**TestNG**

1. TestNG is a unit testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use.
2. A unit may be function or method or module. Unit Testing is a software testing techniques where individual units or components of the software are tested to verify that they function as expected.
3. TestNG provides you full control over the test cases and the execution of the test cases. Due to this reason, TestNG is also known as a testing framework.
4. Created by Cedric Beust, it is used more frequently by developers and testers in test case creation owing to its ease of using multiple annotations, grouping, dependence, prioritization, and parametrization features..
5. If you want to run a test case A before that as a pre-request you need to run multiple test cases before you begin a test case A. You can set and map with the help of TestNG so that pre-request test cases run first and then only it will trigger a test case A. In such way, you can control the test cases.

### **What is TestNG in Selenium?**

TestNG provides advanced features such as annotations, data-driven testing, test sequencing, and parallel testing to help you organize and execute your Selenium tests more efficiently and effectively.

**Some of the benefits of using TestNG in Selenium**:

1. Group test cases into logical units, making managing and maintaining your test suite easier.
2. Run tests in parallel, significantly reducing the time it takes to execute your test suite.
3. TestNG provides a wide range of annotations that you can use to customize your tests, such as **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeMethod, and @AfterMethod**.
4. It supports data-driven testing, allowing you to run the same test case with multiple test data sets.
5. Better reporting and logging features than other testing frameworks make identifying and debugging issues in your tests easier.
   1. Whenever we execute testNG class, java compiler / Interpreter always looks for @Test annotation method to start the execution.
   2. Without @Test, TestNG class will not be executed. @Test annotation method act like main method in TestNG.
   3. In one TestNG class we can have multiple @Test methods, but each test method should have @Test annotation before method signature.
   4. Annotation method return type should be “void” and access specifier can be public /private/protected/default , but method name can be anything( but we have provide manual Test Name as Test)
   5. TestNG class name should be modulename @Test method name should be manual test case name.
   6. Class name and TestNG method name should end with “Test”.
   7. One manual test case contains multiple steps all those steps should be automated using one @test annotation and test name should be manual test case name and end with Test.
   8. In one class we can keep multiple @Test annotation but in real time we are going to maintain maximum 10 to 15 @test script, because maintenance will be easy.

**Priority**

1. Whenever we execute TestNG class by default all the test method will be executed based on alphabetical order in order the change the order of execution , we go for priority.
2. Make every TC independent and try to execute independent If TC fails the other TC.
3. Instead of creating dependency or priority write insert query for creating which reduces the execution time.
4. Lower priority values are executed first, and tests without a priority specified are assigned a default value of 0.

**import org.testng.annotations.Test;**

**public class PriorityExample {**

**@Test(priority = 2)**

**public void testMethod1() {**

**System.out.println("Test Method 1 executed");**

**}**

**@Test(priority = 1)**

**public void testMethod2() {**

**System.out.println("Test Method 2 executed");**

**}**

**@Test(priority = 3)**

**public void testMethod3() {**

**System.out.println("Test Method 3 executed");**

**}**

**}**

**Depends on method**

1. It is used to create dependency between two test script like “test2” is depends on “test1”.
2. IT helps us to check the test dependent test case pass or fail.
3. It test 1 test script get pass , “test2” execution will continue
4. If “test1” fails skip the “test2” execution.
5. @Test ( dependsonmethod=createmethod)

**import org.testng.annotations.Test;**

**public class DependsOnMethodExample {**

**@Test**

**public void login() {**

**System.out.println("Login test executed");**

**}**

**@Test(dependsOnMethods = {"login"})**

**public void placeOrder() {**

**System.out.println("Place Order test executed");**

**}**

**@Test(dependsOnMethods = {"placeOrder"})**

**public void logout() {**

**System.out.println("Logout test executed");**

**}**

**}**

**Invocation** **count**

1. Same test script executed with multiple times with same test data.
2. If one of the iteration fails it will not stops execution and it will not stops execution and it will get into next iteration.
3. It will use same data again and again.
4. Default Value: The default is 1, meaning the test runs only once unless specified.
5. @Test(InvocationCount=10)

**import org.testng.annotations.Test;**

**public class InvocationCountExample {**

**@Test(invocationCount = 3)**

**public void testMethod() {**

**System.out.println("Test method executed");**

**}**

**}**

**Data Provider**

1. In order to execute same test case multiple times with different testdata , we go for @Dataprovider annotation.
2. Data provider annotation always return two-dimensional object array, because we can pass any type of datatype.
3. Data-provider annotation help us to execute same test mulitiple times with the different set of data each @dataprovider annotation.
4. Data-provider annotation play major role in data driven framework where we need to test the application with huge amount of data like ecommerce , booking and banking application.

