TestNG Framework P1 : Introduction

TestNG Introduction

TestNG is a testing testing framework inspired from JUnit and NUnit but introducing some new functionality that make it more powerful and easier to use.

TestNG is an open source automated testing framework; where NG of TestNG means Next Generation. TestNG is similar to JUnit but it is much more powerful than JUnit but still it's inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. Pay special thanks to Cedric Beust who is the creator of TestNG.

TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parameterizing.

Benefits of TestNG

There are number of benefits of TestNG but from Selenium perspective, major advantages of TestNG are :

1) It gives the ability to produce HTML Reports of execution

2) Annotations made testers life easy

3) Test cases can be Grouped & Prioritized more easily

4) Parallel testing is possible

5) Generates Logs

6) Data Parametrise action is possible

Test Case Writing

Writing a test in TestNG is quite simple and basically involves following steps:

Step 1 – Write the business logic of the test

Step 2 – Insert TestNG annotations in the code

Step 3 – Add the information about your test (e.g. the class names, methods names, groups names etc…) in a testng.xml file

Step 4 – Run TestNG

Annotations in TestNG

@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 tag is run.

@AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the 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.

@Test: The annotated method is a part of a test case.

Benefits of using annotations

1) TestNG identifies the methods it is interested in by looking up annotations. Hence method names are not restricted to any pattern or format.

2) We can pass additional parameters to annotations.

3) Annotations are strongly typed, so the compiler will flag any mistakes right away.

4) Test classes no longer need to extend anything (such as Test Case, for JUnit 3).

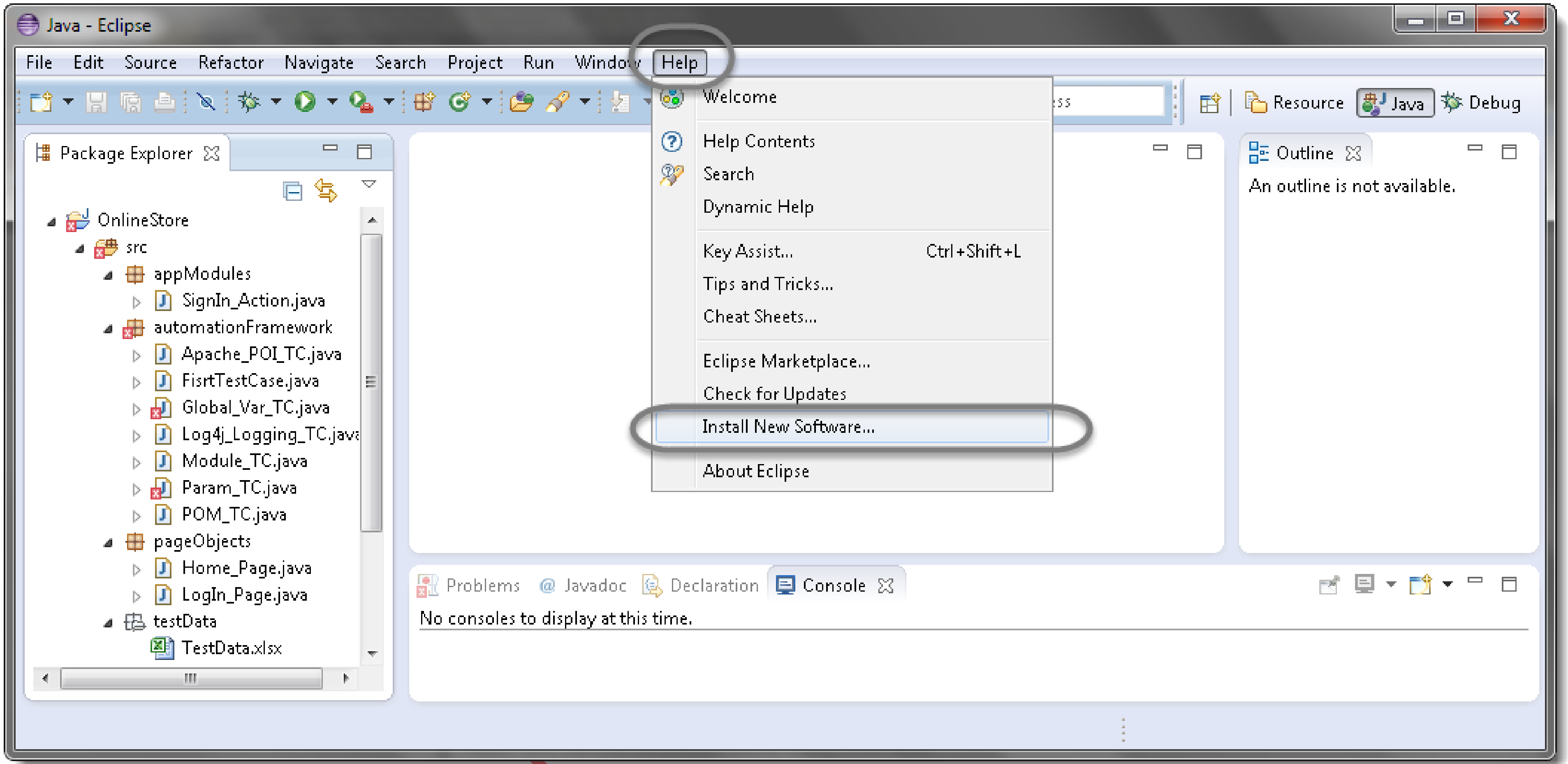
TestNG Framework P2 : Download

Install TestNG in Eclipse

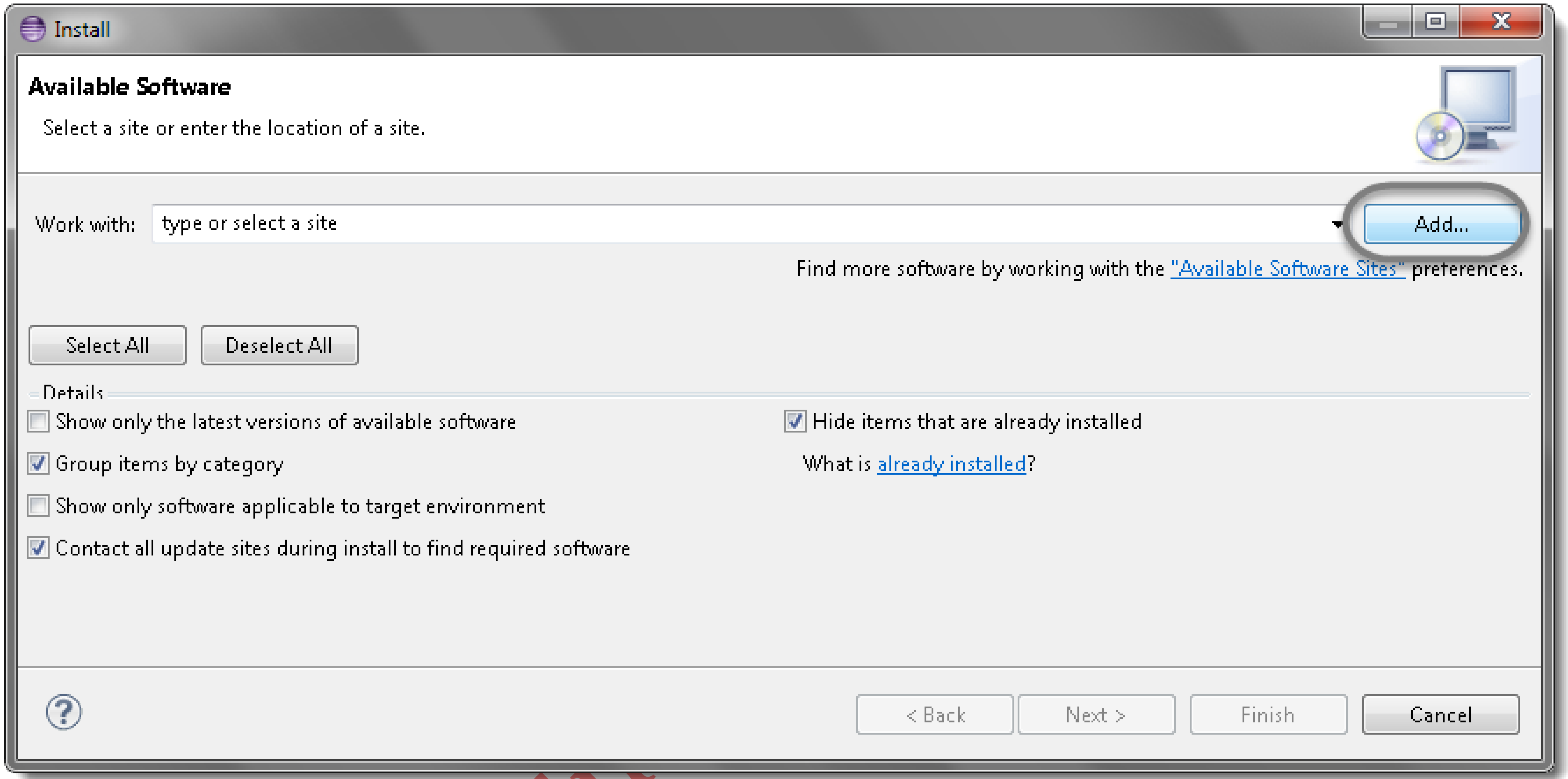
It is easy to install TestNG, as it comes as a plugin for Eclipse IDE. Prerequisite for installing TestNG is your Internet connection should be up & running during installation of this plugin and Eclipse IDE should be installed in your computer. Please see Download and Install Eclipse to setup Eclipse to you system.

Steps to follow:

1) Launch the Eclipse IDE and from Help menu, click "Install New Software".



2) You will see a dialog window, click "Add" button.

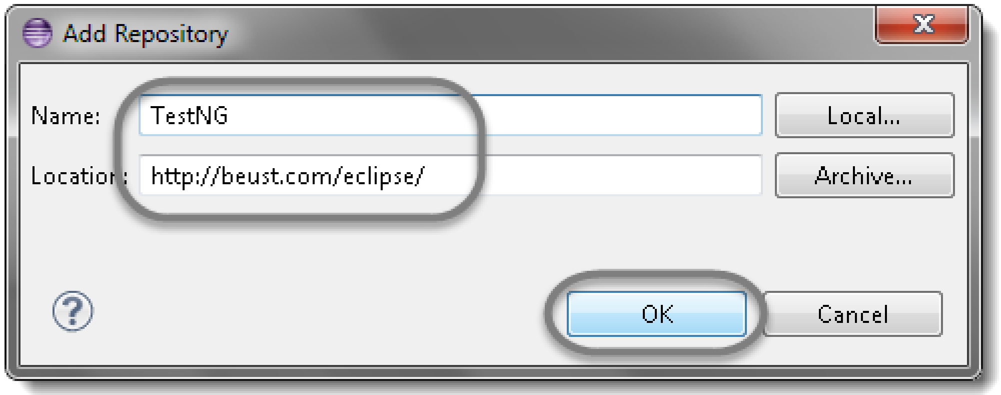


Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

- [ ] Show only the latest versions of available software
- [x] Hide items that are already installed
- [x] Group items by category
- [ ] Show only software applicable to target environment
- [x] Contact all update sites during install to find required software

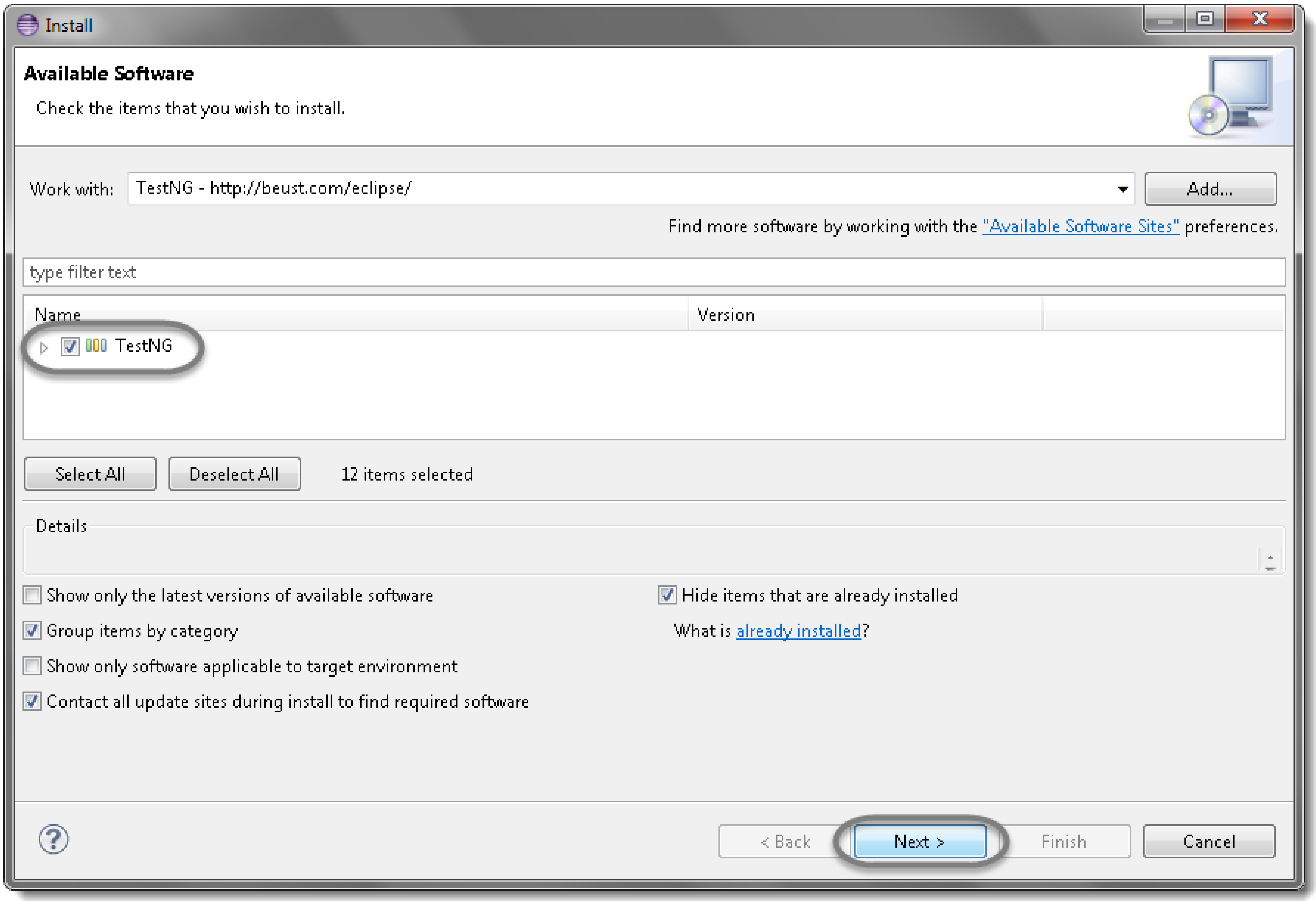
3) Type name as you wish, lets take "TestNG" and type "http://beust.com/eclipse/%E2%80%9D as location. Click OK.



Name: TestNG

Location: http://beust.com/eclipse/

4) You come back to the previous window but this time you must see TestNG option in the available software list. Just Click TestNG and press “Next” button.



Install

Available Software

Check the items that you wish to install.

Work with: TestNG - http://beust.com/eclipse/

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
[✓] TestNG	

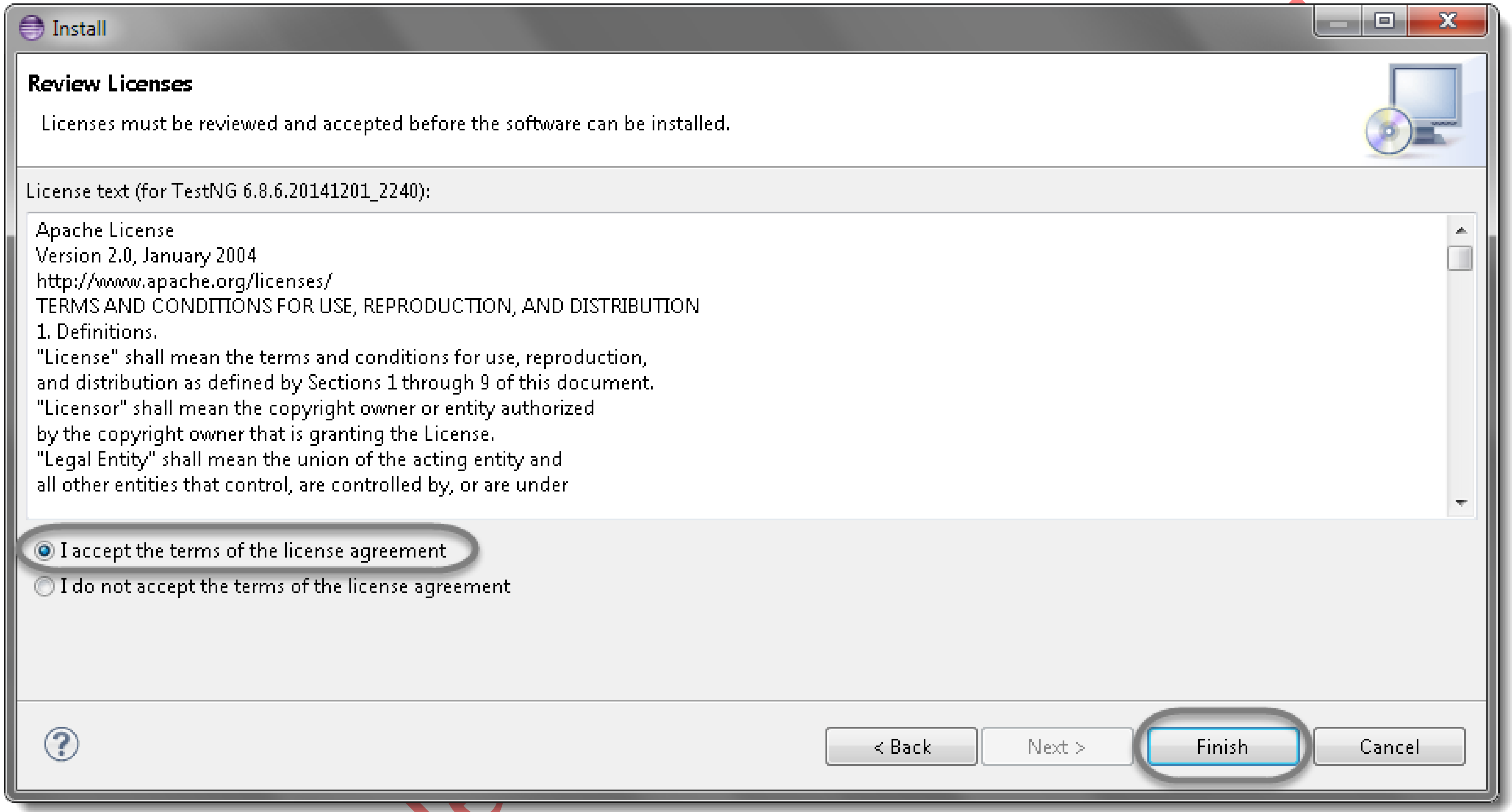
12 items selected

Details

- [ ] Show only the latest versions of available software
- [✓] Group items by category
- [ ] Show only software applicable to target environment
- [✓] Contact all update sites during install to find required software
- [✓] Hide items that are already installed

What is already installed?

