JUNIT Testing

Junit is the java unit testing framework which used for testing of the individual units or components which are actually independent classes. The version in currently in use is Junit 4 which makes use of the Java 5 annotations and contains additional features compared to Junit 3.

Using Junit with SDE

Make sure that the junit jar is in the buildpath of your project.

Inorder to add the junit jar to the build path,

Right click on the project, select build path→ select add libraries →select Junit→ select Junit library version (Select 4) and click finish.

You have successfully added Junit library in your build path.

Creating Test Cases

Manual Code Generation method

1. Select a class for which we wish to create test case class.
   Right click→New→ Junit test case →select methods for which test case to be created → finish.
2. A test case class will be created with the test case methods for the methods you have selected.
3. Apply proper coding for the methods by using the various assertXXXX methods for the output matching.
4. Run the TestCase class by selecting the Runas Junit test.
5. Test results will be displayed in the Junit window.

Automatic Test Code Generation

The automatic test code generation is achieved with help of Junit TestGen plugin.

Using the Junit Code generation Plugin we have the choice for generating the test case for the following modes

1. Simple test case: Generate a testcase class with simple test case methods for the methods for the methods.
2. Boundary condition test case: Generate the testcase class with methods for testing with the boundary conditions for the method arguments like the maximum,minimum as inputs for the numeric type arguments .

3. Full coverage test case: In this case along with the boundary condition test cases, different test case methods will be created for testing the code through the entire looping or branching statements. For e.g.: if we have if else block we have to test the code with values satisfying the if condition as well as not satisfying. In that case only we can ensure the testing through the entire code written in if as well as else block. For this purpose we use the full coverage test codes which contains n number of functions depending on the conditional constructs used.

To achieve this automatic generation, make sure the Testgen2.0.0 plugin is added in eclipse.

How to generate the code?

Right click from inside the class for which you want to generate the testcase class.

- ***Source→Generate Junit test***

You will be getting window showing the options for the above mentioned types. Select the appropriate type and click finish.

Once the test class is created you can make the required modifications in accordance with your assertions to be justified.
After making all the required changes, run the testclass and verify the output in the Junit window.

Code Coverage Analysis

Code Coverage analysis is an important measure used along with unit testing. It gives an analysis of the percentage of code covered during the unit testing.

Coverage Analysis tools

They are the tools which are used for automated analysis of the code coverage during a particular unit testing. Some of the tools are EMMA, Cobertura,Quilt etc.

EclEmma

EclEmma is a free Java code coverage tool for Eclipse, available under the Eclipse Public License. Internally it is based on the great EMMA Java code coverage tool.

How to Use Eclemma?

Select the test class and click the Launch icon select → Coverage as → Junit test for checking the coverage of a particular test class.

You will be getting the coverage details in the coverage window. Coverage reports can also be generated.

Theory Hands on Junit using testgen plug in

Standalone Application:

There are two components to be tested. Use the TestGen plugin for generating the testcase class and methods. Suitable test methods may be added as per the requirement. Unwanted methods may be knocked off and input parameters may be modified if required.

The theory Exercises are placed inside the package "com.ccd.junit.theory"

Trainers can first show case the calculator example to explain about how to use test gen plugin and they can use the telephone Bill calculator to explain about how to write junits to cover if else statements and loops. You can use the same junits developed to show how to use Emma to measure test coverage.

a. **SimpleCalculato**r component having methods for addition, subtraction, division and multiplication. This component has to be performed simple testing.

a. Solution:

b. **TelephoneBillCalculator** component have methods for calculating the telephone bill of the customers. Here the methods are

1. **double calculateBill(Customer cust, int netReading)**
   Calculates the telephone bill of the customer based on the customer details, and the current reading. This method has to be tested with simple test values, boundary condition and for full code coverage.
2. **ArrayList getSeniorcitizenList(List<Customer> customerList)**
   This method takes the customerlist and generates the list of seniorcitizen customers if the customer age greater than 60.This method has to be tested with full code coverage tests and must checked with proper assertions for checking if the list is empty.

Solution:

Important Annotations in JUnit 4

Annotation	Description
@Test	Identifies the method is a test method.
@Before	Will perform the method() before each test
@After	Test method must start with test
@BeforeClass	Will perform the method before the start of all tests
@AfterClass	Will perform the method after all tests have finished
@Ignore	Will ignore the test method
@Test(expected=IllegalArgumentException.class)	Tests if the method throws the named exception
@Test(timeout=100)	Fails if the method takes longer than 100 milliseconds

Assertion Overview

Assert Statement	Description
fail(String)	**To check that a certain part of the code is not reached.**
assertTrue(boolean condition);	**Asserts that the boolean condition is true**
assertsEquals([String message], expected, actual)	**Asserts that the actual value is equal to the expected value. Message is used for information.**
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**