

Practical No. 10 TestNG Framework Introduction

Date: _____

Aim:

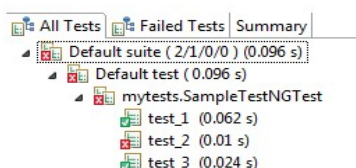
To study TestNG Framework.

Theory:

TestNG is an automation testing framework in which NG stands for “Next Generation”. It was developed by Cedric Beust. TestNG is inspired by JUnit which uses the annotations (@). TestNG overcomes the disadvantages of JUnit and is designed to make end-to-end testing easy. Using TestNG, you can generate a proper report, and you can easily come to know how many test cases are passed, failed, and skipped. You can execute the failed test cases separately. Default Selenium tests do not generate a proper format for the test results. Using TestNG in Selenium, we can generate test results.

Following are the key features of Selenium TestNG:

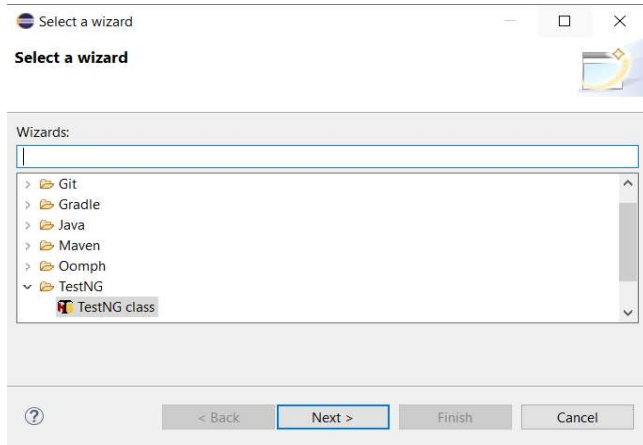
- Generate the report in a proper format including a number of test cases runs, the number of test cases passed, the number of test cases failed, and the number of test cases skipped.
- Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first.
- The same test case can be executed multiple times without loops just by using keyword called ‘invocation count.’
- Using testng, you can execute multiple test cases on multiple browsers, i.e., cross browser testing.
- The TestNG framework can be easily integrated with tools like TestNG Maven, Jenkins, etc.
- Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest
- TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.
- Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.
- WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format like the one shown below.



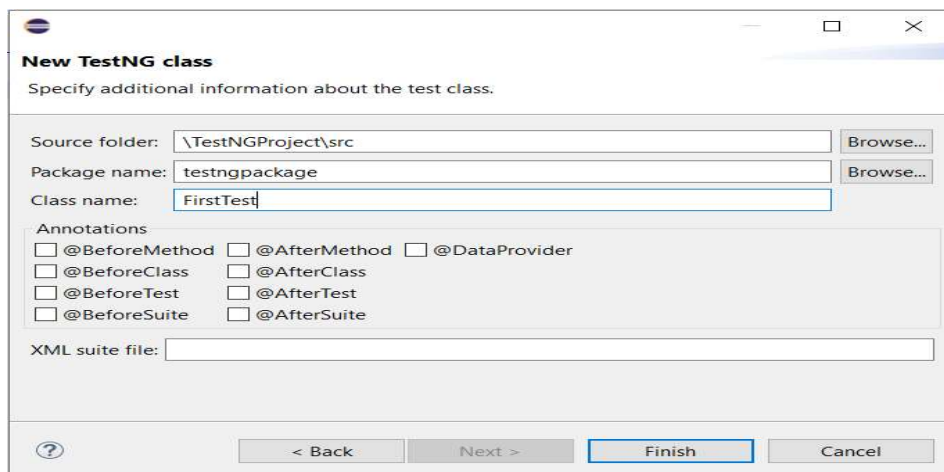
Creating your first TestNG class

Perform the following steps to create your first TestNG class:

1. Go to File | New | Other. This will open a new Add wizard window in Eclipse.
2. Select TestNG class from the Add wizard window and click on Next.



3. On the next window click on the Browse button and select the Java project here you need to add your class.



4. Enter the package name and the test class name and click on Finish.

This window also gives you an option to select different annotations while creating a new TestNG class.

If selected, the plugin will generate dummy methods for these annotations while generating the class.

