TA Lecture-6

TestNG.xml
and
TestNG Reports

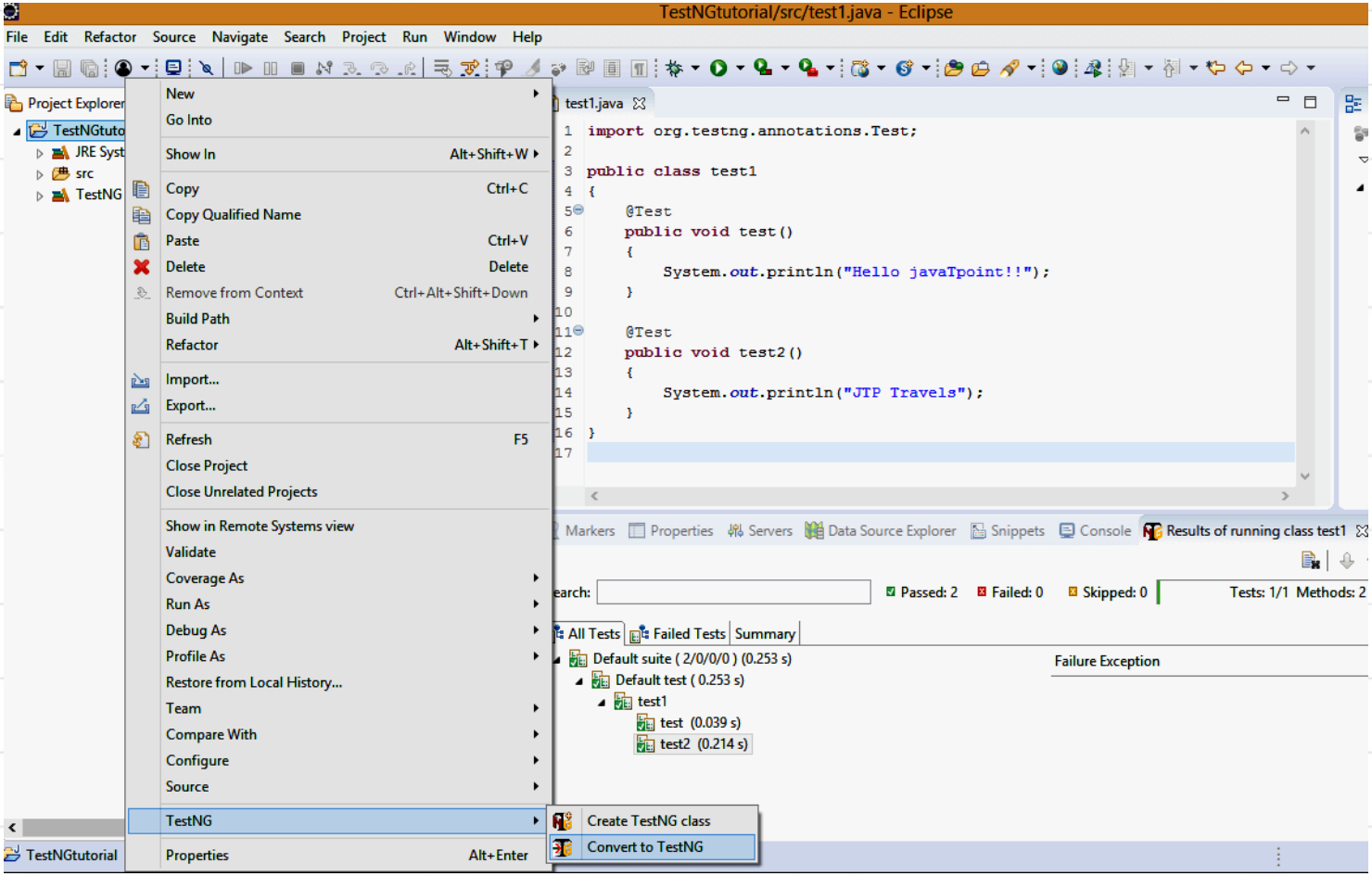
TestNG.xml

TestNG.xml

- **TestNG.xml** file is a configuration file that helps in organizing our tests.
- It allows testers to create and handle multiple test classes, define test suites and tests.
- Major **advantages** of TestNG.xml file are:
  - It provides parallel execution of test methods.
  - It allows the dependency of one test method on another test method.
  - It helps in prioritizing our test methods.
  - It allows grouping of test methods into test groups.
  - It supports the parameterization of test cases using @Parameters annotation.

