**public** **class** DataPro {

@Test(dataProvider="SamsURL")

**public** **void** launchURL(String url){

System.setProperty("webdriver.chrome.driver", "/Users/vn0m93a/Documents/Codebase/Softwares/chromedriver");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.get(url);

System.out.println(driver.getTitle());

driver.close();

}

@DataProvider(name="SamsURL")

**public** Object[][] urlSupply(){

Object[][]data= **new** Object[1][1];

data[0][0]="https://samsclub.com";

**return** data;

}

}

Note: If for @DataProvider , name attribute is not used than,

Name of the method followed by @DataProvider should be used.

**public** **class** DataPro {

@Test(dataProvider = "urlSupply")

**public** **void** launchURL(String url) {

System.setProperty("webdriver.chrome.driver", "/Users/vn0m93a/Documents/Codebase/Softwares/chromedriver");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.get(url);

System.out.println(driver.getTitle());

driver.close();

}

@DataProvider

**public** **static** Object[][] urlSupply(){

Object[][]data= **new** Object[1][1];

data[0][0]="https://samsclub.com";

**return** data;

}

}

7. dataProviderClass

**public** **class** DataPro {

@Test(dataProvider="dp",dataProviderClass=DataProvClass.**class**)

**public** **void** launchURL(String url){

System.setProperty("webdriver.chrome.driver", "/Users/vn0m93a/Documents/Codebase/Softwares/chromedriver");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.get(url);

System.out.println(driver.getTitle());

driver.close();

}

}

**public** **class** DataProvClass {

@DataProvider(name="dp")

**public** **static** Object[][] urlSupply(){

Object[][]data= **new** Object[1][1];

data[0][0]="https://samsclub.com";

**return** data;

}

}

Note 1: In this scenario the DataProvider method was in a different class. In such a case the dataProviderMethod() has to be declared static so that it can be used by a test method in a different class for providing data.

Note 2 : If for @DataProvider , name attribute is not used than,

Name of the method followed by @DataProvider should be used.

Ex:

**public** **class** DataPro {

@Test(dataProvider = "urlSupply", dataProviderClass = DataProvClass.**class**)

**public** **void** launchURL(String url) {

System.setProperty("webdriver.chrome.driver", "/Users/vn0m93a/Documents/Codebase/Softwares/chromedriver");

WebDriver driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.get(url);

System.out.println(driver.getTitle());

driver.close();

}

}

**public** **class** DataProvClass {

@DataProvider

**public** **static** Object[][] urlSupply(){

Object[][]data= **new** Object[1][1];

data[0][0]="https://samsclub.com";

**return** data;

}

}

***\*\*\*ASSERTion: using ASSERT***

TestNG supports assertion of a test using the Assert class.

Ex: Assert.assertEquals("actual","expected");

Asserts helps us to verify the conditions of the test and decide whether test has failed or passed.

A test is considered successful ONLY if it is completed without throwing any exception.

Ex:

verifying if the page title is equal to 'Google' or not. If the page title is not matching with the text /title that we provided, it will fail the test case.

@Test

public void testCaseVerifyHomePage() {

driver= new FirefoxDriver();

driver.navigate().to("http://google.com");

Assert.assertEquals("Google", driver.getTitle());

}

@Test

public void testCaseVerifyHomePage() {

driver= new FirefoxDriver();

driver.navigate().to("http://google.com");

Assert.assertEquals("Gooooogle", driver.getTitle());

}

The above code will throw you an Assertion error as below:

java.lang.AssertionError: expected [Google] but found [Gooooogle]

**Assert class methods:**

assertEqual(String actual,String expected) :-

It takes two string arguments and checks whether both are equal, if not it will fail the test.

assertEqual(String actual,String expected, String message) :- It takes three string arguments and checks whether both are equal, if not it will fail the test and throws the message which we provide.

assertEquals(boolean actual,boolean expected) :- It takes two boolean arguments and checks whether both are equal, if not it will fail the test.

assertEquals(java.util.Collection actual, java.util.Collection expected, java.lang.String message) :-

Takes two collection objects and verifies both collections contain the same elements and with the same order. if not it will fail the test with the given message.

