Annotations

We'll cover the following topics:

* Annotations in TestNG
* Before and After annotations
* Test annotation
* Disabling a test
* Exception test
* Time test
* Parameterization of test
* Passing parameters to the test methods
* Using DataProvider for parameterized tests

Refer below link for further information:

<https://testng.org/doc/documentation-main.html>

Annotation is a feature introduced in Java 5 and is used to add metadata (data about data) to Java source code.

Annotations in TestNG:

One of the basic annotations of TestNG is the @Test annotation. **This annotation marks a method or a class as part of the TestNG test.** If applied at class level this annotation will mark all the public methods present inside the class as test methods for TestNG test. It supports lot of attributes which you can use along with the annotation, which will enable you to use the different features provided by TestNG.

Below table shows attributes for @Test

|  |  |
| --- | --- |
| **Attributes** | **Explanation** |
| alwaysRun | If set to true, this test method will always be run even if it depends on a method that failed. |
| dataProvider | The name of the data provider for this test method. |
| dataProviderClass | The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class. |
| dependsOnGroups | The list of groups this method depends on. |
| dependsOnMethods | The list of methods this method depends on. |
| description | The description for this method. |
| enabled | Whether methods on this class/method are enabled. |
| 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 a failure. |
| groups | The list of groups this class/method belongs to. |
| invocationCount | The number of times this method should be invoked. |
| invocationTimeOut | The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified. |
| priority | The priority for this test method. Lower priorities will be scheduled first. |
| successPercentage | The percentage of success expected from this method |
| singleThreaded | If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with parallel="methods". This attribute can only be used at the class level and it will be ignored if used at the method level. Note: this attribute used to be called sequential (now deprecated). |
| timeOut | The maximum number of milliseconds this test should take. |
| threadPoolSize | The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. Note: this attribute is ignored if invocationCount is not specified |

Before and After annotations are mainly used to execute a certain set of code before and after the execution of test method.

TestNG provides five different kinds of Before and After annotation options:

@BeforeSuite/@AfterSuite

@BeforeTest/@AfterTest

@BeforeGroups/@AfterGroups

@BeforeClass/@AfterClass

@BeforeMethod/@AfterMethod

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

@AfterSuite: 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 be run after all the test methods belonging to the classes inside the <test> tag have run.

@BeforeClass: The annotated method will be run before the first test method in the current class is invoked.

@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.

@BeforeGroups: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

@AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

**package** pack1;

**import** org.testng.annotations.AfterClass;

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

**import** org.testng.annotations.BeforeClass;

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

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

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.BeforeClass;

**import** org.testng.annotations.BeforeGroups;

**import** org.testng.annotations.AfterClass;

**import** org.testng.annotations.AfterGroups;

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

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

**import** org.testng.annotations.BeforeSuite;

**import** org.testng.annotations.AfterSuite;

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

@Test

**public** **void** LoginTest() {

System.***out***.println("In method LoginTest");

}

@Test

**public** **void** LogoffTest() {

System.***out***.println("In method LogoffTest");

}

@BeforeGroups

**public** **void** beforeGroup() {

System.***out***.println("@beforeGroup Test1");

}

@AfterGroups

**public** **void** afterGroup() {

System.***out***.println("@afterGroup Test1");

}

@BeforeMethod

**public** **void** beforeMethod() {

System.***out***.println("@BeforeMethod Test1");

}

@AfterMethod

**public** **void** afterMethod() {

System.***out***.println("@AfterMethod Test1");

}

@BeforeClass

**public** **void** beforeClass() {

System.***out***.println("@BeforeClass Test1");

}

@AfterClass

**public** **void** afterClass() {

System.***out***.println("@AfterClass Test1");

}

@BeforeTest

**public** **void** beforeTest() {

System.***out***.println("@BeforeTest Test1");

}

@AfterTest

**public** **void** afterTest() {

System.***out***.println("@AfterTest Test1");

