**TestNG & JUNIT**

1. These are JAVA frameoworks, used for unit testing
2. JUNIT is used by developers for unit testing
3. TestNG is used by testers
4. Earlier we create testclasses and create method main inside it in order to execute, so if we have 100 testclasses, then we need to click 100 times run button, to execute testmethods prevailing inside classes. So for avoiding this, we use testing frameworks. these frameworks provide some annotations, through which we can run multiple testclasses in one go.
5. if there is dependency in testcases, lets say test A depends on test B, then on failure of test A skips test B for execution.
6. in test suite, if we put individual testclasses in different test, then execution will be happened in sequence.
7. TestNG Parameterization

Lets we have test method, that accepts username and password and we have a list of 100 set of username and password, besides that we want to execute such test method hundred times.

To implement this will create one static method, that is responsible for sending data from list one by one. such method is decorated with @dataprovider annotation

The test method which accepts such data, will have to information about data provider

One test method can accept data from one data provider at a time.

**apache POI** is responsible for communicating with all microsoft related documents.

Link = https://poi.apache.org/spreadsheet/index.html

POI -XSSF = for latest version excel i.e. xlsx

POI-HSSF = for older version excel i.e xls

Walkthrough quick guide, that you want to do with excel i.e. https://poi.apache.org/spreadsheet/quick-guide.html

There are 5 jars files that are responsible for excel reading. You can download such jars from apache site or from maven respository (add dependecy in pom.xml file)

Use "ExcelReader". java file, and refer example "Seventh Example"

**First Example**

**package** DemoTestCases;

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

**public** **class** DemoTestCase1 {

/\*

Annotations

On Adding annotation, clicking run menu, u will find run as => testng.

1. @Test = This will run the method, earlier we need main method to create an object and call the method

2. test-output = On refresh project, u will get such folder and underneath such folder, two special reports are created

1. index.html

2. emailable-report.html

3. When u will bring up such reports, you will find some default things

1. testng-customsuite.xml = This contains name of test suite i.e. Default Suite

2. Suite is collection of test cases

3. Default test case name is "Default Test"

4. test cases is collection of test classes.

\*/

@Test

**public** **void** testLogin()

{

System.***out***.println("DemoTestCase1=>Testing login functionality");

}

@Test

**public** **void** testRegistration()

{

System.***out***.println("DemoTestCase1=>Testing registartion functionality");

}

}

**Second Example**

**package** DemoTestCases;

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

**public** **class** DemoTestCase2 {

/\*

Now scenario is registration test case must be executed first, after then only login test case should be executed

- To implement this, we use priority parameter

\*/

@Test(priority=2)

**public** **void** testLogin()

{

System.***out***.println("DemoTestCase2=>Testing login functionality");

}

@Test(priority=1)

**public** **void** testRegistration()

{

System.***out***.println("DemoTestCase2=>Testing registartion functionality");

}

}

**Third Example**

**package** DemoTestCases;

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.BeforeMethod;

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

**public** **class** DemoTestCase3 {

/\*

Now scenario, there is test method, that must be executed before and after every actual test methods

like before case = lanuch the browser. for this we use "@BeforeMethod"

in between = actual test method will be executed

like after case = quit the browser. for this we use "@AfterMethod"

\*/

@BeforeMethod

**public** **void** launchBrowser()

{

System.***out***.println("DemoTestCase3=>Launching browser");

}

@AfterMethod

**public** **void** closeBrowser()

{

System.***out***.println("DemoTestCase3=>Closing broswer");

}

@Test(priority=2)

**public** **void** testLogin()

{

System.***out***.println("DemoTestCase3=>Testing login functionality");

}

@Test(priority=1)

**public** **void** testRegistration()

{

System.***out***.println("DemoTestCase3=>Testing registartion functionality");

}

}

**Fourth Example**

**package** DemoTestCases;

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.AfterTest;

**import** org.testng.annotations.BeforeMethod;

**import** org.testng.annotations.BeforeTest;

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

**public** **class** DemoTestCase4 {

/\*

Now scenario, we need those method which must invoke once in life cycle i.e in the beg and at the end

\*/

@BeforeTest

**public** **void** init()

{

System.***out***.println("DemoTestCase4=>Intilization properties, dbmanager, logger, mailing, application logs etc");

}

@AfterTest

**public** **void** dispose()

{

System.***out***.println("DemoTestCase4=>Deintilization process, quit from browser windows, which are opened in current session");

}

@BeforeMethod

**public** **void** launchBrowser()

{

System.***out***.println("DemoTestCase4=>Launching browser");

}

@AfterMethod

**public** **void** closeBrowser()

{

System.***out***.println("DemoTestCase4=>Closing broswer");

}

@Test(priority=2)

