**TestNG**

TestNG is a **java unit framework** use for writing/designing of **Test classes**.

<https://github.com/cbeust/testng-eclipse>

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

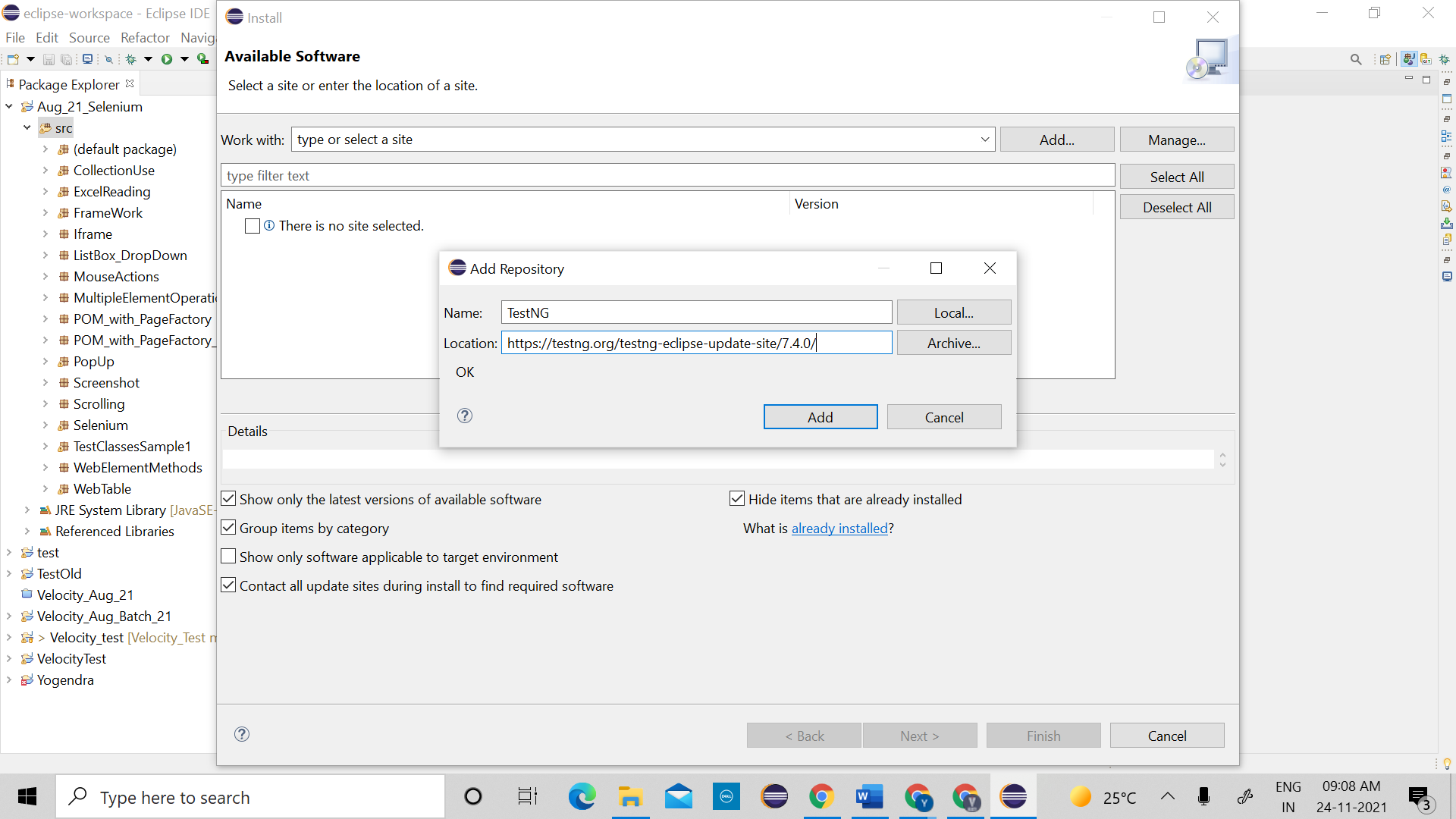
#### Go to downloads🡪 search for Install from update site

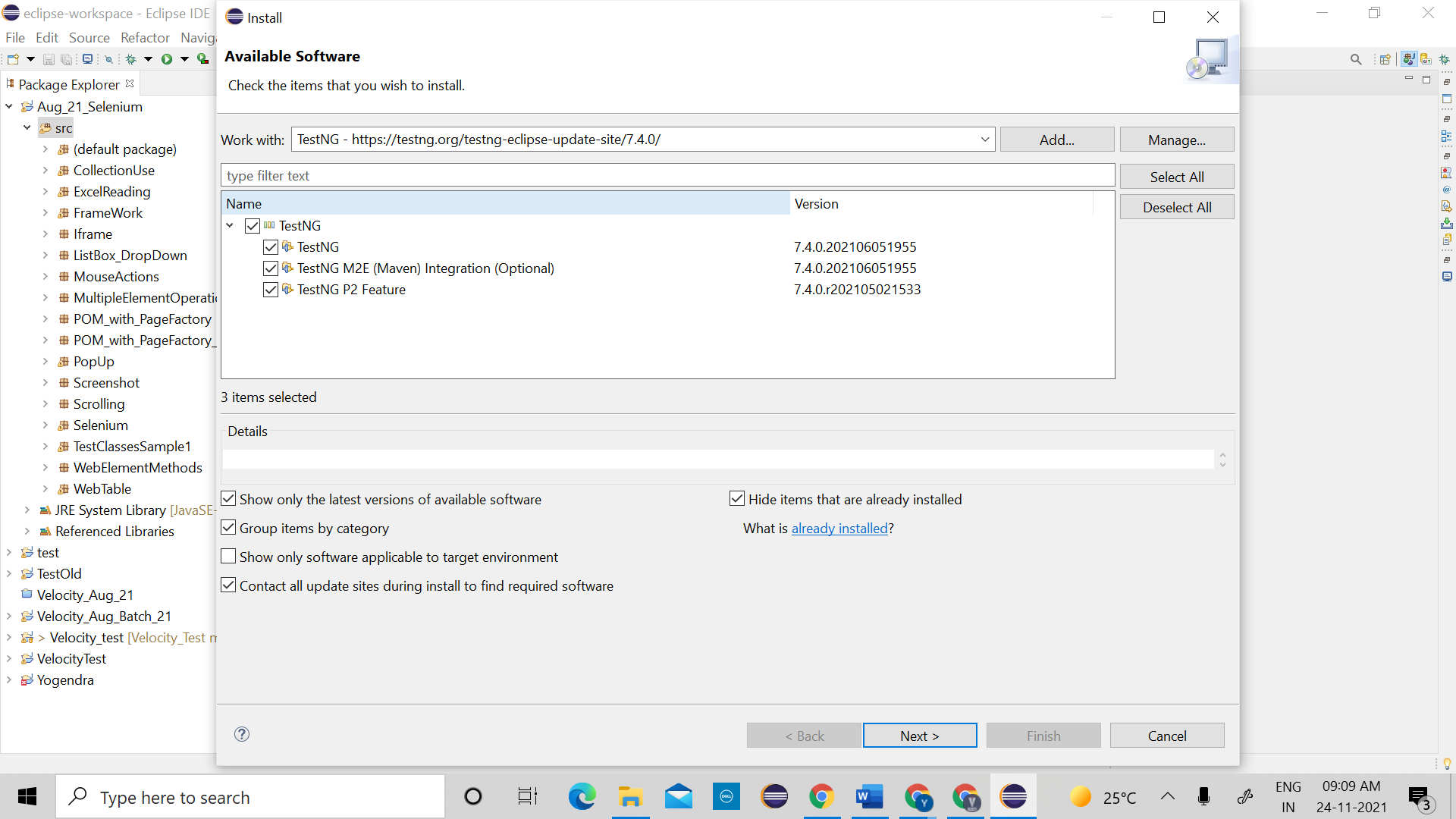
Graphical user interface, text, application, email

