TestNG

What is TestNG?

So far we had been doing Selenium tests without generating a proper format for the test results. From this point on, we shall tackle how to make these reports using a test framework called TestNG.

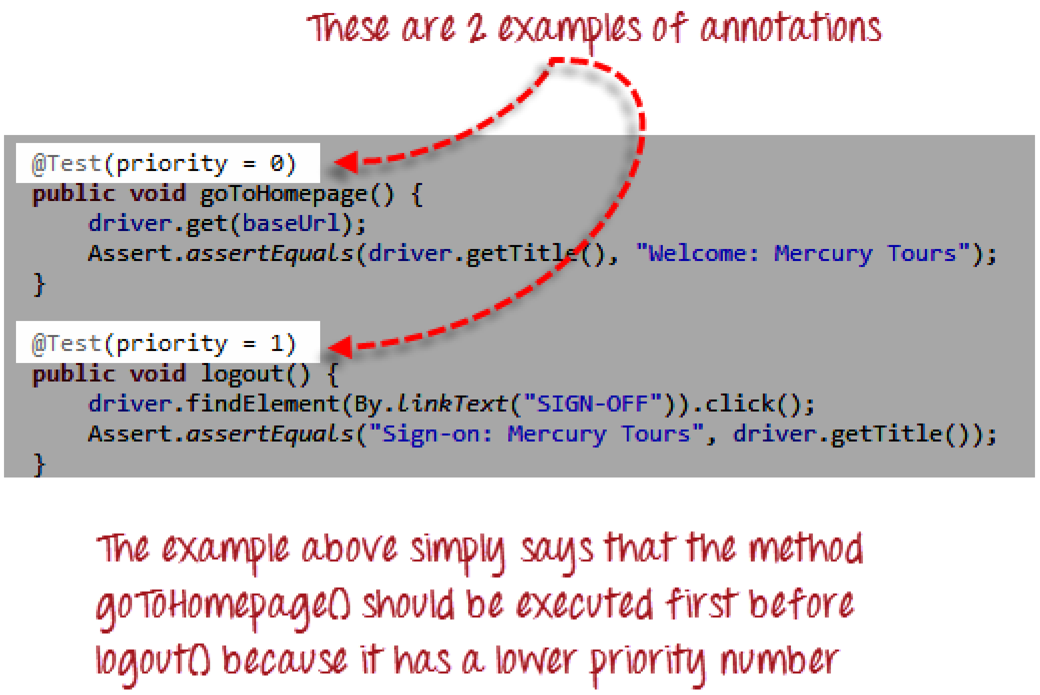
TestNG is a testing framework that overcomes the limitations of another popular testing framework called JUnit. The “NG” means “Next Generation”. Most Selenium users use this more than JUnit because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium.

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.


These are 2 examples of annotations
The example above simply says that the method goToHomepage() should be executed first before logout() because it has a lower priority number

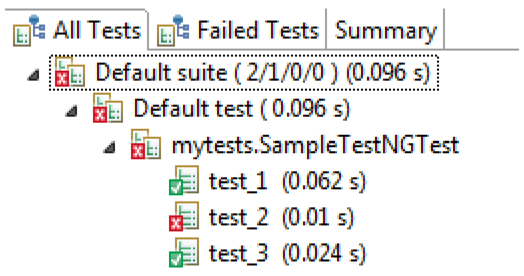
```
@Test(priority = 0)
public void goToHomepage() {
    driver.get(baseUrl);
    Assert.assertEquals(driver.getTitle(), "Welcome: Mercury Tours");
}

@Test(priority = 1)
public void logout() {
    driver.findElement(By.linkText("SIGN-OFF")).click();
    Assert.assertEquals("Sign-on: Mercury Tours", driver.getTitle());
}
```

The ability to run tests in parallel is available in TestNG but not in JUnit, so it is the more preferred framework of testers using Selenium Grid.

Why do we need TestNG in Selenium?

TestNG can generate reports based on our Selenium test results.

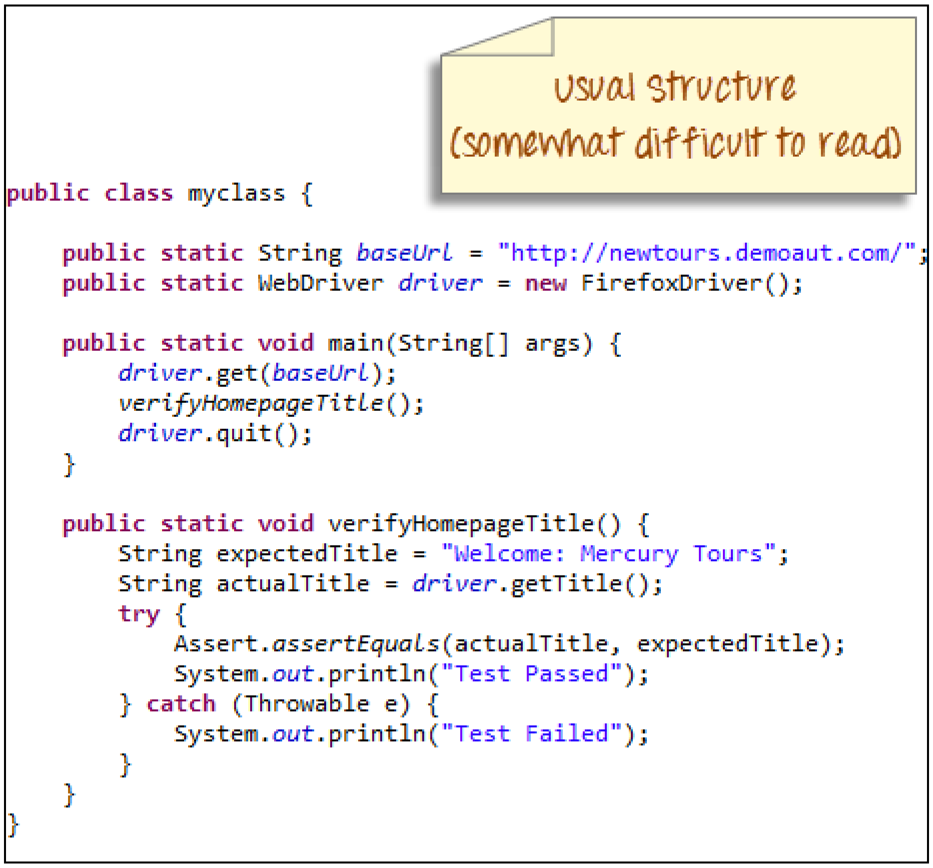
- WebDriver has no native mechanism for generating reports.
- TestNG can generate the report in a readable format like the one shown below.



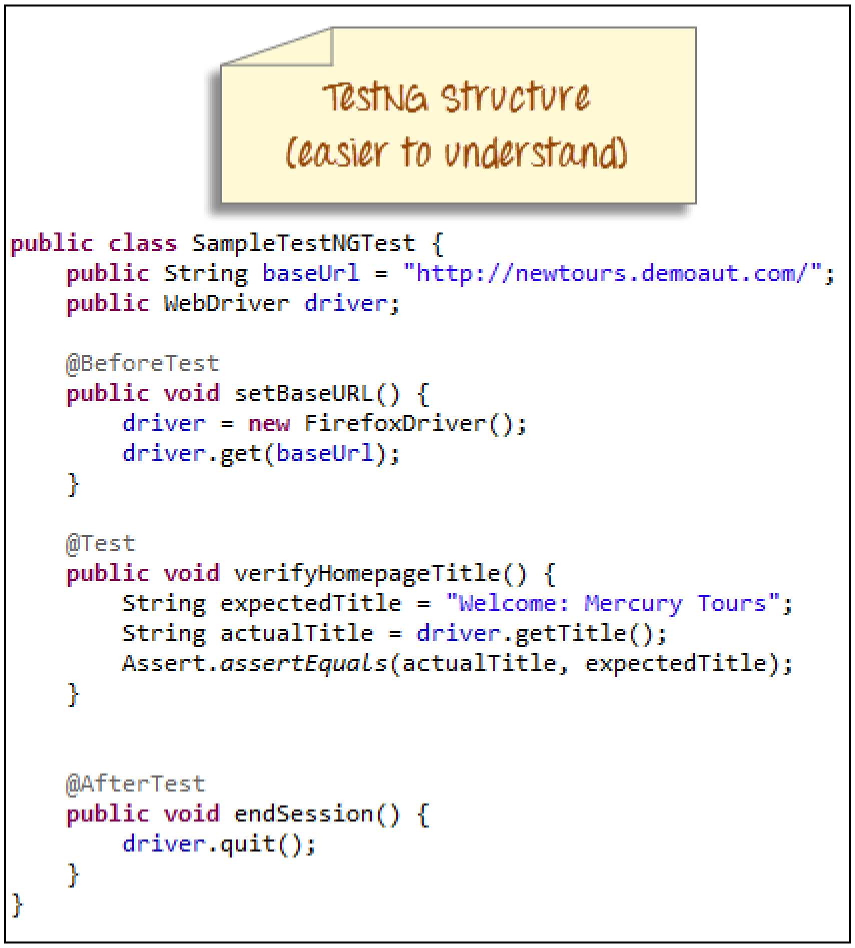
```
Default suite ( 2/1/0/0 ) (0.096 s)
    Default test ( 0.096 s)
        mytests.SampleTestNGTest
            test_1  (0.062 s)
            test_2  (0.01 s)
            test_3  (0.024 s)
```

