ToolsQA.com

**Basics of TestNG**

***What is the TestNG Framework?***

TestNG framework is a testing framework to perform tests in the java programming language. Moreover, the "***NG***" in TestNG abbreviates for "***Next Generation***". Cedric Beust developed it and inspired by the JUnit and NUnit testing framework. Subsequently, you can learn ***What is TestNG Framework*** in detail, here.

***How do you run the TestNG script?***  
TestNG script is run by right-click on the ***TestNG class -> Run As -> TestNG Test***. Subsequently, learn [***How To Install TestNG In Eclipse And IntelliJ***](https://www.toolsqa.com/testng/install-testng/).

***What are the advantages of TestNG?***  
One of the common TestNG interview questions is about the advantages of TestNG.  
TestNG has the following advantages:

* *Firstly, TestNG is capable of producing reports automatically with all the necessary information such as failed tests, passed tests, test execution times, etc*.
* *Secondly, TestNG makes use of annotations such as @BeforeMethod, @Test, etc., which are easily understandable as their naming is after their working*.
* *Thirdly, TestNG provides a grouping of methods by which we can group multiple methods as one unit. In other words, Grouping performs operations on all the tests in a group at once rather than individually*.
* *Fourthly, TestNG provides a test method parameterization, which means we can provide parameters in the TestNG and call the function repeatedly with different values. Moreover, parameterization helps in data-driven testing in TestNG*.
* *Fifthly, TestNG provides the prioritization of methods. In other words, by defining the priorities of the methods in TestNG, we can alter the default execution sequence of the test methods according to our wish*.
* *In addition to the above, TestNG allows parallel testing, which increases efficiency and improves the overall running time of test methods*.
* *With the TestNG framework, you can easily integrate with other tools such as Maven, Jenkins, etc*.
* *Moreover, TestNG provides a feature to run multiple*[***test methods on various browsers***](https://www.toolsqa.com/testng/cross-browser-testing-using-testng/)*to test for cross-browser compatibility issues on your website. It is*[***cross-browser testing***](https://www.toolsqa.com/cross-browser-testing/what-is-cross-browser-testing/).
* *Additionally, TestNG allows us to run the tests separately. So, if you run the tests and only one test failed, you can run this test independently in the next execution*.
* *Moreover, TestNG allows the test methods to depend on each other. Its also called Test Dependency in TestNG*.
* *Lastly, TestNG provides a bunch of assertion methods for testing more efficiently*.

Subsequently, you can learn more about the ***benefits of the TestNG framework*** here.

**TestNG Test Case And Suites**

***What is the difference between a TestNG test and a TestNG test suite?***  
TestNG test suite refers to a collection of tests that we can run simultaneously with the help of the TestNG XML file. On the other side, a TestNG test is a single test case file, and when we say "***we are running a TestNG test case***", we simply mean we are running a single test case file. [***Learn more about TestNG Test Suite***](https://www.toolsqa.com/testng/testng-test-suite/).

***Define the correct order of tags in the TestNG XML file***.  
The correct order followed to run the TestNG suite from the XML file is as follows:

<suite>

<test>

<classes>

<class>

<methods>

The closing tags don't appear here as it is just for demonstration purposes. Subsequently, you can learn more about [***How to run TestNG suites through XML?***](https://www.toolsqa.com/testng/testng-test-suite/)

**TestNG Annotations**

***What are the types of annotations used in TestNG (In the sequence of execution/hierarchy)?***

There are nine types of annotations used in TestNG. In order of their execution sequence, they are as follows:

* *@BeforeSuite*
* *@BeforeTest*
* *@BeforeClass*
* *@BeforeMethod*
* *@Test*
* *@AfterMethod*
* *@AfterClass*
* *@AfterTest*
* *@AfterSuite*

***What are the categories of annotations in TestNG?***

TestNG annotations divide into three categories:

* ***Precondition Annotations***: *The annotations under this category execute before the test. It consists of the following annotations*:
* *@BeforeMethod*
* *@BeforeClass*
* *@BeforeSuite*
* *@BeforeTest*
* ***Test Annotations***: *The annotations under this category are defined just before the test methods. Moreover, it consists of the following annotations*:
* *@Test*
* ***Postcondition Annotations***: *The annotations under this category execute after the test methods. Additionally, it consists of the following annotations*:
* *@AfterMethod*
* *@AfterClass*
* *@AfterTest*
* *@AfterSuite*

Subsequently, you can learn more about it here, [***How to use TestNG annotations using Selenium.***](https://www.toolsqa.com/testng/testng-annotations/)

**TestNG Reports**

***What are the types of reports generated in TestNG by default?***

TestNG generates two types of reports by default after the execution of all the test methods finishes. They are:

* *Emailable Reports*
* *Index Reports*

Subsequently, you can learn more about it here, [***What are emailable and index reports in TestNG?***](https://www.toolsqa.com/testng/testng-reports/)

***Where is the emailable report generated and saved in TestNG?***

Emailable reports generate under the project folder and test-output subfolder. This report is available as "***emailable-report.html***" by default.

***Where is the index report generated and saved in TestNG?***

The index report generates under the project folder and test-output subfolder. Moreover, this report is available as "***index.html***" by default.

Subsequently, you can learn more about it here, [***How to view and analyze reports in TestNG?***](https://www.toolsqa.com/testng/testng-reports/)

**TestNG Priorities**

***What are priorities in TestNG?***

Priorities in TestNG is a parameter which declares the priority to a specific test method. TestNG uses the method of alphabetical execution to execute its test method. Through priorities, we can alter the sequence of the test execution. Additionally, the priority can be set as an integer value and lower this integer value; higher is the priority.

Subsequently, you can learn more about it here, [***what are priorities in TestNG?***](https://www.toolsqa.com/testng/testng-test-priority/)

***How would you set priorities in TestNG?***  
TestNG priority is set by the following syntax:

@Test (priority = 1)

public void func(){

//test code

}

An example of prioritization in TestNG can be as follows:

@Test (priority = 1)

public void CloseBrowser() {

driver.close();

System.out.println("Closing Google Chrome browser");

}

@Test (priority = 0)

public void OpenBrowser() {

System.out.println("Launching Google Chrome browser");

driver.get("https://www.demoqa.com");

}

Subsequently, you can learn more about it here, [***How to set priorities and sequencing in TestNG?***](https://www.toolsqa.com/testng/testng-test-priority/)

***Why do we create the XML file in TestNG?***

We use the XML file in TestNG for many purposes. The TestNG XML file helps us:

* *To run multiple tests in a single execution*.
* *Secondly, it also helps us to include and exclude the test methods and groups*.
* *Thirdly, it also helps us to add dependencies in groups*.
* *Fourthly, it helps to run the test case methods through parameters*.
* *Finally, it assists in the execution of the parallel test execution*.

Learn more about [***How to include and exclude groups in TestNG through the XML file?***](https://www.toolsqa.com/testng/groups-in-testng/)

**TestNG Parameters**

***What is parameterization in TestNG?***

In TestNG, parameterization runs a test method multiple times with different values. Another name for this process is data-driven testing in TestNG. We can acquire Parameterization in TestNG in two ways:

* *Firstly, we can achieve it through the XML file*.
* *Secondly, we can achieve it through the dataproviders in TestNG*.

Learn more about [***How to run parameterized tests in TestNG through XML and dataproviders.***](https://www.toolsqa.com/testng/testng-parameters/)

***What are the optional parameters in TestNG?***

Optional parameters work similarly to the default case in the parameterization in TestNG. We use the optional parameter when no other parameter gets defined for that test case method. Additionally, the @Optional annotation declares the optional parameter. We don't define the @Optional parameter above the test method definition but alongside where the method is declared. Subsequently, the following code snippet demonstrates the declaration of the optional parameters in TestNG:

import org.testng.annotations.Optional;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class Params

{

@Test

@Parameters ("message")

public void OP( @Optional("Optional Parameter Selected") String message) {

System.out.println(message);

}

}

Learn more about [***How to implement optional parameters in TestNG?***](https://www.toolsqa.com/testng/testng-parameters/)

***Write the code snipped for passing values 1 and 2 to the parameters val1 and val2 through the XML file.***

To pass the values into the parameters in TestNG, we use <parameter> tag in the TestNG XML file. Additionally, it contains two attributes:

* *name: the name of the parameter variable.*
* *value: the value to insert in that variable.*

Observe the following XML file denoting the same concept.

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

<suite name="My Test-Suite" >

<test name="QA" >

<parameter name="val1" value="1" />

<parameter name="val2" value="2" />

<classes>

<class name="testNGPackage.Parameter" />

<class name="testNGPackage.Multiple\_Parameters" />

</classes>

</test>

</suite>

Learn more about [***How to pass the parameters in TestNG through the XML file?***](https://www.toolsqa.com/testng/testng-parameters/)

**TestNG Groups**

***What is the importance of groups in TestNG?***

Another important TestNG interview questions are about its importance.

Groups are the collection of multiple test case methods combined into one single unit. By grouping, we can operate directly onto the group, which will reflect on all the test case methods under it. Moreover, in TestNG, we can also create a group of groups as a bigger unit of test methods.

Learn more about [***How to create a group of groups in TestNG?***](https://www.toolsqa.com/testng/groups-in-testng/)

***How do you define groups in TestNG?***

The answer to such TestNG interview questions is that we define the Groups in TestNG by passing the "***groups***" parameter to the Test annotation with the value being the group name. In the below example, the test case method will be under the group named "***group1***".

***@Test ( groups = {"group1"})***  
***//test case method***

***How do you exclude a group from the test execution cycle?***

Excluding a group in TestNG denotes that this particular group refrains from running during the execution, and TestNG will ignore it. Additionally, the name of the group that we want to exclude is defined in the XML file  by the following syntax:

<groups>

<run>

<exclude name = "demo">

</exclude>

</run>

</groups>

Learn more about [***How to exclude groups in TestNG?***](https://www.toolsqa.com/testng/groups-in-testng/)

***Can we use regular expression in TestNG groups? Write a demo XML file for the same.***

Yes, regular expressions can be used in TestNG to execute the groups which have some typical pattern in their name. For example, if I want to run all the groups with a name starting from "***abc***", then I can write the regular expression as abc.\* in the XML file.

