Unit Testing Framework Tool

Available Unit testing framework tools in Current Market :

TestNG ----------> Java, .Net [Eclipse / JDevelopers / Intellj]

Junit -------> Java [Eclipse / JDevelopers / Intellj]

Nunit ---------> .Net [Visual Studio]

Pydev --------> Python [PyChram]

Rspsc ----------> Rubby [Eclipse]

Jasmin ➔ javascript [WebStrom]

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

What is testNG ?

➔TestNG is a unit test **TDD**[test Driven Development] framework , which support java & .Net

➔TestNG is an open source unit test framework, where NG stands for **Next Generation**.

➔TestNg developed as addition plugin for Eclipse

➔TestNG is inspired from JUNIT & NUNIT, it means it as all the feature of Nunit & Junit & also contains addition features that makes TestNg become more powerFull

Installation steps of TestNG:

Go to Eclipse window

Click on help option-> Eclipse Marketplace

- ➔ Write TestNG in find edit box and click on go button.
- ➔ Find TestNG for eclipse division and click on **install** button

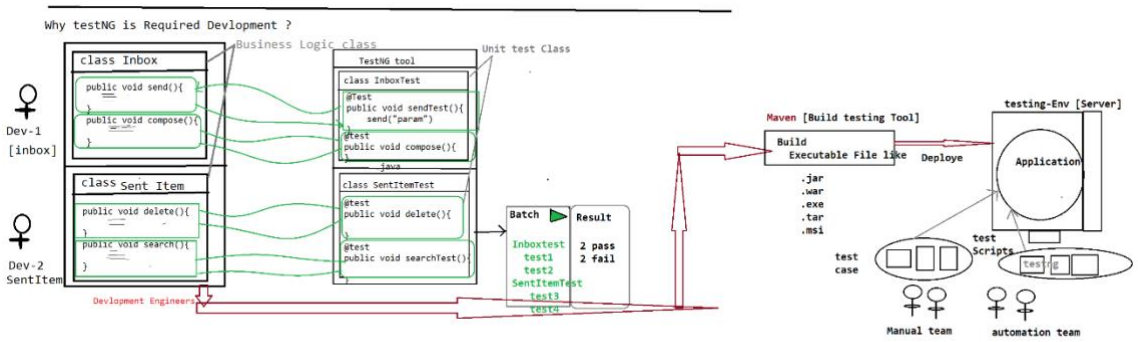
➔ Click on confirm button and I accept the terms and conditions and click on finish

⇨ **In order to verify the** TestNG installation- → Go to windows

→ Show view → others →expand java folder, TestNG symbol will be present

TestNG usage in development:

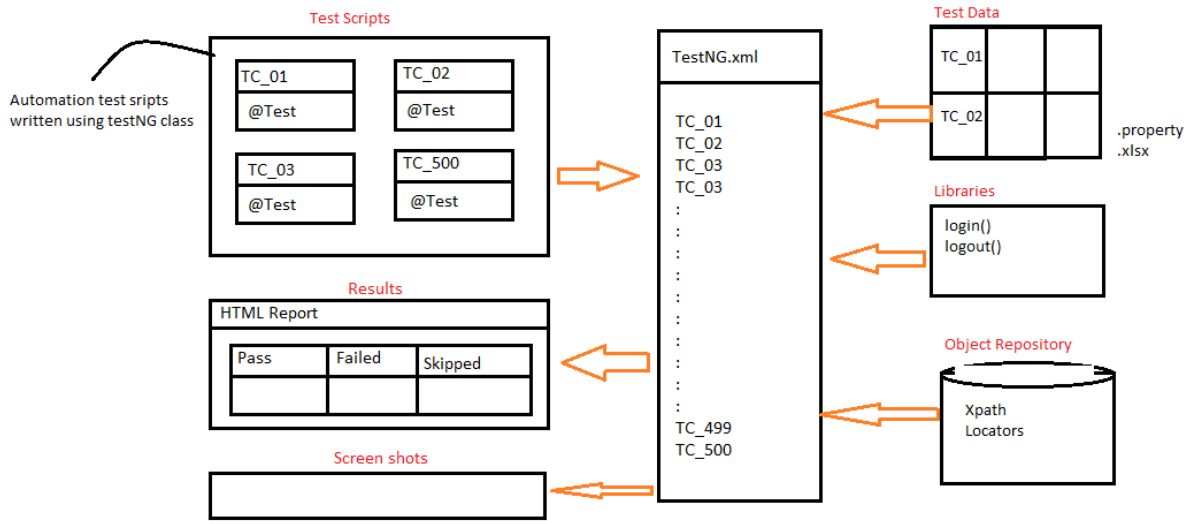
TestNG is used in development to write white box/ unit test cases and each unit test case will be used to test the source code of the application



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

➔Used to achieve WBT

TestNG usage in 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 all framework component & help us to run all the test scripts in batch / parallel / group without any manual intervention

➔TestNG.xml is main controller of the selenium framework , where we start the execution

Why TestNG ? Why not Junit ?

- ➔ **Annotation**
- ➔ **Batch Execution**
- ➔ **Assesrtions**

New functionalities are:

->Html report

->Parallel execution

->Grouping execution

->Additional annotations

->batch execution is easier

-> iTest Listeners [used to take ScrrenShot]

->Retry Analyser [used to rerun the failed test script]

Annotations:

There are 4 blocks in java

1. classs Block
2. Interface Block
3. Enum block
4. Annoation block

```
class A{

}
```

```
interface I{

}
```

```
Enum Key{

}
```

```
Annoation Test{

}
```

Its Java block , which is used to provide metadata(information/instruction) to the JVM , at

the time of execution in RUN-Time.

⇨ Annotation always start with @symbol

➔ @Test

➔ @BeforeMethod

➔ @AfterMethod

➔ @BeforeClass

➔ @AfterClass

-==========addition annotation available only in TestNG

➔ @BeforeTest

➔ @AfterTest

➔ @beforeSuite

➔ @afterSuite

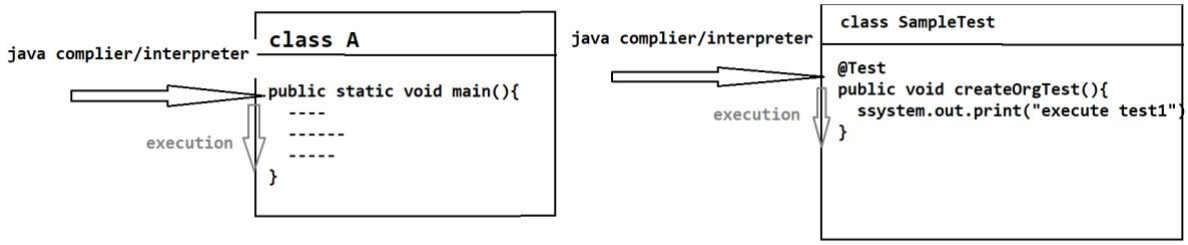
➔ @paramaters

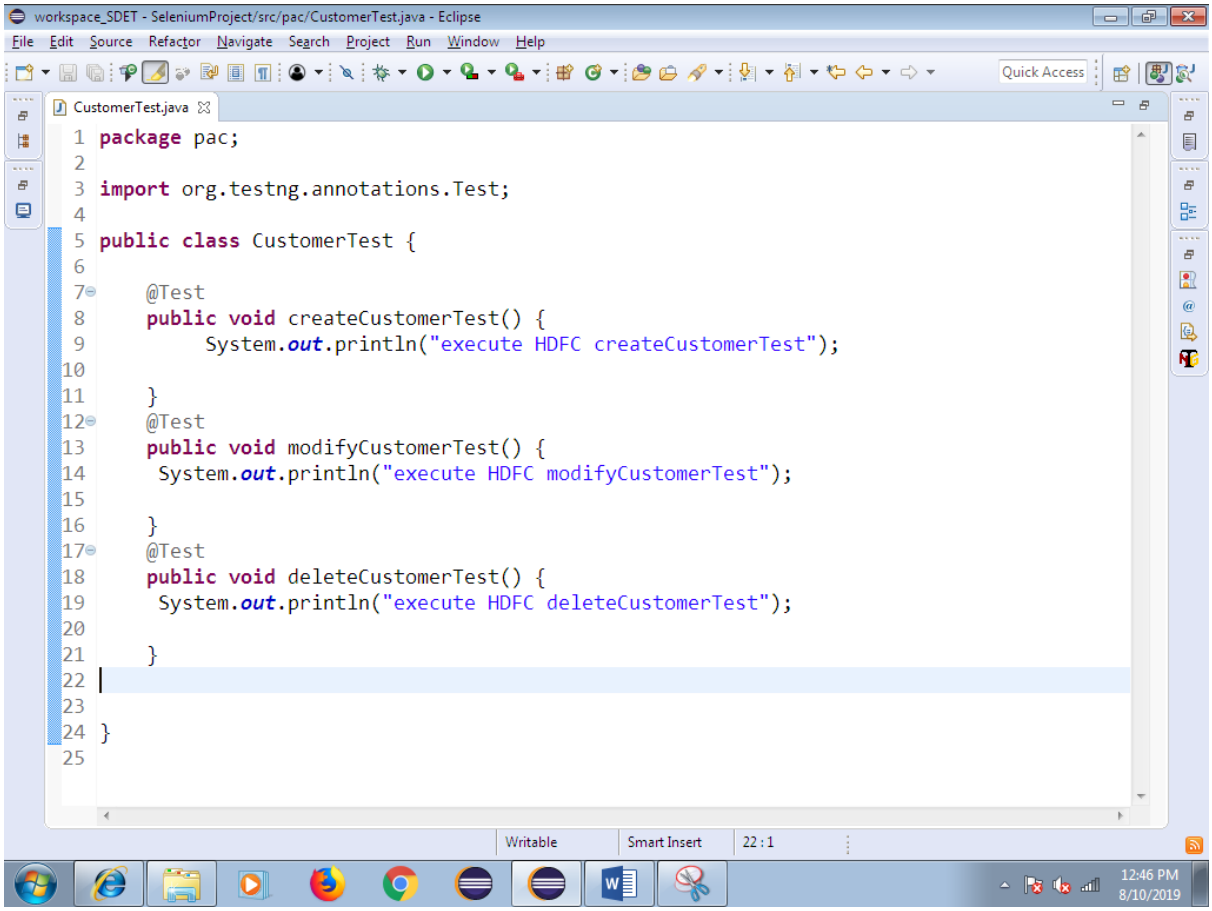
➔ @dataProvider

➔ @Listner

==========

@Test





1. Whenever we execute testng class, javaCompiler/Interpeter always looks for @Test Annotation method to start the execution.
2. **Without @Test , testNG class will not be executed, @test annotation method act like main method in testNG**
3. In one testng class we can have multiple @test methods, but each test method should have @Test annotation before method signature.
4. Annotation method return type should be “void” and access specifier should be public., but method name can be anything (we have provide manual testNameTest).
5. As per the Rule of the Automation , TESTNG class Name should be ModuleNAme , @test method name should be manual testCase Name
6. As per the Rule TestNG class Name & testNG method Name should end With “Test”
7. One Manual test case contains multiple steps all those steps should be automated using one @test annotation & test name should be manual test-case Name & end with Test
8. In one class we can keep multiple @test annotation , but in real time we are going maintain maximum 10 to 15 @test script , because maintenance will be easy