5) Click “I accept the terms of the license agreement” then click Finish.



Install

Review Licenses

Licenses must be reviewed and accepted before the software can be installed.

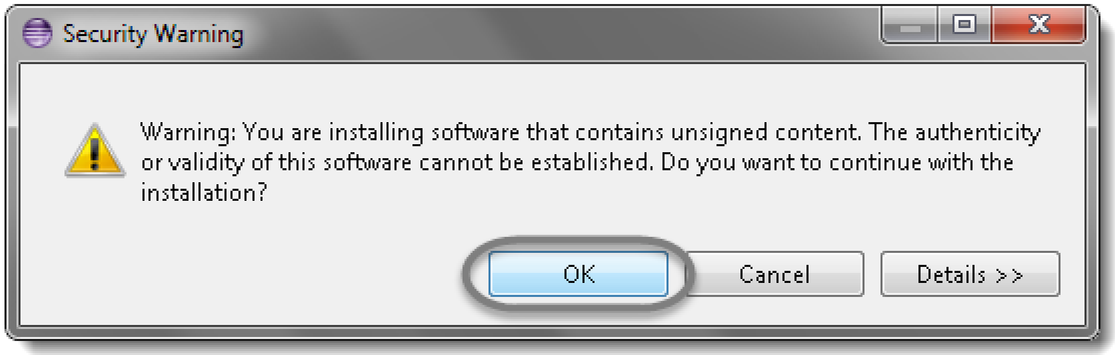
License text (for TestNG 6.8.6.20141201_2240):

```
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.
"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.
"Licensor" shall mean the copyright owner or entity authorized
by the copyright owner that is granting the License.
"Legal Entity" shall mean the union of the acting entity and
all other entities that control, are controlled by, or are under
```

● I accept the terms of the license agreement

○ I do not accept the terms of the license agreement

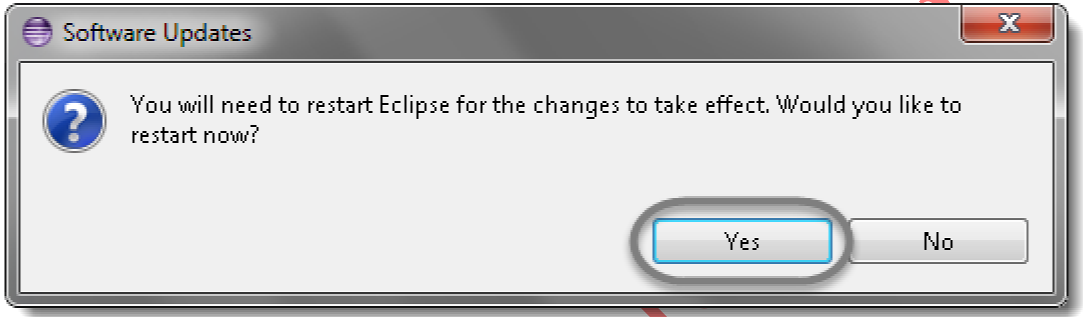
6) You may or may not encounter a Security warning, if in case you do just click OK.


Security Warning

Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the installation?

7) Click Next again on the succeeding dialog box until it prompts you to Restart the Eclipse.

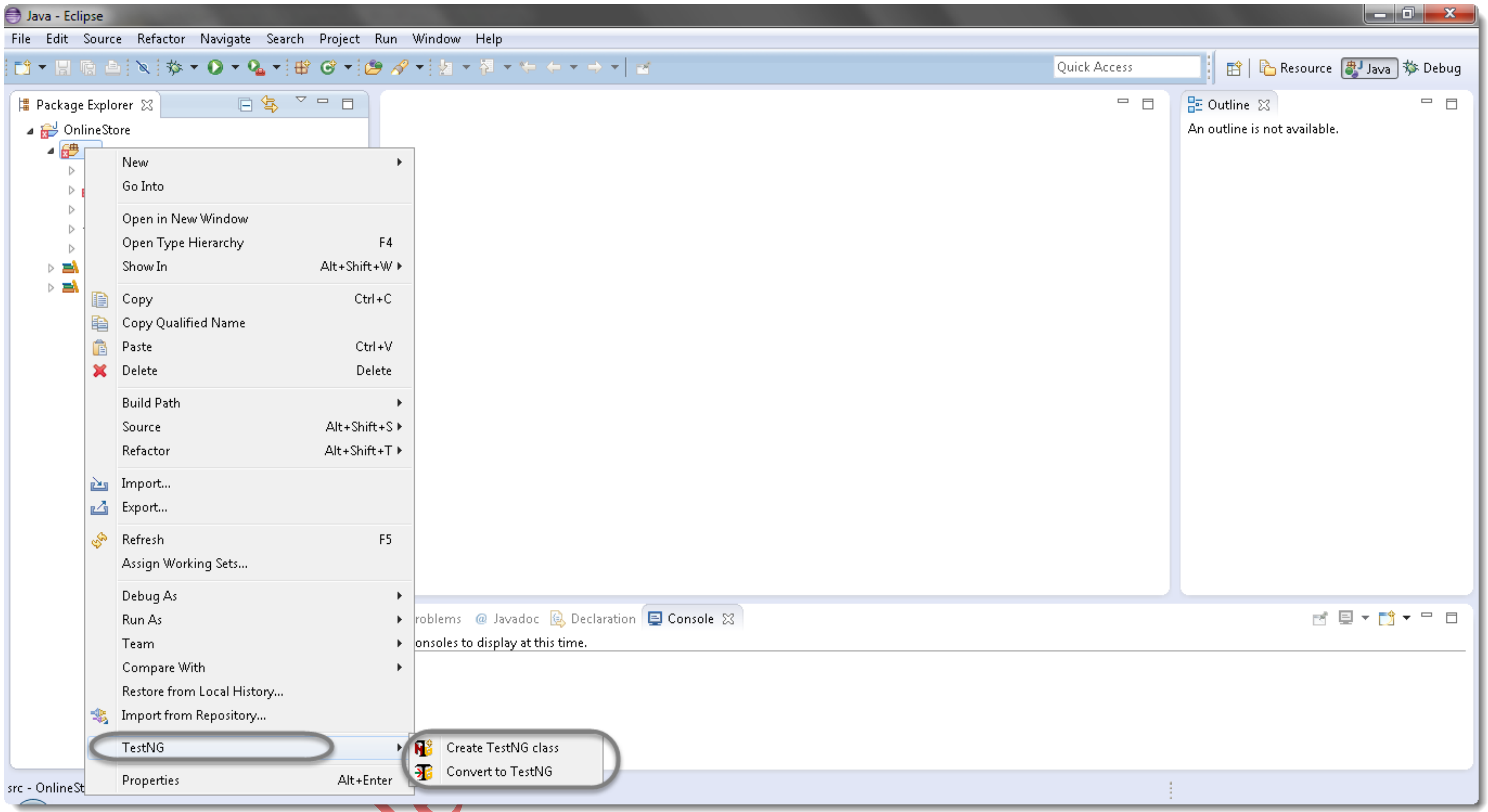
8) You are all done now, just Click Yes.


Software Updates

You will need to restart Eclipse for the changes to take effect. Would you like to restart now?

9) Proceed with your workplace.

10) After restart, verify if TestNG was indeed successfully installed. Right click on you project and see if TestNG is displayed in the opened menu.


Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Resource Java Debug
Package Explorer
OnlineStore
Outline
An outline is not available.
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Debug As
Run As
Team
Compare With
Restore from Local History...
Import from Repository...
TestNG
Properties Alt+Enter
Create TestNG class
Convert to TestNG
roblems Javadoc Declaration Console
onsoles to display at this time.
src - OnlineSt

TestNG Framework P3 : First Test Case With TestNG

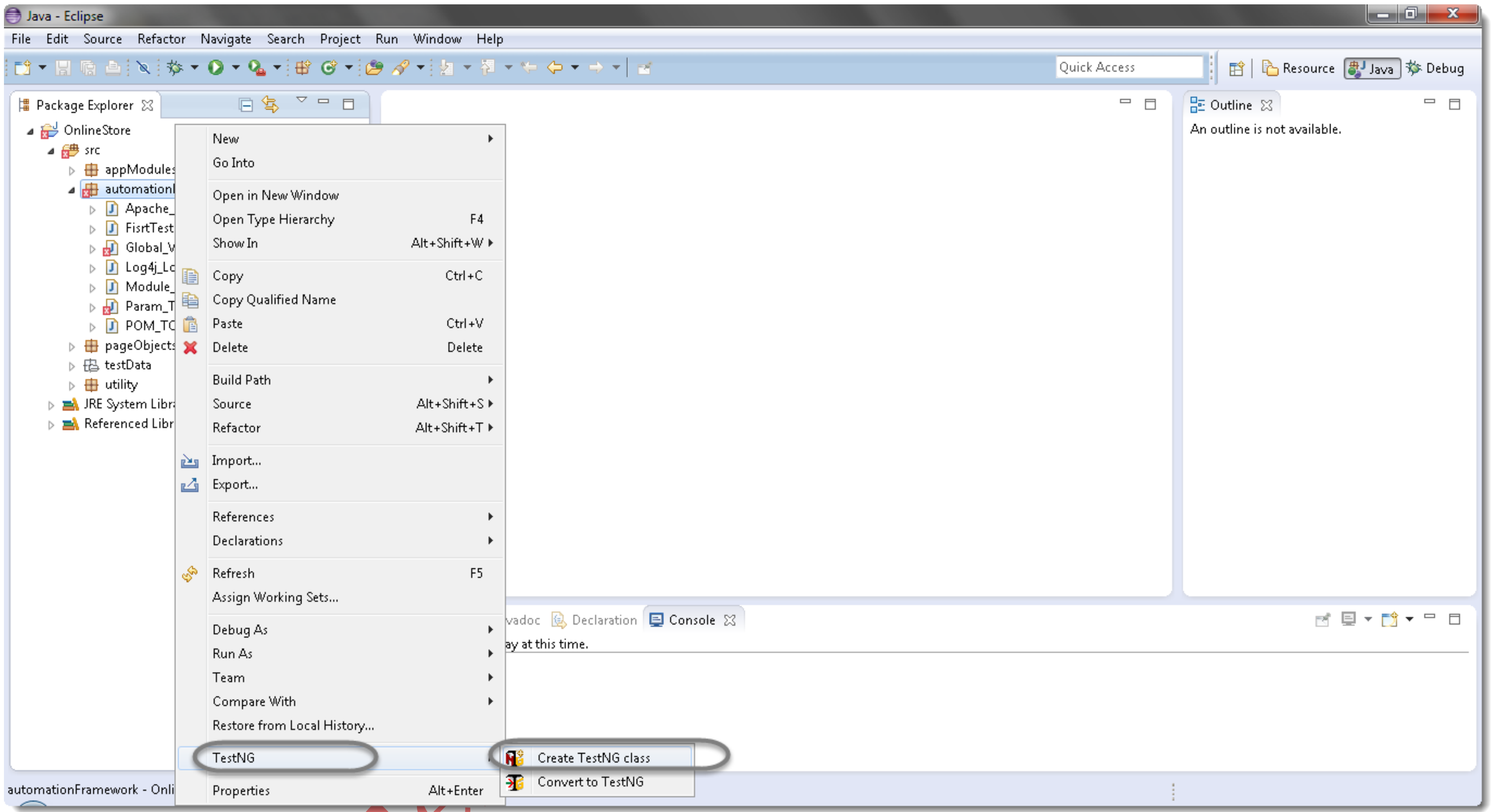
First Test case with TestNG

Steps to follow:

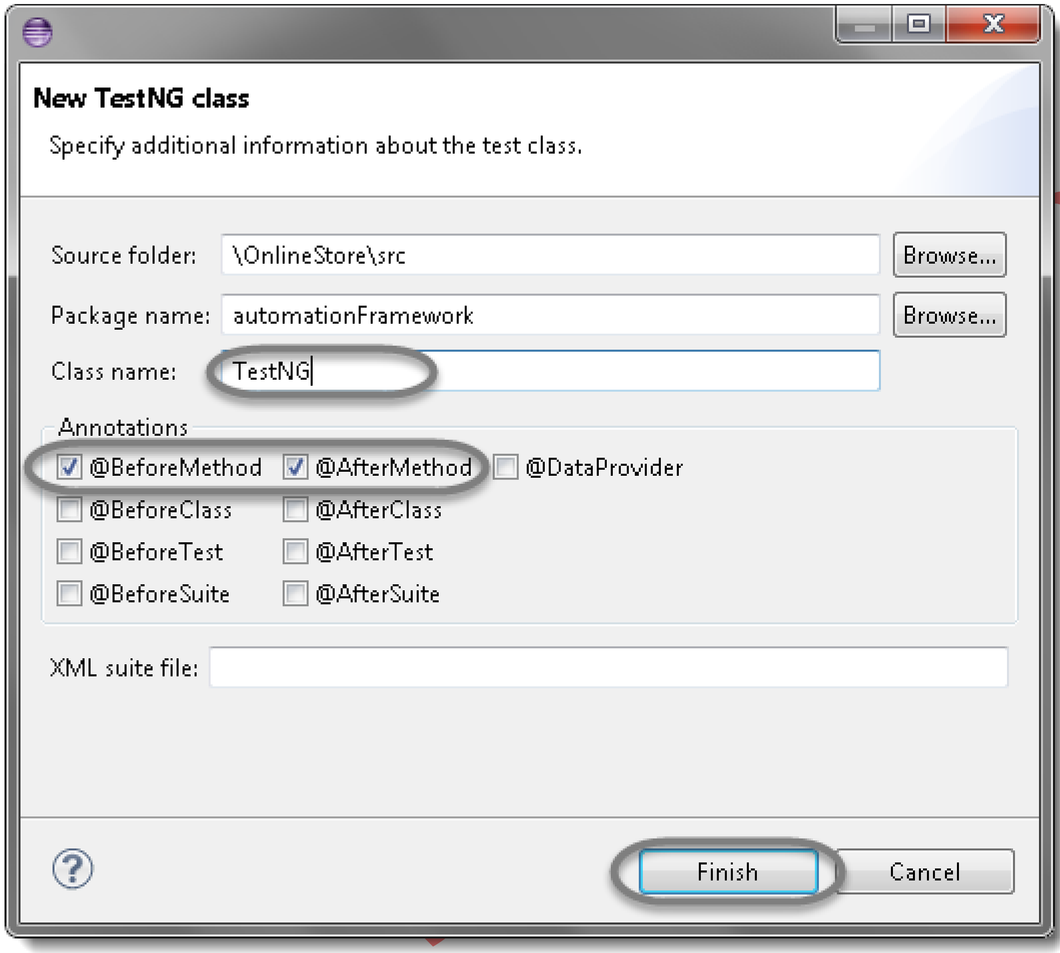
1) Press Ctrl+N , select “TestNG Class” under TestNG category and click Next.

Or

Right click on Test Case folder, go to TestNG and select “TestNG Class“.


Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Resource Java Debug
Package Explorer
OnlineStore
src
appModules
automationI
Apache_
FisrtTest
Global_V
Log4j_Lo
Module_
Param_T
POM_TC
pageObjects
testData
utility
JRE System Libr
Referenced Libr
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Debug As
Run As
Team
Compare With
Restore from Local History...
TestNG
Properties Alt+Enter
Create TestNG class
Convert to TestNG
Outline
An outline is not available.
vadoc Declaration Console
ay at this time.
automationFramework - Onli

2) If your project is set up and you have selected the Test Case folder before creating TestNG class then the source folder and the package name will be prepopullated on the form. Set class name as ‘TestNG‘. Under Annotations, check “@BeforeMethod”, “@AfterMethod” and click Finish. That’s it.


New TestNG class

Specify additional information about the test class.