A demonstration for the above given an example is as follows:

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

<suite name="Test-Suite" >

<test name="ToolsQA" >

<groups>

<run>

<include name = "abc.\*">

</include>

</run>

</groups>

<classes>

<class name="TestNG" />

</classes>

</test>

</suite>

Learn more about [***How to use regular expression in TestNG groups?***](https://www.toolsqa.com/testng/groups-in-testng/)

**TestNG Asserts**

***What do you understand by asserts in TestNG?***

An asset is a piece of code that helps us verify if the expected result and the actual result are equal or not. In TestNG, we leverage the inbuilt "***Assert***" class and a lot of its method to determine whether the test case passed or failed. Additionally, in TestNG, a test case acts as a "***pass***" if none of the assert methods throws an exception during the execution. The syntax for TestNG assert is:

Assert.Method(actual, expected, message);

Learn more about [***TestNG asserts and How to use them in TestNG?***](https://www.toolsqa.com/testng/testng-asserts/)

***Describe any five common TestNG assertions.***

The five common TestNG assertions are:

* *assertEqual(String actual,String expected)*
* *assertEqual(String actual,String expected, String message)*
* *assertEquals(boolean actual,boolean expected)*
* *assertTrue(condition)*
* *assertTrue(condition, message)*
* *assertFalse(condition)*
* *assertFalse(condition, message)*

Although it should be noted that there are a lot more assertions provided by TestNG. Learn more about [***types of assertions in TestNG***](https://www.toolsqa.com/testng/testng-asserts/) and their syntax.

***What are the different types of assert in TestNG?***

There are two types of assert in TestNG:

* *Soft Asserts*
* *Hard Asserts*

Learn more about [***What is soft and hard assert in TestNG?***](https://www.toolsqa.com/testng/testng-asserts/)

***Define soft asserts in TestNG and describe how they are different from hard assert.***

Soft asserts in TestNG means that the execution of the tests would not stop even though the assertion throws an exception in between the execution. In addition to this, TestNG does not include Soft asserts by default in TestNG, so an extra ***org.testng.asserts.Softassert*** package import is required.

Moreover, Soft asserts are different from hard asserts as the hard asserts stop the execution of the test case as soon as the first assertion fails and provides the results. Hard assert includes by default in TestNG.

Learn more about [***What are soft assert and how to use them in TestNG?***](https://www.toolsqa.com/testng/testng-asserts/)

**TestNG Dependent Tests**

***What is meant by dependency in TestNG?***

Dependency in TestNG is a process of making one test dependent on the other test. By providing dependencies in the test methods, we assure that a test method B would only run if test method A runs (*given B depends on A*). Moreover, in TestNG, we can also have one test method dependent on multiple tests.

Learn more about [***single and multiple dependent tests in TestNG.***](https://www.toolsqa.com/testng/testng-dependent-tests/)

***How do you create dependencies in TestNG?***

We can create the dependent tests in TestNG by providing the *dependsonMethods* parameter on the @Test annotation. The value of the attribute is the name of the method on which we want this method to depend. The usage of this method is as follows:

import org.testng.annotations.Test;

public class DependsOnTest {

@Test (dependsOnMethods = { "OpenBrowser" })

public void SignIn() {

System.out.println("User has signed in successfully");

}

@Test

public void OpenBrowser() {

System.out.println("The browser is opened");

}

}

Here, the SignIn method has been made dependent on the OpenBrowser method. Learn more about [***How to create dependencies in TestNG?***](https://www.toolsqa.com/testng/testng-dependent-tests/)

***How do you create dependency through the XML file?***

TestNG also allows us to create dependencies between groups through the TestNG XML file. Such dependencies denote the dependence of one group onto another. The following code demonstrates how to achieve the same goal:

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

<suite name="TestNG XML Dependency Suite" >

<test name="ToolsQA" >

<groups>

<dependencies>

<group depends-on= "openbrowser" name= "login"></group>

</dependencies>

</groups>

<classes>

<class name="GroupDependency" />

</classes>

</test>

</suite>

Here, the login group depends upon the *openbrowser* group. Learn more about [***How to create group dependencies in TestNG?***](https://www.toolsqa.com/testng/testng-dependent-tests/)

***When do we use "dependsOnGroups" in TestNG?***

TestNG gives us the liberty to make a single test depend on a group of tests. When we want to execute in such a manner, we use the *dependsOnGroups* attribute in the TestNG test case file. The value of this attribute is the name of the group on which we want this method to depend. Given below is an example demonstrating the same:

import org.testng.annotations.Test;

public class GroupDependency

{

@Test(dependsOnGroups = { "SignIn" })

public void ViewAcc() {

System.out.println("View Your Dashboardd");

}

@Test(groups = { "SignIn" })

public void OpenBrowser() {

System.out.println("Browser Opened Successfully");

}

@Test(groups = { "SignIn" })

public void LogIn() {

System.out.println("Login Into The Account");

}

}

Learn more about [***How to use dependsOnGroups in TestNG?***](https://www.toolsqa.com/testng/testng-dependent-tests/)

**Miscellaneous**

***What is the significance of "timeout" in TestNG?***

In TestNG, "***timeout***" is a parameter that defines the maximum time that a method can take for execution. The timeout parameter is handy if the execution time is a constraint during testing, or the tester wants to take precautions against the endless execution of tests. We can declare the timeout parameter at:

* ***suite level***: *To put a time constraint on all the methods in the suite*.
* ***method level***: *To put the time constraint on a particular method*.

Syntax:

*@Test (timeout = 1000)*

***What is meant by invocationCount in TestNG?***

The *invocationCount* is an attribute that defines the number of times a test method has to run in a single execution. So, if the *invocationCount* sets as 5, then the test method would run five times each time I execute the TestNG test case.

Syntax for invocationCount:

*@Test (invocationCount = 5)*

***What is meant by parallel test execution in TestNG?***

The parallel test execution means executing different test methods simultaneously, i.e., parallelly in TestNG. It is achieved by creating threads and assigning these threads to different test methods (*which is done automatically and is an operating system's job*). Moreover, running the tests parallelly rather than sequentially is very efficient.

Learn more about [***What is parallel test execution in TestNG, and what are its advantages?***](https://www.toolsqa.com/testng/testng-parallel-execution/)

***On what levels can we apply parallel testing in TestNG?***

Parallel testing can apply at four different levels in TestNG:

* ***Methods***: *This will run the parallel tests on all @Test methods in TestNG*.
* ***Tests***: *All the test cases present inside the <test> tag will run with this value*.
* ***Classes***: *All the test cases present inside the classes that exist in the XML will run in parallel*.
* ***Instances***: *This value will run all the test cases parallelly inside the same instance*.

Learn more about [***How to run tests parallelly in TestNG at different levels with examples?***](https://www.toolsqa.com/testng/testng-parallel-execution/)

***How is exception handling done in TestNG?***

We carry out Exception handling in TestNG by defining the exception at the @Test annotation level. If we proceed in such a manner, the test case will not fail even after raising an exception.

Example:

*@Test (expectedException = numberFormatException.class)*

A tester can write any type of exception here instead of *numberFormatException*.

***Can we disable a test in TestNG? If so, explain how?***

Yes, disabling a test can be achieved in TestNG. Once we disable a test, it will not run in the next execution cycle. Moreover, we accomplish this by using the "***enabled***" attribute.

Syntax:

*@Test (enabled = False)*  
*//code*

Learn more about [***How to skip tests using the enabled parameter in TestNG?***](https://www.toolsqa.com/testng/testng-test-priority/)

***Why is the reporter class used in TestNG?***

The reporter class in TestNG logs the tester defined messages into the reports generated by TestNG. These logged messages then print into the reports, which we can share with the team.

Learn more about [***What is a reporter class in TestNG?***](https://www.toolsqa.com/testng/testng-reporter-log/)

***Define the syntax for generating logs through the reporter class in TestNG.***

Reporter class logs tester-defined messages onto the reports generated by TestNG. Additionally, the syntax for the same is as follows:

*Reporter.log("message");*

Learn more about [***How to generate logs with Reporter class in TestNG?***](https://www.toolsqa.com/testng/testng-reporter-log/)

***What is @Factory annotation in TestNG?***

The need to run multiple test cases in a single test suffices by using the @Factory annotation. The name factory resembles the generation of test class object that is provided by the method under it. Moreover, it is similar to a factory producing a product. The following example shows a factory annotation in TestNG:

@Factory()

public Object[] getTestClasses() {

Object[] tests = new Object[2];

tests[0] = new Test1();

tests[1] = new Test2();

return tests;

}

***Note***: *The test method under @Factory annotation always returns an object array.*

***What is the difference between @Factory and @Dataprovider annotations?***

*@Factory* and *@Dataprovider* are two types of annotations available in TestNG, which look similar in their working but are different.

***@Factory***: *The use of the factory annotation is when the tester needs to execute the test methods multiple times, which are present in the same class. Additionally, we achieve this by creating different instances of the same class*.

***@Dataprovider***: *The dataprovider annotation enables the tester to run a test method multiple times using a different set of data provided by the dataprovider*.

Learn more about [***What are dataproviders and their usage in TestNG?***](https://www.toolsqa.com/testng/testng-dataproviders/)

**TestNG Listeners**

***What are listeners in TestNG?***

Listeners in TestNG are the piece of code that listens to certain events and execute the code associated with that event. As a result, with TestNG listeners, we can change the default behavior of [***TestNG***](https://testng.org/doc/). Moreover, in TestNG, the tester gets the benefit of a lot of listeners who have different functionalities.

Learn more about [***What are TestNG listeners and their types?***](https://www.toolsqa.com/testng/testng-listeners/)

***How are listeners declared in TestNG?***

The listener code in TestNG exists in a separate file than the TestNG test case file. Subsequently, this file contains the listener code and the type of listener to implement is done by "***implementing***" the listener class in the following way:

public class ListenersTestNG implements ITestListener {

public void onStart(ITestContext context) {

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

}

}

To apprise the TestNG test case file about the listener, we declare the @Listener annotation and mentioning the listener class name in the following manner:

@Listeners(ListenersTestNG.class)

public class TestNG {

WebDriver driver = new FirefoxDriver();

@Test //Success Test

public void CloseBrowser() {

driver.close();

}

}

Learn more about [***How to implement listeners in TestNG?***](https://www.toolsqa.com/testng/testng-listeners/)

***What do we need to generate a customized report in TestNG?***

This is amongst many other TestNG interview questions that are asked. A customized report in TestNG generates with the help of TestNG listeners. Using the interface *ITestListener* in TestNG, we can control the events such as method start, method pass, fail, etc., and according to these events, a tester can log appropriate messages.

