**1. What is TestNG?**

TestNG in Selenium is **a Java testing framework, inspired by JUnit and NUnit**. It overcomes the constraints and disadvantages of JUnit. Its entire new set.. The suffix 'NG' stands for Next Generation, signifying the new functionalities that TestNG brings to the table.

**2. What are the types of annotations available in TestNG?**

* **@BeforeMethod**: This will be executed before every @test annotated method.
* **@AfterMethod:** This will be executed after every @test annotated method.
* **@BeforeClass:** This will be executed before first @Test method execution. It will be executed one only time throughout the test case.
* **@AfterClass:** This will be executed after all test methods in the current class have been run
* **@BeforeTest:** This will be executed before the first @Test annotated method. It can be executed multiple times before the test case.
* **@AfterTest**: A method with this annotation will be executed when all @Test annotated methods complete the execution of those classes inside the <test> tag in the TestNG.xml file.
* **@BeforeSuite**: It will run only once, before all tests in the suite are executed.
* **@AfterSuite:** A method with this annotation will run once after the execution of all tests in the suite is complete.
* **@BeforeGroups**: This method will run before the first test run of that specific group.
* **@AfterGroups**: This method will run after all test methods of that group complete their execution.

**3. What is TestNG Assert and list out some common Assertions supported by TestNG?**

Assertions in TestNG are **a way to verify that the expected result and the actual result matched or not**. If we could decide the outcome on different small methods using assertions in our test case, we can determine whether our test failed or passed overall.

There are two types of Assertion:-

1. Hard Assertions.
2. Soft assertions.

These are explained as following below.

**1. Hard Assertions :**   
When any assert statement fails this type of assertion throws an exception immediately and continues with the next test in the test suite.

Hard Assertion can be of following types:-

**1. assertEquals –**   
This is used to compare expected and actual values in the selenium webdriver. The assertion passes with no exception whenever the expected and actual values are same. But, if the actual and expected values are not same then assert fails with an exception and the test is marked as failed.

*Syntax :*

Assert.assertEquals(actual, expected);

**2. assertNotEquals –**   
This is just the opposite of assertEquals. The assertion passes with no exception whenever the expected and actual values are not same. But, if the actual and expected values are same then assert fails with an exception and the test is marked as failed.

*Syntax :*

Assert.assertNotEquals(actual, expected, message);

**3. assertTrue –**   
This type of assertion is used when you are checking if condition is true. That is when we are dealing with boolean values this assertion is used. Whenever test case passes it returns true and if condition is false then it skips the current method and jumps to next.

*Syntax :*

Assert.assertTrue(condition);

**4. assertFalse –**   
It checks if value returned is false or not. Whenever test case passes it aborts the method and gives an exception.

*Syntax :*

Assert.assertFalse(condition);

**5. assertNull –**   
This assertion checks if the object is null or not. It aborts the test if object is null and gives an exception.

*Syntax :*

Assert.assertNull(object);

**6. assertNotNull –**   
This assertion checks if object is null or not. It aborts the test if object is not null that is if object is having any value and gives an exception.

*Syntax :*

Assert.assertNotNull(object);

**2. Soft Assertion :**   
These types of Assertions are the type of assertions do not throw an exception when an assertion fails and continues with the next step after the assert statement.

**4. How to create and run TestNG.xml?**

**Steps to create TestNG XML file**

**Step 1: Create testng xml file**

i. Right click on Project folder, go to **New** and select ‘**File**‘ as shown in below image.

ii. In New file wizard, add file name as ‘**testng xml**‘ as shown in below given image and click on **Finish**button.

iii. It will add **testng xml** file under your project folder.

**Step 2 : Write xml code:**

i. Now add below given code in your testng xml file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <suite name="softwaretestingmaterial">  <test name="testngTest">  <classes>  <class name="softwareTestingMaterial.STMTestNGClass" />  </classes>  </test>  </suite> |

**Note:** You can choose any name for your Test Suite & Test Name as per your need.

ii. After giving appropriate names, now your testng xml file will looks like this:

The hierarchy in the testng xml file is very simple to understand.

Very first tag is the Suite tag<suite>, under that it is the Test tag<test> and then the Class tag<classes>. You can give any name to the suite and the test but you need to provide the correct name to the <classes> tag which is a combination of your **Package** name and **Test Case** name.

eg. **Package Name** is “softwareTestingMaterial”, **Test Case Name** is “STMTestNGClass”. So the C**lass Name** should be softwareTestingMaterial.STMTestNGClass

**Step 3 : Execute a testng xml**

Now let’s run the xml. Run the test by right click on the testng xml file and select **Run As** > **TestNG Suite**.

Once the execution is done, you could view test result under the TestNg console.