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.



```
public class myclass {

    public static String baseUrl = "http://newtours.demoaut.com/";
    public static WebDriver driver = new FirefoxDriver();

    public static void main(String[] args) {
        driver.get(baseUrl);
        verifyHomepageTitle();
        driver.quit();
    }

    public static void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        try {
            Assert.assertEquals(actualTitle, expectedTitle);
            System.out.println("Test Passed");
        } catch (Throwable e) {
            System.out.println("Test Failed");
        }
    }
}
```



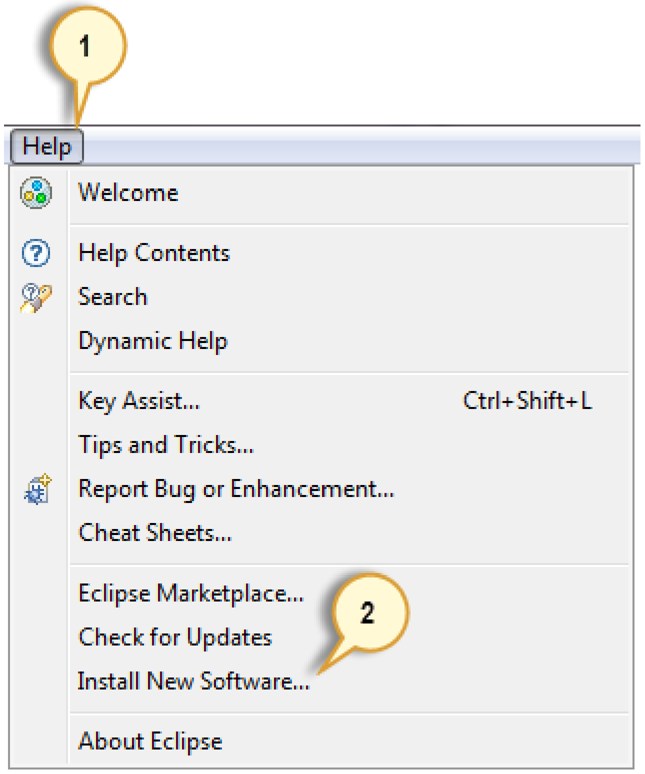
```
public class SampleTestNGTest {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver;

    @BeforeTest
    public void setBaseURL() {
        driver = new FirefoxDriver();
        driver.get(baseUrl);
    }

    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }


    @AfterTest
    public void endSession() {
        driver.quit();
    }
}
```

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

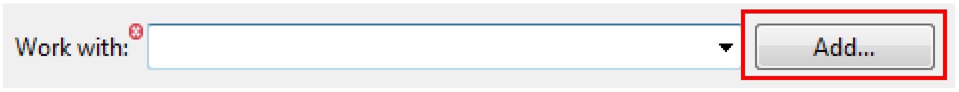
<u>Installing TestNG in Eclipse</u>

Step 1

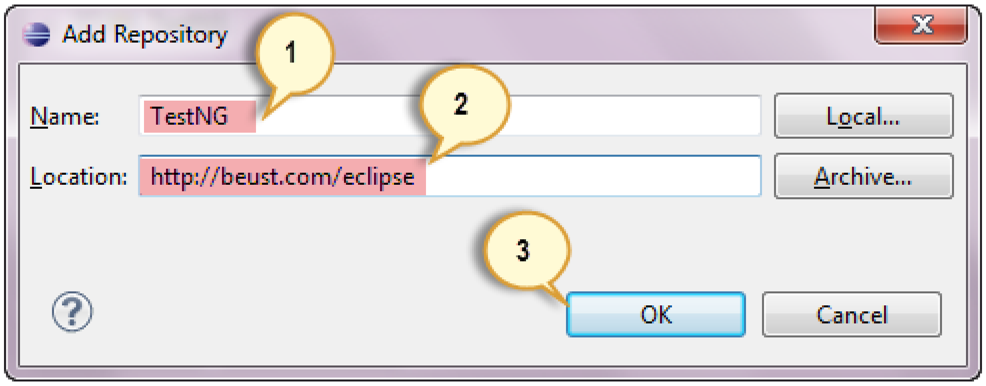
- Launch Eclipse.
- On the menu bar, click Help.
- Choose the “Install New Software…” option.



Step 2
In the Install dialog box, click the Add button



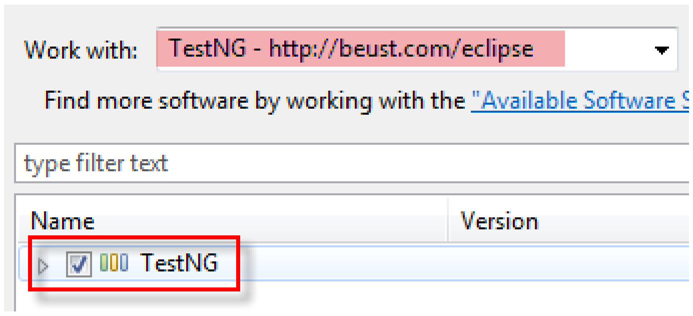
Step 3
1. In "Name", type TestNG.
2. In "Location", type http://beust.com/eclipse.
3. Click OK



Name: TestNG
Location: http://beust.com/eclipse

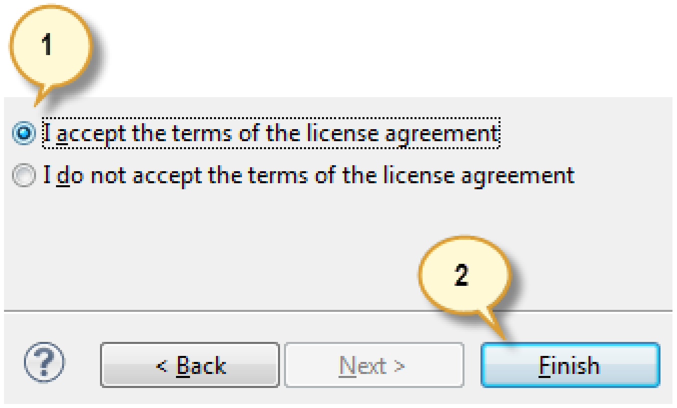
Step 4
- Notice that "TestNG - http://beust.com/eclipse" was populated onto the "Work with:" textbox.

- Check the “TestNG” check box as shown below, then click Next.
- Note: In the latest Eclipse (Kepler) you don't have a checkbox for testNG, instead you click on question mark (help) icon which will open up the form, and you can select all and installation will continue as for the remaining instructions. Thanks Jana for the tip!

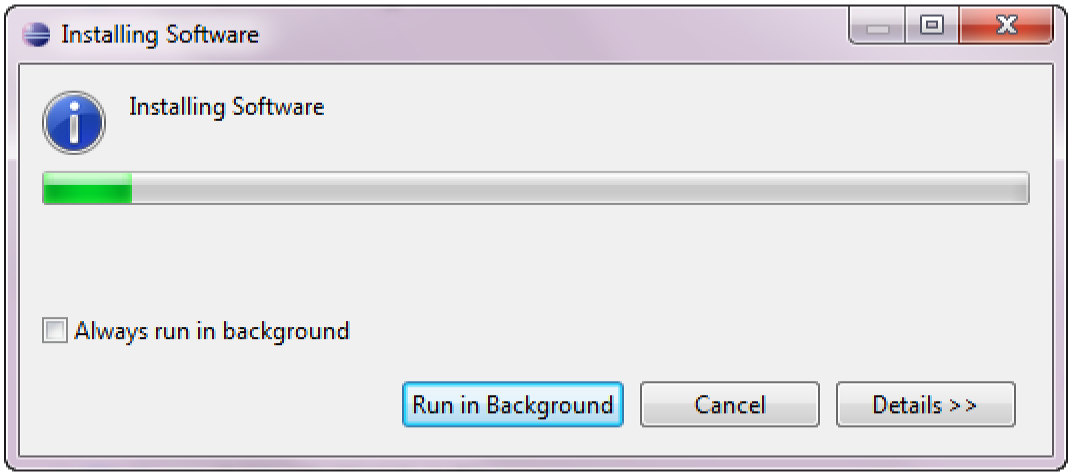

Work with:
TestNG - http://beust.com/eclipse
Find more software by working with the "Available Software S
type filter text
Name
Version
TestNG

Step 5

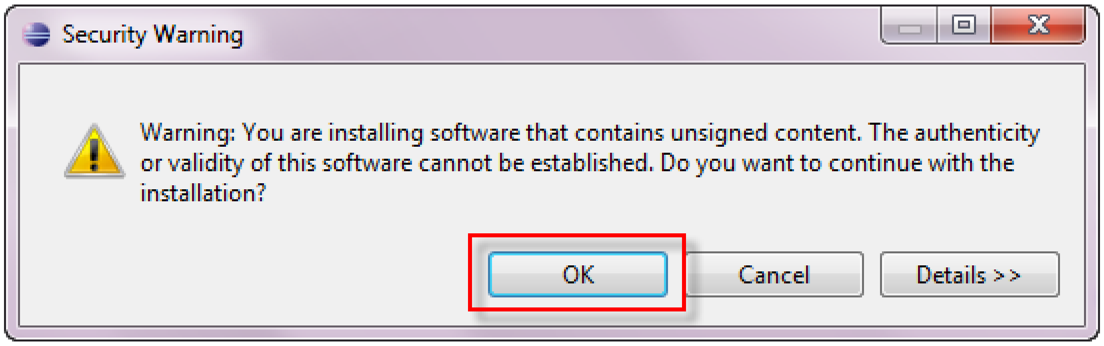
- Click Next again on the succeeding dialog box until you reach the License Agreement dialog.
- Click “I accept the terms of the license agreement” then click Finish.


