**. What is the TestNG Framework?**

Inspired by JUnit, TestNG was created by Cedric Beust in 2004. TestNG is a free and open-source Java test automation framework. It is based on the same principles as JUnit and NUnit. TestNG's advanced and useful features make it a more robust framework than its competitors. Such as annotations being easier to use, HTML reports for implementations, and enabling you to group test cases.

TestNG's NG stands for **'Next Generation**.' It was developed by Cedric Beust and is increasingly utilized by developers and testers in test case generation due to the ease of use of numerous annotations, grouping, dependence, priority, and parametrization capabilities.

We have a whole dedicated learning hub where you will get A to Z information about the [TestNG testing framework](https://www.lambdatest.com/learning-hub/testng) with best practices.



**2. How do you run the TestNG script?**

**Method 1:**

In the first method, you run TestNG scripts from the command line by specifying the TestNG JAR file and the path to your test class file in the java command. Here is the code snippet for a better understanding

java -cp testng.jar:path/to/your/test/**class** com.testng.TestClass

Replace testng.jar with the name of the TestNG JAR file, path/to/your/test/class with the path to the file containing your test class, and com.testng.TestClass with the fully qualified name of your test class.

**Method 2:**

Another method to run the TestNG script is as shown below:

* Launch your Java IDE (e.g. Eclipse, IntelliJ IDEA, NetBeans).
* Open an existing Java project or start a new one.
* Include the TestNG library in the build path of your project. To do so, download the TestNG JAR file and add it to the classpath of your project or by using a build tool such as Maven or Gradle.
* Create a Java class file for your TestNG test cases. This class's methods that include test code should have the @Test annotation.
* On the Project Explorer view, right-click on the test class file and choose "Run As" > "TestNG Test" (or use the keyboard shortcut, usually Ctrl+Shift+F10).
* TestNG will run your class's test methods and provide a report with the results of each test case.

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

TestNG is built on the same foundation as JUnit and NUnit. TestNG is a more robust framework than its competitors due to its advanced and valuable capabilities. Some of the major benefits of TestNG include:

* TestNG allows you to conveniently group test cases, which JUnit does not allow.
* TestNG includes three more levels: @Before/After suite, @Before/AfterTest, and Before/AfterGroup.
* No classes are extended by TestNG. The TestNG framework allows you to define test scenarios that are independent of one another.
* TestNG has extensive reporting options, including HTML and XML reports with detailed information about test execution results. This makes identifying test failures and diagnosing problems much easier
* It also produces logs.
* There are no constraints available in TestNG, such as @beforeclass and @afterclass, which are present in Junit.

**4. What is the difference between a TestNG test and a TestNG test suite?**

The major difference between the TestNG test and the TestNG test suite is aTestNG test is a single Java class with one or more test methods annotated with the @Test annotation. In contrast, a TestNG test suite is an XML file that contains one or more TestNG test definitions.

Here is some other difference between TestNG and TestNG test suites:

* A TestNG test is used to test a specific application functionality or feature, whereas a TestNG test suite is used to bundle related tests together and run them as a unified unit.
* Annotations such as @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest, and others can be used to configure a TestNG test to conduct setup and teardown actions for each test method. To describe the test suite structure and customize its behavior, a TestNG test suite can be configured using various tags such as <suite>, <test>, <classes>, <packages>, and so on.
* A TestNG test runs using a test runner, such as the TestNG plugin for Eclipse or the testng.xml file. While the testng.xml file is used to run a TestNG test suite.

**5. What are some advantages of TestNG over JUnit?**

TestNG is one of the leading automation testing frameworks, and it has several advantages over JUnit. Here is a list of advantages that TestNG has over JUnit:

* Annotations in TestNG are easier to grasp than in Junit.
* It generates HTML reports for use in implementation.
* It also produces logs.
* TestNG allows you to conveniently aggregate test cases, which JUnit does not allow.
* TestNG includes three more levels: @Before/After suite, @Before/AfterTest, and Before/AfterGroup.
* It does not add any new classes. As a result, the TestNG framework allows you to define test cases that are independent of one another.
* Only the TestNG framework allows for parallel execution of test cases, i.e., running many test cases simultaneously.

Also, if you are curious about the difference between JUnit and TestNG, check out our blog on [TestNG vs. JUnit](https://www.lambdatest.com/blog/junit-5-vs-testng/).

**6. Why do we need TestNG in Selenium?**

TestNG is a Selenium test automation testing framework that provides a flexible and powerful way to write and organize test cases. It is a powerful tool for test automation because it includes features such as parallel test execution, test dependency management, flexible test configuration, data-driven testing, and comprehensive reporting.

Selenium Interview Questions

**Note :** We have compiled all [Selenium Interview Questions](https://www.lambdatest.com/learning-hub/selenium-interview-questions) for you. Feel free to visit it. Check it out now!!

**7. What are the basic steps required in writing the TestNG test?**

The following are the basic steps involved in creating a TestNG test:

* Add the TestNG library to your project's classpath after downloading it.
* Make a Java class and annotate it with @Test or another TestNG annotation; in the annotated method, write the test code.
* Use relevant TestNG annotations such as @BeforeMethod, @AfterMethod, @BeforeClass, or @AfterClass to set up the test data and environment.
* Execute the test using TestNG's test runner, which can be invoked from an IDE or the command line.
* Use TestNG's extensive reporting features to analyze test results.