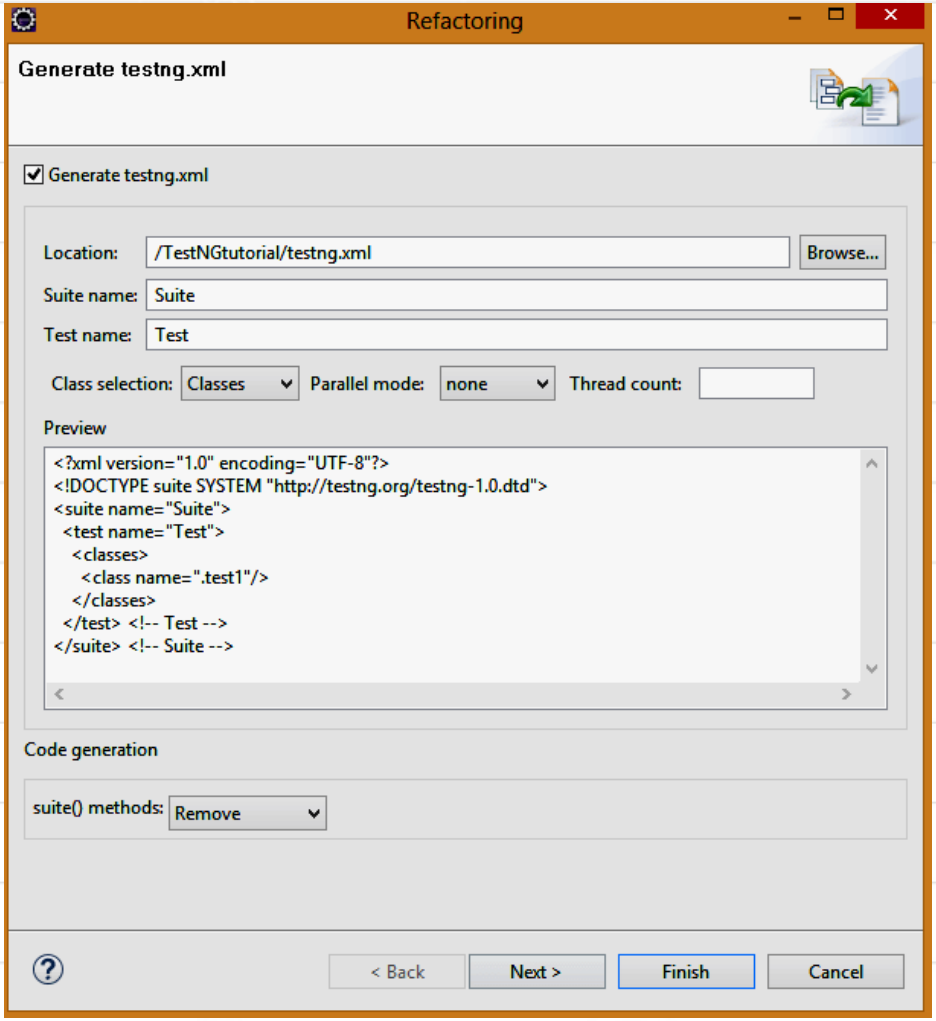
How to Create a TestNG.xml?

- **Right click** on the project.
- Move your cursor down, and click on the **Convert to TestNG**.



How to Create a TestNG.xml? (cont..)

- Click on the **Next** button.



Concepts Used In TestNG.xml

1. **Suite** is represented by one XML file. It can contain one or more tests and is defined by the `<suite>` tag.

   Example: `<suite name="Testing My Project">`

2. **Test** is represented by `<test>` and can contain one or more TestNG classes.

   Example: `<test name="UnitTest">`

3. **Class** is a Java class that contains TestNG annotations. It is represented by the `<class>` tag and can contain one or more test methods.

