(<http://testng.org/doc/documentation-main.html#annotations>)

(<http://www.techbeamers.com/testng-framework-interview-questions-answers/>)

**1) what is the use of framework in automation testing**

Ans: Framework is nothing but it is a separate project which contains packages/classes/methods. Basically framework contains reusable components/code

Ex: In framework, I created classes/methods to handle exceptions/custom reports, reading data from database, excel file.

It will be a jar file in java

It will be DLL in C#

In automation code we add this jar/dll’s (using buildpath> add external jar files option) and call appropriate classes/method Whenever it is required.

Ex: selenium jar file is framework and we add this file jar file in our project

We use classes/methods of selenium framework (WebDriver, Webelement classes, get

method)

**2) what are the things to consider/keep in framework?**

Ans: Create classes/methods to handle exceptions/custom reports, reading data from

database, excel file

**3) different types of frameworks we have in selenium/qtp?**

Ans:

**Data driven framework:** When some test needs to repeat for different data set, Data driven framework gets used. In this framework, Parameters in the test case gets linked to database, excel, csv, text files from there test case run for all defined parameter in the file.

Data-driven testing is creation of test scripts where test data and/or output values are read from data files instead of using the same hard-coded values each time the test runs. This way testers can test how the application handles various inputs effectively. It can be any of the below data files.

**Keyword driven framework**: As Name suggest, Keyword nothing but a code which represent some action, say “login”. in this framework, we map the set of code which perform certain action with a keyword and then we use that keyword across the framework.

Keyword driven testing is a type of functional automation testing framework which also known as table-driven testing or action word based testing.

In Keyword-driven testing we use a table format, usually a spreadsheet, to define keywords or action words for each function that we would like to execute.

**Hybrid framework**: It is nothing but mix of any frameworks together. most popular hybrid automation frameworks are Modular- Data driven and Keyword- data driven.

**Linear automation framework:** linear framework mainly based on record and play and follow the procedural code. this framework especially suites for very small projects and for creation of smoke test suites where only basis tests being executed.

**Structured automation Framework:**In Structured framework**,** test cases are writing in more structured way using loops, if else statements, Switch statement and conditional statement, but  it does not have any functions or modularity to make the framework more flexible.

**Modular automation Framework:**In Modular framework, reusable code put in some functions and functions getting called whenever needed. it makes framework more flexible and easy for maintenance.

**BDD:** **behavior-driven development** is a software development process based on test-driven development (TDD). Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software development and management teams with shared tools and a shared process to collaborate on software development.

**4) what is data driven frame work and when do we use it?**

Ans: When some test needs to repeat for different data set, Data driven framework gets used. In this framework, Parameters in the test case gets linked to database, excel, csv, text files from there test case run for all defined parameter in the file.

Data-driven testing is creation of test scripts where test data and/or output values are read from data files instead of using the same hard-coded values each time the test runs. This way testers can test how the application handles various inputs effectively. It can be any of the below data files.

It is used to test same functionality more than once with different set of data/input data

We store the test data in external files (excel/csv) or table.

**5) what is keyword frame work and when do we use it?**

As Name suggest, Keyword nothing but a code which represent some action, say “login”. in this framework, we map the set of code which perform certain action with a keyword and then we use that keyword across the framework.

Keyword driven testing is a type of functional automation testing framework which also known as table-driven testing or action word based testing.

In Keyword-driven testing we use a table format, usually a spreadsheet, to define keywords or action words for each function that we would like to execute.

The basic working of the Keyword Driven Framework is to divide the Test Case in to four different parts. First is called as Test Step, second is Object of Test Step, third is Action on Test Object and fourth is Data for Test Object.

The above categorization can be done and maintained with the help of Excel spread sheet:

**Test Step**: It is a very small description of the Test Step or the description of the Action going to perform on Test Object.

**Test Object:** It is the name of the Web Page object/element, like Username & Password.

Action: It is the name of the action, which is going to perform on any Object such as click, open browser, input etc.