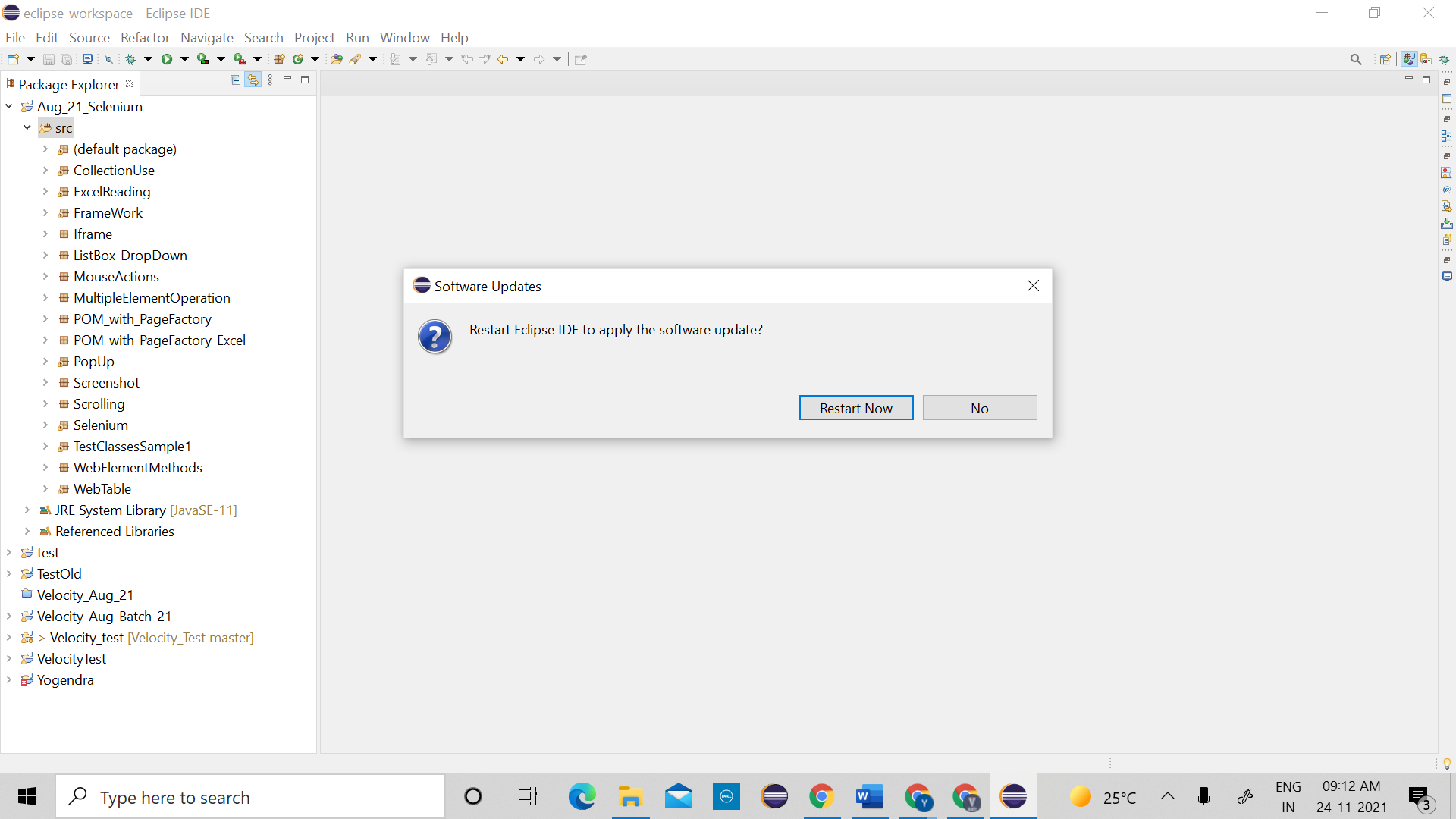
Description automatically generated

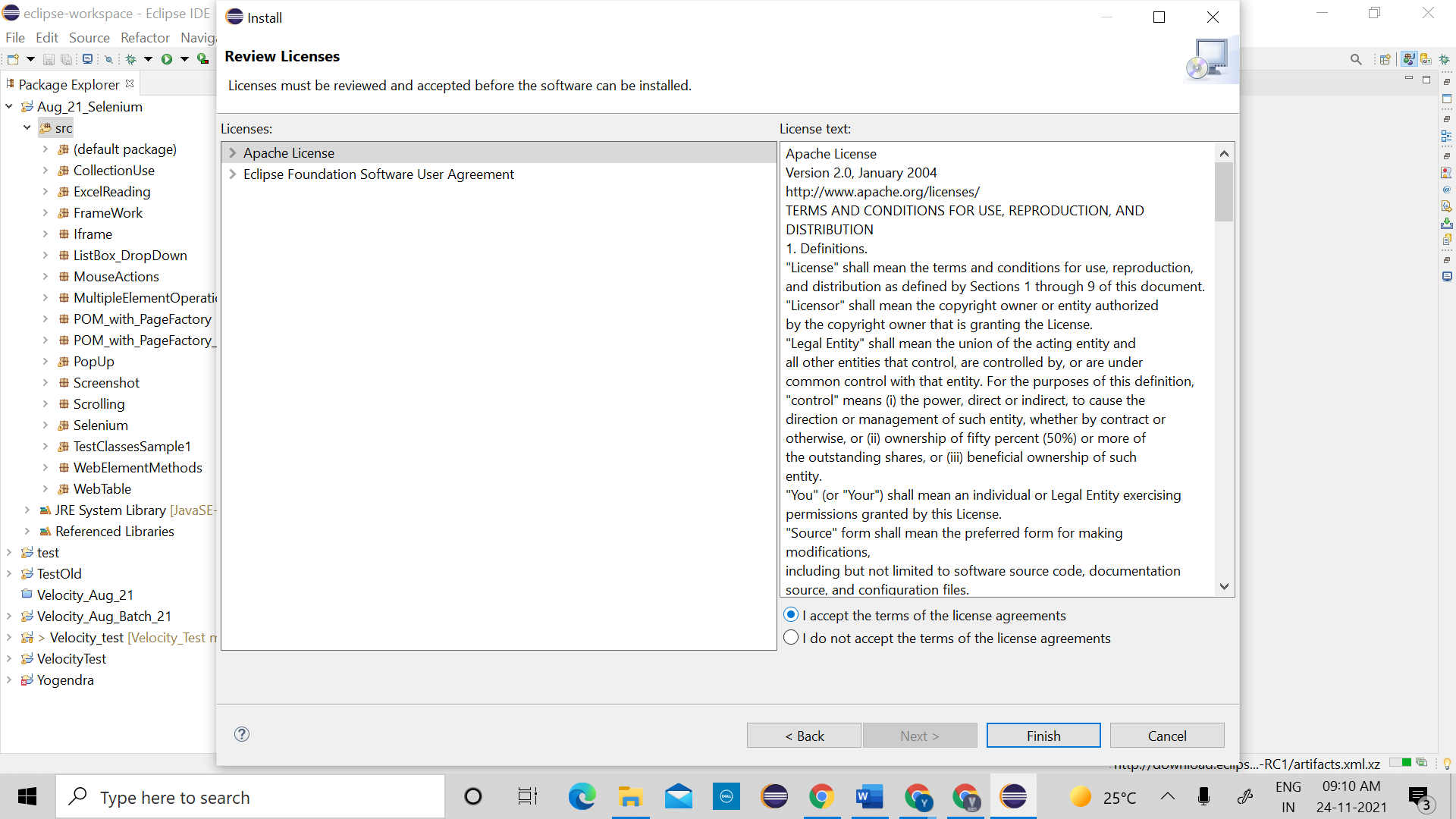
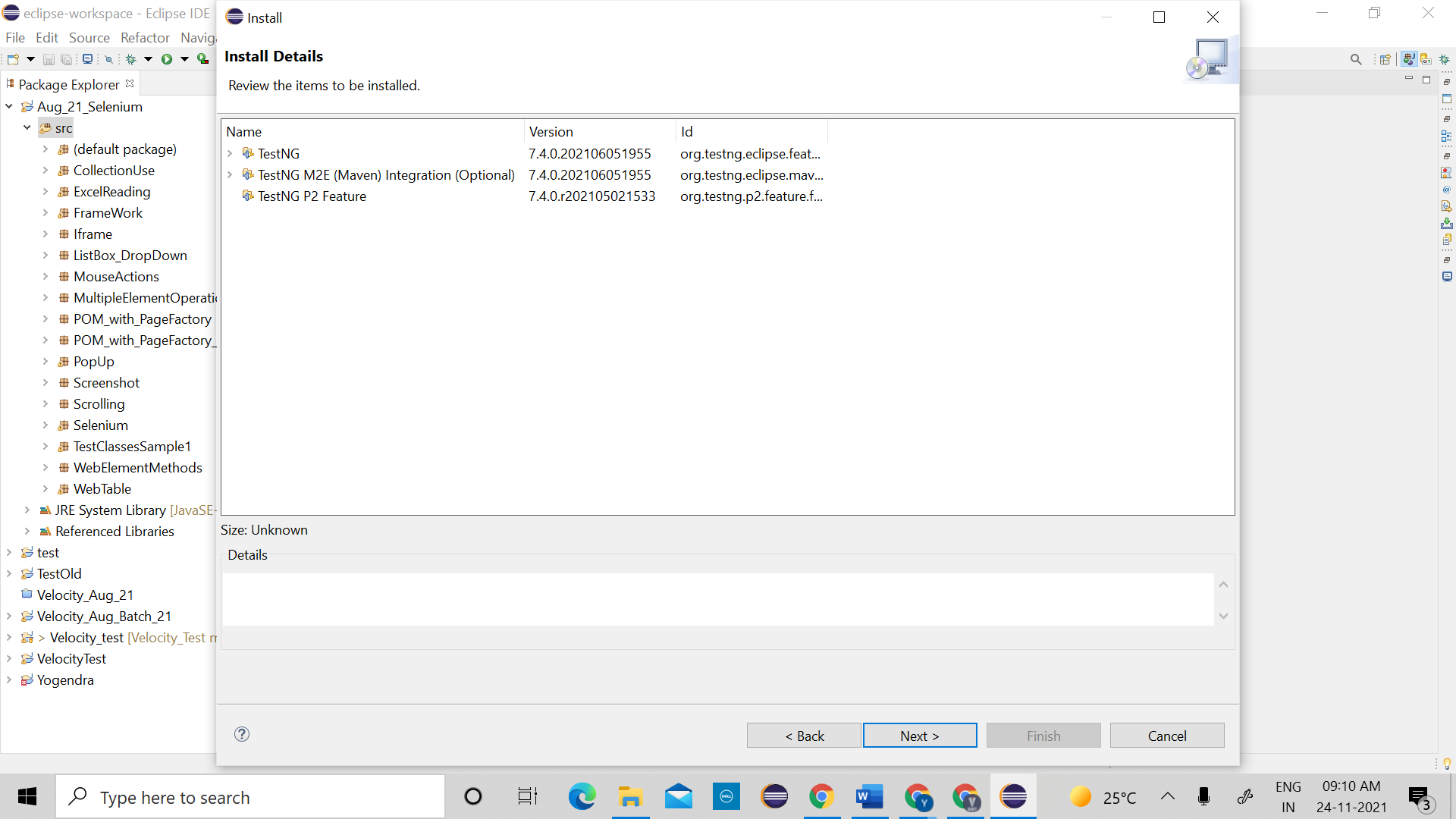
Paste in site place

<https://testng.org/testng-eclipse-update-site/7.4.0/>









To check testNG is installed or not

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

->Example of normal Test Class

============================================================

**package** TestNGStudy;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** NormalClass {

**public** **static** **void** main(String[] args) {

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://kite.zerodha.com/");

}

}

=============================================================

->Example of TestNG Test class

==============================================================

**package** TestNGStudy;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

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

**public** **class** Example1TestNG {

@Test // this for method/ testCase

**public** **void** KiteLogin()

{

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://kite.zerodha.com/");

}

}

=============================================================

1. **Emailable Report:** Report generation is very important when you are doing the Automation Testing as well as for Manual Testing. By looking at the result, you can easily identify how many test cases are passed, failed and skipped. By looking at the report, you will come to know what the status of the project is. Selenium web driver is used for automating the web-application, but it won't generate any reports. The TestNG will generate the default report.