   Example:

```
<classes>
   <class name="com.demo.TestClass1"/>
   <class name="com.demo.TestClass2"/>
   <class name="com.demo.TestClass3"/>
</classes>
```

TestClass1.java

```
public class TestClass1 {
  @Test
  public void m1() {
    System.out.println("Class 1, Test method 1");
  }

  @Test()
  public void m2() {
    System.out.println("Class 1, Test method 2");
  }

  @Test
  public void m3() {
    System.out.println("Class 1, Test method 3");
  }
}
```

TestClass2.java

```
public class TestClass2 {
  @Test
  public void m1() {
    System.out.println("Class 2, Test method 1");
  }

  @Test()
  public void m2() {
    System.out.println("Class 2, Test method 2");
  }
}
```

TestClass3.java

```
public class TestClass3 {
  @Test
  public void m1() {
    System.out.println("Class 3, Test method 1");
  }

  @Test()
  public void m2() {
    System.out.println("Class 3, Test method 2");
  }

  @Test
  public void m3() {
    System.out.println("Class 3, Test method 3");
  }
}
```

TestNG.xml

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

<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="com.demo.TestClass1"/>
      <class name="com.demo.TestClass2"/>
      <class name="com.demo.TestClass3"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

How to Execute Specific Tests From a Large Testing Suite Using TestNG.xml?

```
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="com.demo.TestClass1"/>
        <methods>
          <include name="m1"/>
          <include name="m3"/>
        </methods>
      <class name="com.demo.TestClass2"/>
        <methods>
          <include name="m2"/>
        </methods>
      <class name="com.demo.TestClass3"/>
        <methods>
          <include name="m2"/>
          <include name="m3"/>
        </methods>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

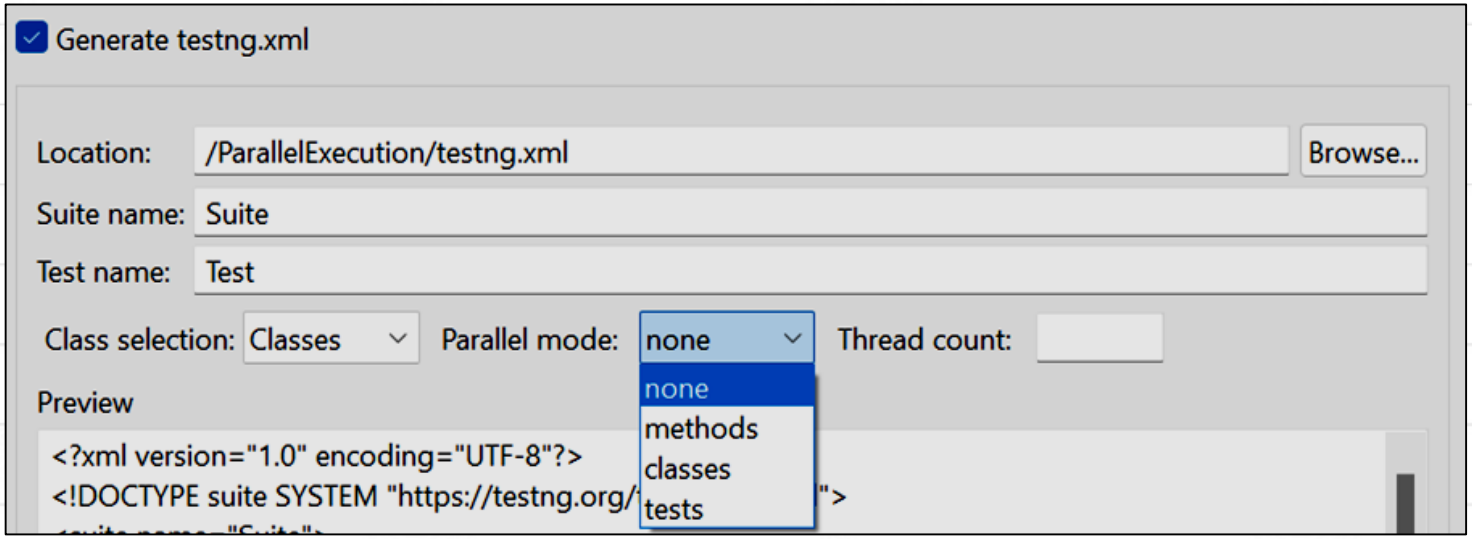
How to Enable or Disable Test Cases From TestNG.xml?

- To enable or disable a test case in TestNG.xml, an attribute called **enabled** is passed which is a boolean attribute (true or false).

