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.

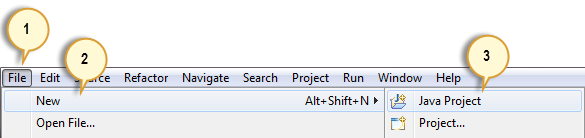
TestNG is inspired from [JUnit](https://www.guru99.com/junit-tutorial.html) which uses the annotations (@).

**First test case using annotations**

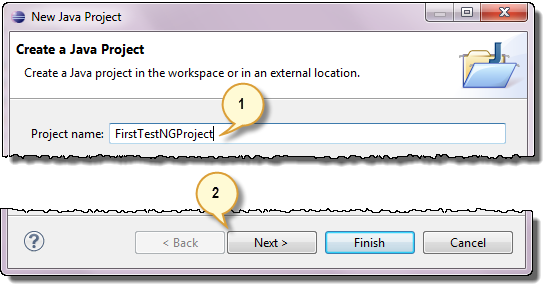
Before we create a test case, we should first setup a new TestNG Project in Eclipse and name it as "FirstTestNGProject".

**Setting up a new TestNG Project**

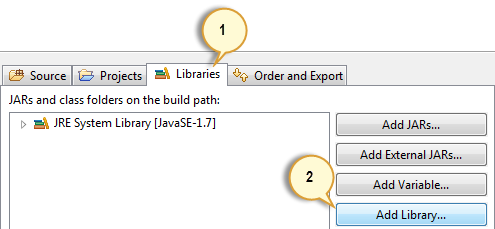
**Step 1:** Click File > New > [Java](https://www.guru99.com/java-tutorial.html) Project

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

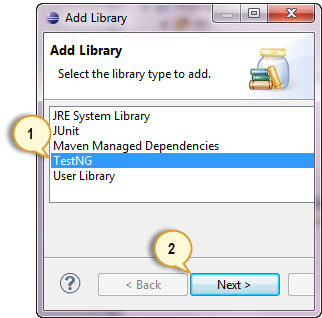
**Step 2:** Type "FirstTestNGProject" as the Project Name then click Next.

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

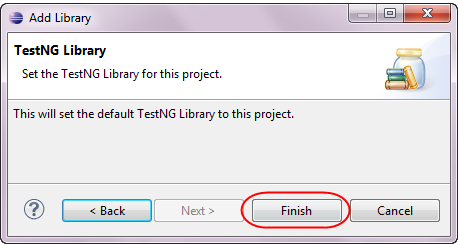
**Step 3:** We will now start to import the TestNG Libraries onto our project. Click on the "Libraries" tab, and then "Add Library…"

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

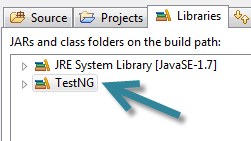
**Step 4:** On the Add Library dialog, choose "TestNG" and click Next.

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

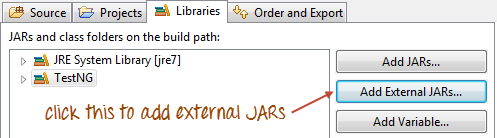
**Step 5:** Click Finish.

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

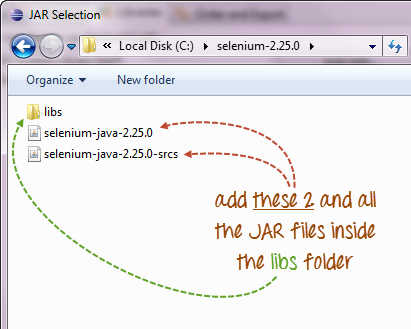
You should notice that TestNG is included on the Libraries list.

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

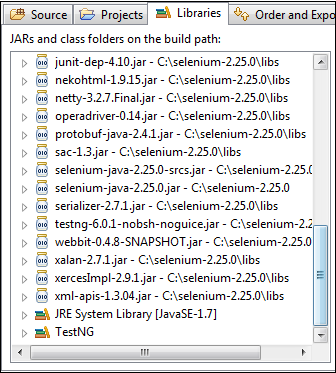
**Step 6:** We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from <http://docs.seleniumhq.org/download/> when we were installing Selenium and Eclipse in the previous chapters.

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

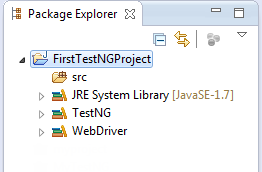
Then, navigate to where you have placed the Selenium JAR files.

