TestNG

TestNG is a testing framework for the Java programming language created by Cédric Beust and inspired by JUnit and NUnit. The design goal of TestNG is to cover a wider range of test categories: unit, functional, end-to-end, integration, etc., with more powerful and easy-to-use functionalities.

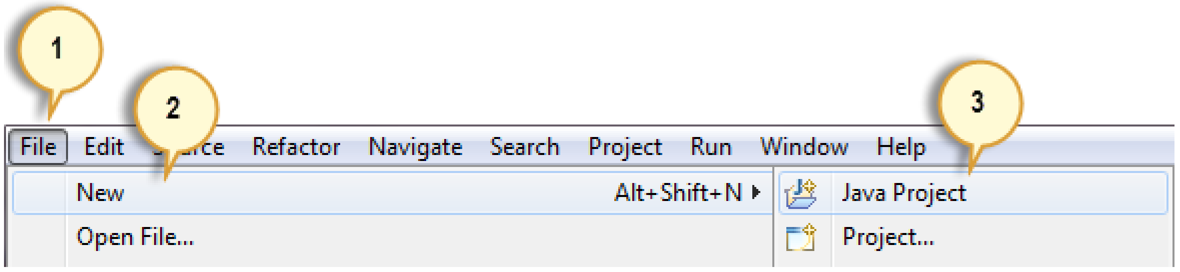
TestNG is inspired from JUnit which uses the annotations (@).

First test case using annotations

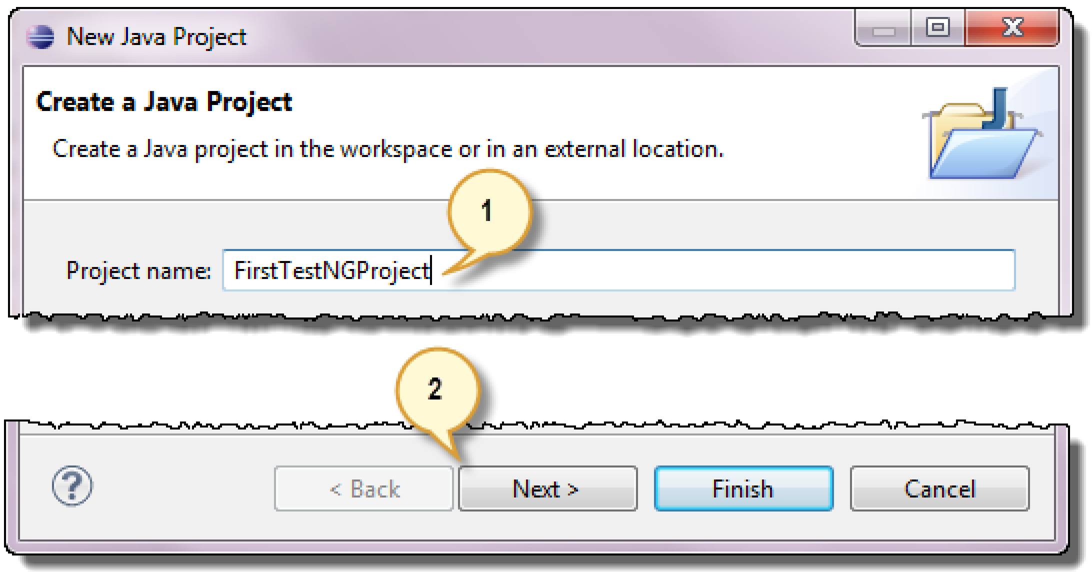
Before we create a test case, we should first setup a new TestNG Project in Eclipse and name it as "FirstTestNGProject".

Setting up a new TestNG Project

Step 1: Click File > New > Java Project



Step 2: Type "FirstTestNGProject" as the Project Name then click Next.

