

TestNG IQ:

1. What is TestNG framework? What is the advantage we get using TestNG framework? Difference between TestNG and Junit? What annotations are there in TestNG that is not in Junit?

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing to integration testing.

The advantages of TestNG are as follows:

- TestNG is an open source framework that help us create test cases in a systematic way.
- TestNG gives lots of annotations which makes the test case creation easy.
- Using TestNG, priorities of the tests and the sequence of execution can be defined.
- Grouping is possible using TestNG.
- TestNG generates HTML reports (Selenium Webdriver cannot generate the test reports alone, TestNG helps SW to achieve this).
- Data parameterization is possible using TestNG.
- TestNG was built on top of Junit

TestNG vs Junit:

- TestNG supports testing in groups but it is not supported in JUnit.
- TestNG has a feature to configure dependency test. Dependency test configuration for software web application is not possible in JUnit.
- Test prioritization, Parallel testing is possible in TestNG. It is not supported by JUnit.
- TestNG compare to Junit provides additional annotations such as:
 - @BeforeSuite
 - @BeforeTest
 - @BeforeGroups
 - @AfterGroups
 - @Aftertest
 - @AfterSuite

2. Explain TestNG annotations? What annotations u have worked on? What is the difference between before method and before test?

Describe all/few **pre and post condition annotation!**

BeforeMethod vs BeforeTest

- **@BeforeMethod** - The annotated method will be run after all the test methods in the current class have been run. On the other hand, @BeforeMethod will be executed just before any function/method with @Test annotation starts.
- **@BeforeTest** - The annotated method will be run before any test method belonging to the classes inside the <test> tag is run. To execute a set-up method before any of the test methods included in the < test > tag in the testng.xml file.

3. How to set test case priority in TestNG? How To Make A Test Dependent On Another? How to run a group of test cases using TestNG? How to disable a test case in TestNG ? How to skip/exclude/block particular Test method/Class from execution in TestNG?

Test Case Priority

We use priority attribute to the @Test annotations. In case priority is not set then the test scripts execute in alphabetical order.

```
@Test
public void firstTest() {
    System.out.println("First Test");
}

@Test(priority = 1) // Assigning the Priority
public void secondTest() {
    System.out.println("Second Test");
}
```

Test Case Dependency

We can enforce TestNG's dependency feature using “dependsOnMethods” attribute to declare dependencies of a test method.

```
@Test
```

```
public void stepOne() {  
    System.out.println("Executing stepOne");  
}  
  
@Test(dependsOnMethods = { "stepOne" })  
public void stepTwo() {  
    System.out.println("Executing stepOne->stepTwo");  
}  
  
@Test(dependsOnMethods = { "stepOne", "stepTwo" })  
public void stepThree() {  
    System.out.println("Executing stepOne->stepTwo->stepThree");  
}
```

Group Test Cases

TestNG allows you to perform groupings of test methods using “groups”

```
@Test(groups= {"Smoke"})  
public void helloTest1() {  
    System.out.println("This test belongs to group 1");  
}  
  
@Test(groups= {"Regression"})  
public void helloTest2() {  
    System.out.println("This test belongs to group 2");  
}
```

Disable a Test Case

To disable the test case we use the parameter `enabled = false` to the `@Test` annotation.

```
@Test(enabled = false)
```

Skip/exclude Test Case

Using Exclude tag in our testng.xml

```
<suite name="Sample Test Suite" >
  <test name="Exclude Test" >
    <classes>
      <class name="AddTestCase">
        <methods>
          <include name="addDepartment" />
          <exclude name="addEmployee" />
        </methods>
      </class>
    </classes>
  </test>
</suite>
```

4. What is suite file? What is the importance of testng.xml file? How to create and run testng.xml?

In a Selenium TestNG project, we use testng.xml file to configure the complete test suite in a single file.

Importance of testng.xml file:

- Allows to include or exclude the execution of test methods and test groups
- Allows to pass parameters to the test cases
- Allows to add group dependencies
- Allows to add priorities to the test cases
- Allows to configure parallel execution of test cases

Create and run testng.xml file:

In TestNG framework, we need to create testng.xml file to create and handle multiple test classes. We do configure our test run, set test dependency, include or exclude any test, method, class or package and set priority etc in the xml file.