**Test Data:** Data can be any value which is needed by the Object to perform any action, like Username value for Username field.

The idea behind the *Keyword Driven* approach in automation is to separate the coding from the test case & test step. This helps a non-technical person to understand the automation very well. With this a manual tester can write Automation scripts. This does not mean that an automation tester is not needed but in any Keyword Driven project, a hard core technical coder is must needed for setting up the framework and to work on regular changes and updates of background automation code. But for an example an automation team can have two manual testers and a automation coder.

**6) what is hybrid framework and when do we use it?**

“A hybrid framework is a framework that is created by combining together the features of the different types of QTP frameworks.”

“Hybrid framework is the most commonly used framework in test automation projects.”

**7) what is POM and modular framework?**

***Page Object***: This is a very popular pattern used these days for Test Automation Frameworks. If you think of a web based application as a network of pages that interact with each other AND there are large number of pages in your application that can change nature frequently, this pattern is the most favorable one to use.

Page-objects (or classes) are defined for each page, where a page contains the html locator information and also the methods that can operate on those page elements. Though we say page-objects (aka. classes) are written for each page, there are some nuances in terms of how we model a Web page into a page-object. A network of page-objects is supposed to model the state of the web application ultimately, however with asynchronous web applications, where parts of page get refreshed (as opposed to full web page. aka. DOM), we have to handle ajax calls based on the javascript library. Anyways, will mention the caveats, pros and cons, situations in which web application (page html source) to page-object modeling becomes the most important part in your framework etc, will be dealt in a separate post at the end of frameworks tutorial.

The advantage of this pattern from the decision-making point is:

* If a html locator information changes for an element, changing it in the page-object reflects across all the scripts – nice right
* Gives us ability to define keywords too as we can define methods inside page-objects
* We can define navigation logic between the pages too as a way to tie the links between the pages

**8) What is testng and junit?**

**TestNG:** TestNG is a testing framework developed in the lines of JUnit and NUnit, however it introduces some new functionalities that make it more powerful and easier to use.

TestNG is designed to cover all categories of tests − unit, functional, end-to-end, integration, etc., and it requires JDK 5 or higher.

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

**9) Writing unit test cases with testng?**

Writing a test in TestNG basically involves the following steps:

* Write the business logic of your test and insert TestNG annotations in your code.
* Add the information about your test (e.g. the class name, the groups you wish to run, etc.) in a testng.xml file or in build.xml.
* Run TestNG.

There are many benefits of using the annotations in the project.

* They are easier to understand.
* You can group the test cases using the appropriate annotation.
* You can do parallel testing.
* You can pass extra parameters to annotations.
* They are strongly typed, so the compiler will catch any error outright.

Steps for executing test case in TestNG:

1) Press **Ctrl+N** , select “**TestNG Class**” under **TestNG**category and click **Next**.

**Or**

Right click on Test Case folder, go to **TestNG**and select “**TestNG Class**“.

2) If the project is set up and we have selected the Test Case folder before creating TestNG class then the source folder and the package name will be prepopullated on the form. Set class name as ‘**TestNG**‘.

Under Annotations, check “**@BeforeMethod**”, “**@AfterMethod**” and click **Finish**. That’s it.

3)Now it will display the newly created TestNg class under the Test Case package(folder). TestNG class will look like the image below with displaying three empty methods. One method f() by default and before & after method, as selected during the creation of the class.

4)Run the test by right click on the test case script and select **Run As** > **TestNG Test**.

TestNG also produce HTML reports. To access those reports go to the**Project** directory and open **test-output** folder.

Open ‘**emailable-report.html**‘, as this is a html report open it with browser.

TestNG also produce ‘**index.html**‘ report and it resides in the same **test-output** folder. This reports gives the link to all the different component of the TestNG reports like **Groups** & **Reporter Output**. On clicking these will display detailed descriptions of execution.

