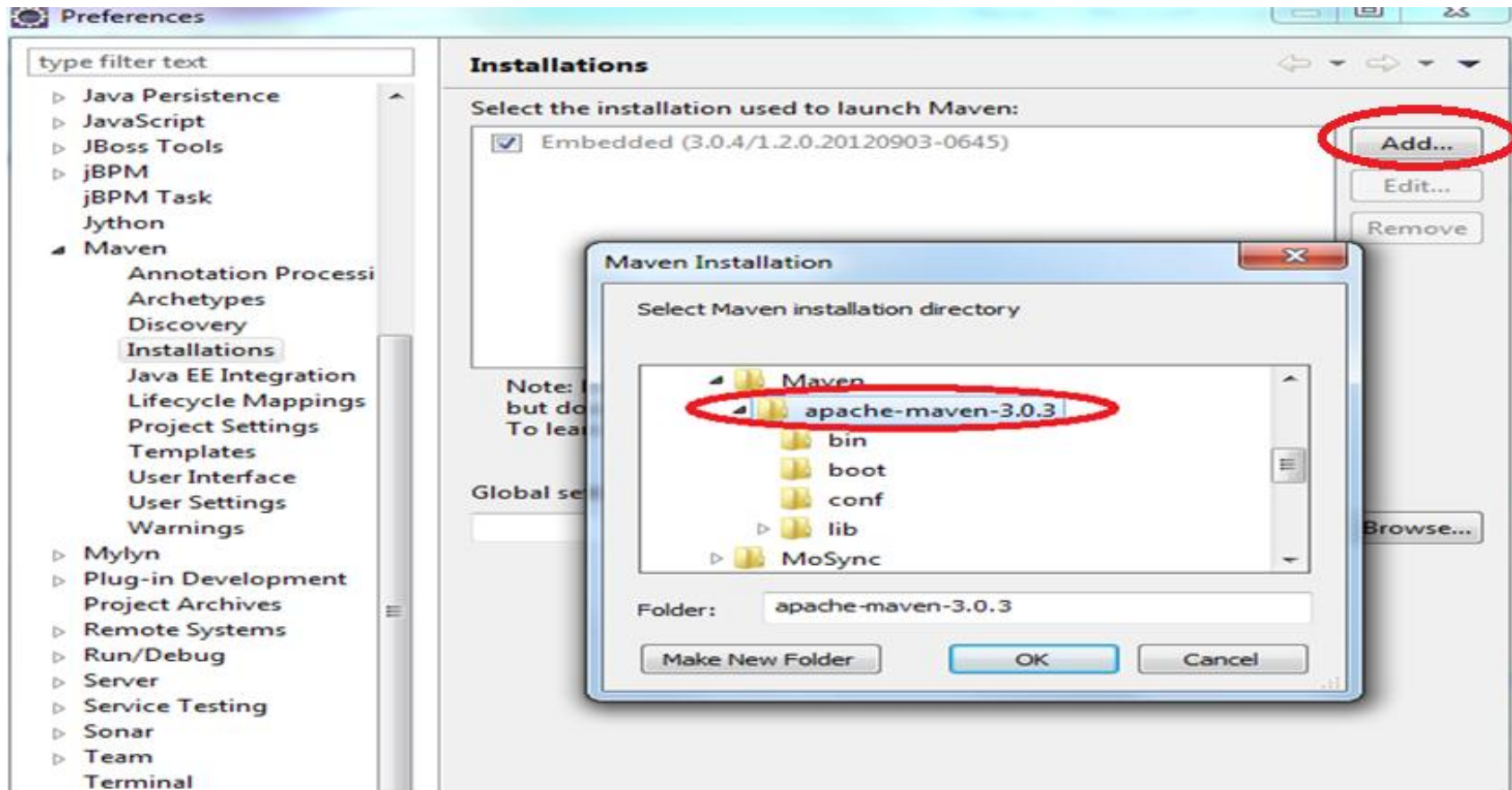
Configuring Maven to Workspace Contd...

- **Step 4** : Click on Maven Tab -> Installations



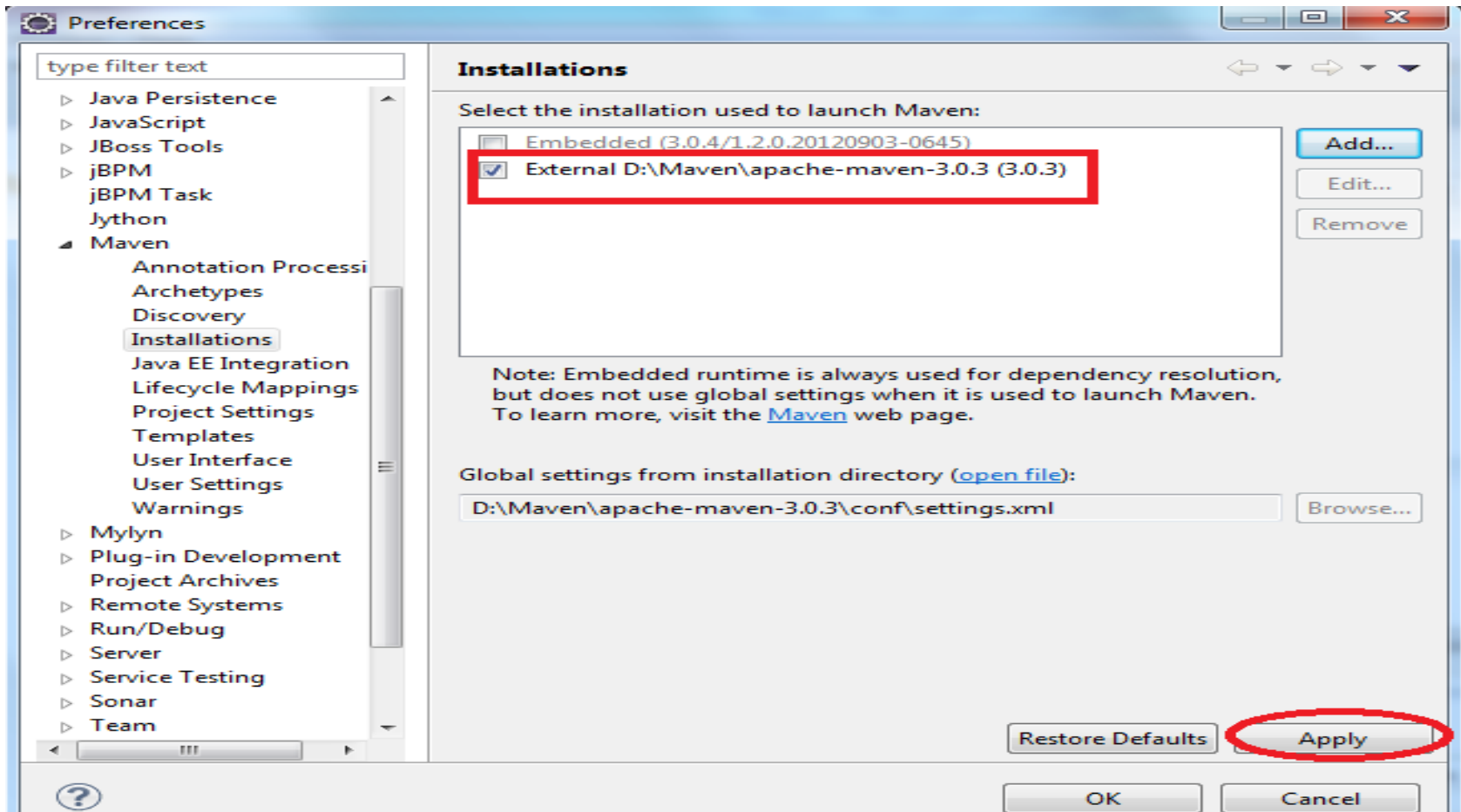
Configuring Maven to Workspace Contd...

- **Step 5** : Click on Add button and specify the maven extracted folder path.



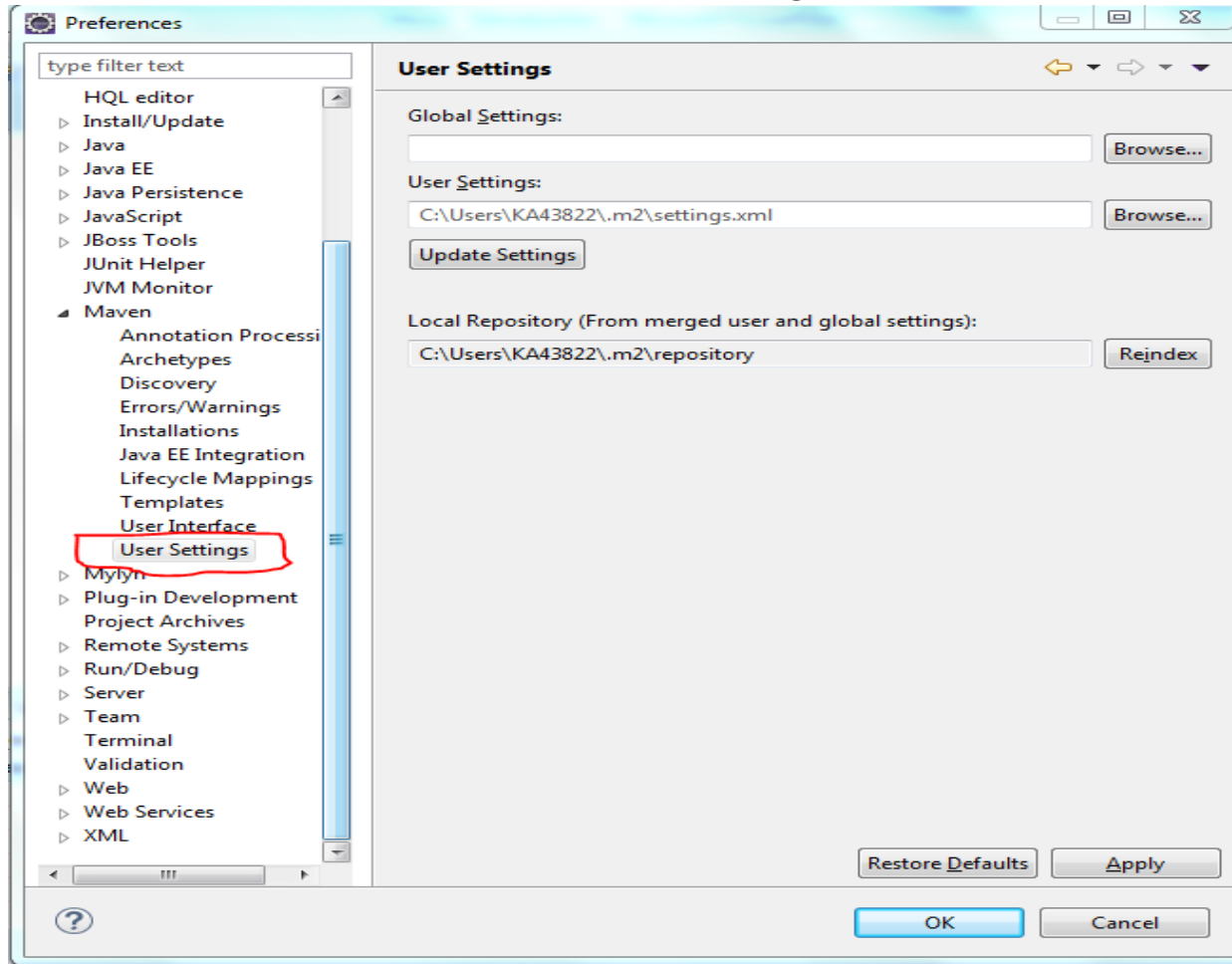
Configuring Maven to Workspace Contd...

