Java means DURGA SOFT..

JAVA TOOLS Material

India's No.1 Software Training Institute
DURGASOFT
www.durgasoft.com Ph: 9246212143 ,8096969696

Junit

Testing:

Testing is nothing but checking the expected results with actual results.

TestCase:

TestCase is a TestPlan where expected results will be compared with actual results with specific input values. To test one functionality it is recommended to write more test cases with both wrong data and write data.

Unit Testing

The testing performed by the programmer on his own piece of code is called as UnitTesting

PeerTesting

The unit testing performed by the programmer on other programmers code is called PeerTesting.

- ✓ Unit Testing, Peer Testing are the responsibility of the programmer .The remaining testing ‘s like Integration Testing,Performance Testing ,Navigation Testing and etc are the responsibility of QA department.

TestSuite

TestSuite provides the environment to run all the TestCases.

What is Junit:

- ✓ **JUnit** is a program used to perform ***unit testing*** of virtually any software. JUnit testing is accomplished by writing test cases using Java, compiling these test cases and running the resultant classes with a JUnit Test Runner.
- ✓ I will explain a little bit about software and unit testing in general. What and how much to test depends on how important it is to know that the tested software works right, in relation to how much time you are willing to dedicate to testing. Since you are reading this, then for some reason you have decided to dedicate at least some time to unit testing.

Example

- ✓ I'm currently developing an SMTP server. An SMTP server needs to handle email addresses of the format specified in RFC documents. To be confident that my SMTP server complies with the RFC, I need to write a test case for each allowable email address type specified in the RFC. If the RFC says that the character "#" is prohibited, then I should write a test case that sends an address with "#" and verifies that the SMTP server rejects it. You don't need to have a formal specification document to work from. If I wasn't aiming for *compliance*, I could just decide that it's good enough if my server accepts email addresses consisting only of letters, numbers and "@", without caring whether other addresses succeed or fail.
- ✓ Another important point to understand is that, everything is better if you make your tests before you implement the features. For my SMTP server, I did some prototyping first to make sure that my design will work, but I did not attempt to satisfy my requirements with that prototype. I am now writing up my test cases and running them against existing SMTP server implementations (like Sendmail). When I get to the implementation stage, my work will be extremely well defined. Once my implementation satisfies all of the unit tests, just like Sendmail does,

then my implementation will be completed. At that point, I'll start working on new requirements to surpass the abilities of Sendmail and will work up those tests as much as possible before starting the second implementation iteration. (If you don't have an available test target, then there sometimes comes a point where the work involved in making a dummy test target isn't worth the trouble... so you just develop the test cases as far as you can until your application is ready).

- ✓ Everything I've said so far has to do with unit testing, not JUnit specifically. Now on to JUnit...
- ✓ JUnit was originally written by **Erich Gamma and Kent Beck**.
- ✓

Installing Junit:

- ✓ **Downloading :**

  You can download JUnit 3.x from **http://www.junit.org/index.htm** in the zipped format.

- ✓ **Installation :**

  Below are the installation steps for installing JUnit:

1. Unzip the JUnit3.x.zip file.

2. Add **junit-3.x.jar** to the CLASSPATH. or we can create a bat file "**setup.bat**" in which we can write "**set CLASSPATH=.;%CLASSPATH%;junit-3.x.jar;**". So whenever we need to use JUnit in our project then we can run this setup.bat file to set the classpath.

Test Coverage -- quantity of test cases

- ✓ With respect to test coverage, an ideal test would have test cases for every conceivable variation type of input, *not every possible instance of input*. You should have test cases for combinations or permutations of all variation types of the implemented operations. You use your knowledge of the method implementation (including possible implementation changes which you want to allow for) to narrow the quantity of test cases way down.

Example:

- ✓ I'll discuss testing the multiplication method of a calculator class as an example. First off, if your multiplcation method hands its parameters (the multiplication operands) exactly the same regardless of order (now and in the future), then you have just dramatically cut your work from covering all permutations to covering all combinations
- ✓ You will need to cover behavior of multiplying by zero, but it will take only a few test cases for this. It is extremely unlikely that the multiplication method code does anything differently for an operand value of 2 different than it would for an operand value of 3, therefore, there is no reason to have a test for "2 x 0" and for "3 x 0".
- ✓ All you need is one test with a positive integer operand and zero, like "46213 x 0". (Two test cases if your implementation is dependent upon input parameter sequence). You may or may not need additional cases to test other *types* of non-zero operands, and for the case of "0 x 0". Whether a test case is needed just depends upon whether your current and future code could ever behave differently for that case.

TestExpressions

- ✓ The most important thing is to write tests like

```
(expectedResult == obtainedResult)
```

Or

```
expectedResult.equals(obtainedResult)
```

- ✓ You can do that without JUnit, of course. For the purpose of this document, I will call the smallest unit of testing, like the expression above, a *test expression* . All that JUnit does is give you flexibility in grouping your test expressions and convenient ways to capture and/or summarize test failures.

Class junit.framework.Assert

- ✓ The Assert class has a bunch of static convenience methods to help you with individual test expressions. For example, without JUnit I might code

```
if (obtainedResult == null || !expectedResult.equals(obtainedResult))

    throw new MyTestException("Bad output for # attempt");
```

- ✓ With JUnit, you can code

```
Assert.assertEquals("Message for
Failure" expectedResult actualResult);
```

- ✓ There are lots of assertXxx() methods to take care of the several common Java types. They take care of checking for nulls. Both assert() failures and failure()s (which are effectively the same thing) throw junit.framework.AssertionFailedError Throwables, which the test-runners and listeners know what to do with. If a test expression fails, the containing test method quits (by virtue of throwing the AssertionFailedError internally, then the test-runner performs cleanup before executing the next test method in the sequence (with the error being noted for reporting now or later). Take a look at the API spec for junit.framework.Assert.

JUnit Failures vs. Errors

- ✓ JUnit test reports differentiate failures vs. errors. A **failure** is a test which your code has explicitly failed by using the mechanisms for that purpose (as described above). Generation of a failure indicates that your time investment is paying off--it points you right to an anticipated problem in the program that is the target of your testing.
- ✓ A JUnit error, on the other hand, indicates an unanticipated problem. It is either a resource problem such as is normally the domain of Java unchecked throwables or it is a problem with your implementation of the test. When you run a test and get an error, it means you really need to fix something, and usually not just the program that you intended to test. Either a problem has occurred outside of your test method (like in test class instantiation, or the test setup methods described in following chapters), or your test method has thrown an exception.

Note:

With some 3.x versions of JUnit, there are/were lifecycle side-effects when Errors were produced. Unless you know otherwise, don't assume that cleanup methods will run after test failure. (This is not a concern with JUnit 4.x, which is reliable in this respect).

Interface junit.framework.Test

- ✓ An object that you can run with the JUnit infrastructure is a Test. But you can't just implement Test and run that object. You can only run specially created

instances of TestCase. (Running other implementations of Test will compile, but produce runtime errors.)

Abstract class junit.framework.TestCase

- ✓ A test case defines the fixture to run multiple tests. To define a test case

  1. implement a subclass of TestCase
  2. define instance variables that store the state of the fixture
  3. initialize the fixture state by overriding setUp()
  4. clean-up after a test by overriding tearDown().

- ✓ setup() method is executed for every test case method execution. In this method we can perform Initialization activity.
- ✓ teardown() method is executed at the end of each Test Case method .In this method we can perform cleanup activity or uninitailiztion.
- ✓ Every Test case class must and should extend the TestCase class
- ✓ Every Test case class should follow the following conventions

  Coding Convention :

  1. Name of the test class must end with "Test".
  2. Name of the method must begin with "test".
  3. Return type of a test method must be void.
  4. Test method must not throw any exception.
  5. Test method must not have any parameter

Class junit.framework.TestSuite

- ✓ A TestSuite is just an object that contains an ordered list of runnable Test objects. TestSuites also implement Test() and are runnable. To run a TestSuite is just to run all of the elemental Tests in the specified order, where by elemental tests, I mean Tests for a single method like we used in the Abstract class junit.framework.TestCase .TestSuites can be nested. Remember that for every elemental Test run, three methods are actually invoked, setUp + test method + tearDown.

This is how TestSuites are used.

```
TestSuite s = new TestSuite();
   s.addTest(new TC("tcm"));
   s.addTest(new TD("tdm"));
   s.addTestSuite(AnotherTestSuiteImplementation.class);
   s.run(new TestResult());
```

Class junit.textui.TestRunner

A command line based tool to run tests.

```
java junit.textui.TestRunner [-wait]   TestCaseClass
```

TestRunner expects the name of a TestCase class as argument. If this class defines a static suite method it will be invoked and the returned test is run. Otherwise all the methods starting with "test" having no arguments are run.

When the wait command line argument is given TestRunner waits until the users types RETURN.

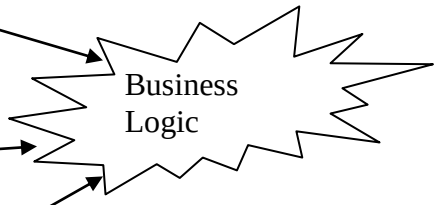
TestRunner prints a trace as the tests are executed followed by a summary at the end.

Examples

- ✓ If you want to work with Junit you need to write three classes
  1. Main Class(i.e our Actual Logic)
  2. TestCase class
  3. Use TestRunner class to run all TestCase classes

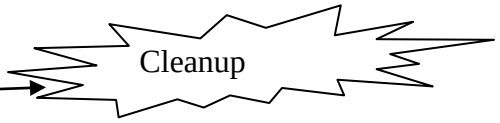
Calc.java

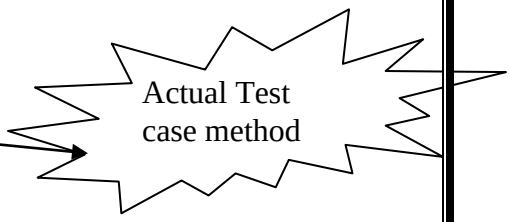
```
1  class  Calc
2  {
3     public int add(int a,int b)
4     {
5        return a+b;
6     }
7     public int sub(int a,int b)
8     {
9        return a-b;
10    }
11    public int div(int a,int b)
12    {
13       return a/b;
14    }
15
16 }
17
```



Write TestCase Class to Test the Functionality

```
import junit.framework.TestCase;
class CalcTest extends TestCase
{
   Calc c;

   //Initialization code
   public void setUp()
   {
      c=new Calc();
   }

   //cleanup code
   public void tearDown()
   {
      c=null;
   }

   //actual testCase methods
   public void testAdd()
   {
      assertEquals(2,c.add(2,1));
   }

}
```





Compile and Running The TestCases

```
D:\JUNIT\JUnitEx>set classpath=D:\JUNIT\junit4.10\junit-4.10.jar;.;
D:\JUNIT\JUnitEx>javac Calc.java
D:\JUNIT\JUnitEx>javac CalcTest.java
D:\JUNIT\JUnitEx>java junit.textui.TestRunner CalcTest
.
Time: 0
OK (1 test)

D:\JUNIT\JUnitEx>_
```

If TestCae Fails

```
D:\JUNIT\JUnitEx>javac CalcTest.java

D:\JUNIT\JUnitEx>java junit.textui.TestRunner CalcTest
.F
Time: 0
There was 1 failure:
1) testAdd(CalcTest)junit.framework.AssertionFailedError: expected:<2> but was:<
3>
        at CalcTest.testAdd(CalcTest.java:21)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0


D:\JUNIT\JUnitEx>
```

JUnit with Eclipse

Preparation

- ✓ Create a new project "com.dsoft.first". We want to create the unit tests in a separate folder. Create therefore a new source folder "test" via right mouse click on your project, select properties and choose the "Java Build Path". Select the tab source code.
- ✓ Press "Add folder" then then press "Create new folder". Create the folder "test".
  - The creation of an separate folder for the test is not mandatory. But it is good advice to keep the test coding separate from the normal coding.

Create a Java class

Create a package "com.dsoft.first" and the following class.

```
package com.dsoft.first;
public class MyClass {
        public int multiply(int x, int y)
        {
                return x / y;
        }
}
```

Create a JUnit test

- ✓ Select your new class, right mouse click and select New ->JUnit Test case, change the source folder to JUnit. Select "New JUnit 4 test". Make sure you change the source folder to test.
- ✓ Press next and select the methods which you want to test.
- ✓ If you have not yet JUnit in your classpath, Eclipse will asked you if it should be added to the classpath.

Create a test with the following code.

```
package com.dsoft.first;
 import org.junit.Test;
import static org.junit.Assert.assertEquals;
 public class MyClassTest {
 @Test
 public void testMultiply() {
 MyClass tester = new MyClass();
 assertEquals("Result", 50, tester.multiply(10, 5));
 }
}
```

Run your test via Eclipse

- ✓ Right click on your new test class and select Run-As-> Junit Test.
- ✓ The test should be failing (indicated via a red bar). This is due to the fact that our multiplier class is currently not working correctly (it does a division instead of multiplication). Fix the bug and re-run test to get a green light.
- ✓ If you have several tests you can combine them into a test suite. All test in this test suite will then be executed if you run the test suite. To create a new test suite, select your test classes, right mouse click-> New-> Other -> JUnit -Test Suite
- ✓ Select next and select the methods you would like to have test created for.

