Using JUnit in SAP NetWeaver Developer Studio

Applies to:

This article applies to SAP NetWeaver Developer Studio and JUnit.

Summary

Test-driven development helps us meet your deadlines by eliminating debugging time, minimizing design speculation and re-work, and reducing the cost and fear of changing working code. JUnit is an open source, regression-testing framework for Java that developers can use to write unit tests as they develop systems.

Author: Kousik Mukherjee

Company: HCL Technologies Ltd., Kolkata

Created on: 11 June 2007

Author Bio

Kousik Mukherjee is working as a Lead Consultant for HCL Technologies Ltd., Kolkata.

Table of Contents

Introduction

Programming bugs have enormous costs: time, money, frustrations, and even lives. It is possible to lessen as much of these pains as possible by creating and continuously executing test cases for our software. In this article I will try to explain a practical and common approach to address programming defects before they make it past our local development environment.

For years now, unit testing has been used by software developers as a process of verification of their programs. But there have been always some doubts in the minds of the programmers about who should write the tests. Apparently the answer is, test cases should be easy to write and execute and can be written and used by the programmers in spite of their busy schedules. Programmers need such kind of tool and JUnit is just the right one.

Originally written by Erich Gamma and Kent Beck, JUnit is a simple, open source framework to write and run repeatable tests. It is an instance of the “xUnit architecture for unit testing frameworks” and the de facto standard unit testing API for Java Development.

SAP NetWeaver Developer Studio provides different test frameworks for testing J2EE application. The integration of JUnit testing framework saves time and effort when performing functional unit testing of code. You can write your own unit tests and embed them in JUnit, thus building a comprehensive test suite incrementally.

Pre-requisites

It is assumed that the reader of this article has knowledge of:

SAP NetWeaver Developer Studio

Writing Test cases

Overview of JUnit Framework

JUnit Goals

Easy to write – Test code should contain no extraneous overhead.

Easy to learn to write – Minimize the barriers to test writing, because the target audience for Junit is programmers who are not usually professional testers.

Quick to execute – Tests should run fast so that they can be run hundreds times per day.

Easy to execute – Tests should run at the touch of a button and present their results in a clear and unambiguous format.

Isolated – Tests should not affect each other. If the order in which the tests are run changes, the results shouldn’t change.

Composable – Should be able to run any number of tests together.

JUnit API

JUnit has a simple API: subclass TestCase for your test cases and calls assertTrue(), assetEquals(), assertNull(), or assertNotNull() from time to time. Most of the time you will encounter these five classes or interfaces.

Assert – A collection of static methods for checking actual values against expected values.

An assert statement enables programmers to test their assumptions about their program. Each assert contains a boolean expression that you believe to be true when the assertion execute. If it is not true, the system will throw an error. By verifying that the boolean expression is indeed true, the assertion confirms the assumptions about the behavior of the program, increasing your confidence that the program is free of errors.

Test – The interface of all objects that act like tests.

TestCase – A single test.

TestSuite – A collection of tests.

TestCase and TestSuite are the two most prominent implementers of Test. Your test case classes will inherit from junit.framework.TestCase. Most of the time tests are run test suites containing a bunch of test cases, all of which are run when the test suite is run. Since it is composite in nature, a suite can contain suites which can contain suites and so on, making it easy to combine tests from various sources and run them together.

TestResult – A summary of the results of running one or more tests.

While you are running all these tests you need somewhere to store all these results: how many tests ran, which failed, and how long they took. TestResult collects results. A single TestResult is used to collect results of all the tests running at one go. When test runs or fails, the fact is noted in the TestResult. At the end of the run, the TestResult contains a summary of all tests.

A graphical test runner might observe the TestResult and update a progress bar every time a test starts. JUnit distinguishes between failures and errors. A failure is a violate JUnit assertion.

Test-First Programming & Test-Driven Development

You can use JUnit to write tests once you are done programming. Remember that tests are more valuable the closer they are written in time to time when an error might have been introduced. So instead of writing tests months after the code is complete, you can write tests days or hours or minutes after the possible introduction of a defect. You can write tests before the possible introduction of a defect.

“Writing a few lines of code, then a test that should run, or even better, to write a test first that won't run, then write the code that will make it run”. This leads to Test-Driven Development.

