***Types Of TestNG Annotations***

In TestNG, there are ten types of annotations:

* ***@BeforeSuite****– The @BeforeSuite method in TestNG runs before the execution of all other test methods.*
* ***@AfterSuite****– The @AfterSuite method in TestNG runs after the execution of all other test methods.*
* ***@BeforeTest****– The @BeforeTest method in TestNG runs before the execution of all the test methods that are inside that folder.*
* ***@AfterTest****– The @AfterTest method in TestNG executes after the execution of all the test methods that are inside that folder.*
* ***@BeforeClass****– The @BeforeClass method in TestNG will run before the first method invokes of the current class.*
* ***@AfterClass****– The @AfterClass method in TestNG will execute after all the test methods of the current class execute.*
* ***@BeforeMethod****– The @BeforeMethod method in TestNG will execute before each test method.*
* ***@AfterMethod****– The @AfterMethod method in TestNG will run after each test method is executed.*
* ***@BeforeGroups****– The @BeforeGroups method in TestNG run before the test cases of that group execute. It executes just once.*
* ***@AfterGroups****– The @AfterGroups method in TestNG run after the test cases of that group execute. It executes only once.*

IN TESTNG.XML

* ***<suite>****– The suite tag can be given any name and denotes the test suite name.*
* ***<test>****– The test tag can be given any name and indicates your test sets.*
* ***<classes>****– This is the combination of your package name and test case name and cannot write anything else.*

***How To Create Groups?***

Before getting more complicated in groups, let’s create a simple group test that we discussed in the above section. In the below code, we will check:

* *Whether we are getting the title of the webpage correctly or not.*
* *Find an element “Sortable” on the website and click the element.*

We will be using our demo website demoqa.com for this task. Refer to our tutorial of ***Selenium Webdriver*** to know in-depth about the code.

Write the following code inside your file named *TestNG.java* (You can choose any TestNG test case file you want).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.chrome.ChromeDriver;  import org.testng.annotations.Test;  import org.testng.Assert;    public class TestNG {  WebDriver driver;      // Saving the expected title of the Webpage      String title = " - Demo Website For Automation";          @Test      public void starting\_point(){       System.out.println("This is the starting point of the test");       //Initialize Chrome Driver       //driver.manage().timeouts().implicitlyWait(1,TimeUnit.SECONDS);       driver = new ChromeDriver();       driver.get("https://demoqa.com/");      }          @Test(groups = { "demo" })      public void checkTitle() {         String testTitle = "Free QA Automation Tools For Everyone";         String originalTitle = driver.getTitle();        Assert.assertEquals(originalTitle, testTitle);      }        @Test(groups = { "demo" })      public void click\_element() {         driver.findElement(By.xpath("//\*[@id=\"sidebar\"]/aside[1]/ul/li[1]/a")).click();          System.out.println("Home Page heading is displayed");      }  } |

In the above code, we have defined three tests and one method each under them. The following are the method names:

* ***starting\_point()****– It will initialize the Chromedriver and will enter the URL demoqa.com automatically.*
* ***check\_title()****– It will check whether the title of the web page is equal to the variable “t****est\_title****” or not. I have placed the wrong title, so this method should fail.*
* ***click\_element()****– This method will find an element called “****Sortable”****on the web page with the help of XPath and will click it.*

We have used the group name called “***demo***” but only on two methods, namely ***check\_title*** and ***click\_element***. Once we are done with the code, we need to tell our XML file about the groups (if we want to run according to the groups).

