**Introduction:**

-TestNg is one of the open resource framework, where NG stands for Next Generation.

-TestNg inspired from JUnit but introduce some new Functionality.

- TestNG is more powerful than JUnit

-Purpose of Java in selenium is to enhance the Tests

-Webdriver is for recognizing the Elements and performing the operations on Elements

-TestNG is unit testing tool which supports both Java and .Net languages

**Benefits Of TestNG Automation Framework:**

* It generates HTML Test Reports
* TestNg annotations are easy to create Test Cases
* Test cases can be grouped(Batch Testing) and prioritiesed(according to the Alphabetic order) more easily by using the TestNG
* Parallel testing is possible
* Parameterization is possible
* Enabling and disabling test cases

**TestNG DataProviders**. When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a **database**, etc…), in such cases parameters can be passed using **Dataproviders**. A **Data Provider** is a method annotated with @**DataProvider**.

Using TestNg we can create test cases, execute test cases, and generate reports in HTML format

There are type of test Ng annotations available @Test, @BeforeMethod, @AfterMethod,

**Manual Test Case EX:**

Test Case ID- TC\_001

Test case Name: Verify title of the Page

Test Case steps: 1. Launch the browser Webdriver driver=new FirefoxDriver();

2. Navigate to Gmail.com driver.get([www.gmail.com](http://www.gmail.com));

String actual =driver.getTitle();Capture the Title of the webpage in reference variable

3. check the page title Assert.assertEquals(actual,”gmail”);

Expected :  
Actual:  
Status:  
  
**AutomationTestCase:**

@Test

Public void verifyTitle()

{

Webdriver driver=new FirefoxDriver();

driver.get(“www.google.com);

String actual=driver.getTitle();

Assert.assertEquals(actual,”Gmail”);

}

Note: Main method is not required for TestNG programs. TestNG programs contain only methods that contain @Test annotations .If we don’t write @Test annotation then the methods are not going to be executed.

Install TestNG In Eclipse ID:

1. Go to Help Menu
2. Tap on install new software
3. Click add and enter name as ‘TestNG’
4. Give the location as <http://beust.com/eclips>’ and click okay
5. Select TestNG
6. Click next->next->accept the agreement->finish ->okay->restart eclips button
7. Execute the above code in eclipse as TestNG code

To copy the email able reports select on the Project in eclipse and rt click on the project and refresh the project . ‘Test Output’ will be generated below the reference libraries . Open the ‘Test Output’ folder and you can see so many formats of output results.

Multiple Test Cases:

It will execute the Test cases in alphabetical order Eg: A,B and then C by default. If you want to control the test execution then use Priority Attributes

Example:

@Test

Public void testA()  
 {  
 Assert.assertEquals(“Google”,”Google”);

}

@Test

Public void testC()  
 {  
 Assert.assertEquals(“facebook”,”Facebook”); //it will also compare the Capitalization ex: f and F

}

@Test

Public void testB()  
 {  
 Assert.assertEquals(“Shahu”,”tarale”);

}

The order will be testA ,testB,and then testC.

**Priority Attribute:**  
The Example of the Test executions as per the user control(Priorities) is as below

Syntax: @Test (priority=1);   
 we can use negative numbers to give the priorities and the test case with lowest priority will get execute first EX: if we are give -1,1,2.

This is known as Attribute of @Test Annotation.

**Description Attribute:(Optional)**

This will allow user to give meaningful description to the test Case

@Test(priority=3, description=”Shahu”); The description will get displayed below the Test Result i.e. below the testcase result(Passed or Failed). The description attribute is optional

In TestNG tests we use Java Programming features, webDriver methods and TestNg annotations  
There are total 10 attribute annotations available on selenium TestNg

**Test Attributes:**

**1. @Test** we can call this as a default priority.  
**2. @Test(priority=int)-:**Test method execution priority. We can execute the test case according to the given priority . **we can use negative number as well to give the priority**. If two test cases has the same priority then the those testcases will be executed according the alphabetical order which is default priority. This is shown as per the below code:  
  
