What is TestNG?

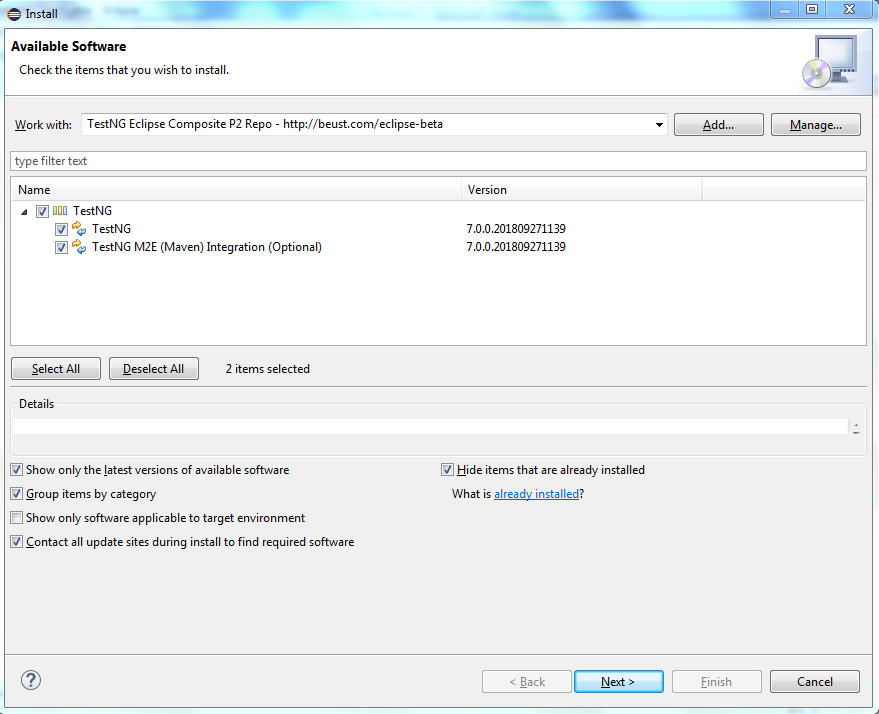
TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

How to install TestNG in Eclipse?