Accelerate TestNG Automation with LambdaTest. [Try LambdaTest Now!](https://accounts.lambdatest.com/selenium-automation-testing-with-testng)

**8. What is the difference between Suite, Test, and Class?**

**Suite:** The highest level of organization in TestNG is a suite. It is a set of one or more tests that can be run concurrently. A suite may include several tests, configurations, and other elements such as listeners, parameters, and groups.

**Test:** A test is a collection of related test cases that can be executed concurrently. A test may contain one or more classes and various parameters and configurations.

**Class:** A class is a Java class that contains one or more test methods. Test methods are marked with @Test and run as part of a test. Other methods for setting up or tearing down the test environment can also be found in a class.

**9. How will you execute methods or test cases in TestNG in a different order/your order?**

We can control the order of method execution in TestNG by using various attributes and annotations. Here are some examples of how to execute methods or test cases in a specific order:

* Priority attribute usage
* Using the dependsOnMethods attribute
* Using the dependsOnGroups property
* Making use of an XML configuration file

**TestNG Interview questions on TestNG XML**

**10. Define the correct order of tags in the TestNG XML file.**

The following is the correct order to run the TestNG suite from the XML file:

<!DOCTYPE suite SYSTEM "https://www.lambdatest.com/">

<**suite** name="MyTestSuite">

<**listeners**>

<!-- Add listeners here -->

</**listeners**>

<**test** name="TestNG QnA">

<**parameter** name="param1" value="value1" />

<**classes**>

<**class** name="com.example.tests.MyTestClass">

<**methods**>

<**include** name="testMethod1" />

<**exclude** name="testMethod2" />

</**methods**>

<**groups**>

<**run**>

<**include** name="group1" />

</**run**>

<**exclude**>

<**group** name="group2" />

</**exclude**>

</**groups**>

<**parameters**>

<**parameter** name="param2" value="value2" />

</**parameters**>

</**class**>

</**classes**>

</**test**>

<**packages**>

<**package** name="com.example.tests.\*" />

</**packages**>

<**method-selector**>

<!-- Add method selector here -->

</**method-selector**>

<**suite-files**>

<!-- Add other TestNG XML files here -->

</**suite-files**>

</**suite**>

Please remember that this is just an example, and the actual tags used may vary depending on your project's specific testing requirements and configurations.

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

The XML file is used in TestNG to configure and customize the test execution process. It contains a number of elements and attributes that define the suite, test cases, test methods, groups, listeners, and other settings that govern how the tests are run.

Here are some reasons why we should create an XML file in TestNG:

* The XML file allows you to configure the test suite, set parameters, and tailor the test execution process to your specific needs.
* You can specify the priority of test methods, and TestNG will execute them in the order you specify.
* You can group the test methods based on their functionalities and execute only the necessary group of methods.
* The test parameters can be specified in the XML file, and TestNG will pass those parameters to the test methods at runtime.
* It allows you to specify which test methods can be executed concurrently, and TestNG will execute them concurrently, saving you time.

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

Here's an example of code that shows how to use the XML file to pass values 1 and 2 to the parameters "val1" and "val2":

// TestNG XML file

<!DOCTYPE suite SYSTEM "https://www.lambdatest.com/">

<**suite** name="MyTestSuite">

<**test** name="MyTest">

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

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

<**classes**>

<**class** name="com.example.MyTestClass" />

</**classes**>

</**test**>

</**suite**>

// Java code

public class MyTestClass {

@Test

public void testMethod(@Parameter("val1") int val1, @Parameter("val2") int val2) {

// Test code goes here

}

}

The "<parameter>" tag is used in this example to specify the values for the "val1" and "val2" parameters in the TestNG XML file. The Java code then declares these parameters with the "@Parameter" annotation and passes their values to the "testMethod()" method.

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

**<test>**

**<suite>**

**<class>**

**<methods>**

**<classes>**

Here is the correct order for testng.xml tags.

**<suite>**

**<test>**

**<classes>**

**<class>**

**<methods>**

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

TestNG supports groups of regular expressions, which might be useful for selecting a subset of tests based on a given pattern.

Here's an example of a TestNG XML file that groups test using regular expressions:

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

<**suite** name="Example Test Suite">

<**test** name="Example Test">

<**groups**>

<**run**>

<**include** name=".\*smoke.\*"/>

</**run**>

</**groups**>

<**classes**>

<**class** name="com.example.tests.Test1"/>

<**class** name="com.example.tests.Test2"/>

<**class** name="com.example.tests.Test3"/>

</**classes**>

</**test**>

</**suite**>

The regular phrase "**.smoke**." is used in this example to pick tests with the term "smoke" in their group name. The include tag specifies the regular expression, and the run tag specifies that only tests that match the regular expression should be executed.

This TestNG XML file will run all tests in the classes **com.example.tests.Test1, com.example.tests.Test2**, and **com.example.tests.Test3** if they are part of a group that fits the regular expression ".smoke."

**15. What is the use of the preserve-order attribute in the TestNG XML file?**

The **preserve-order** attribute in the TestNG XML file specifies whether the test methods in a test class should be executed in the order defined in the Java code. If the preserve-order property is set to true, TestNG will keep the order of test methods as defined in the Java code. TestNG may run the test methods in any order if **preserve-order="false."**

**16. How to Integrate TestNG XML with Maven?**

Add the TestNG dependency to your Maven **pom.xml** file, create a TestNG XML file and add your test classes and configurations, add the **maven-surefire-plugin** to the build section of your pom.xml file, and run your Maven tests using the test command to integrate TestNG XML with Maven.