Learn more about [***What is ITestListener and how to use it in TestNG?***](https://www.toolsqa.com/testng/testng-listeners/)

Conclusively, these were the common TestNG interview questions from my side. Again, I would ask you not to depend entirely on these TestNG interview questions and refer them to gain knowledge about TestNG and its features. So, keep practicing and all the best!!

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Javatpoint.com

|  |
| --- |
| **TestNG Interview Questions** TestNG Interview Questions  A list of top frequently asked **TestNG Interview Questions and answers** are given below. **1) What is TestNG?** TestNG stands for "**Testing Next Generation**". It is an` automation testing framework used for java programming language developed by Credric beust, and it comes after the inspiration from the JUnit framework. TestNG consists of all the features of JUnit framework but also contains some more additional features that make TestNG more powerful. **2) What are the advantages of TestNG?** **The following are the advantages of TestNG are:**   * It generates the report in a proper format, which includes the following information:   + Number of test cases executed.   + Number of test cases passed.   + Number of test cases failed.   + Number of test cases skipped * Multiple test cases can be grouped easily by converting them into a testng.xml file, in which you can set the priority of each test case that determines which test case should be executed first. * With the help of TestNG, you can execute the multiple test cases on multiple browsers known as cross-browser testing. * The TestNG framework can be easily integrated with other tools such as Maven. Jenkins, etc. * Annotations used in a TestNG framework are easily understandable such as @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest. * WebDriver does not generate the reports while TestNG generates the reports in a readable format. * TestNG simplifies the way the test cases are coded. We do not have to write the static main method. The sequence of actions is maintained by the annotations only. * TestNG allows you to execute the test cases separately. For example, if you have six test cases, then one method is written for each test case. When we run the program, five methods are executed successfully, and the sixth method is failed. To remove the error, we need to run only the sixth method, and this can be possible only through TestNG. Because TestNG generates testng-failed.xml file in the test output folder, we will run only this xml file to execute the failed test case.  **3) How to run the test script in TestNG?** You can run the test script in TestNG by clicking right click on the TestNG class, click on "Run As" and then select "TestNG test". **4) What are the annotations used in the TestNG?** **The following are the annotations used in the TestNG are:**   * **Precondition annotations** Precondition annotations are executed before the execution of test methods The Precondition annotations are @BeforeSuite, @BeforeClass, @BeforeTest, @BeforeMethod. * **Test annotation** Test annotation is specified before the definition of the test method. It is specified as @Test. * **Postcondition annotations** The postcondition annotations are executed after the execution of all the test methods. The postcondition annotation can be @AfterSuite, @AfterClass, @AfterTest, @AfterMethod.   AD **5) What is the sequence of execution of all the annotations in TestNG?** **The sequence of execution of all the annotations in TestNG is given below:**   * @BeforeSuite * @BeforeTest * @BeforeClass * @BeforeMethod * @Test * @AfterSuite * @AfterTest * @AfterClass * @AfterMethod  **6) How to set the priorities in TestNG?** If we do not prioritize the test methods, then the test methods are selected alphabetically and executed. If we want the test methods to be executed in the sequence we want, then we need to provide the priority along with the @Test annotation.  **Let's understand through an example.**   1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Test\_methods 4. { 5. @Test(priority=2) 6. **public** **void** test1() 7. { 8. System.out.println("Test1"); 9. } 10. @Test(priority=1) 11. **public** **void** test2() 12. { 13. System.out.print("Test2"); 14. } 15. }  **7) Define grouping in TestNG?** The group is an attribute in TestNG that allows you to execute the multiple test cases. For example, if we have 100 test cases of it\_department and 10 test cases of hr\_department, and if you want to run all the test cases of it\_department together in a single suite, this can be possible only through the grouping.  **Let's understand through an example.**   1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Test\_methods 4. { 5. @Test(groups="it\_department") 6. **public** **void** java() 7. { 8. System.out.println("I am a java developer"); 9. } 10. @Test(groups="it\_department") 11. **public** **void** dot\_net() 12. { 13. System.out.println("I am a .Net developer"); 14. } 15. @Test(groups="it\_department") 16. **public** **void** tester() 17. { 18. System.out.println("I am a software tester"); 19. } 20. @Test (groups="hr") 21. **public** **void** hr() 22. { 23. System.out.print("I am hr"); 24. } 25. }   **testng.xml**   1. ?xml version="1.0" encoding="UTF-8"?> 2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd"> 3. <suite name="Suite"> 4. <test name="It Company"> 5. <groups> 6. <run> 7. <include name="it\_department"/> 8. </run> 9. </groups> 10. <classes> 11. <**class** name="com.javatpoint.Test\_methods"></**class**> 12. </classes> 13. </test> 14. </suite> <!-- Suite -->  **8) What is dependency in TestNG?** When we want to run the test cases in a specific order, then we use the concept of dependency in TestNG.  **Two types of dependency attributes used in TestNG:**   * **dependsOnMethods** The dependsOnMethods attribute tells the TestNG on which methods this test will be dependent on, so that those methods will be executed before this test method.  1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Login 4. { 5. @Test 6. **public** **void** login() 7. { 8. System.out.println("Login page"); 9. } 10. @Test(dependsOnMethods="login") 11. **public** **void** home() 12. { 13. System.out.println("Home page"); 15. } 16. }  * **dependsOnGroups** It is similar to the dependsOnMethods attribute. It allows the test methods to depend on the group of test methods. It executes the group of test methods before the dependent test method.  1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Test\_cases 4. { 5. @Test(groups="test") 6. **public** **void** testcase1() 7. { 8. System.out.println("testcase1"); 9. } 10. @Test(groups="test") 11. **public** **void** testcase2() 12. { 13. System.out.println("testcase2"); 14. } 15. @Test(dependsOnGroups="test") 16. **public** **void** testcase3() 17. { 18. System.out.println("testcase3"); 19. } 20. }  **9) What is timeOut in TestNG?** While running test cases, there can be a case when some test cases take much more time than expected. In such a case, we can mark the test case as a failed test case by using timeOut.  TimeOut in TestNG allows you to configure the time period to wait for a test to get completely executed. It can be configured in two levels:   * **At the suit level:** It will be available to all the test methods. * **At each method level:** It will be available to a particular test method.   The timeOut attribute can be specified as shown below:   1. @Test( timeOut = 700)   The above @Test annotation tells that the test method will be given 700 ms to complete its execution otherwise it will be marked as a failed test case. **10) What is invocationCount in TestNG?** An invocationCount in TestNG is the number of times that we want to execute the same test.   1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Test\_cases 4. { 5. @Test(invocationCount=5) 6. **public** **void** testcase1() 7. { 8. System.out.println("testcase1"); 9. } 11. }   **Output**  TestNG Interview Questions **11) What is the importance of testng.xml file?** **The testng.xml file is important because of the following reasons:**   * It defines the order of the execution of all the test cases. * It allows you to group the test cases and can be executed as per the requirements. * It executes the selected test cases. * In TestNG, listeners can be implemented at the suite level. * It allows you to integrate the TestNG framework with tools such as Jenkins.  **12) How to pass the parameter in test case through testng.xml file?** We can also pass the value to test methods at runtime, we can achieve this by sending parameter values through the testng.xml file. We can use the **@Parameter** annotation:   1. @Parameter("param-name");   **Let's understand through an example:**   1. **package** com.javatpoint; 2. **import** org.openqa.selenium.By; 3. **import** org.openqa.selenium.WebDriver; 4. **import** org.openqa.selenium.chrome.ChromeDriver; 5. **import** org.testng.annotations.Test; 6. **import** org.testng.annotations.Parameters; 7. **public** **class** Web { 8. @Parameters({"text"}) 9. @Test 10. **public** **void** search() 11. { 12. // TODO Auto-generated method stub 13. System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe"); 14. WebDriver driver=**new** ChromeDriver(); 15. driver.get("http://www.google.com/"); 16. driver.findElement(By.name("q")).sendKeys("javatpoint tutorial"); 17. } 18. }   **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="Suite"> 4. <test name="It Company"> 5. <parameter name="text" value="javatpoint"/> 6. <classes> 7. <**class** name="com.javatpoint.Web"></**class**> 8. </classes> 9. </test> 10. </suite> <!-- Suite -->   **On running the testng.xml file, we get the output as shown below:**  TestNG Interview Questions TestNG Interview Questions **13) How can we disable the test case from running?** We can disable the test case from running by using the enabled attribute. We can assign the false value to the enabled attribute, in this way we can disable the test case from running.   1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Test\_cases 4. { 5. @Test(enabled=**false**) 6. **public** **void** testcase1() 7. { 8. System.out.println("testcase1"); 9. } 10. @Test 11. **public** **void** testcase2() 12. { 13. System.out.println("testcase2"); 14. } 16. }  **14) What is the difference between soft assertion and hard assertion?** **Soft Assertion:** In case of Soft Assertion, if TestNG gets an error during @Test, it will throw an exception when an assertion fails and continues with the next statement after the assert statement.  **Hard Assertion:** In the case of Hard Assertion, if TestNG gets an error during @Test, it will throw an AssertException immediately when an assertion fails and stops execution after the assert statement.  **Let's understand through an example.**   1. **package** com.javatpoint; 2. **import** org.testng.Assert; 3. **import** org.testng.annotations.Test; 4. **import** org.testng.asserts.SoftAssert; 5. **public** **class** Assertion { 6. SoftAssert soft\_assert=**new** SoftAssert(); 7. @Test 8. **public** **void** Soft\_Assert() 9. { 10. soft\_assert.assertTrue(**false**); 11. System.out.println("soft assertion"); 12. } 13. @Test 14. **public** **void** Hard\_Assert() 15. { 16. Assert.assertTrue(**false**); 17. System.out.println("hard assertion"); 18. } 19. }   **Output**  TestNG Interview Questions **15) What is the use of @Listener annotation in TestNG?** TestNG provides different kinds of listeners which can perform different actions whenever the event is triggered. The most widely used listener in TestNG is ITestListener interface. The ITestListener interface contains methods such as onTestSuccess, onTestfailure, onTestSkipped, etc.  **Following are the scenarios that can be made:**   * If the test case is failed, then what action should be performed by the listener. * If the test case is passed, then what action should be performed by the listener. * If the test case is skipped, then what action should be performed by the listener.   **Let's understand through an example.**   1. **package** com.javatpoint; 2. **import** org.testng.Assert; 3. **import** org.testng.annotations.Listeners; 4. **import** org.testng.annotations.Test; 5. @Listeners(com.javatpoint.Listener.**class**) 6. **public** **class** Test\_cases 7. { 9. @Test 10. **public** **void** test\_to\_success() 11. { 12. Assert.assertTrue(**true**); 13. } 14. @Test 15. **public** **void** test\_to\_fail() 16. { 17. Assert.assertTrue(**false**); 18. } 20. }   **Listener.java**   1. **package** com.javatpoint; 2. **import** org.testng.ITestContext; 3. **import** org.testng.ITestListener; 4. **import** org.testng.ITestResult; 5. **public** **class** Listener **implements** ITestListener 6. { 7. @Override 8. **public** **void** onTestStart(ITestResult result) { 9. // TODO Auto-generated method stub 10. } 11. @Override 12. **public** **void** onTestSuccess(ITestResult result) { 13. // TODO Auto-generated method stub 14. System.out.println("Success of test cases and its details are : "+result.getName()); 15. } 16. @Override 17. **public** **void** onTestFailure(ITestResult result) { 18. // TODO Auto-generated method stub 19. System.out.println("Failure of test cases and its details are : "+result.getName()); 20. } 21. @Override 22. **public** **void** onTestSkipped(ITestResult result) { 23. // TODO Auto-generated method stub 24. System.out.println("Skip of test cases and its details are : "+result.getName()); 25. } 26. @Override 27. **public** **void** onTestFailedButWithinSuccessPercentage(ITestResult result) { 28. // TODO Auto-generated method stub 29. System.out.println("Failure of test cases and its details are : "+result.getName()); 30. } 31. @Override 32. **public** **void** onStart(ITestContext context) { 33. // TODO Auto-generated method stub 34. } 35. @Override 36. **public** **void** onFinish(ITestContext context) { 37. // TODO Auto-generated method stub 38. }}   **Output**  TestNG Interview Questions **16) What is the use of @Factory annotation?** The @Factory annotation is useful when we want to run multiple test cases through a single test class. It is mainly used for the dynamic execution of test cases.  **Let's understand through an example.**  **testcase1.java**   1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Testcase1 4. { 5. @Test 6. **public** **void** test1() 7. { 8. System.out.println("testcase 1"); 9. } 10. }   **testcase2.java**   1. **package** com.javatpoint; 2. **import** org.testng.annotations.Test; 3. **public** **class** Testcase2 4. { 5. @Test 6. **public** **void** test1() 7. { 8. System.out.println("testcase 2"); 9. } 10. }   **Factory.java**   1. **import** org.testng.annotations.Factory; 2. **public** **class** Factory1 3. { 4. @Factory 5. **public** Object[] getTestClasses() 6. { 7. Object tests[]=**new** Object[2]; 8. tests[0]=**new** Testcase1(); 9. tests[1]=**new** Testcase2(); 10. **return** tests; 11. } 12. }  **17) What is the difference between @Factory and @DataProvider annotation?** **@DataProvider:** It is annotation used by TestNG to execute the test method multiple numbers of times based on the data provided by the DataProvider.  **@Factory:** It is annotation used by the TestNG to execute the test methods present in the same test class using different instances of the respective class.  D |

