**Day 13- Facilitation Guide**

**TestNG**

**Index**

I. Why TestNG? And it’s advantage

II. TestNG Installation and Setting the Perspective to Java in Eclipse

III. Creation of TestNG class

IV. TestNG Annotations

V. Execution of TestNG classes

VI. Launching a Browser and Opening a Website Using Selenium WebDriver with TestNG

**For (2 hrs) ILT**

### **I. Introduction to TestNG**

TestNG (Test Next Generation) is an **open-source automated testing framework** where NG means **Next Generation**. It is inspired by JUnit and NUnit but offers more flexibility and better functionalities. It allows developers and testers to write automated test cases efficiently for various types of tests, including **unit, functional, and integration testing**.

### **Why TestNG?**

* It **overcomes JUnit’s limitations** by providing advanced features such as annotations, parameterized testing, and parallel execution.
* Supports all types of testing like **unit, functional, and integration testing**.
* **Generates HTML reports** after test execution (JUnit lacks built-in reporting features).
* Provides **annotations** for better test case management.
* Allows **grouping and prioritization** of test cases to control execution flow.
* Enables **data parameterization** using built-in annotations.
* Supports **parallel execution** of test methods.

### **Advantages of TestNG**

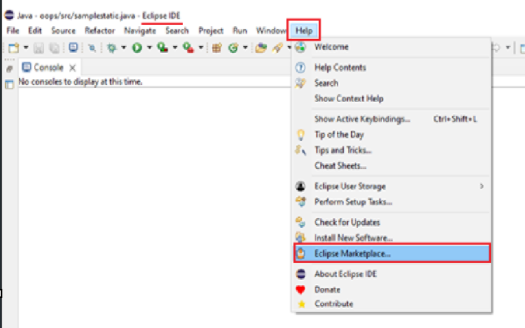
| **Feature** | **Description** |
| --- | --- |
| **Annotations** | Provides annotations like @Test, @BeforeMethod, @AfterMethod, @BeforeClass, @AfterClass, etc., to organize test execution. |
| **Parallel Execution** | Allows running multiple test cases simultaneously, reducing execution time. |
| **Grouping of Test Cases** | Supports test case grouping using the groups attribute, making test management easier. |
| **Data-driven Testing** | Supports parameterized testing using the @DataProvider annotation. |
| **Dependency Management** | Helps set dependencies between test cases using the dependsOnMethods attribute. |
| **Powerful Test Execution** | Allows setting priorities, skipping tests, and rerunning failed tests. |
| **Integration with Other Tools** | Easily integrates with **Selenium, Maven, Jenkins, and other testing tools**. |

## **II. TestNG Installation and Setting the Perspective to Java in Eclipse**

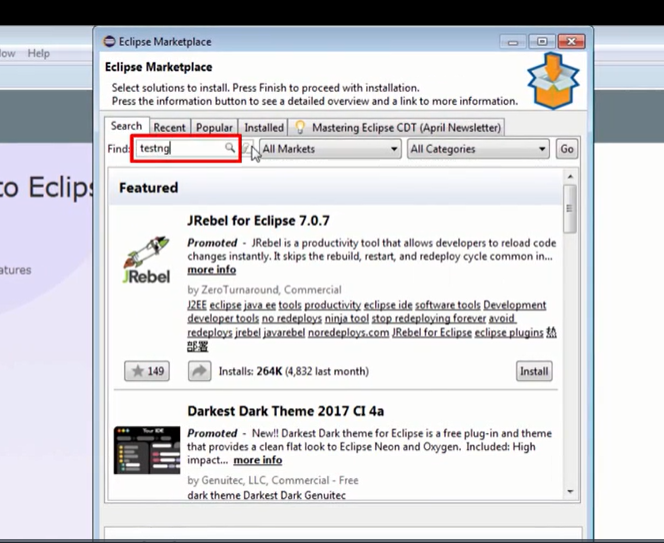
## **Installing TestNG in Eclipse**

#### **Step-by-Step Installation Process**

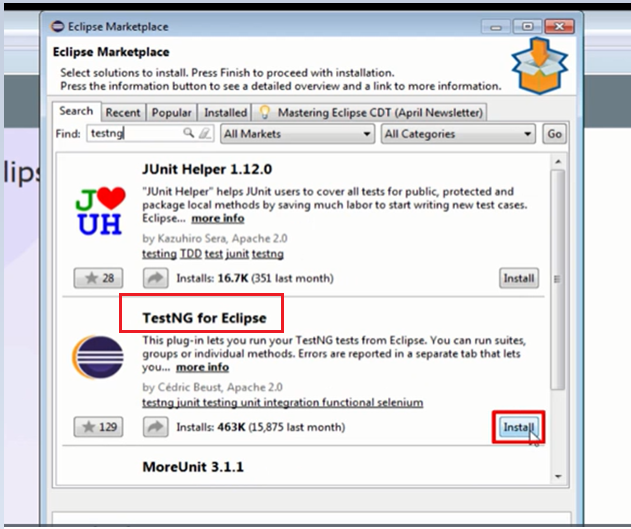
1. **Open Eclipse** → Go to **Help** → Click **Eclipse Marketplace**.



