Basics of TestNG for Interview

1. ***What is the TestNG Framework? What are the advantages of TestNG?***

TestNG framework is a testing framework to perform tests in the java programming language. Moreover, the "***NG***" in TestNG abbreviates for "***Next Generation***".

TestNG has the following advantages:

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

2. ***What is the difference between a TestNG test and a TestNG test suite?***
   TestNG test suite refers to a collection of tests that we can run simultaneously with the help of the TestNG XML file. On the other side, a TestNG test is a single test case file, and when we say "***we are running a TestNG test case***", we simply mean we are running a single test case file.

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

```
<suite>
  <test>
    <classes>
   <class>
<methods>
```

The closing tags don't appear here as it is just for demonstration purposes.

TestNG Annotations

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

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

- `@BeforeSuite`
- `@BeforeTest`
- `@BeforeClass`
- `@BeforeMethod`
- `@Test`
- `@AfterMethod`
- `@AfterClass`
- `@AfterTest`
- `@AfterSuite`

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

TestNG annotations divide into three categories:

- ***Precondition Annotations***: *The annotations under this category execute before the test. It consists of the following annotations*:
- `@BeforeMethod`
- `@BeforeClass`

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

TestNG Reports

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

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

- *Emailable Reports*
- *Index Reports*

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

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

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

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

TestNG Priorities

9. ***What are priorities in TestNG?***

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

10. How would you set priorities in TestNG?

TestNG priority is set by the following syntax:

```
@Test (priority = 1)
public void func(){
 //test code
}
```

An example of prioritization in TestNG can be as follows:

```
       @Test (priority = 1)
        public void CloseBrowser() {
                driver.close();
                System.out.println("Closing Google Chrome browser");
        }

        @Test (priority = 0)
        public void OpenBrowser() {
                System.out.println("Launching Google Chrome browser");
                driver.get("https://www.demoqa.com");
        }
```

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

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

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

TestNG Parameters

12. What is parameterization in TestNG?

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

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

13. What are the optional parameters in TestNG?

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

```
import org.testng.annotations.Optional;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class Params
{
    @Test
    @Parameters ("message")
    public void OP( @Optional("Optional Parameter Selected") String message) {
        System.out.println(message);
    }
}
```

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

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

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

Observe the following XML file denoting the same concept.

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

<suite name="My Test-Suite" >
  <test name="QA" >
    <parameter name="val1" value="1" />
    <parameter name="val2" value="2" />
    <classes>
       <class name="testNGPackage.Parameter" />
       <class name="testNGPackage.Multiple_Parameters" />
    </classes>
   </test>
</suite>
```

TestNG Groups

15. What is the importance of groups in TestNG?

Another important TestNG interview questions are about its importance.

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

16. How do you define groups in TestNG?

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

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

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

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

```
<groups>
  <run>
    <exclude name = "demo">
    </exclude>
   </run>
</groups>
```

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

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

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

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

TestNG Asserts

18. What do you understand by asserts in TestNG?

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

```
Assert.Method(actual, expected, message);
```

Describe any five common TestNG assertions.

The five common TestNG assertions are:

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

Although it should be noted that there are a lot more assertions provided by TestNG.

19. What are the different types of assert in TestNG?

There are two types of assert in TestNG:

- *Soft Asserts*

- *Hard Asserts*

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

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

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

TestNG Dependent Tests

20. What is meant by dependency in TestNG?

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

21. How do you create dependencies in TestNG?

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

```
import org.testng.annotations.Test;
public class DependsOnTest {
  @Test (dependsOnMethods = { "OpenBrowser" })
  public void SignIn() {
          System.out.println("User has signed in successfully");
  }

  @Test
  public void OpenBrowser() {
          System.out.println("The browser is opened");
  }
}
```

Here, the SignIn method has been made dependent on the OpenBrowser method.

22. How do you create dependency through the XML file?

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

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="TestNG XML Dependency Suite" >
   <test name="ToolsQA" >
        <groups>
                <dependencies>
                        <group depends-on= "openbrowser" name=
"login"></group>
                </dependencies>
                </groups>
       <classes>
          <class name="GroupDependency" />
       </classes>
   </test>
 </suite>
```

Here, the login group depends upon the *openbrowser* group.

23. When do we use "dependsOnGroups" in TestNG?

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

```
import org.testng.annotations.Test;

public class GroupDependency
{
    @Test(dependsOnGroups = { "SignIn" })
    public void ViewAcc() {
        System.out.println("View Your Dashboardd");
    }

    @Test(groups = { "SignIn" })
    public void OpenBrowser() {
        System.out.println("Browser Opened Successfully");
    }

    @Test(groups = { "SignIn" })
    public void LogIn() {
        System.out.println("Login Into The Account");
    }
}
```

Miscellaneous

24. What is meant by parallel test execution in TestNG?

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

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

Parallel testing can apply at four different levels in TestNG:

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

26. How is exception handling done in TestNG?

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

```
@Test (expectedException = numberFormatException.class)
```

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

27. What is `@Factory` *annotation in TestNG?*

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

```
@Factory()
        public Object[] getTestClasses() {
                Object[] tests = new Object[2];
                tests[0] = new Test1();
                tests[1] = new Test2();
                return tests;
```

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

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

`@Factory` and `@Dataprovider` are two types of annotations available in TestNG, which look similar in their working but are different.
`@Factory`: *The use of the factory annotation is when the tester needs to execute the test methods multiple times, which are present in the same class. Additionally, we achieve this by creating different instances of the same class*.
`@Dataprovider`: *The dataprovider annotation enables the tester to run a test method multiple times using a different set of data provided by the dataprovider*.
Learn more about ***What are dataproviders and their usage in TestNG?***

TestNG Listeners

29. What are listeners in TestNG?

Listeners in TestNG are the piece of code that listens to certain events and execute the code associated with that event. As a result, with TestNG listeners, we can change the default behavior of ***TestNG***. Moreover, in TestNG, the tester gets the benefit of a lot of listeners who have different functionalities.

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

```
public class ListenersTestNG implements ITestListener {
        public void onStart(ITestContext context) {
                System.out.println("onStart method started");
        }
}
```
To apprise the TestNG test case file about the listener, we declare the `@Listener` annotation and mentioning the listener class name in the following manner:
```
@Listeners(ListenersTestNG.class)
public class TestNG {
        WebDriver driver = new FirefoxDriver();
        @Test  //Success Test
        public void CloseBrowser() {
                driver.close();
        }
}
```