**1.What is TestNG?**

**Answer:**Please refer the below points about TestNG-

* TestNG is an automation testing framework.
* NG stands for "Next Generation".
* Java unit testing framework.
* TestNG is an advance framework designed in a way to leverage the benefits by both the developers and testers.
* TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use.

**2. What are Annotations and what are the different annotations available in TestNG?**

**Answer:**TestNG Annotation is a piece of code which is inserted inside a program or business logic used to control the flow of execution of test methods.

**List of TestNG Annotations**

 Below are the different annotations used in TestNG

* @BeforeSuite: The annotated method will be run only once before all tests in this suite have run.
* @AfterSuite: The annotated method will be run only once after all tests in this suite have run.
* @BeforeTest: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.
* @AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.
* @BeforeClass: The annotated method will be run only once before the first test method in the current class is invoked.
* @AfterClass: The annotated method will be run only once after all the test methods in the current class have run.
* @BeforeMethod: The annotated method will be run before each test method.
* @AfterMethod: The annotated method will be run after each test method.
* @BeforeGroups - The @BeforeGroups annotated method run only once for a group before the execution of all test cases belonging to that group.
* @AfterGroups - The @AfterGroups annotated method run only once for a group after the execution of all test cases belonging to that group.

Please refer link: <https://www.automationtestinginsider.com/2020/03/testng-annotations-and-their-execution.html>

|  |
| --- |
| [Text, letter  Description automatically generated](https://1.bp.blogspot.com/-7158gjpbc58/Xn83xnDP0bI/AAAAAAAAAd8/QwqGTZbgRCQI53bMFpf_4qHhk3N1Lc2rQCLcBGAsYHQ/s1600/TestNGAnnotationsHeirarchy.webp) |
| **TestNG Annotations** |

**Benefits of using TestNG Annotations:**

* TestNG Annotations made the life of testers very easy. Based on your requirements, you can access the test methods, i.e., it has no predefined pattern or format.
* You can pass the additional parameters to TestNG annotations.
* In the case of TestNG annotations, you do not need to extend any test classes.
* TestNG Annotations are strongly typed, i.e., errors are detected at the compile time.

**3. What is JUnit?**

**Answer:**JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which are collectively known as xUnit that originated with sUnit.

**4. What are JUnit annotations?**

**Answer:** Below are theannotations in JUnit

* @Test: This annotation is a replacement of org.junit.TestCase which indicates that public void method to which it is attached can be executed as a test Case.
* @Before: This annotation is used if you want to execute some statement such as preconditions before each test case.
* @BeforeClass: This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.
* @After: This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files, variables, etc.
* @AfterClass: This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.
* @Ignores: This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.
* @Test(timeout=500): This annotation can be used if you want to set some timeout during test execution for e.g. if you are working under some SLA (Service level agreement), and tests need to be completed within some specified time.
* @Test(expected=IllegalArgumentException.class): This annotation can be used if you want to handle some exception during test execution. For, e.g., if you want to check whether a particular method is throwing specified exception or not.

**5. How is TestNG better than JUnit?**

**Answer:**please refer Advantages of TestNG over Junit in below link –

<https://www.automationtestinginsider.com/2020/03/testng-part1-introduction-to-testng.html>

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

**Answer:**TestNG is a Testing framework that covers different types of test designs like a unit test, functional test, end to end test, UI test and integration test. You can run a single or multiple test cases in your Testng code.

You can set test case priority in TestNG by using priority attribute to the @Test annotations. In case priority is not set then the test scripts execute in alphabetical order. Following code snippet prioritizes the test cases:

Now, while running; lower priorities will be scheduled first.

package TestNG;

**import** **org.testng.annotations.**\*;

public **class** **PriorityTestCase**{

**@Test**(priority=**1**)

public void testCase2() {

system.out.println("Test Case 2");

}

**@Test**(priority=**0**)

public void testCase1() {

system.out.println("Test Case 1");

}

}

**Output:**

Test Case 1

Test Case 2

**7. How to ignore test cases?**

**Answer:**When executing TestNG tests, there may be some scenarios where you may have to disable a particular test or a set of tests from getting executed.

For example, consider a scenario where a serious bug exists in a feature due to certain tests belonging to certain scenarios that cannot be executed. As the issue has already been identified we may need to disable the said test scenarios from being executed.

**Ways to Skip / Disable / Ignore Tests in TestNG:**

**1. Suite Level**

Using “exclude” parameter in testng.xml.

TestNg provides an option to include or exclude for Groups, Test Methods, Classes and Packages using include and exclude tags by defining in testng.xml.

Only include methods will be Execute / Run other methods will be skip / ignore.

**Example:**

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

<suite name="Sample Test Suite" verbose="1" >

<test name="Method Test Cases" >

<classes>

<**class** **name**="com.easy.entry.AddTestCase">

<methods>

<include name="addLocationTestCase" />

<include name="addDepartmentTestCase" />

<exclude name="addEmployeeTestCase" />

</methods>

</class>

</classes>

</test>

</suite>

**2. TestClass Level**

To ignore all the tests in class, you need to use @Ignore annotation at the class level.

**@Ignore**

public **class** **IgoneTestClassExample** {

**@Test**

public void testCase1() {

System.out.println("testCase1");

}

**@Test**

public void testCase2() {

System.out.println("testCase2");

}

}

**3. Test Method Level**

**Using @Ignore at test level**

public **class** **IgoneTestClassExample** {

**@Ignore**("not yet ready , Please ignore.")

**@Test**

public void testCase1() {

System.out.println("testCase1");

}

**@Test**

public void testCase2() {

System.out.println("testCase2");

}

}

**Output:**

testCase2

**Using enabled = false at test level**

public **class** **SkipTestClassExample** {

**@Test**

public void testCase1() {

System.out.println("testCase1");

}

**@Test**(enabled = false)

public void testCase2() {

System.out.println("testCase2");

}

}

**Output:**

testCase1

**8. Use of dependson in testng?**

**Answer:**Dependency is a feature in TestNG that allows a test method to depend on a single or a group of test methods. This will help in executing a set of tests to be executed before a test method.

