

Automation Testing for Web



Lecture 01:

TESTNG TESTING FRAMEWORK

Agenda

- Why TestNG and Its Advantages.
- Running testcases in TestNG with out Void main Java.
- TestNG Basic Annotations
- Importance of TestNG xml file.
- Prioritizing the tests using TestNG.
- Controlling the Testcase execution with Exclude Mechanism.
- Executing the Testcases at Package level with regex.
- Test level TestNG Annotations examples.
- Importance of Groups in TestNG .
- Parameterising from TestNG xml file.
- DataProvider Annotation -Parameterizing Testcases.
- Importance of Listeners in TestNG framework.
- TestNG Assertions.

Why TestNG and Its Advantages

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

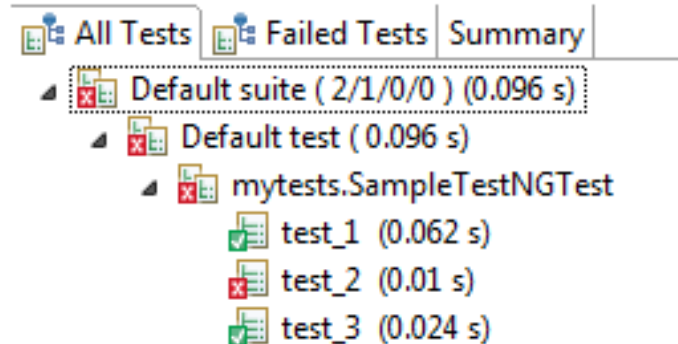
TestNG

Why TestNG and Its Advantages

- Generate the report in a proper format including a number of test cases runs, the number of test cases passed, the number of test cases failed, and the number of test cases skipped.
- Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first.
- Using testng, you can execute multiple test cases on multiple browsers, i.e., cross browser testing.
- The testing framework can be easily integrated with tools like Maven, Jenkins, etc.

Why use TestNG with Selenium?

- Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest
- WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format like the one shown below.



Why use TestNG with Selenium

TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.

```
public static void main(String[] args) {  
    driver.get(baseUrl);  
    verifyHomepageTitle();  
    driver.quit();  
}  
  
public static void verifyHomepageTitle() {  
    String expectedTitle = "Welcome: Mercury Tours";  
    String actualTitle = driver.getTitle();  
    try {  
        Assert.assertEquals(actualTitle, expectedTitle);  
        System.out.println("Test Passed");  
    } catch (Throwable e) {  
        System.out.println("Test Failed");  
    }  
}
```

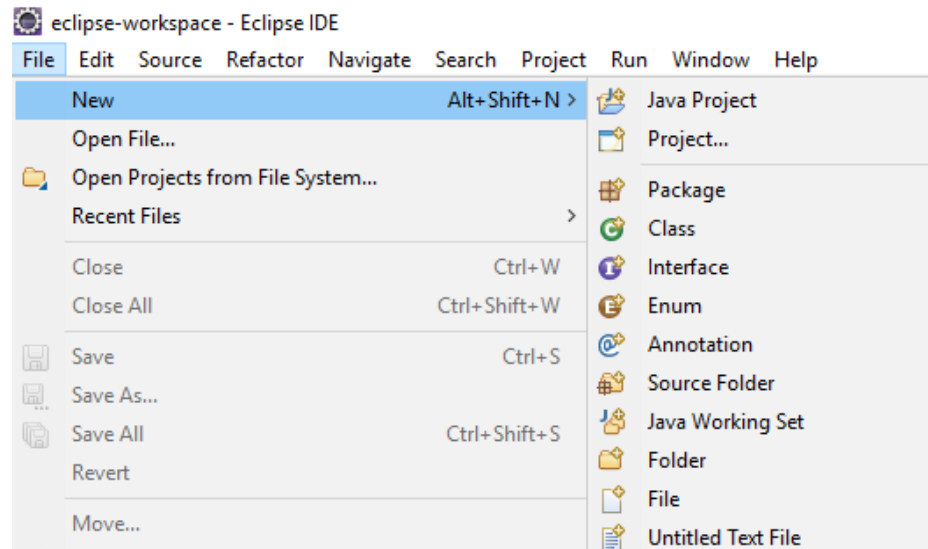
```
@BeforeTest  
public void setBaseUrl() {  
    driver = new FirefoxDriver();  
    driver.get(baseUrl);  
}  
  
@Test  
public void verifyHomepageTitle() {  
    String expectedTitle = "Welcome: Mercury Tours";  
    String actualTitle = driver.getTitle();  
    Assert.assertEquals(actualTitle, expectedTitle);  
}  
  
@AfterTest  
public void endSession() {  
    driver.quit();  
}
```

Why Use TestNG with Selenium?

- Using it you can create data-driven frameworks. You can use the `@DataProvider` annotation to support data-driven testing.
- If you want to run your tests based on the parameters, TestNG supports parameter feature which will make your test cases or test suite very flexible. You can easily pass the parameter at runtime so that based on the requirement you can pass the parameter and you don't have to modify your existing test.
- You can simply group the test cases with the use of TestNG. Suppose you have four categories of smoke, recreation, functional and non-functional. You can group all of these test cases and can run them individually.

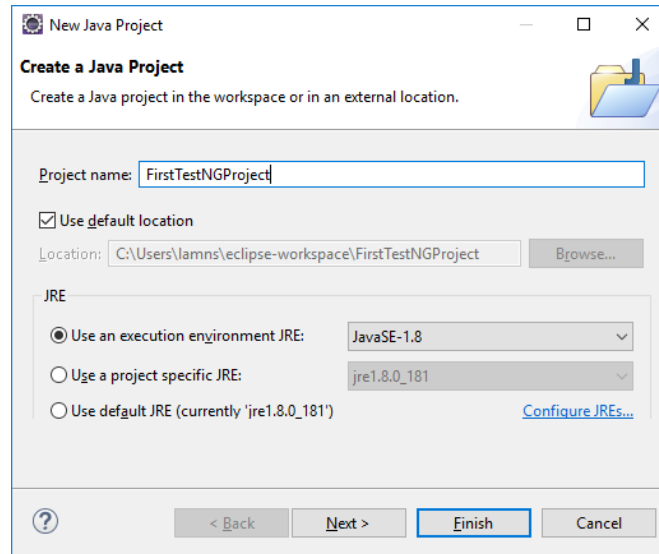
Running testcases in TestNG with out Void main Java

