## eWhat is TestNG?

***TestNG*** is a testing framework inspired from ***JUnit*** and ***NUnit*** but introducing some new functionality that makes it more powerful and easier to use.

It is an open-source automated testing framework; where ***NG*** of Test**NG** means **N**ext **G**eneration. TestNG is similar to JUnit but it is much more powerful than JUnit but still, it's inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. Pay special thanks to Cedric Beust who is the creator of TestNG.

TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parametrizing.

### *****What are the Benefits of TestNG?*****

There are a number of benefits but from Selenium perspective, major advantages of TestNG are :

1. It gives the ability to produce ***HTML Reports*** of execution
2. ***Annotations*** made testers life easy
3. Test cases can be ***Grouped & Prioritized*** more easily
4. ***Parallel*** testing is possible
5. Generates ***Logs***
6. Data ***Parameterization*** is possible

## ***Advantages of TestNG over Junit***

***There are three major advantages of TestNG over JUnit:***

* ***Annotations are easier to understand***
* ***Test cases can be grouped more easily***
* ***Parallel testing is possible***
* ***Dependency Tests are supported for TestNG only. These are tests where one method will not run unless the dependent method runs & passes.***

### *****Test Case Writing process in TestNG*****

Writing a test in TestNG is quite simple and basically involves the following steps:

* ***Step 1*** - Write the business logic of the test
* ***Step 2*** - Insert TestNG annotations in the code
* ***Step 3*** - Add the information about your test (e.g. the class names, methods names, groups names, etc...) in a testng.xml file
* ***Step 4*** - Run TestNG

### *****What are the different Annotations are present in TestNG?*****

* ***@BeforeSuite***: The annotated method will be run before all tests in this suite have run.
* ***@AfterSuite***: The annotated method will be run after all tests in this suite have run.
* ***@BeforeTest***: The annotated method will be run before any test method belonging to the classes inside the tag is run.
* ***@AfterTest***: The annotated method will be run after all the test methods belonging to the classes inside the 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.
* ***@BeforeClass***: The annotated method will be run before the first test method in the current class is invoked.
* ***@AfterClass***: The annotated method will be run after all the test methods in the current class have been run.
* ***@BeforeMethod***: The annotated method will be run before each test method.
* ***@AfterMethod***: The annotated method will be run after each test method.
* ***@Test***: The annotated method is a part of a test case.

***Benefits of using Annotations***

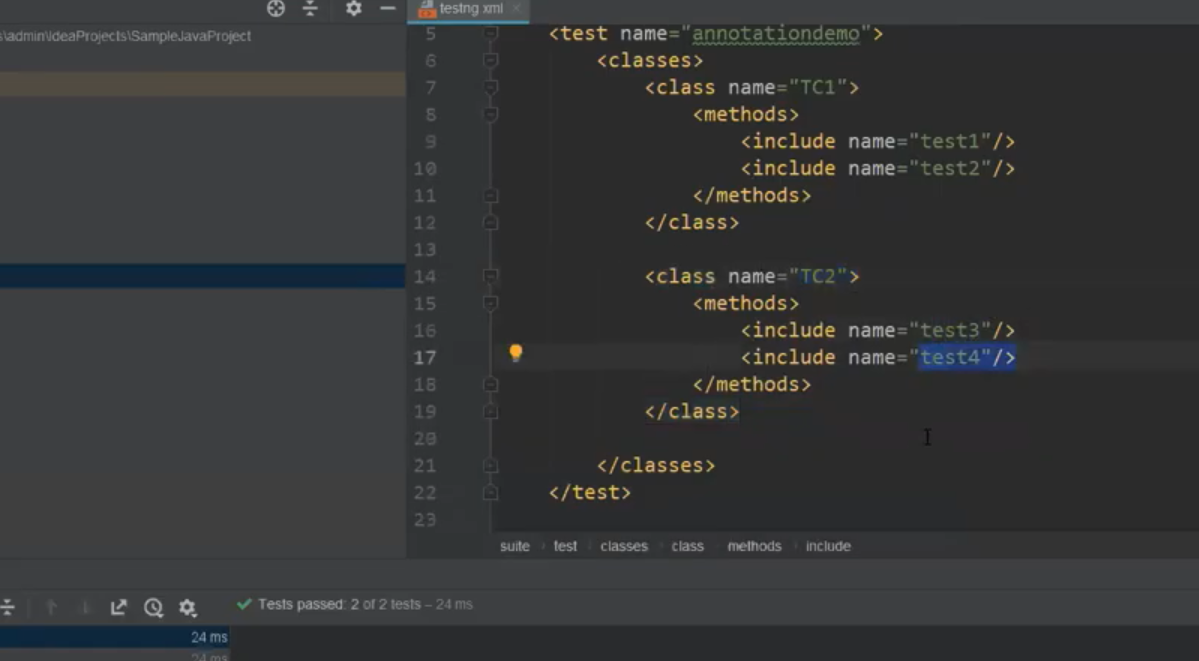
1. It identifies the methods it is interested in by looking up annotations. Hence method names are not restricted to any pattern or format.
2. We can pass additional parameters to annotations.
3. Annotations are strongly typed, so the compiler will flag any mistakes right away.
4. Test classes no longer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