**5. How to set test case priority in TestNG?**

What is test Priority in TestNG?

In TestNG, Priority is an attribute that helps the users define the order in which they want the test cases to be executed.

When you have multiple test cases and want them to run in a particular order, you can use the Priority attribute to set test priority in TestNG. The test cases get executed following an ascending order in the priority list, and hence, test cases with lower priority will always get executed first.

**Syntax for using test Priority in TestNG**

To set test case priority in TestNG, we need to add annotation as **@Test (priority=X)**. In the below-shown example, we have given a priority of 1 to the test case.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Test(priority = 1)  public void myTestCaseWithPriority() {     try {         System.out.println("I am in my first testcase with priority=1");     } catch (Exception e) {     }  } |

**6. What is Parameterized testing in TestNG?**

**Parameterized tests** allow developers to run the same test over and over again using different values.

TestNG lets you pass parameters directly to your test methods in two different ways −

* With testng.xml
* With Data Providers

**7. How to run a group of test cases using TestNG?**

Groups in TestNG are specified in testng. xml under the <suite> or <test> tag. Groups under the <suite> tag apply to all the tests included under the <test> tag in that particular <suite>. To group tests in the source code, you have to **use the @groups attribute of the @Test annotation**

**8. What is the use of @Listener annotation in TestNG?**

@Listeners annotation **defines listeners on a test class**. @Listeners annotated method listens to certain events and keep track of test execution while performing some action at every stage of test execution.

**9. How can we create a data-driven framework using TestNG?**

**Step 1**: Go to the Eclipse IDE and create a project. Add all the dependencies for [TestNG](https://www.browserstack.com/automate/testng), Selenium and Apache POI.

**Step 2**: Create a class file to write the functionality.

import org.openqa.selenium.By;

import org.testng.Assert;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ExcelExample{

@Test(dataProvider="testdata")

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

System.setProperty("webdriver.chrome.driver", "Path of Chrome Driver");

Webdriver driver = new ChromeDriver();

driver.get("<a href="https://www.browserstack.com/users/sign\_in</a>");

driver.findElement(By.name("user[login]")).sendKeys(username);

driver.findElement(By.name("user[password]")).sendKeys(password);

driver.findElement(By.name("commit")).click();

Thread.sleep(5000);

Assert.assertTrue(driver.getTitle().matches("BrowserStack Login | Sign Into The Best Mobile & Browser Testing Tool"), "Invalid credentials");

System.out.println("Login successful");

}

@AfterMethod

void ProgramTermination() {

driver.quit();

}

@DataProvider(name="testdata")

public Object[][] testDataExample(){

ReadExcelFile configuration = new ReadExcelFile("Path\_of\_Your\_Excel\_File");

int rows = configuration.getRowCount(0);

Object[][]signin\_credentials = new Object[rows][2];

for(int i=0;i<rows;i++)

{

signin\_credentials[i][0] = config.getData(0, i, 0);

signin\_credentials[i][1] = config.getData(0, i, 1);

}

return signin\_credentials;

}

}

In the above code, there is a “TestDataExample() method” in which the user has created an object instance of another class named “ReadExcelFile”. The user has mentioned the path to the excel file. The user has further defined a *for loop* to retrieve the text from the excel workbook. But to fetch the data from the excel file, one needs to write a class file for the same.

import java.io.File;

import java.io.FileInputStream;

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

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

public class ReadExcelFile{

XSSFWorkbook work\_book;

XSSFSheet sheet;

public ReadExcelFile(String excelfilePath) {

try {

File s = new File(excelfilePath);

FileInputStream stream = new FileInputStream(s);

work\_book = new XSSFWorkbook(stream);

}

catch(Exception e) {

System.out.println(e.getMessage());

}

}

public String getData(int sheetnumber, int row, int column){

sheet = work\_book.getSheetAt(sheetnumber);

String data = sheet.getRow(row).getCell(column).getStringCellValue();

return data;

}

public int getRowCount(int sheetIndex){

int row = work\_book.getSheetAt(sheetIndex).getLastRowNum();

row = row + 1;

return row;

}

In the code above, the user has used Apache POI libraries to fetch the data from the excel file. Next, it will point to the data present in the excel file and then enter the relevant username and password to the sign in page.

**Note**: The same thing can be done using a Data provider in TestNG. But to fetch the data from the Excel sheet, the user needs Apache POI jar files.

**Note**: Please enter one valid credential to test.

**10.Mention types of data you have handled in Selenium?**

**12.What are the advantages of TestNG?**

* It generate logs.
* You can do parallel testing.
* Annotations helps to set program/function priority easy.
* Allow to generate HTML report of execution.
* You can group test cases.
* You can set test cases priorities.
* You can do data Parameterization.

