PROGRAM NO: 1

1. Build and Run a Java Application with Maven, Migrate the Same Application to Gradle.

- **Introduction to Maven and Gradle: Overview of Build Automation Tools, Key Differences between Maven and Gradle, Installation and Setup.**

Introduction to Maven and Gradle

- **Overview of Build Automation Tools**

Build automation tools help developers streamline the process of building, testing, and deploying software projects. They take care of repetitive tasks like compiling code, managing dependencies, and packaging applications, which make development more efficient and error-free.

Two popular tools in the Java ecosystem are **Maven** and **Gradle**. Both are great for managing project builds and dependencies, but they have some key differences.

Maven

- **What is Maven?** Maven is a build automation tool primarily used for Java projects. It uses an XML configuration file called pom.xml (Project Object Model) to define project settings, dependencies, and build steps.
- **Main Features:**
  - Predefined project structure and lifecycle phases.
  - Automatic dependency management through Maven Central.
  - Wide range of plugins for things like testing and deployment.
  - Supports complex projects with multiple modules.

Gradle

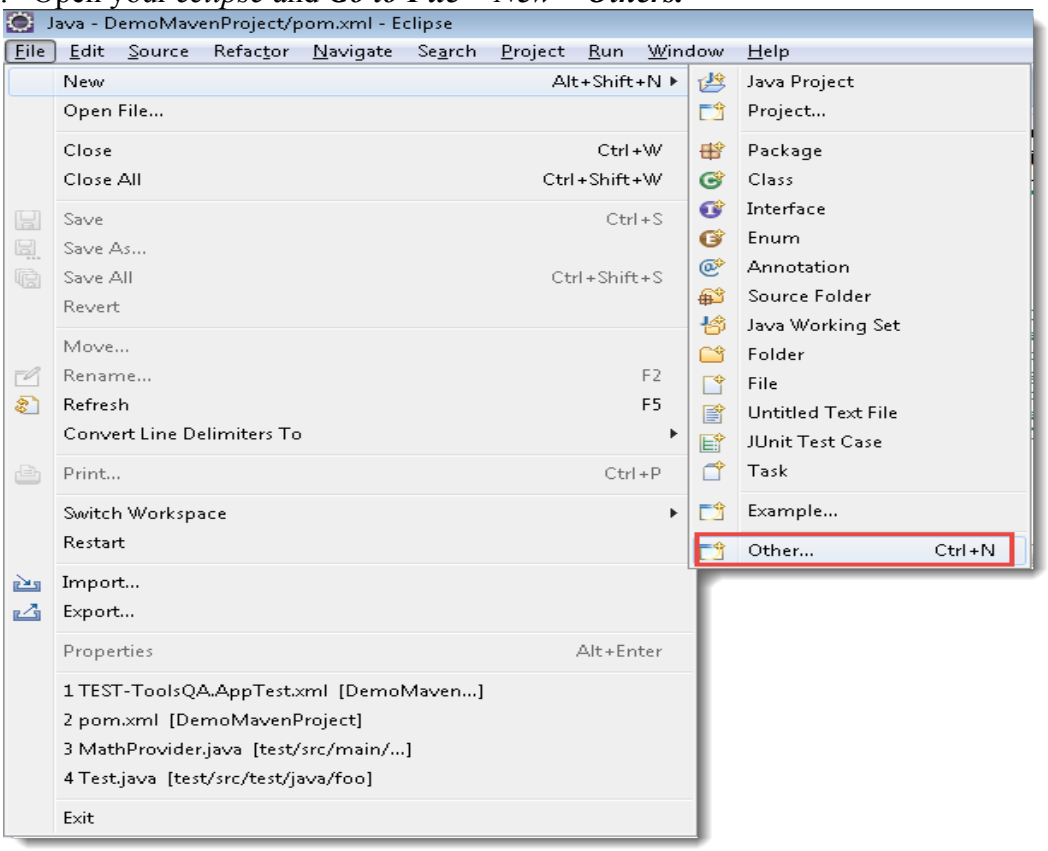
- **What is Gradle?** Gradle is a more modern and versatile build tool that supports multiple programming languages, including Java, Groovy, and Kotlin. It uses a domain-specific language (DSL) for build scripts, written in Groovy or Kotlin.
- **Main Features:**
  - Faster builds thanks to task caching and incremental builds.
  - Flexible and customizable build scripts.
  - Works with Maven repositories for dependency management.
  - Excellent support for multi-module and cross-language projects.
  - Integrates easily with CI/CD pipelines.

