What Maven Provides

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

What POM Contains

POM stands for Project Object Model. It is fundamental unit of work in Maven. 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

What is Build Lifecycle?

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical **Maven Build Lifecycle** consists of the following sequence of phases.

|  |  |  |
| --- | --- | --- |
| **Phase** | **Handles** | **Description** |
| prepare-resources | resource copying | Resource copying can be customized in this phase. |
| validate | Validating the information | Validates if the project is correct and if all necessary information is available. |
| compile | compilation | Source code compilation is done in this phase. |
| Test | Testing | Tests the compiled source code suitable for testing framework. |
| package | packaging | This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml. |
| install | installation | This phase installs the package in local/remote maven repository. |
| Deploy | Deploying | Copies the final package to the remote repository. |

There are always **pre** and **post** phases to register **goals**, which must run prior to, or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.

Maven has the following three standard lifecycles −

* clean
* default(or build)
* site

A **goal** represents a specific task which contributes to the building and managing of a project. It may be bound to zero or more build phases. A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The **clean**and **package** arguments are build phases while the **dependency:copy-dependencies** is a goal.

## What is SNAPSHOT?

SNAPSHOT is a special version that indicates a current development copy. Unlike regular versions, Maven checks for a new SNAPSHOT version in a remote repository for every build.

Now data-service team will release SNAPSHOT of its updated code every time to repository, say data-service: 1.0-SNAPSHOT, replacing an older SNAPSHOT jar.

## Snapshot vs Version

In case of Version, if Maven once downloaded the mentioned version, say data-service:1.0, it will never try to download a newer 1.0 available in repository. To download the updated code, data-service version is be upgraded to 1.1.

In case of SNAPSHOT, Maven will automatically fetch the latest SNAPSHOT (data-service:1.0-SNAPSHOT) every time app-ui team build their project.

## app-ui pom.xml

**app-ui** project is using 1.0-SNAPSHOT of data-service.

<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>app-ui</groupId>

<artifactId>app-ui</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<name>health</name>

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

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>data-service</groupId>

<artifactId>data-service</artifactId>

<version>1.0-SNAPSHOT</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

## data-service pom.xml

**data-service** project is releasing 1.0-SNAPSHOT for every minor change.

<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>data-service</groupId>

<artifactId>data-service</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>health</name>

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

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

</project>

Dependency Order

<groupId>com.companyname.project-group</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<scope>test</scope>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet-core</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

</dependency>

<dependency>

<groupId>org.jmockit</groupId>

<artifactId>jmockit</artifactId>

<version>1.33</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.jdbi</groupId>

<artifactId>jdbi</artifactId>

<version>2.73</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.39</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.12</version>

</dependency>

</dependencies>

What is a Maven Repository?

In Maven terminology, a repository is a 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. The following illustration will give an idea regarding these three types.

* local
* central
* remote

## Local Repository

Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time.

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. It helps to avoid references to dependencies stored on remote machine every time a project is build.

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 available at %M2\_HOME%\conf directory.

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 following URL − <https://repo1.maven.org/maven2/>

Key concepts of Central repository are as follows −

* This repository is managed by Maven community.
* It is not required to be configured.
* It requires internet access to be searched.

## Remote Repository

Sometimes, Maven does not find a mentioned dependency in central repository as well. It then stops the 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.

For example, using below mentioned POM.xml, Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.xml.

## Maven Dependency Search Sequence

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence −

* **Step 1** − Search dependency in local repository, if not found, move to step 2 else perform the further processing.
* **Step 2** − Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4. Else it is downloaded to local repository for future reference.
* **Step 3** − If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
* **Step 4** − Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference. Otherwise, Maven stops processing and throws error (Unable to find dependency).

**MVN CLEAN : Cleans all the artifacts from the repository.**

## REST API HTTP Status Codes and Error Messages Reference

This section provides information about the status codes and error messages that can be received by a REST API client of Oracle Messaging Cloud Service.