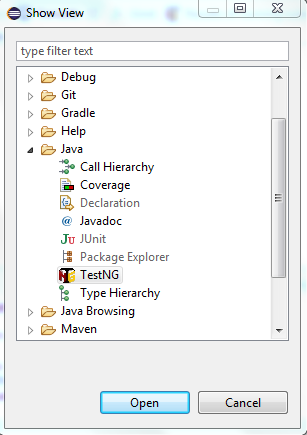
1. Open below site in browser

<https://testng.org/doc/download.html>

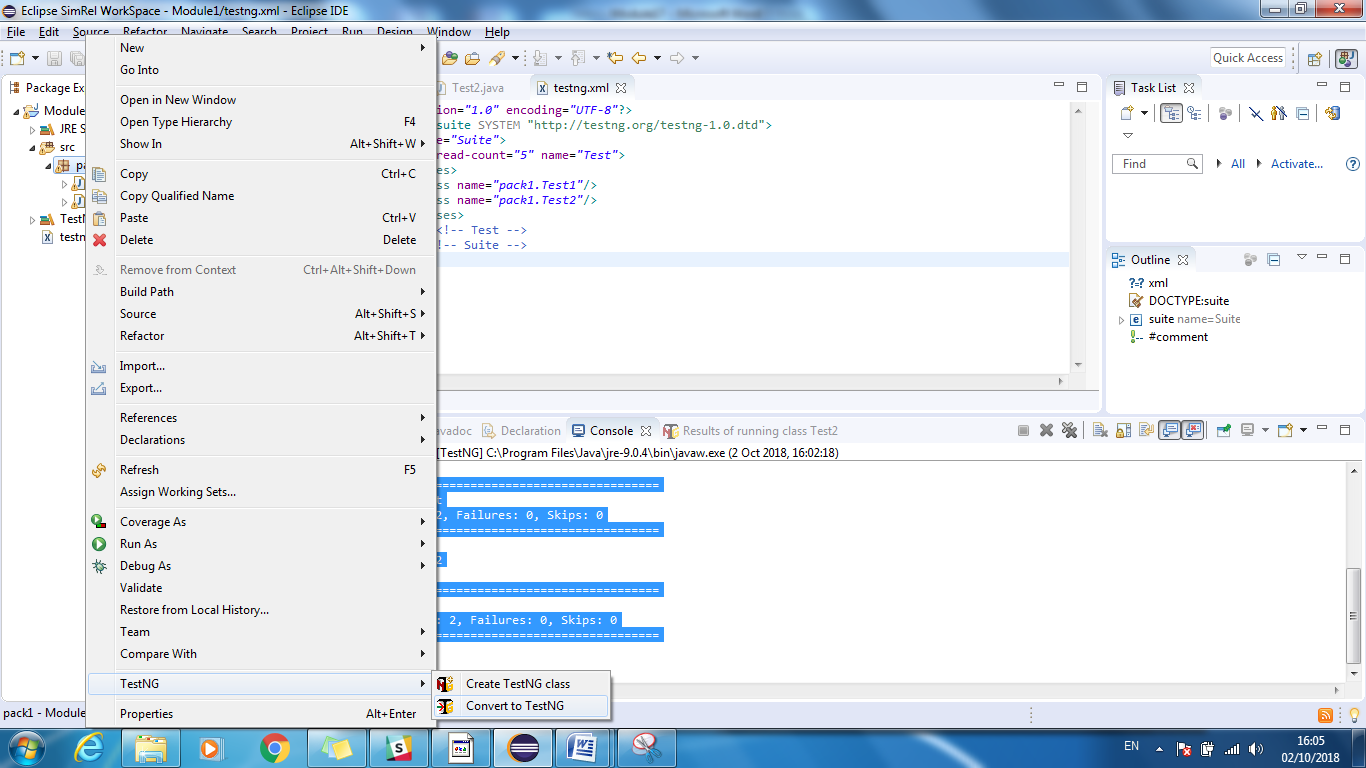
1. Refer to section ’ Install from update site’. Copy link <http://beust.com/eclipse>
2. In Eclipse, click Help->Install New Software
3. Paste the link in Work with field and hit Enter. TestNG option will be displayed.
4. Select all TestNG option and click Next until installation is finished.



In Eclipse navigate to Window->Show View->Other->Java. TestNG option should be visible as displayed below. It means installation of TestNG is successful.



In order to create testing.xml, right click on pack1 and select TestNG->Convert to TestNG as shown below and then click ok.



Below testng.xml is created

<?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=*"pack1.Test1"*/>

<class name=*"pack1.Test2"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Run the testing.xml as TestNG suite and below is the output

[RemoteTestNG] detected TestNG version 6.14.2

@BeforeSuite Test1

beforeSuite Test2

@BeforeTest Test1

beforeTest Test2

@BeforeClass Test1

@BeforeMethod Test1

In method LoginTest

@AfterMethod Test1

@BeforeMethod Test1

In method LogoffTest

@AfterMethod Test1

@AfterClass Test1

Beforeclass Test2

beforeMethod Test2

sendMessagetest method Test2

aftermethod Test2

beforeMethod Test2

sendemailtest method Test2

aftermethod Test2

afterClass Test2

@AfterTest Test1

afterTest test2

@AfterSuite Test1

afterSuite Test2

===============================================

Suite

Total tests run: 4, Failures: 0, Skips: 0

===============================================

**In order to understand testing.xml it’s important to understand the basics of XML DTD**

Refer below link for explanation

<https://www.w3schools.com/xml/xml_dtd_intro.asp>

What is XML DTD (Document Type Definition)?

* A DTD defines the structure and the legal elements and attributes of an XML document.
* With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
* An application can use a DTD to verify that XML data is valid.

