# **TestNG Annotations, Framework, Examples in Selenium**

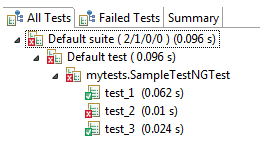
* TestNG is a testing framework that is capable of making Selenium tests easier to understand and of generating reports that are easy to understand.
* The main advantages of TestNG over JUnit are the following.
  + Annotations are easier to use and understand.
  + Test cases can be grouped more easily.
  + TestNG allows us to create parallel tests.
* The Console window in Eclipse generates a text-based result while the TestNG window is more useful because it gives us a graphical output of the test result plus other meaningful details such as:
  + Runtimes of each method.
  + The chronological order by which methods were executed
* TestNG is capable of generating HTML-based reports.
* Annotations can use parameters just like the usual Java methods.

### Why Use TestNG with Selenium?

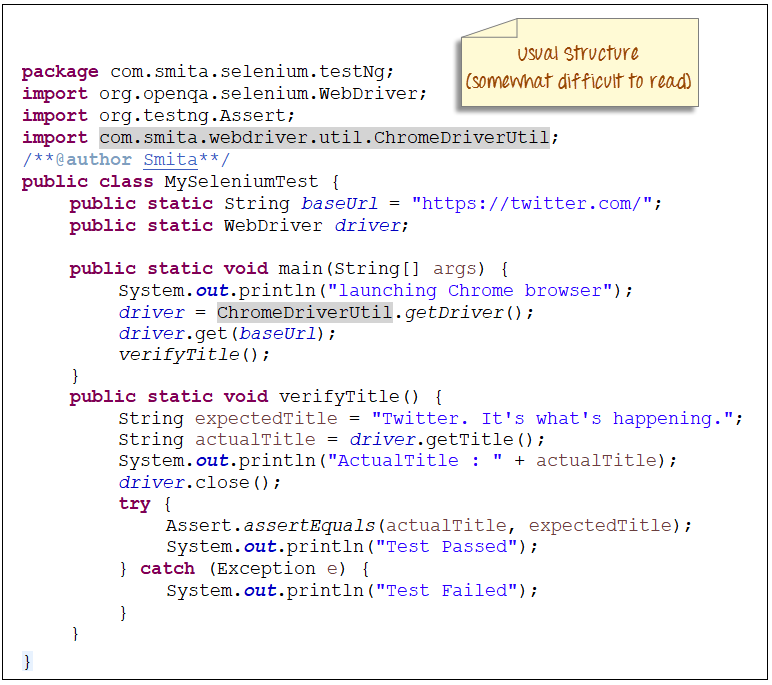
Default Selenium tests do not generate a proper format for the test results. Using TestNG we can generate test results.

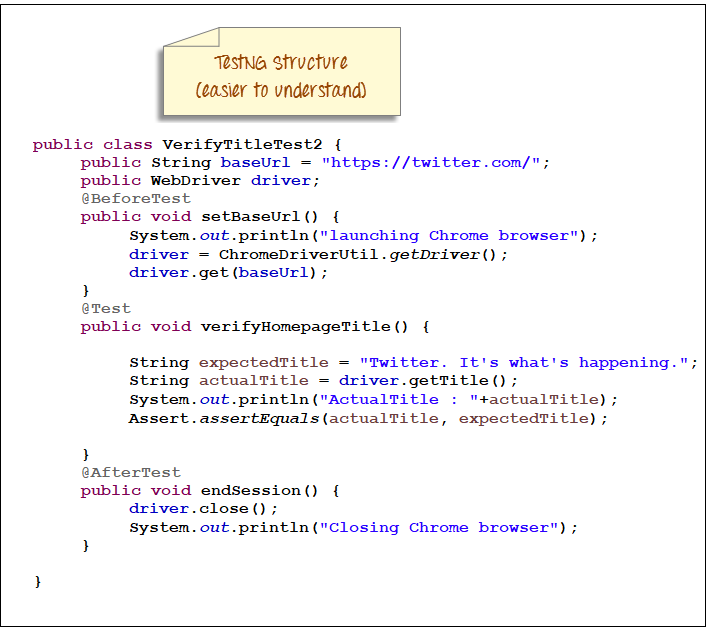
Most Selenium users use this more than Junit because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium. Following are key features of TestNG

* Generate the report in a proper format including a number of test cases runs, the number of test cases passed, the number of test cases failed, and the number of test cases skipped.
* Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first.
* The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'
* Using testng, you can execute multiple test cases on multiple browsers, i.e., cross browser testing.
* The testing framework can be easily integrated with tools like Maven, Jenkins, etc.
* Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest
* WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format like the one shown below.

