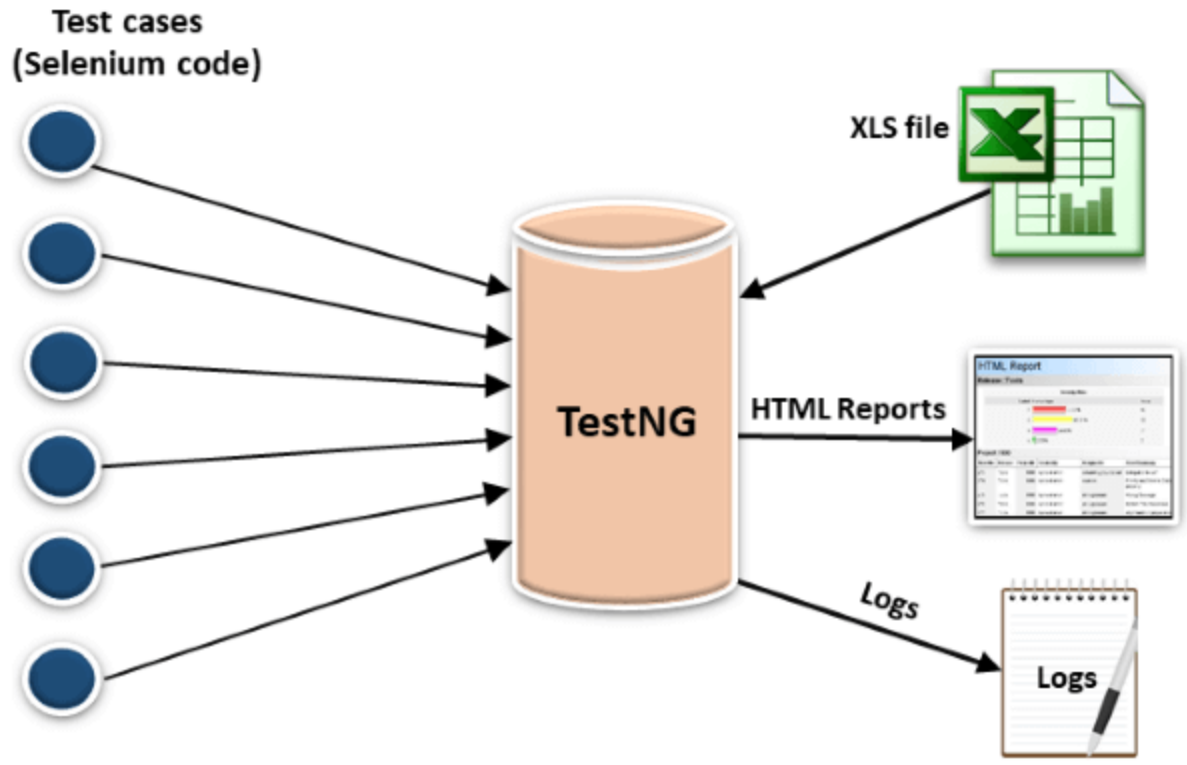
What is TestNG

- TestNG is a very important framework when you are actually developing the framework from scratch level.
- TestNG provides you full control over the test cases and the execution of the test cases. Due to this reason, TestNG is also known as a testing framework.
- Cedric Beust is the developer of a TestNG framework.
- If you want to run a test case A before that as a pre-request you need to run multiple test cases before you begin a test case A. You can set and map with the help of TestNG so that pre-request test cases run first and then only it will trigger a test case A. In such way, you can control the test cases.
- TestNG framework came after Junit, and TestNG framework adds more powerful functionality and easier to use.
- It is an open source automated TestNG framework. In TestNG, NG stands for "**Next Generation**".
- TestNG framework eliminates the limitations of the older framework by providing more powerful and flexible test cases with help of easy annotations, grouping, sequencing and Parameterising.