- **Step 1:** Click File > New > Java Project



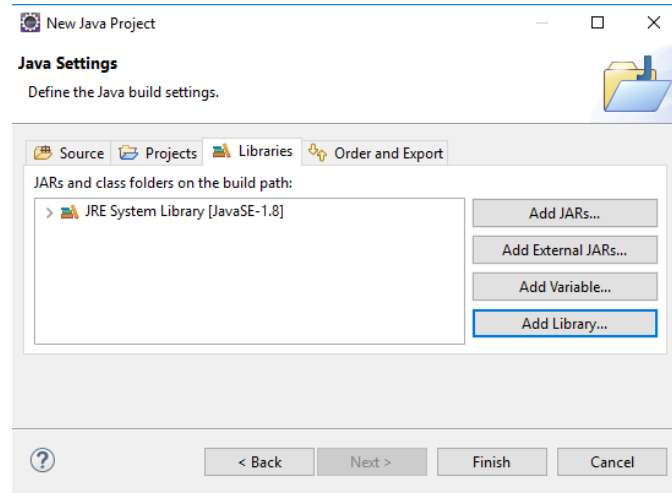
Running testcases in TestNG with out Void main Java

- **Step 2:** Type "FirstTestNGProject" as the Project Name then click. Next



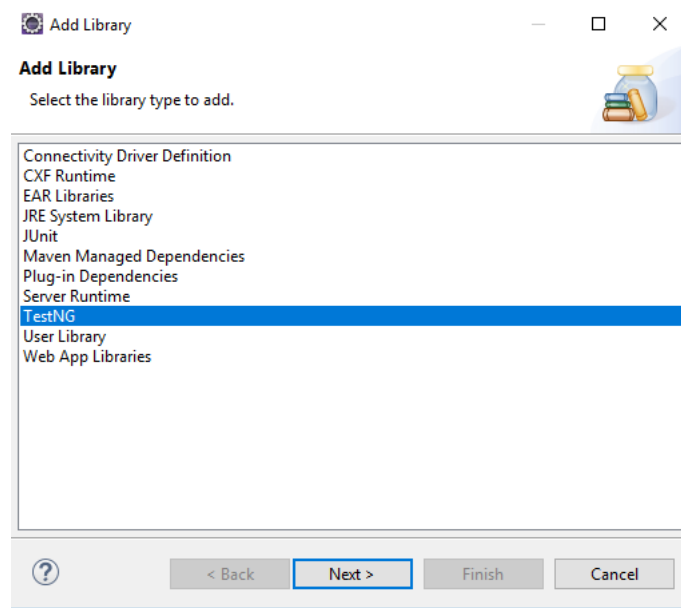
Running testcases in TestNG with out Void main Java

- **Step 3:** We will now start to import the TestNG Libraries onto our project.
Click on the "Libraries" tab, and then "Add Library..."



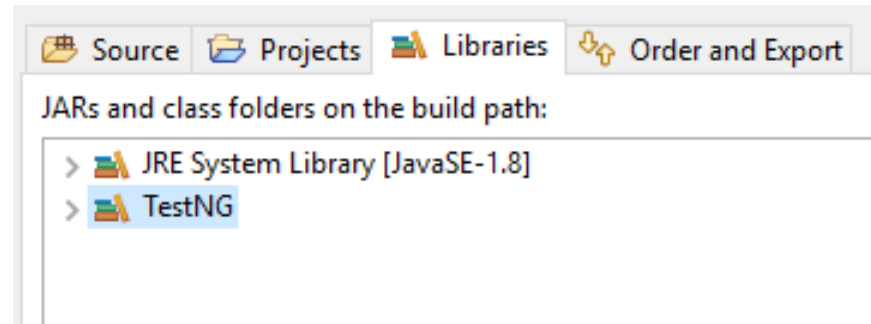
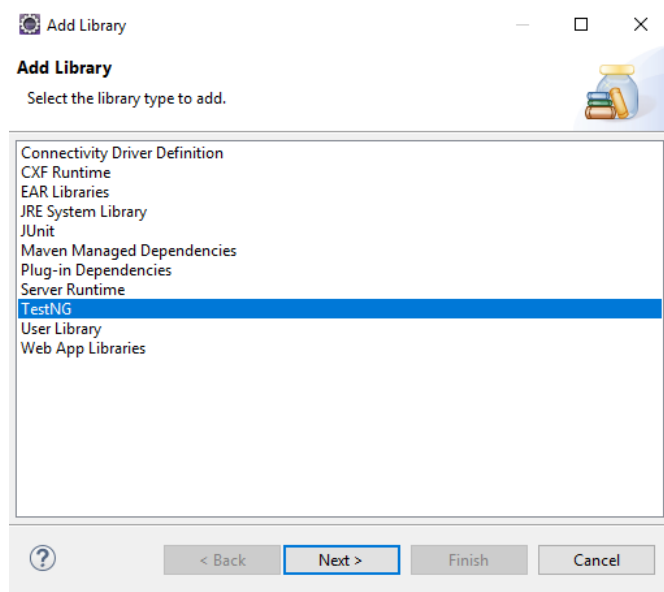
Running testcases in TestNG with out Void main Java

- **Step 4:** On the Add Library dialog, choose "TestNG" and click Next.



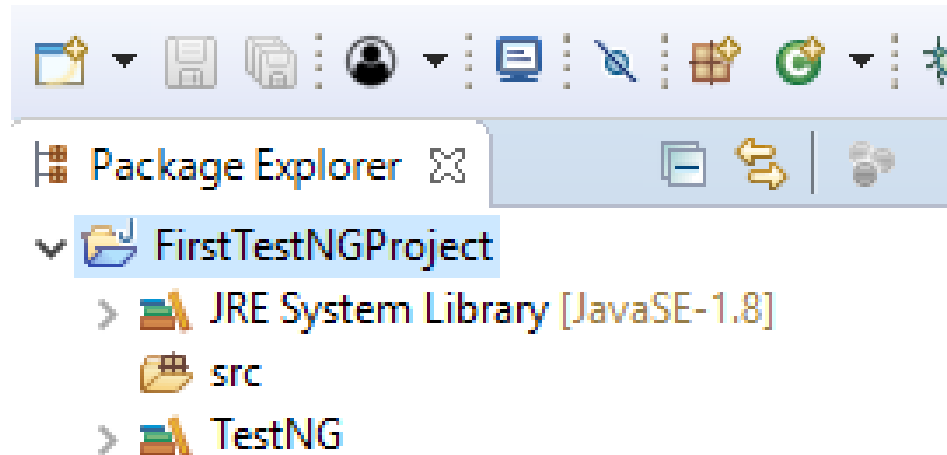
Running testcases in TestNG with out Void main Java

- **Step 5:** On the Add Library dialog, choose "TestNG" and click Next.