How to Verify html report

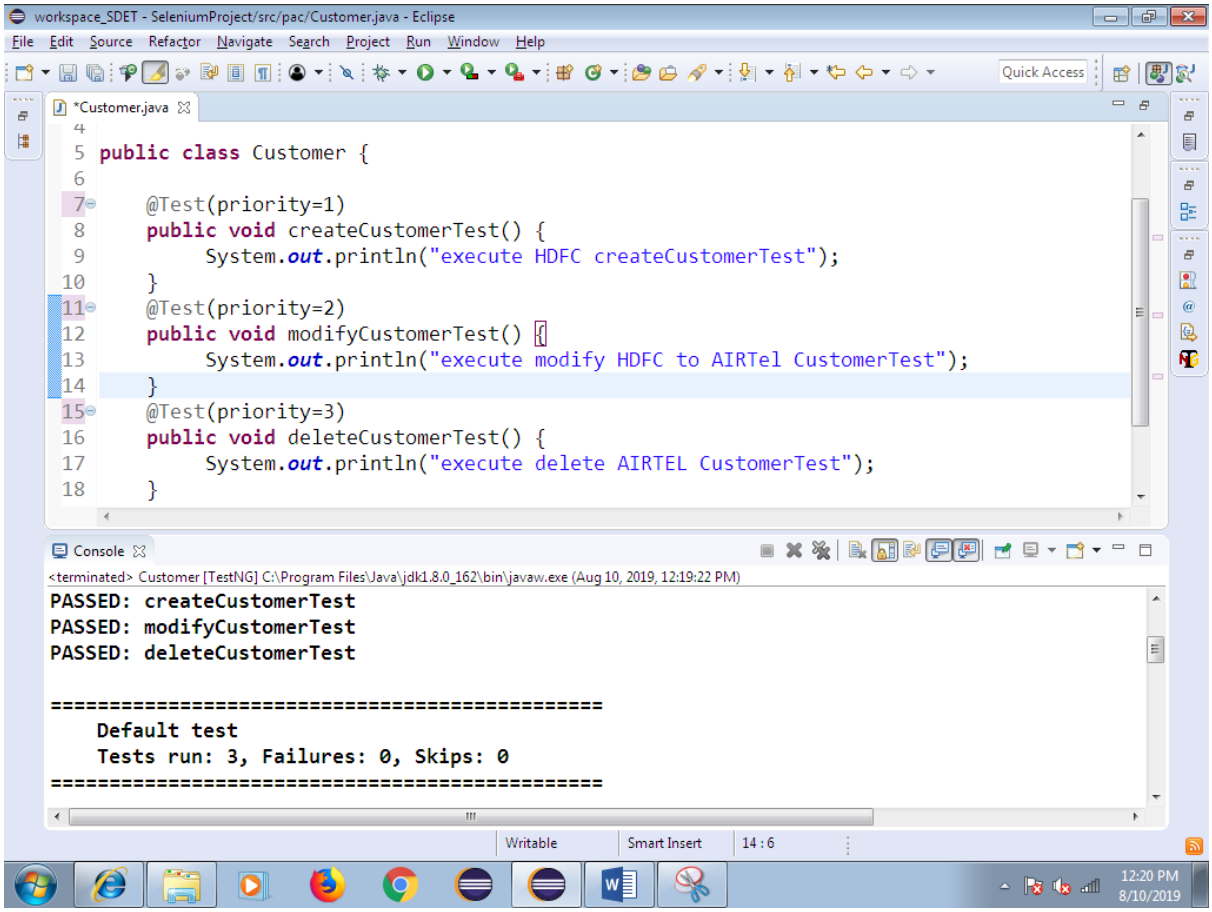
Refresh the project after execution—select project right click and click on refresh

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

Expand test-output folder → select emailable.html and right click → open with→ Open with browser

Priority

Whenever we execute testNG class , bydefault all the test method will be executed based on Alphabetical Order , in order the change the Order of Execution , we go for priority

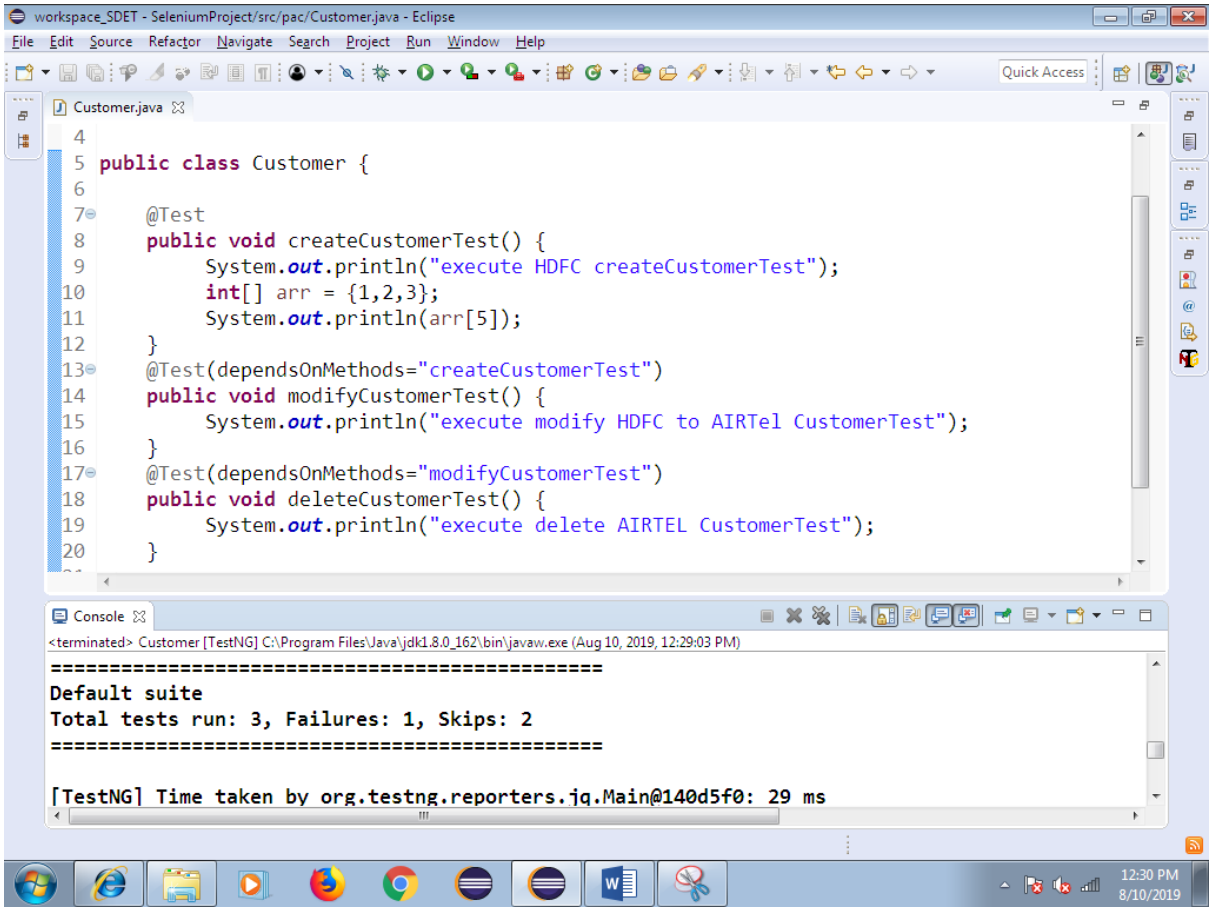


DependsOnMethod :

Its help us to check the dependent test case is pass or fail,

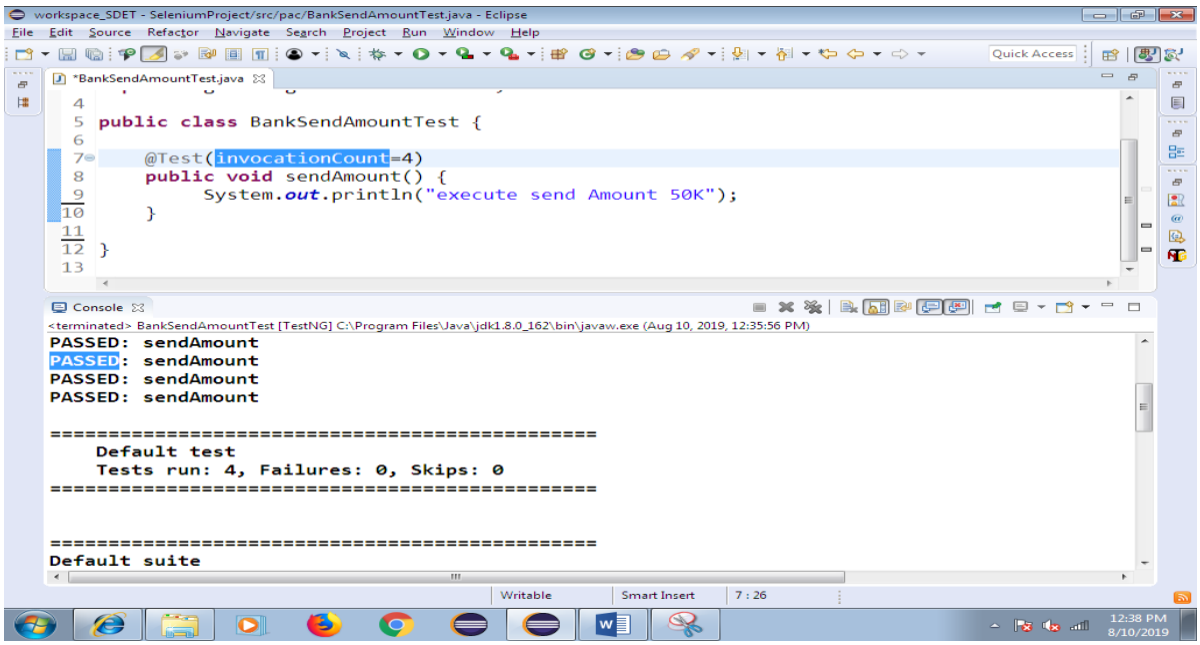
If dependent test-script get pass, execution will continue

If dependent test-script get fail, skip the all other test script which is dependent on first test