Key Differences Between Maven and Gradle

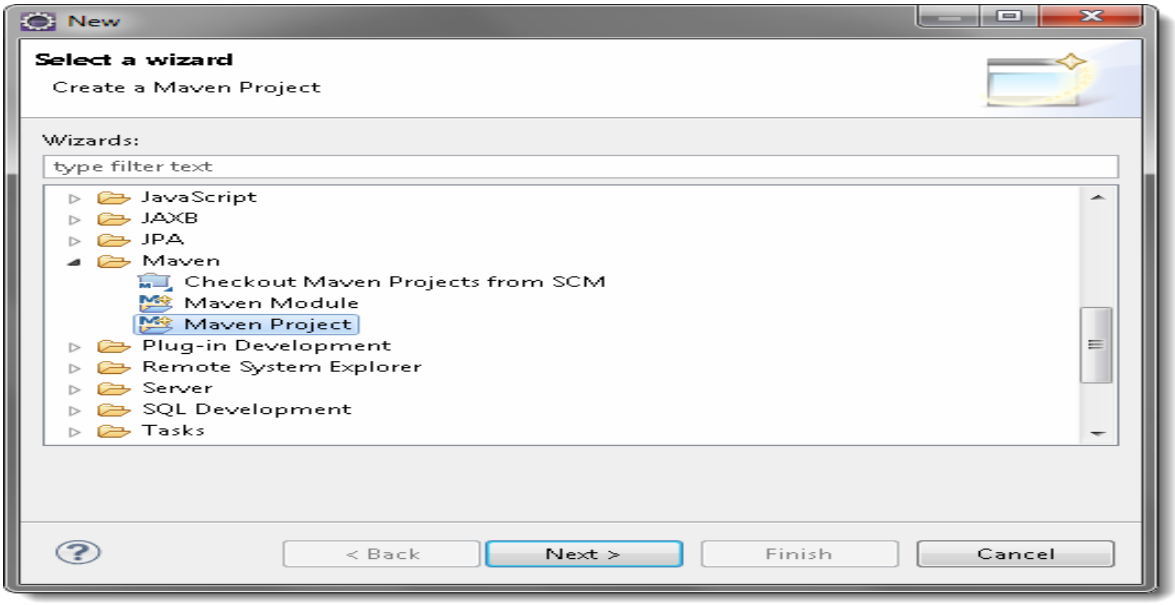
Aspect	Maven	Gradle
Configuration	XML (pom.xml)	Groovy or Kotlin DSL
Performance	Slower	Faster due to caching
Flexibility	Less flexible	Highly customizable
Learning Curve	Easier to pick up	Slightly steeper
Script Size	Verbose	More concise
Dependency Management	Uses Maven Central	Compatible with Maven too
Plugin Support	Large ecosystem	Extensible and versatile