Tip

This does currently not work for JUnit4.0 testcases. See Bug Report

JAVA Means DURGA

- ✓ Change the coding to the following to make your test suite run your test. If you later develop another test you can add it to @Suite.SuiteClasses

```
package mypackage;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses( { MyClassTest.class })
public class AllTests {
}
```

Run your test via code

- ✓ You can also run your test via your own coding. The class "org.junit.runner.JUnitCore" provides the method runClasses() which allows you to run one or several tests classes. As a return parameter you receive an object of type "org.junit.runner.Result". This object can be used to retrieve information about the tests and provides information about the failed tests.
- ✓ Create in your "test" folder a new class "MyTestRunner" with the following coding. This class will execute your test class and write potential failures to the console.

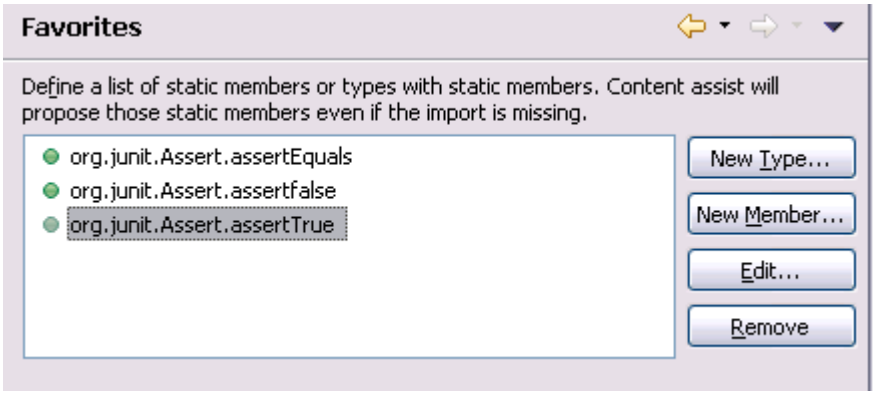
```
package com.dsoft;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class MyTestRunner {
        public static void main(String[] args) {
                Result result = JUnitCore.runClasses(MyClassTest.class);
                for (Failure failure : result.getFailures()) {
                        System.out.println(failure.toString());
                }
        }
}
```

JUnit (more) in Detail

Static imports with Eclipse

JUnit uses a lot of static methods and Eclipse cannot automatically import static imports. You can make the JUnit test methods available via the content assists.

Open the Preferences via Window -> Preferences and select Java > Editor > Content Assist > Favorites. Add then via "New Member" the methods you need. For example this makes the assertTrue, assertFalse and assertEquals method available.



You can now use Content Assist (Ctrl+Space) to add the method and the import.

I suggest to add at least the following new members.

- org.junit.Assert.assertTrue
- org.junit.Assert.assertFalse
- org.junit.Assert.assertEquals
- org.junit.Assert.fail

3.2. Annotations

The following give an overview of the available annotations in JUnit 4.x

Table 1. Annotations

Annotation	Description
@Test public void method()	Annotation @Test identifies that this method is a test method.
@Before public void method()	Will perform the method() before each test. This method can prepare the test environment, e.g. read input data, initialize the class)
@After public void method()	Test method must start with test
@BeforeClass public void method()	Will perform the method before the start of all tests. This can be used to perform time intensive activities for example be used to connect to a database
@AfterClass public void method()	Will perform the method after all tests have finished. This can be used to perform clean-up activities for example be used to disconnect to a

JAVA Means DURGA SOFT

Annotation	Description
	database
@Ignore	Will ignore the test method, e.g. useful if the underlying code has been changed and the test has not yet been adapted or if the runtime of this test is just to long to be included.
@Test(expected=IllegalArgumentException.class)	Tests if the method throws the named exception
@Test(timeout=100)	Fails if the method takes longer then 100 milliseconds

Assert statements

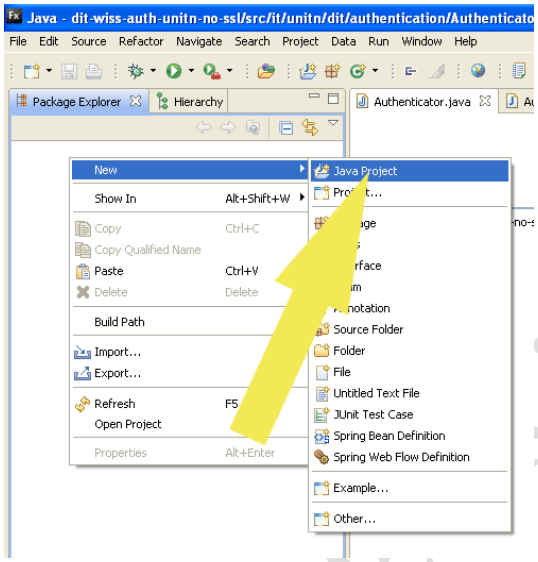
The following gives an overview of the available test methods:

Table 2. Test methods

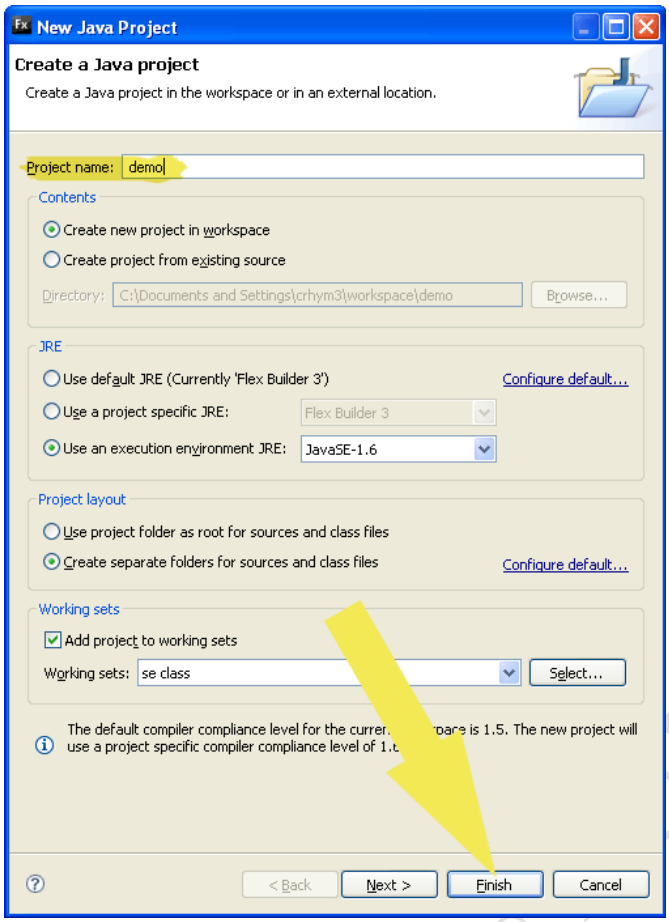
Statement	Description
fail(String)	Let the method fail, might be usable to check that a certain part of the code is not reached.
assertTrue(true);	True
assertsEquals([String message], expected, actual)	Test if the values are the same. Note: for arrays the reference is checked not the content of the arrays
assertsEquals([String message], expected, actual, tolerance)	Usage for float and double; the tolerance are the number of decimals which must be the same
assertNull([message], object)	Checks if the object is null
assertNotNull([message], object)	Check if the object is not null
assertSame([String], expected, actual)	Check if both variables refer to the same object

Statement	Description
assertNotSame([String], expected, actual)	Check that both variables refer not to the same object
assertTrue([message], boolean condition)	Check if the boolean condition is true.

