TestNG

((Test Automation framework))

Test Next Generation

- TestNG is a testing framework that covers different types & test design like unit, Functional, end to end, UI and integration testing. regression,
- Through the testNG we can write the test cases, execute and generate the reports automatically.
- with testng we can execute the failed test cases separately
- we can integrate testng with other in built frame works like Maven, & Jenkins.
- with testNG class we don't need the main method to execute the test cases, internally the testng configured with main method for execution.
- Using the testng we can generate a proper report and you can easily come to know how many test cases passed, failed and skipped, and you can execute failed test cases separately (testng-failed.xml)

features of TestNG

1. Annotations: Based on annotations that provide mechanism to aid your automation scripts.
2. Supports a no of parameters
3. Generates Reports
4. Supports data-driven testing @DataProvider annotations.
5. Support of running only the failed testcase.

(6) Support of 3rd party plugin like Jenkin, Git, maven, cucumber, Junit.

(7) test configuration, grouping related test case for execution. by controlling testng.xml.

(8) Configuring the dependent test cases using 'dependsonMethods'

(9) Parallel Test Execution, test order, Listeners support

(10) Console output HTML report generation and index.html report

Advantages of TestNG :→

- Report in proper format, Passed, failed, skipped.
- Testcase can be grouped and priorities.
- Same testcase can be executed multiple times using keyword 'invocation count'
- Cross Browser testing
- Integrate with maven, Jenkin
- Annotation used are very easy to understand.
- Uncaught exceptions automatically handled without terminating the test and are reported as failed steps in report
- Tests can be run in parallel.

Advantages of TestNG over Junit ;→

(1) parallel testing
(2) Easy to understand annotations
(3) Testcase grouped more easily.

Note :→ TestNG installation and setup refer the screenshot Pdf provided.

TestNG Annotations

@BeforeSuit : The annotated method will be run before all tests in this suite have run.

@AfterSuit : The annotated method will be run after all tests in this suite have run.

@BeforeTest : The annotated method will be run before any test method belonging to the classes inside the <test> tag is run

@AfterTest : The annotated method will run after all the test methods belonging to the classes inside the <test> tag have run.

@BeforeGroups : The list of groups that this configuration method will run before.

@AfterGroups : The list of groups ~~for test method~~ that this configuration method will run after.

@Beforeclass : The annotated method will be run after the all test methods in the current class have been run

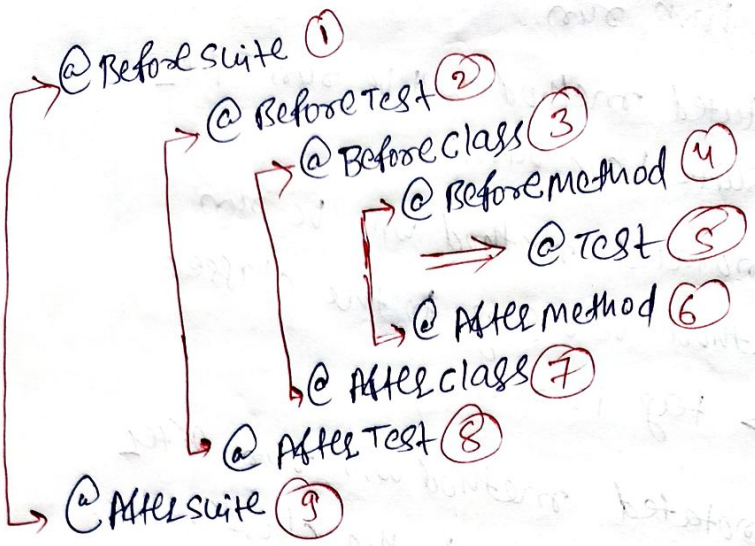
@Afterclass : The annotated method will be run after all the test methods in the current class have been run.

@Beforemethod : The annotated method will be run before each test method.

@Aftermethod : The annotated method will be run after each test method.

@Test : makes method as a test method, Equal to a test case.