Assert.assertTrue(condition) :- It takes one boolean arguments and checks that a condition is true,If it isn't, an AssertionError is thrown.

Assert.assertTrue(condition, message) :- It takes one boolean argument and String message. It Asserts that a condition is true. If it isn't, an AssertionError, with the given message, is thrown.

Assert.assertFalse(condition) :- It takes one boolean arguments and checks that a condition is false,If it isn't, an AssertionError is thrown.

Assert.assertFalse(condition, message) :- It takes one boolean argument and String message. It Asserts that a condition is false. If it isn't, an AssertionError, with the given message, is thrown.

The below is the sample code to use assertions :

@Test

public void testCaseVerifyHomePage() {

driver= new FirefoxDriver();

driver.navigate().to("http://google.com");

Assert.assertEquals("Gooogle", driver.getTitle(), "Strings are not matching");

//Write a code to login and write a method called isUserLoggedInSuccessfully and isUserLoggedOut which returns boolean.

Assert.assertTrue(isUserLoggedInSuccessfully(), "User failed to login");

Assert.assertFalse(isUserLoggedOut())

}

***Normal<Hard> and Soft Assert***

**Hard Assert:**

@Test

***public******void*** *verifyLindeinIn() {*

*driver =* ***new*** *ChromeDriver();*

*driver.manage().window().maximize();*

*driver.get("https://linkedin.com");*

*Assert.assertTrue(driver.getTitle().contains("Link"));*

*System.out.println("Test 1 is successful");*

*driver.close();*

*}*

**SoftAssert:**

@Test

*public void validate01() {*

*SoftAssert softAssert = new SoftAssert();*

*softAssert.assertEquals(true, true);*

*softAssert.assertEquals(5, 5);*

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

*softAssert.****assertAll();***

*}*

***@Parameters***

it is used to pass environmental variables or generic variables which are same for *different classes and Test Cases.*

***TestNG Parameters***

It allows us to automatically run a test case multiple times with different input and validation values.

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

Ex:

**public** **class** BrowserOps {

WebDriver driver;

@Test

@Parameters("url")

**public** **void** browserOps(String url) {

System.setProperty("webdriver.chrome.driver", "/Users/vn0m93a/Documents/Codebase/Softwares/chromedriver");

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

driver.get(url);

System.out.println(driver.getTitle());

// Assert.assertEquals(false, true);

driver.close();

}

}

testing.xml :

<suite name="Suite">

<test thread-count="5" name="Test">

<parameter name="url" value="https://samsclub.com"></parameter>

<classes>

<class name="com.webScenarios.BrowserOps"/>

</classes>

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

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

pass the values to Test Cases from xml file <name,Value pair>

by using

<parameter name="parameterName" value="parameterValue"></parameter>

before the <classes> nodes/tags

// for one parameter - one line as above in testing.xml, for multiple parameters use multiple lines.

then use @Parameters after @Test annotation in java class.

@Parameters({"parameterName”}) // in java class

***Note1:*** Run the tests from testing.xml file, if run from Java class it will not be executed and throws below error.



***Note2: we can supply same parameter to multiple Test cases under different Java classes through testing.xml as below.***

<suite name="Suite">

<test thread-count="5" name="Test">

<parameter name="url" value="https://samsclub.com"></parameter>

<classes>

<class name="com.webScenarios.BrowserOps"/>

<class name="com.webScenarios.Class2"/>

</classes>

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

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

***Note 3: Paramters are supplied to only those test methods in the classes having @Parameters annotation after the @Test.***

**@DataProvider:**

only one attribute: "name"

@DaraProvider(name = "DataProvider\_Name")

if name is not provided / empty --> its going to take method name following it.

it returns: object type : Object [][]

2-D array

used to store Test Data

initialization: Object [][] = new Object[rows][columns];

**Test Data stored will be used during runtime.**

To access the Data from DataProvider: @Test(dataProvider="DataProvider\_Name")

parameterize the method under @Test /Test Method with data types of input variables.

