­­­­­­TestNG

**TESTNG:(Unit testing framework tool)**

**testng Java**

**Dot net**

**Junit --------------java**

**Nunit----------.net**

**Py dev----------phyton**

**Rspsc----------------rubby**

**All unit testing framework tool is implemented as plugin for eclipse IDE, but junit is a default plugin for eclipse IDE.**

**Installation steps of Testng:**

**Goto Eclipse window**

* **click on “help” option-> click on “Eclipse Marketplace”**
* **write “testng “ in find edit box and click on “go” button.**
* **Find “ testng for eclipse” division and click on “install” button**
* **Click on “confirm” button and click on “ I accept the terms and conditions “ radio button ,and click on “finish”**
* **Restart the Eclipse**

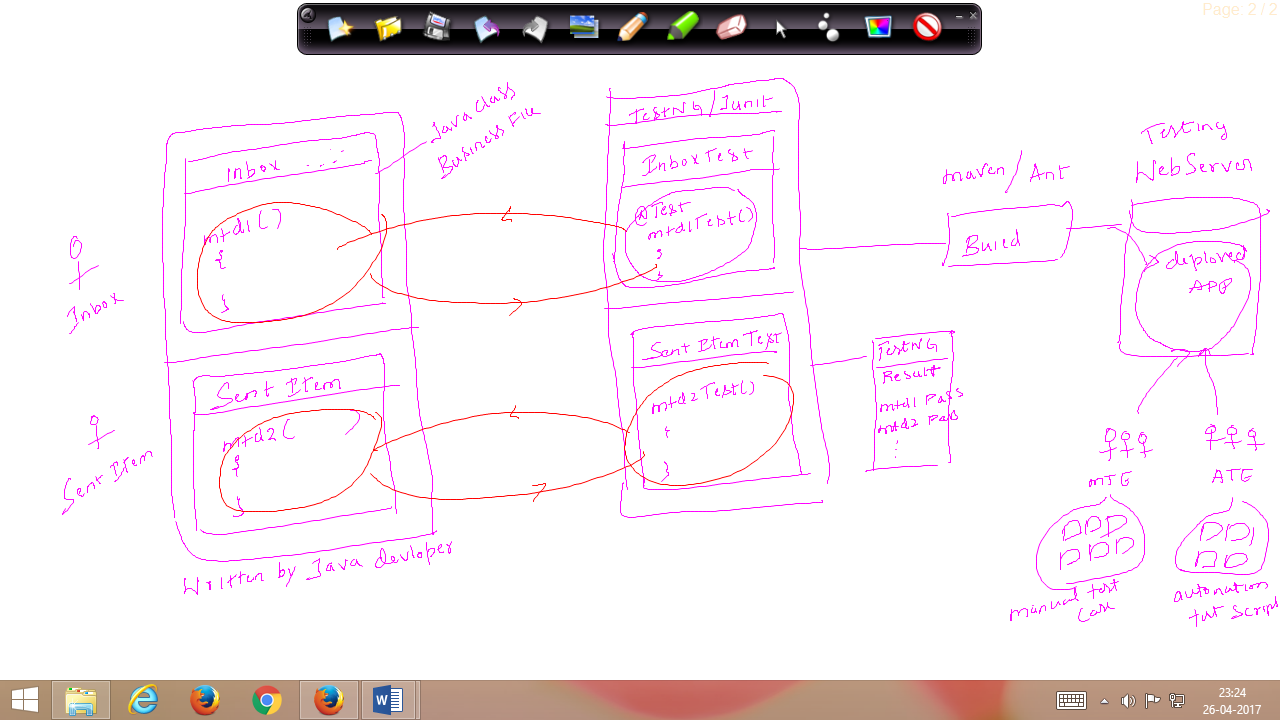
**In order to verify the testng installation-🡪 goto eclipse window & click om “windows” Options🡪show view--🡪 others---🡪expand java folder , then u *will* see testng symbol**

After the TestNg installation , Make Sure TestNG Jar being imported to Eclipse Project

Selct “javaProject”--🡪 Right click 🡪build path 🡪add libraries🡪Selct “TestNG” Option 🡪 click on “finish”

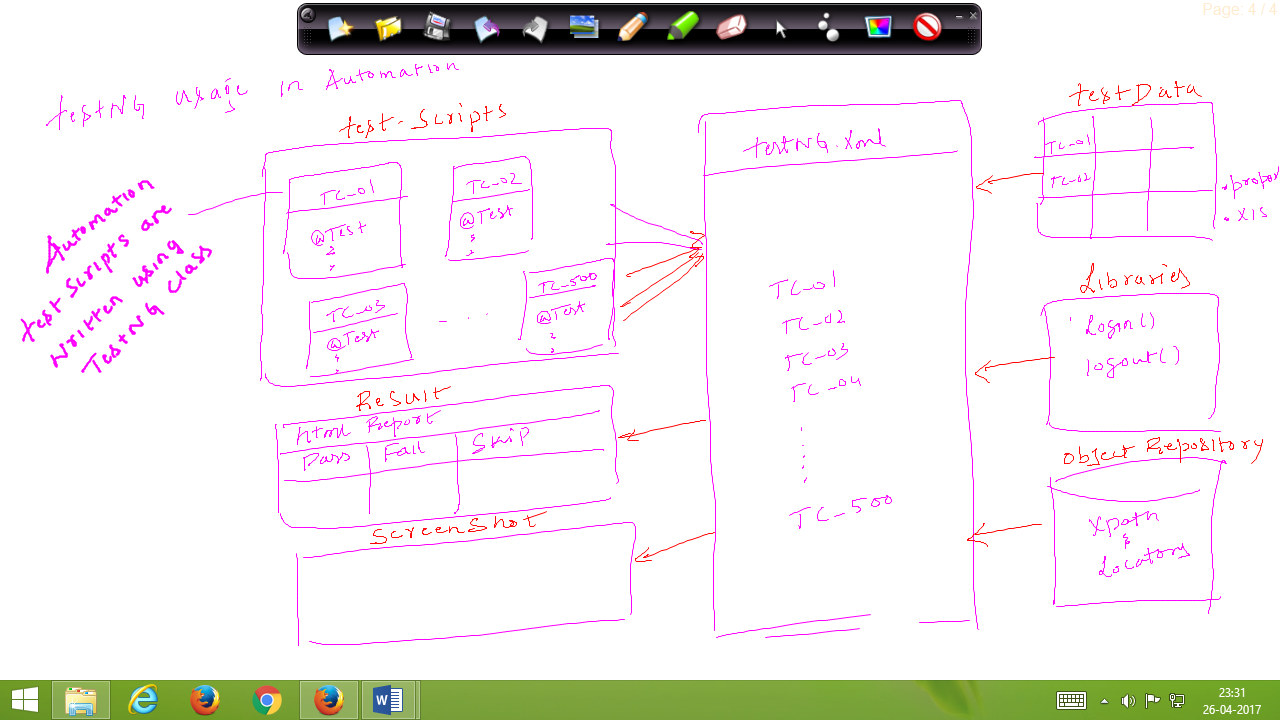
**Testng is used in case of development.**

Testng is used in development,



In case of development, testng will used to develop, unit test cases and each unit test case will test business logic of the source code.

**Testng used in case of selenium automation**



In case of automation, testng will be used to develop all the scripts using testng annotations and achieve batch execution without any manual interaction.

Testng will be used to handle framework component.

Testng is inspired from junit and nunit. but introducing few new functionality, that makes testng become more powerful.

New functionalities are :

->Html report

->Parallel execution

->Grouping execution

->Additonal annotations

->batch execution is easier.

**Annotations:**

**It’s a feature of Java ,**which provide information(meta Data) to testng compiler at the time of execution.

Always start with “@” symbol

Annotation mtds will be executed automatically by compiler , no need to call them

It will be always written with in JAVA class Block

* @Test
* @BeforeMethod
* @AfterMethod
* @BeforeClass
* @AfterClass
* @BeforeTest
* @AfterTest
* @paramaters
* @dataProvider
* @Listener

Development testng scripts typically uses 4 Steps:

Step1 :

* Create testng class
* Select package & right click
* Select testng and click on “create testng class”

Steps2:

* Create testng test method within a class

Publicc class newtest

{

@Test

Public void createCustomer()

{

System.out.println(“execute create customer”);

}

}

Step3:

* Run testng class ----select file—right click----and “run as “ testng test

Steps 4:

Verify html report

Refresh the project after execution—select project right click and click on refresh

Automatically we get **test-Output** folder within the same project.

