1: Tell Me Briefly What Is TestNG And Why Should You Use It?

TestNG is an innovative test framework built on top of the JUnit framework. Its primary consumers are Java developers who use it for the unit testing purpose.

It comes with many useful features like [**TestNG annotations**](https://www.techbeamers.com/testng-annotations-selenium-webdriver/) and [**TestNG assertions**](https://www.techbeamers.com/using-testng-assertions-selenium/) that give a lot of flexibility to testers. Also, there are a couple of solid reasons to use TestNG for automation testing.

* The reporting in the TestNG framework is one of its cornerstone features. It can generate reports both in HTML and XML format. The report provides many indicators for predicting the health of the project. It includes the count of passed/failed cases, the age of a defect and a lot more.
* Since Selenium doesn’t have a built-in report generator, the TestNG report is an ideal candidate to be used for Selenium report generation.
* TestNG provides an excellent exception-handling mechanism to ensure seamless execution of your test automation project.

Q-2: What Are The Core Features Of TestNG Framework?

Some of the notable features of TestNG are as follows.

* **Concurrency –** TestNG can run multiple tests in parallel. It creates a thread for each test and executes them in a thread-safe manner.
* **Dependency –** In TestNG, we can add a test method which depends on a single or a group of test methods. This feature ensures the execution of the dependency methods before the actual test.
* **Prioritization –** We can specify the order of tests using the “priority” parameter inside the @Test annotation.
* **Grouping –** It is a unique feature of TestNG which didn’t exist in the JUnit framework. We can use it for smart/efficient distribution of tests. It lets us create groups with own test methods and groups having other groups. Please note that we can only specify groups in the “TestNG.XML” file. Also, the include/exclude tags can help us customize the test execution in groups.
* **Parameterization –** TestNG has a parameter annotation that allows values to pass as arguments to test methods.
* **Data Sourcing –** TestNG introduces data provider annotation to take input from external sources like excel.

Q-3: What Are The Standard Annotations Available In TestNG?

TestNG annotations control the way the test method below them will get executed. They begin with the “@” symbol.

//TestNG annotation syntax

@<annotation-name>(<attribute-name>=<value>)

public void SampleTest() {

...

}

TestNG supports fifteen different annotations.

1. Test
2. BeforeMethod
3. AfterMethod
4. BeforeClass
5. AfterClass
6. BeforeTest
7. AfterTest
8. BeforeSuite
9. AfterSuite
10. BeforeGroups
11. AfterGroups
12. DataProvider
13. Factory
14. Listeners
15. Parameters

Q-4: What Are The Standard Assertions Available In TestNG?

Assertions not only let us add validation into the tests but also help decide their state (Passed/Failed).

* **assertEquals –** Passes if the arguments match else fails.
* **assertNotEquals –** If the actual and expected values match, then the assertion fails.
* **assertTrue –** Passes if the input condition is true else throws AssertionError.
* **assertFalse –** Passes if the input condition is false else throws AssertionError.
* **assertNull –** Performs the null test on an object. In case of a null, the test aborts with an exception.
* **assertNotNull** – It does the opposite of the “asserNull.” The test method breaks with an exception upon discovering a non-null object.

Q-5: What Are TestNG Hard And Soft Assertions And How Do They Differ?

Hard assertions break a test immediately with an AssertException after the assert method reports failure. If one of the many test cases fails, then the execution continues with the remaining tests.

However, in some cases, we want the test to complete all steps even if one of them fails. TestNG offers to use a soft assertion in such scenarios.

A soft assertion doesn’t throw an exception on failure instead it collects all the errors until it finishes. And that’s how we could manage to run all the steps of a test.

You can read more about these and see examples from the below tutorial.

* [**Hard Assertions in TestNG**](https://www.techbeamers.com/using-testng-assertions-selenium/#hard-assertion)
* [**Soft Assertions in TestNG**](https://www.techbeamers.com/using-testng-assertions-selenium/#soft-assertion)

Q-6: What Is The Purpose Of The “TestNG.XML” File?

The “TestNG.XML” is a configuration file which contains the TestNG project settings. It also represents a TestNG suite which can include one or more tests.

We can do several customizations with this file. The list is as follows.

