TESTNG

What is TestNG

→ TestNG is an automation testing framework in which NG stands for Next Generation.
TestNG is inspired by **Junit** which uses annotations. TestNG overcome the disadvantages of Junit and is designed to make E2E testing easy.

FEATURES :-

1). Generate the report in proper format including the no. of test cases run, pass, failed and skipped.
2). Multiple test cases can be grouped easily.
3). We can assign priority to test cases.
4). Same test cases can be executed multiple time without loops using keyword - "Invocation count".
5). You can execute multiple test cases on the multiple browser.
6). Annotations used are very easy to under ND.
7). TestNG simplifies the way tests are coded. There is no need for static main method. Sequence is regulated by annotations that do no require method to be static.
8). Uncaught exceptions are handled by testNG.

TestNG Annotations

① **@BeforeSuite**

→ The annotated method will run before all tests in this suite have run.

② **@AfterSuite**

→ The annotated method will run After all tests in this suit have run.

③. **@BeforeTest**

➡ The annotated method will run before any test method belonging to classes inside they tag is run.

④. **@AfterTest**

➡ The annotated method will run after all test method belonging to classes inside the tag is run.

⑤ **@BeforeClass**

➡ The annotated method will run before the first test method in current class is invoked.

⑥. **@AfterClass**

➡ The annotated method will run after all the test method in current class is invoked.

⑦. **@BeforeMethod**

➡ The annotated method will run before each test method.

⑧. **@AfterMethod.**

➡ The annotated method will run after each test method.

⑨. **@Test**

➡ The annotated method is a part of test case.

⑩ **@BeforeGroups / @AfterGroups**

After ➡ This method is guranteed to run shorty after the last test method belong to any of specified group is invoked.

Before ➡ This method is guranteed to run shortly before the first test method belong to any of specified group is invoked.

TestNG Annotations

Sign ON	Register	Support	Contact

Flow :-

1). Go to Homepage and Varify title.
2). Click Register and Verify title.
3). Go back to Homepage and Varify title.
4). Click Support and Varify title.
5). Go back to homepage and Varify title.

```
import org.openqa.selenium.*;
import org.testNG.Assert;
import org.testNG.annotations.*;

Public class Test {
    Public string baseurl = "http://demo.test.com";
    String driverpath = PATH/geckodriver.exe;
    Public WebDriver driver;
    Public string expected = null;
    Public string actual = null;

@ BeforeTest
    Public Void launchBrowser( ) {
    S.O.P. ("Launching Firefox browser");
    System.setProperty ("Webdriver.gecko.driver",
                        driverpath);

    driver = new FirefoxDriver();
    driver.get (baseurl);
  }
@ BeforeMethod
    Public void VerifyHomeTitle() {
    String expectedTitle = "Welcome: Home";
    String actualTitle = driver.getTitle();
  Assert.assertEquals (actualTitle, expectedTitle);
  }
```

```
@Test(priority=0)
    Public void register(){
    driver.FindElement(By.id("register")).click();
    expected = "Welcome: Register";
    actual = driver.getTitle();
    Assert.assertEquals(actual, expected);
}

@ Test(Priority=1)
    Public void support(){
    driver.findElement(By.id("support")).click();
    expected = "Welcome: support";
    actual = driver.getTitle();
    Assert.assertEquals(actual, expected);
}

@ AfterMethod
    Public void goToHome(){
    driver.FindElement(By.linkText("Home")).click();
}

@ AfterTest
    Public void close Browser(){
    driver.close();
    }
}
```

ATUL KUMAR (LINKEDIN).
TELEGRAM - NOTES GALLERY.

TestNG Priority

1). If we don't assign priority to testNG method then they will execute in alphabetical order.

2). If two methods are having same priority then they will execute in alphabetical order.

3). **What is Prioritization in TestNG?**

➡ Prioritization in TestNG is a way to provide a sequence to methods so that they do not run out of order.

Syntax = @Test (priority = 1)

Lower the priority number, higher is the priority of the test case method.
Priority value can be negative, zero or Positive.
Priority cannot pass through XML files.
If no priority is assigned then default priority is zero.

4). **How to skip a test in TestNG using Parameter?**

➡ Using 'enable' parameter
ex - @Test (enable = false)

5). **How to execute failed test cases in TestNG?**

➡ Run testing-failed.xml.

Apache POI

- Apache POI in selenium in a widely used API for selenium data driven testing. It is a POI library written in JAVA that gives users on API for manipulating microsoft document like .xls and .xlsx.
- Users can easily create, modify and read/write data into excel.
- POI stands for poor Obfuscation Implementation.
- To read XLS files, HSSF Implementation used.
- To read XLSX files, XSSF Implementation is used.

* Class and Interfaces in POI

XLS classes	Interface	XLSX classes
HSSFWorkbook →	Workbook ←	XSSFWorkbook
HSSFSheet →	Sheet ←	XSSFSheet
HSSFROW →	Row ←	XSSFROW
HSSFCell →	Cell ←	XSSFCell

Parameterization in Selenium

Parameterization in selenium is a process to parameterize test scripts in order to pass multiple data to application at runtime.
It is a strategy of execution which automatically runs the test cases multiple times using different values.
The concept achieved by parameterizing the test scripts called Data Driven Testing.