}

@BeforeSuite

**public** **void** beforeSuite() {

System.***out***.println("@BeforeSuite Test1");

}

@AfterSuite

**public** **void** afterSuite() {

System.***out***.println("@AfterSuite Test1");

}

}

Below is the output when Test1.java is run independently as TestNG Test

[RemoteTestNG] detected TestNG version 6.14.2

@BeforeSuite Test1

@BeforeTest Test1

@BeforeClass Test1

@BeforeMethod Test1

In method LoginTest

@AfterMethod Test1

@BeforeMethod Test1

In method LogoffTest

@AfterMethod Test1

@AfterClass Test1

@AfterTest Test1

PASSED: LoginTest

PASSED: LogoffTest

===============================================

Default test

Tests run: 2, Failures: 0, Skips: 0

===============================================

@AfterSuite Test1

===============================================

Default suite

Total tests run: 2, Failures: 0, Skips: 0

===============================================

**package** pack1;

**import** org.testng.annotations.AfterClass;

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

**import** org.testng.annotations.BeforeClass;

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

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

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.BeforeClass;

**import** org.testng.annotations.AfterClass;

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

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

**import** org.testng.annotations.BeforeSuite;

**import** org.testng.annotations.AfterSuite;

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

@Test

**public** **void** sendMessagetest() {

System.***out***.println("sendMessagetest method Test2");

}

@Test

**public** **void** sendemailtest() {

System.***out***.println("sendemailtest method Test2");

}

@BeforeMethod

**public** **void** beforeMethod() {

System.***out***.println("beforeMethod Test2");

}

@AfterMethod

**public** **void** afterMethod() {

System.***out***.println("aftermethod Test2");

}

@BeforeClass

**public** **void** beforeClass() {

System.***out***.println("Beforeclass Test2");

}

@AfterClass

**public** **void** afterClass() {

System.***out***.println("afterClass Test2");

}

@BeforeTest

**public** **void** beforeTest() {

System.***out***.println("beforeTest Test2");

}

@AfterTest

**public** **void** afterTest() {

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

}

@BeforeSuite

**public** **void** beforeSuite() {

System.***out***.println("beforeSuite Test2");

}

@AfterSuite

**public** **void** afterSuite() {

System.***out***.println("afterSuite Test2");

}

}

Below is the output when Test2.java is run independently as TestNG Test

[RemoteTestNG] detected TestNG version 6.14.2

beforeSuite Test2

beforeTest Test2

Beforeclass Test2

beforeMethod Test2

sendMessagetest method Test2

aftermethod Test2

beforeMethod Test2

sendemailtest method Test2

aftermethod Test2

afterClass Test2

afterTest test2

PASSED: sendMessagetest

PASSED: sendemailtest

===============================================

Default test

Tests run: 2, Failures: 0, Skips: 0

===============================================

afterSuite Test2

===============================================

Default suite

Total tests run: 2, Failures: 0, Skips: 0

@Test

One of the basic annotations of TestNG is the Test annotation. This annotation marks a method or a class as part of the TestNG test. If applied at class level this annotation will mark all the public methods present inside the class as test methods for TestNG test. It supports lot of attributes which you can use along with the annotation, which will enable you to use the different features provided by TestNG. The following is a list of attributes supported by the Test annotation:

|  |  |
| --- | --- |
| **@Test** | **Marks a class or a method as part of the test.** |
| alwaysRun | If set to true, this test method will always be run even if it depends on a method that failed. |
| dataProvider | The name of the data provider for this test method. |
| dataProviderClass | The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class. |
| dependsOnGroups | The list of groups this method depends on. |
| dependsOnMethods | The list of methods this method depends on. |
| Description | The description for this method. |
| Enabled | Whether methods on this class/method are enabled. |
| 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 a failure. |
| Groups | The list of groups this class/method belongs to. |
| invocationCount | The number of times this method should be invoked. |
| invocationTimeOut | The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified. |
| Priority | The priority for this test method. Lower priorities will be scheduled first. |
| successPercentage | The percentage of success expected from this method |
| singleThreaded | If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with parallel="methods". This attribute can only be used at the class level and it will be ignored if used at the method level. Note: this attribute used to be called sequential (now deprecated). |
| Timeout | The maximum number of milliseconds this test should take. |
| threadPoolSize | The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. Note: this attribute is ignored if invocationCount is not specified |

