**TestNG Tutorial**

**packages**

Testng tutorial we can add the package ‘s also

<suite>

<test>

<packages>

<package name=com.test/>

</packages>

</test>

</suite>

This will consider will retain all the classes that has TestNG notation

@Beforesuite: before all the <tests> are started

@aftersuite

:after all the <test> are completed

@BeforeTest: annoted methoed will run before starting the any test methoed from all the <test tags>

@Aftertest:after all the test are finished

@Beforeclass: is class dependent if you have written in the class this will execute before the test methoed in that class starts the execution

@Afterclass: after all the test methoed in that test after class will be executed

@Beforemethoed: before test methoed starts the execution

@Aftermethoed:after test methoed is executed

@Test: test methoed is run

Complete structure of the testng file can be like this

<suite>

<groups>

<run>

<include>

<exclued>

</run>

</groups>

<test>

<classes> / <packages>

<class name=””>/<pakcage>

<metheods>

<include name=”methoedname”>

</methoeds>

</classes>/<packes>

**To allow the Test to return the value**

est methods are annotated with @Test. Methods annotated with @Test that happen to return a value will be ignored, unless you set allow-return-values to true in your testng.xml:

|  |
| --- |
| <suite allow-return-values="true"> |

|  |
| --- |
|  |

|  |
| --- |
| or |

|  |
| --- |
|  |

|  |
| --- |
| <test allow-return-values="true"> |
| **Groups of Groups**   |  | | --- | | test name="Regression1"> |  |  | | --- | | <groups> |  |  | | --- | | <define name="functest"> |  |  | | --- | | <include name="windows"/> |  |  | | --- | | <include name="linux"/> |  |  | | --- | | </define> |  |  | | --- | |  |  |  | | --- | | <define name="all"> |  |  | | --- | | <include name="functest"/> |  |  | | --- | | <include name="checkintest"/> |  |  | | --- | | </define> |  |  | | --- | |  |  |  | | --- | | <run> |  |  | | --- | | <include name="all"/> |  |  | | --- | | </run> |  |  | | --- | | </groups> |  |  | | --- | |  |  |  | | --- | | <classes> |  |  | | --- | | <class name="test.sample.Test1"/> |  |  | | --- | | </classes> |  |  | | --- | | </test> | |

**Parameters in Test ng**

<suite>

<paramtere name=”xyz” value=”abc”>

<test>

<classes>

<class name=org.prasanna.temp>

</classes>

</test>

</suite>

@prameter({“xyz”})

@Test

Public void test(String parm1)

{

Sop(param1)

}

**Optional parameter**

<suite>

<paramtere name=”xyz” value=”abc”>

<test>

<classes>

<class name=org.prasanna.temp>

</classes>

</test>

</suite>

Public void (@optional (“prasanna”) String pram1 )

{ sop(parm1)

}

On any method that already has a @Test, @Before/After or @Factory annotation

DataProvider in TestNG

[1.@Dataprovider(name=”prasanna”)](mailto:1.@Dataprovider(name=”prasanna”))

Public object[][] givedata()

{

Return new object[][]{{“prasanna”,”Testing testng”},{“MKV”,”TestNG”}};

}

@Test(dataprovider=”prasanna”)

Public void testdataprovider(String s1,String s2)

{

Sop(s1+s2)

}

--By default, the data provider will be looked for in the current test class or one of its base classes

If you want to put your data provider in a different class, it needs to be a static method and you specify the class where it can be found in thedataProviderClass attribute:

|  |
| --- |
| public class StaticProvider { |

|  |
| --- |
| @DataProvider(name = "create") |

|  |
| --- |
| public static Object[][] createData() { |

|  |
| --- |
| return new Object[][] { |

|  |
| --- |
| new Object[] { new Integer(42) } |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
| } |

|  |
| --- |
|  |

|  |
| --- |
| public class MyTest { |

|  |
| --- |
| @Test(dataProvider = "create", dataProviderClass = StaticProvider.class) |

|  |
| --- |
| public void test(Integer n) { |

|  |
| --- |
| // ... |

|  |
| --- |
| } |

|  |
| --- |
| } |
| **Dependencies with annotations** |

There are 2 ways of dependencies in the testng

1.Hard dependencie : this will run only when dependent method is run first and passed

@Test(dependsonmethod=”Test1”)

Public void showmsg()

{sop(“show msg”)

}

@Test

Public void Test()

{

Sop(“show test”)

}

Showmsg will execute only when test method is passed

2.softdependencie

@Test(dependsonmethod=”Test1”,alwaysrun=true)

Public void showmsg()

{sop(“show msg”)

}

@Test

Public void Test()

{

Sop(“show test”)

Assert.fail();

}

This will ensure that test methoed would run first even when this fails showmsg will run

-If a method depended upon fails and you have a hard dependency on it (alwaysRun=false, which is the default), the methods that depend on it are **not** marked as FAIL but as SKIP

#### TestNG Listeners

How to specify the TestNG listener

<suite>

<Listners>

<listener class-name=”org.prasanna.Report”/>

<listener class-name=”org.prasanna.test” />

</listeners>

Another way to provide the listeners is

@Listner({org.prasanna.report,org.prasanna.test})

Public void test()

{

}

@Listeners annotation can contain any class that extends org.testng.ITestNGListener **except** IAnnotationTransformer and IAnnotationTransformer2. The reason is that these listeners need to be known very early in the process so that TestNG can use them to rewrite your annotations, therefore you need to specify these listeners in your testng.xml file.

**Listening to method invocation :**

Any class that implements IinvokemethodlistListner would have to override

|  |
| --- |
| public interface IInvokedMethodListener extends ITestNGListener { |

|  |
| --- |
| void beforeInvocation(IInvokedMethod method, ITestResult testResult); |

|  |
| --- |
| void afterInvocation(IInvokedMethod method, ITestResult testResult); |

|  |
| --- |
| } |

**public** **class** Beforesuite **implements** ISuiteListener{

@Override

**public** **void** onStart(ISuite suite) {

System.*out*.println("From ISuiteListener class ----suite is started"+suite.getName());

}

@Override

**public** **void** onFinish(ISuite suite) {

// **TODO** Auto-generated method stub

System.*out*.println("From ISuiteListener class ---- suite is finished"+suite.getName());

}

}

So Testng will devide the methoeds that it need to execute in two parts :

1.methoeds that need to be executed in sequence that methods that has dependencies and dependents

2.methoeds that can run in any order

To handle these methods tesng gives you the interface

|  |
| --- |
| public interface IMethodInterceptor { |

|  |
| --- |
|  |

|  |
| --- |
| List<IMethodInstance> intercept(List<IMethodInstance> methods, ITestContext context); |

|  |
| --- |
|  |

|  |
| --- |
| } |