You can use the below example as a reference to add the **maven-surefire-plugin** to the build section of your Maven pom.xml file and specify the TestNG XML file to use.

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M5</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile>src/test/resources/testng.xml</suiteXmlFile>

</suiteXmlFiles>

</configuration>

</plugin>

</plugins>

</build>

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

You can use the parameter tag within the test tag to pass parameters in the test case via the TestNG XML file. First, You need to specify the parameter name and value within the tag. These parameters can be accessed in the test case by using the @Parameters annotation in the method signature, as shown below:

@Test

@Parameters({"username", "password"})

public void myTestMethod(String username, String password) {

//test method code

}

The myTestMethod method in the preceding example is annotated with the @Parameters annotation, which specifies the parameter names username and password. The method code then uses these parameters to run the test.

**18. How to create an XML file in TestNG?**

* Using a text editor or an IDE, create a new file with the.xml extension.
* To define the DTD, use the <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd"> tag, and the <!—Test suite—> tag to define the test suite.
* Define the test tags within the test suite with the necessary attributes such as name and verbose. And within test tags, define classes or methods that must be executed within each test tag using the class and method tags, respectively.

Here is an example of an XML file in TestNG

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

<**suite** name="MyTestSuite">

<**test** name="MyTest">

<**classes**>

<**class** name="com.example.MyTestClass"/>

</**classes**>

</**test**>

</**suite**>

You should check out our blog on [how to create TestNG XML file](https://www.lambdatest.com/blog/create-testng-xml-file-execute-parallel-testing/) to get a better insight.

**19. How to write regular expressions In testng.xml file to search @Test methods containing the “smoke” keyword.**

You can use the <test> tags include the attribute. Here is an example of code that shows how to write a regular expression in a TestNG XML file:

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

<**suite** name="MyTestSuite" verbose="1">

<**test** name="MyTest" >

<**classes**>

<**class** name="com.example.MyTestClass" />

</**classes**>

<**methods**>

<**include** name=".\*smoke.\*"/>

</**methods**>

</**test**>

</**suite**>

**Annotation related TestNG Interview Questions**

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

[Annotations in TestNG](https://www.lambdatest.com/blog/complete-guide-on-testng-annotations-for-selenium-webdriver/) are used to specify the behavior of test methods. The annotations are carried out in a specific order and hierarchy. The following are the types of annotations used in TestNG in the execution sequence/hierarchy:

Before Suite

Before Test

Before Class

Before Method

**Test** 1

After Method

Before Method

**Test** 2

After Method

After Class

After Test

After Suite

Here is an example that includes all annotations:

**public** **class** **MyTest** {

@BeforeSuite

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

System.out.println("Before Suite");

}

@BeforeTest

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

System.out.println("Before Test");

}

@BeforeClass

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

System.out.println("Before Class");

}

@BeforeMethod

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

System.out.println("Before Method");

}

@Test

**public** void **test1**() {

System.out.println("Test 1");

}

@Test

**public** void **test2**() {

System.out.println("Test 2");

}

@AfterMethod

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

System.out.println("After Method");

}

@AfterClass

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

System.out.println("After Class");

}

@AfterTest

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

System.out.println("After Test");

}

@AfterSuite

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

System.out.println("After Suite");

}

}



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

**Precondition Annotations:** These annotations are executed before the test. It is made up of the following annotations:

@BeforeMethod

@BeforeClass

@BeforeSuite

@BeforeTest

**Test Annotations:** These annotations are defined just before the test methods. It also includes the following annotations:

@Test

**Postcondition Annotations:** These annotations are executed after the test methods. It also includes the annotations listed below:

@AfterMethod

@AfterClass

@AfterTest

@AfterSuite

**Meta-Annotations:** These annotations are used in TestNG to create custom annotations. They provide additional test metadata, such as how the test should be grouped, what data should be used for the test, and which groups should be excluded from the test.

@TestInstance

@Parameters

@DataProvider

@Listeners

@Factory

@Test

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

In TestNG, the @Factory annotation is used to create test instances at runtime. It enables you to generate test classes or instances dynamically based on runtime conditions or parameters.

You can use the @Factory annotation to create multiple instances of the same test class with different data sets or parameters and run them in parallel to improve test execution speed. Here is an example of @factory annotations

@Factory

public static Object**[]** createInstances() {

Object**[]** result = new Object[3];

for (int i = 0; i < result.length; i++) {

result**[i]** = new MyTest(i);

}

return result;

}

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

The **@Factory** annotation is used to create multiple instances of the same test class, each with a different set of input parameters or configurations. This is useful when running the same test with different data sets or configurations or when running tests in parallel to save time during testing.

Meanwhile, **@DataProvider** annotation supply test data to a test method. It allows you to separate the test data from the test logic, making your test code easier to maintain and reuse. @DataProvider can be used to provide test data from a variety of sources, including arrays, Excel sheets, databases, and external files.

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

In TestNG, the @Listener annotation is used to define listeners, which are classes that listen to events that occur during the execution of a TestNG test. Listeners can be used to customize or enhance TestNG's behavior by adding extra functionality such as logging, reporting, or customizing the test execution flow.

You can specify one or more listener classes to be invoked during test execution by using the @Listener annotation. This enables you to extend and customize TestNG's behavior to meet your specific testing requirements.