XML document with an internal DTD

<?xml version="1.0"?>  
<!DOCTYPE note [  
<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>  
]>  
<note>  
<to>Tove</to>  
<from>Jani</from>  
<heading>Reminder</heading>  
<body>Don't forget me this weekend</body>  
</note>

XML document with a reference to an external DTD

<?xml version="1.0"?>  
<!DOCTYPE note SYSTEM "note.dtd">  
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>

“note.dtd”

<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>

All XML documents are made up by the following building blocks:

* Elements

XML elements are declared with the following syntax:

<!ELEMENT element-name category>

or

<!ELEMENT element-name (element-content)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children.

Examples of XML elements are "note", “to”, “from”, “heading”, “body”

Empty elements are declared with the category keyword EMPTY:

<!ELEMENT element-name EMPTY>

Elements with only parsed character data are declared with #PCDATA inside parentheses:

<!ELEMENT element-name (#PCDATA)>

Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>

Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1,child2,...)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children.

Declaring Only One Occurrence of an Element

<!ELEMENT element-name (child-name)>

Declaring Minimum One Occurrence of an Element

<!ELEMENT element-name (child-name+)>

Declaring Zero or More Occurrences of an Element

<!ELEMENT element-name (child-name\*)>

Declaring Zero or One Occurrences of an Element

<!ELEMENT element-name (child-name?)>

Declaring either/or Content

<!ELEMENT element-name child-name1|child-name2|child-name3)>

e.g. <!ELEMENT note (to,from,header,(message|body))>

Declaring Mixed Content

<!ELEMENT note (#PCDATA|to|from|header|message)\*>

* Attributes Syntax

<!ATTLIST element-name

attribute-name attribute-type attribute-value

attribute-name attribute-type attribute-value

attribute-name attribute-type attribute-value

…

>

Attributes provide **extra information about elements**.

The attribute-type can be one of the following:

|  |  |
| --- | --- |
| **Type** | **Description** |
| CDATA | The value is character data |
| (*en1*|*en2*|..) | The value must be one from an enumerated list |
| ID | The value is a unique id |
| IDREF | The value is the id of another element |
| IDREFS | The value is a list of other ids |
| NMTOKEN | The value is a valid XML name |
| NMTOKENS | The value is a list of valid XML names |
| ENTITY | The value is an entity |
| ENTITIES | The value is a list of entities |
| NOTATION | The value is a name of a notation |
| xml: | The value is a predefined xml value |

The attribute-value can be one of the following:

|  |  |  |
| --- | --- | --- |
| **Value** | **Explanation** | **E.g.** |
| value | The default value of the attribute | Syntax  <!ELEMENT square EMPTY> <!ATTLIST square width CDATA "0">  Valid XML: <square width="100" /> |
| #REQUIRED | The attribute is required | <!ATTLIST person number CDATA #REQUIRED>  Valid XML: <person number="5677" />  Invalid XML: <person /> |
| #IMPLIED | The attribute is optional | <!ATTLIST contact fax CDATA #IMPLIED>  Valid XML: <contact fax="555-667788" />  Valid XML: <contact /> |
| #FIXED *value* | The attribute value is fixed | DTD: <!ATTLIST sender company CDATA #FIXED "Microsoft">  Valid XML: <sender company="Microsoft" />  Invalid XML: <sender company="W3Schools" /> |

Enumerated Attribute Values

DTD:  
<!ATTLIST payment type (check|cash) "cash">  
  
XML example:  
<payment type="check" />  
or  
<payment type="cash" />

* Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

|  |  |
| --- | --- |
| **Entity References** | **Character** |
| &lt; | < |
| &gt; | > |
| &amp; | & |
| &quot; | " |
| &apos; | ' |

* PCDATA: parsed character data (It cannot contain any &, <, or > characters; these need to be represented by the &amp; &lt; and &gt; entities, respectively.) Tags inside the PCDATA will be treated as markup and entities will be expanded.
* CDATA: Stands for character data. CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Below is testng-1.0.dtd document

Refer link: <http://testng.org/testng-1.0.dtd.php>

<!--

Here is a quick overview of the main parts of this DTD. For more information,

refer to the <a href="http://testng.org">main web site</a>.

A <b>suite</b> is made of <b>tests</b> and <b>parameters</b>.

