**TestNG**

TestNG is a testing framework inspired by JUnit and NUnit. But introducing some new functionality that makes it more powerful and easier to use.

It is an open-source automated testing framework; where NG of TestNG means Next Generation. TestNG is similar to JUnit but it is much more powerful than JUnit but still, it’s inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. Pay special thanks to Cedric Beust who is the creator of TestNG.

This eliminates most of the limitations of the older framework. It provides the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parametrizing.

***TestNG Topics***

Below is the list of topics that we are going to study in the next chapters of the ***TestNg Tutorial***:

[***TestNG Introduction***](https://www.toolsqa.com/testng/what-is-testng/)***:***What is TestNG or TestNG Framework? The benefits of using TestNG framework with Selenium and Why to use TestNG over JUnit Testing Framework. This article also covers the basics of TestNG Annotations & Listeners.

[***Install TestNG***](https://www.toolsqa.com/testng/install-testng/)***:***How To Install TestNG in Eclipse IDE as well as in IntelliJ to work along with Selenium? Choose the IDE of your choice either Eclipse or IntelliJ and follow the steps to download TestNG and jars to run the test as TestNG test.

[***TestNG Test***](https://www.toolsqa.com/testng/testng-test/)***:***In this article, we will start by creating a fresh TestNG Project and then write a TestNg test in Eclipse and IntelliJ? At the end of the test, we will analyze the created TestNG Reports.

[***TestNG Test Suite***](https://www.toolsqa.com/testng/testng-test-suite/)***:***What is a TestNG test suite and How To create a TestNG test suite? We will look at the steps to create a testng.xml file build test suites in TestNG.

[***TestNG Annotations***](https://www.toolsqa.com/testng/testng-annotations/)***:***What are testNG annotations and what are their benefits? What hierarchy is followed in TestNG annotations and How to use parameters in TestNG with examples?

[***TestNG Groups***](https://www.toolsqa.com/testng/groups-in-testng/)***:***What are groups in TestNG and how to run them in TestNG? How to declare a group inside the group. This also covers the usage of regular expressions in TestNG.

[***TestNG Dependent Tests***](https://www.toolsqa.com/testng/testng-dependent-tests/)***:***What are TestNG dependent tests? How To create single and multiple dependent tests in TestNG and how to create dependencies among them?

[***TestNG Reports***](https://www.toolsqa.com/testng/testng-reports/)***:***How to generate reports in TestNG? What are the different ways to view reports in TestNG? We will also look at the Emailable reports in TestNG.

[***TestNG Parameters***](https://www.toolsqa.com/testng/testng-parameters/)***:***What are the parameters in TestNG? What are Constant, Variable & Optional parameters in TestNG, and how to write tests with those parameters using annotations in TestNG?

[***TestNG DataProviders***](https://www.toolsqa.com/testng/testng-dataproviders/)***:***What are TestNg DatarProviders and How to pass parameters using the TestNG annotations. How to pass a method as a parameter in DataProvider.

[***TestNG Test Priority***](https://www.toolsqa.com/testng/testng-test-priority/)***:***How to set the priority of the tests in TestNG?  In this article, you will learn how to run a specific test before than the other test or vice versa.

[***TestNG Reporter Log***](https://www.toolsqa.com/testng/testng-reporter-log/)***:***How to use TestNG Reporter Log and How to print important messages and do logging in TestNG Reports with Selenium examples.

[***TestNG Asserts***](https://www.toolsqa.com/testng/testng-asserts/)***:***What are asserts in TestNG? How to use TestNG Asserts to validate conditions in Selenium? What are hard assert & soft assert in TestNG and the difference between them?

[***TestNG Cross Browser Testing***](https://www.toolsqa.com/testng/cross-browser-testing-using-testng/)***:***What is cross-browser testing and why do we need cross-browser testing? How to perform cross-browser testing using TestNG with Selenium?

[***TestNG Data Provider with Excel***](https://www.toolsqa.com/selenium-webdriver/testng-data-provider-excel/)***:***What are TestNg DatarProviders and How to pass parameters using the TestNG *DataProviders*. how to pass a method as a parameter in DataProvider.

[***TestNG Parallel Execution***](https://www.toolsqa.com/testng/testng-parallel-execution/)***:***What is parallel execution in TestNG? How to execute parallel testing using methods & classes & suites in TestNG with Selenium WebDriver.

# What is TestNG or TestNG Framework?

[***TestNG***](https://www.toolsqa.com/testng/what-is-testng/) is an open-source testing framework that has evolved as the testing standards and practices evolved over the years. Moreover, the TestNG Framework covers a wide area of tests such as***unit tests***, ***functional tests***, ***end-to-end tests***, ***integration tests***, etc. At the time of writing this tutorial, we are working on **TestNG 7.0.0 version**. Designed “out of frustration from JUnit deficiencies,” the TestNG Framework today stands strong among the bouquet of java testing framework, and the community is strong as well. Subsequently, in this tutorial, we will brief you about the TestNG Framework in general and make you a little familiar with its powerful features, such as annotations and listeners. Indexing the main points of this tutorial, we will be discussing:

* What is TestNG or TestNg Framework?
  + Advantage Of TestNG Over Junit.
  + How to Write TestNG Testcases?
  + Basic TestNG Annotations.
  + TestNG Listeners
  + TestNG Framework With Selenium.

## What is TestNG or TestNG Framework?

***TestNG*** is a testing framework for performing testing in the java programming language and is inspired by [***JUnit***](https://www.toolsqa.com/java/junit-framework/junit-introduction/)and ***NUnit.*** It helps in performing java tests with enhanced functionality and easy-to-use interface with a lot more added functionalities and overcomes JUnit’s deficiencies.

Additionally, the TestNG Framework is an open-source automated testing framework, where **NG** of Test**NG** means **N**ext **G**eneration. The TestNG Framework is better than JUnit, especially when testing integrated classes. Pay special thanks to ***Cedric Beust, who is the creator of*** TestNG!!  How and in what ways it is powerful, is discussed in the next section.

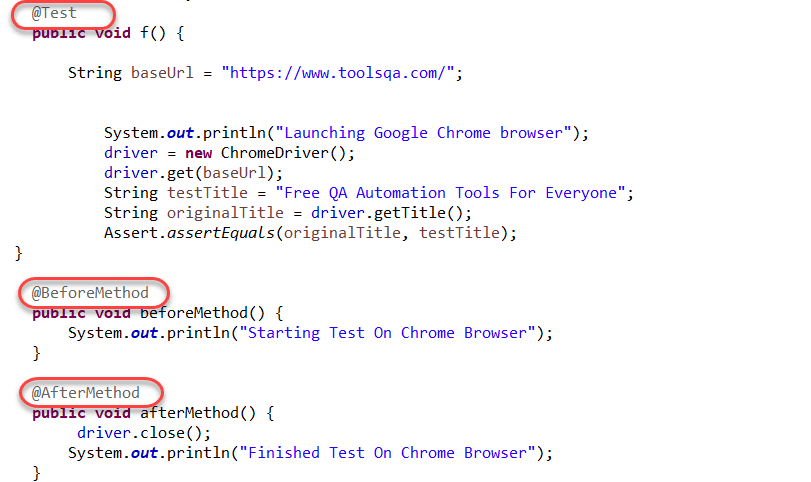
In the test area, the TestNG testing framework provides the ability to the user to write more flexible tests with the help of the TestNG Framework’s [***annotations***](https://www.toolsqa.com/testng/testng-annotations/), [***grouping***](https://www.toolsqa.com/testng/groups-in-testng/), [***sequencing***](https://www.toolsqa.com/testng/testng-test-priority/)***,*** and [***parametrizing***](https://www.toolsqa.com/testng/testng-parameters/). It gives the TestNG Framework an upper hand over JUnit. Additionally, the TestNG Framework provides a lot of benefits for the tester’s community. We will list them out in the next section.

### *****What are the Benefits of TestNG Over JUnit?*****

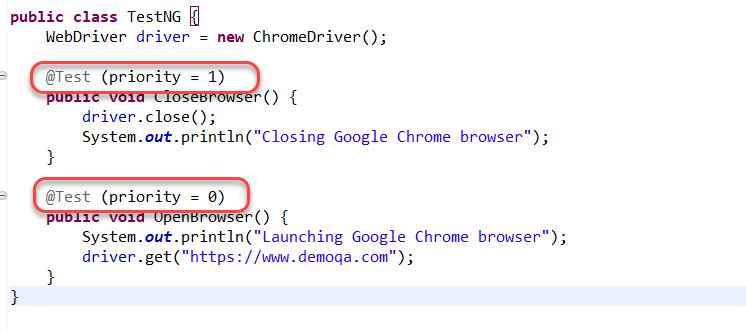
As we mentioned in the introductory section, TestNG is more advanced than JUnit. Below we have listed the significant benefits of TestNG over JUnit and a brief explanation of its features.

1. TestNG gives the ability to produce ***HTML Reports*** of execution. In other words, these reports contain detailed test case results that we can distribute to other team members. TestNG reports are generated automatically and are of two types, emailable reports, and the index report. Subsequently, the detailed discussion on them is in the [***TestNG Reports***](https://www.toolsqa.com/testng/testng-reports/) post.

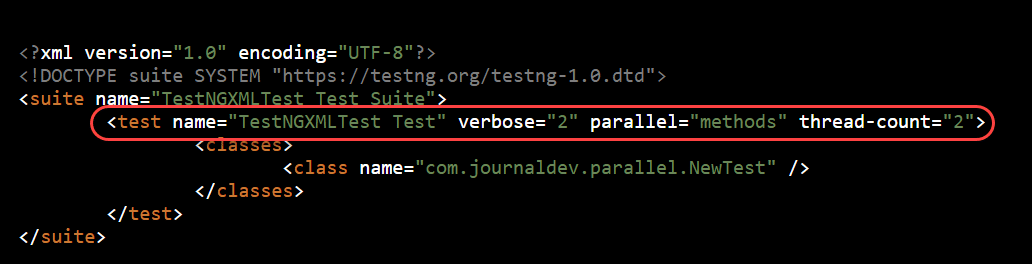
2. TestNG Annotations are very easy to use. In a way, these annotations indicate TestNG about the point when these tests should execute. In TestNG, the tester gets advantages of a lot of annotations and can use them according to their need. Moreover, TestNG annotations are written in English and denote their literal meaning, which makes it easier to remember. For example, @BeforeMethod means executing before the test method, etc. To know more, visit [***TestNG Annotations***](https://www.toolsqa.com/testng/testng-annotations/).



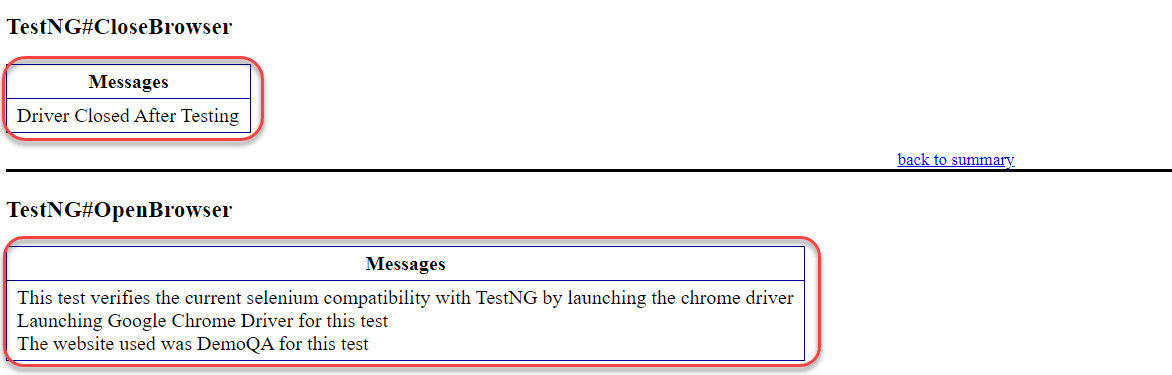
3. Test cases can be ***Grouped & Prioritized*** more easily. That is to say, Grouping the tests in TestNG allows the tester to execute a bunch of tests altogether with just one line of code in the XML file. Since we also need a method to define the order of execution of tests, TestNG helps us by providing a parameter called “priority.” By providing the priority to a test case and assigning it, some integer value changes the order of the sequence of test execution. An example below denotes the same scenario. To know more, you can visit [***what are groups in TestNG***](https://www.toolsqa.com/testng/groups-in-testng/) and [***how to create prioritized tests in TestNG?***](https://www.toolsqa.com/testng/testng-test-priority/)



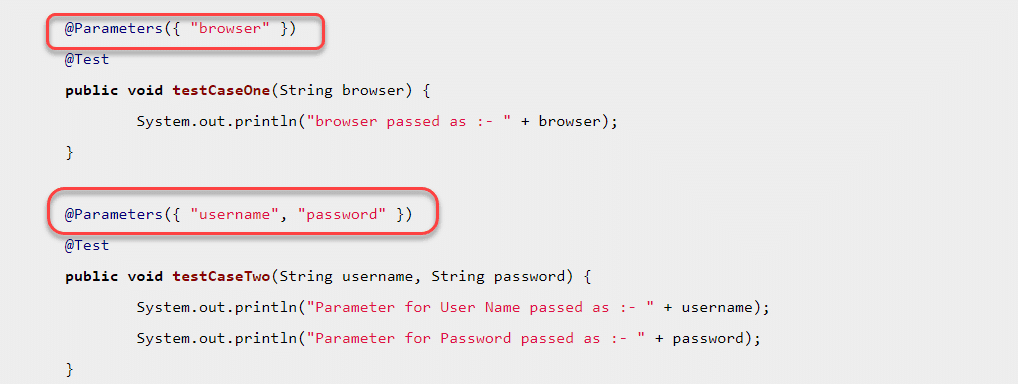
4. Parallel testing is possible with TestNG Framework. Parallel testing refers to the creation of threads and running different methods in different threads parallelly. TestNG also gives us the facility to run not only methods but also classes and even suites parallelly. It saves time and effort for a tester. You can learn more about Parallel Testing and Threads by visiting the [***TestNG parallel execution***](https://www.toolsqa.com/testng/testng-parallel-execution/) post.



5. TestNG Framework helps in generating logs**.** User-defined logs are anything that the tester wants to retain onto the reports. Additionally, these logs get printed over the reports as well as the console when we execute tests in TestNG Framework. These logs play a very vital role when other team members or the tester wants to study these tests after some time. To know more, you can visit [***How to generate reports in TestNG?***](https://www.toolsqa.com/testng/testng-reports/)



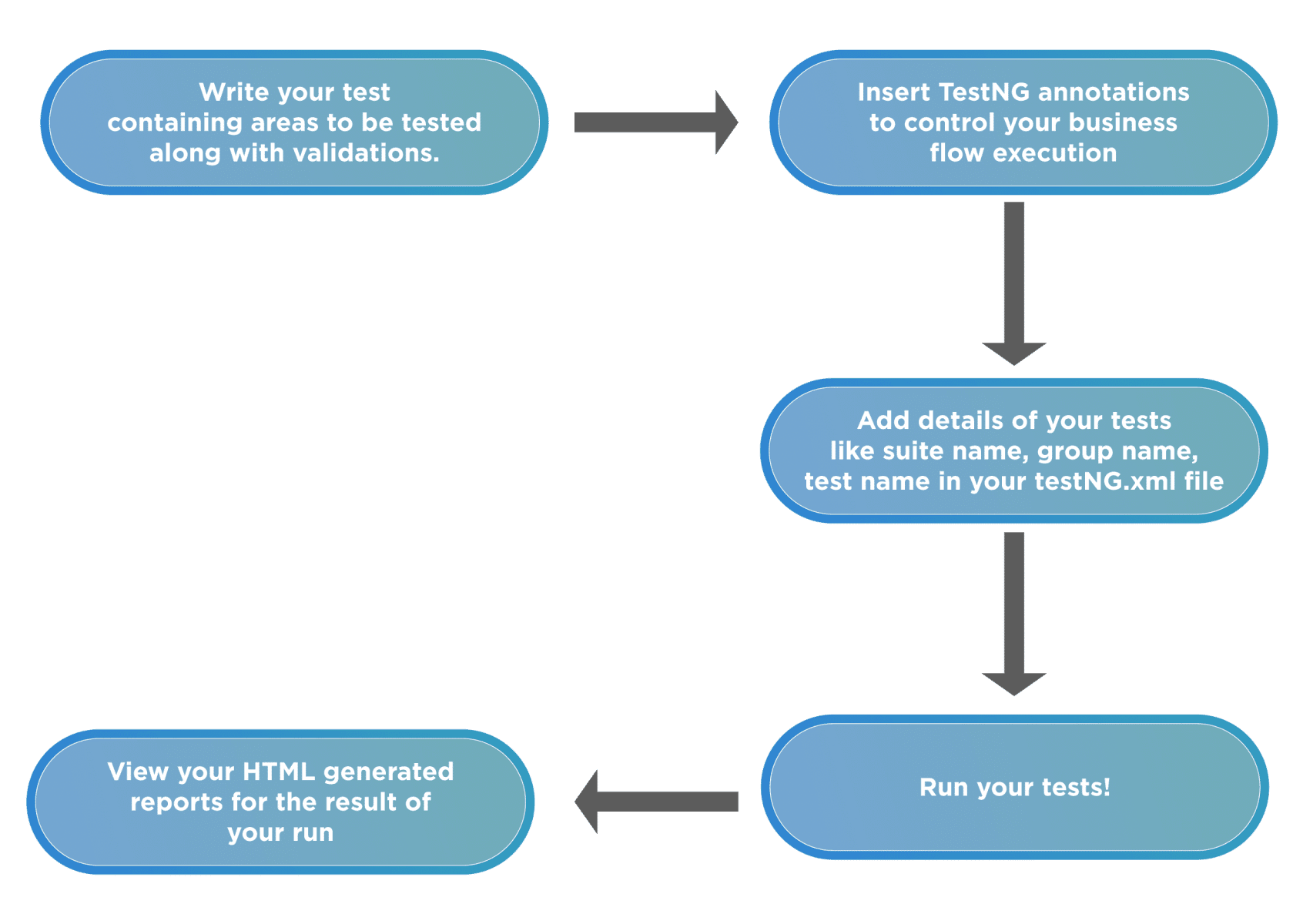
6. Data Parameterization is possible in TestNG Framework. With data parameterization, the tester can pass the data onto the parameters, which help in running the same method again and again with different data values. Moreover, Data parameterization saves a lot of time since the tester need not write multiple tests with different values. Additionally, to know more, you can visit what[***are parameters in TestNG and How to use them?***](https://www.toolsqa.com/testng/testng-parameters/)



These were just the basic concepts available in the TestNG Framework, which makes it stand apart from the JUnit and other similar frameworks. While we move ahead with the course, we will not only discover a lot more topics and features but also study these above-given features in detail, including the exceptions. In the next section, we will see the typical flow followed while writing the test cases in the TestNG Framework.

### *****Test Case Writing Process In TestNG Framework*****

Writing a [***TestNG Test***](https://www.toolsqa.com/testng/testng-test/) is quite simple. The basic flow of writing to running the tests is as below.



Writing the test cases in TestNG involves the following steps:

* ***1st Step*** – Write the business logic of the test.
* ***2nd Step*** – Insert TestNG annotations in the code.
* ***3rd Step***– Add the information about your test (e.g., the class names, methods names, groups names, etc.) in a testng.xml file.
* ***4th Step***– Run TestNG tests.

### *****TestNG Annotations*****

TestNG’s main power comes from its annotations. [***TestNG annotations***](https://www.toolsqa.com/testng/testng-annotations/) are very easy to use and help the testers provide commands to run the tests accordingly.

A tester can run multiple annotations in a single test. Below is the given TestNG annotations list:

* ***@BeforeSuite***: TestNG BeforeSuite method will run before all tests in this suite have run.
* ***@AfterSuite***:  TestNG AfterSuite method will be run after all tests in this suite have run.
* ***@BeforeTest***: TestNG BeforeTest method will execute before any test method belonging to the classes inside the tag is run.
* ***@AfterTest***: TestNG AfterTest method will 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 execute before. TestNG BeforeGroups method is ensured to run immediately before the first test method that belongs to any of these groups is invoked.
* ***@AfterGroups***: The list of groups that this configuration method will execute after. TestNG AfterGroups method is confirmed to run immediately after the last test method that belongs to any of these groups is invoked.
* ***@BeforeClass***: TestNG BeforeClass method will run before the first test method in the current class invokes.
* ***@AfterClass***: TestNG AfterClass method will run after all the test methods in the current class run.
* ***@BeforeMethod***: TestNG BeforeMethod method will run before each test method.
* ***@AfterMethod***: TestNG AfterMethod method will run after each test method.
* ***@Test***: This annotated method is a part of a test case.

### *****TestNG Listeners*****

TestNG listeners are one of the most used features that come bundled with TestNG. Its listeners are the piece of code that “listens” to certain events and act accordingly. TestNG listeners have a lot of advantages in the field of testing as they change the default behavior of TestNG in many aspects. TestNG Framework provides us with a lot of TestNG listeners that are discussed in-depth in the [***What is TestNG Listeners***](https://www.toolsqa.com/testng/testng-listeners/) tutorial. Here is the list of listeners which are provided by TestNG Framework:-

1. ITestListener
2. IReporter Listener
3. ISuiteListener
4. IInvokedMethod Listener
5. IHookable Listener
6. IConfigurationListener
7. IConfigurableListener
8. IAnnotationTransformer Listener
9. IExecution Listener
10. IMethodInterceptor Listener

Researching the TestNG Framework and hearing it from your friends, you will encounter that they are used quite heavily with Selenium. Selenium, if you are unaware, is a web driver that performs efficient tests that run on a browser. Watching this on so many sources, a question might pop-up in your mind that should I use TestNG with Selenium? Or how can I use TestNG? Let’s answer this question.

### *****Should I Use TestNG Framework With Selenium?*****

TestNG Framework is more straightforward, more comfortable, and friendlier when used with Selenium. In the tester’s community, too, TestNG with Selenium is a popular combo, and they use a lot of the TestNG’s features very effectively with Selenium. Therefore, in the course of this tutorial, we will use Selenium’s written tests a lot (and sometimes plain TestNG-Java tests). It does not convey that we have to use TestNG with Selenium only. As you will see during this tutorial, TestNG can be used with whatever type of testing the tester prefers. TestNG is nowhere related to [***Selenium***](https://www.selenium.dev/), but knowing Selenium is an added advantage for the tester.

On the other hand, if you are efficient in Selenium, then you must know that Selenium, in particular, does not have a specific feature to generate test results. TestNG Framework can help testers in this regard and generate test reports along with Selenium to analyze the tests done by the web driver. TestNG also helps in handling the uncaught exceptions while writing the test cases. Don’t worry! We will have a dedicated section in the course for using the [***TestNG Framework with Selenium.***](https://www.toolsqa.com/testng-tutorial/)

## Conclusion

TestNG developed as a framework that could overcome the anomalies of JUnit, which was an already existing framework in Java. Once the TestNG Framework released, it took the tester’s community by storm. As it turned out, not only Beust, but a lot of testers were frustrated by JUnit, and since the TestNG Framework resembled JUnit, nobody had a problem moving to this Framework.

TestNG has a lot of features, one of the most powerful ones being the annotations. Its annotations help the tester determine the meaning of the test method underneath. TestNG parameters work as steroids with annotations and help in the creation of efficient and meaningful test code without writing much logic. Along with this, TestNG provides listeners who listen to the events and execute the code associated with that event. TestNG Framework has strong community support today, and if you are thinking of starting your career journey into it, I will try to provide you with the best content possible on this platform.

Chapter 2:

we introduced you to the [***TestNG framework***](https://www.toolsqa.com/testng/what-is-testng/)***,*** along with its benefits. Additionally, we also made you familiar with a few annotations used in the TestNG framework. Continuing on the same path and before moving to the hands-on experience, we need to install the software on our system. It is effortless to install TestNG in Eclipse as it comes as a plugin.

Moreover, the user just needs to have eclipse installed in their system beforehand. Please visit[***Download and Install Eclipse***](https://toolsqa.com/selenium-webdriver/download-and-start-eclipse/) to setup Eclipse on your system. This tutorial will cover the following:

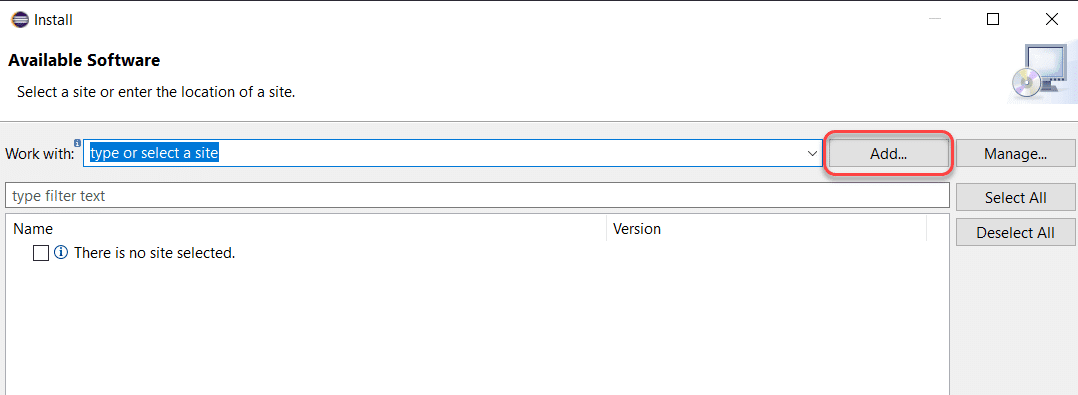
* ***How To Install TestNG In Eclipse?***
* ***How To Install TestNG In IntelliJ?***
  + ***Procedure To Download TestNG Jar in IntelliJ?***
  + ***How To Set Up TestNG Jar In IntelliJ?***

## How To Install TestNG In Eclipse?

The following installation process uses Eclipse Version 4.14.0 as on Mar’20 to install TestNG.