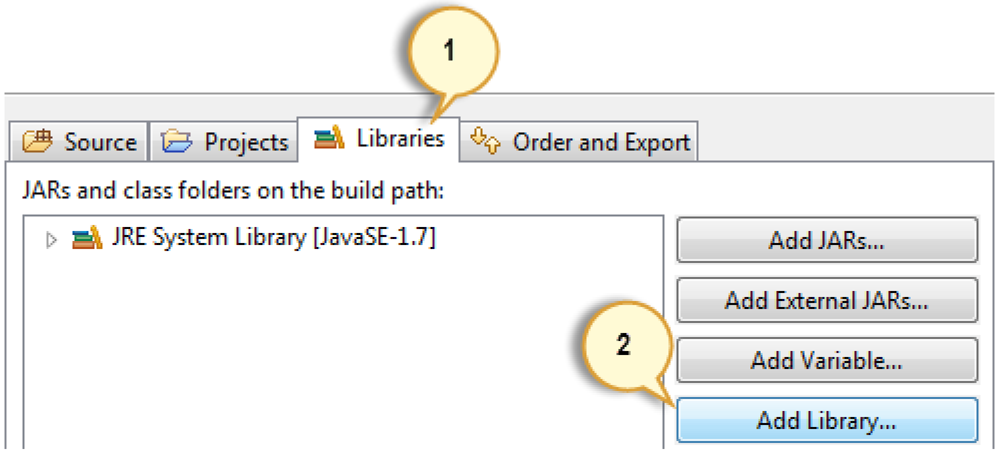


Create a Java Project

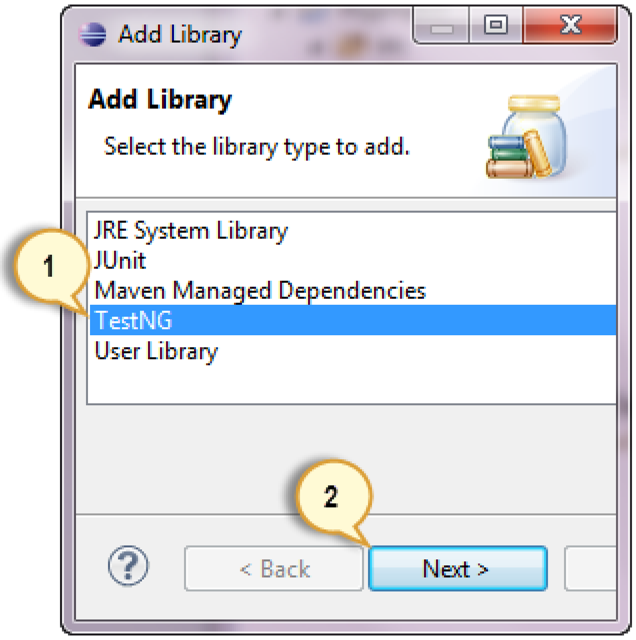
Create a Java project in the workspace or in an external location.

Project name: FirstTestNGProject

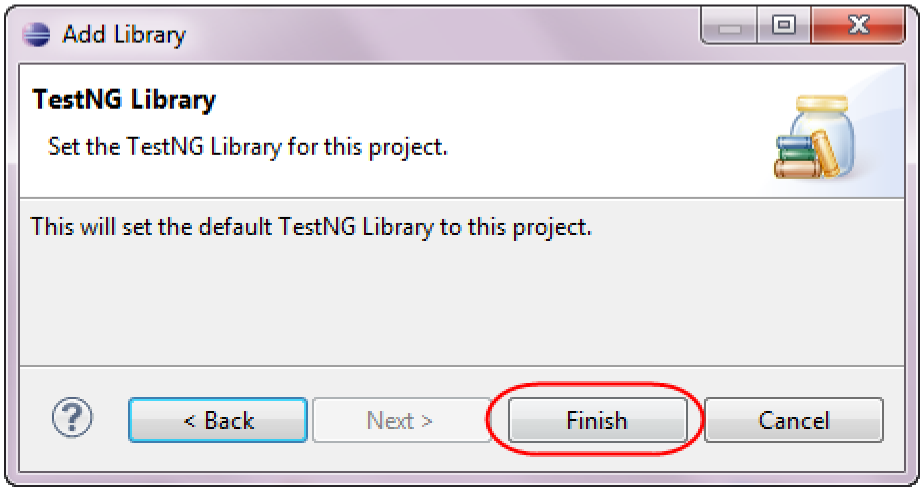
Step 3: We will now start to import the TestNG Libraries onto our project. Click on the "Libraries" tab, and then "Add Library…"



Step 4: On the Add Library dialog, choose "TestNG" and click Next.



Step 5: Click Finish.