To include the name of the groups, create a ***TestNG XML*** like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | *<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >*  <suite name="Test-Suite" >     <test name="" >     <groups>     <run>     <include name = "demo"></include>     </run>     </groups>         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

We have included the name of the group “***demo***” inside our XML file within the ***“include”*** tag. By this, we expect to run only two tests because only two tests are included in the group “***demo.”***

Right-click on the file ***->******Run As – > TestNG Suite***

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

Till now, to run the groups, we have included them inside the “include” tag, which is quite obvious since we want to “include” them in our test run. Similar to this, we can ignore the groups by putting them under the “exclude” tag. This minor tweak can be seen in the XML file below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Test-Suite" >     <test name="" >     <groups>     <run>     <exclude name = "demo">     </exclude>     </run>     </groups>         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

By putting our group “demo” inside the exclude tag, we are requesting TestNG to ignore the test cases under the group “demo”. So, any guesses how many tests will run? Check the below output to see if you are right or wrong.

Only one test runs because it is the only test that was not inside the “demo” group.

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

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

* ***Using attributes dependsOnMethods in @Test annotations***

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class Dependent {      @Test (dependsOnMethods = { "OpenBrowser" })    public void SignIn() {    System.out.println("This will execute second (SignIn)");    }      @Test    public void OpenBrowser() {    System.out.println("This will execute first (Open Browser)");    }  } |

Here, our method *SignIn()* is dependent upon the method *OpenBrowser()*

* ***Using attributes dependsOnGroups in @Test annotations.***

The dependsOnGroups attribute lets us make a test depend on a whole group rather than a single test. For example, see the code below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class GroupDependency  {      @Test(dependsOnGroups = { "SignIn" })      public void ViewAcc() {          System.out.println("SignIn Successful");      }        @Test(groups = { "SignIn" })      public void LogIn() {          System.out.println("Logging In Success");      }  } |

Here, our test method, “***ViewAcc(),***” depends upon the group “***SignIn.”***

Refer ***TestNG Annotations*** to know more about the annotations, their benefits, and how they work in the hierarchy with different possible scenarios.

Let us see the different ways in which we can provide the dependency in TestNG.

***How To Run TestNG Parameters?***

Let’s say we want to add two numbers using the TestNG parameters. Observe the below-given code for the same.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class Params  {      @Test      @Parameters ({"val1", "val2"})      public void Sum(int v1, int v2) {       int finalsum = v1 + v2;          System.out.println("The final sum of the given values is " + finalsum);      }  } |

Can we run it directly using ***Rus As -> TestNG Test***? Try to run it like this. I am sure it won’t.

**Note:***TestNG Parameters are run through the TestNG XML file and not from the test case files directly.*

Head over to the XML file and write the following XML code in it.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="TestNG Parameters Suite">     <test name="Params">        <parameter name="val1" value="2" />        <parameter name="val2" value="3" />        <classes>           <class name="Params" />        </classes>     </test>  </suite> |

In the above XML file, we have defined a tag called parameters which work as follows:

* ***name:****Name of the variable that you declared in the test case file like val1 and val2 in the above example.*
* ***value:****The value of the variable you want to insert.*

***Note****: If you are confused with the suite and test element in the TestNG XML, I request you to refer to the previous tutorials of the****TestNG Test Suite****&****TestNG Test****.*

Run the XML file by ***Rus As -> TestNG*** Suite.

### *****How to give Priority in TestNG test?*****

The following is the syntax for allocating a priority to a test case method.

@Test (priority = 1)

public void func(){

//test code

}

import org.testng.annotations.Test;

import org.openqa.selenium.chrome.ChromeDriver;

public class TestNG {

WebDriver driver = new ChromeDriver();

@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:\\URL");

}

}

@Test (priority = -1)

public void OpenBrowser() {

Reporter.log("This test verifies the current selenium compatibility with TestNG by launching the chrome driver");

        Reporter.log("Launching Google Chrome Driver version 81 for this test");

        driver.get("https:\\URL");

        Reporter.log("The website used was DemoQA for this test", true);

        String expectedTitle = "Free QA Automation Tools For Everyone";

        String originalTitle = driver.getTitle();

        Assert.assertEquals(originalTitle, expectedTitle);

  }

}

## Different types of Asserts in TestNG

There are two types of TestNg Assert:

* ***Hard Assert***
* ***Soft Assert***

### *****Hard Assert in TestNG*****

***Hard Asserts*** are those asserts that stop the test execution when an assert statement fails, and the subsequent assert statements are therefore not validated. It plays a vital role in projects where we have an element without whose validation, asserting other elements is useless. One good example in such cases is the login functionality. If I want to see my past orders, for example, then what is the point of checking this test case when the login validation already failed? ***Hard asserts are the default type of asserts in TestNG,*** and what we used in the previous section was Hard Assert.