**Topics:**

* [Generic Meanings of HTTP Response Status Codes](https://docs.oracle.com/en/cloud/iaas/messaging-cloud/csmes/rest-api-http-status-codes-and-error-messages-reference.html#GUID-8BC6BDC0-4C08-4229-98EA-BBFAA24A923A)
* [Error Key, Status Codes and Error Messages](https://docs.oracle.com/en/cloud/iaas/messaging-cloud/csmes/rest-api-http-status-codes-and-error-messages-reference.html#GUID-F3506024-AF35-4020-98B7-AAEC6C2AC73F)

### **Generic Meanings of HTTP Response Status Codes**

The following table lists HTTP response status codes and their meanings:

| **Response Status Code** | **Meaning** |
| --- | --- |
| 200 Ok | Successful requests other than creations and deletions. |
| 201 Created | Successful creation of a queue, topic, temporary queue, temporary topic, session, producer, consumer, listener, queue browser, or message. |
| 204 No Content | Successful deletion of a queue, topic, session, producer, or listener. |
| 400 Bad Request | The path info doesn't have the right format, or a parameter or request body value doesn't have the right format, or a required parameter is missing, or values have the right format but are invalid in some way (for example, destination parameter does not exist, content is too big, or client ID is in use). |
| 403 Forbidden | The invoker is not authorized to invoke the operation. |
| 404 Not Found | The object referenced by the path does not exist. |
| 405 Method Not Allowed | The method is not one of those allowed for the path. |
| 409 Conflict | An attempt was made to create an object that already exists. |
| 500 Internal Server Error | The execution of the service failed in some way. |

# **JUnit Tutorial | Testing Framework for Java**

JUnit tutorial provides basic and advanced concepts of **unit testing in java** with examples. Our junit tutorial is designed for beginners and professionals.

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

The **org.junit** package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

## Types of unit testing

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

#### **1) Manual Testing**

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

#### **2) Automated Testing**

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

#### **Annotations for Junit testing**

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

**@Test** annotation specifies that method is the test method.

**@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

**@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.

**@Before** annotation specifies that method will be invoked before each test.

**@After** annotation specifies that method will be invoked after each test.

**@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests.

## Assert class

The org.junit.Assert class provides methods to assert the program logic.

#### Methods of Assert class

The common methods of Assert class are as follows:

1. **void assertEquals(boolean expected,boolean actual)**: checks that two primitives/objects are equal. It is overloaded.
2. **void assertTrue(boolean condition)**: checks that a condition is true.
3. **void assertFalse(boolean condition)**: checks that a condition is false.
4. **void assertNull(Object obj)**: checks that object is null.
5. **void assertNotNull(Object obj)**: checks that object is not null.

## What is REST architecture?

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

### **HTTP methods**

Following four HTTP methods are commonly used in REST based architecture.

* **GET** − Provides a read only access to a resource.
* **POST** − Used to create a new resource.
* **DELETE** − Used to remove a resource.
* **PUT** − Used to update a existing resource or create a new resource.

## Introduction to RESTFul web services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

@Path("/Emp")

public class EmployRest {

@POST

@Path("/employInsert/")

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public String insertEmploy(final Employ e) {

String s =EmployFactory.addEmploy(e);

System.out.println(s);

return s;

}

@GET

@Path("{id}")

@Produces(MediaType.APPLICATION\_JSON)

public Employ employeeListById(@PathParam("id") int id) {

Employ empl = EmployFactory.showEmploy(id);

if (empl == null) {

throw new NotFoundException("No such Employee ID: " + id);

}

return empl;

}

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Employ[] employeesList() {

Employ[] employees = EmployFactory.showEmploy();

return employees;

}

package com.hexaware.trainer;

import org.junit.Test;

import static org.junit.Assert.assertNull;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertTrue;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertEquals;

import org.junit.BeforeClass;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.After;

import static org.junit.Assert.assertNotEquals;

public class PracticeTest {

static Practice obj;

@BeforeClass

public static void setUpBeforeClass() throws Exception {

obj = new Practice();

}

@Before

public void setUp() throws Exception {

System.out.println("Test Case ");

}

@After

public void tearDown() throws Exception {

System.out.println("Test Case Ended");

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

obj = null;

System.out.println("Program Ended...");

}

@Test

public void testEvenOdd() {

assertTrue(obj.evenodd(4));

assertFalse(obj.evenodd(5));

}

@Test

public void testMax3() {

assertEquals(5, obj.max3(2,3,5));

assertEquals(5, obj.max3(2,5,3));

assertEquals(5, obj.max3(5,3,2));

}

}