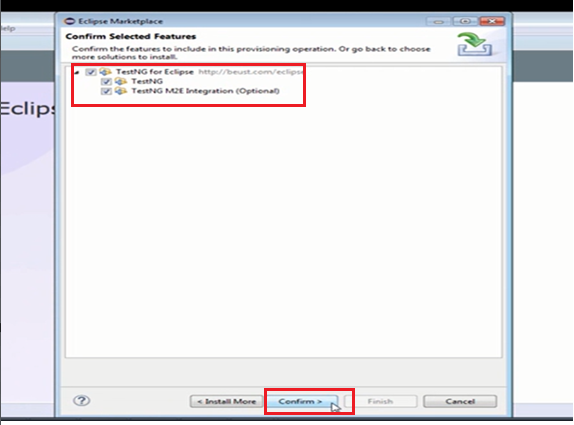
1. In the search bar, type **TestNG** and click **Search**.



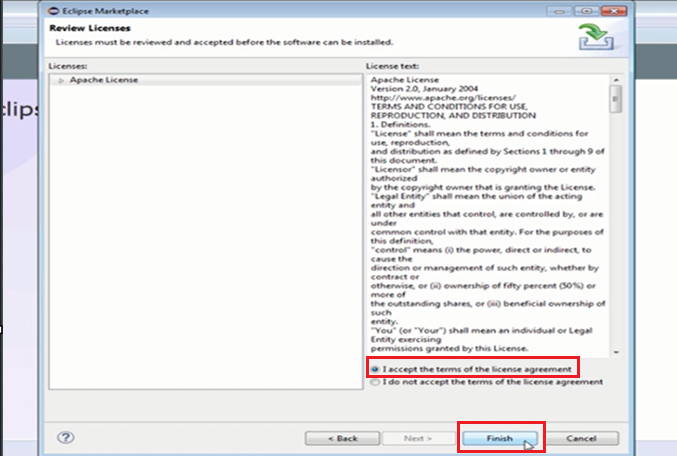
1. Find **TestNG for Eclipse** and click **Install**.



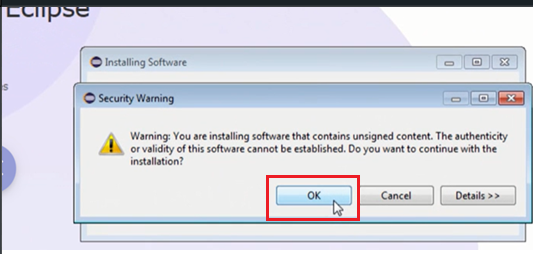
1. Select both options and click **Confirm**.



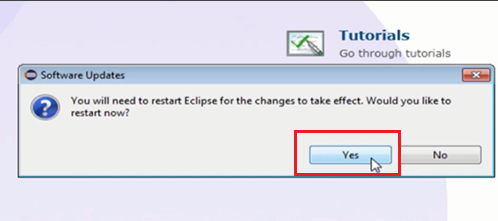
1. Accept the license agreement and click **Finish**.



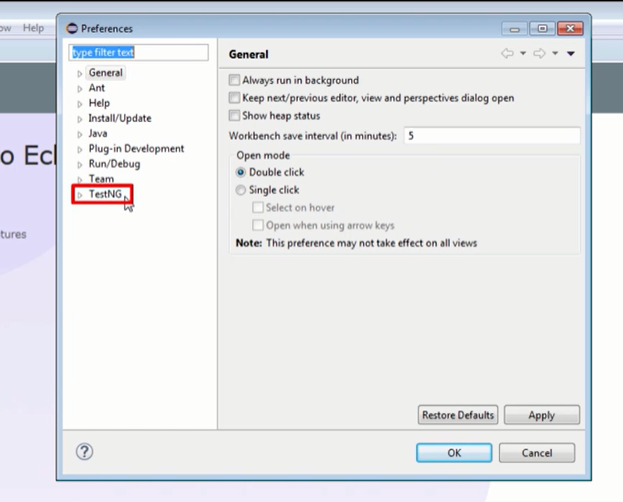
1. Click **OK** to continue installation.



1. Click **Yes** to restart Eclipse for the changes to take effect.



1. After restart, go to **Window → Preferences** and check whether **TestNG is installed**.



#### **Setting the Perspective to Java in Eclipse**

#### **What is a Perspective in Eclipse?**

#### A **Perspective** in Eclipse is a predefined layout that provides specific views, menus, and toolbars optimized for a particular type of development, such as Java, Debugging, or Git.

#### **Why Set the Perspective to Java?**

#### When you set the perspective to **Java**, Eclipse optimizes the environment for Java development by enabling essential tools and features. Below are the key benefits:

#### **1. Java-Specific Views & Tools**

#### **Package Explorer**: Displays the project structure and allows easy navigation.

#### **Outline View**: Provides a structural overview of a Java class, showing methods and fields.

#### **Problems View**: Displays compilation errors, warnings, and other issues.

#### **Javadoc View**: Shows documentation for selected Java elements.

#### **2. Optimized Menus & Toolbars**

#### Provides options like **"Run as Java Application"**, **"Run as JUnit Test"**, etc.

#### Java-related features are more accessible compared to other perspectives like **Debug** or **Git**.

#### **3. Better Coding Assistance**

#### Enables **code completion, syntax highlighting, and debugging support**.

#### Allows **quick fixes** (e.g., using Ctrl + 1 to auto-correct errors).

#### **How to Set the Perspective to Java**

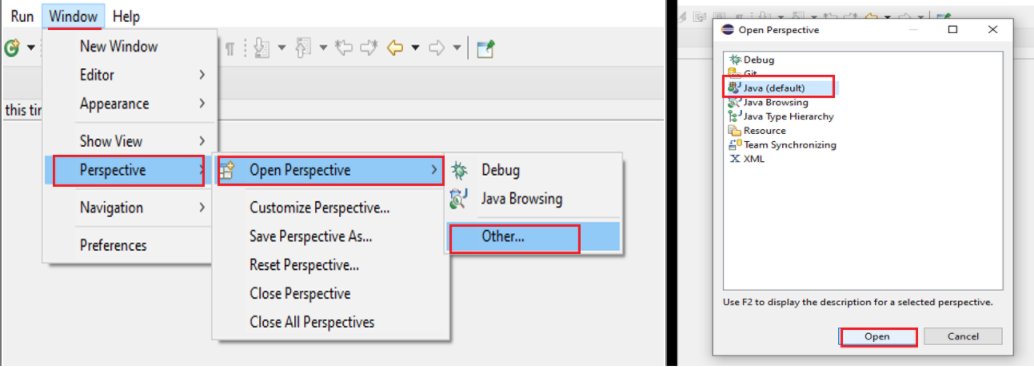
#### Follow these steps to set the Java perspective in Eclipse:

#### **Open Eclipse IDE**.

#### Go to **Window** → **Perspective** → **Open Perspective** → **Other**.

#### In the list of available perspectives, select **Java (default)**.

#### Click **Open**.



#### The IDE will now be set up with Java-related tools and views.

#### **Do We Need to Set Java Perspective for Every Maven/Java Project?**

#### **Once set, the Java perspective remains active** for all Java and Maven projects.

#### If you switch to a different perspective (e.g., **Debug** or **Git**), you may need to manually switch back to the **Java perspective** for a better development experience.

#### **Best Practice**

#### Always ensure you are in the **Java Perspective** while working on Java/Maven projects to enhance your productivity and maintain an efficient workflow.

## **III. Creation of a TestNG Class**

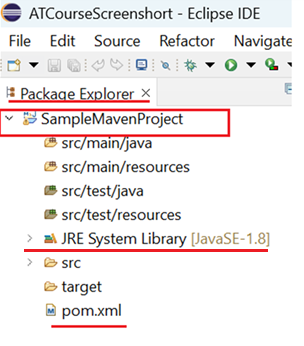
### **Why Create a TestNG Class?**

* Required to **encapsulate methods** annotated with TestNG annotations.
* Used to **write Selenium WebDriver test scripts**.
* Ensures that TestNG **library is added** to the project.

**Adding TestNG to a Java Project (Maven Approach)**

**Step 1: Create a Maven Project in Eclipse/IntelliJ**

1. **Open Eclipse/STS** → Go to **File → New → Maven Project**.
2. Check **"Create a simple project"** and click **Next**.
3. **Provide Project Details**:
   1. **Group Id**: org.anudip // organisation id
   2. **Artifact Id**: SampleMavenProject // project name
   3. **Packaging**: jar
4. **Click Finish** to create the project.



**Step2: Add the following dependency to pom.xml after the <version> tag**

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.4.0</version>

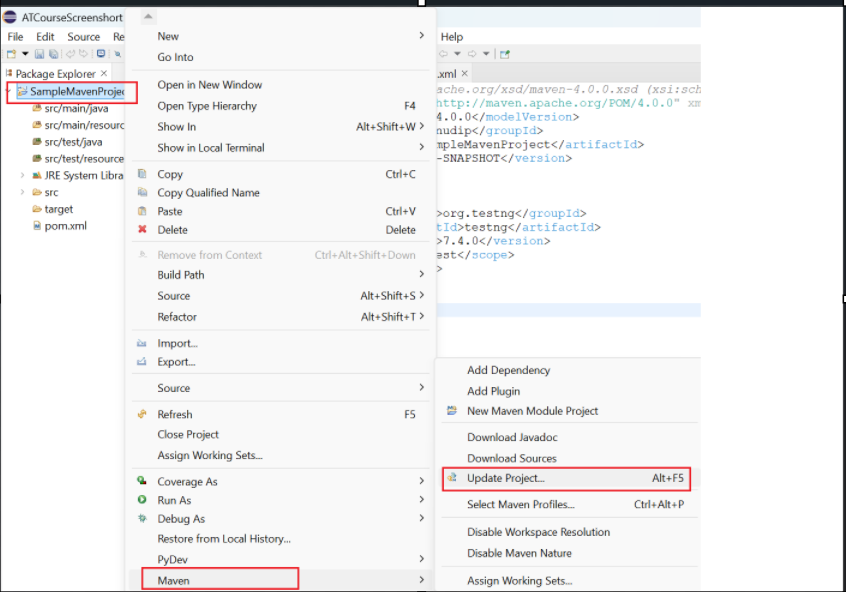
<scope>test</scope>

</dependency>

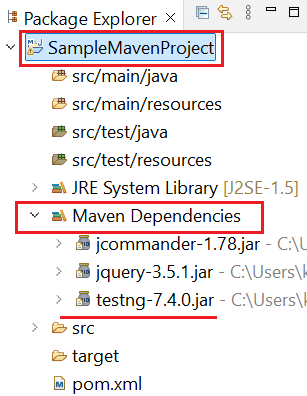
</dependencies>



Step 3: Save the pom.xml file and **update the project** (Right Click → Maven → Update Project).