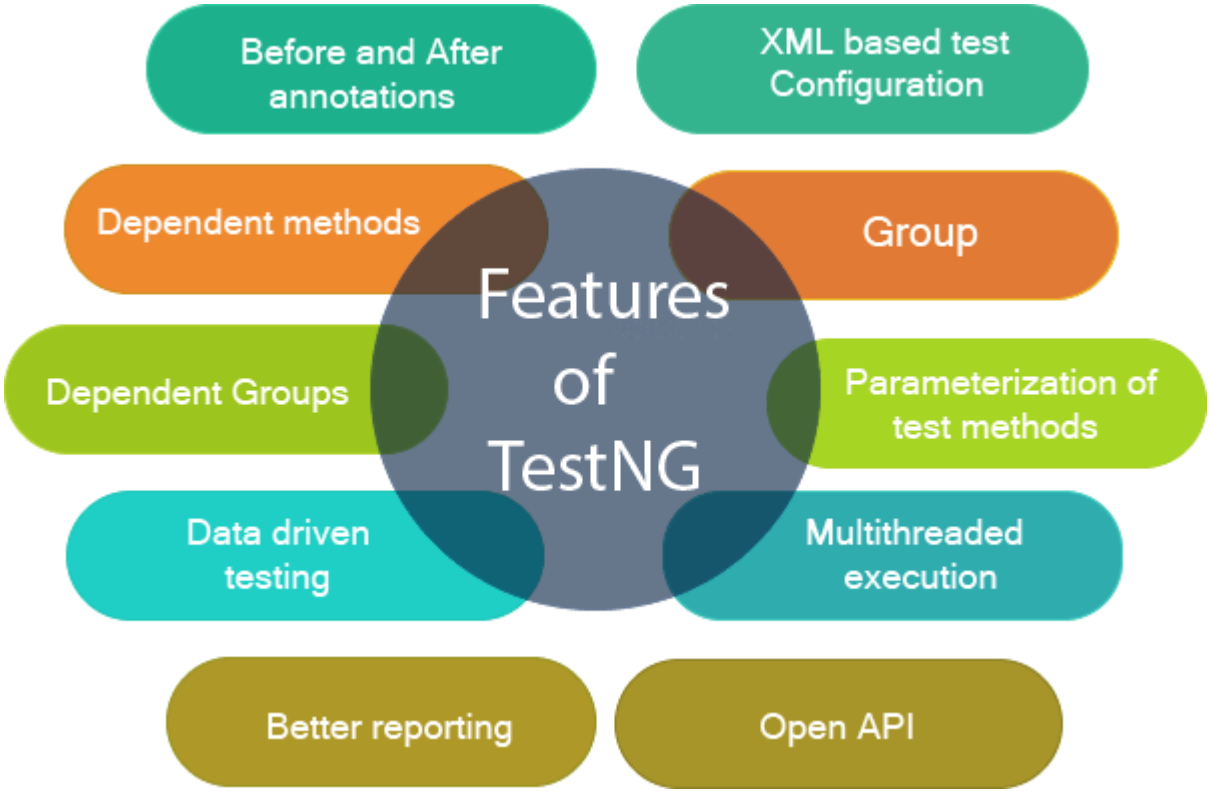
Advantages of TestNG over Junit



- In TestNG, annotations are easier to understand than Junit.
- It produces the HTML reports for implementation.
- It also generates the Logs.
- In TestNG, there is no constraint available such as @beforeclass and @afterclass which is present in Junit.
- TestNG enables you to group the test cases easily which is not possible in JUnit.
- TestNG supports three additional levels such as @Before/After suite, @Before/AfterTest, and Before/AfterGroup.
- TestNG does not extend any class. TestNG framework allows you to define the test cases where each test case is independent of other test cases.
- It allows you to run the test cases of a particular group. Let's consider a scenario where we have created two groups such as 'Smoke' and 'Regression'. If you want to execute the test cases in a 'Regression' group, then this can only be possible in the TestNG framework.
- Parallel execution of test cases, i.e., running multiple test cases is only possible in the TestNG framework.

TestNG Installation and Configuration in Eclipse

Features of TestNG



Multiple Before and After annotation options

Before and after annotations are used to execute a certain set of code before and after executing the test methods. These annotations are used to set the variables or configuration before the start of the execution of test methods and clean up all the variables after the execution ends. Some of the Before and After annotations are @BeforeSuite, @BeforeTest, @BeforeGroups, @BeforeClass, etc.