10. Invocation Count:

Same test-script executed with multiple Times with same test data



11 . Data Provider

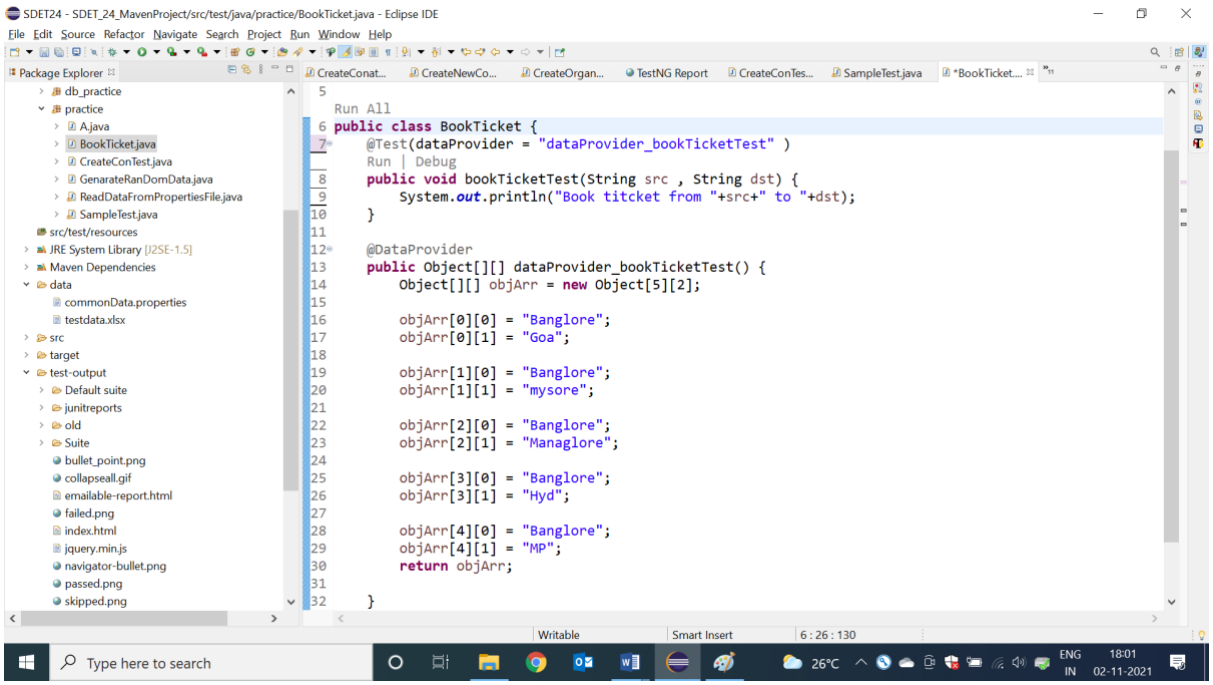
a. In Order to execute same test case multiple Times with different test Data , we go for @DataProvider annotataion
b. Data Provider annotation always return TWO –DIMENSTINAL Object array , because we can pass any type of datatype
c. Data Provider annotation help us to execute same test multiple times with the different set of data , each test-script should have dedicated @dataprovider annotation
d. DataProvider annotation play major role in Data driven framework , where we need to test the application with huge amount of data like Ecommerce , Booking ,banking application
e. In Below example , row count is 5 ➔ it indicates test needs to be executed 5 times

Column count is 2➔ it indicates every iteration 2 arguments will be passed

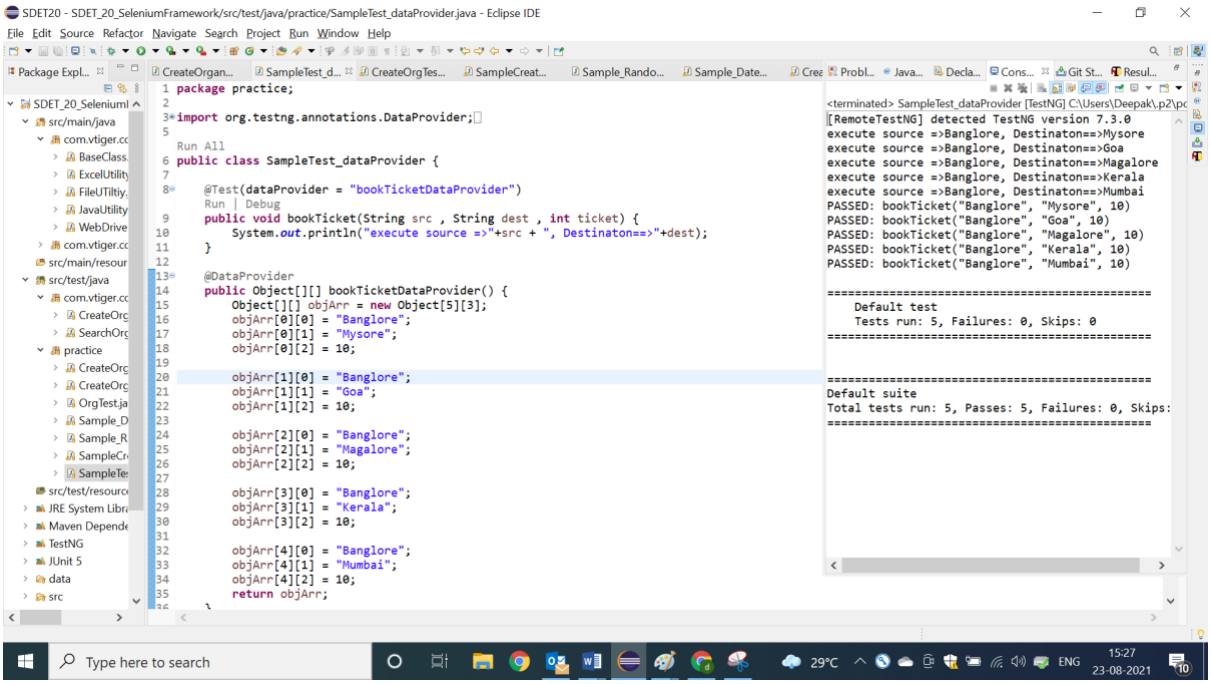
object[5][2]

Banglaore 0,0	Mysore 0,1
Banglore 1,0	Goa 1,1
Banglore 2,0	Mangalore 2,1
3,0 Banglore	3,1 Kerala
Banglore 4,0	Mumbai 4, 1

➔Sample program for data provider

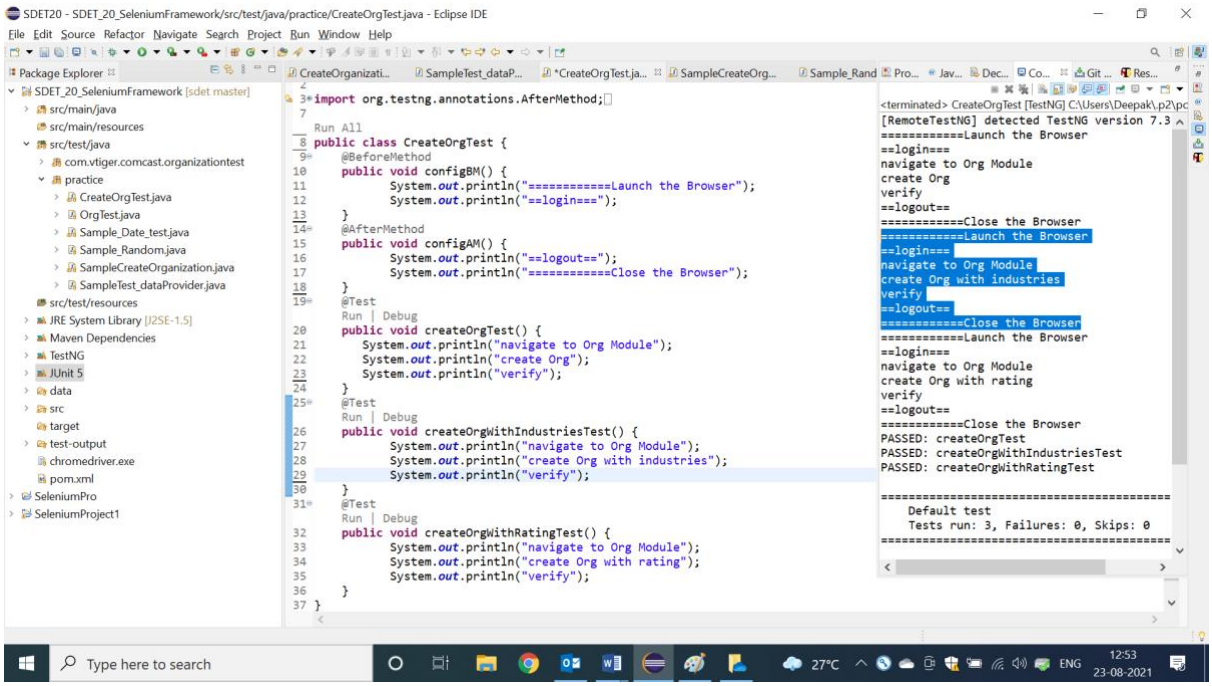


➔Data provider example with 3 arguments

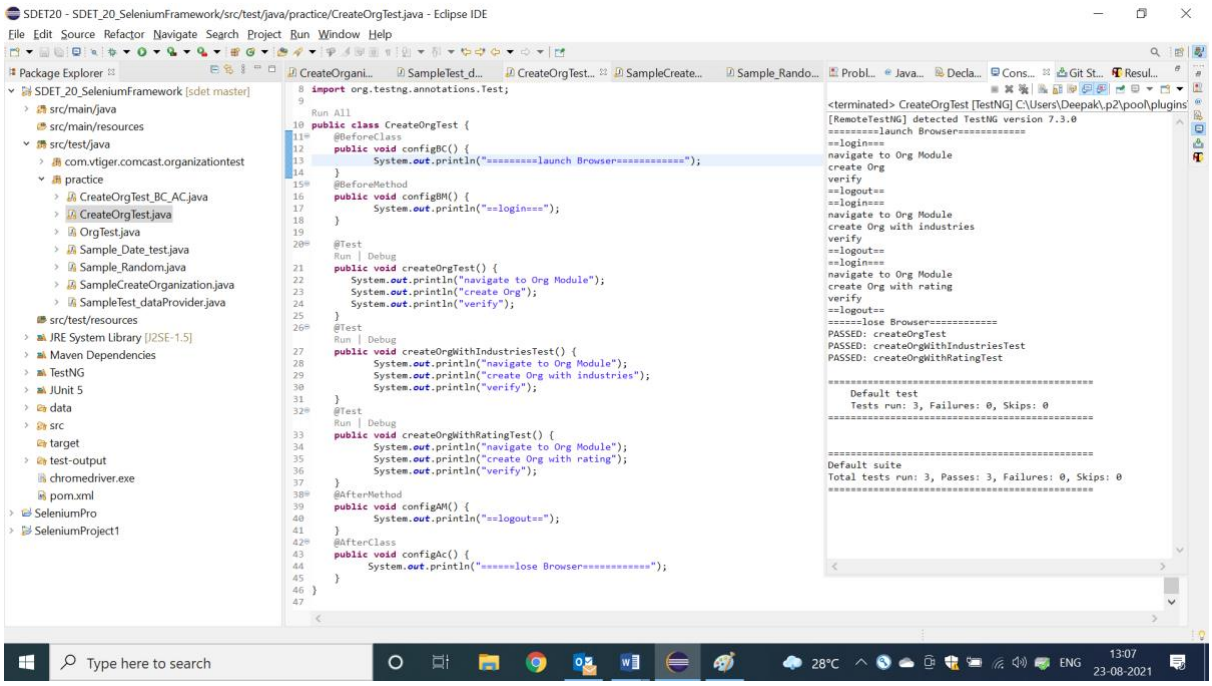


@BeforeMethod @AfterMethod

1. Before method annotations will be executed, before executing each @test method in a class
2. After method annotation will be executed, after executing each @test in a class
3. Beforemethod & aftermethod will not be executed, without @test annotation method, because they are configaration method
4. In order to implement similar pre- condition for all the testcase like “LOGIN code ” we go for @BM
5. In order to implement similar post-condition for all the testcase like “LOGOUT code ” we go for @AM