The dependency on multiple test methods is configured for a test by providing comma separated dependent test method names to the attribute dependsOnMethods while using the Test annotation.

The following example shows a test class where process () test method depends on multiple test methods start() and initi() of the same class.

**public** **class** **MultipleDependencyTest** {

**@Test**

**public** **void** **start**() {

System.out.println("Starting the server");

}

**@Test**(dependsOnMethods = { "start" })

**public** **void** **init**() {

System.out.println("Initializing the data for processing!");

}

**@Test**(dependsOnMethods = { "start", "init" })

**public** **void** **process**() {

System.out.println("Processing the data!");

}

**@Test**(dependsOnMethods = { "process" })

**public** **void** **stop**() {

System.out.println("Stopping the server");

}

}

**Output:**

Starting the server

Initializing the data for processing!

Processing the data!

Stopping the server

As seen in the above console output, process () method executed after start() and init() methods are executed and likewise stop() method is executed after process()  method is executed.

**9. How do you generate reports using testng?**

**Answer:**Report generation is very important when you are doing the Automation Testing as well as for Manual Testing. By looking at the result, you can easily identify how many test cases are passed, failed and skipped.

By looking at the report, you will come to know what the status of the project is.

Selenium web driver is used for automating the web-application, but it won't generate any reports.

There are two ways we can generate reports in testng:

**Using emailable-report.html** - The TestNG will generate the default report.

When you execute testng.xml file, and refresh the project. You will get test-output folder in that folder.

Right click on the emailable-report.html and select the option. Open with the web browser.

**Using index.html** - When you execute testng.xml file, and refresh the project. You will get test-output folder in that folder.

Right click on the index.html and select the option. Open with the web browser.

**0. What are the threads in testng , uses of it?**

**Answer:**we can utilize TestNG to perform load testing by creating multiple threads. We can even re-execute a test the no. of times we want.

TestNG is indeed a quite resourceful test automation framework. And integrating it with Webdriver only makes it better to perform automated testing. In this post, we are going to expose two powerful attributes of @Test annotation of TestNG.

These are <invocationCount> and <threadPoolSize>. The first attribute specifies the exact no. of times a test method will get called.  And the latter sets the total no. of threads that will run to call the test method.

**1. TestNG example using @Test(invocationCount=?) attribute.**

The below code will show how does the <invocationCount> attribute call a method multiple times

**public** **class** **LoadTestMultipleTimes** {

**@Test**(invocationCount = **2**)

**public** **void** **testCase1**() {

System.out.println("testCase1");

}

**@Test**()

**public** **void** **testCase2**() {

System.out.println("testCase2");

}

}

**Output:**

testCase1

testCase1

testCase2

**2. TestNG example using @Test(invocationCount=?, threadPoolSize=?) attribute.**

**public** **class** **LoadTestMultipleTimes** {

**@Test**(invocationCount = **2**,threadPoolSize = **2**)

**public** **void** **testCase1**() {

System.out.printf("Thread Id : %s is started!\n", Thread.currentThread().getId());

System.out.println("testCase1");

}

**@Test**()

**public** **void** **testCase2**() {

System.out.printf("Thread Id : %s is started!\n", Thread.currentThread().getId());

System.out.println("testCase2");

}

}

In this TestNG example, we are using both the attributes together while keeping their value same. It’ll let each test method call run on a separate thread.

We’ve added thread ID and timestamp in the log messages so that you can identify the thread and the execution time of the test running.

**Output:**

Thread Id : 12 is started!

testCase1

Thread Id : 11 is started!

testCase1

Thread Id : 1 is started!

testCase2

**11. Which testng annotations you use for extent reports?**

**Answer:**Please refer below links to get to know about extent report

<https://www.automationtestinginsider.com/2020/05/generate-extent-report-and-attach.html>

<https://www.automationtestinginsider.com/2020/05/extent-report-implementation-using.html>

**12. How to prepare customized html reports using testng?**

**Answer:**We will learn to create a custom emailable report.

We need to implement an IReporter interface to create a custom TestNG Report. So, if you implement IReporter by any Java class then you need to override the unimplemented method as per the requirement to display data in the custom report, which is as below:

package Test;

import java.util.List;

import org.testng.IReporter;

import org.testng.ISuite;

import org.testng.xml.XmlSuite;

public class TestReporterClass implements IReporter{

public void generateReport(List<XmlSuite> xmlSuites, List<ISuite> suites, String outputDirectory) {

// TODO Auto-generated method stub

}

}

**STEPS TO CREATE CUSTOM TESTNG HTML REPORT IN SELENIUM**

Here are the steps which you need to implement to create a custom HTML report in TestNG.

**STEP# 1: IMPLEMENT IREPORTER AND OVERRIDE THE UNIMPLEMENTED METHOD**

Above sample code has the unimplemented method. We will first take its second argument, that is, List<ISuite> suites. We will iterate through the ISuite which is basically iteration over the entire test suite.

Once we defined the iteration through for each loop then we put all the test results (consisting passed and failed tests) of the single suite inside the Map. We further get the key from the result map. This key will help further to find failed and passed

Map<String, ISuiteResult> resultMap = ist.getResults();

Set<String> key = resultMap.keySet();

We further use iteration over the keys and we identify the Context objects of the result.