Expand test-output folder 🡪 select “emailable.html “and right click 🡪 open with🡪 Open with “webbrowser”.

@Test

Whenever we execute testng class, testng compiler always looks for @test Annotation method to start the execution.

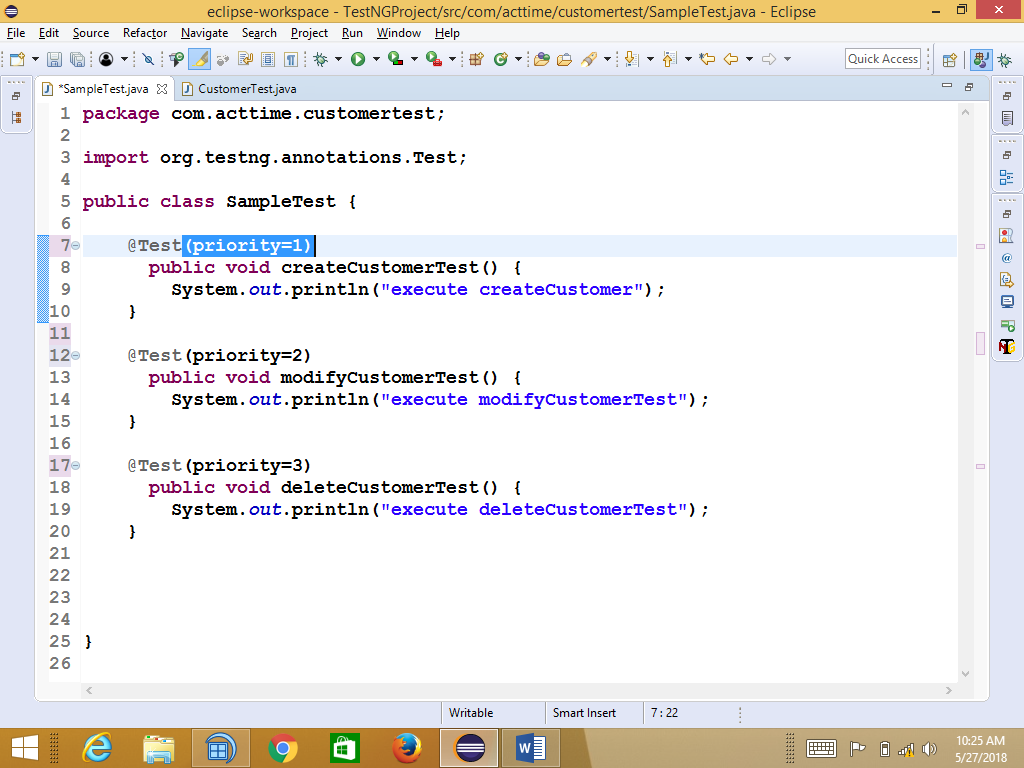
**Without @Test , testng class cannot be executed, @test annotation method act like a main method.**

In one testng class we can have multiple @test methods, but each test method should have @Test annotation before method signature.

Annotation method return type should be “void” and access specifier should be public., but method name can be anything .

As per the TestNG rule “classNAme” & “methodName” should end with Test

WhenEver we execute TestNG class , all the @test mtd will be executed based on alphabetical Order , in order the change Order of the Execution we go for priority



**@BeforeMethod @AfterMethod**

**Before method annotations will be executed, before executing each @test method in a class**

**After method annotation will be executed, after executing each @test in a class**

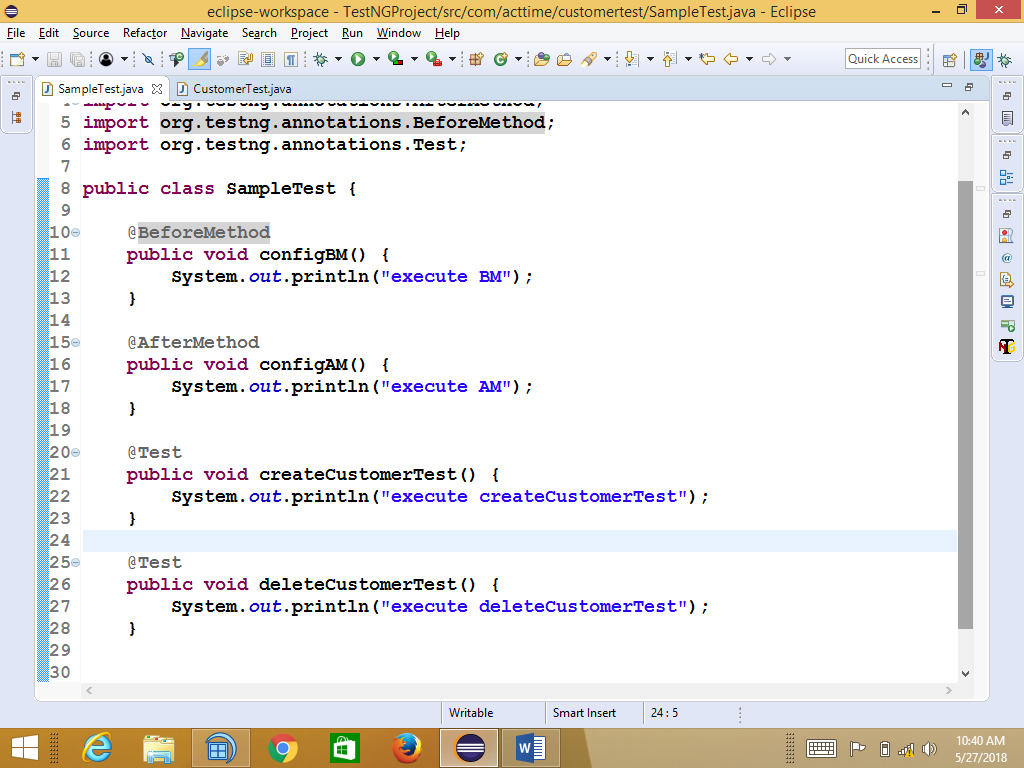
**@Beforemethod &@aftermethod will not be executed, without @test annotation method.**

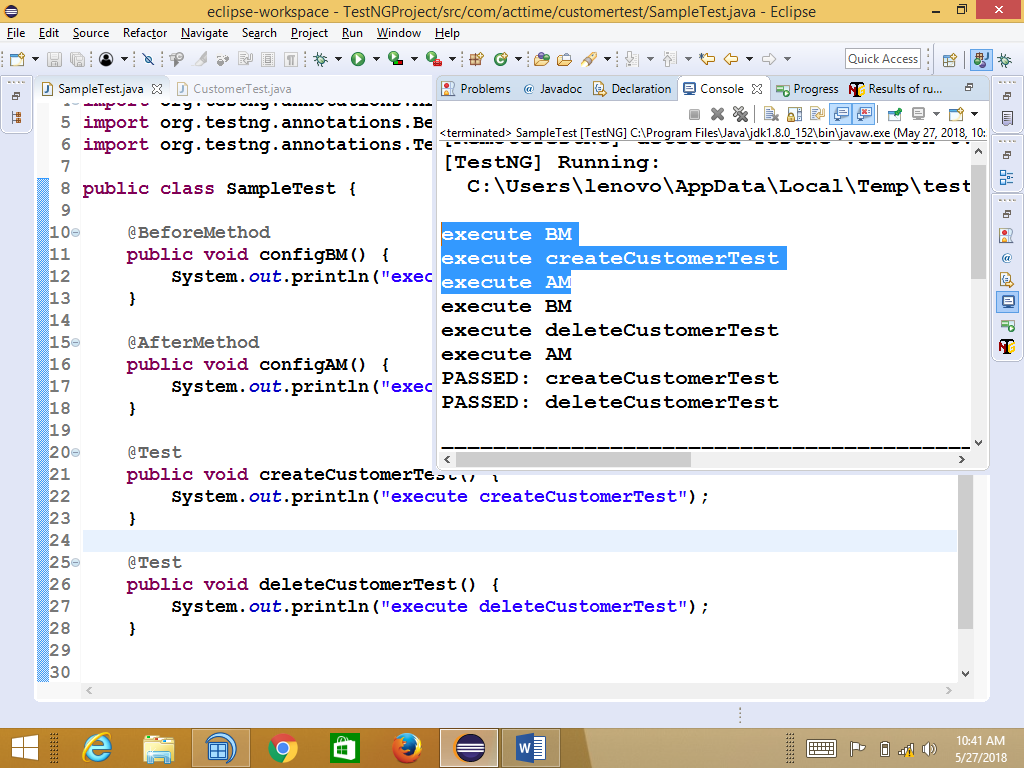
**In order to implement similar pre and post condition for all the test case, we go for @Beforemethod &@aftermethod**