Below class shows usage of attributes like dependsOnMethods, alwaysRun, description

invocationCount, enabled

**package** pack3;

**import** org.testng.Assert;

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

**public** **class** TestClass {

@Test(dependsOnMethods= {"testMethodTwo"},alwaysRun=**true**,description="I am method one")

**public** **void** testMethodOne() {

System.***out***.println("Test method one.");

Assert.*assertEquals*('a', 'b');

}

@Test(invocationCount = 3,priority=3)

**public** **void** testMethodTwo() {

System.***out***.println("Test method two.");

}

@Test(description="I am method three",dependsOnMethods= {"testMethodTwo","testMethodOne"},alwaysRun=**false**)

**public** **void** testMethodThree() {

System.***out***.println("Test method three.");

}

@Test(enabled=**false**,priority=2)

**public** **void** testMethodFour() {

System.***out***.println("Test method four.");

}

@Test(priority=1)

**public** **void** testMethodFive() {

System.***out***.println("Test method five.");

}

}

**Following is the output:**

[RemoteTestNG] detected TestNG version 6.14.2

Test method five.

Test method two.

Test method two.

Test method two.

Test method one.

PASSED: testMethodFive

PASSED: testMethodTwo

PASSED: testMethodTwo

PASSED: testMethodTwo

FAILED: testMethodOne

I am method one

java.lang.AssertionError: expected [b] but found [a]

at org.testng.Assert.fail(Assert.java:96)

at org.testng.Assert.failNotEquals(Assert.java:776)

at org.testng.Assert.assertEqualsImpl(Assert.java:137)

at org.testng.Assert.assertEquals(Assert.java:118)

at org.testng.Assert.assertEquals(Assert.java:610)

at org.testng.Assert.assertEquals(Assert.java:620)

at pack3.TestClass.testMethodOne(TestClass.java:13)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)

at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)

at java.base/java.lang.reflect.Method.invoke(Unknown Source)

at org.testng.internal.MethodInvocationHelper.invokeMethod(MethodInvocationHelper.java:124)

at org.testng.internal.Invoker.invokeMethod(Invoker.java:580)

at org.testng.internal.Invoker.invokeTestMethod(Invoker.java:716)

at org.testng.internal.Invoker.invokeTestMethods(Invoker.java:988)

at org.testng.internal.TestMethodWorker.invokeTestMethods(TestMethodWorker.java:125)

at org.testng.internal.TestMethodWorker.run(TestMethodWorker.java:109)

at org.testng.TestRunner.privateRun(TestRunner.java:648)

at org.testng.TestRunner.run(TestRunner.java:505)

at org.testng.SuiteRunner.runTest(SuiteRunner.java:455)

at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:450)

at org.testng.SuiteRunner.privateRun(SuiteRunner.java:415)

at org.testng.SuiteRunner.run(SuiteRunner.java:364)

at org.testng.SuiteRunnerWorker.runSuite(SuiteRunnerWorker.java:52)

at org.testng.SuiteRunnerWorker.run(SuiteRunnerWorker.java:84)

at org.testng.TestNG.runSuitesSequentially(TestNG.java:1208)

at org.testng.TestNG.runSuitesLocally(TestNG.java:1137)

at org.testng.TestNG.runSuites(TestNG.java:1049)

at org.testng.TestNG.run(TestNG.java:1017)

at org.testng.remote.AbstractRemoteTestNG.run(AbstractRemoteTestNG.java:114)

at org.testng.remote.RemoteTestNG.initAndRun(RemoteTestNG.java:251)

at org.testng.remote.RemoteTestNG.main(RemoteTestNG.java:77)