***Ex: LOGIN TO A WEB APP WITH BASIC LOGIN CREDENTIALS:***

public class Login{

public WebDriver driver;

driver = new FirefoxDriver();

driver.manage().window().maximize();

driver.get("URL");

@Test(dataProvider = "loginCredentials")

public void open(String User,String Pwd){

driver.findElement(By.id("username")).clear();

driver.findElement(By.id("username")).sendKeys(User);

driver.findElement(By.id("password")).clear();

driver.findElement(By.id("password")).sendKeys(Pwd);

driver.findElement(By.id("loginButton")).click();

driver.close();

}

@DaraProvider(name = "loginCredentials")

public Object[][] getData(){

Object[][] data = new Object[3][2];

data [0][0] = "user1";

data [0][1] = "pwd1";

data [1][0] = "user2";

data [1][1] ="pwd2";

data [2][0] ="user3";

data [2][1] ="pwd3";

return data;

}

}

**Read data from Excel to DataProvider in Selenium using TEST NG:**

NOTE: very important:

**No. of rows = no. of Test Runs**

**No. of Columns = No. of parameters of a test method(method under @Test annotation)**

Create a Java Class under Utility as ExcelUtility:

Assumption: Data is present in a single column(only one column)

< logic will be changed based on the number of columns >

public class ExcelUtility {

private static XSSFSheet ExcelWSheet;

private static XSSFWorkbook ExcelWBook;

private static XSSFCell Cell;

public static Object[][] getTestData(String FilePath, String SheetName,int startingRow,int lastRow,int column) throws Exception {

String [][] table = null;

FileInputStream excelFile = new FileInputStream(FilePath);

ExcelWBook = new XSSFWorkbook(excelFile);

ExcelWSheet = ExcelWBook.getSheet(SheetName);

int i = startingRow;

int k= lastRow;

int j=column;

int ci=0;int cj=0;

int totalRows =k-i;

int totalCols=1;

table=new String[totalRows][totalCols];

while(i<=k){

table[ci][cj]=getCellData(i,j);

ci++ ;i++;

}

return (table);

}

public static String getCellData(int RowNum, int ColNum) throws Exception {

Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);

String CellData = Cell.getStringCellValue();

return CellData;

}

}

***@Factory***

***Ex:***

**public** **class** TestClass01 {

**private** String param = "";

**public** TestClass01(String param) {

**this**.param = param;

}

@BeforeClass

**public** **void** beforeClass() {

System.out.println("Before SimpleTest class executed.");

}

@Test

**public** **void** testMethod() {

System.out.println("testMethod parameter value is: " + param);

}

@Factory

**public** Object[] factoryMethod() {

**return** **new** Object[] { **new** TestClass01("one"), **new** TestClass01("two") ,**new** TestClass01("three")};

}

}

o/p:

Before SimpleTest class executed.

testMethod parameter value is: three

Before SimpleTest class executed.

testMethod parameter value is: two

Before SimpleTest class executed.

testMethod parameter value is: one

PASSED: testMethod

PASSED: testMethod

PASSED: testMethod

**Same scenario in DataProvider as below**

@BeforeClass

**public** **void** beforeClass() {

System.out.println("Before class executed");

}

@Test(dataProvider = "dataMethod")

**public** **void** testMethod(String param) {

System.out.println("The parameter value is: " + param);

}

@DataProvider

**public** Object[][] dataMethod() {

**return** **new** Object[][] { { "one" }, { "two" } };

}

o/p:

**Before class executed**

**The parameter value is: one**

**The parameter value is: two**

**PASSED: testMethod("one")**

**PASSED: testMethod("two")**

***Difference between @Factory and @DataProvider:***

**DataProvider**: A test method that uses DataProvider will be executed a multiple number of times based on the data provided by the DataProvider. **The test method will be executed using the same instance of the test class to which the test method belongs.**

**Factory**: A factory will execute all the test methods present inside a test class using a separate instance of the respective class.