**EG**

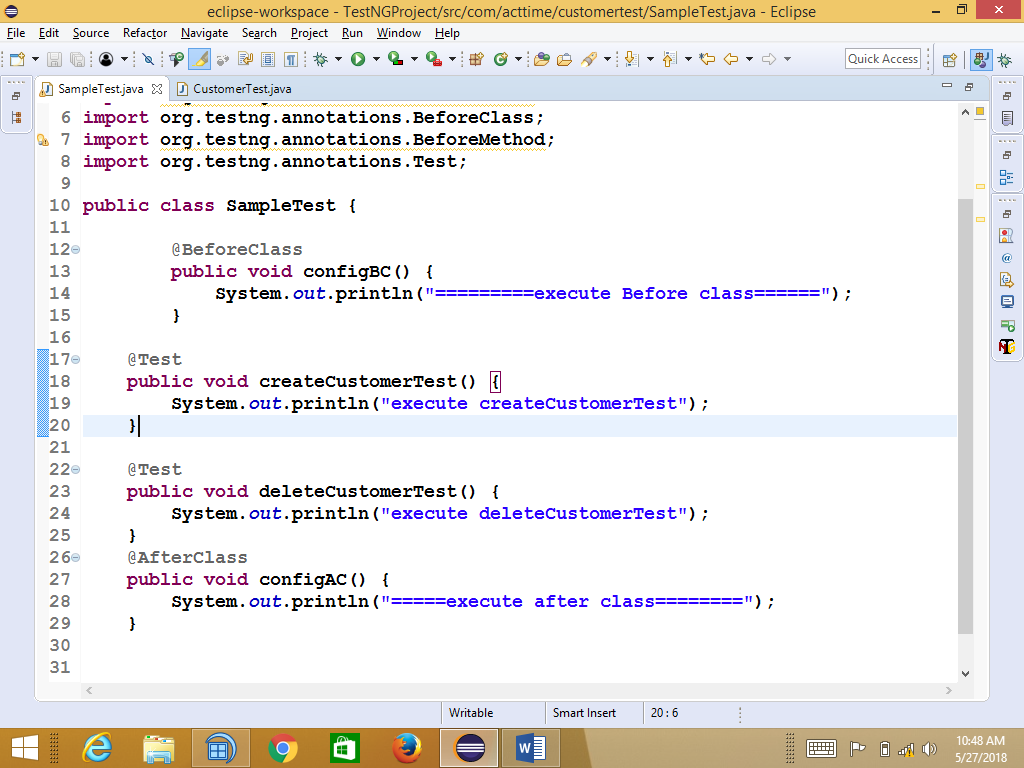
**In order to devlop “login code” we should go for @Beforemethod , because login code common for all the test-scripts**

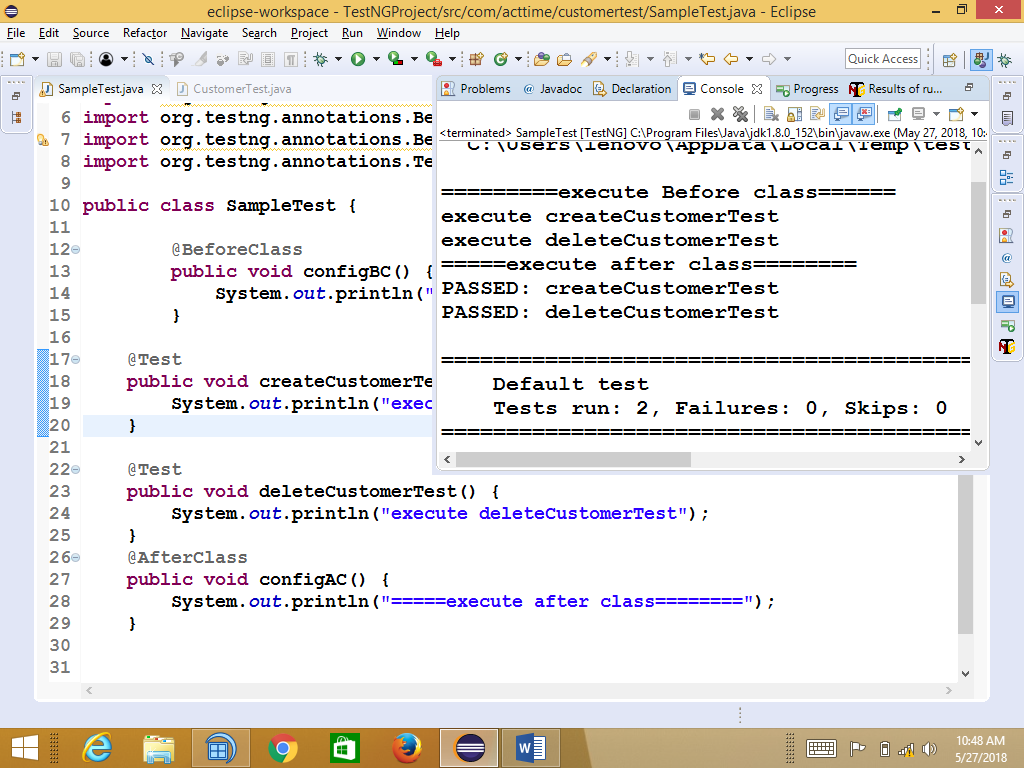
**In order to devlop “logout code” we should go for @Aftermethod , because logout code common for all the test-scripts**





**BeforeClass & afterclass**





BeforeClass Annotation method will be executed, before Executing first @test in a class

AfterClass Annotation method will be executed, after executing all @test in a class

BeforeClass & afterClass annotations will be executed only once in a entire class execution.

It will be used to develop global configuration like launch browser, object initliziation , database-connection

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

As per the automation rule, Test Cases shouldn’t have a dependency between each other. Every test case should have login and logout and it should be unique.

Real Example to use Annotation

package com.acttime.customertest;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Actions;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

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;

public class SampleTest {

WebDriver driver; //global driver Object declaration

@BeforeClass

public void configBC() {

System.out.println("=========launch browser======");

driver = new FirefoxDriver();

}

@BeforeMethod

public void configBm() {

System.out.println("login");

driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

driver.get("http://localhost:80");

driver.findElement(By.name("username")).sendKeys("admin");

driver.findElement(By.name("pwd")).sendKeys("manager");

driver.findElement(By.id("loginButton")).click();

//wait for TASK Image using Custom-Wait

while(true) {

try {

driver.findElement(By.xpath("//div[text()='TASKS']"))

.click();

break;

}catch (Exception e) {

System.out.println("handle the expection");

}

}

}

@Test

public void createCustomerTest() {

}

@Test

public void modifyCustomerTest() throws Throwable {

System.out.println(" createCustomerTest");

driver.findElement(By.xpath("//div[text()='Add New']")).click();

driver.findElement(By.xpath("//div[text()='+ New Customer']")).click();

driver.findElement(By.id("customerLightBox\_nameField")).sendKeys("ICICI\_008\_BTR");

driver.findElement(By.id("customerLightBox\_commitBtn")).click();

//modify the Customer

driver.findElement(By.xpath("(//input[@placeholder='Start typing name ...'])[1]")).sendKeys("ICICI\_008\_BTR");

WebElement wb = driver.findElement(By.xpath("//div[@class='node customerNode selected']/div[4]"));

Actions act = new Actions(driver);

act.moveToElement(wb).click().build().perform();

Thread.sleep(3000);

driver.findElement(By.xpath("//div[@class='customerNamePlaceHolder']/div/div[1]")).click();

driver.findElement(By.xpath("(//input[@placeholder='Enter Customer Name'])[2]")).clear();

driver.findElement(By.xpath("(//input[@placeholder='Enter Customer Name'])[2]")).sendKeys("Airtel\_007\_BTR",Keys.ENTER);

}

@AfterMethod

public void configBM() {

System.out.println("logout");

driver.findElement(By.linkText("Logout")).click();

}

@AfterClass

public void configAC() {

System.out.println("=====close browser========");

driver.close();

}

}

**Batch Execution**

* Collection of multiple test script is called batch, Execute multiple test script through xml in a single click is called batch execution.
* In order to achieve batch execution we go for Testng.xml file.
* Testng xml file always start with suite xml tag followed by test tag and classes.
* In one xml file we can invoke n number of testng classes, but all the classes should be present within a project.