**25. What is the sequence of execution of the annotations in TestNG?**

The annotations are executed in the following order in TestNG:

@BeforeSuite

@BeforeTest

@BeforeClass

@BeforeMethod

@TestMethod

@AfterMethod

@AfterClass

@AfterTest

@AfterSuite

**26. What are the attributes supported by @Test annotation in TestNG?**

The following is an exhaustive list of the attributes supported by the @Test annotation in TestNG:

* alwaysRun
* dataProvider
* dataProviderClass
* description
* enabled
* expectedExceptions
* expectedExceptionsMessageRegExp
* groups
* invocationCount
* invocationTimeOut
* priority
* successPercentage
* testName
* timeOut
* dependsOnGroups
* dependsOnMethods
* skipFailedInvocations
* alwaysResolveDependencies

**27. Which attribute is used to run the test method always?**

The alwaysRun attribute ensures that a test method is always executed, regardless of whether a dependent method fails or is skipped.

**28. Which attribute is used to provide data to test method in Data-driven testing?**

In Data-driven testing, we use the **dataProvider** attribute of the @Test annotation in TestNG to provide data to a test method.

**29. What is the use of the dependsOnMethods attribute in TestNG?**

The dependsOnMethods attribute in TestNG is used to specify a test method's dependency on one or more other test methods. This property allows us to ensure that certain test methods are run in the correct order.

When one test method is dependent on another, TestNG will not run the dependent method until all of its dependencies have passed successfully. The dependent method will be skipped if any of the dependencies fail or are skipped.

**30. What is the use of dependsOnGroups attribute in TestNG?**

The **dependsOnGroups** attribute in TestNG is used to specify a test method's dependency on one or more groups of test methods. This property allows us to ensure that certain groups of test methods are executed in the correct order.

When a test method is dependent on a group of test methods, TestNG will not execute the dependent method until all of the test methods in the specified group have successfully passed. If any of the group's test methods fail or are skipped, the dependent method will also be skipped.

**31. What is the difference between @BeforeTest and @BeforeMethod annotation?**

The @BeforeTest annotation is used to run the setup code before running all of the test methods in the <test> tag of a TestNG XML file. The @BeforeMethod annotation, on the other hand, is used to run the setup code before each individual test method in a class.

**32. How will you make test cases dependent on each other?**

Using the @Test annotation's **dependsOnMethods** attribute, we can make test cases dependent on one another in TestNG. This attribute allows us to specify which test methods must be run before a specific test method can be executed.

**Report related TestNG interview questions**

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

After all of the test methods have been completed, TestNG generates several reports by default. Such as:

* TestNG HTML reports
* TestNG XML reports
* TestNG index reports
* Emailable reports

Apart from these, the TestNG Listener API allows you to create custom reports. Custom listeners can be created by developers to generate reports in a variety of formats, including JSON, PDF, and CSV.

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

TestNG provides several methods for producing test result reports. Here are some common methods for [generating reports for TestNG](https://www.lambdatest.com/blog/how-to-generate-testng-reports-in-jenkins/) results:

* **TestNG Default Reports:** For test results, TestNG provides default HTML reports. These reports include detailed information about the tests, such as test methods, groups, execution time, and test status.
* **TestNG Ant Task:** TestNG includes an Ant task for creating customized HTML reports. You can use this task to create customized reports with logos, headers, and footers.
* **ExtentReports:** ExtentReports is a popular reporting library that generates rich, interactive HTML reports for TestNG results. It has features such as pie charts, line charts, and screenshots.
* **Allure Reports:** Another popular reporting library that provides detailed and interactive HTML reports for TestNG results is Allure Reports. It includes features such as history, tags, and attachments.
* **ReportNG:** ReportNG is a TestNG custom reporting plugin that generates detailed HTML reports with customizable templates. It includes features such as screenshots, pie charts, and test logs.

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

The emailable report is generated by default in the TestNG project's output directory, in the folder named "**test-output**." The emailable report's filename is "**emailable-report.html**."

So, for a TestNG project, the full path to the emailable report would look something like this:

<**project**-dir>/test-output/emailable-report.html

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

The index.html report is one of TestNG's default reports, providing an overview of the test results as well as links to other reports such as the emailable report, **testng-results.xml**, and **testng-failed.xml**.

The index.html report is generated by default in the TestNG project's output directory under the folder named "**test-output**." The index.html report's filename is "**index.html**."

<**project**-dir>/test-output/index.html

It should be noted that the output directory can be specified in the TestNG XML file by using the "outputDirectory" attribute of the "suite" element.

**37. How to use TestNG Reporter Class for the log generation?**

The TestNG Reporter class is a built-in class for logging messages during test execution. Follow these steps to use the Reporter class for log generation:

* Import the Reporter class as follows:

**import** org.testng.Reporter;

* As an argument, pass a message string to the Reporter.log() method:

Reporter.log("Log message");

* As a second argument to the log() method, you can optionally include a log level, such as "debug", "info", "warning", or "error":
* To view the Reporter class's logs, enable the verbose output option in the TestNG XML file:

<suite name="My Test Suite" verbose="1"></suite>

When the verbose output option is enabled, the [Reporter class logs](https://www.lambdatest.com/blog/how-to-use-testng-reporter-log-in-selenium/) are displayed in the console and included in TestNG reports.