Writing a Test

A Test class must extend TestCase

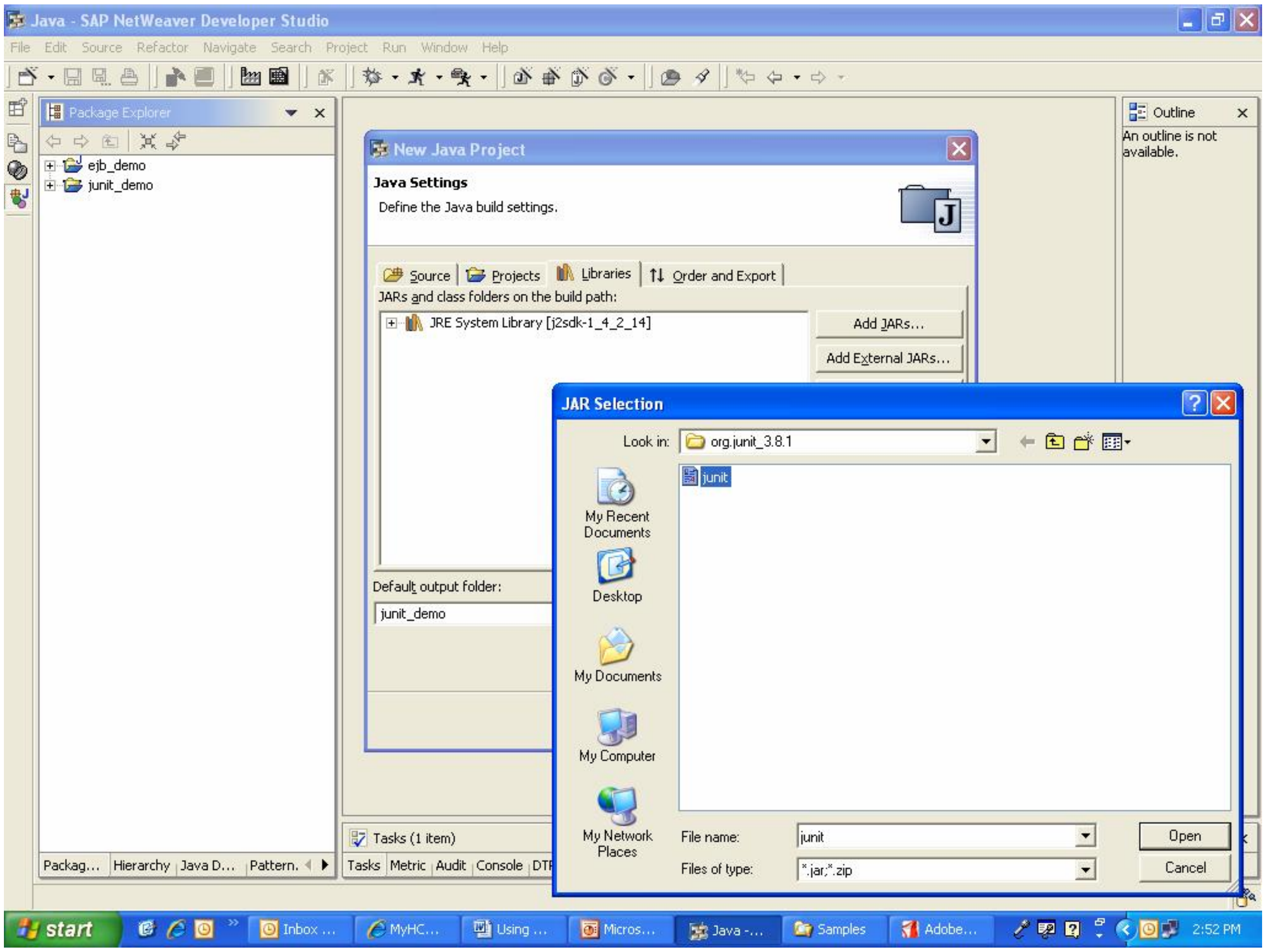
A Test method must begin with test

```
package com.hclt.test;
public class CollectionTest extends TestCase {
  private ArrayList list;
  /**
  * Before test
  * setUp() allocates Java Objects
  **/
  public void setUp() {
        list = new ArrayList(); }
  /** Test **/
  public void testCollection() {
```