TestNG Sequence & Execution



Basic Simple TestNG Program Example:

```
Public class TestNGTest {

    @Test
    Public void HelloTesting()
        {
        System.out.println("Hello welcome to TestNG...!!");

        }
}
```

TestNG Annotations Example program

```
public class TestNGAnnotations {

    @Test
    Public void method1() {

            Sop("from method 1");

    }
```

```
@Test
Public void method2() {
    SOP("From method 2");
}

@Beforemethod
Public void beforemethod()
{
SOP("From Beforemethod");
}

@Beforeclass
Public void beforeclass() {
    SOP("from beforeclass");
}

@AfterMethod
Public void aftermethod() {
    SOP("from after method");
}

@AfterClass
Public void afterclass() {
    SOP("from afterclass");
}

@BeforeTest
Public void beforeTest() {
    SOP("from BeforeTest");
}

@AfterTest
Public void afterTest() {
    SOP("from afterTest");
}
```

```
@BeforeSuite
Public void beforeSuite() {
    Sop("from beforeSuite");
}
@AfterSuite
Public void afterSuite() {
    Sop("After suite from");
}
}
```

Assertion In TestNG

Assert class

Assertions in TestNG are a way to verify that the expected result and the actual result matched or not.

Ex: Assert.MethodName(actual, expected);

Assert.MethodName(actual, expected, message);

Common TestNG Assert methods.

(1) assertEquals(actual, expected)
- Accepts two string arguments
- it will check both are equal, if not it will fail the test.

(2) assertEquals(actual, expected, message)
- It accepts three string arguments
- it test whether both are same, if not it will fail
- also, it will throw a message that we provide

Note: we use other method also assertNULL(obj), assertNOTNULL(obj);

(3) assertEquals (boolean actual, boolean expected)
- tests for equality, if not it fail the test

(4) assertEquals (java.util.Collection actual, java.util.Collection expected, message)
- it accepts two collection type objects
- it checks whether they hold same elements and with the same order.
- it will fail the test if the above condition doesn't meet.
- also, you can show message in the report.

(5) assertTrue (condition)
- It accepts one boolean argument, tests for given condition is true.
- if it fails then <AssertError> would occur.

(6) assertTrue (condition, message)
- accepts two arguments condition boolean and string message.
- it assert that the given condition is true
- if it fail then assertError would occurs with the message you passed

(7) assertFalse (condition)
- accepts one boolean argument and checks that the given condition is false.
- if it doesn't pass, then an <assertError> would occur.

(8) assertFalse (condition, message)
- accepts one boolean condition and string message that asserts the given condition is false
- if it fails then AssertError thrown with message.

Different types of Asserts in TestNG

There are two types

1. Hard Assert
2. Soft Assert

* Hard Assert are those asserts that stop the test execution when an assert statement fails. and the subsequent assert statement are therefore not validated

used when when we have a element without whose validation, asserting other elements is useless.

Hard asserts are the default type of asserts in TestNG.

* Soft Asserts are just opposite of Hard asserts

In soft assert the subsequent asserting keep on running even though one assert validation fails. test execution does't stop.

- Not the default assert method added in TestNG.
- available at Package org.testng.asserts.SoftAssert.
- If you want execution to be continued without the validation failure and also want to see the exception errors we go for soft assert
- Soft asserts also known as "verify"

Ex:

```
SoftAssert sa = new SoftAssert();
sa.assertEqual(actual, expected);  //F
sa.assertEqual(actual1, expected1);  //P
sa.assertEqual(actual2, expected2);  //F
sa.assertAll();
```

Note: assertAll statement is required to see the exceptions : otherwise we can't know what passed and what failed.

Annotations attributes in TESTNG

(1) AlwaysRun

if set to true, this test method will always be run even if it depends on a method that failed.

(2) dataProvider

The name of the data Provider for this test method

(3) dataProviderClass

The class where to look for the data Provider. If not specified, the data Provider will be looking at the class of the current test method or one of its base class if this attribute is specified, the data Provider method needs to be static in the specified class

(4) dependsOnGroups

the list of groups this method depends on

(5) dependsOnMethods

The list of methods this method depends on

(6) description

The description for this method

(7) enabled

whether methods on this class/method are enabled

(8) expectedExceptions

The list of exceptions that a test method is expected to throw, if no exception or a different than one on this list is thrown, this test will be marked as failure.

(9) groups

The list of groups this class/method belongs to.

(10) invocationCount

The number of times this method should be invoked.