### *****Soft Assert in TestNG*****

Soft asserts are just the opposite of hard asserts. In soft asserts, the subsequent assertions keep on running even though one assert validation fails, i.e., the test execution does not stop. ***Soft assert does not include by default in TestNG***. For this, you need to include the package **org.testng.asserts.Softassert.**So, when should we use soft asserts in TestNG? We use soft asserts when we do not care about the failure of specific validations and want the test execution to proceed and also want to see the exception errors.

A good example is multiple validations on an input form. Also, to note that on many platforms, you will see “verify” while learning about asserts. ***Soft asserts are also known as***“Verify” and hence do not get confused about the same.

 softassert.assertEquals(originalTitle, expectedTitle);

        System.out.println("\*\*\* Checking For The Second Title \*\*\*");

// Checking title for – Demo Website to Practice Automation – Demo Website to Practice Automation

        softassert.assertEquals(originalTitle, " – Demo Website to Practice Automation – Demo Website to Practice Automation" );

        softassert.assertAll();

***How To Perform Cross-Browser Testing In TestNG Using Selenium?***

Cross-browser testing requires us to test our website using Selenium on multiple browsers, and as you might remember, if we want to pass different values to the same function, we use TestNG parameters for that. Without using the parameters, we are just writing the same repetitive code again and again. TestNG parameters will help us cut down on lines of code significantly and makes efficient use of TestNG annotations which is what every tester aims for while working on TestNG.

In the below code, we will be demonstrating how to perform cross-browser testing in TestNG using selenium web driver. But, before jumping onto the code, I highly recommend reading the ***TestNG parameters*** tutorial, ***TestNG Annotations*** tutorial, and our ***selenium tutorial***so that you adjust towards the flow of the code.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.openqa.selenium.chrome.\*;  import org.testng.annotations.AfterClass;  import org.testng.annotations.BeforeClass;  import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class MultiBrowser {    public WebDriver driver;    *@Parameters*("browser")    *@BeforeClass*      // Passing Browser parameter from TestNG xml      public void beforeTest(String browser) {      // If the browser is Firefox, then do this      if(browser.equalsIgnoreCase("firefox")) {    //Initializing the firefox driver (Gecko)    driver = new FirefoxDriver();      }else if (browser.equalsIgnoreCase("chrome")) {      //Initialize the chrome driver      driver = new ChromeDriver();      }      // Enter the website address in the browser      driver.get("https:\\URL");      }      // Once Before method is completed, Test method will start    *@Test* public void login() throws InterruptedException {    driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();    }    *@AfterClass* public void afterTest() {    driver.quit();    }    } |

Since we are using ***TestNG parameters***, we need to specify the values from the TestNG XML file that will pass to the test case file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Suite" parallel="none">  <test name="FirefoxTest">  <parameter name="browser" value="firefox" />  <classes>  <class name="MultiBrowser" />  </classes>  </test>  <test name="ChromeTest">  <parameter name="browser" value= "chrome" />  <classes>  <class name="MultiBrowser" />  </classes>  </test>  </suite> |

## ITestListener In TestNG

ITestListener is the most used listener in TestNG with***Selenium webdriver***. The ITestListener ***implements*** since it is an interface, and the class in which we implement the listener overrides the ITestListener methods. ITestListener listens to specific events (depending on its methods) and executes the code written inside the method.  With ITestListener in TestNG, we can also log the events onto the reports using the Selenium web driver. The ITestListener contains the following methods:

* **onStart:** This method invokes when the test class is instantiated and before executing any test method.