**38. What do we need to generate a customized report in TestNg**

In TestNG, you must use a combination of listeners, reporters, and frameworks to generate a customized report. You can take the following steps:

* Make use of the TestNG IReporter interface.
* Implement the ITestListener interface in TestNG.
* Include the listeners and reporters in your testing.
* Run your test suite and look at your customized report.

**39. How to share the project report using TestNG?**

You can distribute the project report generated by TestNG in the following ways:

* **HTML Reports:** TestNG generates HTML reports with a detailed summary of test execution results. You can distribute these reports to others by simply sending the HTML file or hosting the file on a server and sending the link.
* **Email Reports:** TestNG also allows you to generate and send an email report with a summary of the test execution results. In the testng.xml file, you can configure the email settings, including the email recipients, subject, and body.
* **CI/CD tools:** Integration with Continuous Integration/Continuous Deployment (CI/CD) tools: If you use a CI/CD tool like Jenkins or Bamboo, you can integrate TestNG with these tools to automatically generate and share test execution reports.

**Grouping related TestNG interview questions**

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

TestNG groups enable you to categorize your tests and control their execution based on these categories. By grouping your tests, you can easily select which to run and which to skip based on the group to which they belong.

Groups have several advantages, such as:

* You can organize your tests however you see fit, such as by functionality, priority, or environment.
* Run a subset of the tests in your test suite rather than all of them, which can save you time and effort.
* Test execution in different groups concurrently, which allows for faster execution and better test coverage.
* managing and organizing your tests according to groups, making it easier to maintain and update your test suite.

Check out our blog on the same, If you want to learn step-by-step how to [Group test cases in TestNG](https://www.lambdatest.com/blog/grouping-test-cases-in-testng/).

**41. Define grouping in TestNG?**

The process of categorizing test methods into logical groups based on their functionality or purpose is referred to as grouping in TestNG. We can execute a subset of test methods or all test methods in a group by grouping. We can define groups at several levels, including class, package, and suite.

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

Groups are a collection of multiple test case methods that have been combined into a single unit. We can operate directly on the group by grouping, which will reflect on all of the test case methods under it. Furthermore, in TestNG, we can define a group of groups as a larger unit of test methods.

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

The exclusion a group in TestNG indicates that this group will not run throughout the execution and will be ignored by TestNG. Furthermore, the name of the group to be excluded is defined in the XML file using the following syntax:

<**groups**>

<**run**>

<**exclude** name = "TestNG testing">

</**exclude**>

</**run**>

</**groups**>

**44. How to create a Group of Groups in TestNG?**

You can make a group of groups in TestNG by including one or more groups as dependencies on another group. The following are the steps for creating a group of groups in TestNG:

* In your test class, use the @Test annotation with the groups attribute to define groups. As an example:

@Test(groups = {"group1"})

public void testMethod1() {

// test code

}

@Test(groups = {"group2"})

public void testMethod2() {

// test code

}

@Test(groups = {"group3"})

public void testMethod3() {

// test code

}

* Make a new group with the groups you want to group together as dependencies. As an example:

@Test(groups = {"groupOfGroups"}, dependsOnGroups = {"group1", "group2", "group3"})

**public** void **testGroupOfGroups**() {

// test code

}

In the preceding example, the testGroupOfGroups method is a member of the groupOfGroups group and is dependent on the groups group1, group2, and group3.

* Use TestNG to run your tests. The testGroupOfGroups method will only be called if all of the tests in groups 1, 2, and 3 passes.

You can design complicated test hierarchies by using group dependencies, which allow you to run tests in a certain order and ensure that all dependent tests pass before running a given test.

**45. How to group multiple test methods in a single group using TestNG?**

With the @Test annotation's groups feature, you can organize many test methods into a single group in TestNG. Here's how to use TestNG to group many test methods into a single group:

**Syntex:**

@Test(**groups** = {"GroupName"})

**46. How do group multiple test methods in multiple groups?**

With the @Test annotation's groups feature, you can group multiple test methods into multiple groups in TestNG.

To divide a test method into numerous groups, use the groups attribute to give a comma-separated list of group names. As an example:

@Test(groups = {"Science", "Maths"})

**public** void **college**() {

// test code

}

The college method is included in both Science and Maths in the preceding example. Similarly, you can organize numerous test methods into several groups by using the groups attribute to define the same group names for each method.

**47. How to group multiple test methods with Priority?**

The priority attribute of the **@Test** annotation in TestNG allows you to group multiple test methods with different priorities. The **priority** attribute specifies the order in which TestNG should execute the test methods.

To divide a test method into numerous groups, use the groups attribute to give a comma-separated list of group names. As an example:

@Test(priority = 1, groups = {"Maths"})

public void testMethod1() {

// test code

}

@Test(priority = 2, groups = {"Maths"})

public void testMethod2() {

// test code

}

@Test(priority = 3, groups = {"Science"})

public void testMethod3() {

// test code

}

@Test(priority = 4, groups = {"Science"})

public void testMethod4() {

// test code

}

There are two groups in this example, Science and Maths, as well as four test methods, testMethod1(), testMethod2(), testMethod3(), and testMethod4(). The priority attribute specifies the order in which the test methods within their respective groups should be executed.

**48. What are Inclusion & Exclusion Groups in TestNG?**

An inclusion group is a group that is included in test execution. The exclusion group is a group that is excluded from test execution.