There are two ways by which we can achieve parameterization in TestNG.

1). With the help of @ Parameters.
2). With the help of @ DataProvider

1). @ Parameters.

Parameters annotation in TestNG is a way to pass values to test methods as arguments using .xml files.
User may be required to pass values to test methods during runtime @ Parameters annotation can be used in any method having @Test @ Before @After or @ factory annotation.

Parameter annotation with TestNG

```
<?xml.version = "1.0"..... ?>
<!DOCTYPE.....>
< suite name = "Testsuite" thread-count ="3" >
< parameter name = "author" value = "chetan" />
< parameter name = "searchkey" value = "2states" />
< test name = "testauthor">
< parameter name = "searchkey" value = "3Idiots" />
< classes >
< class name = "packagename, Test ">
</class>
</classes>
</test>
</suite>
```

➡ **@ Optional Parameter**

```
@Test
@Parameters ({"author" , "searchkey"})
Public void Test (@Optional ("default") String author,
                  String searchkey)
```

➡ Package Packagename

```
import org.openqa.selenium.*;
import org.testng.annotations.*;
import java.util.*;
Public class Test {

String driverpath = "C:\\geckodriver.exe";

Webdriver driver;
@Test
@Parameters({"author", searchkey"}) Parameter value
                                    cannot be Typecasted
```

(If String is in the testNG then string is also in method.).

```
public void TestParam (@Optional("ABC"), String
        author, String searchkey) throws Exception {

System.setProperty("Webdriver.gecko.driver", driverpath)
driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.seconds)
                                                        ;

driver.get("http://demo.google.com");
WebElement searchText = driver.FindElement(By.name
                                        ("q"));

SearchText.sendkeys(searchkey);
S.O.P.("Welcome" + author + "your search Key is" +
                                    searchkey);
    }
  }
```

@DataProvider.

Dataprovider helps us to send multiple set of data from our excel sheet to test methods.

Design Pattern

1). **Page Object Model :**

Page object model also known as POM, is a design pattern in selenium that creates an object repository for storing all web elements. It is useful in code reusability and improving test case maintanance.

In POM, consider, each webpage in application is consider as class file. Each class file will contain corresponding web page elements.

Advantages of POM :

1). Page object design pattern says that operations & flow in UI should be separated from verification. This makes our code cleaner and easy to understand.

2). Object repository is independent of testcases so we can use same OR for different purposes with different tools.

3). Code become less and Optimized because of the reusable methods.

4). Readable and easy to maintain.

2]. **Page Factory :-**

Page factory is a class provided by the selenium webdriver to support page object design pattern. In page factory, testers use @FindBy annotation. The InitElements method is used to initialize web elements.

- @FindBy(id = "elementId") WebElement element;
  different locators like name, classname, tagname, xpath, css, linktext, partiallinktest can be use.

- **InitElement() :-**

  InitElements is a static method in Page factory class. Using InitElements method one can initialize all the elements located by @FindBy annotation.

- **Lazy Initialization :-**

  AjaxElementLocatorFactory is a lazy load concept in page factory.

Singletone Design Pattern

When we develop a class in a such a way that it can only have one(1) instance at any time is called **singletone design pattern.**

It is very useful when you use need to same object of a class accross all classes or framework.

Singleton class must return same instance again if it is instantiated again.

To create singleton class -

1). Declare constructor of a class as private.
2). Declare static reference variable of class. Static is needed to make it available globally.
3). Declare static method with return type as object of class which should check if class is already instantiated once.

* Whenever you feel you should have single instance of a class, you can use singleton pattern.

```
Class SingletonTest {
    Private static SingletonTest instance = null;
    Private SingletonTest() {
        S.O.P. ("Object created");

Public static SingletonTest getInstanceofClass() {
        if (instance == null)
        instance = new SingletonTest();

    return instance;
}
  }
```

```
Class TestDemo {
    P.S.V.M ( string args [] ) {
SingletonTest First = SingletonTest.getInstance -
                                -Of class ();
SingletonTest second = SingletonTest.getInstance-
                                -OfClass ();
}
    }
```

➡ When you run above program, you'll get "Object created" only once, when you create second instance it will not create / call constructor as it is already Initiated once.

How to use singleton in selenium Webdriver

```
import Org.Openqa.Selenium.*;
Public class SingletonClass {

Private static SingletonClass instance = null;
Private Webdriver driver;
Private SingletonClass () {
 System.setProperty("Webdriver.Chrome.driver",
                    "./execfiles/Chromedriver.exe";
 driver = new ChromeDriver();
}
 Public static SingletonClass getInstanceOfClass () {
     if (instance == null) {
     instance = new SingletonClass();
 }
   return instance;
 }
   Public Webdriver getdriver(){
      return driver;
 }
     }
```

LoadURL.java

```
import org.openqa.selenium.*;
Public class LoadURL {

        P.S.V.M (string args[]) {

Singleton class SC1 = Singleton class.getInstance of Class();
Webdriver d1 = SC1.getdriver();
Singleton class SC2 = Singleton class.getInstance of Class();
Webdriver d2 = SC2.getdriver();
d2.get("http://google.com");
    }
        }
```

➡ When you run LoadURL.java you will see browser launched and url will be opened in same browser. We have instantiated two instance of class, but both give the same instance of driver.