Source folder: \OnlineStore\src

Package name: automationFramework

Class name: TestNG

Annotations

[✓ @BeforeMethod] [✓ @AfterMethod] @DataProvider

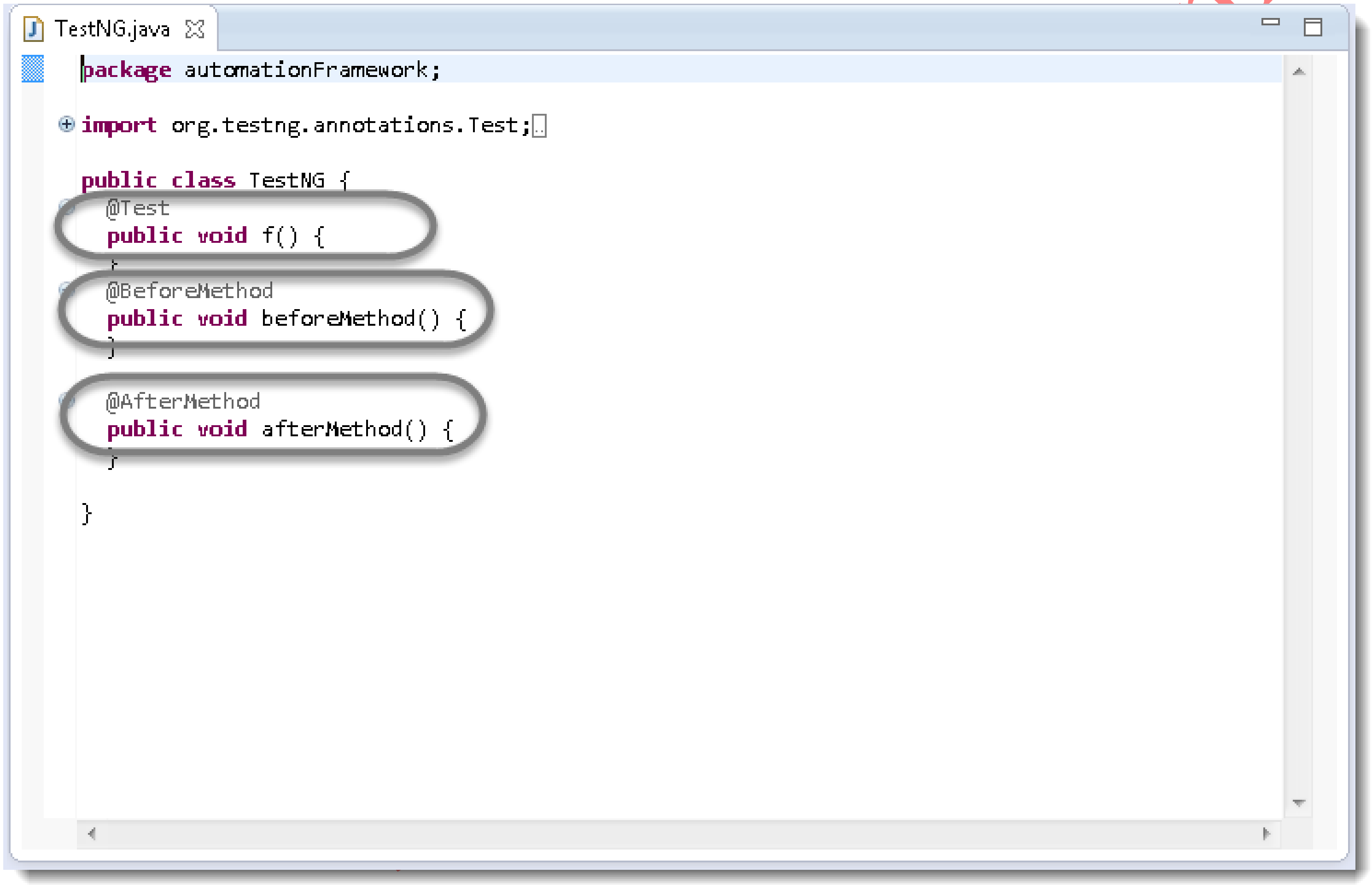
@BeforeClass @AfterClass

@BeforeTest @AfterTest

@BeforeSuite @AfterSuite

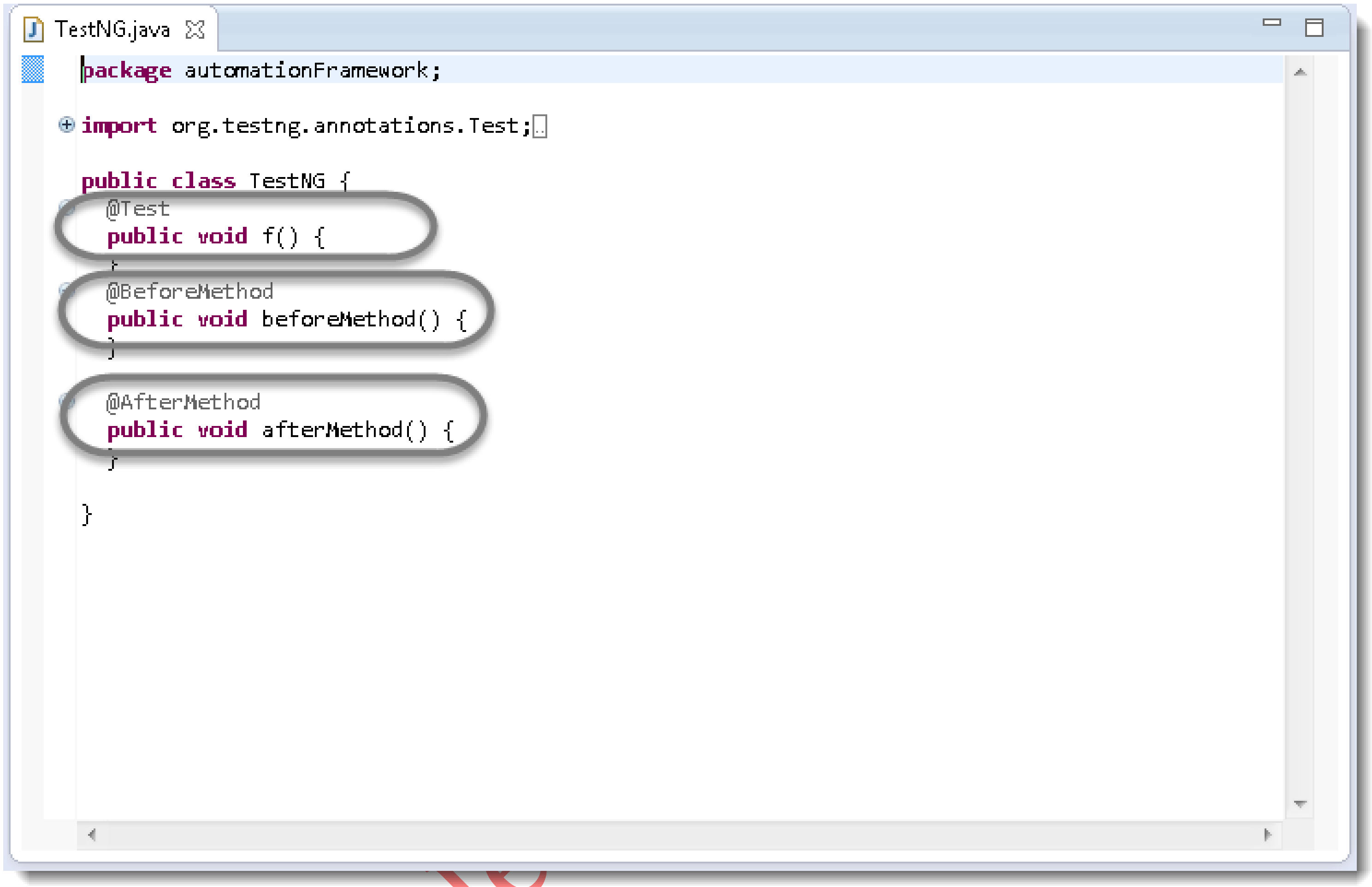
XML suite file:

3) Now it will display the newly created TestNg class under the Test Case package(folder). TestNG class will look like the image below with displaying three empty methods. One method f() by default and before & after method, as selected during the creation of the class.



```
package automationFramework;

import org.testng.annotations.Test;

public class TestNG {
  @Test
  public void f() {
  }
  @BeforeMethod
  public void beforeMethod() {
  }

  @AfterMethod
  public void afterMethod() {
  }

}
```

4) Project explorer will look like this with TestNG class.



```
package automationFramework;

import org.testng.annotations.Test;

public class TestNG {
    @Test
    public void f() {
    }
    @BeforeMethod
    public void beforeMethod() {
    }

    @AfterMethod
    public void afterMethod() {
    }

}
```

Now it is the time to write the first TestNG test case.

5) Let’s take an example of First Test Case and divide the test case in to three parts .

@BeforeMethod : Launch Firefox and direct it to the Base URL

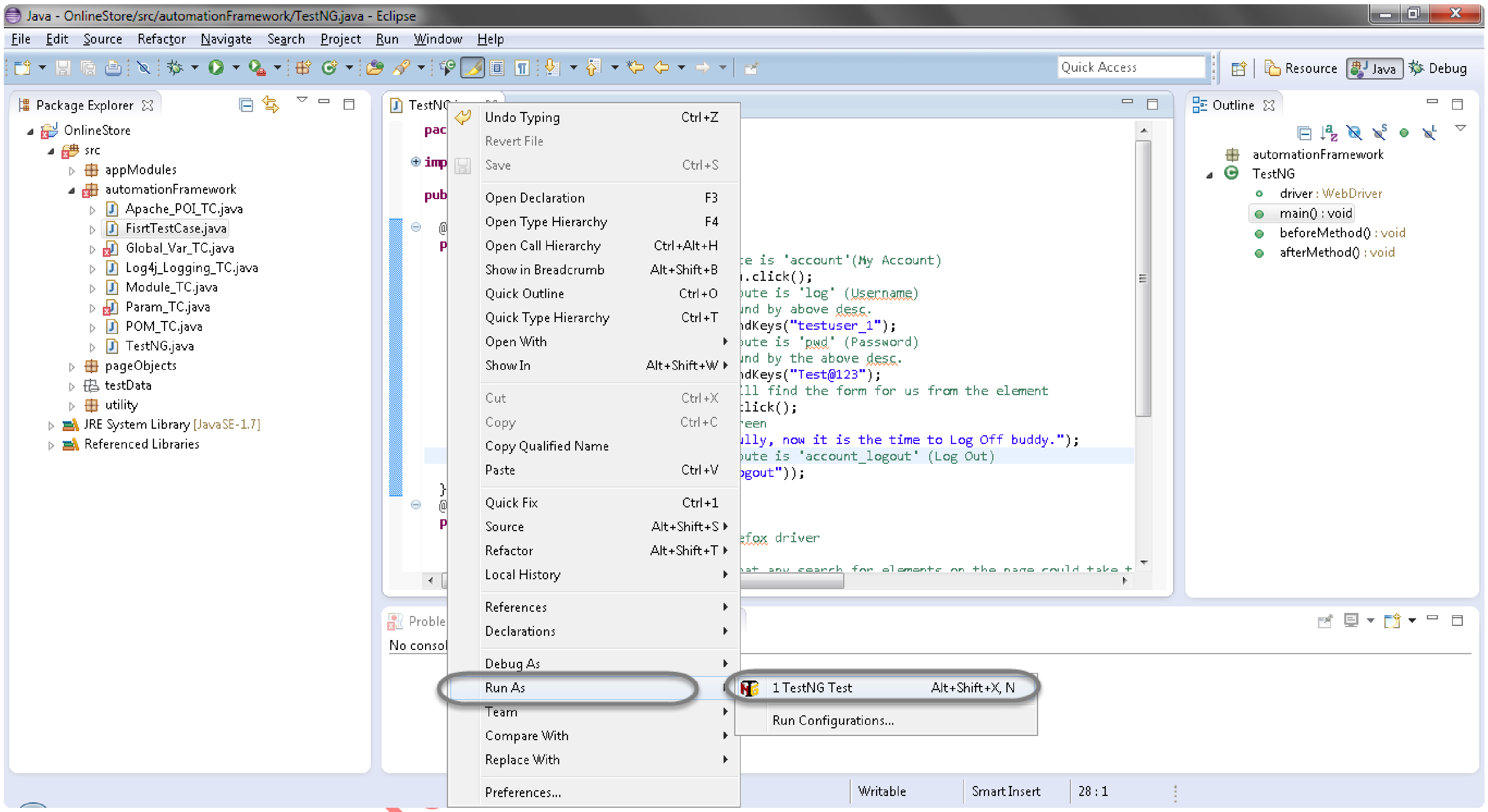
@Test : Enter Username & Password to Login, Print console message

@AfterMethod : Close Firefox browser

```
package automationFramework;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.AfterMethod;

public class TestNG {
  public WebDriver driver;

  @Test
  public void main() {
  // Find the element that's ID attribute is 'account'(My Account)
      driver.findElement(By.id("account")).click();
      // Find the element that's ID attribute is 'log' (Username)
      // Enter Username on the element found by above desc.
      driver.findElement(By.id("email")).sendKeys("yourusername");
      // Find the element that's ID attribute is 'pwd' (Password)
      // Enter Password on the element found by the above desc.
      driver.findElement(By.id("pass")).sendKeys("yourpassword");
      // Now submit the form. WebDriver will find the form for us from the element
      driver.findElement(By.id("loginbutton")).click();
      // Print a Log In message to the screen
      System.out.println(" Login Successfully");
  }
  @BeforeMethod
  public void beforeMethod() {
    // Create a new instance of the Firefox driver
```

```
      driver = new FirefoxDriver();
      //Put a Implicit wait, this means that any search for elements on the page could take the time the implicit
wait is set for before throwing exception
      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
      //Launch the Online Store Website
      driver.get("https://www.facebook.com/");
  }

  @AfterMethod
  public void afterMethod() {
    // Close the driver
      driver.quit();
  }

}
```

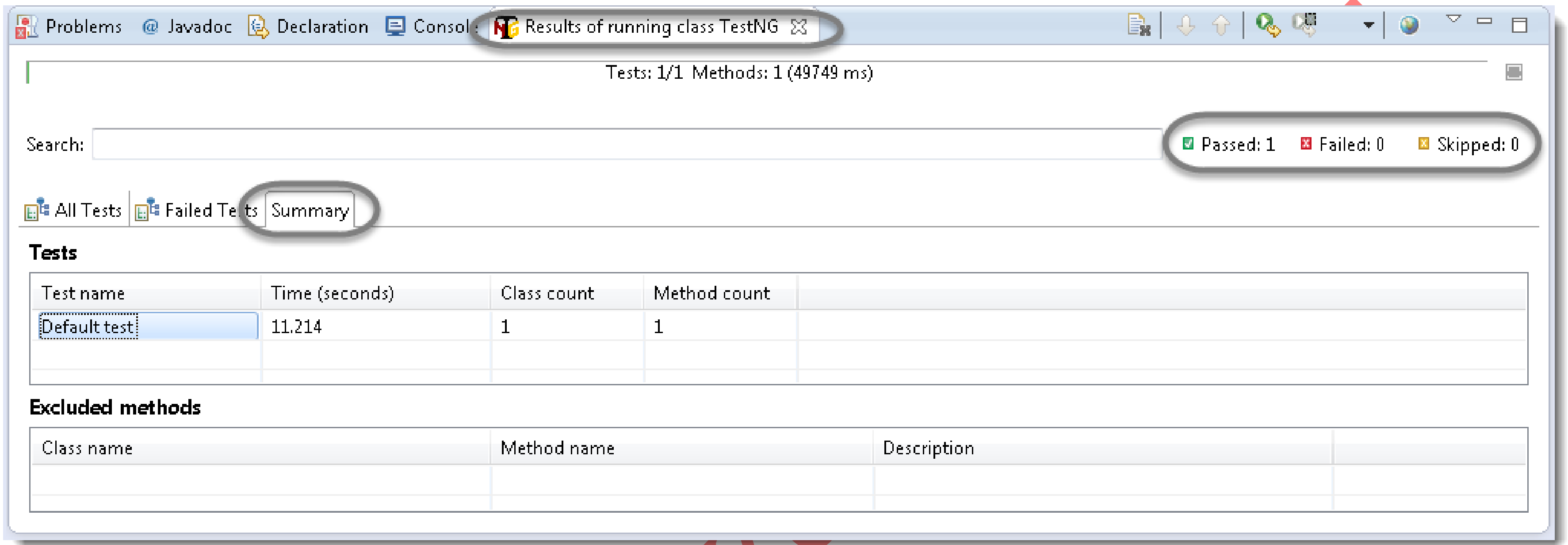
6) Run the test by right click on the test case script and select Run As > TestNG Test.



```
e is 'account'(My Account)
.click();
bute is 'log' (Username)
und by above desc.
ndKeys("testuser_1");
bute is 'pwd' (Password)
und by the above desc.
ndKeys("Test@123");
ill find the form for us from the element
click();
reen
ully, now it is the time to Log Off buddy.");
bute is 'account_logout' (Log Out)
ogout"));

efox driver
```

Results of running the Testng Test Case

7) Give it few minutes to complete the execution, once it is finished the results will look like this in the TestNg Result window.