- **Step 6 :** The maven extracted folder path is added and click **Apply**



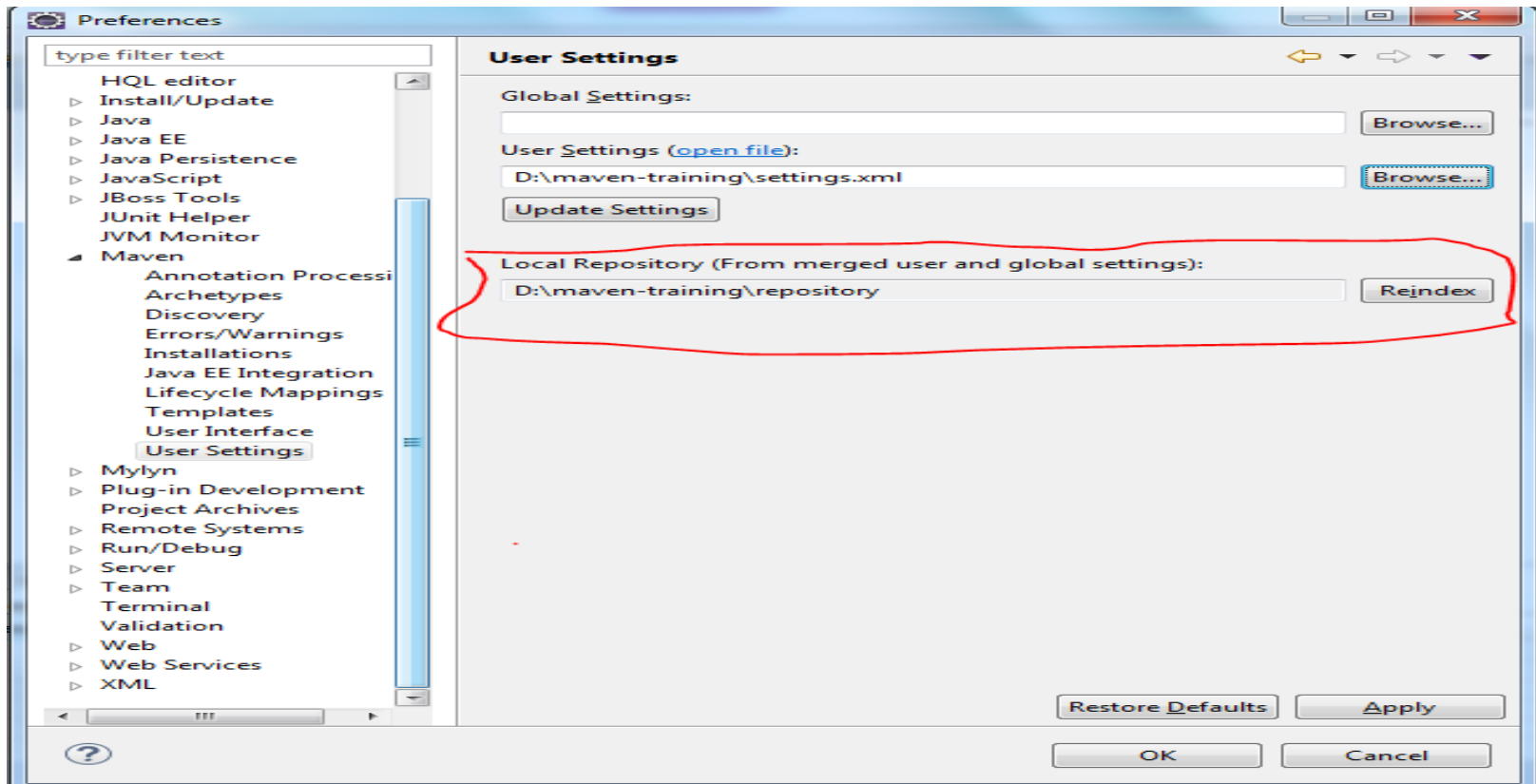
Configuring Maven to Workspace Contd...

- **Step 7 : Click on Maven -> User Settings**



Configuring Maven to Workspace Contd...

- **Step 8 :** In the User Settings tab specify the settings.xml file path location by clicking Browse button.
- **Step 9 :** Keep that settings.xml file in the extracted folder of apache maven.



settings.xml

- The settings.xml file contains elements used to define values which configure Maven execution in various ways, like the pom.xml.
- It should not be bundled to any specific project, or distributed to an audience.
- These include values such as the local repository location, alternate remote repository servers, and authentication information.

```
<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>
```

Local Repository

- Maven local repository is a folder location on your machine.
- Maven local repository keeps your project's all dependencies (library jars, plugin jars etc).
- When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository.
- Maven local repository by default get created by Maven in %USER_HOME% directory. To override the default location, mention another path in Maven settings.xml file.

```
<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: ~/.m2/repository -->
  <localRepository>Your Local path Ex:D:\Repository </localRepository>
```

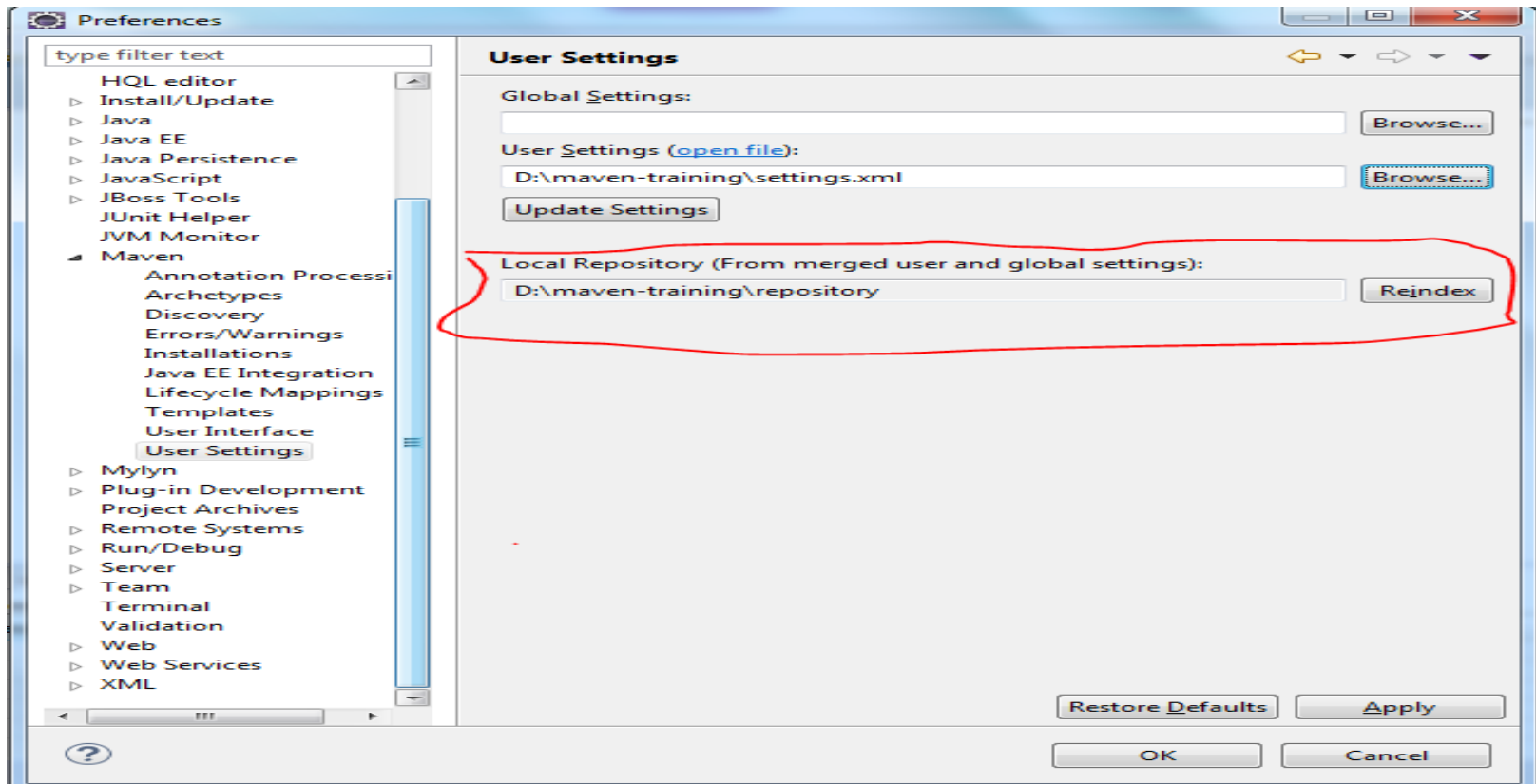
Local Repository

- Local Repository path can be specified in settings.xml file under `<localRepository>` tag

```
settings.xml ×
<server>
  <username>kumaram</username>
  <password>Nissan18</password>
  <id>central</id>
</server>
<server>
  <username>kumaram</username>
  <password>Nissan18</password>
  <id>snapshots</id>
</server>
</servers>
<localRepository>D:\maven-training\repository</localRepository>
<profiles>
  <profile>
    <repositories>
      <repository>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
        <id>central</id>
        <name>libs-release</name>
        <url>http://10.128.184.89:8080/artifactory/libs-release</url>
      </repository>
      <repository>
        <snapshots />
        <id>snapshots</id>
        <name>libs-snapshot</name>
        <url>http://10.128.184.89:8080/artifactory/libs-snapshot</url>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>
```

Local Repository

- Local Repository path is mentioned in Maven -> User Settings tab.
- The path mentioned in settings.xml file is shown here automatically.



Maven Dependency

Dependency management is one of the features of Maven that is best known to users. Maven introduced concept of transitive dependency.