***Syntax: void onStart(ITestContext context);***

* ***onFinish***: This method invokes when all the test methods have run, and calling of all of their configuration methods happens.

***Syntax: void onFinish(ITestContext context);***

* ***onTestStart***:  This method invokes every time a test method is called and executed.

***Syntax: void onTestStart(ITestResult result);***

* **onTestSuccess:** This method is invoked every time a test case passes (succeeds).

***Syntax: void onTestSuccess(ITestResult result);***

* ***onTestFailure***: This method invokes every time a test case fails.

***Syntax: void onTestFailure(ITestResult result);***

* **onTestSkipped:** This method invokes every time a test skips.

***Syntax: void onTestSkipped (ITestResult result);***

* **onTestFailedButWithinSuccessPercentage:** This method invokes when the test method fails as a whole but has passed a certain success percentage, which is defined by the user.

***Syntax: void onTestFailedButSuccessPercentage (ITestResult result);***

***How to implement ITestListener in TestsNG?***

This section will explain step by step how to use listeners in TestNG to invoke various functions. It is important to note here that ***Listeners can implement in two ways in TestNG***:

* **At the class level***: Annotating listeners on each class in the test code.*
* **At the suite level:***Define the class names to implement listeners in the TestNG XML file.*

So first, we will go with implementing the listeners at the class level.

Interview Questions

Ques.1. What is TestNG?  
Ans. TestNG(NG for Next Generation) is a testing framework that can be integrated with selenium or any other automation tool to provide multiple capabilities like assertions, reporting, parallel test execution, etc.  
  
Ques.2. Provide some advantages of TestNG?  
Ans. Following are the advantages of TestNG-  
  
TestNG provides different assertions that help in checking the expected and actual results.  
It provides parallel execution of test methods.  
We can define the dependency of one test method over others in TestNG.  
Also, we can assign priority to test methods in Selenium.  
It permits the grouping of test methods into test groups.  
It allows data-driven testing using @DataProvider annotation.  
It gives inherent support for reporting.  
It has support for parameterizing test cases using @Parameters annotation.  
 Ques.3. How is TestNG different from Selenium WebDriver?  
Ans. Selenium is an automation tool using which we can automate web-based applications. In order to add testing capabilities in the Test Automation suites – Selenium is clubbed with TestNG. With TestNG, we can have different features in our automation suite like different types of assertions, reporting, parallel execution, and parameterization, etc.  
In short, for doing Automation Testing, Selenium helps with ‘automation’ and TestNG helps with ‘testing’ capabilities.  
  
Ques.4. What is the use of testng.xml file?  
Ans. The testng.xml file is used for configuring the whole test suite. In testng.xml file, we can create a test suite, create test groups, mark tests for parallel execution, add listeners, and pass parameters to test scripts. We can also use this testng.xml file for triggering the test suite from the command prompt/terminal or Jenkins.  
  
Ques.5. How can we group test cases like separate test cases for Sanity suite, Regression suite, etc?  
Ans. Using groups attribute in TestNG, we can assign the test methods to different groups.  
  
//Test method belonging to sanity suite only  
@Test(groups = {“sanitySuite”})  
public void testMethod1() {  
//Test logic  
}  
  
//Test method belonging to both sanity and regression suite  
@Test(groups = {“sanitySuite”, “regressionSuite”})  
public void testMethod2() {  
//Test logic  
}  
  
Ques.6. Can we exclude a Test method from getting executed via testng.xml file?  
Ans. Using the exclude tag in testng.xml file, we can exclude a particular test method from getting executed.  
  