A <b>test</b> is made of three parts:

<ul>

<li> <b>parameters</b>, which override the suite parameters

<li> <b>groups</b>, made of two parts

<li> <b>classes</b>, defining which classes are going to be part

of this test run

</ul>

In turn, <b>groups</b> are made of two parts:

<ul>

<li> Definitions, which allow you to group groups into

bigger groups

<li> Runs, which defines the groups that the methods

must belong to in order to be run during this test

</ul>

Cedric Beust & Alexandru Popescu

@title DTD for TestNG

@root suite

-->

<!-- A suite is the top-level element of a testng.xml file -->

<!ELEMENT suite (groups?,(listeners|packages|test|parameter|method-selectors|suite-files)\*) >

<!-- Attributes: -->

<!--

@attr name The name of this suite (as it will appear in the reports)

@attr junit Whether to run in JUnit mode.

@attr verbose How verbose the output on the console will be.

This setting has no impact on the HTML reports.

@attr parallel Whether TestNG should use different threads

to run your tests (might speed up the process)

Do not use "true" and "false" values, they are now deprecated.

@attr parent-module A module used to create the parent injector of all guice injectors used

in tests of the suite

@attr guice-stage The stage with which the parent injector is created

@attr configfailurepolicy Whether to continue attempting Before/After

Class/Methods after they've failed once or just skip remaining.

@attr thread-count An integer giving the size of the thread pool to use

if you set parallel.

@attr annotations If "javadoc", TestNG will look for

JavaDoc annotations in your sources, otherwise it will

use JDK5 annotations.

@attr time-out The time to wait in milliseconds before aborting the

method (if parallel="methods") or the test (parallel="tests")

@attr skipfailedinvocationcounts Whether to skip failed invocations.

@attr data-provider-thread-count An integer giving the size of the thread pool to use

for parallel data providers.

@attr object-factory A class that implements IObjectFactory that will be used to

instantiate the test objects.

@attr allow-return-values If true, tests that return a value will be run as well

-->

<!ATTLIST suite

name CDATA #REQUIRED

junit (true | false) *"false"*

verbose CDATA #IMPLIED

parallel (false | true | none | methods | tests | classes | instances) *"none"*

parent-module CDATA #IMPLIED

guice-stage (DEVELOPMENT | PRODUCTION | TOOL) *"DEVELOPMENT"*

configfailurepolicy (skip | continue) *"skip"*

thread-count CDATA *"5"*

annotations CDATA #IMPLIED

time-out CDATA #IMPLIED

skipfailedinvocationcounts (true | false) *"false"*

data-provider-thread-count CDATA *"10"*

object-factory CDATA #IMPLIED

group-by-instances (true | false) *"false"*

preserve-order (true | false) *"true"*

allow-return-values (true | false) *"false"*

>

<!-- A list of XML files that contain more suite descriptions -->

<!ELEMENT suite-files (suite-file)\* >

<!ELEMENT suite-file ANY >

<!ATTLIST suite-file

path CDATA #REQUIRED

>

<!--

Parameters can be defined at the <suite> or at the <test> level.

Parameters defined at the <test> level override parameters of the same name in <suite>

Parameters are used to link Java method parameters to their actual value, defined here.

-->

<!ELEMENT parameter ANY>

<!ATTLIST parameter

name CDATA #REQUIRED

value CDATA #REQUIRED >

<!--

Method selectors define user classes used to select which methods to run.

They need to implement <tt>org.testng.IMethodSelector</tt>

-->

<!ELEMENT method-selectors (method-selector\*) >

<!ELEMENT method-selector ((selector-class)\*|script) >

<!ELEMENT selector-class ANY>

<!ATTLIST selector-class

name CDATA #REQUIRED

priority CDATA #IMPLIED

>

<!ELEMENT script ANY>

<!ATTLIST script

language CDATA #REQUIRED

>

<!--

A test contains parameters and classes. Additionally, you can define additional groups ("groups of groups")

-->

<!ELEMENT test (method-selectors?,parameter\*,groups?,packages?,classes?) >

<!--

@attr name The name of this test (as it will appear in the reports)

@attr junit Whether to run in JUnit mode.