```
<test name="TestSet1" enabled="false">
   <classes>
      .
      .
      .
   </classes>
</test>
<test name="TestSet2" enabled="true">
   <classes>
      .
      .
      .
   </classes>
</test>
```

How to Perform Parallel Execution of Test Methods Using TestNG.xml?

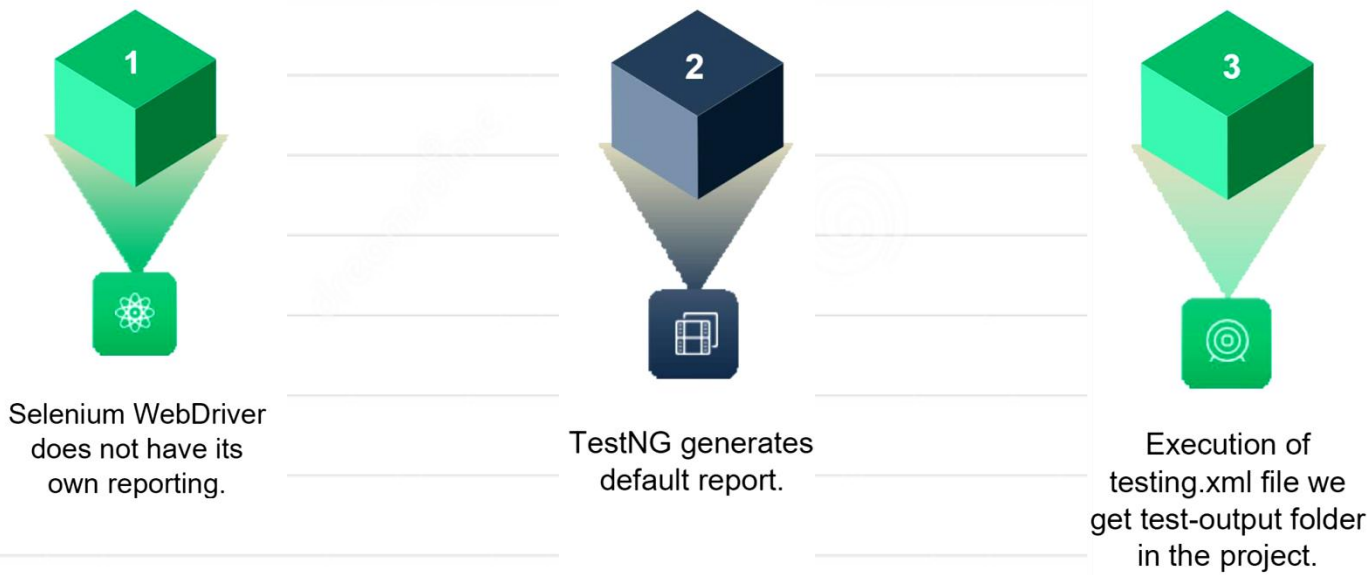
- To enable parallel execution at the test level in TestNG, we can specify the **`parallel`** attribute in the **`<test>`** tag in the TestNG XML configuration file.



- The parallel attribute accepts three values:
  - **methods**: Runs all methods with `@Test` annotation in parallel mode
  - **tests**: Runs all test cases present inside `<test>` tag in the XML in parallel mode
  - **classes**: Runs all test cases present inside classes in the XML in parallel mode

TestNG Reports

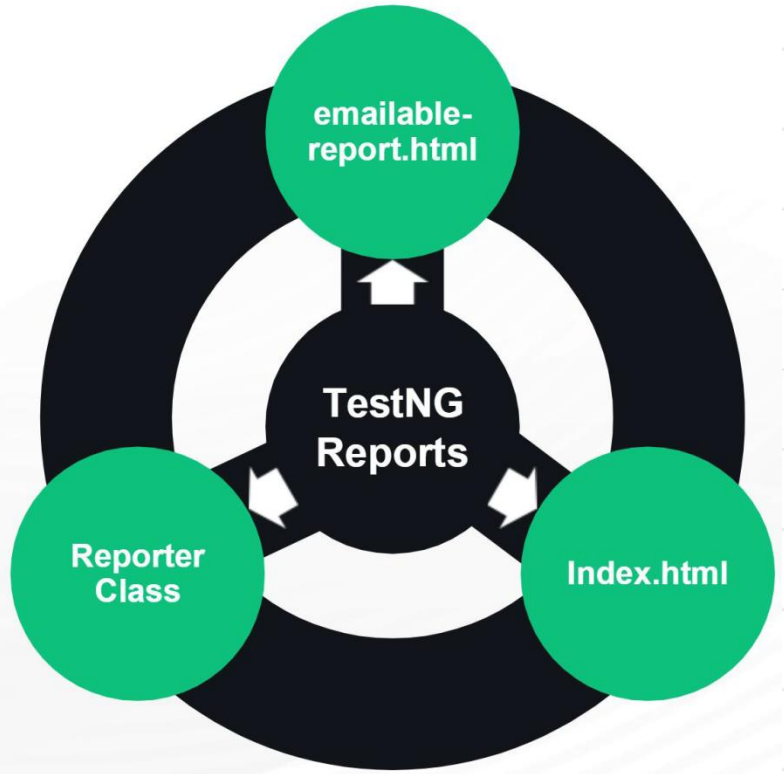
TestNG Reports



- Generation of reports is very important while doing Automation Testing as no end client will be interested in seeing the code rather they will see the reports.
- Reports give complete statistics of how many test cases – passed, failed and skipped.
- When executing testng.xml file and refresh the project, a **test-output** folder will be there in that folder.

Types of TestNG Reports

1. Emailable Reports
2. Index Reports
3. Reporter Class



How to Generate Emailable Report in TestNG?

- Emailable reports are generated in TestNG to let the user send their test reports to other team members.
- To generate emailable reports:
  1. Explore the ***test-output folder***
  2. Right-click on the ***emailable-report.html*** file
  3. Choose ***Open With -> Web Browser***

Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite1							