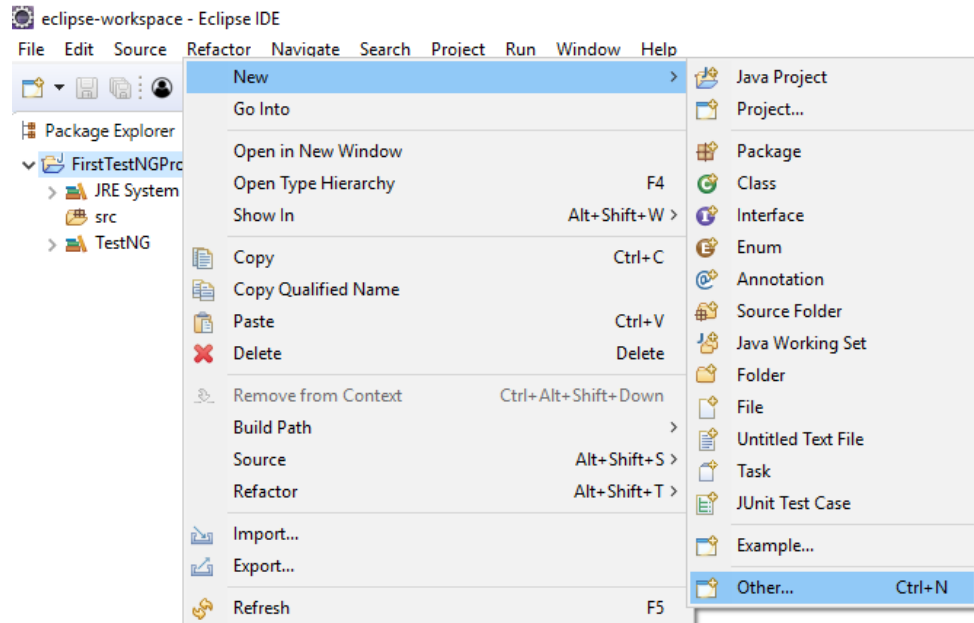
Running testcases in TestNG with out Void main Java

- **Step 6:** Click Finish and verify that our FirstTestNGProject is visible on Eclipse's Package Explorer window.



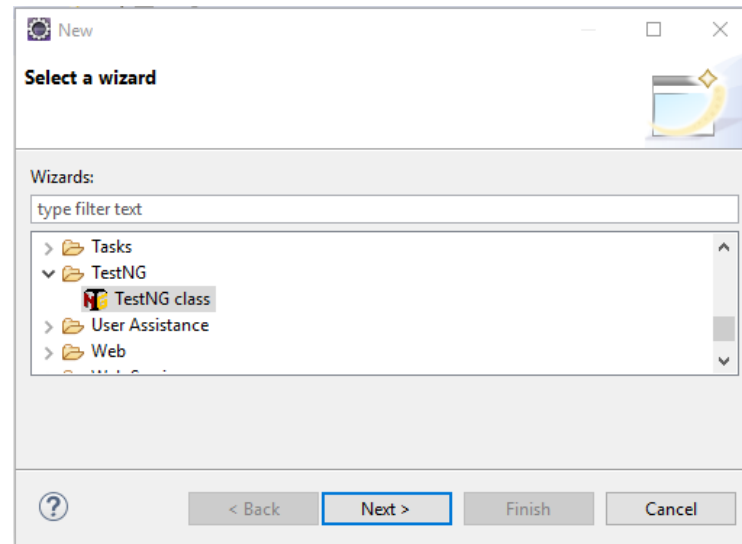
Running testcases in TestNG with out Void main Java

- **Step 7:** Right-click on the "src" package folder then choose New > Other...



Running testcases in TestNG with out Void main Java

- **Step 8:** Click on the TestNG folder and select the "TestNG class" option.
Click Next.



Running testcases in TestNG with out Void main Java

- **Step 9:** Type the values indicated below on the appropriate input boxes and click Finish. Notice that we have named our Java file as "**FirstTestNGFile**".

New TestNG class
Specify additional information about the test class.

Source folder: /FirstTestNGProject/src Browse...

Package name: FirstTestNGFile Browse...

Class name: FirstTestNGFile

Annotations

☐ @BeforeMethod ☐ @AfterMethod ☐ @DataProvider
☐ @BeforeClass ☐ @AfterClass
☐ @BeforeTest ☐ @AfterTest
☐ @BeforeSuite ☐ @AfterSuite

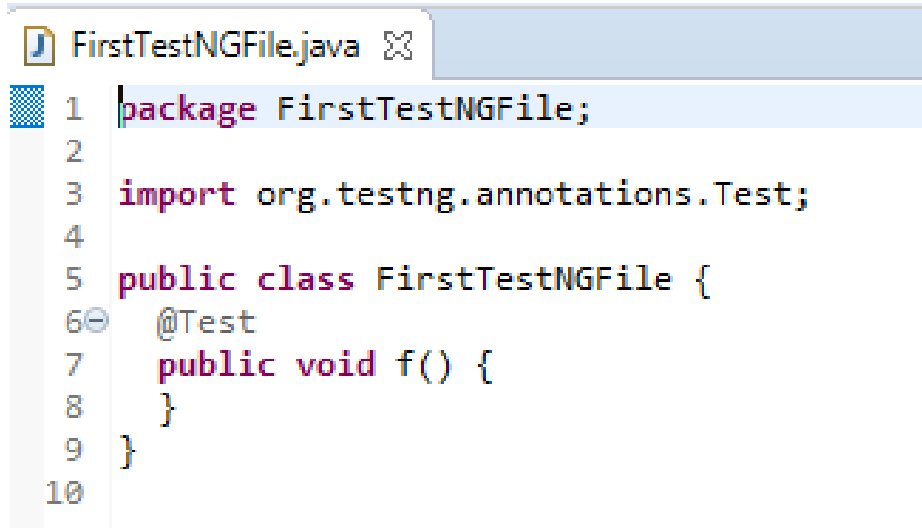
XML suite file:

? < Back Next > Finish Cancel

Running testcases in TestNG with out Void main

Java

Type Eclipse should automatically create the template for our TestNG file shown below.



```
1 package FirstTestNGFile;
2
3 import org.testng.annotations.Test;
4
5 public class FirstTestNGFile {
6     @Test
7     public void f() {
8     }
9 }
10
```