@attr verbose How verbose the output on the console will be.

This setting has no impact on the HTML reports.

Default value: suite level verbose.

@attr parallel Whether TestNG should use different threads

to run your tests (might speed up the process)

Do not use "true" and "false" values, they are now deprecated.

@attr thread-count An integer giving the size of the thread pool to be used if

parallel mode is used. Overrides the suite level value.

@attr annotations If "javadoc", TestNG will look for

JavaDoc annotations in your sources, otherwise it will

use JDK5 annotations.

@attr time-out the time to wait in milliseconds before aborting

the method (if parallel="methods") or the test (if parallel="tests")

@attr enabled flag to enable/disable current test. Default value: true

@attr skipfailedinvocationcounts Whether to skip failed invocations.

@attr preserve-order If true, the classes in this tag will be run in the same order as

found in the XML file.

@attr allow-return-values If true, tests that return a value will be run as well

-->

<!ATTLIST test

name CDATA #REQUIRED

junit (true | false) *"false"*

verbose CDATA #IMPLIED

parallel (false | true | none | methods | tests | classes | instances) #IMPLIED

thread-count CDATA #IMPLIED

annotations CDATA #IMPLIED

time-out CDATA #IMPLIED

enabled (true | false) #IMPLIED

skipfailedinvocationcounts (true | false) *"false"*

preserve-order (true | false) *"true"*

group-by-instances (true | false) *"false"*

allow-return-values (true | false) *"false"*

>

<!--

Defines additional groups ("groups of groups") and also which groups to include in this test run

-->

<!ELEMENT groups (define\*,run?,dependencies?) >

<!ELEMENT define (include\*)>

<!ATTLIST define

name CDATA #REQUIRED>

<!-- Defines which groups to include in the current group of groups -->

<!ELEMENT include ANY>

<!ATTLIST include

name CDATA #REQUIRED

description CDATA #IMPLIED

invocation-numbers CDATA #IMPLIED>

<!-- Defines which groups to exclude from the current group of groups -->

<!ELEMENT exclude ANY>

<!ATTLIST exclude

name CDATA #REQUIRED>

<!-- The subtag of groups used to define which groups should be run -->

<!ELEMENT run (include?,exclude?)\* >

<!ELEMENT dependencies (group\*)>

<!ELEMENT group ANY>

<!ATTLIST group

name CDATA #REQUIRED

depends-on CDATA #REQUIRED>

<!-- The list of classes to include in this test -->

<!ELEMENT classes (class\*,parameter\*) >

<!ELEMENT class (methods|parameter)\* >

<!ATTLIST class

name CDATA #REQUIRED >

<!-- The list of packages to include in this test -->

<!ELEMENT packages (package\*) >

<!-- The package description.

If the package name ends with .\* then subpackages are included too.

-->

<!ELEMENT package (include?,exclude?)\*>

<!ATTLIST package

name CDATA #REQUIRED >

<!-- The list of methods to include/exclude from this test -->

<!ELEMENT methods (include?,exclude?,parameter?)\* >

<!-- The list of listeners that will be passed to TestNG -->

<!ELEMENT listeners (listener\*) >

<!ELEMENT listener ANY>

<!ATTLIST listener

class-name CDATA #REQUIRED >

**Let’s learn about testng.xml**

testng.xml is a configuration file for TestNG. It is used to define test suites and tests in TestNG. It is also used to pass parameters to test methods.

testNG allows you to do the following:

* Create tests with packages
* Create tests using classes
* Create tests using test methods
* Include/exclude a particular package, class, or test method
* Use of regular expression while using the include/exclude feature
* Store parameter values for passing to test methods at runtime
* Configure multithreaded execution options

Follow below steps for creating a test suite:

1. Go to the Eclipse project that we created in the previous chapter.
2. Select the project and then right-click on it and select New | File.
3. Select the project in the File window.
4. Enter text testng.xml in the File name section, and click on Finish.
5. Eclipse will add the new file to your project and will open the file in the editor
6. Add the following snippet to the newly created tesntg.xml file and save it.