Ques.7. State some commonly used TestNG annotations?  
Ans. The commonly used TestNG annotations are-  
  
@Test – @Test annotation marks a method as a test method.  
@BeforeSuite – The annotated method will run only once before all tests in this suite have run.  
@AfterSuite – The annotated method will run only once after all tests in this suite have run.  
@BeforeClass – The annotated method will run only once before the first test method in the current class is invoked.  
@AfterClass – The annotated method will run only once after all the test methods in the current class have been run.  
@BeforeTest – The annotated method will run before any test method belonging to the classes inside the tag is run.  
@AfterTest – The annotated method will run after all the test methods belonging to the classes inside the tag have run.  
@BeforeMethod – The annotated method will run before each test method marked by @Test annotation.  
@AfterMethod – The annotated method will run after each test method marked by @Test annotation.  
@DataProvider – The @DataProvider annotation is used to pass test data to the test method. The test method will run as per the number of rows of data passed via the data provider method.  
  
Ques.8. What is the order of execution of the test method based on the different annotations?  
Ans. The test methods in TestNG follow the Suite->Test->Class->Method sequence combined with the  
Before annotations->Test annotations->After annotations sequence. So, the order of execution is-  
@BeforeSuite  
@BeforeTest  
@BeforeClass  
@BeforeMethod  
@Test  
@AfterMethod  
@AfterClass  
@AfterTest  
@AfterSuite  
  
Ques.9. Explain common assertions provided by TestNG?  
Ans. Some of the common assertions provided by testNG are-  
  
assertEquals(String actual, String expected, String message) and other overloaded data types in parameter  
assertNotEquals(double data1, double data2, String message) and other overloaded data types in parameter  
assertFalse(boolean condition, String message)  
assertTrue(boolean condition, String message)  
assertNotNull(Object object)  
fail(boolean condition, String message)  
true(String message)  
  
Ques.10. How can we disable or prevent a test case from running?  
Ans. By setting the “enabled” attribute as false, we can disable a test method from running.  
  
//In case of a test method  
@Test(enabled = false)  
public void testMethod1() {  
//Test logic  
}  
  
//In case of test method belonging to a group  
@Test(groups = {“NegativeTests”}, enabled = false)  
public void testMethod2() {  
//Test logic  
}  
  
Ques.11. Can we make one test method dependent on others using TestNG?  
Ans. Using dependsOnMethods parameter inside @Test annotation in TestNG we can make one test method run only after the successful execution of the dependent test method.  
  
@Test(dependsOnMethods = { “preTests” })  
  
Ques.12. How can we set the priority of test cases in TestNG?  
Ans. We can define the priority of test cases using the “priority” parameter in @Test annotation. The tests with lower priority value will get executed first. Example-  
  
@Test(priority=1)  
  
Ques.13. What is the default priority of test cases in TestNG?  
Ans. The default priority of a test when not specified is integer value 0. So, if we have one test case with priority 1 and one without any priority then the test without any priority value will get executed first (as default value will be 0 and tests with lower priority are executed first).  
  
Ques.14. How can we run a Test method multiple times in a loop(without using any data provider)?  
Ans. Using invocationCount parameter and setting its value to an integer value, makes the test method to run n number of times in a loop.  
  
@Test(invocationCount = 10)  
public void invocationCountTest(){  
//Test logic  
}  
  
Ques.15. What is threadPoolSize? How can we use it?  
Ans. The threadPoolSize attribute specifies the number of threads to be assigned to the test method. This is used in conjunction with invocationCount attribute. The number of threads will get divided with the number of iterations of the test method specified in the invocationCount attribute.  
  
@Test(threadPoolSize = 5, invocationCount = 10)  
public void threadPoolTest(){  
//Test logic  
}  
  
Ques.16. Difference between soft assertion and hard assertion in TestNG?  
Ans. Soft assertions (SoftAssert) allows us to have multiple assertions within a test method, even when an assertion fails the test method continues with the remaining test execution. The result of all the assertions can be collated at the end using softAssert.assertAll() method.  
  