Running testcases in TestNG with out Void main Java

```
package FirstTestNGFile;

import org.testng.annotations.Test;

public class FirstTestNGFile {
    @Test
    public void One() {

        System.out.println("This is the Test Case number One");
    }

    @Test
    public void Two() {

        System.out.println("This is the Test Case number Two");
    }

    @Test
    public void Three() {

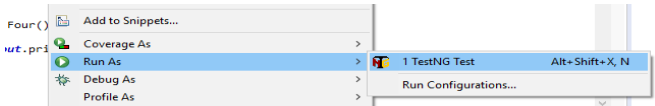
        System.out.println("This is the Test Case number Three");
    }

    @Test
    public void Four() {

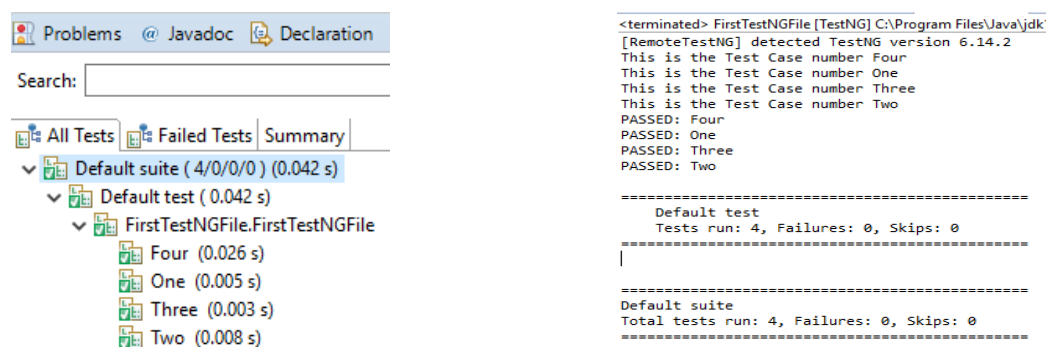
        System.out.println("This is the Test Case number Four");
    }
}
```

Running testcases in TestNG with out Void main Java

To run the test, simply run the file in Eclipse as you normally do. Eclipse will provide two outputs – one in the Console window and the other on the TestNG Results window.



The screenshot shows the Eclipse IDE with a context menu open for a file named 'Four()'. The menu options are: 'Add to Snippets...', 'Coverage As', 'Run As' (highlighted), 'Debug As', and 'Profile As'. The 'Run As' option has a submenu open showing '1 TestNG Test' (highlighted) and 'Run Configurations...'.



The screenshot shows the Eclipse IDE with the TestNG Results window and the Console window open. The TestNG Results window shows a tree view of the test results:

- Default suite (4/0/0/0) (0.042 s)
 - Default test (0.042 s)
 - FirstTestNGFile.FirstTestNGFile
 - Four (0.026 s)
 - One (0.005 s)
 - Three (0.003 s)
 - Two (0.008 s)

The Console window shows the output of the test:

```
<terminated> FirstTestNGFile [TestNG] C:\Program Files\Java\jdk-  
[RemoteTestNG] detected TestNG version 6.14.2  
This is the Test Case number Four  
This is the Test Case number One  
This is the Test Case number Three  
This is the Test Case number Two  
PASSED: Four  
PASSED: One  
PASSED: Three  
PASSED: Two  
  
=====  
Default test  
Tests run: 4, Failures: 0, Skips: 0  
|  
  
=====  
Default suite  
Total tests run: 4, Failures: 0, Skips: 0  
=====
```

Sr. No.

Annotation

Description

1.

@BeforeSuite

This annotation will always be executed before all tests are run inside a TestNG.

2.

@AfterSuite

This annotation will always be executed after all tests are run inside a TestNG.

3.

@BeforeClass

This annotation will always be executed before the first test method in a class is called on.

4.

@AfterClass

This annotation will always be executed after the first test method in a class is called on.

5.

@BeforeTest

Executed before any test method associated with the class inside <test> tag is run.

6.

@AfterTest

This annotation will always be executed before any test method associated with the class inside <test> tag is run.

7.

@BeforeGroups

This annotation will always be executed before the first test method associated with these groups is called on.

8.

@Aftergroups

This annotation will always be executed shortly after the first test method associated with these groups is called on.

9.

@BeforeMethod

This annotation will always be executed before each test method.

10.

@AfterMethod

This annotation will always be executed after each test method.

11.

@DataProvider

This annotation must return object [], it marks a method as supplying data for a test method.

12.

@Factory

This annotation will return the object and marks a method as a factory that will be used by TestNG as test classes.

13.

@Listeners

This annotation defines the listeners on a test class.

14.

@parameters

This annotation describes how parameters can be passed to a test method.

15.

@Test

This annotation marks a class as a part of the test.

@BeforeSuite: The annotated method runs before all tests in this suite.

@BeforeTest: The annotated method runs before test classes.

@BeforeGroups: The annotated method runs before the group's tests.

@BeforeClass: The annotated method runs before the first test method which belongs to any of the test groups.

@BeforeMethod: The annotated method runs before each test method.

@Test: The annotated method is a test method. It executes the test.

@AfterMethod: The annotated method runs after each test method.

