Now test the class FirstTest:

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

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

<suite name=*"DemoSuite"*>

<test name=*"FirstTest"*>

<classes>

<class name=*"com.psr.test.FirstTest"*>

<methods>

<include name=*"TestGoogle"* />

<include name=*"TestFacebook"* />

</methods>

</class>

</classes>

</test>

</suite>

Now test the both classes FirstTest and OrangeHRM:

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

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

<suite name=*"DemoSuite"*>

<test name=*"FirstTest"*>

<classes>

<class name=*"com.psr.test.FirstTest"*>

<methods>

<include name=*"TestGoogle"* />

<include name=*"TestFacebook"* />

</methods>

</class>

</classes>

</test>

<test name=*"OrangeHRM"*>

<classes>

<class name=*"com.psr.test.OrangeHRM"*>

<methods>

<include name=*"navigatetoMyinfo"* />

</methods>

</class>

</classes>

</test>

</suite>

Now run the xml file using TestNG suite.

Parameters in TestNG:  
TestNG parameters are the arguments that we pass to the test methods.  
There are two ways to pass the parameters to the test methods.

1.TestNG parameters.  
2.TestNG DataProviders.

How to use:  
 Suppose we want to set the global variables such as URL settings, username, password or API keys, there are some values which are constant in all the test cases, in such case we use the TestNG parameters.

TestNG parameters are present in XML file. They can be applied either inside the tag or tag.

If we want to apply the parameters to all the test cases, then the parameters are applied inside the tag. If the parameters is specific to a particular folder, then the parameter is applied within a tag.

Passing the value using parameter:

Step 1: Create one class

**package** practise;

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

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

**public** **class** ParameterizedTest {

@Test

@Parameters("myName")

**public** **void** paramterTest(String myName) {

System.***out***.println("Parameterized value is: " + myName);

}

}

Step 2:create one xml file and execute

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

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

<suite name=*"Suite1"*>

<test name=*"Test1"*>

<parameter name=*"myName"* value=*"Partha Saradhi"* />

<classes>

<class name=*"practise.ParameterizedTest"* />

</classes>

</test>

</suite>

Step 3: Execute as TestNG Suite.

Parameters support:  
String

int/Integer

boolean/Boolean

byte/Byte

char/Character

double/Double

float/Float

long/Long

short/Short

Passing the value using Data provider:

Step 1: Create one class in java file

**package** normal;

**public** **class** PrimeNumberChecker {

**public** Boolean validate(**final** Integer PrimeNumber) {

**for**(**int** i = 2; i < (PrimeNumber / 2); i++) {

**if**(PrimeNumber % i == 0) {

**return** **false**;

}

}

**return** **true**;

}

}

Step 2: Create one test/java class

**package** practise;

**import** org.testng.Assert;

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

**import** org.testng.annotations.DataProvider;

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

**import** normal.PrimeNumberChecker;

**public** **class** ParameterWithDataP {

**private** PrimeNumberChecker primeNumberChecker;

@BeforeMethod

**public** **void** initialize() {

primeNumberChecker = **new** PrimeNumberChecker();

}

@DataProvider(name = "test1")

**public** **static** Object[][] primeNumbers() {

**return** **new** Object[][] {{2, **true**}, {6, **false**}, {19, **true**}, {22, **false**}, {23, **true**}};

}

@Test(dataProvider = "test1")

**public** **void** testPrimeNumberChecker(Integer inputNumber, Boolean expectedResult) {

System.***out***.println(inputNumber + " " + expectedResult);

Assert.*assertEquals*(expectedResult, primeNumberChecker.validate(inputNumber));

}

}

Step 3: Create one xml file

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

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

<suite name=*"Suite1"*>

<test name=*"Test1"*>

<classes>

<class name=*"practise.ParameterWithDataP"* />

</classes>

</test>

</suite>

Step 4: Execute as TestNG Suite.

How to enable and disable test cases in xml file:

To disable testcase:

<test name=*"Test1"* enabled=*"false"*>

To enable testcase:

<test name=*"Test1"* enabled=*"true"*>

How to Prioritize your tests in TestNG:

1. What is priority. Why we need to use the priority.  
   Priority is a parameter inside the test annotation.

To execute test method in a certain order we need priority.

1. How do we define the priority.  
   **public** **class** NewTest {

@Test(priority=1)

**public** **void** f1() {

System.***out***.println("f1");

}

@Test(priority=2)

**public** **void** f2() {

System.***out***.println("f2");

}

1. What is the default value to the priority.  
   Zero
2. Can we pass the negative values to prioritize the tests.  
   Yes
3. What happens if we pass same priority value to the multiple methods.  
   It follows the ASCII value order to execute.

How to ignore tests in TestNG:

Step1: Create two TestNG classes in a package and this must be in test/java.

Step2: Add @Test methods in that classes.

Step3: Create one XML file and execute those classes.

Step4: To ignore a method in a class we need to put @Ignore annotation in the method level.

**package** com.psr.test;

**import** org.testng.annotations.Ignore;

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

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

@Ignore

@Test

**public** **void** test1() {

System.***out***.println("test1");

}

@Test

**public** **void** test2() {

System.***out***.println("test2");

}

@Test

**public** **void** test3() {

/;.///. System.***out***.println("test3");

}

}

If we put the ignore annotation in the method level, then that method is not going to be executed.