**How to create testng xml file automatically through eclipse?**

1. **Select all the “testng classes”-> right click-> select “testng” and click on “convert to testng “ and click on “finish”.**

**Automatically you will get the “testng.xml “ file with in same Project**

**2. click “Source” button to View Xml Data**

1. **. In order to run the TestNG.xml , Select “xml file”**🡺 **right click** 🡺**run as**🡺**testNg Suite**

**Batch Execution**

ProjectAndCustTest

@test

@test

@test

<suite>

<test>

<classes>

<class name=”PRojectAndCustTest”>

<class name=”ReportTest”>

</classes>

</test>

</suite>

|  |  |
| --- | --- |
| modules | Mtc |
| ProjAndCust | 10 |
| Report | 10 |

ReportTest

@test

@test

@test

**Grouping Execution**

ProjectAndCustTest

@test RT

@test

@test ST

@test

<suite>

<groups>

<run>

<include name=”RT”>

</run>

</groups>

<test>

<classes>

PRojectAndCustTest

ReportTest

</classes>

</test>

</suite>

|  |  |
| --- | --- |
| modules | Mtc |
| ProjAndCust | 10🡪8 RT  2 ST |
| Report | 10🡪8 RT  2 ST |

ReportTest

@test RT

@test

@test ST

@test

* Collection of similar test scripts across the testing classes is called grouping Execution
* In order achieve grouping execution , each & every test script should have group name

, group will be written along with annotation

* One test script can have multiple group name

@Test(groups={"smokeTest"})

**public** **void** createCustomerTest(){

System.***out***.println("execute createCustomerTest");

}

@Test(groups={"regressionTest","smokeTest"})

**public** **void** modifyCustomerTest(){

System.***out***.println("execute modifyCustomerTest");

}

* In order to invoke grouping execution should declare Group Key in testing.xml file

& group key should be declare before <test> , after <suite> tag

<suite name=*"Suite"*>

<groups>

<run>

<include name=*"smokeTest"*/>

</run>

</groups>

<test name=*"Test"*>

<classes>

<class name=*"pac1.ProjectAndCustomerTest"*/>

<class name=*"pac2.ReportTest"*/>

</classes>

</test>

</suite>

How to disable the test in TESTNG class Execution

@Test(enabled=**false**)

**public** **void** createCustomerTest(){

System.***out***.println("execute createCustomerTest");

}

Or

<suite name=*"Suite"*>

<test name=*"Test"*>

<classes>

<class name=*"pac1.ProjectAndCustomerTest"*>

<methods>

<exclude name=*"deleteCustomerTest"*/>

</methods>

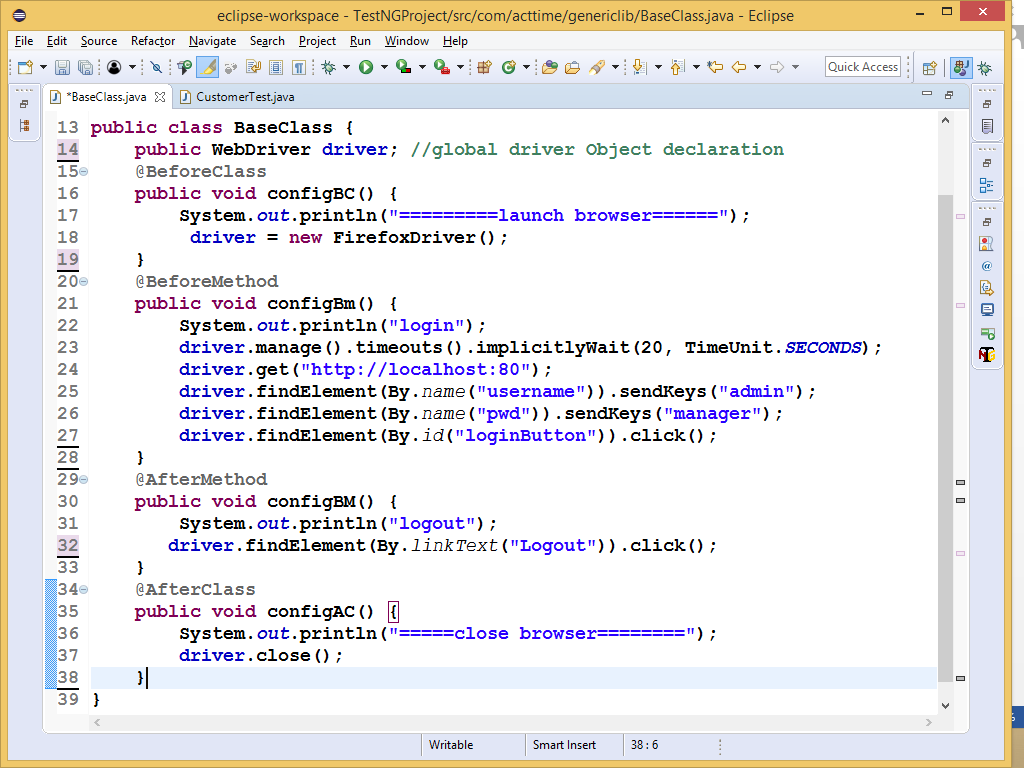
</class>

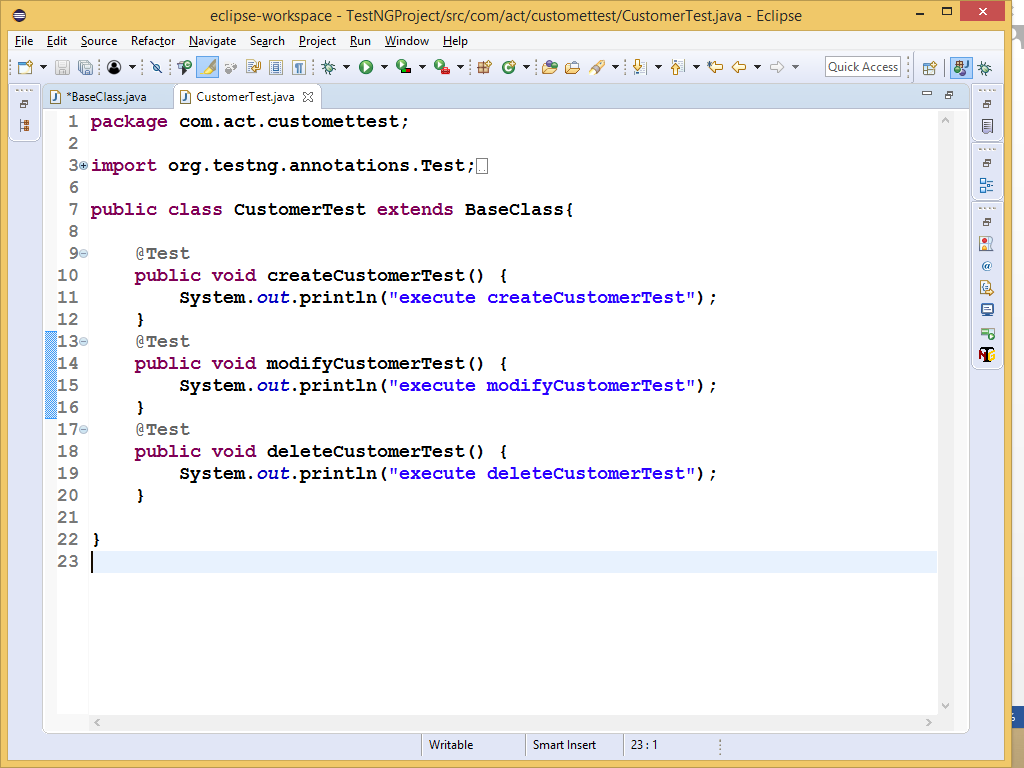
</classes>

</test>

</suite>

RealTime Annotation usage in FrameWork



Test Scripts

**Assertions / checkpoint**

Assertions is a concept of the tesNG , which is used to verify the expected Result of the testscipts

* As per the Automation rule every test scripts should be verified with assert statement
* Test should not verified with Java IF/Else block , because it never fails tesNG test test
* Whenever assert statement fails , testing generate AssertError Excpetion along Line Number of the failed test step

Assertions

HardAssert SoftAssert

Class SoftAssert

Class Assert

🡪assertEquals() 🡪assertEuals()