1) Launch the Eclipse IDE and click “***Install New Software***” in the Help menu.

2) You will see a new installation dialog window, click the “***Add***” button.

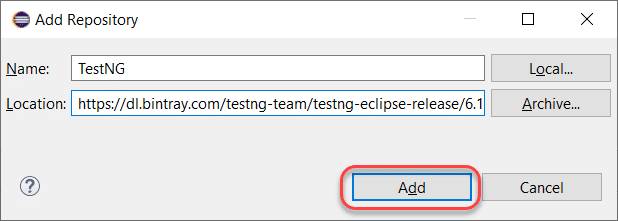


3) Fill out the information as follows:

**Name:** TestNG (depends on the user)

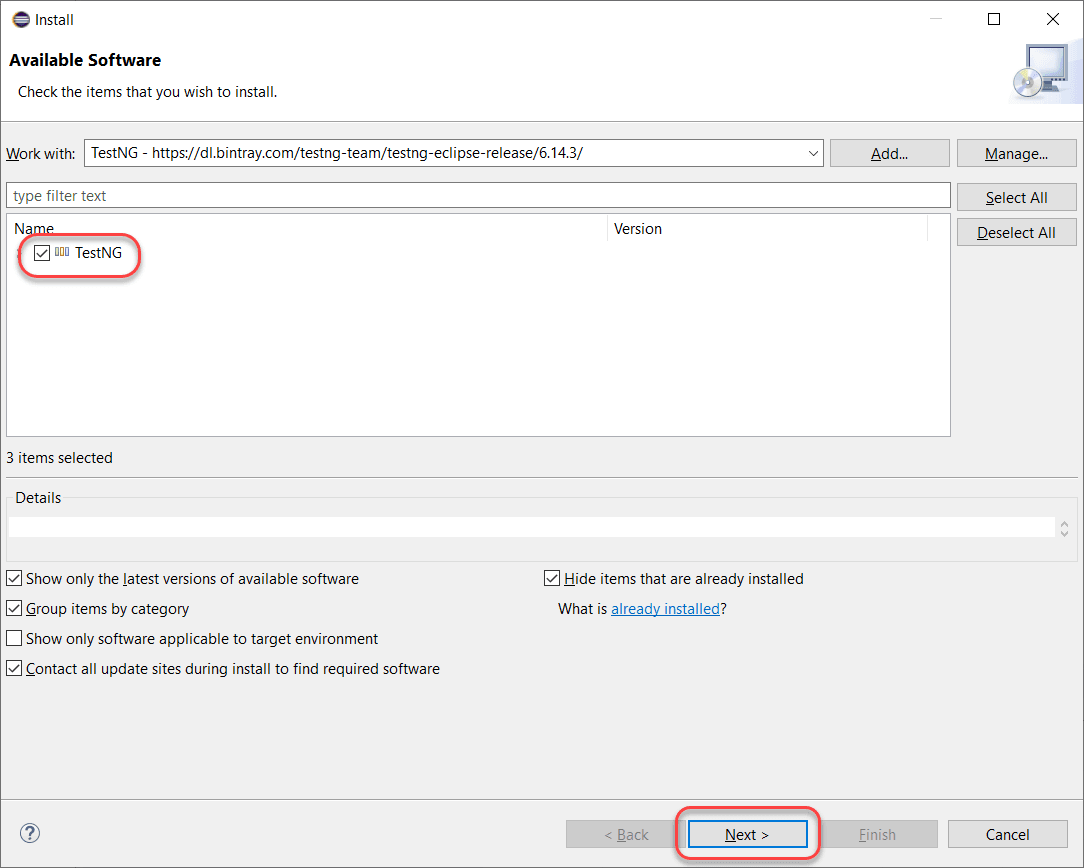
**Location: https://dl.bintray.com/testng-team/testng-eclipse-release/6.14.3/**

Click ***Add***.

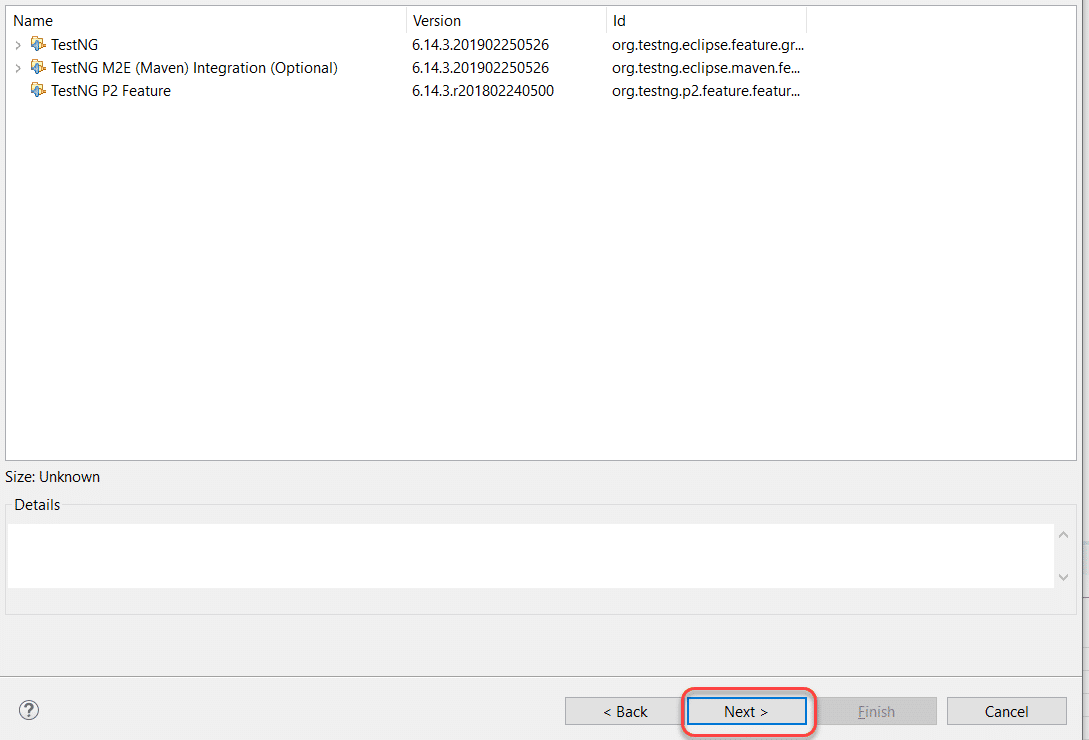


**Note:** There is a deprecation of ***http://beust.com/eclipse/***. For Eclipse06-18, you can install TestNG from the [***Eclipse Marketplace***](https://marketplace.eclipse.org/content/testng-eclipse), and for 09-19, you need to follow the new link as given in this tutorial.

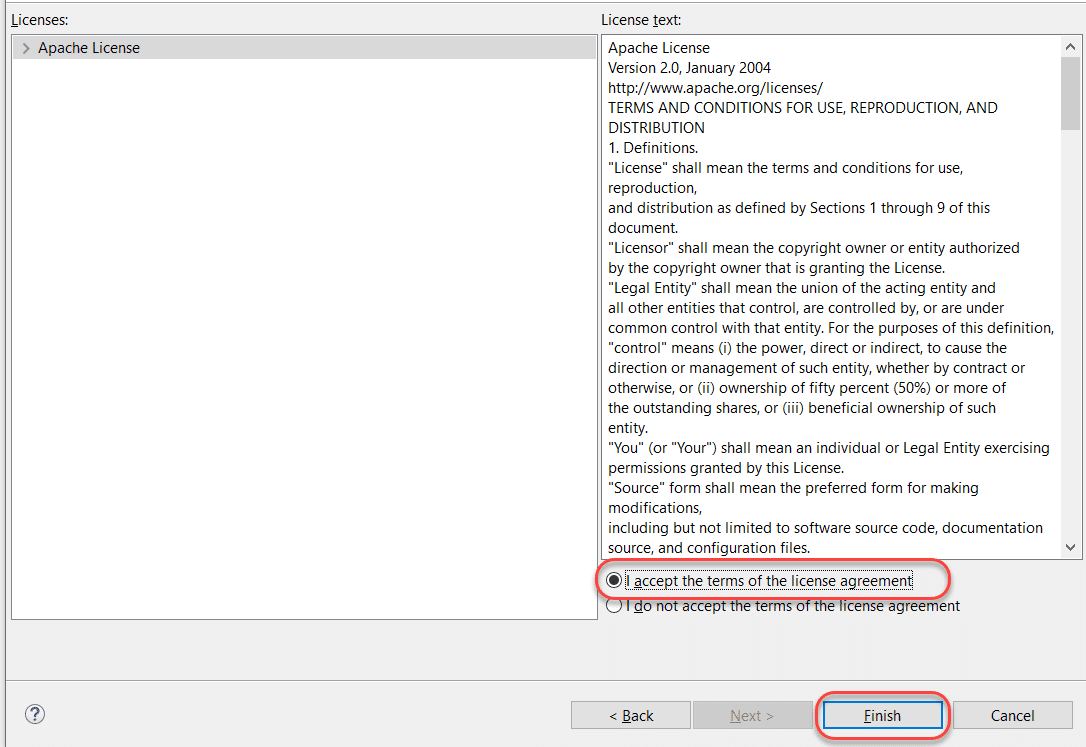
4) Clicking on add redirects us back to the previous window. However, this time you must see the ***TestNG*** option in the available software list. After that, check ***“TestNG”*** and click ***Next***.



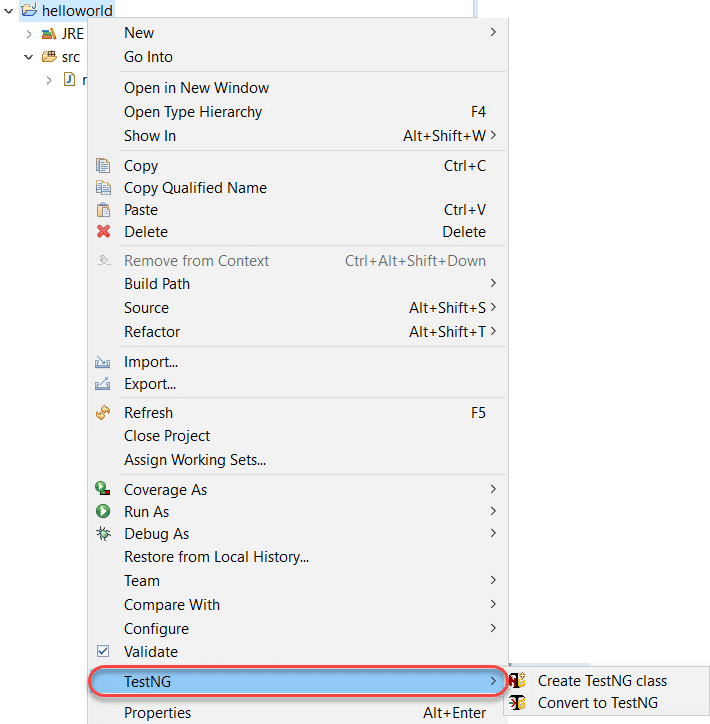
5) Click ***Next*** to install the TestNG dependencies that eclipse calculates by itself.



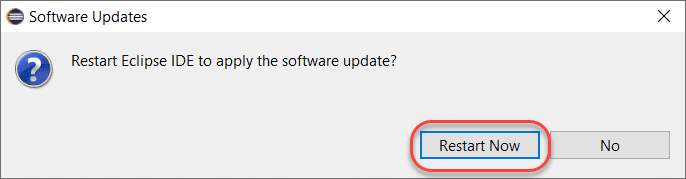
6) After that, ***accept the terms of the license agreement*** then click ***Finish***.



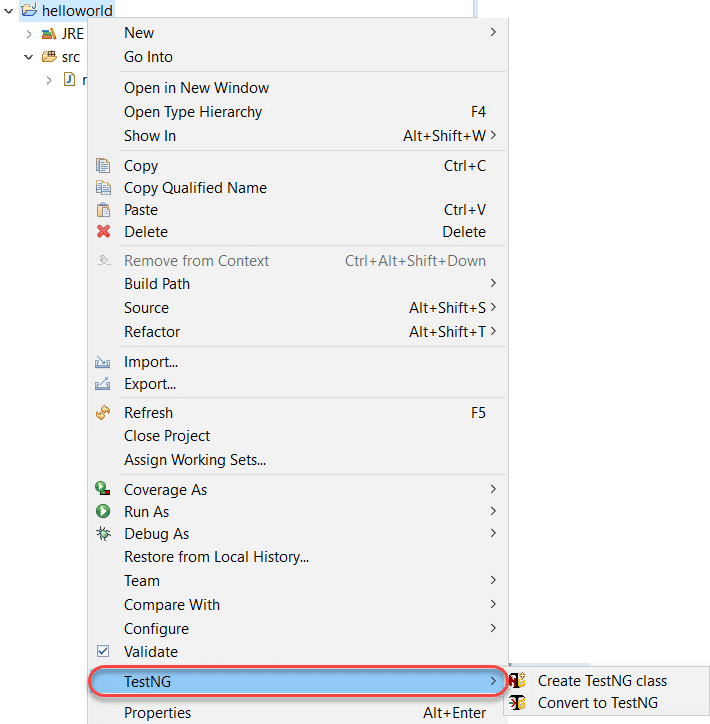
7) You may or may not encounter a Security warning. Click ***Install Anyway***if you do.



8) After that, click “***Restart Now***” to restart the eclipse and finish the installation setup.



9) Finally, after the restart, verify if TestNG installed successfully. Right-click on your project and see if **TestNG**displays in the opened menu.



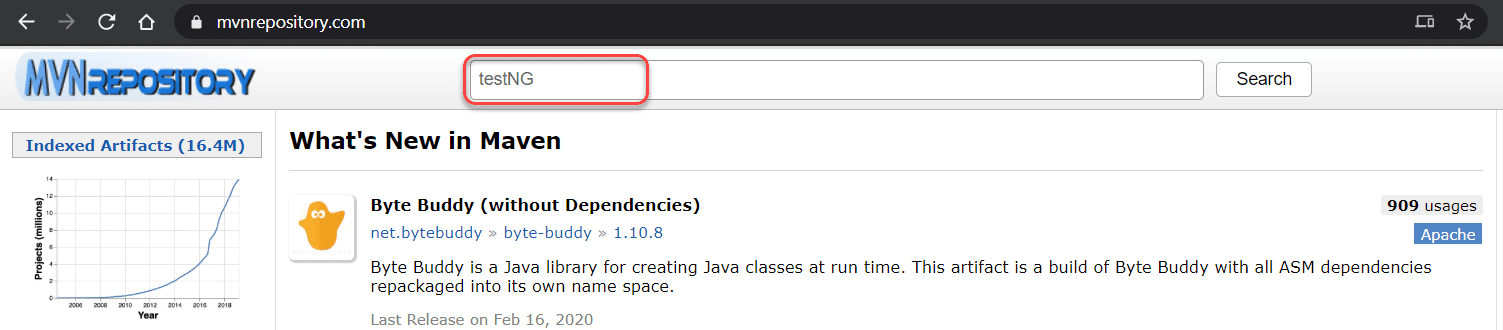
If everything has happened as stated in this post, congratulations! You have TestNG now installed on your system. Subsequently, in the next section, we will look at how to install TestNG on another popular IDE for Java called IntelliJ.

## How To Install TestNG In IntelliJ?

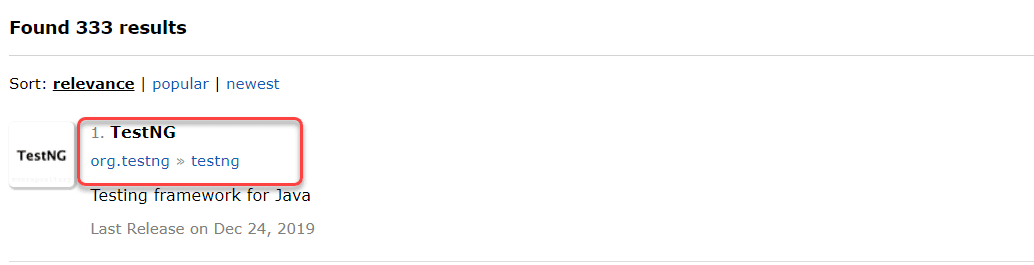
IntelliJ is yet another IDE for running the TestNG test cases other than Eclipse. Intellij requires dependencies to be downloaded externally or through a direct link (if it is a Maven Project). Since we will be running tests on Java, we need to download external TestNG Jar File onto our system.

### *****How To Download TestNG Jar?*****

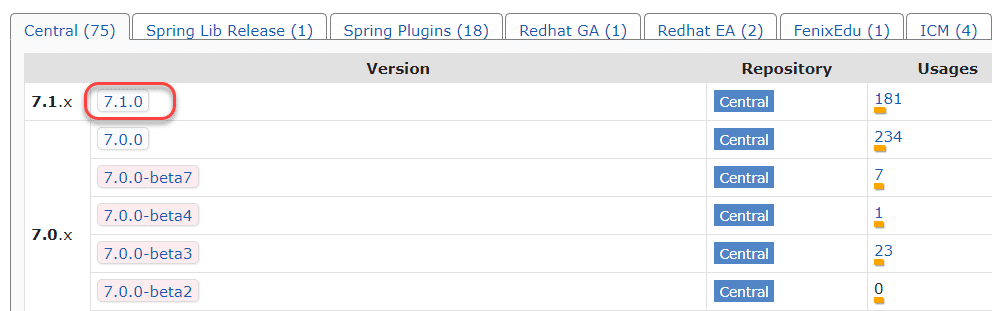
TestNG Jar is very easy to install. Visit [***Maven Repositories***](https://mvnrepository.com/)and search for TestNG in the search bar.



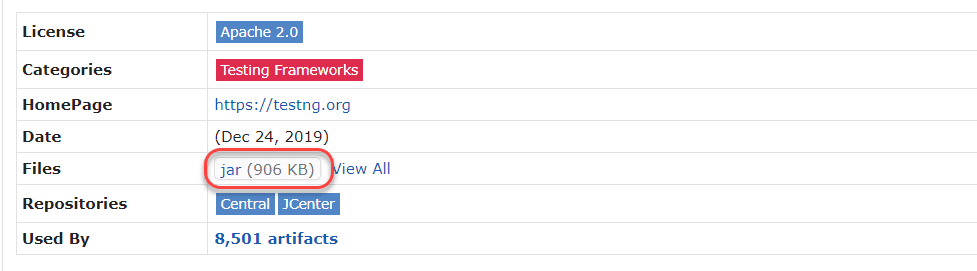
Open the***TestNG***Jar link in the search results.



Select the latest release version that you see. I will use 7.1.0 as its the latest version I have.



Select the “jar” link to download the TestNG jar onto your local system.

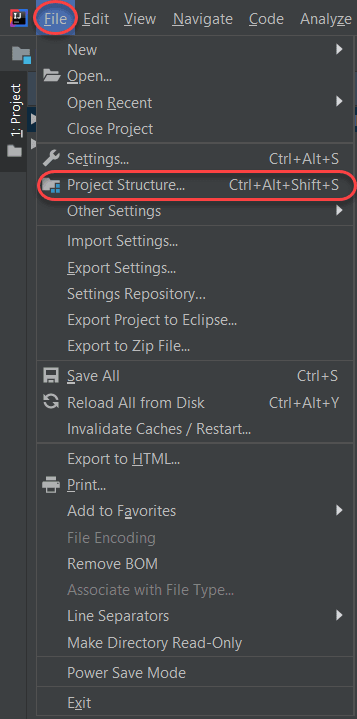


It will download the TestNG jar file in your system. Now we need to add this jar to IntelliJ so that we can use it in our TestNG tests later in the tutorial.

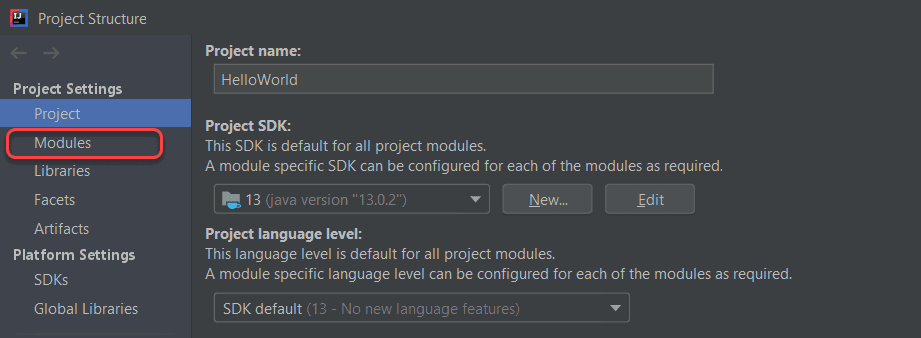
### *****How To Setup TestNG Jar In IntelliJ?*****

Once the TestNG Jar file is downloaded to your system, follow these steps:

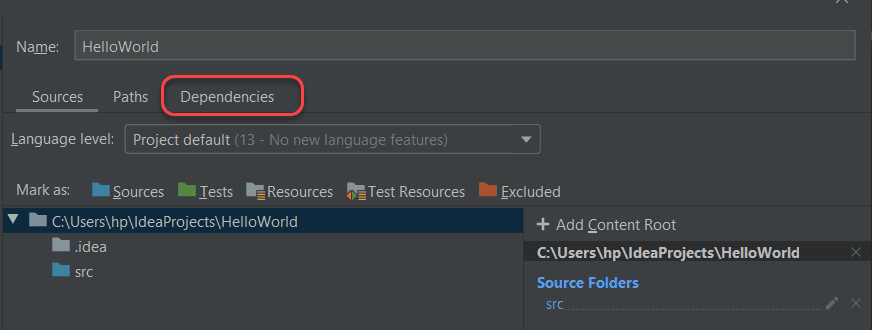
Navigate to ***File -> Project Structure*** in IntelliJ (Ctrl + Alt + Shift +S).



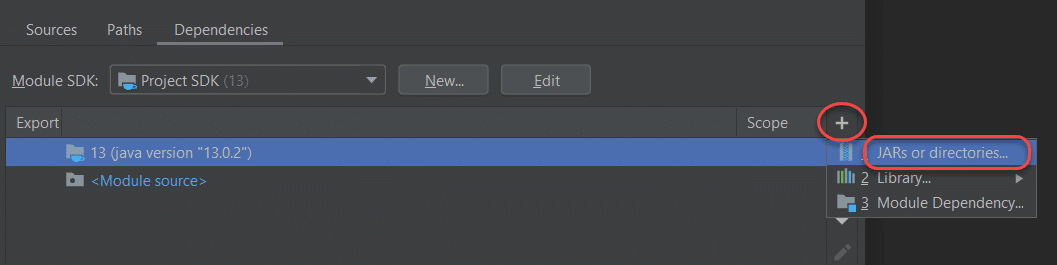
Open ***Modules*** panel.



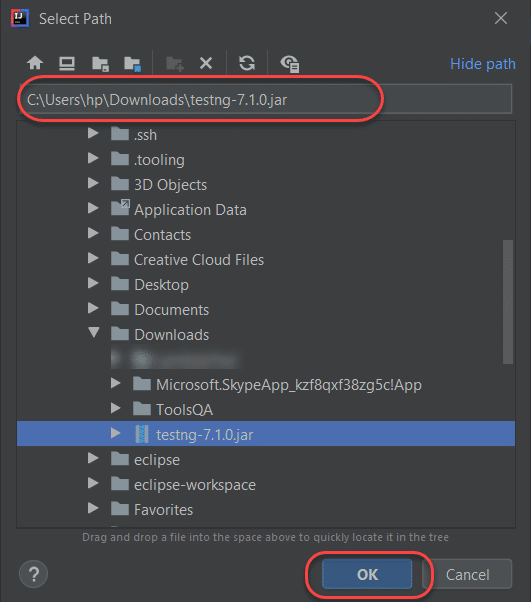
Go to the dependencies tab.



Click the “***+***” sign to add a new dependency and then select **“JARs or directories“.**



Write the path where you downloaded the jar file or navigate directly through the GUI and click Okay.



Select ***Okay*** in the returning panel, and you will have your TestNG installed in IntelliJ.

It was all in the installing section, and honestly, it was quite easy. In this course about [***TestNG***](https://www.toolsqa.com/testng-tutorial/), we will be going ahead with Eclipse and will run TestNG test cases in Eclipse. It is totally up to you what IDE you want to go ahead with as there is not so much difference between the two.