@Test  
public void softAssertionTest(){  
SoftAssert softAssert= new SoftAssert();  
  
//Assertion failing  
softAssert.fail();  
System.out.println(“Failing”);  
  
//Assertion passing  
softAssert.assertEquals(1, 1);  
System.out.println(“Passing”);  
  
//Collates test results and marks them pass or fail  
softAssert.assertAll();  
}  
Here, even though the first assertion fails still the test will continue with execution and print the message below the second assertion.  
Hard assertions on the other hand are the usual assertions provided by TestNG. In case of hard assertion in case of any failure, the test execution stops, preventing the execution of any further steps within the test method.  
  
Ques.17. How to fail a testNG test if it doesn’t get executed within a specified time?  
Ans. We can use the timeOut attribute of @Test annotation. The value assigned to this timeOut attribute will act as an upperbound, i  
f test doesn’t get executed within this time frame then it will fail with timeOut exception.  
  
@Test(timeOut = 1000)  
public void timeOutTest() throws InterruptedException {  
//Sleep for 2sec so that test will fail  
Thread.sleep(2000);  
System.out.println(“Will throw Timeout exception!”);  
}  
  
Ques.18. Can we skip a test case conditionally?  
Ans. Using SkipException, we can conditionally skip a test case. On throwing the skipException, the test method is marked as skipped in the test execution report and any statement after throwing the exception will not get executed.  
  
@Test  
public void testMethod(){  
if(conditionToCheckForSkippingTest)  
throw new SkipException(“Skipping the test”);  
//test logic  
}  
  
Ques.19. How can we make sure a test method runs even if the test methods or groups on which it depends fail or get skipped?  
Ans. Using “alwaysRun” attribute of @Test annotation, we can make sure the test method will run even if the test methods or groups on which it depends fail or get skipped.  
  
@Test  
public void parentTest() {  
Assert.fail(“Failed test”);  
}  
  
@Test(dependsOnMethods={“parentTest”}, alwaysRun=true)  
public void dependentTest() {  
System.out.println(“Running even if parent test failed”);  
}  
Here, even though the parentTest failed, the dependentTest will not get skipped instead it will executed because of “alwaysRun=true”. In case, we remove the “alwaysRun=true” attribute from @Test then the report will show one failure and one skipped test, without trying to run the dependentTest method.  
  
Ques.20. Describe how can we pass the parameter to test script using TestNG?  
Ans. Using @Parameter annotation and ‘parameter’ tag in testng.xml we can pass parameters to test scripts.  
Sample testng.xml –  
  
Sample test script-  
  
public class TestFile {  
@Test  
@Parameters(“sampleParamName”)  
public void parameterTest(String paramValue) {  
System.out.println(“Value of sampleParamName is – ” + sampleParamName);  
}  
  
Ques.21. Explain how can we create a data-driven framework using TestNG?  
Ans. Using @DataProvider we can create a data-driven framework in which data is passed to the associated test method and multiple iterations of the test run for the different test data values passed from the @DataProvider method. The method annotated with @DataProvider annotation return a 2D array of object.  
  
//Data provider returning 2D array of 3\*2 matrix  
@DataProvider(name = “dataProvider1”)  
public Object[][] dataProviderMethod1() {  
return new Object[][] {{“kuldeep”,”rana”}, {“k1″,”r1”},{“k2″,”r2”}};  
}  
  
//This method is bound to the above data provider returning 2D array of 3\*2 matrix  
//The test case will run 3 times with different set of values  
@Test(dataProvider = “dataProvider1”)  
public void sampleTest(String s1, String s2) {  
System.out.println(s1 + ” ” + s2);  
}  
  
Ques.22. Use of @Listener annotation in TestNG?  
Ans. TestNG provides us different kinds of listeners using which we can perform some action in case an event gets triggered. Usually, testNG listeners are used for configuring reports and logging. One of the most widely used listeners in testNG is ITestListener interface. It has methods like onTestSuccess, onTestFailure, onTestSkipped etc. We need to implement this interface creating a listener class of our own. After that using the @Listener annotation we can use specify that for a particular test class our customized listener class should be used.  
  