Green🡪 You are matching with your timeline (daily 10 TC)

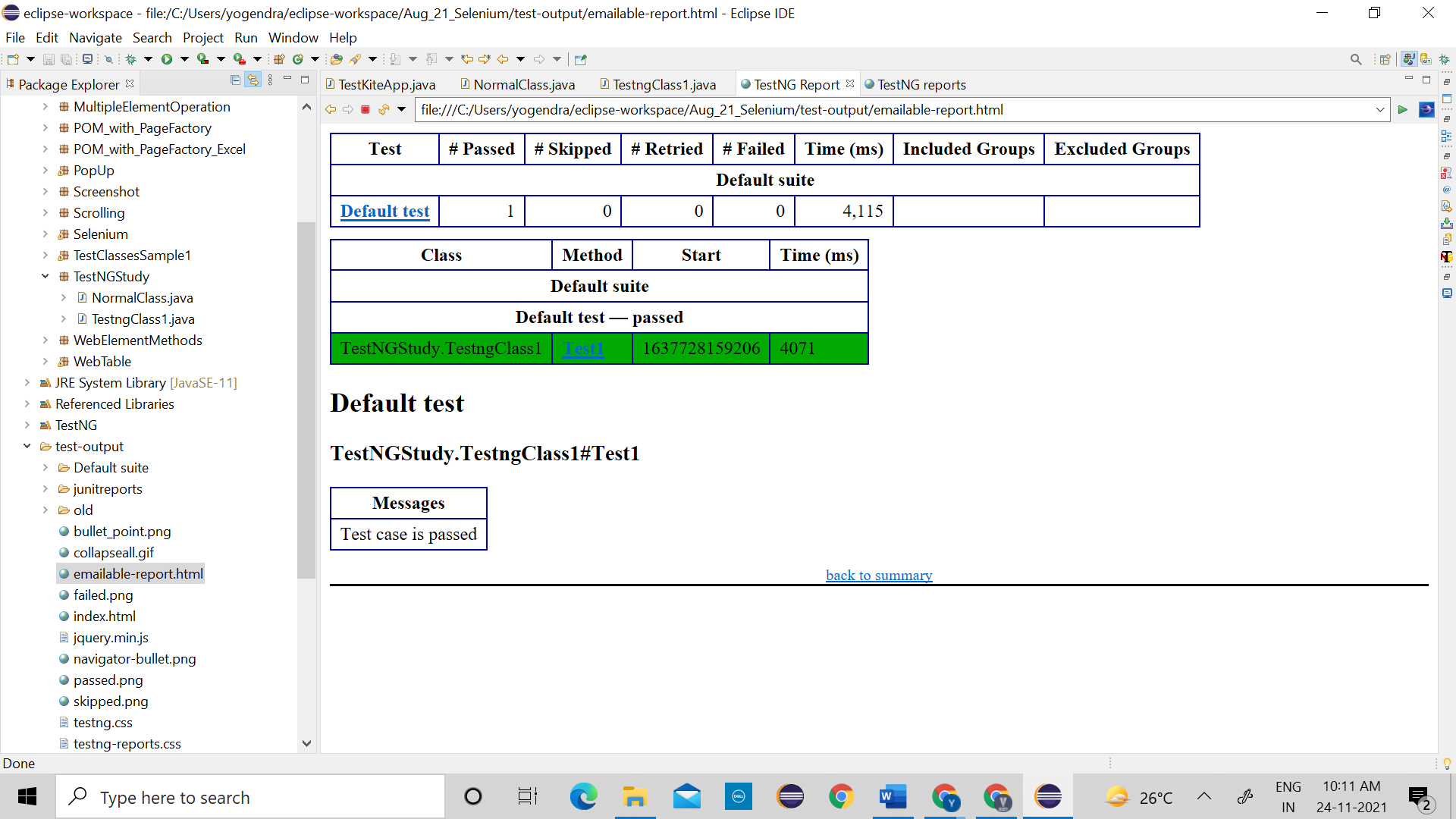
Yellow🡪 You are lagging in your execution (daily 8 TC) 🡪(blocking defect)

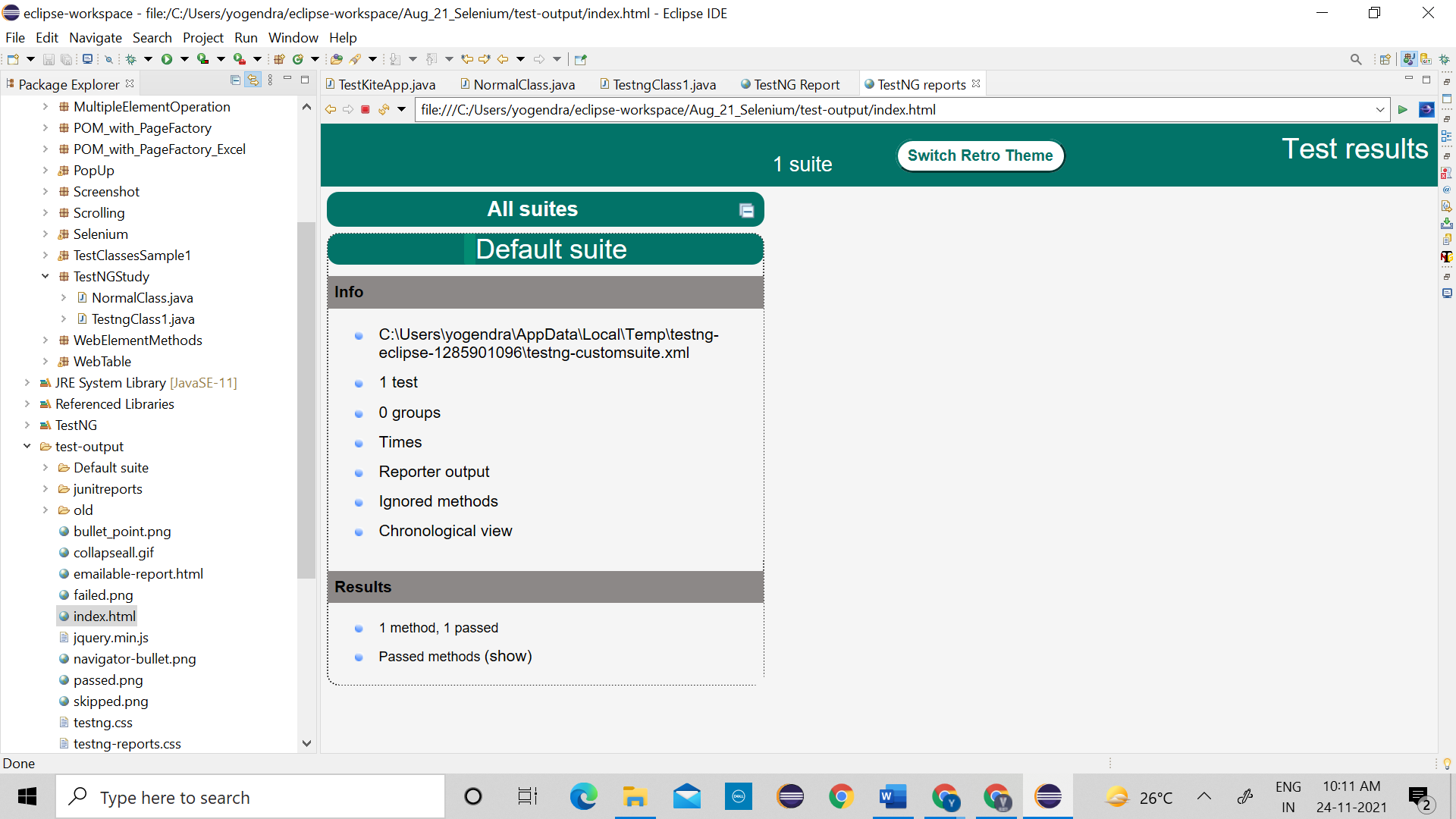
Red🡪 You are far behind your target (No TC executed)

Steps to generate Emailable report🡪

1. Execute Test class and refresh the project.
2. You will get test-output folder.
3. In That folder Right click on the "emailable-report.html" and select the option Open with the web browser or double click on it.

Note:

1. if we use sop() to display text as a output then result will be displayed in console only, not in emailable report.
2. To display text in emailable report we need to use static method log present in Reporter class. eg. Reporter.log("String msg", true)
3. 



======================================================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** Simple2 {

@Test

**public** **void** Display1() {

Reporter.*log*("hi Good morning",**true**);

}

}

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

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

<suite name="Default suite" guice-stage="DEVELOPMENT">

<test thread-count="5" name="Default test" verbose="2">

<classes>

<class name="TestNGStudy.Simple2"/>

</classes>

</test> <!-- Default test -->

</suite> <!-- Default suite -->

**2. TestNG Annotation**

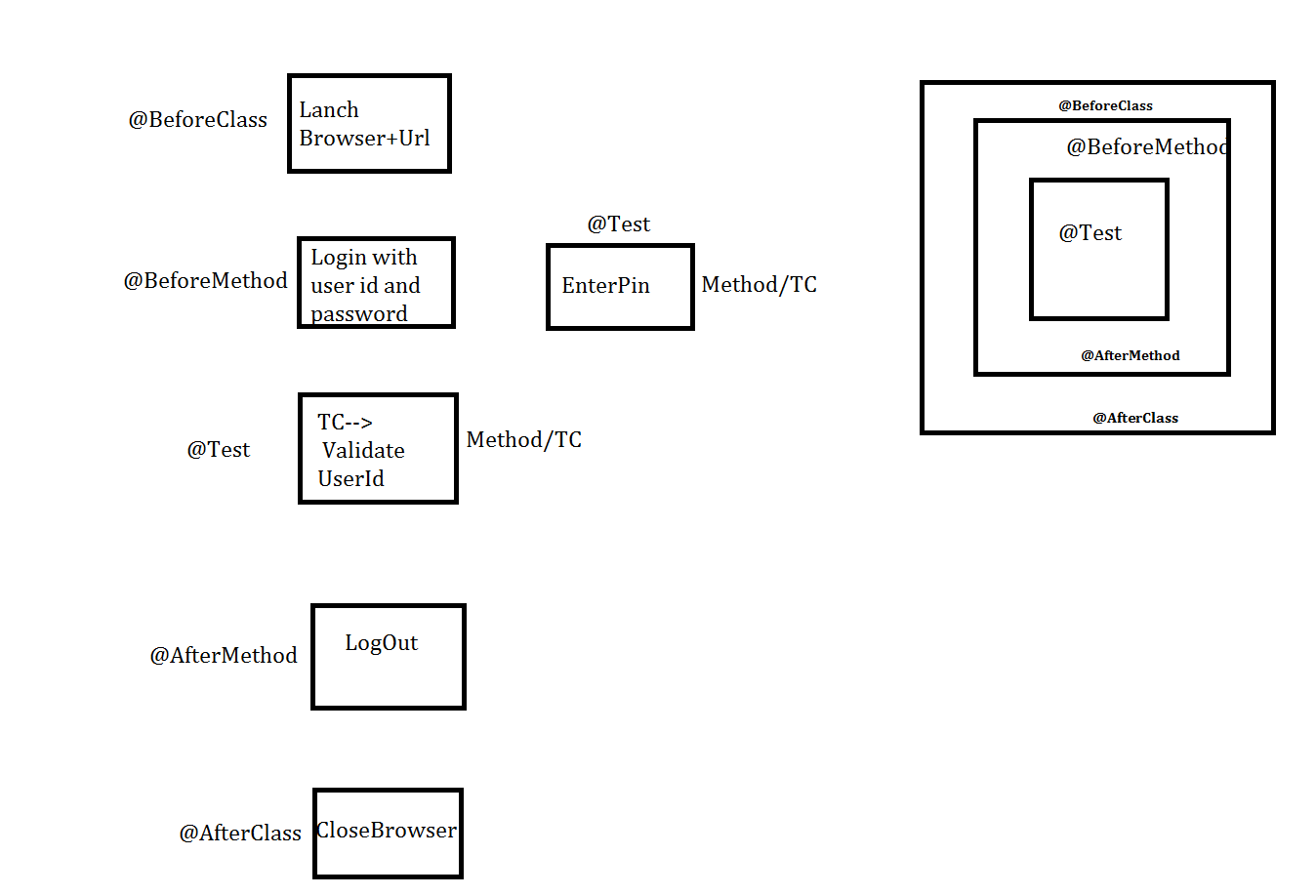
1. @Test

2. @BeforeMethod

3. @AfterMethod

4. @BeforeClass

5. @AfterClass



1. @Test: - Used for execution of test method/TC.
2. @BeforeMethod: - It is used for execution of method before execution of **every test method** with an annotation @Test.
3. @AfterMethod: - It is used for execution of method after execution of **every test method** with an annotation @Test.
4. @BeforeClass: - It is used for execution of method before execution of test class.
5. @AfterClass: - It is used for execution of method after execution of test class.