@BeforeClass & @Afterclass



1. BeforeClass Annotation method will be executed, before Executing first @test in a class
2. AfterClass Annotation method will be executed, after executing all/last test-case with in a class
3. BeforeClass & AfterClass annotations will be executed only once in a entire class execution.
4. It will be used to develop global configuration like launch browser, object initialization
   NOTE: As per the Rule of the Automation , test case should not be have dependency between one to another test , every testcase should be unique (it means every test should have fresh login & logout code)

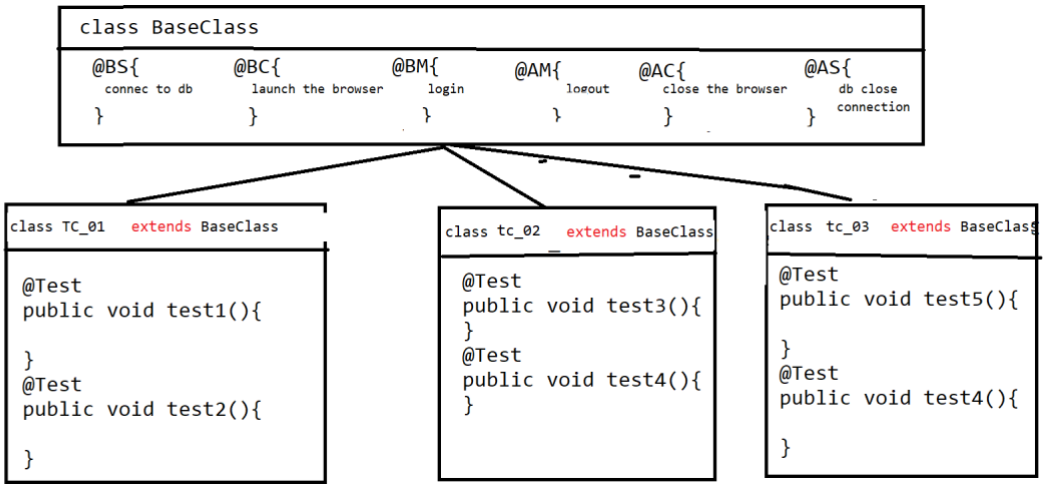
ConFig Annotation methods Usage in REAL Selenium Framework

1. In Real selenium FrameWork , all the configure annotation will be implemented Inside the BaseClass, that is being shared all the Automation engineers via GITGUB
2. BaseClass should be available in generic libraries package.
3. As per the Rule , Every testScripts class should extend BaseClass, so that all the configure annotation will be inherited & executed to the test scripts automatically

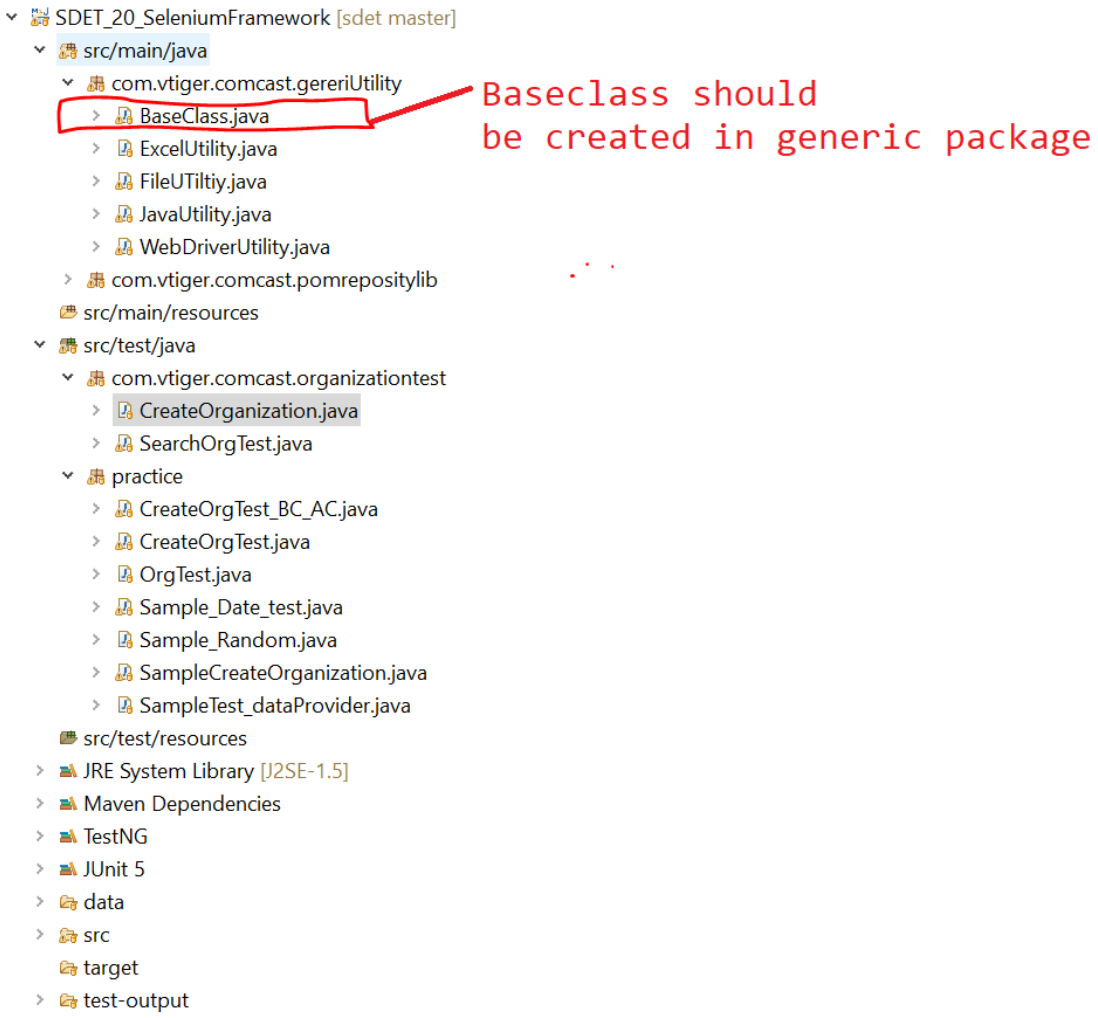
Advantages :

1. Code Reusability
2. Code Optimization
3. Modification is easy
4. Maintenance is easy
5. Test Development is faster
6. Slightly execution time is reduced

EG :



Project Structure in Eclipse



=========Base class Program==========

```
package com.vtiger.comcast.gereriUtility;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;

import com.vtiger.comcast.pomrepositylib.Home;
import com.vtiger.comcast.pomrepositylib.Login;

public class BaseClass {
        public WebDriver driver;
          /*Object Creation for Lib*/
                public JavaUtility jLib = new JavaUtility();
                public WebDriverUtility wLib = new WebDriverUtility();
                public FileUTiltiy fLib = new FileUTiltiy();
                public ExcelUtility eLib = new ExcelUtility();

        @BeforeSuite
        public void configBS() {
```

```
                System.out.println("=======================connect to DB=======================");

        }

        @BeforeClass
        public void configBC() {
                System.out.println("=============Launch the Browser=======");
                driver = new ChromeDriver();
                wLib.waitUntilPageLoad(driver);
                driver.manage().window().maximize();
        }

        @BeforeMethod
        public void configBM() throws Throwable {
                /*common Data*/
                String USERNAME = fLib.getPropertyKeyValue("username");
                String PASSWORD = fLib.getPropertyKeyValue("password");
                String URL = fLib.getPropertyKeyValue("url");
                String BROWSER = fLib.getPropertyKeyValue("browser");
                /* Navigate to app*/
                   driver.get(URL);
                /* step 1 : login */
                Login loginPage = new Login(driver);
                loginPage.loginToApp(USERNAME, PASSWORD);
        }


        @AfterMethod
        public void configAM() {
              /*step 6 : logout*/
                Home homePage = new Home(driver);
        homePage.logout();
        }

        @AfterClass
        public void configAC() {
                System.out.println("=============Close the Browser=======");
                driver.quit();
        }

        @AfterSuite
        public void configAS() {
                System.out.println("=======================close DB=======================");
        }
}
```

==========Sample Test Script code using base class==========

```
package com.vtiger.comcast.organizationtest;

import org.testng.annotations.Test;

import com.vtiger.comcast.gereriUtility.BaseClass;
import com.vtiger.comcast.pomrepositylib.CreateNewOrganization;
import com.vtiger.comcast.pomrepositylib.Home;
import com.vtiger.comcast.pomrepositylib.OrganizationInfo;
import com.vtiger.comcast.pomrepositylib.Organizations;

public class CreateOrganization  extends BaseClass{

        @Test
        public void createOrgTest() throws Throwable {

                int randomInt = jLib.getRanDomNumber();
                /*test script Data*/
                String orgName = eLib.getDataFromExcel("Sheet1", 1, 2) + randomInt;

        /*step 2 : navigate to organization*/
        Home homePage = new Home(driver);
        homePage.getOrganizationLnk().click();
```

```java
/*step 3 : navigate to "create new organization"page by click on "+" image */
Organizations orgPage = new Organizations(driver);
orgPage.getCreateOrgImg().click();

/*step 4 : create organization*/
CreateNewOrganization cno = new CreateNewOrganization(driver);
cno.createOrg(orgName);

/*step 5 : verify the successful msg with org name*/
OrganizationInfo orginfoPage = new OrganizationInfo(driver);
String  actSuccesfullMg =  orginfoPage.getSuccesfullMsg().getText();
if(actSuccesfullMg.contains(orgName)) {
System.out.println(orgName + "==>created successfully");
}else {
System.out.println(orgName + "==> not created successfully");

}
}


@Test
public void createOrgWithIndutriesTest() throws Throwable {
        /*test script Data*/
        int randomInt = jLib.getRanDomNumber();
        String orgName = eLib.getDataFromExcel("Sheet1", 4, 2) + randomInt;
        String industriesType = eLib.getDataFromExcel("Sheet1", 4, 3);
         /*step 2 : navigate to organization*/
Home homePage = new Home(driver);
homePage.getOrganizationLnk().click();

/*step 3 : navigate to "create new organization"page by click on "+" image */
Organizations orgPage = new Organizations(driver);
orgPage.getCreateOrgImg().click();

/*step 4 : create organization*/
CreateNewOrganization cno = new CreateNewOrganization(driver);
cno.createOrg(orgName, industriesType);

/*verify orgname & industry */
OrganizationInfo orginfoPage = new OrganizationInfo(driver);
String  actSuccesfullMg =  orginfoPage.getSuccesfullMsg().getText();
if(actSuccesfullMg.contains(orgName)) {
System.out.println(orgName + "==>created successfully");
}else {
System.out.println(orgName + "==> not created successfully");

}

String actIndustryType = orginfoPage.getIndutryTypeInfo().getText();
if(actIndustryType.equals(industriesType)) {
System.out.println(industriesType + "==>industry is verified successfully");
}else {
System.out.println(industriesType + "==>industry is not verified successfully");

}
}

@Test
public void createOrgWithRatingTest() throws Throwable {
        /*test script Data*/
        int randomInt = jLib.getRanDomNumber();
        String orgName = eLib.getDataFromExcel("Sheet1", 7, 2) + randomInt;
        String rating = eLib.getDataFromExcel("Sheet1", 7, 3);
           /*step 2 : navigate to organization*/
Home homePage = new Home(driver);
homePage.getOrganizationLnk().click();

/*step 3 : navigate to "create new organization"page by click on "+" image */
Organizations orgPage = new Organizations(driver);
orgPage.getCreateOrgImg().click();

/*step 4 : create organization*/
CreateNewOrganization cno = new CreateNewOrganization(driver);
```

```
        cno.createOrg(orgName, rating, true);

        /*verify orgname & industry */
        OrganizationInfo orginfoPage = new OrganizationInfo(driver);
        String  actSuccesfullMg =  orginfoPage.getSuccesfullMsg().getText();
        if(actSuccesfullMg.contains(orgName)) {
        System.out.println(orgName + "==>created successfully");
        }else {
        System.out.println(orgName + "==> not created successfully");

        }
        String actRatingType = orginfoPage.getRatingTypeInfo().getText();

        if(actRatingType.equals(rating)) {
        System.out.println(rating + "==>industry is verified successfully");
        }else {
        System.out.println(rating + "==>industry is not verified successfully");

        }
        }
}
```

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 configuration file

> TestNG xml file always start with suite xml tag fallowed by <test> and <classes.>

> In one xml file we can invoke N-number of testng classes, but all the classes should be present within a project.