**10) what are important elements in testng.xml?**

In a Selenium TestNG project, we use <testng.xml> file to configure the complete test suite into a single file. This file makes it easy to group all the test suites and their parameters in one file. It also gives the ability to pull out subsets of your tests or split several runtime configurations. Few of the tasks which we can group in the <testng.xml> file are as follows.

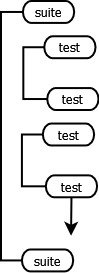
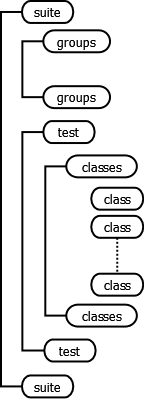
**1-** Can configure test suite comprising of multiple test cases to run from a single place.  
**2-** Can include or exclude test methods test execution.  
**3-** Can mark a group to include or exclude.  
**4-** Can pass parameters in test cases.  
**5-** Can add group dependencies.  
**6-** Can configure parallel test execution.  
**7-** Can add listeners.

testng.xml is the main configuration file that defines the suite and tests. suite is top level element in TestNG configuration file and is defined by one XML file.

If you need to have more suites then you need to define separate testng.xml file for each suite like database\_testng.xml, feature\_testng.xml, performance\_testng.xml etc.

**suite** is the first element of testng.xml. A suite contains one or more test elements.

A test is made of one or more classes and a clas is made or one or more methods.

**11) what are different annotations in testng and junit?**

**(**<http://everythingaboutselenium.blogspot.com/2013/07/all-testng-annotations-with-attributes.html>**)**

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| @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. |
| @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. |
| @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. |
| @BeforeGroups | The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked. |
| @AfterGroups | The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked. |
| @BeforeMethod | The annotated method will be run before each test method. |
| @AfterMethod | The annotated method will be run after each test method. |
| @DataProvider | Marks a method as supplying data for a test method. The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation. |
| @Factory | Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ]. |
| @Listeners | Defines listeners on a test class. |
| @Parameters | Describes how to pass parameters to a @Test method. |
| @Test | Marks a class or a method as a part of the test. |

**Benefits of TestNG:**

* TestNG identifies the methods it is interested in, by looking up annotations. Hence, method names are not restricted to any pattern or format.
* We can pass additional parameters to annotations.
* Annotations are strongly typed, so the compiler will flag any mistakes right away.
* Test classes no longer need to extend anything (such as TestCase, for JUnit 3).

**JUnit:**

**(**<https://www.java2novice.com/junit-examples/junit-annotations/>**)**

The Junit annotations are:

1. @Test
2. @Before
3. @After
4. @BeforeClass
5. @AfterClass
6. @Test (expected = Exception.class)
7. @Test(timeout=100)
8. @Ignore

**@Test:** The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case. To run the method, JUnit first constructs a fresh instance of the class then invokes the annotated method. Any exceptions thrown by the test will be reported by JUnit as a failure. If no exceptions are thrown, the test is assumed to have succeeded.

**@Before:** When writing tests, it is common to find that several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method to be run before the Test method. The @Before methods of super classes will be run before those of the current class.

**@After:**  If you allocate external resources in a Before method you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method. All @After methods are guaranteed to run even if a Before or Test method throws an exception. The @After methods declared in superclasses will be run after those of the current class.

**@BeforeClass:** Sometimes several tests need to share computationally expensive setup (like logging into a database). While this can compromise the independence of tests, sometimes it is a necessary optimization. Annotating a public static void no-arg method with @BeforeClass causes it to be run once before any of the test methods in the class. The @BeforeClass methods of superclasses will be run before those the current class.

The annotations @BeforeClass and @Before are same in functionality. The only difference is the method annotated with @BeforeClass will be called once per test class based, and the method annotated with @Before will be called once per test based.