Create a New Maven Project in Eclipse

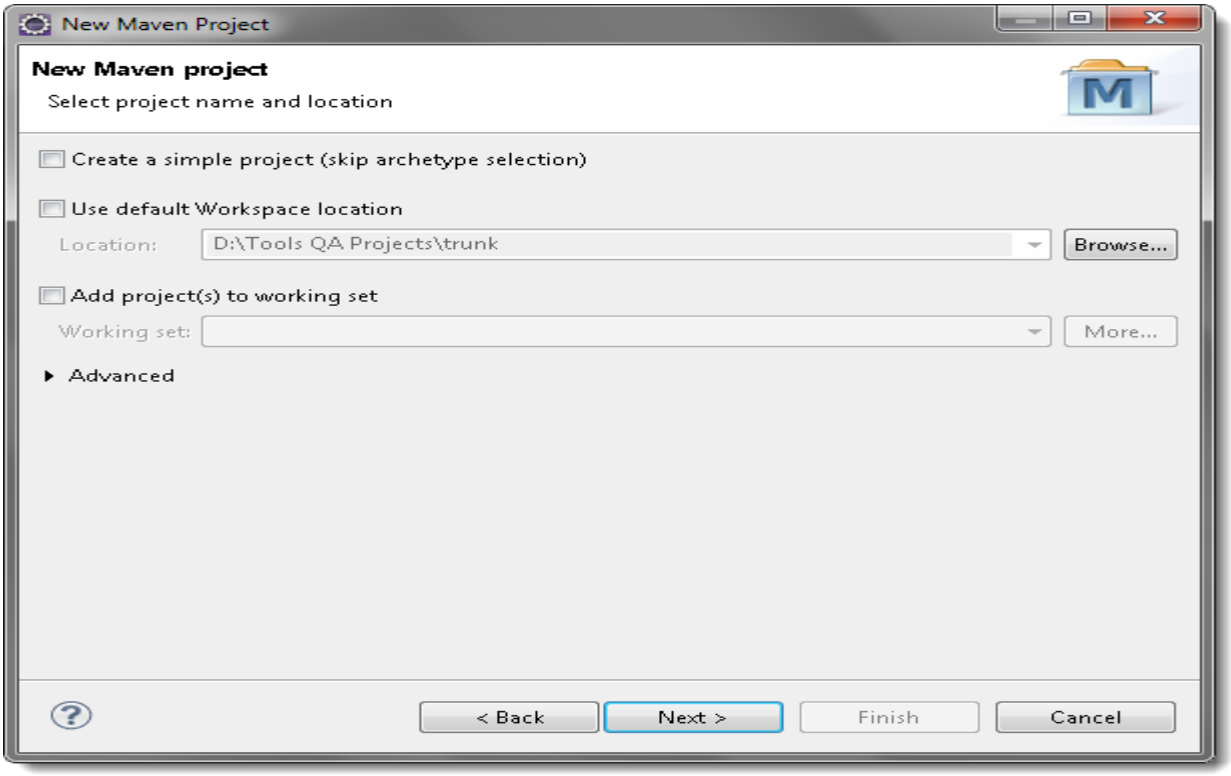
1. Open your *eclipse* and ***Go to File > New > Others.***



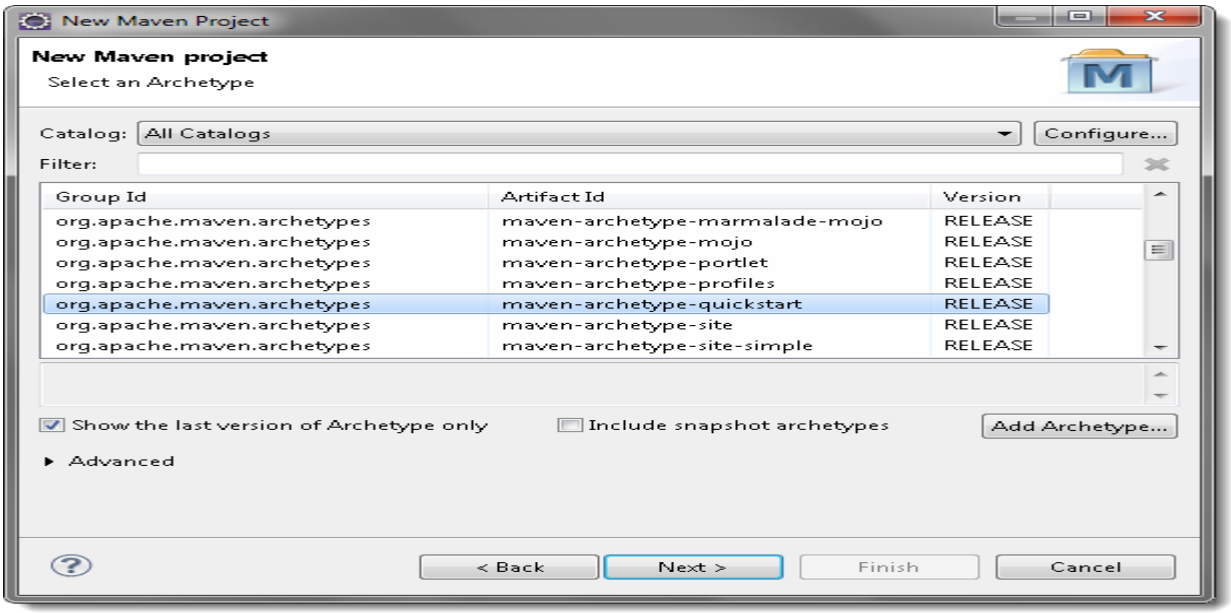
2. Select ***Maven Project*** and *click* on ***Next.***



