**TESTNG**

**Creator** :

Cedirc Beust ( French software engineer) 53 years old (1969)

Work as senior software engineer Google until 2010 and switched to work LinkedIn

The idea of ​​​​creating TestNG was around 2004 (version 1.0 was released on April 28, 2004) when he created and implemented small testing projects with Junit..

NG means [Next Generation]

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:

* Annotations.
* Run your tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc...).
* Test that your code is multithread safe.
* Flexible test configuration.
* Support for data-driven testing (with @DataProvider).
* Support for parameters.
* Powerful execution model (no more TestSuite).
* Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).
* Embeds BeanShell for further flexibility.

Default JDK functions for runtime and logging (no dependencies).

Dependent methods for application server testing.

**Adventages:**

* There are many types of After and Before annotations
* Dependent Run test feature (eg: run B condition is mandatory to run A first)
* Ability to group test methods
* Run Multithread
* Report in HTML
* Can be tested in parallel, Generate bug logs, generate data Parameterized action.

**Annotations**

|  |  |
| --- | --- |
| @BeforeSuite: | The annotated method will be run only once before all tests in this suite have run. |
| @AfterSuite: | The annotated method will be run only once 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 only once before the first test method in the current class is invoked. |
| @AfterClass | The annotated method will be run only once after all the test methods in the current class have run. |
| @BeforeMethod | The annotated method will be run before each test method. |
| @AfterMethod | The annotated method will be run after each test method. |
| @Test | Marks a class or a method as a part of the test. |
| @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. |
| @Paramenters | Describes how to pass parameters to a @Test method. |

**Asserts**

* 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.

**Group test:**

* Groups in TestNG denotes the process of grouping different tests together into a straightforward and running these test together by just running the group in a single command
* It does not even matter if they belong to different classes
* Inside groups: TesNG also allows us to group test inside groups

+ Include/exclude group: besides, TestNG allows us to include groups as well as wxclude them

**Parameterized Test**

* Test methods don’t have to be parameterless. You can us an arbitrayry number of parameter on each of your test method, and you instruct TestNG to pass you the correct paramters with the @Parameters annotation
* TestNG let you pass parameters direcly to your test methods in two differents ways:

+ With testng.xml: you define the simple parameters in the testng.xml file and then reference those parameters in the source files

+With Data Providers:

. Passing parameters using testng.xml is not sufficient when you need to pass complex parameters or parameters that need to be created from java(complex object, objects read from a property file or a database, ect……).

.In this case, you can use Data Provider to supply the values you need to test. A Data Provider is a method on your class that returns an array of array objects. This method annotated with @DataProvider.

**Dependency Test:**

* Sometimes, you need to invoke methods in a test casr in a particular order, or you may want to share some data and state between methods.
* TestNG supports this kind of dependency, as it supports the declaration of explicit dependencies between test methods.
* You can use the attributes **dependsOnMethods** or **depensOnGroups**, found on the **@Test** annotation
* There are two kinds of dependencies: **Hard dependencies** and **Soft dependencies**

**+**Hard Dependencies:

**.** All the methods you depend on must have run and succeeded for you to run

. If at least one failure occurred in your dependencies, you will not be invoked and marked as a SKIP in the report.

+Soft Denpendencies:

.You will always be run after the methods you depend on, even if some of them have failed

.A soft dependency is obtained by adding “alwaysRun=true” in your @Test annotation.