Step5: To ignore classes in a package, put ignore in the class level.  
**package** com.psr.test;

**import** org.testng.annotations.Ignore;

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

@Ignore

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

@Test

**public** **void** test1() {

System.***out***.println("test1");

}

@Test

**public** **void** test2() {

System.***out***.println("test2");

}

@Test

**public** **void** test3() {

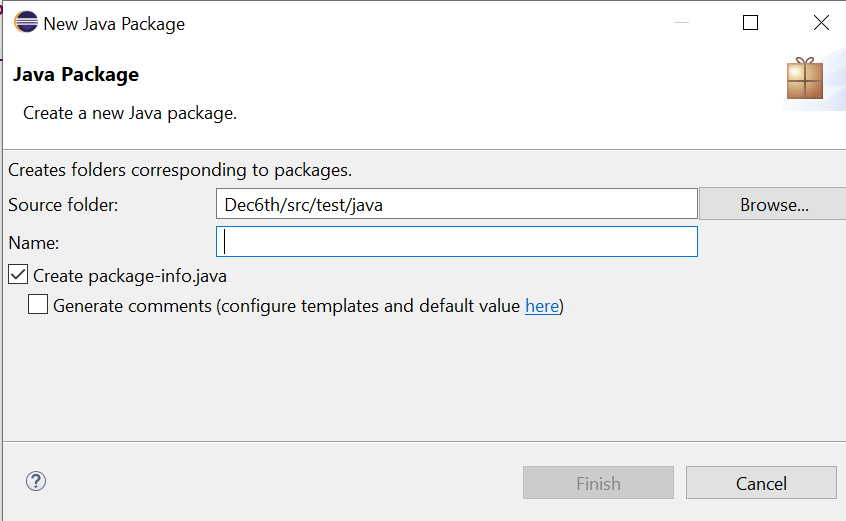
System.***out***.println("test3");

}

}

If we put ignore in the class level, then the methods in that class are not executed.  
If we keep ignore in class level and method level also remains same.

Step6: To ignore a package.  
a.Right click on src/test/java.



Click on “create package-info-java” and enter the package name which you want to ignore and click finish.

Then add ignore annotation, it ignores all the methods and classes in the package.

@Ignore

**package** com.psr.test;

**import** org.testng.annotations.Ignore;

How to group your tests in TestNG:

1. What is grouping. Why do we need grouping.  
   If you want to combine / categorize multiple test under one set, then it is called as grouping.  
   To combine multiple tests in a single set we need grouping.
2. How to define grouping.  
   Suppose if we create two classes “NewTest” and “NewTest1”, add groups under test annotation.

**package** com.psr.groupTest;

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

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

@Test(groups = {"smoke"})

**public** **void** test1() {

System.***out***.println("test1");

}

@Test(groups = {"smoke", "functional"})

**public** **void** test2() {

System.***out***.println("test2");

}

@Test(groups = {"smoke", "functional", "regression"})

**public** **void** test3() {

System.***out***.println("test3");

}

}

**package** com.psr.groupTest;

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

**public** **class** NewTest1 {

@Test(groups = {"smoke"})

**public** **void** test4() {

System.***out***.println("test4");

}

@Test(groups = {"regression", "smoke"})

**public** **void** test5() {

System.***out***.println("test5");

}

@Test(groups = {"functional"})

**public** **void** test6() {

System.***out***.println("test6");

}

@Test

**public** **void** test7() {

System.***out***.println("test7");

}

}

1. Types of groups and how to use them:
2. Test Groups.
3. Exclusion groups.
4. Meta groups (groups in groups).
5. Partial groups.

Create one XML file:

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

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

<suite name=*"Suite1"*>

<test name=*"test"*>

<groups>

<run>

<include name=*"smoke"*/>

</run>

</groups>

<classes>

<class name=*"com.psr.groupTest.NewTest"* />

</classes>

</test>

</suite>

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

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

<suite name=*"Suite1"*>

<test name=*"test"*>

<groups>

<run>

<include name=*"smoke"*/>

<exclude name=*"regression"*/>

</run>

</groups>

<classes>

<class name=*"com.psr.groupTest.NewTest"* />

</classes>

</test>

</suite>

Meta groups(groups in groups)  
<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

<suite name=*"Suite1"*>

<test name=*"test"*>

<groups>

<define name=*"DailyBasis"*>

<include name=*"smoke"*/>

<include name=*"functional"*/>

</define>

<define name=*"WeeklyBasis"*>

<include name=*"regression"*/>

<include name=*"smoke"*/>

</define>

<run>

<include name=*"DailyBasis"*/>

<include name=*"WeeklyBasis"*/>

</run>

</groups>

<classes>

<class name=*"com.psr.groupTest.NewTest"* />

<class name=*"com.psr.groupTest.NewTest1"*/>

</classes>

</test>

</suite>

Note: exclude tag is not applicable under define.

Partial groups:  
If you are assigning any group at the class level, then all the test methods present inside that class will be also part of that group.

**package** com.psr.groupTest;

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

@Test(groups = {"all"})

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

@Test(groups = {"smoke"})

**public** **void** test1() {

System.***out***.println("test1");

}

@Test(groups = {"smoke", "functional"})

**public** **void** test2() {

System.***out***.println("test2");

}

@Test(groups = {"smoke", "functional", "regression"})

**public** **void** test3() {

System.***out***.println("test3");

}

}

1. How to execute the groups at suite level.

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

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

<suite name=*"Suite1"*>

<groups>

<run>

<include name=*"smoke"* />

</run>

</groups>

<test name=*"Testing"*>

<classes>

<class name=*"com.psr.groupTest.NewTest"* />

</classes>

</test>

<test name=*"Testing1"*>

<classes>

<class name=*"com.psr.groupTest.NewTest1"* />

</classes>

</test>

</suite>

If you want to add separate “functional” to only for “NewTest1” class, then add group tags under test tag.