This will add a new TestNG class to your project.

Write the following code to your newly created test class:

```
package test;

import org.testng.annotations.Test;

public class FirstTest {
    @Test
    public void testMethod() {
        System.out.println("First TestNG test");
    }
}
```

Understanding testng.xml

testng.xml is a configuration file for TestNG.

It is used to define test suites and tests in TestNG. It is also used to pass parameters to test methods.

testng.xml provides different options to include packages, classes, and independent test methods in our test suite.

It also allows us to configure multiple tests in a single test suite and run them in a multithreaded environment.

TestNG allows you to do the following:

Create tests with packages

Create tests using classes

Create tests using test methods

Include/exclude a particular package, class, or test method

Use of regular expression while using the include/exclude feature

Store parameter values for passing to test methods at runtime

Configure multithreaded execution options

Creating a test suite

Let's now create our first TestNG test suite using testng.xml. We will create a simple test suite with only one test method.

Perform the following steps for creating a test suite:

1. Go to the Eclipse project that we created in the previous chapter.
2. Select the project and then right-click on it and select New | File.
3. Select the project in the File window
4. Enter text testng.xml in the File name section, and click on Finish.
5. Eclipse will add the new file to your project and will open the file in the editor.

Add the following snippet to the newly created testng.xml file and save it.

```
<suite name="First Suite" verbose="1" >
    <test name="First Test" >
```

```
<classes>
    <class name="test.FirstTest" />
</classes>
</test>
</suite>
```

Running testng.xml

Perform the following steps for executing testng.xml using Eclipse:

1. Open Eclipse and go to the project where we have created the testng.xml file.
2. Select the testng.xml file, right-click on it, and select Run As | TestNG suite.
3. Eclipse will execute the XML file as TestNG suite and you can see the report in Eclipse

What is Annotation in TestNG?

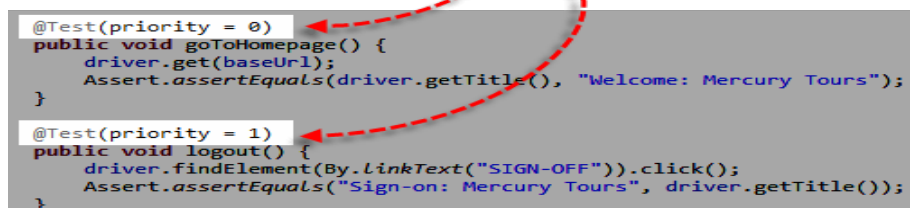
Annotations in TestNG are a powerful feature that allow you to control the flow of test execution. They define how methods in a test class are executed and provide a way to configure the test suite for various tasks such as setting up environments, running tests, cleaning up, and grouping test cases. These annotations help organize the tests and manage dependencies between test methods. TestNG defines its own annotations and builds an execution framework by using it.

Annotations in TestNG are lines of code that can control how the method below them will be executed.

They are always preceded by the @ symbol.

A very early and quick TestNG example is the one shown below.

These are 2 examples of annotations



```
@Test(priority = 0)
public void goToHomepage() {
    driver.get(baseUrl);
    Assert.assertEquals(driver.getTitle(), "Welcome: Mercury Tours");
}

@Test(priority = 1)
public void logout() {
    driver.findElement(By.LinkText("SIGN-OFF")).click();
    Assert.assertEquals("Sign-on: Mercury Tours", driver.getTitle());
}
```

The example above simply says that the method goToHomepage() should be executed first before logout() because it has a lower priority number

Here are some commonly used TestNG annotations:

1. **@Test**: Marks a method as a test method. This is the most essential annotation in TestNG.
2. **@BeforeMethod**: Indicates that a method should be run before each test method in the

current class. It's useful for setting up preconditions.

3. **@AfterMethod**: Indicates that a method should be run after each test method in the current class, typically used for cleanup operations.
4. **@BeforeClass**: Runs once before any of the test methods in the current class. This is often used for setup that is needed for all tests in that class.
5. **@AfterClass**: Runs once after all the test methods in the current class have been executed. It's useful for teardown operations related to the class.
6. **@BeforeSuite**: Runs once before any tests in a test suite, making it ideal for suite-level setup.
7. **@AfterSuite**: Runs once after all tests in a suite have been executed, typically used for cleanup operations at the suite level.
8. **@DataProvider**: Allows you to run a test method multiple times with different sets of data, useful for data-driven testing.
9. **@Parameters**: Used to pass parameters to test methods from XML configuration files.
10. **@Factory**: Marks a method as a factory that returns an array of class objects (Object[]). These class objects will then be used as test classes by TestNG. This is used to run a set of test cases with different values.