* Setup multiple tests to run with a single click.
* Include/Exclude test cases.
* Select a specific group or groups.
* Supply parameters for tests.
* Introduce group-level dependencies.
* Setup test parallelization.
* Configure listeners for tests.

Q-7: Can You Run A TestNG Test Without The “TestNG.XML” File? If Yes, Then How?

Yes, we can run the test straight from the command-line using ant or maven. We may need to pass the required class, method or group name in the command.

However, we can only use a limited set of TestNG features without the “TestNG.XML” file.

Q-8: How To Pass A String Value To A Test Using The “TestNG.XML” File?

We can supply a string value using the parameter tag to the desired test case inside the “TestNG.XML.”

<suite name="TestNG Param Suite" verbose="1">

<parameter name="my-string-param" value="my string value" />

<test name="Param Test">

<classes>

<class name="com.techbeamers.test.ParamTest">

<methods>

<include name="paramTest" />

</methods>

</class>

</classes>

</test>

</suite>

Next, we’ll need to access the string value using the parameter annotation just above the test inside the Java code.

package com.techbeamers.test;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class ParamTest {

/\*\*

\* \* Following method takes one param as input argument. \* Its value is set

\* at the suite level.

\*/

@Parameters({ "my-string-param" })

@Test

public void paramTest(String strValue) {

System.out.println("The input parameter is: " + strValue);

}

}

Q-9: How To Pass An Optional Parameter To A Test Using The “TestNG.XML” File?

We can supply an optional parameter with the help of “@Optional” annotation.

Below is the sample “TestNG.XML” file.

<suite name="Optional Param Suite" verbose="1">

<test name="Optional Param">

 <parameter name="opt-value" value="optional text" />

<classes>

<class name="com.techbeamers.test.OptionalParam" />

</classes>

</test>

</suite>

Here is the Java code to access the optional parameter.

package com.techbeamers.test;

import org.testng.annotations.Optional;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class OptionalParam

{

@Parameters({ "opt-value" })

@Test

public void optionTest(@Optional(opt-value") String strValue) {

System.out.println("The value is: " + strValue);

}

}

Q-10: How To Run Multiple TestNG Tests In Parallel?

TestNG supports concurrent executions of tests. We can utilize this feature to run the tests in different browsers simultaneously.

In the TestNG XML, there is a “suite” tag which has a “parallel” attribute. This tag can parallelize the test execution at different levels.

We can set the “parallel” attribute with any of the following values.

* **Tests –** Execute all tests in parallel that fall under the “test” tag.
* **Classes –** Execute all tests in parallel corresponding to the “classes” mentioned in the “classes” tag
* **methods –** Execute all tests in parallel specified with the @Test annotation.
* **Instances –** Execute those tests in parallel which has the same InstanceId else will run them in a different thread.

Q-11: How To Block A Test Method From Execution In TestNG?

We can set the “enabled” attribute to “false” for a test method to block. Its default value is “true.”

Turning this flag to “false” will block the test from execution.

package com.techbeamers.test;

import org.testng.annotations.Test;

public class SimpleTest {

@Test(enabled = false)

public void testOne() {

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

}

@Test(enabled = true)

public void testTwo(String strValue) {

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

}

}

Q-12: How To Block A TestNG Test Depending On A Condition?

We can utilize TestNG’s annotation transformer to initialize the disabled property of a @Test annotation with a “true” or “false” value.

public class TestNGTest implements IAnnotationTransformer {

public void transform(ITest annotation, Class myClass,

Constructor myConstructor, Method myMethod){

if (is\_test\_disabled(myMethod.getName()))) {

annotation.setEnabled(false);

}

}

public boolean is\_test\_disabled(String strName){

// Check if the test is disabled or not.

}

}

Q-13: How To Make A Test Dependent On Another In TestNG?

We can enforce TestNG’s dependency feature in such a case. It provides the “dependsOnMethods” attribute to declare dependencies of a test method. Please follow the below example.

package com.techbeamers.test;

import org.testng.annotations.Test;

public class TestNGDependency {

@Test

public void stepOne() {

System.out.println("Executing stepOne");

}

@Test(dependsOnMethods = { "stepOne" })

public void stepTwo() {

System.out.println("Executing stepOne->stepTwo");

}

@Test(dependsOnMethods = { "stepOne", "stepTwo" })

public void stepThree() {

System.out.println("Executing stepOne->stepTwo->stepThree");

}

}

Q-14: How To Control The Order Of Test Execution In TestNG?