***TestNG factory is used to create instances of test classes dynamically.***

This is useful if you want to run the test class any number of times.

For example, if you have a test to login into a site and you want to run this test multiple times,

then its easy to use TestNG factory where you create multiple instances of test class and run the tests (may be to **test any *memory leak issues*).**

Whereas, ***dataprovider is used to provide parameters to a test.***

*If you provide dataprovider to a test, the test will be run taking different sets of value each time.*

This is useful for a scenario like where you want to login into a site with different sets of username and password each time.

***@Listeners***

**// to Implement TestNG listeners in:**

Listeners - features of TestNG to **customize logs / reports of Test NG**

listens to certain events of TestNG and respond accordingly.

Complete customization of reports.

**Types of Listeners:(Interfaces)**

IAnnotation Transformer

IAnnotation Transformer 2

IHookable

IInvokedMethodListener

IMethodInterceptor

IReporter

ISuiteListener

ITestListener - used in our projects

**ITestListener** - implementation in two ways

1.extend **ITestListenerAdapter** Class

2. implement **ITestListener** Interface.

Implement at 1. Class Level 2. Suite Level.

public class TestNGListener implements ITestListener { // import ITestListener from testng.org

// click on Add Unimplemented Methods.. will get

@Override

public void onFinish(ITestContext arg0) {

// TODO Auto-generated method stub

}

@Override

public void onStart(ITestContext arg0) {

// TODO Auto-generated method stub

}

@Override

public void onTestFailedButWithinSuccessPercentage(ITestResult arg0) {

// TODO Auto-generated method stub

}

@Override

public void onTestFailure(ITestResult arg0) {

// TODO Auto-generated method stub

}

@Override

public void onTestSkipped(ITestResult arg0) {

// TODO Auto-generated method stub

}

@Override

public void onTestStart(ITestResult arg0) {

// TODO Auto-generated method stub

}

@Override

public void onTestSuccess(ITestResult arg0) {

}

}

// Next.... remove the comments and perform as below...

public class TestNGListener implements ITestListener {

@Override

public void onFinish(ITestContext result) {

}

@Override

public void onStart(ITestContext result) {

}

@Override

public void onTestFailedButWithinSuccessPercentage(ITestResult result) {

}

@Override

public void onTestFailure(ITestResult result) {

System.out.println("The details of Test Failed are"+result.getName()); // getName() - gets name of the Test Case

}

@Override

public void onTestSkipped(ITestResult result) {

System.out.println("The details of Test Skipped are"+result.getName());

}

@Override

public void onTestStart(ITestResult result) {

System.out.println("The details of TestStart are"+result.getName());

}

@Override

public void onTestSuccess(ITestResult result) {

System.out.println("The details of TestSuccess are"+result.getName());

}

}

**// To Implement Listeners in Java Project / Java Class**

@Listeners(package\_name.TestNGListener.class)

// import Listeners from org.testng.annotations

// keep Listeners at Class Level

public class TestNGTestCase{

@Test

public void testMethod(){

// write any method statements which perform some task

// ex: launching a browser

}

}

**// To implement for multiple classes / 2. Implement at Suite Level**

remove

@Listeners(package\_name.TestNGListener.class) before each class

convert the Class in to TestNG (Convert to TestNG)

in the testng.xml , include the below code after the Suite tag.

<listeners>

<listener class-name="package\_name.TestNGListener"/>

</listeners>

// if multiple classes :

add the same node as above.

