**Frame works**

**TestNG:**

It is an open source automated testing framework; where **NG** of test**NG** means **N**ext **G**eneration. TestNG is similar to JUnit but it is much more powerful than JUnit but still it’s inspired by JUnit. It gives the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parametrizing.

The major advantages of TestNG are :

1. It gives the ability to produce ***HTML Reports*** of execution
2. ***Annotations***made testers life easy
3. Test cases can be ***Grouped & Prioritized*** more easily
4. ***Parallel***testing is possible
5. Generates ***Logs***
6. Data ***Parameterization***is possible

Annotations in TestNG

**@BeforeSuite**: The annotated method will be run before all tests in this suite have run.

**@AfterSuite**: The annotated method will be run after all tests in this suite have run.

**@BeforeTest**: The annotated method will be run before any test method belonging to the classes inside the tag is run.

**@AfterTest**: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

**@BeforeGroups**: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

**@AfterGroups**: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

**@BeforeClass**: The annotated method will be run before the first test method in the current class is invoked.

**@AfterClass**: The annotated method will be run after all the test methods in the current class have been run.

**@BeforeMethod**: The annotated method will be run before each test method.

**@AfterMethod**: The annotated method will be run after each test method.

**@Test**: The annotated method is a part of a test case.

## Test Case Grouping

‘**Groups**‘ is one more annotation of TestNG which can be used in the execution of multiple tests. Let’s say you have hundred tests of class vehicle and in it ten method of car, ten method of scooter and so on. You probably like to run all the scooter tests together in a batch. And you want all to be in a single test suite. With the help of grouping you can easily overcome this situation.

@Test (groups = { "Car", "Sedan Car" })

  public void Sedan1() {

  System.out.println("Batch Sedan Car - Test sedan 1");

  }

**TestNG.xml**

<suite name="Suite">

   <test name="Practice Grouping">

      <groups>

         <define name="All">

   <include name="Car"/>

   <include name="Scooter"/>

</define>

<run>

   <include name="All"/>

</run>

  </groups>

<classes>

      <class name="automationFramework.Grouping" />

</classes>

   </test>

</suite>

## Dependent Test:

TestNG allows you to specify dependencies either with:

* Using attributes *dependsOnMethods* in @Test annotations OR
* Using attributes *dependsOnGroups* in @Test annotations.

@Test (dependsOnMethods = { "SignIn" })

## InvocationCount

* The number of times this method should be invoked.

## Skipping a Test Case

Think of a situation where you are required to skip one or more @Test from your testng class. In testng, you can easily able to handle this situation by setting the ‘enabled’ parameter to ‘false’ for e.g.:

## @Test(enabled = false)

## TestNG Parameters:

## ****Parameters****. TestNG lets you pass parameters directly to your test methods with your testng.xml.

## @Test

## @Parameters({ "sUsername", "sPassword" })

## Public void test(String sUsername, String sPassword) { }

## TestNG XML

## <parameter name="sUsername" value="testuser\_1"/>

## <parameter name="sPassword" value="Test@123"/>

## TestNG DataProviders

1)  Define the method credentials() which is defined as a Dataprovider using the annotation. This method returns array of object array.

2) Add a method test() to your DataProviderTest class. This method takes two strings as input parameters.

3) Add the annotation *@Test(dataProvider = “Authentication”)* to this method. The attribute dataProvider is mapped to “Authentication”.

 @DataProvider(name = "Authentication")

  public static Object[][] credentials() {

        return new Object[][] { { "testuser\_1", "Test@123" }, { "testuser\_1", "Test@123" }};

  }

  // Here we are calling the Data Provider object with its Name

  @Test(dataProvider = "Authentication")

  public void test(String sUsername, String sPassword) {

## TestTestNG Listeners

**ISuiteListener**: It has two method in it **onStart()** & **onFinish()**. Whenever a class implements this listener, TestNG guarantees the end-user that it will invoke the methods onStart() and onFinish() before and after running a TestNG Suite. So before TestNG picks up your suite for execution, it first makes a call to onStart() method and runs whatever has been scripted in this method. In a similar way, it again makes a call to onFinish() method after a suite has been run.

**ITestListener**: The working of this listener is also exactly the same as ISuiteListerner but the only difference is that it makes the call before and after the Test not the Suite. It has seven methods in it.

**onFinish()**: Invoked after all the tests have run and all their Configuration methods have been called.

**onStart():**Invoked after the test class is instantiated and before any configuration method is called.

**onTestFailedButWithinSuccessPercentage(ITestResult result):**Invoked each time a method fails but has been annotated with successPercentage and this failure still keeps it within the success percentage requested.

**onTestFailure(ITestResult result):** Invoked each time a test fails.

**onTestSkipped(ITestResult result):**Invoked each time a test is skipped

**onTestStart(ITestResult result):**Invoked each time before a test will be invoked.

**onTestSuccess(ITestResult result):**Invoked each time a test succeeds.

**IInvokedMethodListener**: The working of this listener is also exactly the same as ISuiteListerner & ITestListerner and the only difference is that it makes the call before and after every Method. It has only two methods in it.

**afterInvocattion():**Invoke after each method

**beforeInvocation():** Invoke before each method

## Sites:

## <http://toolsqa.com/selenium-webdriver/testng-introduction/>

## <http://testng.org/doc/documentation-main.html>