**Task #4**

## **What is TestNG?**

TestNG is an open-source advanced testing framework built on the capabilities of JUnit and NUnit. The framework is not only robust but also provides awesome features like [TestNG annotations](https://www.lambdatest.com/blog/complete-guide-on-testng-annotations-for-selenium-webdriver/" \t "https://www.lambdatest.com/blog/testng-in-selenium/_blank), grouping, parameterizations, etc., that help keep tests well-structured and easy to maintain.

The TestNG framework is designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

**Salient features of TestNG:**

* Provides ****HTML Reports**** in an efficient easy to read format, thus catering to WebDriver’s limitation to generate such reports
* Support to ****Group**** multiple test cases to be executed together whenever that particular group is invoked
* Ability to set ****Priority**** among test cases which help to decide which case has to be executed first
* Provides the ability to execute only the failed cases with the help of ****testng-failed.xml****
* Support for [Cross Browser testing](https://www.lambdatest.com/cross-browser-testing" \t "https://www.lambdatest.com/blog/testng-in-selenium/_blank), ****Data Parameterization****, and ****Parallel testing****
* Easy to understand ****Annotations****, which helps control the sequence of execution in automation script without a static main method****Handles Uncaught Exceptions**** inherently, thus preventing sudden test termination.

**TestNG Annotations and Attributes**

Annotation means a note or a comment in any document that is used to provide it certain meaning. TestNG annotations also serve a similar purpose. These are used to provide meaning to any function in the test script and describe its behavior.

Annotations in TestNG are some predefined keywords that help to control the execution of the automation script in a proper flow. One must note, any test which is not annotated in TestNG is ignored while running tests. So it is mandatory to add required annotations for the test cases. Here is one example to understand this.

UnAnnotatedTestExample.java

package test;

import org.testng.annotations.Test;

public class UnAnnotatedTestExample {

@Test

public void testWithAnnotation()

{

System.out.println("This test is annotated");

}

public void testWithoutAnnotation()

{

System.out.println("This test is not annotated");

}

}

package test;

import org.testng.annotations.Test;

public class UnAnnotatedTestExample {

@Test

public void testWithAnnotation()

{

System.out.println("This test is annotated");

}

public void testWithoutAnnotation()

{

System.out.println("This test is not annotated");

}

}

As you can see, in the above example, we have added two test cases, one with @Test annotation and one without it. So the expectation here is only the one with annotation is executed and another one is skipped. Let’s execute the class and verify this.

**TestNG Annotations -**

It is clear from the output, that only the case with annotation got executed. One more thing to note here is that the Test run on the output also shows the total number of test cases as 1 since TestNG reads the number of test cases from @Test annotation and we had only one such case in the file.

TestNG annotations also provide relevant information about the method, class, or test suite they are associated with. These are represented by adding @ prefix. Let us have a look at all these annotations one by one, which we will use in this Selenium TestNG tutorial.

**Types of Annotations**

TestNG currently supports ten types of annotations.

**@Test**

This is the most important annotation as it covers the actual test case logic and executes the automation run. There are some attributes associated with this annotation that helps to serve various use cases. We will learn these attributes in the following sections.

**@Test**

public void testAnnotation()

{

//Write your test case logic here.

}

**@BeforeTest**

This annotation is executed before running any test with the @Test annotation in a class. One example of using this can be to maximize the browser for test cases in a class.

**@BeforeTest**

public void browserSetup()

{

//Code to maximize the browser

}

**@AfterTest**

This annotation is executed after all the test cases in a class are executed. One use case of this annotation would be to write the method to compile a test run report after all tests are run.

**@AfterTest**

public void compileReport()

{

//code to compile a report

}

***@BeforeMethod***

This annotation is executed before any and every test method, i.e., before every @Test annotation method in the class. For example, we have cases to test some dashboard that comes up after login. So instead of writing login steps in each test case, @BeforeMethod annotation can be used to write those steps once in a method to be called before every test case.

**@BeforeMethod**

public void doLogin()

{

//code to do login

}

**@AfterMethod**

This annotation is executed after any and every test method, i.e., after every @Test annotation method in the class. In continuation to the above example, let’s say we need to log out after every test case. In such a scenario, this annotation would help.

**@AfterMethod**

public void doLogout()

{

//code to do logout}

**@BeforeClass**

This annotation is executed before the first test method with @Test annotation in the class is executed. For example, this can be used to navigate to the URL before executing the test.

**@BeforeClass**

public void navigateUrl()

{

//code to navigate to URL

}

**@AfterClass**

This annotation is executed after the last @Test method is executed. The method with this annotation can be used to close the driver after the automation run.

**@AfterClass**

public void closeDriver()

{

//code to close the driver

}

**@BeforeSuite**

This annotation marks the entry point of any automation script execution on TestNG in Selenium. Any method with this annotation is executed before any test in all the classes in that suite. This can be used to do generic setup steps for the execution, like initializing WebDriver for execution.

**@BeforeSuite**

public void initializeDriver()

{

//code to initialize the driver

}

**@AfterSuite**

This annotation in TestNG is the last to be executed after all test methods of all the classes within the suite have been executed. A method with this annotation is mostly used for cleanup activities like closing the active driver sessions.

**@AfterSuite**

public void cleanUp()

{

//code to close all sessions

}

**@BeforeGroups**

TestNG provides the capability to group the tests with similar functionalities in a group with the help of the group attribute of @Test annotation. This helps when we want to execute only a particular type of test case. The method with @BeforeGroup annotation is executed before the first test case for the given group is executed. We can mention the group name inside brackets next to the annotation.

**@BeforeGroups("groupName")**

public void startGroupExecution()

{

//code to execute before particular group test cases.

}

**@BeforeGroups("groupName")**

public void startGroupExecution()

{

//code to execute before particular group test cases.

}

@AfterGroups

This annotation in TestNG is executed after all the test cases of a given group are executed.

**@AfterGroups("groupName")**

public void stopGroupExecution()

{

//code to execute after particular group test cases.

}

**@AfterGroups("groupName")**

public void stopGroupExecution()

{

//code to execute after particular group test cases.

}

With this, we have understood all the annotations used in TestNG. These annotations are highly usable as Very easy to learn as there are no rules on which annotation to use with which method. The tester can use them as per free will and understanding.

Ease of parametrization, which helps in creating groups and other tasks.

Strongly typed, which saves time by highlighting errors while implementing only.

Saves the time and effort to extend classes to define the order of execution, unlike JUnit…