Unit Test	5	0	0	0	52		

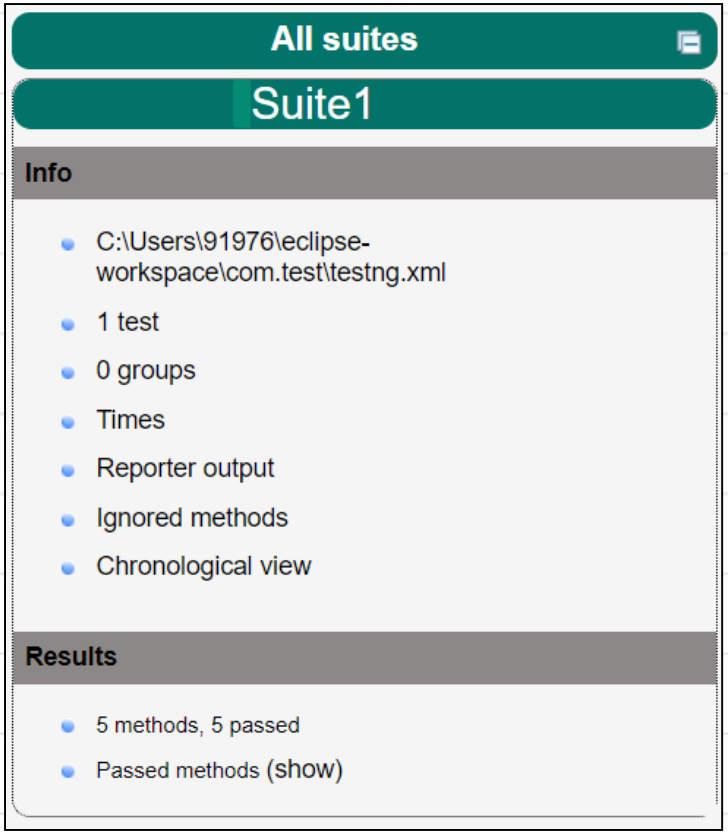
Class	Method	Start	Time (ms)
Suite1			
Unit Test — passed			
com.test.TestClass1	m1	1712942403963	6
	m2	1712942403972	1
	m3	1712942403974	1
com.test.TestClass2	m1	1712942403977	1
	m2	1712942403979	1

How to Generate Index File in TestNG?

- Index reports, contains the index-like structure of different parts of the report, such as failed tests, test file, passed test, etc.

- To generate index reports:

  1. Explore the ***test-output folder***
  2. Right-click on the ***index.html*** file
  3. Choose ***Open With -> Web Browser***



How to Use Reporter Class to Generate TestNG Reports?

- Reporter class is an inbuilt class in TestNG.
- The reporter class in TestNG helps in storing the logs inside the reports that are user-generated or system-generated.
  - We can view the logs directly from there rather than rerunning the test case.
- To use the reporter class:
  1. Add the following syntax: **`Reporter.log(string);`**
  2. Run ***testng.xml***
  3. View the ***emailable-report.html*** in the browser

XSLT Report Generation Using TestNG and ANT

XSLT Report

- XSLT stands for **Extensible Stylesheet Language Transformations**.
- XSLT is basically a transformation language which transforms one XML document into XHTML document which is convenient for any browser to display the test reporting.
- It uses **XPath** to navigate through elements and attributes in XML documents.
- We can customize output files by adding/removing attributes and elements in XML files using XSLT.
- This helps interpreting results quickly and it is supported by all browsers.

Pre-requisite to Generate XSLT Report

