# TestNG Annotations | Order, Example

## What is Annotation?

[***Annotation***](https://www.scientecheasy.com/2021/09/annotations-in-java.html/) is a feature in Java that is used to add metadata to Java source. It was introduced in Java 1.5 version. It can be applied to classes, methods, variables, and parameters.

Examples of certain predefined annotations in Java are @Override, @Deprecated, @SupressWranings, and so on. But Java also allows for user-defined annotations.

Therefore, TestNG uses this feature provided by Java to define its own annotations and develop an execution framework by using it.

Before going to learn TestNG annotations, let’s see first some important terminology related to it.

* **Suite:** A suite is made of one or more tests.
* **Test:** A test is made of one or more classes.
* **Class:** A class is made of one or more methods.

These terminologies are very important because they are connected to the annotations. So keep them in mind.

## What is TestNG Annotation?

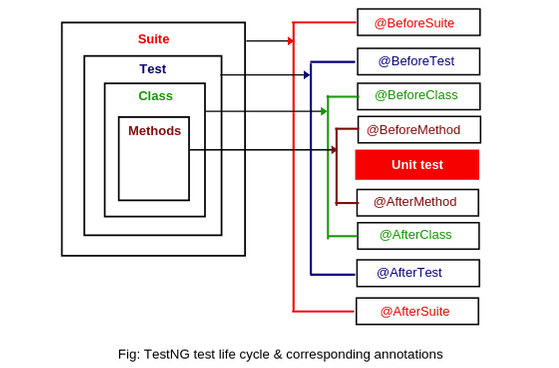
**Annotation in TestNG** is the set of code that controls how method below them has to be executed. i.e. the order of the execution of method below them will be decided by an annotation that you give.

TestNG supports a powerful set of different annotations that run your tests in an organized and predictable manner.

An annotation is always preceded by a @ symbol which allows performing some java logic before and after a certain point.

TestNG life cycle defines the way and order in which the annotated method is executed in order. It will also help you to understand how to write a better, faster, and more maintainable unit test.

The TestNG life cycle is shown in the below figure.

[](https://www.scientecheasy.com/2020/07/testng-annotations.html/)

In TestNG, a test suite is made up of one or more tests that contain a number of test classes. A test class is made up of one or more test methods.

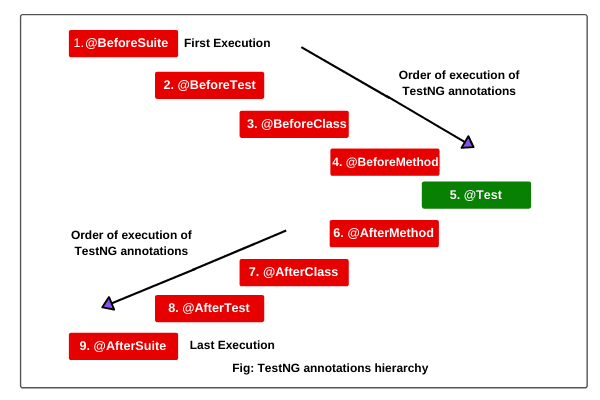
## Hierarchy of TestNG Annotations Order

The order of execution of an annotated method with TestNG annotations is shown in the below figure. The method annotated with @BeforeSuite will be executed first whereas method annotated with @AfterSuite will be executed at last.

**Hierarchy of the TestNG Annotations:**

* @BeforeSuite
* @BeforeTest
* @BeforeClass
* @BeforeMethod
* @Test
* @AfterMethod
* @AfterClass
* @AfterTest
* @AfterSuite

Look at the hierarchy diagram of TestNG annotations in the below figure to understand better.

[](https://www.scientecheasy.com/2020/07/testng-annotations.html/)

## Before and After TestNG Annotations with Example

Basically, Before and After annotations are mainly used to execute a set of code before and after the execution of test methods.

These annotations can be used to set up a configuration before starting of test execution and then to close after the test execution ends.

TestNG provides the following different kinds of Before and After annotations as shown in the above figure. Both TestNG annotations can be used based on test requirements.

Let’s understand a brief explanation that will give you a basic idea of various functionalities offered by TestNG.

**1. @BeforeSuite:** This annotated method will be run before running all the tests in the entire suite. A method annotated with @BeforeSuite will be executed only once. Therefore, @BeforeSuite can be used to set up and start Selenium WebDriver.

**2. @AfterSuite:** This annotated method will be run after running all the tests in this suite. A method annotated with @AfterSuite will be executed only once. Therefore, it can be used to stop the selenium driver.

Let’s take a simple example program based on @BeforeSuite and @AfterSuite.

**Program source code 1:**

package testngtests;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.AfterSuite;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.Test;

public class DriverSetUp

{

// First of all, create WebDriver reference.

