**Introduction to TestNG Framework**  
  
I) Overview  
  
II) Install TestNG and write First TestNG Test Case.  
  
III) Create multiple Test Cases and Run  
  
IV) Execute multiple programs/classes using XML  
-----------------------------------------------  
**I) Overview**  
  
> In Selenium using Java there are two Testing frameworks available,  
  
1) JUnit  
  
2) TestNG  
------------------------------  
**TestNG Testing Framework**  
> TestNG is a testing framework designed to simplify a broad range of Testing needs, from Unit Testing to System Testing.  
  
> Initially developed for Unit Testing, now used for all kinds of Testing.  
  
> TestNG is an open source framework, where NG stands for next generation.  
  
> TestNG inspired from Junit(Java platform) and NUnit (.NET platform), but introducing some new functionalities that make it more powerful and easier to use.  
--------------------------------------------  
Advantages of TestNG  
  
1) TestNG Annotations are easy to create Test Cases  
  
2) Test Cases can be grouped and prioritized more easily.  
  
3) Supports Parameterization.  
  
4) Supports Data driven Testing using Dataproviders.  
  
5) Generates HTML reports  
  
6) Parallel test execution is possible.  
  
7) Readily supports integration with other tools and plug ins like Eclipse IDE, build tools Ant, Maven etc...  
---------------------------------  
Note: Using TestNG we can create Test Cases, group Test Cases, prioritize Test Cases, execute Test Cases and generate Test Reports.  
---------------------------------------------  
**II) Install TestNG and write first Test Case**  
   
In Eclipse  
  
Help menu -> Install New Software -> Click Add  
-> Enter Name as "TestNG"  
-> Enter URL as "http://beust.com/eclipse/"  
-> Select "TestNG"  
-> Next -> Next -> Accept the Agreement -> Finish  
------------------------------------------------------  
Write TestNG Test Case  
  
**Manual Test Case**  
  
Test Case Name: Verify title of the Page  
  
Test Steps:  
  
1) Launch Browser  
  
2) Navigate to gmail.com  
------------------------------  
Verification point  
Capture the Page title and compare with expected  
  
Expected = Gmail  
  
Actual =   
  
Status =  
--------------------------------------------  
TestNG Test Case:  
  
public class Sample {  
@Test  
public void verifyTitle(){  
WebDriver driver = new FirefoxDriver();      
driver.get("https://www.gmail.com");  
String pageTitle = driver.getTitle();  
Assert.assertEquals(pageTitle, "Gmail");  
}  
}  
-------------------------------------------  
Note:  
1) main method is not used for TestNG programs.  
  
2) TestNG programs contains only methods that contain @Test Annotations  
  
3) if we don't write @Test Annotations then the methods are not going to be executed.  
----------------------------------------------------------  
**III) Write Multiple Test Cases**  
   
public class Sample {  
@Test   
public void testA(){  
Assert.assertEquals("Gmail", "Gmail");  
}  
@Test   
public void testC(){  
Assert.assertEquals("Gmail", "Google");  
}  
@Test   
public void testB(){  
Assert.assertEquals("Yahoo", "Yahoo");  
}  
}  
  
Note: TestNG Test cases are executed in Alphabetical order,  
  
If You want to control the Test execution process then use priority attribute.  
---------------------------------------------------------------  
public class Sample {  
@Test (priority = 3)  
public void abcd(){  
Assert.assertEquals("Gmail", "Gmail");  
}  
@Test (priority = 2)  
public void xyz(){  
Assert.assertEquals("Gmail", "Google");  
}  
@Test (priority = 1)  
public void pqr(){  
Assert.assertEquals("Yahoo", "Yahoo");  
}  
}  
-------------------  
General Test Execution Flow:  
abcd  
pqr  
xyz  
-----------------  
pqr  
xyz  
abcd  
--------------------  
public class Sample {  
@Test (priority = 3)  
public void abcd(){  
Assert.assertEquals("Gmail", "Gmail");  
}  
@Test (priority = 1, enabled = false)  
public void xyz(){  
Assert.assertEquals("Google", "Google");  
}  
@Test (priority = 2)  
public void pqr(){  
Assert.assertEquals("Yahoo", "Yahoo");  
}  
}  
--------------------------------  
public class Sample {  
@Test   
public void login(){  
System.out.println("Login Successful");  
}  
@Test (dependsOnMethods = {"login"})  
public void search(){  
System.out.println("Search Successful");  
}  
@Test (dependsOnMethods = {"search"})  
public void advancedSearch(){  
System.out.println("Advanced Search Successful");  
}  
@Test (dependsOnMethods = {"advancedSearch"})  
public void logout(){  
System.out.println("Logout Successful");  
}  
}  
-------------------------------------  
Hard dependency