Conclusively, from the next tutorial onwards, we will try some hands-on exercises with [***TestNG test cases***](https://www.toolsqa.com/selenium-webdriver/testng-testcase/)in Eclipse.

we [***installed TestNG in Eclipse and IntelliJ***](https://www.toolsqa.com/testng/install-testng/). Additionally, we also added the TestNG Jar file to Eclipse and IntelliJ and have it all set up on our system. Now, after the completion of the set-up, its time for some hands-on experience of writing our first ***TestNG Test*** in Eclipse. Subsequently, in this tutorial, we will cover the following:

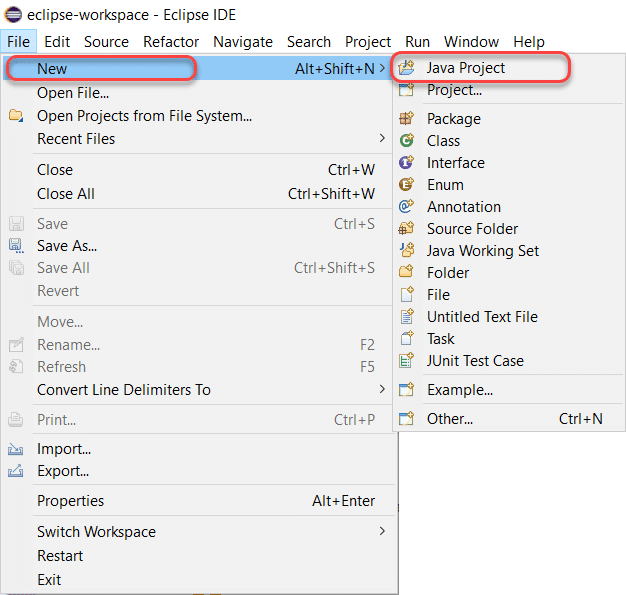
* ***Setting Up A TestNG Project In Eclipse?***
* ***First Test Case With TestNG***
  + ***Downloading Selenium Jar Files For TestNG***
  + ***How To Create A TestNG Test Class In TestNG?***
* ***How To View TestNG Reports?***

Let’s start by setting up a TestNG project.

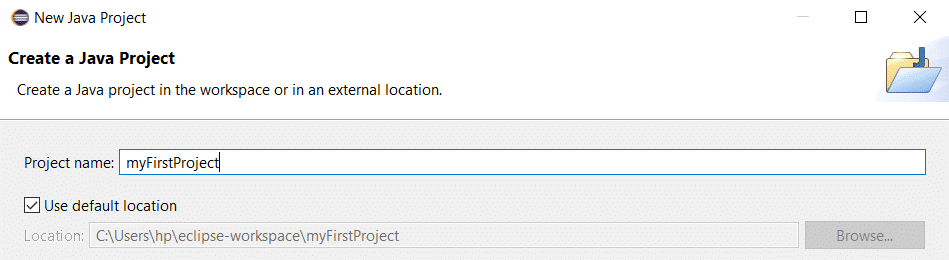
## How To Set Up A TestNG Test Project In Eclipse?

To set up a new TestNG project in Eclipse, open your Eclipse and follow the given steps:

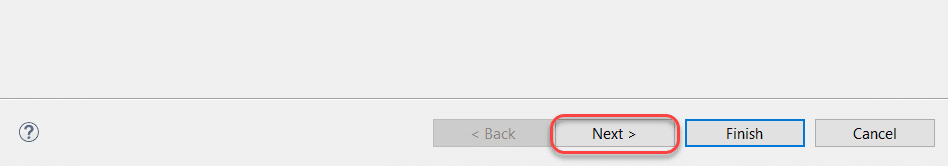
Firstly, navigate To ***File -> New -> Java Project.***



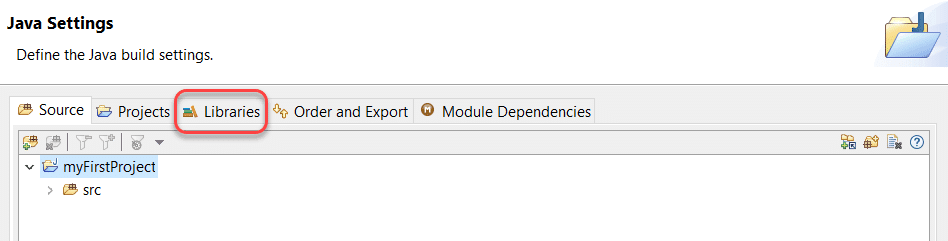
Give it a name of your choice.



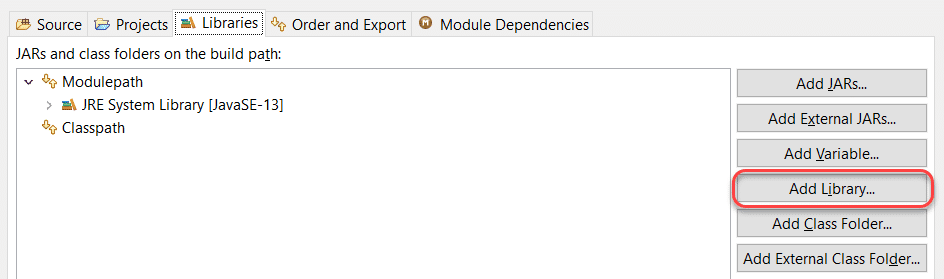
Secondly, click ***Next*** to move to the next panel.



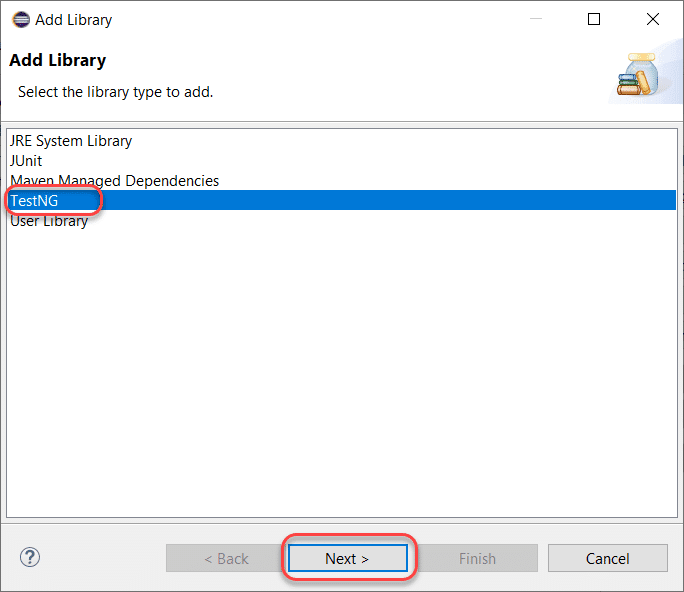
Thirdly, click on***Libraries*** to add TestNG Libraries to your project (Only if Eclipse does not automatically add the TestNG Library).



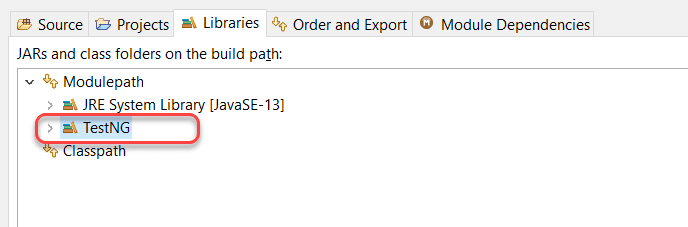
After that, select “***Add Library***” to add the TestNG Library.



Choose ***TestNG*** and click on ***Next***.



Finally, click ***Finish*** to finish adding the TestNg Library in the project.



By this, we have added the TestNG Library to the project. As the next step, we need to make sure that we add the Selenium to the project before moving on to code the first test case.

## How to write a TestNG Test?

Now that we are all set up with TestNG in Eclipse, we will try to write and run our first TestNG test case. But before coding our way through, we need to download Selenium Jar Files.

### *****Download Selenium Jar Files For TestNG*****

TestNG is majorly used with the conjunction of Selenium, so we are also going to write a TestNG test with Selenium. For that, we need to make sure that Selenium WebDriver is also set up in our system. Download the jar files from this link [***Download Selenium Jars.***](https://download.jar-download.com/cache_jars/org.seleniumhq.selenium/selenium-java/3.13.0/jar_files.zip)

Extract the zip file and remember the location where you extracted as we require the location in the further steps.

In the next section, we will create a TestNG class in Eclipse.

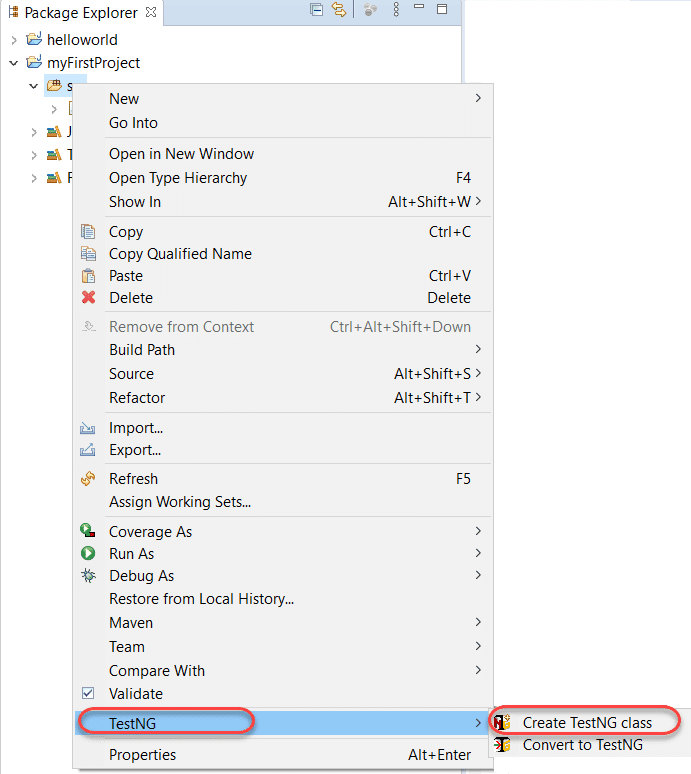
### *****How To Create A TestNG Class In Eclipse*****

Follow the given steps to create our first TestNG class.

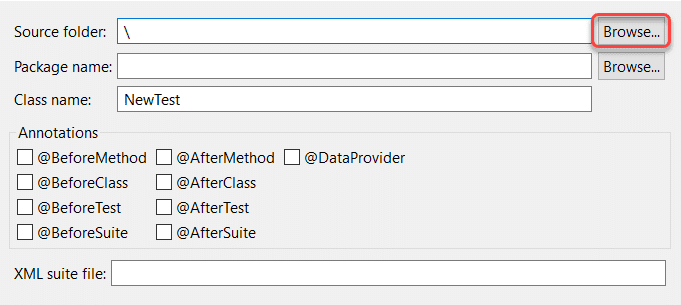
Firstly, press ***Ctrl+N***, then select “***TestNG Class***” under the ***TestNG***category and click ***Next***.

***Or***

Right-click on ***src***, go to **TestNG,**and select “***Create*** ***TestNG Class.”***

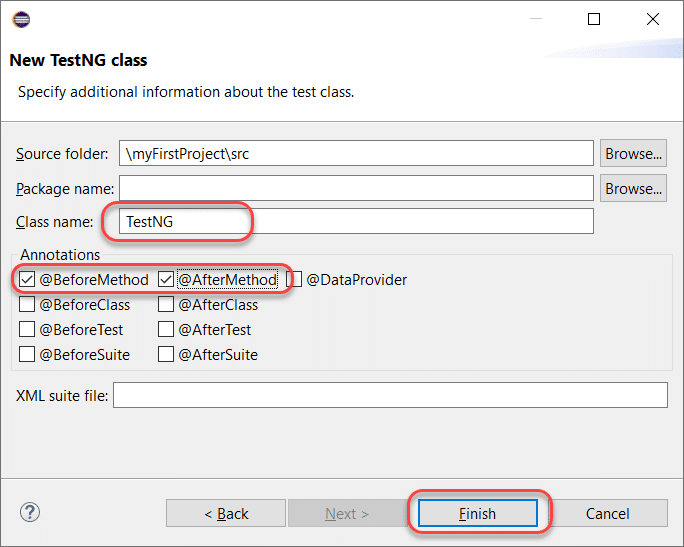


After that, the source folder name will automatically populate in the text field. But if it doesn’t, like my system, you can browse your way through the src folder by clicking on the Browse button.



Thirdly, set class name as ‘TestNG ‘

Leave the Annotations part as it is, for now, we will deal with it in the later tutorials.



***Note:*** To know more about the TestNG Annotations, please refer to What Are TestNG Annotations?

It will display the TestNG.java test file, which is partially created for you. The test case file will contain a default method, f(), along with beforeMethod() and afterMethod() that we checked in the previous step.

Finally, we are all set now by creating our first test class in TestNG. We can now proceed to write the first TestNG test case.

### *****Coding Our First Test Case In TestNG*****

We wrote a straightforward code as a TestNG test case below for you. For understanding the Selenium part, it is recommendable to follow the [***Learn Selenium***](https://www.toolsqa.com/selenium-tutorial/) tutorial. Moreover, we will deal with other TestNG complexities later in the course.

You can copy and paste this code in your Eclipse.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  import org.testng.annotations.BeforeMethod;  import org.testng.annotations.AfterMethod;  import org.openqa.selenium.\*;  import org.openqa.selenium.chrome.ChromeDriver;  import org.testng.Assert;  import org.testng.annotations.\*;    public class TestNG {  WebDriver driver ;  @Test  public void f() {        String baseUrl = "https://www.toolsqa.com/";              System.out.println("Launching Google Chrome browser");          driver = new ChromeDriver();          driver.get(baseUrl);          String testTitle = "Free QA Automation Tools For Everyone";          String originalTitle = driver.getTitle();          Assert.assertEquals(originalTitle, testTitle);    }    @BeforeMethod  public void beforeMethod() {  System.out.println("Starting Test On Chrome Browser");  }    @AfterMethod  public void afterMethod() {  driver.close();  System.out.println("Finished Test On Chrome Browser");  }  } |

Right-click on the test case script and execute the test. After that, select ***Run As*** > ***TestNG Test***.

It will run your tests successfully. We will analyze how those annotations worked in our annotations tutorial. Please note a few points concerning the above-written test case.

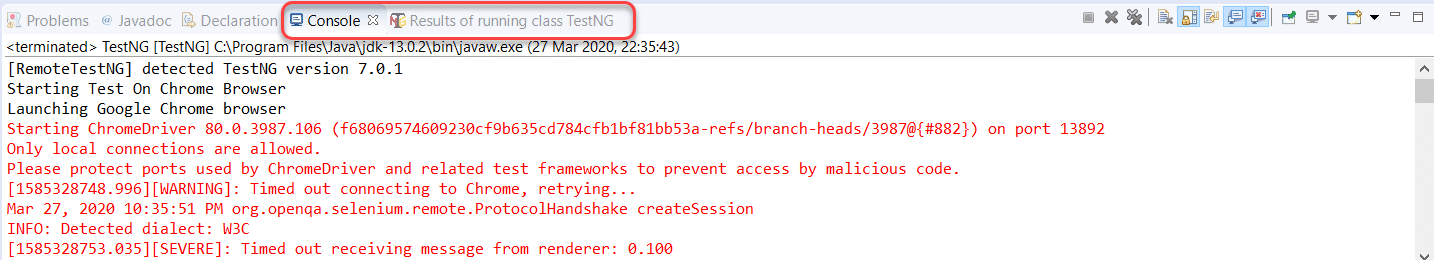
* ***The primary method is not necessary for a TestNG file.***
* ***Moreover, the methods in the TestNG file need not be static in their behavior.***
* ***In addition to the above, @Test annotations tell the underlying methods is a test method.***
* ***Moreover, @BeforeMethod denotes that the underlying method should run before the test method.***
* ***Similarly, @AfterMethod indicates that the underlying method should run after the test method.***

## How To View TestNG Reports?

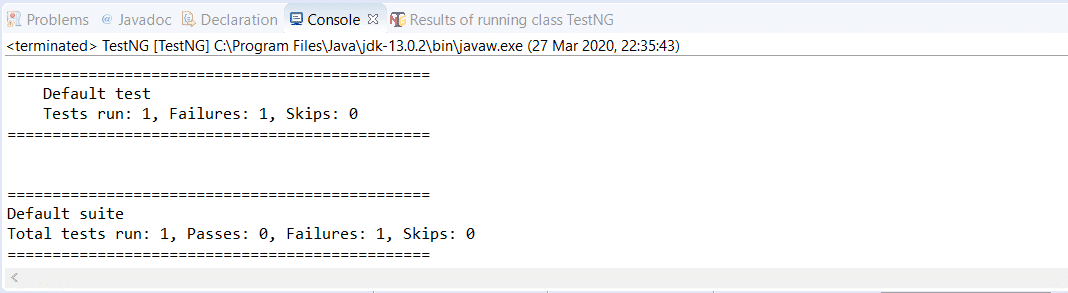
TestNG generates the reports as soon as the tests run. TestNG results are available under two sections:

* ***Console***
* ***TestNG Reports***

The bottom half of the screen shows both of the options.

[](https://www.toolsqa.com/wp-content/uploads/2020/04/console_testng.png)

Scrolling down in the console tab will bring the results of the tests to you.

[](https://www.toolsqa.com/wp-content/uploads/2020/04/testng_results_console.png)

But this is not interesting although it delivers the final aim. Additionally, for a more in-depth view of the tests, we can switch to the TestNG reports section located just beside the console.

This report contains a few elements to analyze. But, we will halt at this point to let you digest the topics as mentioned above and let you code a little around the TestNG test case. Subsequently, in the next section, we shall continue from this point and study how we can generate TestNG reports and different ways of its generation.

we learned about how to create a test and run it in Eclipse. But, often, we do not run a single test on our project. To test the behavior of our software or project, we need to run multiple tests all at once. Moreover, running them manually one by one is not the way. This process of running multiple tests at once is called a test suite, and performing it in TestNG is called ***TestNG Test Suites***. Therefore, it will be the center point of this tutorial covering the following topics:

* **What Is A Test Suite?**
* **How To Create and Run TestNG Test Suite?**
  + **How To Create A TestNG XML File To Execute TestNG Test Suites?**
  + ***Creation of TestNG Test Suites in the XML file.***

## What is a Test Suite?

As I said in the introduction of this tutorial, the collection of [***TestNG Tests***](https://www.toolsqa.com/testng/testng-test/) together is called a ***Test Suite***. A test suite can run multiple tests at once by executing the test suite. Additionally, these test cases can be dependent on each other or may have to be executed in a specific order independently. Moreover, running the TestNG test suite gives us the capability to manage our test execution.

***It is important to remember that the TestNG does not let us define the test suites inside the test code or the main testing source code. Hence, we need to create a TestNG XML file for the same and execute this file.***

Don’t worry. This post will cover all the essentials. Subsequently, let’s see how to create and execute a TestNG Test Suite.

## How To Create and Run TestNG Test Suite?

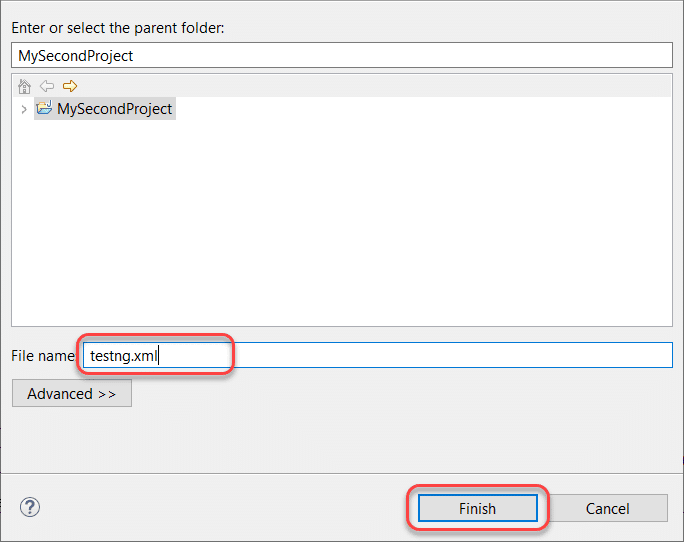
Running a test suite in TestNG requires us to create a TestNG XML file and executing it to achieve the goals. Through this TestNG XML file only, we will be able to create and handle multiple test classes in the TestNG framework. In addition to this, the XML file will be the target file where you will configure your test run, set test dependency, include or exclude any test, method, class or package and set priority, etc.

### *****How To Create A TestNG XML?*****

To create TestNG XML file for running the TestNG test suites, follow the given steps:

1.Right-click on the Project folder, go to **New** and select ***File*** as shown in the below image.

 2. In New file wizard, add filename as ‘***testng.xml***‘ and click on the **Finish**button.



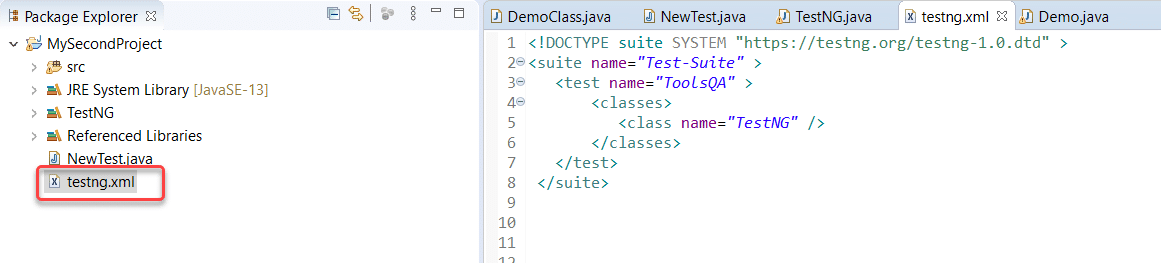
3. Finally, it will add a **testng.xml** file under your project folder, and we are all set to write our first TestNG XML to run TestNG test suites.

After that, add the below-given code in your ***testng.xml*** file.

|  |  |
| --- | --- |
|  | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Test-Suite" >     <test name="ToolsQA" >         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

**Note:** You can select any name for your Test Suite & Test Name as per your requirement. To know more about the TestNG test case file denote here, please refer to [***How To Write Test Cases In TestNG?***](https://www.toolsqa.com/testng/testng-test/)

After giving appropriate names, now your ***testng.xml*** file will look like this:



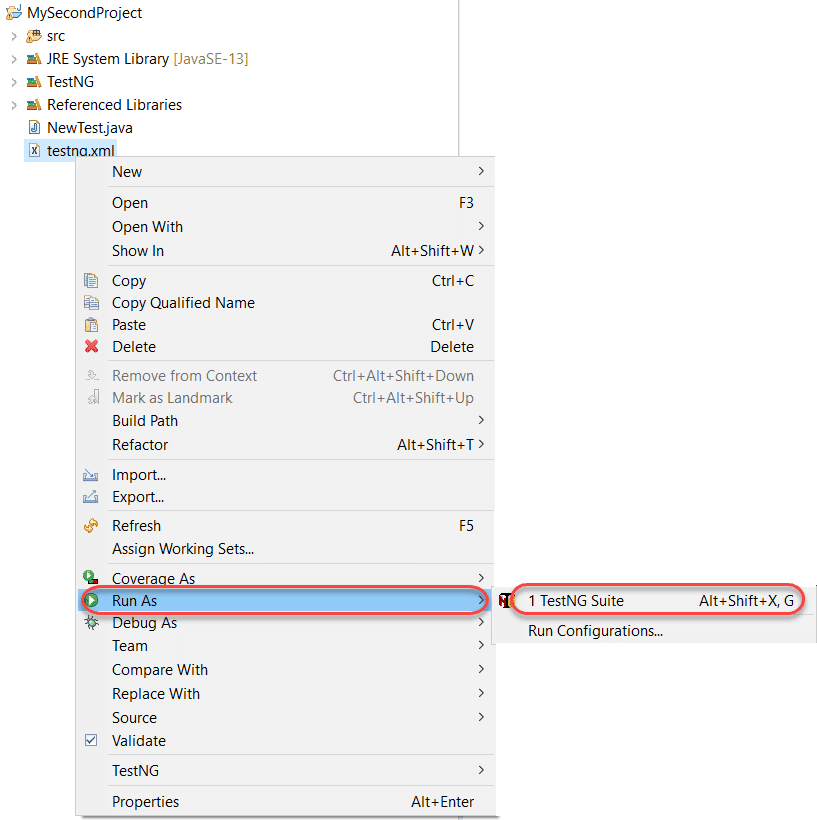
This TestNG XML code does not need any explanation. It is effortless to read and write.

* ***<suite>*** – The suite tag can be given any name and denotes the test suite name.
* ***<test>*** – The test tag can be given any name and indicates your test sets.
* ***<classes>*** – This is the combination of your package name and test case name and cannot write anything else.

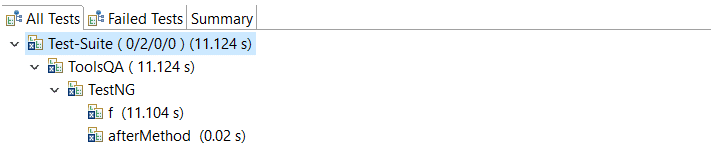
**Note:** that there is no package name in the above-written code. It is because TestNG.java is inside the default package. For example, if the package name is ABC, then this would be written as ABC.TestNG.

### *****How to Run the TestNG Suite?*****

Now it’s time to run the TestNG XML file we just created. Subsequently, run the test by right click on the testng.xml file and select ***Run As*** > ***TestNG Suite***.



It will take a few seconds to start the TestNG execution engine. After that, once the execution is complete, visit the reports section on the bottom half of the Eclipse.



You can view that the TestNG test file has run. You can learn how to read reports in [***How To Analyse the TestNG Reports***](https://www.toolsqa.com/testng/testng-test/) section.

