**UNIT – IV**

**Unit Testing Framework & Selenium API:** Introduction to Selenium, Selenium Locators, Finding Elements, working with elements, Synchronizing Tests, Testng and list of annotations used in Testng.

**What is TestNG?**

**TestNG**is an automation testing framework in which NG stands for “Next Generation”. TestNG is inspired by [JUnit](https://www.guru99.com/junit-tutorial.html)which uses the annotations (@). TestNG overcomes the disadvantages of JUnit and is designed to make [end-to-end testing](https://www.guru99.com/end-to-end-testing.html) easy.

Using TestNG, you can generate a proper report, and you can easily come to know how many test cases are passed, failed, and skipped. You can execute the failed test cases separately.

For example:

* Suppose, you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using testNG). When you run this program first, three methods are executed successfully, and the fourth method is failed. Then correct the errors present in the fourth method, now you want to run only fourth method because first three methods are anyway executed successfully. This is not possible without using TestNG.
* The TestNG in Selenium provides an option, i.e., testng-failed.xml file in test-output folder. If you want to run only failed test cases means you run this XML file. It will execute only failed test cases.

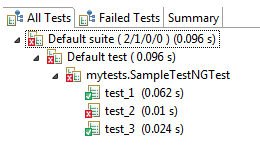
Beside above concept, you will learn more on TestNG, like what are the Advantages of TestNG, how to create test methods using @test annotations, how to convert these classes into testing suite file and execute through the eclipse as well as from the command line.

**Why Use TestNG with Selenium?**

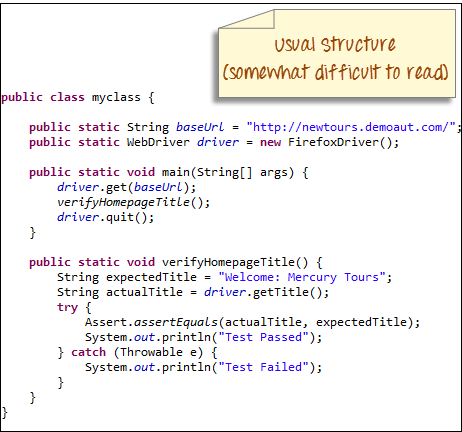
Default Selenium tests do not generate a proper format for the test results. Using TestNG in Selenium, we can generate test results.

Most Selenium users use this more than[Junit](https://www.guru99.com/junit-tutorial.html)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 the key features of Selenium 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](https://www.guru99.com/top-10-cross-browser-testing-tools.html).
* The TestNG framework can be easily integrated with tools like TestNG 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.



* 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.

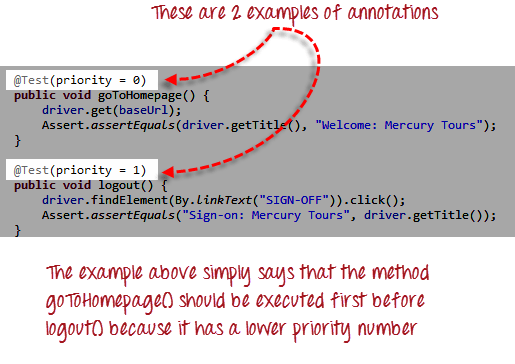
**Advantages of TestNG over JUnit**

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

**What is Annotation in TestNG?lo**

**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 TestNG Example is the one shown below.

 sed in TestNG,”so it is perfectly ok if you do not understand the above TestNG Example just yet. It is just important to note for now that annotations in TestNG are easier to code and understand than in JUnit.

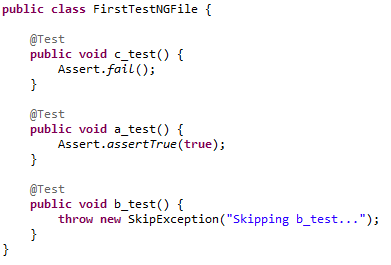
The ability to run tests in parallel is available in TestNG but not in JUnit, so the TestNG framework is more preferred for testers using Selenium Grid.