🡪assertNotEquals() 🡪assertNotEquals()

🡪assertTrue() 🡪assertTrue()

🡪assertFalse() static mtd 🡪assertFalse()

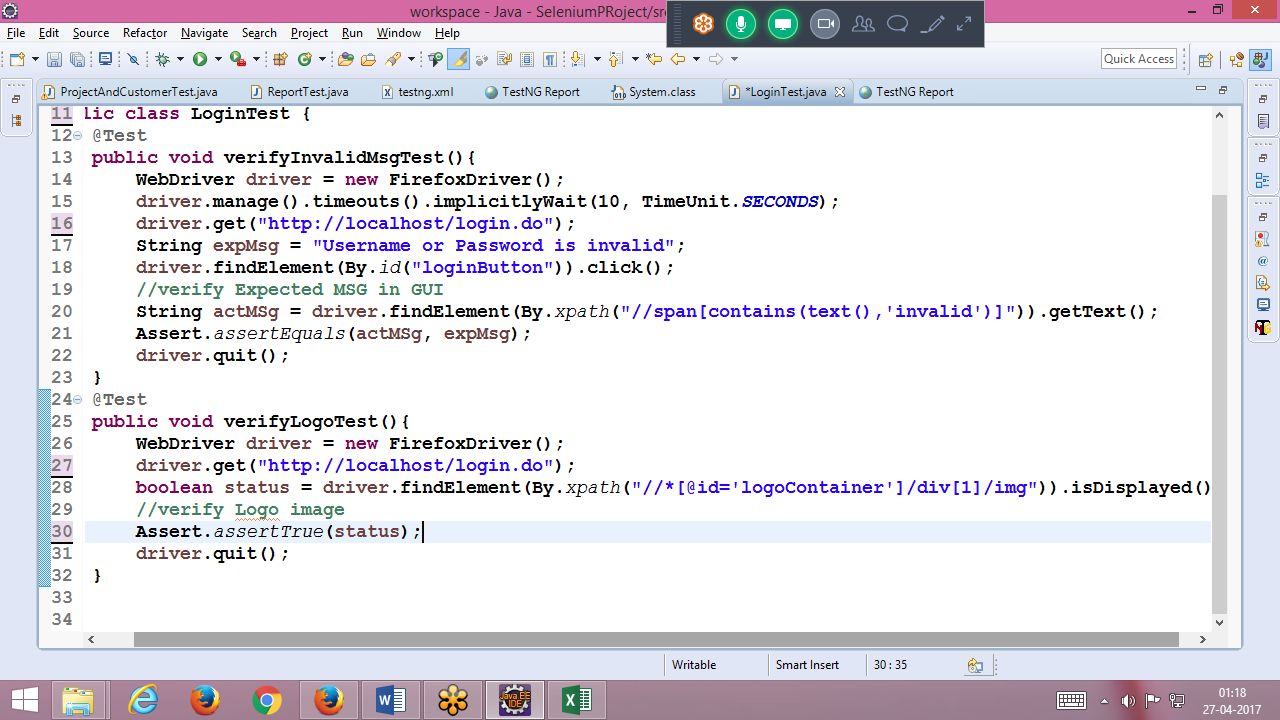
🡪assertNull() 🡪assertNull()

🡪assertAll()

Non static

|  |  |  |  |
| --- | --- | --- | --- |
| testID |  | tc\_01 |  |
| testName |  | verifyInvalidMSg |  |
| testData |  | admin/deepak, "https://online.actitime.com/emc1/login.do" |  |
| steps |  | Actions | Expected Result |
| 1 |  | navigate to Acttime app | Login page should be display |
| 2 |  | click on Login Btn with username/Password | "Username or Password is invalid. Please try again." msg should be display |

|  |  |  |
| --- | --- | --- |
| testID | tc\_02 |  |
| testName | verifyLogo |  |
| testData | admin/deepak, "https://online.actitime.com/emc1/login.do" |  |
| steps | Actions | Expected Result |
| 1 | navigate to Acttime app | Login page should be display |
| 2 | verify Actitime Logo should be display | Actitime Logo should be display |



HardAssert

All mtd are static mtd

In order verify mandatory data go for Hard assert

Whenever hardAssert mtd fails , testNG Generate AssertError exception & stop the current test execution , & cotinue execution with remaining test

@Test

**public** **void** createCustomerTest(){

System.***out***.println("step\_1");

System.***out***.println("step\_2");

Assert.*assertEquals*("A", "B");

System.***out***.println("step\_3");

System.***out***.println("step\_4");

}

@Test

**public** **void** modifyCustomerTest(){

System.***out***.println("=============================");

System.***out***.println("step\_1");

System.***out***.println("step\_2");

System.***out***.println("step\_3");

System.***out***.println("step\_4");

}

Out Put

step\_1

step\_2

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

step\_1

step\_2

step\_3

step\_4

PASSED: modifyCustomerTest

FAILED: createCustomerTest

java.lang.AssertionError: expected [B] but found [A]

Soft Assert

All the mtd are non static mtd

In order to verify non-mandatory data , we go for SoftAssert

Whenever softAssert mtd fails , testNG Generate AssertError exception & continue remaining test step execution

assertAll() mtd should be used at the end of the test-scripts , because its used collect all the error information & display in console

**public** **void** createCustomerTest(){

System.***out***.println("step\_1");

System.***out***.println("step\_2");

SoftAssert s = **new** SoftAssert();

s.assertEquals("A", "B");

System.***out***.println("step\_3");

s.assertEquals("X", "Y");

System.***out***.println("step\_4");

s.assertAll();

}

@Test

**public** **void** modifyCustomerTest(){

System.***out***.println("=============================");

System.***out***.println("step\_1");

System.***out***.println("step\_2");

System.***out***.println("step\_3");

System.***out***.println("step\_4");

}

Output

step\_1

step\_2

step\_3

step\_4

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

step\_1

step\_2

step\_3

step\_4

PASSED: modifyCustomerTest

FAILED: createCustomerTest

java.lang.AssertionError: The following asserts failed:

expected [B] but found [A],

expected [Y] but found [X]

Parallel Execution

Execute multiple test scripts in multiple browser concurrently is called parallel execution

testNg support maximize 5 parallel test runners

in order achieve parallel execution , we should enable parallel & thread count attribute in <suite >

parallel = “tests” 🡪 it means run <test> in parallel

thread-count =5 🡪 its depends on number of <test> runners available in <suite>

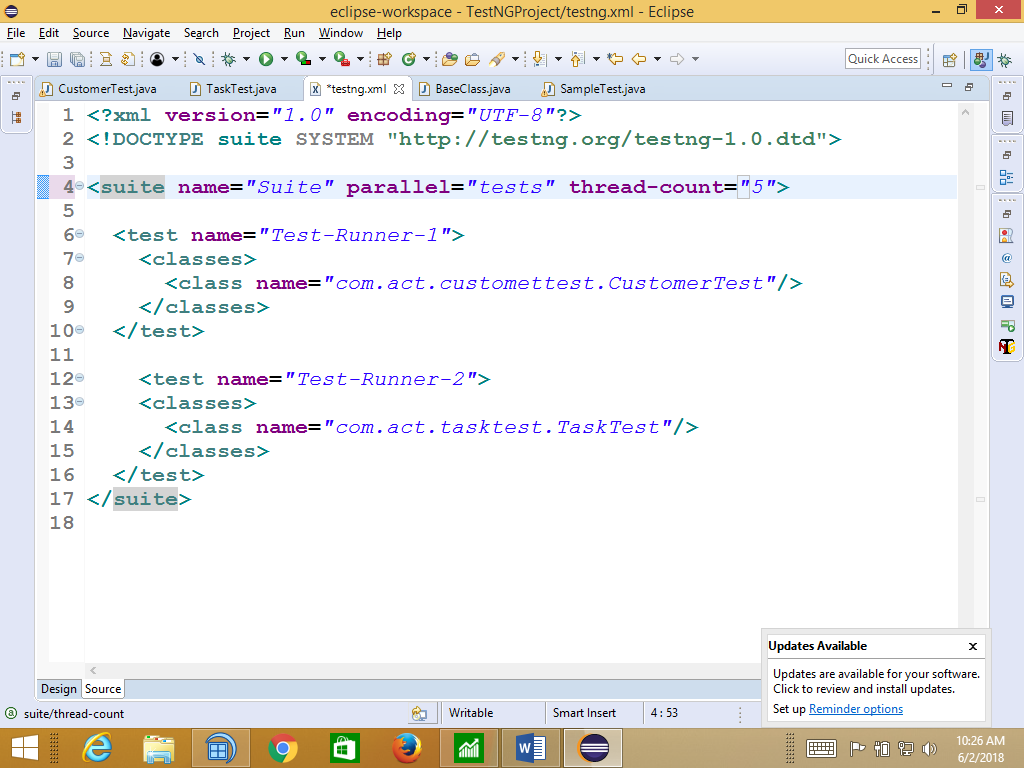
Two types of parallel execution

1. Distribute parallel execution
2. Compatibility parallel execution / Cross Browser
3. Distribute parallel execution:

When even we need to run 200+ of test scripts in short period of time we go for Distribute parallel execution

