TestNG Annotations And Benefits – AUTOMATION TESTING

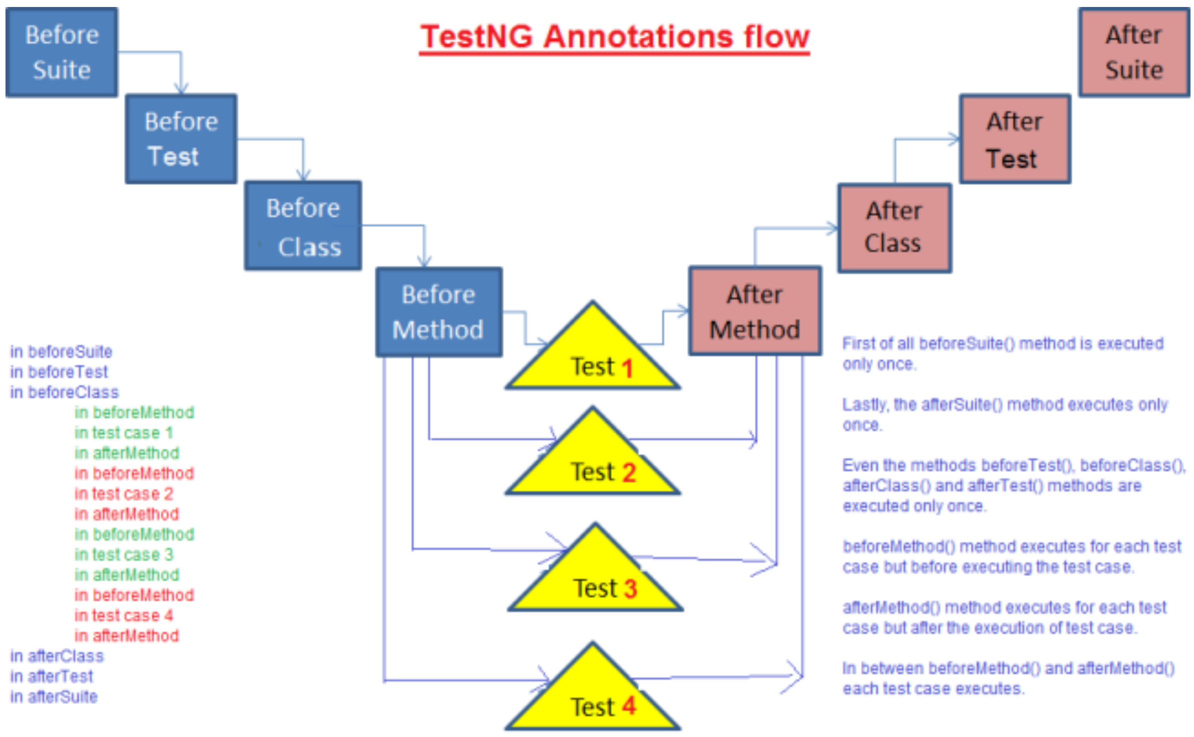
TestNG Annotations:

Here is a quick overview of the annotations available in TestNG along with their attributes.

	Configuration information for a TestNG class:
@BeforeSuite @AfterSuite @BeforeTest @AfterTest @BeforeGroups @AfterGroups @BeforeClass @AfterClass @BeforeMethod @AfterMethod	**@BeforeSuite:** The annotated method will be run before all tests in this suite have run. **@AfterSuite:** The annotated method will be run after all tests in this suite have run. **@BeforeTest**: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run. **@AfterTest**: The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run. **@BeforeGroups**: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked. **@AfterGroups**: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked. **@BeforeClass**: The annotated method will be run before the first test method in the current class is invoked. **@AfterClass**: The annotated method will be run after all the test methods in the current class have been run. **@BeforeMethod**: The annotated method will be run before each test method. **@AfterMethod**: The annotated method will be run after each test method.**Behaviour of annotations in superclass of a TestNG class** The annotations above will also be honored (inherited) when placed on a superclass of a TestNG class. This is useful for example to centralize test setup for multiple test classes in a common superclass. In that case, TestNG guarantees that the "@Before" methods are executed in inheritance order (highest superclass first, then going down the inheritance chain), and the "@After" methods in reverse order (going up the inheritance chain).
alwaysRun	For before methods (beforeSuite, beforeTest, beforeTestClass and beforeTestMethod, but not beforeGroups): If set to true, this configuration method will be run regardless of what groups it belongs to. For after methods (afterSuite, afterClass, ...): If set to true, this configuration method will be run even if one or more methods invoked previously failed or was skipped.
dependsOnGroups	The list of groups this method depends on.
dependsOnMethods	The list of methods this method depends on.
enabled	Whether methods on this class/method are enabled.
groups	The list of groups this class/method belongs to.
inheritGroups	If true, this method will belong to groups specified in the @Test annotation at the class level.