**@AfterClass:** If you allocate expensive external resources in a BeforeClass method you need to release them after all the tests in the class have run. Annotating a public static void method with @AfterClass causes that method to be run after all the tests in the class have been run. All @AfterClass methods are guaranteed to run even if a BeforeClass method throws an exception. The @AfterClass methods declared in superclasses will be run after those of the current class.

The annotations @AfterClass and @After are same in functionality. The only difference is the method annotated with @AfterClass will be called once per test class based, and the method annotated with @After will be called once per test based.

**@Ignore**: Sometimes you want to temporarily disable a test or a group of tests. Methods annotated with Test that are also annotated with @Ignore will not be executed as tests. Also, you can annotate a class containing test methods with @Ignore and none of the containing tests will be executed. Native JUnit 4 test runners should report the number of ignored tests along with the number of tests that ran and the number of tests that failed.

**@Test (expected = Exception.class):** Sometimes we need to test the exception to be thrown by the test. @Test annotation provides a parameter called 'expected', declares that a test method should throw an exception. If it doesn't throw an exception or if it throws a different exception than the one declared, the test fails.

**@Test(timeout=100):** Sometimes we need to measure the performance in terms of time. The @Test annotations provides an optional parameter called 'timeout', which causes a test to fail if it takes longer than a specified amount of clock time (measured in milliseconds).

**12) what is group and suite and parallel execution in testNG?**

**Group Test:** Group test is a new innovative feature in TestNG, which doesn’t exist in JUnit framework. It permits you to dispatch methods into proper portions and perform sophisticated groupings of test methods. Not only can you declare those methods that 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 (or regular expressions), while excluding another set. Group tests provide 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 using the <groups> tag. It can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.

**Suite Test:** “Suite Test” is done when you have to run few unit test together, “Suite Test” bundle this unit test together. XML file is used to run the suite test.

A **test suite** is a collection of test cases intended to test a behavior or a set of behaviors of software program. In TestNG, we cannot define a suite in testing source code, but it is represented by one XML file, as suite is the feature of execution. It also allows flexible configuration of the *tests* to be run. A suite can contain one or more tests and is defined by the <suite> tag.

<suite> is the root tag of your testng.xml. It describes a test suite, which in turn is made of several <test> sections.

The following table lists all the legal attributes that <suite> accepts.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| name | The name of this suite. It is a **mandatory** attribute. |
| verbose | The level or verbosity for this run. |
| parallel | Whether TestNG should run different threads to run this suite. |
| thread-count | The number of threads to use, if parallel mode is enabled (ignored other-wise). |
| annotations | The type of annotations you are using in your tests. |
| time-out | The default timeout that will be used on all the test methods found in this test. |

**Parallel Testing:**

**(**<http://www.guru99.com/parallel-testing.html>**)**

Parallel testing is a testing of multiple applications or subcomponents of one application concurrently to reduce the test time.

When any organization is moving from old system to new system, legacy data is an important part. Transferring this data is a complex process.

In software testing, verifying compatibility of the newly developed system with the old system is done through "parallel testing.

In parallel testing, tester runs two different versions of software concurrently with same input. The aim to find out whether the legacy system and the new system are behaving same or differently. It ensures that the new system is capable enough to run the software efficiently.

**13) how to decide which test cases needs to be automated?**

It is impossible to automate all testing, so it is important to determine what test cases should be automated first.

The benefit of automated testing is linked to how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are ones that are run frequently and require large amounts of data to perform the same action.

Possible ranking criteria include the following:   
- ask the end-user to prioritize the requirements (and test those first);  
- test first the areas of the software that have had most faults in the past;  
- test most those areas of the software that are most complex or critical.  
- tests that would find the most severe failures;  
- tests that would find the most visible failures;  
- tests that would find the most likely faults;

Prioritization in TestNG:

We can use “priority” keyword for it.

**Priority**: The priority for this test method. Lower priorities will be scheduled first.

We use ‘**priority**‘ parameter. **Parameters** are keywords that can modify the annotation’s default function. Priority starts from 0. It can be put with @Test annotation.