EG : assume we have 200 testscripts to be run test on firefox browser , but don’t have to time to wait for long duration , in such case we distribute the test-scripts in two test runners and run the test in two different parallel firefox browser

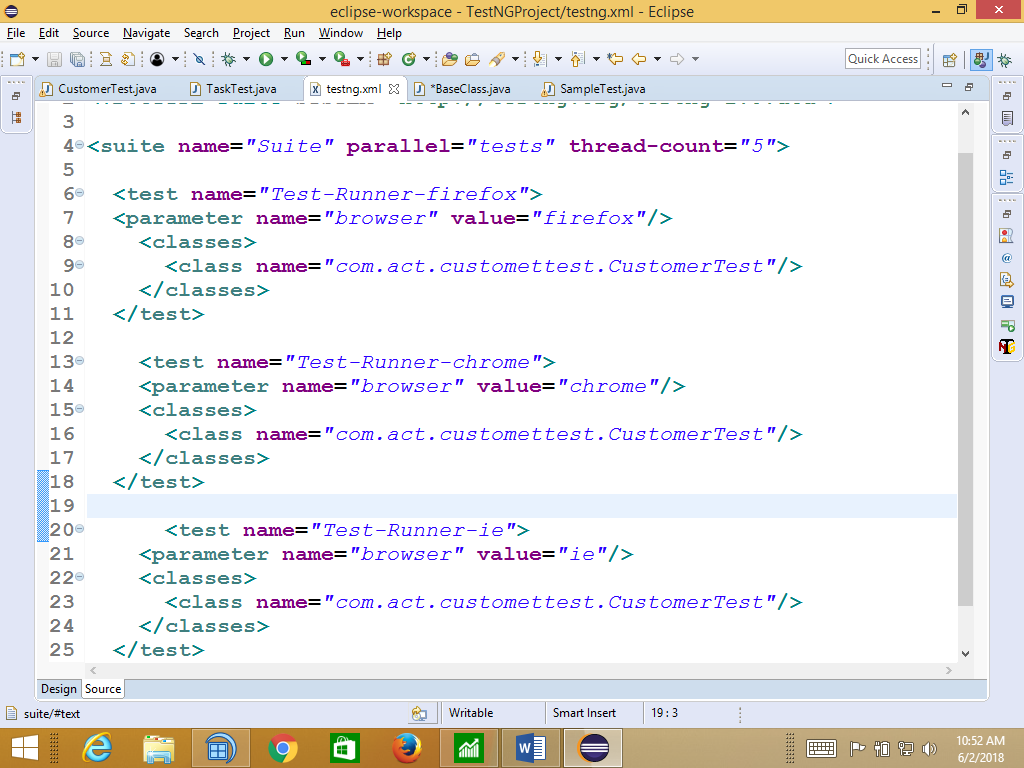
In order to achieve



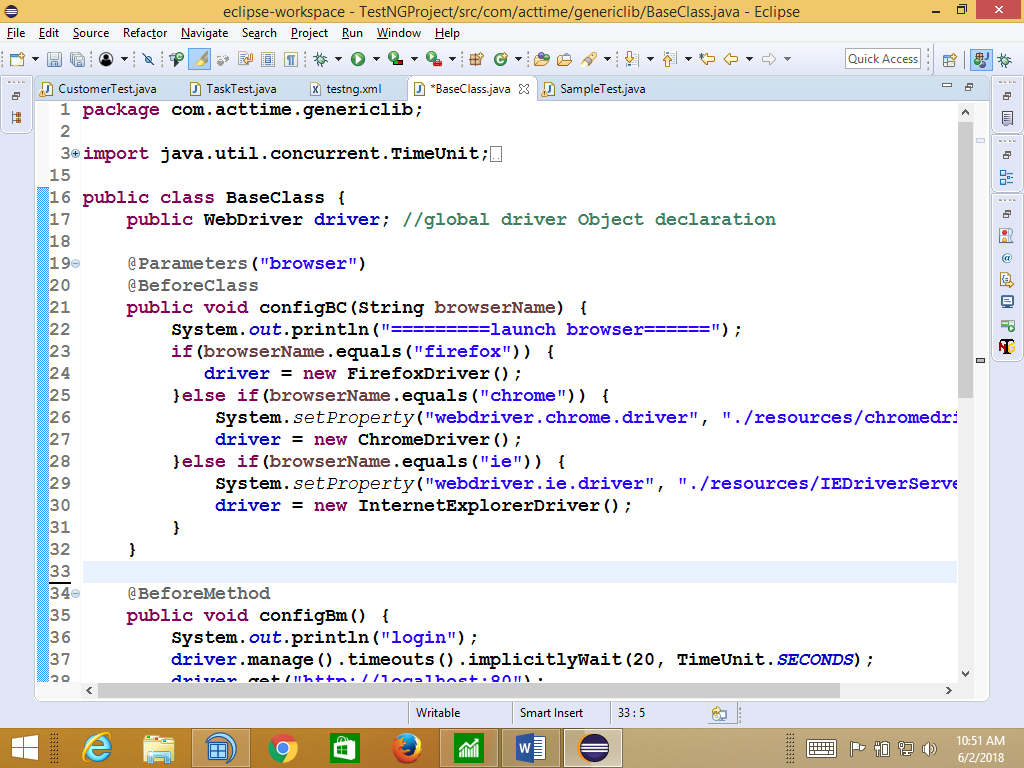
1. Compatibility parallel execution

Execute same test in multiple parallel browser like IE firefox , safari etc ,

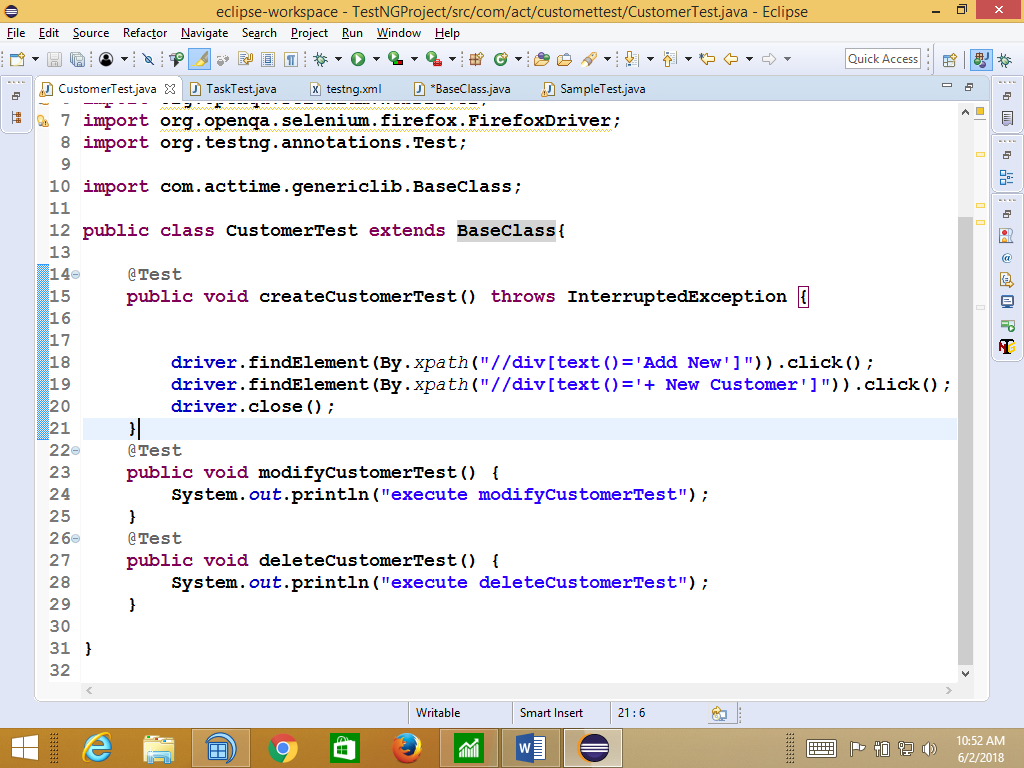
Its is also called as cross browser testing