[](https://www.guru99.com/images/report-0057.png)

* TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.





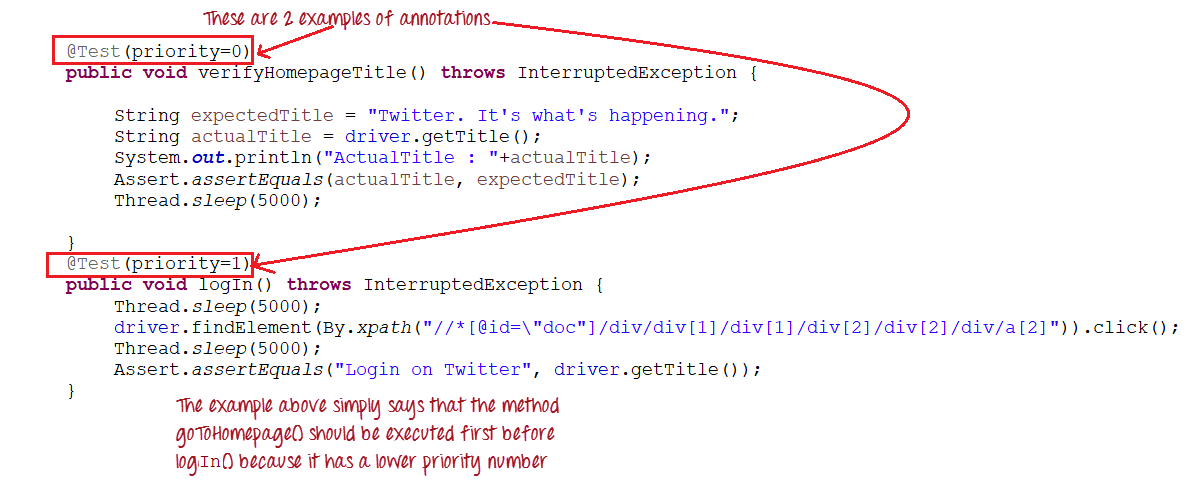
* Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.

## 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

**Annotations in TestNG are lines of code that can control how the method below them will be executed**. They are always preceded by the @ symbol. A very early and quick example is the one shown below.



Annotations will be discussed later in the section named "Annotations used in TestNG,"so it is perfectly ok if you do not understand the above example just yet. It is just important to note for now that annotations in TestNG are easier to code and understand than in JUnit.

**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.

**@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.

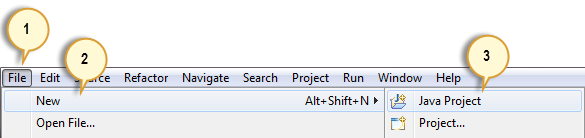
**@Test**: The annotated method is a part of a test case

## First test case using annotations

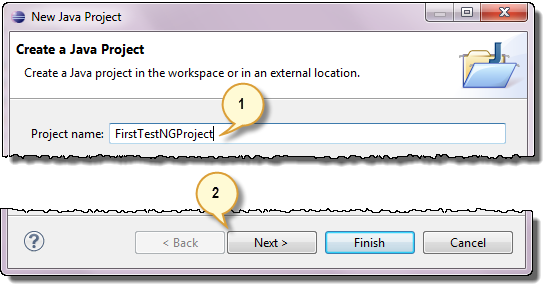
Before we create a test case, we should first setup a new TestNG Project in Eclipse and name it as "060SeleniumWebDriver".

### Setting up a new TestNG Project

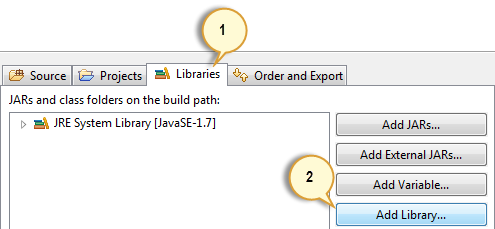
**Step 1:**Click File > New > Java Project

[](https://www.guru99.com/images/step_1-0034.png)

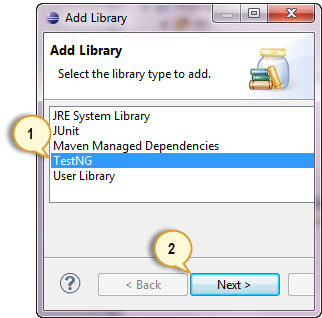
**Step 2:**Type "060SeleniumWebDriver" as the Project Name then click Next.