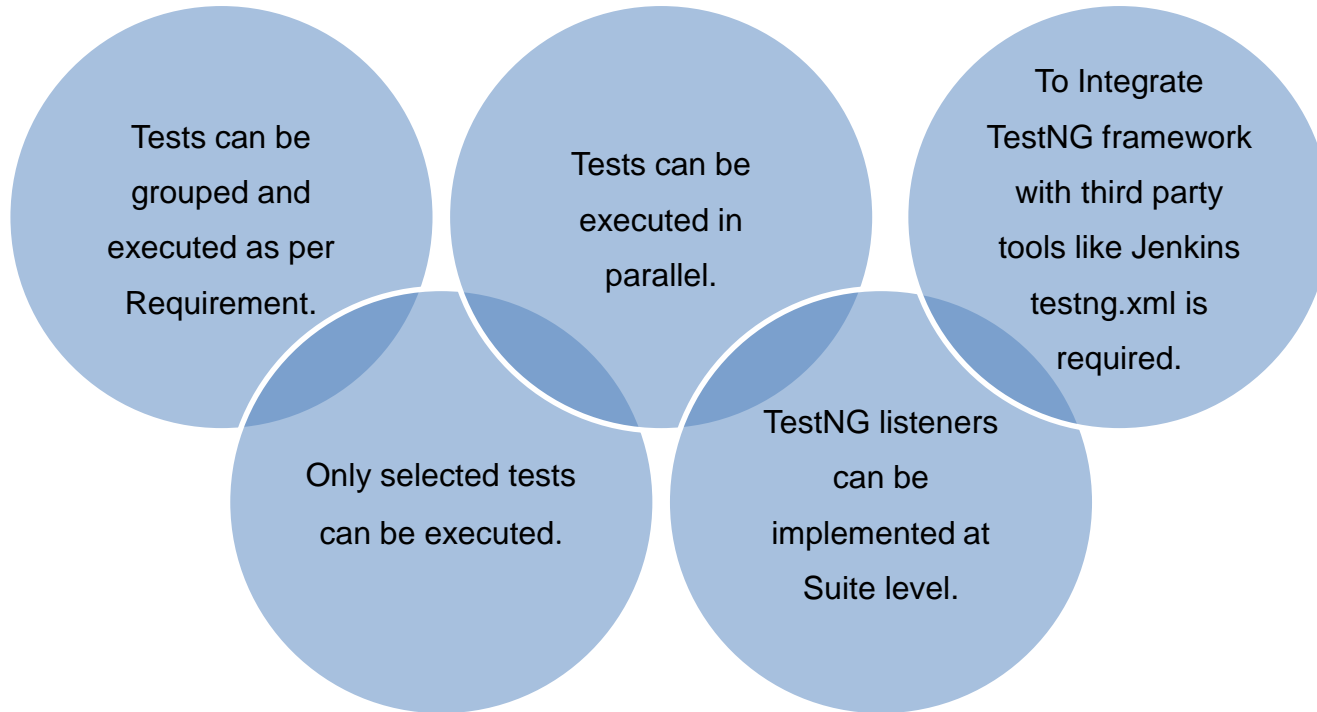
@AfterClass: The annotated method runs after all the test methods in the current test class.

@AfterGroups: The annotated method runs after the last test method which belongs to any of the test groups.

@AfterTest: The annotated method runs after test classes.

@AfterSuite: The annotated method runs after all tests.

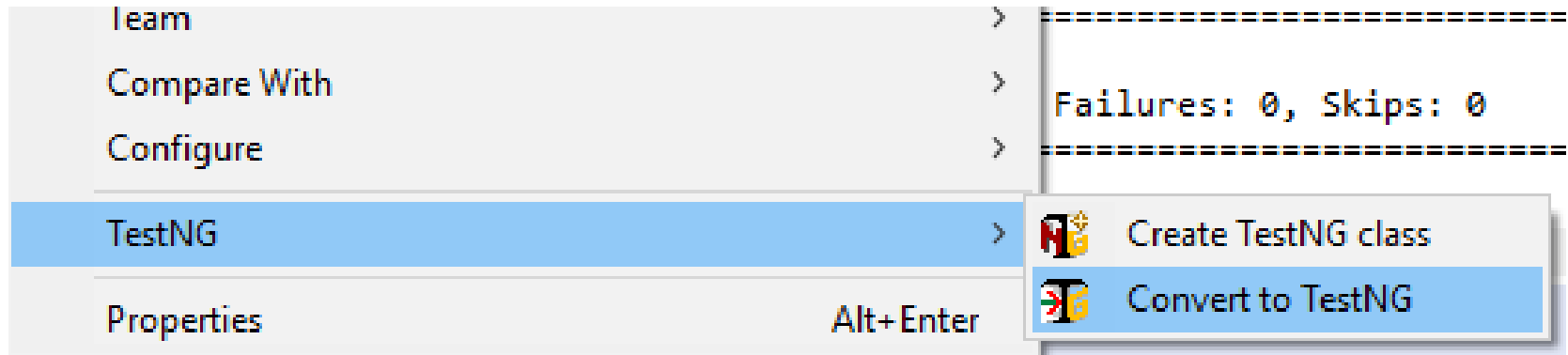

```
public class TestNGAnnotations {  
    @BeforeSuite  
    public void setupSuite() {  
        System.out.println("@BeforeSuite started.");  
    }  
  
    @BeforeTest  
    public void setupTests() {  
        System.out.println("@BeforeTest started.");  
    }  
  
    @BeforeClass  
    public void setupClass() {  
        System.out.println("@BeforeClass started.");  
    }  
  
    @BeforeMethod  
    public void setupTest() {  
        System.out.println("@BeforeMethod has started.");  
    }  
  
    @AfterMethod  
    public void teardownTest() {  
        System.out.println("@AfterMethod has started.");  
    }  
  
    @AfterClass  
    public void teardownClass() {  
        System.out.println("@AfterClass has started.");  
    }  
  
    @AfterTest  
    public void teardownTests() {  
        System.out.println("@AfterTest has started.");  
    }  
  
    @AfterSuite  
    public void teardownSuite() {  
        System.out.println("@AfterSuite has started.");  
    }  
}
```



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test name="Test">
    <classes>
      <class name="packageName.TestName"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

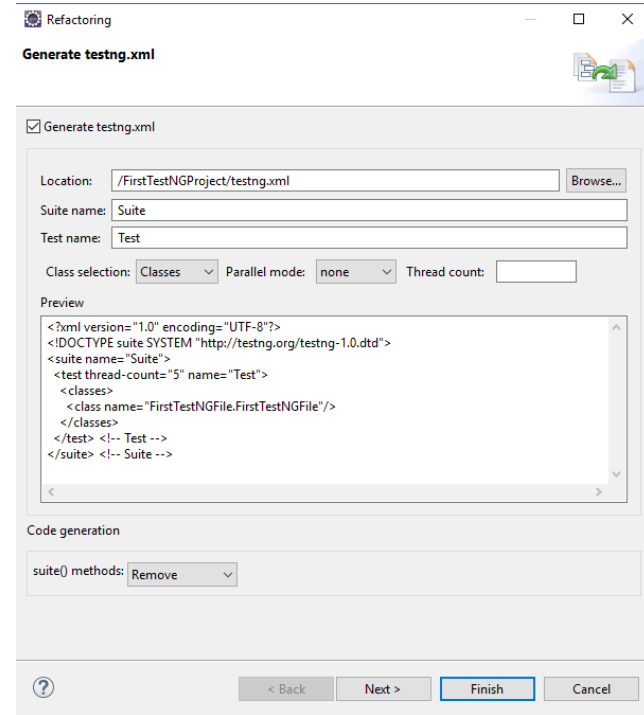
Importance of Testng xml file - Creating testng.xml

- **Step 1:** To create testng.xml, Right click on the java class in the project (that is having tat least one test) --> **TestNG --> Conver to TestNG**