SKIPPED: testMethodThree

I am method three

java.lang.Throwable: Method TestClass.testMethodThree()[pri:0, instance:pack3.TestClass@49e5f737] depends on not successfully finished methods

at org.testng.internal.Invoker.invokeTestMethods(Invoker.java:887)

at org.testng.internal.TestMethodWorker.invokeTestMethods(TestMethodWorker.java:125)

at org.testng.internal.TestMethodWorker.run(TestMethodWorker.java:109)

at org.testng.TestRunner.privateRun(TestRunner.java:648)

at org.testng.TestRunner.run(TestRunner.java:505)

at org.testng.SuiteRunner.runTest(SuiteRunner.java:455)

at org.testng.SuiteRunner.runSequentially(SuiteRunner.java:450)

at org.testng.SuiteRunner.privateRun(SuiteRunner.java:415)

at org.testng.SuiteRunner.run(SuiteRunner.java:364)

at org.testng.SuiteRunnerWorker.runSuite(SuiteRunnerWorker.java:52)

at org.testng.SuiteRunnerWorker.run(SuiteRunnerWorker.java:84)

at org.testng.TestNG.runSuitesSequentially(TestNG.java:1208)

at org.testng.TestNG.runSuitesLocally(TestNG.java:1137)

at org.testng.TestNG.runSuites(TestNG.java:1049)

at org.testng.TestNG.run(TestNG.java:1017)

at org.testng.remote.AbstractRemoteTestNG.run(AbstractRemoteTestNG.java:114)

at org.testng.remote.RemoteTestNG.initAndRun(RemoteTestNG.java:251)

at org.testng.remote.RemoteTestNG.main(RemoteTestNG.java:77)

===============================================

Default test

Tests run: 6, Failures: 1, Skips: 1

===============================================

===============================================

Default suite

Total tests run: 6, Failures: 1, Skips: 1

===============================================

**What happens when methods in TestNG class are not public?**

**Ans: Only public class method is run and others are ignored.**

**Note: @Test is declared at class level and not for every method.**

**package** pack3;

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

@Test

**public** **class** TestClass1 {

**void** testMethodOne() {

System.***out***.println("Test method one.");

}

**public** **void** testMethodTwo() {

System.***out***.println("Test method two.");

}

**private** **void** testMethodThree() {

System.***out***.println("Test method three.");

}

**protected** **void** testMethodFour() {

System.***out***.println("Test method three.");

}

}

Ouput:

[RemoteTestNG] detected TestNG version 6.14.2

Test method two.

PASSED: testMethodTwo

===============================================

Default test

Tests run: 1, Failures: 0, Skips: 0

===============================================

===============================================

Default suite

Total tests run: 1, Failures: 0, Skips: 0

===============================================

Exception test

While writing unit tests there can be certain scenarios where we need to verify that an exception is being thrown by the program during execution. TestNG provides a feature to test such scenarios by allowing the user to specify the type of exceptions that are expected to be thrown by a test method during execution. It supports multiple values being provided for verification. If the exception thrown by the test is not part of the user entered list, the test method will be marked as failed.