1. The project must be configured with the **ANT** build tool.
2. **XSLT Package** inside the project folder.
3. The project must be developed with **TestNG**.

What is ANT?

- ANT stands for **Another Neat** Tool.
- It is a Java-based build tool from computer software development company **Apache**.
- ANT helps us to automate and simplify tasks like build and deployment that include:
  - Compiling the code
  - Packaging the binaries
  - Deploying the binaries to the test server
  - Testing the changes
  - Copying the code from one location to another

Step 1: How to Install ANT?

1. Go to https://ant.apache.org/bindownload.cgi and download the .zip file.

1.9.16 release - requires minimum of Java 5 at runtime

- 1.9.16 .zip archive: apache-ant-1.9.16-bin.zip [PGP] [SHA512]
- 1.9.16 .tar.gz archive: apache-ant-1.9.16-bin.tar.gz [PGP] [SHA512]
- 1.9.16 .tar.bz2 archive: apache-ant-1.9.16-bin.tar.bz2 [PGP] [SHA512]

1.10.14 release - requires minimum of Java 8 at runtime

- 1.10.14 .zip archive: apache-ant-1.10.14-bin.zip [PGP] [SHA512]
- 1.10.14 .tar.gz archive: apache-ant-1.10.14-bin.tar.gz [PGP] [SHA512]
- 1.10.14 .tar.bz2 archive: apache-ant-1.10.14-bin.tar.bz2 [PGP] [SHA512]
- 1.10.14 .tar.xz archive: apache-ant-1.10.14-bin.tar.xz [PGP] [SHA512]

2. Unzip the zip file to a convenient location **C:\apache-ant-1.10.14** by using Winzip, WinRAR, 7-zip or similar tools.

3. Add **C:\apache-ant-1.10.14 \bin** to the System Path Variable.

Step 1: How to Install ANT?

4. To validate the installation, type **ant** at the Command Prompt**:**

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>ant
Buildfile: build.xml does not exist!
Build failed

C:\Windows\System32>
```