ITestContext cntx = resultMap.get(k).getTestContext();

Context help us further to get the map for failed or passed tests, which further gives their respective methods name.

IResultMap failedTest = cntx.getFailedTests();

Collection<ITestNGMethod> failedMethods = failedTest.getAllMethods();

Finally, we get custom details of the execution in the console and custom TestNG report as well.

**Here is the implementation:**

**package** Test;

**import** **java.util.Collection**;

**import** **java.util.Date**;

**import** **java.util.List**;

**import** **java.util.Map**;

**import** **java.util.Set**;

**import** **org.testng.IReporter**;

**import** **org.testng.IResultMap**;

**import** **org.testng.ISuite**;

**import** **org.testng.ISuiteResult**;

**import** **org.testng.ITestContext**;

**import** **org.testng.ITestNGMethod**;

**import** **org.testng.xml.XmlSuite**;

**public** **class** **TestNGReportCustomizationClass** **implements** IReporter {

**public** **void** **generateReport**(List<XmlSuite> xmlSuites, List<ISuite> suites, String outputDirectory) {

**for** (ISuite ist : suites) {

Map<String, ISuiteResult> resultMap = ist.getResults();

Set<String> key = resultMap.keySet();

**for** (String k : key) {

ITestContext cntx = resultMap.get(k).getTestContext();

System.out.println("Suite Name- " + cntx.getName() + "\n Report Directory- " + cntx.getOutputDirectory()

+ "\n Test Suite Name- " + cntx.getSuite().getName() + "\n Start Date and Time of Execution- "

+ cntx.getStartDate() + "\n End Date and Time of Execution- " + cntx.getEndDate());

IResultMap failedTest = cntx.getFailedTests();

Collection<ITestNGMethod> failedMethods = failedTest.getAllMethods();

System.out.println("------Failed Test Case-----");

**for** (ITestNGMethod imd : failedMethods) {

System.out.println(

"Test Case Name- " + imd.getMethodName() + "\n Description- " + imd.getDescription()

+ "\n Priority- " + imd.getPriority() + "\n Date- " + **new** Date(imd.getDate()));

}

IResultMap passedTest = cntx.getPassedTests();

Collection<ITestNGMethod> passedMethods = passedTest.getAllMethods();

System.out.println("------Passed Test Case-----");

**for** (ITestNGMethod imd1 : passedMethods) {

System.out.println(

"Test Case Name- " + imd1.getMethodName() + "\n Description- " + imd1.getDescription()

+ "\n Priority- " + imd1.getPriority() + "\n Date- " + **new** Date(imd1.getDate()));

}

}

}

}

}

**STEP# 2: CREATE A SAMPLE TEST CASE**

You should create a test case in TestNG class. Here is the sample program.

MYSAMPLETEST.JAVA

**package** Test;

**import** **java.util.concurrent.TimeUnit**;

**import** **org.openqa.selenium.WebDriver**;

**import** **org.openqa.selenium.chrome.ChromeDriver**;

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

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

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

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

**public** **class** **MySampleTest** {

WebDriver driver;

**@BeforeClass**

**public** **void** **setUp**() {

System.setProperty("webdriver.chrome.driver", "C:\\Selenium\\chromedriver.exe");

driver = **new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(**10**, TimeUnit.SECONDS);

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

driver.manage().window().maximize();

}

**@AfterClass**

**public** **void** **tearDown**() {

driver.close();

driver.quit();

}

**@Test**(priority = **1**, description = "My Sample Test Fail")

**public** **void** **testMethod1**() {

String expectedTitle = "TestingFailed";

Assert.assertEquals(driver.getTitle(), expectedTitle, "Title not matched");

}

**@Test**(priority = **0**, description = "My Sample Test Pass")

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

**boolean** matchCondition = driver.getTitle().contains("ATI");

Assert.assertTrue(matchCondition, "Title contains the expected value");

}

}

**STEP# 3: UPDATE TESTNG.XML FILE WITH LISTENERS**

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

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

<suite name="Inviul Sample Test Suite">

<listeners>

<listener class-name="Test.TestNGReportCustomizationClass" />

<listener class-name="Test.ListenersDefinitionClass" />

</listeners>

<test name="Inviul Tests">

<classes>

<**class** **name**="Test.MySampleTest" />

</classes>

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

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

In the end, you need to add the IReporter implemented the class as a listener in your testng.xml file.

**13. Architecture of testng?**

**Answer:**Please refer below link

<https://www.oodlestechnologies.com/blogs/testng-framework-and-its-architecture/>

**14. Write the order of testng annotations?**

**Answer:**Please refer the below link

<https://www.automationtestinginsider.com/2020/03/testng-annotations-and-their-execution.html>

**15. How to run failed test cases in testng?**

**Answer:**Please refer the below link

<https://www.automationtestinginsider.com/2020/03/run-failed-test-cases-in-selenium.html>

**16. When to use dataprovider testng anotation?**

**Answer:**TestNG @DataProvider – Test parameters example. An important features provided by TestNG is the testng DataProvider feature. It helps you to write data-driven tests which essentially mean that same test method can be run multiple times with different data-sets.

**17. What is TestNG Assert and list out some common assertions supported by TestNG?**

**Answer:** Assertions in TestNG are a way to verify that the expected result and the actual result matched or not. If we could decide the outcome on different small methods using assertions in our test case, we can determine whether our test failed or passed overall. An example of assertion can be logging into the website, checking the title of the webpage, verifying the functionality of an input box that takes only integers, etc.

We should remember that an assertion in TestNG is successful only if there are no exceptions thrown during the test case execution. TestNG asserts (or assertions) popularly validate the results in TestNG using selenium.