**public** **void** testLogin()

{

System.***out***.println("DemoTestCase4=>Testing login functionality");

}

@Test(priority=1)

**public** **void** testRegistration()

{

System.***out***.println("DemoTestCase4=>Testing registartion functionality");

}

}

**Fifth Example**

**package** DemoTestCases;

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.AfterTest;

**import** org.testng.annotations.BeforeMethod;

**import** org.testng.annotations.BeforeTest;

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

**public** **class** DemoTestCase5 {

/\*

Now scenario is that, we want to execute particular test method only if dependents test methods are executed

- for this we use "dependsOnMethods" parameter

- if more than two methods names will be specified to "dependsOnMethods" parameter, then order of execution is alphabetical order.

if priority is not set to test methods, then they will be executed in the order they are written, besides that they will be called first before any test method.

- like isSignUpButtonVisible, isWebGuiVisible

\*/

@BeforeTest

**public** **void** init()

{

System.***out***.println("DemoTestCase5=>Intilization properties, dbmanager, logger, mailing, application logs etc");

}

@AfterTest

**public** **void** dispose()

{

System.***out***.println("DemoTestCase5=>Deintilization process, quit from browser windows, which are opened in current session");

}

@BeforeMethod

**public** **void** launchBrowser()

{

System.***out***.println("DemoTestCase5=>Launching browser");

}

@AfterMethod

**public** **void** closeBrowser()

{

System.***out***.println("DemoTestCase5=>Closing broswer");

}

@Test(priority=2)

**public** **void** testLogin()

{

System.***out***.println("DemoTestCase5=>Testing login functionality");

}

@Test

**public** **void** isSignUpButtonVisible()

{

System.***out***.println("DemoTestCase5=>Testing signup button");

}

@Test

**public** **void** isWebGuiVisible()

{

System.***out***.println("DemoTestCase5=>Testing website GUI");

}

@Test(dependsOnMethods={"isSignUpButtonVisible"})

**public** **void** isRegistrationGUIVisible()

{

System.***out***.println("DemoTestCase5=>Testing Registration GUI");

}

@Test(priority=1, dependsOnMethods={"isRegistrationGUIVisible"} )

**public** **void** testRegistration()

{

System.***out***.println("DemoTestCase5=>Testing registartion functionality");

}

}

**Sixth Example**

**package** DemoTestCases;

**import** org.testng.Assert;

**import** org.testng.SkipException;

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

**public** **class** DemoTestCase6 {

/\*

- if testng.xml looks like, then it means the testsuite is collection of one testing module.

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

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

<suite name="DemoTestSuite">

<test name="TestFirstModule">

<classes>

<class name="DemoTestCases.DemoTestCase1"/>

<class name="DemoTestCases.DemoTestCase2"/>

<class name="DemoTestCases.DemoTestCase3"/>

<class name="DemoTestCases.DemoTestCase4"/>

<class name="DemoTestCases.DemoTestCase5"/>

<class name="DemoTestCases.DemoTestCase6"/>

</classes>

</test> <!-- TestFirstModule -->

</suite> <!-- DemoTestSuite -->

- if testng.xml looks like, then it means, testsuite is collection of two test modules

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

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

<suite name="DemoTestSuite">

<test name="TestFirstModule">

<classes>

<class name="DemoTestCases.DemoTestCase1"/>

<class name="DemoTestCases.DemoTestCase2"/>

<class name="DemoTestCases.DemoTestCase3"/>

</classes>

</test> <!-- TestFirstModule -->

<test name="TestSecondModule">

<classes>

<class name="DemoTestCases.DemoTestCase4"/>

<class name="DemoTestCases.DemoTestCase5"/>

<class name="DemoTestCases.DemoTestCase6"/>

</classes>

</test> <!-- TestSecondModule -->

</suite> <!-- DemoTestSuite -->

- To run complete suite

right click on testng.xml

run as

TestNG Suite.

\*/

/\*

Here how to pass, fail and skip the test methods

\*/

@Test

**public** **void** demoPassTest()

{

String actualContent = "DemoTestCase6\_hello";

String expectedContent = "DemoTestCase6\_hello";

Assert.*assertEquals*(actualContent, expectedContent);

}

@Test

**public** **void** demoFailTest()

{

String actualContent = "DemoTestCase6\_hello";

String expectedContent = "DemoTestCase6\_bye";

Assert.*assertEquals*(actualContent, expectedContent);

}

@Test

**public** **void** demoSkipTest()

{

**throw** **new** SkipException("DemoTestCase6=>This test has been skipped, as of now, it is not in use");

}

@Test

**public** **void** demoFailAnotherTest()

{

Assert.*assertTrue*(**false**,"DemoTestCase6=>It has been failed, because condition is not met");

}

}