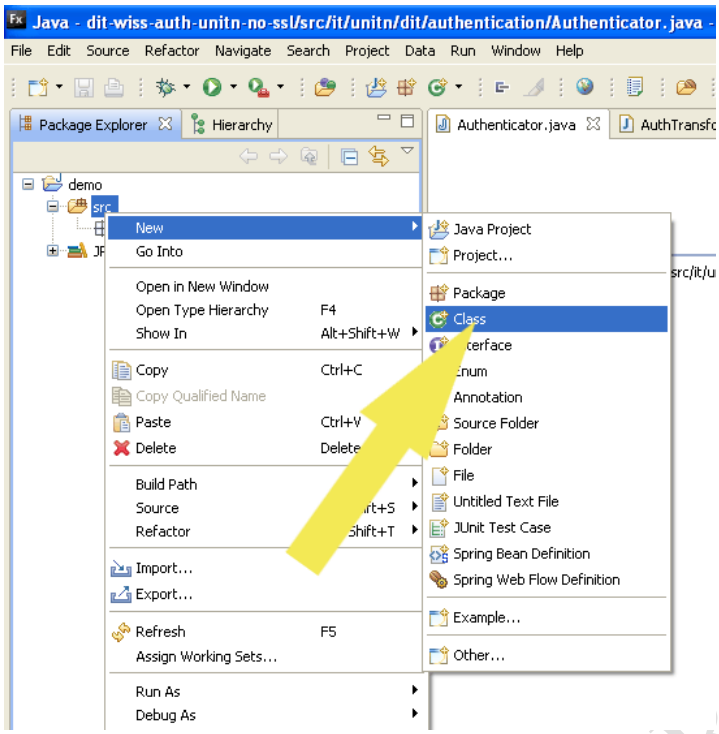
Working with MyEclipse

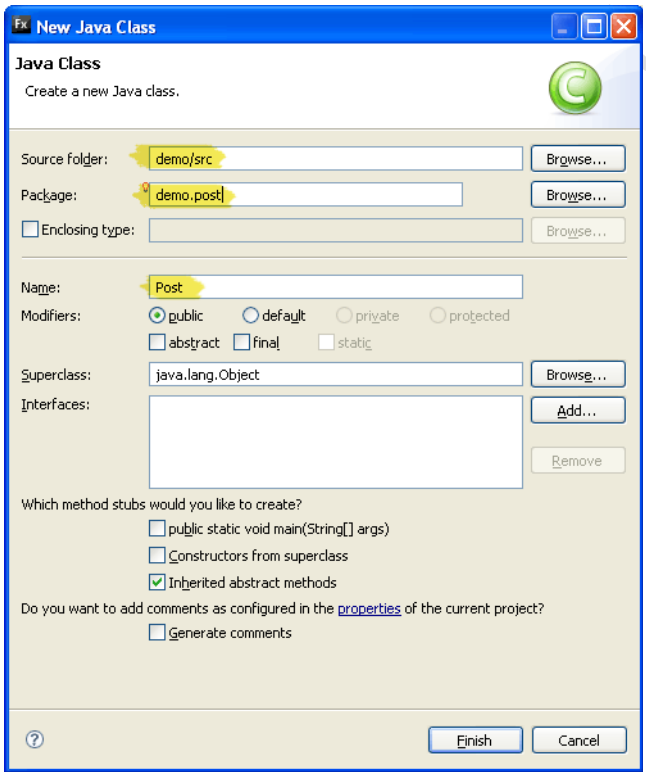


Create a Java Project

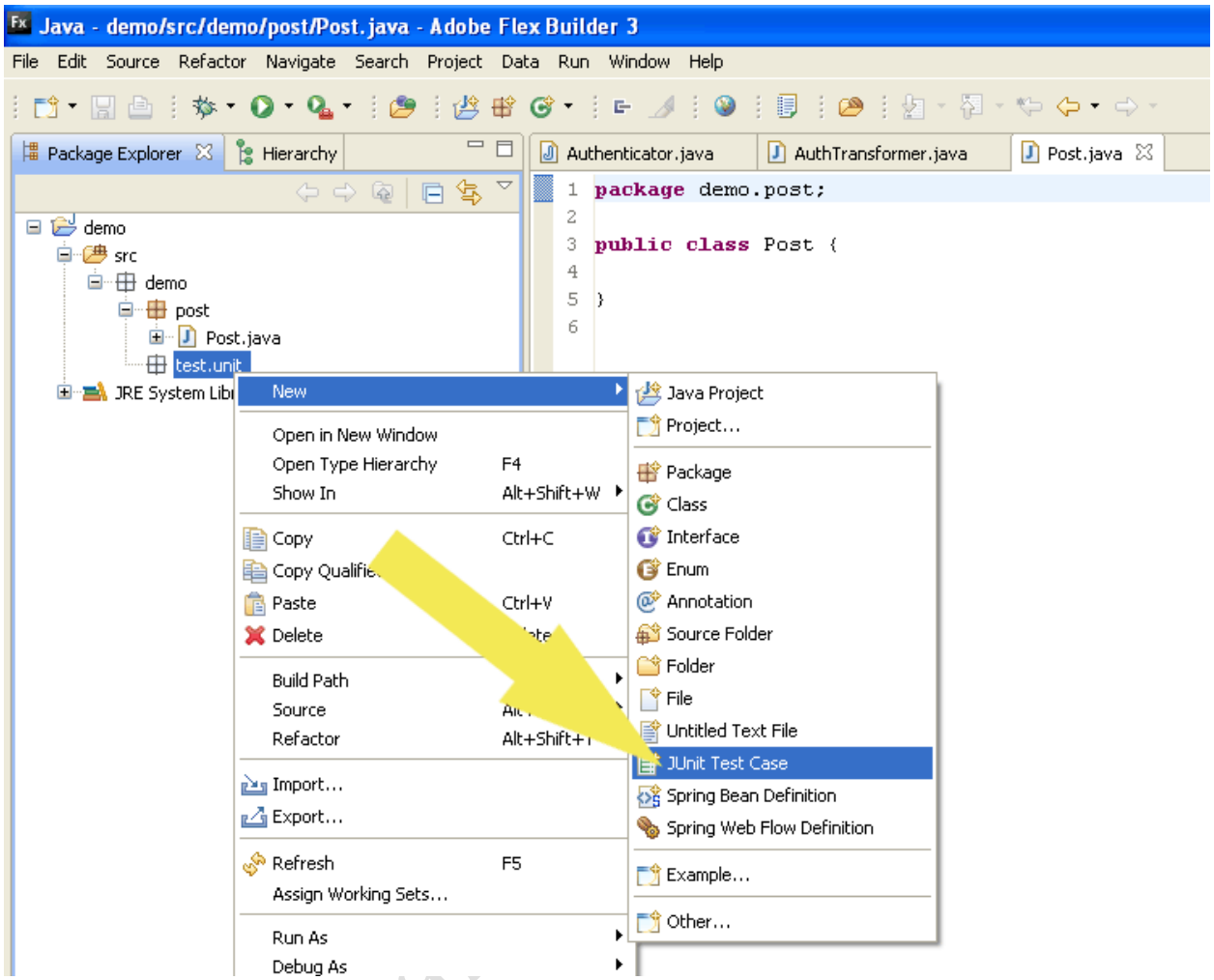


Lets Create a Model class



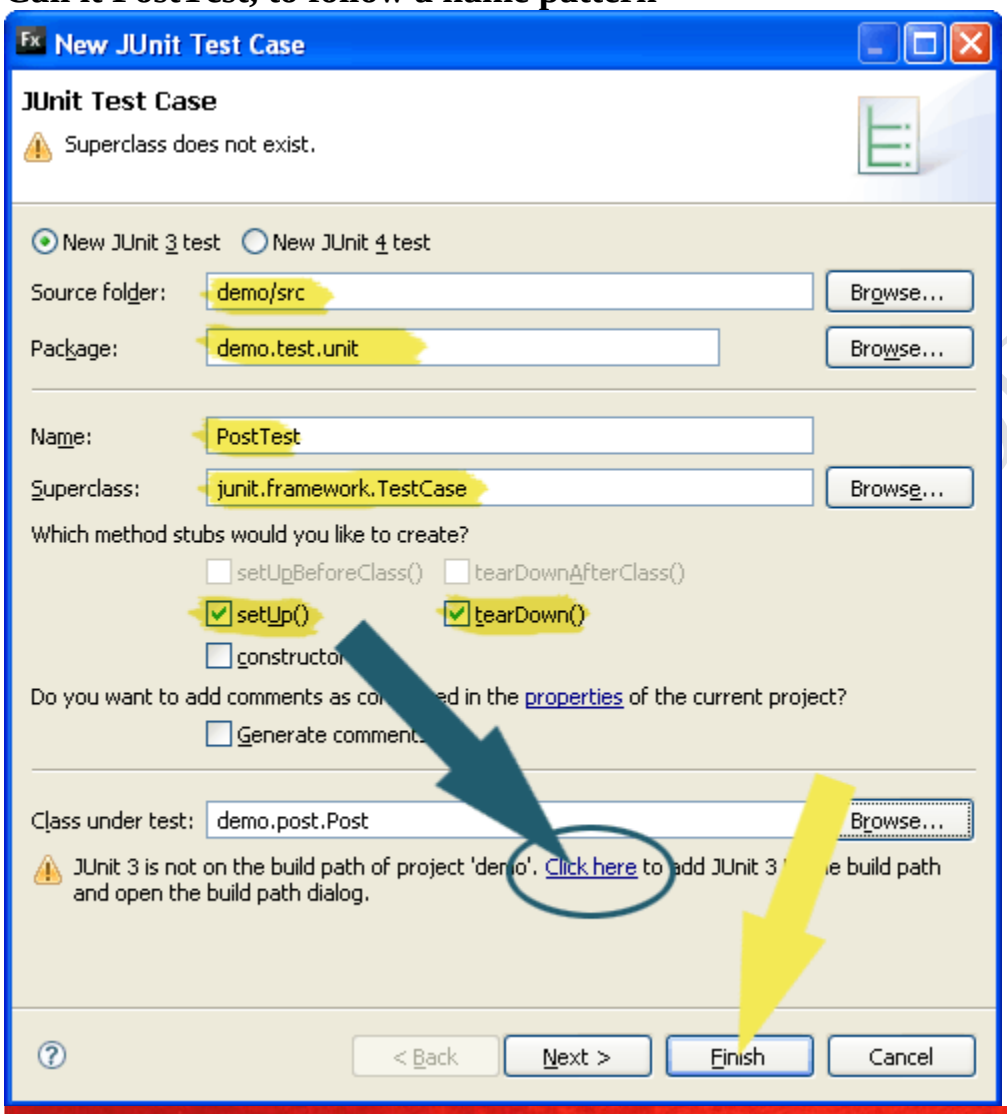


Add a JUNIT TestCase class


Java - demo/src/demo/post/Post.java - Adobe Flex Builder 3
File Edit Source Refactor Navigate Search Project Data Run Window Help
Package Explorer
Hierarchy
Authenticator.java
AuthTransformer.java
Post.java
package demo.post;
public class Post {
}
demo
src
demo
post
Post.java
test.unit
JRE System Libi
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete
Build Path
Source
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Run As
Debug As
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Folder
File
Untitled Text File
JUnit Test Case
Spring Bean Definition
Spring Web Flow Definition
Example...
Other...

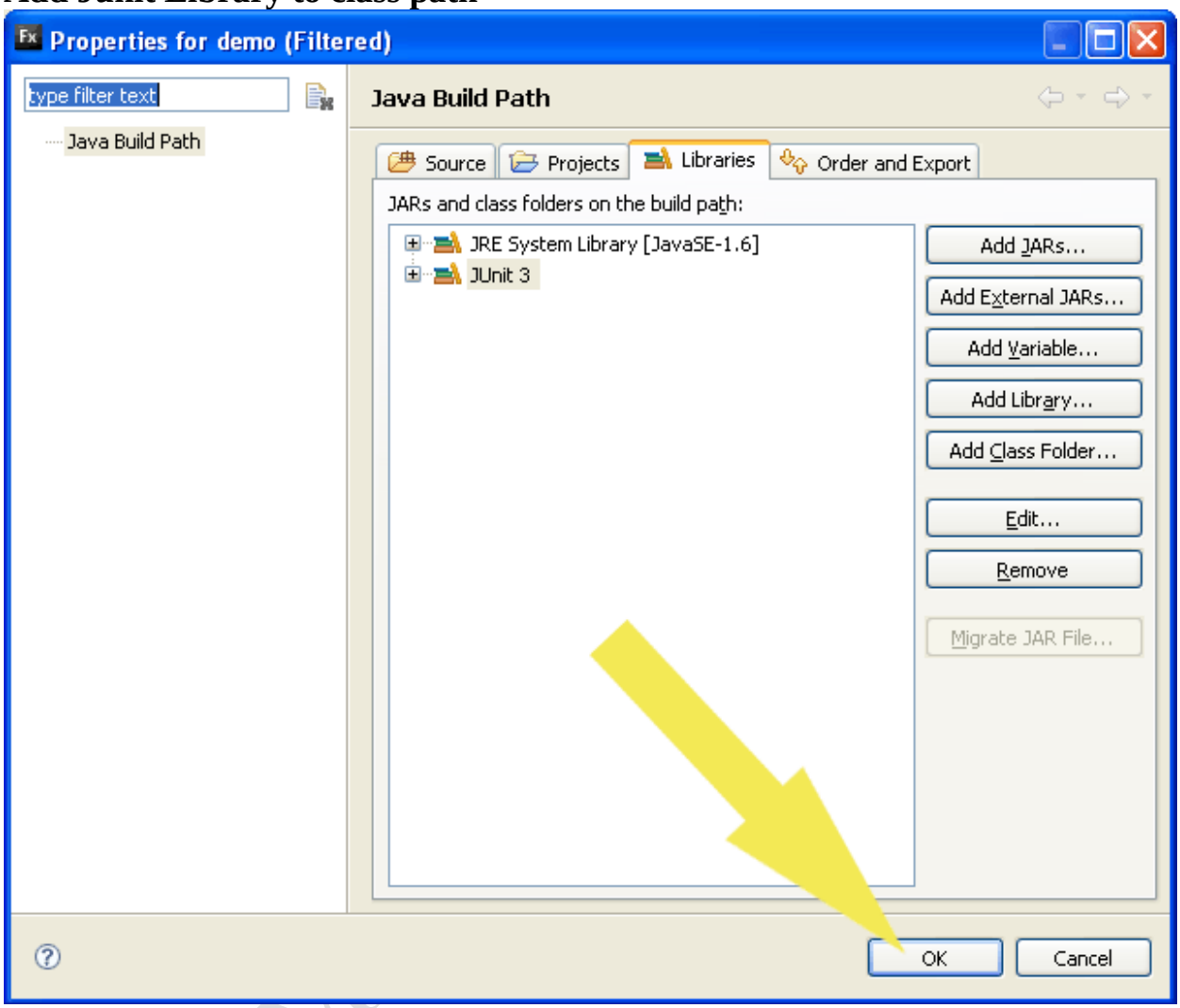
www.durgasoftonlinetraining.com
Online Training
Pre Recorded Video
Classes Training
Corporate Training
Ph: +91-8885252627, 7207212427
+91-7207212428
USA Ph : 4433326786
E-mail : durgasoftonlinetraining@gmail.com

Call it PostTest, to follow a name pattern

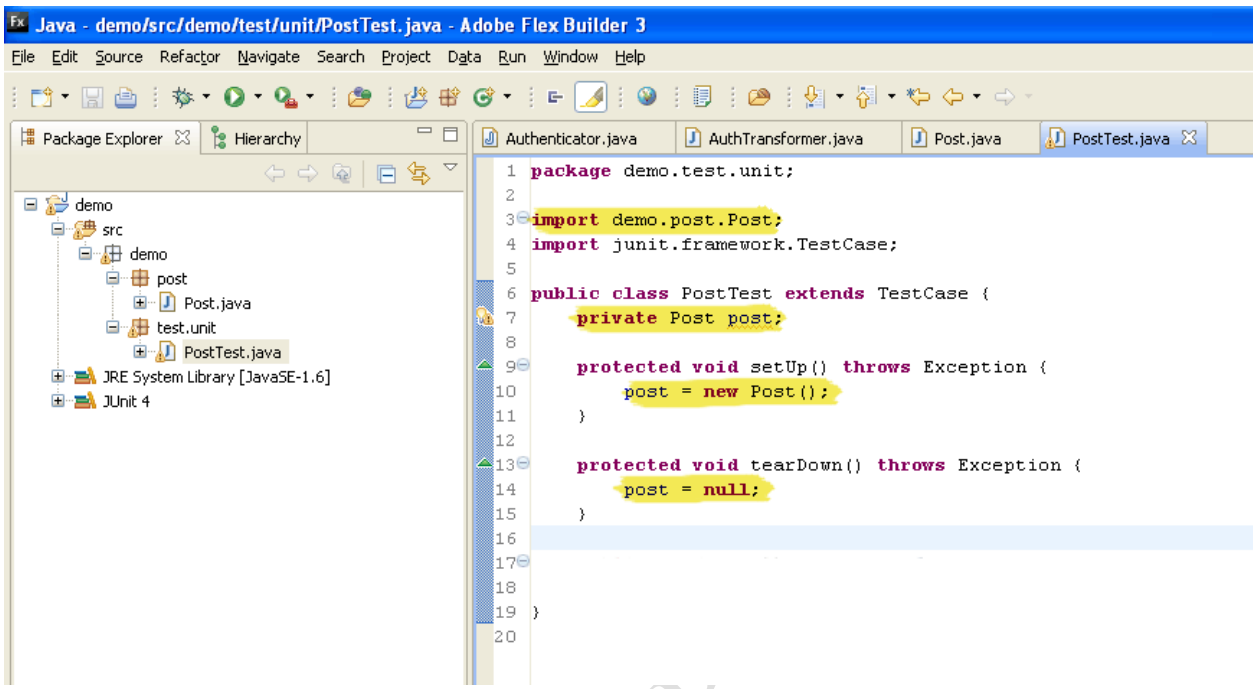


LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...
JAVA MEANS DURGASOFT
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE
AN ISO 9001:2008 CERTIFIED
DURGA SOFTWARE SOLUTIONS
#202 2nd FLOOR
www.durgasoft.com
040-64512786
+91 9246212143
+91 8096969696