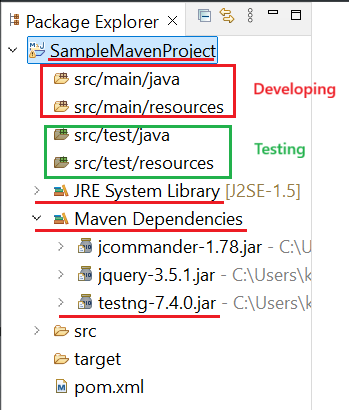


Now you can see the testNG is added as Maven Dependencies in the project



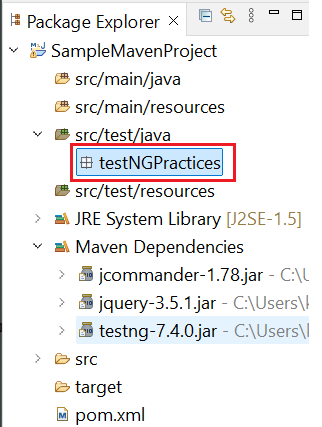
### **Steps to Create a TestNG Class**

#### **Step 1. Create the Maven project and add testNG dependencies using the above steps**



#### **2. Creating a User-Defined Package**

* **Right-click on the src/test/java folder**.
* Create a **user-defined package** “testngpractices”



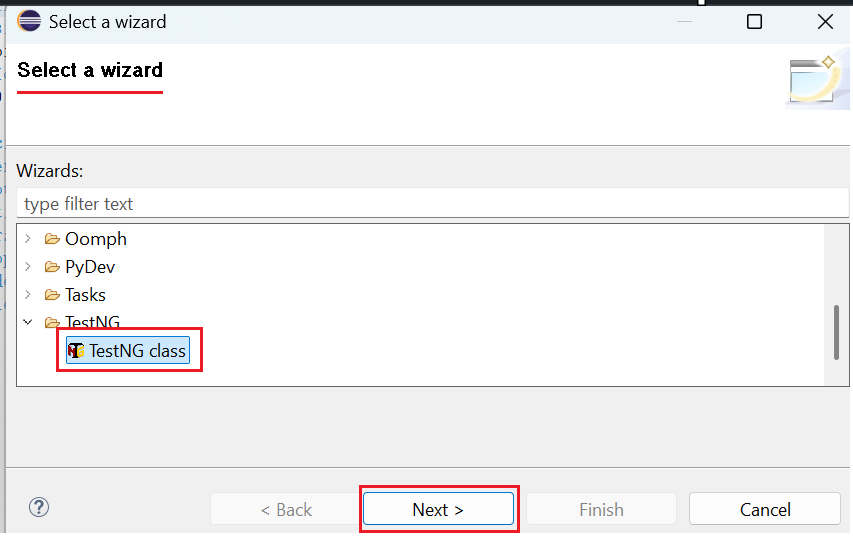
#### **3. Creating a TestNG Class**

#### **3. 1 Right-click on the package(tesngpractices)** → **New → Other…** → **TestNG Class**.



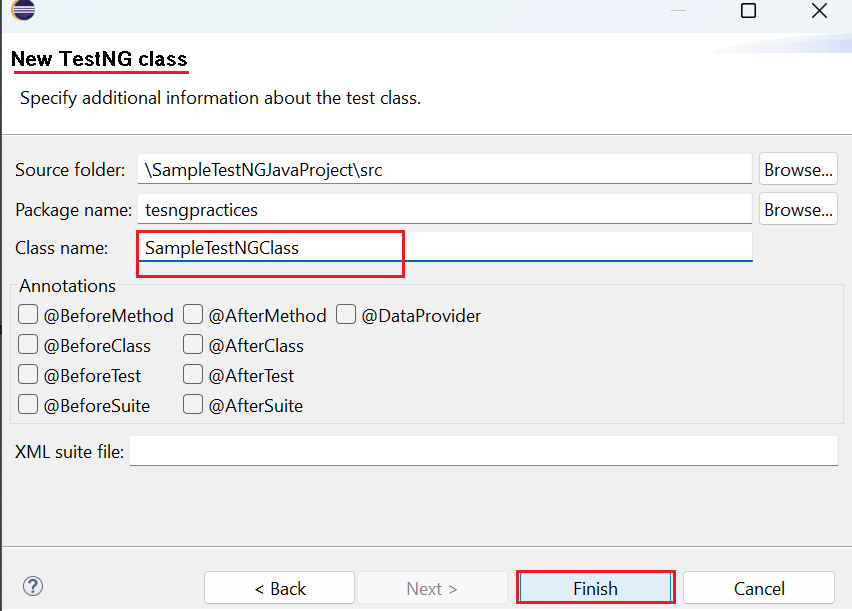
#### **3.2. Selecting TestNG Class Wizard**

* Click **Next**.