**13.What is the importance of testng.xml file?**

* Test cases are executed in groups.
* Test methods can be included or excluded in the execution.
* The execution of multiple test cases from multiple java class files can be triggered.
* Comprises names of the folder, class, method.
* Capable of triggering parallel execution.
* Test methods belonging to groups can be included or excluded in the execution.

**14.How to pass parameter through testng.xml file to a test case?**

## Passing Parameters with *testng.xml*

With this technique, you define the simple parameters in the *testng.xml* file and then reference those parameters in the source files. Let us have an example to demonstrate how to use this technique to pass parameters.

### Create Test Case Class

* Create a java test class, say, ParameterizedTest1.java.
* Add test method parameterTest() to your test class. This method takes a string as input parameter.
* Add the annotation *@Parameters("myName")* to this method. The parameter would be passed a value from testng.xml, which we will see in the next step.

Create a java class file named **ParameterizedTest1.java** in **/work/testng/src**.

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class ParameterizedTest1 {

@Test

@Parameters("myName")

public void parameterTest(String myName) {

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

}

}

### Create testng.xml

Create testng.xml in **/work/testng/src** to execute test case(s).

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

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

<suite name = "Suite1">

<test name = "test1">

<parameter name = "myName" value="manisha"/>

<classes>

<class name = "ParameterizedTest1" />

</classes>

</test>

</suite>

We can also define the parameters at the <suite> level. Suppose we have defined *myName* at both <suite> and <test> levels. In such cases, regular scoping rules apply. It means that any class inside <test> tag will see the value of parameter defined in <test>, while the classes in the rest of the testng.xml file will see the value defined in <suite>.

Compile the test case class using javac.

/work/testng/src$ javac ParameterizedTest1.java

Now, run testng.xml, which will run the *parameterTest* method. TestNG will try to find a parameter named *myName* first in the <test> tag, and then, if it can’t find it, it searches in the <suit> tag that encloses it.

/work/testng/src$ java org.testng.TestNG testng.xml

Verify the output.

Parameterized value is : manisha

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

Suite1

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

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

TestNG will automatically try to convert the value specified in testng.xml to the type of your parameter. Here are the types supported −

* String
* int/Integer
* boolean/Boolean
* byte/Byte
* char/Character
* double/Double
* float/Float
* long/Long
* short/Short

## Passing Parameters with *Dataproviders*

When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a database, etc.), parameters can be passed using Dataproviders.

A Data Provider is a method annotated with **@DataProvider**. This annotation has only one string attribute: its name. If the name is not supplied, the data provider’s name automatically defaults to the method’s name. A data provider returns an array of objects.

The following examples demonstrate how to use data providers. The first example is about @DataProvider using Vector, String, or Integer as parameter, and the second example is about @DataProvider using object as parameter.

### Example 1

Here, the @DataProvider passes Integer and Boolean as parameter.

**Create Java class**

Create a java class called PrimeNumberChecker.java. This class checks if the number is prime. Create this class in **/work/testng/src**.

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;

}

}

**Create Test Case Class**

* Create a java test class, say, **ParamTestWithDataProvider1.java** in **/work/testng/src**.
* Define the method primeNumbers(), which is defined as a Data provider using the annotation. This method returns an array of objects.
* Add the test method testPrimeNumberChecker() to your test class. This method takes an Integer and Boolean as input parameters. This method validates if the parameter passed is a prime number.
* Add the annotation *@Test(dataProvider = "test1")* to this method. The attribute dataProvider is mapped to "test1".

Following are the contents of **ParamTestWithDataProvider1.java**.

import org.testng.Assert;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class ParamTestWithDataProvider1 {

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}};

}

// This test will run 4 times since we have 5 parameters defined

@Test(dataProvider = "test1")

public void testPrimeNumberChecker(Integer inputNumber, Boolean expectedResult) {

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

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

}

}

**Create testng.xml**

Create a testng.xml **/work/testng/src** to execute Test case(s).

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

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

<suite name = "Suite1">

<test name = "test1">

<classes>

<class name = "ParamTestWithDataProvider1" />

</classes>

</test>

</suite>

Compile the Test case class using javac.

/work/testng/src$ javac ParamTestWithDataProvider1.java PrimeNumberChecker.java

Now, run testng.xml.

/work/testng/src$ java org.testng.TestNG testng.xml

Verify the output.

2 true

6 false

19 true

22 false

23 true

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

Suite1

Total tests run: 5, Failures: 0, Skips: 0

**15.What is exception test in TestNG?**

TestNG provides an option of tracing the exception handling of code. You can test whether a code throws a desired exception or not. Here the **expectedExceptions** parameter is used along with the @Test annotation.