XML-based test configuration

Test suites in a Testng are mainly configured by using the XML-based file. Testng.xml file is used to organize and run the test suites. The testng.xml file is used to create the test suites by using classes, test methods, packages as well as by using the test groups. It is also used to pass the parameters to test classes or methods.

Dependent methods

Dependency is a feature of Testng that allows a test method to depend on the single or group of test methods. Dependency works on the principle "depend-on-method" which must be either in the same class or in the inherited base class. This is the most important feature in TestNG that tells the TestNG to run the dependent test method after the execution of a given

test method. You can also configure whether you want dependent test method should be executed or not even after the execution of the given test method fails.

Groups/group of groups

TestNG groups allow you to group the test methods. By using TestNG groups, you can declare the methods in a group as well as you can declare the groups within a group. The Testng group can be used to include a certain set of groups and can exclude another set of groups.

Dependent groups

Similar to the Dependent methods, test methods in a group can depend on the test methods of another group.

Parameterization of test methods

One of the most important feature of TestNG is Parameterization. This feature allows you to pass the arguments as parameters and this achieved by using testng@Parameters annotation. We can pass the parameters to test methods in two ways, i.e., testng.xml file and DataProviders.

Data-driven testing

TestNG allows users to perform data-driven testing. This testing allows users to execute the same test multiple times with multiple sets of data. To achieve the data-driven testing, DataProvider feature is used. DataProvider is a data feeder method that executes the test method with multiple sets of data.

Multithreaded execution

Multithreaded execution is the parallel execution of tests. Multithreading means the execution of multiple parts of software at the same time. Based on the configuration in the XML file, multiple threads are started, and test methods are executed in them. Multithreaded execution saves a lot of execution time.

Better reporting

Testng provides XML and HTML reports by default for test execution. You can even add your own custom reports when required.

Open API

TestNG contains the open API means API is publicly available to the developers. This feature allows you to create your custom extensions in your framework when required.

TestNG Groups

TestNG Groups allow you to perform groupings of different test methods. Grouping of test methods is required when you want to access the test methods of different classes.

Not only you can declare the methods within a specified group, you can also declare another group within a specified group. Thus, TestNG can be asked to include a certain set of groups while excluding another set of groups.

It provides you maximum flexibility by partitioning your test methods in groups and does not require recompilation of test cases if you run your two different sets of test cases back to back.

Groups are specified in the testng.xml file with <groups> tag. Groups can be specified either in the <suite> tag or <test> tag. If the <groups> tag is specified inside the <suite> tag, then it is applied to all the <test> tags of XML file. If the <groups> tag is specified within a particular <test> folder, then it is applied to that particular <test> tag only.

Test cases within Groups

Step 1

```
package TestCasesWithinGroups;

import org.testng.annotations.Test;

public class OrangeHRMLogin {

	@Test(groups = { "SmokeTest" })
	public void OrangeLogin() {
		System.out.println("OrangeHRM User Can able to Login");
	}

	@Test
	public void OrangeAdminLoign() {
		System.out.println("OrangeHRM Admin Can able to Login");
	}

	@Test
	public void OrangeEmployeeLogin() {
		System.out.println("OrangeHRM Employee Can able to Login");
	}

}
```

Step 2

```
package TestCasesWithinGroups;

import org.testng.annotations.Test;

public class EasyCalLogin {
	@Test
	public void EasyCalculaLogin() {
```

```
		System.out.println("EasyCalcula User Can able to Login");
	}

	@Test(groups = { "SmokeTest" })
	public void EasyCalculaAdminLoign() {
		System.out.println("EasyCalcula Admin Can able to Login");
	}

	@Test
	public void EasyCalculaEmployeeLogin() {
		System.out.println("EasyCalcula Employee Can able to Login");
	}

}
```

Step 3

```
package TestCasesWithinGroups;

import org.testng.annotations.Test;

public class DemoOpenCartLogin {

	@Test
	public void DemoOpenLogin() {
		System.out.println("DemoOpenCart User Can able to Login");
	}

	@Test
	public void DemoOpenAdminLoign() {
		System.out.println("DemoOpenCart Admin Can able to Login");
	}

	@Test(groups = { "SmokeTest" })
	public void DemoOpenEmployeeLogin() {
		System.out.println("DemoOpenCart Employee Can able to Login");
	}

}
```