=============================================================

package TestNGStudy;

import org.testng.annotations.Test;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.Reporter;

import org.testng.annotations.AfterClass;

public class AnnotationStudy {

@Test

public void ValidatePin() {

Reporter.log("Pin Validation done-->@Test used", true);

}

@Test

public void ValidateUserId() {

Reporter.log("UserId Validation done--> @Test used", true);

}

@BeforeMethod

public void EnterUserIdAndPassword() {

Reporter.log("UserIdPasswordEntered done--> @BeforeMethod used", true);

}

@AfterMethod

public void Logout() {

Reporter.log("Logout done--> @AfterMethod used", true);

}

@BeforeClass

public void BrowserLaunch() {

Reporter.log("Launch Browser--> @BeforeClass used", true);

}

@AfterClass

public void CloseBrowser() {

Reporter.log("Closed Browser--> @AfterClass used", true);

}

}

======================================================================

**3. TestNG Keyword:**

1. invocationCount

2. priority

3. enabled

4. TimeOut

5. dependsOnMethods

1. **invocationCount**: Sometimes same test method/TC need to be executed multiple times which can be possible by using TestNG keyword "invocationCount"

eg. invocationCount=5;

======================================================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

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

**public** **class** KeyWordInvocationCount {

@BeforeMethod

**public** **void** before()

{

Reporter.*log*("Before Method Running", **true**);

}

@Test(invocationCount = 5)

**public** **void** TC1() {

Reporter.*log*("TC1 Running", **true**);

}

}

======================================================================

1. **priority**: To change test method/TC execution order we need to use TestNG keyword "priority".

eg. priority=1

**Note: priority can be**

|  |  |  |  |
| --- | --- | --- | --- |
| i. bydefault=0 | ii. +ve integer | iii. -ve integer | iv. Duplicate |

**priority can't be**

|  |  |  |  |
| --- | --- | --- | --- |
| i. Decimals | ii. Variables |  |  |

==================================================================

Without priority🡪

============

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** KeywordPriority {

@Test

**public** **void** C()

{

Reporter.*log*("C is running",**true**);

}

@Test

**public** **void** A()

{

Reporter.*log*("A is running",**true**);

}

@Test

**public** **void** B()

{

Reporter.*log*("B is running",**true**);

}

}

================================================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** KeywordPriority {

@Test (priority = 1)

**public** **void** C()

{

Reporter.*log*("C is running",**true**);

}

@Test (priority = 3)

**public** **void** A()

{

Reporter.*log*("A is running",**true**);

}

@Test (priority = 2)

**public** **void** B()

{

Reporter.*log*("B is running",**true**);

}

}

=================================================================

1. **enabled:** Disabling a test method/TC in TestNG can be achieved by setting the enabled attribute of the @Test annotation to false.

eg. enabled=false

====================================================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** KeywordEnabled {

@Test

**public** **void** TC1() {

Reporter.*log*("Running TC1",**true**);

}

@Test (enabled = **false**)

**public** **void** TC2()

{

Reporter.*log*("Running TC2",**true**);

}

@Test

**public** **void** TC3() {

Reporter.*log*("Running TC3",**true**);

}

}

================================================================

1. **TimeOut**: If test class contains multiple test methods if one of the test method is time consuming to execute then TestNG bydefault **fail** that TC & executes the other TC.

eg. @Test(timeOut=8000)

Running TC1—10ms

Running TC2-11-90ms🡪40ms (50ms)

Running TC3-91ms🡪41ms

==========================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** KeywordTimeOut {

@Test

**public** **void** TC1()

{

Reporter.*log*("TC1 is running", **true**);

}

@Test (timeOut = 2000)

**public** **void** TC2() **throws** InterruptedException

{

Reporter.*log*("TC2 is running", **true**);

Thread.*sleep*(3000);

}

@Test

**public** **void** TC3()

{

Reporter.*log*("TC3 is running", **true**);

}

}

=========================================

5. **dependsOnMethods**: If 1 TC execution depends on multiple TC then we need to use "dependsOnMethods" TestNG keyword.

eg. dependsOnMethods= {"TC name"}

======================================================================**package** TestNGStudy;

**import** org.testng.Assert;

**import** org.testng.Reporter;

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

**public** **class** KeywordDependsOnMethod1 {

@Test(priority = 1)

**public** **void** Login()

{

Reporter.*log*("Login sucess",**true**);

Assert.*fail*();

}

@Test(priority = 2)

**public** **void** VerifyPin()

{

Reporter.*log*("Pin Verified",**true**);

}

@Test (dependsOnMethods = "Login")

**public** **void** Logout()

{

Reporter.*log*("Logout success", **true**);

}

}

**Test-Suite:**

* It is xml file which contains all the test classes name which need to be executed.
* It is used to execute all/multiple Test classes.

1000🡪50run (950🡪disable)

<suite name="Suite name">

<test name="Test name">

<classes>

<class name="packageName.className"/>

</classes>

</test>

</suite>

200🡪Class1

100🡪Class2 10🡪inlude1-10, exculude90

Exclude from single class🡪

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

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

<suite name=*"Suite"*>

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

<classes>

<class name=*"TestNGStudy.ExcludeMethodsFromSuite"*>

<methods>

<exclude name=*"TC4"*>

</exclude>

<exclude name=*"TC3"*>

</exclude>

</methods>

</class>

</classes>

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

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

=====================================================================

Include from single class🡪

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

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

<suite name=*"Suite"*>

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

<classes>

<class name=*"TestNGStudy.ExcludeMethodsFromSuite"*>

<methods>

<include name=*"TC1"*/>

<include name=*"TC2"*/>

<include name=*"TC3"*/>

</methods>

</class>

</classes>

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

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

======================================================================

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

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

<suite name=*"Suite"*>

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

<classes>

<class name=*"TestNGStudy.IncludeMethodsFromSuite"*>

<methods>

<exclude name=*"TC7"*/>

<exclude name=*"TC8"*/>

<exclude name=*"TC9"*/>

</methods>

</class>

</classes>

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

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

Include and Exclude from two diff classes

======================================================================

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

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

<suite name=*"Suite"*>

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

<classes>

<class name=*"TestNGStudy.IncludeMethodsFromSuite"*>

<methods>

<exclude name=*"TC7"*/>

<exclude name=*"TC8"*/>

<exclude name=*"TC9"*/>

</methods>

</class>

<class name=*"TestNGStudy.ExcludeMethodsFromSuite"*>

<methods>

<include name=*"TC1"*/>

<include name=*"TC2"*/>

</methods>

</class>

</classes>

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

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

==================================================================

Class1 🡪 Total 50🡪20 Regression Tc’s, 30 Sanity TC

Class2🡪 Total 100🡪50 Regression TC’s, 50 Sanity TC

Regression🡪70

Sanity🡪80

Grouping in **testng.xml**

**================================================================**

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** groupseg1 {

@Test(groups = "regression")

**public** **void** TC1()

{

Reporter.*log*("TC1 regression Running",**true**);

}

@Test(groups = "sanity")

**public** **void** TC2()

{

Reporter.*log*("TC2 sanity Running",**true**);

}

@Test(groups = "regression")

**public** **void** TC3()

{

Reporter.*log*("TC3 regression Running",**true**);

}

@Test(groups = "sanity")

**public** **void** TC4()

{

Reporter.*log*("TC4 sanity Running",**true**);

}

@Test

**public** **void** TC5()

{

Reporter.*log*("TC5 Running",**true**);

}

}

=====================================================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**public** **class** groupeg2 {

@Test (groups = "regression")

**public** **void** TC6()

{

Reporter.*log*("TC6 regression Running",**true**);

}

@Test(groups = "sanity")

**public** **void** TC7()

{

Reporter.*log*("TC7 sanity Running",**true**);

}

@Test(groups = "regression")

**public** **void** TC8()

{

Reporter.*log*("TC8 regression Running",**true**);

}

@Test (groups = "sanity")

**public** **void** TC9()

{

Reporter.*log*("TC9 sanity Running",**true**);

}

@Test

**public** **void** TC10()

{

Reporter.*log*("TC10 Running",**true**);

}

}

======================================================================

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

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

<suite name=*"Suite"*>

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

<groups>

<run>

<exclude name=*"sanity"*/>

</run>

</groups>

<classes>

<class name=*"TestNGStudy.groupseg1"*/>

<class name=*"TestNGStudy.groupeg2"*/>

</classes>

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

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

======================================================================

* **Failed.xml:** While executing the automation scripts, test cases may fail for several reasons. To optimize our next runs, we need to re-run only failed test cases.

1000🡪TC total

Run🡪900 Pass, 100 fail

Steps to execute failed.xml file🡪

1. Create testng. xml file under project folder.

2. execute testng.xml file

3. Refresh the project

4. In the test-output folder 🡪 testng-failed. xml file will be created.

5. execute "testng-failed. xml"

In this way we can execute fail testcases in TestNG class.