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

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

<suite name=*"Suite1"*>

<groups>

<run>

<include name=*"smoke"* />

</run>

</groups>

<test name=*"Testing"*>

<classes>

<class name=*"com.psr.groupTest.NewTest"* />

</classes>

</test>

<test name=*"Testing1"*>

<groups>

<run>

<include name=*"functional"* />

</run>

</groups>

<classes>

<class name=*"com.psr.groupTest.NewTest1"* />

</classes>

</test>

</suite>

1. How to execute the group at suite level + test level.  
   <?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

<suite name=*"Suite1"*>

<groups>

<run>

<include name=*"smoke"* />

</run>

</groups>

<test name=*"Testing"*>

<groups>

<run>

<include name=*"regression"* />

</run>

</groups>

<classes>

<class name=*"com.psr.groupTest.NewTest"* />

</classes>

</test>

<test name=*"Testing1"*>

<groups>

<run>

<include name=*"functional"* />

</run>

</groups>

<classes>

<class name=*"com.psr.groupTest.NewTest1"* />

</classes>

</test>

</suite>

How to create the test dependencies in TestNG:

1. What is a test dependency. Why we need test dependencies.

**package** com.psr.dependencyTests;

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

**public** **class** DependencyTest {

**static** String *trackingNumber* = **null**;

@Test(priority = 1)

**public** **void** CreateShipment() {

System.***out***.println("Create Shipment");

*trackingNumber* = "ABCDEF";

}

@Test(priority = 2)

**public** **void** TrackShipment() **throws** Exception {

**if**(*trackingNumber* !=**null**)

System.***out***.println("Track Shipment");

**else**

**throw** **new** Exception("invalid tracking number");

}

@Test(priority = 3)

**public** **void** CancelShipment() **throws** Exception {

**if**(*trackingNumber* !=**null**)

System.***out***.println("Cancel Shipment");

**else**

**throw** **new** Exception("invalid tracking number");

}

}

If we fails the first method, then other two also fails.  
Simply add print statement that becomes value 0.  
**public** **void** CreateShipment() {

System.***out***.println(5/0);

System.***out***.println("Create Shipment");

*trackingNumber* = "ABCDEF";

}

Now add the dependencies to the to the second and third method.

**package** com.psr.dependencyTests;

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

**public** **class** DependencyTest {

**static** String *trackingNumber* = **null**;

@Test(priority=0)

**public** **void** CreateShipment() {

System.***out***.println("Create Shipment");

*trackingNumber* = "ABCDEF";

}

@Test(priority=1, dependsOnMethods = {"CreateShipment"})

**public** **void** TrackShipment() **throws** Exception {

**if**(*trackingNumber* !=**null**)

System.***out***.println("Track Shipment");

**else**

**throw** **new** Exception("invalid tracking number");

}

@Test(priority=2, dependsOnMethods = {"CreateShipment"})

**public** **void** CancelShipment() **throws** Exception {

**if**(*trackingNumber* !=**null**)

System.***out***.println("Cancel Shipment");

**else**

**throw** **new** Exception("invalid tracking number");

}

}

When we keep the value 0 in the first method, then the first method fails and others are skipped.

1. What happens when priority is set to the test methods.

The priority is ignored and it executes based on dependency.

1. If we want to run second method when first method fails, then we need to add parameter “alwaysrun”.

@Test(priority=1, dependsOnMethods = {"CreateShipment"}, alwaysRun = **true**)

How to create the test group dependencies in TestNG:  
Step1: Create one TestNG class:  
**package** com.psr.dependencyTests;

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

**public** **class** TestGroupDependency {

@Test(groups = {"smoke"})

**public** **void** test1() {

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

}

@Test(groups = {"smoke"})

**public** **void** test2() {

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

}

@Test(groups = {"smoke"})

**public** **void** test3() {

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

}

@Test(dependsOnGroups = "smoke")

**public** **void** test0() {

System.***out***.println("main test");

}

}

Step2: If we fails third method, then fourth will skip.  
simply print the negative value.  
@Test(groups = {"smoke"})

**public** **void** test3() {

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

System.out.println(5/0);

}

If we want to create and run groups in group dependencies the we need to create one xml file. And add those groups in the dependencies tag.

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

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

<suite name=*"Suite1"*>

<test name=*"Test1"*>

<groups>

<dependencies>

<group depends-on=*"sanity"* name=*"smoke"* />

<group depends-on=*"regression"* name=*"sanity"* />

</dependencies>

</groups>

<classes>

<class name=*"com.psr.dependencyTests.TestGroupDependency"* />

</classes>

</test>

</suite>

If we fails any smoke method, then sanity also fails.

What is a Data provider and to use it:

1.What is a Data Provider. Why do we use it.  
Data Provider is the container which pass the data to our test method.