**package** pack3;

**import** java.io.IOException;

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

**public** **class** ExceptionTest {

// expected, throws and actual exception are of same type i.e. IOException

// Test case passes as expected and the actual exception is same

@Test(expectedExceptions = { IOException.**class** })

**public** **void** exceptionTestOne() **throws** IOException {

// **Java code – which actually throws IOException();**

**throw** **new** IOException();

}

// expected is of child class i.e. NullPointerException and NullPointerException

// throws and actual exception are of parent class i.e. Exception

// Test case fails as expected was child exception however actual was parent exception and hence cannot be handled

@Test(expectedExceptions = { IOException.**class**, NullPointerException.**class** })

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

**throw** **new** Exception();

}

// expected is of parent class i.e. Exception

// throws and actual exception are of child class i.e. IOException

// Test case passes as expected was parent exception however actual was child exception hence can be handled

@Test(expectedExceptions = { Exception.**class** })

**public** **void** exceptionTestThree() **throws** IOException {

**throw** **new** IOException();

}

}

**package** pack3;

**import** java.io.IOException;

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

**public** **class** ExceptionMessageTest {

// Verifies the exception message based on the exact error message thrown.

// Expected and Actual exception thrown are same - hence passes

@Test(expectedExceptions={IOException.**class**},expectedExceptionsMessageRegExp="Pass Message test")

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

**throw** **new** IOException("Pass Message test");

}

/\*\*

\* Verifies the exception message using the regular exception.

\* This test verifies that the exception message contains a text "Message" in it.

\*/

@Test(expectedExceptions={IOException.**class**},expectedExceptionsMessageRegExp=".\* Message .\*")

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

**throw** **new** IOException("Pass Message test");

}

// Verifies the exception message based on the exact error message thrown.

// Expected exception is child class i.e. IOException while Actual exception thrown is parent class i.e. Exception - hence fails

@Test(expectedExceptions={IOException.**class**},expectedExceptionsMessageRegExp="Pass Message test")

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

**throw** **new** Exception("Pass Message test");

}

// Verifies the exception message based on the exact error message thrown.

// Expected exception is parent class i.e. Exception while Actual exception thrown is child class i.e. IOException - hence passes

@Test(expectedExceptions={Exception.**class**},expectedExceptionsMessageRegExp="Pass Message test")

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

**throw** **new** IOException("Pass Message test");

}

}

Time test

While running tests there can be cases where certain tests get stuck or may take much more time than expected. In such a case you may need to mark the said test case as fail and then continue. TestNG allows user to configure a time period to wait for a test to completely execute. This can be configured in two ways:

* At suite level: This will be applicable for all the tests in the said TestNG test suite
* At each test method level: This will be applicable for the said test method and will override the time period if configured at the suite level

Let's go ahead and create a sample project to see how this feature works.

**package** pack3;

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

**public** **class** TimeSuite {

// suite level time-out = 500 hence this test fails as it exceeds the suite time-out

@Test(priority=1)

**public** **void** timeTestOne() **throws** InterruptedException {

Thread.*sleep*(700);

System.***out***.println("Inside timeTestOne");

}

// suite level time-out = 500 hence this test passes as its within the suite time-out

@Test(priority=2)

**public** **void** timeTestTwo() **throws** InterruptedException {

Thread.*sleep*(300);

System.***out***.println("Inside timeTestTwo");

}

// suite level time-out = 500 and Test time out is changed to 700. This test fails as it exceeds the test time-out

@Test(timeOut=700,priority=3)

**public** **void** timeTestThree() **throws** InterruptedException {

Thread.*sleep*(800);

System.***out***.println("Inside timeTestThree");

}

// suite level time-out = 500 at suite level and Test time out is changed to 700. This test passes as its within the test time-out

@Test(timeOut=700,priority=4)

**public** **void** timeTestFour() **throws** InterruptedException {

Thread.*sleep*(600);

System.***out***.println("Inside timeTestFour");

}

}

Corresponding testng.xml

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

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

<suite name=*"Suite"* time-out=*"500"*>

<test name=*"Test"*>

<classes>

<class name=*"pack3.TimeSuite"*/>

</classes>

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

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

**TestNG @Parameters**

One of the important features of TestNG is parameterization. This feature allows user to pass parameters to tests as arguments. This is supported by using the testng @Parameters annotation.

There are mainly two ways through which we can provide parameter values to testng tests.