Dependencies consist of:

- GAV
- Scope: compile, test, provided (default=compile), runtime, system
- Type: jar, pom, war, ear, zip (default=jar)

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
    <!-- Adding javaeeapi jar for all layers -->
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>${javaeeapi.version}</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

Transitive Dependencies

- Transitive Dependency Definition:
 - A dependency that should be included when declaring project itself is a dependency
- ProjectA depends on ProjectB
- If ProjectC depends on ProjectA then ProjectB is automatically included
- Only compile and runtime scopes are transitive
- Transitive dependencies are controlled using:
 - Exclusions
 - Optional declarations

Dependency Exclusions

- Exclusions exclude transitive dependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
    <!-- Adding javaeeapi jar for all layers -->
    <dependency>
      <groupId>org.apache.openejb</groupId>
      <artifactId>javaee-api</artifactId>
      <version>6.0-5</version>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.apache.openejb</groupId>
          <artifactId>javaee-api</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>
</project>
```


Dependency Management

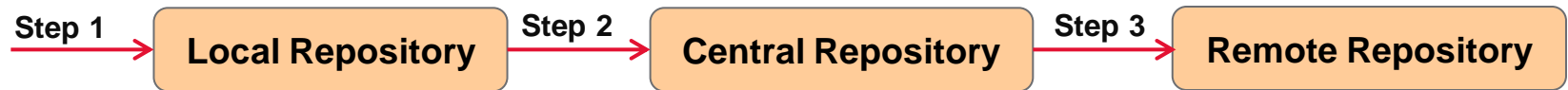
- If more than one project depends on a specific dependency, you can list the dependency in Dependency Management
- What do you do when versions collide?
 - Allow Maven to manage it?
 - Complex and less predictable
 - Take control yourself
 - Manage the version manually

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>3.0.5.RELEASE</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
    </dependency>    <!-- Look ma, no version! -->
  </dependencies>
</project>
```

DAY 2

Maven Repository

- A repository is a place i.e. directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.
- Maven repository are of three types
 - ❖ Local Repository
 - ❖ Central Repository
 - ❖ Remote Repository
- Maven searches for the dependencies in the following order:



Local Repository

- Maven local repository is a folder location on your machine.
- Maven local repository keeps your project's all dependencies (library jars, plugin jars etc).
- When you run a Maven build, then Maven automatically downloads all the dependency jars into the local repository.
- Maven local repository by default get created by Maven in %USER_HOME% directory. To override the default location, mention another path in Maven settings.xml file.

```
<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: ~/.m2/repository -->
  <localRepository>Your Local path Ex:D:\Repository </localRepository>
```

Central Repository

- Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries.
- When Maven does not find any dependency in local repository, it starts searching in central repository using the URL mentioned in configuration file (settings.xml)
- Central repositories can be configured in your local intranet system, by using mirror configurations, can connect local central repository.

```
<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">
  <servers>
  <server>
    <id>deploymentRepo</id>
    <username>repouser</username>
    <password>repopwd</password>
  </server>
  <profile>
    <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>
```

Remote Repository

- Remote repositories refer to any other type of repository, accessed by a variety of protocols such as `file://` and `http://`
- Local and Remote repositories are structured the same way so that scripts can easily be run on either side, or they can be synced for offline use
- Sometime, Maven does not find a mentioned dependency in central repository as well then it stopped build process and output error message to console. To prevent such situation, Maven provides concept of Remote Repository which is developer's own custom repository containing required libraries or other project jars.
- If a remote repository has not been mentioned, Maven simply stops the processing and throws error.

Maven Plugins

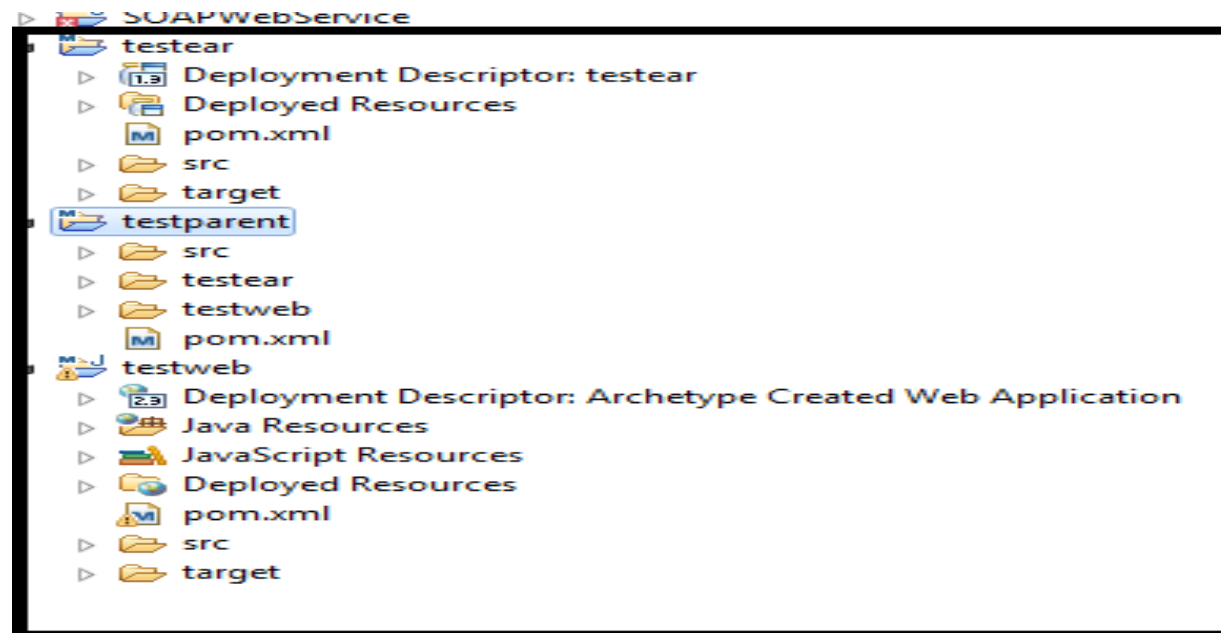
- Maven is actually a plugin execution framework where every task is actually done by plugins.
- A plugin generally provides a set of goals and which can be executed.
- Plugins are specified in pom.xml using plugin element.
- Each plugin have multiple goals.
- Plugin can execute following format `mvn [plugin-name]:[goal-name]`
- Ex: `mvn compiler:compile`

Plugin	Description
Clean	Clean up target after the build. Deletes the target directory.
Compiler	Compile java source files
Jar	Builds jar file from the current project
War	Builds a war file from the current project
Surefire	Runs the junit unit tests . Creates test report
Javadoc	Generates the javadoc
Antrun	Runs a set ant task from any phase mentioned of the build.

Maven Project Structure

- Maven splits project structure into three parts

parent folder
ear folder
web folder



Project Inheritance by using Parent

- POM files can inherit configuration
 - groupId, version
 - Project Config
 - Dependencies
 - Plugin configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <parent>
    <artifactId>maven-training-parent</artifactId>
    <groupId>com.techm.training</groupId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>maven-training</artifactId>
  <packaging>jar</packaging>
</project>
```

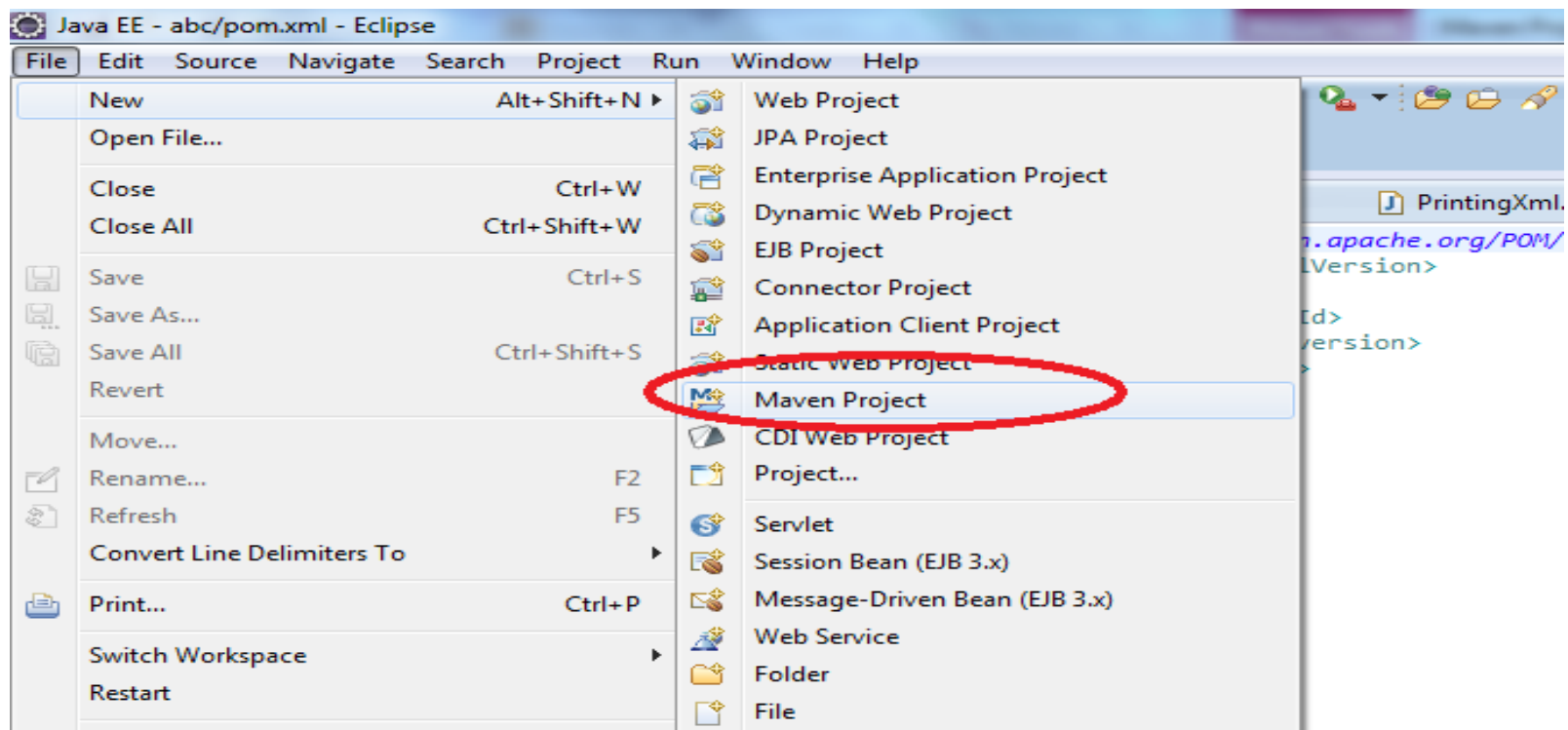
Multi Module Projects

- Maven can do multi-module support
- Each maven project creates one primary artifact
- Parent POM is used to group modules

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.techm.training</groupId>
  <artifactId>maven-training-parent</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>
  <!-- adding all properties -->
  <properties>
    <!-- Define all properties>
  </properties>
  <!-- adding all modules -->
  <modules>
    <module>training-web</module>
    <module>training-ear</module>
  </modules>
</project>
```