To parameterize the data to the test method we use data provider.

package testNG;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class Dataproviders {

@Test(dataProvider = "Data1")

public void Data(String username,String password) throws InterruptedException {

WebDriver a = new ChromeDriver();

a.manage().window().maximize();

a.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

Thread.sleep(1000);

a.findElement(By.name("username")).sendKeys(username);

a.findElement(By.name("password")).sendKeys(password);

a.findElement(By.xpath("//\*[@id=\"app\"]/div[1]/div/div[1]/div/div[2]/div[2]/form/div[3]/button")).click();

//Assert.assertTrue(a.findElement(By.xpath("//p[@class='oxd-userdropdown-name']")).isDisplayed());

Thread.sleep(5000);

a.quit();

}

@DataProvider

public Object[][] Data1() {

Object[][] q = new Object[2][2];

q[0][0] = "Admin";

q[0][1] = "admin123";

q[1][0] = "Admin1";

q[1][1] = "Venkat123";

return q;

}

}

**Parameters of Test method and return types of Data Provider:  
1. Is it mandatory to return object[] [] from the data provider.  
 NO**

**2.What are the return types of a data provider.  
 a. Object[] (Single dimensional array)  
 b. Object[] [] (Multi dimensional array)  
 c. Iterator<Object>  
 d. Iterator<Object[]>**

**Object – any datatype (predefined / user defined)  
String[] – string, string, string**

**Step1: Using single dimensional for String  
package** testNG;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(String s) **throws** InterruptedException {

System.***out***.println(s);

}

@DataProvider

**public** String[] Data1() {

String[] q = **new** String[] {

"pqr",

"abc",

"xyz"

};

**return** q;

}

}

Step2: Using single dimensional for Integer.  
**package** testNG;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(Integer s) **throws** InterruptedException {

System.***out***.println(s);

}

@DataProvider

**public** Integer[] Data1() {

Integer[] q = **new** Integer[] {

5,

6,

2

};

**return** q;

}

}

Step3: Now if we pass one string in the array, then we need to use object.  
**package** testNG;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(Object s) **throws** InterruptedException {

System.***out***.println(s);

}

@DataProvider

**public** Object[] Data1() {

Object[] q = **new** Object[] {

5,

"Hello",

2

};

**return** q;

}

}

Step4: using two dimensional array for string.  
**package** testNG;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(String[] s) **throws** InterruptedException {

System.***out***.println(s[0] + " >> " + s[1]);

}

@DataProvider

**public** String[][] Data1() {

String[][] q = **new** String[][] {

{"Hello", "psr"},

{"Abc", "xyz"},

{"abc", "pqw"}

};

**return** q;

}

}

Step5: Using two dimensional array for username and password.  
**package** testNG;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(String username, String password) **throws** InterruptedException {

System.***out***.println(username + " >> " + password);

System.***out***.println(username + password);

}

@DataProvider

**public** String[][] Data1() {

String[][] q = **new** String[][] {

{"Hello", "psr"},

{"Abc", "xyz"},

{"abc", "pqw"}

};

**return** q;

}

}

Step6: If we keep String[] in the method, then there is no problem with how many arrays are there in the data provider.

@DataProvider

**public** String[][] Data1() {

String[][] q = **new** String[][] {

{"Hello", "psr", "123"},

{"Abc", "xyz"},

{"abc", "pqw"}

};

Step7: using for loop.  
**package** testNG;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(String[] s) **throws** InterruptedException {

System.***out***.println(s[0] + " >> " + s[1]);

**for**(**int** i = 0; i < s.length; i++) {

System.***out***.println(s[i]);

}

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

}

@DataProvider

**public** String[][] Data1() {

String[][] q = **new** String[][] {

{"Hello", "psr", “123”},

{"Abc", "xyz"},

{"abc", "pqw"}

};

**return** q;

}

}

Step8: if we place “123” as 123, then we need to mention as Object.

Iterator<Object>: Iterator is used for reading the values from a collections.  
**package** testNG;

**import** java.util.ArrayList;

**import** java.util.List;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(Object[] s) **throws** Exception {

System.***out***.println(s);

}

@DataProvider

**public** List<String> Data1() {

List<String> data = **new** ArrayList<String>();

data.add("PSR");

data.add("parthu");

**return** data;

}

}

It shows an exception error, because need to import the iterator.

**package** testNG;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(Object[] s) **throws** Exception {

System.***out***.println(s);

}

@DataProvider

**public** Iterator<String> Data1() {

List<String> data = **new** ArrayList<String>();

data.add("PSR");

data.add("parthu");

**return** data.iterator();

}

}

Now we can change the type of collection and its types.

Iterator<Object[]>:  
**package** testNG;

**import** java.util.ArrayDeque;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Queue;

**import** java.util.Set;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(Object[] s) **throws** Exception {

System.***out***.println(s);

}

@DataProvider

**public** Iterator<String[]> Data1() {

Set<String[]> data = **new** HashSet<String[]>();

data.add(**new** String[] {"partha", "abc"});

data.add(**new** String[] {"PSR", "add"});

**return** data.iterator();

};

}

Now if we want mention “123” as 123, then we can use Object.

3.How do we specify the parameters in method signature of a test method.  
 It completely depends on the data.

How to use the dataprovider partially in TestNG:

1. What is Indices. Why do we use indices.  
   Indices are indexes of which data we need to actually pass.

We need indices to pass data providers data partially.

1. How to use indices.  
   **package** testNG;

**import** java.util.ArrayDeque;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Queue;

**import** java.util.Set;

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Returntypes {

@Test(dataProvider = "Data1")

**public** **void** Data(Object[] s) **throws** Exception {

System.***out***.println(s);

}

@DataProvider(indices = {1})

**public** Iterator<String[]> Data1() {

Set<String[]> data = **new** HashSet<String[]>();

data.add(**new** String[] {"partha", "abc", "Apple"});

data.add(**new** String[] {"PSR", "add", "Dog"});

**return** data.iterator();

};

}

1. What happens if we don’t pass use indices.  
   If we don’t use the indices, then entire data is supplied to the method.