Implementation

1. Create two packages in TestNg project, each package containing two classes. Each class will contain two test methods. Create and execute a TestNG test suite(testng.xml) to include first test method from each class.

FirstPackage:

FirstClass:

```
package firstPackage;
```

```
import org.testng.annotations.Test;
```

```
public class firstClass {
```

```
    @Test
```

```
    public void firstMethod() {
```

```
        System.out.println("This is First Method in First Class of First Package");
```

```
}
```

```
@Test
```

```
public void secondMethod() {
```

```
    System.out.println("This is Second Method in First Class of First Package");
```

```
}
```

```
}
```

SecondClass:

```
package firstPackage;
```

```
import org.testng.annotations.Test;
```

```
public class secondClass {
```

```
    @Test
```

```
    public void firstMethod() {
```

```
        System.out.println("This is First Method in Second Class of First Package");
```

```
    }
```

```
    @Test
```

```
    public void secondMethod() {
```

```
        System.out.println("This is Second Method in Second Class of First Package");
```

```
    }
```

```
}
```

SecondPackage:

FirstClass:

```
package secondPackage;
```

```
import org.testng.annotations.Test;
```

```
public class firstClass {  
    @Test  
    public void firstMethod() {  
        System.out.println("This is First Method in First Class of Second Package");  
    }  
  
    @Test  
    public void secondMethod() {  
        System.out.println("This is Second Method in First Class of Second Package");  
    }  
}
```

SecondClass:

```
package secondPackage;
```

```
import org.testng.annotations.Test;
```

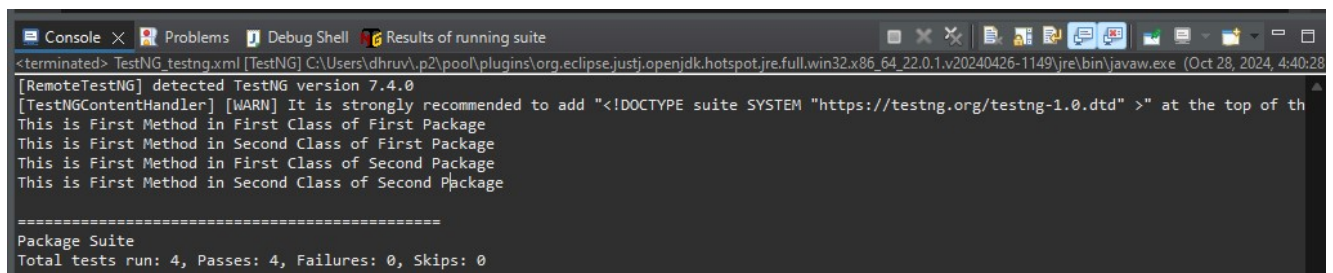
```
public class secondClass {  
    @Test  
    public void firstMethod() {  
        System.out.println("This is First Method in Second Class of Second Package");  
    }  
  
    @Test  
    public void secondMethod() {  
        System.out.println("This is Second Method in Second Class of Second Package");  
    }  
}
```

Testng.xml:

```
<suite name="Package Suite" verbose="1">
```

```
<test name="Package Test">
  <classes>
    <class name="firstPackage.firstClass">
      <methods>
        <include name="firstMethod"/>
      </methods>
    </class>
    <class name="firstPackage.secondClass">
      <methods>
        <include name="firstMethod"/>
      </methods>
    </class>
    <class name="secondPackage.firstClass">
      <methods>
        <include name="firstMethod"/>
      </methods>
    </class>
    <class name="secondPackage.secondClass">
      <methods>
        <include name="firstMethod"/>
      </methods>
    </class>
  </classes>
</test>
</suite>
```