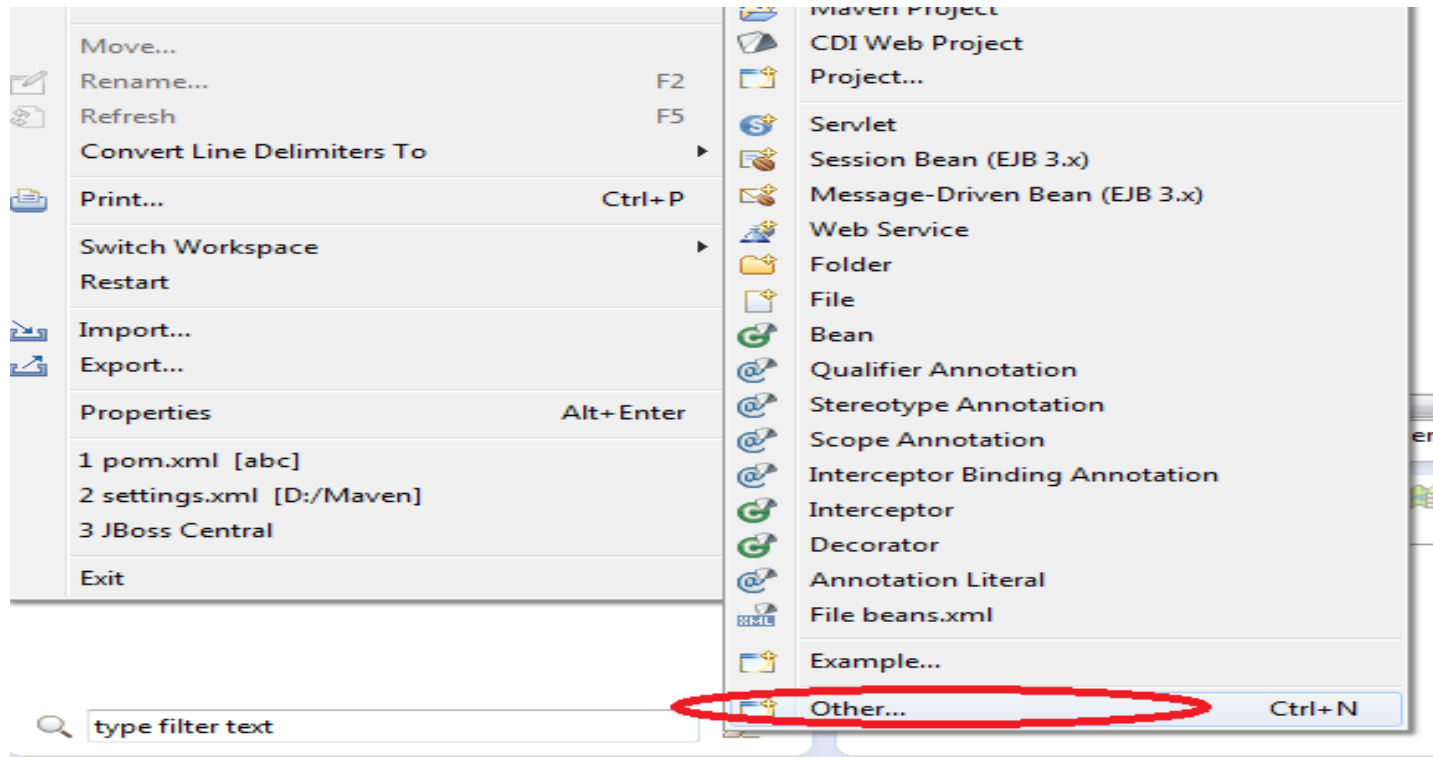
Creating the parent folder

- After configuring Maven to workspace successfully, we can create a maven project.
- Click on File menu -> New -> Maven Project.



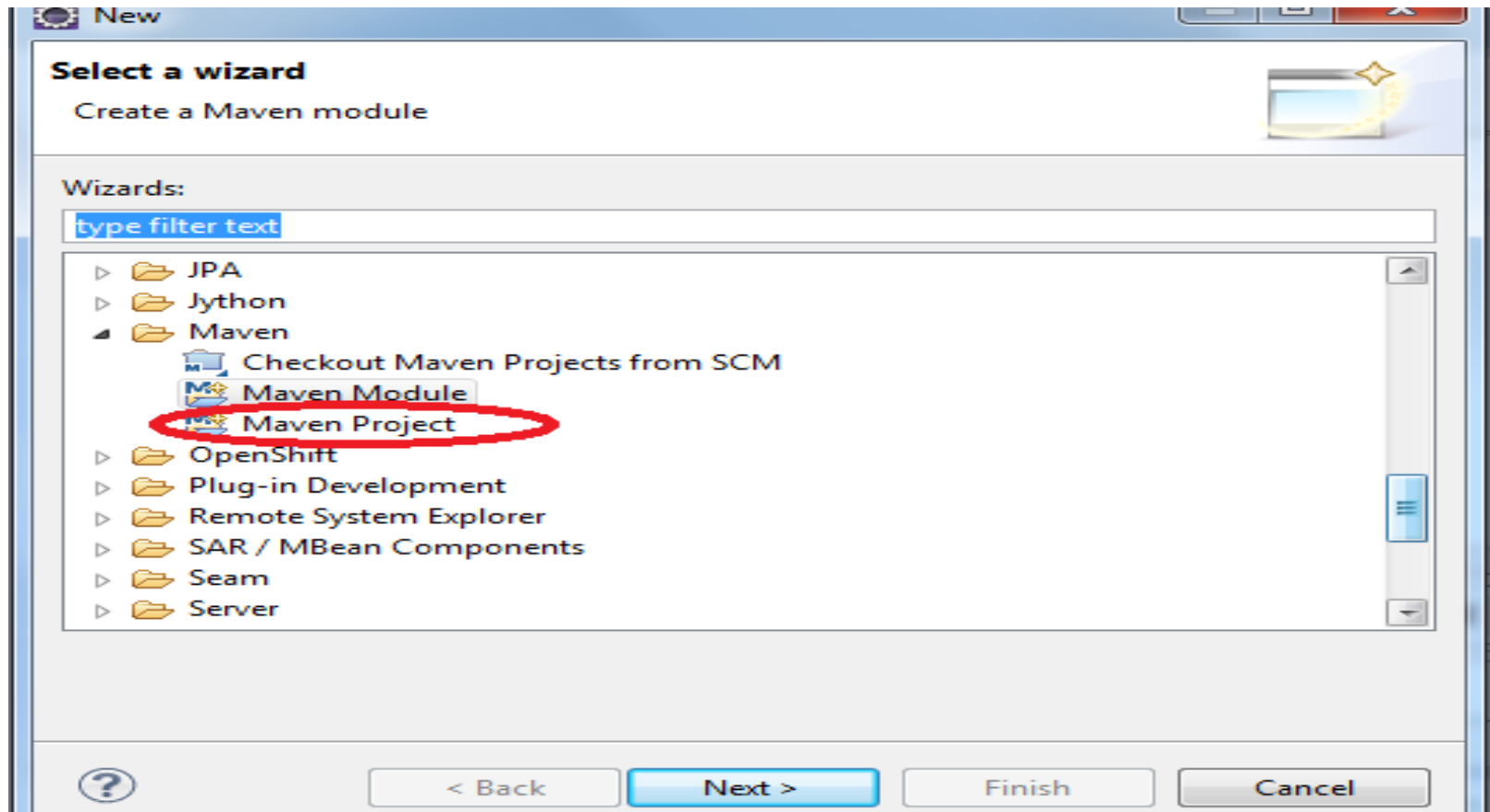
Creating the parent folder contd..

- If Maven project is not listed under the New menu options, click File -> New -> Other...



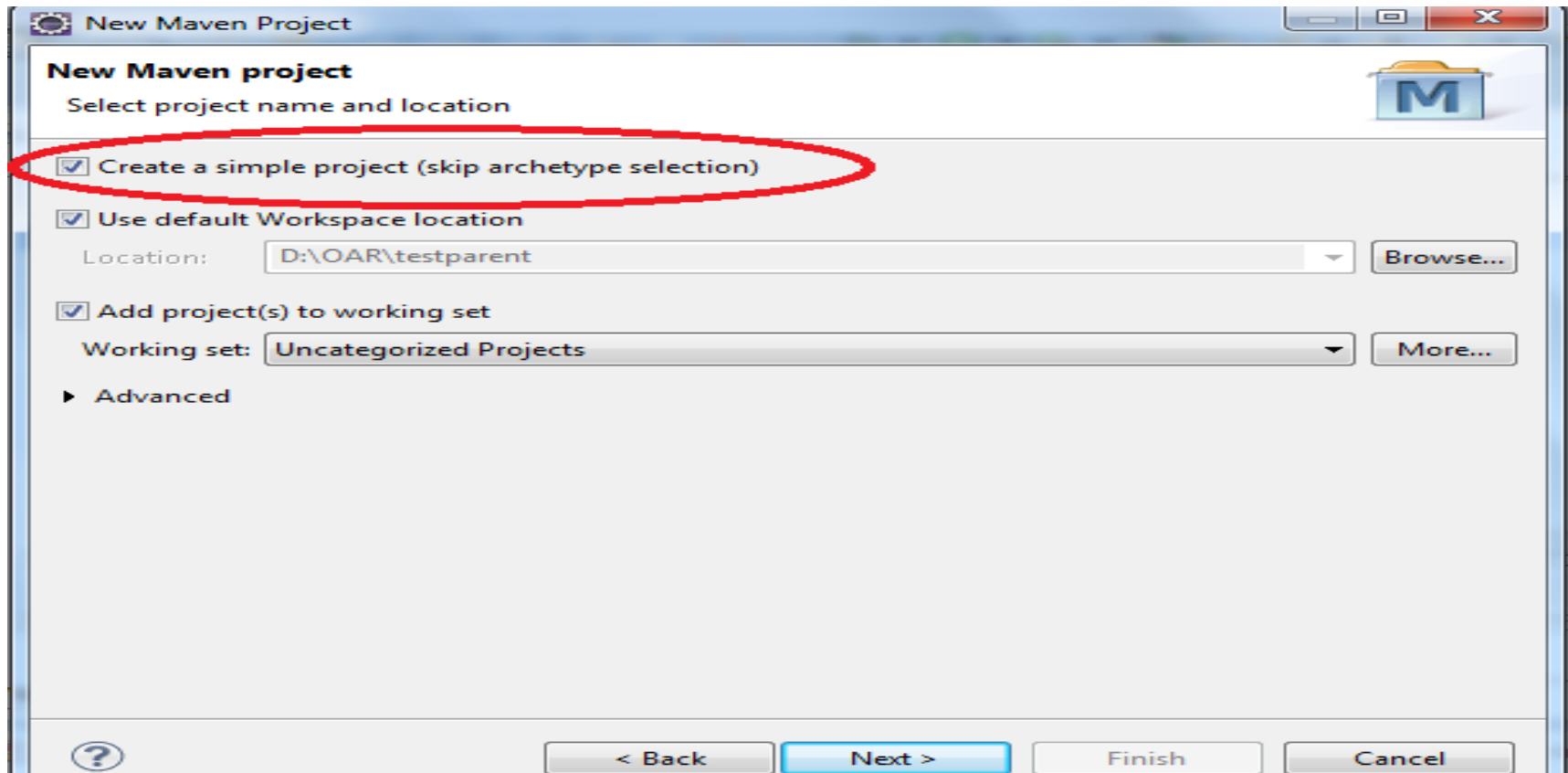
Creating the parent folder contd..