(11) invocationTimeOut

The maximum number of ms this test should take for the cumulated time of all the invocation counts. This attribute will be ignored if invocation count is not specified.

(12) Priority

- The priority for this test method, lower priorities will be scheduled first.
- First Preference : Non-Prioritized method based on alphabetical order.
- second preference : Prioritized methods with highest priority (i.e 0). if some priority level then, based on alphabetical order of same Prioritized methods.

(13) Success Percentage

The percentage of success expected from this method.

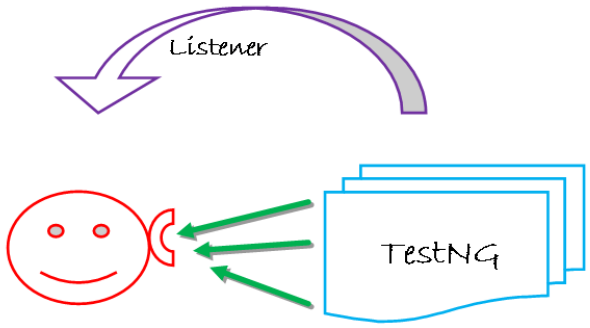
~~(14) SingleThreaded~~

(14) timeout
threadPoolSize }

- Size of thread Pool for method
- method will be invoked multipal threads as per invocationCount
- invocation count not given ignored.

TestNG Listeners

Listeners are TestNG annotations that literally “listen” to the events in a script and modify TestNG behaviour accordingly. These listeners are applied as interfaces in the code. For example, the most common usage of listeners occurs when taking a screenshot of a particular test that has failed along with the reason for its failure. Listeners also help with logging and generating results.



Types of TestNG Listeners

Listeners are implemented in code via interfaces to modify TestNG behaviour. Listed below are the most commonly used TestNG listeners:

- IAnnotationTransformer
- IExecutionListener
- IHookable
- IInvokedMethodListener
- IMethodInterceptor
- IReporter
- ISuiteListener
- ITestListener

These listeners can be implemented in TestNG in the following ways:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
<listener class-name="Listener"/>
</listeners>
  <test name="Test">
    <classes>
      <class name="ParallelTestWithMultiThread"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

- Using the listener annotation(@Listeners) in a testNG class as below:
  @Listeners(com.Example.Listener.class)

Note: The **@Listener** will be, by default, applicable to the complete suite – similar to the **testNG.xml** file. One can choose to restrict its scope to the current class. It is a best practice to apply listeners in the testNG.xml file for neat framework design and understanding.

ITestListener:

This is the most frequently used TestNG listener. ITestListener is an interface implemented in the class , and that class overrides the ITestListener defined methods. The ITestListener listens to the desired events and executes the methods accordingly.

The *ITestListener* contains the following methods:

- ***onStart***: *This method invokes when the test class is instantiated and before executing any test method*.

  Syntax: ***void onStart(ITestContext context);***

- ***onFinish***: *This method invokes when all the test methods have run, and calling of all of their configuration methods happens*.

Syntax: ***void onFinish(ITestContext context);***

- ***onTestStart***: *This method invokes every time a test method is called and executed*.

  Syntax: ***void onTestStart(ITestResult result);***

- ***onTestSuccess***: *This method is invoked every time a test case passes (succeeds)*.

  Syntax: ***void onTestSuccess(ITestResult result);***

- ***onTestFailure***: *This method invokes every time a test case fails*.

  Syntax: ***void onTestFailure(ITestResult result);***

- ***onTestSkipped***: *This method invokes every time a test skips*.

  Syntax: ***void onTestSkipped (ITestResult result);***

- ***onTestFailedButWithinSuccessPercentage***: *This method invokes when the test method fails as a whole but has passed a certain success percentage, which is defined by the user*.

  Syntax: ***void onTestFailedButSuccessPercentage (ITestResult result);***

Now in the above syntaxes, you must be wondering about the words ***ITestContext*** and ***ITestResult***. So, the term ***'ITestResult' is an interface that describes the result of the test***. Therefore '*result*' has been passed as its instance in the syntax. Whereas ' *ITestContext* ' ***is a class that defines an instance*** '*context*', ***which contains all the information about the test run***. We can use this information to pass to our listeners, and they can proceed with their queries.

Now, let's look at an example showcasing the use of this listener.