**16. How to create Group of Groups in TestNG?**

Groups in Groups

We can also specify a group within another group. The groups which are defined in another groups are known as Meta Groups.

**Let's understand through an example:**

**Step 1:** Open the Eclipse.

**Step 2:** We create a java project named as "**Groups\_in\_Groups**".

**Groups\_in\_Groups.java**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Groups\_in\_Groups
4. {
5. @Test(groups= {"Smoke"})
6. **public** **void** test1()
7. {
8. System.out.println("test1");
9. }
10. @Test(groups= {"Regression"})
11. **public** **void** test2()
12. {
13. System.out.println("test2");
14. }
15. @Test
16. **public** **void** test3()
17. {
18. System.out.println("test3");
19. }}

**Step 3:** Now we create a testng.xml file where we configure the above class.

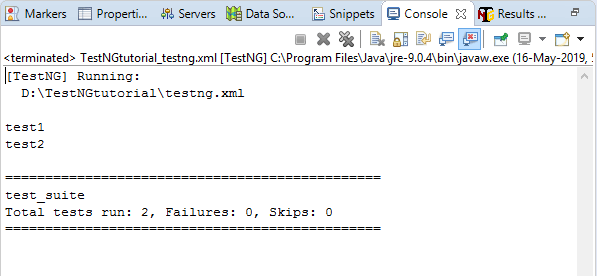
**testng.xml file**

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3. <suite name="test\_suite">
4. <test name="Groups in Groups">
5. <groups>
6. <define name="Group 1">
7. <include name="Smoke"/>
8. <include name="Regression"/>
9. </define>
10. <run>
11. <include name="Group 1"/>
12. </run>
13. </groups>
14. <classes>
15. <**class** name="com.javatpoint.Groups\_in\_Groups"/>
16. </classes>
17. </test> <!-- Test -->
18. </suite> <!-- Suite -->

In the above xml file, we define a new group within another group named as "Group 1" and we have include those test cases which are tagged with "Smoke" and "Regression".

**Step 4:** Run the testng.xml file.

**Output**



**17.How to run test cases in parallel using TestNG?**

TestNG allows to run tests parallelly or in separate threads in following ways:

* **Parallel suites:**If you are running several suite files (e.g. testng1.xml testng2.xml"), and you want each of these suites to be run in a separate thread. Use the following command line flag to specify the size of a thread pool:
* java org.testng.TestNG -suitethreadpoolsize 3 testng1.xml testng2.xml testng3.xml

* **Parallel tests, classes and methods:**Use *parallel* attribute on the*<suite>* tag respectively (for methods,test,classes, instances).
* <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 testing is used heavily with Selenium because of the importance of cross-browser testing. With so many browsers in market today with a different version, create a browser matrix and run the tests parallelly. This will save us lot of time and other resources.

## Create testng.xml

Create testng.xml in **/work/testng/src** to execute test case(s).

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

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

<suite name = "Parallel Testing Suite">

<test name = "Parallel Tests" parallel = "methods">

<classes>

<class name = "TestParallel" />

</classes>

</test>

</suite>

Compile the TestParallel class using javac.

/work/testng/src$ javac TestParallel.java

Now, run testng.xml.

/work/testng/src$ java org.testng.TestNG testng.xml

Verify the output.

Inside method1()

Inside method2()

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

Parallel Testing Suite

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

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

**18.How to exclude a particular test method from a test case execution?**

You can disable or exclude the test cases by **using the enable attribute to the @Test annotation and assign False value to the enable attribute**.

**19..How to exclude a particular test group from a test case execution?**

**In TestNG, test cases can be disabled in two ways:**

* You can disable the test case in a @Test annotation.
* You can disable the test case in the XML file.

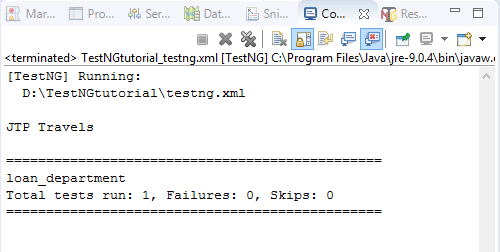
## TestNG @Test enable parameter

You can disable or exclude the test cases by using the enable attribute to the @Test annotation and assign False value to the enable attribute.

1. **package** day1;
2. **import** org.testng.annotations.Test;
4. **public** **class** module1
5. {
6. @Test(enabled=**false**)
7. **public** **void** test1()                                          // First test case
8. {
9. System.out.println("Hello javaTpoint!!");
10. }
12. @Test
13. **public** **void** test2()                                          // Second test case
14. {
15. System.out.println("JTP Travels");
16. }}

In the above code, we created two test cases, i.e., test1() and test2(), and we disable the first test case by assigning the **False** value to the enable attribute. On disabling the first test case, the only second test case will run.