Reasons for fail TC

1. envirnment issue

2. script error

3. bug/defect

======================================================================

**package** TestNGStudy;

**import** org.testng.Assert;

**import** org.testng.Reporter;

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

**public** **class** FailTCExecution {

@Test(timeOut = 100)

**public** **void** TC6() **throws** InterruptedException

{

Reporter.*log*("TC6 Running",**true**);

Thread.*sleep*(200);

}

@Test

**public** **void** TC7()

{

Reporter.*log*("TC7 Running",**true**);

}

@Test

**public** **void** TC8()

{

Reporter.*log*("TC8 Running",**true**);

Assert.*fail*();

}

@Test

**public** **void** TC9()

{

Reporter.*log*("TC9 Running",**true**);

}

@Test

**public** **void** TC10()

{

Reporter.*log*("TC10 Running",**true**);

}

}

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

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

<suite name=*"Failed suite [Suite]"* guice-stage=*"DEVELOPMENT"*>

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

<classes>

<class name=*"TestNGStudy.FailTCExecution"*>

<methods>

<include name=*"TC6"*/>

<include name=*"TC8"*/>

</methods>

</class> <!-- TestNGStudy.FailTCExecution -->

</classes>

</test> <!-- Test(failed) -->

</suite> <!-- Failed suite [Suite] -->

===============================================

Parallel execution🡪

==============================================

**package** TestNGStudy;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.Reporter;

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

**public** **class** Parallel1 {

@Test

**public** **void** KiteLogin() **throws** InterruptedException

{

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://kite.zerodha.com/");

Reporter.*log*("KiteLaunched",**true**);

Thread.*sleep*(2000);

driver.close();

}

}

==============================================

**package** TestNGStudy;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.Reporter;

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

**public** **class** Parallel2 {

@Test

**public** **void** FacebookLaunch() **throws** InterruptedException

{

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://en-gb.facebook.com/");

Reporter.*log*("facebook Launche",**true**);

Thread.*sleep*(2000);

driver.close();

}

}

===============================================

**package** TestNGStudy;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.Reporter;

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

**public** **class** Parallel3 {

@Test

**public** **void** VCTCLaunch() **throws** InterruptedException

{

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://vctcpune.com/");

Reporter.*log*("VCTC Launche",**true**);

Thread.*sleep*(2000);

driver.close();

}

}

===============================================

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

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

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

<test name=*"ParallelTest1"*>

<classes>

<class name=*"TestNGStudy.Parallel1"*/>

</classes>

</test> <!-- ParallelTest1 -->

<test name=*"ParallelTest2"*>

<classes>

<class name=*"TestNGStudy.Parallel2"*/>

</classes>

</test> <!-- ParallelTest2 -->

<test name=*"ParallelTest3"*>

<classes>

<class name=*"TestNGStudy.Parallel3"*/>

</classes>

</test> <!-- ParallelTest3 -->

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

**# Verification using TestNG// Assert**

Example to verify checkbox is selected or not.

public class Sample {

public static void main(String[] args) {

System.setProperty("webdriver.chrome.driver",

"===path===\\chromedriver.exe");

WebDriver driver= new ChromeDriver();

WebElement rv = driver.findElements("====Xpath===);"

if(rv.isSelected())

{

System.out.println("checkbox is selected");

}

else

{

System.out.println("checkbox is not selected");

}

}

}

If above verification process is used to verify expected result of a test case, length of Test

script will take more time for execution.

To reduce length of test script we need to use **Assert class** for verification which contains

**static** methods.

Important **static** methods present in **Assert** class **(Hard Assert)**

All static method should be imported from org.TestNG

**1) assertEquals()**

**2) assertNotEquals()**

**3) assertTrue()**

**4) assertFalse()**

**5) assertNull()**

**6) assertNotNull()**

**7) fail()**

1] assertEquals()

Used to verify expected and actual results. If both results are same then output is pass

otherwise fail.

Ex.

@Test

public void TC1()

{

String ExpectedResult="VCTCPune";

String ActualResult="VCTC";

Assert.assertEquals(ActualResult, ExpectedResult,"Result is not matching");

Reporter.log("TC1 is running", true);

}

========|==========|=======

2] assertNotEquals()

Used to verify expected and actual results. If both results are not same then output is

pass otherwise fail.

Ex.

@Test

public void TC2()

{

String ExpectedResult="VCTCPune";

String ActualResult="VCTC";

Assert.assertNotEquals(ActualResult, ExpectedResult, "Result is matching");

Reporter.log("TC2 is running", true);

}

========|========|=======

3] assertTrue()

This method is use to verify conditions are true or false. If condition is true output is

pass otherwise fail.

Ex.

@Test(enabled = false)

public void TC3()

{

//boolean output= Title.isSelected();

//boolean output= Title.isEnabled();

//boolean output= Title.isMultiple();

//boolean output= Title.isDisplayed();

boolean Result=false;

Assert.assertTrue(Result, "Result is false");

Reporter.log("TC3 is running", true);

}

========|=======|=======

4] assertFalse()

Use to verify conditions are true or false, if condition is true output is fail and if

condition is false then output is pass.

Ex.

@Test(enabled = false)

public void TC4()

{

//boolean output= Title.isDisplayed();

boolean Result=true;

Assert.assertFalse(Result,"Result is true");

Reporter.log("TC4 is running", true);

}

========|=======|=======

5] assertNull()

This method is use to verify components or text fields empty or not if it is empty

output is pass otherwise fail.

Ex.

@Test(enabled = false)

public void TC5()

{

String str=null;

Assert.assertNull(str,"Value is not null");

Reporter.log("TC5 is running", true);

}

========|=======|=======

6] assertNotNull()

Use to verify components or text fields empty or not, if it is not empty output is pass

otherwise fail.

Ex.

@Test (enabled = false)

public void TC6()

{

String str="ABC";

Assert.assertNotNull(str,"Value is null");

Reporter.log("TC6 is running", true);

}

========|=======|========

7] fail()

This method is used to intentionally failed test method.

Ex.

@Test

public void TC7()

{

Reporter.log("Assert fail check", true);

Assert.fail();

Reporter.log("TC7 is running", true);

}

In a test class if one of the test method is fail then TestNG will stop execution of failed test

methods & after test methods execution will be continued

Ex.

@test

public void test1()

{

Reporter.log("Hello", true);

Assert.fail(); //output- failed

}

@test

public void test2()

{

Reporter.log("Hello", true); // output- Passed

}

In a test class if one of the test method is failed and that test method execution required for

other test method execution then other test methods will be skipped.

Ex.

@test

public void test1()

{

Reporter.log("Hello", true);

Assert.fail(); //output- failed

}

@test(dependsOnMethods= "test1")

public void test2()

{

Reporter.log("Hello", true); // output- skipped

}

=========|========|=======

# Disadvantages of Assert class**(HardAssert):**

If a test class is containing multiple test methods and one of the test method contains

multiple verification point, And while executing, if one verification is failed then rest of the

verification will not be verified and TestNG will execute next method by failing verification

field method.

Ex.

public class SampleAssert{

@test

public void sample()

{

String str1 = "Hello";

Assert.assertEquals(str1, "Hi");

String str1 = "Hi";

Assert.assertEquals(str2, "Hi");

Reporter.log("Hi", true);

}

}

**# Soft Assert**

-To overcome assert class drowback we need to use soft assert.

-It is a class which contains **non-static** methods use to do verification.

-Soft assert will do verification if any executes the rest of verification in a test method

Ex.

package TestNGStudy;

import org.testng.Reporter;

import org.testng.annotations.Test;

import org.testng.asserts.SoftAssert;

public class SoftAssertStudy {

SoftAssert soft= new SoftAssert();

@Test

public void TC1()

{

String ExpectedResult="VCTCPune";

String ActualResult="VCTC";

soft.assertEquals(ActualResult, ExpectedResult, "Value is not

matching");

Reporter.log("TC1 softAssert is running",true);

soft.assertNotNull(ActualResult);

Reporter.log("running TC",true);

soft.assertAll(); //If we don’t write assertAll then it will not notify

}

}

======================================================================

Hard assert example🡪

======================================================================

**package** TestNGStudy;

**import** org.testng.Assert;

**import** org.testng.Reporter;

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

**public** **class** HardAssertStudy {

@Test(enabled = **false**)

**public** **void** TC1()

{

String ExpectedResult="VCTCPune";

String ActualResult="VCTC";

Assert.*assertEquals*(ActualResult, ExpectedResult,"Result is not matching");

Reporter.*log*("TC1 is running", **true**);

}

@Test(enabled = **false**)

**public** **void** TC2()

{

String ExpectedResult="VCTCPune";

String ActualResult="VCTC";

Assert.*assertNotEquals*(ActualResult, ExpectedResult, "Result is matching");

Reporter.*log*("TC2 is running", **true**);

}

@Test(enabled = **false**)

**public** **void** TC3()

{

//boolean output= Title.isDisplayed();

**boolean** Result=**false**;

Assert.*assertTrue*(Result, "Result is false");

Reporter.*log*("TC3 is running", **true**);

}

@Test(enabled = **false**)

**public** **void** TC4()

{

//boolean output= Title.isDisplayed();

**boolean** Result=**true**;

Assert.*assertFalse*(Result,"Result is true");

Reporter.*log*("TC4 is running", **true**);

}

@Test(enabled = **false**)

**public** **void** TC5()

{

String str=**null**;

Assert.*assertNull*(str,"Value is not null");

Reporter.*log*("TC5 is running", **true**);

}

@Test (enabled = **false**)

**public** **void** TC6()

{

String str="ABC";

Assert.*assertNotNull*(str,"Value is null");

Reporter.*log*("TC6 is running", **true**);

}

@Test

**public** **void** TC7()

{

Reporter.*log*("Assert fail check", **true**);

Assert.*fail*();

Reporter.*log*("TC7 is running", **true**);

}

}

======================================================================

SoftAssert Example🡪

======================================================================

**package** TestNGStudy;

**import** org.testng.Reporter;

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

**import** org.testng.asserts.SoftAssert;

**public** **class** SoftAssertStudy {

SoftAssert soft= **new** SoftAssert();

@Test

**public** **void** TC1()

{

String ExpectedResult="VCTCPune";

String ActualResult="VCTC";

soft.assertEquals(ActualResult, ExpectedResult, "Value is not matching");

Reporter.*log*("TC1 softAssert is running",**true**);

soft.assertNotNull(ActualResult);

Reporter.*log*("running TC",**true**);

soft.assertAll();

}

}

====================================================================

**Cross Browser Testing Single parameter/Serial Testing/ Parallel Testing🡪 Change in Testng.xml only code will be same for all(observe xmls only)**

=====================================================================

**package** TestNGStudy;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.firefox.FirefoxDriver;

**import** org.testng.annotations.Parameters;

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

**public** **class** CrossBrowserTesting {

@Parameters("browserName")

@Test

**public** **void** BrowserTest(String browserName)

{

WebDriver driver=**null**;

**if** (browserName.equals("chrome"))

{

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

driver= **new** ChromeDriver();

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

}

**else** **if** (browserName.equals("firefox"))

{

System.*setProperty*("webdriver.gecko.driver", "F:\\Velocity\\May-2021 Class\\test2\\geckodriver.exe");

driver= **new** FirefoxDriver();

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

}

driver.get("https://kite.zerodha.com/");

driver.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***);

//provide userid

driver.findElement(By.*xpath*("//input[@id='userid']")).sendKeys("DV1510");

//provide password

driver.findElement(By.*xpath*("//input[@id='password']")).sendKeys("Year@123");

//click on login button

driver.findElement(By.*xpath*("//button[@class='button-orange wide']")).click();

//provide pin

driver.findElement(By.*xpath*("//input[@id='pin']")).sendKeys("959594");

//click on continue

driver.findElement(By.*xpath*("//button[@class='button-orange wide']")).click();

//find userid

String ActualUserid = driver.findElement(By.*xpath*("//span[@class='user-id']")).getText();

String ExpecxtedUserid="DV1510";

**if**(ActualUserid.equals(ExpecxtedUserid))

{

System.***out***.println("TC passed user id is matching");

}

**else** {

System.***out***.println("TC failed user id is not-matching");

}

}

}

======================================================================

This xml for single browser will open according to parameter value (chrome/firefox)🡪

====================================================================

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

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

<suite name=*"Suite"*>

<test name=*"ChromeTest"*>

<parameter name=*"browserName"* value**=*"chrome"*/>**

<classes>

<class name=*"TestNGStudy.CrossBrowserTesting"*/>

</classes>

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

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

=============================================

This xml serial multiple browser will open according to parameter value (chrome/firefox) we added **extra Test In this**