* Through testng.xml XML configuration file
* Through DataProviders

1. Through testng.xml XML configuration file

If you need to pass some simple values such as String types to the test methods at runtime, you can use this approach of sending parameter values through testng XML configuration files. You have to use the @Parameters annotation for passing parameter values to the test method.

Below ParameterTest has two methods which gets parameters from testng.xml.

Note: Below can’t run independently as it will fail. It has to be run from testng.xml

**package** pack3;

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

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

**public** **class** ParameterTest {

// Following method takes two parameters as input. Value of the parameter is

// defined at suite level.

@Parameters({ "suite-param1", "suite-param2" })

@Test()

**public** **void** prameterTestTwo(String param1, String param2) {

System.***out***.println("Test two param1 is: " + param1);

System.***out***.println("Test two param2 is: " + param2);

}

// Following method takes one parameter as input. Value of the parameter is

// defined at test level.

@Parameters({ "suite-param3" })

@Test

**public** **void** prameterTestOne(String param1) {

System.***out***.println("Test one suite param1 is: " + param1);

}

}

Below is the corresponding testng.xml

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

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

<suite name=*"Suite"* time-out=*"500"*>

<test name=*"Test1"*>

<parameter name=*"suite-param1"* value=*"Jigar"*></parameter>

<parameter name=*"suite-param2"* value=*"Mehta"*></parameter>

<parameter name=*"suite-param3"* value=*"I am learning TestNg"*></parameter>

<classes>

<class name=*"pack3.ParameterTest"*/>

</classes>

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

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

Another way of passing parameters through testng.xml for ParameterTest

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

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

<suite name=*"Suite"* time-out=*"500"*>

<test name=*"Test1"*>

<parameter name=*"suite-param1"* value=*"Jigar"*></parameter>

<parameter name=*"suite-param2"* value=*"Mehta"*></parameter>

<parameter name=*"suite-param3"* value=*"I am learning TestNg"*></parameter>

<classes>

<class name=*"pack3.ParameterTest"*>

<methods>

<include name=*"prameterTestOne"*>

<!-- Overrides suite level parameters -->

<parameter name=*"suite-param3"* value=*"Hello World"*></parameter>

</include>

<include name=*"prameterTestTwo"*></include>

</methods>

</class>

</classes>

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

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

Test with @Optional annotation

**package** pack3;

**import** org.testng.annotations.Optional;

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

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

**public** **class** ParameterTest {

@Parameters({ "optional-parameter" })

@Test

**private** **void** prameterTestThree(@Optional("s1 optional value") String s1) {

System.***out***.println("s1 = " + s1);

}

}

In below testng.xml parameter name *optional-parameter is passed to prameterTestThree*

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

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

<suite name=*"Suite"* time-out=*"500"*>

<test name=*"optional-test"*>

<classes>

<class name=*"pack3.ParameterTest"*>

<methods>

<include name=*"prameterTestThree"*>

<parameter name=*"optional-parameter"*

value=*"value passed from*

*testng"*></parameter>

</include>

</methods>

</class>

</classes>

</test>

</suite>

Below is output:

[RemoteTestNG] detected TestNG version 6.14.2

s1 = value passed from testng

===============================================

Suite

Total tests run: 1, Failures: 0, Skips: 0

===============================================

In below testng.xml parameter values are not passed. Optional value from ParameterTest class is printed.

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

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

<suite name=*"Suite"* time-out=*"500"*>

<test name=*"optional-test"*>

<classes>

<class name=*"pack3.ParameterTest"*>

<methods>

<include name=*"prameterTestThree"*>

</include>

</methods>

</class>

</classes>

</test>

</suite>

Below is the output

[RemoteTestNG] detected TestNG version 6.14.2

s1 = s1 optional value

===============================================

Suite

Total tests run: 1, Failures: 0, Skips: 0

===============================================

DataProvider

One of the important features provided by TestNG is the DataProvider feature. It helps the user to write data-driven tests, that means same test method can be run multiple times with different datasets. DataProvider is the second way of passing parameters to test methods. It helps in providing complex parameters to the test methods as it is not possible to do this from XML.