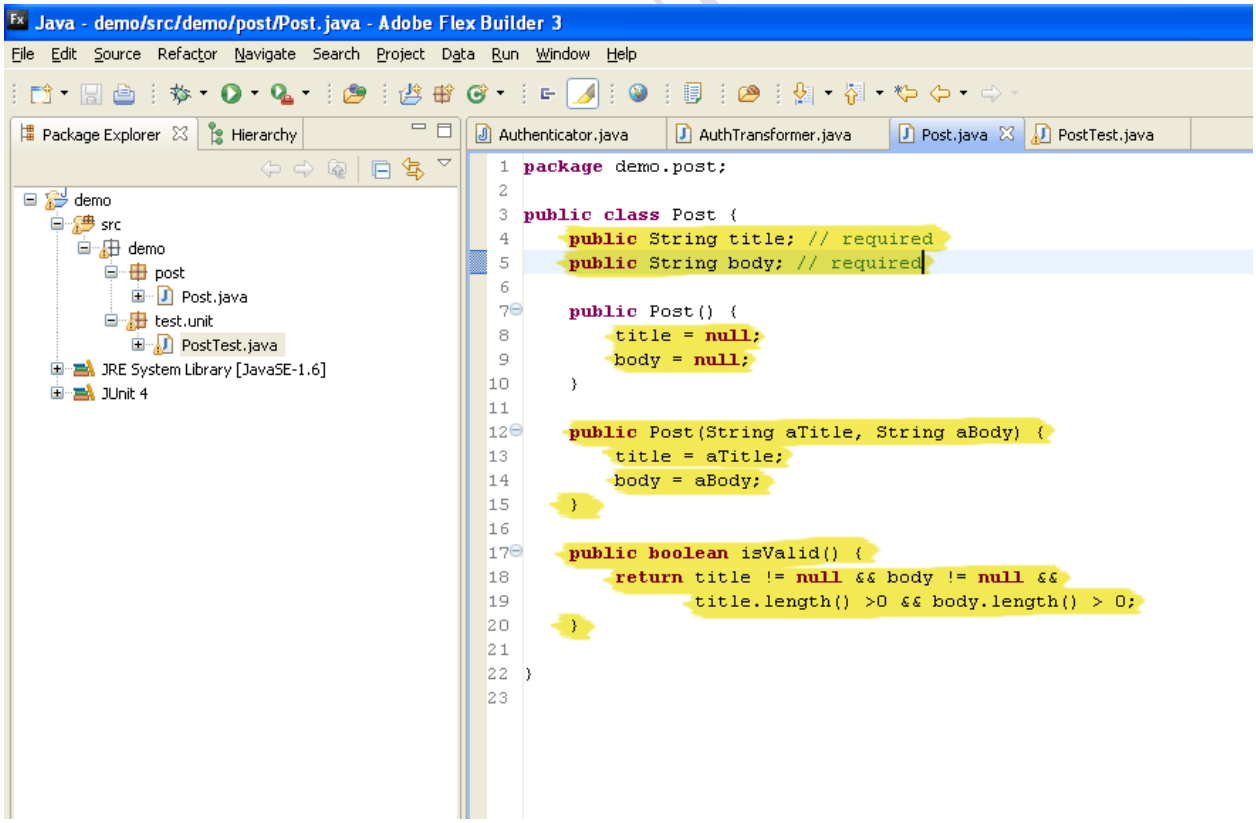
Add Junit Library to class path



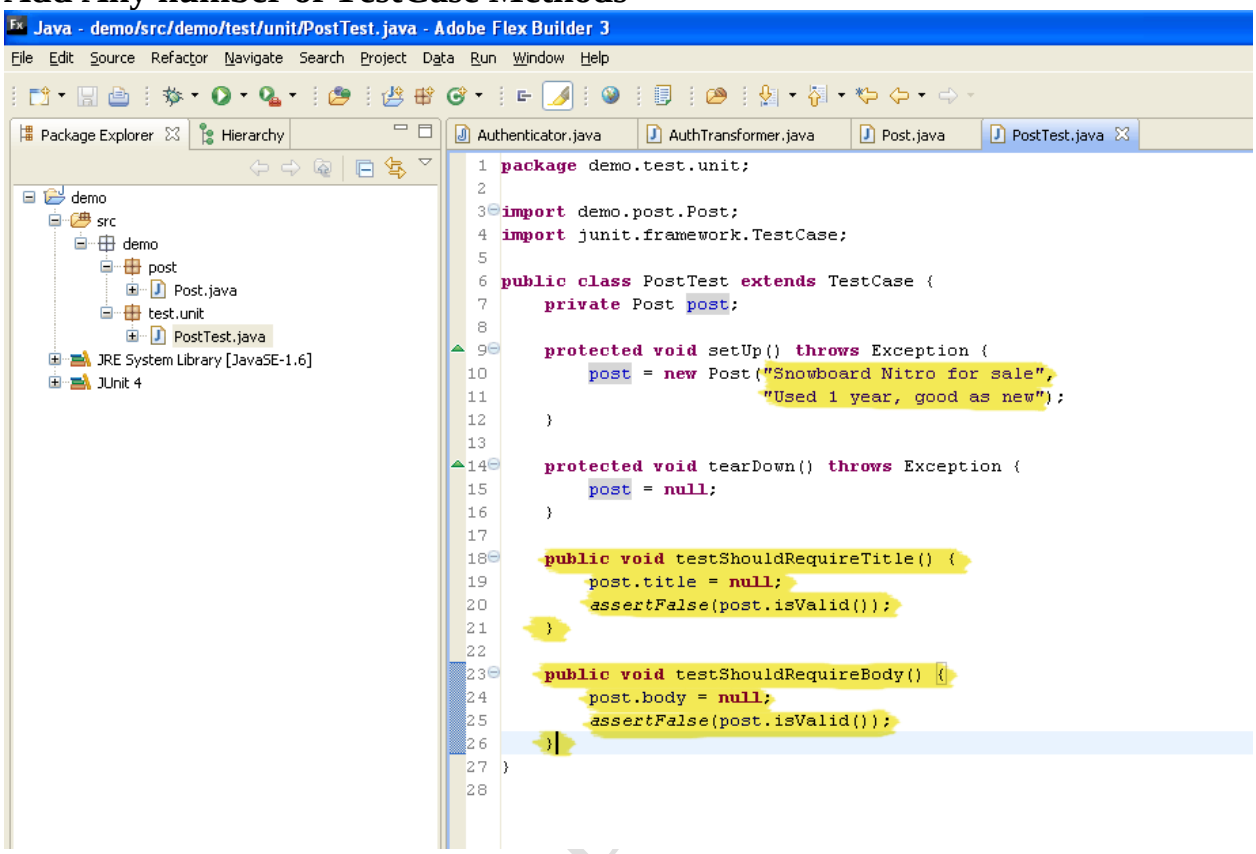
Write Your own TestCases



Test Your Functionalities

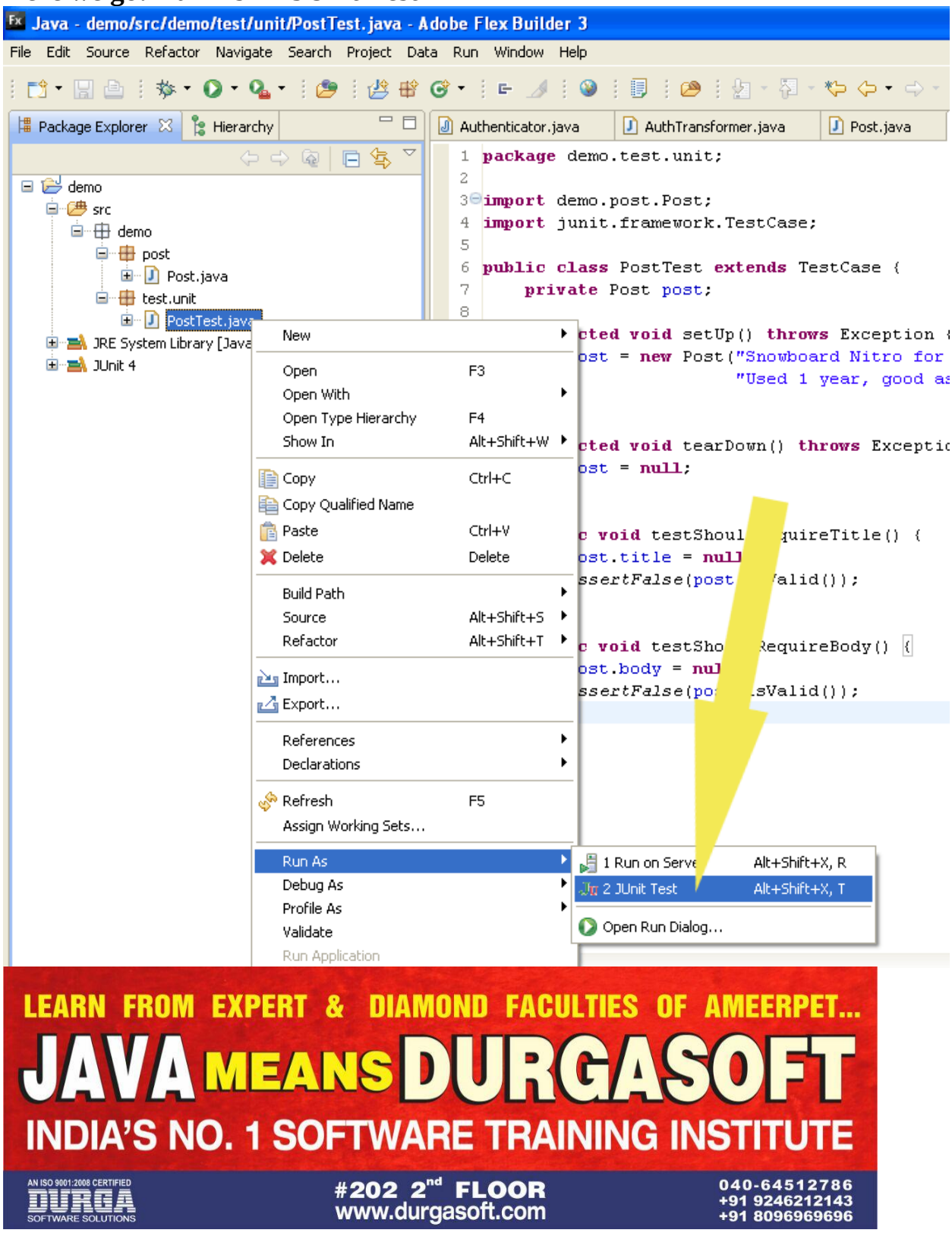


Add Any number of TestCase Methods



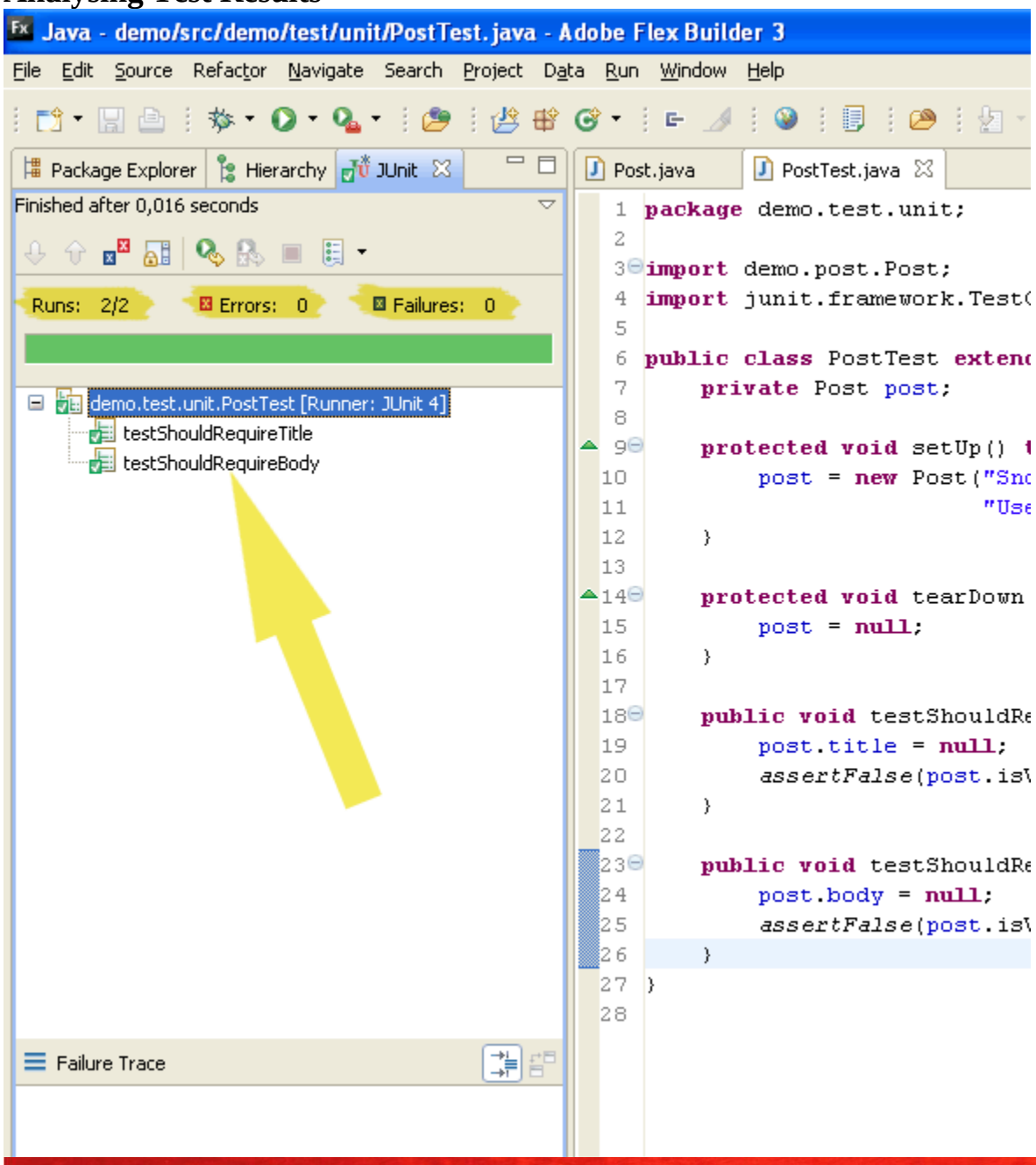
Runnig TestCase

Here we go: Run As => JUnit Test



LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...
JAVA MEANS DURGASOFT
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE
AN ISO 9001:2008 CERTIFIED DURGA SOFTWARE SOLUTIONS
#202 2nd FLOOR
www.durgasoft.com
040-64512786
+91 9246212143
+91 8096969696