**Annotations used in TestNG**

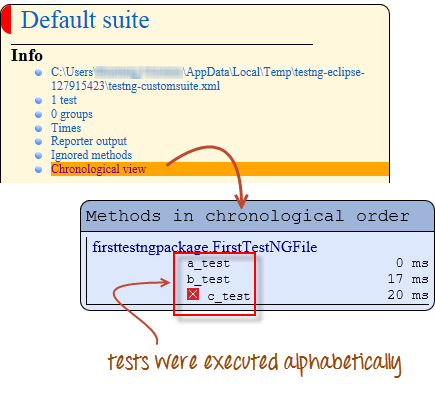
In the previous section, you have been introduced to the @Test annotation. Now, we shall be studying more advanced annotations and their usages.

**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.



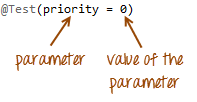
Run this code and on the generated index.html page, click “Chronological view.”



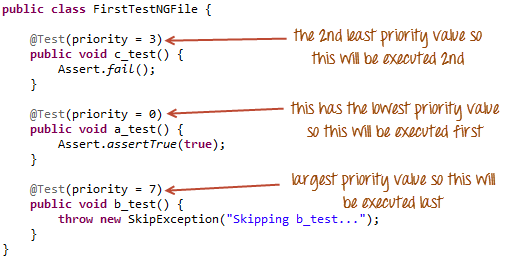
**Parameters**

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**.

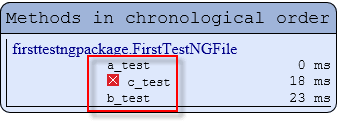
* Parameters require you to assign a value to them. You do.this by placing a “=” next to them, and then followed by the value.
* Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.



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.



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



**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)



@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 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://demo.guru99.com/test/newtours/";

String driverPath = "C:\\geckodriver.exe";

public WebDriver driver ;

@BeforeTest

public void launchBrowser() {

System.out.println("launching firefox browser");

System.setProperty("webdriver.gecko.driver", driverPath);

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 terminateBrowser(){

driver.close();

}

}

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**. Let’s understand with a TestNG Example and try to rearrange the annotation blocks such that your code would look similar to the one below.

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://demo.guru99.com/test/newtours/";

String driverPath = "C:\\geckodriver.exe";

public WebDriver driver ;

@AfterTest //Jumbled

public void terminateBrowser(){

driver.close();

}

@BeforeTest //Jumbled

public void launchBrowser() {

System.out.println("launching firefox browser");

System.setProperty("webdriver.gecko.driver", driverPath);

driver = new FirefoxDriver();

driver.get(baseUrl);

}

@Test //Jumbled

public void verifyHomepageTitle() {

String expectedTitle = "Welcome: Mercury Tours";

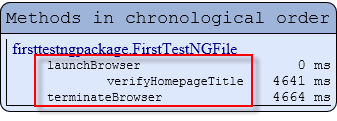
String actualTitle = driver.getTitle();

Assert.assertEquals(actualTitle, expectedTitle);

}

}

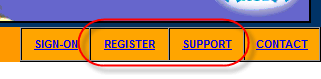
Run the code above and notice that



@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.

package firsttestngpackage;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.\*;

@Test

public class firsttestngfile {

public String baseUrl = "http://demo.guru99.com/test/newtours/";

String driverPath = "C:\\geckodriver.exe";

public WebDriver driver;

public String expected = null;

public String actual = null;

@BeforeTest

public void launchBrowser() {

System.out.println("launching firefox browser");

System.setProperty("webdriver.gecko.driver", driverPath);

driver= new FirefoxDriver();

driver.get(baseUrl);

}

@BeforeMethod

public void verifyHomepageTitle() {

String expectedTitle = "Welcome: Mercury Tours";

String actualTitle = driver.getTitle();

Assert.assertEquals(actualTitle, expectedTitle);

}

@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

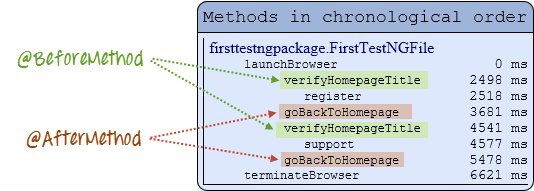
public void terminateBrowser(){

driver.close();

}

}

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.

**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

**Conclusion**

* TestNG is a [testing](https://www.guru99.com/software-testing-introduction-importance.html) 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](https://www.guru99.com/parallel-testing.html).
* 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 TestNG methods.
* TestNG Data provider is a way to pass parameters into the test function which passes different values in test cases in a single execution.