[](https://www.guru99.com/images/step_2-0037.png)

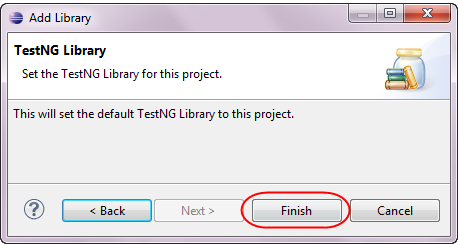
**Step 3:**We will now start to import the TestNG Libraries onto our project. Click on the "Libraries" tab, and then "Add Library…"

[](https://www.guru99.com/images/step_3-0038.png)

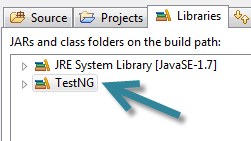
**Step 4:**On the Add Library dialog, choose "TestNG" and click Next.

[](https://www.guru99.com/images/step_4-0039.png)

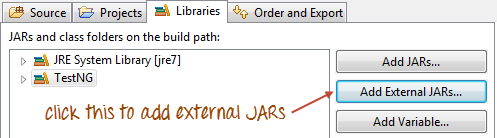
**Step 5:**Click Finish.

[](https://www.guru99.com/images/step_5-0041.png)

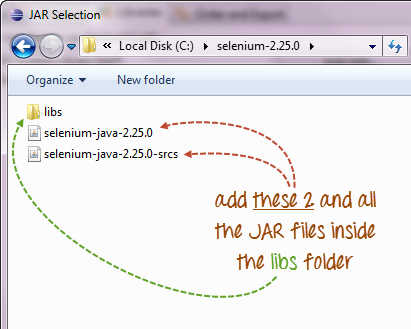
You should notice that TestNG is included on the Libraries list.

[](https://www.guru99.com/images/step_5_b-0040.png)

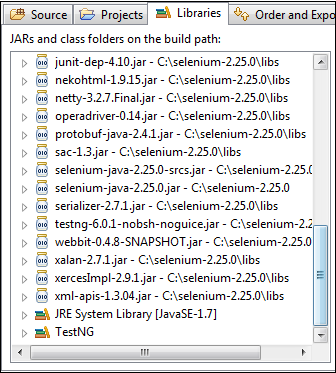
**Step 6:**We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from <http://docs.seleniumhq.org/download/> when we were installing Selenium and Eclipse in the previous chapters.

[](https://www.guru99.com/images/step_6a.png)

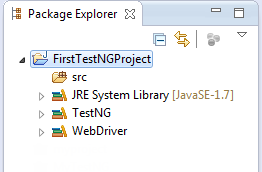
Then, navigate to where you have placed the Selenium JAR files.

[](https://www.guru99.com/images/step_6b.png)

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

[](https://www.guru99.com/images/step_6c.png)

**Step 7:**Click Finish and verify that our 060SeleniumWebDriver is visible on Eclipse's Package Explorer window.

[](https://www.guru99.com/images/step_7-0045.png)

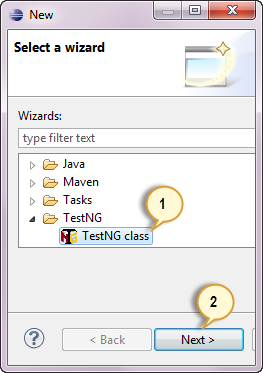
## Creating a New TestNG Test File

Now that we are done setting up our project, let us create a new TestNG file.