@DataProvider	**Marks a method as supplying data for a test method. The annotated method must return an Object[][] where each Object[] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.**
name	The name of this data provider. If it's not supplied, the name of this data provider will automatically be set to the name of the method.
parallel	If set to true, tests generated using this data provider are run in parallel. Default value is false.
@Factory	**Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[].**
@Listeners	**Defines listeners on a test class.**
value	An array of classes that extend org.testng.ITestNGListener.
@Parameters	**Describes how to pass parameters to a @Test method.**
value	The list of variables used to fill the parameters of this method.
@Test	**Marks a class or a method as part of the test.**
alwaysRun	If set to true, this test method will always be run even if it depends on a method that failed.
dataProvider	The name of the data provider for this test method.
dataProviderClass	The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class.
dependsOnGroups	The list of groups this method depends on.
dependsOnMethods	The list of methods this method depends on.
description	The description for this method.
enabled	Whether methods on this class/method are enabled.
expectedExceptions	The list of exceptions that a test method is expected to throw. If no exception or a different than one on this list is thrown, this test will be marked a failure.
groups	The list of groups this class/method belongs to.
invocationCount	The number of times this method should be invoked.
invocationTimeOut	The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified.
priority	The priority for this test method. Lower priorities will be scheduled first.
successPercentage	The percentage of success expected from this method
singleThreaded	If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with parallel="methods". This attribute can only be used at the class level and it will be ignored if used at the method level. Note: this attribute used to be called sequential (now deprecated).
timeOut	The maximum number of milliseconds this test should take.
threadPoolSize	The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. Note: this attribute is ignored if invocationCount is not specified

Basic annotations and its execution process-



```
in beforeSuite
in beforeTest
in beforeClass
        in beforeMethod
        in test case 1
        in afterMethod
        in beforeMethod
        in test case 2
        in afterMethod
        in beforeMethod
        in test case 3
        in afterMethod
        in beforeMethod
        in test case 4
        in afterMethod
in afterClass
in afterTest
in afterSuite
```

First of all beforeSuite() method is executed only once.

Lastly, the afterSuite() method executes only once.

Even the methods beforeTest(), beforeClass(), afterClass() and afterTest() methods are executed only once.

beforeMethod() method executes for each test case but before executing the test case.

afterMethod() method executes for each test case but after the execution of test case.

In between beforeMethod() and afterMethod() each test case executes.

Lets see the order of methods called using the below script:

```
import org.testng.annotations.AfterClass;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterSuite;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class TestngAnnotation {

  // Test Case 1

  @Test

  public void testCase1() {

    System.out.println(“in Test Case 1”);

  }

  // Test Case 2

  @Test

  public void testCase2() {

    System.out.println(“in Test Case 2”);
  }
  @BeforeMethod
  public void beforeMethod() {
    System.out.println(“in Before Method”);
  }
  @AfterMethod
  public void afterMethod() {
    System.out.println(“in After Method”);
  }
  @BeforeClass
  public void beforeClass() {
    System.out.println(“in Before Class”);
  }
  @AfterClass
  public void afterClass() {
    System.out.println(“in After Class”);
  }
  @BeforeTest
  public void beforeTest() {
    System.out.println(“in Before Test”);
  }
  @AfterTest
  public void afterTest() {
    System.out.println(“in After Test”);
  }
  @BeforeSuite
  public void beforeSuite() {
    System.out.println(“in Before Suite”);
  }
  @AfterSuite
  public void afterSuite() {
    System.out.println(“in After Suite”);
  }
```

}

Output:-

```
[TestNG] Running:

in Before Suite
in Before Test
in Before Class
in Before Method
in Test Case 1
in After Method
in Before Method
in Test Case 2
in After Method
in After Class
in After Test
in After Suite

===============================================
Default suite
Total tests run: 2, Failures: 0, Skips: 0

===============================================
```

Summary of TestNG Annotations:-

TestNG is more flexible because of its annotations. What do they offer to us?

So there are controlled annotation TestNG:

- Annotations**@BeforeSuite**, **@AfterSuite** Indicate the methods that are executed once before / after execution of all tests. It is convenient to have any difficult settings common to all tests, for example, you can create a pool of database connections.
- Annotations**@BeforeGroups**, **@AfterGroups** refer to methods that run before / after the first / last test belonging to a given group.
- Annotations**@BeforeClass**, **@AfterClass** define the methods that are executed once before / after execution of all tests in the class. The most suitable for testing of a specific service, which does not change its status as a result of the test.
- Annotations**@BeforeTest**, **@AfterTest** define methods that are executed once before / after the execution of the test (the one that includes the test classes, not to be confused with the test methods). Here you can store the settings of a group of interrelated services or a service if it is tested by several test classes.
- Methods with**@BeforeMethod**, **@AfterMethod** annotations will be executed before/after each test method.

All of these annotations have the following options:

- **enabled**– can be temporarily disabled by setting the value to false
- **groups**– define, for which groups will be executed
- **inheritGroups**– if true (and the default value is true), the method will inherit the group of the test class
- **timeOut**– the time after which "fall down" method and drags along all dependent tests from its description – the name used in the report
- **dependsOnMethods**– methods from which they are depended, will first be executed, and then this method
- **dependsOnGroups**– the groups from which they are depended
- **alwaysRun**– if set to true, will always be called regardless of which groups it belongs to, not applicable to **@BeforeGroups**, **@AfterGroups**.