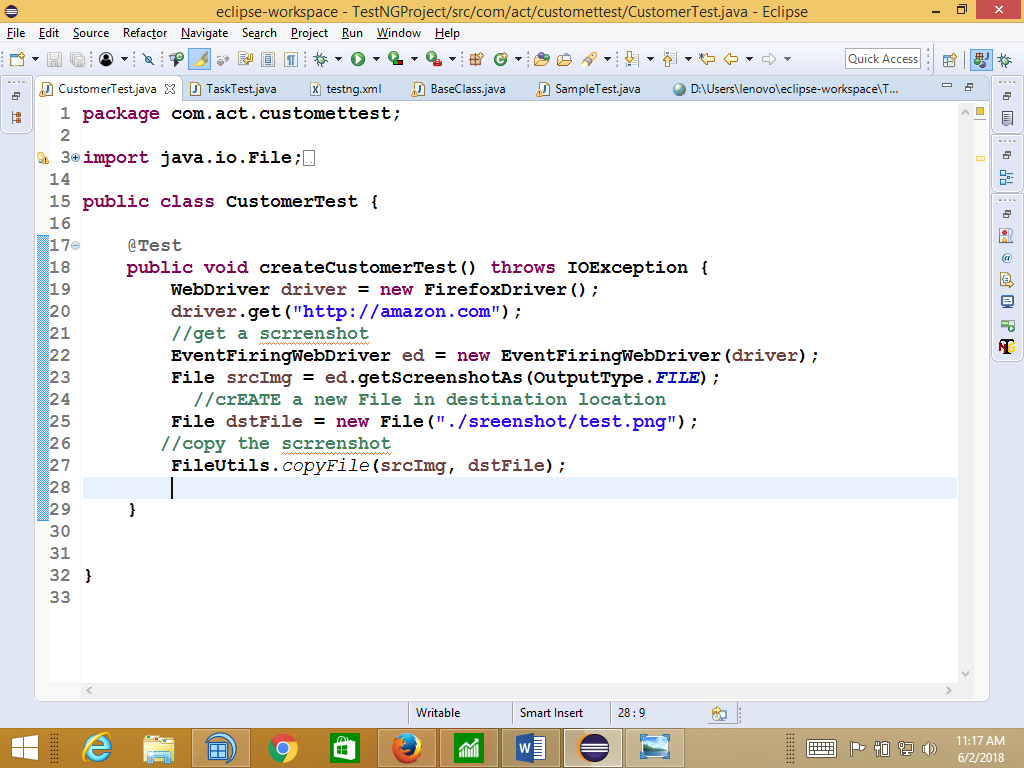
Base class



Testngtest



ScreenShot



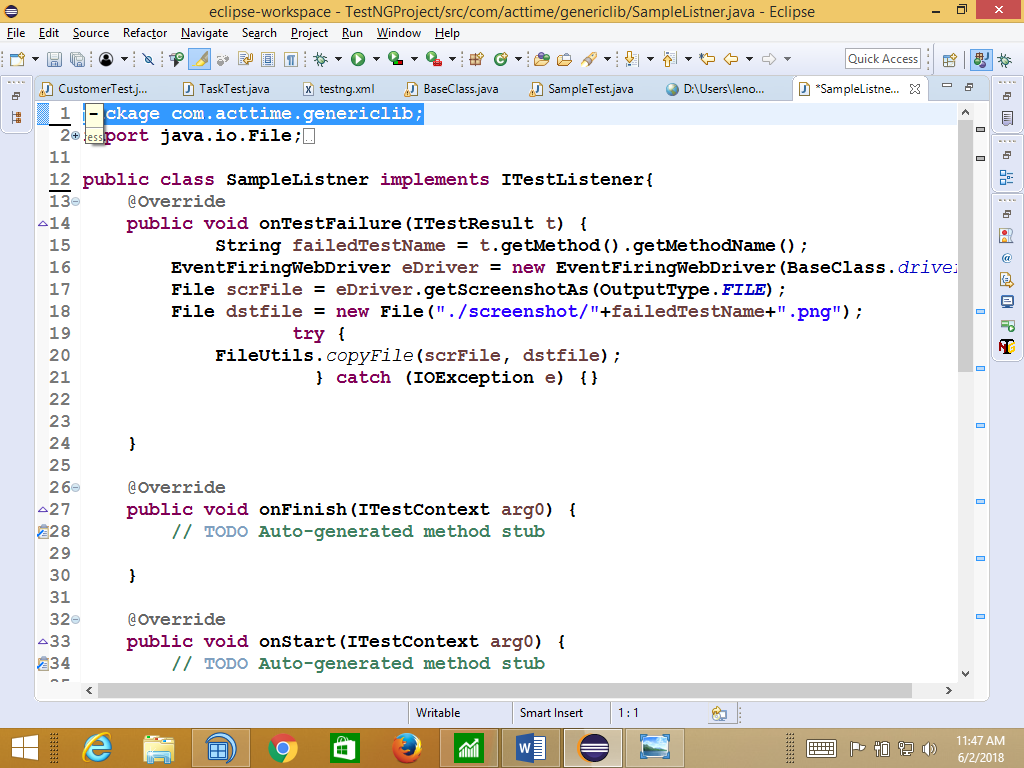
Listener :

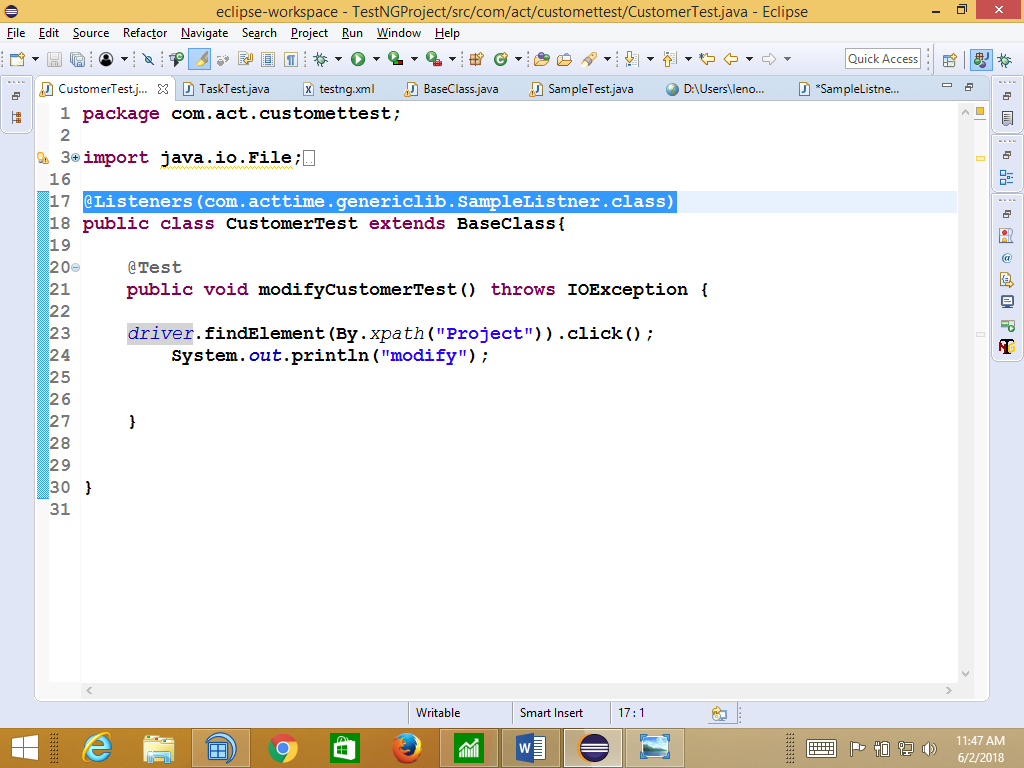
@Listener is a testNg annoataion , which is used to minter the test-Scripts execution at the time of execution , its will generate the failure event when test is getting failed ,

When ever we used @Listener annotation in testNG test scripts , mandate to implement ITESTListener intrerface & implement “ontestFaulire ()” scrrenshot code inside the mtd

whenever TestNg Testr failed , @ Listener automatically generate a failure event & send failure failure event to Listener implement class to take a screenshot