> All the class should fallowed by packageNAme

EG :

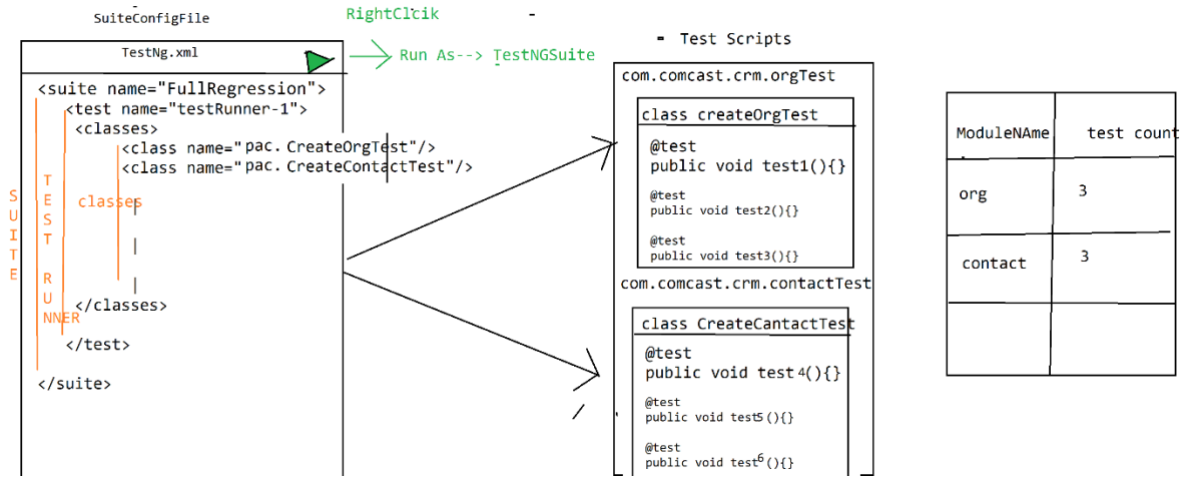
<class name=”com.comcast.orgtest.CreateOrg”></class>

How to create testNG xml file automatically through eclipse?

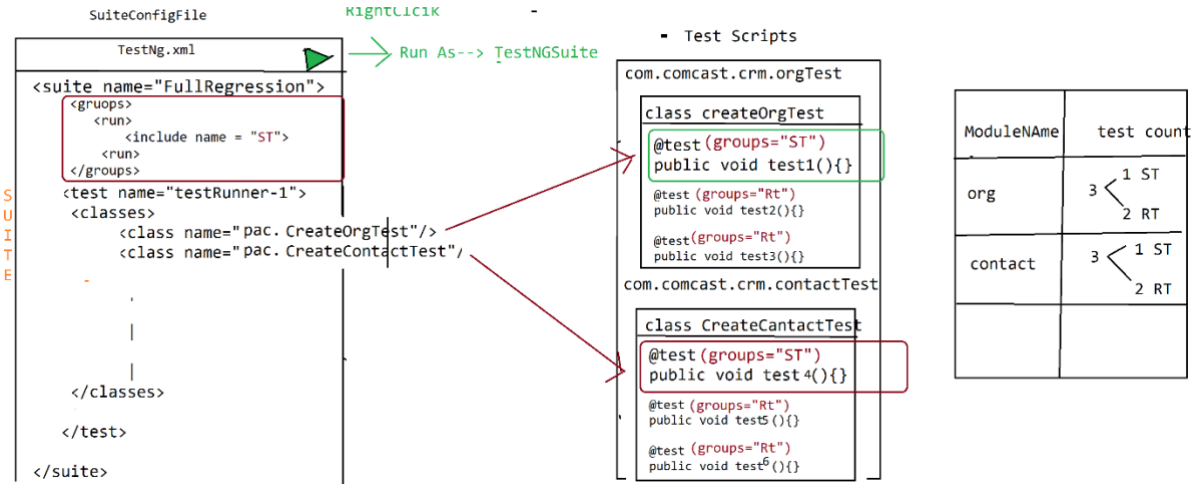
Select all the testNG classes or packages -> right click-> select➔ testNG ➔and click on “Convert to testing” and➔ click on finish.

⇨ Automatically you will get the testng.xml file with in the project
⇨ In order to edit xml File➔ double click on testing.xml➔ click on “Source”

Batch Execution : [Full regression testing]



Grouping Execution:



- ➔ 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 name will be written along with annotation
- ➔ In grouping execution, all configure annotation should have group name , other wise those annotation will not participate in grouping execution like @BeforeSuite @BeforeClass , @BeforeMethode etc
  EG :

```
@BeforeSuite(groups = {"smokeTest","regressionTest"})
public void configBS() {
        System.out.println("======Execute BeforeSuite=============");
}
```

Smoke Test:

```
@Test(groups={"smokeTest"})
public void createCustomerTest()
{
        System.out.println("execute createCustomerTest");
```

```
        }

    @Test(groups={"regressionTest")
        public void modifyCustomerTest()
    {
            System.out.println("execute modifyCustomerTest");
    }
```

➔ In order to invoke grouping execution should declare Group Key in testing.xml file & group keep should be declared before <test>, after <suite> tag

Smoke Test:

```
<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>
```

➔We can invoke multiple groupkey in one XML File

```
<suite name="Suite">
    <groups>
        <run>
            <include name="smokeTest"/>
            <include name="regressionTest"/>
        </run>
    </groups>
 <test name="Test">
  <classes>
   <class name="pac1.ProjectAndCustomerTest"/>
    <class name="pac2.ReportTest"/>

  </classes>
 </test>
```

➔One test can have multiple Group name

```
@Test(groups={"regressionTest","smokeTest"})
    public void modifyCustomerTest()
{
            System.out.println("execute modifyCustomerTest");
}
```

Regional Regression Test:

➔To execute particular test cases across the Suite is called regional regression testing

➔in Real time , impact area is given by developer / test lead , based on that idea XML will be created

➔ Whenever we want execute particular @test method inside class , we go for <method> & <include>

```
<?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="com.comcast.crm.contacttest.CreateContactTest">
            <methods>
               <include name="createdContactTest"/>
            </methods>
      </class>
      <class name="com.comcast.crm.orgtest.CreateOrgTest">
            <methods>
               <include name="createdOrgWithRatingTest"/>
            </methods>
      </class>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Parallel Execution:

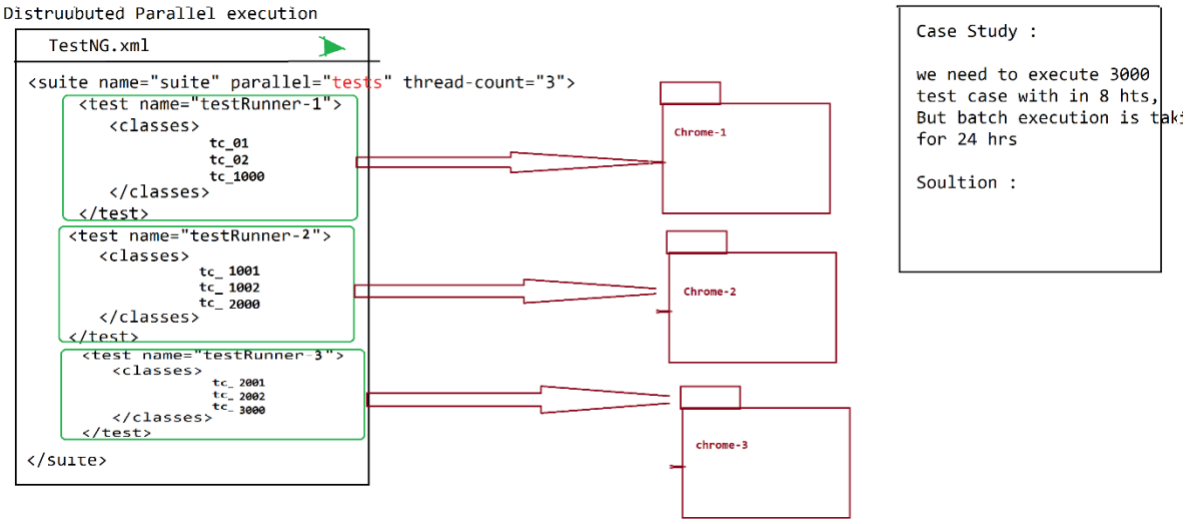
There are 3 types of Parallel Execution

1. **Distributed Parallel execution:**
2. **Cross browser parallel execution**
3. **Cross platform parallel execution**

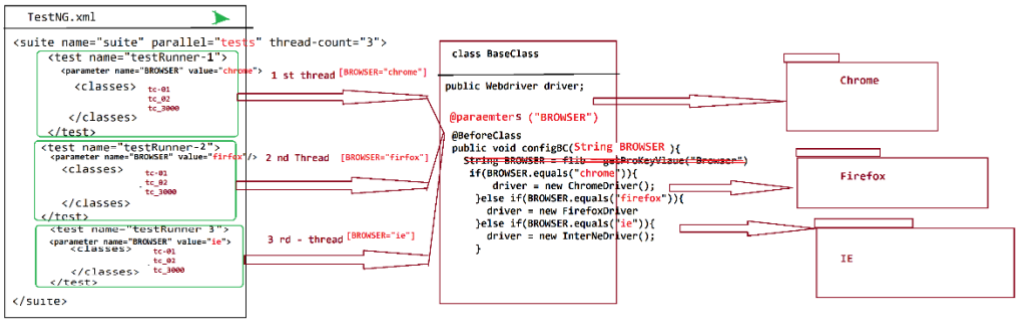
1. **Distributed Parallel execution:**

➔Distribute the test case across the multiple test runner & execute, each <test> Test runner in parallel is called Distributed Parallel execution

- ⇨ we reduce the suite execution time, so that we can get the result early
- ⇨ in order the archive parallel execution we should enable parallel=”tests” & thread-count=5 in <suite> , then create multiple test runner & distribute the testcase
- ⇨ maximum thread count is 5
- ⇨ Thread count should be same as number of <test> testRunner



2 Cross browser parallel execution / Browser Compatibility testing



1. Execute same set of testcase in different Browser parallel is called cross browser testing
2. To achieve cross browser testing we should we <parameter> in XML file & @parameters annotation inside the testScript

3. **<parameter > is used to specify the browser data for each <test>**

EG :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel= "tests" thread-count="2">
  <test name="Test-Runner-Chrome">
   <parameter name="BROWSER" value="chrome"/>
    <classes>
       <class name="com.vtiger.comcast.organizationtest.CreateOrganization"/>
      <class name="com.vtiger.comcast.organizationtest.SearchOrgTest"/>

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

  <test name="Test-Runner-Firefox">
  <parameter name="BROWSER" value="firefox"/>
    <classes>
    <class name="com.vtiger.comcast.organizationtest.CreateOrganization"/>
      <class name="com.vtiger.comcast.organizationtest.SearchOrgTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

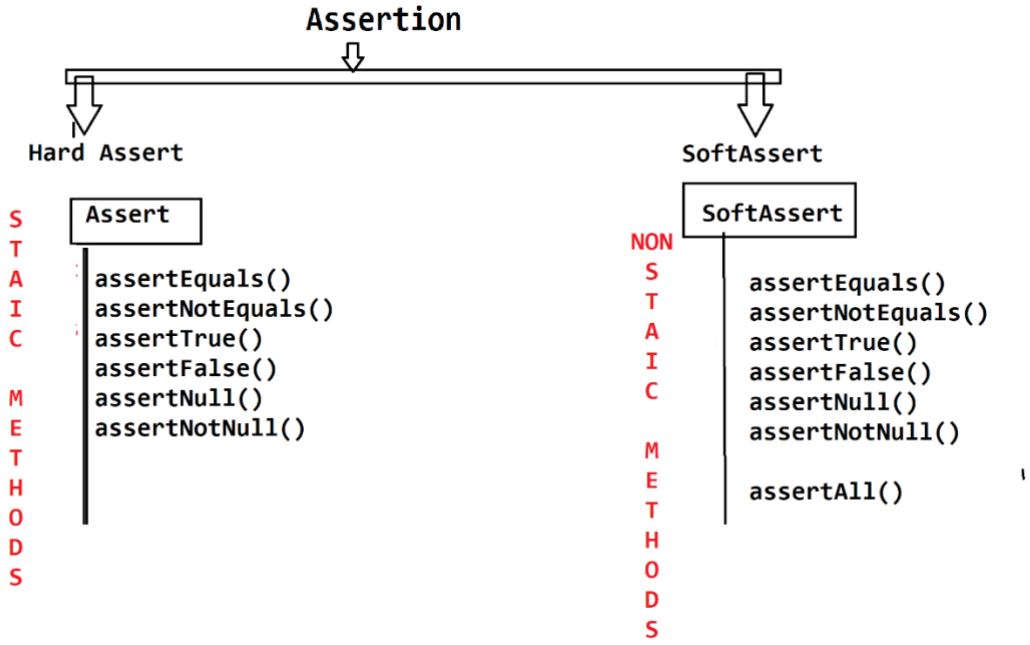
4. @parameters annotation will be used to receive the data (Eg : browser , platform etc) from the XML file to test Scripts [Like BaseClass]

Note : Replace below annotation inside the balseclass