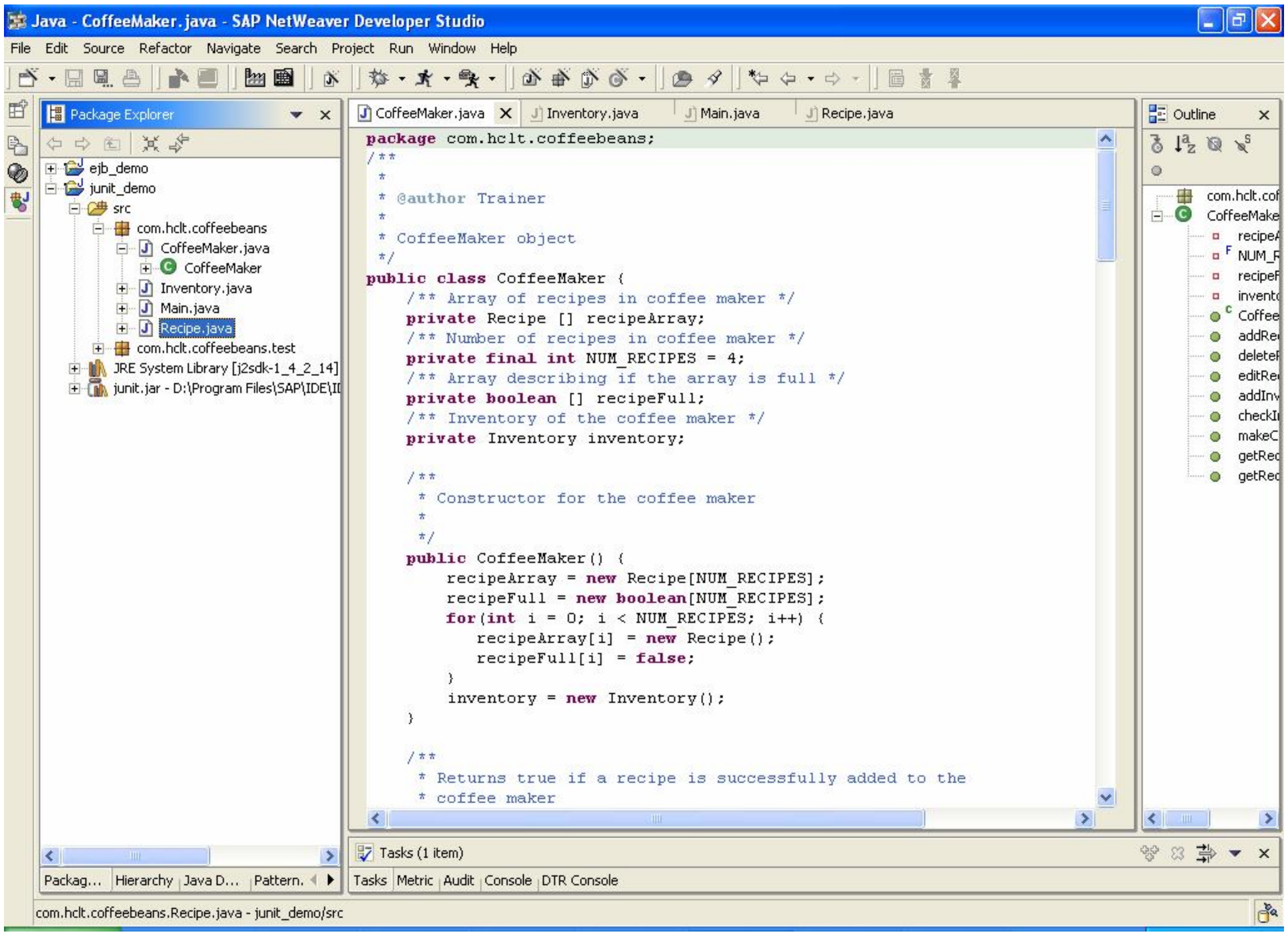
```
            assertTrue(list.isEmpty()); }
      /**
      * After test
      * If you allocate many megabytes of objects in setUp(), you may want to
      * clear the variables pointing to those objects so that they can be
      * garbage-collected because the test case objects are not
      * garbage-collected at any predictable time
      **/
      public void tearDown() {
            list.clear(); }
   }
```