**Output**



In the above output, we observe that only the second test case run without any failure.

**20.How to disable a test case in TestNG ?**

Sometimes, it happens that our code is not ready and the test case written to test that method/code fails. In such cases, **annotation @Test(enabled = false)** helps to disable this test case. If a test method is annotated with @Test(enabled = false), then the test case that is not ready to test is bypassed.

**In TestNG, test cases can be disabled in two ways:**

* You can disable the test case in a @Test annotation.
* You can disable the test case in the XML file.

## TestNG @Test enable parameter

You can disable or exclude the test cases by using the enable attribute to the @Test annotation and assign False value to the enable attribute.

**Let's understand through an example:**

53.6M

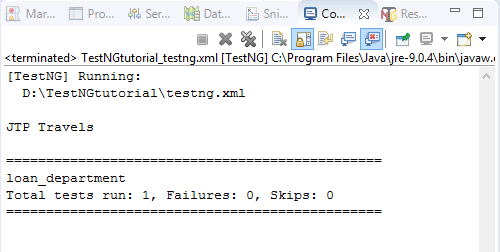
1K

C++ vs Java

1. **package** day1;
2. **import** org.testng.annotations.Test;
4. **public** **class** module1
5. {
6. @Test(enabled=**false**)
7. **public** **void** test1()                                          // First test case
8. {
9. System.out.println("Hello javaTpoint!!");
10. }
12. @Test
13. **public** **void** test2()                                          // Second test case
14. {
15. System.out.println("JTP Travels");
16. }}

In the above code, we created two test cases, i.e., test1() and test2(), and we disable the first test case by assigning the **False** value to the enable attribute. On disabling the first test case, the only second test case will run.

**Output**



In the above output, we observe that only the second test case run without any failure.

### Disable the enable attribute in the XML file.

**Step1:** Till now, we have created the test cases of the Personal loan department.

**module1.java**

1. **package** day1;
2. **import** org.testng.annotations.Test;
4. **public** **class** module1
5. {
6. @Test
7. **public** **void** test1()
8. {
9. System.out.println("Hello javaTpoint!!");
10. }
12. @Test
13. **public** **void** test2()
14. {
15. System.out.println("JTP Travels");
16. }}

**module2.java**

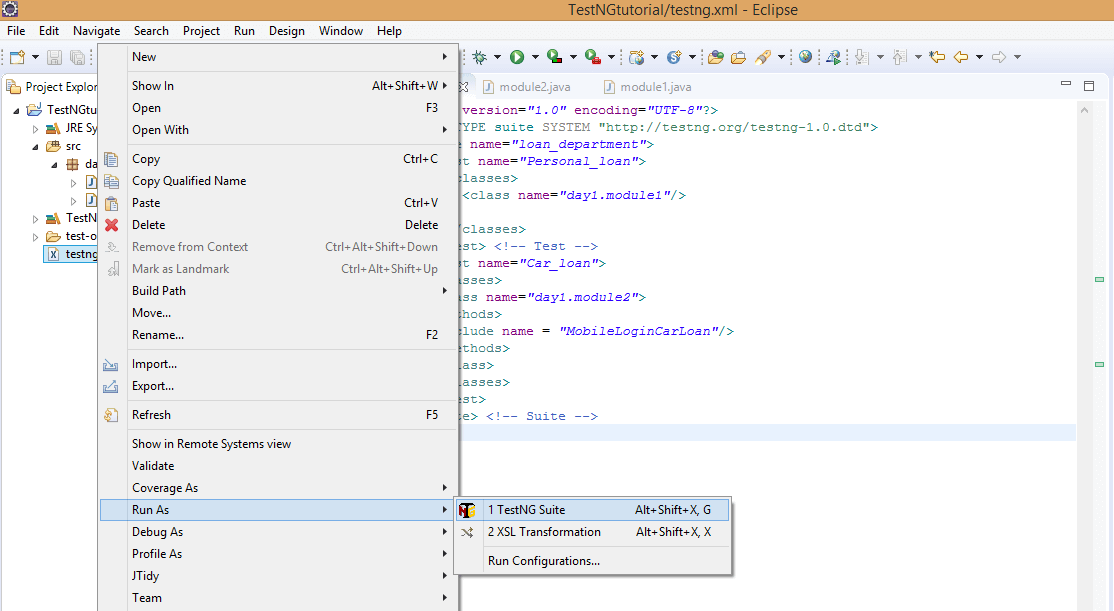
1. **package** day1;
3. **import** org.testng.annotations.Test;
5. **public** **class** module2
6. {
7. @Test
8. **public** **void** WebLoginCarLoan()
9. {
10. System.out.println("WebLoginCarLoan");
11. }
12. @Test
13. **public** **void** MobileLoginCarLoan()
14. {
15. System.out.println("MobileLoginCarLoan");
16. }
17. @Test
18. **public** **void** APILoginCarLoan()
19. {
20. System.out.println("APILoginCarLoan");
21. }
23. }