```
@Parameters("BROWSER")
        @BeforeClass
        public void configBC(String BROWSER) {
                System.out.println("=============Launch the Browser=======");
                if(BROWSER.equals("chrome")) {
                    driver = new ChromeDriver();
                }else if(BROWSER.equals("firefox")) {
                        driver = new FirefoxDriver();
                }else if(BROWSER.equals("ie")) {
                        driver = new InternetExplorerDriver();
                }else {
                         driver = new ChromeDriver();
                }
                wLib.waitUntilPageLoad(driver);
                driver.manage().window().maximize();
        }
```

Assertion / CheckPoint

- **Assertion is a feature available in TestNG used to validate test scripts expected results**
- **As per the Rule of the automation every expected result should be verified with Assert statements, because java "if else "statement will not have capability to fail the testNG test Scripts**
- **There are 2 types of Assertions in TESTNG**



Hard Assertion	Soft Assertion
All methods are static in nature	All methods are non-static in nature
It does not allow further execution of test if the line containing hard assert gets failed.	Next steps would be executed even if the line containing soft assertion gets failed.
Whole test case gets failed if at least 1 hard assert fails.	AssertAll() extra lines of code are required to track the fail status.
To verify mandatory fields we go for hard assert	To verify non mandatory fields we go for soft assert

HardAssert

Whenever hardAssert method fails, testNG generate AssertError exception & stop the current test execution & continue 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");
	}
```

Out Put

```
step_1
step_2
============================
step_1
step_2
step_3
PASSED: modifyCustomerTest
FAILED: createCustomerTest
java.lang.AssertionError: expected [B] but found [A]
```

Soft Assert

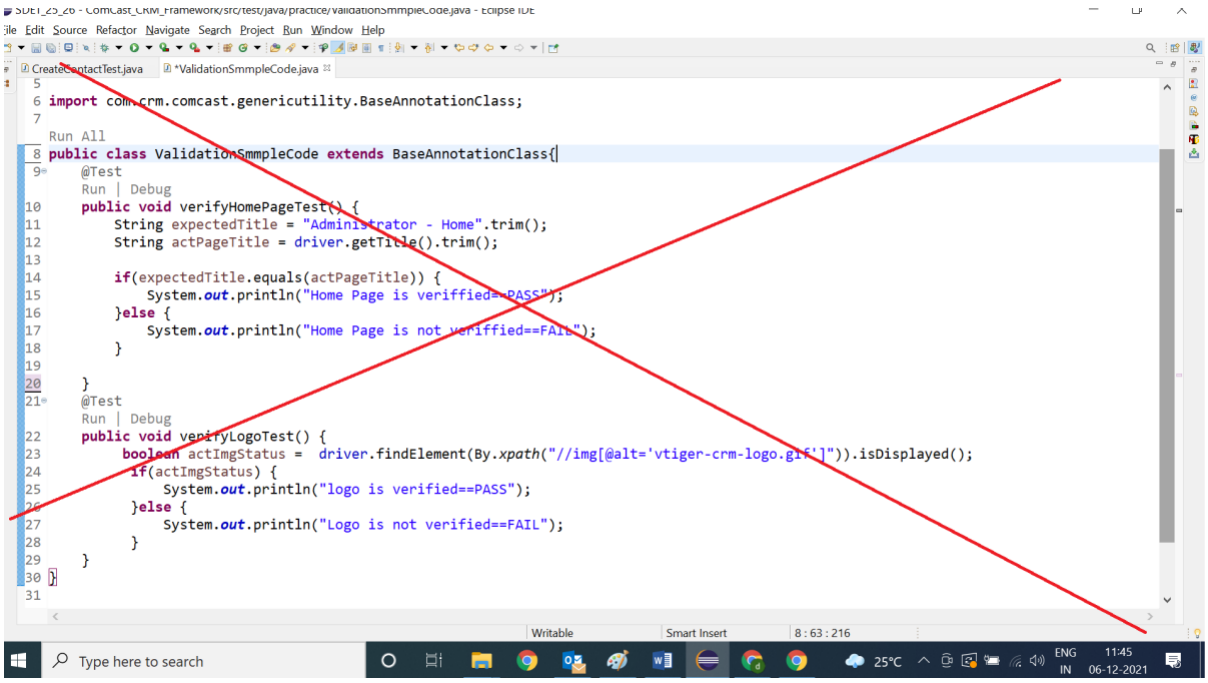
Whenever softAssert mtd fails , testNG Generate AssertError exception & continue execution with remaining steps of same testScript