Tests: 1/1 Methods: 1 (49749 ms)

Search:

Passed: 1 Failed: 0 Skipped: 0

Tests

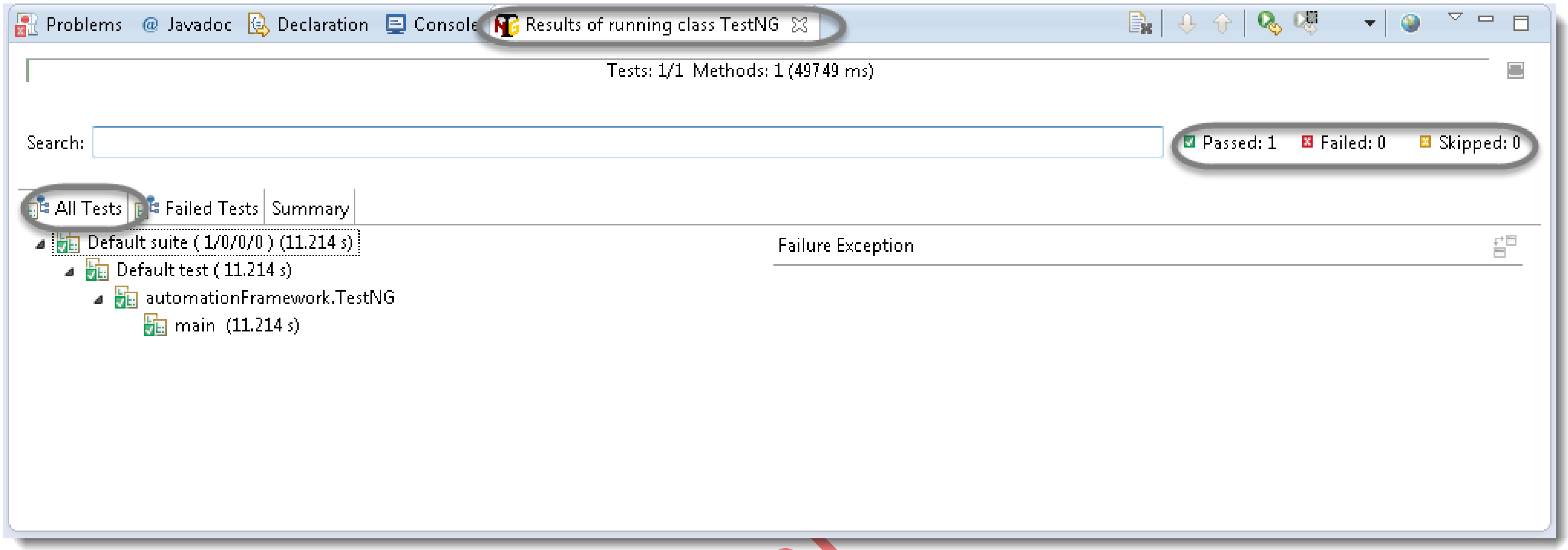
Test name	Time (seconds)	Class count	Method count
Default test	11.214	1	1

Excluded methods

Class name	Method name	Description

It displayed 'passed : 1'. This means test is successful and Passed.

There are 3 sub tabs. "All Tests", "Failed Tests" and "Summary". Just click "All Tests" to see what is there.



Tests: 1/1 Methods: 1 (49749 ms)

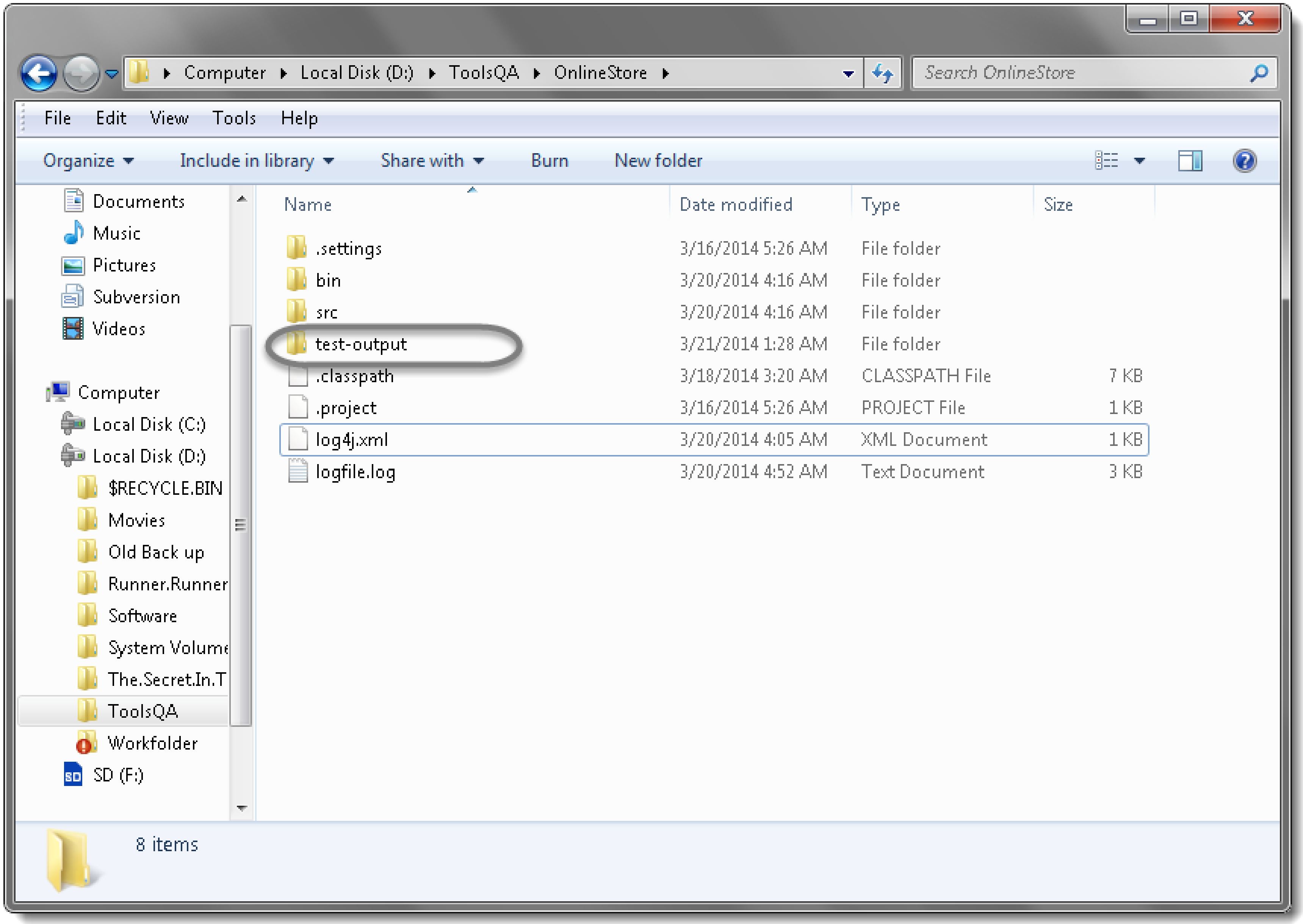
Search:

Passed: 1 Failed: 0 Skipped: 0

```
Default suite ( 1/0/0/0 ) (11.214 s)
  Default test ( 11.214 s)
    automationFramework.TestNG
      main (11.214 s)
```

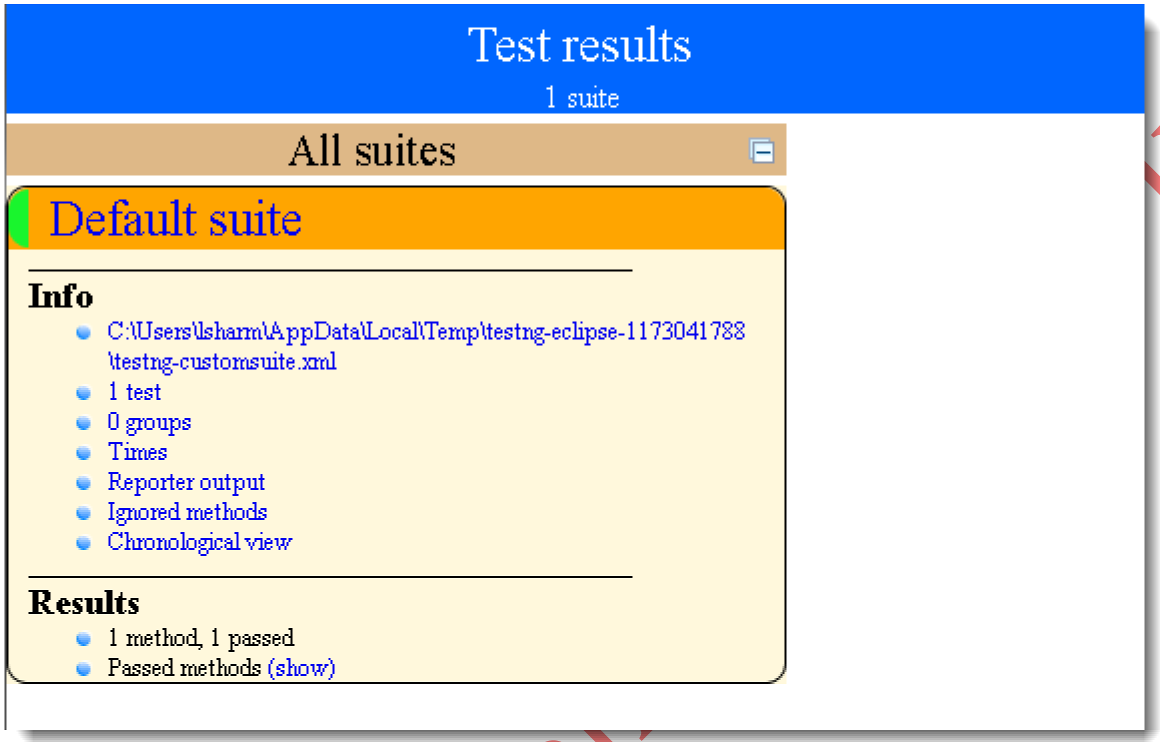
As you see, there is information of which test cases are executed and their duration. Take look to other tabs. Better than Junit right?

8) TestNG also produce HTML reports. To access those reports go to the Project directory and open test-output folder.



Name	Date modified	Type	Size
.settings	3/16/2014 5:26 AM	File folder	
bin	3/20/2014 4:16 AM	File folder	
src	3/20/2014 4:16 AM	File folder	
test-output	3/21/2014 1:28 AM	File folder	
.classpath	3/18/2014 3:20 AM	CLASSPATH File	7 KB
.project	3/16/2014 5:26 AM	PROJECT File	1 KB
log4j.xml	3/20/2014 4:05 AM	XML Document	1 KB
logfile.log	3/20/2014 4:52 AM	Text Document	3 KB

9) TestNG also produce 'index.html' report and it resides in the same test-output folder. This reports gives the link to all the different component of the TestNG reports like Groups & Reporter Output. On clicking these will display detailed descriptions of execution. In the advance chapter of TestNG we will go though each of the TestNG topics.



Test results

1 suite

All suites

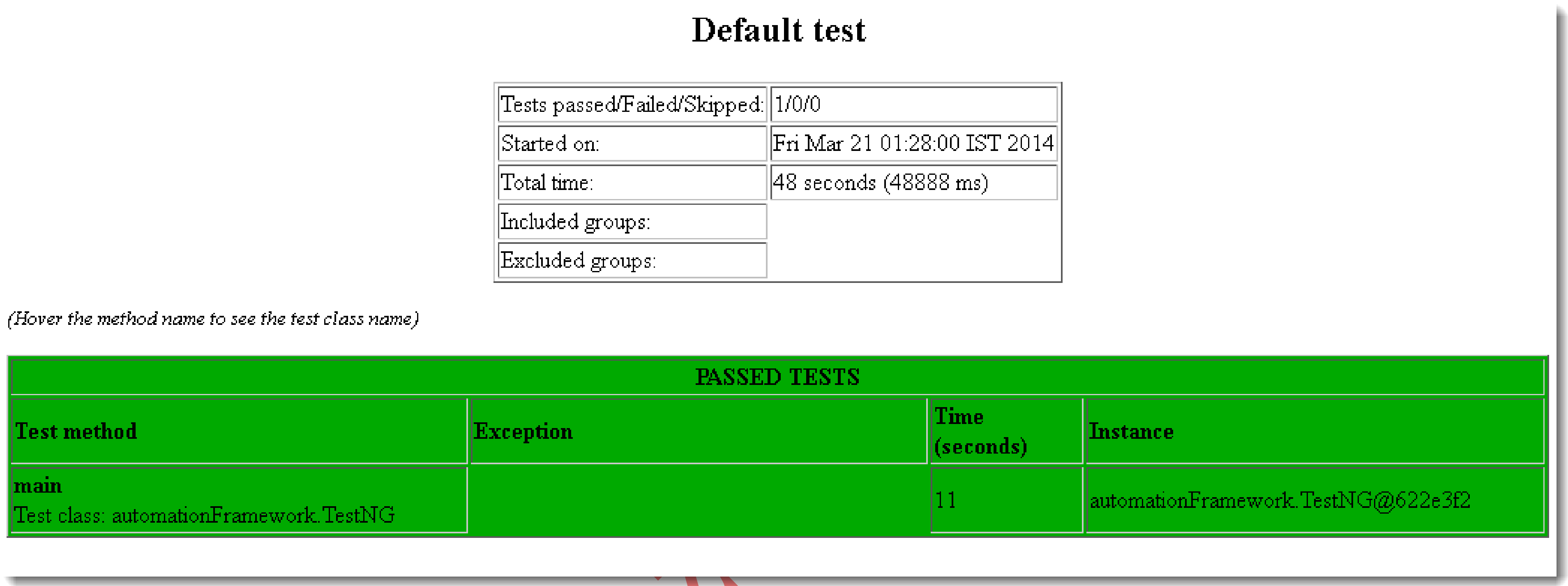
Default suite

Info

- C:\Users\lsharm\AppData\Local\Temp\testng-eclipse-1173041788 \testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results

- 1 method, 1 passed
- Passed methods (show)



Default test

Tests passed/Failed/Skipped:	1/0/0
Started on:	Fri Mar 21 01:28:00 IST 2014
Total time:	48 seconds (48888 ms)
Included groups:	
Excluded groups:	

(Hover the method name to see the test class name)

PASSED TESTS			
Test method	**Exception**	**Time (seconds)**	**Instance**
main Test class: automationFramework.TestNG		11	automationFramework.TestNG@622e3f2

TestNG Framework P4 : Test Suite

TestNG Test Suite

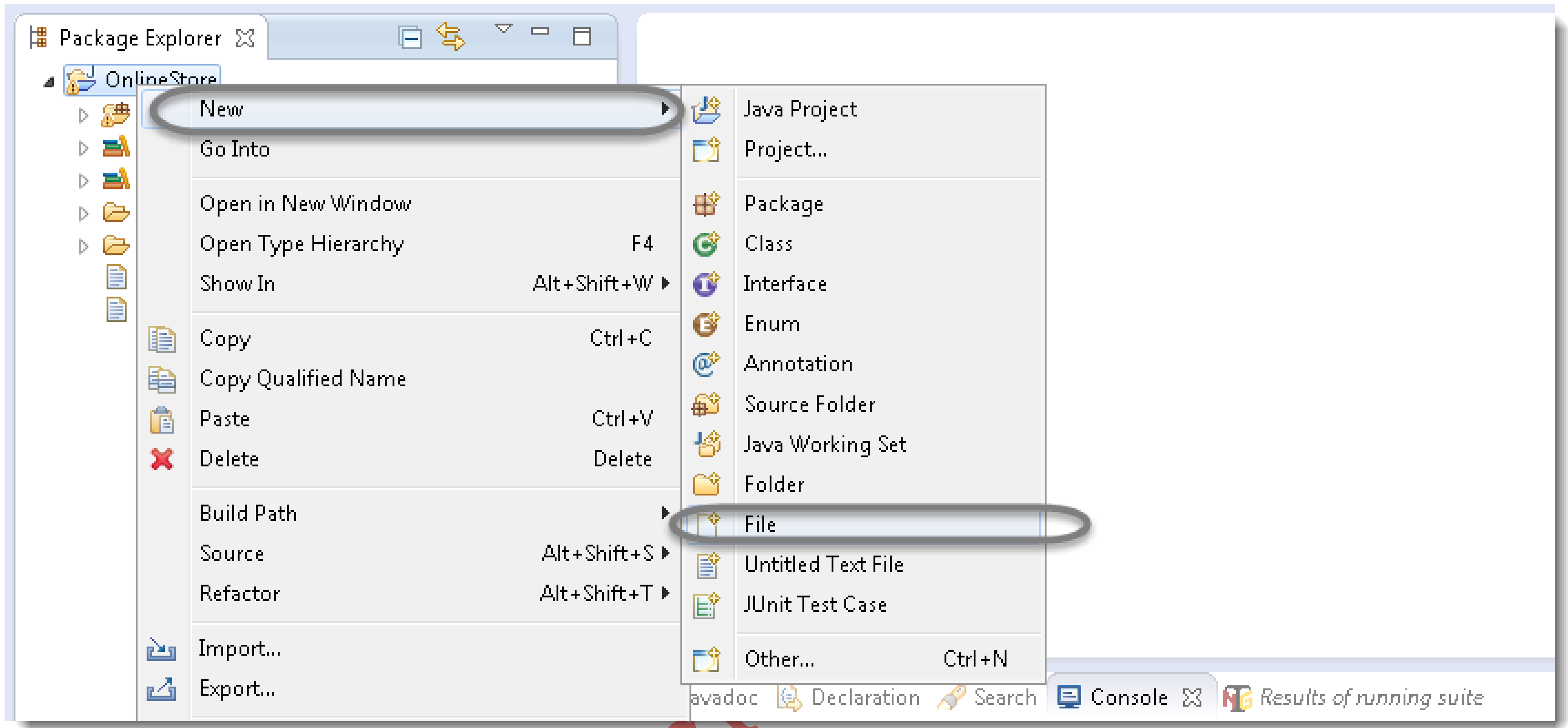
In any project, you will end up to a place where you need to execute so many test cases on a run. Running a set of test cases together is call executing a Test Suite. Those test cases can be dependent to each other or may have to be executed in a certain order. TestNg gives us the capability to manage our test execution.