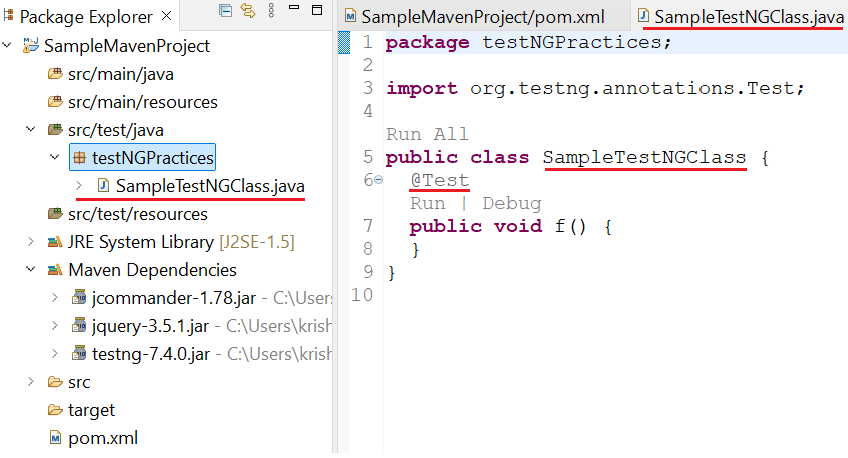
#### **3.3. Naming the TestNG Class**

* Enter a name for the TestNG class as “SampleTestNGClass” and click **Finish**.



#### **4. Checking the Created TestNG Class**

* When a **TestNG class** is created in Eclipse, it automatically includes a method with the @Test annotation by default.



* The extension of a **TestNG class** is .java, as it is a standard Java class that uses TestNG annotations for test execution.

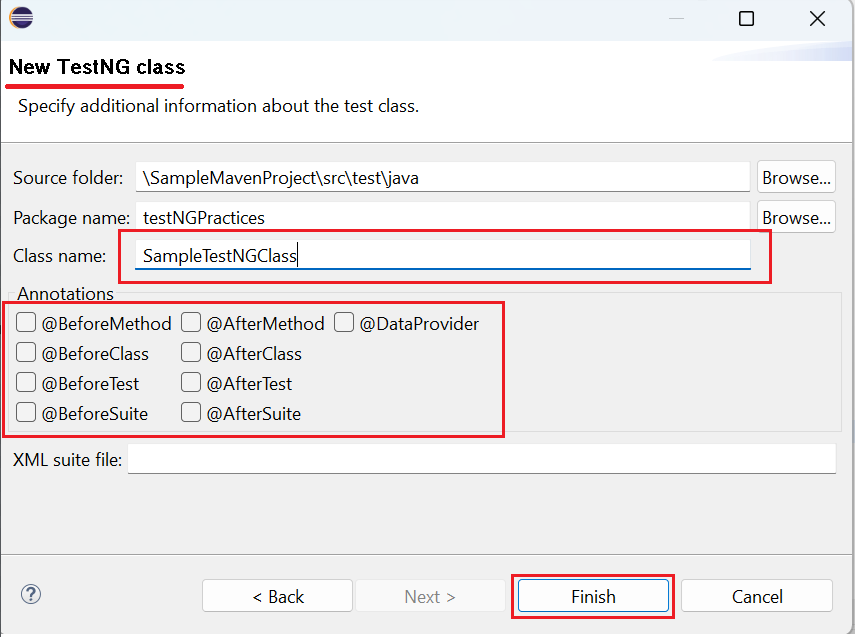
IV. **TestNG Annotations**

What Are TestNG Annotations?

Annotations in TestNG control the order of method execution in the test script. If a method is not prefixed with an annotation, it will be ignored and not executed as part of the test.

**Types of TestNG Annotations and Execution Order**

| **Annotation** | **Execution Order** | **Description** |
| --- | --- | --- |
| @BeforeSuite | 1st | Runs once before all tests in a suite. |
| @BeforeTest | 2nd | Runs once before any <test> tag in testng.xml. |
| @BeforeClass | 3rd | Runs once before the first method in a class. |
| @BeforeMethod | 4th | Runs before each @Test method. |
| @Test | 5th | Defines the actual test case. |
| @AfterMethod | 6th | Runs after each @Test method. |
| @AfterClass | 7th | Runs once after all methods in the class. |
| @AfterTest | 8th | Runs once after all tests in a <test> tag in testng.xml. |
| @AfterSuite | 9th | Runs once after all test execution is complete. |

****

**Understanding the Execution Flow of TestNG Annotations**

To understand how TestNG annotations work, consider the following example:

import org.testng.annotations.\*;

public class TestNGExecutionFlow {

@BeforeSuite

public void beforeSuite() {

System.out.println("Before Suite Execution");

}

@BeforeTest

public void beforeTest() {

System.out.println("Before Test Execution");

}

@BeforeClass

public void beforeClass() {

System.out.println("Before Class Execution");

}

@BeforeMethod

public void beforeMethod() {

System.out.println("Before Method Execution");

}

@Test

public void testMethod1() {

System.out.println("Executing Test Method 1");

}

@Test

public void testMethod2() {

System.out.println("Executing Test Method 2");

}

@AfterMethod

public void afterMethod() {

System.out.println("After Method Execution");

}

@AfterClass

public void afterClass() {

System.out.println("After Class Execution");

}

@AfterTest

public void afterTest() {

System.out.println("After Test Execution");

}

@AfterSuite

public void afterSuite() {

System.out.println("After Suite Execution");

}

}

**Expected Execution Order**

Before Suite Execution

Before Test Execution

Before Class Execution

Before Method Execution

Executing Test Method 1

After Method Execution

Before Method Execution

Executing Test Method 2

After Method Execution

After Class Execution

After Test Execution

After Suite Execution

**V. Execution of a TestNG Class**

A TestNG class can be executed in two ways:

1. **As a TestNG Test Execution – Running a testNG class directly without using an XML file.**
2. **As a TestNG Suite Execution– Running a testNG class using a testng.xml configuration file**.