- In the Select Wizard, Choose Maven Project and click Next.



Creating the parent folder contd..

- In the New Maven Project Window, enable **Create a simple project** Check box and click Next button.



Creating the parent folder contd..

- In this window give the project name in Group Id, ParentFolder name in Artifact Id.
- Since it is a parent folder, select packaging as POM and click Finish.

New Maven Project
Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging: (highlighted with a red oval)

Name:

Description:

Parent Project

Group Id:

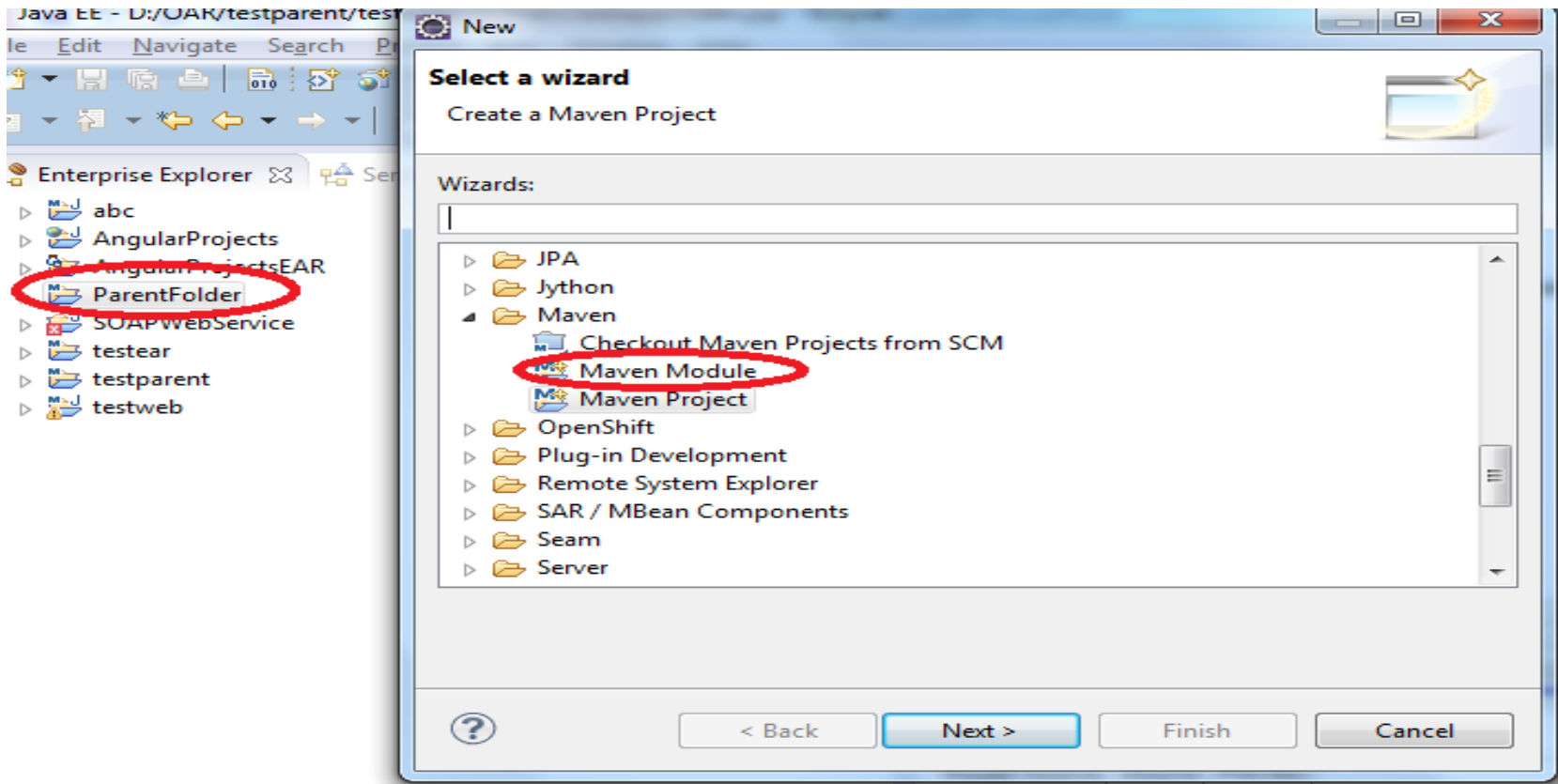
Artifact Id:

Version:

Advanced

Adding ear module to parent

- Right click on ParentFolder -> New -> Other-> Maven Module.
- Then Click Next.



Adding ear module to parent contd...

- Enter the EARmodule name in **ModuleName** Tab.
- Check **Create a simple project** check box is enabled and click **Next**.

New Maven Module
Select the module name and parent

☒ Create a simple project (skip archetype selection)

Module Name:

Parent Project:

☐ Add project(s) to working set

Working set:

► Advanced

Adding ear module to parent contd...

- Since it is a EAR module, type ear in **Packaging** tab and click **Finish**.

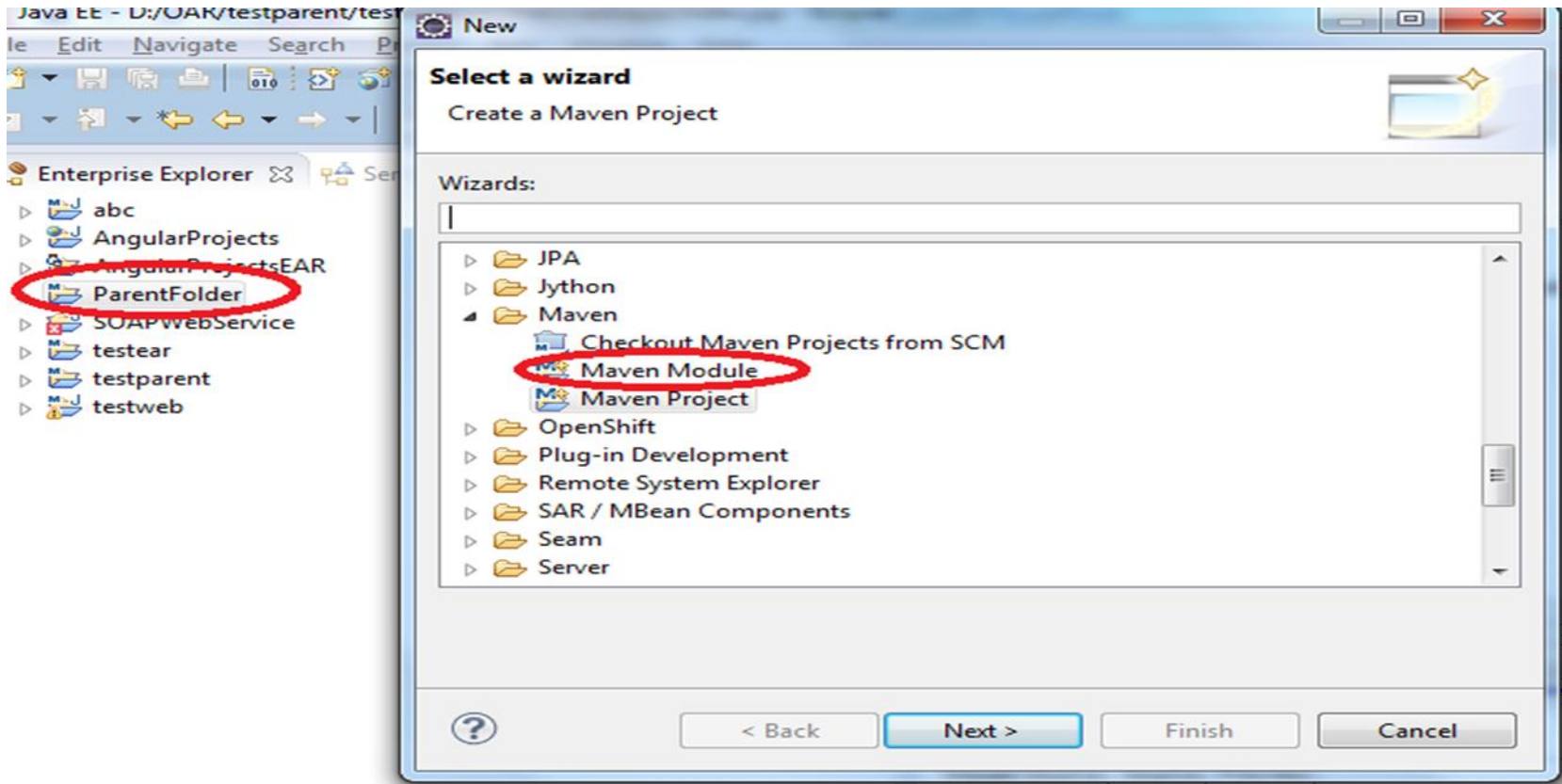
The screenshot shows the 'New Maven Module' dialog box with the following fields:

- Artifact**
 - Group Id: ProjectName
 - Artifact Id: EARModule
 - Version: 0.0.1-SNAPSHOT
 - Packaging: ear** (highlighted with a red circle)
 - Name: EARModule name
 - Description: This is Project's EAR Module
- Parent Project**
 - Group Id: ProjectName
 - Artifact Id: ParentFolder
 - Version: 0.0.1-SNAPSHOT
- Advanced** (collapsed)

At the bottom, there are buttons for '< Back', 'Next >', **Finish**, and 'Cancel'.