How to create the dataprovider in a separate class:

1. Can we separate the data provider from the test class.  
   Yes
2. What is dataprovider class. Why do we use the dataprovider class.  
   Dataprovider is a parameter inside the test annotation.  
   To specify where our dataprovider is present we are using dataprovider class attribute.
3. How to use the dataprovider class.  
   Step1: create one class with dataprovider  
    **package** testNG;

**import** org.testng.annotations.DataProvider;

**public** **class** DProviderSep {

@DataProvider(indices = {1})

**public** String[] Data1() {

String[] data = **new** String[] {

"parthu",

"venkat",

"dinesh"

};

**return** data;

}

}

Step2: create one class with @Test and data provider.  
**package** testNG;

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

**public** **class** DataPSeparateClass {

@Test(dataProvider = "Data1", dataProviderClass = DProviderSep.**class** )

**public** **void** Data(Object[] s) **throws** Exception {

System.***out***.println(s);

}

}

1. Can we store multiple dataprovider in a dataprovider class.  
   Yes, but name should be different.
2. Can we have multiple dataprovider classes in a project.  
   Yes

Integration of excel with dataprovider:

1. Libraries available for reading the data from excel file.  
   Excel file is a spread sheet file which has rows and columns.  
   Extension of excel file 🡪 xls, xlsx.  
   xls is a older version and xlsx is a newer version.  
     
   Libraries:  
   Apache POI, JXL  
   Apache POI supports both version.  
   JXL supports only older version but performance is faster when compared to Apache POI.
2. Downloading and Installation of Apache POI java library.  
   Go to maven repository and copy Apache POI common and Apache POI API bases on OPC and OOXML Schemas stable versions and paste it in the POM.Xml.
3. How to read data from excel file dynamically.

**package** integrationExcel;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

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

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**public** **class** ReadDataFrExcel {

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

File excelfile = **new** File("C:\\Users\\ADMIN\\eclipse-workspace\\Dec6th\\src\\test\\resources\\Test.xlsx");

//System.out.println(excelfile.exists());

FileInputStream f1 = **new** FileInputStream(excelfile); //to read raw data from excel

//Now i want to all the data in the excel file so i am converting the data into workbook format

XSSFWorkbook workbook = **new** XSSFWorkbook(f1);//HSSF for older versions //pass the XSSFWorkbook as input Stream

XSSFSheet sheet = workbook.getSheet("Sheet1"); //to get sheet from workbook

//System.out.println(sheet.getPhysicalNumberOfRows());//to get no:of rows in a sheet

**int** rowscount = sheet.getPhysicalNumberOfRows();

**int** columnscount = sheet.getRow(0).getLastCellNum();

//System.out.println(sheet.getLastRowNum()); //to get last row index number

**for** (**int** i = 0; i < rowscount - 1; i++) {

**for** (**int** j = 0; j < columnscount; j++) {

DataFormatter df = **new** DataFormatter();

System.***out***.println(df.formatCellValue(sheet.getRow(i).getCell(j)));

//System.out.println(sheet.getRow(i).getCell(j).getStringCellValue());

}

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

}

workbook.close();

f1.close();

}

}

Now integrate this data with the data provider:

**package** integrationExcel;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.util.Arrays;

**import** java.util.Iterator;

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

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**public** **class** ReadDataFrExcel {

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

File excelfile = **new** File("C:\\Users\\ADMIN\\eclipse-workspace\\Dec6th\\src\\test\\resources\\Test.xlsx");

//System.out.println(excelfile.exists());

FileInputStream f1 = **new** FileInputStream(excelfile); //to read raw data from excel

//Now i want to all the data in the excel file so i am converting the data into workbook format

XSSFWorkbook workbook = **new** XSSFWorkbook(f1);//HSSF for older versions //pass the XSSFWorkbook as input Stream

XSSFSheet sheet = workbook.getSheet("Sheet1"); //to get sheet from workbook

//System.out.println(sheet.getPhysicalNumberOfRows());//to get no:of rows in a sheet

**int** rowscount = sheet.getPhysicalNumberOfRows();

**int** columnscount = sheet.getRow(0).getLastCellNum();

//System.out.println(sheet.getLastRowNum()); //to get last row index number

String[][] data = **new** String[rowscount - 1][columnscount];

**for** (**int** i = 0; i < rowscount - 1; i++) {

**for** (**int** j = 0; j < columnscount; j++) {

DataFormatter df = **new** DataFormatter();

//System.out.println(df.formatCellValue(sheet.getRow(i).getCell(j)));

data[i][j] = df.formatCellValue(sheet.getRow(i + 1).getCell(j));

//System.out.println(sheet.getRow(i).getCell(j).getStringCellValue());

}

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

}

workbook.close();

f1.close();

**for** (String[] dataArr : data) {

System.***out***.println(Arrays.*toString*(dataArr));

}

}

}

1. How to create data provider by reading the data from excel file.
2. How to maintain the data provider in a separate class.  
     
   Now create one class in the same package or different package and pass the application data and execute with adding data provider names.  
     