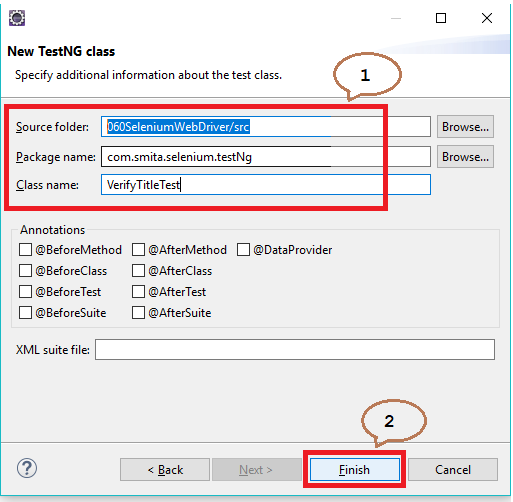
**Step 1:**Right-click on the "src" package folder then choose New > Other…

[](https://www.guru99.com/images/step_1-0029.png)

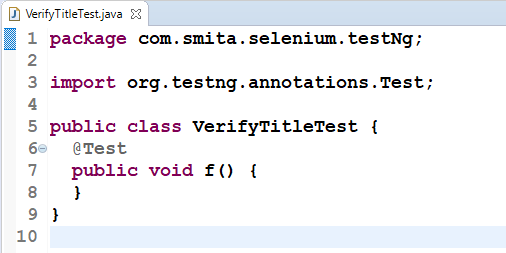
**Step 2:**Click on the TestNG folder and select the "TestNG class" option. Click Next.

[](https://www.guru99.com/images/step_2-0030.png)

**Step 3:**Type the values indicated below on the appropriate input boxes and click Finish. Notice that we have named our Java file as " **VerifyTitleTest**".



Eclipse should automatically create the template for our TestNG file shown below.



### 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.

**package com.smita.selenium.testNg;**

**import org.openqa.selenium.WebDriver;**

**import org.testng.Assert;**

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

**import com.smita.webdriver.util.ChromeDriverUtil;**

**/\*\*@author Smita\*\*/**

**public class VerifyTitleTest {**

**public String baseUrl = "https://twitter.com/";**

**public WebDriver driver;**

**@Test**

**public void verifyHomepageTitle() {**

**System.*out*.println("launching Chrome browser");**

**driver = ChromeDriverUtil.*getDriver*();**

**driver.get(baseUrl);**

**String expectedTitle = "Twitter. It's what's happening.";**

**String actualTitle = driver.getTitle();**

**System.*out*.println("ActualTitle : "+actualTitle);**

**driver.close();**

**Assert.*assertEquals*(actualTitle, expectedTitle);**

**}**

**}**

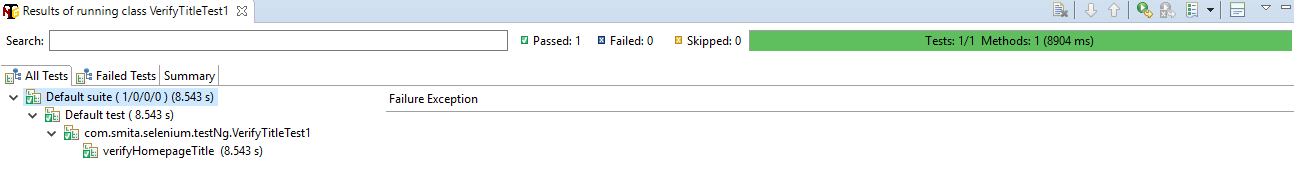
Notice the following.

* 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, we have set the verifyHomepageTitle() method to be our test case, so we placed an '@Test' annotation above it.
* 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.

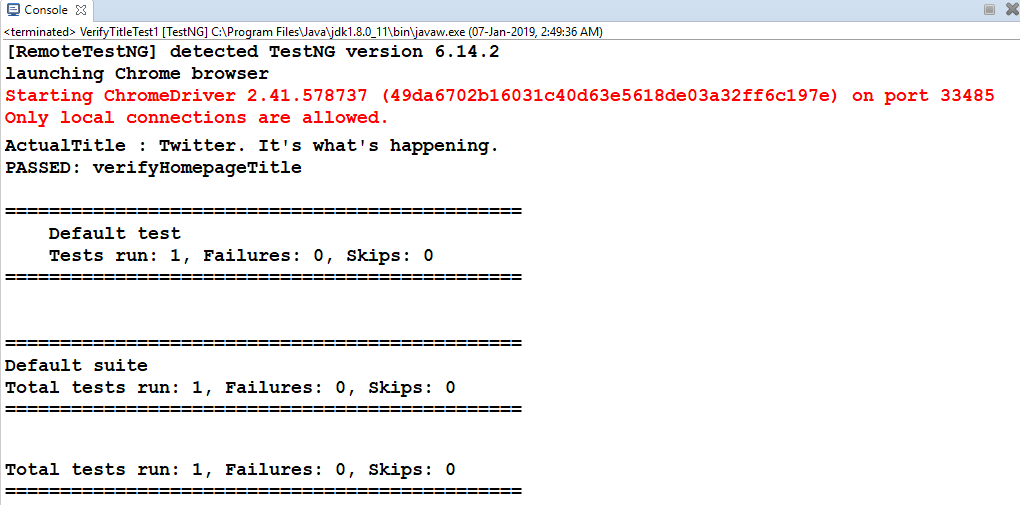
You may have multiple test cases (therefore, multiple @Test annotations) in a single TestNG file. This will be tackled in more detail later in the section "Annotations used in TestNG."