To use the DataProvider feature in your tests you have to declare a method annotated by DataProvider and then use the said method in the test method using the dataProvider attribute in the Test annotation.

A Data Provider is a method on your class that returns an array of array of objects.

Create below class file and run it.

**package** pack3;

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

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

**public** **class** SameClassDataProvider {

@DataProvider(name = "dp1")

**public** Object[][] dataprovider1() {

**return** **new** Object[][] {

{ "jigar", "mehta" },

{ "abhishek", "mishra" }

};

}

@Test(dataProvider = "dp1")

**public** **void** test\_name(String firstname, String lastname) {

System.***out***.println(firstname + " " + lastname);

}

}

Below is output

[RemoteTestNG] detected TestNG version 6.14.2

jigar mehta

abhishek mishra

PASSED: test\_name("jigar", "mehta")

PASSED: test\_name("abhishek", "mishra")

===============================================

Default test

Tests run: 2, Failures: 0, Skips: 0

===============================================

===============================================

Default suite

Total tests run: 2, Failures: 0, Skips: 0

===============================================

Another example of Dataprovider

**package** pack3;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Set;

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

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

**public** **class** SameClassDataProvider {

@DataProvider(name = "dp1")

**public** Object[][] dataprovider1() {

HashMap<String, Integer> hm1 = **new** HashMap<>();

hm1.put("Chemistry", 10);

hm1.put("Maths", 20);

hm1.put("English", 10);

HashMap<String, Integer> hm2 = **new** HashMap<>();

hm2.put("Chemistry", 100);

hm2.put("Maths", 200);

hm2.put("English", 100);

**return** **new** Object[][] {

{ hm1 },

{ hm2 }

};

}

@Test(dataProvider = "dp1")

**public** **void** test\_name(HashMap<String, Integer> m1) {

Set<String> set = m1.keySet();

Iterator<String> ite = set.iterator();

**while** (ite.hasNext()) {

String s1 = ite.next();

System.***out***.println(s1 + " " + m1.get(s1));

}

}

}

**After running below is the output**

[RemoteTestNG] detected TestNG version 6.14.2

Maths 20

English 10

Chemistry 10

Maths 200

English 100

Chemistry 100

PASSED: test\_name({Maths=20, English=10, Chemistry=10})

PASSED: test\_name({Maths=200, English=100, Chemistry=100})

===============================================

Default test

Tests run: 2, Failures: 0, Skips: 0

===============================================

===============================================

Default suite

Total tests run: 2, Failures: 0, Skips: 0

===============================================

**DataProvider in different class**

DataProvider "dp1" as is defined in different class.

**package** pack2;

**import** java.util.HashMap;

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

**public** **class** DataProviderClass {

@DataProvider(name = "dp1")

**public** Object[][] dataprovider1() {

HashMap<String, Integer> hm1 = **new** HashMap<>();

hm1.put("Chemistry", 10);

hm1.put("Maths", 20);

hm1.put("English", 10);

HashMap<String, Integer> hm2 = **new** HashMap<>();

hm2.put("Chemistry", 100);

hm2.put("Maths", 200);

hm2.put("English", 100);

**return** **new** Object[][] { { hm1 }, { hm2 } };

}

}

**package** pack3;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Set;

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

**import** pack2.DataProviderClass;

**public** **class** SameClassDataProvider {

@Test(dataProvider = "dp1", dataProviderClass = DataProviderClass.**class**)

**public** **void** test\_name(HashMap<String, Integer> m1) {

Set<String> set = m1.keySet();

Iterator<String> ite = set.iterator();

**while** (ite.hasNext()) {

String s1 = ite.next();

System.***out***.println(s1 + " " + m1.get(s1));

}

}

}