**Note:**

* **At least one method must be annotated with @Test for the .java file to execute successfully in TestNG.**
* **By default, the @Test annotation is included when we create a TestNG class without selecting any specific annotations.**

Before executing the test, ensure that the TestNG class is created with primary annotations.

**Execution of TestNG Class**

**a. Running a TestNG Class Without an XML File(TestNG Test Execution)**

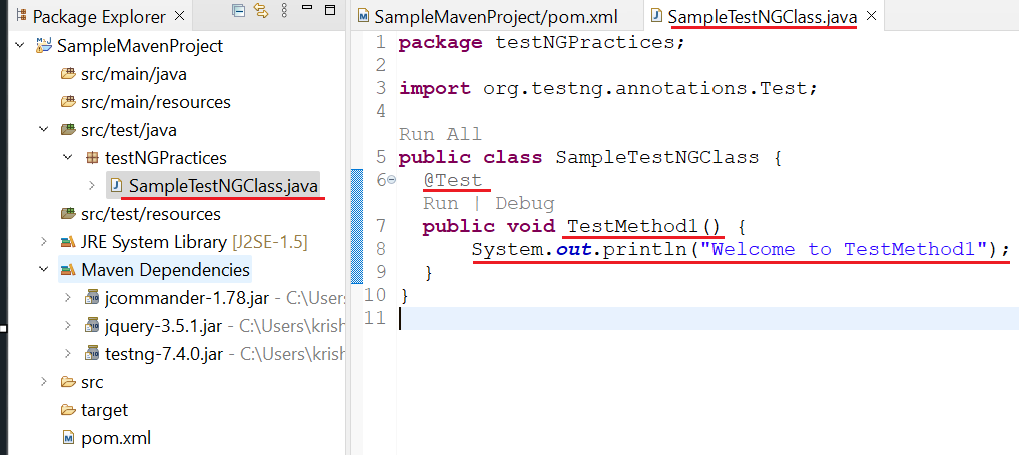
In TestNG, you can execute test cases directly without creating a separate XML configuration file (testng.xml). This method allows for quick and simple test execution, making it ideal for small-scale testing and debugging.

This approach is useful when you need to test individual classes quickly without additional configuration. It is particularly beneficial for beginners learning TestNG, as it simplifies execution without requiring XML setup.

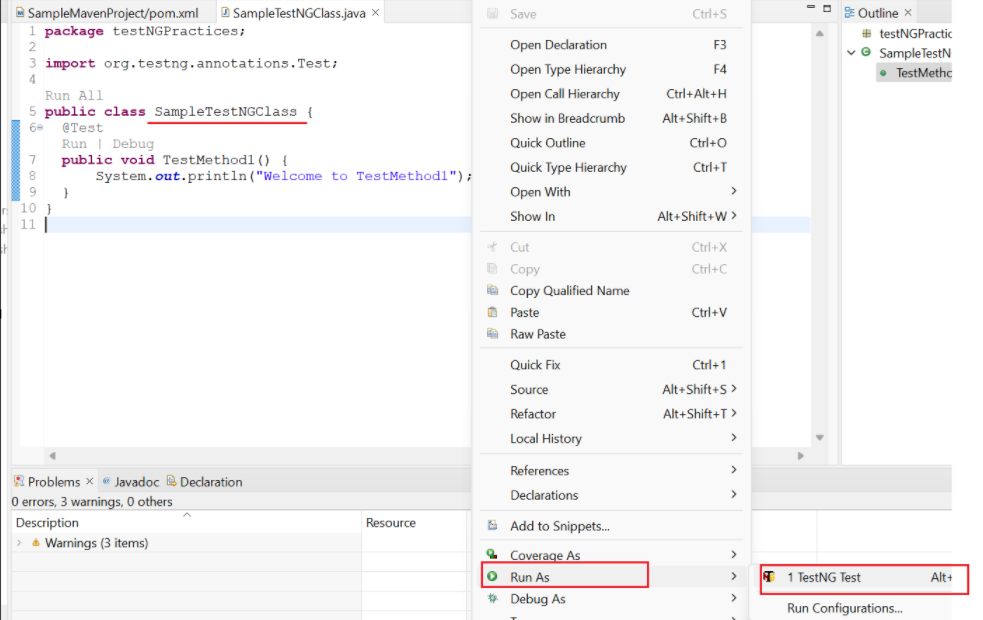
Executing a TestNG class directly from an IDE like Eclipse or IntelliJ IDEA eliminates the need for complex configurations, allowing developers and testers to focus on writing test cases and validating their behavior.

**Steps to Execute a TestNG Class**

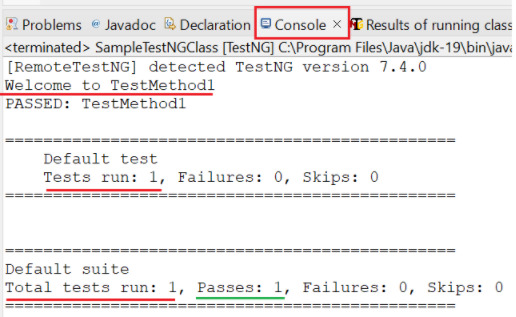
1. **Ensure that TestNG is installed** in Eclipse and add TestNG in your Maven project through pom.xml
2. **Create a TestNG class** with at least one @Test method.



1. **Run the class directly**:
   * **Right-click on the SampleTestNGClass .java file** → Select **Run As** → Click **TestNG Test**.



1. **Observe the output** in the console.