@Override

@Override annotation is used when we override a method in sub class(Child Class).

1) If programmer makes any mistake such as wrong method name, wrong parameter types while overriding, you would get a compile time error.

As by using this annotation you instruct compiler that you are overriding this method.

If you don’t use the annotation then the sub class method would behave as a new method (not the overriding method) in sub class.

2) It improves the readability of the code. So if you change the signature of overridden method then all the sub classes that

overrides the particular method would throw a compilation error, which would eventually help you to change the signature

in the sub classes. If you have lots of classes in your application then this annotation would really help you to identify the classes that require changes when you change the signature of a method.

***##########################################################***

***\* TO GENERATE testng.xml REPORT:***

RIGHT CLICK ON ANY "CLASS FILE"

--> TESTNG

--> CONVERT TO TESTNG

-->"testng.xml" FILE GETS CREATED.

Note: testing.xml file can have any name but the file extension should be .xml

***\* WORKING ON TESTNG.XML***

right click on testng.xml --> open--> body of xml:

<?xml version="1.0" encoding="UTF-8"?>

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

<suite name="Suite">

<test name="Test">

<classes>

<class name="package\_name.Class\_name"/>

</classes>

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

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

***NOTE:***

first two lines are autogenerated - need not bother about it

The values of suite name and test name can be customised as per our wish

\* under <classes> we can add N no of classes from same or different packages and all can be run

together one by one from top to bottom. (*BUT THIS IS NOT PARALLEL TESTING)*

\* value of class\_name syntax: package\_name.Class\_name

**Ex: <?xml version="1.0" encoding="UTF-8"?>**

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

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

**<test thread-count="5" name="Test">**

**<classes>**

**<class name="Test01"></class>**

**</classes>**

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

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

***Note: for default package – no need of package name.***

***SPECIAL ANNOTATION:***

***@BeforeGroups***

***@AfterGroups***

\* both will be used when required if we use parameter "group" in @Test

Ex: @Test(groups ={"Functional","Sanity"})

***TO PERFORM GROUP TESTING:***

Add this piece of code in the above code after <test name = ""> and before <classes>

<groups>

<run>

<include name="GROUP\_NAME"></include>

</run>

</groups>

and the actual code becomes

<?xml version="1.0" encoding="UTF-8"?>

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

<suite name="Suite">

<test name="Test">

<groups>

<run>

<include name="GROUP\_NAME"></include>

</run>

</groups>

<classes>

<class name="package\_name.Class\_name"/>

</classes>

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

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

\****NOTE:*** we can execute N no. of groups together by including

<include name="GROUP\_NAME"></include> under one another

and can also from different classes just by adding

<class name="package\_name.Class\_name"/>

one below another / one next to another.

***include and exclude***

*suite name="Suite" parallel="classes" thread-count="3">*

*<groups>*

*<run>*

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

*<exclude name="groupNameExcluded"></exclude>*

*</run>*

*</groups>*

*<test thread-count="5" name="Test">*

*<classes>*

*<class name="com.testngOps.ParallelTest"/>*

*<class name="com.testngOps.Parallel2"></class>*

*<class name="com.testngOps.Parallel3"></class>*

*<class name="com.testngOps.Parallel4"></class>*

*<class name="com.testngOps.Parallel5"></class>*

*</classes>*

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

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

***Actual working with testing.xml***

Xml file with Class names

|  |
| --- |
| <suite name="Suite1" verbose="1" > |

|  |
| --- |
| <test name="Nopackage" > |

|  |
| --- |
| <classes> |

|  |
| --- |
| <class name="NoPackageTest" /> |

|  |
| --- |
| </classes> |

|  |
| --- |
| </test> |

|  |
| --- |
|  |

|  |
| --- |
| <test name="Regression1"> |

|  |
| --- |
| <classes> |

|  |
| --- |
| <class name="test.sample.ParameterSample"/> |

|  |
| --- |
| <class name="test.sample.ParameterTest"/> |

|  |
| --- |
| </classes> |

|  |
| --- |
| </test> |

|  |
| --- |
| </suite> |

Xml file with package name

|  |
| --- |
| <suite name="Suite1" verbose="1" > |

|  |
| --- |
| <test name="Regression1"   > |

|  |
| --- |
| <packages> |

|  |
| --- |
| <package name="test.sample" /> |

|  |
| --- |
| </packages> |

|  |
| --- |
| </test> |

|  |
| --- |
| </suite> |

In the above code:

TestNG will look at all the classes in the package test.sample and will retain only classes that have TestNG annotations.

We can also define new groups inside testng.xml and specify additional details in attributes, such as whether to run the tests in parallel, how many threads to use, whether you are running JUnit tests, etc...

By default, TestNG will run your tests in the order they are found in the XML file. If you want the classes and methods listed in this file to be run in an unpredictable order, set the preserve-order attribute to false

[view source](http://testng.org/doc/documentation-main.html#viewSource)

[print](http://testng.org/doc/documentation-main.html#printSource)[?](http://testng.org/doc/documentation-main.html#about)

|  |  |
| --- | --- |
| <test name="Regression1" preserve-order="false"> | |
| <classes> |

|  |
| --- |
|  |
| <class name="test.Test1"> | |

|  |
| --- |
| <methods> |
| <include name="m1" /> | |

|  |  |
| --- | --- |
| <include name="m2" /> | |
| </methods> |

|  |  |
| --- | --- |
| </class> | |
|  |

|  |  |
| --- | --- |
| <class name="test.Test2" /> | |
|  |

|  |  |
| --- | --- |
| </classes> | |
| </test> |

***PARALLEL TESTING***: "parallel" "thread-count"

EXECUTING MORE THAN ONE CLASS PARALLELY / NOT IN A SEQUENTIAL MANNER./ Executing Simultaneously

it uses MultiThreading concept of Java.( process is one but instances are many)

TO DO IT:

testng.xml --> <suite name="Suite\_name" parallel = "classes" thread-count = "2">

***NOTE:*** IN TESTNG: the value of parallel can be 5 : classes/falses/instances/methods/tests

in real time we use classes/ but can use tests and methods also.

default value of thread-count:5 / but we use up to 3 because of memory consumption by thread

Ex:

<?xml version="1.0" encoding="UTF-8"?>

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

<suite name="Parallel Test Suite" parallel="classes" thread-count="2">

<test name="Sanity Test">

<classes>

<class name="com.TestNG.Class1"/>

<class name="com.TestNG.Class2"/>

</classes>

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

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

**Maximum Thread count**

**// to execute failed test cases in Selenium webdriver:**

**Why Test Case Fails:**

1. Application is down.
2. Server is not responding
3. Network Issue
4. Scripting Issue
5. ***Application Issue - validation fail - genuine bug***

public class ExecuteTestCase{

@Test

public void executeTC(){

statements;

}

// convert to TestNG

testng.xml

refresh the project --> "test-output" folder is generated and inside it we get "TestScenario" Folder

which will have testng-failed.xml (will have xml code of only failed test cases)

now Run this xml

will get a folder "Failed Suite" under test-output

- failed test cases report

Fix the Script and Re Run the XML

***//1. To fix this using a Java program<Run only failed Test Cases>***

public class TestRunner{{

public static void main(String []args){

TestNG runner = new TestNG(); // import TestNG Class from org.testng

List<String> list = new ArrayList<String>(); // collections concept, import ArrayListfrom java.util

list.add("path\_xml");// add the path of xml of failed test case to execute.

runner.setTestSuites(list);

runner.run();