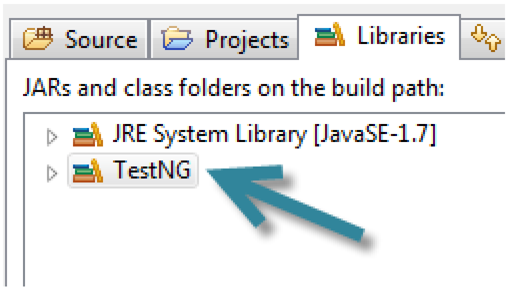

Add Library

TestNG Library

Set the TestNG Library for this project.

This will set the default TestNG Library to this project.

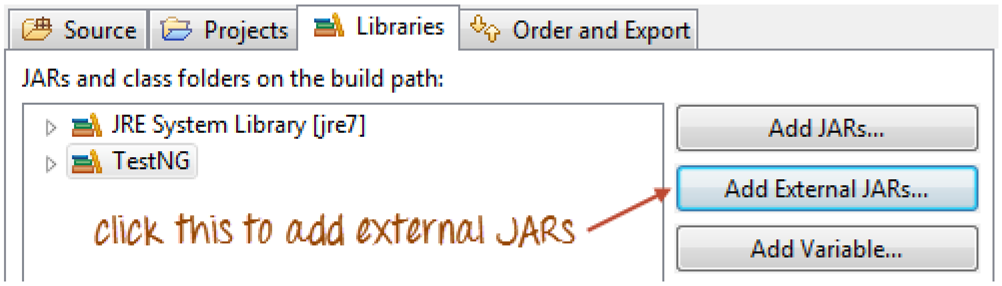
You should notice that TestNG is included on the Libraries list.



JARs and class folders on the build path:

- JRE System Library [JavaSE-1.7]
- TestNG

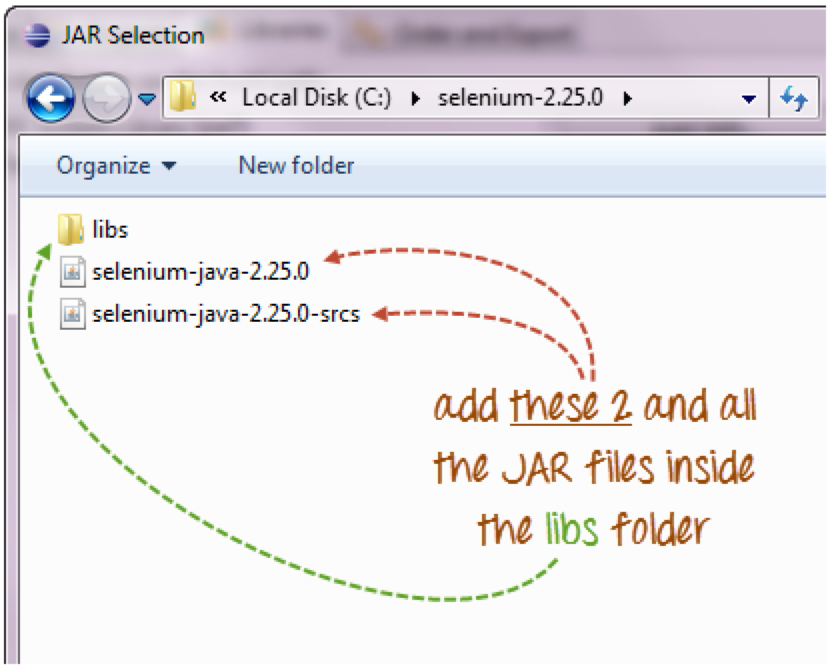
Step 6: We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from http://docs.seleniumhq.org/download/ when we were installing Selenium and Eclipse in the previous chapters.



JARs and class folders on the build path:

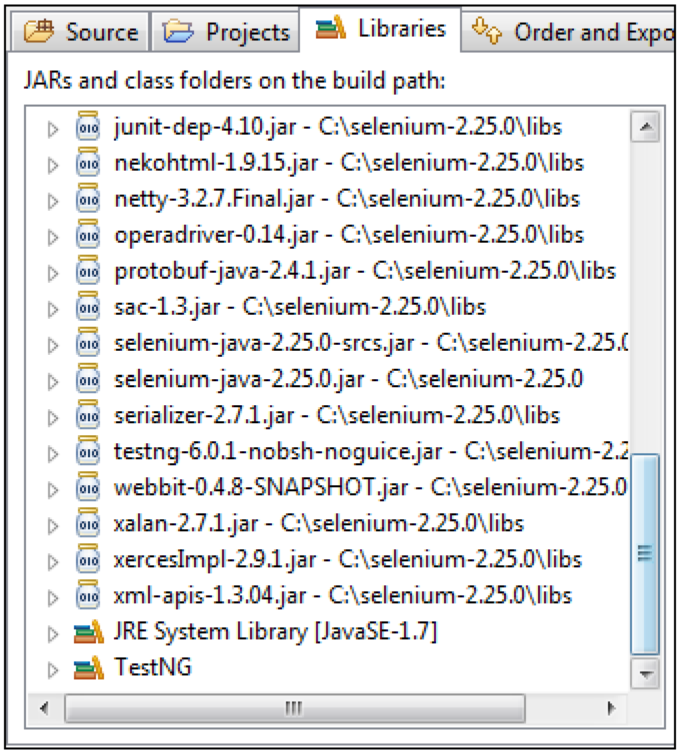
- JRE System Library [jre7]
- TestNG

Then, navigate to where you have placed the Selenium JAR files.



JAR Selection
Local Disk (C:) ▸ selenium-2.25.0 ▸
Organize ▾
New folder
libs
selenium-java-2.25.0
selenium-java-2.25.0-srcs
add these 2 and all the JAR files inside the libs folder

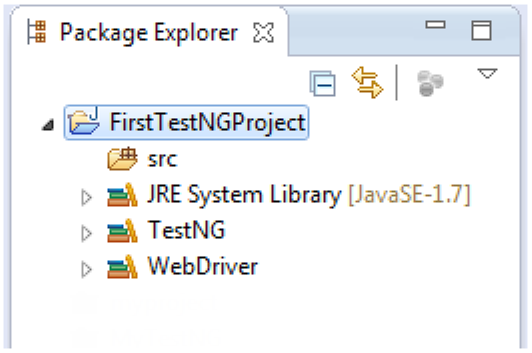
After adding the external JARs, your screen should look like this.

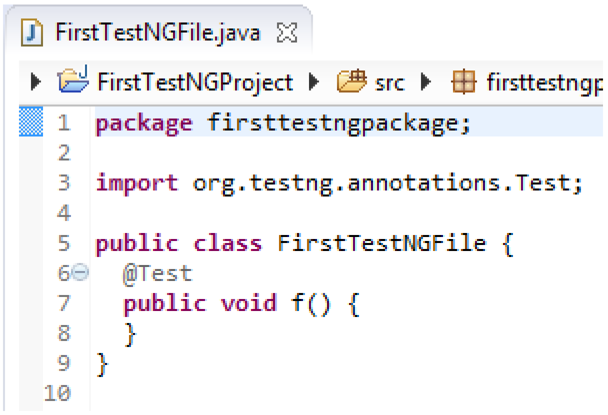


JARs and class folders on the build path:

```
junit-dep-4.10.jar - C:\selenium-2.25.0\libs
nekohtml-1.9.15.jar - C:\selenium-2.25.0\libs
netty-3.2.7.Final.jar - C:\selenium-2.25.0\libs
operadriver-0.14.jar - C:\selenium-2.25.0\libs
protobuf-java-2.4.1.jar - C:\selenium-2.25.0\libs
sac-1.3.jar - C:\selenium-2.25.0\libs
selenium-java-2.25.0-srcs.jar - C:\selenium-2.25.(
selenium-java-2.25.0.jar - C:\selenium-2.25.0
serializer-2.7.1.jar - C:\selenium-2.25.0\libs
testng-6.0.1-nobsh-noguice.jar - C:\selenium-2.2
webbit-0.4.8-SNAPSHOT.jar - C:\selenium-2.25.0
xalan-2.7.1.jar - C:\selenium-2.25.0\libs
xercesImpl-2.9.1.jar - C:\selenium-2.25.0\libs
xml-apis-1.3.04.jar - C:\selenium-2.25.0\libs
JRE System Library [JavaSE-1.7]
TestNG
```

Step 7: Click Finish and verify that our FirstTestNGProject is visible on Eclipse's Package Explorer window.





```
package firsttestngpackage;

import org.testng.annotations.Test;

public class FirstTestNGFile {
  @Test
  public void f() {
  }
}
```