^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**Software Testing materials**

# 30 Most Popular TestNG Interview Questions And Answers

By[Rajkumar](https://www.softwaretestingmaterial.com/)Updated onDecember 4, 2020

## Most Popular TestNG Interview Questions:

In this post, we will see TestNG Interview Questions with Answers. Our main focus is on Selenium TestNG Interview Questions and also we write some [Selenium Interview Questions](https://www.softwaretestingmaterial.com/selenium-interview-questions/) too. Before going ahead, let’s see some unavoidable Interview Questions such as [What Are The Reasons For Choosing Software Testing As Your Career](https://www.softwaretestingmaterial.com/choose-software-testing-as-a-career/) and [Explain Your Selenium Test Automation Framework](https://www.softwaretestingmaterial.com/explain-test-automation-framework/). I don’t want to take much time of yours but I couldn’t move further without mentioning about this inevitable question in any interview i.e., [Tell Me About Yourself](https://www.softwaretestingmaterial.com/6-important-interview-questions/). Click on the link to get some idea on how to answer [Tell Me About Yourself](https://www.softwaretestingmaterial.com/6-important-interview-questions/). So, Let’s move on to the actual post.

Don’t miss the following posts:

* [100+ Selenium Interview Questions And Answers](https://www.softwaretestingmaterial.com/selenium-interview-questions/)
* [Java Interview Questions for Selenium Testers](https://www.softwaretestingmaterial.com/java-interview-questions/)
* [Framework Interview Questions](https://www.softwaretestingmaterial.com/test-automation-framework-interview-questions/)

Here is a video tutorial “**Selenium TestNG Interview Questions And Answers**”:

Please be patient. The video will load in some time.

If you liked this video, then please subscribe to our [YouTube Channel](https://www.youtube.com/channel/UCIJGI_3XgnfUaSNQD8D2IMQ) for more video tutorials.

**1. What is TestNG?**

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing to integration testing. [For more information](https://www.softwaretestingmaterial.com/testng-introduction/).

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

1. TestNG provides parallel execution of test methods
2. It allows to define dependency of one test method over other method
3. It allows to assign priority to test methods
4. It allows grouping of test methods into test groups
5. It has support for parameterizing test cases using @Parameters annotation
6. It allows data driven testing using @DataProvider annotation
7. It has different assertions that helps in checking the expected and actual results
8. Detailed (HTML) reports

**3. What are the annotations available in TestNG?**

@BeforeTest  
@AfterTest  
@BeforeClass  
@AfterClass  
@BeforeMethod  
@AfterMethod  
@BeforeSuite  
@AfterSuite  
@BeforeGroups  
@AfterGroups  
@Test

[Practical Example](https://www.softwaretestingmaterial.com/testng-annotations/)

**4. Can you arrange the below testng.xml tags from parent to child?**



|  |  |
| --- | --- |
| 1  2  3  4  5 | <test>  <suite>  <class>  <methods>  <classes> |

The correct order of the TestNG tags are as follows



|  |  |
| --- | --- |
| 1  2  3  4  5 | <suite>  <test>  <classes>  <class>  <methods> |

**5. How to create and run testng.xml ?**

In TestNG framework, we need to create **testng.xml** file to create and handle multiple test classes. We do configure our test run, set test dependency, include or exclude any test, method, class or package and set priority etc in the xml file.

[View Complete Post](https://www.softwaretestingmaterial.com/create-testng-xml-file/)

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

In a Selenium TestNG project, we use *testng.xml* file to configure the complete test suite in a single file. Some of the features are as follows.

* testng.xml file allows to include or exclude the execution of test methods and test groups
* It allows to pass parameters to the test cases
* Allows to add group dependencies
* Allows to add priorities to the test cases
* Allows to configure parallel execution of test cases
* Allows to parameterize the test cases

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

We could define the parameters in the *testng.xml* file and then reference those parameters in the source files.

Create a java test class, say, ParameterizedTest.*java*and add a test method say parameterizedTest() to the test class. This method takes a string as input parameter. Add the annotation @Parameters(“browser”) to this method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // TestNG Interview Questions  public class ParameterizedTest {  @Test  @Parameters("browser")  public void parameterizedTest(String browser){  if(browser.equals("firefox")){  System.out.println("Open Firefox Driver");  }else if(browser.equals("chrome")){  System.out.println("Open Chrome Driver");  }  }  } |

The parameter would be passed a value from testng.xml, which we will see in the next step.

We could set the parameter using the below syntax in the testng.xml file.



|  |  |
| --- | --- |
| 1 | <parameter name="browser" value="firefox"/> |

Here, name attribute represents the parameter name and value represents the value of that parameter.

[Practical Example](https://www.softwaretestingmaterial.com/testng-parameterization-using-xml/)

**8. What is TestNG Assert and list out common TestNG Assertions?**

TestNG Asserts help us to verify the condition of the test in the middle of the test run. Based on the TestNG Assertions, we will consider a successful test only if it is completed the test run without throwing any exception.

Some of the common assertions supported by TestNG are

* assertEqual(String actual,String expected)
* assertEqual(String actual,String expected, String message)
* assertEquals(boolean actual,boolean expected)
* assertTrue(condition)
* assertTrue(condition, message)
* assertFalse(condition)
* assertFalse(condition, message)

[For Complete Post](https://www.softwaretestingmaterial.com/testng-asserts/)

**9. What is Soft Assert in TestNG?**

Soft Assert collects errors during *@Test*. Soft Assert does not throw an exception when an assert fails and would continue with the next step after the assert statement.

If there is any exception and you want to throw it then you need to use assertAll() method as a last statement in the @Test and test suite again continue with next @Test as it is.

[Practical Example](https://www.softwaretestingmaterial.com/soft-assert/)

**10. What is Hard Assert in TestNG?**

Hard Assert throws an AssertException immediately when an assert statement fails and test suite continues with next *@Test*

[Practical Example](https://www.softwaretestingmaterial.com/soft-assert/)

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

TestNG gives an option for tracing the Exception handling of code. You can verify whether a code throws the expected exception or not. The expected exception to validate while running the test case is mentioned using the **expectedExceptions** attribute value along with @Test annotation.