JUnit in SAP NetWeaver Developer Studio

- Launch NetWeaver Developer Studio
- Create a new java project. To create a new project follow the below mentioned steps
    - From the File menu, select New... Project. The New Project window is displayed.
    - Select Java, and then Create a Java Project. Click Next. You can select J2EE and create an EJB Module Project or Web Module Project also (I have used Java Project for my example).
    - Enter a name for the project viz. junit_demo. Click next.
    - Java Settings window is displayed. In this window select the Libraries tab and add external jars. Add junit.jar. Click Finish.

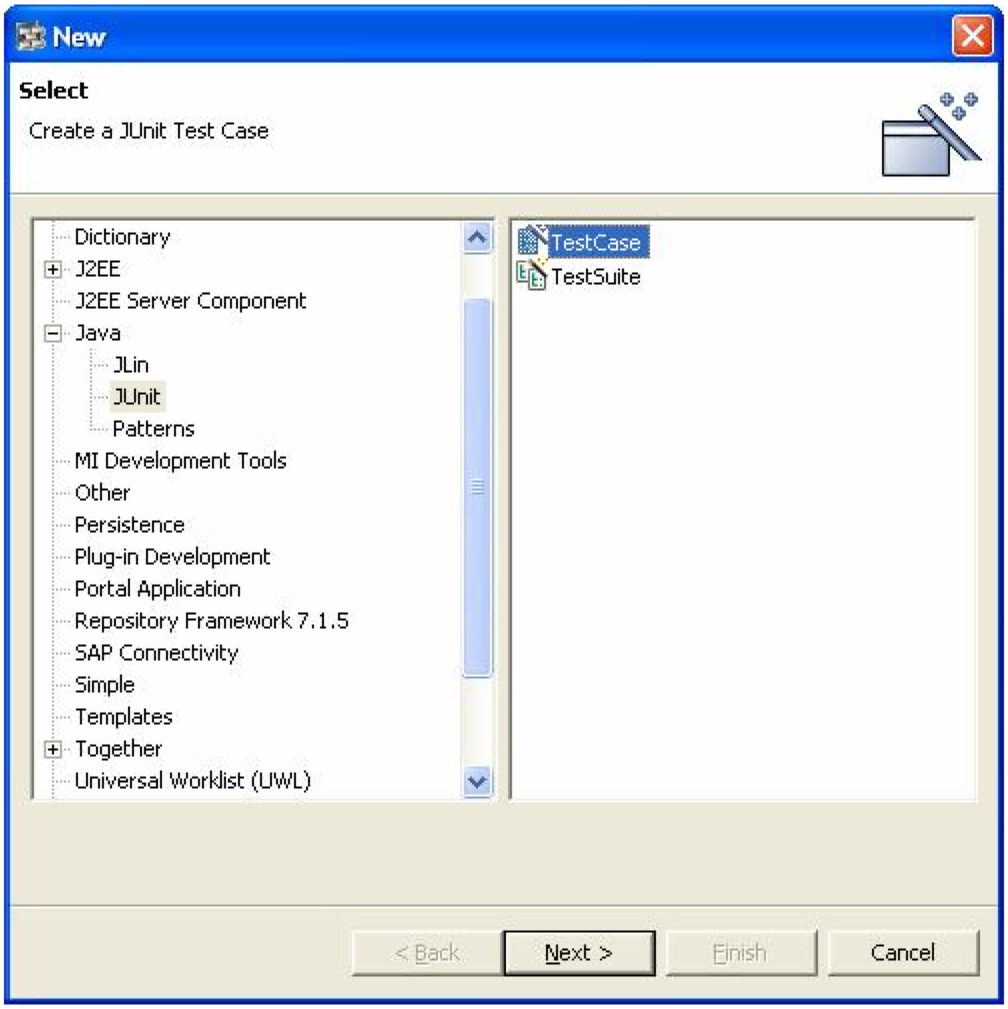


- Right click on your project → Select New… Package. Create a new package named ***com.hclt.coffeebeans***. Use this package to write your all java classes.