**module1.java** contains the test cases of Personal loan while **module2.java** contains the test cases of a car loan.

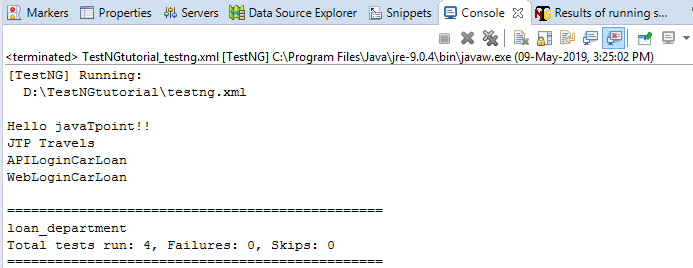
**Step 2:** If we want to disable the MobileLoginCarLoan test case in car loan module, then we need to add the <method> tag in xml file which has access to all the methods of a class.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3. <suite name="loan\_department">
4. <test name="Personal\_loan">
5. <classes>
6. <**class** name="day1.module1"/>
7. </classes>
8. </test> <!-- Test -->
9. <test name="Car\_loan">
10. <classes>
11. <**class** name="day1.module2">
12. <methods>
13. <exclude name = "MobileLoginCarLoan"/>
14. </methods>
15. </**class**>
16. </classes>
17. </test>
18. </suite> <!-- Suite -->

**Step 3:** Run the **testng.xml** file. Right click on the **testng.xml** file, and move the cursor down, you will see the **Run As** and then click on the **1 TestNG Suite**.



**Output**



In the above output, we observe that all the test cases run successfully except the test case "**MobileLoginCarLoan**" which we have excluded.

#### Note: Suppose we have multiple test cases and you want to include only one or two test cases, in such situation, we use <include> tag. If we use the <exclude> tag, then it becomes very tedious to exclude all the test cases.

**Let's understand through an example of <include> tag.**

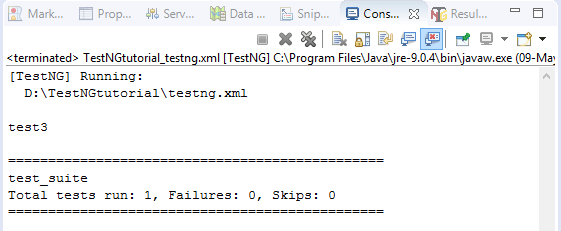
**test.java file**

1. **package** day1;
3. **import** org.testng.annotations.Test;
5. **public** **class** test
6. {
7. @Test
8. **public** **void** test1()
9. {
10. System.out.println("test1");
11. }
13. @Test
14. **public** **void** test2()
15. {
16. System.out.println("test2");
17. }
19. @Test
20. **public** **void** test3()
21. {
22. System.out.println("test3");
24. }
25. @Test
26. **public** **void** test4()
27. {
28. System.out.println("test3");
29. }
30. }

**testng.xml file**

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3. <suite name="test\_suite">
4. <test name="test">
5. <classes>
6. <**class** name="day1.test">
7. <methods>
8. <include name="test3"/>
9. </methods>
10. </**class**>
11. </classes>
12. </test> <!-- Test -->
14. </suite> <!-- Suite -->

**Output**



In the above output, we observe that only one test case is running, i.e., test3 as we have included only test3 in <include> tag in an XML file.

**21.How to skip a @Test method from execution in TestNG?**

TestNG supports multiple ways to skip or ignore a **@Test** execution. Based on requirement, the user can skip a complete test without executing it at all or skip a test based on a specific condition. If the condition meets at the time of execution, it skips the remaining code in the test.

One can use the following ways to skip a **@Test** execution −

* Use the parameter **enabled=false** at **@Test**. By default, this parameter is set as True.
* Use **throw new SkipException(String message)** to skip a test.
* **Conditional Skip** − User can have a condition check. If the condition is met, it will throw a **SkipException** and skip the rest of the code.

In this article, we will illustrate how to skip a test in a class using the above three ways.

### Approach/Algorithm to solve this problem

* **Setp 1** − Create a TestNG class, **NewTestngClass**
* **Setp 2** − Write three different @Test methods in the class **NewTestngClass**, as shown in the programming code section.

**1st @Test Method** It is **enabled=false**. It won’t execute at all and TestNG ignores it completely at runtime. Even in consolidated run details, it doesn’t consider it, so only two tests will be executed.