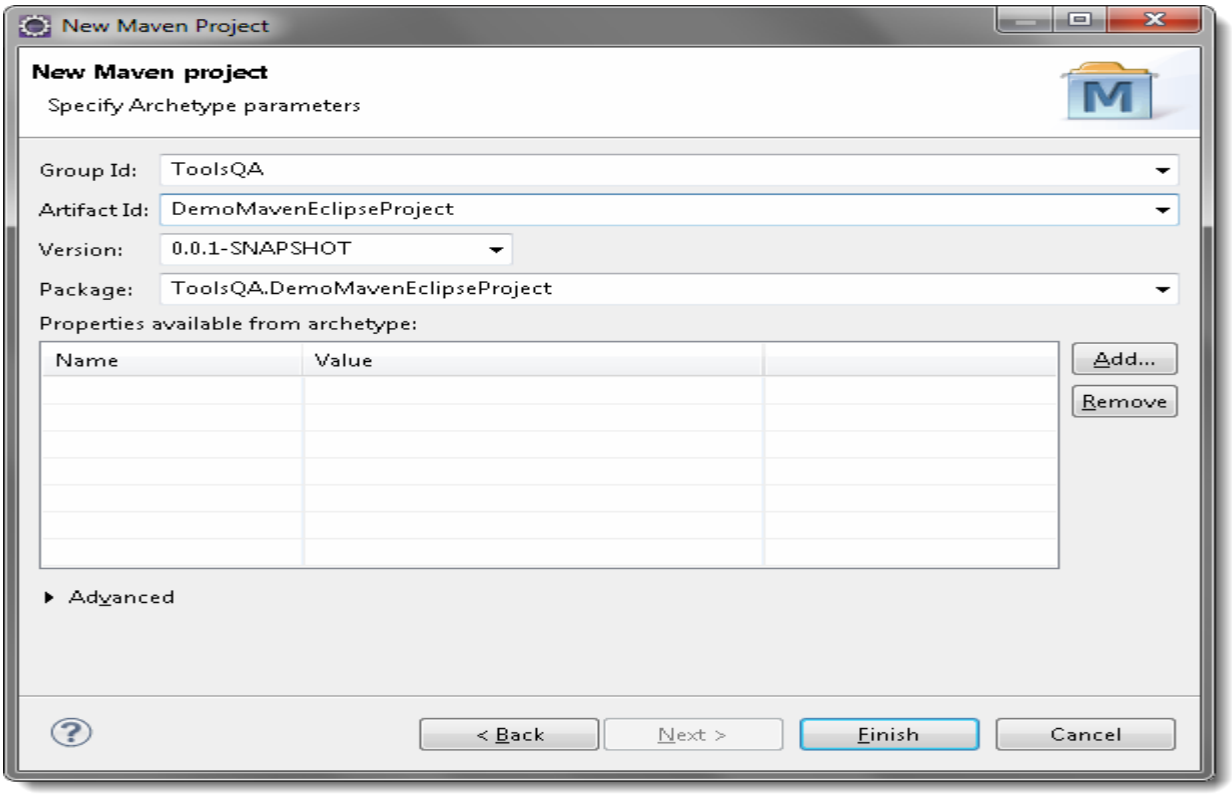
3. *Un-check* the '*Use default Workspace location*' and with the help of the ***Browse*** button choose your *workspace* where you would like to set up your *Maven project.*



