1. **TestNG:**

**U can control test cases , how they have to run, execution, group**

1. **Install and configure:**

>Testng.org: you can add .jar’s to build path or install testNG plugin for eclipse

> installing Plugin- goto testNG site> eclipse>how to install> copy URL for testNG plugin for eclipse

> Go to eclipse>help>install new soft> paste url in work with>wait>tick testNG>next>next>finish

1. **How to run test without Java compiler**

> Create new java Project

1. Add new package xyz- for storing all java class file

2. Create new class file in xyz package> don’t tick on public static ……..

3. create one public method eg.

**@Test**

**Public void FirstTest()**

**{**

**System.out.println("I am first test case");**

**}**

**@Test**

**Public void SecondTest()**

**{**

**System.out.println("I am second test case");**

**}**

> The @Test adds testNG annotations, any method after @Test will be treated as test case by TestNG

> For above Demo is name for test case

> run as TesNG test o/p test will run

> u can create multiple test cases from single class

1. **Creating TestNG XML file**

>Right click on project >> TestNG>> convert to TestNG

>XML hierarchy :::: test suite>>>test folder>>>test case

**<suite name=” name of suite”>**

**<test name="name of test">**

**<classes>**

**<class name="test.name of class file created in package of java project">**

**</class>**

**</classes>**

**</test>**

**</suite>**

1. **Editing TestNG XML:**

> you can changes or model the test cases based on functions (viz- mobile brands/tv brands etc) and trigger as we want

> **how to Exclude particular test cases from one class:** As we know method means test case , so we need to exclude that particular method from running syntax in XML as follows

**<suite name=” name of suite”>**

**<test name="name of test">**

**<classes>**

**<class name="test.name of class file created in package of java project">**

**<methods>**

**<exclude name="name of method"></exclude>**

**</methods>**

**</class>**

**</classes>**

**</test>**

**</suite>**

> **how to Include only particular test cases from one class:** as above XML syntax we need only to **add <include name=”name of test case or method”></include>**

**Scenario:** if we have many brands mobiles / washing machine/ tv in single class file and we want only exclude the washing machines of all brands then add **washingmachine.\*** in exclude syntax , Note: in framework naming convention is most important as for washing machines of all brands method name must start with washingmachineXYZ/ABC etc

**>How to add package in TestNG XML:** The Syntax for adding package in TestNG XML as follows which allow to execute all classes and all the test cases/methods from each class when we rung testNG XML as Suite

**<suite name="Name of suite">**

**<test name="Name of test">**

**<packages>**

**<package name="name of package in which all the class files are created"></package>**

**</packages>**

**</test>**

**</suite>**

**>TestNG Annotations:**

**a.** @BeforeTest : This annotation is used for executing the particular test case before running any testcase/method from test in testNG XML file which includes the class which have beforetest testcase

**Note:** This annotation is used when there is need of prerequisite to execute eg. Server to start etc

**b.** @AfterTest : This annotation is used for executing the particular test case after all testcase/method from test in testNG XML file which includes the class which have afterTest testcase

**c.** @BeforeSuite & @AfterSuite: it is used to set global environment variables to framework i.e before executing testcases in whole framework if there is need to set env’s that will be take care by this annotations

**we can set this annotation in any testcase/method of any class in the whole suite**

**d.** @BeforeMethod & @AfterMethod: these annotations are used when we need to execute particular test case before starting every testcase(methods in class) in class and after execution of each test case in single class file , this annotation is for class file only, it don’t have any relation with XML

**e.** @BeforeClass & @AfterClass: these annotations are used when we need to execute particular test case before start executing the testcases(methods in class) in that class and after execution of last test case of that class file , this annotation is for class file only , it don’t have any relation with XML

**>Helping Attributes in TestNG:**

**Grouping:** this is used for executing particular set of test cases , it is like running the smoke test . this is Helper Attribute

**a.** @Test(groups= {"smoke"}):we need to add the tag like this here “smoke” is just a name for representation we can take any name we want. We have to add this syntax for the test cases which we need to execute across the classes

**<suite name="name of the suite">**

**<test name="name of the test">**

**<groups>**

**<run>**

**<include name="name of the group"></include> (we can also exclude the particular group of test cases from running)**

**</run>**

**</groups>**

**<classes>**

**<class name="test.name of the class"/>**

**<class name="test. name of the class "/>**

**<class name="test. name of the class "/>**

**<class name="test. name of the class "/>**

**</classes>**

**</test>**

**</suite>**

**b.** @Test(dependsOnMethods={"MethodName"})**:** this feature(Helper Attribute) is used when one test case/ Method is depends on other test case/Method.

**@Test(dependsOnMethods={"RunFirst"})**

**public void RunSecond()**

**{**

**System.out.println("I am running Second");**

**}**

**@Test**

**public void RunFirst()**

**{**

**System.out.println("I am running First");**

**}**

**Explanation: this feature is useful when one test case is depends on another. In above case the RunSecond test case is depends on RunFirst test case that means if RunFirst test case is not executed then RunSecond test case will give some error in Output .**

**c.**  @Test(enabled=**false**): **This helper attribute is used when we want to skip particular test case/Method because of known issue/failure. Then for that particular test case this helper attribute is used.**

**d.**  @Test(timeout=**4000**): **This helping attribute is used to wait the execution of test case when there is condition e.g.test case is invoking the server**

**>Parameterization: This is used when there is some values like url,username, passwords etc which are common across the test cases in framework, so we use parameter in TestNG XML file**

**<suite name=” name of suite”>**

**<parameter name="URL" value="https://wetesterr.com/"/> (This is example) OR**

**<test name="name of test">**

**<parameter name="URL" value="https://wetesterr.com/"/> (This is example) THis**

**<parameter name="Username" value="yogesh"/>(This is example)**

**<classes>**

**<class name="test.name of class file created in package of java project">**

**</class>**

**</classes>**

**</test>**

**</suite>**

**And in class following are the syntax**

**@Parameters({"URL",”Username”})**

**@Test**

**public void parameter(String urlname, String name)**

**{**

**System.out.println(urlname);**

**System.out.println(name);**

**}**

**output will be:** [**https://wetesterr.com/**](https://wetesterr.com/)

**yogesh**

in the above example we have set parameter at suite level that means we can use this URL parameter at any class in this frame work.

Like this we can set parameter at test folder level so that we can parameterize values at test level.

**>Data Provider Annotations:** This annotation is used for parameterizing with multiple data sets and using those data sets for running tests with multiple combinations.

This is specific to test case/method only not entire test folder or test suite

The data provider can be defined as follows

Here we took the example as username and password for 3 different users and pass them to test case or method

**@Test(dataProvider="dProvider")**

**public void Dp(String username, String password)**

**{**

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

**System.out.println(username);**

**System.out.println(password);**

**}**

**@DataProvider**

**public Object[][] dProvider()**

**{**

**//prime user details--1st customer**

**//General user details-2nd customer**

**//new user details-- 3rd customer**

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

**data [0][0]="primeuser";**

**data [0][1]="primepass";**

**data [1][0]="generaluser";**

**data [1][1]="generalpass";**

**data [2][0]="newuser";**

**data [2][1]="newpass";**

**return data;**

**}**

**The output will be as follows:**

**Login Details**

**primeuser**

**primepass**

**Login Details**

**generaluser**

**generalpass**

**Login Details**

**newuser**

**newpass**

**>TestNG listeners:** This annotation is used when there is condition in test eg. If want to grab a screenshot after test get failed then the listeners can be use in such scenario.

**ITestListener** : this is an interface which have all the method for TestNG listeners. This interface implements TestNG Listeners to the framework

**package test;**

**import org.testng.ITestListener;**

**public class Listeners implements ITestListener {**

**}**

**Implements :** this is the keyword to call the interface in the class

TestNG as follows:

**<suite name=” name of suite”>**

**<listeners>**

**<listener class-name="test. name of class file created for defining listeners in package of java project "/>**

**</listeners>**

**<test name="name of test">**

**<classes>**

**<class name="test.name of class file created in package of java project">**

**<methods>**

**<exclude name="name of method"></exclude>**

**</methods>**

**</class>**

**</classes>**

**</test>**

**</suite>**

***===========================================END OF TESTNG==================================================***

**\*\*Driving Global Environment values from External Files to the Test in Java\*\***

1. **For defining the values globally we need to create .properties file in the project package eg as follows:**

**File name : data.properties**

**File content:** browser=chrome

url=https://wetesterr.com

**To get the values from .properties file to Java class we need to use properties class in java as follows:**

Properties prop=**new** Properties();

**We need to create file object for giving information about .properties file to the properties class, for that we need to create file object for this location, for that we need to create file object by class fileinputstream as follows:**

FileInputStream fis=**new** FileInputStream("path of .properties file");

**Here file object is ready now we need to provide this file object information to the properites object as follows:**

prop.load(fis);

**now we can get the values from .properties file to the test case/method as follows:**

prop.getProperty("here is the property we need to fetch from .proeprties file");

**we can modify the the properties from class file also as follows:**

prop.setProperty("browser", "firefox");

**to write back the changes made in .properties we need to set FileOutputStream as follows:**

FileOutputStream fos=**new** FileOutputStream("path of .properties file");

prop.store(fos, **null**);

***===========================================END OF Global env to class==================================================***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\*\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**\***

**All about MAVEN**

**It is an Software project Management and build management Tool for Java frameworks**

1. Why Maven:

* It is a Central Repository to get dependencies
* Maven is used to maintain the common structure across the organization
* Flexibility in Integration with CI tools
* Plugins for test framework execution

1. Maven Terminologies

* ArtifactId: it is a file, usually jar that gets deployed to a maven repository
* GroupId: it is helpful to identify project uniquely across all the projects
* Archtype:generate: it is a template for maven project

1. Creating new Maven project

* Create new maven project in eclipse
* After creating new maven project copy required dependencies from maven repo eg. testNG, selenium, Appium, restAssured etc

1. Plugin used for executing all maven test cases

* **Surefire plugin** is used to execute all tests in Maven project
* We need that plugin in POM.xml file
* Copy surefire plugin from there official site and paste it in pom.xml above <dependencies>

1. Running Maven project

* There is 3 steps/commands to run the maven project from CMD
* **1. mvn clean:** this is used for cleaning the build if there is previously run project/tests
* **2.** **mvn compile:** it will scan and check all syntax of test and if there is any syntax error it will inform as BUILD SUCCESS or BUILD FAILED
* **3.** **mvn test:** This command will trigger the test execution

1. The test java file must end with test to run with maven eg SeleniumTest.java
2. Running testNG.xml from maven project

* In surefire plugin page > navigate to Using TestNG and froms

**Using suite XML files** Copy <Configuration> and paste it in pom.xml at the end of surefire plugin

This allows flexible configuration of the tests to be run

<configuration>

<suiteXmlFiles>

**<suiteXmlFile>testng.xml</suiteXmlFile> //pass the path of testNG.xml**

</suiteXmlFiles>

</configuration>

1. Profiling:

* It is used when there is multiple XML’s in maven project and we need to run certain XML
* In POM.xml we can add plugin under build >>pluginMangement for different profiles which we can differ by different ID’s as follows:

<profiles>

<profile>

<id>Name of Porfile</id>

<build>

Plugins which have surefire plugin with tesNG path

<build>

<profile>

<profiles>

Command use to run particular profile : mvn test -PRegression(here regression is name of profile defined in pom.xml)