<suite name=*"First Suite"* verbose=*"1"* >

<test name=*"First Test"* >

<classes>

<class name=*"test.FirstTest"* />

</classes>

</test>

</suite>

Follow below steps for executing testng.xml using Eclipse:

1. Open Eclipse and go to the project where we have created the testng.xml file.
2. Select the testng.xml file, right-click on it, and select Run As | TestNG suite.
3. Eclipse will execute the XML file as TestNG suite

How to create testng XML with multiple tests in Eclipse?

1. Create below testNG class files

**package** pack1;

**import** org.testng.annotations.Test;

**public** **class** FirstTestClass {

@Test

**public** **void** firstTest\_firstMethod() {

System.***out***.println("First test and first method");

}

@Test

**public** **void** firstTest\_secondMethod() {

System.***out***.println("First test and Second method");

}

}

**package** pack1;

**import** org.testng.annotations.Test;

**public** **class** SecondTestClass {

@Test

**public** **void** secondTest\_firstMethod() {

System.***out***.println("Second test and first method");

}

@Test

**public** **void** secondTest\_secondMethod() {

System.***out***.println("Second test and second method");

}

}

**package** pack2;

**import** org.testng.annotations.Test;

**public** **class** ThirdTestClass {

@Test

**public** **void** thirdTest\_firstMethod() {

System.***out***.println("Third test and first method");

}

@Test

**public** **void** thirdTest\_secondMethod() {

System.***out***.println("Third test and second method");

}

}

**package** pack2;

**import** org.testng.annotations.Test;

**public** **class** FourthTestClass {

@Test

**public** **void** fourthTest\_firstMethod() {

System.***out***.println("Fourth test and first method");

}

@Test

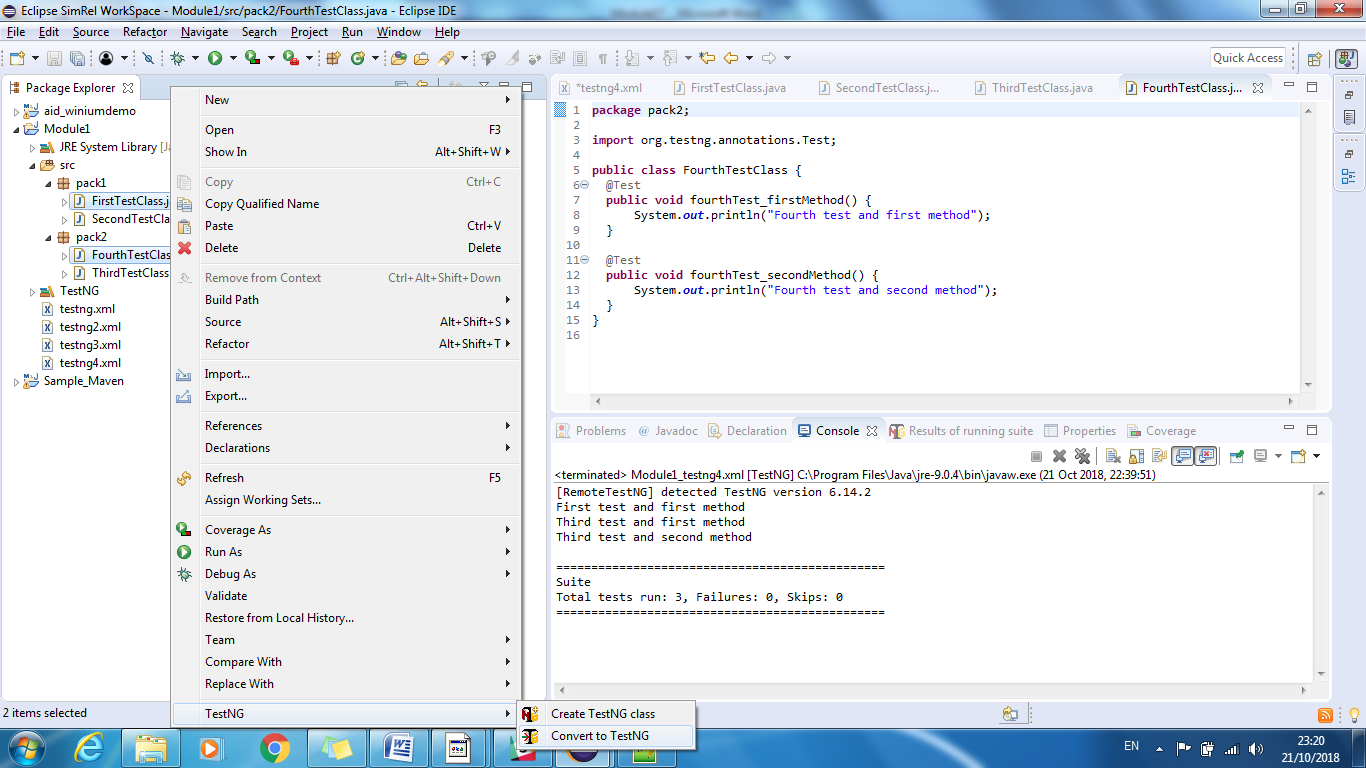
**public** **void** fourthTest\_secondMethod() {