==========================================

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

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

<suite name=*"Suite"*> **<!-- for serial running-->**

<test name=*"ChromeTest"*>

<parameter name=*"browserName"* value=*"chrome"*/>

<classes>

<class name=*"TestNGStudy.CrossBrowserTesting"*/>

</classes>

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

<test name=*"FirefoxTest"*>

<parameter name=*"browserName"* value=*"firefox"*/>

<classes>

<class name=*"TestNGStudy.CrossBrowserTesting"*/>

</classes>

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

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

==================================================================

This This xml parallel multiple browser will open according to parameter value (chrome/firefox) we added **extra Test In this and added “parallel=”tests” in suite** 🡪

=====================================================================

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

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

**<suite name=*"Suite"* parallel=*"tests"*> <!-- for parrel running-->**

<test name=*"ChromeTest"*>

<parameter name=*"browserName"* value=*"chrome"*/>

<classes>

<class name=*"TestNGStudy.CrossBrowserTesting"*/>

</classes>

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

<test name=*"FirefoxTest"*>

<parameter name=*"browserName"* value=*"firefox"*/>

<classes>

<class name=*"TestNGStudy.CrossBrowserTesting"*/>

</classes>

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

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

======================================================================

POM class1 for login to Kite🡪

package TestNGPOM;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class KiteLoginPapge1 {

//

@FindBy(id = "userid") private WebElement UserID;

@FindBy(xpath="//input[@id='password']") private WebElement Password;

@FindBy(xpath = "//button[@class='button-orange wide']") private WebElement LoginButton;

public KiteLoginPapge1(WebDriver driver) {

PageFactory.initElements(driver, this);

}

public void SendUserId(String UserName)

{

UserID.sendKeys(UserName);

}

public void SendPassword(String pwd)

{

Password.sendKeys(pwd);

}

public void ClickOnLoginButton()

{

LoginButton.click();

}

}

=================================================================

POM class2 for Pin page🡪

**package** TestNGPOM;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.support.FindBy;

**import** org.openqa.selenium.support.PageFactory;

**public** **class** KitePinPage2 {

@FindBy(xpath = "//input[@id='pin']") **private** WebElement Pin;

@FindBy(xpath = "//button[@class='button-orange wide']") **private** WebElement ContinueButton;

**public** KitePinPage2(WebDriver driver)

{

PageFactory.*initElements*(driver, **this**);

}

**public** **void** EnterPin(String PinValue)

{

Pin.sendKeys(PinValue);

}

**public** **void** ClickOnContinue()

{

ContinueButton.click();

}

}

=========================================================

POM class3 for Home page🡪

**package** TestNGPOM;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.support.FindBy;

**import** org.openqa.selenium.support.PageFactory;

**public** **class** KiteHomePage3 {

@FindBy(xpath = "//span[@class='user-id']") **private** WebElement Userid;

**public** KiteHomePage3(WebDriver driver) {

PageFactory.*initElements*(driver, **this**);

}

**public** String UserIdValidation()

{

String ActualUserid = Userid.getText();

**return** ActualUserid;

}

}

=========================================================

Test Class Using TestNG

=======================================================

**package** TestNGStudy;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.util.concurrent.TimeUnit;

**import** org.apache.poi.EncryptedDocumentException;

**import** org.apache.poi.ss.usermodel.Sheet;

**import** org.apache.poi.ss.usermodel.WorkbookFactory;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.chrome.ChromeOptions;

**import** org.testng.Assert;

**import** org.testng.Reporter;

**import** org.testng.annotations.AfterClass;

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

**import** org.testng.annotations.BeforeClass;

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

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

**import** TestNGPOM.KiteHomePage3;

**import** TestNGPOM.KiteLoginPapge1;

**import** TestNGPOM.KitePinPage2;

**public** **class** KiteTestUsingPOM {

KiteLoginPapge1 page1;

KitePinPage2 page2;

KiteHomePage3 page3;

WebDriver driver;

Sheet Mysheet;

@BeforeClass

**public** **void** LaunchBrowser() **throws** EncryptedDocumentException, IOException

{

Reporter.*log*("Browser Launched", **true**);

FileInputStream Myfile= **new** FileInputStream("C:\\Users\\user\\Desktop\\mytesting.xlsx");

Mysheet = WorkbookFactory.*create*(Myfile).getSheet("Sheet4");

ChromeOptions options=**new** ChromeOptions();

options.addArguments("--disable-notifications");

//options.addArguments("headless");

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://kite.zerodha.com/");

driver.manage().timeouts().implicitlyWait(100, TimeUnit.***SECONDS***);

page1= **new** KiteLoginPapge1(driver);

page2= **new** KitePinPage2(driver);

page3= **new** KiteHomePage3(driver);

}

@BeforeMethod

**public** **void** LoginKiteApplication()

{

Reporter.*log*("LoginKiteApplication", **true**);

String Userid1 = Mysheet.getRow(0).getCell(0).getStringCellValue();

String Password1 = Mysheet.getRow(0).getCell(1).getStringCellValue();

**double** PinValue = Mysheet.getRow(0).getCell(2).getNumericCellValue();

String PinValues=Double.*toString*(PinValue);

page1.SendUserId(Userid1);

page1.SendPassword(Password1);

page1.ClickOnLoginButton();

page2.EnterPin(PinValues);

page2.ClickOnContinue();

}

@Test

**public** **void** ValidateUserID()

{

Reporter.*log*("ValidateUserID running", **true**);

String ExPectedUserid = Mysheet.getRow(0).getCell(0).getStringCellValue();

String ActualUserID=page3.UserIdValidation();

Assert.*assertEquals*(ActualUserID, ExPectedUserid,"UseriD Not matching TC Failed");

Reporter.*log*("UserID Matching TC passed", **true**);

}

@AfterMethod

**public** **void** LogoutKiteApplication()

{

Reporter.*log*("Application logged Out", **true**);

}

@AfterClass

**public** **void** CloseBrowser()

{

Reporter.*log*("Closing Browser", **true**);

driver.close();