Below is a listener class that implements ITestListener:

```
import org.testng.ITestContext;

import org.testng.ITestListener;

import org.testng.ITestResult;

public class MyListener implements ITestListener {

@Override
```

```
public void onFinish(ITestContext contextFinish) {

System.out.println("onFinish method finished");

}

@Override

public void onStart(ITestContext contextStart) {

System.out.println("onStart method started");

}

@Override

public void onTestFailedButWithinSuccessPercentage(ITestResult result) {

System.out.println("Method failed with certain success percentage"+ result.getName());

}

@Override

public void onTestFailure(ITestResult result) {

System.out.println("Method failed"+ result.getName());

}

@Override

public void onTestSkipped(ITestResult result) {

System.out.println("Method skipped"+ result.getName());

}

@Override

public void onTestStart(ITestResult result) {

System.out.println("Method started"+ result.getName());
```

```
}



@Override

public void onTestSuccess(ITestResult result) {

System.out.println("Method passed"+ result.getName());

}

}
```

The class below contains four methods, showcasing one method being passed, one method being failed, one method being skipped and one method being passed with a defined success percentage:

```
import org.testng.Assert;

import org.testng.SkipException;

import org.testng.annotations.Test;

import org.testng.asserts.SoftAssert;



public class ItestListenerWithExample {

int i=0;

@Test

public void testMethod1()

{

System.out.println("This method will pass and will invoke the onTestSuccess method of ITestlistener");

int i=10;
```

```
Assert.assertEquals(i, 10);

}



@Test

public void testMethod2()

{

System.out.println("This method will fail and will invoke the onTestFailure method of ITestlistener");

int i=10;

Assert.assertEquals(i, 11);

}



@Test

Public  void testMethod3()

{

System.out.println("This method will skip and will invoke the onTestSkipped method of ITestlistener");

throw new SkipException("Skipping this test case.");

}



@Test(successPercentage=50, invocationCount=5)

public  void testMethod4()

{

i++;
```

```
System.out.println("Test Failed But Within Success Percentage Test Method, invocation count: " + i);

if (i == 1 || i == 2) {

System.out.println("this will be Failed");

Assert.assertEquals(i, 100);

}

}

}
```

Below is the testNG xml file:

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

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

<suite name="Suite">

	<listeners>

		<listener class-name="com.abc.MyListener"/>

	</listeners>

	<test name="Listeners_program">

		<classes>

		<class name="com.abc.ItestListenerWithExample"></class>

		</classes>

	</test>

</suite>
```

```
[RemoteTestNG] detected TestNG version 6.9.9
[TestNG] Running:
  C:\Users\ss251550\eclipse-workspace_automate\Automate_Active_MQ_Process\testng.xml

onStart method started
Method startedtestMethod1
This method will pass and will invoke the onTestSuccess method of ITestlistener
Method passedtestMethod1
Method startedtestMethod2
This method will fail and will invoke the onTestFailure method of ITestlistener
Method failedtestMethod2
Method startedtestMethod3
This method will skip and will invoke the onTestSkipped method of ITestlistener
Method skippedtestMethod3
Method startedtestMethod4
Test Failed But Within Success Percentage Test Method, invocation count: 1
this will be Failed
Method failed with certain success percentagetestMethod4
Method startedtestMethod4
Test Failed But Within Success Percentage Test Method, invocation count: 2
this will be Failed
Method failed with certain success percentagetestMethod4
Method startedtestMethod4
Test Failed But Within Success Percentage Test Method, invocation count: 3
Method passedtestMethod4
Method startedtestMethod4
Test Failed But Within Success Percentage Test Method, invocation count: 4
Method passedtestMethod4
Method startedtestMethod4
Test Failed But Within Success Percentage Test Method, invocation count: 5
Method passedtestMethod4
onFinish method finished

===============================================
Parent_Suite
Total tests run: 8, Failures: 3, Skips: 1
===============================================
```

Exploring testng.xml

1. testng.xml with class declaration

```
<?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.atipune.testng.basics.CrossBrowserTesting"/>
      <class name="com.atipune.testng.basics.TestNgAttributes"/>
      <class name="com.atipune.testng.basics.TestNGAnnotations"/>
      <class name="com.atipune.testng.basics.SoftAssertTest"/>
      <class name="com.atipune.testng.basics.groupAll"/>
      <class name="com.atipune.testng.basics.TestNGAsserts"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```