//Hard dependencies. All the methods you depend on must

// have run and succeeded for you to run.

// If at least one failure occurred in your dependencies,

// you will not be invoked and marked as a SKIP in the report.

// Soft dependencies--You will always be run after

// the methods you depend on, even if some of them have failed.

// This is useful when you just want to make sure that your test

// methods are run in a certain order but their success

// doesn't really depend on the success of others.

// A soft dependency is obtained by adding "alwaysRun=true"

// in your @Test annotation.

@Test (dependsOnMethods ={"methodName"})  
  
Soft Dependency  
@Test (dependsOnMethods ={"methodName"}, alwaysRun=true)  
--------------------------------------------------  
public class Sample {  
public WebDriver driver;  
@Test (priority=1)  
public void launchBrowser(){  
driver = new FirefoxDriver();      
}  
  
@Test (priority=2)  
public void verifyPageTitle1(){  
driver.get("https://www.gmail.com");  
Assert.assertEquals("Gmail", driver.getTitle());  
}  
  
@Test(priority=3)  
public void verifyPageTitle2(){  
driver.get("https://in.yahoo.com/");  
Assert.assertEquals("Yahoo", driver.getTitle());  
}  
@Test (priority=4)  
public void closeBrowser(){  
driver.close();  
}  
}  
--------------------------------------  
Test Execution Flow  
1) closebrowser  
2) launchBrowser  
3) verifyPageTitle1  
4) verifyPageTitle2  
  
Test Execution Flow (As per priorities) :  
  
1) launchBrowser  
2) verifyPageTitle1  
3) verifyPageTitle2  
4) closeBrowser  
--------------------------------------------------  
BeforeMethod and AfterMethod Annotations  
  
@BeforeMethod - Pre-condition for every Test case in a Class/Program  
@AfterMethod Post-condition for every Test case in a Class/Program  
  
**Example:**  
  
@BeforeMethod  
public void launchBrowser(){  
driver = new FirefoxDriver();      
}  
  
@Test   
public void verifyPageTitle1(){  
driver.get("https://www.gmail.com");  
Assert.assertEquals("Gmail", driver.getTitle());  
}  
  
@Test  
public void verifyPageTitle2(){  
driver.get("https://in.yahoo.com/");  
Assert.assertEquals("Yahoo", driver.getTitle());  
}  
@AfterMethod  
public void closeBrowser(){  
driver.close();  
}  
}  
--------------------------------------  
Test Execution Flow:  
launchBrowser -pre-condition for every test case.  
closeBrowser -post-condition for every test case  
  
verifyPageTitle1  
verifyPageTitle2  
---------------------------  
launchBrowser  
verifyPageTitle1  
closeBrowser  
  
launchBrowser  
verifyPageTitle2  
closeBrowser  
---------------------------------------------  
BeforeClass and AfterClass Annotations  
  
@BeforeClass -Pre-condition for All Test cases in a Class/Program  
@AfterClasee -Post-condition for All Test cases in a Class/Program  
  
**Example:**  
   
@BeforeClass  
public void launchBrowser(){  
driver = new FirefoxDriver();      
}  
  
@Test   
public void verifyPageTitle1(){  
driver.get("https://www.gmail.com");  
Assert.assertEquals("Gmail", driver.getTitle());  
}  
  
@Test  
public void verifyPageTitle2(){  
driver.get("https://in.yahoo.com/");  
Assert.assertEquals("Yahoo", driver.getTitle());  
}  
@AfterClass  
public void closeBrowser(){  
driver.close();  
}  
}