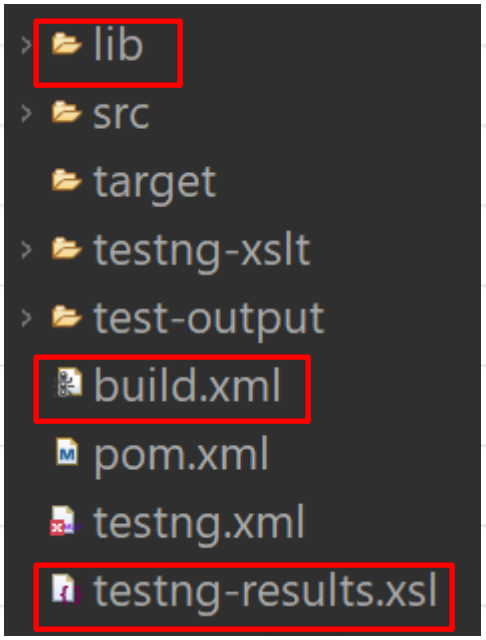
You will get **Build Failed** status since no build generated yet.

Step 2: How to Configure XSLT?

1. Download and extract XSLT file from here :

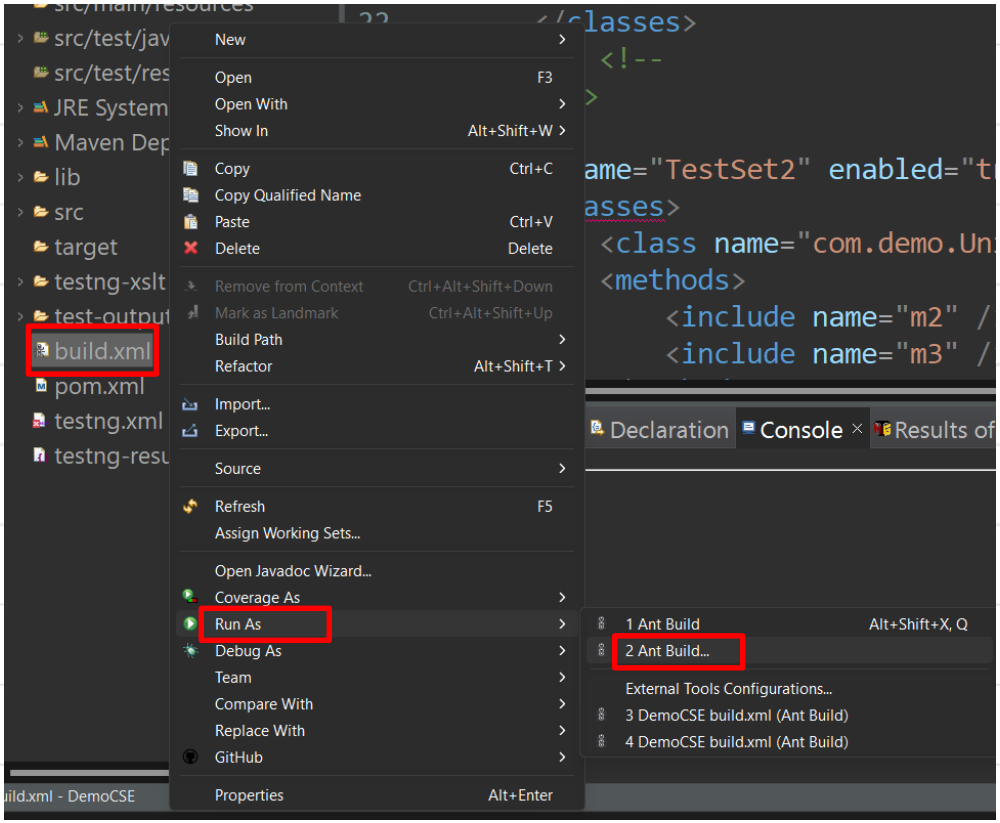
lib	02-06-2014 11:29 PM	File folder	
build.xml	17-11-2013 12:53 AM	xmlfile	2 KB
testng-results	08-05-2013 09:39 AM	XSL Stylesheet	58 KB