In TestNG framework, we need to create testng.xml file to create and handle multiple test classes. This is the xml file where you will configure your test run, set test dependency, include or exclude any test, method, class or package and set priority etc.

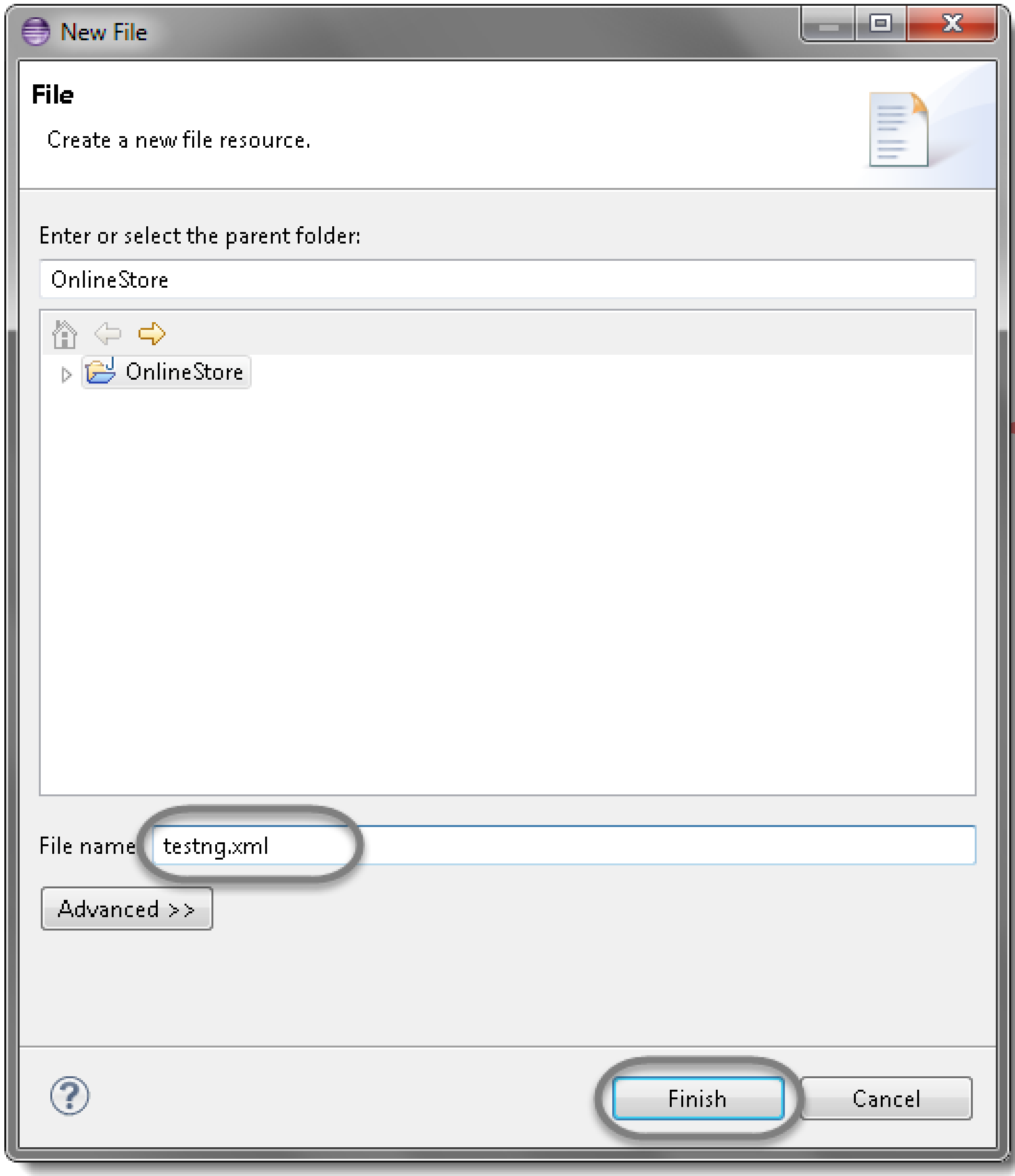
How to do it…

Step 1 : Create a TestNG XML

1) Right click on Project folder, go to New and select 'File' as shown in below image.



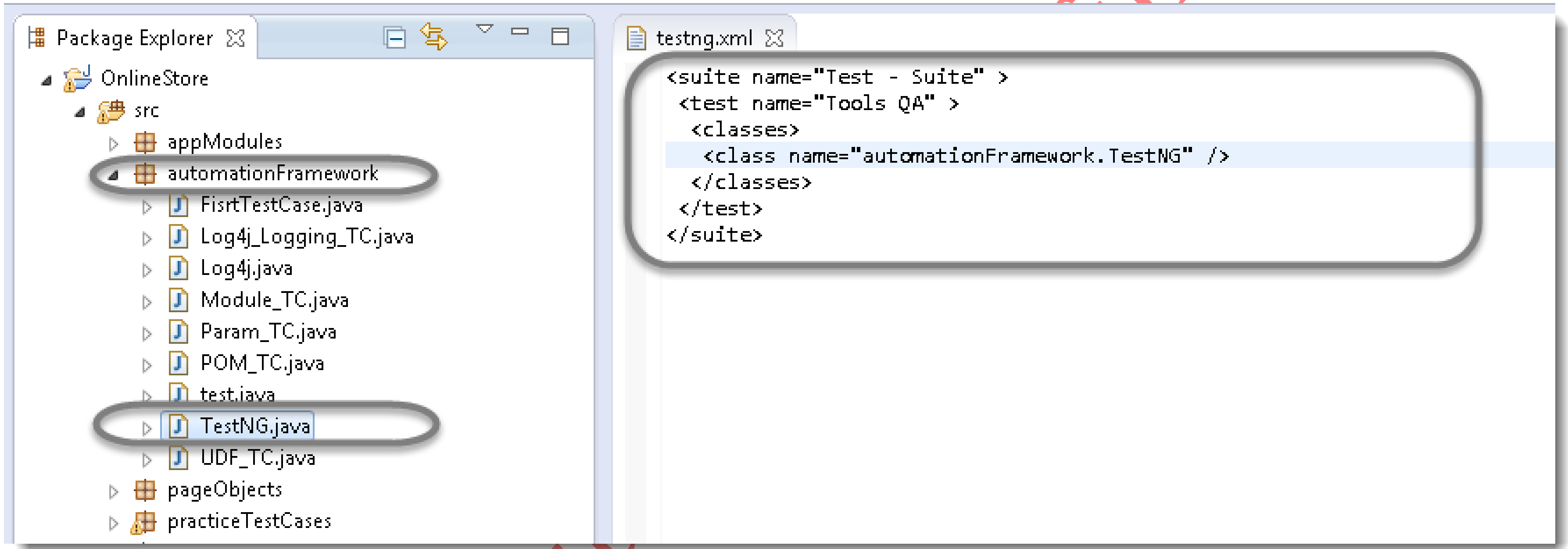
2) In New file wizard, add file name = 'testng.xml' as shown in below given image and click on Finish button.


New File
File
Create a new file resource.
Enter or select the parent folder:
OnlineStore
OnlineStore
File name
testng.xml
Advanced >>
Finish
Cancel

3) It will add testng.xml file under your project folder.

Step 2 : Write xml code ?

Now add below given code in your testng.xml file.see the image below



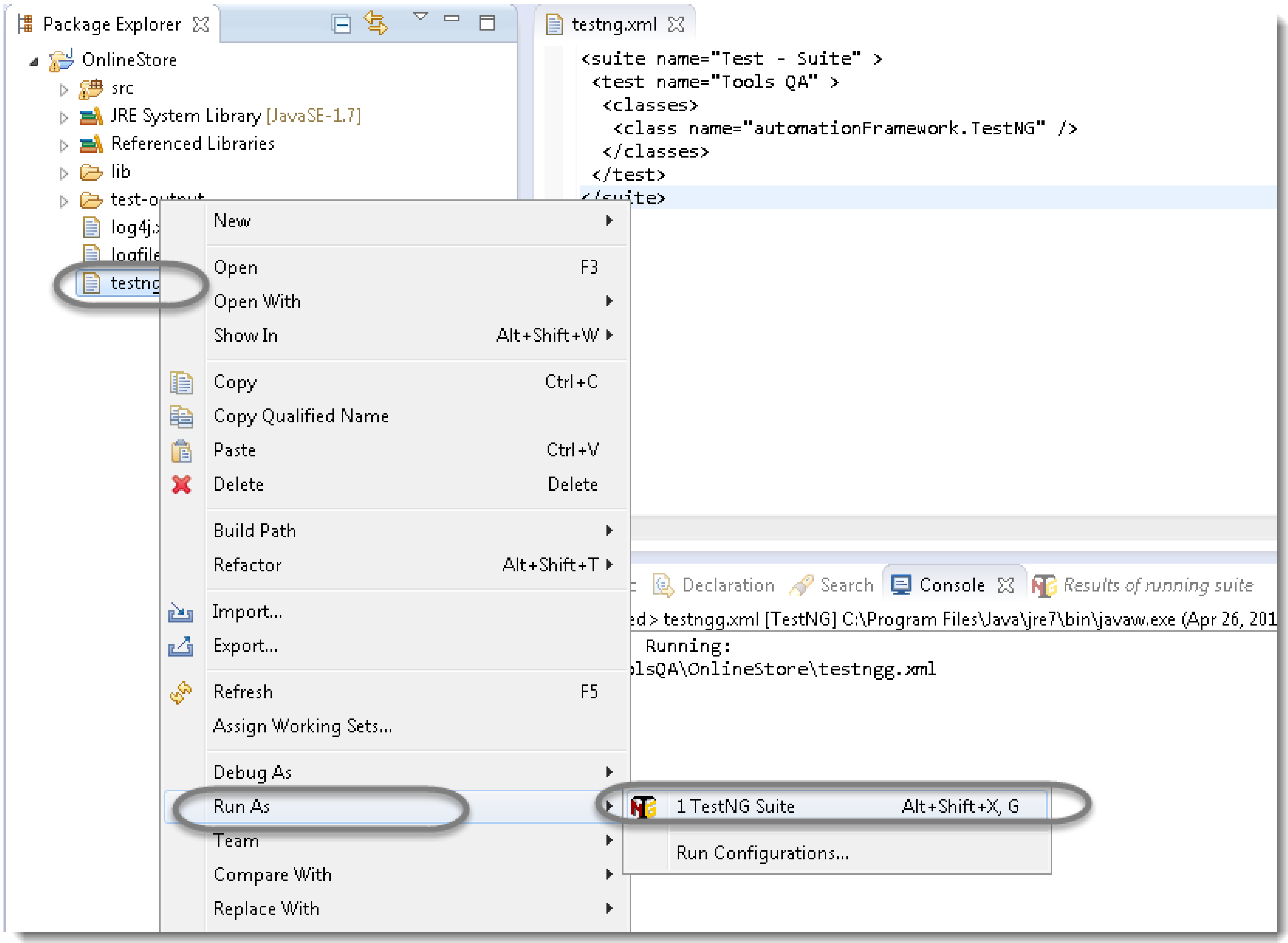
```
<suite name="Test - Suite" >
 <test name="Tools QA" >
  <classes>
   <class name="automationFramework.TestNG" />
  </classes>
 </test>
</suite>
```

Hope you have understood the xml code, as it is quite simple hierarchy:

Very first tag is the Suite tag, under that it is the Test tag and then the Class tag. You can give any name to the suite and the test but you need to provide the correct name to the tag which is a combination of your Package name and Test Case name.

Step 3 : Execute a testng.xml

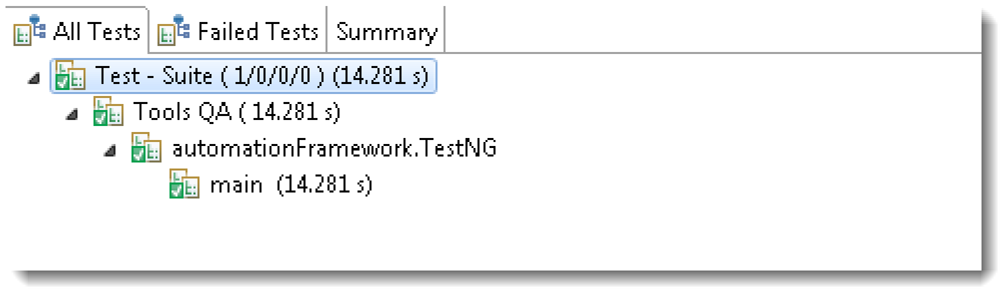
Now it’s time to run the xml. Run the test by right click on the testng.xml file and select Run As > TestNG Suite.



```
<suite name="Test - Suite" >
 <test name="Tools QA" >
  <classes>
   <class name="automationFramework.TestNG" />
  </classes>
 </test>
</suite>
```

```
ed> testngg.xml [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 26, 201
 Running:
olsQA\OnlineStore\testngg.xml
```

It will take few seconds to start the testng execution engine and soon you will notice that your test will run and complete. Once the execution is complete, you can view test execution result under the TestNg console.

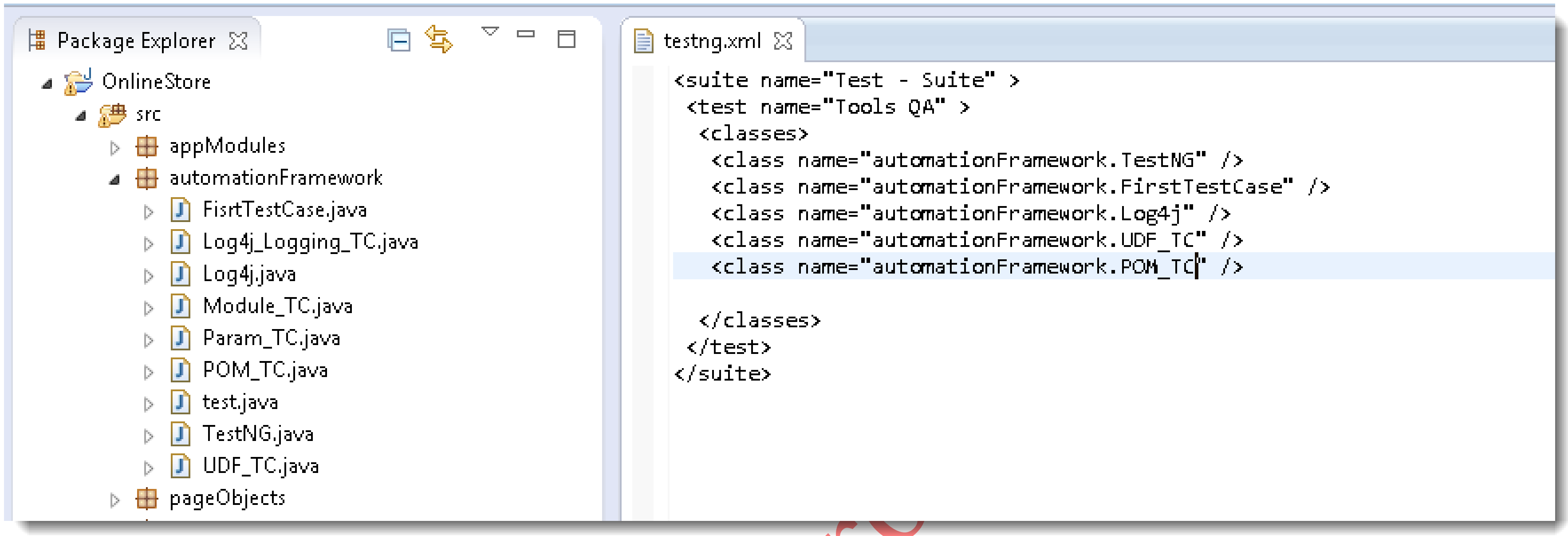


```
Test - Suite ( 1/0/0/0 ) (14.281 s)
  Tools QA ( 14.281 s)
    automationFramework.TestNG
      main  (14.281 s)
```

This is the one simple example of creating and running testng.xml file in eclipse.

Building a Test Suite

Now when you have learned how to build the xml, now it's time to learn how to build a Test Suite using testng.xml. It is again not a complex task, all you need to do is to add your test cases to your xml file in tag.



```
<suite name="Test - Suite" >
 <test name="Tools QA" >
  <classes>
   <class name="automationFramework.TestNG" />
   <class name="automationFramework.FirstTestCase" />
   <class name="automationFramework.Log4j" />
   <class name="automationFramework.UDF_TC" />
   <class name="automationFramework.POM_TC" />

  </classes>
 </test>
</suite>
```

The above test will execute only those tests, which are mentioned in the testng.xml. The rest of the test cases under 'automationFramework' package will remain untouched.

TestNG Framework P5: Annotations , Groups and OnDepends

TestNG Annotations, Groups & OnDepends

TestNG Annotations

In the TestNG Introduction chapter we have came across different annotations used in TestNG Framework but so far we have used just three(Before, After & Test). All though these are the most frequently used annotations but who know how far you will go with your framework and may like to use other useful TestNG annotations.

Before that I would like you to give a small idea on Annotations hierarchy or Annotations levels in TestNG.