1
I accept the terms of the license agreement
I do not accept the terms of the license agreement
2
< Back
Next >
Finish

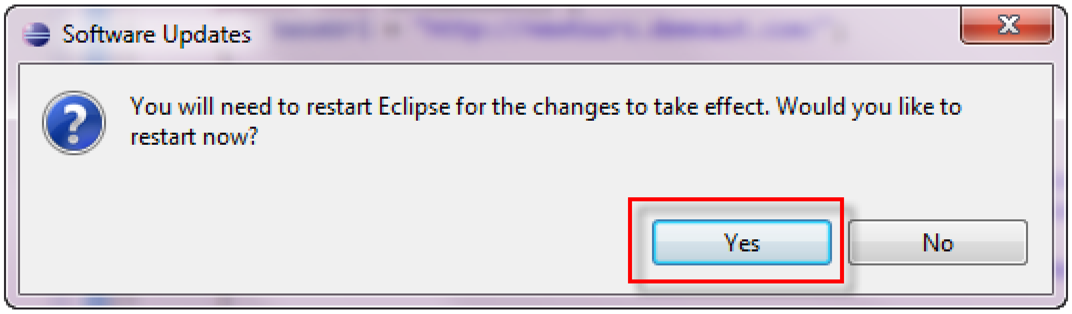
Step 6
Wait for the installation to finish



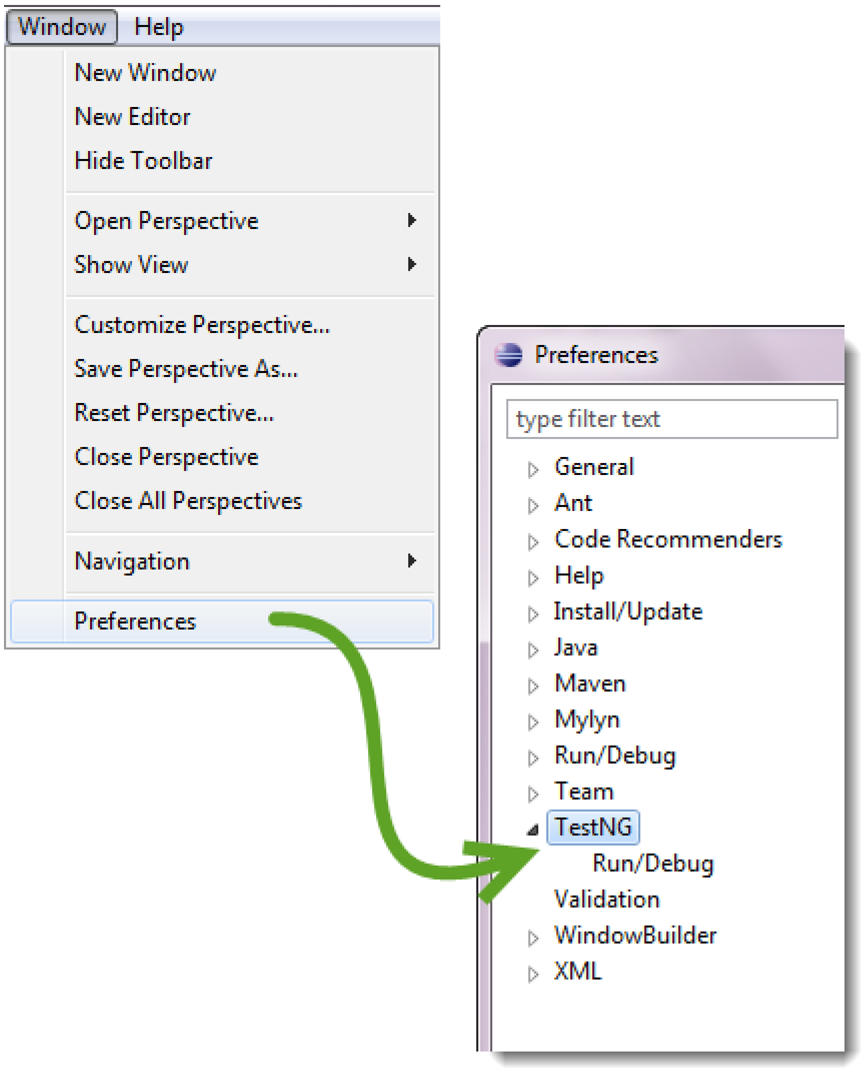
If you encounter a Security warning, just click OK


Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

Step 7
When Eclipse prompts you for a restart, just click Yes.

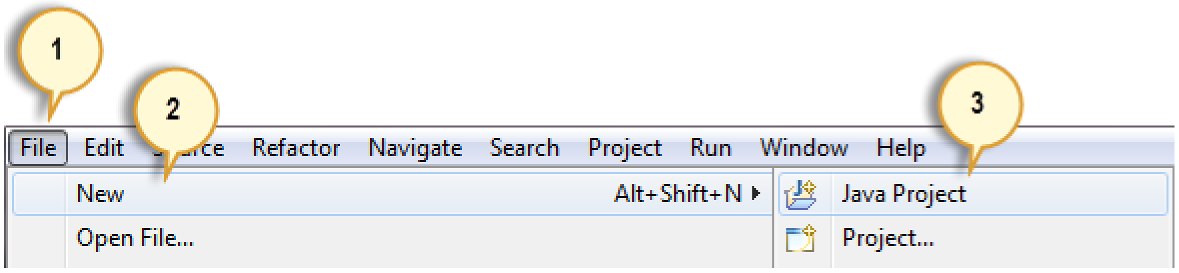

You will need to restart Eclipse for the changes to take effect. Would you like to restart now?

Step 8
After restart, verify if TestNG was indeed successfully installed. Click Window > Preferences and see if TestNG is included on the Preferences list.

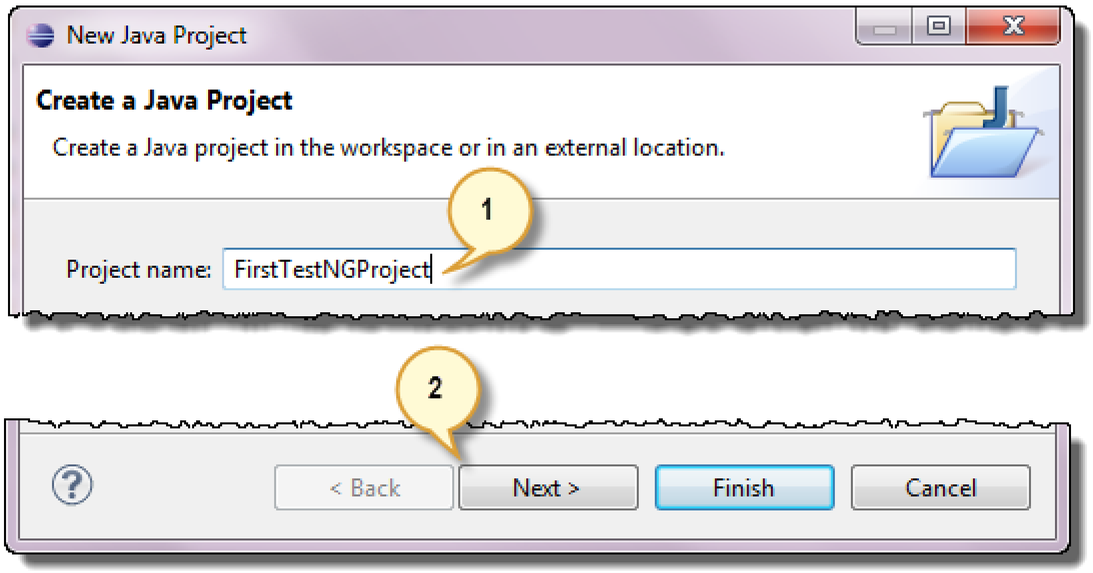


First test case using annotations

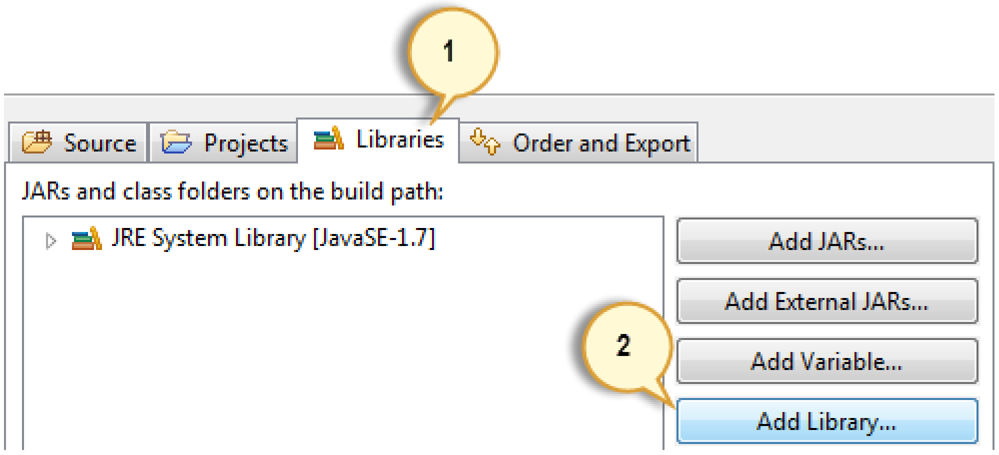
Before we create a test case, we should first setup a new TestNG Project in Eclipse and name it as "FirstTestNGProject".
Setting up a new TestNG Project
Step 1
Click File > New > Java Project