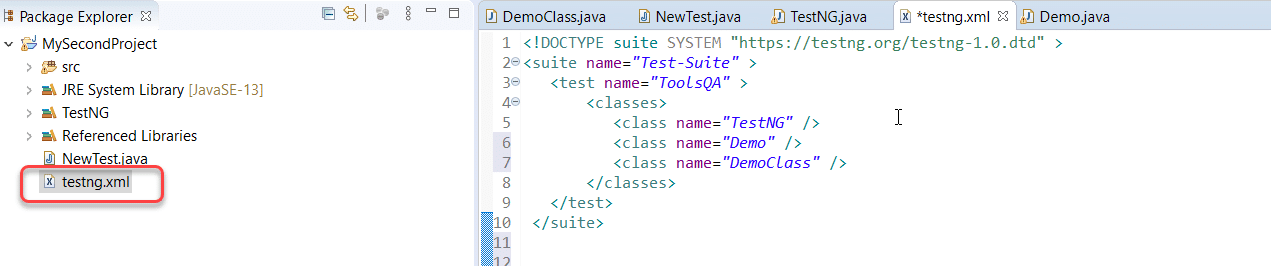
To conclude, this was our simple example of creating and running a testng.xml file in Eclipse.

### *****How to Create a TestNG Test Suite?*****

In the previous section of executing the TestNG XML file, we just ran one test. How did we figure that out? The class tag of the XML file denotes the tests you want to run.

Moreover, we just mentioned a single class tag as the focus was mainly on the XML file creation. Now when you have learned how to build the XML, it’s time to learn ***how to build a Test Suite (multiple tests) using testng.xml***.

It is again not a complicated task since one class denoted one test. All you need to do is to add your test case names to your XML file in ***<classes>*** tag.



The above test will execute only those tests, which are in the ***testng.xml*** file’s class tags. You can go ahead and check the reports to see how the tests have run. The rest of the test cases will remain untouched.  Additionally, the [***TestNG***](https://testng.org/doc/) is a straightforward and robust framework that derives power from the annotations it offers. In the next tutorial, we will look into [***TestNG annotations***](https://www.toolsqa.com/selenium-webdriver/testng-annotations-groups-depends/)and how to implement them in our tests.

we discussed the [***annotations in TestNG***](https://www.toolsqa.com/testng/testng-annotations/)and how to use them in our tests and how to run multiple tests all at once, i.e., running a [***TestNG test suite***](https://www.toolsqa.com/testng/testng-test-suite/). Discussing the hierarchy, we learned how TestNG decides which test to run first and which one to run next. This tutorial is in the continuation of the previous one. This tutorial is about ***TestNG Groups***. TestNG groups combine the tests into groups and let the tester choose which ones to run and which ones to ignore. Along with it, we can combine different groups in TestNG, use a regular expression, and much more. Ready to learn the following out of this tutorial? :

* ***What are TestNG Groups?***
* ***Group of Groups in TestNG***
* ***How To Include and Exclude Groups In TestNG?***
* ***Regular Expressions and TestNG Groups***

## What Are TestNG Groups?

Groups in TestNG denotes the process of grouping different tests together into a straightforward group and running these tests together by just running the group in a single command. It does not even matter if they belong to different classes.

As an example, let’s say you have a hundred tests of a class ToolsQA and in it ten methods of front-end design, ten methods of functional tests, and so on. You probably like to run all the front-end tests together in a batch. And you want all of them to be in a single test suite. With the help of grouping, you can easily overcome this situation by including all the front-end tests into one group.

TestNG also allows us to group test inside groups, which we discussed in the later section of this tutorial. Grouping saves us from defining many classes in our test source code and then running these classes separately, resulting in avoiding the wastage of our time. This situation also helps us in preventing the recompilation of test cases again and again, according to our needs.

***It is important to note that Groups are declared in the testng.xml file in the TestNG and can be found inside <test> tag or <suite> tag.***

***Also, remember that the groups defined in <test> tag apply to only that particular test tag, but the groups defined in <suite> tag apply to all the <test> tags in the XML file.***

|  |  |
| --- | --- |
|  | <suite name="test\_suite">     <groups>        <run>           <include name="UI Test"/>        </run>     </groups>    <test name="Check Login Page">       <classes>           <class name="com.demoqa.Check\_Login\_Page"/>           </classes>       </test>       <test name="Response Status">      <classes>         <class name="com.demoqa.Response\_Status"/>      </classes>      </test>  </suite> |

Since the “groups” is inside the suite tag, it will run all the tests in the XML file. For the second case, see the following tweak in the above code:

|  |  |
| --- | --- |
|  | <suite name="test\_suite">    <test name="Check Login Page">        <groups>           <run>              <include name="UI Test"/>           </run>        </groups>       <classes>           <class name="com.demoqa.Check\_Login\_Page"/>           </classes>       </test>       <test name="Response Status">      <classes>         <class name="com.demoqa.Response\_Status"/>      </classes>      </test>  </suite> |

Now the groups will work only inside the test tag with the name Check Login Page.

### *****How To Create Groups?*****

Before getting more complicated in groups, let’s create a simple group test that we discussed in the above section. In the below code, we will check:

* Whether we are getting the title of the webpage correctly or not.
* Find an element “Sortable” on the website and click the element.

We will be using our demo website demoqa.com for this task. Refer to our tutorial of [***Selenium Webdriver***](https://www.toolsqa.com/selenium-tutorial/) to know in-depth about the code.

Write the following code inside your file named TestNG.java (You can choose any TestNG test case file you want).

|  |  |
| --- | --- |
|  | import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.chrome.ChromeDriver;  import org.testng.annotations.Test;  import org.testng.Assert;    public class TestNG {  WebDriver driver;      // Saving the expected title of the Webpage      String title = "ToolsQA - Demo Website For Automation";          @Test      public void starting\_point(){       System.out.println("This is the starting point of the test");       //Initialize Chrome Driver       //driver.manage().timeouts().implicitlyWait(1,TimeUnit.SECONDS);       driver = new ChromeDriver();       driver.get("https://demoqa.com/");      }          @Test(groups = { "demo" })      public void checkTitle() {         String testTitle = "Free QA Automation Tools For Everyone";         String originalTitle = driver.getTitle();        Assert.assertEquals(originalTitle, testTitle);      }        @Test(groups = { "demo" })      public void click\_element() {         driver.findElement(By.xpath("//\*[@id=\"sidebar\"]/aside[1]/ul/li[1]/a")).click();          System.out.println("Home Page heading is displayed");      }  } |

In the above code, we have defined three tests and one method each under them. The following are the method names:

* ***starting\_point()*** – It will initialize the Chromedriver and will enter the URL demoqa.com automatically.
* ***check\_title()*** – It will check whether the title of the web page is equal to the variable “t***est\_title***” or not. I have placed the wrong title, so this method should fail.
* ***click\_element()*** – This method will find an element called “***Sortable”*** on the web page with the help of XPath and will click it.

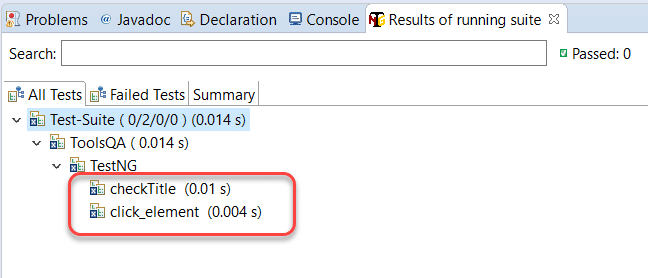
We have used the group name called “***demo***” but only on two methods, namely ***check\_title*** and ***click\_element***. Once we are done with the code, we need to tell our XML file about the groups (if we want to run according to the groups).

|  |  |
| --- | --- |
|  | *<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >*  <suite name="Test-Suite" >     <test name="ToolsQA" >     <groups>     <run>     <include name = "demo"></include>     </run>     </groups>         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

We have included the name of the group “***demo***” inside our XML file within the ***“include”*** tag. By this, we expect to run only two tests because only two tests are included in the group “***demo.”***

Right-click on the file ***->*** ***Run As – > TestNG Suite***

Check the results on the bottom half of the screen.



As expected, only two tests ran, which were inside the group “***demo”.*** Notice that they both must have failed since we were initializing the Chromedriver in the first test, which never ran.

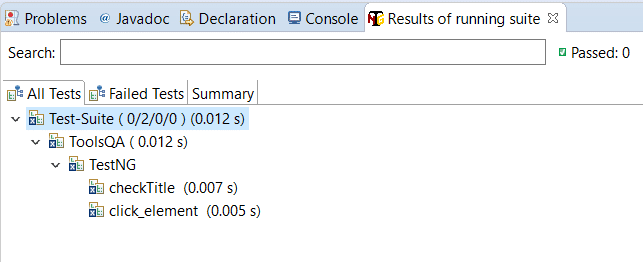
You can create your groups and shuffle the tests in these groups to see the results. But did you know, we can also include groups inside of groups?

## TestNG Groups Inside Groups

***TestNG provides the flexibility of providing groups inside another group*** and running them according to your needs. We can term them “***nested groups,***” but this is no official term. Let’s create a group inside a group in our XML file.

|  |  |
| --- | --- |
|  | *<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >*  <suite name="Test-Suite" >     <test name="ToolsQA" >     <groups>     <define name = "SuperGroup">     <include name = "demo"></include>     </define>     <run>     <include name = "SuperGroup"></include>     </run>     </groups>         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

Above, we created a new group with the name ‘***SuperGroup***‘ and included our group demo into it. Then we called the newly created group (SuperGroup) for execution by including it in the run tag. The output will be like this:



It will be the same as the previous output. As a practice, you can create multiple groups and include all inside this one group. You can try different scenarios.

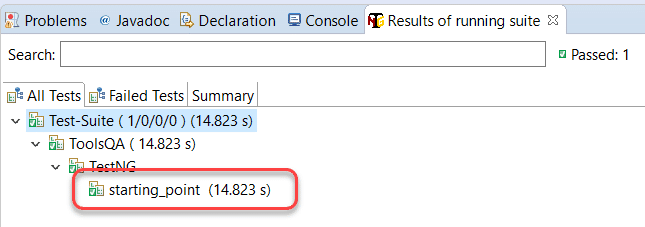
Simple, isn’t it? TestNG is very easy to learn, and groups just provide extra power to these tests and annotations. The above section just told us about including other groups in TestNG groups. It raises a simple question in my mind, and I am sure your’s too. What if we want to exclude a particular group and run the rest? This question must have been raised while developing TestNG, and hence they provided the solution for it.

### *****How To Ignore (Exclude) a group in TestNG?*****

Till now, to run the groups, we have included them inside the “include” tag, which is quite obvious since we want to “include” them in our test run. Similar to this, we can ignore the groups by putting them under the “exclude” tag. This minor tweak can be seen in the XML file below.

|  |  |
| --- | --- |
|  | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Test-Suite" >     <test name="ToolsQA" >     <groups>     <run>     <exclude name = "demo">     </exclude>     </run>     </groups>         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

By putting our group “demo” inside the exclude tag, we are requesting TestNG to ignore the test cases under the group “demo”. So, any guesses how many tests will run? Check the below output to see if you are right or wrong.



Only one test runs because it is the only test that was not inside the “demo” group.

This leaves us with one more point to remember while creating groups. A regular expression helps in defining a generalized pattern to include in the field we apply them. For example, using regular expressions we can command the computer to include only those people from the database who live in a place starting from A and ending from D. It becomes a generalized way. Can we use regular expressions inside the groups in TestNG? We definitely can.

### *****How To Use Regular Expressions With TestNG Groups?*****

TestNG Groups gives us the freedom to include regular expressions inside the include/exclude tag. By this, we can just define a generalised pattern which is there inside more than one groups so that we do not need to write the complete group names. To find this, we have changed the group name to ***demo1*** and ***demo2*** for checkTitle() and click\_element() methods respectively.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.chrome.ChromeDriver;  import org.testng.annotations.Test;  import org.testng.Assert;    public class TestNG {  WebDriver driver;      // Saving the expected title of the Webpage      String title = "ToolsQA - Demo Website For Automation";          @Test      public void starting\_point(){       System.out.println("This is the starting point of the test");       //Initialize Chrome Driver       //driver.manage().timeouts().implicitlyWait(1,TimeUnit.SECONDS);       driver = new ChromeDriver();       driver.get("https://demoqa.com/");      }          @Test(groups = { "demo1" })      public void checkTitle() {         String testTitle = "Free QA Automation Tools For Everyone";         String originalTitle = driver.getTitle();        Assert.assertEquals(originalTitle, testTitle);      }        @Test(groups = { "demo2" })      public void click\_element() {         driver.findElement(By.xpath("//\*[@id=\"sidebar\"]/aside[1]/ul/li[1]/a")).click();          System.out.println("Home Page heading is displayed");      }  } |

Now we have a pattern in the groups i.e. the word “***demo***” is common in both the groups. Include the regex in your XML file and check whether TestNG accepts the regular expression or not.

|  |  |
| --- | --- |
|  | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Test-Suite" >     <test name="ToolsQA" >     <groups>     <run>     <include name = "demo.\*">     </include>     </run>     </groups>         <classes>            <class name="TestNG" />         </classes>     </test>  </suite> |

Notice the expression demo.\* in the include tag. demo.\* denotes that whatever group name starts with demo (no matter what it has ahead of it) should be included in the test. Check the output in your eclipse and see if it has both demo1 and demo2 running or not. It will!!

This was all about using groups in [***TestNG***](https://testng.org/doc/). Remember that groups are very important since you always need to group the tests in bigger projects rather than executing them one by one. Practice grouping of tests as much as you can. In the next tutorial, we will show you another type of tests called the dependency test in TestNG.

In the last tutorial, we discussed the concept of [***Groups in TestNG***](https://www.toolsqa.com/testng/groups-in-testng/) and how to use them in different ways in TestNG. In TestNG, we often require to run the test in a specific order. Along with that, we may require that a test must run only when another test has run. For example, I want testB to run if testA has run. By this, I denote that testB is dependent on testA, and these are called ***Dependent Tests in TestNG***. There are different types of TestNG dependent tests, and we intend to cover all those in this tutorial. So the index will look like:

* **What Are TestNG Dependent Tests?**
  + **Single Dependent Test Methods**
  + **Multiple Dependent Test Methods**
  + **Inherited Dependent Test Methods**
* **Group Dependent Tests In TestNG**
* ***TestNG Dependent Tests in XML Suite***

## What Are Dependent Test In TestNG?

Often, we want to run our test cases in a particular order in TestNG. We may use the [***priority parameter***](https://www.toolsqa.com/testng/testng-annotations/) for that, no doubt, but priority will run all the cases without looking for the relationship we want to define (alphabetically for the same priority). The dependent tests in TestNG determine the dependency of a test on a single or group of tests. In this case, we say that a test is dependent on another test. It is similar to saying a browser is dependent on the internet. No internet means no purpose in running the browser. Providing dependencies among tests also helps us in the scenarios where we want to share the state or data between the methods. So, first things first, how do we specify these dependent tests in TestNG?

### *****How To Use dependsOn attribute in TestNG?*****

TestNG allows you to specify dependencies in the following two ways:

* ***Using attributes dependsOnMethods in @Test annotations***

The dependsOnMethods lets us make a test depend on a particular method. For example, look at the following code:

|  |  |
| --- | --- |
|  | public class Dependent {      @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)");    }  } |

Here, our method SignIn() is dependent upon the method OpenBrowser()

* ***Using attributes dependsOnGroups in @Test annotations.***

The dependsOnGroups attribute lets us make a test depend on a whole group rather than a single test. For example, see the code below:

|  |  |
| --- | --- |
|  | public class GroupDependency  {      @Test(dependsOnGroups = { "SignIn" })      public void ViewAcc() {          System.out.println("SignIn Successful");      }        @Test(groups = { "SignIn" })      public void LogIn() {          System.out.println("Logging In Success");      }  } |

Here, our test method, “***ViewAcc(),***” depends upon the group “***SignIn.”***

Refer [***TestNG Annotations***](https://www.toolsqa.com/testng/testng-annotations/) to know more about the annotations, their benefits, and how they work in the hierarchy with different possible scenarios.

Let us see the different ways in which we can provide the dependency in TestNG.

### Single Dependent Test Methods In TestNG

A single dependent test in TestNG is declared when a single test depends on another test. It is similar to the example we saw above. We use dependsOnMethods for the same purpose.

Take a look over the below example:

|  |  |
| --- | --- |
|  | import org.testng.annotations.Test;  public class DependsOnTest {    @Test (dependsOnMethods = { "OpenBrowser" })    public void SignIn() {    System.out.println("User has signed in successfully");    }      @Test    public void OpenBrowser() {    System.out.println("The browser is opened");    }      @Test (dependsOnMethods = { "SignIn" })    public void LogOut() {    System.out.println("The user logged out successfully");    }  } |

The above code contains three functions:

* ***SignIn()*** – This function depends on the method OpenBrowser (depicted by dependsOnMethod).
* ***OpenBrowser()*** – This method does not depend on any method; hence it is a standard test case.
* ***LogOut()-***  This method depends on the method SignIn() (depicted by dependsOnMethod).

If you think about it, it is helping us in testing in a more significant way. By making one test depend upon another test, we can ensure that SignIn happens after the browser has opened. Logout happens only when the SignIn i. You can use these kinds of logic queries to bring more out of TestNG in your tests.

Let’s verify the output and see if it matches our expectations.

The result is as expected. It brings us to our next question whether we can make one test rely on more than one test? Let’s explore in the next section.

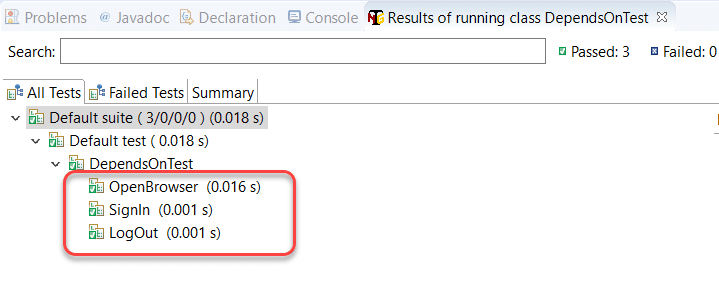
### *****Multiple Dependent Tests In TestNG*****

A single test depends on multiple tests in TestNG. We make use of defining various dependencies for the same. Observe the code below:

|  |  |
| --- | --- |
|  | import org.testng.annotations.Test;  public class DependsOnTest  {      @Test      public void OpenBrowser() {          System.out.println("Opening The Browser");      }        @Test(dependsOnMethods = { "SignIn", "OpenBrowser" })      public void LogOut() {          System.out.println("Logging Out");      }        @Test      public void SignIn() {          System.out.println("Signing In");      }  } |

In the above example, I have created multiple dependencies by making the test method “LogOut” depend upon two different tests, which are “SignIn” and “OpenBrowser.” The Logout() method has been moved to the middle intentionally to demonstrate the use of dependent tests in TestNG. Logically, it would come at the last.

The above code will give the following output in Eclipse:



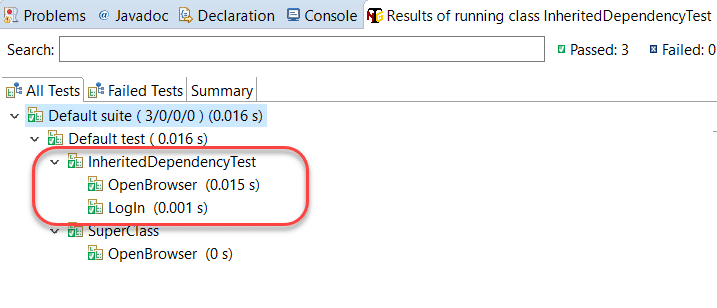
As expected, the LogOut() method executed after the methods on which it depended has executed.

### *****Inherited Dependent Test Methods In TestNG*****

Inheritance is a popular concept in object-oriented programming paradigms, and especially if you have worked in Java, you must have heard and used it a lot. In the previously used methods, we used all the tests of the same class in TestNG. ***In inherited dependent test methods in TestNG, we create dependency among the methods that belong to different classes, and one of the classes inherits the functionalities of another class***.

|  |  |
| --- | --- |
|  | import org.testng.annotations.Test;    class SuperClass  {      @Test      public void OpenBrowser() {          System.out.println("BrowserOpened");      }  }    public class InheritedDependencyTest extends SuperClass  {      @Test(dependsOnMethods = { "OpenBrowser" })      public void LogIn() {          System.out.println("Logged In");      }  } |

As seen in the above code, the InheritedDependencyTest class inherits the SuperClass and hence can depend on any function described in SuperClass. Consequently, the above code will result in the following output:



Please note that the name of the TestNG class is InheritedDependencyTest, which is the base class in the above code. As a practice, you can swap superclass and base class names to analyze the response from TestNG.

## What Are Group Dependent Test?

In the above sections, we just created dependencies of a single test over another test (single or multiple). While creating dependencies in multiple tests, we have to write each test name separately manually. For example, if I want to make a test depend on three other tests, I need all the three names in the dependsOnMethod attribute. What if we want to create a dependent test that depends on a group of tests? If we could do that, we can have a group of a test run before this test and then we just need to mention a single group name instead of multiple test names.

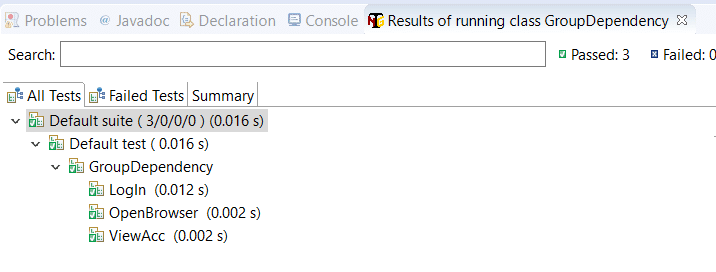
Fortunately, ***TestNG provides the functionality of making a test depend on a group of tests.*** Let’s see the following code to understand the same.

|  |  |
| --- | --- |
|  | import org.testng.annotations.Test;    public class GroupDependency  {      @Test(dependsOnGroups = { "SignIn" })      public void ViewAcc() {          System.out.println("View Your Dashboardd");      }        @Test(groups = { "SignIn" })      public void OpenBrowser() {          System.out.println("Browser Opened Successfully");      }        @Test(groups = { "SignIn" })      public void LogIn() {          System.out.println("Login Into The Account");      }  } |

In the above code, we created a group called SignIn and a test method ViewAcc that depends on that group.

The group SignIn contains two tests under it called OpenBrowser (describing the successful event of opening a browser) and LogIn (describing the successful login event).

Run the file to see that the ViewAcc will run in the last.



As expected, the ViewAcc runs after all the group tests have run on which it depended.

But, notice the tests run in the group SignIn. We intended to run the OpenBrowser test first and LogIn after that. I mean, that makes logical sense. But in the above run, LogIn has run before the OpenBrowser method. That would create problems as how would you LogIn if the browser has not yet opened.

So, any guesses why that happened?

***Since the group and the annotation is the same, TestNG will run the methods in the alphabetical order.*** Try renaming the LogIn as something like Testing(). Since T comes after O, OpenBrowser will run first. For more on this, please refer to [***TestNG Annotations and its hierarchy***](https://www.toolsqa.com/testng/testng-annotations/).

## TestNG Dependent Test In XML Suite

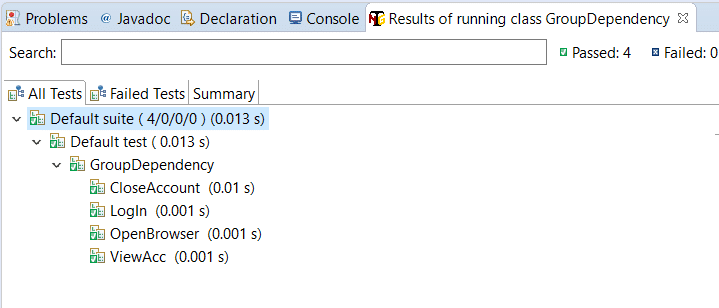
All that we discussed in the above sections of this tutorial was somehow related to the changes done in the TestNG test case file. In this section, we will move our dependency commands over to the XML file.

***TestNG lets you create dependencies between groups in the XML file.*** So, if you have multiple groups in the TestNG file, you can create the dependent tests in between them in the XML file.

Let’s create multiple groups in our TestNG test case file first before jumping to the XML file.

|  |  |
| --- | --- |
|  | import org.testng.annotations.Test;    public class GroupDependency  {      @Test(groups = { "viewacc" })      public void ViewAcc() {          System.out.println("View Your Dashboardd");      }        @Test(groups = { "openbrowser" })      public void OpenBrowser() {          System.out.println("Browser Opened Successfully");      }        @Test(groups = { "login" })      public void LogIn() {          System.out.println("Login Into The Account");      }        @Test(groups = {"logout"})      public void CloseAccount() {       System.out.println("Closing The Account");      }  } |

In the above code, I have created four groups and declared one method in each of them. Notice that there are no dependencies in this file so the file will currently run the methods in the alphabetical order as follow:



We don’t want to close our account before logging in.