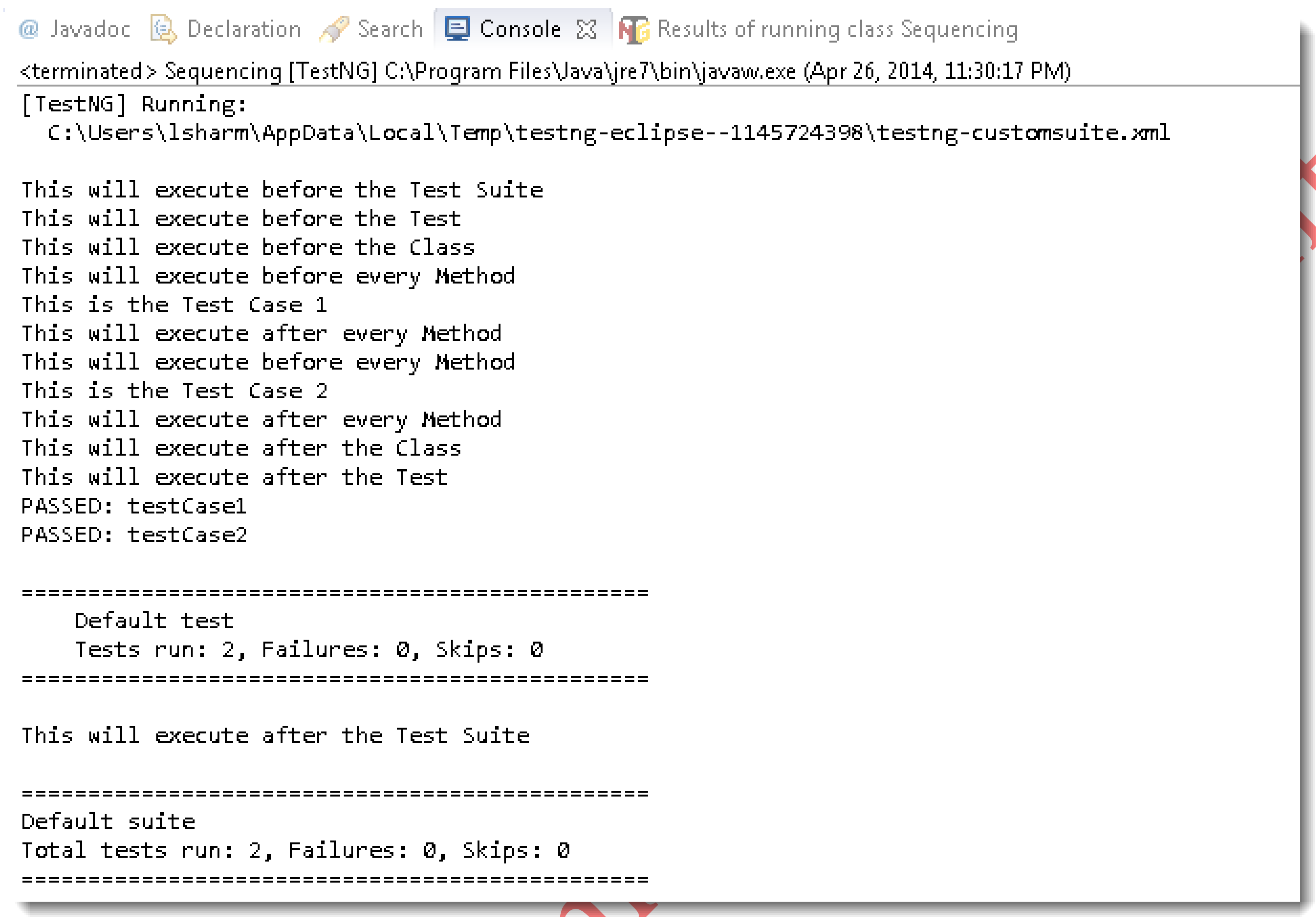
```

<suite>
     <test>
           <classes>
                  <method>
                        <test>
                  </method>
           </classes>
     </test>
</suite>
```

It says that @Test is the smallest annotation here. @Method will be executed first, before and after the execution of @Test. The same way @Class will be executed first, before and after the execution of @Method and so on.

Now with the below example it will be clear to you easily.

```
package automationFramework;

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 Sequencing {

        @Test
        public void testCase1() {
            System.out.println("This is the Test Case 1");
        }

        @Test
        public void testCase2() {
            System.out.println("This is the Test Case 2");
        }

        @BeforeMethod
        public void beforeMethod() {
            System.out.println("This will execute before every Method");
        }

        @AfterMethod
        public void afterMethod() {
            System.out.println("This will execute after every Method");
        }

        @BeforeClass
        public void beforeClass() {
            System.out.println("This will execute before the Class");
        }

        @AfterClass
        public void afterClass() {
            System.out.println("This will execute after the Class");
        }

        @BeforeTest
        public void beforeTest() {
            System.out.println("This will execute before the Test");
        }

        @AfterTest
        public void afterTest() {
            System.out.println("This will execute after the Test");
        }

        @BeforeSuite
        public void beforeSuite() {
            System.out.println("This will execute before the Test Suite");
        }

        @AfterSuite
        public void afterSuite() {
            System.out.println("This will execute after the Test Suite");
        }

    }
```

Output of the above code will be like this:



```
<terminated> Sequencing [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 26, 2014, 11:30:17 PM)
[TestNG] Running:
  C:\Users\lsharm\AppData\Local\Temp\testng-eclipse--1145724398\testng-customsuite.xml

This will execute before the Test Suite
This will execute before the Test
This will execute before the Class
This will execute before every Method
This is the Test Case 1
This will execute after every Method
This will execute before every Method
This is the Test Case 2
This will execute after every Method
This will execute after the Class
This will execute after the Test
PASSED: testCase1
PASSED: testCase2

===============================================
    Default test
    Tests run: 2, Failures: 0, Skips: 0
===============================================

This will execute after the Test Suite

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

It is clearly visible that the @Suite annotation is the very first and the very lastly executed. Then @Test followed by @Class. Now if you notice, the @Method has executed twice. As @Test is a method in the class, hence @Method will always executed for each @Test method.

Test Case Grouping

‘Groups‘ is one more annotation of TestNG which can be used in the execution of multiple tests. Let’s say you have hundred tests of class vehicle and in it ten method of car, ten method of scooter and so on. You probably like to run all the scooter tests together in a batch. And you want all to be in a single test suite. With the help of grouping you can easily overcome this situation.

How to do it…

1) Create two methods for Car, two methods for Scooter and one method in conjunction with Car & Sedan Car.

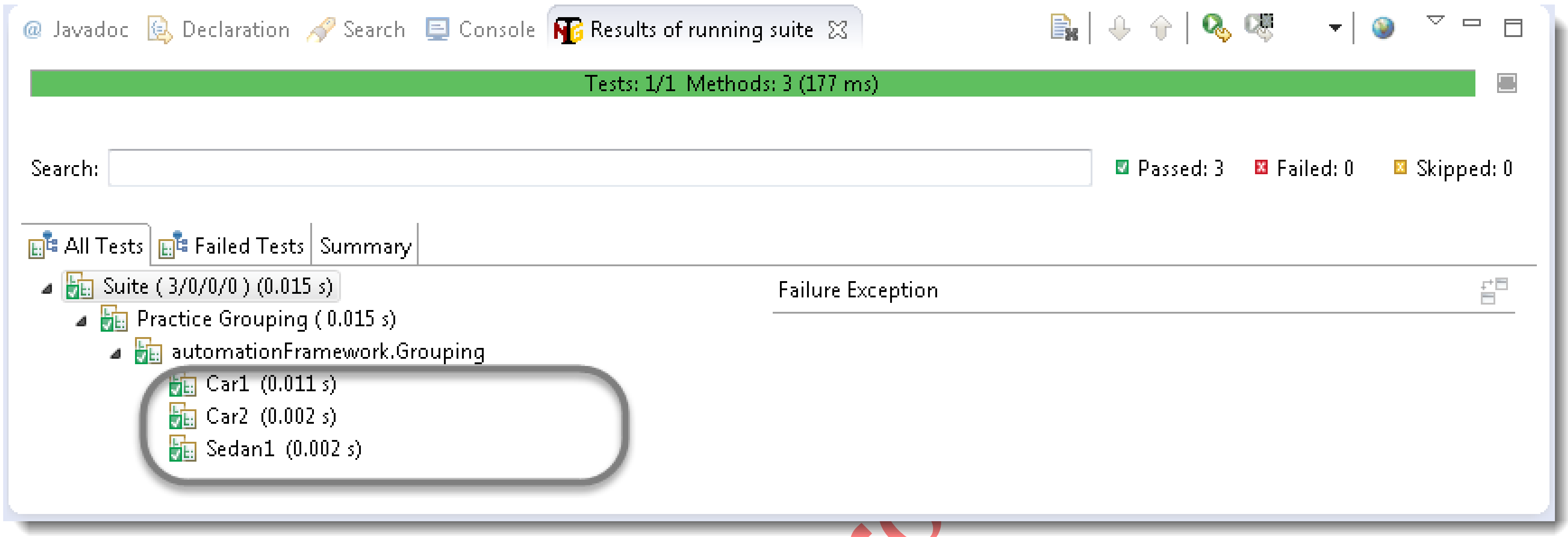
2) Group them separately with using (groups = { ” Group Name” })

```
package automationFramework;

import org.testng.annotations.Test;

public class Grouping {
  @Test (groups = { "Car" })
  public void Car1() {
    System.out.println("Batch Car - Test car 1");
  }
  @Test (groups = { "Car" })
  public void Car2() {
    System.out.println("Batch Car - Test car 2");
  }
  @Test (groups = { "Scooter" })
  public void Scooter1() {
    System.out.println("Batch Scooter - Test scooter 1");
  }
  @Test (groups = { "Scooter" })
  public void Scooter2() {
    System.out.println("Batch Scooter - Test scooter 2");
  }
  @Test (groups = { "Car", "Sedan Car" })
  public void Sedan1() {
    System.out.println("Batch Sedan Car - Test sedan 1");
  }
}
```

3) Create a testng xml like this:

```
  <suite name="Suite">
      <test name="Practice Grouping">
          <groups>
        <run>
        <include name="Car" />
        </run>
    </groups>
    <classes>
        <class name="automationFramework.Grouping" />
    </classes>
      </test>
  </suite>
```

4) Run the test by right click on the testng.xml file and select Run As > TestNG Suite. Output will be like this in TestNg console:



Tests: 1/1 Methods: 3 (177 ms)

Search:

Passed: 3 Failed: 0 Skipped: 0

All Tests | Failed Tests | Summary

```
Suite ( 3/0/0/0 ) (0.015 s)
  Practice Grouping ( 0.015 s)
    automationFramework.Grouping
      Car1  (0.011 s)
      Car2  (0.002 s)
      Sedan1  (0.002 s)
```

Failure Exception

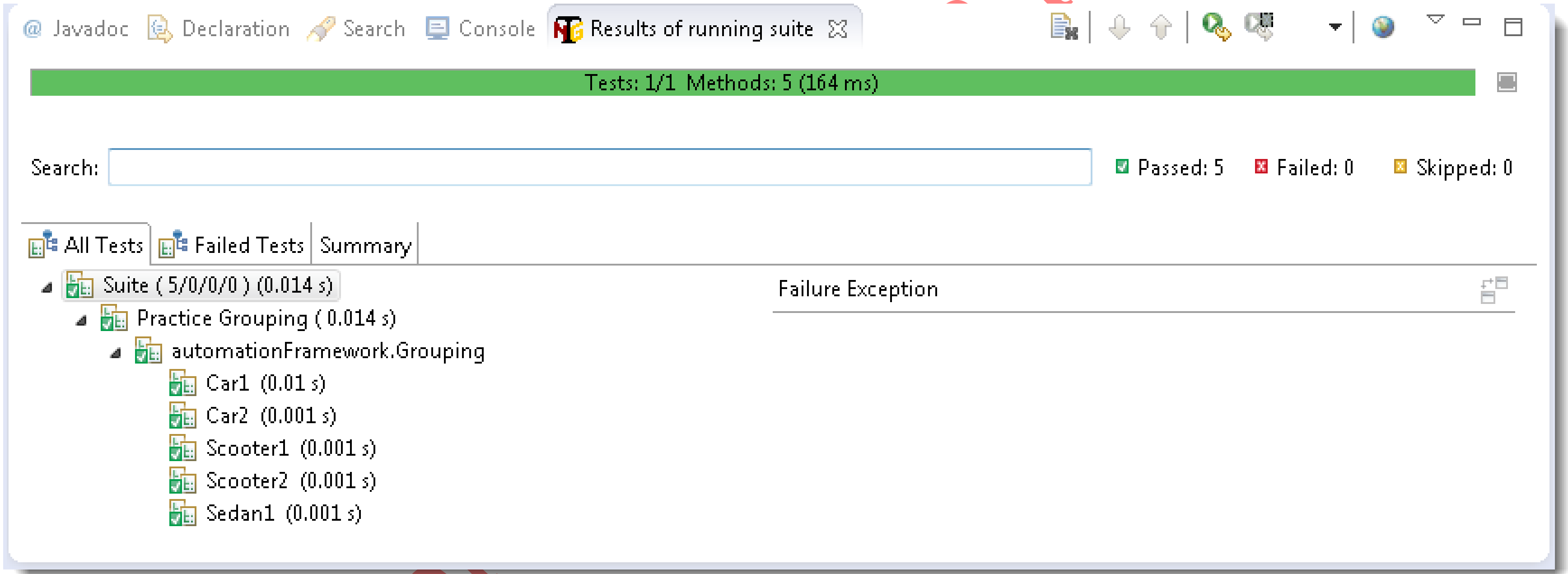
Note: We have just call the group ‘Car’ from the xml and it also executed the test for Sedan Car, as we have mentioned the ‘Car’ as well while declaring the group of Sedan Car.

Clubbing of groups is also possible, take a look at the below xml:

```

<suite name="Suite">
   <test name="Practice Grouping">
      <groups>
         <define name="All">
            <include name="Car"/>
            <include name="Scooter"/>
        </define>
      <run>
            <include name="All"/>
      </run>
```

```
            </groups>
        <classes>
             <class name="automationFramework.Grouping" />
       </classes>
        </test>
  </suite>

```

You can see that we have created a new Group with the name ‘All’ and include other groups in it. Then simply called the newly created group for execution. The output will be like this:



```
Tests: 1/1  Methods: 5 (164 ms)

Search:                     Passed: 5   Failed: 0   Skipped: 0

All Tests | Failed Tests | Summary
Suite ( 5/0/0/0 ) (0.014 s)                 Failure Exception
  Practice Grouping ( 0.014 s)
    automationFramework.Grouping
      Car1  (0.01 s)
      Car2  (0.001 s)
      Scooter1  (0.001 s)
      Scooter2  (0.001 s)
      Sedan1  (0.001 s)
```

Dependent Test

Sometimes, you may need to invoke methods in a Test case in a particular order or you want to share some data and state between methods. This kind of dependency is supported by TestNG as it supports the declaration of explicit dependencies between test methods.

TestNG allows you to specify dependencies either with:

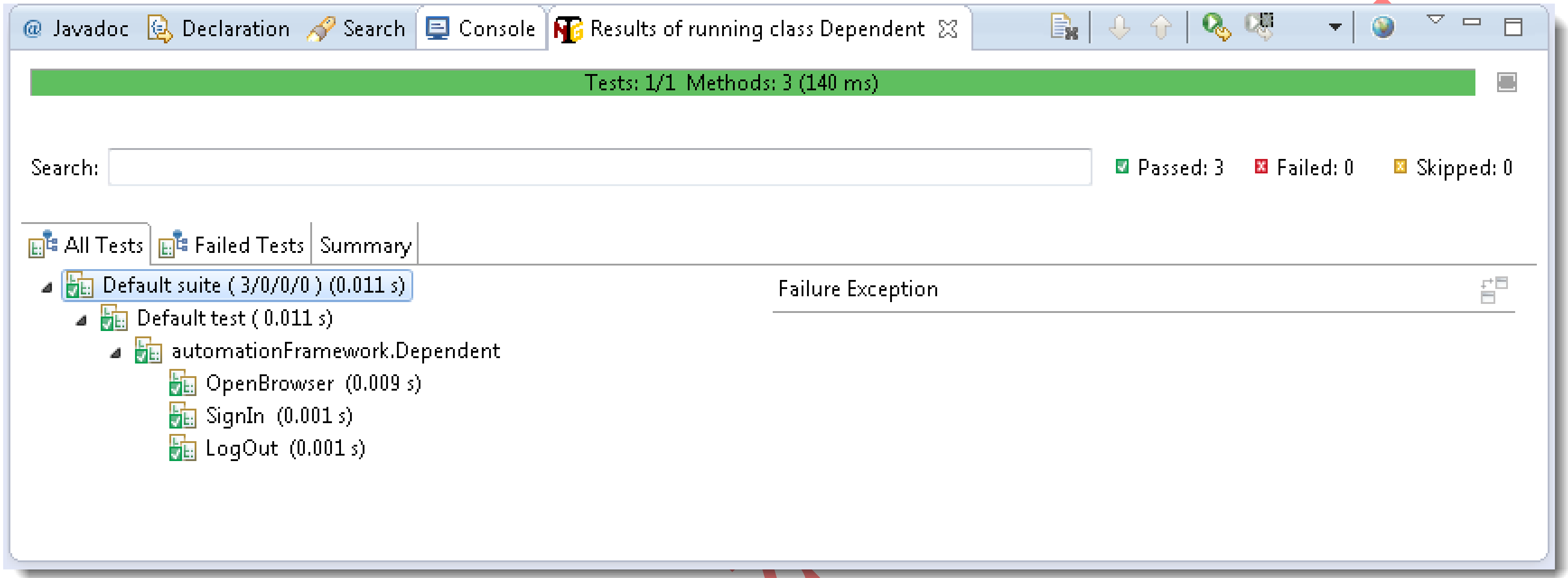
Using attributes dependsOnMethods in @Test annotations OR

Using attributes dependsOnGroups in @Test annotations.

Take a look over the below example:

```

package automationFramework;

import org.testng.annotations.Test;

public class Dependent {
  @Test (dependsOnMethods = { "OpenBrowser" })
  public void SignIn() {
    System.out.println("This will execute second (SignIn)");
  }
  @Test
  public void OpenBrowser() {
    System.out.println("This will execute first (Open Browser)");
  }
  @Test (dependsOnMethods = { "SignIn" })
  public void LogOut() {
    System.out.println("This will execute third (Log Out)");
  }

```

The output will be like this:



```
Tests: 1/1  Methods: 3 (140 ms)

Search:                                   Passed: 3   Failed: 0   Skipped: 0

Default suite ( 3/0/0/0 ) (0.011 s)
  Default test ( 0.011 s)
    automationFramework.Dependent
      OpenBrowser  (0.009 s)
      SignIn  (0.001 s)
      LogOut  (0.001 s)
```

TestNG Framework P6 : Prioritizing and Sequencing

TestNG Prioritizing & Sequencing

Multiple Tests

There will be situations when you want to put number of tests under a single test class and like to run all in single shot. With the help of TestNG '@Test' annotations we can execute multiple tests in single TestNG file.

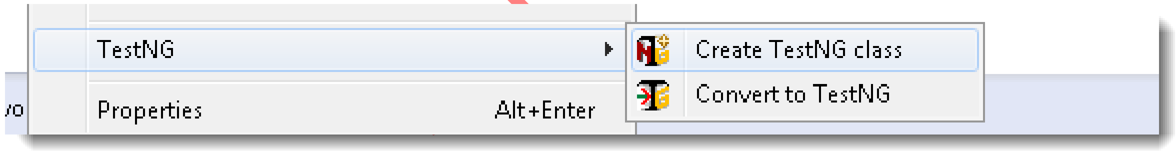
Take an example of four different tests under one testng class and print the test sequence on the console.

How to do it…

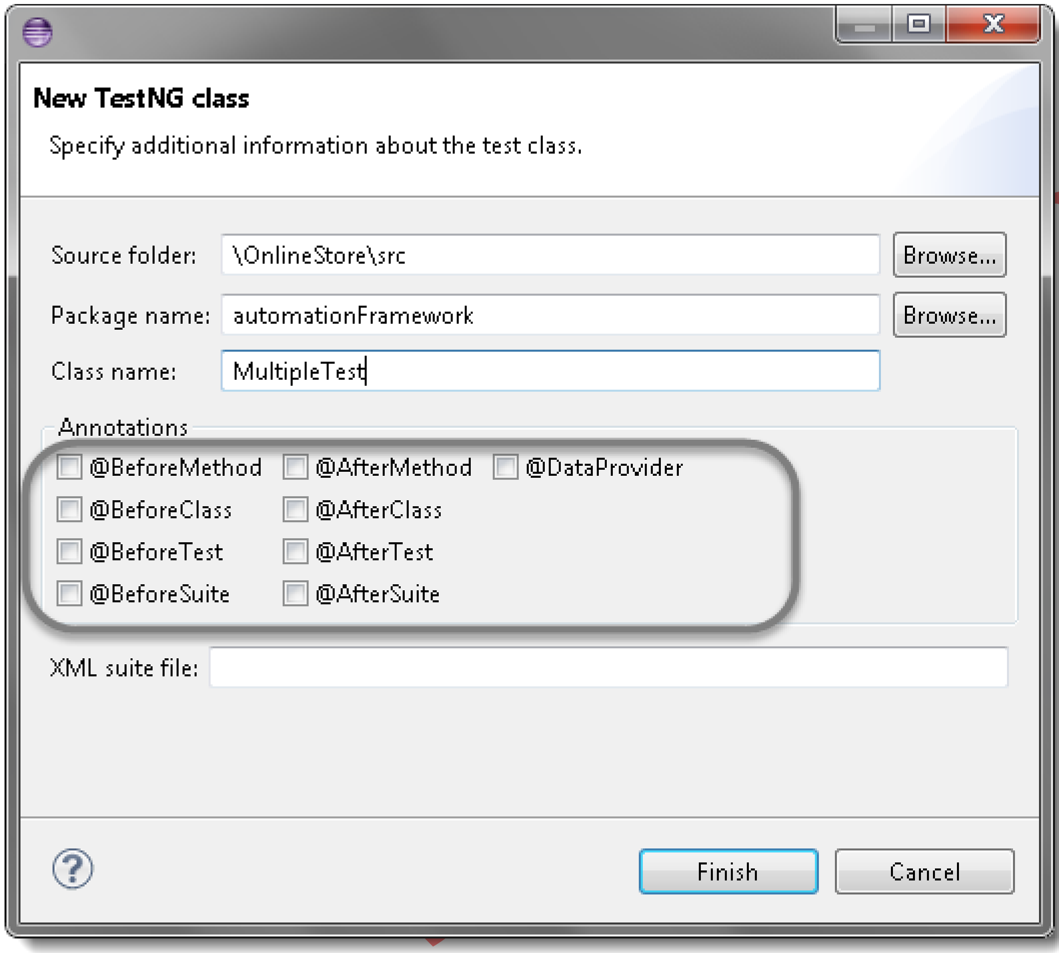
1) Press Ctrl+N , select "TestNG Class" under TestNG category and click Next.

Or

Right click on Test Case folder, go to TestNG and select "TestNG Class".



2) If your project is set up and you have selected the Test Case folder before creating TestNG class then the source folder and the package name will be pre-populated on the form. Set class name as 'TestNG'. Leave rest of the settings untouched, do not check for "@BeforeMethod", "@AfterMethod" for now and click Finish. That's it.


New TestNG class

Specify additional information about the test class.

Source folder: \OnlineStore\src

Package name: automationFramework

Class name: MultipleTest

Annotations

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

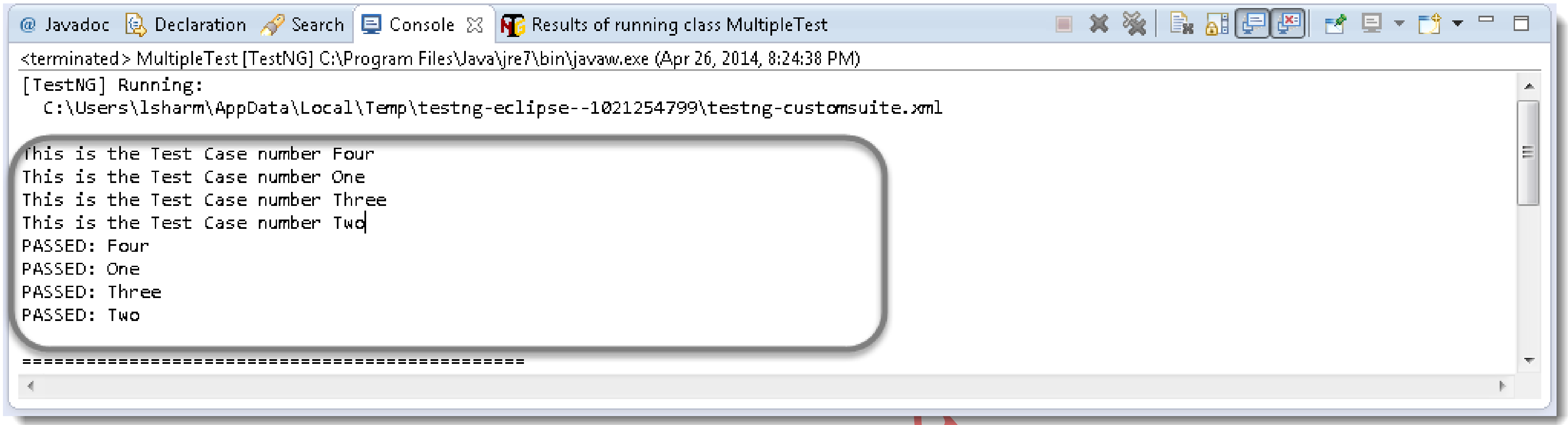
XML suite file:

3) By default a new class will have only one @Test method.

Add two more methods by yourself and put your code accordingly in methods. Code will look like:

```
package automationFramework;

import org.openqa.selenium.WebDriver;
import org.testng.annotations.Test;

public class MultipleTest {
  public WebDriver driver;
  @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");
  }
}
```

This will enable you to execute all four tests with just one testng class. Take a look on the output.



```
<terminated> MultipleTest [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 26, 2014, 8:24:38 PM)
[TestNG] Running:
  C:\Users\lsharm\AppData\Local\Temp\testng-eclipse--1021254799\testng-customsuite.xml

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

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

Attention: By default, methods annotated by @Test are executed alphabetically. Take a look over the next topic to see how to prioritize @Test.

Sequencing & Prioritizing

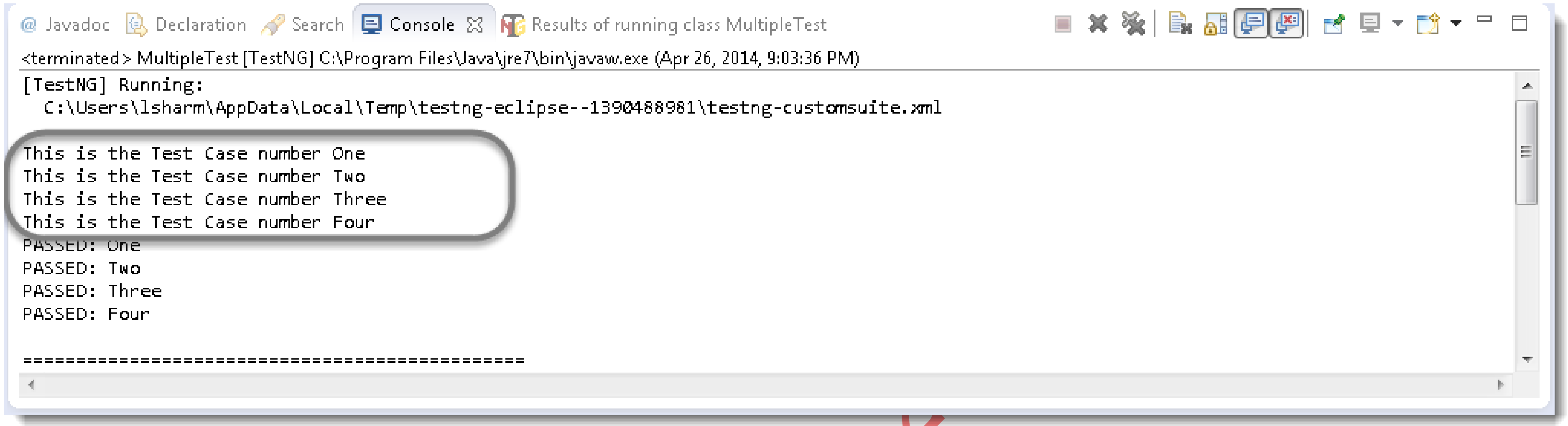
You need to use the ‘priority‘ parameter, if you want the methods to be executed in your order. Parameters are keywords that modify the annotation’s function.

Let’s take the same above example and execute all @Test methods in right order. Simply assign priority to all @Test methods starting from 0(Zero).

```


package automationFramework;

import org.openqa.selenium.WebDriver;
import org.testng.annotations.Test;

public class MultipleTest {
  public WebDriver driver;
  @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");
  }
}
```

Note: TestNG will execute the @Test annotation with the lowest priority value up to the largest.

Output of the above:



```
<terminated> MultipleTest [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 26, 2014, 9:03:36 PM)
[TestNG] Running:
  C:\Users\lsharm\AppData\Local\Temp\testng-eclipse--1390488981\testng-customsuite.xml

This is the Test Case number One
This is the Test Case number Two
This is the Test Case number Three
This is the Test Case number Four
PASSED: One
PASSED: Two
PASSED: Three
PASSED: Four

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

Skipping a Test Case

Think of a situation where you are required to skip one or more @Test from your testng class. In testng, you can easily able to handle this situation by setting the ‘enabled’ parameter to ‘false’ for e.g.:

@Test(enabled = false)

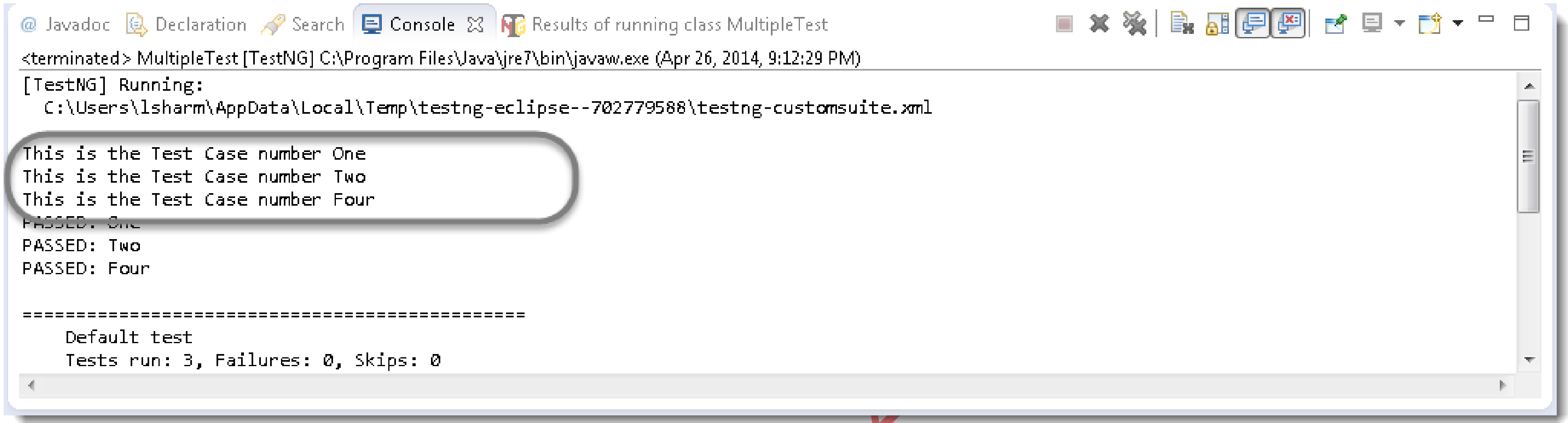
To use two or more parameters in a single annotation, separate them with a comma:

@Test(priority = 3, enabled = false)

Again take the same example and set the value false for the third test.

```

package automationFramework;

import org.openqa.selenium.WebDriver;
import org.testng.annotations.Test;

public class MultipleTest {
  public WebDriver driver;
  @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, enabled = false)
  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");
  }
}

```

Output of the above example:



```
<terminated> MultipleTest [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe (Apr 26, 2014, 9:12:29 PM)
[TestNG] Running:
  C:\Users\lsharm\AppData\Local\Temp\testng-eclipse--702779588\testng-customsuite.xml

This is the Test Case number One
This is the Test Case number Two
This is the Test Case number Four
PASSED: One
PASSED: Two
PASSED: Four

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
```

TestNG Framework P7 : Reporters and Asserts

TestNG Reporters & Asserts

TestNG Reporters

TestNG is a Framework and so far we have already seen the many different powerful features of TestNG. It almost gives you all the important things you are required to complete the Framework.

TestNG Reporter Logs

TestNG also gives us the logging facility for the test. For example during the running of test case user wants some information to be logged in the console. Information could be any detail depends upon the purpose. Keeping this in mind that we are using Selenium for testing, we need the information which helps the User to understand the test steps or any failure during the test case execution. With the help of TestNG Logs it is possible to enable logging during the Selenium test case execution.

In selenium there are two types of logging. High level logging and Low level logging. In low level logging you try to produce logs for the every step you take or every action you make in your automation script. In high level logging you just try to capture main events of your test.

Everybody has their own style of logging and I have mine too. I am also a big fan of Log4j logging and that's why I do not mix log4j logging with testng logging but on the same side I make to use of both of its. I perform low level logging with log4j and high level logging with testng reporter logs.

How to do it…

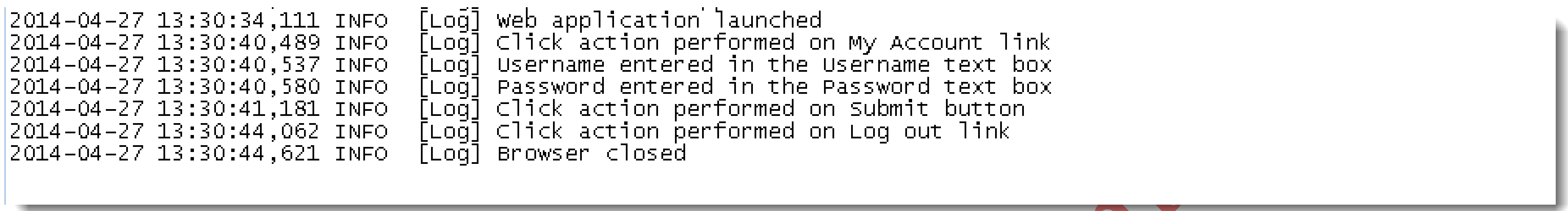
1) Write a test case for Sign In application and implement Log4j logging on every step.

2) Insert Reporter logs on the main events of the test.