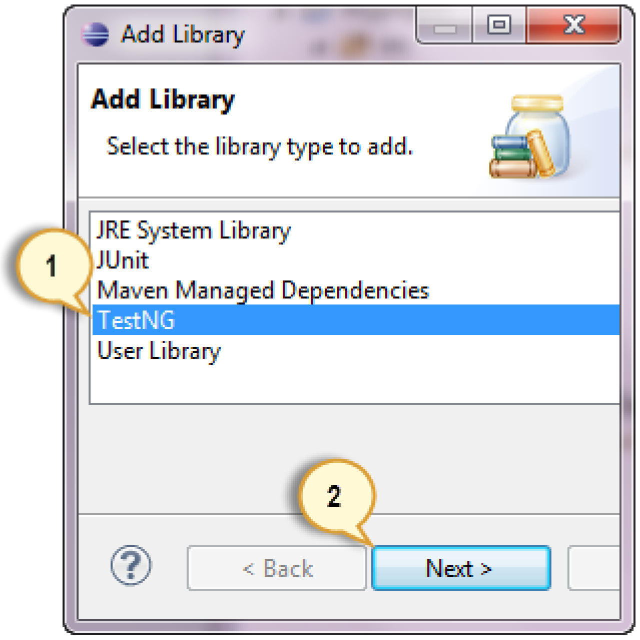
Step 2
Type “FirstTestNGProject” as the Project Name then click Next.


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
1
Project name: FirstTestNGProject
2
< Back
Next >
Finish
Cancel

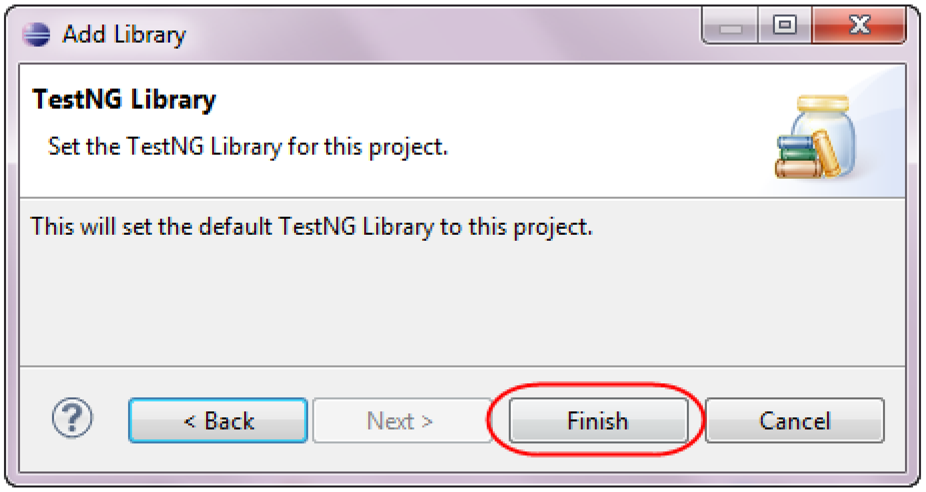
Step 3
We will now start to import the TestNG Libraries onto our project. Click on the “Libraries” tab, and then “Add Library…”



Step 4
On the Add Library dialog, choose “TestNG” and click Next.

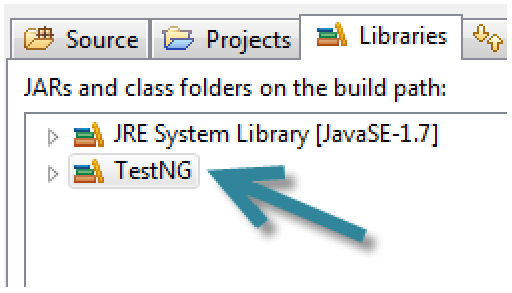


Step 5
Click Finish.

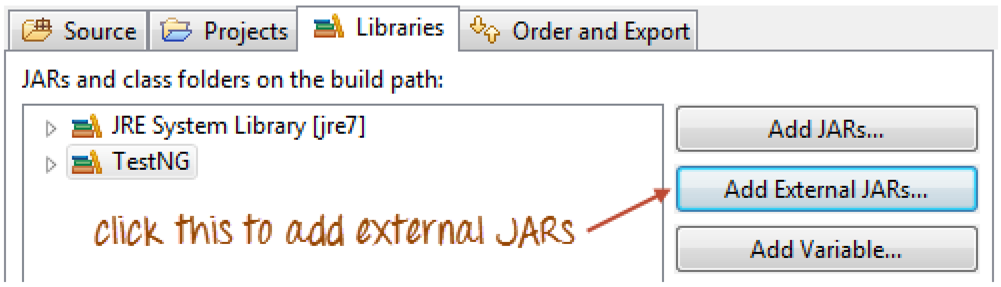


This will set the default TestNG Library to this project.

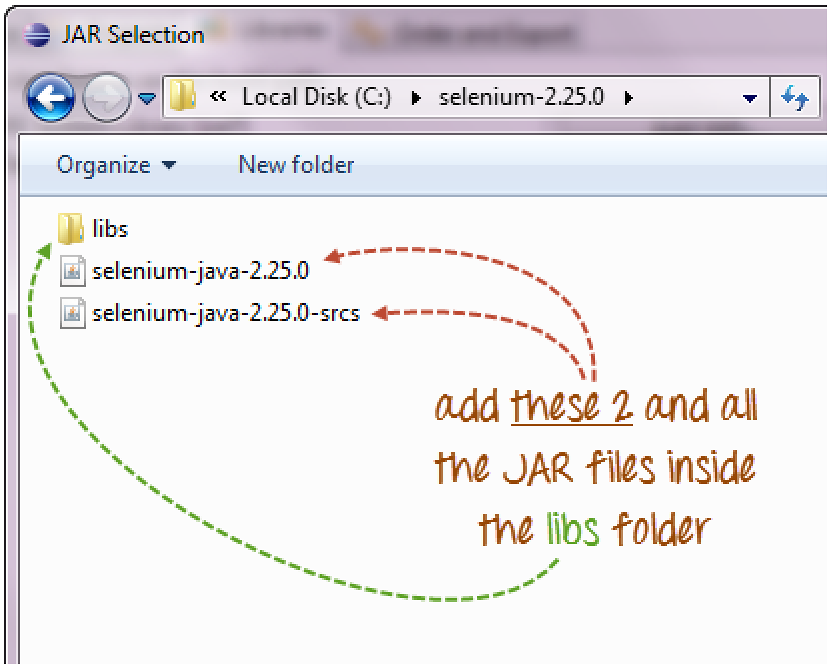
You should notice that TestNG is included on the Libraries list.


Source
Projects
Libraries
JARs and class folders on the build path:
JRE System Library [JavaSE-1.7]
TestNG

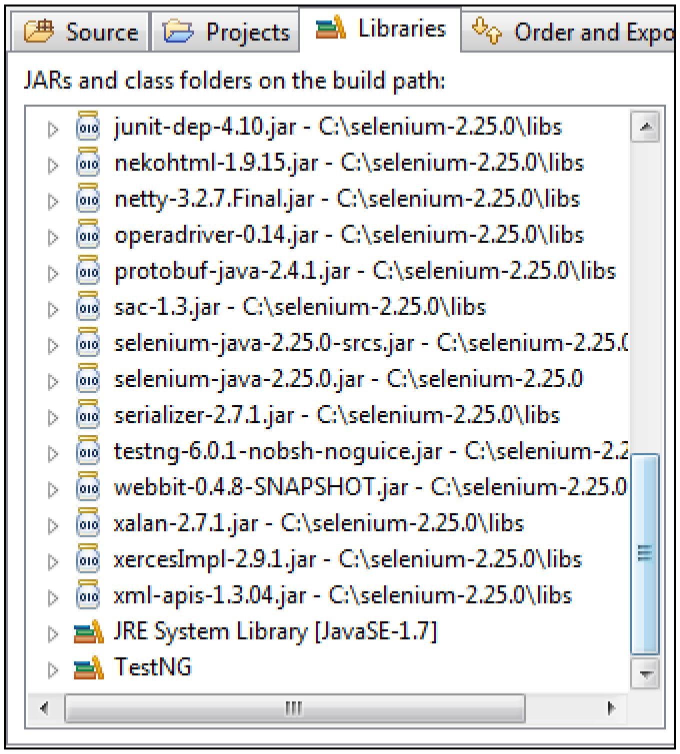
Step 6
We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from http://docs.seleniumhq.org/download/ when we were installing Selenium and Eclipse in the previous chapters.


Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
JRE System Library [jre7]
TestNG
Add JARs...
Add External JARs...
Add Variable...
click this to add external JARs

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

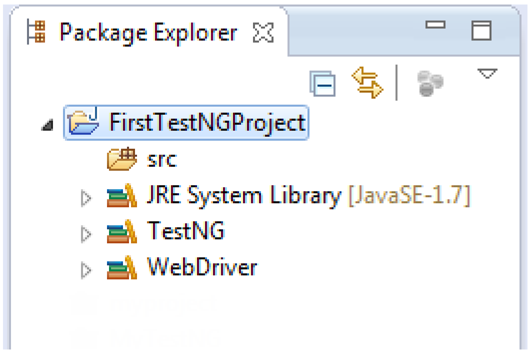
After adding the external JARs, your screen should look like this.