[Practical Example](https://www.softwaretestingmaterial.com/testng-exception/)

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

We use priority attribute to the @Test annotations. In case priority is not set then the test scripts execute in alphabetical order.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | // TestNG Interview Questions  package TestNG;  import org.testng.annotations.\*;  public class PriorityTestCase{  @Test(priority=0)  public void testCase1() {  system.out.println("Test Case 1");  }  @Test(priority=1)  public void testCase2() {  system.out.println("Test Case 2");  }  } |

Output:



|  |  |
| --- | --- |
| 1  2 | Test Case 1  Test Case 2 |

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

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

There are two ways to set these parameters:

* using testng.xml – [Practical Example](https://www.softwaretestingmaterial.com/testng-parameterization-using-xml/)
* using Data Providers – [Practical Example](https://www.softwaretestingmaterial.com/testng-parameterization-using-dataproviders/)

**14. How can we create data driven framework using TestNG?**

By using @DataProvider annotation,  we can create a Data Driven Framework.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | // TestNG Interview Questions  @DataProvider(name="getData")  public Object[][] getData(){  //Object [][] data = new Object [rowCount][colCount];  Object [][] data = new Object [2][2];  data [0][0] = "FirstUid";  data [0][1] = "FirstPWD";  data[1][0] = "SecondUid";  data[1][1] = "SecondPWD";  return data;  } |

[Practical Example](https://www.softwaretestingmaterial.com/testng-parameterization-using-dataproviders/)

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

TestNG allows you to perform sophisticated groupings of test methods. Not only can you declare that methods belong to groups, but you can also specify groups that contain other groups. Then TestNG can be invoked and asked to include a certain set of groups (or regular expressions) while excluding another set.  This gives you maximum flexibility in how you partition your tests and doesn’t require you to recompile anything if you want to run two different sets of tests back to back.

Groups are specified in your testng.xml file and can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.



|  |  |
| --- | --- |
| 1  2  3  4 | @Test (groups = { "smokeTest", "functionalTest" })  public void loginTest(){  System.out.println("Logged in successfully");  } |

[Practical Example](https://www.softwaretestingmaterial.com/testng-groups/)

### **TestNG Interview Questions 16 – 33**

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

Groups can also include other groups. These groups are called *MetaGroups*. For example, you might want to define a group *all* that includes *smokeTest*and functionalTest. Let’s modify our testng.xml file as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <groups>     <define name="all">  <include name="smokeTest"/>  <include name="functionalTest"/>     </define>     <run>           <include name="all" />     </run>  </groups> |

[Practical Example](https://www.softwaretestingmaterial.com/testng-groups/)

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

we can use “parallel” attribute in testng.xml to accomplish parallel test execution in TestNG

The parallel attribute of suite tag can accept four values:

tests – All the test cases inside <test> tag of testng.xml file will run parallel  
classes – All the test cases inside a java class will run parallel  
methods – All the methods with @Test annotation will execute parallel  
instances – Test cases in same instance will execute parallel but two methods of two different instances will run in different thread.



|  |  |
| --- | --- |
| 1 | <suite name="softwaretestingmaterial" parallel="methods"> |

[Practical Example](https://www.softwaretestingmaterial.com/parallel-test-execution-testng/)

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

By adding the exclude tag in the testng.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <classes>    <class name="TestCaseName">       <methods>         <exclude name="TestMethodNameToExclude"/>       </methods>    </class>  </classes> |

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

By adding the exclude tag in the testng.xml



|  |  |
| --- | --- |
| 1  2  3  4  5 | <groups>      <run>  <exclude name="TestGroupNameToExclude"/>      </run>  </groups> |

[Practical Example](https://www.softwaretestingmaterial.com/testng-groups/)

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

To disable the test case we use the parameter enabled = false to the @Test annotation.



|  |  |
| --- | --- |
| 1 | @Test(enabled = false) |

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

By using throw new SkipException()

Once SkipException() thrown, remaining part of that test method will not be executed and control will goes directly to next test method execution.



|  |  |
| --- | --- |
| 1 | throw new SkipException("Skipping - This is not ready for testing "); |

[Practical Example](https://www.softwaretestingmaterial.com/how-to-skip-testng-test/)

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

To ignore the test case we use the parameter enabled = false to the @Test annotation.



|  |  |
| --- | --- |
| 1 | @Test(enabled = false) |

[Practical Example](https://www.softwaretestingmaterial.com/how-to-ignore-testng-test/)

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

TestNG allows two ways to declare the dependencies.

Using attributes dependsOnMethods in @Test annotations – [Practical Example](https://www.softwaretestingmaterial.com/testng-dependencies/)Using attributes dependsOnGroups in @Test annotations – [Practical Example](https://www.softwaretestingmaterial.com/testng-dependencies/)

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

TestNG offers two ways to produce a report.

**Listeners** implement the interface *org.testng.ITestListener* and are notified in real time of when a test starts, passes, fails, etc…

**Reporters** implement the interface *org.testng.IReporter* and are notified when all the suites have been run by TestNG. The IReporter instance receives a list of objects that describe the entire test run.

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

TestNG listeners are used to configure reports and logging. One of the most widely used listeners in testNG is ITestListener interface. It has methods like onTestStart, onTestSuccess, onTestFailure, onTestSkipped etc. We should implement this interface creating a listener class of our own. Next we should add the listeners annotation (@Listeners) in the Class which was created.

[Practical Example](https://www.softwaretestingmaterial.com/testng-listeners/)

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

Regular expression to find @Test methods containing keyword “smoke” is as mentioned below.



|  |  |
| --- | --- |
| 1  2  3 | <methods>       <include name=".\*smoke.\*"/>  </methods> |

**27. What is the time unit we specify in test suites and test cases?**

We specify the time unit in test suites and test cases is in milliseconds.

**28. List out various ways in which TestNG can be invoked?**

TestNG can be invoked in the following ways

* Using Eclipse IDE
* Using ant build tool
* From the command line
* Using IntelliJ’s IDEA

**29. How To Run TestNG Using Command Prompt?**

Run the TestNG using command prompt

Open command prompt and use the below code



|  |  |
| --- | --- |
| 1  2  3  4  5 | C:\Users\Admin\Desktop\STMSeleniumTutorial\workspace\SoftwareTestingMaterial    set classpath=C:\Users\Admin\Desktop\STMSeleniumTutorial\workspace\SoftwareTestingMaterial\bin;C:\Users\Admin\Desktop\STMSeleniumTutorial\workspace\SoftwareTestingMaterial\lib\\*    java org.testng.TestNG C:\Users\Admin\Desktop\STMSeleniumTutorial\workspace\SoftwareTestingMaterial\testng.xml |

**30. What is the use of @Test(invocationCount=x)?**

The invocationcount attribute tells how many times TestNG should run a test method



|  |  |
| --- | --- |
| 1  2 | @Test(invocationCount = 10)  public void testCase1(){ |

In this example, the method *testCase1* will be invoked ten times

**31. What is the use of @Test(threadPoolSize=x)?**

The threadPoolSize attribute tells to form a thread pool to run the test method through multiple threads.

**Note:** This attribute is ignored if invocationCount is not specified



|  |  |
| --- | --- |
| 1 | @Test(threadPoolSize = 3, <code class="plain">invocationCount = </code><code class="value">10</code>) public void testCase1(){ |

In this example, the method *testCase1* will be invoked from three different threads

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

The maximum number of milliseconds a test case should take.



|  |  |
| --- | --- |
| 1  2 | @Test(threadPoolSize = 3, invocationCount = 10,  timeOut = 10000)  public void testCase1(){ |

In this example, the function testCase1 will be invoked ten times from three different threads. Additionally, a time-out of ten seconds guarantees that none of the threads will block on this thread forever.

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

@Factory: A factory will execute all the test methods present inside a test class using a separate instance of the respective class with different set of data.

@DataProvider: A test method that uses DataProvider will be executed the specific methods multiple number of times based on the data provided by the DataProvider. The test method will be executed using the same instance of the test class to which the test method belongs.

I would like to conclude this post “TestNG Interview Questions” here.

Final words, Bookmark this post “TestNG Interview Questions” for future reference. After reading this post “TestNG Interview Questions”, if you find that we missed some important questions, please comment below we would try to include those with answers.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

[Software Testing Help](https://www.softwaretestinghelp.com/)

# 20 Most Popular TestNG Interview Questions And Answers

Last Updated:[May 4, 2022](https://www.softwaretestinghelp.com/testng-interview-questions/)

**Top Frequently Asked TestNG Interview Questions and Answers with Examples to Help in Your Preparation:**

A list of the most popular and frequently asked interview questions and answers on TestNG are explained here in this article.

Simple examples are added at the concerned places for your easy understanding of the concept. I’m sure that these questions would help you to crack any TestNG interview successfully.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/08/interview-questionson-software-testng.png)

### Frequently Asked TestNG Interview Questions

**Q #1) What is TestNG?**

**Answer:**TestNG is the framework created for executing unit tests in java program by the developers.

TestNG is also used by the software testers to efficiently run the automated test scripts created in Selenium Webdriver. Its full form is the “Testing New Generation” framework.

It is inspired by “JUnit” which is another framework for unit testing Java programs. In addition to all the features in JUnit, TestNG has its new features which make it more powerful.

**Q #2) How will you install TestNG in Eclipse?**

**Answer:**

**Follow the below steps to install TestNG on Eclipse:**

1. Go to Eclipse -> Click on “Help” -> Click on “Install New Software”.
2. Click on the “Add” button, Enter the name(Preferably TestNG) in the “Next” textbox. Enter in the “Location” textbox and click on the “OK” action button.
3. Check the TestNG checkbox and click on the “Next” action button. The installation will start and the Eclipse will restart after installation.
4. Right-click on the project in Eclipse -> Select build path -> Configure Build Path.
5. Select the library tab -> Click on Add library button -> Select TestNG-> Click on Next -> Click on Finish and Apply and close.

**Q #3) How to run the TestNG script?**

**Answer:**

**To run the TestNG script:**

Right-click on the class in Eclipse, click on “Run as” and select “TestNG test”.

**OR**

Directly click on the Run button on the toolbar of the Eclipse.