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class DataProviderExample {

// Define a DataProvider

@DataProvider(name = "loginData")

public Object[][] provideData() {

return new Object[][] {

{"user1", "pass1"}, // First set of data

{"user2", "pass2"}, // Second set of data

{"user3", "pass3"} // Third set of data

};

}

// Test method using DataProvider

@Test(dataProvider = "loginData")

public void loginTest(String username, String password) {

System.out.println("Testing with Username: " + username + ", Password: " + password);

}

}

**Enable**

In the context of TestNG, enabled=false is an attribute used in the @Test annotation to disable a test case. When a test is marked with enabled=false, it is ignored during the test execution and won't be run.

Why use enabled=false?

* To temporarily skip tests without deleting or commenting them.
* Useful during debugging or when a test is incomplete.

import org.testng.annotations.Test;

public class TestExample {

@Test(enabled = true) // This test will run

public void test1() {

System.out.println("Test 1 is running");

}

@Test(enabled = false) // This test will be skipped

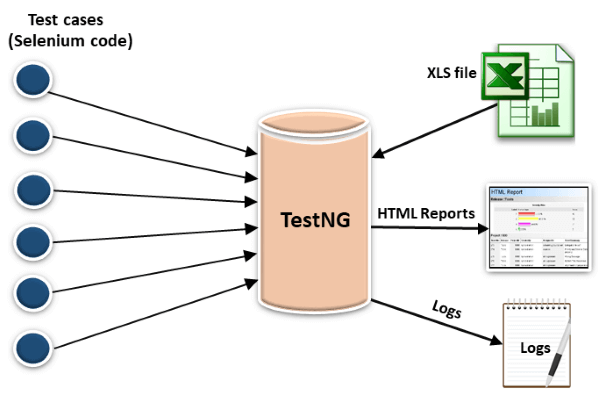
public void test2() {

System.out.println("Test 2 is running");

}

}

Advantages of TestNG over Junit



* It produces the HTML reports for implementation.
* It also generates the Logs.
* TestNG enables you to group the test cases easily which is not possible in JUnit.
* TestNG supports three additional levels such as @Before/After suite, @Before/AfterTest, and Before/AfterGroup.
* Parallel execution of test cases, i.e., running multiple test cases is only possible in the TestNG framework.
* Batch Execution of test cases.

|  |  |  |
| --- | --- | --- |
| Sl no | Test NG | Junit |
| 1 | Flexible test configuration and execution options | Not flexible |
| 2 | It supports powerful features such as test group, Batch and parallel execution , prioritization, and test dependencies | Not available |
| 3 | the DataProvider annotation to pass data to a test method | JUnit requires you to write custom code to handle data-driven testing |
| 4 | TestNG generates HTML reports with detailed information about test execution, including test results, failures, and errors | generates simple text-based reports |
| 5 | advanced annotations such as **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeGroups, @AfterGroups, @BeforeClass, and @AfterClass** | @Test , @B, @A |
| 6 | TestNG needs to be installed explicitly | Junit is default |

### **Why use TestNG with Selenium?**

One of the drawbacks of Selenium is that it does not have a proper format for the test results. By using the TestNG framework in Selenium, you can:

* Generate the report in a proper format.
* Include the number of test cases run; tests passed, failed, and skipped in the report.
* Group test cases by converting them to testing.xml
* Use invocation count and execute multiple tests without using loops
* Perform [cross browser testing](https://www.browserstack.com/cross-browser-testing)
* Easily understand annotations

### **TestNG Annotations**

An annotation tag provides information about the method, class, and suite. It helps to define the execution approach of your test cases and the different features associated with it. Below are the major annotations used:

* ***@Test***– This is the root of TestNG test cases. To use TestNG, all methods should be annotated with this annotation. Below is an example:

@Test

public void setupTestNG()

{

System.out.println(“this is a test annotation method”)

}

A few attributes associated with the TestNG annotation are:

1. **Description**: You can describe your test case under the description, stating what it does

@Test(description=”This test validates login functionality”)

1. **Priority**: You can [prioritize the order of your test methods in TestNG](https://www.browserstack.com/guide/prioritizing-tests-in-testng-with-selenium) by defining a priority. Based on the defined priority, the test shall execute in that order.

@Test(priority=1)

1. **DependsOnMethod**: This attribute works miracles if one test case is dependent on another. For example, to view your profile details, you need to login to the application. So, your profile test case is dependent on the login test case

@Test(dependsOnMethod=”Login”)

1. **Enabled**: Using this attribute, you can choose to execute or skip the execution of this test case. Setting it to true execute it and putting it to false will skip the test from the execution cycle

@Test(enabled=’true’)

1. **Groups**: Using this attribute, you can club your test cases into a single group and specify the group you wish to execute in your TestNG XML file. The test cases clubbed to that group will only be executed, and the rest will be skipped

@Test(groups=”Smoke Suite”)

While the above ones should help you get started, other major annotations are:

* **@BeforeMethod and @AfterMethod** – These annotations run before and after each test method
* **@BeforeClass and @AfterClass** – These annotations run once before and after the first*@test* method in a class
* **@BeforeTest and @AfterTest** – The BeforeTest annotation runs before the *@BeforeClass* annotation and the AfterTest annotation runs after the *@AfterClass* annotation
* **@BeforeSuite and @AfterSuite**– These annotations run before and after any test annotated method in a class respectively. These annotations start the beginning of a test and the end of it, for all the classes in a suite

Talking about the execution order of these annotations, they execute in the below order:

***@BeforeSuite -> @BeforeTest -> @BeforeClass -> @BeforeMethod -> @Test -> @AfterMethod -> @AfterClass -> @AfterCTest -> @AfterSuite***

### **TestNG Assertions**

Like JUnit, TestNG provides multiple-level assertions to validate your actual results against your expected results. Few of the commonly used assertions are:

1. **assertTrue**– This assertion verifies whether the defined condition is true or not. If true, it will pass the test case. If not, it will fail the test case

Assert.assertTrue(condition);

1. **assertFalse**– This assertion verifies whether the defined condition is false or not. If false, it will pass the test case. If not, it will fail the test case

Assert.assertFalse(condition);

1. **assertEquals**– This assertion compares the expected value with the actual value. If both are the same, it passes the test case. If not, it fails the test case. You can compare strings, objects, integer values etc. using this assert

Assert.assertEquals(actual,expected);

1. **assertNotEquals**: This is just opposite to what assertEquals does. If actual matches the expected, the test case fails, else the test case passes

Assert.assertNotEquals(actual,expected,Message)

### **What is Parameterization in TestNG?**

Parameterization is a powerful feature of TestNG that allows you to write more efficient and effective tests by reusing the same test logic with different input data.. It is useful when you need to test a particular functionality with different input values to ensure that it works correctly in all scenarios.

**Batch execution in TestNG**

1. Collection of multiple test script is called batch, execute multiple test script through xml in a single click is called batch execution.
2. In order to achieve batch execution we go for TestNG xml configuration file.
3. TestNG xml file always start with suite xml tag followed by test and classes tag
4. In one xml file we can invoke n number of TestNG classes but all the classes should be present within a project.
5. All the class should followed by packageName.classname.

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

**<suite name="Batch Execution Suite" parallel="false">**

**<test name="Test Batch">**

**<classes>**

**<class name="com.example.tests.TestClass1"/>**

**<class name="com.example.tests.TestClass2"/>**

**<class name="com.example.tests.TestClass3"/>**

**</classes>**

**</test>**

**</suite>**

**Group Execution in TestNG**

1. Collection of similar / group related test scripts across the TestNG classes is called grouping execution.
2. In order to achieve group execution each and every test script should have group name and it should be written along with @Test annotation.
3. In group execution , all configure annotation should have group name, otherwise those annotation will not participate in group execution.
4. In order to achieve group execution should declare group key in Testng.xml file and group key should be declared before test tag and after suite tag.

**import org.testng.annotations.Test;**

**public class TestClass1 {**

**@Test(groups = "smoke")**

**public void smokeTest() {**

**System.out.println("Smoke Test from TestClass1");**

**}**

**@Test(groups = "regression")**

**public void regressionTest() {**

**System.out.println("Regression Test from TestClass1");**

**}**

**}**

**<suite name="Grouped Test Suite">**

**<test name="Smoke Tests">**

**<groups>**

**<run>**

**<include name="smoke"/>**

**</run>**

**</groups>**

**<classes>**

**<class name="com.example.tests.TestClass1"/>**

**<class name="com.example.tests.TestClass2"/>**

**</classes>**

**</test>**

**</suite>**

**Distributed Parallel execution**

1. Distribute the TC across the threads execute all threads parallelly ( in same browser)
2. Distribute the test case across the multiple test runner and execute each <test> Test runner in parallel is called distributed parallel execution
3. We reduce the suite execution time so that we can getthe result early.
4. In order to achieve parallel execution we should enable parallel = “tests” and thread.count=5 in <suite>, then create multiple test runner and distribute the test case.
5. Maximum thread count is 5
6. Thread count should be same as number of <test> testRunner.

| **Scenario** | **Description** | **TestNG Configuration** | **Behavior** |
| --- | --- | --- | --- |
| **Same Browser in Parallel** | Tests run on the same browser type (e.g., Chrome) but in **separate browser instances**. | parallel="methods" or parallel="tests" | Multiple test methods or test groups are executed in parallel **on the same browser**, using different instances (e.g., opening multiple tabs or windows in the same browser). |
| **Different Browsers in Parallel** | Tests run on **different browsers** (e.g., Chrome, Firefox) in parallel. | parallel="tests" or parallel="classes" | Test methods or test groups are executed in parallel on different browsers, each browser running in its own thread. |