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

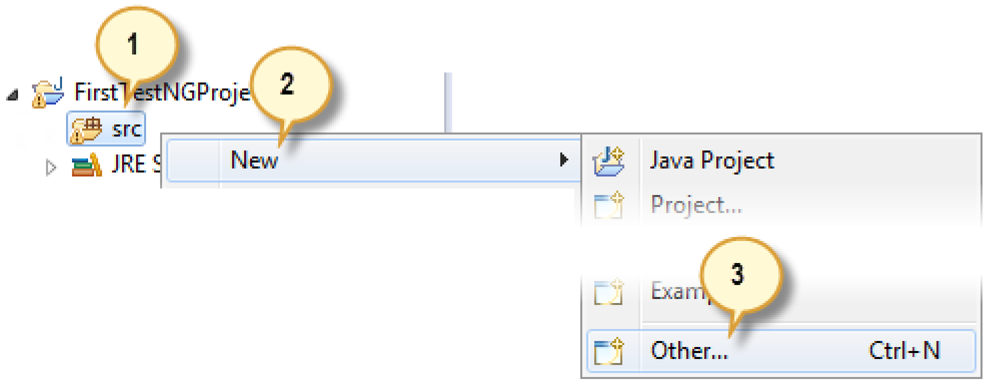
Step 7
Click Finish and verify that our FirstTestNGProject is visible on Eclipse’s Package Explorer window.



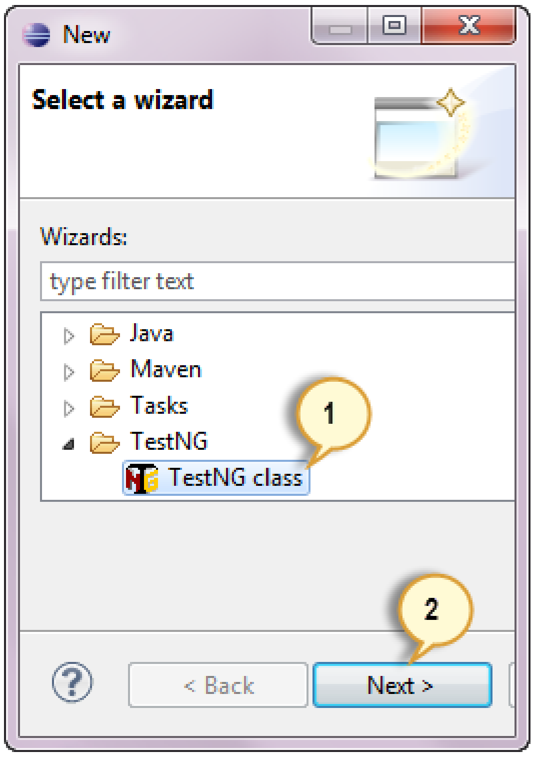
FirstTestNGProject
src
JRE System Library [JavaSE-1.7]
TestNG
WebDriver

Creating a New TestNG Test File

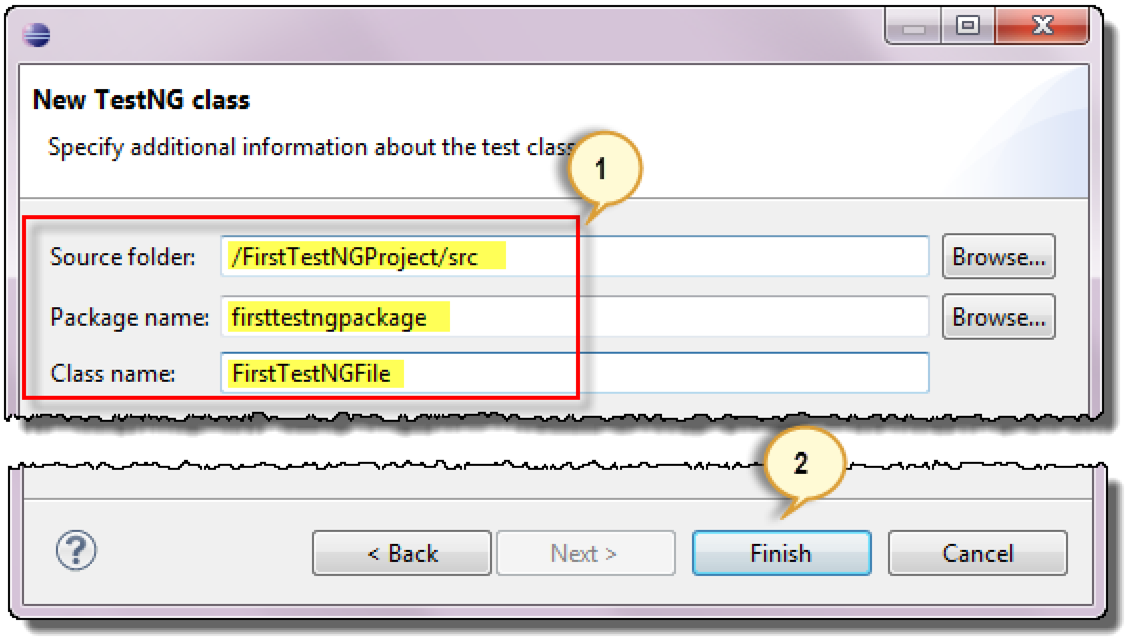
Now that we are done setting up our project, let us create a new TestNG file.
Step 1
Right-click on the “src” package folder then choose New > Other…



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



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


New TestNG class

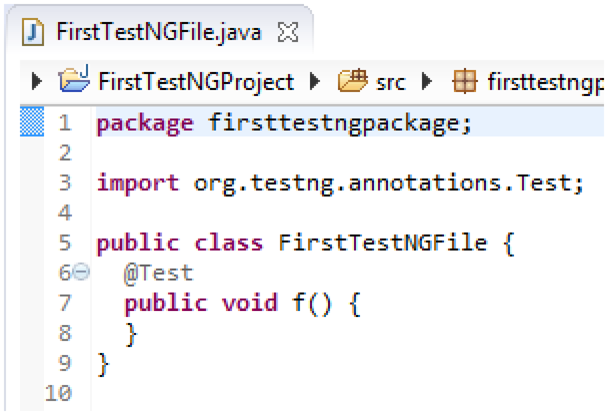
Specify additional information about the test class

Source folder: /FirstTestNGProject/src

Package name: firsttestngpackage

Class name: FirstTestNGFile

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



```
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.

```java
package firsttestngpackage;

import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class FirstTestNGFile {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver = new FirefoxDriver();