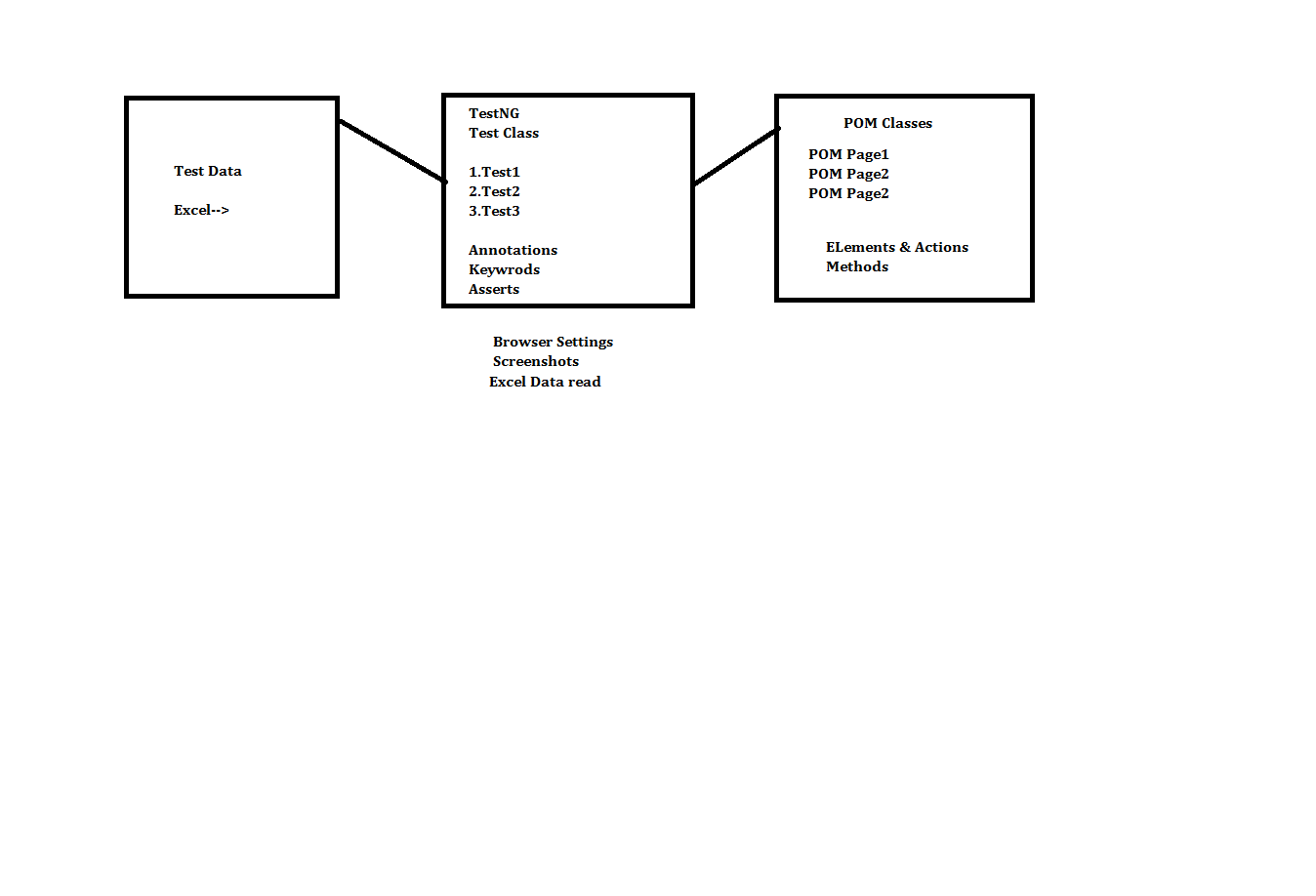
}

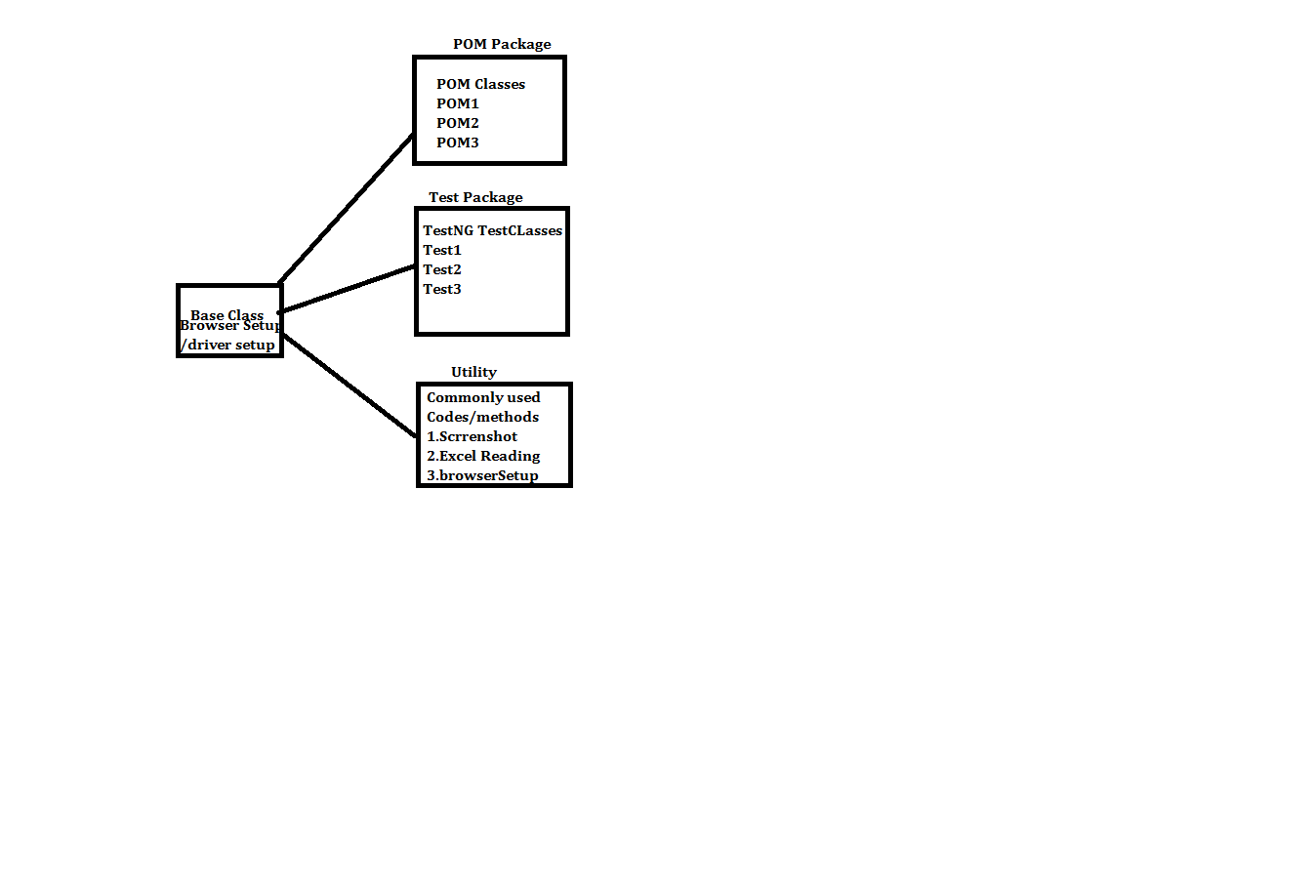
}

**TestNG advantages**

* It has different assertions that help in checking the expected and actual results.
* It allows to assign priority to test methods
* It allows to define dependency of one test method over other method
* It provide Detailed (HTML) reports/ Emailable report
* It allows grouping of test methods into test class
* TestNG provides parallel execution of test methods
* TestNG provides multibrowser/CT testing of test methods

|  |  |  |
| --- | --- | --- |
|  | Junit | TestNG |
| Parallel execution | JUnit does not support to run parallel tests. | TestNG can run parallel tests. |
| Supports Annotation | It does not support advanced annotation. | It supports advanced annotation. |
| Dependency tests | The dependency tests are missing in JUnit. | Dependency tests are present in TestNG. |
| Grouping tests | Grouping tests together is not possible in JUnit. | Tests can be grouped together and run parallel. |





==============================================================

Base Class🡪

==========================================================

**package** KiteTestBaseUtility;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.chrome.ChromeOptions;

**public** **class** Base {

WebDriver driver;

**public** **void** browserInitialize()

{

ChromeOptions options=**new** ChromeOptions();

options.addArguments("--disable-notifications");

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get("https://kite.zerodha.com/");

driver.manage().timeouts().implicitlyWait(100, TimeUnit.***SECONDS***);

}

}

===========================================================

Utility Class🡪

============================================================

package KiteTestBaseUtility;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import org.apache.poi.EncryptedDocumentException;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.WorkbookFactory;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.io.FileHandler;

public class Utility {

public static String getDatafromExcel(int rowIndex, int columnIndex) throws EncryptedDocumentException, IOException

{

FileInputStream Myfile= new FileInputStream("C:\\Users\\user\\Desktop\\mytesting.xlsx");

Sheet Mysheet = WorkbookFactory.create(Myfile).getSheet("Sheet4");

String value = Mysheet.getRow(rowIndex).getCell(columnIndex).getStringCellValue();

return value;

}

public static void captureScreenshot(WebDriver driver, int TCID) throws IOException

{

File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

File dest= new File("F:\\Velocity\\May-2021 Class\\Java\\Screenshot eg\\TestCaseID"+TCID+".png");

FileHandler.copy(src, dest);

}

}

=============================================================

POM1🡪

package TestNGPOM;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class KiteLoginPapge1 {

@FindBy(id = "userid") private WebElement UserID;

@FindBy(xpath="//input[@id='password']") private WebElement Password;

@FindBy(xpath = "//button[@class='button-orange wide']") private WebElement LoginButton;

@FindBy(xpath = "//span[contains(text(),'Password should be')]") private WebElement PasswordErrorMsg;

@FindBy(xpath = "//span[contains(text(),'User ID should')]") private WebElement UserIdErrorMsg;

public KiteLoginPapge1(WebDriver driver) {

PageFactory.initElements(driver, this);

}

public void SendUserId(String UserName)

{

UserID.sendKeys(UserName);

}

public void SendPassword(String pwd)

{

Password.sendKeys(pwd);

}

public void ClickOnLoginButton()

{

LoginButton.click();

}

public String getUserIdErrorMsg()

{

String ActualUserIdErrorMsg=UserIdErrorMsg.getText();

return ActualUserIdErrorMsg;

}

public String getPasswordErrorMsg()

{

String ActualPasswordErrorMsg = PasswordErrorMsg.getText();

return ActualPasswordErrorMsg;

}

}

===============================================================

POM2🡪

package TestNGPOM;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class KitePinPage2 {

@FindBy(xpath = "//input[@id='pin']") private WebElement Pin;

@FindBy(xpath = "//button[@class='button-orange wide']") private WebElement ContinueButton;

public KitePinPage2(WebDriver driver)

{

PageFactory.initElements(driver, this);

}

public void EnterPin(String PinValue)

{

Pin.sendKeys(PinValue);

}

public void ClickOnContinue()

{

ContinueButton.click();

}

}

==================================================================

POM3🡪

package TestNGPOM;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class KiteHomePage3 {

@FindBy(xpath = "//span[@class='user-id']") private WebElement Userid;

@FindBy(xpath="//a[contains(text(),'Logout')]") private WebElement logout;

public KiteHomePage3(WebDriver driver) {

PageFactory.initElements(driver, this);

}

public String UserIdValidation()

{

String ActualUserid = Userid.getText();

return ActualUserid;

}

public void clickOnUserid()

{

Userid.click();

}

public void ClickOnLogout()

{

logout.click();

}

}

======================================================================

TestClass🡪

**package** KiteTestBaseUtility;

**import** java.io.IOException;

**import** org.apache.poi.EncryptedDocumentException;

**import** org.testng.Assert;

**import** org.testng.Reporter;

**import** org.testng.annotations.AfterClass;

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

**import** org.testng.annotations.BeforeClass;

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

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

**import** org.testng.asserts.SoftAssert;

**import** TestNGPOM.KiteHomePage3;

**import** TestNGPOM.KiteLoginPapge1;

**import** TestNGPOM.KitePinPage2;

**import** bsh.util.Util;

**public** **class** TestLoginLogout **extends** Base {

KiteLoginPapge1 page1;

KitePinPage2 page2;

KiteHomePage3 page3;

@BeforeClass

**public** **void** LaunchBrowser()

{

Reporter.*log*("Launching Browser", **true**);

browserInitialize();

page1= **new** KiteLoginPapge1(driver);

page2= **new** KitePinPage2(driver);

page3= **new** KiteHomePage3(driver);

}

@BeforeMethod

**public** **void** LoginApplication() **throws** EncryptedDocumentException, IOException

{

Reporter.*log*("Application Logged in", **true**);

page1.SendUserId(Utility.*getDatafromExcel*(0, 0));

page1.SendPassword(Utility.*getDatafromExcel*(0, 1));

page1.ClickOnLoginButton();

page2.EnterPin(Utility.*getDatafromExcel*(0, 2));

page2.ClickOnContinue();

}

@Test

**public** **void** ValidateUserID() **throws** EncryptedDocumentException, IOException

{

**int** TCID=555;

Reporter.*log*("Running "+TCID, **true**);

String ActualUserID = page3.UserIdValidation();

String ExpectedUserID = Utility.*getDatafromExcel*(0, 3);

Assert.*assertEquals*(ActualUserID, ExpectedUserID,"User ID not matching TC failed");

Reporter.*log*("User ID matching TC Passed ", **true**);

Utility.*captureScreenshot*(driver, TCID);

}

@AfterMethod

**public** **void** LogOutFormApplication() **throws** InterruptedException

{

Thread.*sleep*(2000);

Reporter.*log*("Looging out ", **true**);

page3.clickOnUserid();

page3.ClickOnLogout();

}

@AfterClass

**public** **void** CloseBroswer() **throws** InterruptedException

{

Thread.*sleep*(2000);

Reporter.*log*("Closing Browser ", **true**);

driver.close();

}

}

======================================================================

Property file selenium

Step 1) Creating a properties file in eclipse

1. Right-click on the main project folder and Select New-> Other->select General -> File and click on 'Next' button->

Provide a valid file name with the extension '.properties' on the new file resource window and click on 'Finish' button.

Step 2) Storing data into properties file🡪Data is stored in properties file in the form of key-value pairs, with the key being unique across the file.