Please refer the below link

<https://www.automationtestinginsider.com/2020/03/assertions-in-testng.html>

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

**Answer: What is testng.xml?**

testng. xml file is a configuration file in TestNG.

It is used to define test suites and tests. It provides different options to include packages, classes and independent test methods in our test suite. It also allows us to configure multiple tests in a single test suite and run them in multi threaded environment.

**Importance of testng.xml**

In TestNG, you can define multiple test cases in a single class whereas, in Java, you can define only one test in a single class in the main() method. In Java, if you want to create one more test, then you need to create another java file and define the test in the main() method.

Instead of creating test cases in different classes, we recommend you to use TestNG framework that allows you to create multiple test cases in a single class.

Please refer complete details about testing.xml here:

<https://www.automationtestinginsider.com/2020/03/in-this-post-we-will-discuss-about.html>

**19. What is parameterized testing in TestNG?**

**Answer:**Parameterization [Data driven test] is an execution strategy, which allows us to run a test case automatically, multiple times with different input values.

To pass multiple data to the application at runtime, we need to parameterize our test scripts.

There are two ways by which we can achieve parameterization in TestNG

1. With the help of Parameters annotation and TestNG XML file.

2. with the help of DataProvider annotation.

Please refer the below link for complete details and program

<https://www.automationtestinginsider.com/2020/03/parameterization-using-testng.html>

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

**Answer:**TestNG allows you to perform ordered groupings of test methods. You can not only 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 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.

@Test (groups = { "smokeTest", "functionalTest" })

public void loginTest(){

System.out.println("Logged in successfully");

}

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

**Answer:**

**What is Listeners in TestNG?**

Listener is defined as interface that modifies the default TestNG's behavior. As the name suggests Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface. It allows customizing TestNG reports or logs. There are many types of TestNG listeners available.

There are many types of listeners which allows you to change the TestNG's behavior.

Below are the few TestNG listeners:

* IAnnotationTransformer ,
* IAnnotationTransformer2 ,
* IConfigurable ,
* IConfigurationListener ,
* IExecutionListener,
* IHookable ,
* IInvokedMethodListener ,
* IInvokedMethodListener2 ,
* IMethodInterceptor ,
* IReporter,
* ISuiteListener,
* ITestListener

Above Interface are called TestNG Listeners. These interfaces are used in selenium to generate logs or customize the TestNG reports.

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

**Answer:**Please refer answer of question #19

**23. How to pass parameters through testng.xml to a test case?**

**Answer:**Please refer answer of question #19

**24. Explain DataProviders in TestNG using an example**

**Answer: Answer:**Please refer answer of question #19

Please refer the below links as well:

<https://www.automationtestinginsider.com/2020/06/data-driven-framework-part-1.html>

<https://www.automationtestinginsider.com/2020/06/data-driven-framework-part-2.html>

**25. Can I call a single data provider method for multiple functions and classes?**

**Answer:**Yes, the same DataProvider can be used in multiple functions and classes by declaring DataProvider in separate class and then reusing it in multiple classes.

**26. How to skip a @Test method or a code block in TestNG?**

**Answer:**

**There are two ways you can skip Test Method**

**1. Using @Ignore at test level**

public class IgoneTestClassExample {

    @Ignore("not yet ready , Please ignore.")

    @Test

    public void testCase1() {

        System.out.println("testCase1");

      }

     @Test

    public void testCase2() {

        System.out.println("testCase2");

      }

}

**Output:**

testCase2

**2. Using enabled = false at test level**

public class SkipTestClassExample {

    @Test

    public void testCase1() {

        System.out.println("testCase1");

      }

     @Test(enabled = false)

    public void testCase2() {

        System.out.println("testCase2");

      }

}

**Output:**

testCase1

**27. What is Soft Assertion in selenium and how can you mark a test case as failed by using soft assertion?**

**Answer:**Soft Assertions (Verify):

It is a custom assert mechanism supported by TestNG’s “org.testng.asserts.Softassert” package. We use it when a test has to continue execution even after an assertion fails in the sequence.

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.

**public** **class** **SoftAssertionDemo** {

**@Test**

**public** **void** **testCase1**() {

SoftAssert obj= **new** SoftAssert();

System.out.println("==================================");

System.out.println("Test1 Start");

obj.assertEquals("Hi", "HI");

System.out.println("Test1 End");

System.out.println("==================================");

obj.assertAll();

}

**@Test**

**public** **void** **testCase2**() {

SoftAssert obj1= **new** SoftAssert();

System.out.println("==================================");

System.out.println("Test2 Start");

obj1.assertEquals("HELLO", "HELLO");

System.out.println("Test2 End");

System.out.println("==================================");

obj1.assertAll();

}

}

**Output:**

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

Test1 Start

Test1 End

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

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

Test2 Start

Test2 End

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

PASSED: testCase2

FAILED: testCase1

java.lang.AssertionError: The following asserts failed:

expected [HI] but found [Hi]

**28. Explain what is a Group Test in TestNG?**

**Answer:**TestNG Groups allow you to perform groupings of different test methods. Grouping of test methods is required when you want to access the test methods of different classes.

Not only you can declare the methods within a specified group, you can also declare another group within a specified group. Thus, TestNG can be asked to include a certain set of groups while excluding another set of groups.

It provides you maximum flexibility by partitioning your test methods in groups and does not require recompilation of test cases if you run your two different sets of test cases back to back.

Groups are specified in the testng.xml file with <groups> tag. Groups can be specified either in the <suite> tag or <test> tag. If the <groups> tag is specified inside the <suite> tag, then it is applied to all the <test> tags of XML file. If the <groups> tag is specified within a particular <test> folder, then it is applied to that particular <test> tag only.