2. Copy and store the files in your project:



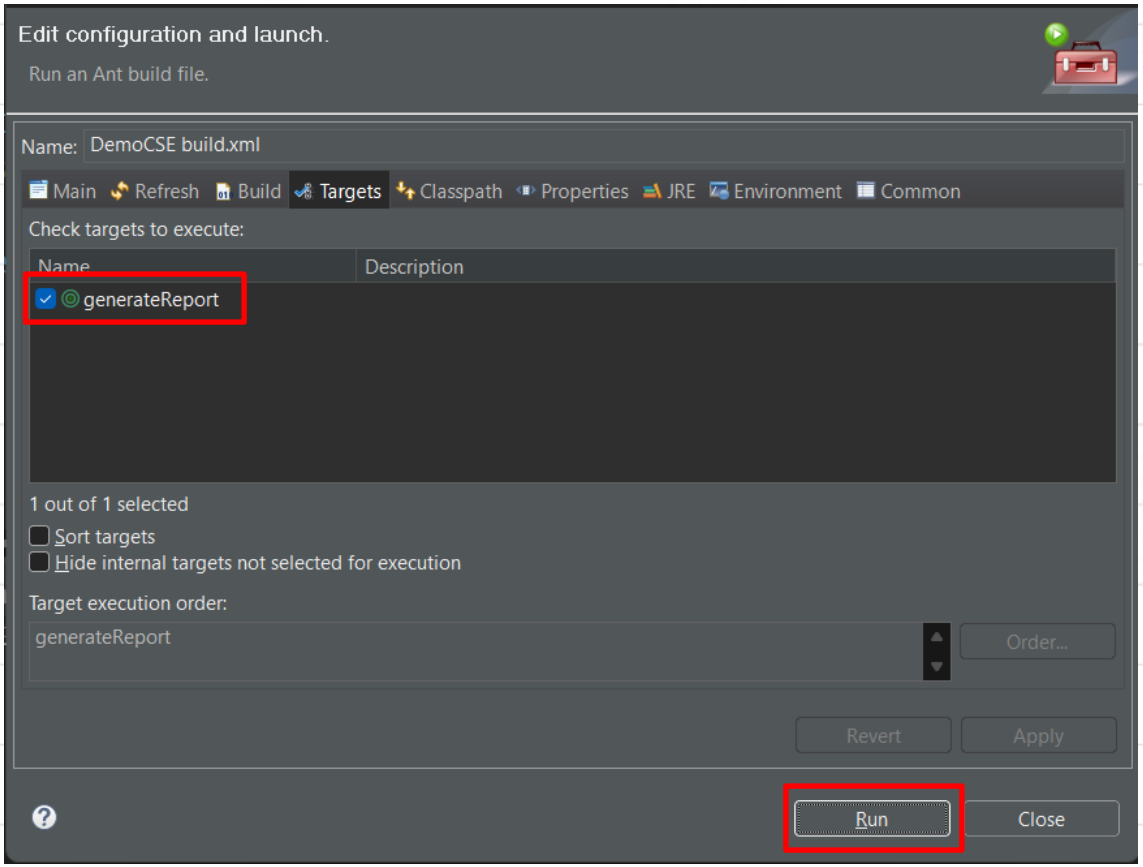
Step 3: How to Generate XSLT Report with TestNG and ANT?

1. Go to your project in Eclipse. Perform right click on **build.xml** and Run as **Ant Build**.



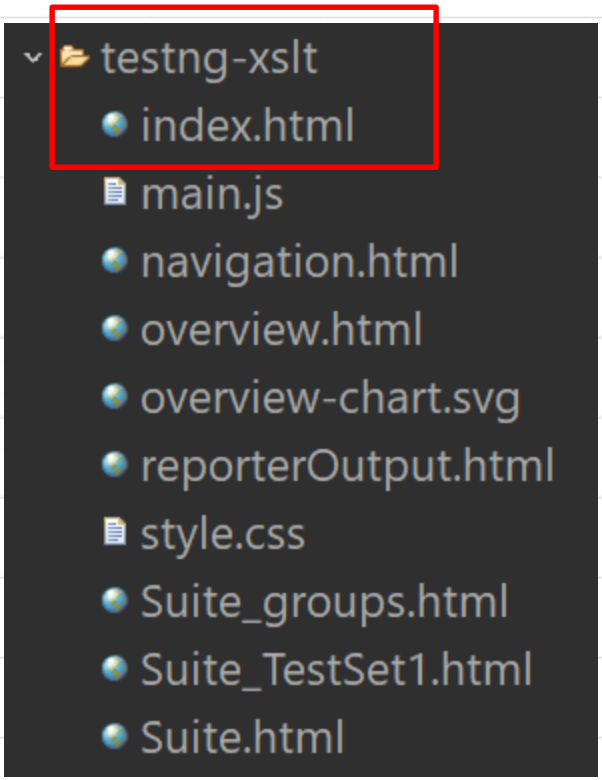
Step 3: How to Generate XSLT Report with TestNG and ANT?

2. In the next window, select **generateReport** and click **Run**.



Step 3: How to Generate XSLT Report with TestNG and ANT?

3. **testng-xslt** folder will be added to the project. Open it and run **index.html**.



Step 3: How to Generate XSLT Report with TestNG and ANT?

4. The **report** will be displayed in the browser.