### 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.







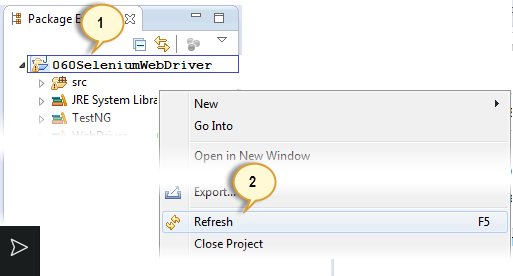
## Checking reports created by TestNG

The Console window in Eclipse gives a text-based report of our test case results while the TestNG Results window gives us a graphical one.

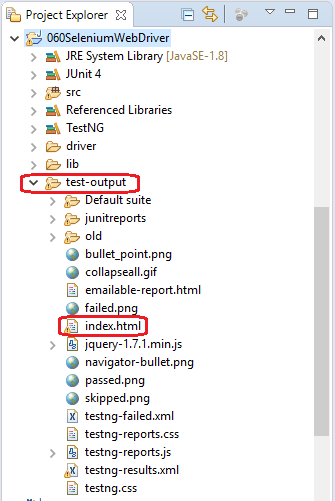
### Generating HTML Reports

TestNG has the ability to generate reports in HTML format.

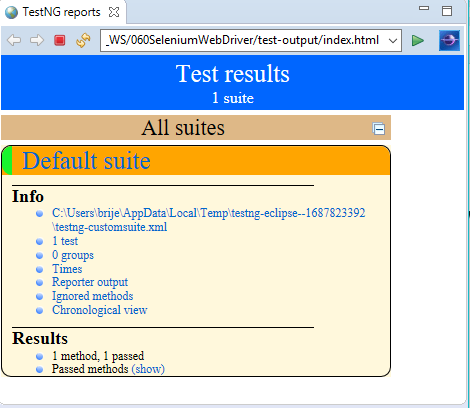
**Step 1:**After running our VerifyHomepageTitle that we created in the previous section, right-click the project name (060SeleniumWebDriver) in the Project Explorer window then click on the "Refresh" option.



**Step 2:**Notice that a "test-output" folder was created. Expand it and look for an index.html file. This HTML file is a report of the results of the most recent test run.



**Step 3:**Double-click on that index.html file to open it within Eclipse's built-in web browser. You can refresh this page any time after you rerun your test by simply pressing F5 just like in ordinary web browsers.

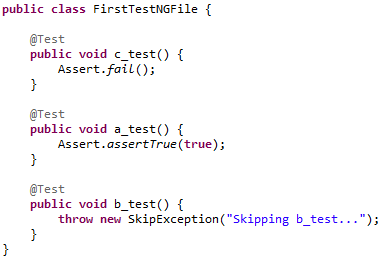


## Annotations used in TestNG

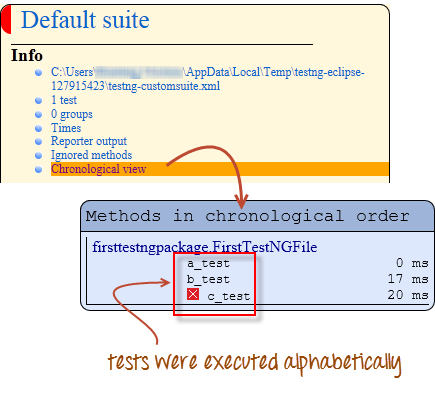
In the previous section, you have been introduced to the @Test annotation. Now, we shall be studying more advanced annotations and their usages.