****

**Example Code**

**Create a TestNG class with @BeforeMethod, @AfterMethod @Test**

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.Test;

public class TestExecutionExample {

@BeforeMethod

public void beforeMethod() {

System.out.println("Before Test Method");

}

@Test

public void testMethod1() {

System.out.println("Executing Test Method 1");

}

@Test

public void testMethod2() {

System.out.println("Executing Test Method 2");

}

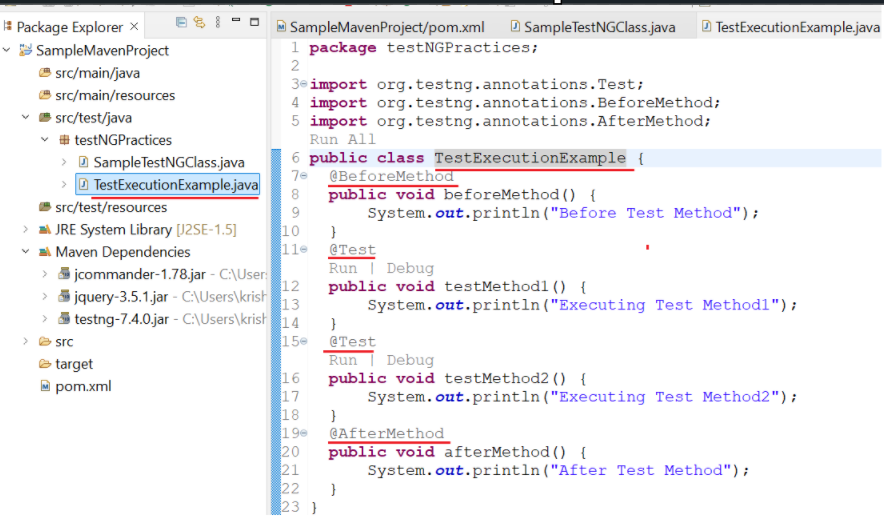
@AfterMethod

public void afterMethod() {

System.out.println("After Test Method");

}

}



**Expected Console Output**

Before Test Method

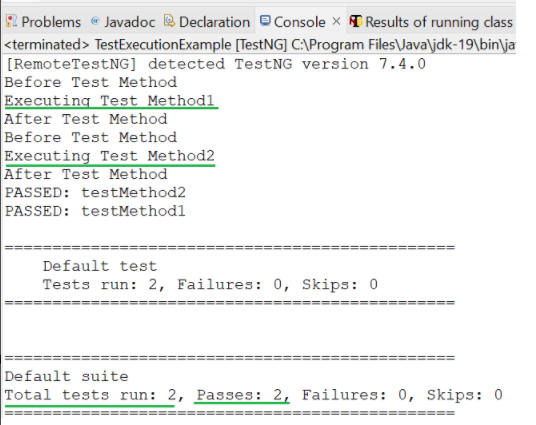
Executing Test Method 1

After Test Method

Before Test Method

Executing Test Method 2

After Test Method



**Understanding the Code Structure**

| **Annotation** | **Purpose** | **Execution Order** |
| --- | --- | --- |
| @BeforeMethod | Runs **before each test method** | 1st |
| @Test | Represents a **test case** | 2nd |
| @AfterMethod | Runs **after each test method** | 3rd |

**Execution Flow of the Code**

1. @BeforeMethod executes before every test.
2. testMethod1() runs.
3. @AfterMethod executes after testMethod1().
4. @BeforeMethod executes again before testMethod2().
5. testMethod2() runs.
6. @AfterMethod executes after testMethod2().

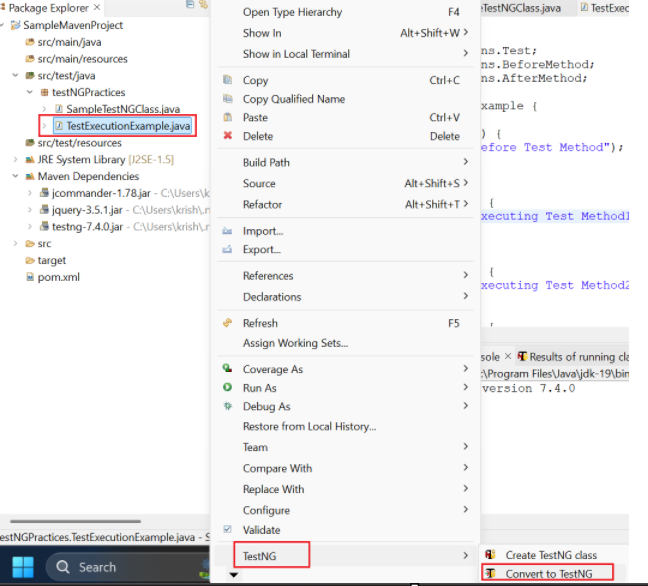
**b. Running a TestNG Class Using testng.xml File (TestNG Suite Execution)**

In this method, a testng.xml file is created and used to run TestNG tests.