Let’s hop on to the XML file and create some dependencies.

|  |  |
| --- | --- |
|  | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="TestNG XML Dependency Suite" >     <test name="ToolsQA" >     <groups>     <dependencies>     <group depends-on= "openbrowser" name= "login"></group>     <group depends-on= "login" name= "viewaccount"></group>     <group depends-on= "viewaccount" name= "logout"></group>     </dependencies>     </groups>         <classes>            <class name="GroupDependency" />         </classes>     </test>  </suite> |

In the “***dependencies***” tag, I have created the flow of groups that I want to execute. There are a total of three components in each dependency that I have created.

***<group>*** – The tag you need to specify to tell XML that we are talking about the groups.

***depends-on*** –  The name of the group on which you want this group to depend.

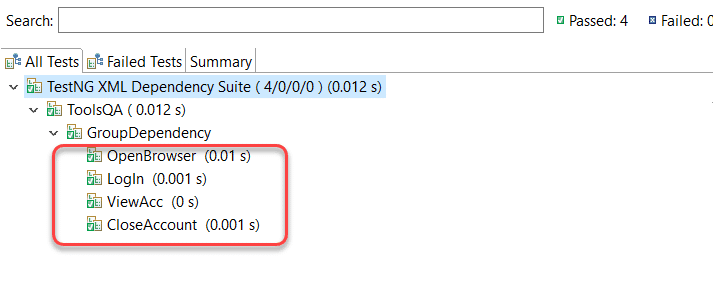
***name-*** Name of the group that you want to depend on.

<group depends-on = “openbrowser” name = “login”></group>

***The above code states that the group named “login” should depend upon the group with the name “openbrowser.”***

Similarly, you can analyze the other two dependencies in the above XML code.

Running the test suite will give us the following output:



Since before CloseAccount was running before every other test, this time, we have run it in the correct order by creating the dependency.

I hope that was a clear and crisp tutorial about the dependent tests in TestNG. Till this point, we have covered a lot by which it would be straightforward to write good tests in [***TestNG***](https://testng.org/doc/) and use different types of tests also. Practice your way as much as you can to write good tests in TestNG. We will move on to our next tutorial.

# TestNG Reports

In the last tutorial, we wrote and executed a [***test case in TestNG***](https://www.toolsqa.com/testng/testng-test/). When we run a test case in TestNG or a [***test suite***](https://www.toolsqa.com/testng/testng-test-suite/)***,*** if that matters, the results are shown at the bottom half of the screen (we will be talking in terms of eclipse only). These results are in terms of reports, and ***TestNG Reports*** required a separate section in this course as they are very detailed, explicit, and complex. Apart from this, TestNG has it’s own default way of developing the reports in the transferrable format. Consequently, this post will combine the following topics:

* ***TestNG Reports’ Dashboard in Eclipse***
  + ***Also, TestNG Console Report***
  + ***TestNG “Report” Section In Eclipse***
* ***How To Generate and View Emailable Report In TestNG?***
* ***Additionally, how To Generate and View Index Report In TestNG?***
* ***How To Use Reporter Class To Generate TestNG Reports?***

Before we try to generate reports that we can view and send outside of Eclipse, let’s explore the Eclipse results dashboard and the meaningful insights it gives.

## TestNG Reports’ Dashboard In Eclipse

It is important to note that we will walk through this tutorial in continuation of the previous tutorial about the [***TestNG test case***](https://www.toolsqa.com/testng/testng-test/). In that tutorial, we ran a test case that would open the browser and close it using Selenium. If you do not know how to run the tests in TestNG, we recommend to read that tutorial first. The test source code looked like this:

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  import org.testng.annotations.BeforeMethod;  import org.testng.annotations.AfterMethod;  import org.openqa.selenium.\*;  import org.openqa.selenium.chrome.ChromeDriver;  import org.testng.Assert;  import org.testng.annotations.\*;    public class TestNG {  WebDriver driver ;  @Test  public void f() {        String baseUrl = "https://www.toolsqa.com/";              System.out.println("Launching Google Chrome browser");          driver = new ChromeDriver();          driver.get(baseUrl);          String testTitle = "Free QA Automation Tools For Everyone";          String originalTitle = driver.getTitle();          Assert.assertEquals(originalTitle, testTitle);    }    @BeforeMethod  public void beforeMethod() {  System.out.println("Starting Test On Chrome Browser");  }    @AfterMethod  public void afterMethod() {  driver.close();  System.out.println("Finished Test On Chrome Browser");  }  } |

When we run this test, there are two separate sections in the Eclipse where these reports are visible.

* ***Console***
* ***Report Section***

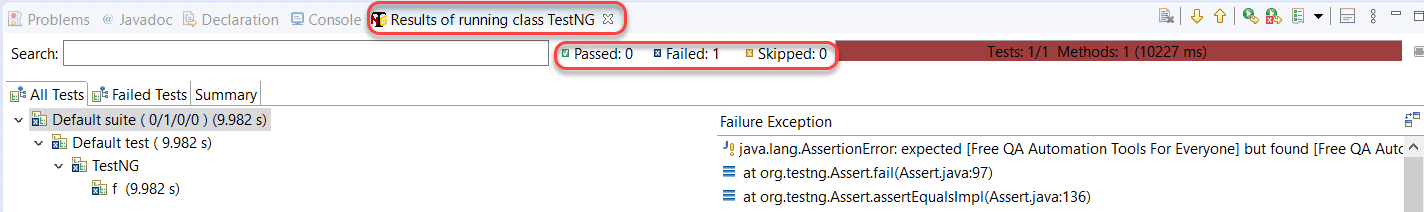
### *****Console Reports In TestNG*****

Console reports in TestNG are short and simple, which just denote the overall summary of the test.

Along with these short stats, the console also shows a bunch of process commands from TestNG that we do not need to bother. If there is any hindrance in running these tests, the error commands display in the console tab only. The console works like a typical console in every language.

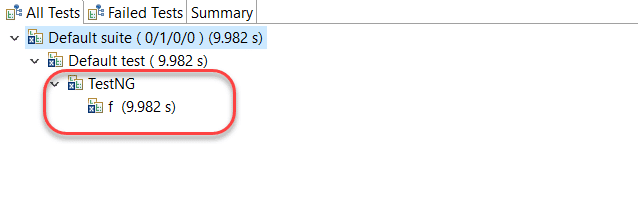
### *****TestNG Report Section In Eclipse*****

Alongside the console tab, the reports tab lies in Eclipse that generates a more in-depth view than what we had in the console.



The top section of the report contains the same summary that we saw in the console part. We have marked it in the above screenshot.

Below the summary, TestNG provides the class name and the function name that were part of the tests.



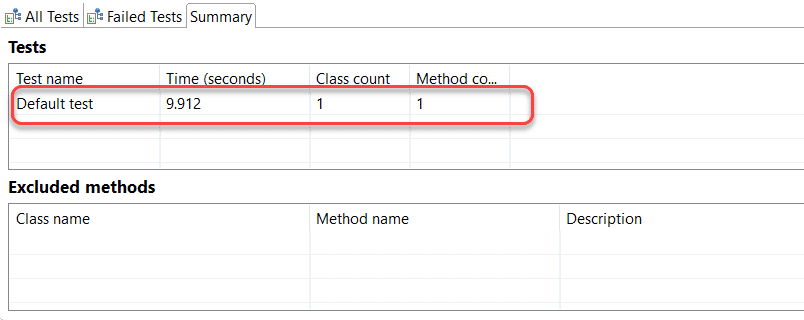
Along with that, the time is taken to execute the “f()” test method is available alongside. It is the default view that comes under the “***All Tests***” part.

***Failed Tests*** and ***Summary*** of the tests are also visible with the different tabs besides All Tests, as shown in the below image.



The above screenshot is of the ***Failed Tests*** tab, and it looks the same as the ***All Tests*** tab. It is because we had one test, and that failed in execution. The UI is similar to both of them.

The summary tab of the TestNG reports section contains a few different things. Click on the “***Summary***” tab:



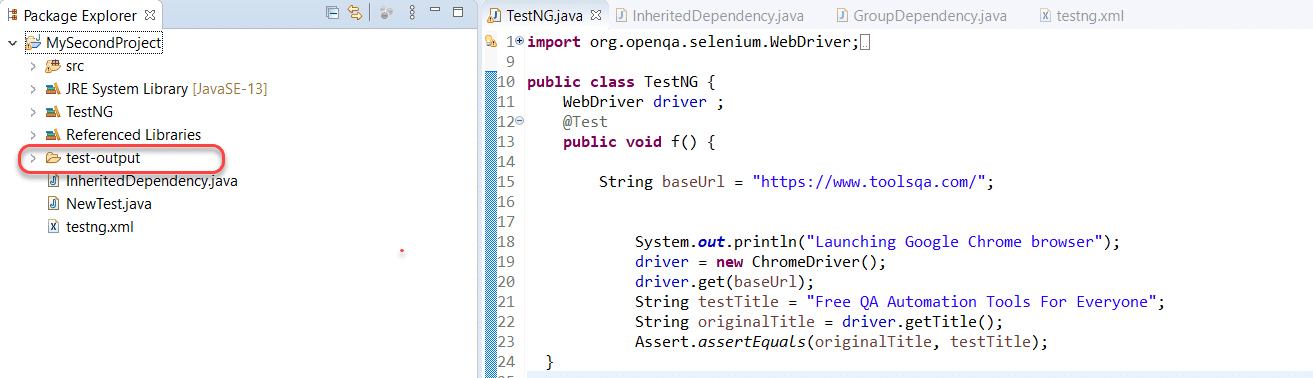
The summary will show a complete summary of the different tests that we executed. If you run a test suite, then there will be multiple tests; otherwise, if you have executed a single test like me, a single test will be seen. The Time (seconds) column will show the total time that took to execute the test with the other two columns showing the count of classes and methods in the tests, respectively.

It was about the Eclipse reports or summary when we run the tests. But, ultimately, we need to email these reports to other team members as well. For this, we need to make use of what is called an “emailable-report” in TestNG. Let’s see how to generate that.

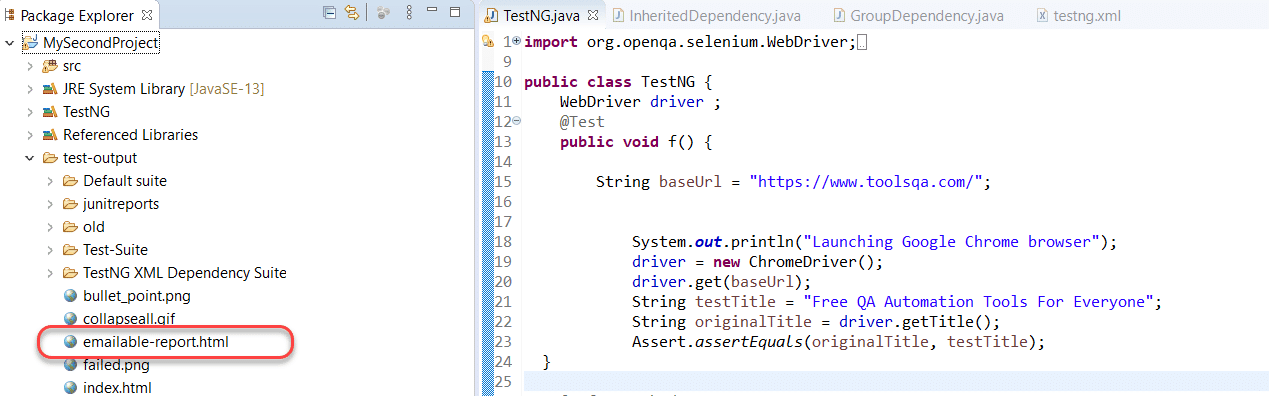
## How To Generate Emailable Report In TestNG?

Emailable reports are generated in TestNG to let the user send their test reports to other team members. Emailable-reports do not require any extra work from the tester, and they are a part of overall test execution. To generate emailable reports, first, run the TestNG test class if you have not already.

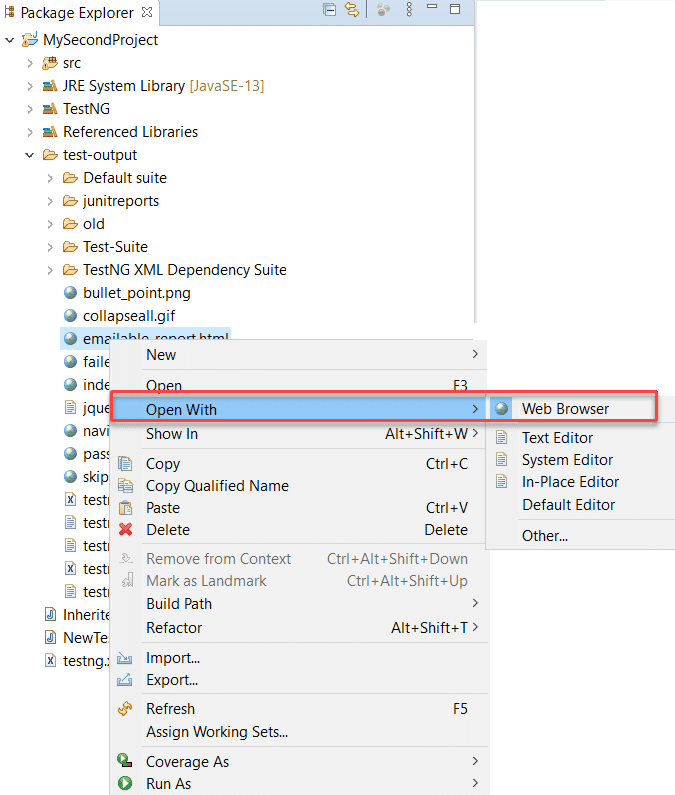
Once we have run the test case, a new folder generates in the same directory with the name ***test-output***.



Explore this folder. It will contain multiple files in it. We will talk about them at different points in this course. For this tutorial, we will focus on an ***emailable-report.html*** file.



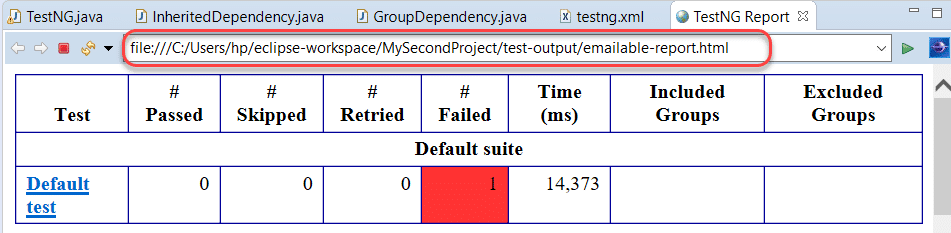
Right-click on the file. Choose ***Open With -> Web Browser***.



It will open the report inside Eclipse. For a better view, copy the URL from the address

bar in Eclipse

.



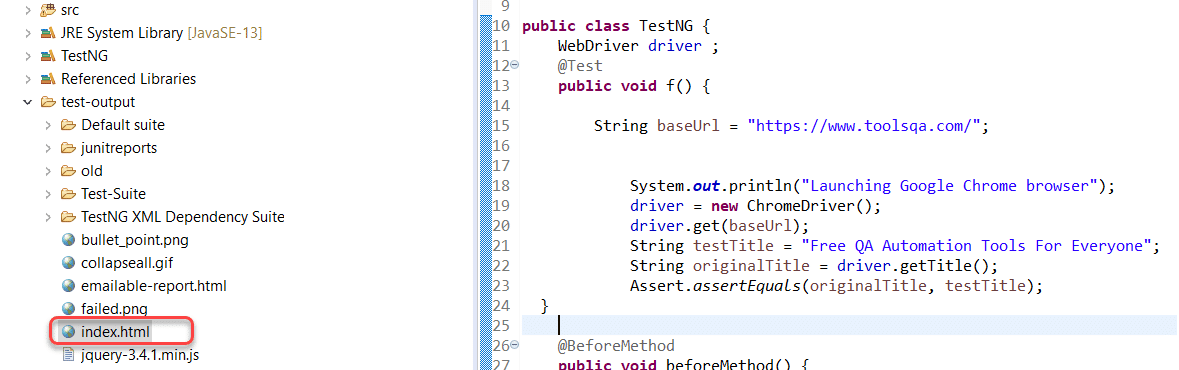
Paste this URL to any browser of your choice to see the emailable-report in a better view.

Another important file that resides inside the test-output folder is ***index.html***. Let’s have a look at that.

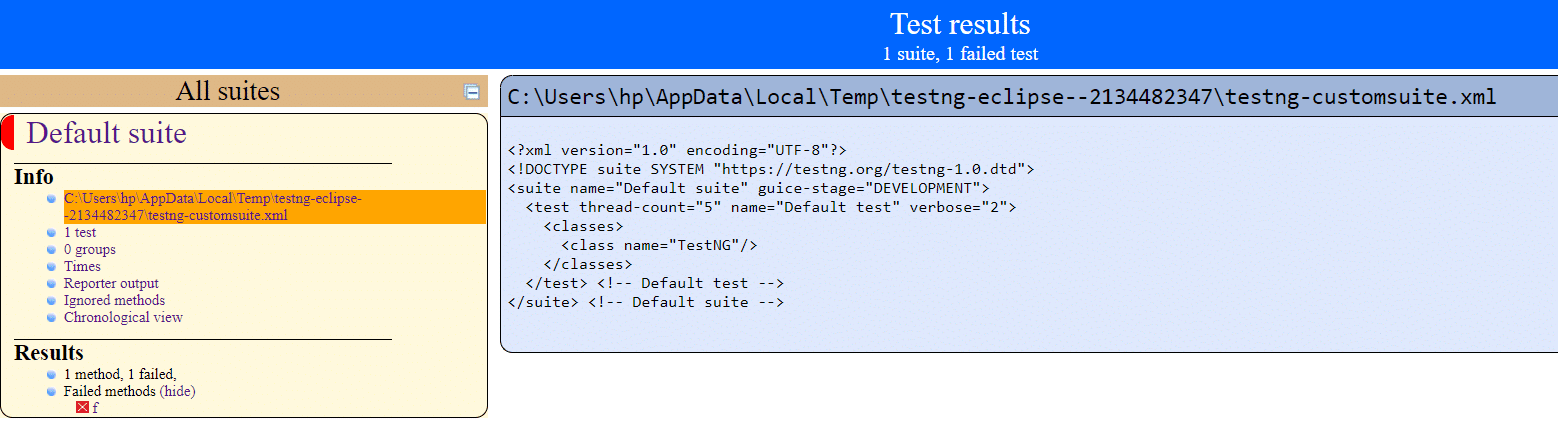
## How To Generate Index File In TestNG?

Emailable reports are a type of summary reports that one can transfer to other people in the team through any medium. Index reports, on the other hand, contains the index-like structure of different parts of the report, such as failed tests, test file, passed test, etc.

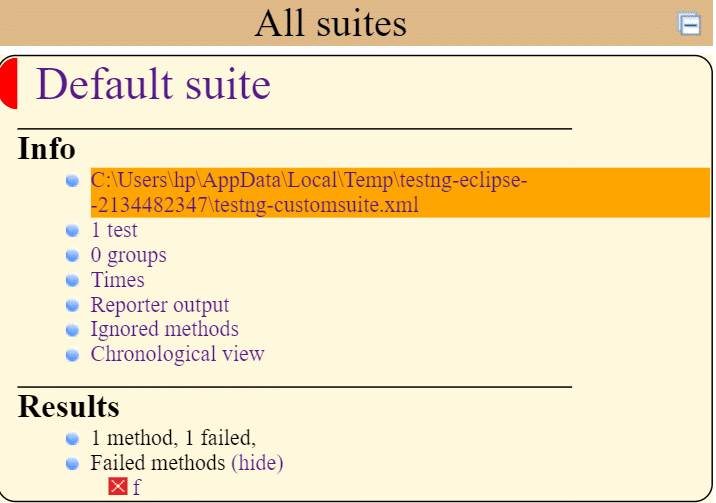
To open the ***index.html*** file, locate it in the same test-output folder.



Similar to the last report, open this one in the browser.



We can divide this report into two parts. ***The left part contains the index,*** and this is the reason it is called an index report, while the right part contains the explored content of that index.



More straightforwardly, whatever you choose in this index is projected on the right side. In the above example, the highlighted XML index projects on the right. It is more handy and user-friendly to operate.

## How To Use Reporter Class To Generate TestNG Reports?

***Reporter class is an inbuilt class in TestNG***. The reporter class in TestNG helps in storing the logs inside the reports that are user-generated or system-generated so that in the future, when we look at the report, we can view the logs directly from there rather than rerunning the test case.

To use the reporter class, we use the following syntax:

***Reporter.log(string);***

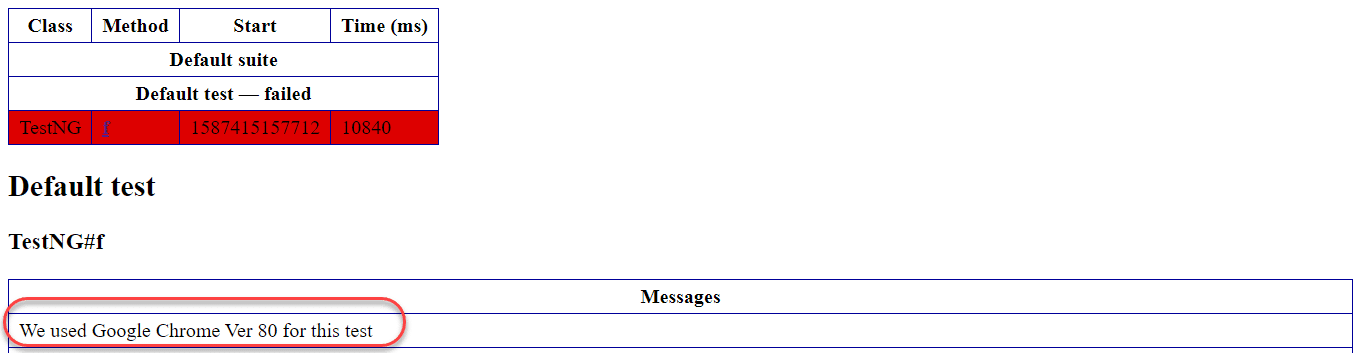
Simply we are calling the “***log***” function of the Reporter class of TestNG.

Observe the following code snippet from the test case code that we used at the start of this tutorial.

|  |  |
| --- | --- |
|  | public void f() {        String baseUrl = "https://www.toolsqa.com/";          System.out.println("Launching Google Chrome browser");          driver = new ChromeDriver();          driver.get(baseUrl);          Reporter.log("We used Google Chrome Ver 80 for this test");          String testTitle = "Free QA Automation Tools For Everyone";          String originalTitle = driver.getTitle();          Assert.assertEquals(originalTitle, testTitle);    } |

We want to log the string “We used Google Chrome Ver 80 for this test” onto the reports. Follow the above-said procedures to run the test case and generate the emailable-report.

Once done, open the emailable report in the browser:



And here we go, we have a message logged into the report for every team member that receives it. Great! It was all about the generation of TestNG reports and viewing them in Eclipse and your browser. Reports serve a similar purpose as your result card in exams. They help us analyze the complete summary without reading a single line of the code. Since reports contain a vast number of elements (including those which are put by the tester in the code), we cannot explain everything in the tutorial or rather this course. Although we will pick up important stuff as we progress. Therefore, I would recommend to give some time on these reports and navigate to every link you see. It would help a lot in the future.

## Common Questions  About TestNG Reports

***Can the tester customize the TestNG reports?***

Yes, the tester is free to customize the [***TestNG***](https://testng.org/doc/) reports according to his will. For this purpose, we use two interfaces in TestNG:

* ***ITestListener Interface***
* ***IReporter Interface***

**Can we generate a PDF report instead of HTML reports in TestNG?**

Yes, TestNG allows generating PDF reports. The tester needs to download external Java APIs for this and read the documentation on how to use them. Moreover, they are readily available over the internet.

***Do TestNG reports need external code to write?***

No, there is no need to write any code to generate reports in TestNG. In other words, the reports generation happens by default.

***What are the two ways to generate a report in TestNG?***

We can generate the TestNG reports in two ways:

* Emailable Reports
* Index Reports

# TestNG Parameters

In the test environment, as well as the development environment, we often create functions. We create ***Functions*** so that we do not have to run or write the same code again and again, which is similar in structure but passes different values as input. For example, if I want to add 2 and 3, I will write the logic. But what if again at some point I want to add 3 and 5, then again 5 and 8 and so on. Should I write the logic every time? No way!! The same logic applies to TestNG. We do not run a single test only once, like in a lifetime. It’s run on different values to give the best results like entering different usernames etc. We use TestNG parameters for this, and they are of various kinds. In this tutorial, we will learn the following concepts in TestNG parameters:

* What are the TestNG Parameters?
  + How to run TestNG Parameters?
  + How to Define TestNG Parameters at the Suite level?
* Overriding power in TestNG Parameters.
* Optional Parameters.

## What are the TestNG parameters?

[***Parameters***](https://en.wikipedia.org/wiki/Parameter_(computer_programming)) in TestNG is similar to [**annotations in TestNG**](https://www.toolsqa.com/testng/testng-annotations/)in their declaration. Similar to parameters in any other [***programming language***](https://www.toolsqa.com/python/what-is-programming/), they are declared to pass some values onto the function. A simple reason to use parameters is that they let us run a function many times with different values or to run different functions with the same values. ***Parameters pass the values in the runtime***. An example of using the parameters in TestNG can be entering different values in an input box. Although you might think that you can change the variable name that is entering the value in the input box, but the test source codes are so long that you would prefer using parameters on any day.