```
package automationFramework;

import java.util.concurrent.TimeUnit;

import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Reporter;
import org.testng.annotations.Test;

import utility.Log;

public class ReporterLogs {
  private static WebDriver driver;
  private static Logger Log = Logger.getLogger(Log.class.getName());
    @Test
  public static void test() {
      DOMConfigurator.configure("log4j.xml");
        driver = new FirefoxDriver();
        Log.info("New driver instantiated");
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        Log.info("Implicit wait applied on the driver for 10 seconds");
        driver.get("https://www.facebook.com/");
        Log.info("Web application launched");
        // Our first step is complete, so we produce a main event log here for our reports.
        Reporter.log("Application Lauched successfully | ");

        driver.findElement(By.id("email")).sendKeys("yourusername");
        Log.info("Username entered in the Username and email text box");
        driver.findElement(By.id("pass")).sendKeys("yourpassword");
        Log.info("Password entered in the Password text box");
        driver.findElement(By.id("loginbutton")).click();
        Log.info("Click action performed on Submit button");
        // Here we are done with our Second main event
        Reporter.log("Sign In Successful | " );
        driver.quit();
        Log.info("Browser closed");
        // This is the third main event
        Reporter.log("User is Logged out and Application is closed | ");
  }

}
```

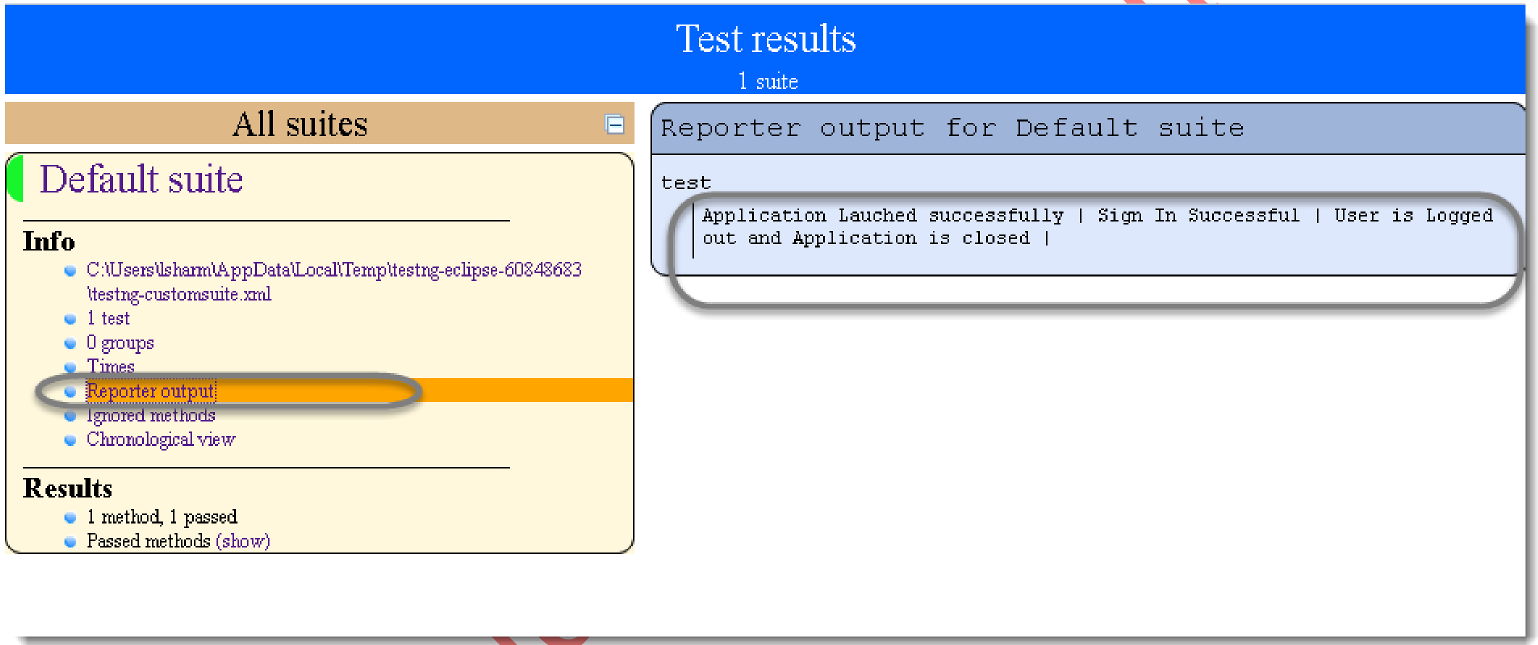
3) Run the test by right click on the test case script and select Run As > TestNG Test.

Your Log4j logging output will look like this:



```
2014-04-27 13:30:34,111 INFO  [Log] web application launched
2014-04-27 13:30:40,489 INFO  [Log] Click action performed on My Account link
2014-04-27 13:30:40,537 INFO  [Log] Username entered in the Username text box
2014-04-27 13:30:40,580 INFO  [Log] Password entered in the Password text box
2014-04-27 13:30:41,181 INFO  [Log] Click action performed on Submit button
2014-04-27 13:30:44,062 INFO  [Log] Click action performed on Log out link
2014-04-27 13:30:44,621 INFO  [Log] Browser closed
```

But your Reporters log will look like this:



Test results
1 suite

All suites

Default suite

Info
- C:\Users\lsharm\AppData\Local\Temp\testng-eclipse-60848683\testng-customsuite.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results
- 1 method, 1 passed
- Passed methods (show)

Reporter output for Default suite

```
test
Application Lauched successfully | Sign In Successful | User is Logged
out and Application is closed |
```

Log4j logging will help you to report a bug or steps taken during the test, on the other hand reporters log will help you to share the test status with leadership. As leadership is just interested in the test results, not the test steps.

I also use reporter's logs on the verification during the test. For example

```
if(Text1.equals(Text2)){
       Reporter.log("Verification Passed forText");
  }else{
       Reporter.log("Verification Failed for Text");
  }
```

TestNG Asserts

TestNG also gives us the power to take decisions in the middle of the test run with the help of Asserts. With this we can put various checkpoints in the test. Asserts are the most popular and frequently used methods while creating Selenium Scripts. In selenium there will be many situations in the test where you just like to check the presence of an element. All you need to do is to put an assert statement on to it to verify its existence.

Different Asserts Statements

1) Assert.assertTrue() & Assert.assertFalse()

```
package automationFramework;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class Asserts {
  private static WebDriver driver;
  @Test
  public void f() {
    driver = new FirefoxDriver();
      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
      driver.get("https://www.facebook.com/");

      // Here driver will try to find out My username textbox on the application
      WebElement text_username = driver.findElement(By.id("email"));

      //Test will only continue, if the below statement is true
      //This is to check whether the textbox is displayed or not
      Assert.assertTrue(text_username.isDisplayed());

      //My username text will be type only if the above condition is true
      myAccount.sendKeys("your_username");
  }
}



```

Note: Assert true statement fails the test and stop the execution of the test, if the actual output is false. Assert.assertFalse() works opposite of Assert.assertTrue(). It means that if you want your test to continue only if when some certain element is not present on the page. You will use Assert false, so it will fail the test in case of the element present on the page.

2) Assert.assertEquals()

```
@Test
  public void test() {
    String sValue = "Ta bao Lan";
    System.out.println(" What is your full name");
    Assert.assertEquals("Ta Bao Lan", sValue);
    System.out.println(sValue);
  }
```

It also works the same way like assert true and assert fail. It will also stop the execution, if the value is not equal and carry on the execution, if the value is equal.

TestNG Framework P8 : Parameters and Data Provider

TestNG Parameters & Data Provider

TestNG Parameters

Everybody knows the importance of Parametrization in testing and in automation testing. It allows us to automatically run a test case multiple times with different input and validation values. As Selenium Webdriver is more an automated testing framework than a ready-to-use tool, you will have to put in some effort to support data driven testing in your automated tests. I usually prefer to use Microsoft Excel as the format for storing my parameters but so many of my followers have requested to write an article on TestNG Data Provider.

TestNG again gives us another interesting feature called TestNG Parameters. TestNG lets you pass parameters directly to your test methods with your testng.xml.

How to do it…

Let me take a very simple example of LogIn application, where the username and password is required to clear the authentication.

1) Create a test on my demo OnlineStore application to perform LogIn which takes the two string argument as username & password.

2) Provide Username & Password as parameter using TestNG Annotation.

```
package automationFramework;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;

import org.testng.annotations.Parameters;

public class TestngParameters {
  private static WebDriver driver;
  @Test
  @Parameters({ "sUsername", "sPassword" })
  public void test(String sUsername, String sPassword) {
    driver = new FirefoxDriver();
      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
      driver.get("https://www.facebook.com");
      driver.findElement(By.id("email")).sendKeys(sUsername);
      driver.findElement(By.id("pass")).sendKeys(sPassword);
      driver.findElement(By.id("loginbutton")).click();
      driver.quit();
  }
}
```

3) The parameter would be passed values from testng.xml which we will see in the next step.

```
<suite name="Suite">
    <test name="ToolsQA">
  <parameter name="sUsername" value="yourusername"/>
  <parameter name="sPassword" value="yourpassword"/>
        <classes>
            <class name="automationFramework.TestngParameters" />
        </classes>
    </test>
</suite>
```

Now, run the testng.xml, which will run the parameterTest method. TestNG will try to find a parameter named sUsername & sPassword.

TestNG DataProviders

When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a database, etc…), in such cases parameters can be passed using Dataproviders. A Data Provider is a method annotated with @DataProvider. A Data Provider returns an array of objects.

Let us check out the same Sign In examples using Dataproviders.

How to do it…

1) Define the method credentials() which is defined as a Dataprovider using the annotation. This method returns array of object array.

2) Add a method test() to your DataProviderTest class. This method takes two strings as input parameters.

3) Add the annotation @Test(dataProvider = “Authentication”) to this method. The attribute dataProvider is mapped to “Authentication”.

Test will look like this:

```
package automationFramework;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class DataProviderTest {
  private static WebDriver driver;

  @DataProvider(name = "Authentication")
  public static Object[][] credentials() {
        return new Object[][] { { "yourusername", "yourpassword" }, { "yourusername_another",
"yourpassword_another" }};
  }

  // Here we are calling the Data Provider object with its Name
  @Test(dataProvider = "Authentication")
  public void test(String sUsername, String sPassword) {
     driver = new FirefoxDriver();
      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

```
        driver.get("https://www.facebook.com");
        driver.findElement(By.id("email")).sendKeys(sUsername);
        driver.findElement(By.id("pass")).sendKeys(sPassword);
        driver.findElement(By.id("loginbutton")).click();
        driver.quit();
    }
}
```

Run the test by right click on the test case script and select Run As > TestNG Test. Give it few minutes to complete the execution, once it is finished the results will look like this in the TestNg Result window.

Note: As the test data is provided two times, the above test executed two times completely.

TestNG Framework P9 : Multi Browser and Parallel Testing

Multi Browser, Cross Browser & Parallel Testing using TestNG

When the time comes to turn your site from mock-up to something fully functional, you'll want to make sure that it works great for everyone visiting your site whether they're using Internet Explorer, Firefox, or any other browser. Testing your website with multiple combinations of browsers is known as Cross Browser testing.

Your site will look different in different browsers. That’s because browsers understand some code slightly differently. Your designer should be testing to make sure that your site works well in all modern browsers. But as a tester we need to make sure that functionality should at least tested on Internet Explorer, Firefox, Safari & Google Chrome browser.

Multi Browser Testing using Selenium TestNG

In every project it is required to perform multi-browser testing to make sure that the functionality is working as expected with every browser to give equal user experience to all of the wide range of audience. It takes a considerable time to test everything on every browser and when we have used automation to reduce the testing efforts then why don’t we perform the multi-browser testing using automation. TestNG gives us functionality to perform same test on different browsers in a simple and easy way.

How to do it…

1)Create your Script to test a LogIn application using TestNG class.

2) Pass ‘Browser Type’ as parameters using TestNG annotations to the before method of the TestNG class. This method will launch only the browser, which will be provided as parameter.

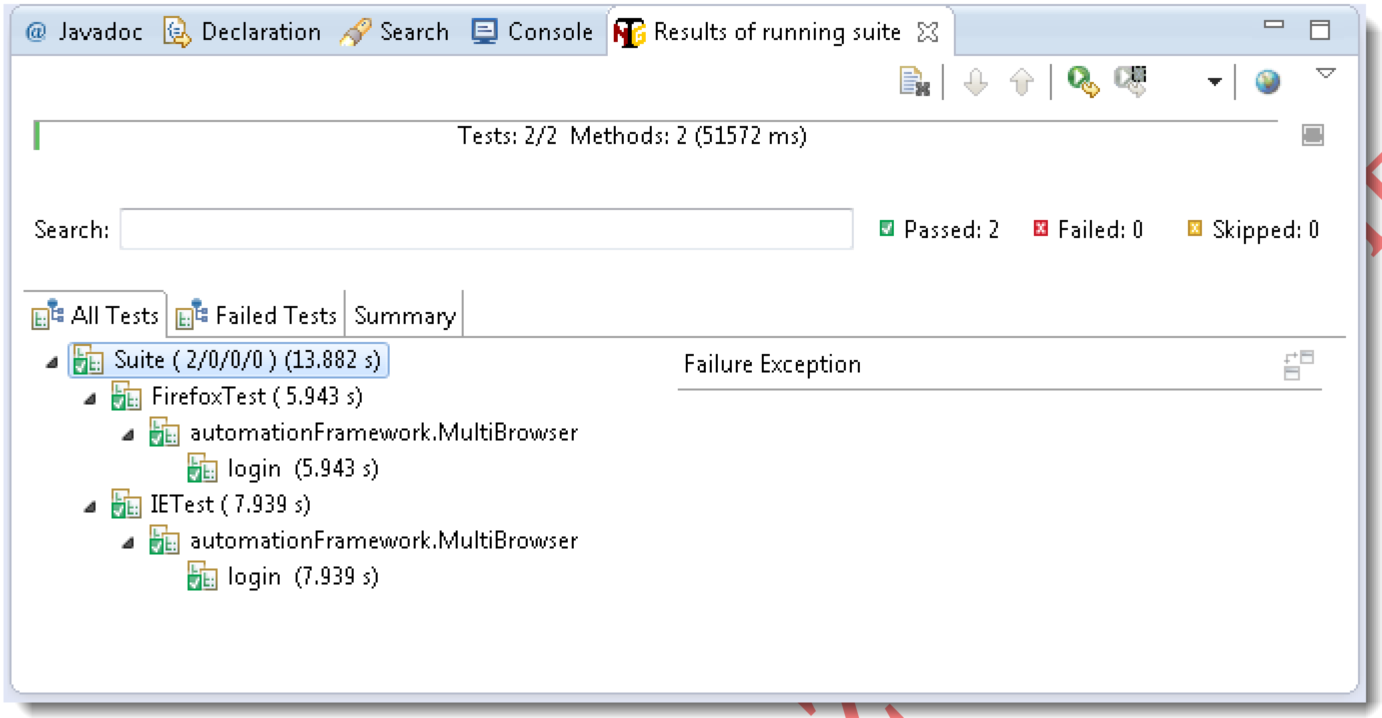
```
package automationFramework;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class MultiBrowser {
  public WebDriver driver;

  @Parameters("browser")
  @BeforeClass
  // Passing Browser parameter from TestNG xml
  public void beforeTest(String browser) {
  // If the browser is Firefox, then do this
  if(browser.equalsIgnoreCase("firefox")) {
    driver = new FirefoxDriver();
  // If browser is IE, then do this
  }else if (browser.equalsIgnoreCase("ie")) {
    // Here I am setting up the path for my IEDriver
    System.setProperty("webdriver.ie.driver", "D:\OnlineStore\drivers\IEDriverServer.exe");
    driver = new InternetExplorerDriver();
  }
  // Doesn't the browser type, lauch the Website
   driver.get("https://www.facebook.com");
  }

  // Once Before method is completed, Test method will start
  @Test
  public void login() throws InterruptedException {
      driver.findElement(By.id("email")).sendKeys("your username");
      driver.findElement(By.id("pass")).sendKeys("your password");
      driver.findElement(By.id("loginbutton")).click();
      driver.quit();
  }

  @AfterClass public void afterTest() {
      driver.quit();
  }
}
```

3) Create a TestNG XML for running your test. Configure the TestNG XML for passing parameters i.e. to tell which browser should be used for Running the Test.

```

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite" parallel="none">
 <test name="FirefoxTest">
 <parameter name="browser" value="firefox" />
 <classes>
 <class name="automationFramework.MultiBrowser" />
 </classes>
 </test>

 <test name="IETest">
 <parameter name="browser" value="ie" />
 <classes>
 <class name="automationFramework.MultiBrowser" />
 </classes>
 </test>
</suite>

```

Note: You can set any number of Browsers here and just for the example purpose I have set up only two main browsers.

4) Now it's time to run the xml. Run the test by right click on the testng.xml file and select Run As > TestNG Suite.



Tests: 2/2 Methods: 2 (51572 ms)

Search:

Passed: 2 Failed: 0 Skipped: 0

```
Suite ( 2/0/0/0 ) (13.882 s)
  FirefoxTest ( 5.943 s)
    automationFramework.MultiBrowser
      login (5.943 s)
  IETest ( 7.939 s)
    automationFramework.MultiBrowser
      login (7.939 s)
```

Note: TestNg will execute the test one by one. You may like to perform parallel tests, next topic will cover that.

Parallel Tests using TestNG

Using the feature provided by TestNG for Parallel Executions. just take the above example for Sign In application with two different browsers. This time all we want is to execute test in both browsers simultaneously.

Now just set the ‘parallel‘ attribute to ‘tests‘ in the above used xml and give a run again. This time you will notice that your both browsers will open almost simultaneously and your test will run in parallel.

```

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite" parallel="tests">
 <test name="FirefoxTest">
 <parameter name="browser" value="firefox" />
 <classes>
 <class name="automationFramework.MultiBrowser" />
 </classes>
 </test>

 <test name="IETest">
 <parameter name="browser" value="ie" />
 <classes>
 <class name="automationFramework.MultiBrowser" />
 </classes>
 </test>
</suite>
```

Note: You may see some intermittent issues using parallel testing. I will not recommend you this rather run one by one only.