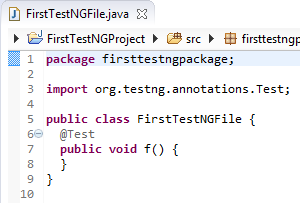
[](https://www.guru99.com/images/step_6b.png)

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 FirstTestNGProject is visible on Eclipse's Package Explorer window.

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

[](https://www.guru99.com/images/step_4-0033.png)

**Coding Our First Test Case**

Let us now create our first [Test Case](https://www.guru99.com/test-case.html) 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();

}

}

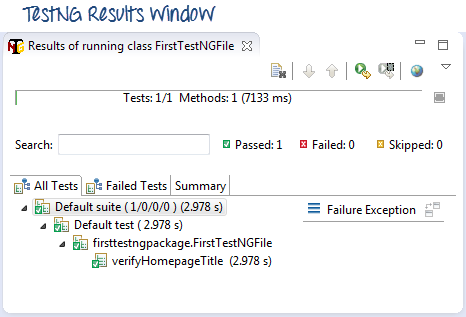
Notice the following.

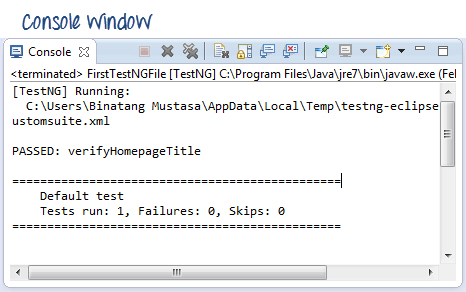
* TestNG does not require you to have a main() method.
* Methods need not be static.
* We used the @Test annotation. **@Test is used to tell that the method under it is a test case**. In this case.
* Since we use annotations in TestNG, we needed to import the package org.testng.annotations.\*.
* We used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

[](https://www.guru99.com/images/testng_results_window-0025.png)

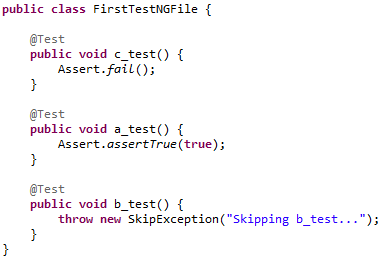
[](https://www.guru99.com/images/console_window-0023.png)

**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)

**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.

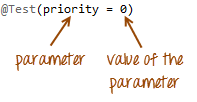
**@AfterMethod**: The annotated method will be run after each test method.

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

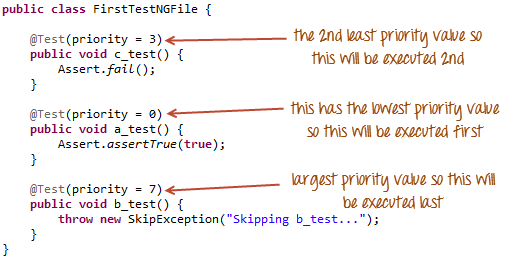
**Parameters**

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.

[](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)

# 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](http://www.inviul.com/log4j-logging-selenium/) 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";

# 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();//for printing report

DataProvider

The DataProvider in TestNG are another way to pass the arguments in the Test annotation.

DataProvider pass the different values to the @Test in single Execution . It is a part of the inbuilt

TestNG data-driven testing for which TestNG is quite popular.

Syntax : @DataProvide(name=”name\_of\_dataprovider”)

Public Object[][] getData(){

Return object[][]{value}

}

* The TestNG DataProvider (the annotation part) contains only one single attribute, which is its name. It is always a string type in nature. For example, “name\_of\_dataprovider,” as mentioned above.
* DataProviders are not declared on top of the functions like [***TestNG parameters***](https://www.toolsqa.com/testng/testng-parameters/)but have a method of their own, which in regular speaking terms called a dataprovider method. For example, dpMethod here.
* If the tester has not specified the name of the dataprovider, then the method name becomes the dataprovider name by default.
* TestNG dataprovider returns a 2d Array of objects.
* The method then performs a data-driven test for each value that you have specified.
* The dataprovider name calls the dataprovider method, and if there is no name specified by the tester, then the dataprovider method is the default name used in the receiving @Test case.

DataProvider example when it is in same class.