example

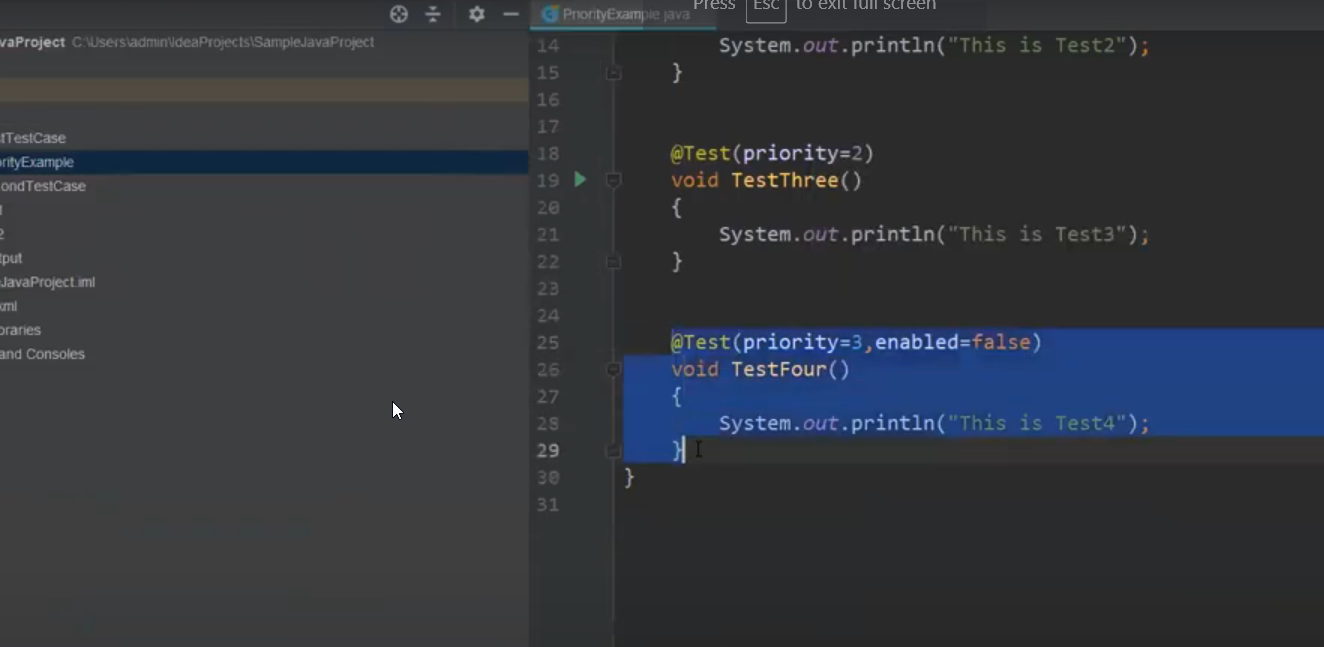
<https://www.javatpoint.com/testng-beforesuite-annotation>