   Class with Application details:

**package** integrationExcel;

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

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

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

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

**public** **class** DataProvi {

@Test(dataProvider = "getData", dataProviderClass = ReadDataFrExcel.**class**)

**public** **void** Data(String username,String password) **throws** InterruptedException {

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

a.manage().window().maximize();

a.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

Thread.*sleep*(1000);

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

a.findElement(By.*name*("password")).sendKeys(password);

a.findElement(By.*xpath*("//\*[@id=\"app\"]/div[1]/div/div[1]/div/div[2]/div[2]/form/div[3]/button")).click();

Thread.*sleep*(5000);

a.quit();

}

}

Class with File data:  
  
**package** integrationExcel;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.util.Arrays;

**import** java.util.Iterator;

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

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**import** org.testng.annotations.DataProvider;

**public** **class** ReadDataFrExcel {

@DataProvider

**public** String[][] getData() **throws** Exception {

File excelfile = **new** File("C:\\Users\\ADMIN\\eclipse-workspace\\Dec6th\\src\\test\\resources\\Test.xlsx");

FileInputStream f1 = **new** FileInputStream(excelfile);

XSSFWorkbook workbook = **new** XSSFWorkbook(f1);

XSSFSheet sheet = workbook.getSheet("Sheet1");

**int** rowscount = sheet.getPhysicalNumberOfRows();

**int** columnscount = sheet.getRow(0).getLastCellNum();

String[][] data = **new** String[rowscount - 1][columnscount];

**for** (**int** i = 0; i < rowscount - 1; i++) {

**for** (**int** j = 0; j < columnscount; j++) {

DataFormatter df = **new** DataFormatter();

data[i][j] = df.formatCellValue(sheet.getRow(i + 1).getCell(j));

}

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

}

workbook.close();

f1.close();

**return** data;

}

}

Now execute the application class with TestNG.

Parallel execution with data provider:  
1.How to execute tests parallelly with data provider.  
package testNG;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class Dataproviders {

@Test(dataProvider = "Data1")

public void Data(String username,String password) throws InterruptedException {

WebDriver a = new ChromeDriver();

a.manage().window().maximize();

a.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

Thread.sleep(1000);

a.findElement(By.name("username")).sendKeys(username);

a.findElement(By.name("password")).sendKeys(password);

a.findElement(By.xpath("//\*[@id=\"app\"]/div[1]/div/div[1]/div/div[2]/div[2]/form/div[3]/button")).click();

//Assert.assertTrue(a.findElement(By.xpath("//p[@class='oxd-userdropdown-name']")).isDisplayed());

Thread.sleep(5000);

a.quit();

}

@DataProvider(parallel = true)

public Object[][] Data1() {

Object[][] q = new Object[2][2];

q[0][0] = "Admin";

q[0][1] = "admin123";

q[1][0] = "Admin1";

q[1][1] = "Venkat123";

return q;

}

}

2.How to control the thread count when using data provider.

package testNG;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class Dataproviders {

@Test(dataProvider = "Data1")

public void Data(String username,String password) throws InterruptedException {

WebDriver a = new ChromeDriver();

a.manage().window().maximize();

a.get("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login");

Thread.sleep(1000);

a.findElement(By.name("username")).sendKeys(username);

a.findElement(By.name("password")).sendKeys(password);

a.findElement(By.xpath("//\*[@id=\"app\"]/div[1]/div/div[1]/div/div[2]/div[2]/form/div[3]/button")).click();

//Assert.assertTrue(a.findElement(By.xpath("//p[@class='oxd-userdropdown-name']")).isDisplayed());

Thread.sleep(5000);

a.quit();

}

@DataProvider(parallel = true)

public Object[][] Data1() {

Object[][] q = new Object[4][2];

q[0][0] = "Admin";

q[0][1] = "admin123";

q[1][0] = "Admin1";

q[1][1] = "Venkat123";

q[2][0] = "Partha";

q[2][1] = "Partha123";

q[3][0] = "Admin";

q[3][1] = "test123";

return q;

}

}

Create one xml file and execute it.  
<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

<suite name=*"Suite"* data-provider-thread-count=*"3"*>

<test name=*"Test"*>

<classes>

<class name=*"testNG.Dataproviders"* />

</classes>

</test>

</suite>

How to use timeout & expected exceptions attributes in TestNG:  
1.What is Timeout.

Timeout is an attribute that is present inside the test annotation.

2.How to use timeout.

**package** timeOutException;

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

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

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Timeout {

@Test(timeOut = 3000)

**public** **void** Test() {

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

driver.get("https://www.instagram.com/");

driver.quit();

}

}

If we give three seconds for execution and the program executes more than three seconds of time it throws “Method TimeOutException”.

3.What is expected exceptions.

4.How to use expected exception.

**package** timeOutException;

**import** java.util.NoSuchElementException;

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

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

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

**import** org.testng.annotations.DataProvider;

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

**public** **class** Timeout {

@Test(expectedExceptions = NoSuchElementException.**class**)

**public** **void** Test() {

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

driver.get("https://www.instagram.com/");

driver.quit();

}

}

We can use more than one expected exception attribute in a single test.

Execute your tests in parallel using TestNG:

1. What is parallel execution?  
     
   Parallel – Side by side  
   Sequential – One after another  
   E.g. for sequential: parallel=”tests” thread – count=”1”  
     
   Executing tests side by side is called Parallel testing.
2. Why do we need Parallel testing?  
     
   To save time while executing tests.
3. How to perform the Parallel testing?  
     
   Step1: Create one package and three classes in that and add test methods in all classes.  
     