Analysing Test Results



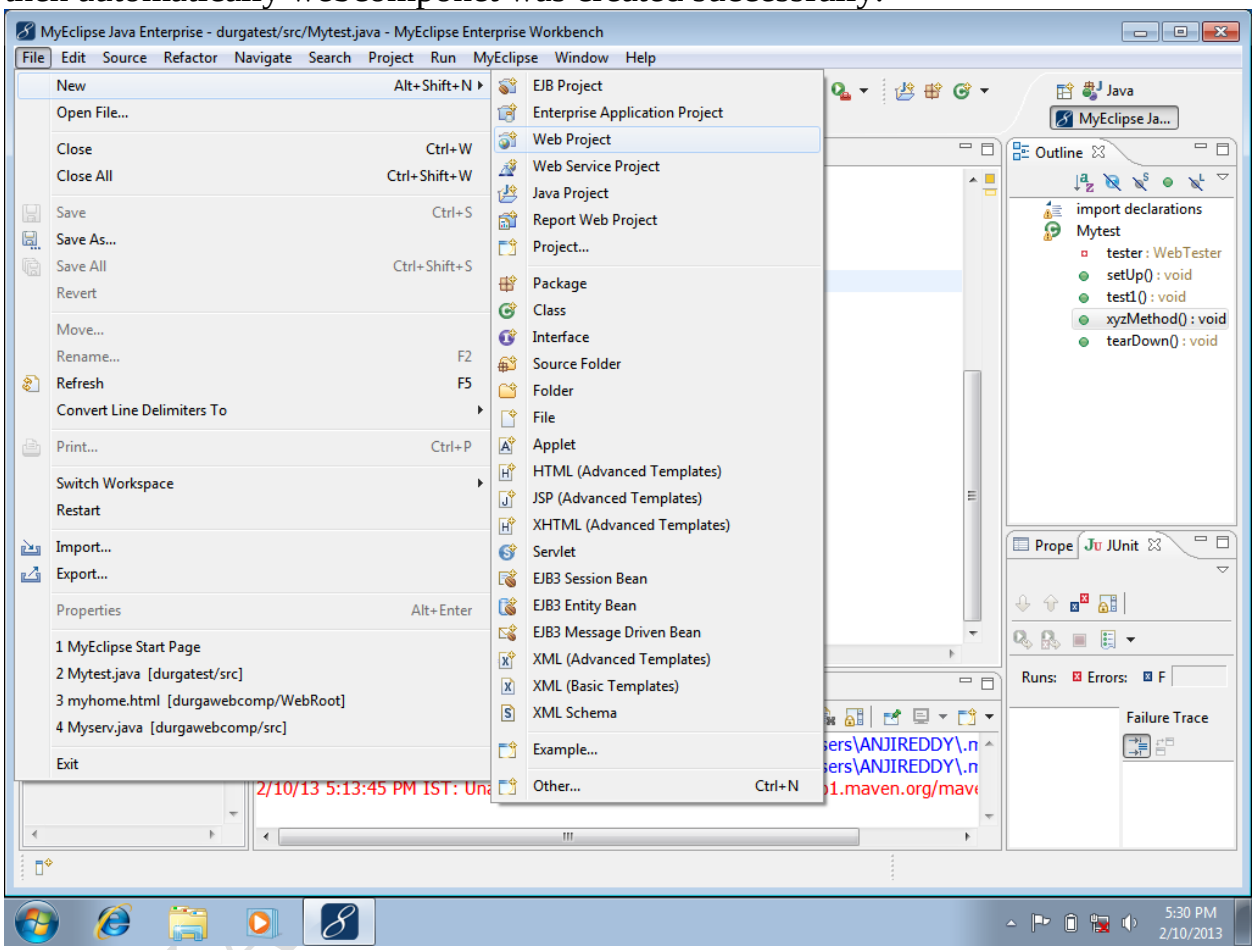
LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...
JAVA MEANS DURGASOFT
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE
AN ISO 9001:2008 CERTIFIED DURGA SOFTWARE SOLUTIONS
#202 2nd FLOOR
www.durgasoft.com
040-64512786
+91 9246212143
+91 8096969696

Junit webcomponentdevelopment and Testing Steps

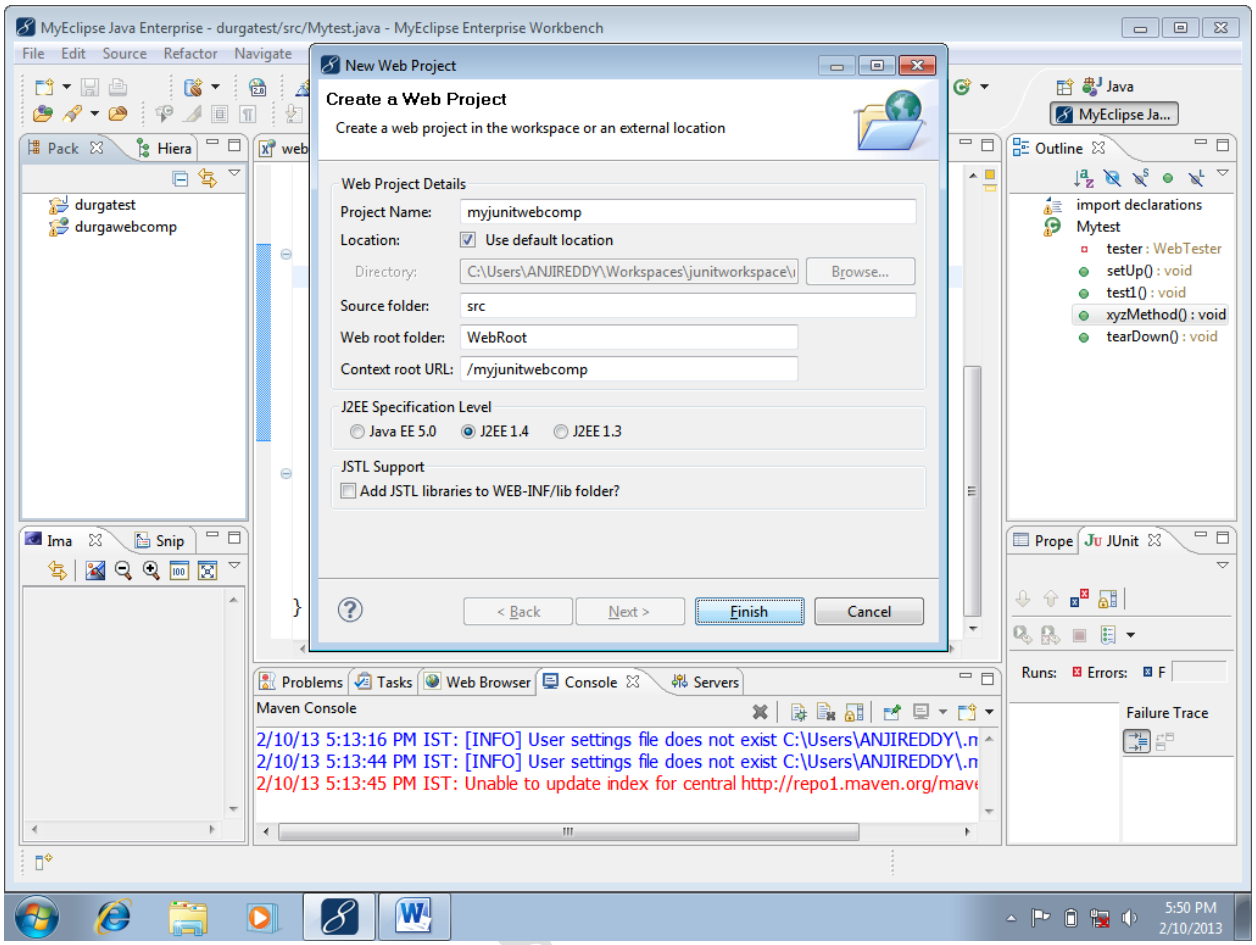
→First of all we need to create new work space(work space is nothing but a small floder)in MyEclipse IDE.

→File->New->webproject->enter the projectname(myjunitwebcomp)->ok->finish.

then automatically webcomponet was created successfully.

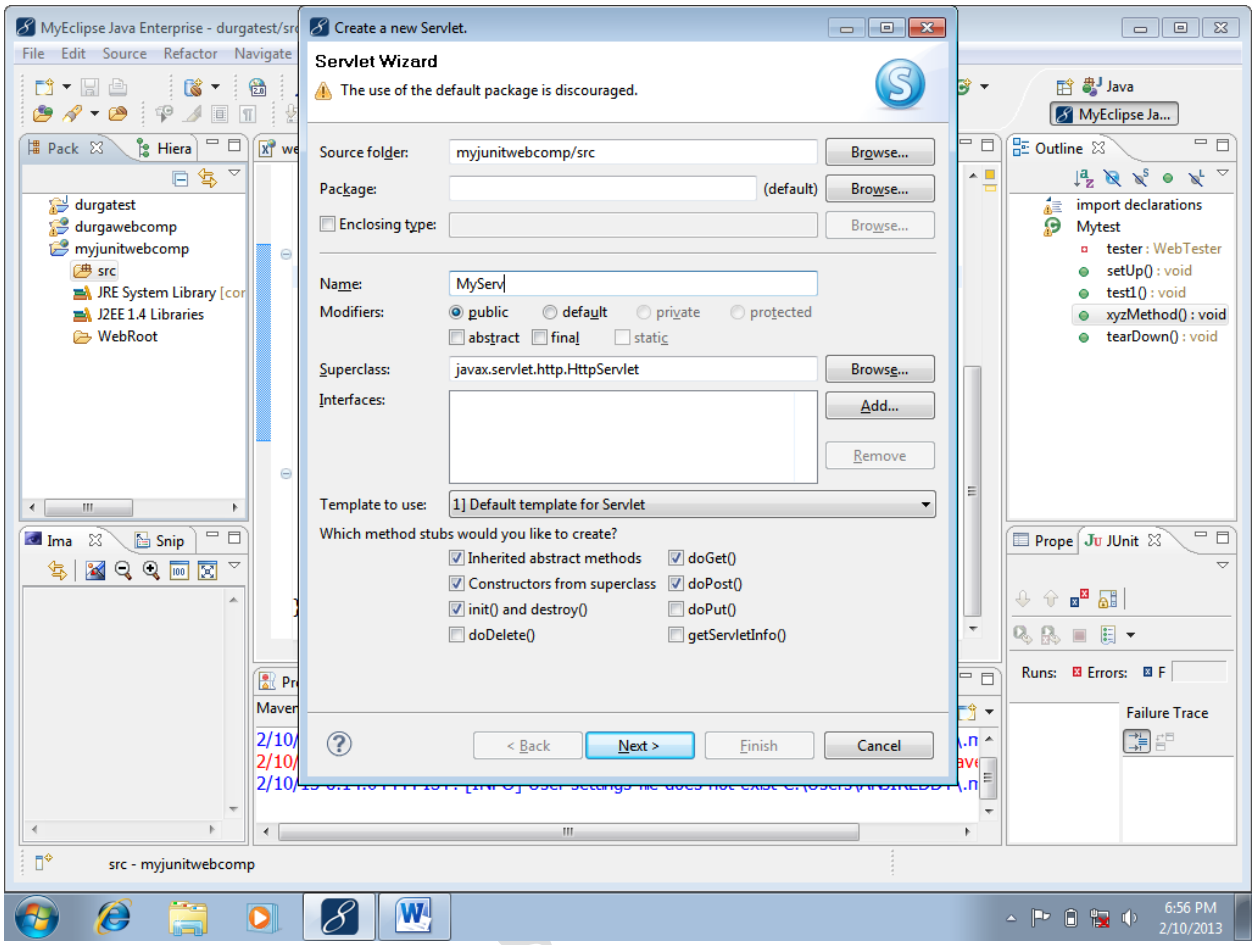


LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...
JAVA MEANS DURGASOFT
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE
AN ISO 9001:2008 CERTIFIED
DURGA SOFTWARE SOLUTIONS
#202 2nd FLOOR
www.durgasoft.com
040-64512786
+91 9246212143
+91 8096969696