**2nd @Test Method** It throws **SkipException**. TestNG will print the first line of code and skip the rest as soon as it reaches to **SkipExecution** code.

**3rd @Test Method**

It is conditional skip. The code checks whether the **DataAvailable** parameter is True or False. If it is False, it throws the **SkipException** and skips the test. But, if **DataAvailable** is True, it won’t throw the **SkipException** and continue the execution.

* **Step 3** − Create the **testNG.xml** as given below to run the TestNG classes.
* **Step 4** − Finally, run the **testNG.xml** or directly testNG class in IDE or compile and run it using command line.

## Example

Use the following code for the common TestNG class, **NewTestngClass**

### src/ NewTestngClass.java

import org.testng.SkipException;

import org.testng.annotations.Test;

public class NewTestngClass {

@Test(enabled=false)

public void testcase1(){

System.out.println("Testcase 1 - Not executed");

}

@Test

public void testcase2(){

System.out.println("Testcase 2 - skip exception example");

throw new SkipException("Skipping this exception");

}

@Test

public void testcase3(){

boolean DataAvailable=false;

System.out.println("Test Case3 - Conditional Skip");

if(!DataAvailable)

throw new SkipException("Skipping this exception");

System.out.println("Executed Successfully");

}

}

### testng.xml

This is a configuration file that is used to organize and run the TestNG test cases. It is very handy when limited tests are needed to execute rather than the full suite.

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

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

<suite name = "Suite1">

<test name = "test1">

<classes>

<class name = "NewTestngClass"/>

</classes>

</test>

</suite>

### Output

Testcase 2 - skip exception example

Test ignored.

Test Case3 - Conditional Skip

Test ignored.

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

Suite1

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

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

**22.How to Ignore a test case in TestNG?**

Sometimes, it happens that our code is not ready and the test case written to test that method/code fails. In such cases, annotation **@Test(enabled = false)** helps to disable this test case.

If a test method is annotated with *@Test(enabled = false)*, then the test case that is not ready to test is bypassed.

Now, let's see @Test(enabled = false) in action.

## Create a Class

Create a java class to be tested, say, **MessageUtil.java** in **/work/testng/src**.

/\*

\* This class prints the given message on console.

\*/

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage() {

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

## Create Test Case Class

* Create a java test class, say, **IgnoreTest.java** in **/work/testng/src**.
* Add test methods, testPrintMessage(), and, testSalutationMessage(), to your test class.
* Add an Annotation @Test(enabled = false) to the method testPrintMessage().

Following are the contents of **IgnoreTest.java**.

import org.testng.Assert;

import org.testng.annotations.Test;

public class IgnoreTest {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test(enabled = false)

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

}

## Create testng.xml

Create testng.xml in **/work/testng/src** to execute test case(s).

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

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

<suite name = "Suite1">

<test name = "test1">

<classes>

<class name = "IgnoreTest" />

</classes>

</test>

</suite>

Compile the MessageUtil and test case classes using javac.

/work/testng/src$ javac MessageUtil.java IgnoreTest.java

Now, run the testng.xml, which will not run testPrintMessage() the test case defined in provided the Test Case class.

/work/testng/src$ java org.testng.TestNG testng.xml

Verify the output. testPrintMessage() test case is not tested.

Inside testSalutationMessage()

Hi!Manisha

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

Suite1

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

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

**23.How TestNG allows to state dependencies?**

TestNG allows you to specify dependencies either with −

* Using attribute *dependsOnMethods* in @Test annotations, OR.
* Using attribute *dependsOnGroups* in @Test annotations.

## Example Using *dependsOnMethods*

### Create a Class

Create a java class to be tested, say, **MessageUtil.java** in **/work/testng/src**.

public class MessageUtil {

private String message;

// Constructor

// @param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage() {

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

### Create Test Case Class

* Create a java test class, say, DependencyTestUsingAnnotation.java in **/work/testng/src**.
* Add test methods, testPrintMessage() and testSalutationMessage(), and initEnvironmentTest(), to your test class.
* Add attribute **dependsOnMethods = {"initEnvironmentTest"}** to the @Test annotation of **testSalutationMessage()** method.

Following are the **DependencyTestUsingAnnotation.java** contents.

import org.testng.Assert;

import org.testng.annotations.Test;

public class DependencyTestUsingAnnotation {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test(dependsOnMethods = { "initEnvironmentTest" })

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

@Test

public void initEnvironmentTest() {

System.out.println("This is initEnvironmentTest");

}

}

### Create testng.xml

Create testng.xml in **/work/testng/src** to execute test case(s).

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

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

<suite name = "Suite1">

<test name = "test1">

<classes>

<class name = "DependencyTestUsingAnnotation" />

</classes>

</test>

</suite>

Compile the MessageUtil, Test case classes using javac.