**First case: When <groups> tag is defined inside the <suite> tag.**

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

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

<suite name="test\_suite">

<groups>

<run>

<include name="SmokeTest"/>

</run>

</groups>

<test name="Personal Loan">

<classes>

<**class** **name**="com.javatpoint.Personal\_loan"/>

</classes>

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

<test name="Home Loan">

<classes>

<**class** **name**="com.javatpoint.Home\_loan"/>

</classes>

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

<test name="Car Loan">

<classes>

<**class** **name**="com.javatpoint.Car\_loan"/>

</classes>

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

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

In the above case, we provide a group name, i.e., SmokeTest to three test cases of three different classes.

**Second case: When <groups> tag is defined inside the <test> tag.**

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

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

<suite name="test\_suite">

<test name="Loan">

<groups>

<run>

<include name="SmokeTest"/>

</run>

</groups>

<classes>

<**class** **name**="com.javatpoint.Personal\_loan"/>

<**class** **name**="com.javatpoint.Home\_loan"/>

<**class** **name**="com.javatpoint.Car\_loan"/>

</classes>

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

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

**Tests belonging to multiple Groups**

@Test(groups= {"Group A","Group B"})

public void testcase1()

{

System.out.println("Test case belonging to both Group A and Group B");

}

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

**Answer:**The annotation like @Factory and @DataProvider are mainly used to reiterate the same test class with different test data. These annotations will help the user to use the same class seamlessly without duplicating the test class code.

**@DataProvider Annotation**

A test method that uses DataProvider will be executed multiple numbers of times based on the data provided by the DataProvider. That means this annotation parametrizes the particular test method and executes the test number of times based on the data provided by the DataProvider method.

The condition that needs to be met here is that the method marked as @DataProvider must return a 2D Object array (Object[][]) where each Object[] will be used as the input parameter to an iteration of the test method which uses the data provider.

The Test method will be executed using the same instance of the test class to which the test method belongs.

**@Factory Annotation**

It can be used to execute all the test methods present inside a test class with multiple sets of data, using the separate instance of the class.

Using this, we can instantiate a class multiple times rather than just a method.

The factory method should return an Object[]. This can be an array of Method calls or class objects.

The Test method will be executed using the separate instance of the respective class.

**Example:**

**Using @DataProvider**

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

{

**@BeforeClass**

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

System.out.println("Before class executed");

}

**@Test**(dataProvider = "dp1")

**public** **void** **testMethod**(String param) {

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

}

**@DataProvider(name = "dp1")**

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

**return** **new** Object[][] { { "one" }, { "two" } };

}

}

**Output:**

Before class executed

The parameter value is: one

The parameter value is: two

PASSED: testMethod("one")

PASSED: testMethod("two")

**Using @Factory**

The SimpleTest class contains the testMethod() and beforeClass() methods.

The constructor of the test class takes a String argument value. Both beforeClass() and testMethod() print a message onto console.

**public** **class** **SimpleTest**

{

**private** String param = "";

**public** **SimpleTest**(String param) {

**this**.param = param;

}

**@BeforeClass**

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

System.out.println("Before SimpleTest class executed.");

}

**@Test**

**public** **void** **testMethod**() {

System.out.println("testMethod parameter value is: " + param);

}

}

**public** **class** **SimpleTestFactory**

{

**@Factory**

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

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

**new** **SimpleTest**("one"),

**new** **SimpleTest**("two")

};

}

}

**Output:**

Before SimpleTest class executed.

testMethod parameter value is: two

Before SimpleTest class executed.

testMethod parameter value is: one

PASSED: testMethod

PASSED: testMethod

**30. What is the difference between @BeforeMethod and @BeforeClass ?**

**Answer:**@BeforeMethod: The annotated method will be run before each test method. @BeforeClass: The annotated method will be run before the first test method in the current class is invoked. @BeforeTest: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.

**31. What are the different attributes for @Test annotation?**

**Answer:**Please refer below link to understand different attributes in testing

<https://www.automationtestinginsider.com/2020/03/different-attributes-in-testng.html>

**32. How can we run test cases in parallel using TestNG?**

**Answer:**Please refer below link to understand to run test cases in parallelin testing

<https://www.automationtestinginsider.com/2020/03/parallel-testing-in-testng.html>

**33. Suppose I want to check a particular exception in TestNG. How will you check?**

**Answer:**TestNG provides an option of tracing the exception handling of code. You can test whether a code throws a desired exception or not. Here the expectedExceptions parameter is used along with the @Test annotation. Now, let's see @Test(expectedExceptions) in action.

**public** **class** **MessageUtil** {

**private** String message;

//Constructor

//@param message to be printed

**public** **MessageUtil**(String message) {

**this**.message = message;

}

// prints the message

**public** **void** **printMessage**() {

System.out.println(message);

**int** a =**0**;

**int** b = **1**/a;

}

}

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

**public** **class** **ExpectedExceptionTest** {

String message = "ExceptionTest";

MessageUtil messageUtil = **new** MessageUtil(message);

**@Test**(expectedExceptions = ArithmeticException.class)

**public** **void** **testPrintMessage**() {

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

messageUtil.printMessage();

}

}

**Output:**

Inside testPrintMessage()

ExceptionTest

**34. How Cross Browser testing is handled in Selenium?**

**What is Cross Browser Testing?**

Cross Browser Testing is a type of functional test to check that your web application works as expected in different browsers?

**Why do we need?**

A web application can be opened in any browser by the end user. For example, some people prefer to open in Firefox, some in chrome, ie etc. We need to ensure that the web application will work as expected in all popular browsers so that more people can access it and use it.

This motive can be fulfilled with Cross Browser testing of the product.