5. How to execute the same test multiple times?

Use invocationCount with @Test method itself

@Test(invocationCount=10) // This method will be executed 10 times

6. How to run test cases in parallel using TestNG?

Use “parallel” attribute in testng.xml to accomplish parallel test execution in TestNG

tests – All the test cases inside <test> tag of testng.xml file will run parallel

classes – All the test cases inside a java class will run parallel

methods – All the methods with @Test annotation will execute parallel

<suite name="SmokeSuite" parallel="methods">

7. How do you pass values from xml file and execute test using them? How will you control data using XML? Difference between Parameters & Parameter.

TestNG allows user to pass values to test methods as arguments by using parameter annotations through testng.xml file.

@Parameters annotation can be placed on any method that has a @Test, @Before/After annotation.

The XML parameters are mapped to the Java parameters in the same order as they are found in the annotation, and TestNG will issue an error if the numbers don't match.

In testng.xml, parameter values can be set at both suite and test level. If we have two parameters with the same name, the one defined in will have the precedence.

Providing Parameters within the class/methods

```
public class TestParameters {  
  
    @Parameters({ "browser" })  
    @Test  
    public void testCaseOne(String browser) {  
        System.out.println("browser passed as :- " + browser);  
    }  
  
    @Parameters({ "username", "password" })  
    @Test  
    public void testCaseTwo(String username, String password) {  
        System.out.println("User Name is " + username);  
    }  
}
```

```
        System.out.println("Parameter is " + password);
    }
}
```

Providing Parameters in xml file

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Parameterization Test Suite">
    <test name="Testing Parameterization">
        <parameter name="browser" value="Firefox"/>
        <parameter name="username" value="testuser"/>
        <parameter name="password" value="testpassword"/>
        <classes>
            <class name="com.parameterization.TestParameters" />
        </classes>
    </test>
</suite>
```

8. How can we create data driven framework using TestNG? What will be the logic behind fetching data from data provider and inserting it on UI?

Data Driven framework is focused on separating the test scripts logic and the test data from each other and allows us to create test automation scripts by passing different sets of test data. The test data set is kept in the external files or resources such as MS Excel Sheets, MS Access Tables, SQL Database, XML files etc.

Data-driven concept is achieved by `@DataProvider` annotation in TestNG. The `@Test` method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.

If you want to provide the test data, the DataProvider way, then we need to declare a method that returns the data set in the form of two dimensional object array `Object[][]`.

Simple DataProvider Scenario:

```
@Test(dataProvider="getData")
public void test1(String UID, String PWD) {
    System.out.println("testing using "+ UID+" and "+PWD);
}
```

```
@DataProvider
public Object[][] getData() {
    Object[][] data=new Object[3][2];

    data[0][0]="UID 1";
    data[0][1]="PWD 1";
    data[1][0]="UID 2";
    data[1][1]="PWD 2";
    data[2][0]="UID 3";
    data[2][1]="PWD 3";
    return data;
}
```

DataProvider Scenario with Excel:

Functions to read Excel:

```
public void openExcel(String xlPath, String sheetName) {
    try {
        fis = new FileInputStream(xlPath);
        wb = new XSSFWorkbook(fis);
        sheet = wb.getSheet(sheetName);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String getExcelData(int rowNum, int cellNum) {
    String
    cellValue=sheet.getRow(rowNum).getCell(cellNum).toString();
    return cellValue;
}

public int rowCount() {
    int rowNum=sheet.getLastRowNum();
}
```

```
        return rowNum;  
    }  
  
    public int columnCount() {  
        return sheet.getRow(0).getLastCellNum();  
    }
```

Creating DataProvider

```
@DataProvider(name="Employee Details")  
public Object[][] getData(){  
    /*Open excel, get count of rows and columns  
    * loop through all rows and columns and get value of each cell  
    * value from each cell will store into Object[][] */  
  
    ExcelUtility obj=new ExcelUtility();  
    obj.openExcel(Constants.XL_FILEPATH, "EmployeeDetails");  
  
    //retrieve all values from excel and store 2D Object array  
    int row=obj.getRowNum();  
    int cell=obj.getColNum(0);  
  
    Object[][] data=new Object[row][cell];  
    //loop through each row & column  
    for (int i=1; i<row; i++) {  
        for (int y=0; y<cell; y++) {  
            //retrieve value from excel  
            String value=obj.getCellData(i, y);  
            //store inside 2D array  
            data[i][y]=value;  
        }  
    }  
    return data;  
}
```

Calling DataProvider in Specific Test

```
@Test(dataProvider="Employee Details")
```



```
public void addingEmployee(String fName, String lName, String uName, String
pwd) {