**Q #4) What are the annotations used in TestNG?**

**Answer: There are three sections of annotation in TestNG:**

**(i) Precondition annotations:**These are the TestNG annotations that are executed before the test.

@BeforeSuite, @BeforeClass, @BeforeTest, @BeforeMethod are the precondition annotations.

**(ii) Test annotation:**This is the annotation which is only mentioned before the test case (Before the method written to execute the test case)

@Test is the test annotation

**(iii) Postcondition annotation:**These are the annotations that are executed after the test case. (After the method is written to execute the test case)

@AfterSuite, @AfterClass, @AfterTest, @AfterMethod are the postcondition annotations

**Q #5) What is the sequence of execution of the annotations in TestNG?**

**Answer: The Sequence of execution of the annotations is as follows:**

@BeforeSuite

@BeforeTest

@BeforeClass

@BeforeMethod

@Test

@AfterMethod

@AfterClass

@Aftertest

@AfterSuite

**Q #6) What are the advantages of TestNG?**

**Answer: The advantages of TestNG are as follows:**

* It is an open-source framework, hence it is easy to configure.
* Using TestNG we can systematically create the test cases.
* It gives lots of annotations which in turn makes the test case creation easy.
* Using TestNG, priorities of the tests and the sequence of execution can be defined.
* Grouping is possible using TestNG.
* It generates HTML reports (Selenium Webdriver cannot generate the test reports alone, it helps SW to achieve this).
* Data parameterization is possible using TestNG.
* In addition to all the functionalities of JUnit, TestNG has its functionalities, which in turn makes it more powerful.

**Q #7) How to set priorities in TestNG?**

**Answer:** There are always more than one test or method in the class. If we do not prioritize these tests or methods, then the methods are selected alphabetically and executed while execution.

If we want to run the tests in the sequence we want, then we need to set the priority along with the @Test annotation.

**This can be done as follows:**

@Test (priority=1), @Test (priority=2)

**Consider the following Example:**

@Test (priority=2)

**public** **void** getText()

{

driver.findElement(By.id(“id”)).getText();

}

@Test(priority=1)

**public** **void** clickelement()

{

driver.findElement(By.id(“id”)).click();

}

In the above example, clickelement() will get executed first as the priority is set to 1.

And, getText() will get executed after clickelement() as its priority is set to 2.

**Q #8) How to share the project report using TestNG?**

**Answer: There are a few ways to do so:**

**(i)** After the execution of the TestNG class, there is one tab called “Result of running class “ which is generated next to the console.

We can copy this and share it.

**(ii)** After the execution of the TestNG class,

* Right-click on the project name and refresh
* Click on the “Test-output” folder
* Right-click on the “index.html” file and select properties
* Copy the link next to “Location”

We can share this link to see the basic HTML test report which is generated by TestNG.

This is the file that gets generated on your machine automatically after the execution of the class using TestNG.

**Q #9) How will you define grouping in TestNG?**

**Answer:**We can define grouping in TestNG using groups attribute as shown below:

@Test(groups=”title”)

**Q #10) What is a dependency on TestNG?**

**Answer:**There are some methods on which many methods are dependent on.

**For Example,** If we want to test any application, and if the login page of the application is not working then we won’t be able to test the rest of the scenarios.

So, LoginTest is the method on which many tests are dependent on.

**Hence, we will write as follows:**

@Test(dependsOnMethods=”LoginTest”)

Public void homePageLaunched()

{

}

The above code shows that homePageLaunched() method is completely dependent on LoginTest() method.

If LoginTest() is passed, only then the homePageLaunched() method gets executed

**Q #11) What is InvocationCount in TestNG?**

**Answer:**If we want to execute a test case “n” number of times, then we can use the invocationCount attribute as shown in the below example.

**Example:**

@Test(invocationCount=8)

Public void print()

{

}

In the above example, the print() method will get executed 8 times.

**Q #12) What is timeOut in TestNG?**

**Answer:**If any method in the script takes a long time to execute, then we can terminate that method using “timeout” in TestNG.

@Test(timeout = 5000)

In this case, the method will get terminated in 5000 ms (5 seconds) and the test case is marked as “Failed”.

**Q #13) How to handle exceptions in TestNG?**

**Answer:**If there are some methods from which we expect some exceptions, then we can mention the exception in @Test annotation so that the test case does not fail.

**Example:** If a method is expected to have “numberFormatException” exception, then the test case will fail because of this exception if no try-catch block is specified.

But we can do it in TestNG by using “expectedException” attribute as follows.

@Test(expectedException=numberFormatException.class)

Then the test case will run without failing.

**Q #14) What are the common TestNG assertions?**

**Answer: Common TestNG assertions include:**

**(i)** Assert.assetEquals(String actual, String expected);

* It accepts two strings.
* If both the strings are equal, the test case executes successfully otherwise the test case fails.

**(ii)** Assert.assertEquals(String actual, String expected, String message)

* It accepts two strings.
* If both the strings are equal, the test case executes successfully otherwise the test case fails.
* The message is printed if the test case fails.

**(iii)** Assert.assertEquals(boolean actual, boolean expected)

* It accepts two boolean values.
* If both the boolean values are equal, the test case executes successfully otherwise the test case fails.

**(iv)** Assert.assertTrue(<condition(t/f)>)

* It accepts a boolean value.
* The assertion passes if the condition is True, else an assertion error is displayed.

**(v)** Assert.assertFalse(<condition(t/f)>)

* It accepts a boolean value.
* The assertion passes if the condition is False, else an assertion error is displayed.

**(vi)** Assert.assertTrue(<condition(t/f)>,message)

* It accepts a boolean value.
* The assertion passes if the condition is True, else an assertion error is displayed with the mentioned message.

**(vii)** Assert.assertFalse(<condition(t/f)>,message)

* It accepts a boolean value.
* The assertion passes if the condition is False, else an assertion error is displayed with the mentioned message.

**Q #15) How to disable a test in TestNG?**

**Answer:**

**To disable a test in TestNG, we have to use the “enabled” attribute as follows:**

@Test(enabled=”false”)

**Q #16) What are the types of Asserts in TestNG?**

**Answer:**To validate the results (pass/fail), we have to use the assertion.

**There are two types of assert in TestNG:**

**(i) Hard Assert:**

Hard Assert is the normal assert which is used to do validations in the TestNG class.

**We have to use Assert class for hard assert as follows:**

Assert.assertEquals(actual value, expected value);

If the hard assert fails, then none of the code gets executed after the assert statement.

**(ii) Soft Assert:**

If we want to continue the test execution even after the assert statement fails, then we have to use soft assert.

To create a soft assert, **we have to create an object of a “softAssert” class as follows:**

softAssert sassert = new softAssert();  
sassert.assertAll();

So now if the test case fails, the execution is not terminated when we use soft assert.

**Q #17) How to pass parameter in the test case through the testng.xml file?**

**Answer:**If we have a class in which a login method is defined, then we can pass the login parameters to this login method from the testing.xml file

**We will have to use the “@parameters” annotation as follows:**

@Parameters({"user\_name","password"})

@Test

**public** **void** loginapp()

{

driverget(“appname”);

driver.findElement(By.id(“login”)).sendkeys(user\_name);

driver.findElement(By.id(“password”)).sendkeys(password);

}

**Now, go to the testng.xml file and enter the parameters there as follows:**

<Suite name = “suitename”>

<test name =”testname”>

<parameter name =”user\_name” value=”user1”/>

<parameter password =”password” value =”pass1”/>

<Classes>

<class name =”passingparameters”/>

<classes/>

<test/>

<Suite/>

**Q #18) What is the need to create a testng.xml file?**

**Answer:**When we test a project using Selenium Webdriver, it has a lot of classes on it. We cannot choose these classes one by one and put them for automation. Hence we need to create a suite so that all the classes run in a single test suite.

We can achieve this by creating a testing.xml file.

**Q #19) How to create an XML file in TestNG?**

**Answer:**Go to the src folder -> click on file ->enter the name of the file(mostly written testing.xml)

Then, Click on finish.

We have a blank XML file. Here, we have to mention the project name and the classes to be executed along with the package name as shown below.

<Suite name = "Testing project">

<test name = "testing feature 1">

<classes>

<class name = "packagename.name of class1"/>

<class name = "packagename.name of class1"/>

<class name = "packagename.name of class1"/>

<class name = "packagename.name of class1"/>

</classes>

</test>

</Suite>

To run this file, we have to go to testng.xml in the package explorer right click and run as -> TestNG suite

**Q #20) How to throw a SKIP Exception in TestNG?**

**Answer:**If we want to SKIP any Test using testing, then we have to use the SKIP exception in TestNG.

**It is written as follows:**

**public** **void** skipExc()

{

System.out.println("SKIP me");

**throw** **new** skipException(“Skipping skipExc”);

}

}

***We wish you all the best for your interview!!***

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

TestNG is an open-source automated testing framework; where NG of TestNG means Next Generation. TestNG is similar to JUnit but it is much more powerful than JUnit but still, it’s inspired by JUnit.

This eliminates most of the limitations of the older framework. It provides the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parametrizing.

Below is the list of topics which we are going to study in the next chapters of TestNg Tutorial:

* TestNG Introduction
* Install TestNG
* TestNG Test
* TestNG Test Suite
* TestNG Annotations
* TestNG Groups
* TestNG Dependent Tests
* TestNG Reports
* TestNG Parameters
* TestNG DataProviders
* TestNG Test Priority
* TestNG Reporter Log
* TestNG Asserts
* TestNG Cross Browser Testing
* TestNG Data Provider with Excel
* TestNG Parallel Execution
* TestNG Listeners
* Retry Failed Tests in TestNG
* Implement IRetryAnalyzer
* TestNG Vs JUnit

**What are the Benefits of TestNG?**

There are a number of benefits but from Selenium perspective, major advantages of TestNG are :

* It gives the ability to produce HTML Reports of execution
* Annotations made testers life easy
* Test cases can be Grouped & Prioritized more easily
* Parallel testing is possible
* Generates Logs
* Data Parameterization is possible

**Test Case Writing process in TestNG**

Writing a test in TestNG is quite simple and basically involves the following steps:

Step 1 - Write the business logic of the test

Step 2 - Insert TestNG annotations in the code

Step 3 - Add the information about your test (e.g. the class names, methods names, groups names, etc...) in a testng.xml file

Step 4 - Run TestNG

**What are the different Annotations are present in TestNG?**

* @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 tag is run.
* @AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.
* @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.
* @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.
* @Test: The annotated method is a part of a test case.