/work/testng/src$ javac MessageUtil.java DependencyTestUsingAnnotation.java

Now, run the testng.xml, which will run the testSalutationMessage() method only after the execution of initEnvironmentTest() method.

/work/testng/src$ java org.testng.TestNG testng.xml

Verify the output.

This is initEnvironmentTest

Inside testPrintMessage()

Manisha

Inside testSalutationMessage()

Hi!Manisha

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

Suite1

Total tests run: 3, Failures: 0, Skips: 0

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

## Example Using *dependsOnGroups*

You can also have methods that depend on entire groups. Let's have an example to demonstrate this.

### Create a Class

Create a java class to be tested, say, **MessageUtil.java** in **/work/testng/src**.

public class MessageUtil {

private String message;

// Constructor

// @param message to be printed

public MessageUtil(String message) {

this.message = message;

}

// prints the message

public String printMessage() {

System.out.println(message);

return message;

}

// add "Hi!" to the message

public String salutationMessage() {

message = "Hi!" + message;

System.out.println(message);

return message;

}

}

### Create Test Case Class

* Create a java test class, say, DependencyTestUsingAnnotation.java.
* Add test methods, testPrintMessage() testSalutationMessage(), and initEnvironmentTest() to your test class, and add them to the group "init".
* Add the attribute **dependsOnMethods = {"init.\*"}** to the @Test annotation of **testSalutationMessage()** method.

Create a java class file named **DependencyTestUsingAnnotation.java** in **/work/testng/src**.

import org.testng.Assert;

import org.testng.annotations.Test;

public class DependencyTestUsingAnnotation {

String message = "Manisha";

MessageUtil messageUtil = new MessageUtil(message);

@Test(groups = { "init" })

public void testPrintMessage() {

System.out.println("Inside testPrintMessage()");

message = "Manisha";

Assert.assertEquals(message, messageUtil.printMessage());

}

@Test(dependsOnGroups = { "init.\*" })

public void testSalutationMessage() {

System.out.println("Inside testSalutationMessage()");

message = "Hi!" + "Manisha";

Assert.assertEquals(message, messageUtil.salutationMessage());

}

@Test(groups = { "init" })

public void initEnvironmentTest() {

System.out.println("This is initEnvironmentTest");

}

}

In this example, testSalutationMessage() is declared as depending on any group, matching the regular expression "init.\*", which guarantees that the methods testPrintMessage() and initEnvironmentTest() will always be invoked before testSalutationMessage().

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. Skipped methods will be reported as such in the final report (in a color that is neither Red nor Green in HTML), which is important since skipped methods are not necessarily failures.

### Create testng.xml

Create testng.xml in **/work/testng/src** to execute test case(s).

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

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

<suite name = "Suite1">

<test name = "test1">

<classes>

<class name = "DependencyTestUsingAnnotation" />

</classes>

</test>

</suite>

Compile the MessageUtil, Test case classes using javac.

/work/testng/src$ javac MessageUtil.java DependencyTestUsingAnnotation.java

Now, run the testng.xml, which will run the *testSalutationMessage()* method only after the execution of *initEnvironmentTest()* method.

/work/testng/src$ java org.testng.TestNG testng.xml

Verify the output.

This is initEnvironmentTest

Inside testPrintMessage()

Manisha

Inside testSalutationMessage()

Hi!Manisha

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

Suite1

Total tests run: 3, Failures: 0, Skips: 0

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

## *dependsOnGroups* Vs *dependsOnMethods*

* On using groups, we are no longer exposed to refactoring problems. As long as we don’t modify the dependsOnGroups or groups attributes, our tests will keep running with the proper dependencies set up.
* Whenever a new method needs to be added in the dependency graph, all we need to do is put it in the right group and make sure it depends on the correct group. We don’t need to modify any other method.

**24.What are the different ways to produce reports for TestNG results**?

* [Method-1: emailable-report.html](https://www.guru99.com/testng-report.html#7)
* [Method-2: index.html](https://www.guru99.com/testng-report.html#8)
* [Method-3: Reporter Class](https://www.guru99.com/testng-report.html#9)

**25.How to write regular expression in testng.xml file to search @Test methods containing “smoke” keyword.**

**27.What does the test timeout mean in TestNG?**

The timeOut is **a helper attribute in TestNG that can put an end to the execution of a test method if that method takes time beyond the timeOut duration**. A timeOut time is set in milliseconds, after that the test method will be marked Failed.

**28.What are @Factory and @DataProvider annotation?**

@DataProvider gives you the power to run a test method with different sets of data, and @Factory gives you the power to run all methods inside a test class with different sets of data.