**Please refer below link for complete cross browser testing:**

<https://www.automationtestinginsider.com/2020/03/cross-browser-testing-in-testng.html>

**35. Explain the structure of testng.xml file?**

**Answer:**please refer the below image for basic structure of testng.xml

|  |
| --- |
| [Diagram  Description automatically generated](https://1.bp.blogspot.com/-TFCat2hGk4c/Xzfy6tKCMzI/AAAAAAAAAnE/Pvx90KZJI202bRT9FyTNy25wQaF8F_zGACLcBGAsYHQ/s399/testng_structure.png) |
| **testng.xml** |

**36. What are the different methods of Assert?**

**Answer:**All the assertions are in the Assert class.

public class Assert extends java.lang.Object

This class provides a set of assertion methods, useful for writing tests. Only failed assertions are recorded. Some of the important methods of Assert class are as follows −

* void assertEquals(boolean expected, boolean actual)- Checks that two primitives/objects are equal.
* void assertTrue(boolean condition)- Checks that a condition is true.
* void assertFalse(boolean condition) - Checks that a condition is false.
* void assertNotNull(Object object) - Checks that an object isn't null.
* void assertNull(Object object) - Checks that an object is null.
* void assertSame(object1, object2) - The assertSame() method tests if two object references point to the same object.
* void assertNotSame(object1, object2) - The assertNotSame() method tests if two object references do not point to the same object.
* void assertArrayEquals(expectedArray, resultArray); - The assertArrayEquals() method will test whether two arrays are equal to each other.

**36. What are the different methods of Assert?**

**Answer:**All the assertions are in the Assert class.

public class Assert extends java.lang.Object

This class provides a set of assertion methods, useful for writing tests. Only failed assertions are recorded. Some of the important methods of Assert class are as follows −

* void assertEquals(boolean expected, boolean actual)- Checks that two primitives/objects are equal.
* void assertTrue(boolean condition)- Checks that a condition is true.
* void assertFalse(boolean condition) - Checks that a condition is false.
* void assertNotNull(Object object) - Checks that an object isn't null.
* void assertNull(Object object) - Checks that an object is null.
* void assertSame(object1, object2) - The assertSame() method tests if two object references point to the same object.
* void assertNotSame(object1, object2) - The assertNotSame() method tests if two object references do not point to the same object.
* void assertArrayEquals(expectedArray, resultArray); - The assertArrayEquals() method will test whether two arrays are equal to each other.

**37. What the difference between include and exclude in TestNG?**

**Answer:**TestNg provides an option to include or exclude Groups, Test Methods, Classes and Packages using include and exclude tags by defining in testng.xml.

We will create a Class with three Test Methods. In that we will include two test methods and try to exclude one test method.

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

**public** **class** **AddTestCase** {

**@Test**

**public** **void** **addLocationTestCase**() {

System.out.println("Im in add location test case");

}

**@Test**

**public** **void** **addDepartmentTestCase**() {

System.out.println("Im in add department test case");

}

**@Test**

**public** **void** **addEmployeeTestCase**() {

System.out.println("Im in add employee test case");

}

}

In the above class example, we have created three test methods, 'addLocationTestCase', 'addDepartmentTestCase', and

'addEmployeeTestCase'.

In the below testng.xml file we will exclude 'addEmployeeTestCase' and try to execute the program. We need to first add the class name and in that class , we need to define the methods which needs to be included and excluded

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

<suite name="Sample Test Suite" verbose="1" >

<test name="Method Test Cases" >

<classes>

<**class** **name**="com.easy.entry.AddTestCase">

<methods>

<include name="addLocationTestCase" />

<include name="addDepartmentTestCase" />

<exclude name="addEmployeeTestCase" />

</methods>

</class>

</classes>

</test>

</suite>

After running the above testng.xml file, we will get the output as It will just run the test methods which are included in the class. And will not execute the test methods which are in excluded.

**38. How to execute the single selected method in TestNG?**

**Answer:**To include only a particular @Test method to run like below:

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

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

<suite name="Suite">

  <test name="Test">

    <classes>

      <class name ="Full path to the Test class" />

         <methods>

            <include name="testMethodName" />

         </methods>

    </classes>

  </test>

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

**39. How the packages and classes are structured in TestNG.xml?**

**Answer:**Please refer Hierarchy of the TestNG Annotations in below link:

<https://www.automationtestinginsider.com/2020/03/testng-annotations-and-their-execution.html>

<https://www.automationtestinginsider.com/2020/03/in-this-post-we-will-discuss-about.html>

**40. Tell me a login page script in TestNG?**

**Answer:**

**import** **java.util.concurrent.TimeUnit**;

**import** **org.openqa.selenium.By**;

**import** **org.openqa.selenium.WebDriver**;

**import** **org.openqa.selenium.WebElement**;

**import** **org.openqa.selenium.chrome.ChromeDriver**;

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

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

**public** **class** **LoginUsingSelenium** {

**@Test** **public** **void** login() {

// TODO Auto-generated method stub

System.setProperty("webdriver.chrome.driver", "path of driver");

WebDriver driver=**new** ChromeDriver(); driver.manage().window().maximize();

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

WebElement username=driver.findElement(By.id("username"));

WebElement password=driver.findElement(By.id("password"));

WebElement login=driver.findElement(By.xpath("//button[text()='Sign in']"));

username.sendKeys("example@gmail.com"); password.sendKeys("password");

login.click(); String actualUrl="https://www.linkedin.com/feed/";

String expectedUrl= driver.getCurrentUrl();

Assert.assertEquals(expectedUrl,actualUrl);

}

}

**41. What is the difference between @Parameters and @DataProviders in TestNG?**

**Answer:**DataProviders pass the different parameters on a single test in a single execution, whereas parameters pass the parameters just once per execution in TestNG.