    @Test
    public void verifyHomepageTitle() {
        driver.get(baseUrl);
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
        driver.quit();
    }
}
```

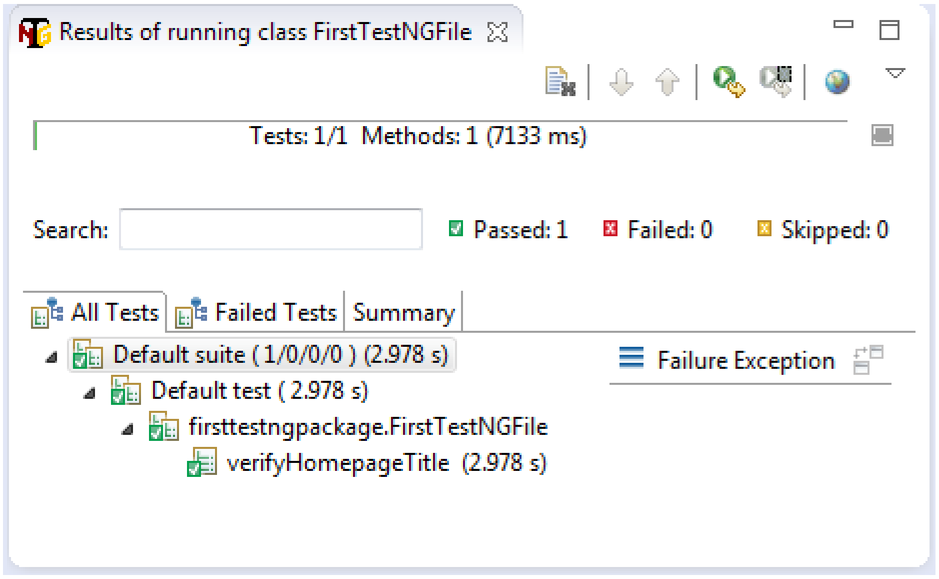
Notice the following.

- 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".

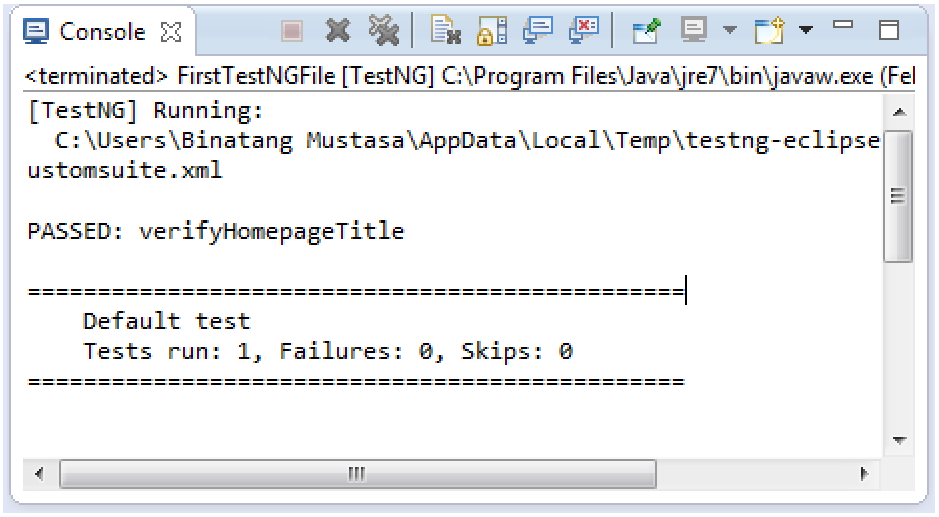
Running the Test

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.

TestNG Results Window


Results of running class FirstTestNGFile
Tests: 1/1 Methods: 1 (7133 ms)
Search:
Passed: 1
Failed: 0
Skipped: 0
All Tests
Failed Tests
Summary
Default suite (1/0/0/0) (2.978 s)
Default test (2.978 s)
firsttestngpackage.FirstTestNGFile
verifyHomepageTitle (2.978 s)
Failure Exception

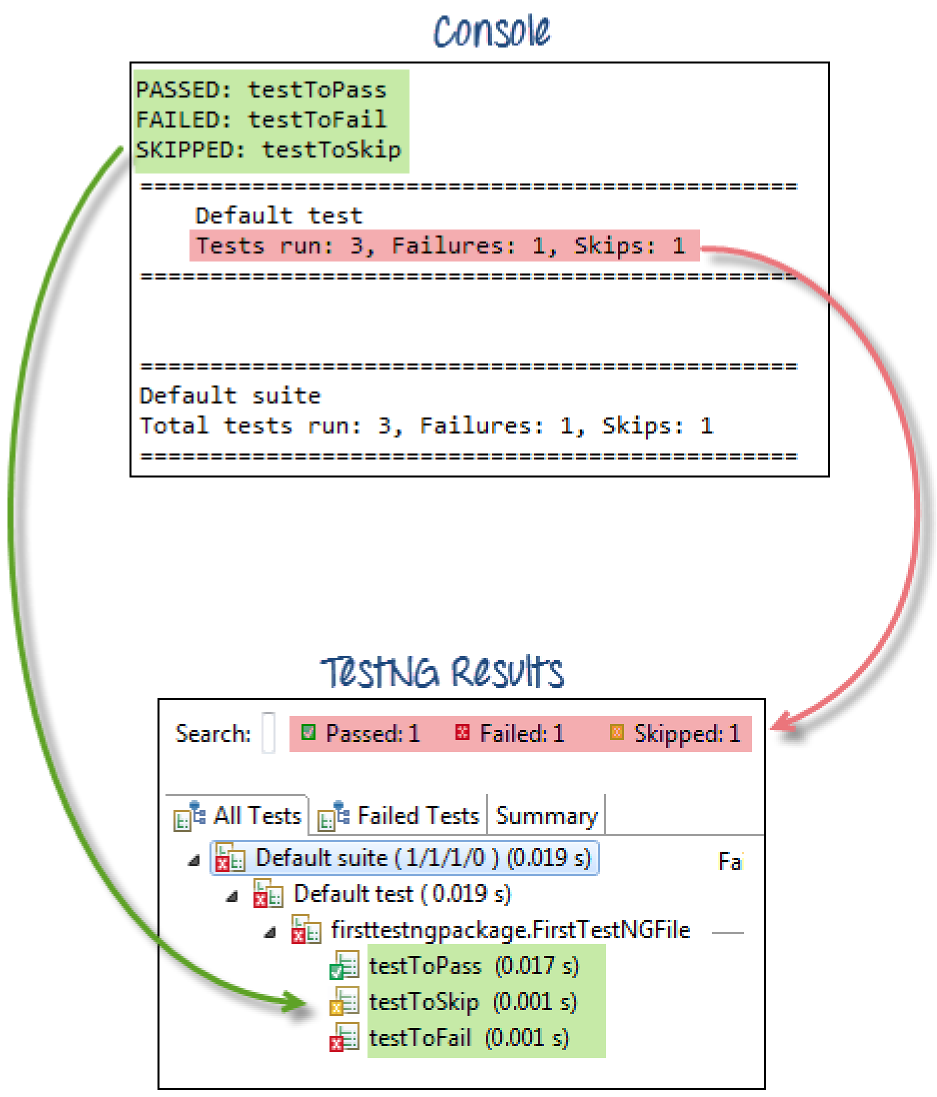
Console Window



```
<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
===============================================
```

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.



```
PASSED: testToPass
FAILED: testToFail
SKIPPED: testToSkip
===============================================
    Default test
    Tests run: 3, Failures: 1, Skips: 1
===============================================


===============================================
Default suite
Total tests run: 3, Failures: 1, Skips: 1
===============================================
```

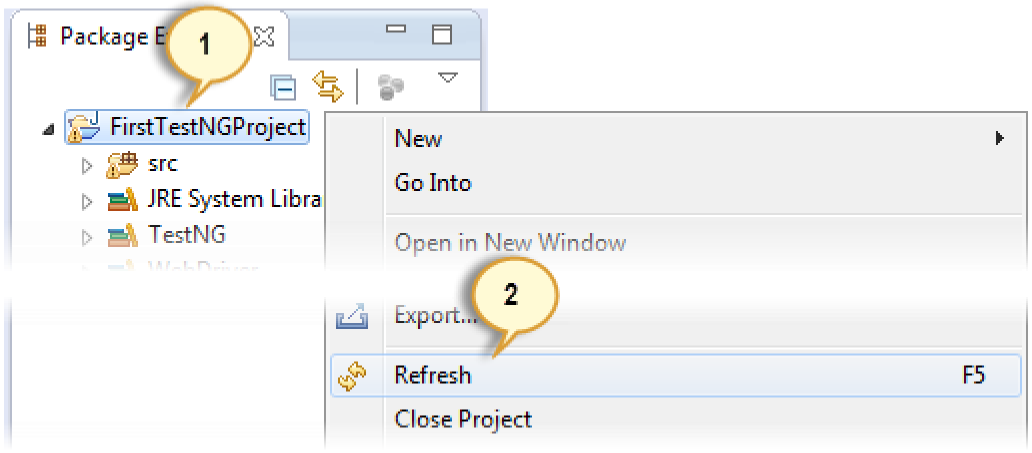
```
Default suite ( 1/1/1/0 ) (0.019 s)
  Default test ( 0.019 s)
    firsttestngpackage.FirstTestNGFile
      testToPass  (0.017 s)
      testToSkip  (0.001 s)
      testToFail  (0.001 s)
```

Generating HTML Reports

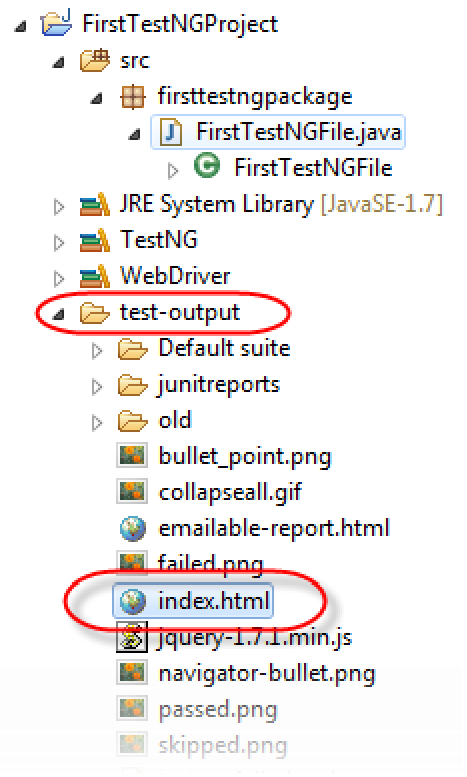
TestNG has the ability to generate reports in HTML format.

Step 1

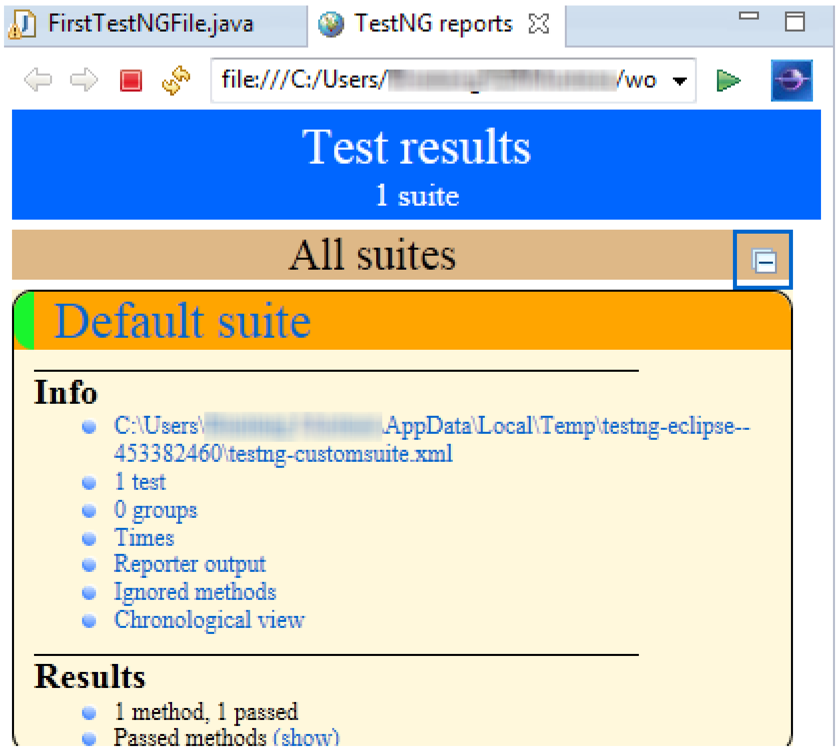
After running our FirstTestNGFile that we created in the previous section, right-click the project name (FirstTestNGProject) in the Project Explorer window then click on the “Refresh” option.



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.



Test results
1 suite

All suites

Default suite

Info

- C:\Users\[redacted]AppData\Local\Temp\testng-eclipse--453382460\testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results

- 1 method, 1 passed
- Passed methods (show)

Annotations used in TestNG

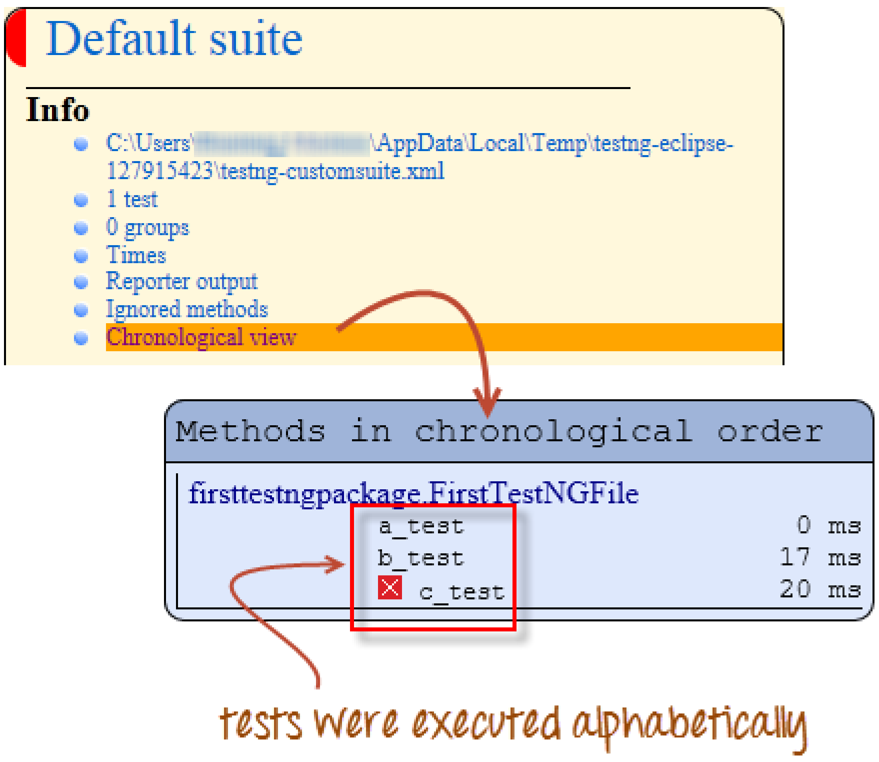
In the previous section, you have been introduced to the @Test annotation. Now, we shall be studying more advanced annotations and their usages.

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.