**14) what are the steps in automation testing or when do we automation testing?**

**When to automate?** Ensure timing of automation is right. At minimum- product should be stable, and manual testing should be in place.

**Test tool selection**

**Define scope of automation**

**Planning, Design and development**

**Test execution**

**Maintenance**

**Test Automation Process:**

**#1 Test Automation - Plan**

This is the first step in the test automation process. The major action item here is to create a plan that specifies Purpose, Scope, Strategies, Major Requirements, Schedule, Budget.

**#2 Test Automation - Design and Development**

The major action item here is to create a detailed automation solution. This will adress the major objectives and meet all the automation requirements. This is more of a detailed breakup adress majority of the automation plan items.

In the development phase the various test automation framework and scripts are developed.

**#3 Test Automation Tool - Preparation**

The major action item here is evaluate the various automation tools and decide a on a tool to be used for the project. This is more of a feasibility study. In this stage an in-house tool can also be developed (if feasible). Once the tool is decided upon, the tool is deployed with the various required configuration required for the project.

**#4 Test Automation solution - Deployment**

Once the tool and the scripts are ready, they are integrated together and deployed on the test environment.

**#5 Test Automation - Review**

The working of the automation solution is reviewed and to identify issues and limitations and provide feedback. This will help to further enhance the solution.

**Switching from Manual Testing Process to Automated Testing Process**

Manual Test Process

1. Perform user actions.
2. Wait for processes to complete.
3. Verify AUT (Application under Test) functions as expected.
4. Repeat steps until all applications are verified compliant
5. Go back to step one.

Automated Test Process

1. Generate automated script.
2. Synchronize script playback to application performance.
3. Add verification.
4. Run test or suite of tests.
5. Go back to step four.

Ans: write manual test cases (if they are not there)

out of all manual test cases we will pick test cases that are of regression testing

(test cases that needs to be executed)

put all positive scenarios in regression

we don’t put any UI related changes in regression testing.

**15) what is the use of data provider annotation?**

 It helps you to write data-driven tests, which essentially means that **same test method can be run multiple times with different data-sets**.

DataProvider is the second way of passing parameters to test methods.  It helps in providing complex parameters to the test methods as it is not possible to do this from XML.

To use the DataProvider feature in your tests you have to declare a method annotated by @DataProvider and then use the said method in the test method using the ‘***dataProvider***‘ attribute in the Test annotation.

When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a database, etc…), in such cases parameters can be passed using Dataproviders. A Data Provider is a method annotated with @DataProvider. A Data Provider returns an array of objects.

1)  Define the method credentials() which is defined as a Dataprovider using the annotation. This method returns array of object array.

2) Add a method test() to your DataProviderTest class. This method takes two strings as input parameters.

3) Add the annotation *@Test(dataProvider = “Authentication”)* to this method. The attribute dataProvider is mapped to “Authentication”.

**16) Difference between junit and testNG?**

Both TestNG and Junit are Testing framework used for unit testing. TestNG is similar to JUnit.

Few more functionalities are added to it that makes TestNG more powerful than JUnit.

TestNG is a testing framework inspired by JUnit and NUnit.

1. In TestNG, Parameterized test configuration is very easy while It is very hard to configure Parameterized test in JUnit.
2. TestNG support group test but it is not supported in JUnit.
3. TestNG has a feature to configure dependency test. Dependency test configuration for software web application is not possible in JUnit.
4. TestNG support @BeforeTest, @AfterTest, @BeforeSuite, @AfterSuite, @BeforeGroups, @AfterGroups which are not supported in JUnit.
5. Test prioritization, Parallel testing is possible in TestNG. It is not supported by JUnit.

