27.05.2022 TestNG (Test New Generation)

Testing (Test new generation)

Jar / Dependencies Required:

testing 6.14.3

testng 6.14.3

jcommander 1.27 (Supporting jar)

TestNG is nothing but Test New... where it is a open source test Auto... framework for JAVA.

Advantages or About Testng or Why not Junit ?

-> It provides the default HTML reports.
-> Easy to set priority (we can prioritized the test cases)
-> Easy to pass parameters (we can pass input as datas ... testng.xml file using parameters)
-> Data provider is possible (we can pass the bulk of data... at a time for the positive and negative testcases ... (field validations).
-> cross browser testing (To check the compatibility and ... of the application and parallel execution is possible (To ... the time consumption).
-> Automatically rerun the failed testcases.
-> Both HardAssert and SoftAssert is possible (verify)
-> We can group the bulk of testcases and then we can ... execute the testcases by using the group name.

Steps:-

1) Download the testng jar with latest version 6.14.3 / 6.7 with supporting jar jcommander version 1.27.
2) Add the testng plugin the eclipse marketplace.
3) Add the testng jar into eclipse buildpath.

Note :- NoClassDefinitionFound error will come here we need to download one more jar called JCommander version 1.27. or

AWT - Exception → Abstract window Tool.

Annotations :-

1) @BeforeSuite
2) @BeforeTest
3) @BeforeClass (J) ---> launchbrowser
4) @Beforegroup
5) @BeforeMethod (J) ---> StartTime
6) @Test (J) ---> Buisness logic
7) @AfterMethod (J) ---> endTime.
8) @AfterGroup --
9) @AfterTest
10) @AfterClass (J)
11) @AfterSuite

<Suite> —
<Test> —
<classes> —
<class> —

NOTE :-

1. private methods allowed in testng
2. @Beforeclass and @Afterclass methods need not be static

Two Types of Frame Work

TDD → Test Driven Development — Junit → Java unit Testing.
TestNG →
BDD — Behaviour Driven Development → Cucumber.

unit Testing → White Box Testing & Black Box Testing Test by Internal coding.

TDD - only Technical people can ~~start~~ understand.
BDD → Tech and Non-Technical can understand.

(Cucumber with Junit
Cucumber with TestNG)

TestNG Dependency → 6.14.3 Stable version.

Help → Eclipse marketplace → Find → TestNG → plugin. → holy tights

TestNG for eclipse → Install

Jcommander, hamcrest → just a supporting file we are not using it.

Program for TestNG:

```
public class Sample {
    WebDriver driver;

    @BeforeClass
    private void beforeclass (){
WebDriverManager.chromedriver().setup();
driver = new ChromeDriver();
driver.manage().window().maximize();
driver.get("https://www.facebook.com/");
  }
@AfterClass
  private void afterclass(){
      driver.quit();
   }
@BeforeMethod
  private void startTime(){
      Date date = new Date();
      System.out.println(date);
  }
@AfterMethod
  private void endTime(){
      Date date = new Date();
      System.out.println(date);
  }
@Test
  private void tc1(){
  WebElement txtuserName = driver.findElement(By.id("email"));
  txtusername.sendkeys("1234567");
  WebElement txtpassword = driver.findElement(By.id("pass"));
  txtpassword.sendkeys("hiboo@123");
  WebElement btnlogin = driver.findElement(By.name("login"));
  btnlogin.click();
  }
```

Priority :- Evening class :-

* By default, the test methods are executed in alphabetical or ascending order.

* When we want to change the order of the test methods we can go for the concept called priority.

* Default priority for any test is 0

* priority we can give -ve or +ve value (@Test with most negative integer will be runned first & @Test with most positive integer will be executed last).

Example :-

```
@Test(priority = -17)
```

Ignoring the particular test from execution

enabled = false.

Ex!

```
@Test(enabled = false)
```

To run the same testcases in multiple times :

invocationCount

Ex!

```
@Test(invocationCount = 100)
```

program for TestNG using priority:

```
public class Employee {
    @Test (priority = 5) //4
    private void tc1 () {
    }
    @Test (priority = 10) //5
    private void tc2 () {
    }
    @Test
    private void tc3 () { //3
    }
    @Test (priority = -5) //2
    private void tc4 () {
    }
    @Test (priority = -10) //1
    private void tc5 () {
    }
}.
```

output

tc5
tc4
tc3
tc2
tc1

Default value
Priority — 0
enabled → True
invocation count → 1

```
@Test (enabled = false)   // check.
private void tc3 () {
}
```
→ To ignore the test case and its default value is true.

```
@Test (invocationCount = 10)
private void tc3 () {
}
```
→ To check the stability of particular function. It count or perform action for 10 times. The default value is 1 (1 time execute)

Testing Asserts (To set the verification point / validation point)

To verify business logic we can use Assert

Testing has both Hard Assert and Soft Assert.
→ only JunIt.

Hard Assert:

When any Assert gets failure, all the remaining lines of particular @test will not be executed and the particular @Test

will be marked as failed.

Assert --> class.

* Methods in Assert (Static methods - so to call those methods we no need to create object.)

assertTrue()
assertEquals()

SoftAssert: (verify)

When any Assert gets failure, all the remaining lines of particular @Test will be executed and the particular @Test will be marked as passed.

SoftAssert --- class

* methods in SoftAssert: (non-Static methods -- so to call those method we need to create object)

assertTrue()
assertEquals()
assertAll(). --> End of @Test method only should use.
It shows all the failed test cases.

rest of the lines in test will executed.

Hard Assert

Mandatory fields
like cardNo
cvv
cardType

Soft Assert
optional fields

program for Hard Assert:

```
@Test
private void tc1(){
    WebDriver driver;

@Test
private void tc1(){
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.get("https://www.facebook.com/");
    Assert.assertTrue(driver.getCurrentUrl().contains("facebook"),
                                                "verify URL");

    WebElement txtusername.sendKeys("123567");
    WebElement txtpassword = driver.
Assert.assertEquals(txtpassword.getAttribute("value"), "Hello@123",
                                                "verify password");
```

Program for Soft Assert:

```
@Test
private void tc1() {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.get("https://www.facebook.com");
    SoftAssert s = new SoftAssert();
    s.assertTrue(driver.getCurrentUrl().contains("xyz"), "verify");
    WebElement txtuserName = driver.findElements(By.id("email"));
    txtuserName.sendKeys("1234567");

    btnlogin.click;
    driver.quit();
    s.assertAll();
```

28.05.2022

Suite Level Execution:

If you want to execute more than one class we can go for suite level execution with help of testing.xml file.

Create testing.xml file:

Select all classes which you want to run.

Right click --> click on testNG option.

Then click convert to testNG.

click finish button.

testNG.xml file created.

To run the testng.xml file --> Right click --> Run as --> TestNG Suite

Parameters:

If you want to pass the parameters (inputs to fill the text boxes), we can use parameter one tag in testing.xml file we can pass more than one parameters.

@parameters ({"parametername"})
@Test

@optional:

When the passed parameters is mismatched in class, we can use @optional one annotation.

If parameter's name in testing.xml and class file are mismatched it will be taking value from @optional.

TestNG program: TestNG.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Suite SYSTEM "http://testng.org/testing-1.0.dtd">
<Suite name="Suite">
    <test name="Test">
        <classes>
            <class name="org.hexa.Company"/>
            <class name="org.hexa.Employee"/>
        </classes>
    </test> <!---Test-->
</Suite> <!--Suite-->.
```

```
<Suite name="Suite">
    <test name="Test">
        <parameter name="userName" value="mani@gmail.com"/>
        <parameter name="password" value="mani@123"/>
        <classes>
            <class name="org.hexa.Company"/>
            <class name="org.hexa.Employee"/>
            <class name="org.hexa.Hello"/>
        </classes>.
```

Program for testNG.xml :-

```
public class Employee {
    @parameters ({ "userName", "password" }) -> testing parameters
    @Test
    private void tc1 (String s1, String s2){
        s.o.p (s1);
        s.o.p (s2);
    }
}
```

give TestNG.xml -> Rightclick ---> Run as - TestNG Suite

program using Selenium :-

```
public class Employee {
    @parameters ({ "userName", "password" })
    @Test
    private void tc1 (String s1, String s2) {
        WebDriverManager.chromedriver().setup();
        WebDriver driver = new ChromeDriver();
        driver.manage.window().maximize();
        driver.get("https://www.facebook.com");
        WebElement txtuserName = driver.findElement(By.id("email"));
        txtuserName.sendkeys(s1);
        WebElement txtpassword = driver.findElement(By.id("pass"));
        txtpassword.sendkeys(s2);
        WebElement btnlogin = driver.findElement(By.name("login"));
        btnlogin.click();
        driver.quit();
    }
}
```

testing.xml:

```
<Suite name = "parameter">
    <parameter name = "password" value = "hello@123" />
    <test name = "Test 1">
        <parameter name = "username" value = "mani@gmail.com" />

    <classes>
        <class name = "org.hexa.Employee" />

    </classes>
    </test> <!-- Test -->
    <test name = "Test 2">
        <parameter name = "username" value = "ram@gmail.com" />
        <classes>
            <class name = "org.hexa.Employee" />
        </classes>
    </test> <!-- Test -->
    <test name = "Test 3">
        <parameter name = "username" value = "arun@gmail.com" />
        <classes>
            <class name = "org.hexa.Employee" />

        </classes>
```

mismatch parameter name is case sensitive. — TestNG Exception

program for mismatching parameters.

=> case sensitive mismatch

```
@parameters({"username", "password"})
@Test
private void test(@Optional("Hello") String x, String y){
        s.o.p(x);
        s.o.p(y);
}
```

output

Hello

Hello@123

Evening class :

* Data provider : It is used to provide bulk of datas into a one t

```
Public class Sample {
    @Test (dataprovider = "res")
    private void tc1 (String S1 , String S2) {
        System.out.println (S1);
        S.O.P (S2);

@Dataprovider (name = "res")
public object [ ] [ ] datas () {
    // DataType VariableName [ ] [ ] = new DataType [rowsize] [column
    object obj [ ] [ ] = new object [2] [2];
        obj [0] [0] = "mani@gmail.com";
        obj [0] [1] = "mani@123";
        obj [1] [0] = "abu@gmail.com";
        obj [1] [1] = "abu@123";
        return obj ;
    }
```

Output
mani@gmail.com
mani@123
abu@gmail.com
abu@123
passed : tc1 ("mani@gmail.com"
passed :

(or)

```
object obj [ ] [ ] = new object [ ] [ ] {{"mani@gmail.com", "mani@123"}
                                        ,{"abu@gmail.com", "abu@123"}};
return obj;
```

```
@Dataprovider (name = "res", parallel = true)
public object [ ] [ ] datas () { }
```

```
@Test (dataprovider = "res")
private void tc1 (String S1, String S2) {
    WebDriverManager.chromedriver().setup();
    WebDriver driver = new ChromeDriver();
    driver.get ("https://www.facebook.com");
    WebElement txtusername = driver.findElement (By.id ("email"))
```

```
txtUserName.sendKeys("s");
WebElement txtpassword = driver.findElement(By.id("pass"));
txtpassword.sendKeys("s");
WebElement btnlogin = driver.findElement(By.name("login"));
}
```

> Execute browser at a time.

```
@DataProvider (name = "res", parallel = true)
public Object [][] datas(){

Object obj[][] = new Object[2][2]{{"mani@gmail.com", "mani@123"},
                                  {"abu@gmail.com", "abu@123"}};

return obj;
}.
```

program for accessing into another class using DataProvider:

```
public class Hello{
    @Test (dataprovider = "res", dataproviderClass = Sample.class)
private void tc21 (String x, String y) {
        s.o.p(x);
        s.o.p(y);
```

just by accessing the Sample class

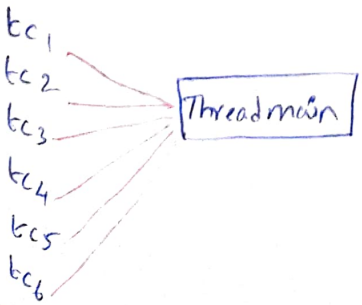
29.05.2022

(*) Java program always execute in Mainthread

(*) default Thread count is '5' (*) interview.

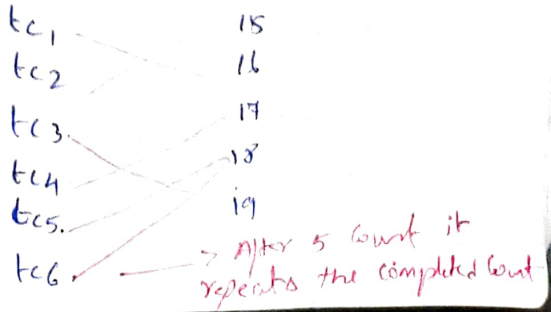
* Thread is a Execution platform.

Sequential Execution



Default thread count is - 5
maximum 20 - 25 only
should give ordbc System will hang

parallel Execution



Parallel Execution of methods

testng.xml code

```
<!DocType Suite System ....">
<Suite name = "parallelExecution" parallel = "methods"
    thread-count = "7" >
    <test name = "Test1" >
        <classes >
            <class name = "org.heca.Company" />
            </classes>
        </test> <!--Test -->
```

→ Execution of Methods Parallel

output

15
17
18
12
16
14
13

Program:

```
import org.testng.annotations.*
public class Company{
    @Test
    private void tc1(){
        s.o.p (Thread.currentThread().getId());
    }
    @Test
    private void tc2(){
        s.o.p (Thread.currentThread().getId());
    } .... tc3 .. tc4 .. tc5 .. tc6 ... tc7.
}
```

parallel Execution:

tc1 ... 1 minute
tc2 ..
tc3 ...

tc100 --> 100 minutes.

1) Methods
2) Classes
3) tests

Parallel Execution of classes:

XML code:

```
<Suite name = "parallel Execution" parallel = "classes">
  <test name = "Test1">
    <classes>
      <class name = "org.hexa.company" />
      <class name = "org.hexa.Employee" />
      <class name = "org.hexa.hexa.Employee" />
      <class name = "org.hexa.Sample" />
    </classes>
  </test> <!-- Test -->
```

Output:

15
14
13
12
13
15
14
12
15
14
13
12

Program:

Need to create four different class as name as above.

Parallel Execution of Test:

```
<Suite name = "parallel Execution" parallel = "Tests" >
  <test name = "chromeTest1">
    <parameter name = "browserName" value = "chrome" />
  <classes>
    <class name = 'org.hexa.Employee" />

  </classes>
  </test> <!--- Test --->
  <test name = "firefoxTest">
    <parameter name = "browserName" value = "firefox" />
    <classes>
      <class name = "org.hexa.Employee" />
    </classes>
  </test> <!-- Test -->
  <test name = "ieTest">
    <parameter name = "browserName" value = "ie" />
    <classes>
      <class name = "org.hexa.Employee" />
    </classes>
  </test> <!-- Test -->
```

```
public class Employee {
    WebDriver driver;
    @parameters ({"browserName"3)
    @Test
    private void tc1 (String browserName) {
        if (browserName.equals ("chrome")) {
            S.o.p ("chrome");
        WebDriverManager.chromedriver().setup();
            driver = new chromeDriver();
        } else if (browserName.equals ("firefox")) {
            S.o.p ("firefox");
            WebDriverManager.firefoxdriver().setup();
            driver = new chromedriver();
            driver.setup ("https://www.facebook.com");
        } else if
```

Evening class :-

Re-Run the failed testcases in testng :-

1) Manually :

After every Execution ---- all the failed methods are recorded in testng --failed.xml. (which is present in test --- output folder)

If we run testng --- failed.xml only testcases failed in the previous run alone will run.

2. Automatically when you know which testcases is getting failed

Beside your @Test mention the class name where you have implemented IRetryAnalyzer interface.

using retry() method we can re run the failed testcases any specified number of times.

3. Automatically When you dont know which testcases in getting failed

-> Include <Listeners> tag in testng.xml. inside <Listener>.

-> Mention listener class Name (class where we have implemented IAnnotationTransformer interface)

- In the class where we have implemented IAnnotationTransformer interface using getRetryAnalyzer() and setRetryAnalyzer() methods We can invoke the class where we have implemented IRetryAnalyzer.

Two types.

* Manually ReRunning the failed test case
* Automatic ReRunning the failed test case

* project refresh -> test-output - testng-failed.xml (manually ReRunning the test cases)
* Default Html Report -> emailable-report.html

program for RE - Run the testcases :

Manual RE-Running

```
public class Employee {

    @Test
    private void tc1(){
        S.O.P("test1")
    }
    @Test
    private void tc2(){
        Assert.assertTrue(false);
    }
    @Test
    private void tc3(){
        Assert.assertTrue(false);
    }
```

-> It shows in test output folder what are test get failed and can RE run by testoutput - Rightclick -> Run As ->

tc2 failed ---> failed.class ---> min=0, max=3 ---> retry.

method is responsible for rerun -> 0<3 -> min++ (0+1=1) -> return true

-> once again it rerun.

Automation Re-Running

Run - Run program using

```
public class Employee {
    @Test
    private void tc1() {
        S.o.p("test1");
    }
    @Test (retryAnalyzer = Failed.class)
    private void tc2() {
        // Assert.assertTrue(false);
        S.o.p("test1");
    }
    @Test (retryAnalyzer = failed.class)
    private void tc3() {
        Assert.assertTrue(false);
    }
```

failed program:

```
import

public class Failed Implement IRetryAnalyzer {
    int min = 0, max = 5;
    public boolean retry(ITestResult Result) {
        if (min < max) {
            min++;
            return true;
        }
        return false;
    }
}
```

→ default. filled arguments.

① IretryAnalyser : (I)

retry() : (m)

transform - (M)

ITestAnnotation - (I)
getRetryAnalyzer - (M)
SetRetryAnalyzer - (M)

<Listners> → Captures failed test cases from class execution

<getRetryAnalyzer> - Captures failed test case from <Listers> Interface.

Xml :-

```
<Suite name = "rerun"  parallel = "test">
  <listners>
    <Listener class-name = "org.hexa.failedAll" />
  <listners>
  <test name = "test">
    <classes>
      <class name = "org.hexa.Employee" />
    </classes>
  <test> <!--Test -->
</Suite> <!--Suite -->.
```

Program :-

```
public class FailedAll implements IAnnotationTransformer {
public void transform (ITestAnnotation annotation, class testclass,
                       Constructor testConstructor, method testMethod){
IRetryAnalyzer analyzer = annotation.getRetryAnalyzer();
  if (analyzer == null) {  => It return null value.
    annotation.SetRetryAnalyzer (Failed.class).
  }
}
```

→ Default value for class Interface, String =

TestNG :- Grouping

Grouping in TestNG :

If you want to ignore 5 testcases from execut
you can use (enabled = false) for the particular 5 test

Included :

If you want to run 300 testcases from 1000 testc
--> We cannot give (enabled = false) for all the 700
testcases. For that we can group our testcases, give nam
to all 300 testcases you want to be run @Test(group.
and include that particular group in testng.xml.

```
<groups>
<run>
<include name = "Smoke" > </include>
</run>
</groups>
```

The test cases with group name "Smoke" only will be exec

Excluded :-

If you want to ignore 300 testcases fro 1000 tes
give group name @Test (groups = "Smoke")
Exclude that particular group in testng.xml.

```
<groups>
<run>
<exclude name = "Smoke" > </excluded>
</run>
</groups>
```

The testcases with group name "Smoke" only will be i
program is there in Test Note.

xml file

```
<Suite name = "suite" >
    <groups>
        <run>
            <include name = "reg" />
        </run>
    </groups>
    <test name = "group">
        <classes>
            <class name = "org.test.Employee" />
            <class name = "org.test.Company" />
            <class name = "org.tc.client" />
        </classes>
    </test> <!-- Test -->
</suite> <!-- Suite -->
```

<Exclude name = "reg" />

```
public class Employee {
    @Test (groups = "reg")
    private void tc1() {
    }
    @Test (groups = "smoke")
    private void tc2() {
    }
    @Test (groups = "sanity")
    private void tc3() {
    }
    @Test (groups = "reg")
    private void tc4() {
    }
    @Test (groups = "E2E")
    Ly (End to End)
```

```
public class Company {
    @Test (groups = "smoke")
}
```

```
client {
}
```