***Syntax:***

***@Parameters ({“a”, “b”})***

where a and b are the values that pass to the function.

As we progress into this tutorial, we will see some additional characteristics of parameters in TestNG. For now, let us see how to run TestNG parameters on a simple class.

### *****How To Run TestNG Parameters?*****

Let’s say we want to add two numbers using the TestNG parameters. Observe the below-given code for the same.

|  |  |
| --- | --- |
|  | import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class Params  {      @Test      @Parameters ({"val1", "val2"})      public void Sum(int v1, int v2) {       int finalsum = v1 + v2;          System.out.println("The final sum of the given values is " + finalsum);      }  } |

Can we run it directly using ***Rus As -> TestNG Test***? Try to run it like this. I am sure it won’t.

**Note:**TestNG Parameters are run through the TestNG XML file and not from the test case files directly.

Head over to the XML file and write the following XML code in it.

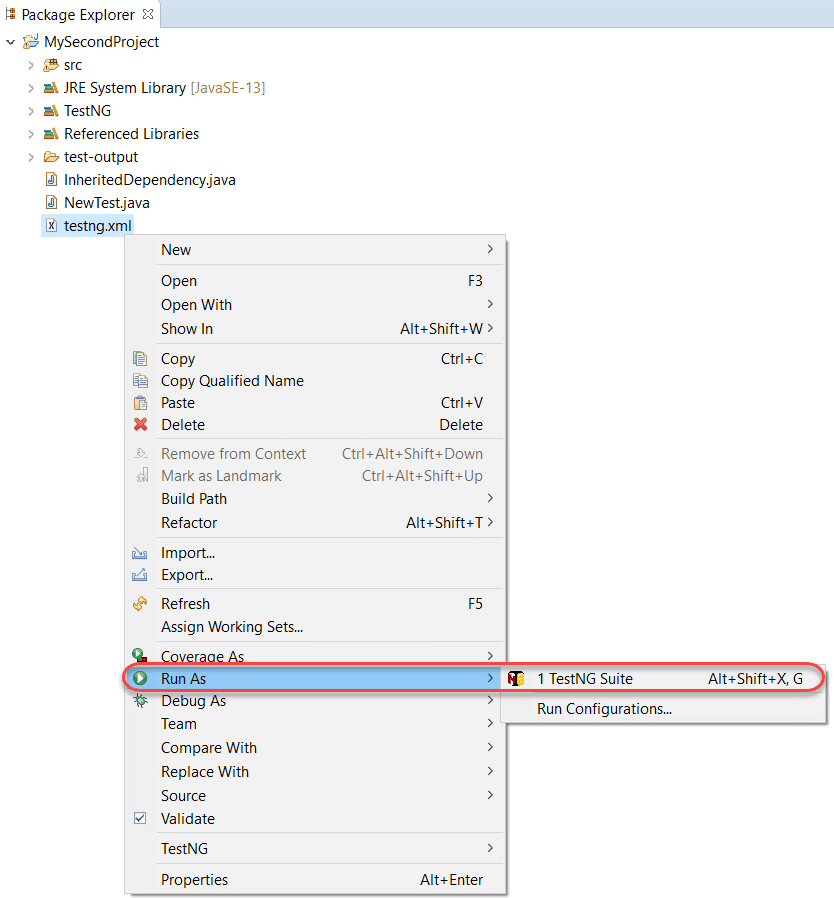
|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="TestNG Parameters Suite">     <test name="Params">        <parameter name="val1" value="2" />        <parameter name="val2" value="3" />        <classes>           <class name="Params" />        </classes>     </test>  </suite> |

In the above XML file, we have defined a tag called parameters which work as follows:

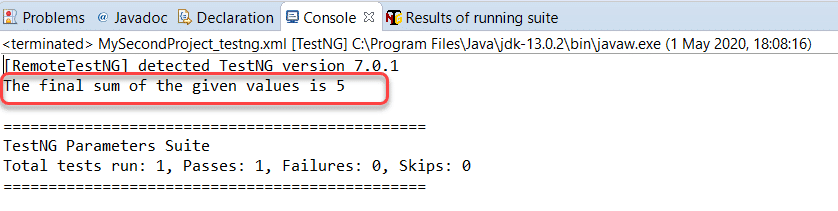
* ***name:*** Name of the variable that you declared in the test case file like val1 and val2 in the above example.
* ***value:*** The value of the variable you want to insert.

***Note***: If you are confused with the suite and test element in the TestNG XML, I request you to refer to the previous tutorials of the [***TestNG Test Suite***](https://www.toolsqa.com/testng/testng-test-suite/) & [***TestNG Test***](https://www.toolsqa.com/testng/testng-test/).

Run the XML file by ***Rus As -> TestNG*** Suite.



The output will be visible in the console.



Now it is so easy just to run this file to see the changes in the test cases.

### *****How to Define Parameters in TestNG at the Suite Level?*****

We can provide the parameters in TestNG at both suite and test levels. In the above test run, we declared parameters at the test case level. In the following code, we will be declaring them on the suite level. To observe this behavior, we will define two different tests along with the parameters.

|  |  |
| --- | --- |
|  | import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class Params  {      @Test      @Parameters ({"val1", "val2"})      public void Sum(int v1, int v2) {       int finalsum = v1 + v2;          System.out.println("The final sum of the given values is " + finalsum);      }        @Test      @Parameters ({"val1", "val2"})      public void Diff (int v1, int v2) {       int finaldiff = v1 - v2;       System.out.println("The final difference of the given values is " + finaldiff);      }  } |

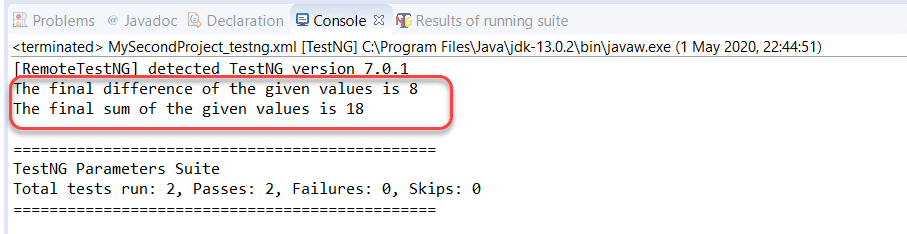
In the above file, the Sum function adds the numbers and Diff function to show the difference between them.

See that we have used the same variable value in both the methods. When we define the parameters at the suite level, our XML file looks as follows:

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="TestNG Parameters Suite">     <parameter name="val1" value="3" />     <parameter name="val2" value="50" />     <test name="Params">        <classes>           <class name="Params" />        </classes>     </test>  </suite> |

**Note:** The parameter tag has moved from being under the <test> to <suite>.

Run this XML file and see that both of the methods would have executed taking the values as 3 and 50, respectively.



As you see, we achieved this by defining the values just once. By defining the parameters in the suite level, you can not only pass the same values in the different methods of the same class but also on different methods in different classes.

Below I have created another file that contains another method called ‘Multiply ‘ to multiply two numbers.

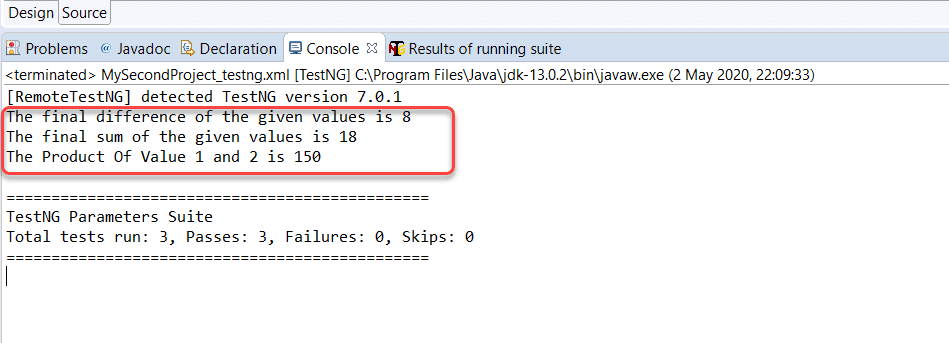
|  |  |
| --- | --- |
|  | import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class Multiply  {        @Test      @Parameters ({"val1", "val2"})      public void mul(int v1, int v2) {       int prod = v1\*v2;          System.out.println("The Product Of Value 1 and 2 is " + prod);      }    } |

***Note:*** that this class also contains the same parameters variable names as in the TestNG class we declared above.

Since we want to run all the three functions (Sum, Diff, and Mul) using the same set of variables, our XML file will look like this:

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="TestNG Parameters Suite">     <parameter name="val1" value="3" />     <parameter name="val2" value="50" />     <test name="Params">        <classes>           <class name="Params" />        </classes>     </test>     <test name="Multiply">        <classes>           <class name="Multiply" />        </classes>     </test>  </suite> |

Execute this TestNG suite to see that all the three methods have run even though being in different classes.



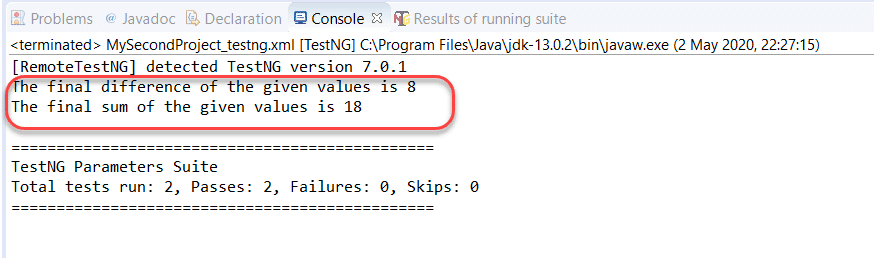
So, we can just declare, pass, and execute variables at a single place only once to run it over any number of files we want. It is convenient. But, you might be wondering what if we declare the same variables at both the levels?

## Order Of Execution In TestNG XML

A tester can declare the variables at the suite level, and test level both, but that would make one of them trivial. Let’s see how does TestNG gives us the results when we encounter such a situation.

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="TestNG Parameters Suite">     <parameter name="val1" value="3" />     <parameter name="val2" value="50" />     <test name="Params">        <parameter name="val1" value="13" />        <parameter name="val2" value="5" />        <classes>           <class name="Params" />        </classes>     </test>  </suite> |

Execute the TestNG suite and observe the results:



By this result, you can already guess which test took preference. TestNG gives preference to the Parameters defined at the test level over the parameters set at the suite level.

## Optional Parameters In TestNG

Optional parameters are yet another type of parameter which kind of acts like the “default” attribute of the switch case in programming languages. ***So, if no parameter value is specified, the optional parameter value is taken.***Optional parameters are defined similarly to Parameters (with annotations), but the place they occupy is different.

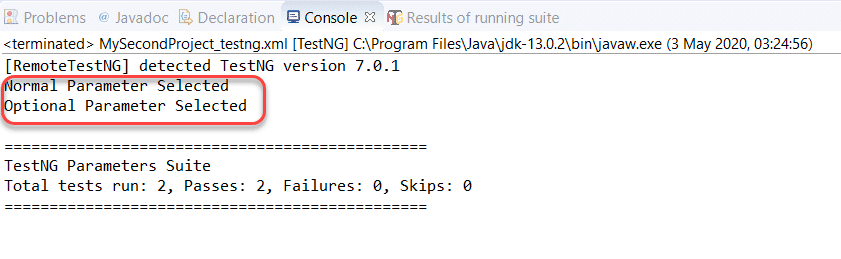
In the following code, I have set a method OP that has an optional parameter notation with the message ***“Operational Parameter Selected.”*** So, if this method gets nothing passed onto the @Parameters, this message will be saved inside the string variable message.

|  |  |
| --- | --- |
|  | import org.testng.annotations.Optional;  import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class Params  {      @Test      @Parameters ("message")      public void OP( @Optional("Optional Parameter Selected") String message) {          System.out.println(message);      }  } |

In the XML file, I have passed the parameters only in the second method.

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="TestNG Parameters Suite">     <test name="Params">        <parameter name="message" value="Normal Parameter Selected" />        <classes>           <class name="Params" />        </classes>     </test>     <test name="Params 2">        <classes>           <class name="Params" />        </classes>     </test>  </suite> |

Run the above TestNG test suite and see how the output comes onto the console.



As expected, the second test executed with the optional parameters as I declared no parameters in the XML file.

It brings us to the end of this tutorial about a very comfortable topic that is very easy to understand. ***@Parameters*** gives us the power to execute a large number of varying tests by declaring the variable values just once. It makes our testing part easy and efficient. Remember that you can use @Parameters annotation with any method that has a @Test, @Before / @After, or @Factory annotation. With this note, we will move onto another type of similar concept called “***Data Provider in TestNG***“.

Category: [TestNG](https://www.toolsqa.com/category/testng/)

# TestNG DataProviders

In the last tutorial, we discussed the [**TestNG Parameters**](https://www.toolsqa.com/testng/testng-parameters/) and how to use them for passing the values to your test class from the XML file. But, there is a problem with TestNG parameters. They worked very well to pass the value and run the tests, but that happens only once per execution. What if I want to run the same test with multiple values? Can I do that with parameters? No, I can’t since this is a very common operation while testing; there needs to be a standard way to accomplish this goal. It is where ***TestNG DataProviders*** come into the picture, and this tutorial will cover just that. Along with the introduction, we will learn the following to use TestNG dataproviders efficiently:

* What are DataProviders in TestNG?
  + DataProvider syntax
* How To Use DataProviders In TestNG?
* Inherited DataProviders in TestNG.
  + MultiValue DataProviders in TestNG.
* DataProviders With Method As Parameters.

## What are DataProviders in TestNG?

The DataProviders in TestNG are another way to pass the parameters in the test function, the other one being TestNG parameters. DataProviders pass different values to the [**TestNG Test Case**](https://www.toolsqa.com/testng/testng-test/)in a single execution and in the form of [**TestNG Annotations**](https://www.toolsqa.com/testng/testng-annotations/). It is a part of the inbuilt TestNG data-driven testing for which TestNG is quite popular. DataProviders help in passing the parameters in different ways. These will be confusing if discussed here. Hence their division into separate sections. The next article will brief you on the syntax of TestNG DataProviders.

### *****DataProvider Syntax:*****

The TestNG DataProvider is used in the following manner:

@DataProvider (name = “name\_of\_dataprovider”)

public Object[][] dpMethod() {

return new Object [][] { values}

}

After the introduction of this syntax, there are a few things that you should take note of before writing a test case:

* 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 list 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.

## How To Use DataProvider In TestNG?

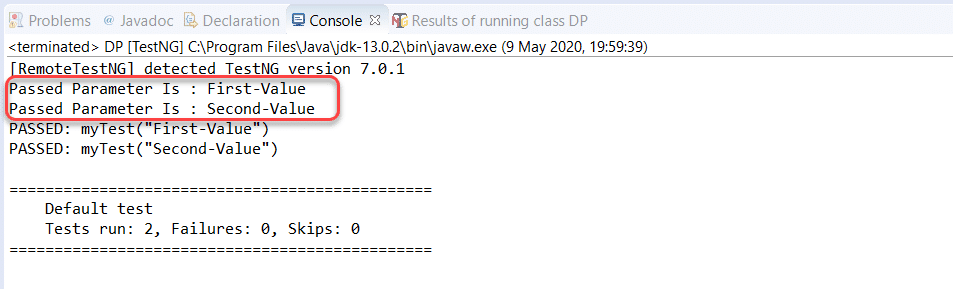
If you have understood the above-said points, using dataproviders is very easy. We will start with a straightforward and basic DataProvider test first. Observe the following code, which contains the @DataProvider.

|  |  |
| --- | --- |
|  | import org.testng.annotations.DataProvider;  import org.testng.annotations.Optional;  import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class DP  {  @DataProvider (name = "data-provider")  public Object[][] dpMethod(){  return new Object[][] {{"First-Value"}, {"Second-Value"}};  }        @Test (dataProvider = "data-provider")      public void myTest (String val) {          System.out.println("Passed Parameter Is : " + val);      }  } |

**Note:** You need to import the DataProvider in TestNG by adding the line import org.testng.annotations.DataProvider;

In the above code, I am trying to pass the values “First-Value” and “Second-Value” to the Test method “myTest” with the help of the DataProvider method “dpMethod().” Please refer to the syntax section to recall the points once again.

Run the code with ***Run As -> TestNG Test*** and see the output.



**Note:**Unlike parameters in TestNG, the dataproviders can be run directly through the test case file.

Both the values appear in the output. It means that even though we ran the file once, the test case method ran twice with different values.

So, even though it is a small and simple code, you might know with how messy codes can get if you have while testing. Wouldn’t it be better if we could declare TestNG dataprovider in another class and our test case into another? It is inheriting the dataproviders.

## Inherited DataProvider In TestNG

Dataprovider and the test case method can also be in two different classes. It is inheriting the dataprovider since we are inheriting it from another file. It’s required to slightly improve the above code to run the test case like this.

The following is the @Test file below :

|  |  |
| --- | --- |
|  | import org.testng.annotations.Test;    public class DataProvider {    @Test (dataProvider = "data-provider", dataProviderClass = DP.class)      public void myTest (String val) {          System.out.println("Current Status : " + val);      }  } |

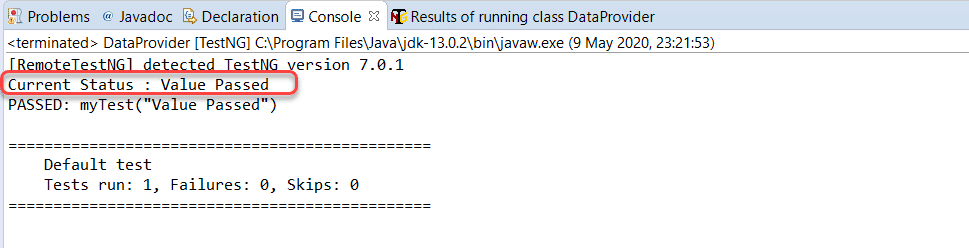
The only difference here is that along with the name of dataProvider; you now need to provide the dataProviderClass by the same attribute name.

Since I told this method that my dataprovider class is DP.java, I will create another file DP.java and write my dataprovider code there.

DP.java looks like this:

|  |  |
| --- | --- |
|  | import org.testng.annotations.DataProvider;  import org.testng.annotations.Optional;  import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class DP  {  @DataProvider (name = "data-provider")  public Object[][] dpMethod(){  return new Object[][] {{"Value Passed"}};  }      } |

Run the test file and see if the output is “Value Passed” or not.



The output says that the variable value “Value Passed” was actually passed to the method. This is great!! In the above codes, though, if you notice, we have just passed a single parameter per execution of the test case. Practically, we need much more. In the next section, we will see how to pass multiple parameters in the TestNG dataprovider.

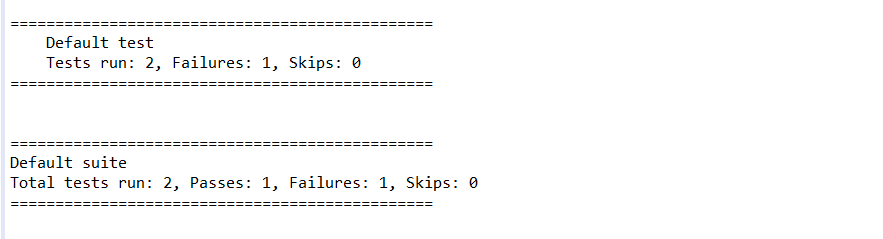
### *****How To Pass Multiple Parameters In TestNG DataProviders?*****

Passing multiple parameters is just similar to the single parameters, but we will be providing multiple values in a single parameter. Observe the following test code, which checks if the sum of two integers is as expected or not.

|  |  |
| --- | --- |
|  | import org.testng.Assert;  import org.testng.annotations.DataProvider;  import org.testng.annotations.Test;    public class DProvider {  @DataProvider (name = "data-provider")  public Object[][] dpMethod(){  return new Object[][] {{2, 3 , 5}, {5, 7, 9}};  }      @Test (dataProvider = "data-provider")      public void myTest (int a, int b, int result) {          int sum = a + b;          Assert.assertEquals(result, sum);      }  } |

In the above code, I have passed three values a,b and result to check if the sum is equal to result or not.

The output is as follows:



Can you guess which test failed?

Alright! So, we have passed multiple parameters into the same test method. As an upgrade to this situation, we can pass the same dataprovider to various test methods also. Just provide the same name in every test method, and you are good to go. You can try that as a practice lesson on your own.

## DataProviders With Method As A Parameter

In the above cases, we have used one way to provide the dataprovider to another test class, i.e., by creating a dataprovider method for each method that will be calling it. It is alright, but we will unnecessarily increase the lines of code in the java file, which is considered a bad coding practice. If I can do the same job for seven lines instead of 10, I should go for it. It is the reason that dataproviders also accept a method as a parameter, and then we can just check the method name and provide the parameters according to it.

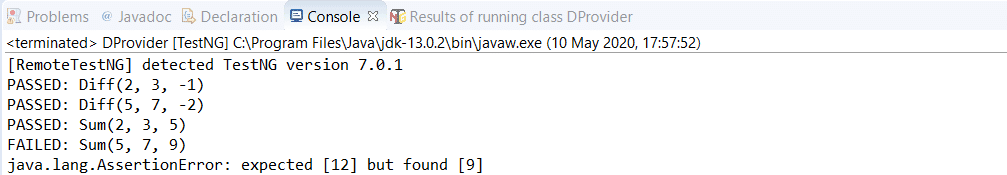
Observe the following code:

|  |  |
| --- | --- |
|  | import org.testng.Assert;  import java.lang.reflect.Method;    import org.testng.annotations.DataProvider;  import org.testng.annotations.Test;  public class DProvider {  @DataProvider (name = "data-provider")  public Object[][] dpMethod (Method m){  switch (m.getName()) {  case "Sum":  return new Object[][] {{2, 3 , 5}, {5, 7, 9}};  case "Diff":  return new Object[][] {{2, 3, -1}, {5, 7, -2}};  }  return null;    }      @Test (dataProvider = "data-provider")      public void Sum (int a, int b, int result) {          int sum = a + b;          Assert.assertEquals(result, sum);      }      @Test (dataProvider = "data-provider")      public void Diff (int a, int b, int result) {          int diff = a - b;          Assert.assertEquals(result, diff);      }  } |

This code provides a switch case to check the name of the method and return the parameters according to the method name.

***Import Required:***import java.lang.reflect.Method;

Run the above code and see how the output compares:



A single execution failed where the expectation was the sum of 5 and 7 to be 9, whose failure was bound to happen. You can see how in 5 lines, I created dataprovider for two different test methods. Notice that for every method added to the given dataprovider code, you just need to add two lines:

case “method” :

        return statement.

which can also happen in a single line. I am doing this instead of adding the 7 – 8 lines typically. You can compare how efficient this is.

It was all from my side on the topic of dataproviders in TestNG. If you get the codes used in this tutorial, there is no need to tell you how efficient dataproviders are in passing the parameters. On the other hand, it would be wrong to assume that they are a better choice than[**Parameters in TestNG**](https://www.toolsqa.com/testng/testng-parameters/). Both of these concepts are used differently and depends on the needs of the tester. So, master both of them with different scenarios and different datasets. We will move onto our next topic now.

## Common Questions On TestNG DataProviders

***How do you give parameters in TestNG?***

There are two ways to pass the parameters in TestNG:

* TestNG Parameters
* TestNG DataProviders

***What is the difference between DataProvider and Parameter in TestNG?***

DataProviders pass the different parameters on a single test in a single execution, whereas parameters pass the parameters just once per execution in [***TestNG***](https://testng.org/doc/).

# TestNG Test Case Priority And Sequence

In the article about running our [**first test case in TestNG**](https://www.toolsqa.com/testng/testng-test/), we learned how test cases need to be ***alphabetically for a serial sequential run*** or else they could execute out of our will. A common problem here is that we cannot work with this flow all the time. Suppose I have a test called ProductTest, which we define before the ConsumerTest, but I want ProductTest to run first because the final result can only evaluate if ProductTest runs first and provides values to ConsumerTest. I need to always think of something lexicographic in order even though it does not make any sense. It is where we harness the power of ***TestNG Test Priority and Test Sequence***, and this post is about that. The table of contents for this post are:

* What Is Prioritization In TestNG?
  + How to give Priority in TestNG Test?
  + How To Run Prioritized Tests In TestNG Using Selenium?
* Same Priority Tests In TestNG
  + Analyzing Test Sequence with Test Priority in TestNG with Selenium
* How To Skip A Test Case In TestNG?

## What Is Prioritization In TestNG?

Prioritization in TestNG is a way to provide a sequence to the methods so that they do not run out of order. Since alphabetically running test cases in TestNG have no logical sequence (concerning the tests and code), providing priority to these test cases helps us managing our tests’ execution.

Priority in TestNG test cases is a parameter with attribute value as “priority.”

### *****How to give Priority in TestNG test?*****

The following is the syntax for allocating a priority to a test case method.

@Test (priority = 1)

public void func(){

//test code

}

Here the test method “func” has a priority of 1.

It is important to note a couple of points regarding priority in TestNG:

* Definition of Priority in TestNG test methods can only be the @Test methods.
* Lower the priority number; higher is the priority of the test case method.
* Priority in TestNG contains only integer value. The value can be negative, zero, or positive.
* If a tester defines a priority in decimal in TestNG, it needs to convert first to Integer (through typecasting).
* One method is allowed to have only one priority in TestNG.
* Priority cannot pass through the XML files.