POM	Page Factory
1). It is an approach for design Patterns.	1). It is a class provide by selenium Webdriver.
2). It helps in separating Page Objects and scripts	2). It is a technique to implement POM.
3). 'By' annotation is used to define page objects.	3). @AndBy annotation is used to describe page objects.
4). There is cache storage to Perform tasks	4). There is no need for cache storage.
5). POM does not Provide lazy initialization.	5). Page factory does not Provide lazy initialization.
6). Need to initialize every page object individually.	6). All page objects are initialized using the InitElements( ) method.

Asserts in Selenium

Assert in selenium webdriver is used for varifying or validating scenario under test.
Based on the result of Assert, outcome of test case is decided.

Some widely used categories in selenium are :-

1). assertEquals , assertNotEquals.
2). assertTrue , assertFalse.
3). assertNull , assertNotNull.
4). assertsame , assertNotsame.

- **Types of Assert :-**

1). **Hard Assert :-** As the name indicate, test execution is halted when condition as a part of assert are not met. Hard assert usually throw an Assertion error. (java.long.AssertionError) and test case marked as failed as soon as hard assert condition is failed.
The assertion error should be handled in try... catch block.

2). **Soft Assert :-** Soft assert are used when the test method execution need not to be halted when assertion condition is not met.
In case of soft assert, errors are accumulated in each @Test execution and Assert All ( ) methods throw assert encountered during execution.

To get current page URL
⇒ driver.getcurrentURL( );

By default, asserts used in selenium are hard assert.

You need to import org.testng.Assert package for throwing appropriate asserts.

- import org.testng.asserts.SoftAssert;
- Assertion hassert = new Assertion();
- SoftAssert sassert = new SoftAssert();

—— × ——

```
driver.get(url)
driver.navigate().To();
driver.navigate().refresh();
driver.getCurrentURL();
driver.findElement(By.id(ID)).sendkeys(Keys.F5);
    driver.navigate().back();
    driver.navigate().forward();
    driver.close()
            VS driver.quit()
```

17. driver.close() method shall close the browser which is in focus.
driver.quit() method closes all the browsers.

27. driver.close() method closes active Webdriver instance.
driver.quit() method closes all the active Webdriver instance.

• RemoteWebDriver.

Selenium RemoteWebDriver is used to execute browser automation suite on remote machine. RemoteWebDriver class Implements webdriver Interface to execute scripts through RemoteWebDriver server on remote machine.

```
FirefoxOption firefoxopt = new firefoxOptions( );
WebDriver driver = new RemoteWebDriver
                   (new URL(URL), firefoxopt);
```

ATUL KUMAR (LINKEDIN).
TELEGRAM - NOTES GALLERY.

TestNG

1). `@Test(description = "Validate 200 status code for GET request")`

2). enabled = false to skip test case execution.

```
@Test(description = "Validate 200 status code",
      enabled = false)
```

3). alwaysRun = true ⇒ If set to true, this test method will always be run even if depends on a method that failed.

```
@Test(description = "Validate 200 status code",
      alwaysRun = true)
```

4). Groups

```
@Test(description = "Validate 200 status code",
      groups = {"smokeSuite", Regression-Suite"})
```

5]. ★ **How to rerun failed testcases in selenium ?**

➡ Rerun using testng - failed.xml.

→ 1). After first run of an automated test run, Right click on Project - Click on Refresh.

2). A folder will be generated name 'test-output' folder. Inside 'test-Output' folder, you could find 'testng -failed.xml'.

3). Run 'testng - failed.xml' to execute failed test cases again.

- **IRetryAnalyzer**

Create a class - RetryFailedTestCases to implement IRetryAnalyzer.

```
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

Public class RetryFailedTestCases Implements
    IRetryAnalyzer {
    Private int retrycnt = 0;
    Private int maxretrycnt = 2;

Public boolean retry (ITestResult result) {
        if (retrycnt  maxretrycnt) {
        S.O.P ("Retrying" + result.getName());
        retrycnt ++;
        return true;
    }
      return false;
}
    }
```

```
import org.testng.IRetryAnalyzer;
import org.testng.IAnnotationTransformer;
import org.testng.annotations.ITestAnnotations;

Public class RetryListner class implements
                IAnnotationTransformer {

    @ Override
    Public void transform (ITestAnnotation testannotation,
            class testclass, Constructor testConstructor,
            Method testMethod) {

IRetryAnalyzer retry = testannotation.
                    getRetryAnalyzer();

if (retry == null) {

    testannotation.setRetryAnalyzer (Retry failed
                                Testcases.class);
}
    }
        }
```

—x—

```
< listeners >
  < listener class-name = "PackageName.Retry-
                          Listenerclass" / >
< listeners >
```