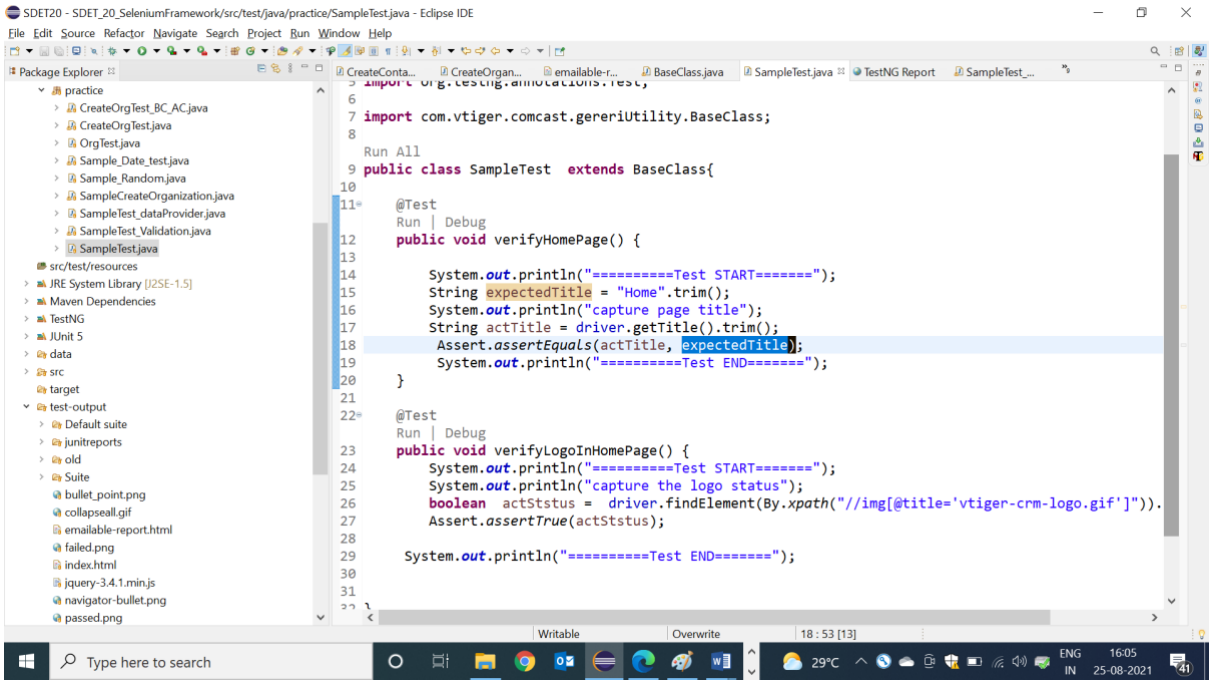
```
	public void createCustomerTest(){
		System.out.println("step_1");
		System.out.println("step_2");
		SoftAssert s = new SoftAssert();
		s.assertEquals("A", "B");
		System.out.println("step_3");
		s.assertEquals("X", "Y");
		System.out.println("step_4");
		s.assertAll();
	}
	@Test
	public void modifyCustomerTest(){
		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]

Assertion usage in Real Time

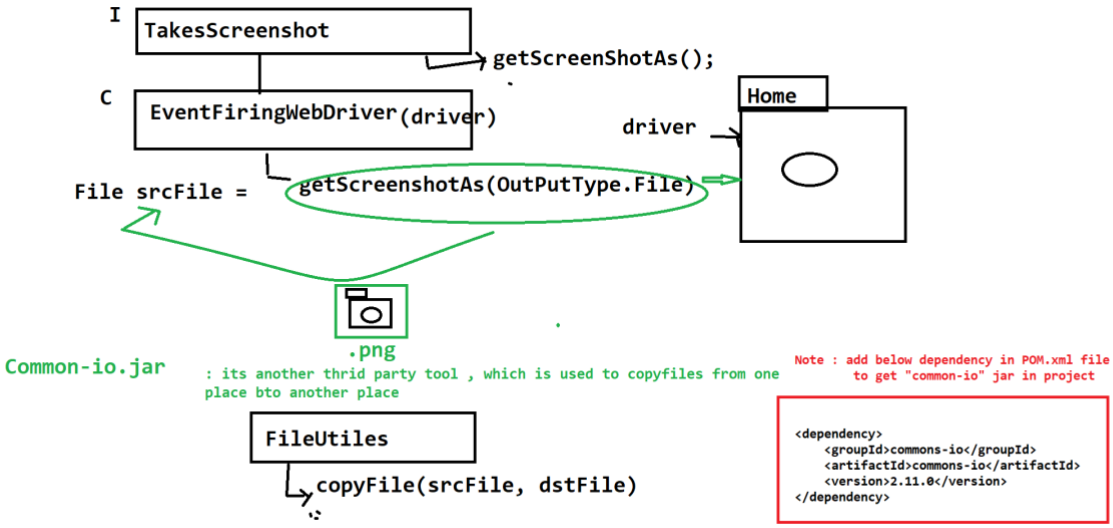




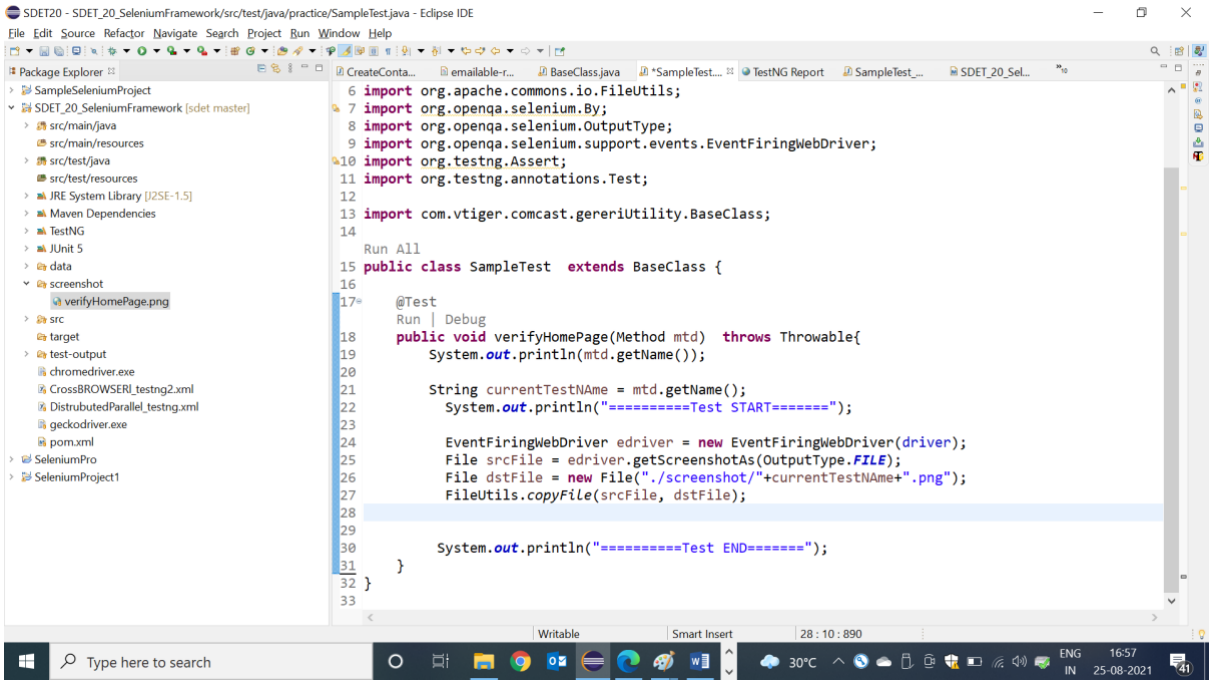
Advantages of Assertion:

- ⇨ It's used to fail the TESTNG test scripts
- ⇨ It's used for test scripts validation
- ⇨ It's generate "AssertErrorException" & reason of the failure + failed line number whenever test is failed
- ⇨ We can compare any 2 primitive variable or array or Collection or MAP in single line

Screen Shot

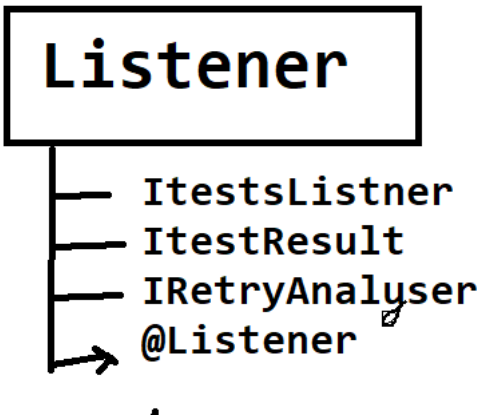


WebDriver Code to take a Screenshot



Listener

- Listen is feature available in TESTNG , which is used to capture runtime events during execution & perform appropriate action based on eventtype



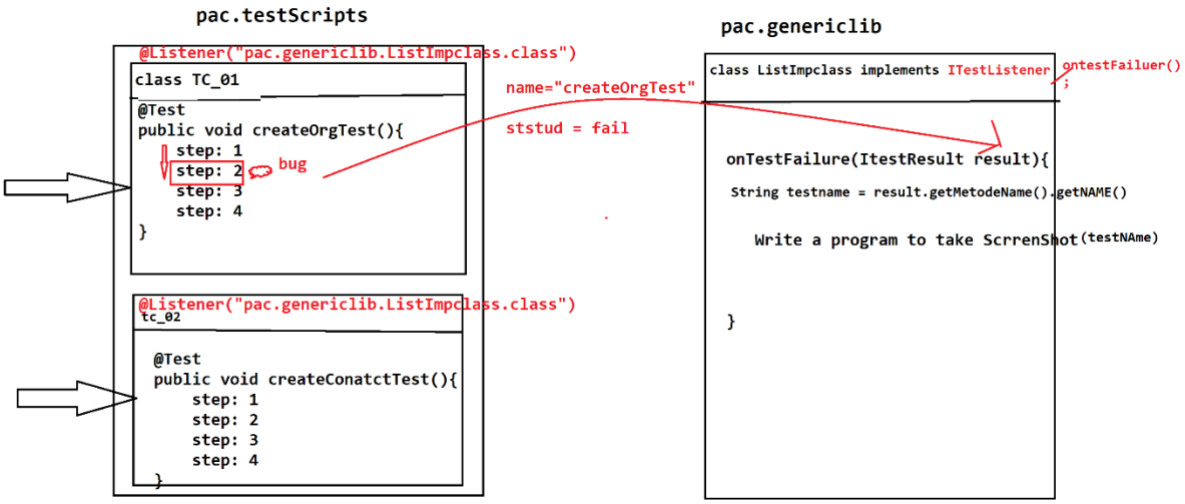
ITestListener:

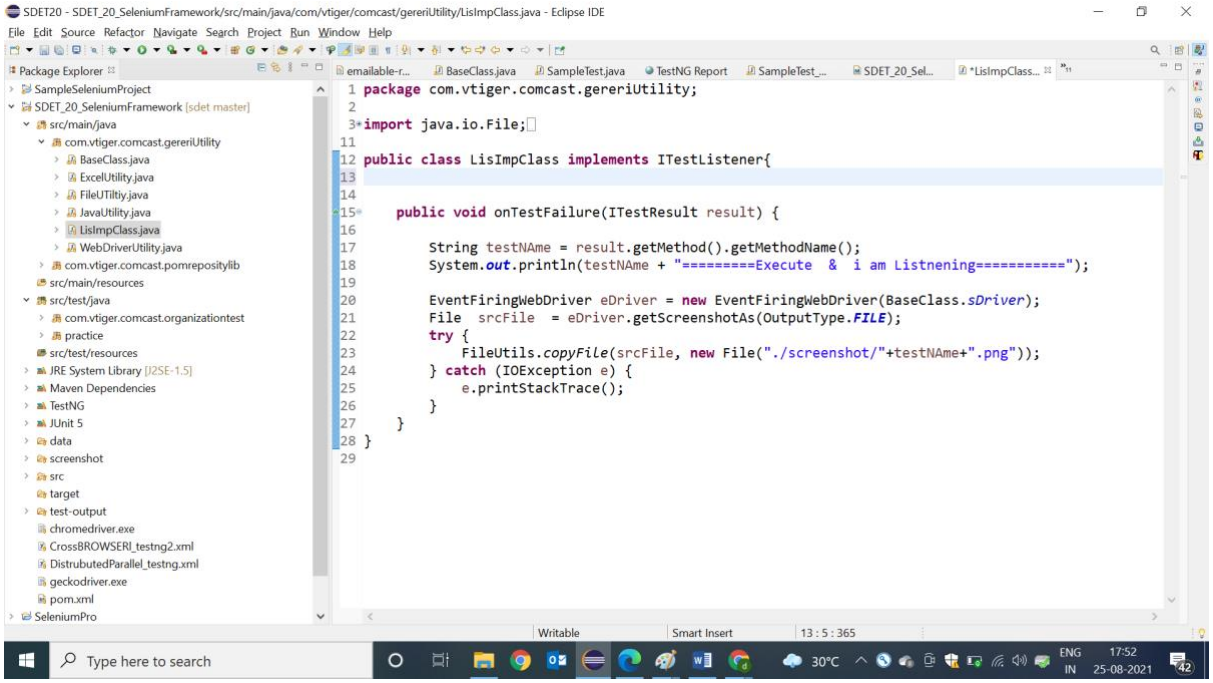
- This is a special feature in testNG which enables the user to take run time events whenever the test scripts fail/pass. Implementation class for ITestListener is mandatory to use Listener feature
- @Listener is testing annotation , which is used to monitor the test execution in the runtime & generate a runtime event to Listener Implementation class , if test is pass / fail