**Benefits of using Annotations**

* It identifies the methods it is interested in by looking up annotations. Hence method names are not restricted to any pattern or format.
* We can pass additional parameters to annotations.
* Annotations are strongly typed, so the compiler will flag any mistakes right away.
* Test classes no longer need to extend anything (such as Test Case, for JUnit 3).

**How To Install TestNG In Eclipse?**

The following installation process uses Eclipse Version 4.14.0 as on Mar'20 to install TestNG.

1. Launch the Eclipse IDE and click “Install New Software” in the Help menu.
2. You will see a new installation dialog window, click the "Add" button.
3. Fill out the information as follows:

Name: TestNG (depends on the user)

Click Add.

1. Clicking on add redirects us back to the previous window. However, this time you must see the TestNG option in the available software list. After that, check "TestNG" and click Next.
2. Click Next to install the TestNG dependencies that eclipse calculates by itself.
3. After that, accept the terms of the license agreement then click Finish.
4. After that, accept the terms of the license agreement then click Finish.
5. You may or may not encounter a Security warning. Click Install Anyway if you do.
6. After that, click "Restart Now" to restart the eclipse and finish the installation setup.
7. Finally, after the restart, verify if TestNG installed successfully. Right-click on your project and see if TestNG displays in the opened menu.

**To create TestNG project**

* Firstly, navigate To File -> New -> Java Project.
* Give it a name of your choice.
* Secondly, click Next to move to the next panel.
* Thirdly, click on Libraries to add TestNG Libraries to your project (Only if Eclipse does not automatically add the TestNG Library).
* After that, select "Add Library" to add the TestNG Library.
* Choose TestNG and click on Next.
* Finally, click Finish to finish adding the TestNg Library in the project.

How To Create A TestNG Class In Eclipse

* Firstly, press Ctrl+N, then select “TestNG Class” under the TestNG category and click Next.
* Or
* Right-click on src, go to TestNG, and select "Create TestNG Class".
* After that, the source folder name will automatically populate in the text field. But if it doesn't, like my system, you can browse your way through the src folder by clicking on the Browse button.
* Thirdly, set class name as 'TestNG '
* It will display the TestNG.java test file, which is partially created for you. The test case file will contain a default method, f(), along with beforeMethod() and afterMethod() that we checked in the previous step.
* Finally, we are all set now by creating our first test class in TestNG. We can now proceed to write the first TestNG test case.

Add the selenium jar files to the project

**TestNG Test Suites:** The collection of TestNG Tests together is called a Test Suite The process of running multiple tests at once is called a test suite, and performing it in TestNG is called TestNG Test Suites.

Running a test suite in TestNG requires us to create a TestNG XML file and executing it to achieve the goals. Through this TestNG XML file only, we will be able to create and handle multiple test classes in the TestNG framework. In addition to this, the XML file will be the target file where you will configure your test run, set test dependency, include or exclude any test, method, class or package and set priority, etc.

**Create textng.xml file**

Right click project 🡪 new🡪file🡪textng.xml🡪ok

**Run the test Suite**

Run the test by right click on the testng.xml file and select Run As > TestNG Suite.un the test by right click on the testng.xml file and select Run As > TestNG Suite.

**What Are TestNG Annotations?**

Annotations, in general, mean "a comment" or "a note" on a diagram, etc. to denote its meaning. TestNG also uses them for the same reason. TestNG annotations are the code that is written inside your source test code logic to control the flow of the execution of tests. It is essential to annotate your methods in TestNG to run the tests. TestNG will ignore the method which does not contain an annotation since it won't know when to execute this method.

A TestNG annotation starts from the symbol "@" and whatever follows is the annotation name.

TestNG contains a hierarchy among the annotations. This hierarchy is as follows (top being the highest priority):

@BeforeSuite

@BeforeTest

@BeforeClass

@BeforeMethod

@Test

@AfterMethod

@AfterClass

@AfterTest

@AfterSuite

**Multiple Test Case Scenario**

Numerous test cases can run by setting the priority of the test in the test methods.

If there are multiple @Test cases and priority is not specified, TestNG runs the test cases in the alphabetical order.

**Test Priority in TestNG**

Although TestNG annotations decide in which order the tests will run, priorities do more or less the same job.

The priorities are an additional option that we can put to use with the test annotations. This attribute decides the priority of the annotation. But remember that priority check happens after the annotation check by TestNG. So the TestNG annotation hierarchy is followed first and then priority-based execution. The larger the priority number, the lower is its priority.

If there same priorities for different methods under TestNG annotations. In that case, TestNG runs the test cases in the alphabetical order. same priorities for different methods under TestNG annotations. In that case, TestNG runs the test cases in the alphabetical order.

The test cases without the priority attribute are given the "priority" and executed before the methods with priority. The test cases without the priority attribute are given the "priority" and executed before the methods with priority.

TestNG.java

|  |
| --- |
| **import** org.testng.annotations.AfterClass;  **import** org.testng.annotations.AfterMethod;  **import** org.testng.annotations.AfterSuite;  **import** org.testng.annotations.AfterTest;  **import** org.testng.annotations.BeforeClass;  **import** org.testng.annotations.BeforeMethod;  **import** org.testng.annotations.BeforeSuite;  **import** org.testng.annotations.BeforeTest;  **import** org.testng.annotations.Test;    **public** **class** TestNG {  @Test(priority =2)  **public** **void** z\_testCase1() {  System.***out***.println("This is the z\_testCase1");  }    @Test(priority =2)  **public** **void** a\_testCase2() {  System.***out***.println("This is the a\_testCase2");  }  @Test  **public** **void** d\_method() {  System.***out***.println("This is D Method");  }    @Test  **public** **void** c\_method() {  System.***out***.println("This is C Method");  }  @BeforeMethod  **public** **void** beforeMethod() {  System.***out***.println("This will execute before every Method");  }    @AfterMethod  **public** **void** afterMethod() {  System.***out***.println("This will execute after every Method");  }    @BeforeClass  **public** **void** beforeClass() {  System.***out***.println("This will execute before the Class");  }    @AfterClass  **public** **void** afterClass() {  System.***out***.println("This will execute after the Class");  }    @BeforeTest  **public** **void** beforeTest() {  System.***out***.println("This will execute before the Test");  }    @AfterTest  **public** **void** afterTest() {  System.***out***.println("This will execute after the Test");  }    @BeforeSuite  **public** **void** beforeSuite() {  System.***out***.println("This will execute before the Test Suite");  }    @AfterSuite  **public** **void** afterSuite() {  System.***out***.println("This will execute after the Test Suite");  }  } |

**Output**

[RemoteTestNG] detected TestNG version 7.4.0

This will execute before the Test Suite

This will execute before the Test

This will execute before the Class

This will execute before every Method

This is C Method

This will execute after every Method

This will execute before every Method

This is D Method

This will execute after every Method

This will execute before every Method

This is the a\_testCase2

This will execute after every Method

This will execute before every Method

This is the z\_testCase1

This will execute after every Method

This will execute after the Class

This will execute after the Test

This will execute after the Test Suite

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

Test-Suite

Total tests run: 4, Passes: 4, Failures: 0, Skips: 0

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

**What Are TestNG Groups?**

Groups in TestNG denotes the process of grouping different tests together into a straightforward group and running these tests together by just running the group in a single command. It does not even matter if they belong to different classes.

It is important to note that Groups are declared in the testng.xml file in the TestNG and can be found inside <test> tag or <suite> tag.

GroupDemo.java

|  |
| --- |
| @Test(priority =1 ,groups = { "demo" })  **public** **void** starting\_point(){  System.***out***.println("This is the starting point of the test");  System.*setProperty*("webdriver.chrome.driver", "C:/Drivers/chromedriver\_win32/chromedriver.exe");  driver = **new** ChromeDriver();  driver.get("https://demoqa.com/");  System.***out***.println("End of starting point");  }      @Test(priority =2 ,groups = { "demo" })  **public** **void** checkTitle() {  String testTitle = "Free QA Automation Tools For Everyone";  String originalTitle = driver.getTitle();  System.***out***.println("testTitle = " + testTitle);  System.***out***.println("originalTitle = " + originalTitle);  Assert.*assertEquals*(originalTitle, testTitle);  }    @Test(priority =3 )  **public** **void** click\_element() {  System.***out***.println("Home Page heading is displayed");  } |

testng.xml

|  |
| --- |
| <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Test-Suite" >  <test name="ToolsQA" >  <groups>  <run>  <include name = "demo"></include>  </run>  </groups>  <classes>  <class name="TestNG" />  <class name ="GroupsDemo" />  </classes>  </test>  </suite> |

**How To Ignore (Exclude) a group in TestNG?**

|  |
| --- |
| <test name="ToolsQA" >  <groups>  <run>  <exclude name = "demo"> </exclude>  </run>  </groups>  <classes>  <class name="TestNG" />  </classes>  </test> |

**How To Use Regular Expressions With TestNG Groups?**

|  |
| --- |
| @Test  public void starting\_point(){  System.out.println("This is the starting point of the test");  }    @Test(groups = { "demo1" })  public void checkTitle() {  String testTitle = "Free QA Automation Tools For Everyone";  }    @Test(groups = { "demo2" })  public void click\_element() {  System.out.println("Home Page heading is displayed");  } |

|  |
| --- |
| <test name="ToolsQA" >  <groups>  <run>  <include name = "demo.\*">  </include>  </run>  </groups> |

**How To Use dependsOn attribute in TestNG?**

TestNG allows you to specify dependencies in the following two ways:

Using attributes dependsOnMethods in @Test annotations

The dependsOnMethods lets us make a test depend on a particular method. For example, look at the following code:

|  |
| --- |
| @Test (dependsOnMethods = { "OpenBrowser" })  **public** **void** SignIn() {  System.***out***.println("User has signed in successfully");  }  @Test  **public** **void** OpenBrowser() {  System.***out***.println("The browser is opened");  }  @Test (dependsOnMethods = { "SignIn" })  **public** **void** LogOut() {  System.***out***.println("The user logged out successfully");  } |

output

The browser is opened

User has signed in successfully

The user logged out successfully

|  |
| --- |
| @Test(dependsOnGroups = { "SignIn" })  public void ViewAcc() {  System.out.println("SignIn Successful");  }    @Test(groups = { "SignIn" })  public void LogIn() {  System.out.println("Logging In Success");  } |