As per the rule @listnere annotation should be written before class declaration Block





Reporting

Reporting feature will be helpful to debug the test ,when we run the test suite with 100 ‘s of test scripts

As per the automation rule should not use hardcoded report like “System.out.println(“msg”)”,

Instead we go testNG advanced class like “Reporter”

Reporter classes will be used to insert user defined report in to High level html report

**package** pac;

**import** org.testng.Reporter;

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

**public** **class** NewTest{

@Test

**public** **void** createTest(){

Reporter.*log*("login to test");

Reporter.*log*("create customer");

Reporter.*log*("modify customer");

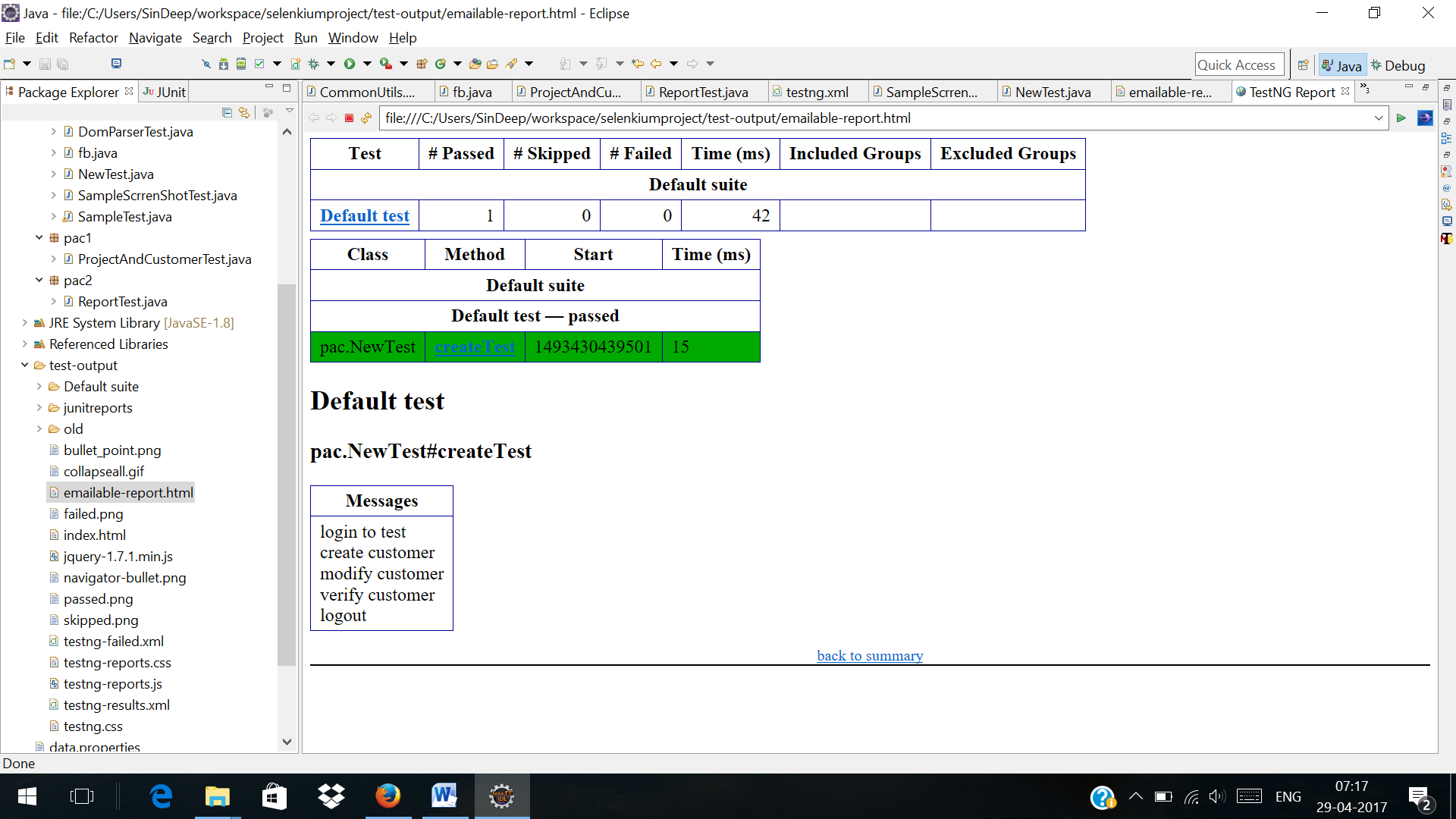
Reporter.*log*("verify customer");

Reporter.*log*("logout");

}

}

Out put



Data Driven Testing , Framework Libraries

LIBRARIES

**package** com.acttime.genericLib;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.util.Properties;

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

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

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

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

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

/\*\*

\*

\* **@author** Deepak

\*

\*/

**public** **class** FileDataUtiles {

String exelPath = "./data/testdata.xlsx";

String filePath = "./data/commonData.properties";

/\*\*

\*

\* **@param** sheetName , rownum , columNum

\* **@return** String data

\* **@throws** Throwable

\*

\* its used to read data from Excel-WorkBook based on user Input

\*/

**public** String getExcelData(String sheetName, **int** rowNum, **int** colNum) **throws** Throwable {

FileInputStream fis = **new** FileInputStream(exelPath);

Workbook wb = WorkbookFactory.*create*(fis);

Sheet sh = wb.getSheet(sheetName);

Row row = sh.getRow(rowNum);

String data = row.getCell(colNum).getStringCellValue();

wb.close();

**return** data;

}

/\*\*

\*

\* **@param** sheetName

\* **@param** rowNum

\* **@param** celNum

\* **@param** data

\* **@throws** Throwable

\*

\* its used to write data back Excel based on user input

\*/

**public** **void** setEcelData(String sheetName, **int** rowNum, **int** celNum, String data) **throws** Throwable {

FileInputStream fis = **new** FileInputStream(exelPath);

Workbook wb = WorkbookFactory.*create*(fis);

Sheet sh = wb.getSheet(sheetName);

Row row = sh.getRow(rowNum);

Cell cel = row.createCell(celNum);

cel.setCellValue(data);

FileOutputStream fos = **new** FileOutputStream(exelPath);

wb.write(fos);

wb.close();

}

/\*\*

\*

\* **@return**

\* **@throws** Throwable

\* its always return properteis File Object & load all the Keys availble in

\* commondata.properteis File in data folder

\*/

**public** Properties getPropertiesFileObj() **throws** Throwable {

FileInputStream fis = **new** FileInputStream(filePath);

Properties pObj = **new** Properties();

pObj.load(fis);

**return** pObj;

}

}

**Test Scripts**

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

**package** com.acttime.customer;

**import** java.io.FileInputStream;

**import** java.util.Properties;

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

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

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

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

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

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

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

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

**import** com.acttime.genericLib.FileDataUtiles;

/\*\*

\*

\* **@author** Deepak

\*

\*/

**public** **class** CreateCustomer {

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

/\* read the data from ExternalResource \*/

FileDataUtiles lib = **new** FileDataUtiles();

/\* read data \*/

Properties pObj = lib.getPropertiesFileObj();

String url = pObj.getProperty("url");

String username = pObj.getProperty("username");

String password = pObj.getProperty("password");

String customerName = lib.getExcelData("Sheet1", 2, 2);

/\*step 1 : login \*/

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

driver.manage().timeouts().implicitlyWait(20, TimeUnit.***SECONDS***);

driver.get(url);

driver.findElement(By.*name*("username")).sendKeys(username);

driver.findElement(By.*name*("pwd")).sendKeys(password);

driver.findElement(By.*id*("loginButton")).click();

//step 2 : navigate to Task Page

Thread.*sleep*(5000);

driver.findElement(By.*xpath*("//div[text()='TASKS']/..")).click();

//step 3 : navigate to creaTAE customer Page

driver.findElement(By.*xpath*("//div[text()='Add New']")).click();

driver.findElement(By.*xpath*("//div[text()='+ New Customer']")).click();

//step 4 : create Customer

driver.findElement(By.*id*("customerLightBox\_nameField")).sendKeys(customerName);

driver.findElement(By.*xpath*("//span[text()='Create Customer']")).click();

//step 5 :verify

//step 6 : logout

// driver.findElement(By.id("logoutLink")).click();

driver.close();

}

}

data