    AddEmployee employee = new AddEmployee();
    CommonMethods.enterValue(employee.firstName, fName);
    CommonMethods.enterValue(employee.lastName, lName);
    CommonMethods.click(employee.chxLoginDetails);
    CommonMethods.enterValue(employee.userName, uName);
    CommonMethods.enterValue(employee.password, pwd);
    CommonMethods.enterValue(employee.confPwd, pwd);
    CommonMethods.click(employee.btnSave);
}
```

9. How do you run multiple TestNG Suite Files?

We have to configure our testng.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="MultipleSuite">
  <suite-files>
    <suite-file path="regression.xml"></suite-file>
    <suite-file path="smoke.xml"></suite-file>
    <suite-file path="sanity.xml"></suite-file>
  </suite-files>
</suite>
```

10. What is TestNG Listener? How did you use it? How to get screenshot on failure?

TestNG listener mainly used to generate the report for the test. Also, you can capture screenshot when there is a test failure. TestNG events are like on Test Failure, onTestSkipped, onTestSuccess, etc.

Using ITestListener interface and method onTestFailure we can get a screenshot of our failed test case.

//create method in common methods

```
public static String takeScreenShot(String fileName) {
    TakesScreenshot ts=(TakesScreenshot)driver;
    File src=ts.getScreenshotAs(OutputType.FILE);

    dest=System.getProperty("user.dir")+"/target/screenshots/"+fileName+".png";
        try {
            FileUtils.copyFile(src, new File (dest));
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Unable to take screenshot");
        }
        return dest;
    }
```

//call method in Listener class

```
public class Listeners extends BaseClass implements ITestListener {

    @Override
    public void onTestFailure(ITestResult result) {
        System.out.println("Failed:" + result.getName());
        logger.fail("Failed:" + result.getName());
        String screen = CommonMethods.takeScreenShot(result.getName() + "");
        System.out.println(screen);

        try {
            logger.addScreenCaptureFromPath(screen);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

//specify Listeners in .xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
```

```
<suite name="Smoke Suite">

    <groups>
        <run>
            <include name="smoke"></include>
        </run>
    </groups>

    <listeners>
        <listener class-name="com.syntax.utils.Listeners" />
    </listeners>

    <test thread-count="5" name="Smoke Test">
        <packages>
            <package name="com.syntax.testcases" />
        </packages>

    </test> <!-- Test -->
</suite> <!-- Suite -->
```

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

TestNG library brings a very convenient reporting feature. Once we execute our tests, TestNG generates a test output folder at the root of the project. It combines two kinds of reports.

Detailed Report - You can find this report in the <index.html> file. It combines the detailed information like the errors, test groups, execution time, step-by-step logs and TestNG XML file.

Summary Report- It is the trimmed version and informs about the no. of “Passed”/”Failed”/”Skipped” cases. You can see it from the <emailable-report.html> file. It’s an email friendly report which you can embed and share with the stakeholders.

Listener- is defined as interface that modifies the default TestNG 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.

ExtendReports - Extent Report is an HTML reporting library for Selenium WebDriver for Java. It is a great tool we can use within our automation framework to make excellent execution reports.

```
public static ExtentHtmlReporter htmlReport;  
public static ExtentReports report;  
public static ExtentTest test;
```

```
@BeforeTest(alwaysRun=true)  
public void generateReport() {  
    ConfigsReader.readProperties(Constants.CREDENTIALS_FILEPATH);  
    //create an object of extentReport and htmlReporter  
    htmlReport=new ExtentHtmlReporter(Constants.REPORT_FILEPATH);  
    report = new ExtentReports();  
    report.attachReporter(htmlReport);  
    //provide some info (optional)  
    report.setSystemInfo("OS", Constants.OS_NAME);  
    report.setSystemInfo("Environment", "Test");  
    report.setSystemInfo("Browser", ConfigsReader.getProperty("browser"));  
    report.setSystemInfo("QA Engineer", Constants.USER_NAME);  
}  
  
@AfterTest (alwaysRun = true)  
public void flushReport() {  
    report.flush();  
}
```

12. Suppose there are class A and Class B and class B extends class A

Class A	Class B
@beforemethod	@beforemethod
@aftermethod	@aftermethod
@beforeclass	@beforeclass
@afterclass	@afterclass
@beforetest	@beforetest
@aftertest	@aftertest



What is the order of execution of testng annotations in above case?