Step 4

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">
	<groups>
		<run>
			<include name="SmokeTest" />
		</run>
	</groups>

	<test name="OrangeHRM Login">
		<classes>
			<class name="TestCasesWithinGroups.OrangeHRMLogin" />
		</classes>
	</test>

	<test name="EasyCal Login">
```

```
		<classes>
			<class name="TestCasesWithinGroups.EasyCalLogin" />
		</classes>
	</test>

	<test name="DemoOpenCart Login">
		<classes>
			<class name="TestCasesWithinGroups.DemoOpenCartLogin" />
		</classes>
	</test>

</suite>
```

Second case: When <groups> tag is defined inside the tag.

```
package BelongingToMultipleGroups;

import org.testng.annotations.Test;

public class MultipleGroups {
	@Test(groups = { "Group A" })
	public void testcase1() {
		System.out.println("Test case1 belonging to Group A");
	}

	@Test(groups = { "Group A", "Group B" })
	public void testcase2() {
		System.out.println("Test case2 belonging to both Group A and Group B");
	}

	@Test(groups = { "Group B" })
	public void testcase3() {
		System.out.println("Test case3 belonging to Group B");
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

	<test name="Group A">
		<groups>
			<run>
```

```
                                        <include name="Group A" />
                                </run>
                        </groups>
                        <classes>
                                <class name="BelongingToMultipleGroups.MultipleGroups" />
                        </classes>
        </test>

        <test name="Group B">
                        <groups>
                                <run>
                                        <include name="Group B" />
                                </run>
                        </groups>
                        <classes>
                                <class name="BelongingToMultipleGroups.MultipleGroups" />
                        </classes>
        </test>

</suite>
```

Including/Excluding Groups

```
package IncludingExcludingGroups;

import org.testng.annotations.Test;

public class IncludingExcludingDemo {
        @Test(groups = { "Include Group" })
        public void test_case1() {
                System.out.println("This is test case 1");
        }

        @Test(groups = { "Exclude Group" })

        public void test_case3() {
                System.out.println("This is test case 3");
        }

        @Test(groups = { "Include Group" })
        public void test_case2() {
                System.out.println("This is test case 2");
        }

        @Test(groups = { "Exclude Group" })

        public void test_case4() {
                System.out.println("This is test case 4");
        }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="test_suite">
	<test name="Include and Exclude Group">
		<groups>
			<run>
				<include name="Include Group" />
				<exclude name="Exclude Group" />
			</run>
		</groups>
			<classes>
				<class name="IncludingExcludingGroups.IncludingExcludingDemo" />
			</classes>
	</test>
</suite>
```

Groups in TestNG

```java
package seleniumWithTestNG;

import org.testng.annotations.Test;

public class GroupTest {
	@Test(groups = { "Apple" })
	public void apple1() {
		System.out.println("Test Apple device 1");
	}

	@Test(groups = { "Apple" })
	public void apple2() {
		System.out.println("Test Apple device 2");
	}

	@Test(groups = { "MI" })
	public void mi1() {
		System.out.println("Test MI device 1");
	}

	@Test(groups = { "MI" })
	public void mi2() {
		System.out.println("Test MI device 2");
	}

	@Test(groups = { "Moto" })
	public void motog1() {
		System.out.println("Test Moto device 1");
	}

	@Test(groups = { "Moto" })
	public void motog2() {
		System.out.println("Test Moto device 2");
	}

	@Test(groups = { "Lenova" })
	public void lenova1() {
		System.out.println("Test Lenova device 1");
	}

	@Test(groups = { "Lenova" })
	public void lenova2() {
		System.out.println("Test Lenova device 2");
```

```
	}
}
```

TestNGWithPriority

```
package seleniumWithTestNG;

import org.testng.annotations.Test;

public class TestWithPriority {

	@Test(priority = 1)
	public void startEngine() {
		System.out.println("Engine started");
	}

	@Test(priority = 2)
	public void putFirstGear() {
		System.out.println("Car is in first Gear");
	}

	@Test(priority = 3)
	public void putSecondGear() {
		System.out.println("Car is in second Gear");
	}

	@Test(priority = 4)
	public void putThirdGear() {
		System.out.println("Car is in third Gear");
	}
}
```