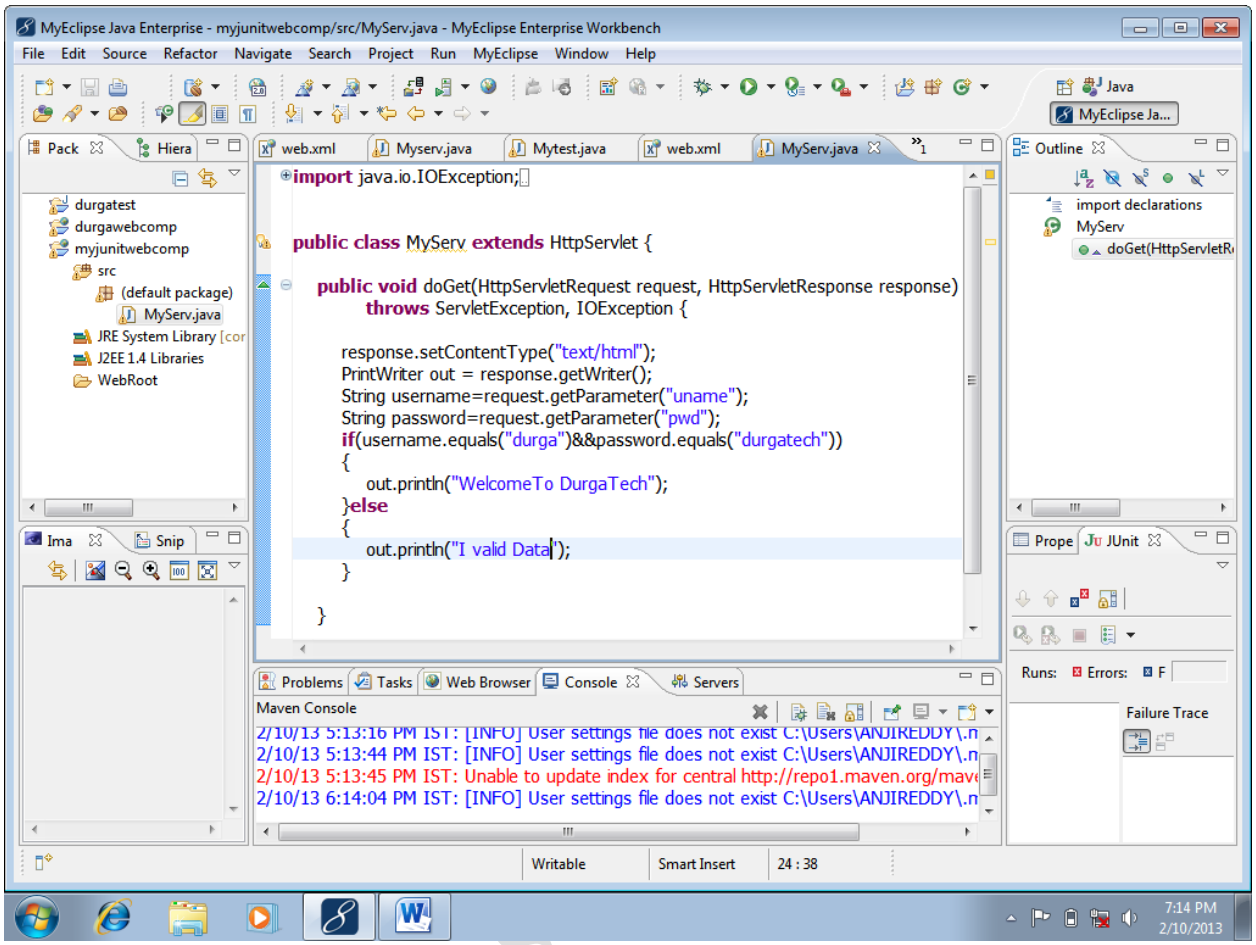


→Now we have to start project development code

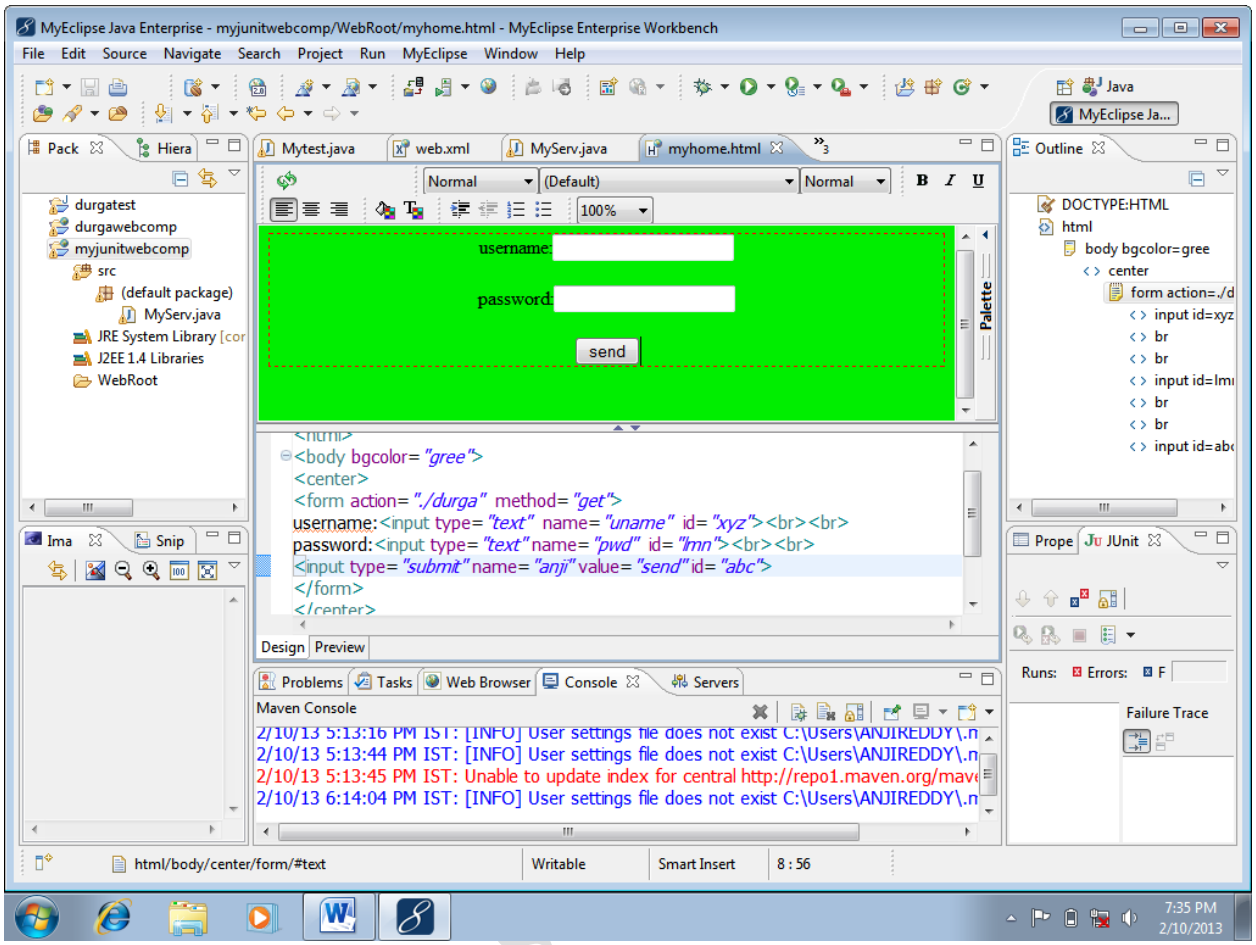
→Extactprojectfolder(myjunitwebcomp)->rightclick on src folder->and take one servlet class and we can develop servlet component code.


Create a new Servlet.
Servlet Wizard
The use of the default package is discouraged.
Source folder: myjunitwebcomp/src
Package:
Enclosing type:
Name: MyServ
Modifiers: public default private protected
abstract final static
Superclass: javax.servlet.http.HttpServlet
Interfaces:
Template to use: 1] Default template for Servlet
Which method stubs would you like to create?
Inherited abstract methods
Constructors from superclass
init() and destroy()
doDelete()
doGet()
doPost()
doPut()
getServletInfo()
< Back
Next >
Finish
Cancel
src - myjunitwebcomp
6:56 PM
2/10/2013

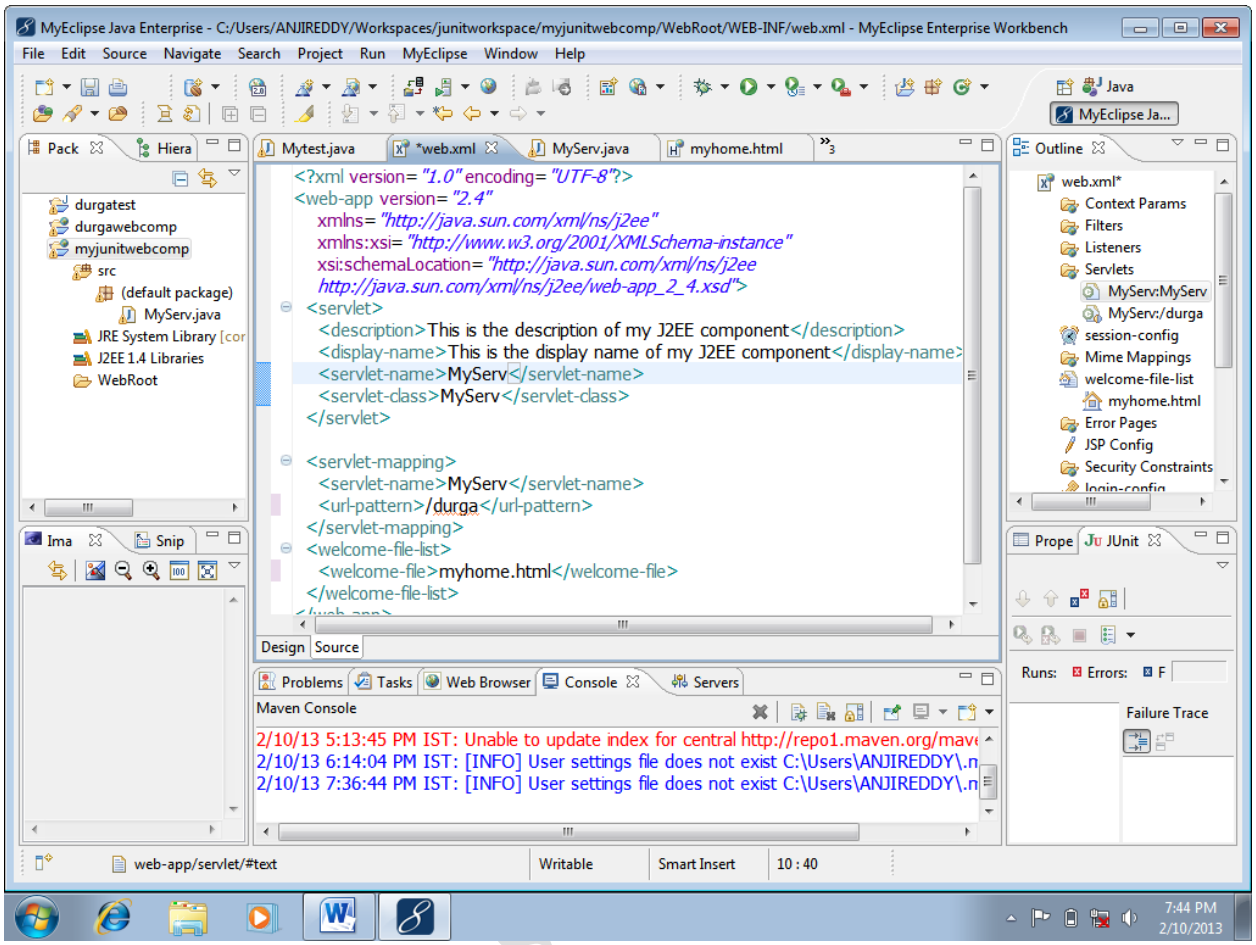
www.durgasoftonlinetraining.com
Online Training
Pre Recorded Video
Classes Training
Corporate Training
Ph: +91-8885252627, 7207212427
+91-7207212428
USA Ph : 4433326786
E-mail : durgasoftonlinetraining@gmail.com



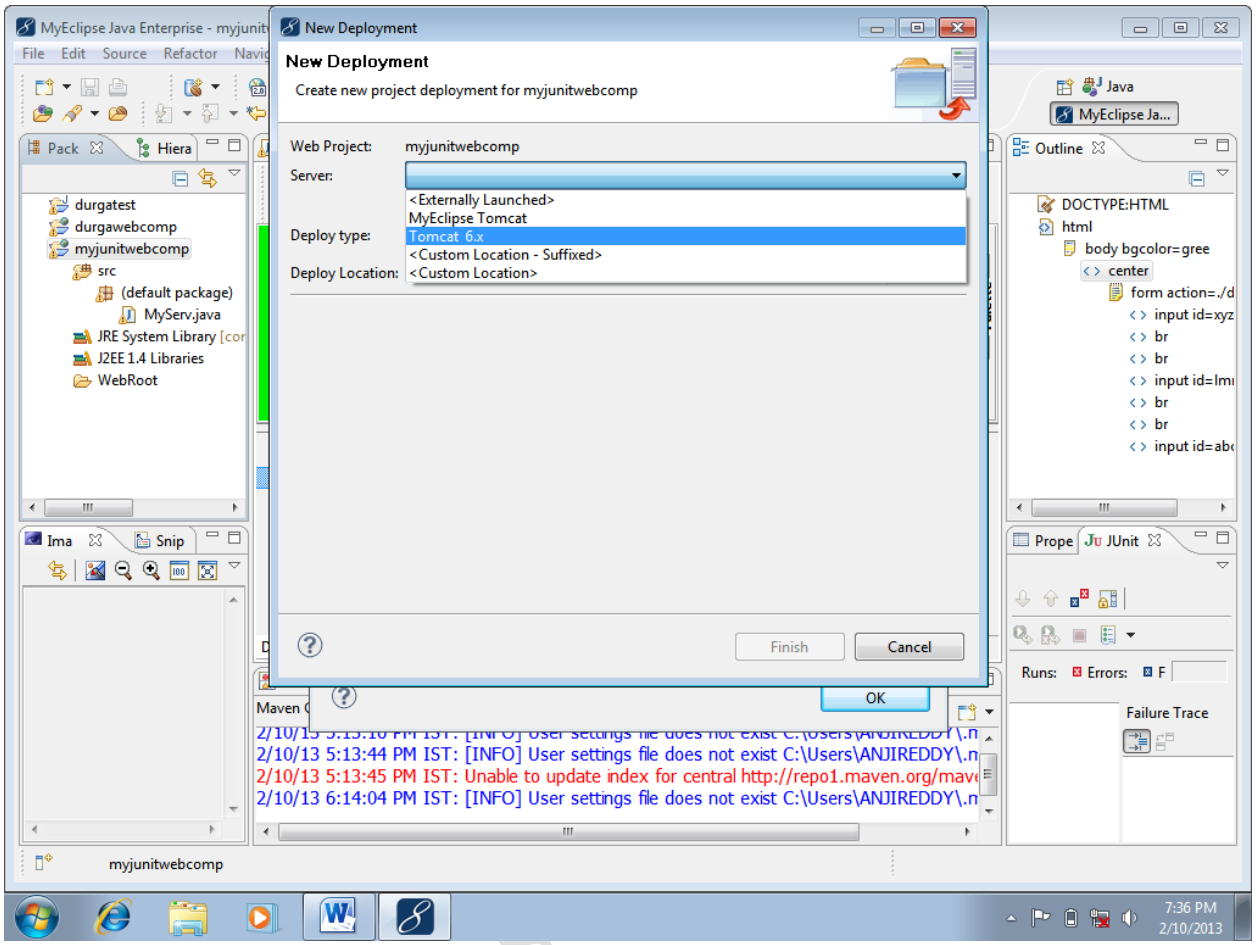
→Now by using html we can develop user details form
→Right click on projectfolder and create user details form and develop the code as follows.
→open the web.xml file and whater url we are passing in a myhome.html same url we can use in web.xml file also
→and finally configure the server and deployee the web application into server
→Now start the server, firt of all test the application manuvally then it is success.
→After that we can go for junit testing and develop the testcases of the webcomponent