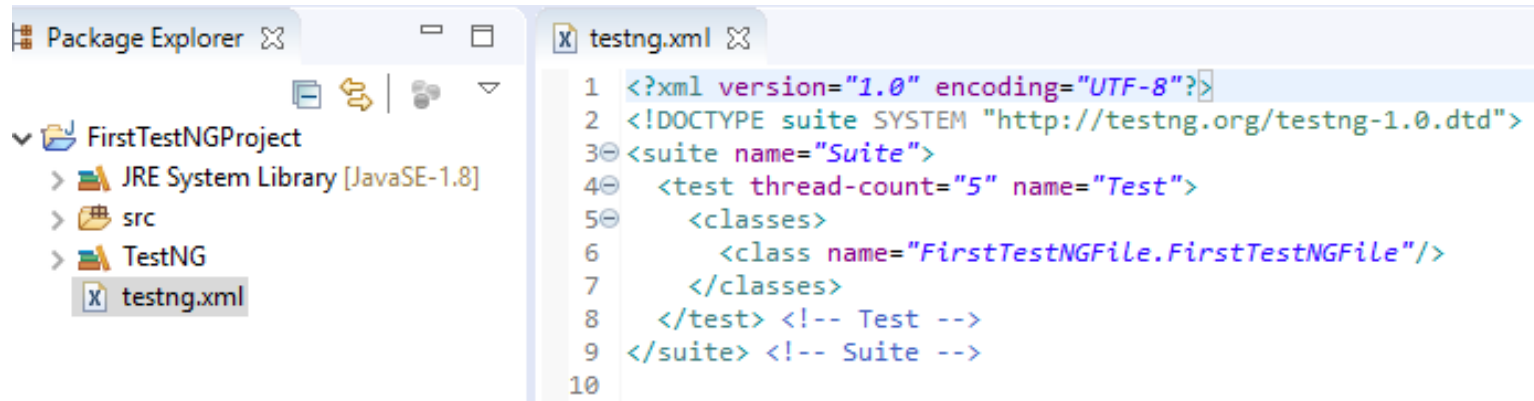
Importance of Testng xml file - Creating testng.xml

- **Step 2:** To A window popup will be shown,
Click Finish



Importance of Testng xml file - Creating testng.xml

- **Step 3:** You will be navigated to the previous window and you should see testng.xml file in the eclipse window as shown in the below screen shot



Prioritizing the tests using TestNG

```
public class FirstTestNGFile {  
    @Test  
    public void One() {  
        System.out.println("This is the Test Case number One");  
    }  
  
    @Test  
    public void Two() {  
        System.out.println("This is the Test Case number Two");  
    }  
    |  
    @Test  
    public void Three() {  
        System.out.println("This is the Test Case number Three");  
    }  
  
    @Test  
    public void Four() {  
        System.out.println("This is the Test Case number Four");  
    }  
}
```

All Tests	Failed Tests	Summary
▼ Default suite (4/0/0/0) (0.042 s)		
▼ Default test (0.042 s)		
▼ FirstTestNGFile.FirstTestNGFile		
Four (0.026 s)		
One (0.005 s)		
Three (0.003 s)		
Two (0.008 s)		

Prioritizing the tests using TestNG

You need to use the '**priority**' parameter, if you want the methods to be executed in your order. Simply assign priority to all @Test methods starting from **0(Zero)**.

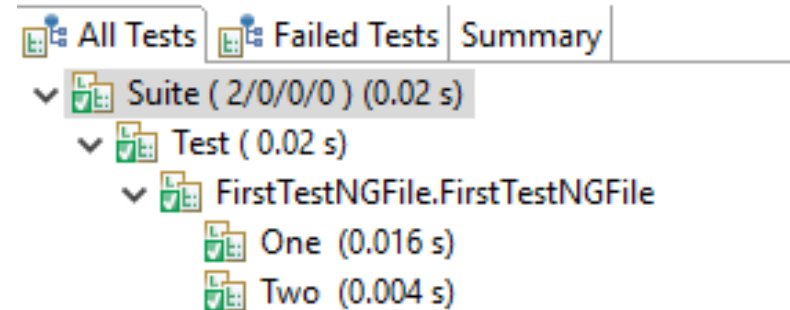
```
public class FirstTestNGFile {  
    @Test(priority = 0)  
    public void One() {  
        System.out.println("This is the Test Case number One");  
    }  
  
    @Test(priority = 1)  
    public void Two() {  
        System.out.println("This is the Test Case number Two");  
    }  
  
    @Test(priority = 2)  
    public void Three() {  
        System.out.println("This is the Test Case number Three");  
    }  
  
    @Test(priority = 3)  
    public void Four() {  
        System.out.println("This is the Test Case number Four");  
    }  
}
```

The screenshot displays the TestNG test results interface. At the top, there are three tabs: 'All Tests', 'Failed Tests', and 'Summary'. The 'All Tests' tab is selected. Below the tabs, the test results are shown in a hierarchical tree structure. The root node is 'Default suite (4/0/0/0) (0.028 s)'. Under this, there is a node 'Default test (0.028 s)'. Under 'Default test', there is a node 'FirstTestNGFile.FirstTestNGFile'. Under 'FirstTestNGFile.FirstTestNGFile', there are four test cases listed in order of execution: 'One (0.018 s)', 'Two (0.003 s)', 'Three (0.003 s)', and 'Four (0.004 s)'. Each test case is preceded by a green checkmark icon, indicating that all tests passed.

Controlling the Testcase execution with Exclude Mechanism

TestNG provides an option to **include or exclude Groups, Test Methods, Classes and Packages** using include and exclude tags by defining in testng.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="FirstTestNGFile.FirstTestNGFile">
        <methods>
          <include name="One" />
          <include name="Two" />
          <exclude name="Three" />
        </methods>
      </class>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```



Executing the Testcases at Package level with regex

```
public class FirstTestNGFile {  
    @Test(priority = 0)  
    public void One() {  
        System.out.println("This is the Test Case number One");  
    }  
  
    @Test(priority = 1)  
    public void Two() {  
        System.out.println("This is the Test Case number Two");  
    }  
  
    @Test(priority = 2)  
    public void Three() {  
        System.out.println("This is the Test Case number Three");  
    }  
  
    @Test(priority = 3)  
    public void Four() {  
        System.out.println("This is the Test Case number Four");  
    }  
}
```

```
public class FirstTestNGFile2 {  
    @Test(priority = 0)  
    public void Five() {  
        System.out.println("This is the Test Case number Five");  
    }  
  
    @Test(priority = 1)  
    public void Six() {  
        System.out.println("This is the Test Case number Six");  
    }  
  
    @Test(priority = 2)  
    public void Seven() {  
        System.out.println("This is the Test Case number Seven");  
    }  
  
    @Test(priority = 3)  
    public void Eight() {  
        System.out.println("This is the Test Case number Eight");  
    }  
}
```