- @Listener annotation will be declared in every testScripts class before class definition block
- ListImplemenation class helps us to receive the failure events from the @Listener & perform appropriate actions

Advantages of Listener :

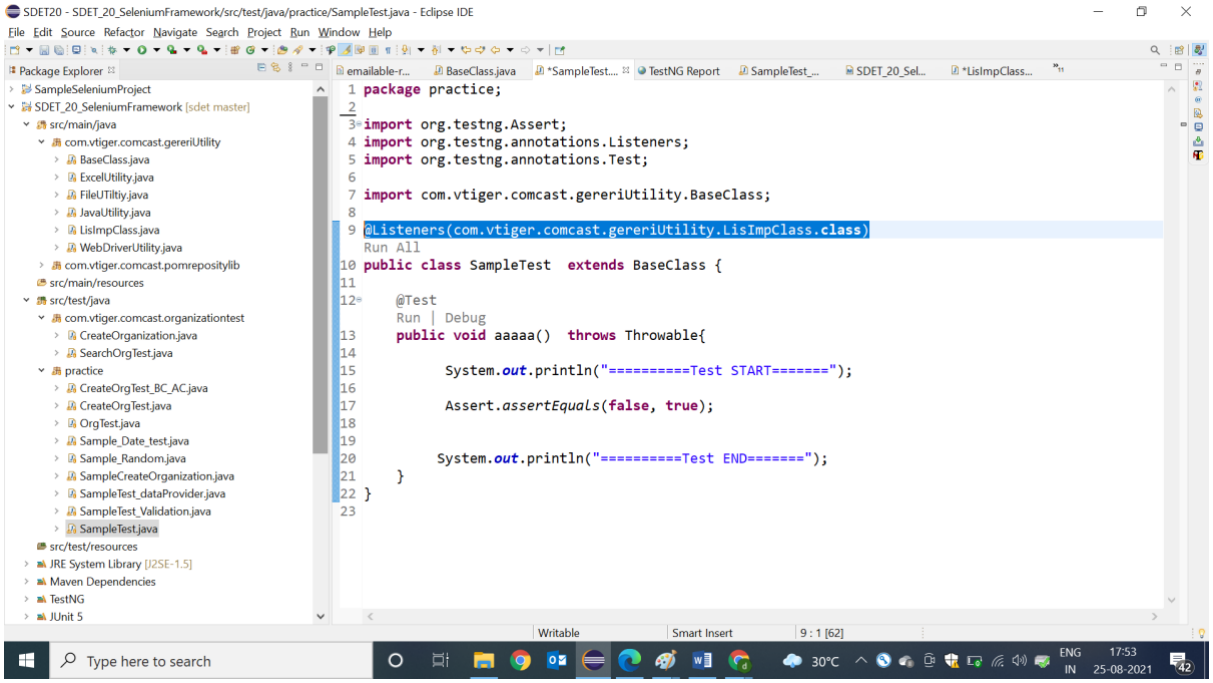
- We can use Listener to take a screenshot for the failed test case, when we execute in bulk
- We can also use Listener for connect to DB , Launch browser & login precondition program
- We can also use Listener for Extend Report Configuration

Listener Implementation class :

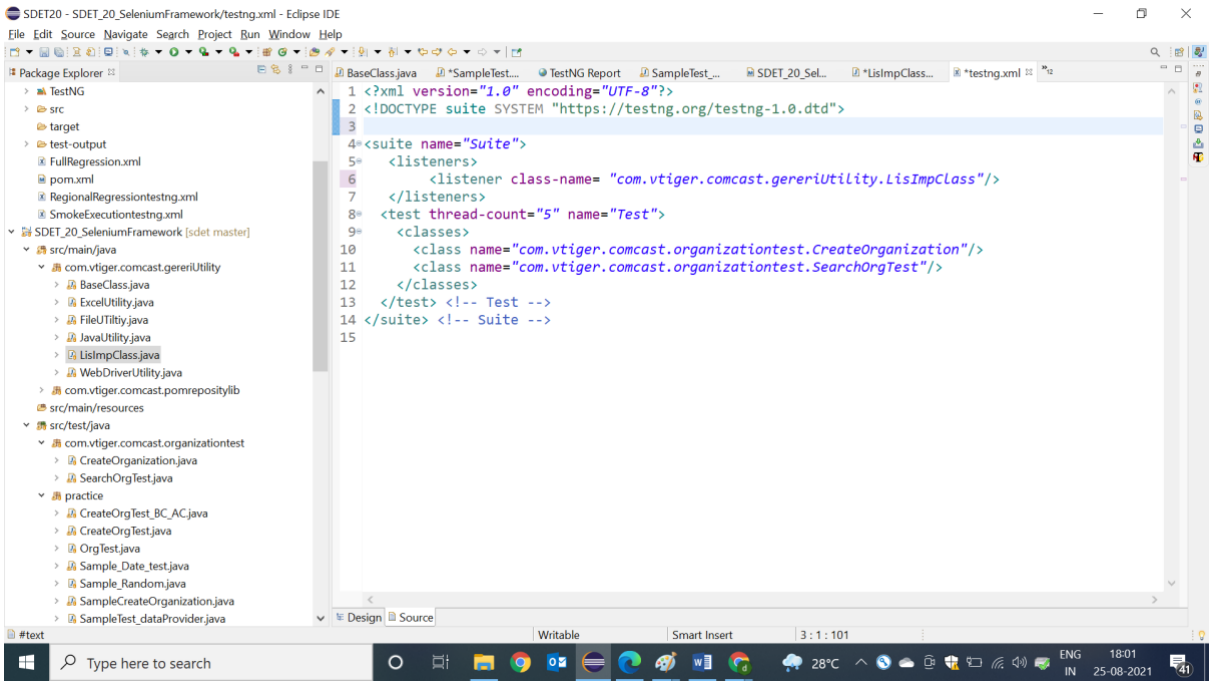




Sample test for Listener :

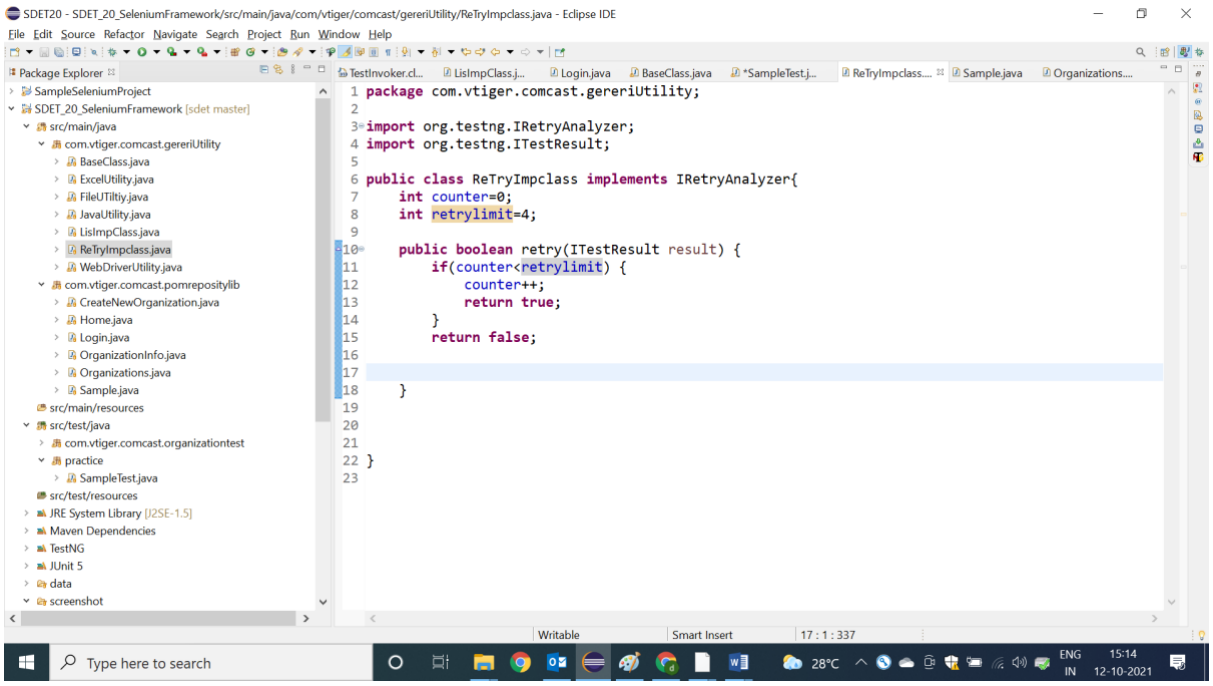


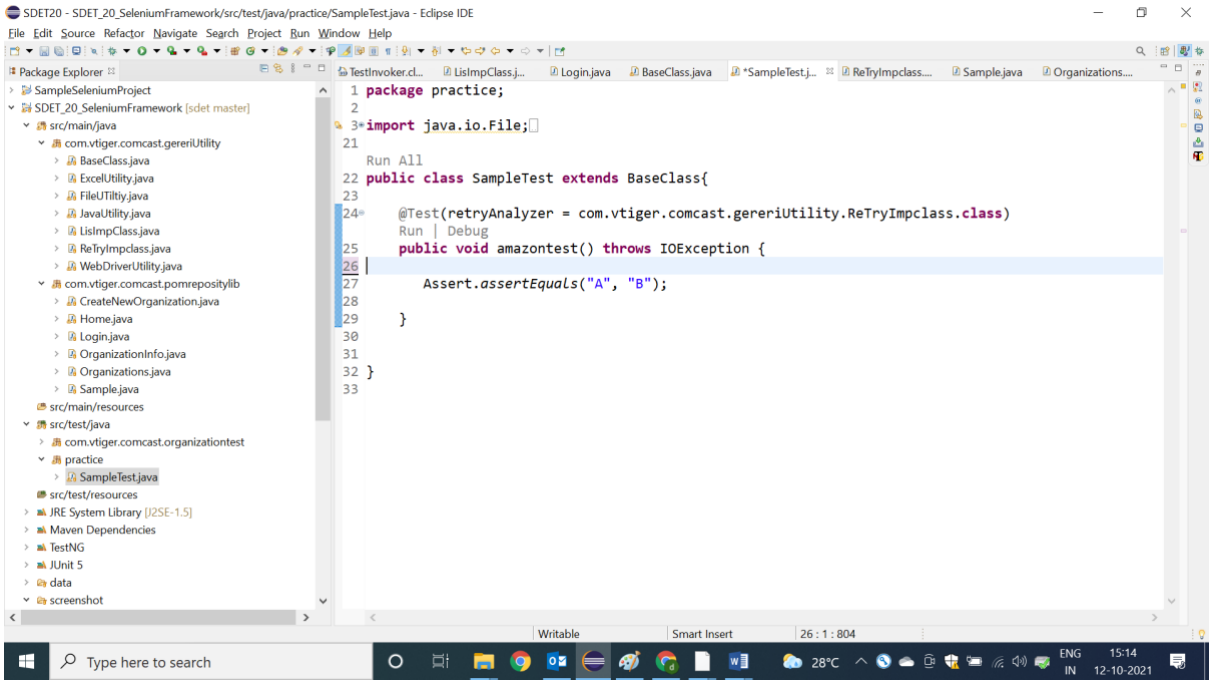
How to use Listener in TestNG.xml



RetryAnalyzer:

- ⇨ This feature of TestNG helps the user to rerun the test script when ever test is getting failed
- ⇨ In order to use this feature we have to implements RetryAnalyzer interface & override retry method
- ⇨ Inside the retry method, we should specify the upper limit so that test script will get executed specified limit when test is getting failed.





When ever we execute 1000 test scripts in batch, out of 1000 / 100 test scripts got failed , next time when a get new build , I wanted execute only failed test scripts , then is your approach ?

After the batach execution, refresh the project & go to test-output folder which is created inside the projectFolder , then execute “test-failed.xml” file

- ⇨ Testng-failed.xml file is created automatically by testing-tool itself for every failed execution reports .