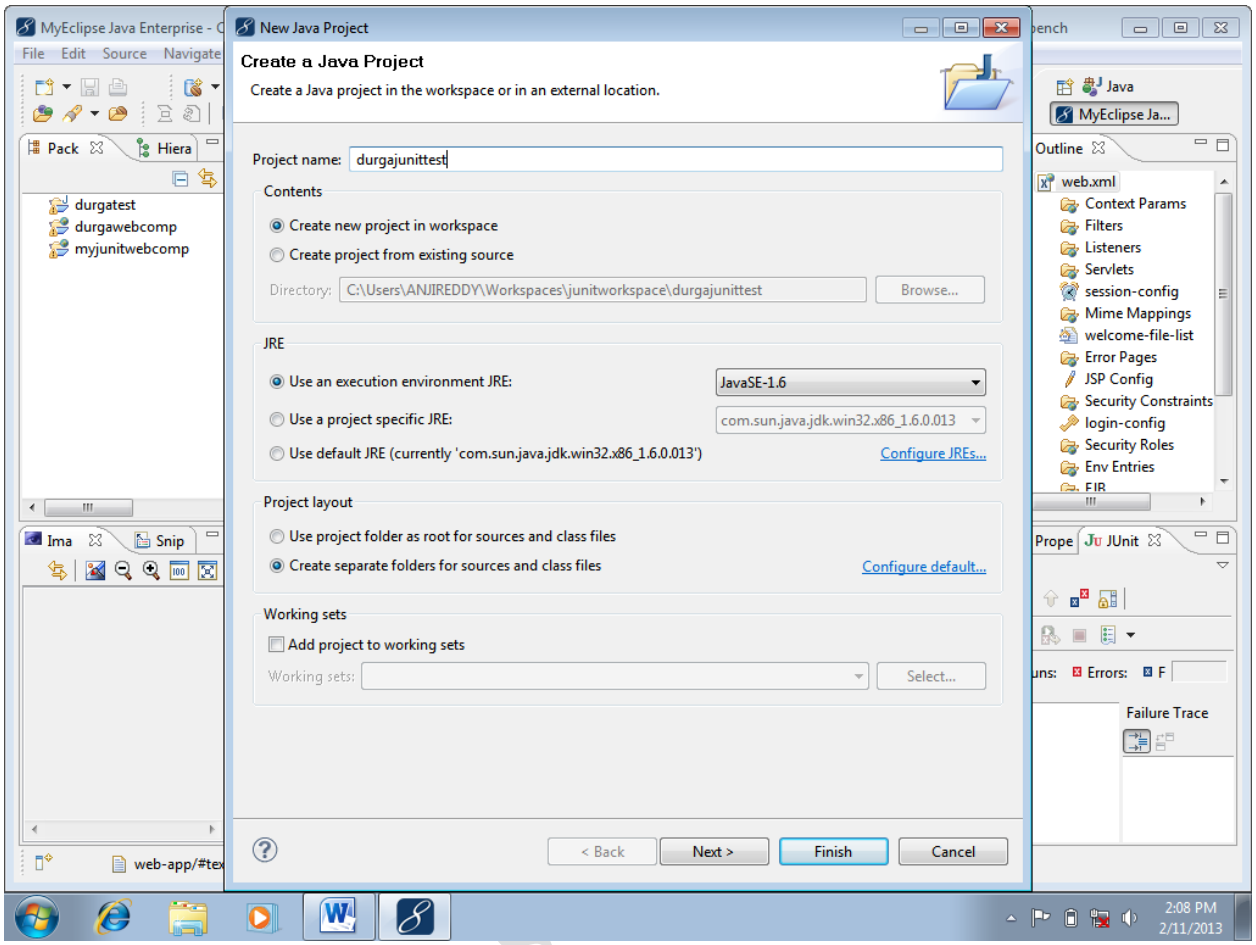
www.durgajobs.com
Continuous Job Updates for every hour
Fresher Jobs
Govt Jobs
Bank Jobs
Walk-ins
Placement Papers
IT Jobs
Interview Experiences
Complete Job information across India


MyEclipse Java Enterprise - C:/Users/ANJIREDDY/Workspaces/junitworkspace/myjunitwebcomp/WebRoot/WEB-INF/web.xml - MyEclipse Enterprise Workbench
File Edit Source Navigate Search Project Run MyEclipse Window Help
Mytest.java
*web.xml
MyServ.java
myhome.html
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<servlet>
<description>This is the description of my J2EE component</description>
<display-name>This is the display name of my J2EE component</display-name>
<servlet-name>MyServ</servlet-name>
<servlet-class>MyServ</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServ</servlet-name>
<url-pattern>/durga</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>myhome.html</welcome-file>
</welcome-file-list>
Design Source
Problems Tasks Web Browser Console Servers
Maven Console
2/10/13 5:13:45 PM IST: Unable to update index for central http://repo1.maven.org/mave
2/10/13 6:14:04 PM IST: [INFO] User settings file does not exist C:\Users\ANJIREDDY\.n
2/10/13 7:36:44 PM IST: [INFO] User settings file does not exist C:\Users\ANJIREDDY\.n
Outline
web.xml*
Context Params
Filters
Listeners
Servlets
MyServ:MyServ
MyServ:/durga
session-config
Mime Mappings
welcome-file-list
myhome.html
Error Pages
JSP Config
Security Constraints
Runs: Errors:
Failure Trace
web-app/servlet/#text
Writable
Smart Insert
10 : 40
7:44 PM
2/10/2013

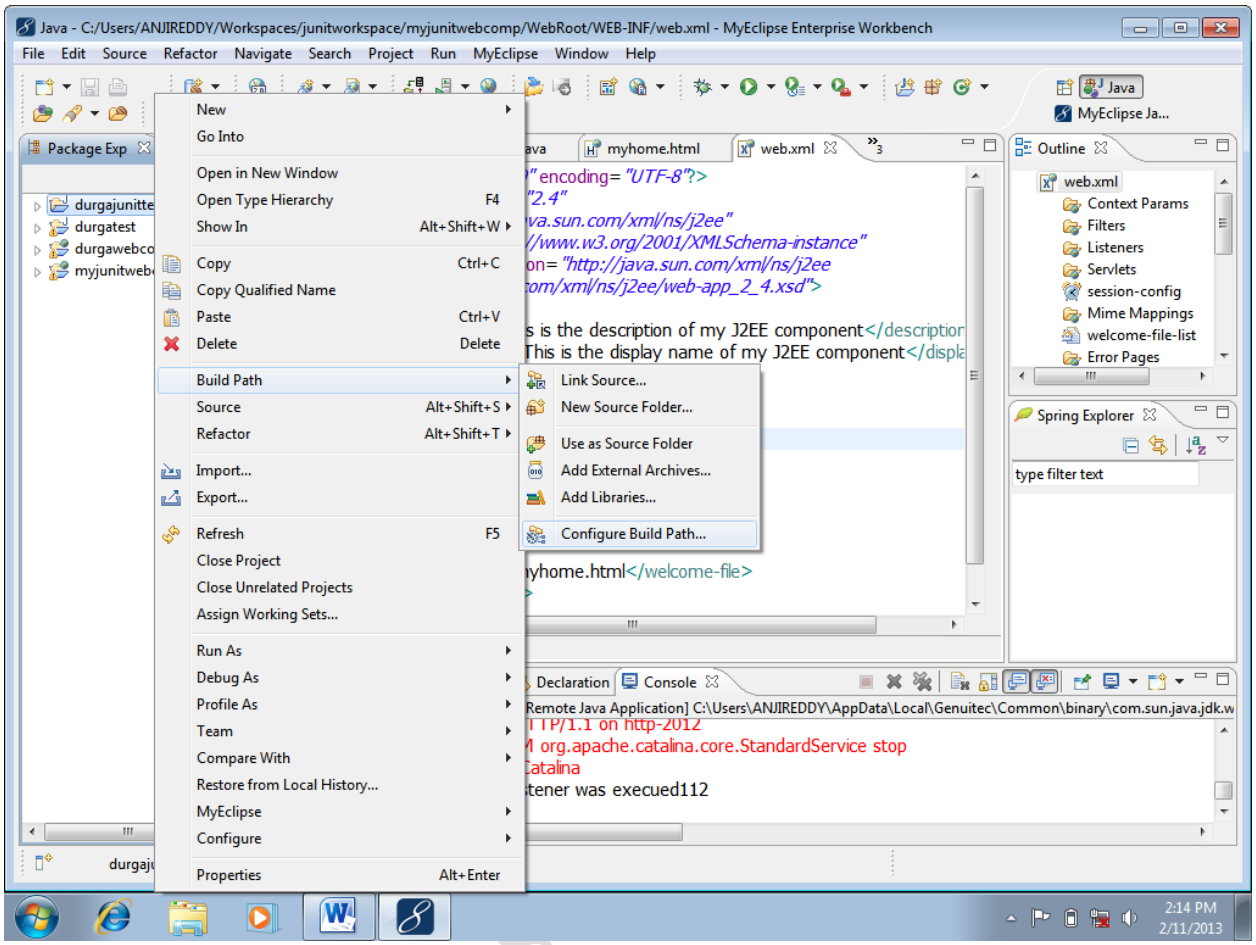
FREE TRAINING VIDEOS
YouTube
3000+
VIDEOS
www.youtube.com/durgasoftware



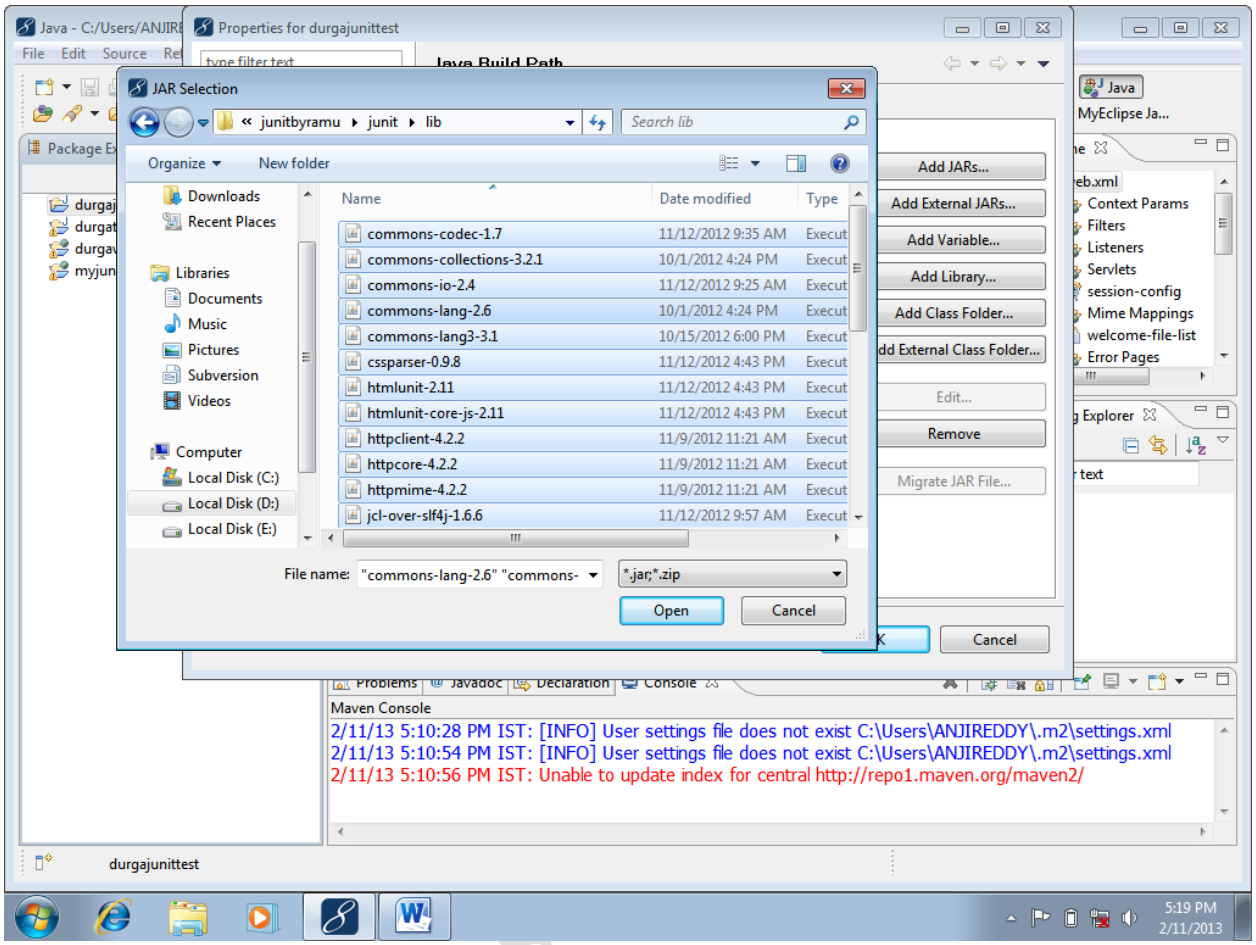
→Now we have to start junit component
→create normal java project and add all the junit related jar files as followes
→file->new->javaproject->enterprojectname(ex:durgajunitproject)->next->finish.
→right click on project->buildpath->cofigurebuildpath->addlibraries ->addexternaljarfiles->add all jar files related to junit.