Executing the Testcases at Package level with regex

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="FirstTestNGFile.FirstTestNGFile" />
      <class name="FirstTestNGFile.FirstTestNGFile2" />
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Executing the Testcases at Package level with regex

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="FirstTestNGFile.FirstTestNGFile" />
      <class name="FirstTestNGFile.FirstTestNGFile2" />
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

The screenshot displays the TestNG test results interface. At the top, there are three tabs: 'All Tests', 'Failed Tests', and 'Summary'. The 'All Tests' tab is selected. Below the tabs, a tree view shows the test execution results. The root node is 'Suite (8/0/0/0) (0.039 s)', which is expanded. Under 'Suite', there is a 'Test (0.039 s)' node, also expanded. Under 'Test', there are two main categories: 'FirstTestNGFile.FirstTestNGFile' and 'FirstTestNGFile.FirstTestNGFile2'. Under 'FirstTestNGFile.FirstTestNGFile', there are four test cases: 'Four (0.015 s)', 'One (0.005 s)', 'Three (0.004 s)', and 'Two (0.003 s)'. Under 'FirstTestNGFile.FirstTestNGFile2', there are four test cases: 'Eight (0.002 s)', 'Five (0.004 s)', 'Seven (0.002 s)', and 'Six (0.004 s)'. All test cases are marked with a green checkmark icon, indicating they passed successfully.

Executing the Testcases at Package level with regex

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <packages>
      <package name="FirstTestNGFile"/>
    </packages>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <packages>
      <package name=".*"/>
    </packages>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

TestNG has great feature to define annotations on a class instead of a each test method.

If say suppose there are 10 test methods, where adding **@Test** on a class level is simpler than adding **@Test** for each method.

```
@Test
public void One() {
    System.out.println("This is the Test Case number One");
}

@Test
public void Two() {
    System.out.println("This is the Test Case number Two");
}

@Test
public void Three() {
    System.out.println("This is the Test Case number Three")
}

@Test
public void Four() {
    System.out.println("This is the Test Case number Four");
}
```

Test level TestNG Annotations examples

```
@Test
public void Three() {
    System.out.println("This is the Test Case number Three");
    int e = 1 / 0;
}

@Test(dependsOnMethods={"Three"})
public void Four() {
    System.out.println("This is the Test Case number Four");
}
```

Search:

✓ Passed: 2

✗ Failed: 1

⏸ Skipped: 1



All Tests Failed Tests Summary

- ✓ Default suite (2/1/1/0) (0.015 s)
 - ✗ Default test (0.015 s)
 - ✗ FirstTestNGFile.FirstTestNGFile
 - ✗ Three (0.012 s)
 - ✓ Two (0.002 s)
 - ✓ One (0.001 s)
 - ✗ Four (0 s)

Failure Exception

java.lang.Throwable: Method FirstTe
at org.testng.remote.AbstractRemot

Group test is a new innovative feature in TestNG, which doesn't exist in JUnit framework. It permits you to dispatch methods into proper portions and perform sophisticated groupings of test methods.

```
@Test(groups = { "Regression", "Smoke" })
public void firstTest() {
    System.out.println("1st Test is Started.");
}

@Test(groups = { "Regression", "Smoke" })
public void secondTest() {
    System.out.println("2nd Test is Started.");
}

@Test(groups = { "Regression" })
public void thirdTest() {
    System.out.println("3rd Test is Started.");
}

@Test(groups = { "Medium" })
public void fourthTest() {
    System.out.println("4th Test is Started.");
}

@Test(groups = { "Regression" })
public void fifthTest() {
    System.out.println("5th Test is Started.");
}

@Test(groups = { "Medium" })
public void sixthTest() {
    System.out.println("6th Test is Started.");
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
    <test thread-count="2" name="Test">
        <groups>
            <run>
                <include name="Smoke" />
                <include name="Regression" />
            </run>
        </groups>
        <classes>
            <class name="TestNGExample.TestNGGrouping" />
        </classes>
    </test> <!-- Test -->
</suite> <!-- Suite -->
```

All Tests	Failed Tests	Summary
Suite (4/0/0/0) (0.017 s)		
Test (0.017 s)		
TestNGExample.TestNGGrouping		
fifthTest (0.01 s)		
firstTest (0 s)		
secondTest (0.006 s)		
thirdTest (0.001 s)		

Parameterising from TestNG xml file

TestNG allows the user to pass values to test methods as arguments by using parameter annotations through testng.xml file.

```
public class TestParameters {  
    @Parameters({ "username", "password" })  
    @Test  
    public void testCaseTwo(String username, String password) {  
        System.out.println("Parameter for User Name passed as :- " + username);  
        System.out.println("Parameter for Password passed as :- " + password);  
    }  
}
```

```
<terminated> TestNGExample_TestParameters.xml [TestNG] C:\Progra  
[RemoteTestNG] detected TestNG version 6.14.2  
Parameter for User Name passed as :- testuser  
Parameter for Password passed as :- testpassword  
  
=====  
Suite  
Total tests run: 1, Failures: 0, Skips: 0  
=====
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  
<suite name="Suite">  
    <test thread-count="5" name="Test">  
        <parameter name="username" value="testuser" />  
        <parameter name="password" value="testpassword" />  
        <classes>  
            <class name="TestNGExample.TestParameters" />  
        </classes>  
    </test> <!-- Test -->  
</suite> <!-- Suite -->
```

How to Run a Test Case multiple times with different Values as Input ?

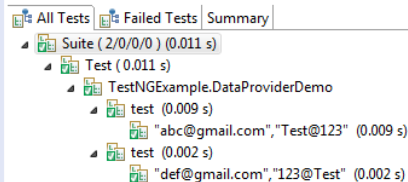


DataProvider Annotation -Parameterizing Testcases

In **TestNG**, there's a concept known as data driven testing, which allows testers to automatically run a test case multiple times, with different input and validation values. You'd want to pass complex parameters or parameters that need to be created from Java, in such cases parameters can be passed using Dataproviders.