@Listeners(PackageName.CustomizedListenerClassName.class)  
  
public class TestClass {  
WebDriver driver= new FirefoxDriver();@Test  
public void testMethod(){  
//test logic  
}  
}  
  
Ques.23. What is the use of @Factory annotation in TestNG?  
Ans. @Factory annotation helps in the dynamic execution of test cases. Using @Factory annotation we can pass parameters to the whole test class at run time. The parameters pas  
sed can be used by one or more test methods of that class.  
Example – there are two classes TestClass and the TestFactory class. Because of the @Factory annotation, the test methods in class TestClass will run twice with the data “k1” and “k2”  
  
public class TestClass{  
private String str;  
  
//Constructor  
public TestClass(String str) {  
this.str = str;  
}  
  
@Test  
public void TestMethod() {  
System.out.println(str);  
}  
}  
  
public class TestFactory{  
//The test methods in class TestClass will run twice with data “k1” and “k2”  
@Factory  
public Object[] factoryMethod() {  
return new Object[] { new TestClass(“K1”), new TestClass(“k2”) };  
}  
}  
  
Ques.24. Difference between @Factory and @DataProvider annotation?  
Ans. @Factory method creates instances of test class and run all the test methods in that class with a different set of data.  
Whereas, @DataProvider is bound to individual test methods and run the specific methods multiple times.  
  
Ques.25. How can we run test cases in parallel using TestNG?  
Ans. In order to run the tests in parallel just add these two key-value pairs inside the suite tag of the testng.xml file-  
  
parallel=”{methods/tests/classes}”  
thread-count=”{number of thread you want to run simultaneously}”.  
  
 IMPORTANT POINTS ON TESTNG  
1. We can run the test case with the help of TestNG annotation without using the java compiler.  
  
2. Everything that you want to execute should be inside a method. We can then execute the test with the @test annotation followed by the method.  
  
3. Using TestNG we can define multiple test cases in a single class.  
  
4. Using the TestNG.XML file we can run the complete suite with a single click.  
  
5. Structure of TestNG.XML file is: TEST SUITE >> TEST FOLDERS(MODULES) >> TEST CASES  
  
6. In the XML file we can write the script to exclude any method from the class by using exclude tag.  
  
7. Also, if we can use include a tag in the XML file,  we can include the specific method to execute.  
  
8. Naming convention plays a very important role, which helps us to control the execution very easily using regular expression we can exclude/include n number of test cases. For example, if you want to exclude all the test cases which start with the word mobile then in the XML file inside the method tag we will use  
  
9. @BeforeTest and @AfterTest annotation are at the test folder level.  
  
10. @BeforeSuit and @afterSuit have scope to the entire XML file.  
  
11. @BeforeMethod has the scope at the class level and it executes before every method of that class and @AfterMethod same.  
  
12. @BeforeClass and @AfterClass have scope to the class level which executes before and after executing any method of that class.  
  
13. Order of Execution of @Test annotation is alphabetic as per method name  
  
14. Suppose you have to run a few test cases from a package having different classes, there is a concept of groups in the testNG. We will tag the test case and then will define the groups in the XML file.  
  
15. Using dependsOnMethods attribute we can control the flow of execution of a test case.  
  
16. If you want to exclude a test case you can use the annotation like @Test(enable=false)  
  
17. We can set time out also for a test case like @Test(timeOut=4000)  
  
18. Using parameterization you can define data in the XML file and then use it in the class for the method.  
  
19. In the XML file, you can define the parameter at suit level and at folder/test level  
  
20. How to run a test case with multiple sets of data: for that we will use @DataProvider annotation above the data provider method  
  
21. How will you use parametrization in TestNG, there will be 2 ways, 1st defining the data in the XML file or using data provider annotation in the class  
  
@Parameters({“URL”,”Username“})  
  
@Test(groups=”Smoke”)