Keeping these points in mind, we are ready to run our first test with declared priority methods using selenium.

### *****How To Run Prioritized Tests In TestNG Using Selenium?*****

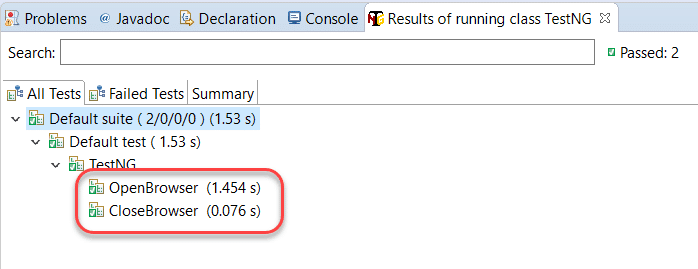
Writing a test case with priority in TestNG is similar to how we write a typical [***test case in TestNG***](https://www.toolsqa.com/testng/testng-test/) but with a “priority” attribute.

Observe the following code, which has two methods: OpenBrowser and CloseBrowser.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 1)  public void CloseBrowser() {  driver.close();  System.out.println("Closing Google Chrome browser");  }    @Test (priority = 0)  public void OpenBrowser() {  System.out.println("Launching Google Chrome browser");  driver.get("https://www.demoqa.com");  }  } |

In the OpenBrowser method, I am trying to open the browser and enter the URL “www.demoqa.com.” The “CloseBrowser” method, however, is used to close the driver. The priorities set are **0 for OpenBrowser** and **1 for CloseBrowser**, so I expect the OpenBrowser method to run first.

Execute the above TestNG test file to check the output.



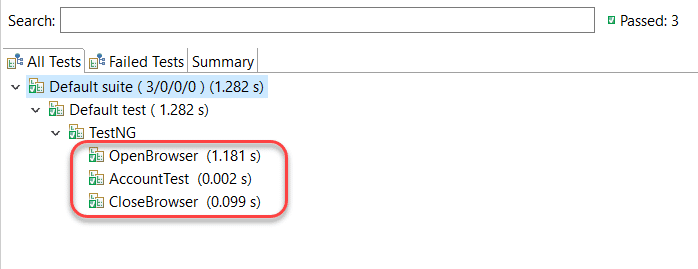
As expected, the OpenBrowser method ran first because of a lower priority. ***Had I not declared the priority here, it would have run alphabetically,*** i.e., CloseBrowser first and then OpenBrowser.

## Tests with Same Priority in TestNG

If priority is deciding the sequence of tests in TestNG, then a simple question arises in our minds: what if I declare the same priority to all the tests in TestNG? Let’ see this case by observing the following code:

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 1)  public void CloseBrowser() {  driver.close();  System.out.println("Closing Google Chrome browser");  }    @Test (priority = 0)  public void OpenBrowser() {  driver.get("https://www.demoqa.com");  System.out.println("Launching Google Chrome browser");  }    @Test (priority = 1)  public void AccountTest(){  System.out.println("Some tests for Customer Account");  }  } |

In the above code, I have set the priority of AccountTest and CloseBrowser as one while OpenBrowser retains its priority of 0. Run this code to see the output:



The OpenBrowser ran first, no questions there!! But, an interesting pattern appears in the methods containing similar priorities (1). Even though the CloseBrowser method was declared first, AccountTest was the one that ran before CloseBrowser. A good explanation for this behaviour is that ***if two or more methods have the same priorities in TestNG, then their running test sequence is alphabetic***. Since “A” comes before “C,” the method AccountTest ran first.

But let’s wait and think at this moment for a second by recollecting the memories of test cases and priorities. If all the test cases with no priorities run alphabetically and all the test cases with similar priorities also run alphabetically, then there must be some relation between these two situations.

Can you guess it by shooting in the air?

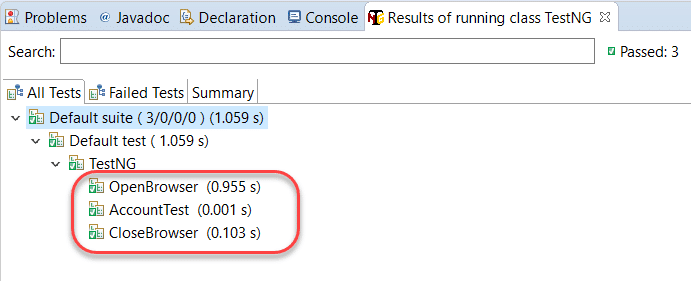
***The test methods with no priority assigned have a default priority equal to 0. It means if we define no priorities, all the test methods will be assigned priority 0, and a similar priority case will apply.*** Let’s prove this fact with some test code in the next section.

### *****Analyzing Test Sequence with Test Priority In TestNG with Selenium*****

The below-given code is the same as the code we used above. But this time, I have reassigned the priorities of all the methods.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 0)  public void CloseBrowser() {  driver.close();  System.out.println("Closing Google Chrome browser");  }    @Test (priority = -1)  public void OpenBrowser() {  System.out.println("Launching Google Chrome browser");  driver.get("https://www.demoqa.com");  }    @Test  public void AccountTest(){  System.out.println("Some tests for Customer Account");  }  } |

In the above test code, the method OpenBrowser contains priority as -1, CloseBrowser as 0, and no priority assignment happens to AccountTest. Let’s see the output after running the above selenium code in Eclipse.



Looking at the output of this test code, we prove three main points in TestNG priority:

1. We can assign negative priorities to a method.
2. With a method with no priority, the priority is set to 0 by default.

Observe that the AccountTest method ran before CloseBrowser even without having any priority because both sets to priority = 0, and hence, they run alphabetically.

Hence, we can change the sequence of tests in TestNG using priorities.

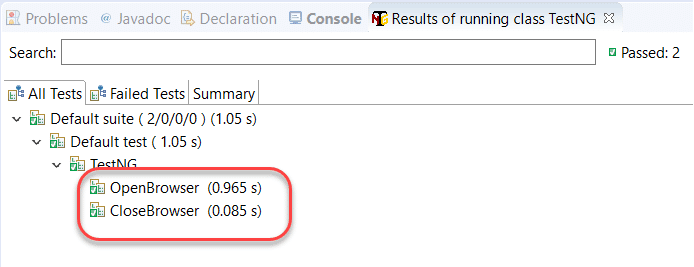
## How to Skip tests in TestNG using Parameters?

In this tutorial, until now, we learned that we could give priority to the test cases and change the sequences of the test methods execution. If we do not, TestNG assigns the priority as zero. But, still, with or without the priority, the method will execute. Often, we are required just to skip a test case method and perform testing. It is skipping, and we carry it through the “enabled” parameter.

Let’s see the following code meant to skip the test CloseAccount.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 0)  public void CloseBrowser() {  driver.close();  System.out.println("Closing Google Chrome browser");  }    @Test (priority = -1)  public void OpenBrowser() {  System.out.println("Launching Google Chrome browser");  driver.get("https://www.demoqa.com");  }    @Test (enabled = false)  public void AccountTest(){  System.out.println("Some tests for Customer Account");  }  } |

Execute the above test case code and see the output.



There you go. We have skipped a test case and broke the typical sequence of the test case in TestNG.

Conclusively, it was all from my side in this tutorial about priorities and sequencing in TestNG. Test priorities are very useful in running the code in the sequence we want with minimum to no changes in the code part, like shuffling the functions. Additionally, it is a light yet powerful topic in TestNG. Subsequently, we will move on to our next tutorial now.

## Common Questions On Priority In TestNG

***How do you give priority in TestNG?***

A tester can provide a priority value to the test case by ***defining the priority parameter*** with @Test annotation. Moreover, if there is no priority defined, the default priority is zero (0) for that test case.

***Can we give a negative priority in TestNG?***

Negative priorities are acceptable in [***TestNG***](https://testng.org/doc/). However, you can provide an integer value to the priority parameter, including zero.

# TestNG Reporter Log

In the tutorial about [**generating reports in TestNG**](https://www.toolsqa.com/testng/testng-reports/), we discussed how we could create different types of reports in TestNG. The report types included the ***emailable reports*** and ***index reports*** with steps included in Eclipse while using selenium tests. So, are we discussing something different here other than reports? Actually, no! ***TestNG reporter is an inbuilt class in TestNG, which helps in logging the messages in the output reports***. Since logs play a vital role in every tributary of computer science, logging with ***TestNG Reporter Logs*** helps the testers and other team members analyze the details of the code without actually looking at the test code. Consequently, this tutorial will guide you through the following topics:

* What is the Reporter Class in TestNG?
  + Syntax for using Reporter class
* How to Log Messages in Reports using TestNG Reporter Log?
  + Additionally, how To View Logged Messages In Emailable Report In TestNG?
  + How To View Reporter Logs In The Console In TestNG?

## What is a Reporter class in TestNG?

Reporter class in [TestNG](https://testng.org/doc/documentation-main.html) helps the testers log the messages on to the test execution reports. These messages commonly denote the status, information, or any other detail about the test execution. But, with no hard and fast rules, these messages can be anything. A straightforward example of logging with the TestNG reporter log can be logging the chrome driver version used for testing with selenium or logging the inputs you provided into the function etc. Reporter class is elementary to use with and requires just an import into the test code.

### *****Syntax For Using TestNG Reporter Log:*****

Reporter.log(“message”);

**Import Required:***import org.testng.Reporter;*

That’s it! You just have to call the log function of the reporter class, and the message passed will be logged onto the final reports. ***Remember to import the class as the reporter class is a separate class available in the org.testng package.*** It needs to be imported first before using it.

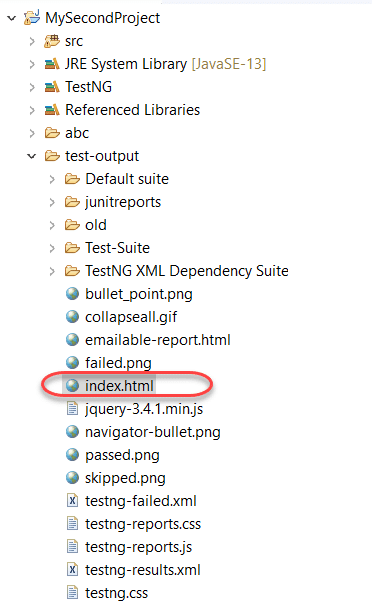
## How to Log messages with TestNG Reporter Log using Selenium?

In this section, I will demonstrate how to log the messages in the reports using the reporter class in TestNG. Observe the following code:

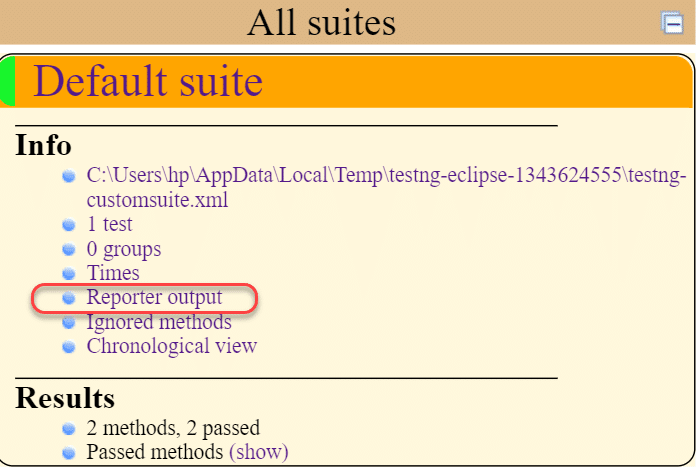
|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.Reporter;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 0)  public void CloseBrowser() {  driver.close();  Reporter.log("Driver Closed After Testing");  }    @Test (priority = -1)  public void OpenBrowser() {  Reporter.log("This test verifies the current selenium compatibility with TestNG by launching the chrome driver");          Reporter.log("Launching Google Chrome Driver for this test");            driver.get("https://www.demoqa.com");            Reporter.log("The website used was DemoQA for this test");    }  } |

Notice all the ***Reporter.log*** messages typed with the code.

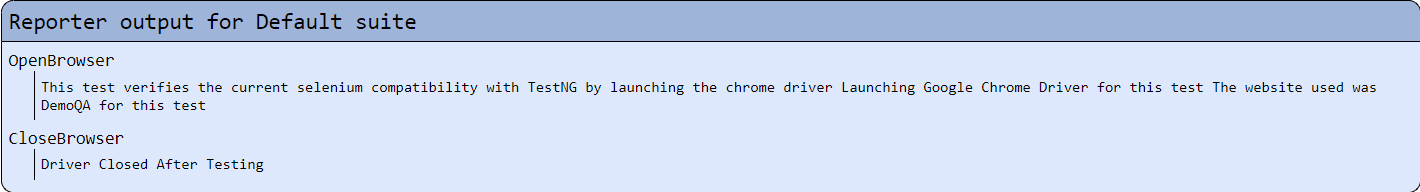
Once you run the code, you will notice that nothing appears on the console. It is because the reporter class has logged the messages on to the reports only. Go to the test-output folder and open index.html report, as shown in the following image:



Open the index.html report in the browser and move to the ***Reporter Output*** section in the report.



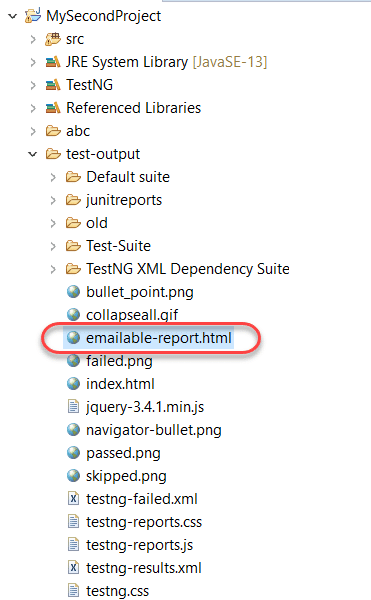
It will open a panel with all the logged messages visible in the panel.



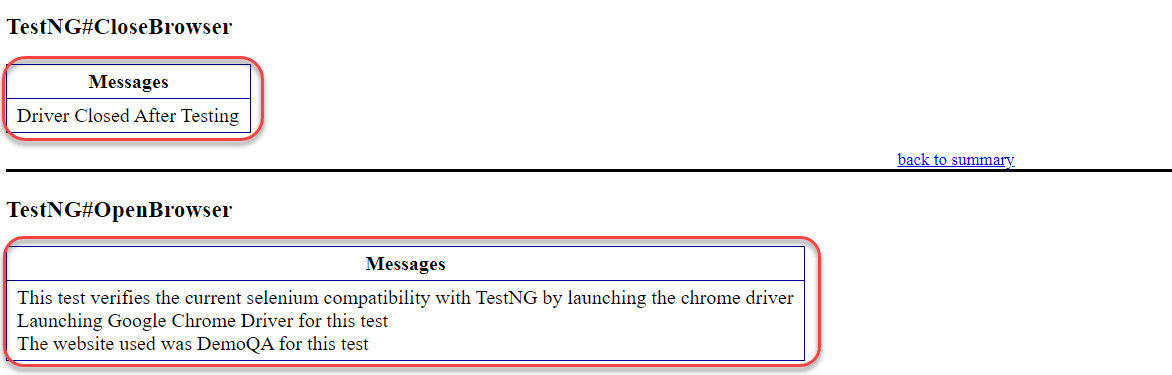
In this way, you can see the logged messages concerning every method in the test case class. Can we see similar outputs in the other type of report?

### *****How to view Log messages in TestNG Emailable Report?*****

The logged messages are also available in the emailable report similar to the index report. Once you have run the code in eclipse, open the emailable-report from the test-output folder.



Open this report in the browser.



A nicer version of what we saw in the index reports is available here. So, as you must be thinking, the messages are available only in the reports and do not appear in the console. You can check your console! Can we make some adjustments in the same regard?

### *****How to view TestNG Reporter Logs in the Console?*****

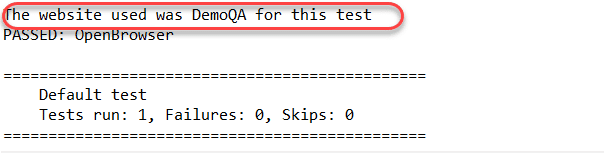
The log function of the reporter class can also log the messages onto the console simultaneously while logging the messages on the reports. For this, we need to pass an additional boolean parameter. Observe the following syntax for the same:

Reporter.log(“message”, true);

We can see it in the following code:

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.Reporter;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = -1)  public void OpenBrowser() {  Reporter.log("This test verifies the current selenium compatibility with TestNG by launching the chrome driver");          Reporter.log("Launching Google Chrome Driver version 81 for this test");            driver.get("https://www.demoqa.com");            Reporter.log("The website used was DemoQA for this test", true);                  driver.close();    }  } |

In the above code, we have just mentioned the boolean value “true” once with the message, “The website used was DemoQA for this test.” Run this test and check the console:



The message gets logged into the console as well in the reports.

The reporter class provides four methods in general:

1. Reporter.log(String s);
2. Reporter.log(String s, Boolean logToStandardOut);
3. Next is, Reporter.log(String s, int level);
4. And, Reporter.log(String s, int level, Boolean logToStandardOut);

Logging the messages is an essential task while performing the testing. It is considered a perfect testing practice in the community. You can imagine a scenario where you are stuck somewhere while testing and wants to take the help of the community online. To make them understand the situation, you paste your code so that they can review it and provide support. Through the provided logged messages, the testers community will be able to understand within seconds what you are trying to do rather than understanding the codes. Now, strategically, we can divide logging into two parts: ***low-level logging***and***high-level logging***.

***Low-level logging is when you want to log every step you are taking while testing***. [***Log4J***](https://www.toolsqa.com/selenium-webdriver/keyword-driven-framework/log4j-logging/) is a popular choice for that. ***High-level logging is when we just log the critical steps that the tester wants***, which the TestNG Reporter Log does.

It was all from my side on TestNG Reporter Logs. In the next tutorial, we will understand the [***TestNG Asserts***](https://www.toolsqa.com/testng/testng-asserts/) and how to use them efficiently in TestNG.

# TestNG Asserts

In this [**TestNG tutorial series**](https://www.toolsqa.com/testng-tutorial/)up till now, we have executed a lot of tests in selenium many TestNG functions. But during these executions, we never declared a test “passed” or “failed,” and of course, we have not studied any formula to do that. Since we are now well versed with all the significant concepts in TestNG, it is time to execute some actual tests using [***selenium webdriver***](https://www.toolsqa.com/selenium-tutorial/) in TestNG. For the same purpose, we will use ***TestNG Asserts***.

TestNG asserts are the most frequently used methods in TestNG and are so common that it is hard to find a TestNG code without the asserts. TestNG asserts the tester decides whether the test was successful or not, along with the exceptions. This post brings you all out of assert with detailed explanations focussing on:

* What are Assertions in TestNG?
  + Syntax for TestNG Assertions
* How to use Assertions in TestNG?
  + How to use messages as a parameter in TestNG Asserts?
* Different types of Asserts in TestNG
  + Hard Asserts
  + Soft Asserts
* How to use Soft Assert in TestNG?
* Commonly used TestNG Assert methods

## What are Assertions in TestNG?

***Assertions in TestNG are a way to verify that the expected result and the actual result matched or not***. If we could decide the outcome on different small methods using assertions in our test case, we can determine whether our test failed or passed overall. An example of assertion can be logging into the website, checking the title of the webpage, verifying the functionality of an input box that takes only integers, etc.

We should remember that an assertion in TestNG is successful only if there are no exceptions thrown during the test case execution. TestNG asserts (or assertions) popularly validate the results in TestNG using selenium.

### *****Syntax for TestNG Assertions:*****

Although there are many methods for assertions (later in this article), the generic syntax is:

***Assert.Method( actual, expected)***

The parameter as you see contains three values:

* ***Actual:*** The actual value that the tester gets like if the tester’s assertion is on the title of the page then what was the actual title of the page goes here.
* ***Expected:*** The value that you expect like if the tester’s assertion is on the title of the page then what value of title do you expect goes here.

Let’s explore the assert in TestNG by writing a simple test case.

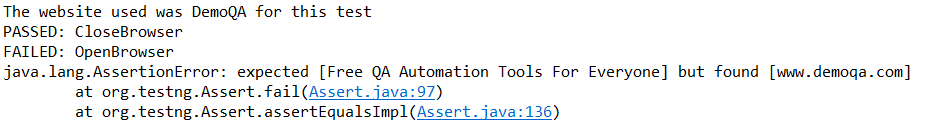
## How to use Asserts in TestNG using Selenium?

As mentioned above, the assert statements just require the implementation of assert methods. However, since we perform assertions to verify the values, we will pass those as a parameter as “actual\_value” and “expected\_value.” In the following code, we will try to assert conditions on the web page’s title name using selenium webdriver.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.Reporter;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;  import org.testng.Assert;    public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 0)  public void CloseBrowser() {  driver.close();  Reporter.log("Driver Closed After Testing");  }    @Test (priority = -1)  public void OpenBrowser() {  Reporter.log("This test verifies the current selenium compatibility with TestNG by launching the chrome driver");          Reporter.log("Launching Google Chrome Driver version 81 for this test");            driver.get("https://www.demoqa.com");            Reporter.log("The website used was DemoQA for this test", true);          String expectedTitle = "Free QA Automation Tools For Everyone";          String originalTitle = driver.getTitle();          Assert.assertEquals(originalTitle, expectedTitle);    }  } |

We have a combination of features of TestNG in the above code, such as [**TestNG Annotation**](https://www.toolsqa.com/testng/testng-annotations/), [***TestNG priority***](https://www.toolsqa.com/selenium-webdriver/testng-prioritizing-sequencing/)***,*** and [***TestNG Reporter***](https://www.toolsqa.com/testng/testng-reporter-log/). You can visit their respective tutorials to know their work.

Run the above-written code to execute the TestNG asserts.



So, our test method OpenBrowser failed because of the expected title, and the actual titles were different. It determined that our test failed. Since selenium tests require a lot of validations, you can practice using different things in the assert methods using TestNG.

### *****How to use Message as a Parameter in TestNG Asserts?*****

In the above example, you may notice that an AssertionError is thrown by Java which is quite complete in its own form. But, in TestNG asserts, we can mention a message as well in the parameter that will be displayed when the assertion fails along with the assertion error. So, as a slight modification to the syntax, this will work as a third parameter making syntax as follows:

***Assert.Method( actual, expected, message)***

The parameter as you see contains three values:

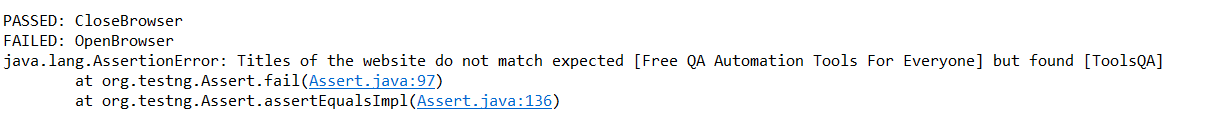
* ***Actual:*** The actual value that the tester gets like if the tester’s assertion is on the title of the page then what was the actual title of the page goes here.
* ***Expected:*** The value that you expect like if the tester’s assertion is on the title of the page then what value of title do you expect goes here.
* ***Message***: A string message to display only in case of an error when the assert fails.

Let’s explore the assert in TestNG by writing a simple test case.

The below-given code is exactly the same code as mentioned in the previous section. Therefore, I have mentioned just the OpenBrowser function since the changes are made only in this part of the code. In the following code, we have used “***Titles of the website do not match***” as the message parameter which is expected to be seen on the console if the assertion fails.

|  |  |
| --- | --- |
|  | public void OpenBrowser() {            driver.get("https://www.demoqa.com");            Reporter.log("The website used was DemoQA for this test", true);          String expectedTitle = "Free QA Automation Tools For Everyone";          String originalTitle = driver.getTitle();          Assert.assertEquals(originalTitle, expectedTitle, "Titles of the website do not match");    } |

Run this on Eclipse to find the message along with the assertion exception error thrown by TestNG (Java).



As expected, the message that we passed is visible. Applying a message to the assertion makes it more meaningful and easy to understand for other team members.

## Different types of Asserts in TestNG

There are two types of TestNg Assert:

* ***Hard Assert***
* ***Soft Assert***

### *****Hard Assert in TestNG*****

***Hard Asserts*** are those asserts that stop the test execution when an assert statement fails, and the subsequent assert statements are therefore not validated. It plays a vital role in projects where we have an element without whose validation, asserting other elements is useless. One good example in such cases is the login functionality. If I want to see my past orders, for example, then what is the point of checking this test case when the login validation already failed? ***Hard asserts are the default type of asserts in TestNG,*** and what we used in the previous section was Hard Assert.

### *****Soft Assert in TestNG*****

Soft asserts are just the opposite of hard asserts. In soft asserts, the subsequent assertions keep on running even though one assert validation fails, i.e., the test execution does not stop. ***Soft assert does not include by default in TestNG***. For this, you need to include the package **org.testng.asserts.Softassert.**So, when should we use soft asserts in TestNG? We use soft asserts when we do not care about the failure of specific validations and want the test execution to proceed and also want to see the exception errors.

A good example is multiple validations on an input form. Also, to note that on many platforms, you will see “verify” while learning about asserts. ***Soft asserts are also known as***“Verify” and hence do not get confused about the same.