@DataProvider

```
public class DataProviderDemo {  
    @DataProvider(name = "FB_Login")  
    public static Object[][] credentials() {  
        return new Object[][] { { "abc@gmail.com", "Test@123" }, { "def@gmail.com", "123@Test" } };  
    }  
    @Test(dataProvider = "FB_Login")  
    public void test(String username, String password) {  
        System.out.println("Parameter for User Name passed as :- " + username);  
        System.out.println("Parameter for Password passed as :- " + password);  
    }  
}
```



The screenshot shows the TestNG test results interface. It includes tabs for 'All Tests', 'Failed Tests', and 'Summary'. The 'All Tests' tab is active, displaying a tree structure of the test suite. The suite 'Suite (2/0/0/0) (0.011 s)' contains a 'Test (0.011 s)' which in turn contains 'TestNGExample.DataProviderDemo'. Under this class, there are two 'test' methods: one for 'abc@gmail.com,Test@123' (0.009 s) and another for 'def@gmail.com,123@Test' (0.002 s). Both tests are marked as passed.

Importance of Listeners in TestNG framework

Listener is defined as interface that modifies the default TestNG's behavior. As the name suggests Listeners "listen" to the event defined in the selenium script and behave accordingly. It is used in selenium by implementing Listeners Interface. It allows customizing TestNG reports or logs. There are many types of TestNG listeners available.

A TestNG listener always extends ***org.testng.ITestNGListener*** and TestNG provides us below listener types:

IExecutionListener

IAnnotationTransformer

ISuiteListener

ITestListener

IConfigurationListener

IMethodInterceptor

IInvokedMethodListener

IHookable

IReporter

ITestListener

- **onStart** is invoked after the test class is instantiated and before any configuration method is called.
- **onTestSuccess** is invoked on success of a test.
- **onTestFailure** is invoked on failure of a test.
- **onTestSkipped** is invoked whenever a test is skipped.
- **onTestFailedButWithinSuccessPercentage** is invoked each time a method fails but is within the success percentage requested.
- **onFinish** is invoked after all the tests have run and all their Configuration methods have been called.

Importance of Listeners in TestNG framework

```
public class SampleTest {
    @Test
    public void test1() {
        System.out.println("I am in test1 test method and it will pass.");
    }

    @Test(expectedExceptions=RuntimeException.class)
    public void test2() {
        System.out.println("I am in test2 test method and it will fail.");
    }

    @Test
    public void test3() {
        throw new SkipException("Skipping the test3 test method!");
    }

    private int i=0;
    @Test(successPercentage=60, invocationCount=5)
    public void test4() {
        i++;
        System.out.println("test4 test method, invocation count: " + i);
        if (i == 1 || i == 2) {
            System.out.println("test4 failed!");
            Assert.assertEquals(i, 8);
        }
    }
}
```

```
public class Listener implements ITestListener{
    @Override
    public void onStart(ITestResult iTestResult) {
        System.out.println("I am in onStart method " + getTestMethodName(iTestResult) + " start");
    }

    @Override
    public void onSuccess(ITestResult iTestResult) {
        System.out.println("I am in onSuccess method " + getTestMethodName(iTestResult) + " succeed");
    }

    @Override
    public void onFailure(ITestResult iTestResult) {
        System.out.println("I am in onFailure method " + getTestMethodName(iTestResult) + " failed");
    }

    @Override
    public void onSkipped(ITestResult iTestResult) {
        System.out.println("I am in onSkipped method " + getTestMethodName(iTestResult) + " skipped");
    }

    @Override
    public void onTestFailedButWithinSuccessPercentage(ITestResult iTestResult) {
        System.out.println("Test failed but it is in defined success ratio " + getTestMethodName(iTestResult));
    }

    @Override
    public void onStart(ITestContext iTestContext) {
        System.out.println("I am in onStart method " + iTestContext.getName());
    }

    @Override
    public void onFinish(ITestContext iTestContext) {
        System.out.println("I am in onFinish method " + iTestContext.getName());
    }

    private static String getTestMethodName(ITestResult iTestResult) {
        return iTestResult.getMethod().getConstructorOrMethod().getName();
    }
}
```

Importance of Listeners in TestNG framework

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <listeners>
    <listener class-name="ITestListenerTest.Listener" />
  </listeners>
  <test thread-count="5" name="Test">
    <classes>
      <class name="ITestListenerTest.SampleTest" />
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

```
<terminated> TestNGExample_SampleTest.xml [TestNG] C:\Program File
[RemoteTestNG] detected TestNG version 6.14.2
I am in onStart method Test
I am in onStart method test1 start
I am in test1 test method and it will pass.
I am in onTestSuccess method test1 succeed
I am in onStart method test2 start
I am in test2 test method and it will fail.
I am in onTestFailure method test2 failed
I am in onStart method test3 start
I am in onTestSkipped method test3 skipped
I am in onStart method test4 start
test4 test method, invocation count: 1
test4 failed!
Test failed but it is in defined success ratio test4
I am in onStart method test4 start
test4 test method, invocation count: 2
test4 failed!
Test failed but it is in defined success ratio test4
I am in onStart method test4 start
test4 test method, invocation count: 3
I am in onTestSuccess method test4 succeed
I am in onStart method test4 start
test4 test method, invocation count: 4
I am in onTestSuccess method test4 succeed
I am in onStart method test4 start
test4 test method, invocation count: 5
I am in onTestSuccess method test4 succeed
I am in onFinish method Test

=====
Suite
Total tests run: 8, Failures: 3, Skips: 1
=====
```

TestNG provides several assertions the most common assertions are **Assert.assertTrue()**, **Assert.assertFalse()**, **Assert.assertEquals()**.

Assert.assertTrue(Condition, Message)

If the condition is not true, an **AssertionError**, with the given message, is thrown.

```
@Test
public void assertTrueExample () {
    //If check is true, test will pass.
    System.out.println("AssertTrue Example");
    String myString = "swtestacademy.com";
    //Condition, Message
    Assert.assertTrue(myString.equals("swtestacademy.com"), "AssertTrue test is failed!");
    System.out.println("AssertTrue test is Passed!\n");
}
```


TestNG provides several assertions the most common assertions are **Assert.assertTrue()**, **Assert.assertFalse()**, **Assert.assertEquals()**.

Assert.assertEquals(String actual,String expected, String message)

If actual and expected conditions are not equal, **AssertionError** with the given message is thrown.

```
@Test
public void assertEqualsExample () {
    System.out.println("AssertEquals Example");
    String myString = "swtestacademy.com";
    //Actual String, Expected String, Message
    Assert.assertEquals("swtestacademy.com", myString, "AssertEquals test is failed!");
    System.out.println("AssertEquals Test is Passed!\n");
}
```

<https://testng.org/doc/documentation-main.html>

Happy Coding