Coding Our First Test Case

Let us now create our first Test Case that will check if Mercury Tours' homepage is correct. Type your code as shown below.

```
public class firsttestngfile {
    public String baseUrl = "http://demo.guru99.com/test/newtours/";
    String driverPath = "C:\\geckodriver.exe";
    public WebDriver driver ;

  @Test
  public void verifyHomepageTitle() {

      System.out.println("launching chrome browser");
      System.setProperty("webdriver.chrome.driver", driverPath);
      driver = new chromeDriver();
      driver.get(baseUrl);
      String expectedTitle = "Welcome: Mercury Tours";
      String actualTitle = driver.getTitle();
      Assert.assertEquals(actualTitle, expectedTitle);
      driver.close();
  }
}
```

Notice the following.

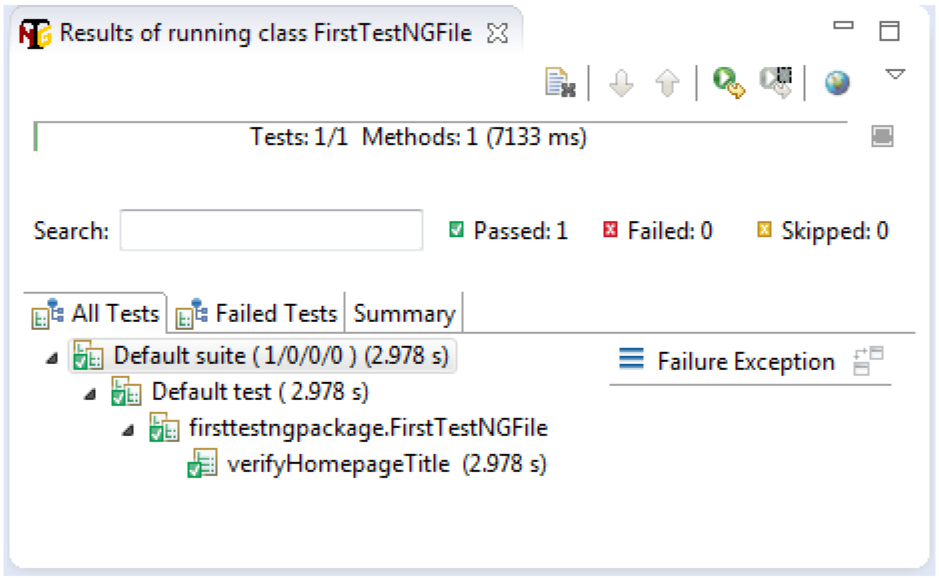
- TestNG does not require you to have a main() method.
- Methods need not be static.
- We used the @Test annotation. **@Test is used to tell that the method under it is a test case**. In this case.
- Since we use annotations in TestNG, we needed to import the package org.testng.annotations.*.
- We used the Assert class. **The Assert class is used to conduct verification operations in TestNG**. To use it, we need to import the org.testng.Assert package.

You may have multiple test cases (therefore, multiple @Test annotations) in a single TestNG file. This will be tackled in more detail later in the section "Annotations used in TestNG."

Running the Test

To run the test, simply run the file in Eclipse as you normally do. Eclipse will provide two outputs – one in the Console window and the other on the TestNG Results window.