4. Select the *archetype,* for now just select the '*maven-aechetype-quickstart*' and *click* on ***Next.***

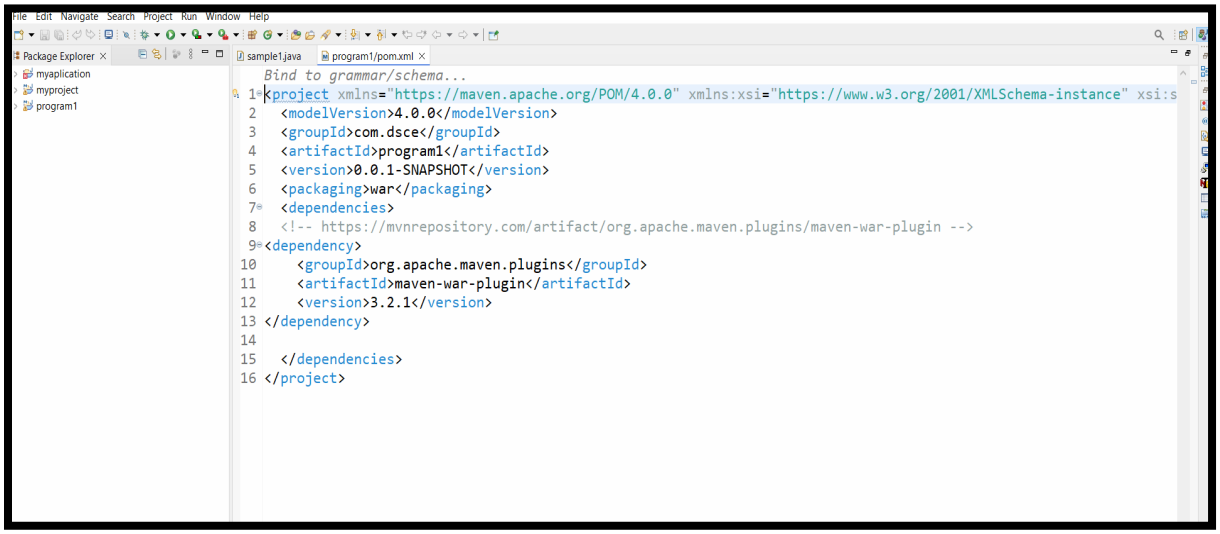


5. Specify the ***Group Id*** & ***Artifact Id*** and *click* on ***Finish.***

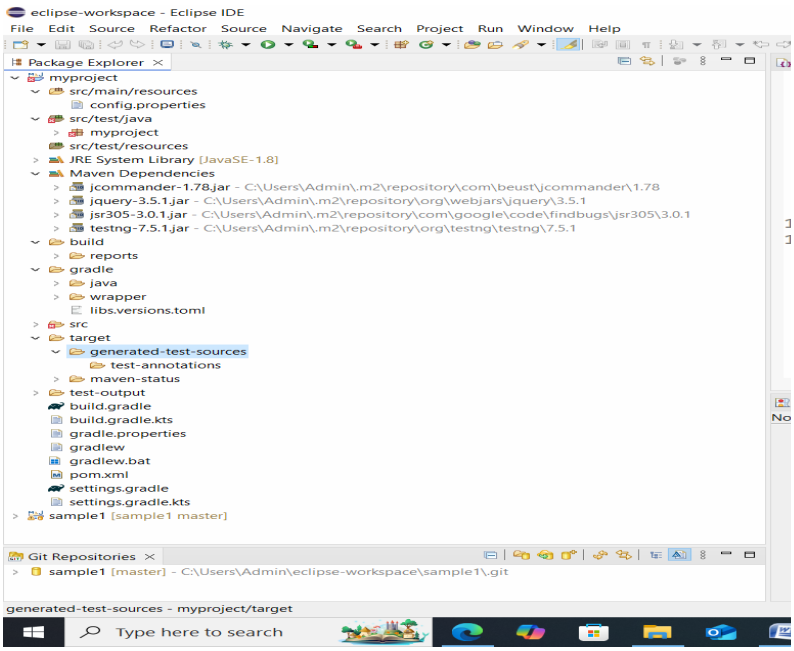


Note: *Here the 'artifactId' is your project name.*

6) Go to the project location to see the newly created maven project. Now open the *pom.xml* file, which resides in the project folder. By default the POM is generated like this:



7) Look at the default folder structure of the Maven project.



Root Directory (Project Directory):

- This is the top-level directory of your project.
- It contains the `pom.xml` file, which is the Project Object Model, the heart of Maven's configuration.

Key Subdirectories:

1. **src/:**
    - This directory contains all the source code for your project.
    - **src/main/java/:**
        - Contains the Java source code for your application.
        - The package structure of your Java classes mirrors the directory structure within this folder.
    - **src/main/resources/:**
        - Contains resources used by your application, such as configuration files, property files, and images.
    - **src/main/webapp/** (for web applications):
        - Contains web related files, such as HTML, CSS, JavaScript, and JSP files.
        - Often contains a WEB-INF folder.
    - **src/test/java/:**
        - Contains the Java source code for your unit tests.
        - The package structure mirrors the src/main/java/ directory.
    - **src/test/resources/:**
        - Contains resources used by your unit tests.
2. **target/:**
    - This directory is created by Maven during the build process.