Output:



```
<terminated> TestNG_testng.xml [TestNG] C:\Users\dhruv\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (Oct 28, 2024, 4:40:28)
[RemoteTestNG] detected TestNG version 7.4.0
[TestNGContentHandler] [WARN] It is strongly recommended to add "<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >" at the top of th
This is First Method in First Class of First Package
This is First Method in Second Class of First Package
This is First Method in First Class of Second Package
This is First Method in Second Class of Second Package

=====
Package Suite
Total tests run: 4, Passes: 4, Failures: 0, Skips: 0
=====
```


2. Create a test class with all kinds of Before and After annotations and execute it using a testng.xml.

Annotations.java:

```
package demo;
```

```
import org.testng.annotations.AfterClass;
```

```
import org.testng.annotations.AfterMethod;
```

```
import org.testng.annotations.AfterSuite;
```

```
import org.testng.annotations.AfterTest;
```

```
import org.testng.annotations.BeforeClass;
```

```
import org.testng.annotations.BeforeMethod;
```

```
import org.testng.annotations.BeforeSuite;
```

```
import org.testng.annotations.BeforeTest;
```

```
import org.testng.annotations.Test;
```

```
public class AnnotationsTesting {
```

```
    @BeforeSuite
```

```
    public void beforeSuite() {
```

```
        System.out.println("Before Suite Method");
```

```
    }
```

```
    @AfterSuite
```

```
    public void afterSuite() {
```

```
        System.out.println("After Suite Method");
```

```
}
```

```
@BeforeTest
```

```
public void beforeTest() {
```

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

```
}
```

```
@AfterTest
```

```
public void afterTest() {
```

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

```
}
```

```
@BeforeClass
```

```
public void beforeClass() {
```

```
    System.out.println("Before Class Method");
```

```
}
```

```
@AfterClass
```

```
public void afterClass() {
```

```
    System.out.println("After Class Method");
```

```
}
```

```
@BeforeMethod
```

```
public void beforeMethod() {
```

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

```
}
```

```
@AfterMethod
```

```
public void afterMethod() {
```

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

```
}
```

```
@Test
```

```
public void testOneMethod() {
```

```
    System.out.println("Test One Method");
```

```
}
```

```
@Test
```

```
public void testTwoMethod() {
```

```
    System.out.println("Test Two Method");
```

```
}
```

```
@Test
```

```
public void testThreeMethod() {
```

```
    System.out.println("Test Three Method");
```

```
}
```

```
}
```

testng-annotations.xml:

```
<suite name="Annotations Suite" verbose="1">
```

```
    <test name="First Test Case">
```

```
<classes>

  <class name="demo.AnnotationsTesting">

    <methods>

      <include name="testOneMethod"/>

      <include name="testTwoMethod"/>

    </methods>

  </class>

</classes>

</test>

<test name="Second Test Case">

  <classes>

    <class name="demo.AnnotationsTesting">

      <methods>

        <include name="testThreeMethod"/>

      </methods>

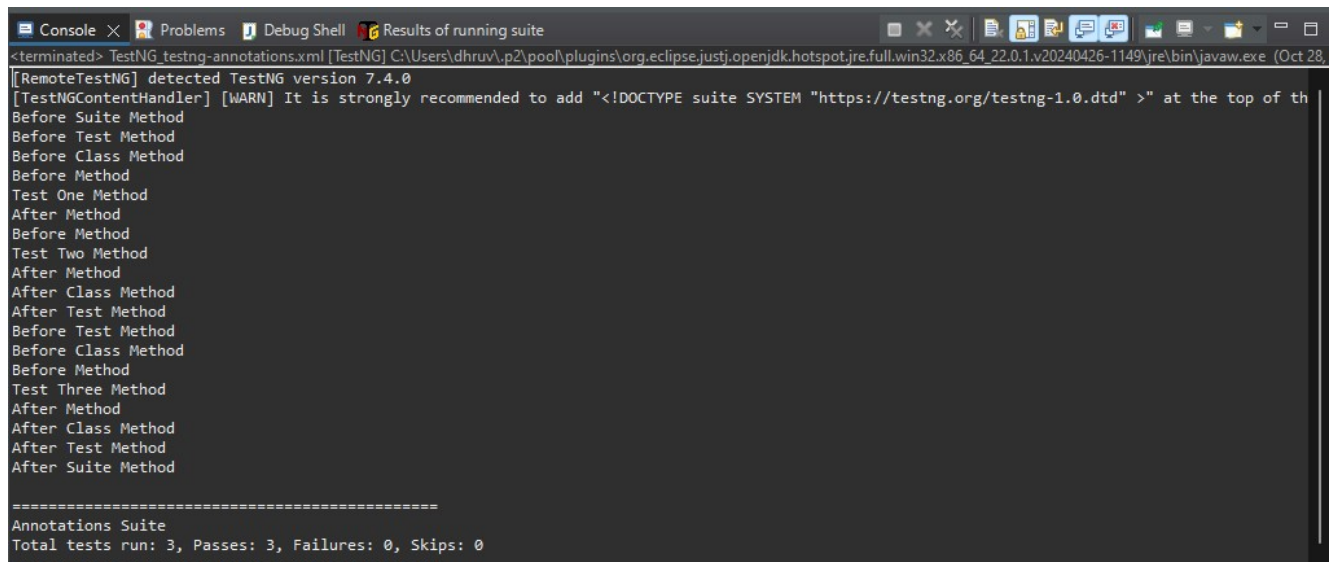
    </class>

  </classes>

</test>

</suite>
```