   🡪  
   **package** parallell;

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

**public** **class** Test1 {

@Test

**public** **void** Testmethod() {

System.***out***.println("Test1 >>> Testmethod >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod1() {

System.***out***.println("Test1 >>> Testmethod1 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod2() {

System.***out***.println("Test1 >>> Testmethod2 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod3() {

System.***out***.println("Test1 >>> Testmethod3 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod4() {

System.***out***.println("Test1 >>> Testmethod4 >>" + Thread.*currentThread*().getId());

}

}

🡪  
**package** parallell;

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

**public** **class** Test2 {

@Test

**public** **void** Testmethod5() {

System.***out***.println("Test2 >>> Testmethod5 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod6() {

System.***out***.println("Test2 >>> Testmethod6 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod7() {

System.***out***.println("Test2 >>> Testmethod7 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod8() {

System.***out***.println("Test2 >>> Testmethod8 >>" + Thread.*currentThread*().getId());

}

}

🡪

**package** parallell;

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

**public** **class** Test3 {

@Test

**public** **void** Testmethod9() {

System.***out***.println("Test3 >>> Testmethod9 >>" + Thread.*currentThread*().getId());

}

@Test

**public** **void** Testmethod10() {

System.***out***.println("Test3 >>> Testmethod10 >>" + Thread.*currentThread*().getId());

}

}  
  
Step2: Execute all the three classes in xml file at a time.

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

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

<suite name=*"Suite1"*>

<test name=*"Testing1"*>

<classes>

<class name=*"parallell.Test1"* />

</classes>

</test>

<test name=*"Testing2"*>

<classes>

<class name=*"parallell.Test2"* />

</classes>

</test>

<test name=*"Testing3"*>

<classes>

<class name=*"parallell.Test3"* />

</classes>

</test>

</suite>

🡪Parallel and thread – count are the two parameters that we use at the parallel execution concept.  
  
Step3: Add parallel and thread – count at suite level.  
  
<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

<suite name=*"Suite1"* parallel=*"none"* thread-count=*"2"*>

<test name=*"Testing1"*>

<classes>

<class name=*"parallell.Test1"* />

</classes>

</test>

<test name=*"Testing2"*>

<classes>

<class name=*"parallell.Test2"* />

</classes>

</test>

<test name=*"Testing3"*>

<classes>

<class name=*"parallell.Test3"* />

</classes>

</test>

</suite>

🡪If parallel is none or false, then TestNG will not consider the thread – count.  
🡪You cannot provide thread – count as negative, because it throws an “Illegal Argument Exception”.  
  
Step4: Now we are introducing parallel testing.  
If we give parallel=”tests” and thread – count=”2” at suite level for all the three tests, then two tests run in a same time, and third test wait until one test is completed and then executes.  
  
Step5: If we give thread – count as 3 for three tests, then it executes in three threads.  
  
🡪If number of tests are less than number of threads then remaining threads will be idle.  
  
🡪parallel = “tests” can be use only in suite level, not in test level.

🡪Now use parallel=”classes” or “methods” and thread – count=”2”at test level.

How to use Invocation count, Invocation timeout & ThreadPool size in TestNG:

1. What is Invocation count?  
     
   Invocation count is an attribute or parameter which tells the test method like how many times the same test method should be executed.
2. Why do we need Invocation count?  
     
   Suppose if we use “for loop” to iterate the method, then if it avoids the execution, then we use Invocation count to execute the method multiple times.
3. How to use Invocation count?  
     
   **package** invocationCount;

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

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

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

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

**public** **class** Count {

@Test(invocationCount = 4)

**public** **void** TestCount() **throws** Exception {

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

driver.get("https://www.instagram.com/");

Thread.*sleep*(2000);

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

driver.findElement(By.*name*("password")).sendKeys("Partha@4434");

Thread.*sleep*(2000);

driver.findElement(By.*xpath*("//div[@class='\_ab8w \_ab94 \_ab99 \_ab9f \_ab9m \_ab9p \_abak \_abb8 \_abbq \_abb- \_abcm']")).click();

driver.quit();

}

}

🡪During iteration, if any iteration fails, it continues to the next iteration.

1. What is Invocation timeout?  
     
   To restrict the method in a given time.
2. Why do we use the Invocation timeout?  
     
   To control the method by using timeout.
3. How to use Invocation timeout?

**package** invocationCount;

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

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

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

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

**public** **class** Count {

@Test(invocationCount = 4, invocationTimeOut = 10000)

**public** **void** TestCount() **throws** Exception {

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

driver.get("https://www.instagram.com/");

Thread.*sleep*(2000);

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

driver.findElement(By.*name*("password")).sendKeys("Partha@4434");

Thread.*sleep*(2000);

driver.findElement(By.*xpath*("//div[@class='\_ab8w \_ab94 \_ab99 \_ab9f \_ab9m \_ab9p \_abak \_abb8 \_abbq \_abb- \_abcm']")).click();

driver.quit();

}

}

🡪Without using Invocation count, the use of Invocation timeout is useless.  
🡪If we give 10 seconds to iterate the method 4 times, if the method executes before 10 seconds then no error will be shown otherwise then it error.

1. What is ThreadPool size?  
     
   To create the number of threads for invocation count.
2. Why do we use ThreadPool size?  
     
   To execute the method count with the thread class.
3. How to use ThreadPool size?  
     
   **package** invocationCount;

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

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

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

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

**public** **class** Count {

@Test(invocationCount = 4, threadPoolSize = 2)

**public** **void** TestCount() **throws** Exception {

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

driver.get("https://www.instagram.com/");

Thread.*sleep*(2000);

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

driver.findElement(By.*name*("password")).sendKeys("Partha@4434");

Thread.*sleep*(2000);

driver.findElement(By.*xpath*("//div[@class='\_ab8w \_ab94 \_ab99 \_ab9f \_ab9m \_ab9p \_abak \_abb8 \_abbq \_abb- \_abcm']")).click();

driver.quit();

}

}

🡪Without using Invocation count, ThreadPool size is useless.