(<http://www.guru99.com/junit-vs-testng.html>)

**17) How to generate reports using testNG?**

**(**<http://www.techbeamers.com/generate-reports-selenium-webdriver/>**)**

Selenium Webdriver doesn’t have a built-in reporting feature but there are plugins like the TestNG and JUnit which can add this functionality.

Here we have 3 unique techniques to generate reports in Selenium Webdriver.

1. TestNG HTML report generation
2. JUnit-Style Report Generation.
   * Simple Junit reports
   * Convert Junit to HTML reports
3. Generating extent HTML reports

TestNG HTML report generation:

Once we execute the 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.

**Generate Reports (index.html) in TestNG.**

**Summary Report.**

It is the trimmed version and informs about the test pass/fail/skip count. 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.

**Steps To Generate Reports Using TestNG.**

**Step-1) Select The TestNG Reporting Interface.**

TestNG supplies two interfaces to implement reporting in Selenium Webdriver. You can apply any of these two in your projects.

1. Testers commonly use the <[ITestListener](http://testng.org/javadocs/org/testng/ITestListener.html)> Interface.
2. Another is the <***IReporter***> Interface which is least popular.

**Step-2) Generate Reports Using The <ITestListener> Interface.**

**Step-3) Verify the Report Generation Process.**

**18) Creating Jar file (separate project) and use that JAR file in selenium project file**

To create a new JAR file in the workbench:

1. In the Package Explorer, you can optionally pre-select one or more Java elements to export. (These will be automatically selected in the JAR package specification wizard page, described in Step 4.)
2. Either from the context menu or from the menu bar's **File** menu, select **Export**.
3. Expand the **Java** node and select **JAR file**. Click **Next**.
4. In the **JAR File Specification** page, select the resources that you want to export in the **Select the resources to export**field.
5. Select the appropriate checkbox to specify whether you want to **Export generated class files and resources** or **Export Java source files and resources**. **Note**: Selected resources are exported in both cases.
6. In the **Select the export destination** field, either type or click **Browse** to select a location for the JAR file.
7. Select or clear the **Compress the contents of the JAR file** checkbox.
8. Select or clear the **Overwrite existing files without warning** checkbox. If you clear this checkbox, then you will be prompted to confirm the replacement of each file that will be overwritten.
9. **Note**: The overwrite option is applied when writing the JAR file, the JAR description, and the manifest file.
10. You have two options:
    * Click **Finish** to create the JAR file immediately.
    * Click **Next** to use the JAR Packaging Options page to [set advanced options](http://help.eclipse.org/neon/topic/org.eclipse.jdt.doc.user/tasks/tasks-34.htm), create a JAR description, or [change the default manifest](http://help.eclipse.org/neon/topic/org.eclipse.jdt.doc.user/tasks/tasks-35.htm).

**19. Include Exception Manager class in framework project under proper package?**

**20. Include class for reading data from excel file under proper package**

**21. Include class for reading data from database table file under proper package**

**22. create maven project?**

**(**<http://www.tech-recipes.com/rx/39279/create-a-new-maven-project-in-eclipse/>**)**

When creating a project in Eclipse, one may use Maven to manage dependencies more easily and to resolve transitive dependencies automatically. Maven projects have a consistent structure for each project created, and it is possible to create this structure automatically within Eclipse.

1. In the **Eclipse IDE**, navigate to **File > New > Other…** in order to bring up the project creation wizard.

2. Scroll to the **Maven** folder, open it, and choose **Maven Project**. Then choose **Next**.

3. You may choose to **Create a simple project** or forgo this option. For the purposes of this tutorial, we will choose the simple project. This will create a basic, Maven-enabled Java project. If you require a more advanced setup, leave this setting unchecked, and you will be able to use more advanced Maven project setup features. Leave other options as is, and click **Next**.

4. Now, you will need to enter information regarding the Maven Project you are creating. You may visit the Maven documentation for a more in-depth look at the Maven Coordinates (http://maven.apache.org/pom.html#Maven\_Coordinates). In general, the **Group Id** should correspond to your organization name, and the **Artifact Id** should correspond to the project’s name. The version is up to your discretion as is the packing and other fields. If this is a stand-alone project that does not have parent dependencies, you may leave the **Parent Project** section as is. Fill out the appropriate information, and click **Finish**.