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: durgajunittest
Contents
Create new project in workspace
Create project from existing source
Directory: C:\Users\ANJIREDDY\Workspaces\junitworkspace\durgajunittest
Browse...
JRE
Use an execution environment JRE: JavaSE-1.6
Use a project specific JRE: com.sun.java.jdk.win32.x86_1.6.0.013
Use default JRE (currently 'com.sun.java.jdk.win32.x86_1.6.0.013')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel
durgatest
durgawebcomp
myjunitwebcomp
web.xml
Context Params
Filters
Listeners
Servlets
session-config
Mime Mappings
welcome-file-list
Error Pages
JSP Config
Security Constraints
login-config
Security Roles
Env Entries
Failure Trace
2:08 PM
2/11/2013

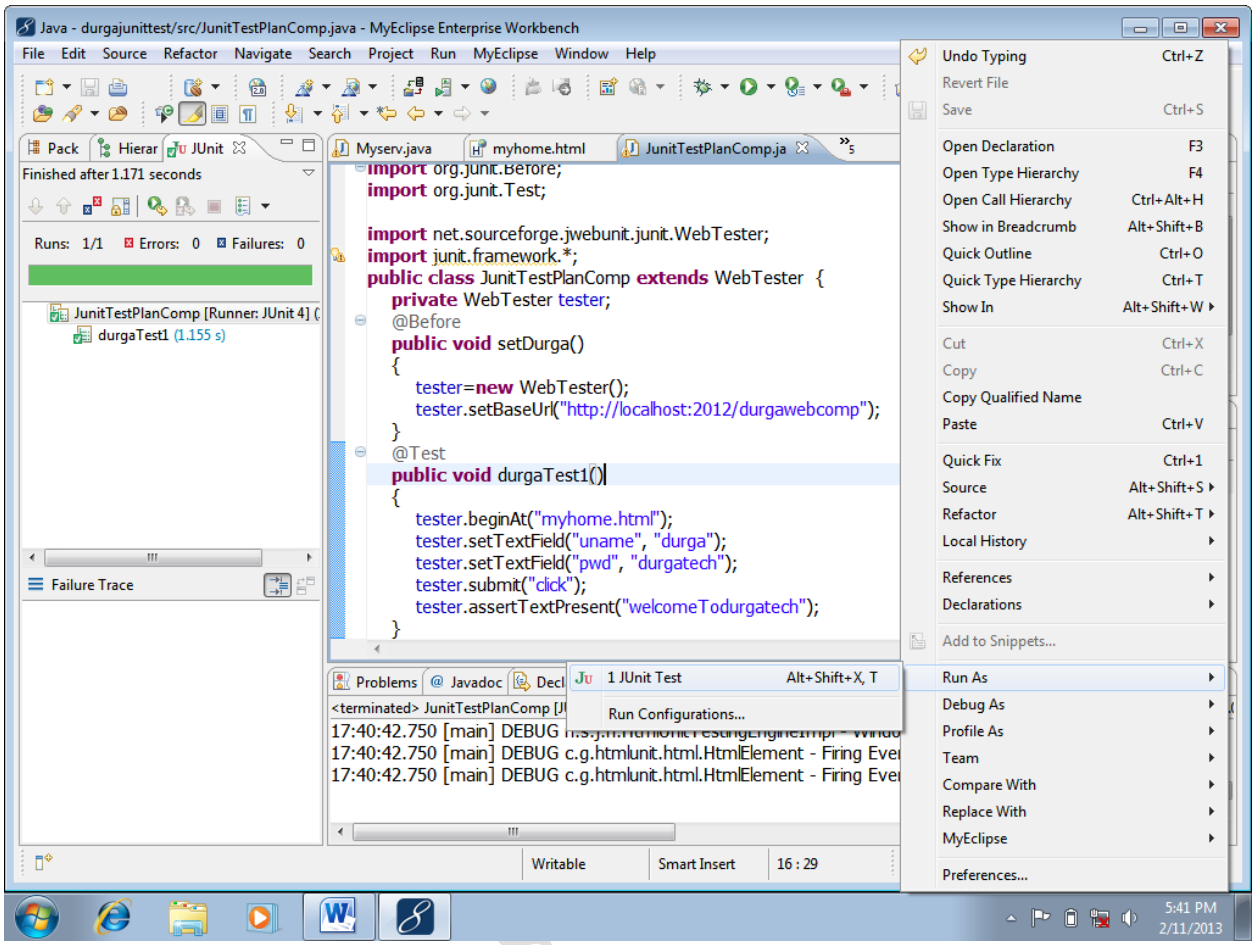
www.durgasoftonlinetraining.com
Online Training
Pre Recorded Video
Classes Training
Corporate Training
Ph: +91-8885252627, 7207212427
+91-7207212428
USA Ph : 4433326786
E-mail : durgasoftonlinetraining@gmail.com


Java - C:/Users/ANJIREDDY/Workspaces/junitworkspace/myjunitwebcomp/WebRoot/WEB-INF/web.xml - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project Run MyEclipse Window Help
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
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
MyEclipse
Configure
Properties Alt+Enter
Link Source...
New Source Folder...
Use as Source Folder
Add External Archives...
Add Libraries...
Configure Build Path...
Package Exp
myhome.html
web.xml
Outline
Context Params
Filters
Listeners
Servlets
session-config
Mime Mappings
welcome-file-list
Error Pages
Spring Explorer
type filter text
Declaration
Console
org.apache.catalina.core.StandardService stop
2:14 PM
2/11/2013

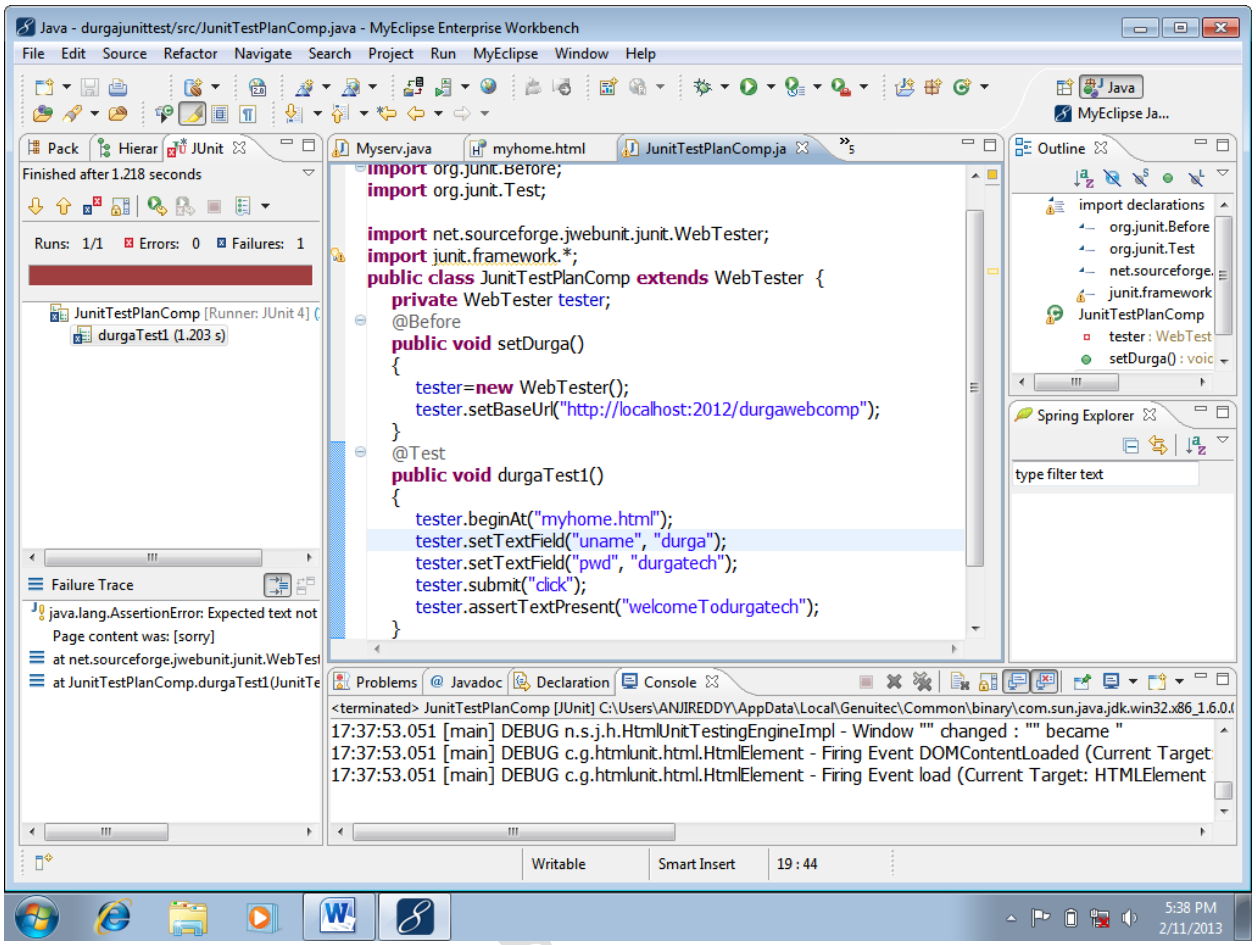
FREE TRAINING VIDEOS
YouTube
3000+
VIDEOS
www.youtube.com/durgasoftware



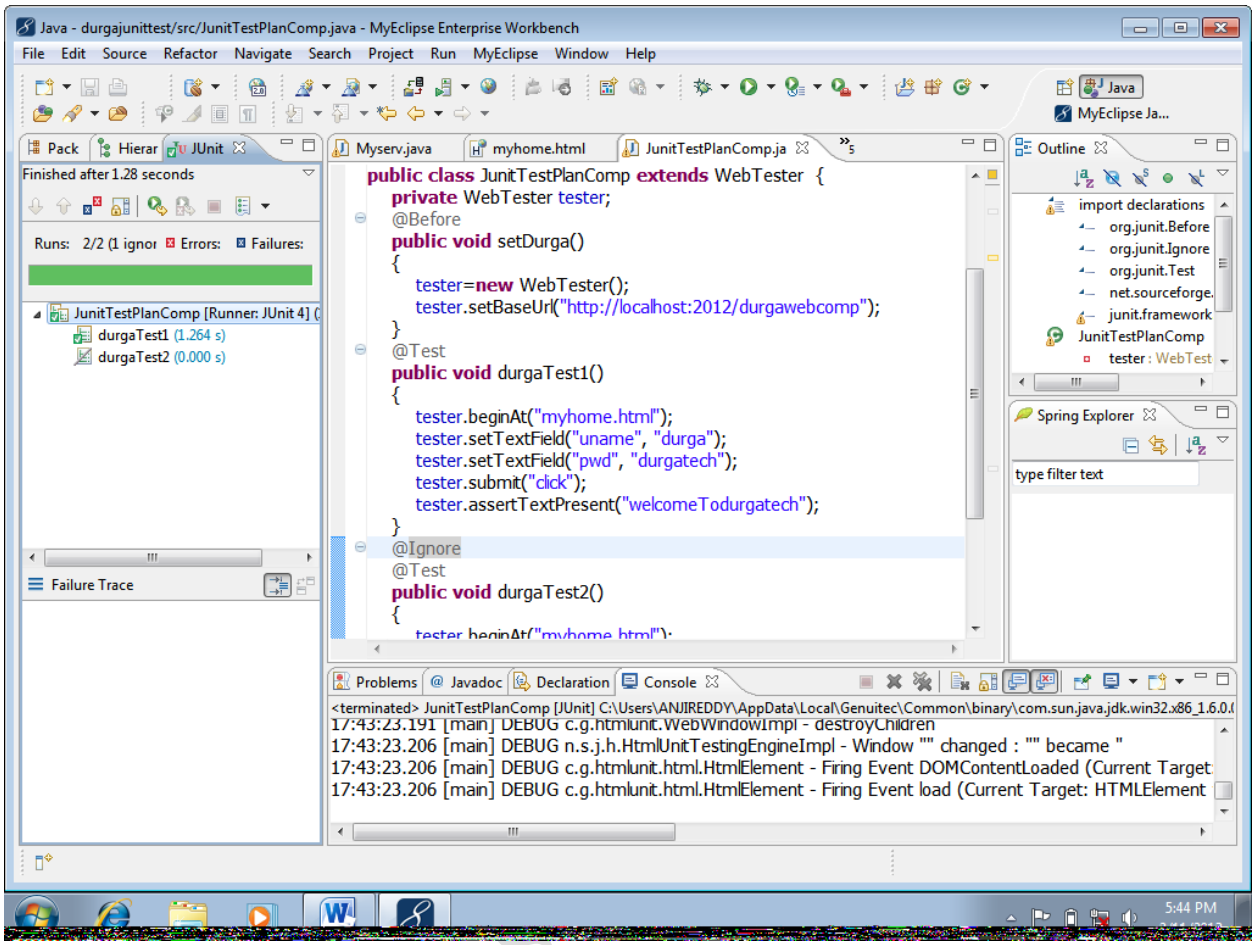
→now test the application.



www.durgajobs.com
Continuous Job Updates for every hour
Fresher Jobs
Govt Jobs
Bank Jobs
Walk-ins
Placement Papers
IT Jobs
Interview Experiences
Complete Job information across India



The above case is failure case.



This is an Ignore case

LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com