TestNG Listeners

1. What is a listener in TestNG?  
     
   Listener is defined as interface that modifies the default TestNG’s behaviour.
2. Why do we need a listener in TestNG?

If you want to perform any actions before execution we need listener.

1. List some of the Listeners available in TestNG?  
     
   IAnnotationTransformer  
   IAnnotationTransformer2  
   IHookable  
   IInvokedMethodListener  
   IMethodInterceptor  
   IReporter  
   ISuiteListener  
   ITestListener  
     
   🡪Now we will go with ITestListener.

🡪  
  
**package** listeners;

**import** org.testng.ITestContext;

**import** org.testng.ITestListener;

**import** org.testng.ITestResult;

**public** **class** ItestListeners **implements** ITestListener {

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

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

}

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

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

}

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

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

}

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

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

}

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

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

}

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

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

}

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

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

}

}

🡪

**package** listeners;

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

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

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

**import** org.testng.annotations.BeforeTest;

**import** org.testng.annotations.Listeners;

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

@Listeners(ItestListeners.**class**)

**public** **class** Listen {

**public** **static** WebDriver *a*;

@BeforeTest

**public** **void** Apple() {

*a* = **new** ChromeDriver();

}

@Test(testName = "Launch")

**public** **void** Launch() **throws** Exception {

*a*.get("https://innojc.kredily.com/company/dashboard/");

}

@Test(testName = "Login")

**public** **void** Login() **throws** Exception {

Thread.*sleep*(5000);

*a*.findElement(By.*id*("signInFormEmailAddress")).sendKeys("8096291806");

Thread.*sleep*(1000);

*a*.findElement(By.*id*("signInFormPassword")).sendKeys("Partha@4434");

}

@Test(testName = "Sign\_In")

**public** **void** Sign\_In() **throws** Exception {

Thread.*sleep*(1000);

*a*.findElement(By.*id*("signinSubmitBtn")).click();

}

}

Now execute with TestNG test.  
🡪If we place 2 seconds timeout for test method, it fails because it not takes any keys.

Capture Screenshot for only failed tests in TestNG:  
  
Create three classes and pass the data  
  
🡪  
  
**package** Screenshots;

**import** java.io.File;

**import** java.io.IOException;

**import** java.time.LocalDateTime;

**import** java.time.format.DateTimeFormatter;

**import** org.apache.commons.io.FileUtils;

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

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

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

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

**import** org.testng.ITestResult;

**import** org.testng.annotations.AfterTest;

**import** org.testng.annotations.BeforeTest;

**public** **class** Browser {

**public** **static** WebDriver *a*;

**public** **static** String *datetimeformat*;

@BeforeTest

**public** **void** Start() {

*a* = **new** ChromeDriver();

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

}

@AfterTest

**public** **void** Stop() {

*a*.quit();

}

**public** **void** CaptureScreenshot(String fileName) {

**if**(*datetimeformat* == **null**) {

LocalDateTime myDateObj = LocalDateTime.*now*();

DateTimeFormatter myFormatObj = DateTimeFormatter.*ofPattern*("dd-MM-yyyy HH mm ss");

*datetimeformat* = myDateObj.format(myFormatObj);

}

TakesScreenshot takesScreenshot = (TakesScreenshot) *a*;

File sourceFile = takesScreenshot.getScreenshotAs(OutputType.***FILE***);

File destFile = **new** File("C:\\Users\\ADMIN\\OneDrive - Innojc Technologies Pvt. Ltd\\Pictures\\Screenshots\\"+*datetimeformat*+'/'+fileName);

**try** {

FileUtils.*copyFile*(sourceFile, destFile);

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("Screenshot saved successfully");

}

}

🡪  
**package** Screenshots;

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

**import** org.testng.annotations.Listeners;

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

@Listeners(ITest.**class**)

**public** **class** FirstTest **extends** Browser {

@Test(testName = "Launch")

**public** **void** Launch() **throws** Exception {

*a*.get("https://innojc.kredily.com/company/dashboard/");

}

@Test(testName = "Login")

**public** **void** Login() **throws** Exception {

Thread.*sleep*(1000);

*a*.findElement(By.*id*("signInFormEmailAddress")).sendKeys("8096291806");

Thread.*sleep*(1000);

*a*.findElement(By.*id*("signInFormPassword")).sendKeys("Partha@4434");

}

@Test(testName = "Sign\_In")

**public** **void** Sign\_In() **throws** Exception {

Thread.*sleep*(1000);

*a*.findElement(By.*id*("signinSubmitBtn")).click();

}

}

🡪  
**package** Screenshots;

**import** org.testng.ITestListener;

**import** org.testng.ITestResult;

**public** **class** ITest **extends** Browser **implements** ITestListener{

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

CaptureScreenshot(result.getMethod().getMethodName()+".jpg");

}

}

Now execute the application data class, it stores a screenshots at defined path.  
  
Default reports generated by TestNG:

1. What is meant by reports?  
     
   TestNG reports are the default HTML reports which are generated once the test cases are executed using TestNG.
2. Does TestNG generate any reports by default?  
     
   Yes.
3. Overview of the TestNG default reports?  
     
   project >> right click >> properties >> show in system explorer >> project name >> test output >> emailable report or index.
4. Do we really use the TestNG default reports?  
     
   No, we cannot use in real time.  
   But we use the extent reports.

Extent reports Integration with TestNG:  
  
We can create the Extent reports by using two ways.  
🡪Annotation level.  
🡪Listeners level.  
  
  
  
  
 -------END-------