OBJECT™ TECHNOLOGIES

Introduction to Maven

What Will Be Covered

- What is build tool
- What is Maven
- Installing and using Maven
- Benefits of Maven
- Build life cycle, phase and goal
- What is managed by Maven
- Convention over configuration
- Features of Maven
- Maven POM
- Maven directory structure
- Project dependencies
- Maven repositories

What is build tool

OBJECT TECHNOLOGIES

- A build tool is a tool that automates everything related to building the software project. Building a software project typically includes one or more of these activities:
  - Generating source code (if auto-generated code is used in the project).
  - Generating documentation from the source code.
  - Compiling source code.
  - Packaging compiled code into JAR files or ZIP files.
  - Installing the packaged code on a server, in a repository or somewhere else.
- Any given software project may have more activities than these needed to build the finished software. Such activities can normally be plugged into a build tool, so these activities can be automated too.
- The advantage of automating the build process is to minimize the risk of humans making errors while building the software manually. Additionally, an automated build tool is typically faster than a human performing the same steps manually.

What is Maven

OBJECT

- Maven tool is developed by Apache Software Foundation
- It is written in java language and used to build and manage projects in C#, scala and other languages
- Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework.
- It helps to build projects, dependency and documentation
- Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.
- Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration etc

Installing and using Maven

OBJECT

Step 1 :Verify java installation

Open console and execute the following java command

Step 2 : Set JAVA Environment

Set the JAVA_HOME environment variable to point to the base directory location where Java is installed on your machine.

Step 3 : Download Maven Archive Download Maven 2.2.1 from http://maven.apache.org/download.html

Step 4 : Extract the Maven Archive Extract the archive, to the directory you wish to install Maven 3.3.1. The subdirectory apache-maven-3.3.1 will be created from the archive