@Test(priority=1)  
Public void testA()  
{  
 Assert.assertEquals(“Google”,”Google”);  
}  
@Test(priority=2)  
Public void testC()  
{  
Assert.assertEquals(“facebook”,”Facebook”); //it will also compare the Capitalization ex: f and F

}@Test(priority=3)

Public void testB()  
 {  
 Assert.assertEquals(“Shahu”,”tarale”);

}

**3. @Test(description=”---------“):** This is used to give the description of the testcase. This is a optional attribute of TestNG.  
@Test(priority=3, description="Verify login")

**public** **void** login()

{

System.***out***.println("Login Is suucssfull");;

}  
the out of this will appear as below  
PASSED: login

Verify login  
[**4.@Test(enabled=boolean)**](mailto:4.@Test(enabled=boolean))**–** Ignores the test cases which has the False Boolean value. This is different than the Skipping the testcase it will disable the testcse and the format will be as below  
@Test(priority=-10,enabled=**false**)

**public** **void** search()

{

System.***out***.println("Seaching it");  
By default all the testcases are **enabled.**Even if the testcase has the higher priority it will be engored if it is disabled  
  
  
**5. @Test(dependsOnMethods={“method name”})** – dependsOnMethods attribute is used when we want to execute a method depends on the status of the another method i.e. if the method on which the our methods depends on got passed then only the our will get executed. suppose we have 50 testcases and we have used priority attribute and given the priorities to them like 1 to 50 and now we have to add 2 more testcase in the position of 23rd and 30th position that time we have to change the priorities of whole tests.

If the dependent testcase failed then the testcase which is depends on that testcase will be “Skipped”. Which is shown in below example  
@Test(priority=-11,dependsOnMethods={"logout"})

**public** **void** search()

{

System.***out***.println("Seaching it");;

}

@Test(priority=-10)

**public** **void** logout()

{

Assert.*assertEquals*("abc","xyz");

}

Output of this will be as below  
Default suite

Total tests run: 3, Failures: 1, Skips: 1

There are two type on dependencies attribute used with “dependsOnMethods” attribute . The below example is a **hard dependency**.  
**@Test(priority=-11,dependsOnMethods={"logout"})**In hard dependency the dependent testcase gets skipped if the depending method fails.  
The **soft dependency** is shown as below   
**@Test(priority=-11,dependsOnMethods={"logout"},alwaysRun=true)**

In soft dependency the dependent testcase will always run it won’t get skipped even if the depending method is failing

[**6.@Test(alwaysRun=true)**](mailto:6.@Test(alwaysRun=true))**: -** If the tester wants to execute the testcase in any condition then we can use alwaysRun=true attribute with @Test annotation. The Example of this attribute is as below  
@Test(priority=3, description="Verify login")

**public** **void** login()

{

System.***out***.println("Login Is suucssfull");;

}

@Test(priority=-11,dependsOnMethods={"logout"},alwaysRun=**true**)

**public** **void** search()

{

System.***out***.println("Seaching it");;

}

@Test(priority=-10)

**public** **void** logout()

{

Assert.*assertEquals*("adb","xyz");

System.***out***.println("logged out");

}  
here the dependent method the search() will get executed even if logout() method is failing. The output of the above code is as below  
PASSED: login

Verify login

PASSED: search

FAILED: logout

java.lang.AssertionError: expected [xyz] but found [adb]  
  
Default test

Tests run: 3, Failures: 1, Skips: 0  
  
There are **two type on dependencies** attribute used with “dependsOnMethods” attribute. The below example is a **hard dependency**.  
**@Test(priority=-11,dependsOnMethods={"logout"})**In hard dependency the dependent testcase gets skipped if the dependending method fails.  
The **soft dependency** is shown as below   
**@Test(priority=-11,dependsOnMethods={"logout"},alwaysRun=true)**

In soft dependency the dependent testcase will always run it wont get skipped even if the depending method is failing  
If we use **alwaysRun** attribute with **dependsOn Methods**  attribute then it will become **Soft Dependency.  
  