Open file in Eclipse and store some data

eg- URL= <https://kite.zerodha.com/>

UN=”DV1510” PWD=”Year@123”

Step 3) Reading data from properties file

Properties obj = new Properties();

FileInputStream objfile = new FileInputStream(System.getProperty("user.dir")+"\\credentials.properties");

obj.load(objfile);

String value= obj.getProperty("URL");

===================================================================

Property file🡪

==================================================================

URL=https://kite.zerodha.com/

UN=DV1510

PWD=Year@123

PIN=959594

UN1=DV0000

==============================================

Utility Class🡪

**package** KiteTestBaseUtility;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** java.util.Properties;

**import** org.openqa.selenium.OutputType;

**import** org.openqa.selenium.TakesScreenshot;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.io.FileHandler;

**public** **class** Utility123 {

**public** **static** **void** captureScreenshot(WebDriver driver, **int** TCID) **throws** IOException

{

File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.***FILE***);

File dest= **new** File("F:\\Velocity\\May-2021 Class\\Java\\Screenshot eg\\TestCaseID"+TCID+".png");

FileHandler.*copy*(src, dest);

}

**public** **static** String getDatafromPropertiesFile(String key) **throws** IOException

{

Properties obj= **new** Properties();

FileInputStream Myfile= **new** FileInputStream("F:\\selenium\\sel\\Java\\TestingSelenium\\Test2\\KiteCredentials.properties");

obj.load(Myfile);

String vaule = obj.getProperty(key);

**return** vaule;

}

}

======================================================================

Base Class🡪

**package** KiteTestBaseUtility;

**import** java.io.IOException;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.chrome.ChromeOptions;

**public** **class** Base123 {

WebDriver driver;

**public** **void** browserInitialize() **throws** IOException

{

ChromeOptions options=**new** ChromeOptions();

options.addArguments("--disable-notifications");

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

driver.get(Utility123.*getDatafromPropertiesFile*("URL"));

driver.manage().timeouts().implicitlyWait(100, TimeUnit.***SECONDS***);

}

}

=====================================================================

POM1🡪 Same as previous POMS no change

POM2🡪 Same as previous POMS no change

POM3🡪 Same as previous POMS no change

=====================================================================

Test Class🡪

**package** KiteTestBaseUtility;

**import** java.io.IOException;

**import** org.apache.poi.EncryptedDocumentException;

**import** org.testng.Assert;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**import** org.testng.annotations.AfterClass;

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

**import** org.testng.annotations.BeforeClass;

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

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

**import** TestNGPOM.KiteHomePage3;

**import** TestNGPOM.KiteLoginPapge1;

**import** TestNGPOM.KitePinPage2;

**public** **class** TestLoginLogoutUsingPropertyFile **extends** Base123{

KiteLoginPapge1 page1;

KitePinPage2 page2;

KiteHomePage3 page3;

**int** TCID=123;

@BeforeClass

**public** **void** LaunchBrowser() **throws** IOException

{

Reporter.*log*("Launching Browser", **true**);

browserInitialize();

page1= **new** KiteLoginPapge1(driver);

page2= **new** KitePinPage2(driver);

page3= **new** KiteHomePage3(driver);

}

@BeforeMethod

**public** **void** LoginApplication() **throws** IOException

{

Reporter.*log*("Application Logged in", **true**);

page1.SendUserId(Utility123.*getDatafromPropertiesFile*("UN"));

page1.SendPassword(Utility123.*getDatafromPropertiesFile*("PWD"));

page1.ClickOnLoginButton();

page2.EnterPin(Utility123.*getDatafromPropertiesFile*("PIN"));

page2.ClickOnContinue();

}

@Test

**public** **void** ValidateUserID() **throws** EncryptedDocumentException, IOException

{

Reporter.*log*("Running "+TCID, **true**);

String ActualUserID = page3.UserIdValidation();

String ExpectedUserID = Utility123.*getDatafromPropertiesFile*("UN1");

Assert.*assertEquals*(ActualUserID, ExpectedUserID,"User ID not matching TC failed");

Reporter.*log*("User ID matching TC Passed ", **true**);

//Utility123.captureScreenshot(driver, TCID);

}

@AfterMethod

**public** **void** LogOutFormApplication(ITestResult result) **throws** InterruptedException, IOException

{

**if**(result.getStatus()==ITestResult.***FAILURE***)

{

Utility123.*captureScreenshot*(driver, TCID);

}

Thread.*sleep*(2000);

Reporter.*log*("Looging out ", **true**);

page3.clickOnUserid();

page3.ClickOnLogout();

}

@AfterClass

**public** **void** CloseBroswer() **throws** InterruptedException

{

Thread.*sleep*(2000);

Reporter.*log*("Closing Browser ", **true**);

driver.close();

}

}

====================================================================

**Listener**

There are different interfaces provided by Java that allow you to modify TestNG behavior. These interfaces are further known as TestNG Listeners in Selenium WebDriver. TestNG Listeners also allow you to customize the tests logs or report according to your project requirements. TestNG Listeners in Selenium WebDriver are modules that listen to certain events and keep track of test execution while performing some action at every stage of test execution.

TestNG Listeners in Selenium WebDriver can be implemented at two levels:

=====================================================================

**Class level**: In this, you can implement listeners for each particular class, no matter how many test cases it includes.

===================================================================

package ListenerStudy;

import org.testng.Assert;

import org.testng.Reporter;

import org.testng.annotations.Listeners;

import org.testng.annotations.Test;

**@Listeners(ListenerStudy.Listener.class)**

public class TestClass {

@Test

public void TC1()

{

Reporter.log("TC1 Running",true);

}

@Test

public void TC2()

{

Reporter.log("TC2 Running",true);

}

@Test

public void TC3()

{

Assert.fail();

Reporter.log("TC3 Running",true);

}

@Test(dependsOnMethods = "TC3")

public void TC4()

{

Reporter.log("TC4 Running",true);

}

}

======================================================================

**Listener class🡪**

**package** ListenerStudy;

**import** org.testng.ITestListener;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**public** **class** Listener **implements** ITestListener {

@Override

**public** **void** onTestFailure(ITestResult result) {

Reporter.*log*("Take Screenshot",**true**);

}

@Override

**public** **void** onTestSuccess(ITestResult result) {

Reporter.*log*("Test Passed Successful",**true**);

}

@Override

**public** **void** onTestSkipped(ITestResult result) {

Reporter.*log*("This test is skipped",**true**);

}

}

===================================================================

**Suite level**: In this, you implement listeners for a particular suite which includes several classes of test cases.

====================================================================

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

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

<suite name="Suite">

<listeners>

<listener class-name="KiteTestBaseUtility.Listener"></listener>

</listeners>

<test name="Test">

<classes>

<class name="KiteTestBaseUtility.TestLoginLogoutUsingPropertyFile"/>

</classes>

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

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

======================================================================

**Types of TestNG Listeners in Selenium WebDriver**

There are numerous TestNG listeners in Selenium WebDriver, some of them are used very frequently by the testing community.

**ITestListener**

IAnnotationTransformer

IInvokedMethodListener

ISuiteListener

IReporter

IConfigurable

IExecutionListener

IHookable

IMethodInterceptor

IConfigurationListener

**ITestListener**

ITestListener is the most adopted TestNG listener in Selenium WebDriver. It provides you with an easy to implement interface through a normal Java class, where the class overrides every method declared inside the ITestListener. By using this TestNG listener in Selenium WebDriver, you can change the default behaviour of your test by adding different events to the methods. It also defines a new way of logging or reporting.

The following are some methods provided by this interface:

* **onStart**: This method is invoked before any test method gets executed. This can be used to get the directory from where the tests are running.
* **onFinish**: This method is invoked after all tests methods gets executed. This can be used to store information of all the tests that were run.
* **onTestStart**: This method is invoked before any test methods are invoked. This can be used to indicate that the particular test method has been started.
* **onTestSkipped**: This method is invoked when each test method is skipped. This can be used to indicate that the particular test method has been skipped.
* **onTestSuccess**: This method is invoked when any test method succeeds. This can be used to indicate that the particular test method has successfully finished its execution.
* **onTestFailure**: This method is invoked when any test method fails. This can be used to indicate that the particular test method has failed. You can create an event for taking a screenshot which will show where the test has been failed.

For every ITestListener method we usually pass the following arguments:

* “ITestResult” interface along with its instance, “result,” which describes the result of a test. Note: If you want to trace your exception through ITestResult then you need to avoid try/catch handling.
* “ITestContext” interface along with its instance “context” which describes the test context containing all the information of the given test run.

Extent Reporter

ExtentReports is an open-source reporting library used in selenium test automation.

Extent reports become the first choice of Selenium Automation Testers, even though

Selenium comes with inbuilt reports using frameworks like JUnit and TestNG

=====================================================================

How to take screen shot of failed TC🡪

**We have made changes in base class and listener only Test class you can use old one just comment out capturescreenshot from test class**

**Base class🡪**

**=================================================================**

**package** SuiteLevelListerner;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.OutputType;

**import** org.openqa.selenium.TakesScreenshot;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.openqa.selenium.chrome.ChromeOptions;

**import** org.openqa.selenium.io.FileHandler;

**public** **class** Base {

**static** WebDriver *driver*;

**public** **void** browserInitialize()

{

ChromeOptions options=**new** ChromeOptions();

options.addArguments("--disable-notifications");

System.*setProperty*("webdriver.chrome.driver", "F:\\Velocity\\May-2021 Class\\test2\\chromedriver\_win32 (1)\\chromedriver.exe");

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

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

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

*driver*.manage().timeouts().implicitlyWait(100, TimeUnit.***SECONDS***);

}

**public** **void** captureScreenshot( String TCName) **throws** IOException

{

File src = ((TakesScreenshot)*driver*).getScreenshotAs(OutputType.***FILE***);

File dest= **new** File("F:\\Velocity\\May-2021 Class\\Java\\Screenshot eg\\TestCase"+TCName+".png");

FileHandler.*copy*(src, dest);

}

}

**================================================================**

**Utility🡪 remove screenshot code from utility**

package SuiteLevelListerner;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import org.apache.poi.EncryptedDocumentException;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.WorkbookFactory;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.io.FileHandler;

public class Utility {

public static String getDatafromExcel(int rowIndex, int columnIndex) throws EncryptedDocumentException, IOException

{

FileInputStream Myfile= new FileInputStream("C:\\Users\\user\\Desktop\\mytesting.xlsx");

Sheet Mysheet = WorkbookFactory.create(Myfile).getSheet("Sheet4");

String value = Mysheet.getRow(rowIndex).getCell(columnIndex).getStringCellValue();

return value;

}

// public static void captureScreenshot(WebDriver driver, int TCID) throws IOException

// {

// File src = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

// File dest= new File("F:\\Velocity\\May-2021 Class\\Java\\Screenshot eg\\TestCaseID"+TCID+".png");

// FileHandler.copy(src, dest);

//

// }

}

=================================================================

**package** SuiteLevelListerner;

**import** java.io.IOException;

**import** org.testng.ITestContext;

**import** org.testng.ITestListener;

**import** org.testng.ITestResult;

**public** **class** Listener **extends** Base **implements** ITestListener {

Base b= **new** Base();

@Override

**public** **void** onTestStart(ITestResult result) {

}

@Override

**public** **void** onTestSuccess(ITestResult result) {

}

@Override

**public** **void** onTestFailure(ITestResult result) {

System.***out***.println("Test Failed plz try again faild name is "+result.getName());

String TCName = result.getName();

**try** {

b.captureScreenshot(TCName);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

@Override

**public** **void** onTestSkipped(ITestResult result) {

}

@Override

**public** **void** onStart(ITestContext context) {

}

@Override

**public** **void** onFinish(ITestContext context) {

}

}

==========================================================

TestNG.xml -🡪

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

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

<suite name=*"Suite"*>

<listeners>

<listener class-name=*"SuiteLevelListerner.Listener"*></listener>

</listeners>

<test name=*"Test"*>

<classes>

<class name=*"SuiteLevelListerner.TestLoginLogout"*/>

</classes>

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

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

==============================================================