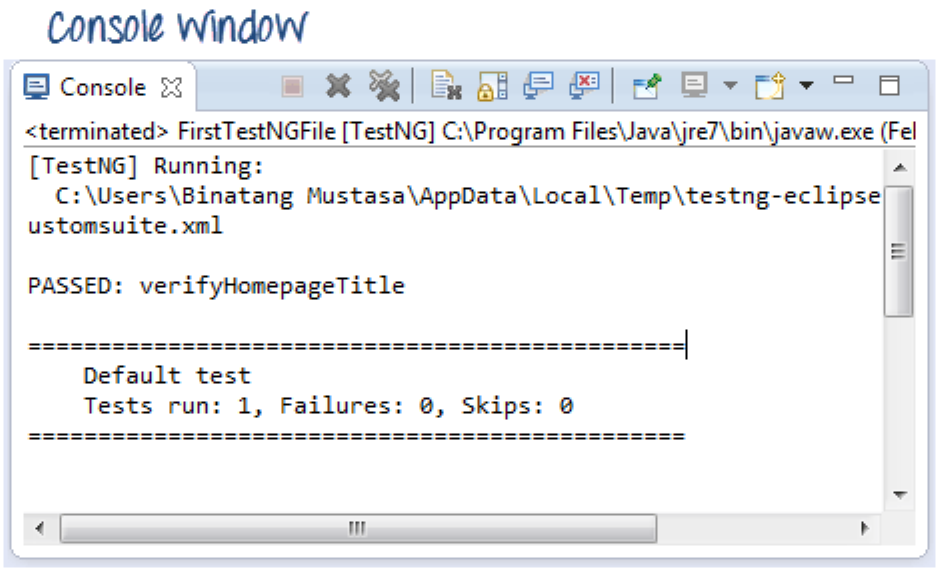
TestNG Results Window



Tests: 1/1 Methods: 1 (7133 ms)

Passed: 1 Failed: 0 Skipped: 0

```
Default suite ( 1/0/0/0 ) (2.978 s)
  Default test ( 2.978 s)
    firsttestngpackage.FirstTestNGFile
      verifyHomepageTitle (2.978 s)
```



```
<terminated> FirstTestNGFile [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Fel
[TestNG] Running:
  C:\Users\Binatang Mustasa\AppData\Local\Temp\testng-eclipse
ustomsuite.xml

PASSED: verifyHomepageTitle

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================
```

Annotations used in TestNG

In the previous section, you have been introduced to the @Test annotation. Now, we shall be studying more advanced annotations and their usages.

Multiple Test Cases

We can use multiple @Test annotations in a single TestNG file. By default, methods annotated by @Test are executed alphabetically. See the code below. Though the methods c_test, a_test, and b_test are not arranged alphabetically in the code, they will be executed as such.

```
public class FirstTestNGFile {

    @Test
    public void c_test() {
        Assert.fail();
    }

    @Test
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}
```

TestNG Annotations

@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.

.

@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.

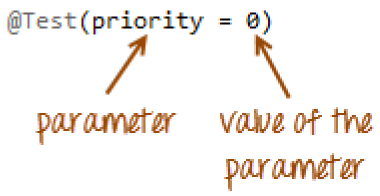
@AfterMethod: The annotated method will be run after each test method.

@Test: The annotated method is a part of a test case

Parameters

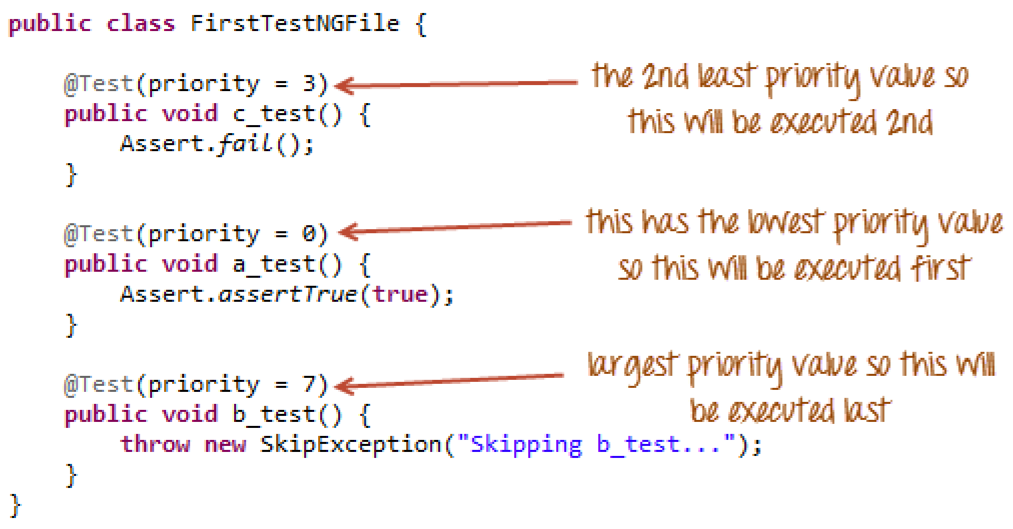
If you want the methods to be executed in a different order, use the parameter "priority". **Parameters are keywords that modify the annotation's function**.