Step 5 : Set Maven Environment Variable

Set maven home to the base Maven directory

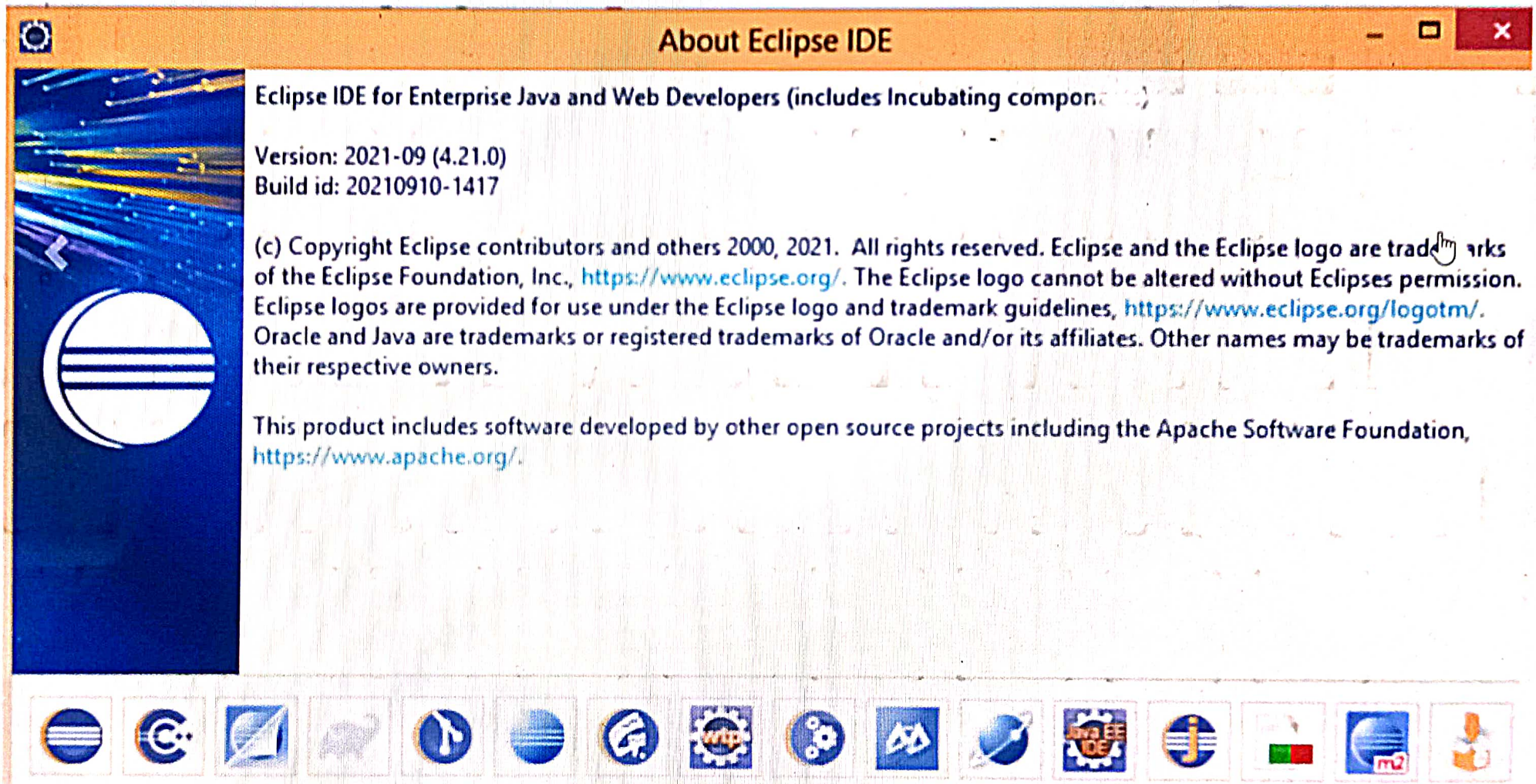
Step 6 : Verify Maven Installation

open console and execute the following mvn command.

Installing and using Maven

OBJECT TECHNOLOGIES

- Most Eclipse IDE downloads already include support for the Maven build system. It can be verified in Help-> About and check if you can see the Maven logo (with the M2E) sign.



Benefits of Maven

OBJECT TECHNOLOGIES

- **simple project setup** that follows best practices: Maven tries to avoid as much configuration as possible, by supplying project templates (named archetypes)
- **dependency management**: it includes automatic updating, downloading and validating the compatibility, as well as reporting the dependency closures (known also as transitive dependencies)
- **isolation between project dependencies and plugins**: with Maven, project dependencies are retrieved from the dependency repositories while any plugin's dependencies are retrieved from the plugin repositories, resulting in fewer conflicts when plugins start to download additional dependencies
- **central repository system**: project dependencies can be loaded from the local file system or public repositories, such as Maven Central

Build Life Cycles, Phases and Goals

OBJECT TECHNOLOGIES

- The build process in Maven is split up into build life cycles, phases and goals.
- A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals.
- When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal.
- If a life cycle is requested executed, all build phases in that life cycle are executed. If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too
- The following list shows the most important Maven lifecycle phases:
  - validate – checks the correctness of the project
  - compile – compiles the provided source code into binary artifacts
  - test – executes unit tests
  - package – packages compiled code into an archive file
  - integration-test – executes additional tests, which require the packaging
  - verify – checks if the package is valid
  - install – installs the package file into the local Maven repository
  - deploy – deploys the package file to a remote server or repository.

Build Life Cycles, Phases and Goals

OBJECT TECHNOLOGIES

- Maven has 3 built-in build life cycles. These are:
  - default
  - clean
  - site
- **The default life cycle** handles everything related to compiling and packaging your project.
- **The clean life cycle** handles everything related to removing temporary files from the output directory, including generated source files, compiled classes, previous JAR files etc.
- **The site life cycle** handles everything related to generating documentation for your project. In fact, site can generate a complete website with documentation for your project.
- You can execute either a whole build life cycle like clean or site, a build phase like install which is part of the default build life cycle.
- Executing the install build phase really means executing all build phases before the install phase, and then execute the install phase after that.

What is managed by Maven

OBJECT TECHNOLOGIES

Maven provides developers ways to manage the following:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution
- Mailing list

Convention over Configuration

OBJECT TECHNOLOGIES

- Maven uses Convention over Configuration, which means developers are not required to create build process
- Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure.
- Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.
- ${basedir} a project location

Item	Default
source code	${basedir}/src/main/java
Resources	{basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

Features of Maven

OBJECT

- Simple project setup that follows best practices.
- Consistent usage across all projects.
- Dependency management including automatic updating.
- Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation
- Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.

Maven POM

OBJECT

- POM stands for Project Object Model.
- It is an XML file that resides in the base directory of the project as pom.xml.
- The POM contains information about the project and various configuration detail used by Maven to build the project(s). POM also contains the goals and plugins.
- While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.
- Some of the configuration that can be specified in the POM are following -
  - project dependencies
  - plugins
  - goals
  - build profiles
  - project version
  - developers
  - mailing list
- Before creating a POM, we should first decide the project **group**(groupId), its **name** (artifactId) and its version as these attributes help in uniquely identifying the project in repository

Maven POM

OBJECT TECHNOLOGIES

```
<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>com.companyname.project-group</groupId>
    <artifactId>project</artifactId>
    <version>1.0</version>
</project>
```

- It should be noted that there should be a single POM file for each project.
- All POM files require the **project** element and three mandatory fields: **groupId, artifactId, version.**
- Projects notation in repository is **groupId:artifactId:version.**

Maven directory structure

OBJECT

- The Maven directory structure is a standard directory structure which Maven expects to build a project.

```
myprojectdir
  - pom.xml

  - .mvn
    - jvm.config

  - src
    - main
      - java
      - resources
    - test
      - java
      - resources

  - target
```

- The myprojectdir is the root dir for your project.
- The src directory is the root directory for both application and test source code.
- The application source code goes into src/main/java . Any resource files (e.g property files) needed by your application goes into the src/main/resources directories.
- The target directory contains all the final products that are the result of Maven building your project.