**42. How to create Suites in TestNG?**

**Answer:**Please refer **Different ways to setup testng.xml** section in below link:

<https://www.automationtestinginsider.com/2020/03/in-this-post-we-will-discuss-about.html>

**43. How to prioritize the tests in TestNG at Class level and Suite level?**

**Answer:**

**1. at Test Method Level:**

Here you have to consider 3 points:

**If you are not using any priority in your test method then TestNG assign by default priority=0 to the Test Method and test method will be executed in alphabetical order**

**public** **class** **ClassA** {

**@Test**

**public** **void** **testC**() {

System.out.println("ClassC");

}

**@Test**

**public** **void** **testA**() {

System.out.println("ClassA");

}

**@Test**

**public** **void** **testB**() {

System.out.println("ClassB");

}

}

**Output:**

ClassA

ClassB

ClassC

**If there is same priority assign to test methods then execution order will be alphabetically.**

**public** **class** **ClassA** {

**@Test**(priority=**1**)

**public** **void** **testC**() {

System.out.println("ClassC");

}

**@Test**(priority=**1**)

**public** **void** **testA**() {

System.out.println("ClassA");

}

**@Test**

**public** **void** **testB**() {

System.out.println("ClassB");

}

}

**Output:**

ClassB

ClassA

ClassC

**In Case of different priority, execution will be from minimum to maximum priority:**

**public** **class** **ClassA** {

**@Test**(priority=-**1**)

**public** **void** **testC**() {

System.out.println("ClassC");

}

**@Test**(priority=**1**)

**public** **void** **testA**() {

System.out.println("ClassA");

}

**@Test**(priority=**0**)

**public** **void** **testB**() {

System.out.println("ClassB");

}

}

**Output:**

ClassC

ClassB

ClassA

**2. At Suite Level:** classes will be executed in the order as they defined

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

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

<suite name="Suite">

<test thread-count="5" name="Test">

<classes>

<**class** **name**="testNG.ClassB"/>

<**class** **name**="testNG.ClassA"/>

</classes>

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

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

In the above example ClassB will be executed first and then ClassA

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

**Answer:**We have below test class which has three test methods, which belongs to different groups**.**

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

**public** **class** **Groups** {

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

**public** **void** **testcase1**() {

System.out.println("Test case belonging to Sanity");

}

**@Test**(groups = { "Sanity", "Smoke" })

**public** **void** **testcase2**() {

System.out.println("Test case belonging to both Sanity and Smoke");

}

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

**public** **void** **testcase3**() {

System.out.println("Test case belonging to Regression");

}

}

**Now let’s say if we want to execute all the groups at once. Refer below testing.xml**

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

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

<suite name="Suite">

<test thread-count="5" name="Test">

<groups>

<run>

<include name="Sanity"></include>

<include name="Smoke"></include>

<include name="Regression"></include>

</run>

</groups>

<classes>

<**class** **name**="testNG.Groups" />

</classes>

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

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

**Output:**

Test case belonging to Sanity

Test case belonging to both Sanity and Smoke

Test case belonging to Regression

**45. How to exclude a particular group from a test case execution using TestNG?**

**Answer:**we can take the help of above example. Suppose we want to remove **Regression** group.

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

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

<suite name="Suite">

<test thread-count="5" name="Test">

<groups>

<run>

<include name="Sanity"></include>

<include name="Smoke"></include>

<exclude name="Regression"></exclude>

</run>

</groups>

<classes>

<**class** **name**="testNG.Groups" />

</classes>

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

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

**Output:**

Test case belonging to Sanity

Test case belonging to both Sanity and Smoke

**46. How to write regular expressions in testng.xml file to search @Test methods containing “smoke” keyword?**

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

<methods>

     <include name=".\*smoke.\*"/>

</methods>

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

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

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

**Answer:**TestNG can be invoked in the following ways

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

**49. How to run TestNG using command prompt?**

**Answer:**

Step 1: Open notepad

Step 2: Paste the below lines of code - You may need to add your project location. In the example, project location is set as 'F:\Selenium\TestNGBatchExample'.

Step 3: Save the file as 'testNGBatchFile.bat' in location that you want to save.

set projectLocation=F:\Selenium\TestNGBatchExample

cd %projectLocation%

set classpath=%projectLocation%\bin;%projectLocation%\lib\\*

java org.testng.TestNG %projectLocation%\testng.xml

pause

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

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

@Test(invocationCount = 5)

public void testCase1(){

}

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

**Answer:**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

@Test(threadPoolSize = 3, invocationCount = 10)

public void testCase1(){

}

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

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

**Answer:**The maximum number of milliseconds a test case should take.

@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.

**53. Is it possible to pass test data through testng.xml file, if yes how?**

**Answer:**TestNG allows the user to pass values to test methods as arguments by using parameter annotations through testng. xml file. Sometimes it may be required for us to pass values to test methods during run time. ... The @Parameters annotation can be placed on any method that has a @Test, @Before/After or @Factory annotation.

**54. Do you run test cases in parallel with TestNG? If yes how many threads and does it cause any problem?**

**Answer:**TestNG provides multiple ways to execute tests in separate threads. In testng.xml, if we set 'parallel' attribute on the tag to 'tests', testNG will run all the ‘@Test’ methods in tag in the same thread, but each tag will be in a separate thread.

If we want to run methods/classes in separate threads, we need to set 'parallel' attribute on the tag to 'methods' / 'classes'

This helps us to run test methods / classes / tests in parallel. By using parallel execution, we can reduce the 'execution time' as tests are executed simultaneously in different threads.