- It contains the compiled classes, generated JAR/WAR files, and other build artifacts.
- Contents of this folder are usually deleted when the clean lifecycle phase of maven is executed.

3. **pom.xml:**
   - This is the Project Object Model file.
   - It contains the configuration for your Maven project, including dependencies, build settings, and plugins.

The student's record should include the following contents and snapshots pasted.

MyApp.java

```
import java.util.ResourceBundle;
public class MyApp
{
public int userlogin(string inuser,string inpwd)
{
ResourceBundle rb= ResourceBundle.getBundle("config");
String username=rb.getstring("username");
String password=rb.getstring("password");
If(inuser.equals(username)&& inpassword(password))
        return 1;
else
        return 0;
}
```

Config.properties

```
username=abc
password=abc@1234
```

MyAppTest.java code

```
package myproject;
import org.testng.Assert;
import org.testng.annotations.Test;

import com.myproject.app;

public class apptest {
  @Test
  public void testlogin1()
  {
            app myapp=new app();
```

```
                Assert.assertEquals(0,myapp.userlogin("abc","abc1234"));
        }
    @Test
    public void testlogin2()
    {
    app myapp=new app();
    Assert.assertEquals(1,myapp.userlogin("abc","abc@1234"));
    }
    }
```

In pom.xml add the following dependency to run Testcases.

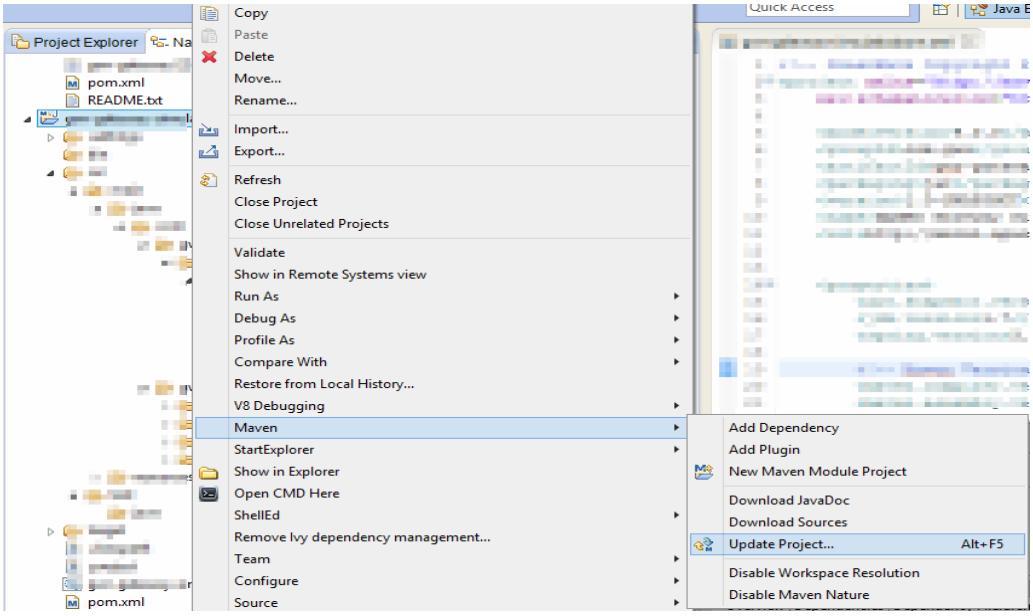
```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.testng/testng
-->
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.5.1</version>
    <scope>test</scope>
</dependency>

  </dependencies>
```

When you build your Maven project (e.g., using `mvn compile` or `mvn test`), Maven reads the `pom.xml` file.

It then downloads the specified TestNG dependency from the Maven repository and stores it in your local Maven repository.