We can set the “preserve-order” attribute to “true” in the “TestNG.XML.”

<test name="MyTestOrder" preserve-order="true">

    <classes>

        <class name="TestNGOrder">

            <methods>

                <include name="stepOne" />

                <include name="stepTwo" />

            </methods>

        </class>

    </classes>

</test>

Alternatively, we can specify a pre-defined sequence value in the “priority” attribute for each test.

package com.techbeamers.test;

import org.testng.annotations.Test;

public class TestNGOrder {

@Test(priority = 1)

public void stepOne() {

System.out.println("Executing stepOne");

}

@Test(priority = 2)

public void stepTwo() {

System.out.println("Executing stepOne->stepTwo");

}

@Test(priority = 3)

public void stepThree() {

System.out.println("Executing stepOne->stepTwo->stepThree");

}

}

Q-15: What If The Priority Of A Test Case In TestNG Is Zero?

By default, every test case in TestNG gets a zero priority.

**Let’s assume:** If there exist two cases – one with priority 1 and another with the default value, then the latter will execute first.

Please note – A test case with lower priority value will always have higher precedence.

Q-16: What Is DataProvider Annotation In TestNG And Why You Use It?

The TestNG framework introduced the data provider annotation to support [**data-driven testing**](https://www.techbeamers.com/writing-data-driven-tests-using-webdriver/). It allows us to use both internal and external sources of input.

If the input size is small, then we can keep the data inside the project files. However, in most cases, the need is to have a big dataset to feed into the test cases. And no tester or a developer would like to bloat his/her code with such a massive set of input.

Hence, the solution is to use the TestNG “@DataProvider” annotation. It can allow a test to accept input from external data sources like CSV, MS Excel, and many others.

Please find below a simple example of using the “@DataProvider” annotation to read input from a 2-D array.

package com.techbeamers.test;

import org.testng.annotations.Test;

import org.testng.annotations.DataProvider;

public class DataDrivenTest {

@Test(dataProvider="empSalary")

public void processSalary(String empName, int empSal) {

System.out.println("Employee: " + empName);

System.out.println("Salary: " + empSal);

}

@DataProvider

public Object[][] empSalary() {

return new Object[][]{{"John", 15000}, {"Andi", 18000}};

}

}

Q-17: What Is Factory Annotation In TestNG And Why You Use It?

The “@Factory” as the name suggests is an annotation which can produce tests at runtime using the given data-sets/conditions.

This annotation solves a common problem of executing tests with different datasets without explicitly creating them.

Another advantage it brings is to allow the parameters to pass to test classes while instantiating them.

Check out the below example to understand it in depth.

package com.techbeamers.test;

import org.testng.TestNG;

public class TestClass

{

private int arg;

public TestClass(int arg) {

this.arg = arg;

}

@Test

public void firstTest() {

int param = arg + 1;

System.out.println("firstTest(): " + param);

}

@Test

public void secondTest() {

int param = arg + 2;

System.out.println("secondTest(): " + param);

}

}

public class TestFactory

{

@Factory

public Object[] factoryMethod() {

return new Object[] { new TestClass(0), new TestClass(1) };

}

}

* **Must Read –**[**How to use TestNG Factory Method?**](https://www.techbeamers.com/use-testng-factory-method-load-testing/)

Q-18: How Is The @Factory Annotation Different From The @DataProvider?

The @Factory annotation leads to following actions.

* It creates the instances of a test class at runtime.
* All the test methods of the TestNG class executes with the given dataset.

Whereas, if we use the @DataProvider, then the following happens.

* The data provider is only applicable for the test methods which comes immediately after it.
* It can run a specific method the no. of times equals to the row count returned by the data provider handler.

Q-19: What Are Listeners In TestNG And Why You Use Them?

TestNG framework defines a set of classes known as listeners to monitor the different states of the tests.

The Listeners extend the following marker interface – “org.testng.ITestListener”. Using them, we can improvise the behavior of a TestNG test.

Below is the list of listeners that TestNG implements.