Output:



The screenshot shows the Eclipse IDE's console window with the following output:

```
<terminated> TestNG_testng-annotations.xml [TestNG] C:\Users\dhruv\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (Oct 28, 2024 10:28:15 AM)
[RemoteTestNG] detected TestNG version 7.4.0
[TestNGContentHandler] [WARN] It is strongly recommended to add "<!DOCTYPE suite SYSTEM 'https://testng.org/testng-1.0.dtd' >" at the top of the suite XML file.
Before Suite Method
Before Test Method
Before Class Method
Before Method
Test One Method
After Method
Before Method
Test Two Method
After Method
After Class Method
After Test Method
Before Test Method
Before Class Method
Before Method
Test Three Method
After Method
After Class Method
After Test Method
After Suite Method

=====
Annotations Suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
=====
```

3. Create TestNG class containing three test methods using test annotation out of which any two methods are enabled and remaining method is disabled. Use appropriate attributes of test annotation.

Enabled.Java:

```
package demo;
```

```
import org.testng.annotations.Test;

@Test(enabled = true)
public class EnableTest {

    public void testMethodOne() {
        System.out.println("Test Method One");
    }

    @Test(enabled = true)
    public void testMethodTwo() {
        System.out.println("Test Method Two");
    }

    @Test(enabled = false)
    public void testMethodThree() {
        System.out.println("Test Method Three");
    }
}
```

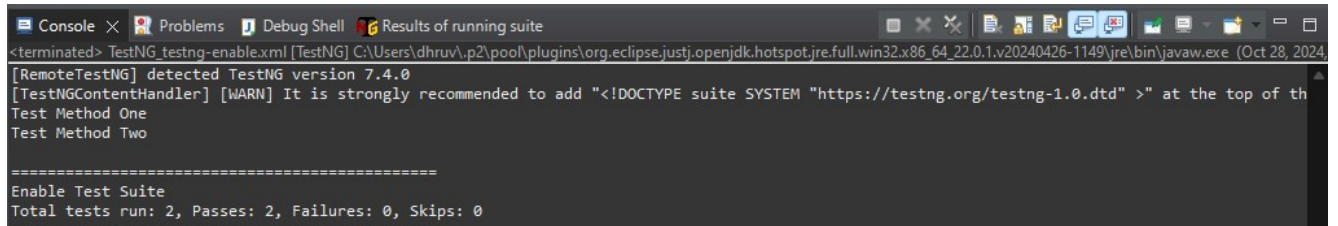
testng-enabled.xml:

```
<suite name="Enable Test Suite" verbose="1">
  <test name="enabled Test">
    <classes>
      <class name="demo.EnableTest"/>
    </classes>
```

```
</test>
```

```
</suite>
```

Output:

A screenshot of the Eclipse IDE's console window. The title bar shows 'Console', 'Problems', 'Debug Shell', and 'Results of running suite'. The console output shows the execution of a TestNG suite. It starts with a warning from RemoteTestNG about the version (7.4.0) and a recommendation to add a DOCTYPE declaration to the XML file. The output then shows 'Test Method One' and 'Test Method Two' being executed. Finally, it displays a summary: 'Enable Test Suite', 'Total tests run: 2, Passes: 2, Failures: 0, Skips: 0'.