```java
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...");
    }
}
```

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



Default suite

Info

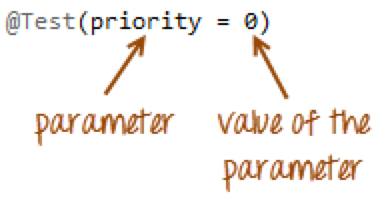
- C:\Users\[redacted]\AppData\Local\Temp\testng-eclipse-127915423\testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

```
Methods in chronological order

firsttestngpackage.FirstTestNGFile
    a_test                    0 ms
    b_test                   17 ms
    c_test                   20 ms
```

Parameters

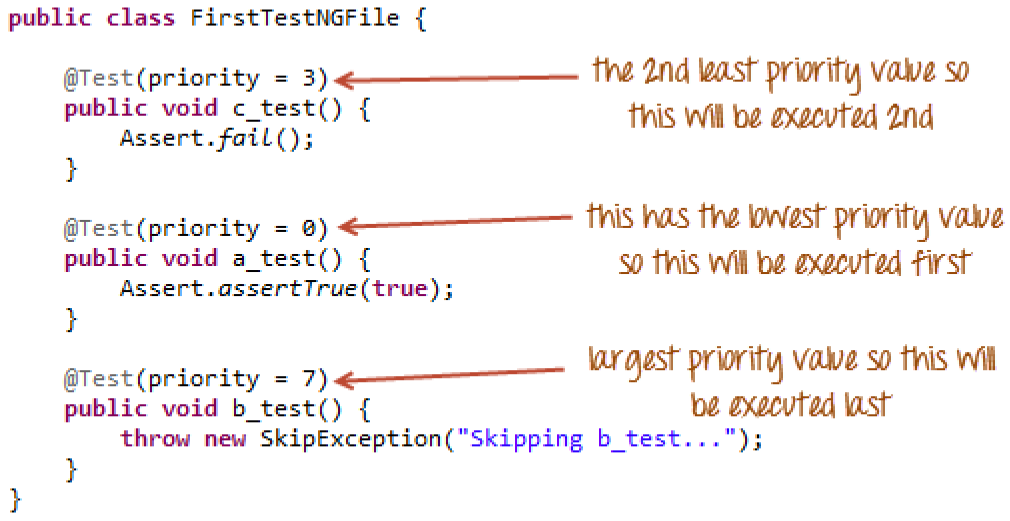
If you want the methods to be executed in a different order, use the parameter "priority". **Parameters are keywords that modify the annotation's function**.

- 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.



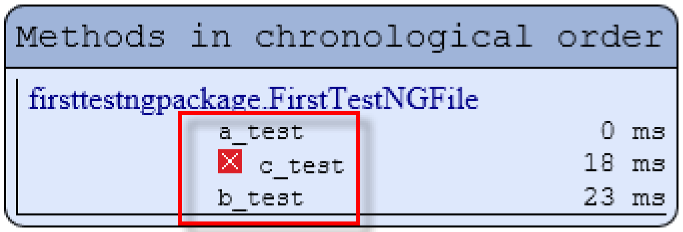
```
public class FirstTestNGFile {

    @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

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



```
Methods in chronological order
firsttestngpackage.FirstTestNGFile
        a_test          0 ms
      ☒ c_test         18 ms
        b_test         23 ms
```

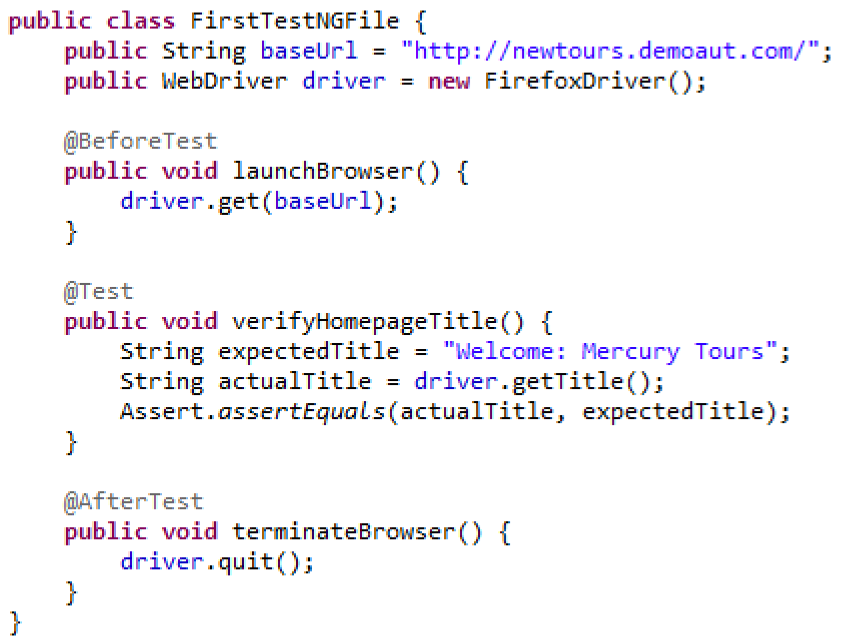
Multiple Parameters

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.

@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.