}

}

**//2. To run failed TCs using Retry and IRetryListener**

***public******class*** *Retry* ***implements*** *IRetryAnalyzer {*

***public******static******final*** *Logger log = Logger.getLogger(Retry.****class****.getName());*

***private******int*** *retryCount = 0;*

***private******int*** *maxRetryCount = 1;*

*@Override*

***public******boolean*** *retry(ITestResult result) {*

***if*** *(retryCount < maxRetryCount) {*

*log("Retrying test " + result.getName() + " with status " + getResultStatusName(result.getStatus())*

*+ " for the " + (retryCount + 1) + " time(s).");*

*retryCount++;*

***return******true****;*

*}*

***return******false****;*

*}*

***public*** *String getResultStatusName(****int*** *status) {*

*String resultName =* ***null****;*

***if*** *(status == 1)*

*resultName = "SUCCESS";*

***if*** *(status == 2)*

*resultName = "FAILURE";*

***if*** *(status == 3)*

*resultName = "SKIP";*

***return*** *resultName;*

*}*

***public******void*** *log(String data) {*

*log.info(data);*

*Reporter.log(data);*

*}*

*}*

*public class RetryListener implements IAnnotationTransformer {*

*@Override*

*public void transform(ITestAnnotation arg0, Class arg1, Constructor arg2, Method arg3) {*

*IRetryAnalyzer retry = arg0.getRetryAnalyzer();*

*if (retry == null) {*

*arg0.setRetryAnalyzer(Retry.class);*

*}*

*}*

*}*

**Note:**

**testng.xml file - "Runner File" / "Template File"**

**DataProvider using Map:**

https://www.youtube.com/watch?v=hkrqODhKV48

***Execute testng.xml file in different ways.***

***1. Eclipse***

***2. InteliJ***

***3. Ant / Maven***

***4. Command Line***

***5. Batch File.***

In ***IDE*** <**Eclipse and InteliJ**>you can just right click on testng.xml file and click on Run as 'TestNG' will invoke your tests.

And we can also invoke TestNG.xml with doing some simple configuration to **maven pom.xml**

***Execute TestNG Tests in Jenkins: to invoke batch file with Jenkins.***

1.create 'New Job' Item, Click on it.

2.Enter Job name and select "Free-Style Software Project" and then click on 'OK' button.

3.should see header as 'Advanced Project Options' and with a button 'Advanced'. Now Click on "Advanced" option which will display multiple options.

4.Select 'Use custom workspace' and specify your project location (workspace directory) in Directory

5. Scroll down to see 'Build' option with 'Add Build Step'.

In this drop down select value 'Execute Windows batch command' which Runs a Windows batch script for building the project. The text that we enter in the text box will be executed as a batch file.

The script will run with the workspace as the current directory which we have specified in step 4.

6. Click on Save button. New Job will be created with the configure options that we have defined.

That's It. We have Done.

***<TO CONVERT TESTNG.XML FILE TO A BATCH FILE .bat> and Execute Testng.xml using batch file.***

***Note: batch files are specific to Windows.***

After creating testng.xml file, create a batch file by adding the below commands in it.

Note: Before doing this , Add a lib folder in workspace of respective folder if it does not exist and include all the APIs used for the respective project

in the same folder.

Step 1: Open notepad

Step 2: Paste the below lines of code - You may need to add your project location.

set projectLocation=F:\Selenium\TestNGBatchExample

cd %projectLocation%

set classpath=%projectLocation%\bin;%projectLocation%\lib\\*

java org.testng.TestNG %projectLocation%\testng.xml

pause

In the example, project location is set as 'F:\Selenium\TestNGBatchExample'.

Step 3: Save the file as 'testNGBatchFile.bat' in the same Project location

Note: use " "while saving .bat file

Note: added 'pause' statement to prevent auto-closing of console after the execution,which will print a nice message as 'Press any key to continue . . . ' so that we can view the output.

Or, if we don't want "Press any key to continue . . ." message you can just ignore that.don't add any spaces around the equal sign, if so the SET commands will not work.