4. Write a TestNg script and configure xml file to prioritize test methods.

Priority.java:

```
package demo;
```

```
import org.testng.annotations.Test;
```

```
public class PriorityTest {
```

```
    @Test(priority = 2)
```

```
    public void FirstMethod() {
```

```
        System.out.println("This is First Method");
```

```
    }
```

```
    @Test(priority = 3)
```

```
    public void SecondMethod() {
```

```
        System.out.println("This is Second Method");
```

```
    }
```

```
    @Test(priority = 1)
```

```
    public void ThirdMethod() {
```

```
        System.out.println("This is Third Method");
```

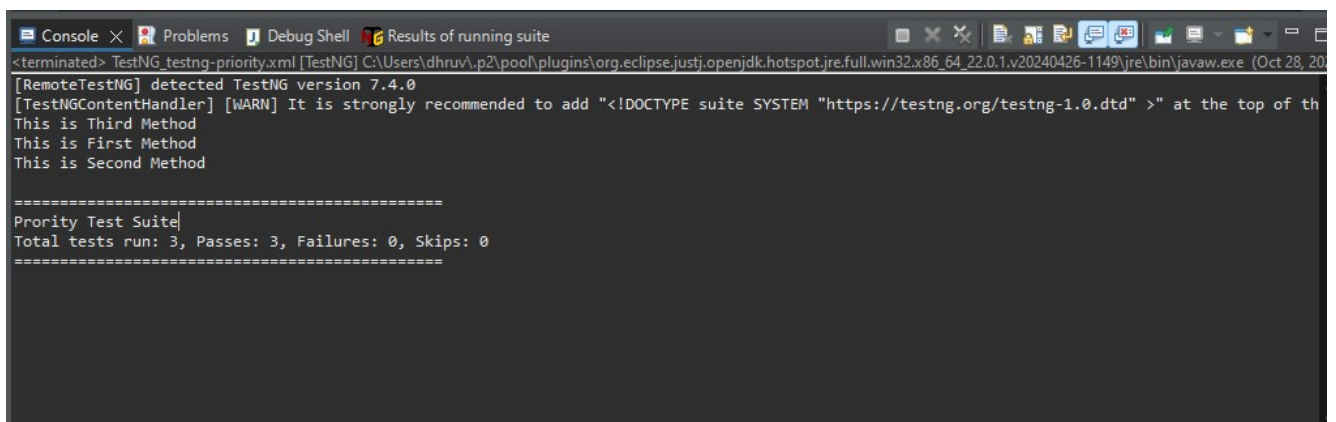
```
    }
```

```
}
```

Testng-priority.xml:


```
<suite name="Priority Test Suite" verbose="1">
  <test name="Priority Test">
    <classes>
      <class name="demo.PriorityTest"/>
    </classes>
  </test>
</suite>
```

Output:

A screenshot of an IDE console window titled 'Console'. The output shows TestNG version 7.4.0 detected, a warning about the DOCTYPE, and the execution of a 'Priority Test Suite'. The suite contains three methods: 'This is Third Method', 'This is First Method', and 'This is Second Method'. The final summary indicates 'Total tests run: 3, Passes: 3, Failures: 0, Skips: 0'.

```
<terminated> TestNG: testng-priority.xml [TestNG] C:\Users\dhruv\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (Oct 28, 2024)
[RemoteTestNG] detected TestNG version 7.4.0
[TestNGContentHandler] [WARN] It is strongly recommended to add "<!DOCTYPE suite SYSTEM 'https://testng.org/testng-1.0.dtd' >" at the top of the suite file
This is Third Method
This is First Method
This is Second Method

=====
Priority Test Suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
=====
```

Conclusion: Understood how to test application using TestNG frameworks.

After performing this Practical/lab, students are expected to answer following questions

Q.1 What is TestNG Framework?

Q.2 What is testing.xml file?