In testNG we can achieve parallel execution by two ways. One with testng.xml file and we can configure an independent test method to run in multiple threads.

**55. What’s TestNG Listener Class & why do we use it?**

**Answer:**TestNG Listeners also allows you to customize the tests logs or report according to your project requirements.

TestNG Listeners in Selenium WebDriver are modules that listens to certain events and keep track of test execution while performing some action at every stage of test execution.

TestNG Listeners in Selenium WebDriver can be implemented at two levels:

Class level: In this, you implement listeners for each particular class no matter how much test cases it includes.

Suite level: In this, you implement listeners for a particular suite which includes several classes as test cases.

There are numerous TestNG listeners in Selenium WebDriver, some of them are used very frequently by the testing community & some are almost forgotten. In this TestNG tutorial, I will demonstrate the most popular TestNG listeners with examples but before that, let me enlist the various TestNG listeners in Selenium WebDriver.

* ITestListener
* IAnnotationTransformer
* IInvokedMethodListener
* ISuiteListener
* IReporter
* IConfigurable
* IExecutionListener
* IHookable
* IMethodInterceptor
* IConfigurationListener

**56. How will you get the browser values from testng?**

**Answer:**Please refer below link

<https://www.automationtestinginsider.com/2020/03/cross-browser-testing-in-testng.html>

**57. Can we run testNG class code without using any TestNg annotation?**

**Answer:**annotations belong to what is called reflection and meta-programming. Also, it's not necessary to have main() method in your tests, but you can use main() method to run the TestNg tests if you want.

**58. How to Install TestNG In Eclipse? How do you verify that TestNg Is Installed properly In Eclipse?**

**Answer:**Please refer below link

<https://www.automationtestinginsider.com/2020/03/testng-part1-introduction-to-testng.html>

**59. What is Error Collector in TestNG? What is its use?**

**Answer:**This class allows the collection of errors during the process of retrieving the test data for the test method parameters.

**60. Detail about TestNG Test Output folder.?**

**Answer:**By default the report files (HTML & XML) are written to a folder named test-output under your workspace. Netbeans however overrides this location. It places output to build/test/results folder. Please re-run the TestNG test suite and watch results folder. All required files will be generated there.

**61. What is the difference between WebDriver Listeners and TestNG Listeners?**

**Answer:**Listeners are those which will be listening to any state change (event).

So in case of WebDriver, whenever anyone clicks on a button on a web page, then automatically a method will be called (actionPerformed() in ActionListener case).

But in case of TestNG, there are the actual "listeners" which listen (here the method gets called automatically) to the test execution events. A few examples are: onStart(), beforeStart(), afterFinish(), and onFinish() etc. Mostly, the TestNG automation teams implement their own custom listeners for custom Logging and reporting.

WebDriver listeners too do a similar job...of logging and/or reporting. But then both of them work on different event sets. WebDriver works on different automation events whereas TestNG works on different test's related events. The point to note is that, with WebDriver listeners, "Logging" happens before/after event occurrence.

**62. Explain how does TestNG allow you to state dependencies with an example?**

**Answer:**Sometimes, you may need to invoke methods in a test case in a certain order. Here comes TestNG Dependencies into the picture. TestNG allows you to specify dependencies either with annotations or in XML.

TestNG allows you to specify dependencies either with:

Using attribute dependsOnMethods in @Test annotations, OR.

Using attribute dependsOnGroups in @Test annotations.

**1.Let’s see first dependsOnMethods in @Test annotations**

public class DependsOnMethodsTestCase {

@Test

public void testCase1(){

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

}

@Test

public void testCase2(){

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

}

}

**testng.xml**

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

 <suite name="testdependsOn">

<test name="testngTest">

<classes>

<class name="com.test.DependsOnMethodsTestCase" />

</classes>

</test>

</suite>

**Output**

Test Case 1

Test Case 2

**Now we add the dependsOnMethods attribute to the @Test Annotations and execute the same program.**

public class DependsOnMethodsTestCase {

@Test(dependsOnMethods = {"testCase2"})

public void testCase1(){

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

}

@Test

public void testCase2(){

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

}

}

**Execute the same testng.xml which was placed above and see the difference in Console Output**

**Output**

Test Case 2

Test Case 1

**2. Let’s see Dependencies with XML:**

public class DependsOnMethodsTestCase {

@Test(groups = {"FirstGroup"})

public void testCase1(){

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

}

@Test(groups = {"SecondGroup"})

public void testCase2(){

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

}

}

**testng.xml**

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

<suite name=" testdependsOn ">

<test name="testngTest">

<groups>

                    <dependencies>

                          <group name="FirstGroup" depends-on="SecondGroup"></group>

                    </dependencies>

                </groups>

<classes>

<class name="com.test.DependsOnMethodsTestCase" />

</classes>

</test>

</suite>

**Output**

Test Case 2

Test Case 1

**63. Sequence of execution of below annotations: @Test @BeforeGroups @AfterGroups @BeforeSuite @AfterSuite @BeforeMethod @AfterMethod @BeforeClass @AfterClass?**

**Answer:**Please refer below link

<https://www.automationtestinginsider.com/2020/03/testng-annotations-and-their-execution.html>

**64. Explain execution seq-- @Test(priority=1) @Test(priority=2) @Test(priority=0) @Test(priority=-1)**

**Answer:**priority sequence would be: -1, 0, 1, 2.

**65. Explain execution seq-- @Test(priority=1) @Test(priority=2) @Test()**

**Answer:**If priority not assigned then automatically it will assigned 0 priority, hence the sequence would be: 0, 1, 2.

**Please refer below YouTube video to understand the explanation of above Q/A**