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

TestNG allows us to make a single test rely on a group of tests. We use the **dependsOnGroups** attribute in the TestNG test case file to execute in this manner. The name of the group on which we want this method to rely is the value of this attribute. An illustration of this is provided below:

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

**public** **class** **MyTests** {

@Test(groups = {"login"})

public void testLogin() {

// test login functionality

}

@Test(groups = {"search"}, dependsOnGroups = {"login"})

public void testSearch() {

// test search functionality

}

@Test(groups = {"checkout"}, dependsOnGroups = {"search"})

public void testCheckout() {

// test checkout functionality

}

}

**Assertion related interview Questions for TestNG**

**50. What do you understand by asserting in TestNG?**

[Asserting in TestNG](https://www.lambdatest.com/blog/asserts-in-testng/) is a code element that assists us in determining whether the expected and actual results are equal. To determine whether the test case passed or failed, we use the built-in "Assert" class and many of its methods in TestNG. Furthermore, in TestNG, a test case is considered a "pass" if none of the assert methods throw an exception during execution. The syntax for TestNG assert is as follows:

**import** org.testng.**Assert**;

// **perform** **assertion**

**Assert**.assertEquals(expectedValue, actualValue, message);



**Also Read:** A list of 70 [Cucumber Interview Questions](https://www.lambdatest.com/learning-hub/cucumber-interview-questions) and Answers

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

**assertEquals(expectedValue, actualValue, message):** It compares two values and determines if they are equal.

**assertTrue(condition, message):** This assertion helps whether or not the specified condition is true.

**assertFalse(condition, message):** This assertion defines if the specified condition is true or false.

**assertNull(object, message):** This assertion determines whether or not the specified object is null.

**assertNotNull(object, message):** This assertion specifies if the specified object is null.

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

In TestNG, there are two types of assertions:

* Soft Asserts
* Firm Assertion

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

Soft asserts in TestNG mean that the tests will continue to run even if the assertion throws an exception in the middle of the execution. Furthermore, TestNG does not include Soft asserts by default, so an additional **org.testng.asserts.Softassert** package import is required.

Soft assertions differ from hard assertions in that when a hard assertion fails, the test terminates immediately, and the remaining test steps are skipped. When a soft assertion fails, however, the test continues to run, and any subsequent assertions are also run. Only when the test is completed are the test results, including soft assertion failures, reported, but the test is not terminated.

**Dependency-related interview questions for TestNG**

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

Dependencies are used in TestNG to specify the order in which test methods should be executed. A dependency is a relationship between two or more test methods that specifies that one or more tests must be run before another.

Here is an example of dependency:

@Test(dependsOnMethods = {"testMethod1"})

public void testMethod2(){

// test method code

}

@Test

public void testMethod1(){

// test method code

}

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

In TestNG, we can create dependent tests by specifying the **dependonMethods** parameter on the @Test annotation. The attribute's value is the name of the method on which we want this method to rely. This method is applied as follows:

@Test

public void loginTest() {

// code for login test

}

@Test(dependsOnMethods = {"loginTest"})

public void dashboardTest() {

// code for dashboard test

}

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

Through the TestNG XML file, we can also create dependencies between groups. Such dependencies denote a group's reliance on another. Here's an example of how to use the testng.xml file to create a dependency between two test methods:

<test name="MyTest">

<classes>

<**class** name="com.example.MyTestClass">

<methods>

<include name="loginTest" />

<include name="dashboardTest" dependsOnMethods="loginTest" />

</methods>

</class>

</classes>

</test>

**57. How TestNG allows to declare dependencies?**

TestNG gives two methods to declare dependencies

* dependsOnMethods
* dependsOnGroups

**58. How many types of dependencies can you achieve by using TestNG?**

TestNG is a Java testing framework that allows for various types of dependencies between test methods. The following dependencies can be achieved by using TestNG:

|  |  |
| --- | --- |
| **Dependencies** | **Attribute** |
| Method Dependencies | dependsOnMethods |
| Group Dependencies | dependsOnGroups |
| Class Dependencies | dependsOnClasses |
| Soft Dependencies | alwaysRun |
| Priority Dependencies | priority |

**Micillenious Interview questions for TestNG**

**59. What are the priorities in TestNG?**

Priorities are used in TestNG to define the order in which test methods should be executed. in the @Test annotation, it is used to assign priorities to test methods. Test methods with lower priority values will be executed first, followed by methods with higher priority values.

In our dedicated blog, you can learn practical examples of how to set[priorities in TestNG](https://www.lambdatest.com/blog/prioritizing-tests-in-testng-with-selenium/).

**60. How to set priorities in TestNG?**

Priorities for test methods in TestNG can be set using the priority attribute in the @Test annotation. Here's an example:

@Test(priority=1)

public void testMethod1() {

// test code

}

@Test(priority=2)

public void testMethod2() {

// test code

}

@Test(priority=3)

public void testMethod3() {

// test code

}

In this example, testMethod1() is the most important and will be executed first, followed by testMethod2() and, finally, testMethod3(). You can set the execution order of your test methods by assigning any integer value to the priority attribute.

**61. What is parameterization in TestNG?**

[Parameterization in TestNG](https://www.lambdatest.com/blog/parameterization-in-testng-dataprovider-and-testng-xml-examples/), refers to the process of passing parameters to a test method so that it can be run multiple times with different data sets. This feature is especially helpful when running the same test with different input values or test scenarios.