Diagram, schematic

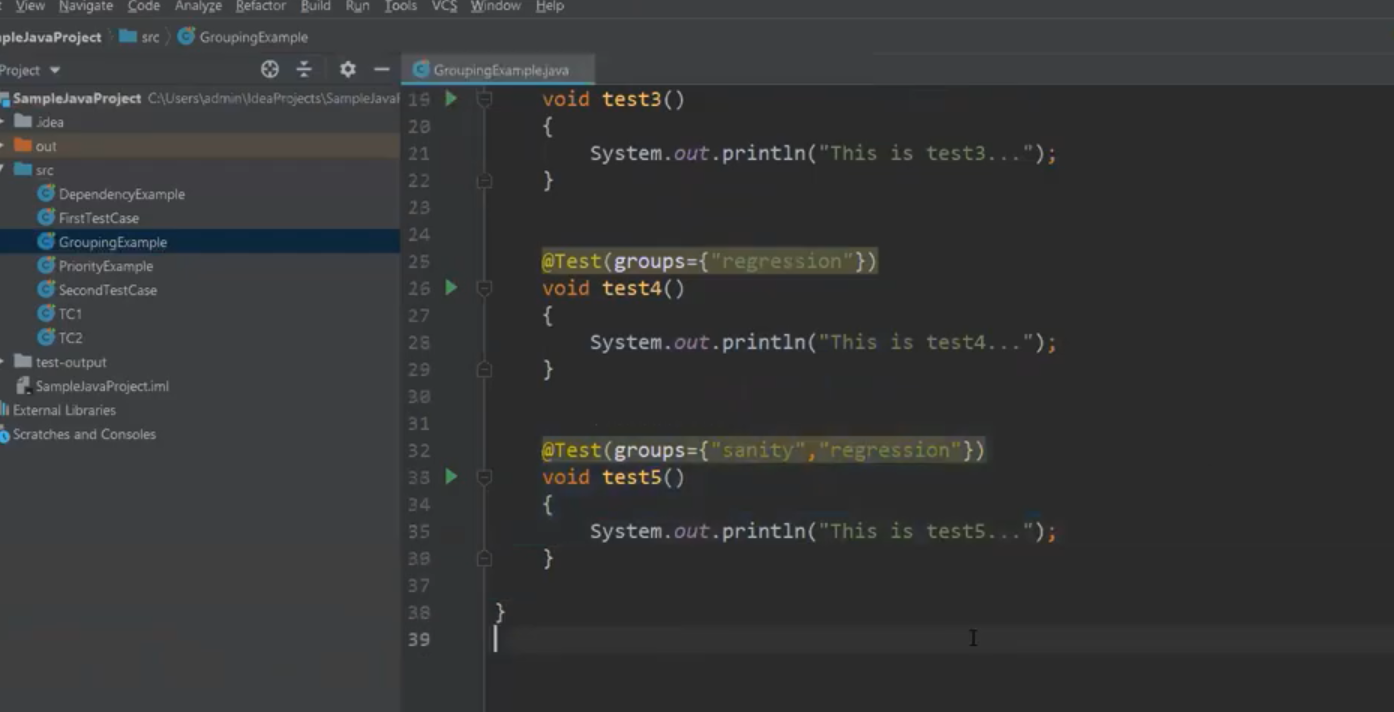
Description automatically generated



Prioritizing and disabling tests



Grouping to sanity or regression



<https://www.softwaretestinghelp.com/testng-annotations-in-selenium/>

TestNG annotations are lines of code that are inserted in the program/ business logic to control how the methods below are to be run.

The annotations differ depending on your project requirements. However, the flow of execution will be the same.

**Prerequisites:**

* Install TestNG in Eclipse. Check [this tutorial on Installation guide](https://www.softwaretestinghelp.com/testng-framework-selenium-tutorial-12/).
* JDK – [Java Development Kit](https://www.softwaretestinghelp.com/webdriver-eclipse-installation-selenium-tutorial-9/)
* Annotations can be used only with Java 1.5 version or higher

Before writing test scripts or setting up a project, we should know the hierarchy in which the annotations work. The execution will always remain the same.

