4.17 Running Tests with Maven



This section will guide you to know:

1. How to use Maven-surefire-plugin
2. What is Maven Java Project
3. Few of the Maven and JUnit5 examples
4. What are the Java Classes used to run the Maven Script
5. The Maven Test

This guide has six subsections, namely:

* + 1. Writing code to demonstrate Maven-surefire-plugin
    2. Explaining the structure of a Maven project
    3. Demonstrating Maven + JUnit5 examples
    4. Demonstrating Java Classes to run the Maven Script
    5. Executing Maven Test
    6. Pushing the code to GitHub repositories

**Step 4.17.1:** Writing a code to demonstrate Maven-surefire-plugin

Below is the Surefile Plugin. This Plugin (code below) needs to be added to the Page Object Model “POM” file in xml format.

* Add the below Surefile to POM

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.0</version>

</plugin>

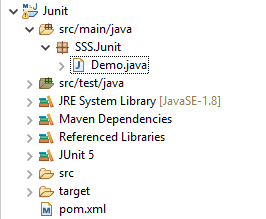
</plugins>

</build>

**Step 4.17.2:** Explaining the structure of a Maven project

* Below is a simple Java project which will guide us on how to run the unit test classes in the Maven project. This is the directory structure for creating a Maven project which essentially includes the Junit test cases, JRE system library, maven dependencies, Junit 5 library, and the POM XML file.

**Directory Structure**



**Step 4.17.3:** Demonstrating Maven + JUnit5 examples

* Below is the sample POM file with all the dependencies and other annotations:

<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>

<artifactId>junit-jupiter-params</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>Junit</name>

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

<properties>

<maven.compiler.source>1.9</maven.compiler.source>

<maven.compiler.target>1.9</maven.compiler.target>

</properties>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.0</version>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.2.0</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>5.2.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<version>5.0.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<groupId>org.junit.jupiter</groupId>

</project>

**Step 4.17.4:** Demonstrating Java Classes to run the Maven Script

* We have already created a complete Maven project structure above with Java source code. Now we will create different Java classes in the *./src/main/*java/packages/class(s). It also created an example test class in *./src/test/*. In the root folder, there is a *pom.xml* file.
  + - 1. **MagicBuilder.java**

MagicBuilder.java

**package** SSS.Junit;  
public class MagicBuilder {

public static int getLucky() {

return 7;

}

}

Textbuild.java

**package** SSS.Junit;  
  
public class Textbuild {

public static String getHelloWorld(){

return "hello world";

}

public static int getNumber10(){

return 10;

}

}

**4.17.4.2 Test class for MagicBuilder**

TestMagicBuilder.java

**package** SSS.Junit;  
  
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestMagicBuilder {

@Test

public void testLucky() {

assertEquals(7, MagicBuilder.getLucky());

}

}

* + - 1. **Test class for Textbuild**

pom.xml

**package** SSS.Junit;  
  
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class Msgbuild {

@Test

public void testHelloWorld() {

assertEquals("hello world", Textbuild.getHelloWorld());

}

@Test

public void testNumber10() {

assertEquals(10, Textbuild.getNumber10());

}

}

## **Step 4.17.5:** Executing Maven Test

* While we run the POM and class files with Maven, the below results will be generated at the console:
  + - 1. **Run all test classes**

$ mvn test

[INFO] -------------------------------------------------------

[INFO] T E S T S

[INFO] -------------------------------------------------------

[INFO] Running SSS.Junit.TestMagicBuilder

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in SSS.Junit.TestMagicBuilder

[INFO] Running SSS.Junit.Msgbuild

[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 s - in SSS.Junit.Msgbuild

[INFO]

* + - 1. **Run a single test class Msgbuild**

Terminal

$ mvn -Dtest=Msgbuild test

[INFO] -------------------------------------------------------

[INFO] T E S T S

[INFO] -------------------------------------------------------

[INFO] Running SSS.Junit.Msgbuild

[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in SSS.Junit.Msgbuild

[INFO]

* + - 1. **Run a single test method testHelloWorld() from the test class Msgbuild**

Terminal

$ mvn -Dtest=Msgbuild#testHelloWorld test

[INFO] -------------------------------------------------------

[INFO] T E S T S

[INFO] -------------------------------------------------------

[INFO] Running SSS.Junit.Msgbuild

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in SSS.Junit.Msgbuild

[INFO]

**Step 4.17.6:** Pushing the code to GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add . 

Commit the changes using the following command:

git commit . -m “Changes have been committed.”

Push the files to the folder you initially created using the following command:

git push -u origin master