WebDriver driver;

@BeforeSuite

public void setUp()

{

// Create an object of FirefoxDriver class.

driver = new FirefoxDriver();

}

@Test

public void launch()

{

driver.get("https://www.google.com"); // It will open Google home page in the browser.

}

@AfterSuite

public void closeDriver()

{

driver.close();

}

}

**3. @BeforeTest:** Method annotated with @BeforeTest will be executed before running each unit test. It will be executed only once, before executing the test method within the same test class.  
  
@BeforeTest can be used to set up some data before doing each test.

**4. @AfterTest:** Method annotated with @AfterTest will be executed after running each unit test. It will be executed only once, after finishing the execution of all test methods within the same test class.

@AfterTest can be used to generate a screenshot of test results after the execution of all test methods and mail it to the stakeholders.

Let’s automate one scenario based on @BeforeTest and @AfterTest.

**Scenario to automate:**

1. Launch the Firefox browser and maximize the browser.
2. Open the web page [google.com](https://www.google.com/).
3. Wait for some time to load webpage successfully.
4. Get the title of web page.
5. Close the browser.

Let’s create a program to automate the following scenario. Program source code is as follows:

**Program source code 2:**

package testngtests;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.AfterSuite;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class BeforeTestEx

{

// First create WebDriver reference.

WebDriver driver;

@BeforeSuite

public void setUpDriver()

{

// Create an object of FirefoxDriver class.

driver = new FirefoxDriver();

// Maximize browser.

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

}

@BeforeTest

public void launchURL()

{

String URL = "https://www.google.com";

driver.get(URL);

// Wait for some time to load web page successfully.

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

}

@Test

public void getTittle()

{

String getTitle = driver.getTitle();

System.out.println("Title of web page: " +getTitle);

}

@AfterTest

public void testResult()

{

// Put the code to take screenshot of test result.

}

@AfterSuite

public void closeBrowser()

{

driver.close();

}

}

Output:

Title of webpage: Google

**5. @BeforeGroups:** The annotation notifies the system that method annotated with @BeforeGroups will run before any of the test methods of the specified group is executed.

**6. @AfterGroups:** Method annotated with @AfterGroups will run after any of the test methods of the specified group is executed.

**7. @BeforeClass:** This annotated method will be executed only once before any of the tests in the current class are run. It can be used to set up the configuration and initialization which is common to all test methods in the current class.

For example, you can set up the driver configuration which will be common for all tests in the class.

**8. @AfterClass:** This annotated method will be executed only once after finishing all the tests of that current class. It can be used to stop the driver after finishing all test execution.

**9. @BeforeMethod:** Method annotated with @BeforeMethod will be executed before each method annotated with @Test. It can be used to set that data that is repeating before each @Test annotated method.

**10. @AfterMethod:** Method annotated with @BeforeMethod will be executed after each and every method annotated with @Test.

Let’s take a simple example program based on the above TestNG annotations. Look at the following source code.

**Program source code 3:**

package testngtests;

import org.testng.annotations.AfterClass;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

public class BeforeTestEx

{

@BeforeClass

public void beforeClass()

{

System.out.println("Before class");

}

@BeforeMethod

public void beforeMethod()

{

System.out.println("Before method");

}

@Test

public void test1()

{

System.out.println("Test1 is executed");

}

@Test

public void test2()

{

System.out.println("Test2 is executed");

}

@AfterMethod

public void afterMethod()

{

System.out.println("After method");

}

@AfterClass

public void afterClass()

{

System.out.println("After class");

}

}

Output:

Before class

Before method

Test1 is executed

After method

Before method

Test2 is executed

After method

After class

PASSED: test1

PASSED: test2

## Other Kinds of TestNG Annotations

TestNG provides some other important kinds of annotations that are frequently used.

**1. @Test:** Methods annotated with @Test are called test methods that serve as a unit test. In @Test methods, we will write the logic of the application which we want to automate.

We can pass attributes to our test method. The output depends on these test methods for success or failure.

**2. @DataProvider:** It marks a method as a data providing method for a test method. It returns an object double array(Object [ ][ ]) as data to the test method.

**3. @Parameters:** This @Parameters annotation is used to pass parameters to a test method. The values of parameters are provided using the testng.xml file at runtime.

This parameter annotation can be used with any of the Before/After, Factory, Test annotated methods.

**4. @Factory:** This annotation marks an annotated method as a factory that returns an array of class objects (Object [ ]). These class objects will then be used as test classes by TestNG. It is used to execute a set of test cases with different values.

**5. @Listeners:** This annotation is defined as an array of test listeners classes that extends org.testng.ITestNGListener. It is used to track the execution status and to customize TestNG reports or logs.

We will learn in more detail in the further tutorial about these TestNG annotations one by one with example scenarios.