The "@DataProvider" annotation in TestNG can be used to parameterize code. The [DataProvider annotation](https://www.lambdatest.com/blog/how-to-use-dataproviders-in-testng-with-examples/) specifies a method that provides data to a test method. The data provider method returns a two-dimensional array of objects, with each row representing a set of data used to run the test method.



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

Optional parameters behave similarly to the default case in TestNG parameterization. When no other parameters are defined for that test case method, we use the optional parameter. The **@Optional** annotation also declares the optional parameter. We don't define the **@Optional** parameter above the test method definition but rather alongside it.

Following that, the following code snippet shows how to declare optional parameters in TestNG:

@Test

public void testMethod(@Optional("default") String optionalParam) {

// Test code goes here

}

The "**@Optional**" annotation is used in this example to specify a default value for the "optionalParam" parameter. If this parameter is not supplied when the test method is run, its value will be "**default.**"

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

The timeout attribute in TestNG is used to specify the maximum amount of time (in milliseconds) that a test method should be allowed to run before being forcefully terminated. This property is useful when a test method becomes stuck in an infinite loop or takes too long to complete, causing the entire test suite to stall. It can be declared at

**suite level:** To impose a time limit on all methods in the suite.

**method level:** To impose a time constraint on a specific method.

**Syntex:**

@**Test**(timeout = 5000)

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

The invocationCount attribute specifies how many times a test method must be executed in a single execution. Hence, if I set the invocationCount to 5, the test function will be invoked five times each time I run the TestNG test case.

Code for invocationCount:

@Test (invocationCount = 10)

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

[Parallel test execution in TestNG](https://www.lambdatest.com/blog/parallel-test-execution-in-testng/) means running multiple test cases or test suites on multiple threads at the same time. TestNG is designed to support the parallel execution of tests, which allows testers to run multiple tests concurrently, reducing overall test execution time.

There are two ways to for parallel testing in TestNG:

* Parallel tests at the method level
* Parallel tests at the suite level

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

Parallel testing is possible in TestNG at multiple levels:

* **Methods:** This will run all @Test methods in TestNG in parallel.
* **Tests:** This value will be used to run all test cases contained within the <test>tag.
* **Classes:** All of the test cases contained within the XML classes will run in parallel.
* **Instances:** This value will run all test cases concurrently within the same instance.

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

Exceptions during test execution can be handled in two ways by TestNG:

* The "expectedExceptions" attribute
* The "catch" block method

**Example:**

@Test(expectedExceptions = ArithmeticException.class)

public void testDivideByZero() {

int a = 5;

int b = 0;

int c = a / b;

}

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

Yes, you can disable a test in TestNG by annotating the test method with **@Test(enabled=false)**. Once you disable the test, it won't execute when you run the test script again.

Here is an example of it:

@Test(enabled=**true**)

**public** void **enabledTest**() {

// test code here

}

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

During test execution, the Reporter class in TestNG is used to generate additional logs or custom output. It allows developers or testers to log additional information or messages during test execution, which can help with debugging or troubleshooting test failures.

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

The Reporter.log() method in TestNG can be used to generate logs via the Reporter class. This method's syntax is as follows:

Reporter.log("TestNG web app testing");

**71. What are listeners in TestNG?**

Listeners are classes in TestNG that can be used to customize and extend TestNG's behavior during test execution. They enable custom actions to be performed before or after specific events, such as the start or end of a test case or suite, the success or failure of a test, or the generation of a test report.

TestNG comes with a set of built-in listeners that you can use right away, or you can create your own custom listeners by implementing the necessary interfaces.

**72. How to declare listeners in TestNG?**

There are two ways to declare listeners in TestNG

* Adding listeners to an XML configuration file:

In the TestNG XML configuration file, add the <listeners> tag inside the <suite> tag to declare listeners. Here's an illustration:

<suite name="My Test Suite">

<listeners>

<listener **class**-name="com.example.MyTestListener"/>

</listeners>

<test name="My Test">

<classes>

<class name="com.example.MyTestClass"/>

</classes>

</test>

</suite>

* Annotating the test class with @Listeners:

Listeners can also be declared by annotating your test class with the @Listeners annotation. Here's an example:

**import** org.testng.annotations.Listeners;

@Listeners(com.example.MyTestListener.class)

**public** **class** **MyTestClass** {

// test methods...

}

**73. What are the different listeners TestNG provides?**

TestNG includes a number of built-in listeners that you can use to customize and extend your test suite's behavior. Here's a quick rundown of the most common listeners in TestNG:

* IInvokedMethodListener
* IMethodInterceptor
* IAnnotationTransformer
* IAnnotationTransformer2
* IReporter
* ISuiteListener
* IHookable
* ITestListener

**74. How to Implement TestNG iAnnotationtransformer ?**

To implement the **IAnnotationTransformer** interface in TestNG, you need to create a new Java class that implements the IAnnotationTransformer interface, implement the **transform()** method to modify the **ITestAnnotation** object associated with the test method, and add the listener to your test suite by including it in your TestNG XML configuration file or annotating your test class with the **@Listeners** annotation.

**75. What is the default value for the TestNG Priority?**

The default priority for a test method in TestNG is 0. This means that if a test method is not explicitly assigned a priority using the @Test annotation, it will be assigned a priority of 0 by default.

**76. How to achieve TestNG itestlistener implementation?**

Sure, here are the step-by-step instructions for implementing TestNG ITestListener:

* Make a Java class that conforms to the ITestListener interface.
* Override the necessary methods like **onStart(), onTestSuccess(), onTestFailure(), onTestSkipped(),** and **onFinish()**.
* In each of these methods, add the custom actions you want to perform, such as logging, reporting, or modifying test results.
* Make a separate Java file for the listener class.
* Include the tag and specify the full class name of your listener in your TestNG XML configuration file to add the listener class.
* Alternatively, you can use the @Listeners annotation on your test class and specify the full class name of your listener.
* When you run your test suite, the listener methods will be executed in response to the test events, allowing you to take custom actions.

**77. What is an exception test in TestNG?**

An exception test in TestNG is a type of test that determines whether or not a method throws a specific exception. When an exception is thrown, it is used to ensure that a method behaves as expected.

To create an exception test in TestNG, use the @Test annotation's **expectedExceptions** attribute. This attribute specifies the type of exception that the method is expected to throw.

**78. How will you install TestNG in Eclipse?**

You can install TestNG in Eclipse by following these steps:

* Navigate to the Help menu in Eclipse.
* From the dropdown menu, select Eclipse Marketplace.
* Enter "TestNG" in the search bar.
* Click the Install button after selecting the TestNG for Eclipse plugin from the search results.
* Accept the license agreement and follow the installation prompts.
* When the installation is finished, restart Eclipse.
* TestNG will be available in Eclipse after installation, and you will be able to create TestNG classes and run TestNG tests within the Eclipse environment.

**79. How to throw a SKIP Exception in TestNG?**

In TestNG, use the following code snippet to throw an SKIP exception:

throw **new** SkipException("Skipping this test **method** **due** **to** **some** **reason**...");

The SkipException class in TestNG is a built-in exception class that can be used to skip a test method during execution. The message passed to the SkipException class's constructor is displayed in the test results, indicating why the test was skipped.

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

Refer the question number 67

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

The time unit specified in TestNG test suites and test cases in seconds.

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

<**suite** name="MyTestSuite" verbose="1" time-out="180">

<!-- Test suite contents -->

</**suite**>

In this example, we have set the test suite timeout to 180 seconds.

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

TestNG can be launched in a number of different ways.

* With the help of the command line, where you specify the TestNG JAR file and the test suite XML file with the java command.
* Invoke with the integrated development environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans.
* You can also use build tools such as Maven, and Gradle
* Use a continuous integration (CI) server such as Jenkins, Travis CI, or CircleCI to run TestNG tests.
* Using a test runner such as the TestNG Eclipse plugin, TestNG Maven plugin, or TestNG Ant task to run TestNG tests.

**TestNG Interview questions for experienced**

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

You can use TestNG to build a data-driven framework by following these steps:

* Make a TestNG XML file with the test parameters and data provider. A data provider is a method that provides test data to other methods.
* Implement the data provider method to get test data from a data source like a CSV file, Excel sheet, or database.
* Add the data provider attribute to the @Test annotation of the test method to indicate that the data provided by the data provider method will be used by the test method.
* Use the test data to execute the test logic in the test method.

@Test(dataProvider = "testData")

public void testMethod(String firstName, String lastName, int age) {

System.out.println("Name: " + firstName + " " + lastName + ", Age: " + age);

// Test logic using the test data

}

}

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

You can use the **enabled** attribute of the **@Test** annotation to exclude a specific test method from a TestNG test case execution. The enabled attribute specifies whether or not a test method should be executed. If the enabled attribute is set to **false**, the test method will be skipped during the execution of the test case.

@Test(enabled = **false**)

**public** void **testMethod2**() {

// Test logic

}

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

The **org.testng.SkipException** class can be used to prevent a @Test method from being executed in TestNG. This class is used to throw a skip exception, which instructs TestNG to skip the test method execution.

Here is the code snippet for better understanding:

**import** org.testng.SkipException;

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

**public** **class** **MyTest** {

@Test

**public** **void** **testMethod2**() {

**throw** **new** SkipException("Skipping testMethod2");

}

}

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

Here is the code snippet to run TestNG in the command prompt:

C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriol

set classpath=C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriolin;C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriollib\*

java org.testng.TestNG C:UsersAdminDesktopLTlearninghubworkspaceTestNGtestingtutoriol estng.xml

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

In TestNG, the @Test(threadPoolSize=x) annotation is used to specify the number of threads to use when running a test method in parallel. The x parameter is an integer value that indicates the thread pool size.

When you use this annotation, TestNG will create a pool of threads and distribute test method instances among them. Each thread will independently and concurrently execute its assigned test method instance with the other threads.

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

In TestNG, the @Test(invocationCount=x) annotation is used to specify how many times a test method should be invoked during the test execution. The x parameter is an integer value that specifies how many times the test method should be run.

When this annotation is used, TestNG will run the test method as many times as specified, each time with a new test instance. This is useful in testing scenarios where the test method must be run multiple times with different input data or conditions.

**89. What is the thread count in TestNG?**

The thread-count attribute in TestNG is used to specify the number of threads that will be used to run the test methods. This attribute is used in the TestNG XML file's <suite> tag.

When the thread-count attribute is set to a value greater than one, TestNG will create multiple threads and distribute the test methods among them for parallel execution.

**90. What is verbose in TestNG?**

The **verbose** attribute in TestNG is used to limit the amount of information displayed in the console output during test execution.

The verbose attribute accepts integer values ranging from 0 to 10, with 0 being the least verbose and 10 being the most verbose. The default value is 1, which displays basic test execution information.

**91. How many ways by which c**