- Parameters require you to assign a value to them. You do.this by placing a "=" next to them, and then followed by the value.
- Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.



```
@Test(priority = 0)
```

TestNG will execute the @Test annotation with the lowest priority value up to the largest. There is no need for your priority values to be consecutive.



```
public class FirstTestNGFile {

    @Test(priority = 3)
    public void c_test() {
        Assert.fail();
    }

    @Test(priority = 0)
    public void a_test() {
        Assert.assertTrue(true);
    }

    @Test(priority = 7)
    public void b_test() {
        throw new SkipException("Skipping b_test...");
    }
}
```

the 2nd least priority value so this will be executed 2nd
this has the lowest priority value so this will be executed first
largest priority value so this will be executed last

invocationCount

```
package org.softpost;

import org.testng.annotations.Test;

public class InvocationTest {


  @Test(invocationCount = 5)
    public void test1(){

        System.out.println("Test Run Number mention in test  ");
    }
}
```

Invocation count is used when you want to run the same tests multiple times. Below example illustrates how to use invocation count in TestNG. In below example, test1 will be executed 5 times

Group’s

‘**Groups**‘ is one more annotation of TestNG which can be used in the execution of multiple tests. Let’s say you have hundred tests of class vehicle and in it ten method of car, ten method of scooter and so on. You probably like to run all the scooter tests together in a batch. And you want all to be in a single test suite. With the help of grouping you can easily overcome this situation.

How to do it…

1) Create two methods for Car, two methods for Scooter and one method in conjunction with Car & Sedan Car.

2) Group them separately with using (groups = { " Group Name" })

```
 @Test (groups = { "Car" })

  public void Car2() {

  System.out.println("Batch Car - Test car 2");

  }

  @Test (groups = { "Scooter" })

  public void Scooter1() {

  System.out.println("Batch Scooter - Test scooter 1");

  }
```

Dependens on group's

Using attributes *dependsOnGroups* in @Test annotations.

```
@Test (dependsOnGroups = { " Car " })

  public void SignIn() {

  System.out.println("This will execute second (SignIn)");

  }

@Test (dependsOnGroups = { " Car "," Scooter " })

  public void SignIn() {

  System.out.println("This will execute second (SignIn)");

  }
```

Using attributes *dependsOnMethods* in @Test annotations

```
@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)");

}

@Test (dependsOnMethods = { "SignIn" , " OpenBrowser " })
public void LogOut() {

System.out.println("This will execute third (Log Out)");

}
```

What is TestNG Listener?

TestNG listener formally called as ITestListener, which is an interface in TestNG. A normal Java class implements ITestListener and overrides all the methods written inside it. Each method corresponds to an event of your Selenium Project.

ITestListener effectively helps in creating a new way of logging. We can implement log4j logging along with the implementation of ITestListener.

```
public class ListenersDefinitionClass implements ITestListener{

public void onTestStart(ITestResult result) {
// TODO Auto-generated method stub
}

public void onTestSuccess(ITestResult result) {
// TODO Auto-generated method stub
}

public void onTestFailure(ITestResult result) {
```

```
// TODO Auto-generated method stub
}

public void onTestSkipped(ITestResult result) {
// TODO Auto-generated method stub
}

public void onTestFailedButWithinSuccessPercentage(ITestResult result) {
// TODO Auto-generated method stub
}

public void onStart(ITestContext context) {
// TODO Auto-generated method stub
}

public void onFinish(ITestContext context) {
// TODO Auto-generated method stub
}

}
```

Implementation of Listener by using @Listeners annotation

Here we will keep testng.xml file normally implemented whereas we implement the listener in a specific class file by using @Listeners annotation.

```
@Listeners(ListenersDefinitionClass.class)
public class TestngListeners {
WebDriver driver;
 @Test()
  public void getTitle() {

  System.out.println("Test ");

  }

}
```

Assertions

Assertions in TestNG. **Asserts** helps us to verify the conditions of the test and decide whether test has failed or passed. A test is considered successful ONLY if it is completed without throwing any exception. There is a difference between SoftAssert and Hard **Assert**.

```
String Et = "ankur.fb.com";
            String At = "ankur.fb.com";
```

```java
Assert.assertEquals(At, Et);
System.out.println("Actual is matching with expected titel");

boolean Ex = true;
boolean Ax = true;
Assert.assertEquals(Ax, Ex);
System.out.println("Actual is matching with expected titel");

Assert.assertNotEquals(s, s2, "Working");

Assert.assertTrue(true);// same for false

SoftAssert sa = new SoftAssert();
sa.assertEquals(9, 19);
System.out.println("Failjasdfs");
sa.assertAll();
```