TestNG Annotations:**

There are different type of TestNG annotations are available.

1. @BeforeMethod –Execute Before every test case in a class
2. @AfterMethod - Executes after every test case in a class
3. @BeforeClass – Executes before all testcases in a class
4. @AfterClass - Execute After all test cases in a class
5. @BeforeTest - Executes before all test case from all classes in xml file
6. @AfterTest - Executes after all test case from all classes in xml file

Consider the below scenario for the end to end flow.  
Login  
Logout  
Search  
Advanced Search  
Buy products  
  
The execution flow for the above scenario is as below  
1.Login  
2.Search  
3. Logout  
4. Login  
5. Advanced Search  
6.Logout  
7.Login  
8.Buy Products  
9.Logout  
  
the code for this will be  
 @BeforeMethod

**public** **void** login()

{

System.***out***.println("Login Successfull");

}

@Test()

**public** **void** search()

{

System.***out***.println("search Successfull");

}

@Test(dependsOnMethods={"search"})

**public** **void** AdvancedSearch()

{

System.***out***.println("Advanced Search Successfull");

}

@Test(dependsOnMethods={"AdvancedSearch"})

**public** **void** BuyProduct()

{

System.***out***.println("BuyProdct Successfull");

}

@AfterMethod

**public** **void** logout()

{

System.***out***.println("logout Successfull");}

The Output of the above code will be  
Login Successfull

search Successfull

logout Successfull

Login Successfull

Advanced Search Successfull

logout Successfull

Login Successfull

BuyProdct Successfull

logout Successfull

PASSED: search

PASSED: AdvancedSearch

PASSED: BuyProduct

As per above result it is clear that the precondition and post condition will get executed for each and every testcase in the scenario.

If we use the method which is used in @BeforeMethod annotation with dependsOnMethods attribute it will give the below error

@BeforeMethod

**public** **void** login()

{

System.***out***.println("Login Successfull");

}

@Test(dependsOnMethods={"logout"})

**public** **void** hello()

{

System.***out***.println("Hello is successful");

}

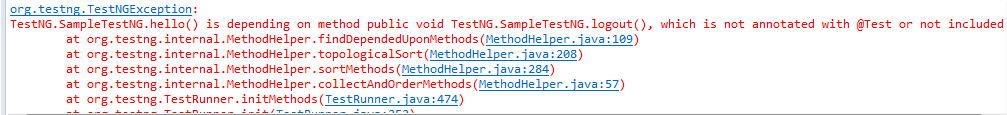
@AfterMethod

**public** **void** logout()

{

System.***out***.println("logout Successfull");

}

The above code will produce below error  


In all the above scenarios   
login – precondition for every testcases

logout – post-condition for every testcases

What if the user don’t want to use the pre and post conditions for some test case. This will be explained as below

Login  
Logout  
Search  
Advanced Search  
Buy products

Execution flow for the above scenario  
Login  
Search  
Advanced Search  
Buy products  
Logout

For this scenario we can use @BeforeClass and AfterClass annotations.

**@BeforeClass and @AfterClass:**

These annotations are used to give the common Pre conditions and Common post condition for all the test cases available in the TestNg class. The example of the @BeforeClass and @AfterClass is as below.

@BeforeClass

**public** **void** login()

{

System.***out***.println("Login Successfull");

}

@Test( priority=1)

**public** **void** search()

{

System.***out***.println("search Successfull");

}

@Test(dependsOnMethods={"search"})

**public** **void** AdvancedSearch()

{

System.***out***.println("Advanced Search Successfull");

}

@Test(dependsOnMethods={"AdvancedSearch"})

**public** **void** BuyProduct()

{

System.***out***.println("BuyProdct Successful");

}

@AfterClass

**public** **void** logout()

{

System.***out***.println("logout Successful");

}

The execution flow of the above code is as shown below

Login Successfull

search Successfull

Advanced Search Successfull

BuyProdct Successfull

logout Successful

The difference between the @BeforeMethod, @AfterMethod and @BeforeClass, AfterClass is that @BeforeMethod, @AfterMethod will be applicable only for the methods but @BeforeClass, AfterClass are applicable for entire class

Use @BeforeMethod and AfterMethod when there are no priorities used in between testcases. And use the @BeforeClass and @AfterClass when there are use of the priorities in the code. To execute the code with priorities then use above code.

**Parallel Execution:**

**Executing multiple programs using XML file**:( XML- extensible mark up language):

The difference between HTML and xml is that in HTML we have predefined tags and in XML we can create or use our own Tags.

The XML format for multiple execution is as below.

<suite name= “Ecommerce”>

<test name =”UserTest”>

<classes>

<class name =packasgename.classname\>

<class name =packasgename.classname\>

<class name =packasgename.classname\>

<\classes>

<\test>

<\suite>

Tags available in above XML code are

Suite tag ,test tag, classes tag, class tag

**Create XML file in eclipse :**

1. Select java project and rt. Click
2. Click on new and select Other option
3. Enter TestNG in the search box
4. Select TestNG class and tap Okay
5. Tap brows button in front of the Source folder > double click on the package> select “src” folder and tap on okay button.
6. Tap brows button in front of the Package Name > select the package in which the source code is available> tap on okay button.
7. Enter the class name. you can give any class name
8. Enter xml file name as “tesNG.xml”. you can give any name but it should have .xml extension
9. Add the class names into the xml file by using below syntax

<class name =packasgename.classname 1\>

<class name =packasgename.classname 2\>

<class name =packasgename.classname 3\>

1. Rt click on the xml file created and run it as TestNG Suite

The xml code for the multiple class execution is as below

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

<test name=*"Test"*>

<classes>

<class name=*"TestNG.SampleTestNG"*/>

<class name=*"TestNG.TestNGBegin"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

The output for the above code is

**@BeforeTest and @AfterTest:**

This is used to give the pre and post conditions for all classes involved in Multiple class execution xml.file. we have to use @BeforeTest and @AfterTest condition in any one of the class of xml file and the pre and post conditions will be applicable for all the testcases in all the class used in xml.

This is a multiple classes level pre and post conditions annotations

@BeforeMethod –Execute Before every test case in a class

@AfterMethod - Executes after every test case in a class

@BeforeClass – Executes before all testcases in a class

@AfterClass - Execute After all test cases in a class

@BeforeTest - Executes before all test case from all classes in xml file

@AfterTest - Executes after all test case from all classes in xml file

**Grouping Test Cases:**

Group testing is a technique used to execute multiple similar kind of test cases at the same time. For example if you want to execute group of regression testcase then we can use the Group testing technique. The xml file for the group testing is as below.

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

<test name=*"Test"*>

<groups>

<run>

<include name=*"regressions"*>

</include>

</run>

</groups>]

<classes>

<class name=*"TestNG.GroupingTest"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

The TestNG class for the Group testing the regressions is as below

@BeforeGroups(groups ="regressions")

**public** **void** login()

{

System.***out***.println("Login Successful");

}

@AfterGroups(groups ="regressions")

**public** **void** logout()

{

System.***out***.println("Logout Successful");

}

@Test

**public** **void** search()

{

System.***out***.println("Search Successful");

}

@Test(groups={"sanity"}, priority=0)

**public** **void** fundtransfer()

{

System.***out***.println("Fund Transfered");

}

@Test(groups={"sanity", "regressions"})

**public** **void** advancedsearch()

{

System.***out***.println("Advanced Search Successful");

}

@Test(groups={"regressions"})

**public** **void** regressions()

{

System.***out***.println("regressed");

}

@Test(groups={"regressions"})

**public** **void** rgressed()

{

System.***out***.println("regressed done");

}