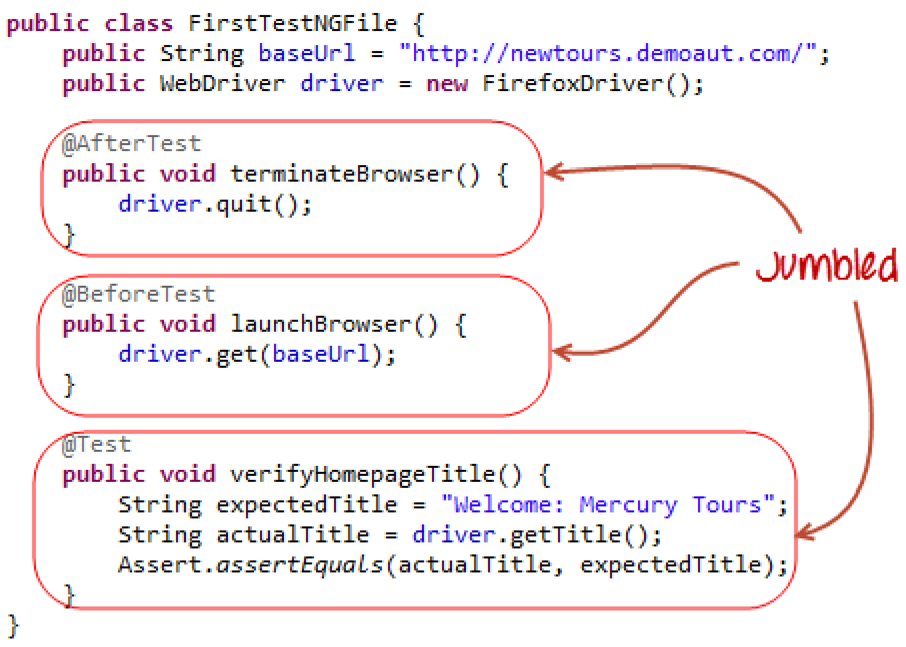


```
public class FirstTestNGFile {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public void launchBrowser() {
        driver.get(baseUrl);
    }

    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }

    @AfterTest
    public void terminateBrowser() {
        driver.quit();
    }
}
```

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

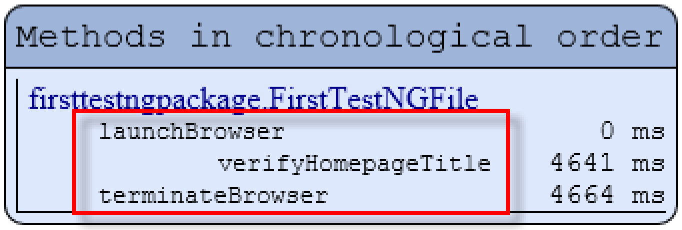
- 1st – launchBrowser()
- 2nd – verifyHomepageTitle()
- 3rd – terminateBrowser()

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.



```
public class FirstTestNGFile {
    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver = new FirefoxDriver();

    @AfterTest
    public void terminateBrowser() {
        driver.quit();
    }

    @BeforeTest
    public void launchBrowser() {
        driver.get(baseUrl);
    }

    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle);
    }
}
```

Run the code above and notice that

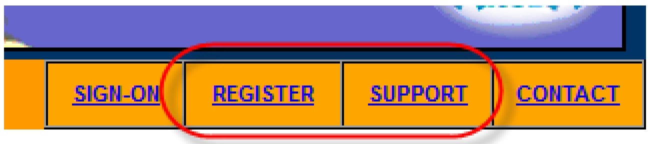


```
Methods in chronological order
firsttestngpackage.FirstTestNGFile
  launchBrowser                          0 ms
          verifyHomepageTitle         4641 ms
  terminateBrowser                    4664 ms
```

@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.**

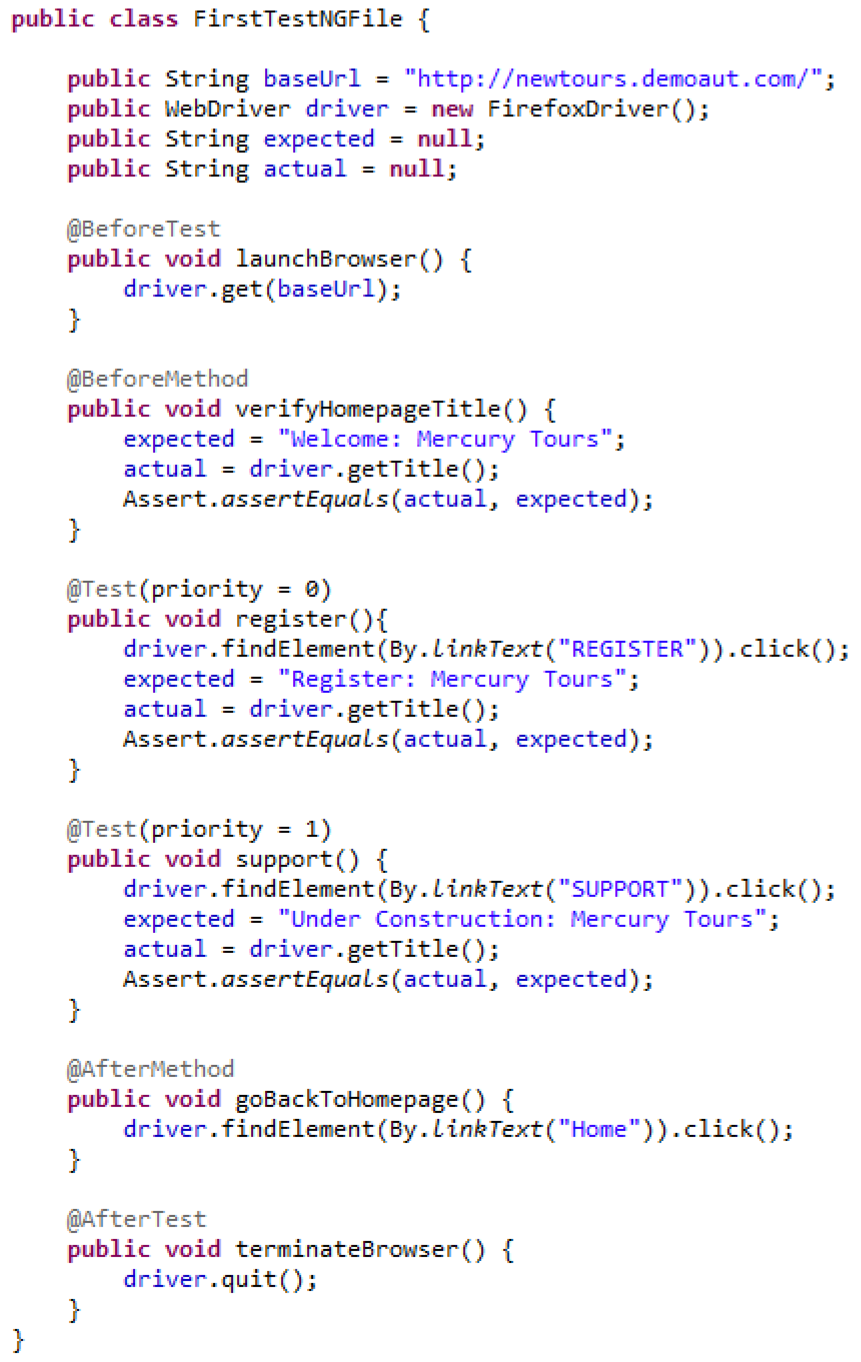
In Mercury Tours, suppose we like to verify the titles of the target pages of the two links below.



The flow of our test would be:

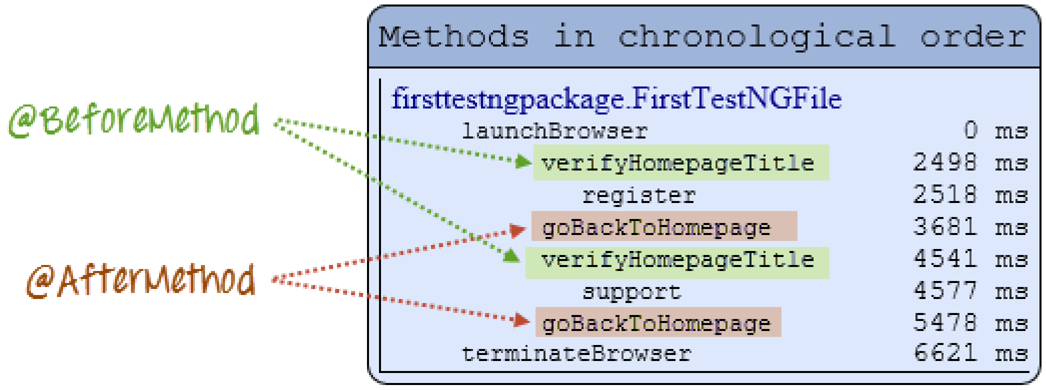
- Go to the homepage and verify its title.
- Click REGISTER and verify the title of its target page.
- Go back to the homepage and verify if it still has the correct title.
- Click SUPPORT and verify the title of its target page.
- Go back to the homepage and verify if it still has the correct title.

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



```java
public class FirstTestNGFile {

    public String baseUrl = "http://newtours.demoaut.com/";
    public WebDriver driver = new FirefoxDriver();
    public String expected = null;
    public String actual = null;

    @BeforeTest
    public void launchBrowser() {
        driver.get(baseUrl);
    }

    @BeforeMethod
    public void verifyHomepageTitle() {
        expected = "Welcome: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected);
    }

    @Test(priority = 0)
    public void register(){
        driver.findElement(By.linkText("REGISTER")).click();
        expected = "Register: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected);
    }

    @Test(priority = 1)
    public void support() {
        driver.findElement(By.linkText("SUPPORT")).click();
        expected = "Under Construction: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected);
    }

    @AfterMethod
    public void goBackToHomepage() {
        driver.findElement(By.linkText("Home")).click();
    }

    @AfterTest
    public void terminateBrowser() {
        driver.quit();
    }
}
```

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



```
Methods in chronological order
firsttestngpackage.FirstTestNGFile
    launchBrowser                    0 ms
      verifyHomepageTitle         2498 ms
        register                  2518 ms
      goBackToHomepage            3681 ms
      verifyHomepageTitle         4541 ms
        support                   4577 ms
      goBackToHomepage            5478 ms
    terminateBrowser              6621 ms
```

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.

Summary of 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