### 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.

[](https://www.guru99.com/images/code-0008.png)

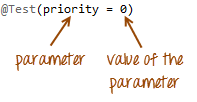
Run this code and on the generated index.html page, click "Chronological view."

[](https://www.guru99.com/images/chronological-0007.png)

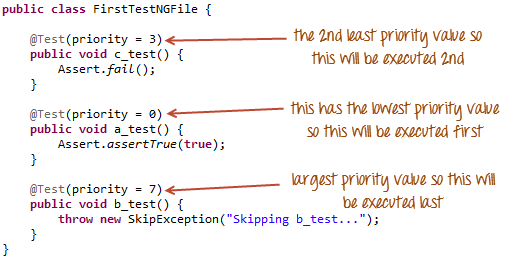
### Parameters

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**.

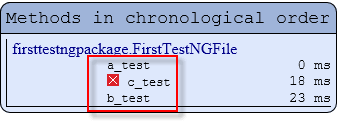
* Parameters require you to assign a value to them. You do.this by placing a "=" next to them, and then followed by the value.
* Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.

[](https://www.guru99.com/images/parameter_and_value-0011.png)

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.

[](https://www.guru99.com/images/whole_code-0013.png)

The TestNG HTML report will confirm that the methods were executed based on the ascending value of priority.

[](https://www.guru99.com/images/result_of_priority-0012.png)

### Multiple Parameters

Aside from "priority," @Test has another parameter called "alwaysRun" which can only be set to either "true" or "false." **To use two or more parameters in a single annotation, separate them with a comma** such as the one shown below.

@Test(priority = 0, alwaysRun = true)

[](https://www.guru99.com/images/3-4-2013_9-06-43_PM.jpg)

@BeforeTest and @AfterTest

|  |  |
| --- | --- |
| **@BeforeTest** | methods under this annotation will be executed **prior to the first test case in the TestNG file**. |
| **@AfterTest** | methods under this annotation will be executed **after all test cases in the TestNG file are executed**. |

Consider the code below.

**package com.smita.selenium.testNg;**

**import org.openqa.selenium.WebDriver;**

**import org.testng.Assert;**

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

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

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

**import com.smita.webdriver.util.ChromeDriverUtil;**

**/\*\*@author Smita\*\*/**

**public class VerifyHomePageTitle {**

**public String baseUrl = "https://twitter.com/";**

**public WebDriver driver;**

**@BeforeTest**

**public void launchBrowser() {**

**System.*out*.println("launching Chrome browser");**

**driver = ChromeDriverUtil.*getDriver*();**

**driver.get(baseUrl);**

**}**

**@Test**

**public void verifyHomepageTitle() {**

**String expectedTitle = "Twitter. It's what's happening.";**

**String actualTitle = driver.getTitle();**

**System.*out*.println("ActualTitle : "+actualTitle);**

**Assert.*assertEquals*(actualTitle, expectedTitle);**

**}**

**@AfterTest**

**public void terminateBrowser() {**

**driver.close();**

**System.*out*.println("Closing Chrome browser");**

**}**

**}**

Applying the logic presented by the table and the code above, we can predict that the sequence by which methods will be executed is:

* 1st - launchBrowser()
* 2nd - verifyHomepageTitle()
* 3rd - terminateBrowser()

**The placement of the annotation blocks can be interchanged without affecting the chronological order by which they will be executed**. For example, try to rearrange the annotation blocks such that your code would look similar to the one below.

**package com.smita.selenium.testNg;**

**import org.openqa.selenium.WebDriver;**

**import org.testng.Assert;**

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

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

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

**import com.smita.webdriver.util.ChromeDriverUtil;**

**/\*\*@author Smita\*\*/**

**public class VerifyHomePageTitle2 {**

**public String baseUrl = "https://twitter.com/";**

**public WebDriver driver;**

**@AfterTest //Jumbled**

**public void terminateBrowser() {**

**driver.close();**

**System.*out*.println("Closing Chrome browser");**

**}**

**@BeforeTest //Jumbled**

**public void launchBrowser() {**

**System.*out*.println("launching Chrome browser");**

**driver = ChromeDriverUtil.*getDriver*();**

**driver.get(baseUrl);**

**}**

**@Test //Jumbled**

**public void verifyHomepageTitle() {**

**String expectedTitle = "Twitter. It's what's happening.";**

**String actualTitle = driver.getTitle();**

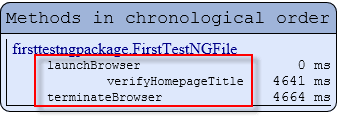
**System.*out*.println("ActualTitle : "+actualTitle);**