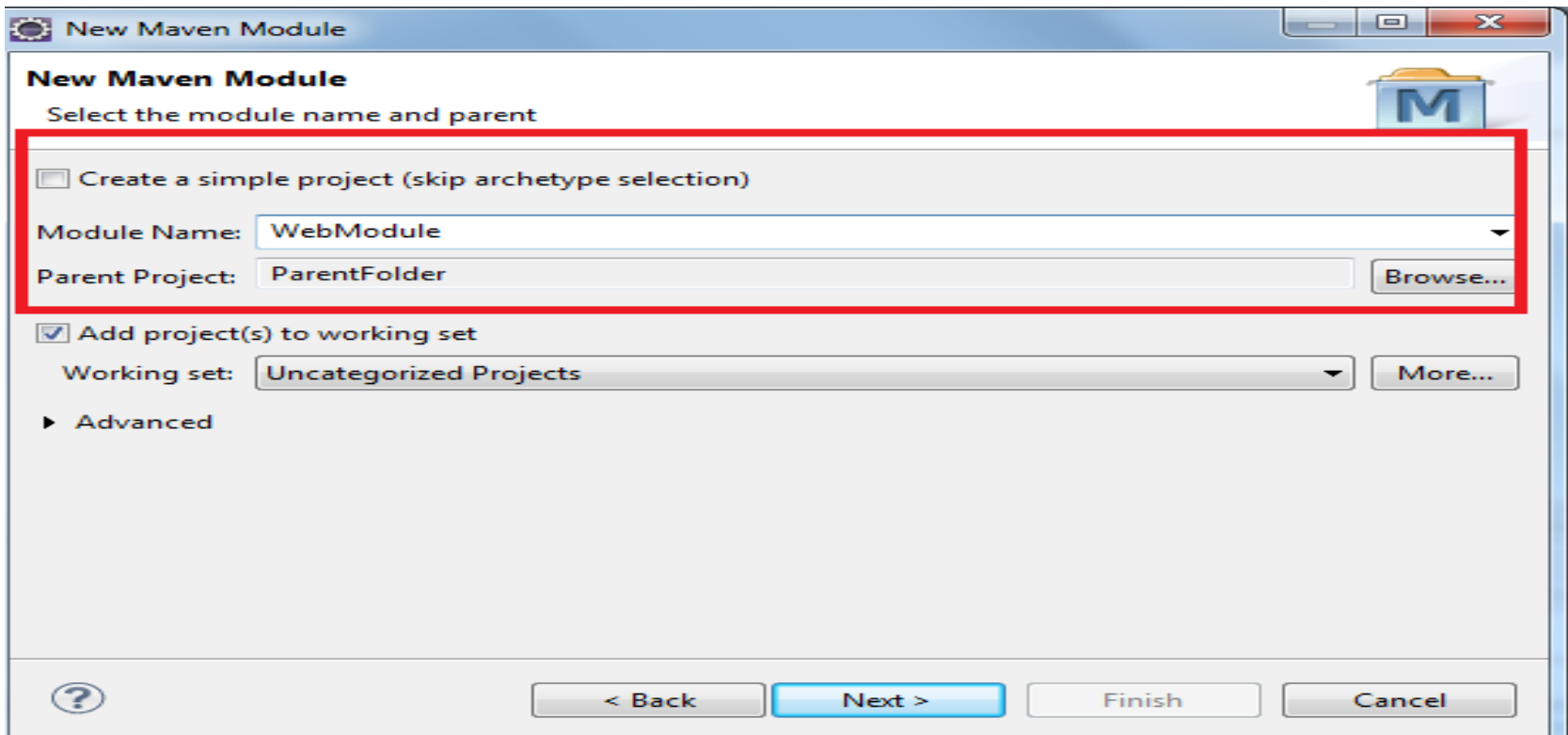
Adding Web module to parent

- Right click on ParentFolder -> New -> Other-> Maven Module.
- Then Click Next.



Adding Web module to parent contd...

- Enter the Webmodule name in **ModuleName** Tab.
- Uncheck **Create a simple project** check box and click **Next**.



New Maven Module
Select the module name and parent

☐ Create a simple project (skip archetype selection)

Module Name:

Parent Project:

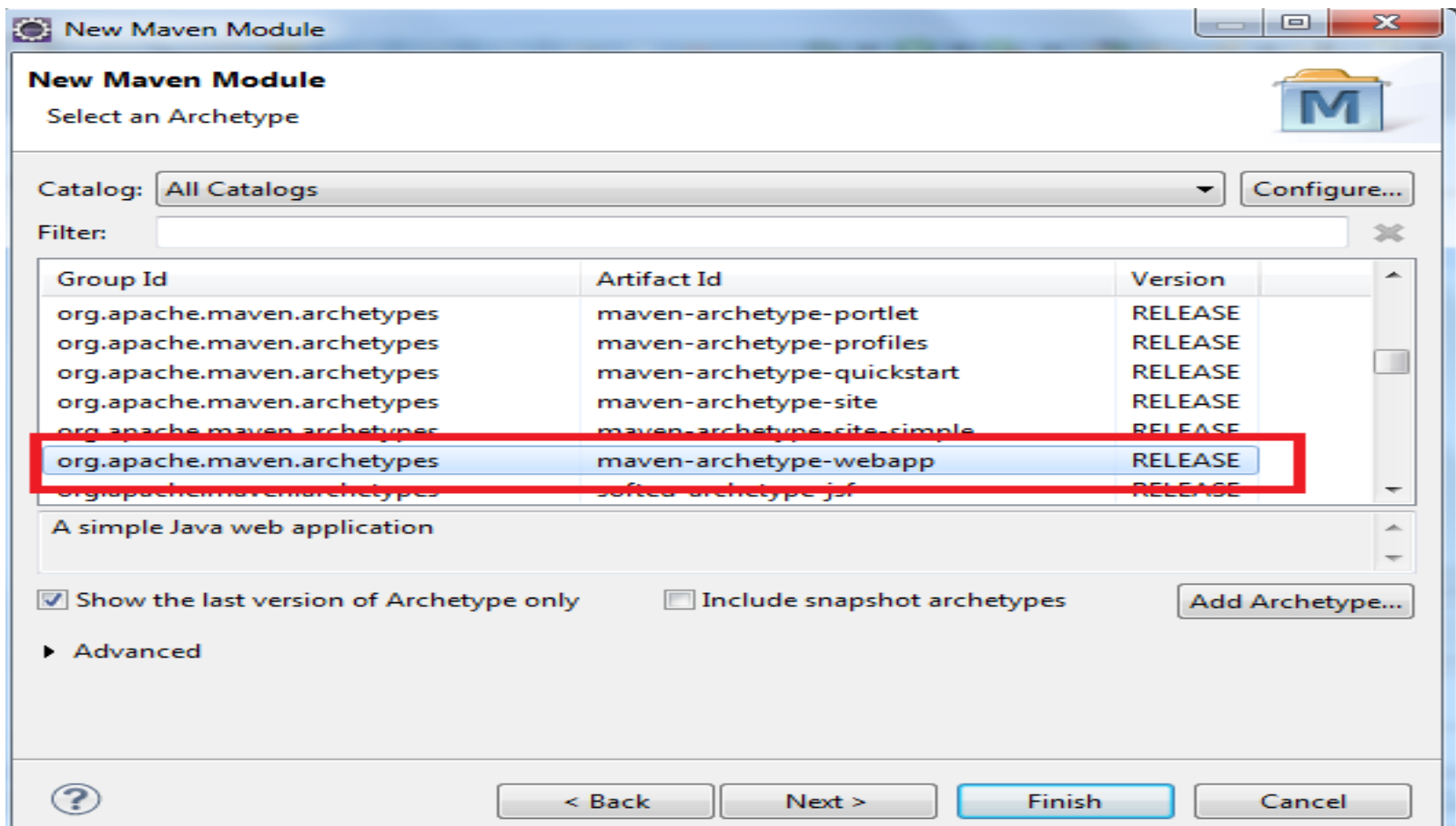
☒ Add project(s) to working set

Working set:

► Advanced

Adding Web module to parent contd...

- In the New Maven Module select the maven-archetype-webapp and click Next



Adding Web module to parent contd...

- Verify the details and click Finish

New Maven Module
Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

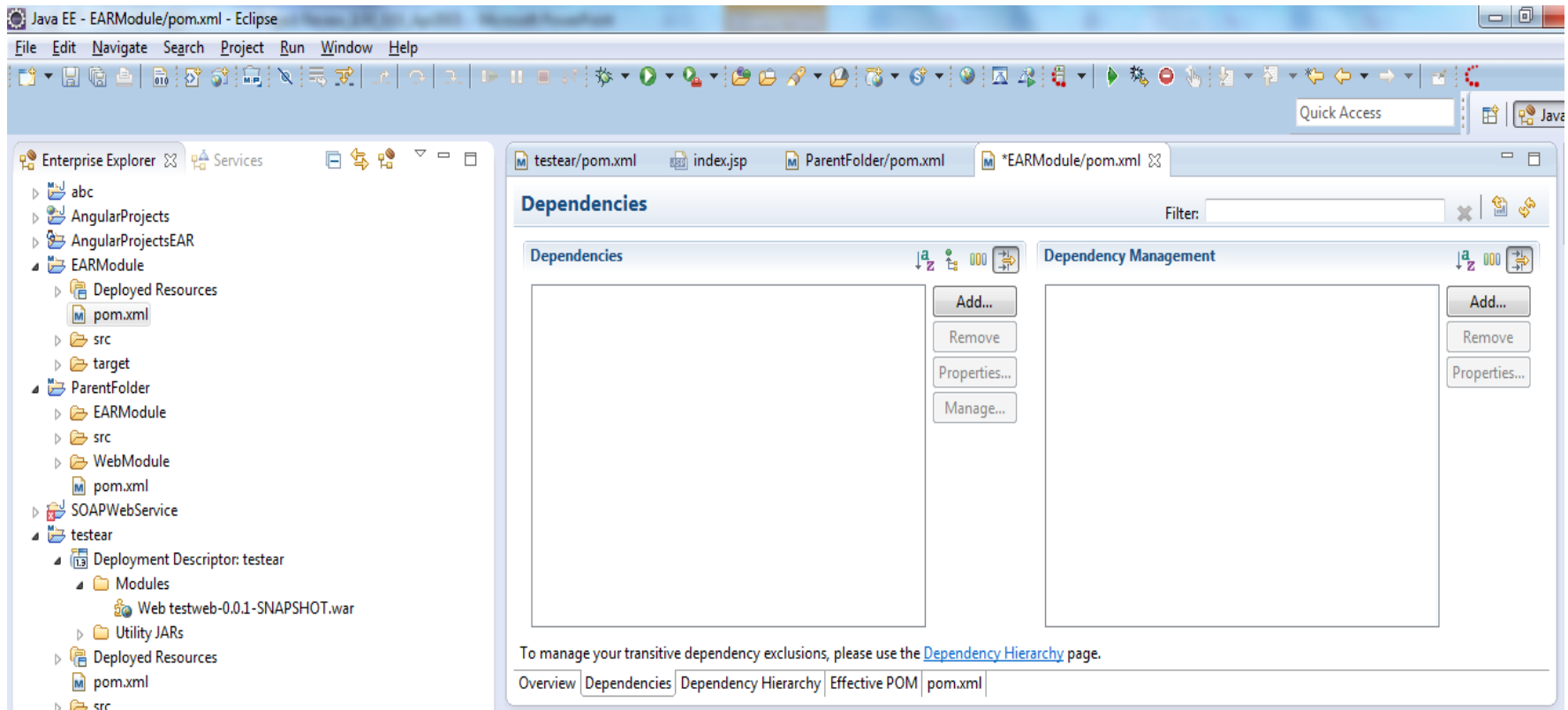
Properties available from archetype:

Name	Value

► Advanced

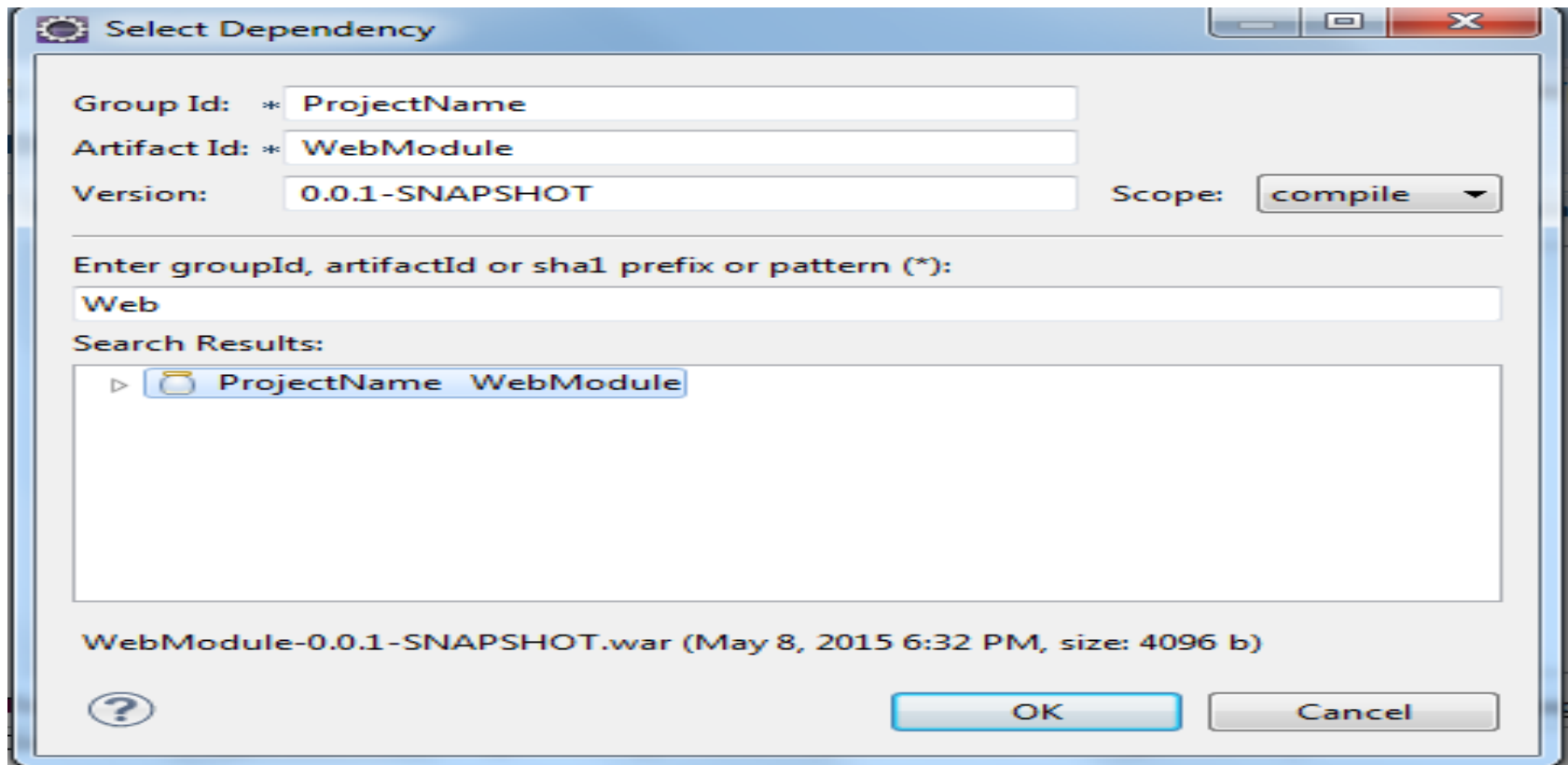
Adding war dependency to ear module

- We have to add WAR dependencies to EAR module. To add dependencies open pom.xml from EAR module



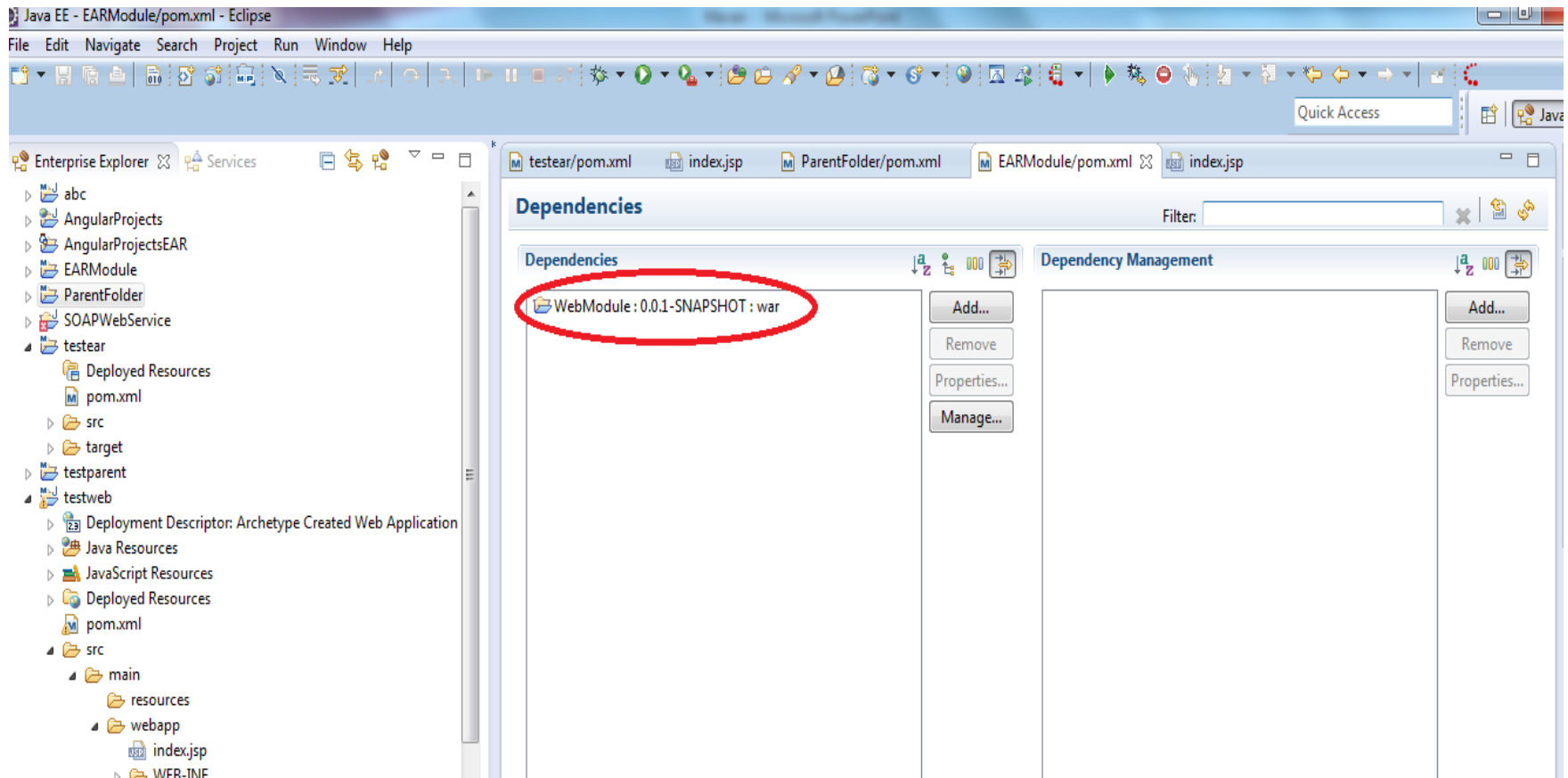
Adding war dependency to ear module contd...

- We have to enter the webmodule name in the Enter groupId: textbox
- In Search results we can find the web module to be added and click **OK**



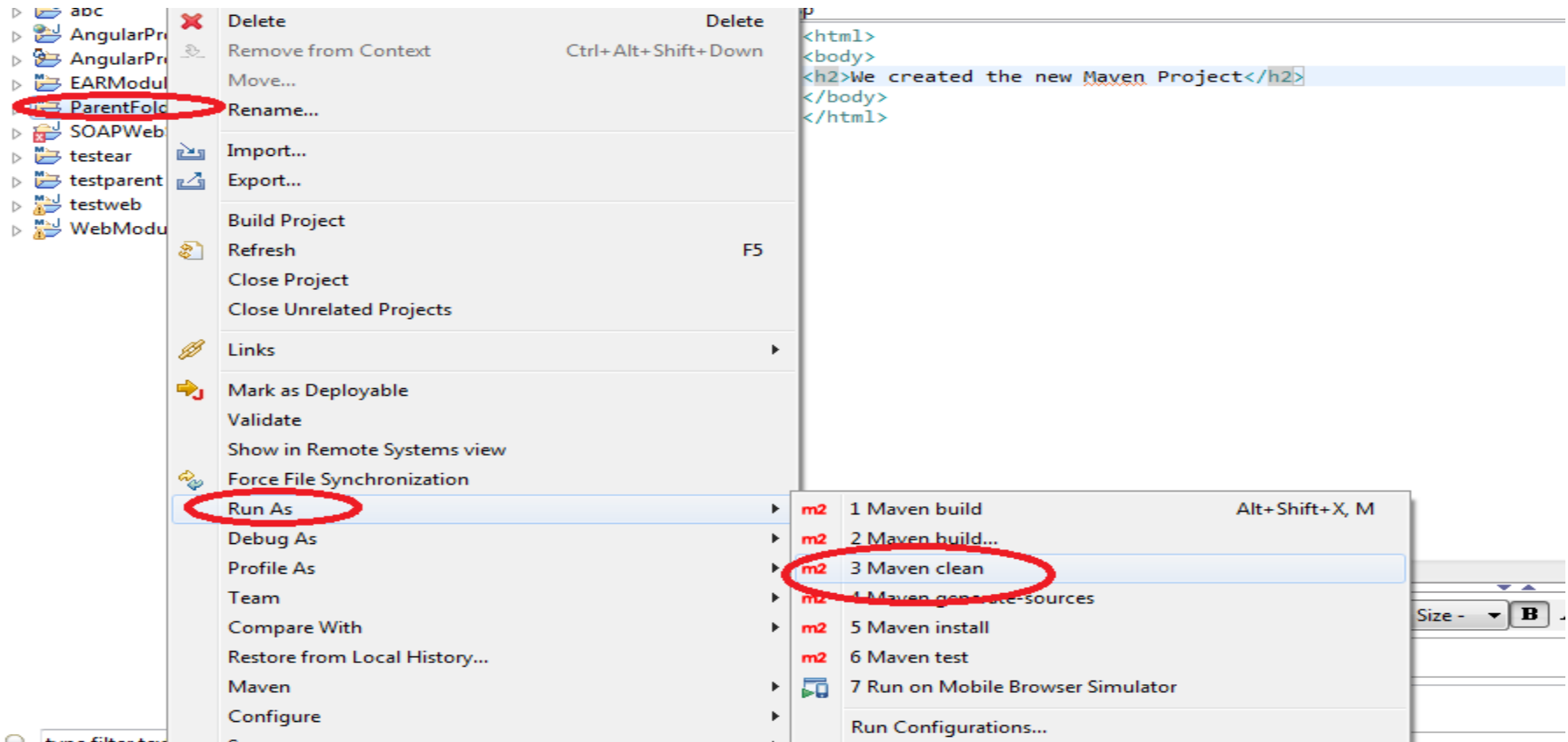
Adding war dependency to ear module contd...