**public** **class** DataProviderDemo {

@DataProvider

**public** String[][] getData() {

String st[][] = **new** String[3][2];

st[0][0] = "Ankit";

st[0][1] = "ankit@123";

st[1][0] = "Alia";

st[1][1] = "Alia@12";

st[2][0] = "admin";

st[2][1] = "manager";

**return** st;

}

@Test(dataProvider = "getData")

**public** **void** m1(String un, String pw, String email) {

Reporter.*log*(un + " " + pw + " " + email, **true**);

}

}

For using dataprovider in same class we just have to use dataProvider in method name which will take dataprovider method name as the input.

Example of data provider when the test annotation is in different class.

@Test(dataPtroviderClass=”DataProviderDemo.clas” ,dataProvider = "getData")

**public** **void** m1(String un, String pw, String email) {

Reporter.*log*(un + " " + pw + " " + email, **true**);

}

}

**Reading data from excel file using dataProvider.**

**public** **class** ReadData {

@DataProvider

**public** String[][] getData() **throws** Exception {

String st[][];

FileInputStream fis = **new** FileInputStream(".\\data.xlsx");

Workbook w = WorkbookFactory.*create*(fis);

Sheet sh = w.getSheet("Sheet1");

st = **new** String[sh.getLastRowNum()][sh.getRow(0).getLastCellNum()];

**for** (**int** i = 1; i < sh.getPhysicalNumberOfRows(); i++) {

**for** (**int** j = 0; j < sh.getRow(i).getLastCellNum(); j++) {

st[i - 1][j] = sh.getRow(i).getCell(j).toString();

}

}

**return** st;

}

}

**public** **class** DataExcelDemo {

@Test(dataProviderClass = ReadData.**class**, dataProvider = "getData")

**public** **void** m1(String un, String pw) {

Reporter.lof(“username“ + un + “password” + pw,true);

}

}

How to run two class parallel using testing.

TestNg can run two classes at the same time or simultaneously using different Threads.

**public** **class** ParallelClass1 {

@Test

**public** **void** m() {

Reporter.log(“Class1”,true);

}

}

**public** **class** ParallelClass2 {

@Test

**public** **void** m() {

Reporter.log(“Class2”,true);

}

}

**testng.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* thread-count=*"5"* parallel=*"classes"*>

<test thread-count=*"5"* name=*"Test"*>

<classes>

<class name=*"Org.qsp.testngdemo.ParallelClass1"* />

<class name=*"Org.qsp.testngdemo.ParallelClass2"* />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

we have defined two attributes 'parallel' and 'thread-count' at suite level. As we want classes to be executed in parallel, we have provided 'parallel="classes''. And 'thread-count' attribute is to used to pass the number of maximum threads to be created.

**How to run all the test annotation of a class parallel**

**public** **class** ParallelClass {

@Test

**public** **void** m1() {

System.***out***.println("One");

}

@Test

**public** **void** m2() {

System.***out***.println("Two");

}

}

**testing.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* thread-count=*"5"* parallel=*"methods"*>

<test thread-count=*"5"* name=*"Test"*>

<classes>

<class name=*"Org.qsp.testngdemo.ParallelClass"* />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

How to do parallel browser testing?

To do parallel browser testing we have to take the help of @parameter.

@parameter will help us to read the data from suite file and pass it to .java file .

Some times we have to pass the value to the @test during run time. Like we can pass the specific browser name .

**public class Parallel2 {**

**WebDriver driver;**

**@Parameters("browser")**

**@BeforeTest**

**public void setup(String browser) throws Exception {**

**if (browser.equalsIgnoreCase("firefox")) {**

**System.*setProperty*("webdriver.firefox.driver",".\\Softwares\\geckodriver.exe");**

**driver = new FirefoxDriver();**

**} else if (browser.equalsIgnoreCase("chrome")) {**

**System.*setProperty*("webdriver.chrome.driver", ".\\jar\\chromedriver.exe");**

**driver = new ChromeDriver();**

**}**

**}**

**@Test**

**Public void m1(){**

**driver.manage().window().maximize();**

**driver.get("https://www.google.com/");**

**}**

**}**

**testng.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"* parallel=*"tests"*>

<test thread-count=*"5"* name=*"TestChrome"*>

<parameter name=*"browser"* value=*"chrome"* />

<classes>

<class name=*"Parallel2"* />

</classes>

</test> <!-- Test -->

<test thread-count=*"5"* name=*"TestFirefox"*>

<parameter name=*"browser"* value=*"firefox"* />

<classes>

<class name=*" Parallel2"* />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->