For example, compile and run the below script and notice the execution order. It will be as following:

* BeforeSuite
* BeforeTest
* BeforeClass
* BeforeMethod
* Test Case 1
* AfterMethod
* BeforeMethod
* Test Case 2
* AfterMethod
* AfterClass
* AfterTest
* AfterSuite

**Example:**

|  |
| --- |
| **public** **class** test {    @BeforeMethod  **public** **void** beforeMethod() {  System.out.println(" Before Method will execute before every test method");  }    @AfterMethod  **public** **void** afterMethod() {  System.out.println("After Method will execute after every test method ");  }    @BeforeClass  **public** **void** beforeClass() {  System.out.println("Before Class will always execute prior to Before Method and Test Method ");  }    @AfterClass  **public** **void** afterClass() {  System.out.println("After Class will always execute later to After Method and Test method");  }    @BeforeTest  **public** **void** beforeTest() {  System.out.println("Before Test will always execute prior to Before Class, ,Before Method and Test Method ");  }    @AfterTest  **public** **void** afterTest() {  System.out.println("After Test will always execute later to After Method, After Class ");  }    @BeforeSuite  **public** **void** beforeSuite() {  System.out.println(“Before Suite will always execute prior to all annotations or tests in the suite.");  }    @AfterSuite  **public** **void** afterSuite() {  System.out.println("After suite will always execute at last when all the annotations or test in the suite have run.");  }    @Test  **public** **void** testCase1() {  System.out.println("This is my First Test Case 1");  }    @Test  **public** **void** testCase2() {  System.out.println("This is my Second Test Case 2");  }    } |

**We can break the test script process in to below steps:**

1. Write the business logic of your test and insert above [TestNG annotations](http://testng.org/doc/documentation-main.html#annotations) in your code
2. Add the information about your test (e.g. the class name, the groups, methods you wish to run, etc.) in a testng.xml file.
3. Run TestNG

But the question still remains – what information should we provide in the above annotations?

**Take a look at the important steps we can achieve using the above annotations:**

**#1)** **@Test**

This is the main part of our automation script where we will write the business logic, the things which we want to automate. We can pass attributes to our test method.

**Below are the lists of attributes that we can pass to our Test method:**

* **alwaysRun**: This is used when we want to make sure a method always runs even if the parameters on which the method depends, fails. If set to true, this test method will always run. Eg: @Test(alwaysRun = true)
* **dataProvider**: TestNG dataProvider is used to provide any data for parameterization. Eg. @Test(dataProvider = “Hello”).
* **dataProviderClass**: This is the class from where we pass the data to data provider. In our case dataProvider class name is “Hello”.
* **dependsOnGroups**: It is the list of groups this method depends on. Eg: @Test (groups = { “City” ,”State” })
* **dependsOnMethods:**This command is used to execute a method based on its dependent method. Eg: @Test (dependsOnMethods = { “OpenBrowser” ,”database is up” })
* **description**: It is the description for the method. Eg: @Test(description = “test method”)
* **invocationCount**: It refers to the number of times a method should be invoked. It will work as a loop. Eg: @Test(invocationCount = 7) . Hence, this method will execute 7 times.
* **invocationTimeOut**: This refers to the maximum number of milliseconds a method should take for all the invocationCount to complete. This attribute will be ignored if invocationCount is not specified. Eg: @Test(invocationCount =7,invocationTimeOut = 30 )
* **priority**: This command sets the priority of the test method. Lower priorities will be scheduled first. Eg: @Test(priority =1 )

**#2) @BeforeSuite and @AfterSuite**

In @BeforeSuite annotated method, you can setup and start [selenium](http://testng.org/doc/selenium.html) drivers and in @AfterSuite annotated method, you can stop Selenium drivers

**Example**:

|  |
| --- |
| **public** **class** TestSuiteSetup () {    @BeforeSuite(alwaysRun = **true**)  **public** **void** setupSuite() {  WebDriver driver = **new** FirefoxDriver();  }    @AfterSuite(alwaysRun = **true**)  **public** **void** tearDown() {  driver().close();  }    } |

**#3) @BeforeClass and @AfterClass**

In @BeforeClass annotated method, you can setup your firefox properties, initialize your driver and so on and in @AfterClass annotated method, you can stop driver

**Example**:

|  |
| --- |
| @BeforeClass(description = "Set capabilities for your Firefox browser and set time it should wait for a page to load.")    **public** **static** **void** firefoxSetUp() **throws** MalformedURLException {    DesiredCapabilities capability = DesiredCapabilities.firefox();  driver = **new** FirefoxDriver(capability);  driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);  driver.manage().window().setSize(**new** Dimension(1920, 1080));    }    @AfterClass(description = "close your firefox driver")    **public** **void** afterclass(){  driver.close();  } |