* **IExecutionListener –** It keeps track of a TestNG run from the beginning to end.
* **IAnnotationTransformer –** It can customize a TestNG annotation and make it work accordingly.
* **ISuiteListener –** It monitors the start and finish of a test suite.
* **ITestListener –** It keeps checking on different states of a TestNG test.
* **IConfigurationListener –** It tracks the events triggered w.r.t. configuration methods.
* **IMethodInterceptor –** It gives us the ability to change the list of test methods during a TestNG run.
* **IInvokedMethodListener –** It adds handlers to invoke before calling the TestNG before and after methods.
* **IHookable –** It allows setting up a callback method to trigger before a test method.
* **IReporter –** It lets us produce a test report for overall execution.

Q-20: What Are The Methods Available In TestNG’s ITestListener Interface?

ITestListener interface can capture the following events/methods.

* **onStart –** It gets triggered after the instantiation of the test class and before the call to any configuration method.
* **onTestSuccess –** It gets triggered if a test succeeds.
* **onTestFailure –** It gets triggered if a test fails.
* **onTestSkipped –** It gets triggered for every test getting ignored.
* **onTestFailedButWithinSuccessPercentage –** It gets triggered for every test that fails but remain below the success percentage.
* **onFinish –** It gets triggered after all the tests finish execution and no configuration method is left.

Q-21: How To Group Tests At Method-Level In TestNG?

We can specify the group names in the “groups” attribute of @Test annotation just before the test methods.

It is useful for executing all tests just by mentioning the group.

In the below example, we’ve one epic, two stories and a no. of tests under them. We’ve created story-centric groups and specified them at method-level.

package com.techbeamers.test;

import org.testng.annotations.Test;

public class TestEpic

{

    @Test(groups={"story1","story2"})

    public void funcTest1() {

        System.out.println("test1");

    }

    @Test(groups="story1")

    public void funcTest2() {

        System.out.println("test2");

    }

    @Test(groups="story2")

    public void funcTest3() {

        System.out.println("test3");

    }

}

Q-22: How To Group Tests At Class-Level In TestNG?

If most of our tests fall under the same group, then no need to repeat the “groups” attribute for each test method. We can annotate the class with @Test and use the “groups” attribute. It’ll then apply to all the public methods of the class.

For example, in the below test class “TestEpic,” the group “story1” defined at class level will cascade to all of its public test methods.

package com.techbeamers.test;

import org.testng.annotations.Test;

@Test(groups="story1")

public class TestEpic

{

    @Test

    public void funcTest1() {

        System.out.println("test1");

    }

    @Test

    public void funcTest2() {

        System.out.println("test2");

    }

    @Test(groups="story2")

    public void funcTest3() {

        System.out.println("test3");

    }

}

Q-23: How Will You Execute Tests From A TestNG Group?

We can follow any of the two methods to run a specific group in TestNG.

* Run via TestNG XML file.
* Execute from Eclipse IDE.

In the TestNG XML, there is a “groups” tag which has subtags namely “include”/”exclude.” we can add our groups to the “include” tag to notify TestNG to run all the tests inside it.

The framework will search the included groups in the test classes added to the “classes” tag. We can also ignore a group by adding it to the “exclude” element.

In the below example runs tests for the “story1” group.

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

<suite name="Suite">

    <test name="story1">

        <groups>

            <run>

                <include name="story1" />

                <exclude name="story2" />

            </run>

        </groups>

        <classes>

            <class name="com.techbeamers.test.TestEpic" />

        </classes>

    </test>

</suite>

For executing groups from Eclipse, we can follow the below steps.

* Start Eclipse, click the “Run” menu, and press “Run Configurations.”
* Click on TestNG, then on the “New Configuration” and provide a label.
* Under the Project menu, press the “Browse” button and choose the project.
* Move to the “Groups” section, again press the “Browse” button to list the groups.
* Choose a group to run.
* Press the “Apply” button and then go ahead to “Run.”

Q-24: How To Run A Single TestNG Test Using Maven?

To run a single TestNG test, we should use the following Maven command.

mvn -Dtest=<Test Class>#<Test Method> test

The above command also supports wildcard characters in class/method names.

Q-25: How To Run A Set Of TestNG Groups Using Maven?

If we have several test groups, e.g., story1, story2, story3, then we can run all of them with a single command.

mvn test -Dgroups=story1,story2,story3