System.***out***.println("Fourth test and second method");

}

}

1. Select these two classes, right click on it, select TestNG, select Convert to TestNG as shown in below diagram.



1. Below code is generated

This is also called testng XML with multiple tests

<?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=*"pack2.FourthTestClass"*/>

<class name=*"pack1.FirstTestClass"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

**testng.xml to run tests with a package e.g. pack1**

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

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

<suite name=*"Suite"* thread-count=*"5"*>

<test name=*"FirstTest"*>

<packages>

<package name=*"pack1"*></package>

</packages>

</test>

</suite>

**testng.xml to run tests with a method e.g. pack1**

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

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

<suite name=*"Suite"* thread-count=*"5"*>

<test name=*"FirstTest"*>

<classes>

<class name=*"pack1.FirstTestClass"*>

<methods>

<include name=*"firstTest\_firstMethod"*></include>

</methods>

</class>

</classes>

</test>

</suite>

creating a test suite with package, class, and test method

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

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

<suite name=*"Suite"* thread-count=*"5"*>

<test name=*"test1"*>

<packages>

<package name=*"pack1"*></package>

<package name=*"pack2"*></package>

</packages>

</test>

<test name=*"test2"*>

<classes>

<class name=*"pack2.ThirdTestClass"*></class>

</classes>

</test>

<test name=*"test3"*>

<classes>

<class name=*"pack2.FourthTestClass"*>

<methods>

<include name=*"fourthTest\_firstMethod"*></include>

</methods>

</class>

</classes>

</test>

</suite>

test suite to include a particular package

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

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

<suite name=*"Suite"* thread-count=*"5"*>

<test name=*"test1"*>

<packages>

<package name=*"pack1.\*"*> <!-- adds all packages within package 'pack1' -->

<include name=*"pack1"*></include> <!-- includes only package 'pack1 and ignores other' -->

</package>

</packages>

</test>

</suite>

test suite to exclude a particular package

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

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

<suite name=*"Suite"* thread-count=*"5"*>

<test name=*"test1"*>

<packages>

<package name=*"pack1.\*"*> <!-- adds all packages within package 'pack1' -->

<exclude name=*"pack1"*></exclude> <!-- excludes only package 'pack1 and run others' -->

</package>

</packages>

</test>

</suite>

test suite to exclude a particular method

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

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

<suite name=*"Suite"* thread-count=*"5"*>

<test name=*"test1"*>

<classes>

<class name=*"pack1.FirstTestClass"*>

<methods>

<exclude name=*"firstTest\_firstMethod"*></exclude>

</methods>

</class>

</classes>

</test>

</suite>

using regular expressions for test

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

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

<suite name=*"Regular Exp Suite"* verbose=*"1"*>

<test name=*"Regular Exp Test"*>

<classes>

<class name=*"pack1.SecondTestClass"*>

<methods>

<include name=*".\*first.\*"* />

</methods>

</class>

</classes>

</test>

</suite>