- Now the war is added as a dependency to ear



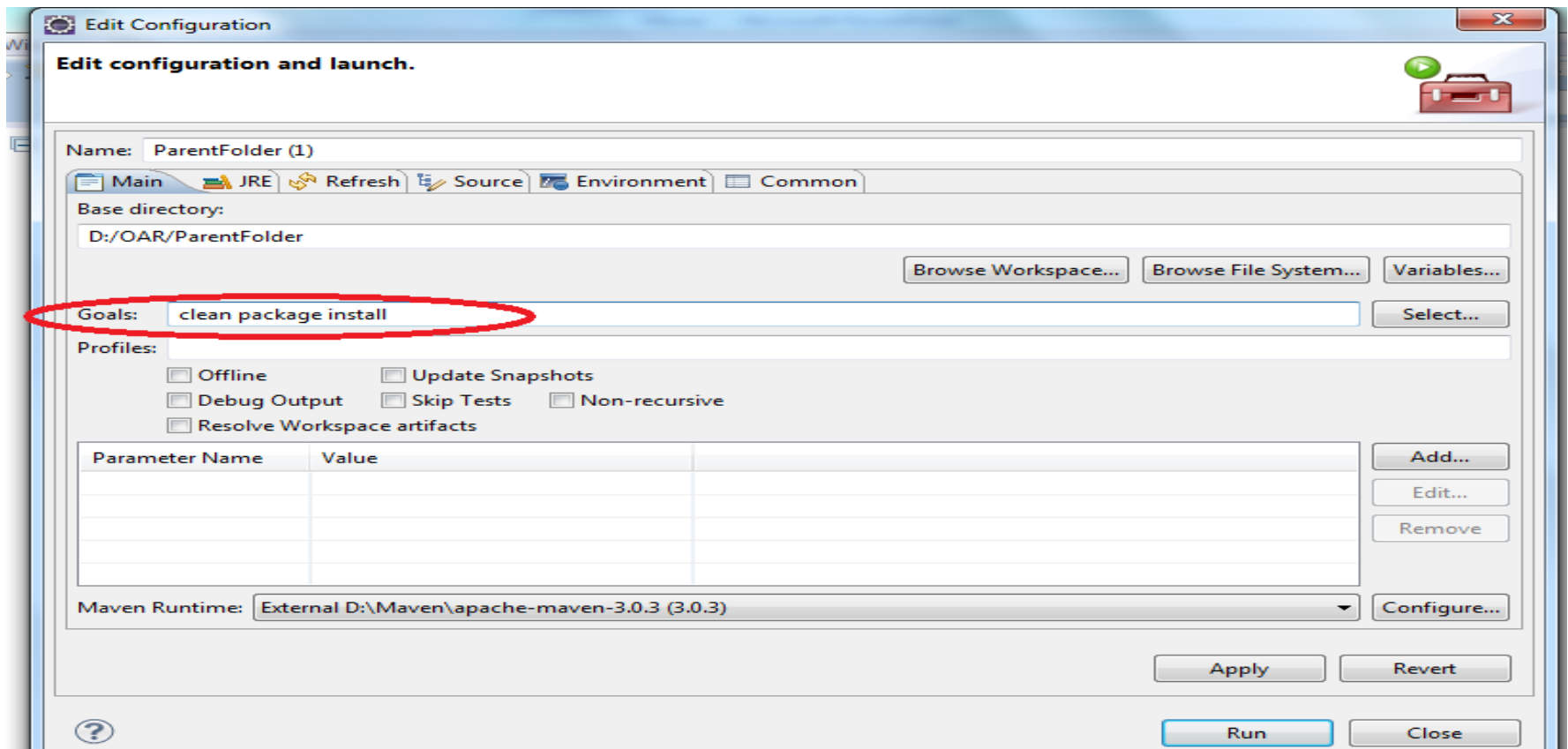
Maven Clean

- Right Click on ParentFolder -> Run As -> Maven clean



Maven Build

- After Maven Clean, Right click on ParentFolder -> Run As -> Maven Build...
- Set the goal as **clean package install** and then click Apply
- Then Apply Run



Maven Build Profiles

Build profile is a set of configuration values which can be used to set or override default values of Maven build.

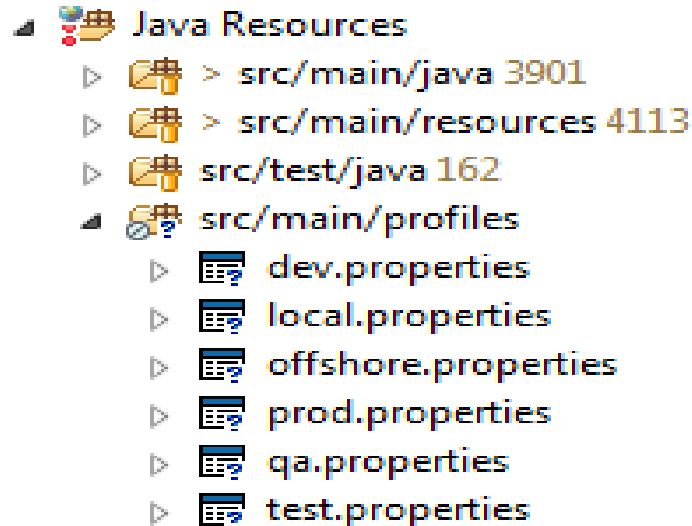
- Using a build profile, you can customize build for different environments such as Production v/s Development environments
- Profiles are specified in pom.xml file.
- Profiles modify the POM at build time, and are used to give parameters different target environments.

Build Profile Activation types

Build profile can be activated in different types.

- Explicitly using command console input.
- Through maven settings.
- Based on environment variables (User/System variables).
- OS Settings (for example, Windows family)

Profile Activation samples



- 1 . To build application for development environment **local** (Developers)
- 2. To build application for **offshore** environment (TechM server for testers and local access)
- 3. To build application for **dev** environment (Customer Development)
- 4. To build application for **test** environment (Customer Test)
- 5. To build application for **qa** environment (Customer UAT)
- 6. To build application for **prod** environment (Customer Prod)

Profile Activation samples

➤ Profile activation via Environment variables

- local (Developers)
`maven clean package install -P local`
- offshore Test environment
`maven clean package install -P offshore-dev`
- Customer development environment
`maven clean package install -P dev`
- Customer test environment
`maven clean package install -P test`
- Customer UAT environment
`maven clean package install -P qa`
- Production environment
`maven clean package install -P prod`

```
<!-- The configuration of the local profile to TechM Server -->
<profile>
  <id>local</id>
  <properties>
    <environment>local</environment>
  </properties>
</profile>

<!-- The configuration of the offshore dev profile for deployment to localhost, -->
<profile>
  <id>offshore-dev</id>
  <properties>
    <environment>offshore</environment>
  </properties>
</profile>

<!-- The configuration of the Customer QA profile -->
<profile>
  <id>qa</id>
  <properties>
    <environment>qa</environment>
  </properties>
</profile>

<!-- The configuration of the production profile -->
<profile>
  <id>prod</id>
  <properties>
    <environment>prod</environment>
  </properties>
</profile>
</profiles>
```

Profile Activation samples

➤ Profile activation via Operating System

- Customer test environment
maven clean package install -P test

```
<profile>
  <id>test</id>
  <activation>
    <os>
      <name>Windows XP</name>
      <family>Windows</family>
      <arch>x86</arch>
      <version>5.1.2600</version>
    </os>
  </activation>
</profile>
```

➤ Profile activation via Maven 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">

  <activeProfiles>
    <activeProfile>alwaysActiveProfile</activeProfile>
    <activeProfile>anotherAlwaysActiveProfile</activeProfile>
  </activeProfiles>
```

Reference links

<https://maven.apache.org/download.cgi>

<https://maven.apache.org/settings.html>

<http://maven.apache.org/guides/>

<http://www.tutorialspoint.com/maven/>

Questions?

Thank You

Disclaimer

Tech Mahindra Limited, herein referred to as TechM provide a wide array of presentations and reports, with the contributions of various professionals. These presentations and reports are for informational purposes and private circulation only and do not constitute an offer to buy or sell any securities mentioned therein. They do not purport to be a complete description of the markets conditions or developments referred to in the material. While utmost care has been taken in preparing the above, we claim no responsibility for their accuracy. We shall not be liable for any direct or indirect losses arising from the use thereof and the viewers are requested to use the information contained herein at their own risk. These presentations and reports should not be reproduced, re-circulated, published in any media, website or otherwise, in any form or manner, in part or as a whole, without the express consent in writing of TechM or its subsidiaries. Any unauthorized use, disclosure or public dissemination of information contained herein is prohibited. Unless specifically noted, TechM is not responsible for the content of these presentations and/or the opinions of the presenters. Individual situations and local practices and standards may vary, so viewers and others utilizing information contained within a presentation are free to adopt differing standards and approaches as they see fit. You may not repackage or sell the presentation. Products and names mentioned in materials or presentations are the property of their respective owners and the mention of them does not constitute an endorsement by TechM. Information contained in a presentation hosted or promoted by TechM is provided “as is” without warranty of any kind, either expressed or implied, including any warranty of merchantability or fitness for a particular purpose. TechM assumes no liability or responsibility for the contents of a presentation or the opinions expressed by the presenters. All expressions of opinion are subject to change without notice.