Project dependencies

OBJECT TECHNOLOGY

- Project often needs external Java APIs or frameworks which are packaged in their own JAR files. These JAR files are needed on the classpath when you compile your project code.
- Keeping the project up-to-date with the correct versions of these external JAR files can be a comprehensive task. Each external JAR may again also need other external JAR files etc.
- Downloading all these external dependencies (JAR files) recursively and making sure that the right versions are downloaded is managed easily by maven.
- Information about external libraries needed by your project and its version is mentioned POM file, then Maven downloads them for you and puts them in your local Maven repository.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.8.1</version>
  <scope>test</scope>
</dependency>
```

Maven repositories

OBJECT

- **Maven repositories are directories of packaged JAR files with extra meta data.**
- **Maven has three types of repository:**
  - **Local repository**
  - **Central repository**
  - **Remote repository**
- **Maven searches these repositories for dependencies in the above sequence. First in the local repository, then in the central repository, and third in remote repositories if specified in the POM.**
- **Local Repository**
- A local repository is a directory on the developer's computer. This repository will contain all the dependencies Maven downloads. The same Maven repository is typically used for several different projects. Thus Maven only needs to download the dependencies once, even if multiple projects depends on them
- **Central Repository**
- The central Maven repository is a repository provided by the Maven community. By default Maven looks in this central repository for any dependencies needed but not found in your local repository.
- **Remote Repository**
- A remote repository is often used for hosting projects internal to your organization, which are shared by multiple projects

Maven Installation

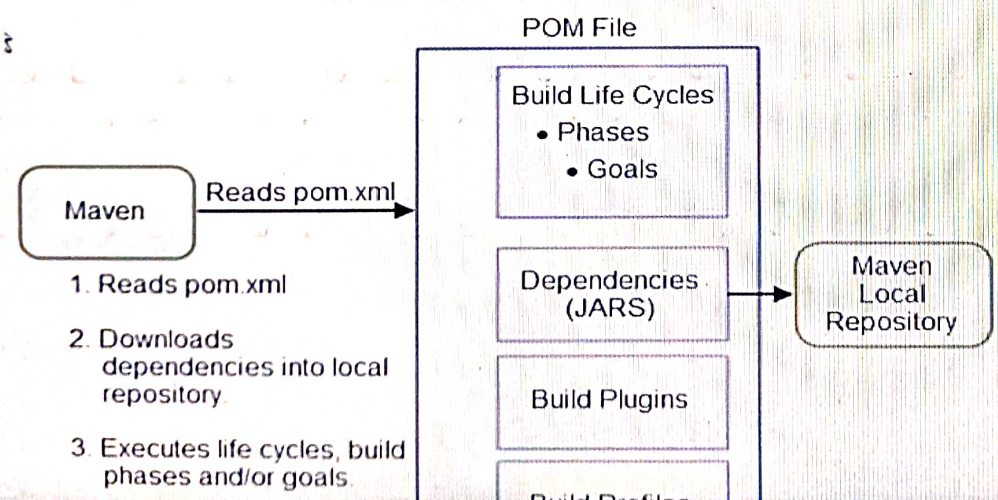
1. To install Maven on your own system (computer), go to the Maven download page and follow the instructions there. In summary, what you need to do is:
2. Set the JAVA_HOME environment variable to point to a valid Java SDK (e.g. Java 8).
3. Download and unzip Maven.
4. Set the M2_HOME environment variable to point to the directory you unzipped Maven to.
5. Set the M2 environment variable to point to M2_HOME/bin (%M2_HOME%\bin on Windows, $M2_HOME/bin on unix).
6. Add M2 to the PATH environment variable (%M2% on Windows, $M2 on unix).
7. Open a command prompt and type 'mvn -version' (without quotes) and press enter.
8. After typing in the mvn -version command you should be able to see Maven execute, and the version number of Maven written out to the command prompt.

Note: Maven uses Java when executing, so you need Java installed too (and the JAVA_HOME environment variable set as explained above). Maven 3.0.5 needs a Java version 1.5 or later. I use Maven 3.3.3 with Java 8 (u45).

Maven Overview

Maven is centered around the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

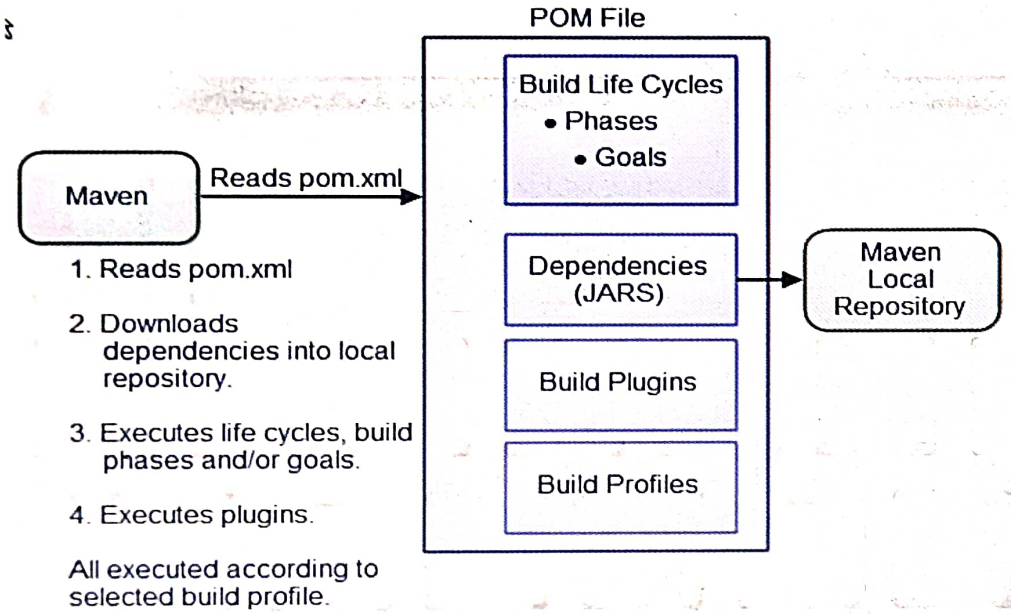
Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contains:



Maven Overview

Maven is centered around the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contains:



POM Files

When you execute a Maven command you give Maven a POM file to execute the commands on. Maven will then execute the command on the resources described in the POM.

Build Life Cycles, Phases and Goals

The build process in Maven is split up into build life cycles, phases and goals. A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal. If a life cycle is requested executed, all build phases in that life cycle are executed. If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too.

Build Life Cycles, Phases and Goals

The build process in Maven is split up into build life cycles, phases and goals. A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal. If a life cycle is requested executed, all build phases in that life cycle are executed. If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too.

Dependencies and Repositories

One of the first goals Maven executes is to check the dependencies needed by your project. Dependencies are external JAR files (Java libraries) that your project uses. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository. The local repository is just a directory on your computer's hard disk. You can specify where the local repository should be located if you want to (I do). You can also specify which remote repository to use for downloading dependencies. All this will be explained in more detail later in this tutorial.

Build Plugins

Build plugins are used to insert extra goals into a build phase. If you need to perform a set of actions for your project which are not covered by the standard Maven build phases and goals, you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java if you need to.

Maven Directory Structure

The Maven directory structure is a standard directory structure which Maven expects to build a project.

Directory Layout

A typical Maven project has a pom.xml file and a directory structure based on defined conventions:

```
maven-project
  ----pom.xml
  ----README.txt
  ----NOTICE.txt
  ----LICENSE.txt
  ----src
```

Directory Layout

A typical Maven project has a pom.xml file and a directory structure based on defined conventions:

```
maven-project
  ----pom.xml
  ----README.txt
  ----NOTICE.txt
  ----LICENSE.txt
 ----src
    ----main
        ----java
        ----resources
        ----filters
        ----webapp
    ----test
        ----java
        ----resources
    ----filters
        ----it
        ----site
        ----assembly
```

The Root Directory

This directory serves as the root of every Maven project.

Let's take a closer look at the standard files and subdirectories that are typically found at root:

1. maven-project/pom.xml - defines dependencies and modules needed during the build lifecycle of a Maven project
2. maven-project/LICENSE.txt - licensing information of the project
3. maven-project/README.txt - summary of the project
4. maven-project/NOTICE.txt - information about third-party libraries used in the project

The Root Directory

This directory serves as the root of every Maven project.

Let's take a closer look at the standard files and subdirectories that are typically found at root:

1. maven-project/pom.xml - defines dependencies and modules needed during the build lifecycle of a Maven project
2. maven-project/LICENSE.txt - licensing information of the project
3. maven-project/README.txt - summary of the project
4. maven-project/NOTICE.txt - information about third-party libraries used in the project
5. maven-project/src/main - contains source code and resources that become part of the artifact
6. maven-project/src/test - holds all the test code and resources
7. maven-project/src/it - usually reserved for integration tests used by the Maven Failsafe Plugin
8. maven-project/src/site - site documentation created using the Maven Site Plugin
9. maven-project/src/assembly - assembly configuration for packaging binaries

The src/main Directory

As the name indicates, src/main is the most important directory of a Maven project. Anything that is supposed to be part of an artifact, be it a jar or war, should be present here.

Its subdirectories are:

src/main/java ? Java source code for the artifact
src/main/resources ? configuration files and others such as i18n files, per-environment configuration files, and XML configurations
src/main/webapp ? for web applications, contains resources like JavaScript, CSS, HTML files, view templates, and images

The src/test Directory

The directory src/test is the place where tests of each component in the application reside.

Note that none of these directories or files will become part of the artifact. Let's see its subdirectories:

1. src/test/java ? Java source code for tests
2. src/test/resources ? configuration files and others used by tests

artifact, be it a jar or war, should be present here.

Its subdirectories are:

src/main/java ? Java source code for the artifact
src/main/resources ? configuration files and others such as i18n files, per-environment configuration files, and XML configurations
src/main/webapp ? for web applications, contains resources like JavaScript, CSS, HTML files, view templates, and images

The src/test Directory

The directory src/test is the place where tests of each component in the application reside.

Note that none of these directories or files will become part of the artifact. Let's see its subdirectories:

1. src/test/java ? Java source code for tests
2. src/test/resources ? configuration files and others used by tests

The webapp directory contains Java web application, if the project is a web application.

The webapp directory is the root directory of the web application. The webapp directory contains the WEB-INF directory.

Assignments

1. Develop a simple application using maven and add the dependency of mysql-connector jar file. See that jar file in the local repository.
2. Develop a web application using maven and add the dependency of jstl and see that jar added in the local repository. Use core tag from JSTL in the JSP and execute it

Important Questions for: Maven Build Tool

Interview Questions

Explain Maven