**#4) @BeforeMethod and @AfterMethod**

In @BeforeMethod annotated method, you can check database connection before executing your test method and in @AfterMethod annotated method, you can close your data base connection

**Example**:

|  |
| --- |
| @BeforeMethod(description="connect to database")    **public** **void** beforemethod() **throws** SQLException{    //check database connection is up  String databaseurl = "jdbc:oracle://192.168.1.258/myDB";  DriverManager.getConnection(databaseurl, "username", "password");  } |
| @AfterMethod(description="close database connection")    **public** **void** aftermethod() **throws** SQLException{  //check database connection is closed  String databaseurl = "jdbc:oracle://192.168.1.258/myDB";  Connection connect = DriverManager.getConnection(databaseurl, "username", "password");    **if**(connect!=**null**)  connect.close();  } |

**#5) @BeforeTest and @AfterTest**

In @BeforTest method, you can set your firefox profile preferences and in @AfterTest method, you can put some code which will generate the test result and mail it to the stake holders

**Example**:

|  |
| --- |
| @BeforeTest (description="set your firefox profile preferences according to your project requirement")    **public** **void** single\_run(){  FirefoxProfile firefoxProfile = **new** FirefoxProfile();  firefoxProfile.setPreference ("browser.download.folderList",2);  firefoxProfile.setPreference ("browser.download.manager.showWhenStarting",**false**);  firefoxProfile.setPreference ("browser.download.dir","E:\\reports\\");  firefoxProfile.setPreference ("browser.helperApps.neverAsk.saveToDisk","csv");  driver = **new** FirefoxDriver(firefoxProfile);  String baseUrl = "www.gmail.com";  }    @AfterTest (description="")  **public** **void** teardown(){  System.out.println(“testcase”);  //a code which will send the test details report  } |

The most important aspect that should be noted here while working with annotations is that your system should be equipped with Java 1.5 version or higher, otherwise Eclipse might show an error that annotations are not supported on your system.

Now, consider a case where your system has the right version of Java needed for annotations but the error still appears.

Something like below:

***Syntax error, annotations are only available if source level is 1.5 or greater.***

What will you do now? There are **three options to rectify this situation.**

Let’s go through it one by one:

**Option #1:**

* Go to Eclipse and right click on your project
* Select Properties
* Click on Java Compiler
* Make sure your Compiler Compliance Level is 1.5 or higher
* Save the settings and your problem is solved

**Option #2:**

* Go to Window Tab in Eclipse
* Select Preferences
* Click on Java and then on Compiler
* Make sure your Compiler Compliance Level is 1.5 or higher
* Save the settings and your problem is solved

**Option #3:**

Check your Java Home Path by setting the correct Java environment path variable.

### Conclusion:

Through this article, we tried to discuss some of the important annotations and attributes which are frequently used by testers. However, there are more annotations in TestNG which are not frequently used such as @AfterGroups, @BeforeGroups and so on which are used when you are working with groups in your project test script.

So use the above annotations according to your requirements. It is always advisable not to do your project setup in test method. In test method write the core business logic which is to be tested.

Make sure your system is equipped with Java 1.5 versions or higher, otherwise Eclipse might show an error that annotations are not supported on your system.

### Invoke DataProvider from different class

By default, DataProvider resides in the same class where test method is or its base class. To put it in some other class we need to make data provider method as static and in test method we need to add an attribute **dataProviderClass** in**@Test** annotation.

@Test(dataProvider="SearchProvider",dataProviderClass=DataproviderClass.class)

Creating TestNG.xml in project in Intellij

Install Create TestNG XML plugin to create TestNG.XML in the project.

Right click on the project and click on create TestNG.XML

To see test-0utput folder after running test through TestNG.XML

From Edit Configurations

1. On the "Listeners tab", Use default listeners should be checked.
2. And Output directory should be set correctly.

Or

You can add Listener like "EmailableReporter" or "FailedReporter" in the "Run Configurations", it will generate test-output directory in your workspace.

https://www.toolsqa.com/testng/what-is-testng/

<https://www.softwaretestinghelp.com/testng-annotations-in-selenium/>

<https://www.softwaretestinghelp.com/testng-annotations-and-listeners/>

<https://www.softwaretestinghelp.com/testng-framework-selenium-tutorial-12/>