2. testng.xml with Package declaration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="SuiteWithPackage">
  <test thread-count="5" name="TestPackage">
  <packages>
     <package name="com.atipune.testng.basics" />
  </packages>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

3. testng.xml with *Suite With Package And Test Configuration*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="SuiteWithPackageAndTestConfig" thread-
count="2">
        <packages>
     <package name="com.atipune.testng.basics" />
     </packages>


     <test name="TestPackageConfig">
    <groups>
    <define name="all">
     <include name="regresion"/>
     <include name="sanity"/>
    </define>

    <run>
     <include name="all" />
     <exclude name="broken"/>
    </run>
    </groups>
     </test> <!-- Test -->

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

4.Executing multiple suites

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="SuperSuite"  >
     <suite-files >
          <suite-file path="testng.xml"/>
          <suite-file path="testng2.xml"/>
          <suite-file path="testng3.xml"/>
          <suite-file path="testng4.xml"/>
     </suite-files>
</suite> <!-- Suite -->
```

5.Suite Parallel attribute

The parallel attribute on the <suite> tag can take one of following values:

<suite name="My suite" parallel="methods" thread-count="5">

<suite name="My suite" parallel="tests" thread-count="5">

<suite name="My suite" parallel="classes" thread-count="5">

<suite name="My suite" parallel="instances" thread-count="5">

- parallel="methods": TestNG will run all your test methods in separate threads.Dependent methods will also run in separate threads but they will respect the order that you specified.
- parallel="tests": TestNG will run all the methods in the same <test> tag in the same thread, but each <test> tag will be in a separate thread. This allows you to group all your classes that are not thread safe in the same <test> and guarantee they will all run in the same thread while taking advantage of TestNG using as many threads as possible to run your tests.
- parallel="classes": TestNG will run all the methods in the same class in the same thread,but each class will be run in a separate thread.
- parallel="instances": TestNG will run all the methods in the same instance in the same thread,but two methods on two different instances will be running in different threads.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="ParallelSuite" parallel="methods" thread-count="2">

  <test name="ParallelTest1">
    <groups>
    <define name="all">
      <include name="regresion"/>
      <include name="sanity"/>
    </define>

    <run>
      <include name="all" />
      <exclude name="broken"/>
    </run>

    </groups>

    <packages>
      <package name="com.atipune.testng.basics"/>
    </packages>

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

  <test name="ParallelTest2">
    <classes>
           <class name="com.atipune.testng.basics.TestNGAsserts">
           <methods>
           <include name="verifyHomePageTitle"/>
           <include name="verifyFirstBookTitle"/>
           </methods>
      </class>
    </classes>
  </test> <!-- Test -->


  <test name="ParallelTest3">
    <classes>
           <class name="com.atipune.testng.basics.TestNgAttributes">
    </class>
    </classes>
  </test> <!-- Test -->

  </suite>
```

5.Cross browser Testing with TestNg

Cross-browser testing is the process of testing our website on different browsers and operating systems. With cross-browser testing, we make sure that the site is rendered the same in every browser

Need Cross-browser testing?

- Cross-browser testing using TestNG ensure a better performance on different browsers and OS.
- Image orientations mess up a lot of the time. We can take care of it.
- The tester and the developer can assure how JavaScript renders on different browsers.
- One can track Font-size issues.
- The unsupported tags can be revealed, which can be taken care of by turnaround codes.

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="CrossBrowserTestingSuite" parallel="tests" thread-
count="5">

 <test name="FirefoxTest">
     <parameter name="browser" value="firefox" />
     <classes>
        <class name="com.atipune.testng.basics.CrossBrowserTesting" />
     </classes>
 </test>


 <test name="ChromeTest">
     <parameter name="browser" value= "chrome" />
     <classes>
        <class name="com.atipune.testng.basics.CrossBrowserTesting" />
     </classes>
 </test>

 <test name="EdgeTest">
     <parameter name="browser" value= "edge" />
     <classes>
        <class name="com.atipune.testng.basics.CrossBrowserTesting" />
     </classes>
 </test>

</suite>
```