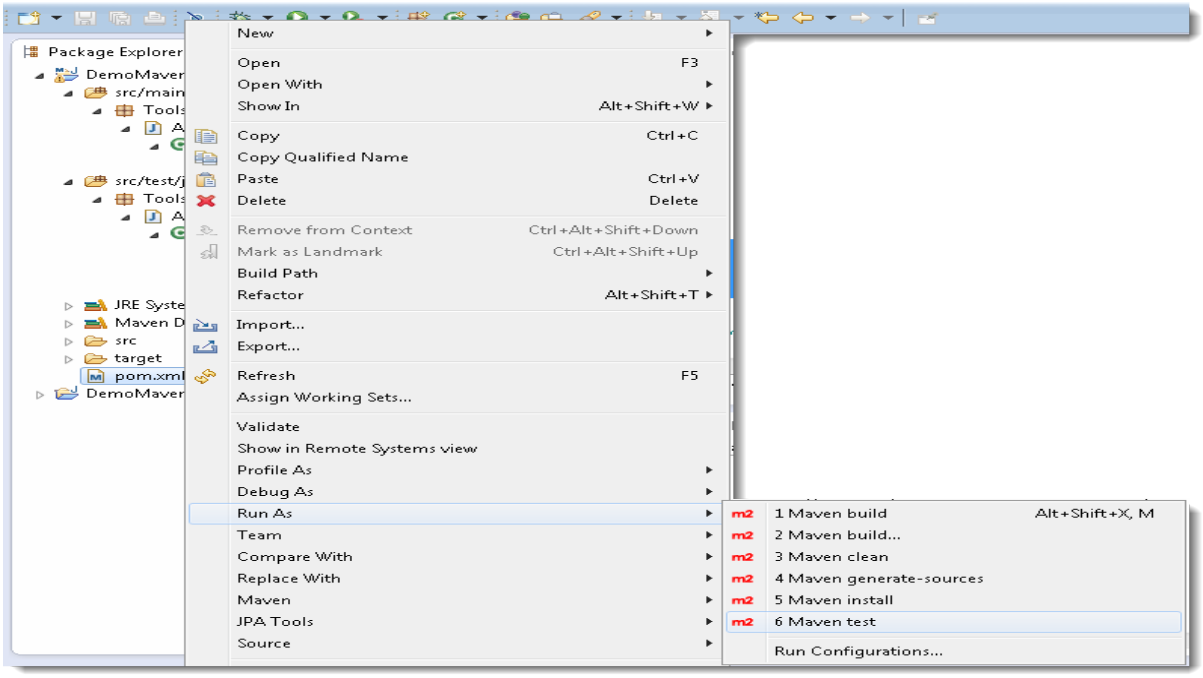
- Your project can then use the TestNG library.

Dependencies can be updated by using "Maven --> Update Project.." after pom.xml file modification.