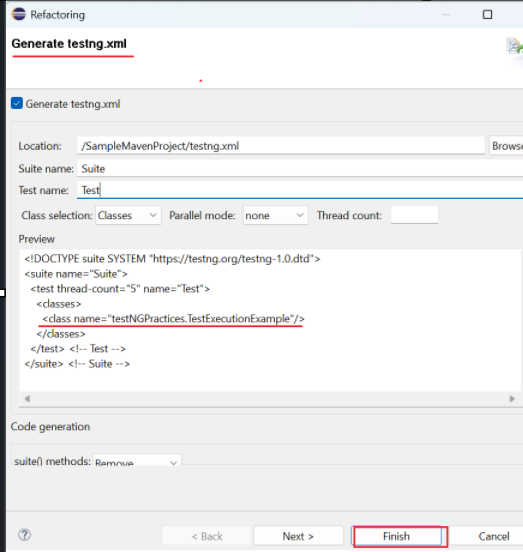
**Steps to Execute a TestNG Class Using an XML File**

**Step 1:** **Convert the TestNG Class to an XML File**

* Right-click on the .java file → Choose **TestNG** → Click **Convert to TestNG**.



* Click **Finish** to create the testng.xml file.



**Step 2:** see **the Generated testng.xml File**

Example:

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="MyTestSuite">

<test name="MyTest">

<classes>

<class name="testNGPractices.TestExecutionExample"/>

</classes>

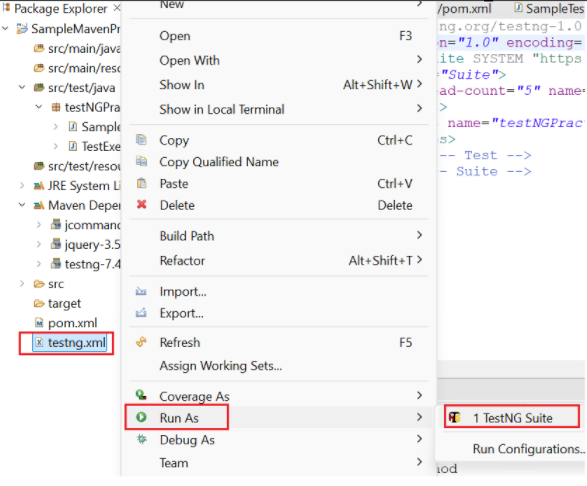
</test>

</suite>



**Step 3:** **Run the TestNG Suite**

* Right-click on testng.xml → Select **Run As** → Click **TestNG Suite**.



**Expected Output:**

Before Test Method

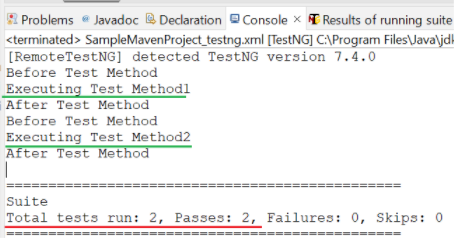
Executing Test Method 1

After Test Method

Before Test Method

Executing Test Method 2

After Test Method



**Advantages of Running Tests Using testng.xml**

* **Allows batch execution of multiple test cases.**
* **Enables parallel execution of tests.**
* **Supports parameterized testing for dynamic test execution.**

**Comparison: TestNG Test vs. TestNG Suite**

| **Feature** | **TestNG Test** | **TestNG Suite (testng.xml)** |
| --- | --- | --- |
| Execution | Run individual test classes | Run multiple test classes |
| Batch Execution | No | Yes |
| Parameterization | Limited | Supports passing parameters from XML |
| Parallel Execution | No | Yes |
| Use in CI/CD | Not ideal | Ideal for automation pipelines |

**VI. Launching a Browser and Opening a Website Using Selenium WebDriver with TestNG**

**1. Introduction**

TestNG is widely used for automated testing in Selenium, allowing structured test execution. In this section, we will focus on how to launch a browser and open a website using TestNG.

**2. Setting Up Selenium with TestNG**

Prerequisites:

Install TestNG in Eclipse (Help → Eclipse Marketplace → Search & Install TestNG).

Add Selenium & TestNG Dependencies (Maven Project)

Use WebDriverManager for managing browser drivers dynamically.

Add the following dependencies to pom.xml:

<dependencies>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.4.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.15.0</version> <!-- Use the latest version -->

</dependency>

<!-- https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager -->

<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>

<version>5.9.2</version>

</dependency>

</dependencies>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Writing a Basic TestNG Script to Open a Browser

Example: Launch Chrome and Open Google using WebDriverManager

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;

import org.testng.annotations.\*;

public class OpenBrowserTestNG {

WebDriver driver;

@BeforeMethod

public void setup() {

WebDriverManager.chromedriver().setup();

driver = new ChromeDriver();

driver.manage().window().maximize();

}

@Test

public void openGoogle() {

driver.get("https://www.google.com");

System.out.println("Google is opened successfully.");

}

@AfterMethod

public void teardown() {

driver.quit();

}

}

Explanation of Code:

@BeforeMethod → Sets up the browser before each test case using WebDriverManager.

@Test → Opens a website (https://www.google.com).

@AfterMethod → Closes the browser after execution.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Running the TestNG Class in Eclipse

Ensure the dependencies are correctly added.

Right-click the Java file → Run As → TestNG Test.

Console Output Example:

Google is opened successfully.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Using testng.xml for Execution

Instead of running the test class directly, we can execute it using a testng.xml file.

Steps to Use testng.xml

Convert Java Class to TestNG XML

Right-click the class file → TestNG → Convert to TestNG.

This will create testng.xml.

Modify testng.xml

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="LaunchBrowserSuite">

<test name="LaunchBrowserTest">

<classes>

<class name="OpenBrowserTestNG"/>

</classes>

</test>

</suite>

Run the TestNG Suite:

Right-click testng.xml → Run As → TestNG Suite.