## How to use Soft Assert in TestNG?

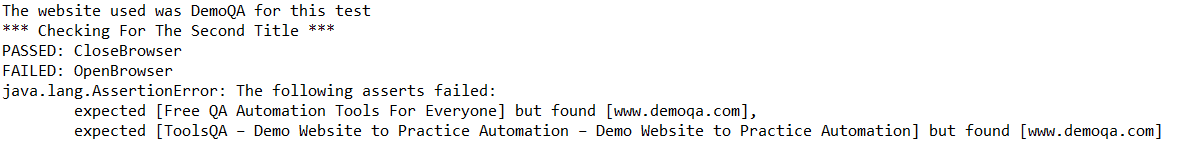
The following code will demonstrate the use of soft asserts in TestNG. In this code, we are validating the title of the web page, bypassing two different expected titles.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.WebDriver;  import org.testng.asserts.SoftAssert;  import org.testng.Reporter;  import org.testng.annotations.Test;  import org.openqa.selenium.chrome.ChromeDriver;  public class TestNG {  WebDriver driver = new ChromeDriver();    @Test (priority = 0)  public void CloseBrowser() {  driver.close();  Reporter.log("Driver Closed After Testing");  }    @Test (priority = -1)  public void OpenBrowser() {  Reporter.log("This test verifies the current selenium compatibility with TestNG by launching the chrome driver");          Reporter.log("Launching Google Chrome Driver version 81 for this test");            driver.get("https://www.demoqa.com");          SoftAssert softassert = new SoftAssert();          Reporter.log("The website used was DemoQA for this test", true);          String expectedTitle = "Free QA Automation Tools For Everyone";          String originalTitle = driver.getTitle();          softassert.assertEquals(originalTitle, expectedTitle);          System.out.println("\*\*\* Checking For The Second Title \*\*\*");  // Checking title for ToolsQA – Demo Website to Practice Automation – Demo Website to Practice Automation          softassert.assertEquals(originalTitle, "ToolsQA – Demo Website to Practice Automation – Demo Website to Practice Automation" );          softassert.assertAll();    }  } |

Important points to remember concerning soft assert that we can notice in the above code are:

* Soft assert requires the external import of the package ***import org.testng.asserts.SoftAssert;***.
* An object of the SoftAssert runs the assert statements.
* The object should have a life within the same test method in which we declared it.
* object.assertAll() statement is required to see the exceptions; otherwise, the tester won’t know what passed and what failed.

Run the above code and see the output:



It shows that our assertions executed and which ones failed. But the test execution did not stop.

## Commonly used TestNG Assert Methods

All the TestNG Assert statements work in the same vein when we talk about the basic structure of its execution. But, like any other method, it can take different parameters and perform various types of validations in the test case methods. Since asserts are so crucial in TestNG and used so commonly, we will discuss some of the most common assertion methods.

* ***Assert.assertEqual***(String actual, String expected): Pass the actual string value and the expected string value as parameters. Validates if the actual and expected values are the same or not.
* ***Assert.assertEqual***(String actual, String expected, String message): Similar to the previous method just that when the assertion fails, the message displays along with the exception thrown.
* ***Assert.assertEquals***(boolean actual, boolean expected): Takes two boolean values as input and validates if they are equal or not.
* ***Assert.assertTrue***(condition)***:*** This method asserts if the condition is true or not. If not, then the exception error is thrown.
* ***Assert.assertTrue***(condition, message):  Similar to the previous method with an addition of message, which is shown on the console when the assertion fails along with the exception.
* ***Assert.assertFalse***(condition): This method asserts if the condition is false or not. If not, then it throws an exception error.
* ***Assert.assertFalse***(condition, message): Similar to the previous method but with an addition of a message string which is shown on the console when the assertion fails, i.e., the condition is true.
* ***public static void assertEquals*** ( Object actual, Object expected, String message): Asserts whether the two objects passed are equal or not. If not, the message and the exception error appears. The message parameter is optional.
* ***public static void assertEquals***(String actual, String expected, String message): Asserts whether two strings are equal or not. If not, the message along with the exception error displays. The message parameter is optional.

And there are many more which you can easily find on the internet (I recommend the https://www.javadoc.io/doc/org.testng/testng/6.8.17/org/testng/Assert.html for the same), but the ones above are the most common. TestNG asserts are very important for someone working on TestNG. They are the core of test case methods, and since a lot of the times we are using TestNG with selenium projects, they help us reduce the chances of error to a minimum. As far as I know, making efficient use of asserts in a test case is a sign of a perfect tester. So keep practicing TestNG asserts, and with this note, we will head on to our next tutorial.

## Commonly Questions on TestNG Asserts

***How do I implement verify in***[***TestNG***](https://testng.org/doc/documentation-main.html)***?***

Verify is another name for soft asserts. Soft asserts can implement in the test code with the help of SoftAssert class in TestNG. Read this tutorial to know more.

***What does “assert fail” mean in TestNG?***

Assert fail refers to the failure of the assertion test method. The conditions for failing depends totally on the assertion methods. When an assertion fails, they throw an exception error onto the console describing the failed test (only in hard asserts).

***Can we apply the assert statements inside the if-statements?***

Yes, assert statements are just like any other methods that return a boolean true or false value. A tester can leverage the return value and can use assert statements at any place in the code.

# Cross Browser Testing using TestNG

Multi-browser testing or cross-browser testing is the essential requirement for your website in current times. With so many browsers in a race to capture the market today, the time when internet explorer ruled the world has gone. Today, the user could be sitting in front of any browser, and as a developer, you cannot just say, “well, let’s suppose all the users are on chrome.” Since the development of such types of websites is so important, so is their testing. For testing the websites efficiently that takes control over the browser drivers, we need Selenium. For using Selenium efficiently, we need [***TestNG***](https://www.toolsqa.com/testng/what-is-testng/). Hence, in this tutorial, we will be going through the process of ***Cross Browser Testing using TestNG with Selenium***. This tutorial will include:

* What is Cross-Browser Testing?
  + Need for Cross-Browser Testing.
  + A practical example of Cross-Browser Testing in different browsers.
  + How to perform Cross-Browser Testing using TestNG?

## What is Cross-Browser Testing?

[***Cross-browser testing***](https://www.toolsqa.com/cross-browser-testing/what-is-cross-browser-testing/) is the process of testing our website on different browsers and operating systems. With cross-browser testing, we make sure that the site is rendered the same in every browser. We can perform cross-browser testing either manually or in an automated way, but the manual method is very tedious. The reason being that while performing cross-browser testing, we do not only care about the browsers but their different versions and the operating systems too. So just imagine the permutations of so many browsers with so many versions (Chrome is on 83) and operating systems. Thus, a better way is to choose an automated way.

We perform the automated cross-browser testing with the help of Selenium and TestNG, but before learning the code, it’s better to understand why we perform cross-browser testing.

### *****Need for Cross-Browser Testing*****

As I mentioned above, nobody knows what browser user loves to open the websites. It can be Google Chrome, Safari, or Opera Mini. Consider you, as a developer, have Google Chrome on your desktop on which you are developing the website. Since you have been developing the website concerning Chrome, it will render correctly on the browser. But now, when you publish the site, you are assuming that every one of your users is on Google Chrome. You must keep in mind that Google Chrome has a 62% share in browser statistics. So, you just lost 38% of your users without even letting them see the product.

Additionally, what if you incorporated some tags that were supported by Chrome in only its latest version? It will hit more than half of that 62%, and we have not yet taken into account the resolutions and operating systems. So, why does it happen?

With the race of capturing more and more customers, every browser manufacture tends to do something new with its product. It includes the engine on which you build the browser. ***Since the engine is different, the way a browser understands the web languages (HTML, JavaScript, and CSS) is different***. So, the position property of  CSS would run fine on Chrome but will choke on Safari because of the absence of –WebKit tag. It makes cross-browser testing using TestNG a very crucial job. If I summarize the main big things that make cross-browser testing using TestNG, then it would be as follows:

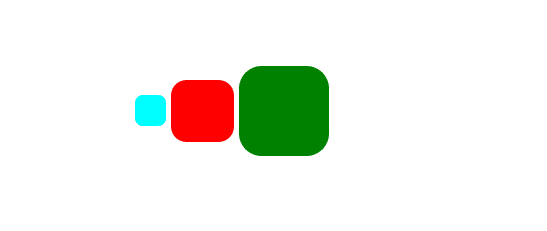
* Cross-browser testing using TestNG ensure a better performance on different browsers and OS.
* Image orientations mess up a lot of the time. We can take care of it.
* The tester and the developer can assure how JavaScript renders on different browsers.
* One can track Font-size issues.
* The unsupported tags can be revealed, which can be taken care of by turnaround codes.

## Practical Example Of Cross-Browser Testing In Browsers

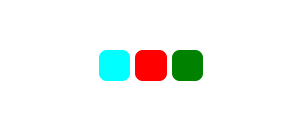
To this point, you must have understood that the idea of cross-browser testing comes from the fact that the engines on which we build the browsers render the web languages differently. We discussed the reasons for the same above. Since this is just theoretical, a real-life practical example would clear the air and help you understand more why cross-browser testing in TestNG is of utmost importance.

In this scenario, we will talk about a CSS property called ***Zoom***. The zoom property is given by the CSS to help developers magnify an element onto the web page. For example, I can magnify my image, a div box, a section, or anything else. It comes handy with just a single line of code helping us achieve our goals. But the problem with zoom property is that it is not supported well with ***Firefox*** and ***Opera Mini,*** which are among the major browsers in the market today. Consider a web page I coded that makes three squares which use zoom property for a different square to magnify it.

The following is the screenshot from Google Chrome:



For the same code, the webpage looks like the following in the Firefox browser:



It happens because Firefox does not support the zoom property. There will be many elements like this on your website that are going to render differently on different browsers, and if they breakdown the website, all the hard work is washed down the river. I hope you have got the importance of cross-browser testing in real life.

### *****How To Perform Cross-Browser Testing In TestNG Using Selenium?*****

Cross-browser testing requires us to test our website using Selenium on multiple browsers, and as you might remember, if we want to pass different values to the same function, we use TestNG parameters for that. Without using the parameters, we are just writing the same repetitive code again and again. TestNG parameters will help us cut down on lines of code significantly and makes efficient use of TestNG annotations which is what every tester aims for while working on TestNG.

In the below code, we will be demonstrating how to perform cross-browser testing in TestNG using selenium web driver. But, before jumping onto the code, I highly recommend reading the [**TestNG parameters**](https://www.toolsqa.com/testng/testng-parameters/) tutorial, [**TestNG Annotations**](https://www.toolsqa.com/testng/testng-annotations/) tutorial, and our [**selenium tutorial**](https://www.toolsqa.com/selenium-tutorial/)so that you adjust towards the flow of the code.

|  |  |
| --- | --- |
|  | import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.openqa.selenium.chrome.\*;  import org.testng.annotations.AfterClass;  import org.testng.annotations.BeforeClass;  import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class MultiBrowser {    public WebDriver driver;    *@Parameters*("browser")    *@BeforeClass*      // Passing Browser parameter from TestNG xml      public void beforeTest(String browser) {      // If the browser is Firefox, then do this      if(browser.equalsIgnoreCase("firefox")) {    //Initializing the firefox driver (Gecko)    driver = new FirefoxDriver();      }else if (browser.equalsIgnoreCase("chrome")) {      //Initialize the chrome driver      driver = new ChromeDriver();      }      // Enter the website address in the browser      driver.get("https://www.demoqa.com");      }      // Once Before method is completed, Test method will start    *@Test* public void login() throws InterruptedException {    driver.findElement(By.xpath("//\*[@id=\"app\"]/div/div/div[2]/div/div[1]/div/div[1]")).click();    }    *@AfterClass* public void afterTest() {    driver.quit();    }    } |

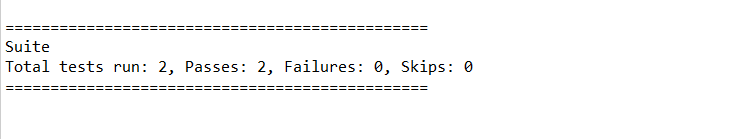
Since we are using [**TestNG parameters**](https://www.toolsqa.com/testng/testng-parameters/), we need to specify the values from the TestNG XML file that will pass to the test case file.

|  |  |
| --- | --- |
|  | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="Suite" parallel="none">  <test name="FirefoxTest">  <parameter name="browser" value="firefox" />  <classes>  <class name="MultiBrowser" />  </classes>  </test>  <test name="ChromeTest">  <parameter name="browser" value= "chrome" />  <classes>  <class name="MultiBrowser" />  </classes>  </test>  </suite> |

The above code snippet does the following job;

* Initialize a parameter with the name browser.
* Initialize the browser driver depending on the parameter value. For a browser value equal to Chrome, initialize a chrome driver, and so on.
* Open the website “demoqa.com” on the browser and click an element with the help of Selenium.
* Specify the browser values in the TestNG XML file.

Once we understand the above code, run the code with the parameter values as Firefox and Chrome***,*** as mentioned above in the XML file.



So, both of my test cases passed successfully running the website on different browsers. You can test your website by using different [***TestNG Asserts***](https://www.toolsqa.com/testng/testng-asserts/)and selenium functions.

Cross-browser testing is essential when it comes to the development of a website. The mobile revolution has changed the way websites are viewed today compared to the last decade. With billions of mobile users in the world, the developer cannot ignore this number. The website can open on any device, any browser, any version, any OS, and any resolution. We, as developers and testers need to be prepared for this before the catastrophe happens. Thankfully, with the help of Selenium, we can perform cross-browser testing in TestNG and be assured of our website. I recommend testing a website entirely as a tester to gain confidence and experience in cross-browser testing in [***TestNG***](https://testng.org/doc/).

# TestNG Data Provider with Excel

## Data-Driven Testing

A key benefit of automating functional testing is the ability to test large volumes of data on the system quickly. But you must be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations with minimal effort. Test Automation Frameworks must-have capability to integrate with spreadsheets and provide powerful calculation features.

## Apache POI (Excel)

Most commercial automated software tools on the market support some sort of data-driven testing, which allows you to automatically run a test case multiple times with different input and validation values. As Selenium Webdriver is more an automated testing framework than a ready-to-use tool, you will have to put in some effort to support data-driven testing in your automated tests. I usually prefer to use Microsoft Excel as the format for storing my parameters. An additional advantage of using Excel is that you can easily outsource the test data administration to someone other than yourself, someone who might have better knowledge of the test cases that need to be run and the parameters required to execute them.

## TestNG Data Providers

When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a database, etc…), in such cases parameters can be passed using Dataproviders. A Data Provider is a method annotated with @DataProvider. A Data Provider returns an array of objects.

Let us check out the same Sign In examples using Data Providers with an Excel datasheet.

## How to do it…

Here we will follow a simple step by step process to Implement Excel with TestNg Data Provider.

**Step 1:** Create a test case of Login Application with TestNG Data Provider.

**Step 2:**  Create a Test Datasheet.

**Step 3:** Create functions to Open & Read data from Excel

**Step 4:** Create a TestNg test case for accepting data from Excel using Data Provider.

**Step 5:** Run the test against the Test Case name in the Test Data file.

## Step 1: Create a test case of LogIn Application with TestNG Data Provider

1) Create a TestNG class ‘DataProviderTest’ by Pressing Ctrl+N, select ‘Create TestNG Class’ under TestNG category and Under Annotations, check ‘DataProvider’ and click Finish.

2) By default, the DataProvider name is ‘dp’, change it to ‘Authentication’. This method returns array of object array.

3) Add a method Registration\_data() to your Test class. This method takes two strings as input parameters.

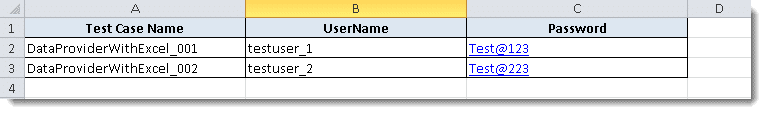
4) Write script for LogIn Application under method @Test.

|  |  |
| --- | --- |
|  | package automationFramework;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.By;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.firefox.FirefoxDriver;    import org.testng.annotations.DataProvider;    import org.testng.annotations.Test;    public class DataProviderTest {        private static WebDriver driver;      @DataProvider(name = "Authentication")      public static Object[][] credentials() {            // The number of times data is repeated, test will be executed the same no. of times            // Here it will execute two times            return new Object[][] { { "testuser\_1", "Test@123" }, { "testuser\_1", "Test@123" }};      }      // Here we are calling the Data Provider object with its Name      @Test(dataProvider = "Authentication")      public void test(String sUsername, String sPassword) {          driver = new FirefoxDriver();          driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);          driver.get("https://www.store.demoqa.com");          driver.findElement(By.xpath(".//\*[@id='account']/a")).click();          // Argument passed will be used here as String Variable          driver.findElement(By.id("log")).sendKeys(sUsername);          driver.findElement(By.id("pwd")).sendKeys(sPassword);          driver.findElement(By.id("login")).click();          driver.findElement(By.xpath(".//\*[@id='account\_logout']/a")).click();          driver.quit();      }    } |

## Step 2:  Create a Test Datasheet

1) Create a ‘[**New Package**](https://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/)‘ file and name it as ‘**testData’**, by right click on the Project and select **New** > **Package**. I always place my Test Data file under a separate test data folder.

2) Place an **Excel** file in the above-created package location and save it as **TestData.xlsx**. Fill the data in the excel like below image:

[](https://toolsqa.com/wp-content/uploads/2014/05/TestNg-Excel.png)

## Step 3: Create functions to Open & Read data from Excel

We need a way to open this Excel sheet and read data from it within our Selenium test script. For this purpose, I use the Apache POI library, which allows you to read, create and edit Microsoft Office-documents using Java. The classes and methods we are going to use to read data from Excel sheet are located in the org.apache.poi.hssf.usermodel package.

|  |  |
| --- | --- |
|  | package utility;            import java.io.FileInputStream;    import java.io.FileNotFoundException;    import java.io.FileOutputStream;    import java.io.IOException;    import org.apache.poi.xssf.usermodel.XSSFCell;    import org.apache.poi.xssf.usermodel.XSSFRow;    import org.apache.poi.xssf.usermodel.XSSFSheet;    import org.apache.poi.xssf.usermodel.XSSFWorkbook;        public class ExcelUtils {    private static XSSFSheet ExcelWSheet;    private static XSSFWorkbook ExcelWBook;    private static XSSFCell Cell;    private static XSSFRow Row;    public static Object[][] getTableArray(String FilePath, String SheetName) throws Exception {       String[][] tabArray = null;       try {       FileInputStream ExcelFile = new FileInputStream(FilePath);       // Access the required test data sheet       ExcelWBook = new XSSFWorkbook(ExcelFile);       ExcelWSheet = ExcelWBook.getSheet(SheetName);       int startRow = 1;       int startCol = 1;       int ci,cj;       int totalRows = ExcelWSheet.getLastRowNum();       // you can write a function as well to get Column count       int totalCols = 2;       tabArray=new String[totalRows][totalCols];       ci=0;       for (int i=startRow;i<=totalRows;i++, ci++) {      cj=0;       for (int j=startCol;j<=totalCols;j++, cj++){       tabArray[ci][cj]=getCellData(i,j);       System.out.println(tabArray[ci][cj]);    }    }    }    catch (FileNotFoundException e){    System.out.println("Could not read the Excel sheet");    e.printStackTrace();    }    catch (IOException e){    System.out.println("Could not read the Excel sheet");    e.printStackTrace();    }    return(tabArray);    }    public static String getCellData(int RowNum, int ColNum) throws Exception {    try{    Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);    int dataType = Cell.getCellType();    if  (dataType == 3) {    return "";    }else{    String CellData = Cell.getStringCellValue();    return CellData;    }catch (Exception e){    System.out.println(e.getMessage());    throw (e);    }    }    } |

## Step 4: Create a TestNg test case for accepting data from Excel using Data Provider

1) Create a TestNG class ‘DataProviderWithExcel’ by Pressing Ctrl+N , select ‘Create TestNG Class‘ under TestNG category and Under Annotations, check ‘@BeforeMethod‘, ‘@AfterMethod‘ & ‘DataProvider‘ and click Finish.

3) Add a method Registration\_data() to your Test class. This method takes two strings as input parameters.

4) Now divide the test case into three parts :

@BeforeMethod : Launch Firefox and direct it to the Base URL

@Test : Enter Username & Password to Login, Print console message and Log out

@AfterMethod : Close Firefox browser

Test Case will look like this:

|  |  |
| --- | --- |
|  | package practiceTestCases;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.By;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.firefox.FirefoxDriver;    import org.testng.annotations.AfterMethod;    import org.testng.annotations.BeforeMethod;    import org.testng.annotations.Test;    import org.testng.annotations.DataProvider;    import utility.ExcelUtils;    public class DataProviderWithExcel\_001 {    WebDriver driver;        @BeforeMethod        public void beforeMethod() throws Exception {        driver = new FirefoxDriver();            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);            driver.get("https://www.store.demoqa.com");    }    @Test(dataProvider="Authentication")        public void Registration\_data(String sUserName,String sPassword)throws  Exception{            driver.findElement(By.xpath(".//\*[@id='account']/a")).click();            driver.findElement(By.id("log")).sendKeys(sUserName);    System.out.println(sUserName);            driver.findElement(By.id("pwd")).sendKeys(sPassword);    System.out.println(sPassword);            driver.findElement(By.id("login")).click();            System.out.println(" Login Successfully, now it is the time to Log Off buddy.");            driver.findElement(By.xpath(".//\*[@id='account\_logout']/a")).click();    }        @DataProvider        public Object[][] Authentication() throws Exception{             Object[][] testObjArray = ExcelUtils.getTableArray("D://ToolsQA//OnlineStore//src//testData//TestData.xlsx","Sheet1");             return (testObjArray);    }        @AfterMethod        public void afterMethod() {           driver.close();         }    } |

**Note:** This LogIn test will execute two times as there are two users credentials in data provider Array.

## Step 5: Run the test against the Test Case name in the Test Data file

1) This means that your test should be run once only with the data which is mentioned against the Test Case name. To do this we need to tweak the Excel utility class, plus need to add few more function to fetch out the current Test Case name & the row number which contain the Test Case name.

|  |  |
| --- | --- |
|  | package utility;    import java.io.FileInputStream;    import java.io.FileNotFoundException;    import java.io.FileOutputStream;    import java.io.IOException;    import org.apache.poi.xssf.usermodel.XSSFCell;    import org.apache.poi.xssf.usermodel.XSSFRow;    import org.apache.poi.xssf.usermodel.XSSFSheet;    import org.apache.poi.xssf.usermodel.XSSFWorkbook;        public class ExcelUtils {    private static XSSFSheet ExcelWSheet;    private static XSSFWorkbook ExcelWBook;    private static XSSFCell Cell;    private static XSSFRow Row;    //This method is to set the File path and to open the Excel file, Pass Excel Path and Sheetname as Arguments to this method    public static void setExcelFile(String Path,String SheetName) throws Exception {       try {    // Open the Excel file    FileInputStream ExcelFile = new FileInputStream(Path);    // Access the required test data sheet    ExcelWBook = new XSSFWorkbook(ExcelFile);    ExcelWSheet = ExcelWBook.getSheet(SheetName);    } catch (Exception e){    throw (e);    }    }    public static Object[][] getTableArray(String FilePath, String SheetName, int iTestCaseRow)    throws Exception    {       String[][] tabArray = null;       try{       FileInputStream ExcelFile = new FileInputStream(FilePath);       // Access the required test data sheet       ExcelWBook = new XSSFWorkbook(ExcelFile);       ExcelWSheet = ExcelWBook.getSheet(SheetName);       int startCol = 1;       int ci=0,cj=0;       int totalRows = 1;       int totalCols = 2;       tabArray=new String[totalRows][totalCols];       for (int j=startCol;j<=totalCols;j++, cj++)       {       tabArray[ci][cj]=getCellData(iTestCaseRow,j);       System.out.println(tabArray[ci][cj]);       }    }    catch (FileNotFoundException e)    {    System.out.println("Could not read the Excel sheet");    e.printStackTrace();    }    catch (IOException e)    {    System.out.println("Could not read the Excel sheet");    e.printStackTrace();    }    return(tabArray);    }    //This method is to read the test data from the Excel cell, in this we are passing parameters as Row num and Col num    public static String getCellData(int RowNum, int ColNum) throws Exception{       try{      Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);      String CellData = Cell.getStringCellValue();      return CellData;      }catch (Exception e){    return"";    }    }    public static String getTestCaseName(String sTestCase)throws Exception{    String value = sTestCase;    try{    int posi = value.indexOf("@");    value = value.substring(0, posi);    posi = value.lastIndexOf(".");    value = value.substring(posi + 1);    return value;    }catch (Exception e){    throw (e);    }    }    public static int getRowContains(String sTestCaseName, int colNum) throws Exception{    int i;    try {    int rowCount = ExcelUtils.getRowUsed();    for ( i=0 ; i<rowCount; i++){    if  (ExcelUtils.getCellData(i,colNum).equalsIgnoreCase(sTestCaseName)){    break;    }    }    return i;    }catch (Exception e){    throw(e);    }    }    public static int getRowUsed() throws Exception {    try{    int RowCount = ExcelWSheet.getLastRowNum();    return RowCount;    }catch (Exception e){    System.out.println(e.getMessage());    throw (e);    }    }    } |

**Final Test Case**

1) Get the Test Case name.

2) With the Test Case name, get the row number of the Excel sheet for the test.

3) Get the data from the excel sheet for the fetched test row.