TestNG Dependent Test In XML Suite

TestNG lets you create dependencies between groups in the XML file. So, if you have multiple groups in the TestNG file, you can create the dependent tests in between them in the XML file.

|  |
| --- |
| <test name="ToolsQA" >  <groups>  <dependencies>  <group depends-on= "openbrowser" name= "login"></group>  <group depends-on= "login" name= "viewaccount"></group>  <group depends-on= "viewaccount" name= "logout"></group>  </dependencies>  </groups>  <classes>  <class name="GroupDependency" />  </classes>  </test> |

**What are the two ways to generate a report in TestNG?**

By default TestNG can generate the reports in two ways:

Emailable Reports

Index Reports

**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.

|  |
| --- |
| **In java file**  @Test(enabled=**false**)  **in xml file**  <methods>    <exclude name = "MobileLoginCarLoan"/>   </methods> |

|  |
| --- |
| **Reg ex in xml file**  <methods>  <exclude name="employee.\*"/>   : Do not execute the tests starting with employee  <include name="Mobile.\*"/>   : Execute all tests starting with Mobile  </methods> |

**TestNG Annotation Attributes**

We can also explicitly specify the attributes in a @Test annotation. Test attributes are the test specific, and they are specified at the right next to the @Test annotation.

Some of the common attributes are described below:

* description
* timeOut
* priority
* dependsOnMethods
* enabled
* groups

|  |
| --- |
| @Test(attribute="value")  **public** **void** method()  @Test(description="This is testcase1")  **public** **void** jira()  @Test(dependsOnMethods= {"WebStudentLogin"})  **public** **void** jira()  @Test(priority=2)  **public** **void** apple()  @Test(enabled=**false**)  **public** **void** jira()  @Test(groups= {"software company"})  **public** **void** infosys()  @Test(timeOut=200)  **public** **void** testcase1() **throws** InterruptedException |

**TestNG Parameters**

TestNG Parameters are the arguments that we pass to the test methods. There are two ways through which we can pass the parameters to the test methods:

* TestNG Parameters
* TestNG DataProviders

Suppose we want to set the global variables such url settings, username, password or API Keys, there are some values which are constant in all the test cases, in such case we use the TestNG Parameters.

TestNG Parameters are present in the xml file.

|  |
| --- |
| public class Sum  {  @Test  @Parameters({"a","b"})  public void add(int c, int d)  {  int sum=c+d;  System.out.println("Sum of two numbers : "+sum);  }  }  testng.xml  <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name="Suite">  <parameter name="a" value="5"/>  <parameter name="b" value="3"/>  <test name="Sum">    <classes>    <class name= "com.javatpoint.Sum"/>    </classes> |

**TestNG Listeners**

TestNG provides the @Listeners annotation which listens to every event that occurs in a selenium code. Listeners are activated either before the test or after the test case. It is an interface that modifies the TestNG behavior. For example, when you are running a test case either through selenium or appium and suddenly a test case fails. We need a screenshot of the test case that has been failed, to achieve such scenario, TestNG provides a mechanism, i.e., Listeners. When the test case failure occurs, then it is redirected to the new block written for the screenshot.

Listeners are implemented by the ITestListener interface. An ITestListener interface has the following methods:

* onTestStart(): An onTestStart() is invoked only when any test method gets started.
* onTestSuccess(): An onTestSuccess() method is executed on the success of a test method.
* onTestFailure(): An onTestFailure() method is invoked when test method fails.
* onTestSkipped(): An onTestSkipped() run only when any test method has been skipped.
* onTestFailedButWithinSuccessPercentage(): This method is invoked each time when the test method fails but within success percentage.
* onStart(): An onStart() method is executed on the start of any test method.
* onFinish(): An onFinish() is invoked when any test case finishes its execution.

|  |
| --- |
| import org.testng.Assert;  import org.testng.annotations.Listeners;  import org.testng.annotations.Test;  @Listeners(com.javatpoint.Listener.class)  public class Class1  {  @Test  public void sum()  {  int sum=0;  int a=5;  int b=7;  sum=a+b;  System.out.println(sum);  }  @Test  public void testtofail()  {  System.out.println("Test case has been failed");  Assert.assertTrue(false);  }  } |

**Listener.java**

|  |
| --- |
| package com.javatpoint;  import org.testng.ITestContext;  import org.testng.ITestListener;  import org.testng.ITestResult;  public class Listener implements ITestListener  {  @Override  public void onTestStart(ITestResult result) {  // TODO Auto-generated method stub  }  @Override  public void onTestSuccess(ITestResult result) {  // TODO Auto-generated method stub  System.out.println("Success of test cases and its details are : "+result.getName());  }  @Override  public void onTestFailure(ITestResult result) {  // TODO Auto-generated method stub  System.out.println("Failure of test cases and its details are : "+result.getName());  }  @Override  public void onTestSkipped(ITestResult result) {  // TODO Auto-generated method stub  System.out.println("Skip of test cases and its details are : "+result.getName());  }  @Override  public void onTestFailedButWithinSuccessPercentage(ITestResult result) {  // TODO Auto-generated method stub  System.out.println("Failure of test cases and its details are : "+result.getName());  }  @Override  public void onStart(ITestContext context) {  // TODO Auto-generated method stub  }  @Override  public void onFinish(ITestContext context) {  // TODO Auto-generated method stub  }  } |

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name="Suite">  <test name="Listeners">  <classes>  <class name="com.javatpoint.Class1"></class>  </classes>  </test>  </suite> <!-- Suite --> |

TestNG allows you to execute the test cases separately. For example, if you have six test cases, then one method is written for each test case. When we run the program, five methods are executed successfully, and the sixth method is failed. To remove the error, we need to run only the sixth method, and this can be possible only through TestNG. Because TestNG generates testng-failed.xml file in the test output folder, we will run only this xml file to execute the failed test case.

**What is dependency in TestNG?**

When we want to run the test cases in a specific order, then we use the concept of dependency in TestNG.

Two types of dependency attributes used in TestNG:

dependsOnMethods

The dependsOnMethods attribute tells the TestNG on which methods this test will be dependent on, so that those methods will be executed before this test method.

dependsOnGroups

It is similar to the dependsOnMethods attribute. It allows the test methods to depend on the group of test methods. It executes the group of test methods before the dependent test method.

9) **What is timeOut in TestNG?**

While running test cases, there can be a case when some test cases take much more time than expected. In such a case, we can mark the test case as a failed test case by using timeOut.

TimeOut in TestNG allows you to configure the time period to wait for a test to get completely executed. It can be configured in two levels:

At the suit level: It will be available to all the test methods.

At each method level: It will be available to a particular test method.

The timeOut attribute can be specified as shown below:

@Test( timeOut = 700)

**What is invocationCount in TestNG?**

An invocationCount in TestNG is the number of times that we want to execute the same test.

|  |
| --- |
| @Test(invocationCount=5) |

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

The testng.xml file is important because of the following reasons:

It defines the order of the execution of all the test cases.

It allows you to group the test cases and can be executed as per the requirements.

It executes the selected test cases.

In TestNG, listeners can be implemented at the suite level.

It allows you to integrate the TestNG framework with tools such as Jenkins.

**How to pass the parameter in test case through testng.xml file?**

We can also pass the value to test methods at runtime, we can achieve this by sending parameter values through the testng.xml file. We can use the @Parameter annotation:

@Parameter("param-name");

**What is the difference between soft assertion and hard assertion?**

Soft Assertion: In case of Soft Assertion, if TestNG gets an error during @Test, it will throw an exception when an assertion fails and continues with the next statement after the assert statement.

Hard Assertion: In the case of Hard Assertion, if TestNG gets an error during @Test, it will throw an AssertException immediately when an assertion fails and stops execution after the assert statement.

|  |
| --- |
| package com.javatpoint;  import org.testng.Assert;  import org.testng.annotations.Test;  import org.testng.asserts.SoftAssert;  public class Assertion {  SoftAssert soft\_assert=new SoftAssert();  @Test  public void Soft\_Assert()  {  soft\_assert.assertTrue(false);  System.out.println("soft assertion");  }  @Test  public void Hard\_Assert()  {  Assert.assertTrue(false);  System.out.println("hard assertion");  }  } |

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

TestNG provides different kinds of listeners which can perform different actions whenever the event is triggered. The most widely used listener in TestNG is ITestListener interface. The ITestListener interface contains methods such as onTestSuccess, onTestfailure, onTestSkipped, etc.

Following are the scenarios that can be made:

* If the test case is failed, then what action should be performed by the listener.
* If the test case is passed, then what action should be performed by the listener.
* If the test case is skipped, then what action should be performed by the listener.

**What is the use of @Factory annotation?**

The @Factory annotation is useful when we want to run multiple test cases through a single test class. It is mainly used for the dynamic execution of test cases.

Let's understand through an example.

testcase1.java

|  |
| --- |
| package com.javatpoint;  import org.testng.annotations.Test;  public class Testcase1  {  @Test  public void test1()  {  System.out.println("testcase 1");  }  } |

Testcase2.java

|  |
| --- |
| package com.javatpoint;  import org.testng.annotations.Test;  public class Testcase2  {  @Test  public void test1()  {  System.out.println("testcase 2");  }  } |

Factory1 .java

|  |
| --- |
| import org.testng.annotations.Factory;  public class Factory1  {  @Factory  public Object[] getTestClasses()  {  Object tests[]=new Object[2];  tests[0]=new Testcase1();  tests[1]=new Testcase2();  return tests;  }  } |

**What is the difference between @Factory and @DataProvider annotation?**

@DataProvider: It is annotation used by TestNG to execute the test method multiple numbers of times based on the data provided by the DataProvider.

@Factory: It is annotation used by the TestNG to execute the test methods present in the same test class using different instances of the respective class.