5. You will now notice that your project has been created. You will place your Java code in **/src/main/java**, resources in **/src/main/resources**, and your testing code and resources in **/src/test/java** and **/src/test/resources** respectively.

6. Open the **pom.xml** file to view the structure Maven has set up. In this file, you can see the information entered in Step 4. You may also use the tabs at the bottom of the window to change to view **Dependencies**, the **Dependency Hierarchy**, the **Effective POM**, and the raw xml code for the pom file in the **pom.xml** tab.

Now we have a new Java project with Maven enabled.

**23. Configure testNG in eclipse?**

(<http://www.techbeamers.com/install-testng-in-eclipse-ide/>)

We have three ways to install TestNG in eclipse.

* 1. Install TestNG in Eclipse [directly from the Eclipse Marketplace](http://www.techbeamers.com/install-testng-in-eclipse-ide/#method1).
  2. Installing TestNG Plugin in Eclipse [using the “Install New Software…” feature](http://www.techbeamers.com/install-testng-in-eclipse-ide/#method2).
  3. Install TestNG in Eclipse IDE via [offline Jar files](http://www.techbeamers.com/install-testng-in-eclipse-ide/#method3).

**24. Create methods with different annonations like beforeclass, beforetest, test, dataprovider, afterclass, aftertest, beforesuite, aftersuite, parameters?**

**package** TestNG;

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

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

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

**import** org.testng.annotations.BeforeTest;

**import** org.testng.annotations.AfterTest;

**import** org.testng.annotations.BeforeSuite;

**import** org.testng.annotations.AfterSuite;

**public** **class** TestNGAnnotations

{

@Test

**public** **void** Test()

{

System.***out***.println("@Test");

}

@BeforeClass

**public** **void** BeforeClass()

{

System.***out***.println("@BeforeClass");

}

@AfterClass

**public** **void** AfterClass()

{

System.***out***.println("@AfterClass");

}

@BeforeTest

**public** **void** BeforeTest()

{

System.***out***.println("@BedforeTest");

}

@AfterTest

**public** **void** AfterTest()

{

System.***out***.println("@AfterTest");

}

@BeforeSuite

**public** **void** BeforeSuite()

{

System.***out***.println("@BeforeSuite");

}

@AfterSuite

**public** **void** AfterSuite()

{

System.***out***.println("@AfterSuite");

}

}

**Have to learn about @DataProviders and @Parameters**

**25. Configure testng.xml for creating test suites, specifying test classes, parameters?**

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

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

<suite guice-stage="DEVELOPMENT" name="Default suite">

<test verbose="2" name="Default test">

<classes>

<class name="TestNG.TestNGAnnotations"/>

</classes>

</test> <!-- Default test -->

</suite> <!-- Default suite -->

**26. Testng reports generation using Listeners for report generation (itestlistener, isuitelistener)?**

There are two ways to generate a report with TestNG:

* Listeners: For implementing a listener class, the class has to implement the *org.testng.ITestListener* interface. These classes are notified at runtime by TestNG when the test starts, finishes, fails, skips, or passes.
* Reporters: For implementing a reporting class, the class has to implement an *org.testng.IReporter* interface. These classes are called when the whole suite run ends. The object containing the information of the whole test run is passed to this class when called.

**27. page object model**

**28. page factory:**

In order to support the [PageObject](https://github.com/SeleniumHQ/selenium/wiki/PageObjects.md) pattern, WebDriver's support library contains a factory class.

Page Factory is an inbuilt page object model concept for Selenium WebDriver but it is very optimized.

Here as well we follow the concept of separation of Page Object repository and Test methods. Additionally, with the help of PageFactory class we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements