**TestNG:**

* ‘Test Next Generation’ is a unit testing framework.
* Using TestNG we will run multiple test classes & generate result.
* TestNG is available as plugin for ‘Eclipse’.

**Using TestNG, we can perform following actions:**

1.Execute multiple classes

2.Generate the execution results in HTML format

3.Re-execute only the failed test scripts

4.Control the execution flow of the scripts

**Install TestNG:** **TestNG –**

**Steps to install testNG**

<https://www.techbeamers.com/install-testng-in-eclipse-ide/>

**For Downloading**

[**https://github.com/cbeust/testng-eclipse**](https://github.com/cbeust/testng-eclipse)

**For installing online**

<https://testng.org/testng-eclipse-update-site>

1. In ‘Eclipse’ go to ‘Help’ & select ‘Eclipse Marketplace’
2. Search for ‘TestNG’.
3. Click on ‘Install’ button of ‘TestNG’. Follow the default instructions then finish. It will restart the eclipse.

* Before using ‘TestNG’ we should associate with ‘Java project’.

**Associate TestNG with Java project:**

1. Right click on ‘Java project’.
2. Go to ‘Build Path’.
3. Select ‘Add Library’.
4. Select ‘TestNG’.
5. Click on ‘Next’ & click ‘Finish’.

**TestNG class:**

|  |
| --- |
| public class **Demo**  **{**  **@Test *//Test Method***  **public void name()**  **{**  **Reporter.log("**My *Name is sudeer***",** **true);**  **}**  **}** |

* The above java class is called as ‘**TestNG class**’ or ‘**Test class**’ because it has test method.

1.We write the selenium code in the TestNG class

2.TestNG class is a java class which contains **test method** instead of **main method**

3.Any method written below @Test is called as test method

[4.@Test](mailto:4.@Test) is one of the TestNG annotation

5.Annotation is syntactic metadata

6.When we run any TestNG class, it automatically generates the result in html format inside **test-output** folder(This folder will be automatically created by TestNG inside your java project folder)

* To see the report, ‘**Refresh**’ the java project, expand ‘**test-output**’ folder and right click on ‘**emailable-report.html**’. Go to ‘**Open with**’ & select ‘**Web Browser**’.
* To export this html report into ‘Excel Report’, right click on the page & select ‘Export to Microsoft Excel’.

7.In-order to write any information into the html report as well as into console, we use **log()** method of **Reporter** class.

Write and execute TestNG class

**How do you run ‘Test method’ multiple times without using looping statement?**

**🡪** By using ‘**invocationCount’**

**Note:**

1. Default ‘invocationCount’ is 1. If we give 0 or negative numbers it will not execute the ‘Test method’.
2. Decimal values & variables are not allowed.
3. ‘invocationCount’ is used to run the ‘Test method’ multiple times with same data. To run the ‘Test method’ multiple times with different data we use ‘DataProvider’ (**@DataProvider**).

**How do you run multiple Test classes in TestNG?**

**🡪** By using **‘TestNG Suite’** file.

**Create ‘Suite File’:**

1. Right click on ‘Java Project’.
2. Go to ‘TestNG’.
3. Select ‘Convert to TestNG’.
4. Click on ‘Finish’.

* It generates ‘**testng.xml’** file inside the java project. Double click it to see the content of the xml file

**Run TestNG ‘Suite file’:**

1. Right click on ‘TestNG.xml’.
2. Go to ‘Run As’.
3. Select ‘TestNG Suite’

**Q) If a ‘Test class’ contains multiple ‘Test methods’ then in which order they are executed?**

**🡪** It executes based on alphabetical order of ‘Test method’ names.

**Q) How do you execute ‘Test methods’ in required order?**

**🡪** By using ‘**priority**’

**Ex:-** **@Test(priority=1)**

* **Default priority is zero.**
* If two ‘Test methods’ have same priority than those two methods will be executed based on alphabetical order.
* Priority value can be negative but we cannot use decimal numbers & variables.

**Important Annotations of TestNG:**

1. @BeforeMethod
2. @AfterMethod
3. @BeforeClass
4. @AfterClass
5. @BeforeTest
6. @AfterTest
7. @BeforeSuite
8. @AfterSuite
9. @DataProvider
10. @Test

**Note:**

1. **‘@BeforeMethod**’ will be executed **before** the execution of each & **every** ‘**Test method**’ iteration.
2. **‘@AfterMethod**’ will be executed **after** **every** ‘**Test method**’ iteration.
3. **‘@BeforeClass**’ will execute **before** every ‘**Test class**’.
4. **‘@AfterClass**’ will execute **after** every ‘**Test class**’.

See the program

**Inheritance in TestNG:**

Generally, for every manual test case we create a TestNG class and each test case may have some common steps. Instead of writing the common steps in every class, we create a parent class where the common code is written then we make all the remaining TestNG classes to extend this parent class

**See the programs-** S33\_BaseClass, S34\_Test1 and S35\_Test2 classes—its xml file as well

**TestNG Groups:**

By default when we execute the testng suit file, it runs all the TestNG classes. In-order to execute only required methods, we should create the groups and mention it in TestNg suite file as filtering condition. (“Include” and “Exclude” are the filtering conditions)

|  |
| --- |
| public class TestNGGroups  {  **@BeforeMethod(alwaysRun=true)**  public void login()  {  Reporter.log("Login", true);  }  **@AfterMethod(alwaysRun=true)**  public void logout()  {  Reporter.log("Logout", true);  }  **@Test(groups= {"Customer", "Smoke"})**  public void createCustomer()  {  Reporter.log("Create Customer", true);  }  **@Test(enabled=false, groups= {"Customer"})**  public void deleteCustomer()  {  Reporter.log("Delete Customer", true);  }  **@Test(groups= {"Project”, “Smoke"})**  public void createProject()  {  Reporter.log("Create Project", true);  }  **@Test(groups= {"Project"})**  public void deleteProject()  {  Reporter.log("Delete Project", true);  }  } |

In Automation if we want to execute particular set of ‘Test Scripts’ we need to update ‘TestNG.xml’

|  |  |
| --- | --- |
| ***TestNG.xml:***  <suite name="Suite">  **<groups>**  **<run>**  **<include name="Smoke"/>**  **</run>**  **</groups>**  <test name="Test">  <classes>  <class name="TestNG\_New.TestNGGroups"/>  </classes>  </test>  </suite> | ***Output:***  Login  Create Customer  Logout  Login  Create Project  Logout |

|  |  |
| --- | --- |
| ***TestNG.xml:***  <suite name="Suite">  **<groups>**  **<run>**  **<include name="Customer"/>**  **</run>**  **</groups>**  <test thread-count="5" name="Test">  <classes>  <class name="TestNG\_New.TestNGGroups"/>  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> | ***Output:***  Login  Create Customer  Logout |

|  |  |
| --- | --- |
| ***TestNG.xml:***  <suite name="Suite">  **<groups>**  **<run>**  **<exclude name="Customer"/>**  **</run>**  **</groups>**  <test thread-count="5" name="Test">  <classes>  <class name="TestNG\_New.TestNGGroups"/>  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> | ***Output:***  Login  Create Project  Logout  Login  Delete Project  Logout |

**Note:**

* ‘**alwaysRun=true’** indicates the method belongs to all groups & that method is executed irrespective of any groups (include or exclude)
* ‘**enabled=false**’ indicates the specific ‘Test method’ should be skipped during the run time.

**Q) How do you compare the result in TestNG?**

**🡪** Using ‘**Assert**’ class. We compare actual result with expected result using ‘**assertEquals( )’** method of ‘Assert’ class.

|  |
| --- |
| public class **CompareResult\_TestNG**  **{**  **@Test**  public void **createUser()**  {  String eResult="Sudeer Kumar"; *//To be taken from excel file*  String aResult="Sudeer Raj"; *//To be taken from application*  **Assert.assertEquals**(***aResult, eResult***);  Reporter.log("Done", true); *//This statement will not execute*  }  } |
| ***Output:***  **FAILED:** createUser  java.lang.AssertionError: **expected** [Sudeer Kumar] **but found** [Sudeer Raj] |

**Q) How do you perform verification in TestNG? How do you compare expected & actual results?**

**🡪** Using ‘**Assert**’ class.

**Note:**

1. All the methods present in ‘Assert’ class are **static.**
2. When we use ‘assertEquals()’ method of ‘Assert’ class it will compare the actual result with expected result then it will report the status (PASSED or FAILED) in TestNG report.
3. If comparison fails ‘**Assert’** will **not execute** the remaining statements of current ‘Test method’.
4. In order to **continue the execution** of current ‘Test method’ **even after comparison fails** we should use ‘**SoftAssert**’ class.
5. All the methods present in the ‘SoftAssert’ class are **non-static**.

|  |
| --- |
| public class **CompareResult\_TestNG\_SoftAssert**  {  **@Test**  public void **createUser()**  {  **SoftAssert s=new SoftAssert();**  String eResult="Vijay Mutalik"; //To be taken from excel file  String aResult="Vijay M"; //To be taken from application  **s.assertEquals(aResult, eResult);**  Reporter.log("**Done**", true);  **s.assertAll();**  }  } |
| ***Output:***  **Done**  **FAILED:** createUser  java.lang.AssertionError: The following asserts failed:  **expected** [Vijay Mutalik] **but found** [Vijay M] |

**Note:-** Do not write any code after ‘**assertAll()**’ statement.

**Q) What are the differences between ‘Assert’ & ‘SoftAssert’?**

|  |  |
| --- | --- |
| **Assert** | **SoftAssert** |
| 1. All methods are **static**. 2. It will **not execute** the remaining statements of current ‘Test method’ if comparison fails. 3. We **do not call** ‘**assertAll()**’ method. | 1. All methods are **non-static** 2. It will **execute** the remaining statement of current ‘Test method’ if the comparison fails. 3. We **must call** ‘**assertAll()**’ method **at the end.** |

**Q) How do you execute a ‘Test method’ only when another ‘Test method’ is passed?**

🡪 By using ‘**dependsOnMethods**’

|  |
| --- |
| public class **Test\_DependsOn\_OtherTest**  {  **@Test**  public void **createUser()**  {  SoftAssert s=new SoftAssert();  s.assertEquals("Vijay", "Vijay Mutalik");  Reporter.log("Created", true);  s.assertAll();  }  **@Test(dependsOnMethods="createUser")**  public void **deleteUser()**  {  Reporter.log("Delete User", true);  }  } |

* In the above example ‘**deleteUser()**’ test method will executes only if ‘**createUser()**’ test method **passes**. ‘**deleteUser()**’ test method will **skip** if the ‘**createUser()**’ method fails.

**POM (Page Object Model)**

**Q) How do you handle ‘StaleElementReference’ exception?**

**🡪** Using **‘Page Object Model (POM)’**

**Q) What is Page Object Model?**

**🡪** POM is a ‘**Java Design Pattern’**

**Q) How do you declare an element in the POM class?**

**🡪** Using ‘**FindBy**’ annotation **(@FindBy)**

**Q) What is the syntax of ‘FindBy’ annotation?**

|  |  |
| --- | --- |
| ***Syntax:***  **@FindBy(*locator*=”***value***”)**  **access\_specifier** ***WebElement*** element\_name**;** | ***Example:***  **@FindBy(id*="****username****"*)**  **private *WebElement*** unTB**;** |

**Q) How do you handle multiple elements (findElements) in POM class?**

**🡪** Using ‘**FindBy**’ annotation only

|  |  |
| --- | --- |
| ***Syntax:***  **@FindBy(*locator*=”***value***”)**  **access\_specifier** ***List<WebElement>*** element\_name**;** | ***Example:***  **@FindBy(xpath*="****//a****"*)**  **private *List<WebElement>*** allLinks**;** |

**Q) How do you initialize the elements present in POM class?**

**🡪** Using ‘**initElements()**’ **method** of ‘**PageFactory**’ **class**. I.e. using “**PageFactory.initElements()**” statement we can initialize the elements in POM class.

**Q) What is the syntax of ‘initElements()’ method?**

**🡪** **PageFactory.initElements(*driver*, *POMClassObject)*;**

**Ex:-** PageFactory.initElements(*driver*, *l*); [Here *l*=LoginPage() ]

**Q) What happens if we do not use ‘initElements()’ method of PageFactory?**

**🡪** We get ‘**NullPointer**’ exception.

**Important Notes:**

1. In POM class, in the place of ‘locator value’ we should not use variables.

**Ex:-** ***String*** xp=”//a”;

**@FindBy(**xpath=xp**)**

1. But we can use constants.

**Ex:-** ***final String*** xp=”//a”;

**@FindBy(**xpath=xp**)**

1. In POM, we develop 2-types of classes.

Page Class

Test Class

1. Number of ‘Page class’ should be same as number of web pages present in the application whereas number of ‘Test class’ should be same as number of manual test cases.
2. ‘Page class’ contains elements & methods hence ‘Page class’ is called with different names such as

Element Repository Class

Object Repository Class

Page Object Model Class

Page Object Repository Class

**Q) What is the difference between ‘Page Object Model’ & ‘PageFactory’?**

**🡪** ‘Page Object Model’ is one of the ‘Java Design Pattern’ whereas ‘PageFactory’ is a class.

**Page Object Model class for Login Page:**

|  |
| --- |
| public class **POM\_LoginPage**  **{**  ***//Declaration***  @FindBy(id="username")  private WebElement unTB;  @FindBy(name="pwd")  private WebElement pwTB;  @FindBy(xpath="//div[.='Login ']")  private WebElement loginBTN;  ***//Initialization***  **public POM\_LoginPage(*WebDriver* driver) *//Constructor***  **{**  **PageFactory.initElements(driver,this);**  **}**  **//Utilization**  public void setUserName(String un)  {  unTB.sendKeys(un);  }  public void setPassword(String pw)  {  pwTB.sendKeys(pw);  }  public void clickLogin()  {  loginBTN.click();  }  **}** |
| public class **POM\_LoginTest**  **{**  **static**  **{**  String key="webdriver.chrome.driver";  String value="./Drivers/chromedriver.exe";  System.setProperty(key, value);  **}**  public static void main(String[] args) throws InterruptedException  **{**  String a[][]=new String[2][2];  a[0][0]="admin";  a[0][1]="abcd";  a[1][0]="admin";  a[1][1]="manager";  WebDriver driver=new ChromeDriver();  driver.get("http://localhost:8080/login.do");  POM\_LoginPage l=new POM\_LoginPage(driver);  for(int i=0;i<=1;i++)  {  String un=a[i][0];  String pw=a[i][1];  System.out.println(un+" "+pw);  l.setUserName(un);  l.setPassword(pw);  l.clickLogin();  }  driver.close();  **}**  **}** |