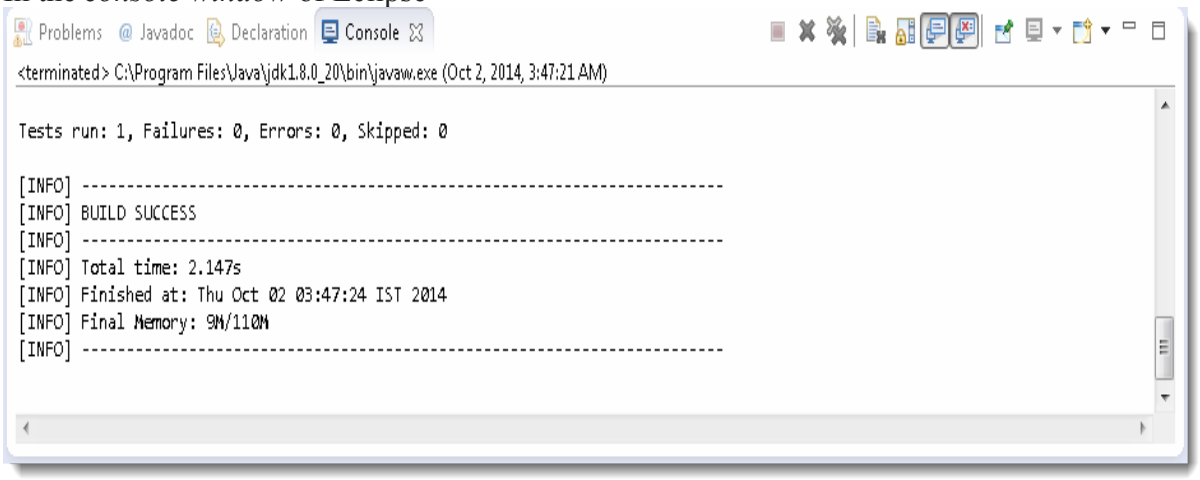


Run your first Maven Test

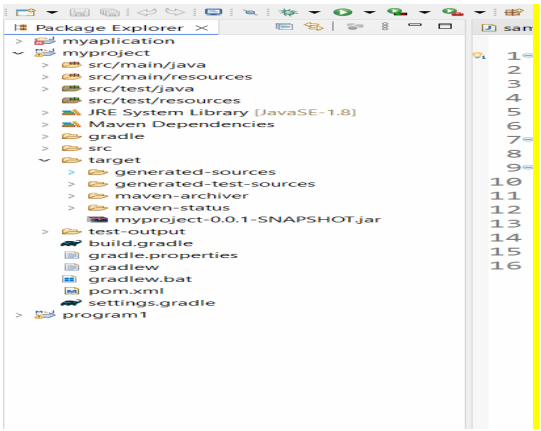
1. Right-click on the ***pom.xml*** and go to ***Run As > Maven test.***



2. In the *console window* of Eclipse



3. Go to '*target*' folder to view the .jar/.war file



How to install Gradle

1. **Download Gradle:**
   Visit the Gradle Downloads Page and download the latest binary ZIP file.
2. **Extract the ZIP File:**

   - Right-click the downloaded ZIP file and select **Extract All…** or use any extraction tool like WinRAR or 7-Zip.

   Move the Folder:
   - After extraction, move the extracted **Gradle folder** (usually named **gradle-x.x.x**) to a convenient directory like C:\Program Files\.

   Navigate to the bin Folder:
   - Open the **Gradle folder**, then navigate to the **bin** folder inside.
   - Copy the path from the File Explorer address bar (e.g., **C:\Program Files\gradle-x.x\bin**).

   Set Environment Variables:

- Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
- Click **Environment Variables**.
- Under **System Variables**:

    - Find the **path**, double click on it and click **New**.
    - Paste the full path to the bin folder of your Gradle directory (e.g., **C:\Program Files\gradle-x.x.x\bin**).

Save the Changes:

- Click **OK** to close the windows and save your changes.

Verify the Installation:

- Open a terminal or Command Prompt and run: **gradle -v** .if it shows the Gradle version, the setup is complete.
- Run **gradle init** in command prompt and check gradle folder is created in eclipse.

Make sure to run this project we have to install and configure the testNg plugin in eclipse.

Method 1: Using the Eclipse Marketplace (Recommended)

1. **Open Eclipse Marketplace:**
    - In Eclipse, go to Help > Eclipse Marketplace....
2. **Search for TestNG:**
    - In the "Find" field, type "TestNG" and press Enter.
3. **Install TestNG:**
    - You should see "TestNG for Eclipse" in the search results.
    - Click the "Install" button next to it.
4. **Review and Confirm:**
    - Eclipse will show you the features that will be installed.
    - Confirm the installation and accept the license agreements.
5. **Restart Eclipse:**
    - Eclipse will prompt you to restart. Click "Restart Now" to complete the installation.

Method 2: Installing via Install New Software

1. **Find the TestNG Update Site URL:**
    - The TestNG update site URL is typically: https://testng.org/testng-eclipse-update-site/.
2. **Open Install New Software:**
    - In Eclipse, go to Help > Install New Software....
3. **Add the Update Site:**
    - In the "Work with" field, paste the TestNG update site URL and press Enter.
4. **Select TestNG:**
    - Eclipse will display the available TestNG components.
    - Check the box next to "TestNG" (and any other relevant components).
5. **Click Next and Finish:**

    - Click "Next" to review the installation details.
    - Click "Next" again, accept the license agreements, and click "Finish."
6. **Restart Eclipse:**
    - Eclipse will prompt you to restart. Click "Restart Now" to complete the installation.

Verifying the Installation:

1. **Create a Java Project (if you don't have one):**
    - Go to File > New > Java Project.
2. **Create a New TestNG Class:**
    - Right-click on your project or a source folder.
    - Go to New > Other....
    - In the "New" dialog, type "TestNG" and select "TestNG Class."
    - Click "Next" and follow the wizard to create a TestNG class. If you see the TestNG wizard, that means TestNG is correctly installed.
3. **Run a TestNG Test:**
    - Once the test class is created, you should be able to right click the java file, and see Run As -> TestNG Test.

If you encounter any issues, make sure your Eclipse installation is up to date and that you have a stable internet connection during the installation process.