**Assert.*assertEquals*(actualTitle, expectedTitle);**

**}**

**}**

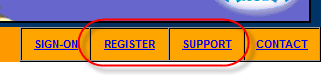
Run the code above and notice that

[](https://www.guru99.com/images/result-0006.png)

@BeforeMethod and @AfterMethod

|  |  |
| --- | --- |
| **@BeforeMethod** | methods under this annotation will be executed **prior to each method in each test case**. |
| **@AfterMethod** | methods under this annotation will be executed **after each method in each test case.** |

In Mercury Tours, suppose we like to verify the titles of the target pages of the two links below.

[](https://www.guru99.com/images/3_links_from_aut-0001.png)

The flow of our test would be:

* Go to the homepage and verify its title.
* Click SignUp and verify the title of its target page.
* Go back to the homepage and verify if it still has the correct title.
* Click HelpCenter and verify the title of its target page.
* Go back to the homepage and verify if it still has the correct title.

The code below illustrates how @BeforeMethod and @AfterMethod are used to efficiently execute the scenario mentioned above.

**package com.smita.selenium.testNg;**

**import org.openqa.selenium.By;**

**import org.openqa.selenium.WebDriver;**

**import org.testng.Assert;**

**import org.testng.annotations.AfterMethod;**

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

**import org.testng.annotations.BeforeMethod;**

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

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

**import com.smita.webdriver.util.ChromeDriverUtil;**

**/\*\* @author Smita \*\*/**

**public class VerifyTitleTestPriority {**

**public String baseUrl = "https://twitter.com/";**

**public WebDriver driver;**

**@BeforeTest**

**public void launchBrowser() {**

**System.*out*.println("launching Chrome browser");**

**driver = ChromeDriverUtil.*getDriver*();**

**driver.get(baseUrl);**

**}**

**@BeforeMethod**

**public void verifyHomepageTitle() throws InterruptedException {**

**String expectedTitle = "Twitter. It's what's happening.";**

**String actualTitle = driver.getTitle();**

**System.*out*.println("On Home Page : " + actualTitle);**

**Assert.*assertEquals*(actualTitle, expectedTitle);**

**Thread.*sleep*(5000);**

**}**

**@Test(priority = 0)**

**public void logIn() throws InterruptedException {**

**Thread.*sleep*(5000);**

**driver.findElement(By.*xpath*("//\*[@id=\"doc\"]/div/div[1]/div[1]/div[2]/div[2]/div/a[2]")).click();**

**String actualTitle = driver.getTitle();**

**System.*out*.println("On Login Page : " + actualTitle);**

**Thread.*sleep*(5000);**

**Assert.*assertEquals*("Login on Twitter", actualTitle);**

**}**

**@Test(priority = 1)**

**public void helpCenter() throws InterruptedException {**

**Thread.*sleep*(5000);**

**driver.findElement(By.*xpath*("//\*[@id=\"doc\"]/div/div[3]/ul/li[2]/a")).click();**

**String actualTitle = driver.getTitle();**

**System.*out*.println("On Help Page : " + actualTitle);**

**Thread.*sleep*(5000);**

**Assert.*assertEquals*("Help Center", actualTitle);**

**}**

**@AfterMethod**

**public void goBackToHomepage() throws InterruptedException {**

**Thread.*sleep*(5000);**

**System.*out*.println("Going Back to Home Page....");**

**driver.navigate().back();**

**}**

**@AfterTest**

**public void terminateBrowser() {**

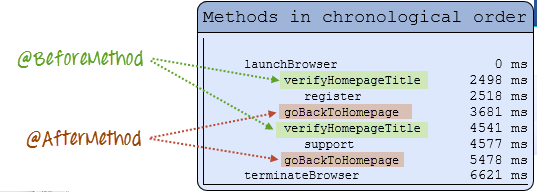
**driver.close();**

**System.*out*.println("Closing Chrome browser");**

**}**

**}**

After executing this test, your TestNG should report the following sequence.



Simply put, @BeforeMethod should contain methods that you need to run **before** each test case while @AfterMethod should contain methods that you need to run **after** each test case.