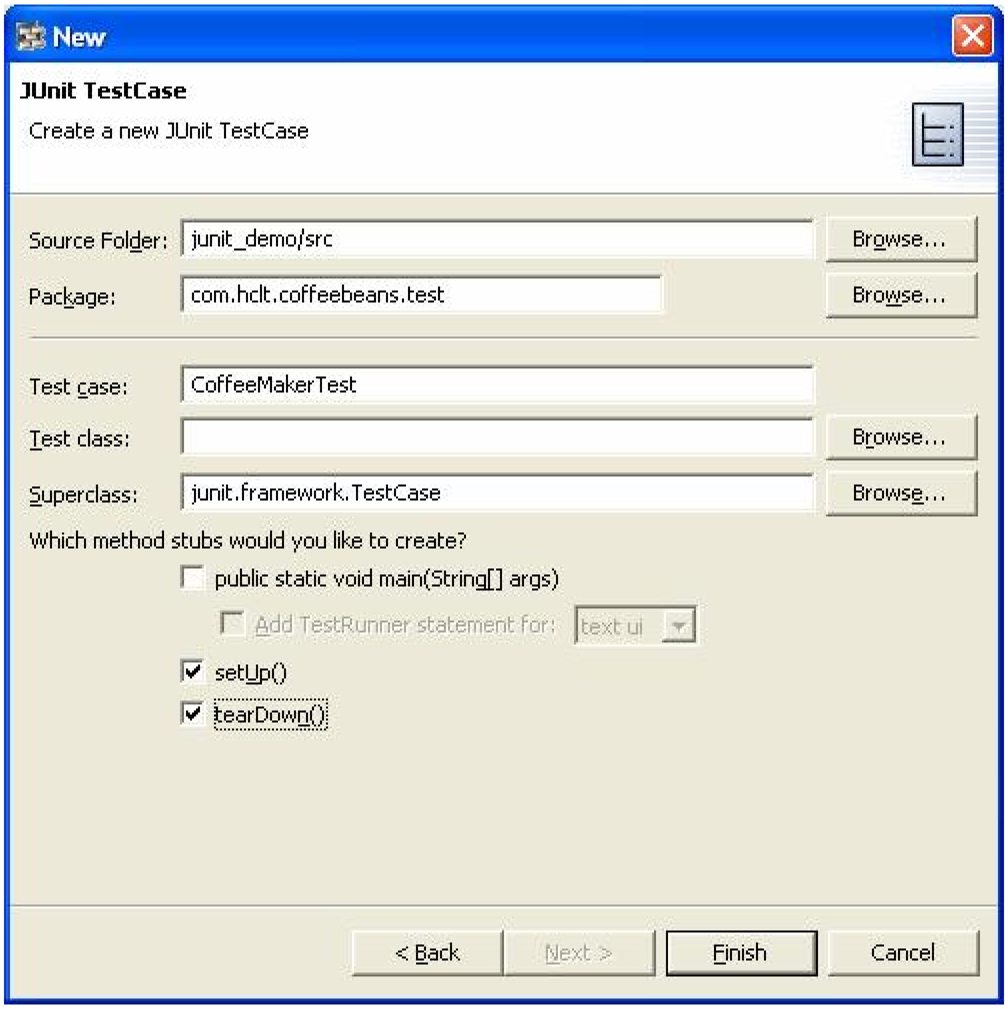

Java - CoffeeMaker.java - SAP NetWeaver Developer Studio
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ejb_demo
junit_demo
src
com.hclt.coffeebeans
CoffeeMaker.java
CoffeeMaker
Inventory.java
Main.java
Recipe.java
com.hclt.coffeebeans.test
JRE System Library [j2sdk-1_4_2_14]
junit.jar - D:\Program Files\SAP\IDE\II
CoffeeMaker.java
Inventory.java
Main.java
Recipe.java
Outline
Tasks (1 item)
Packag... Hierarchy Java D... Pattern.
Tasks Metric Audit Console DTR Console
com.hclt.coffeebeans.Recipe.java - junit_demo/src

```
package com.hclt.coffeebeans;
/**
 *
 * @author Trainer
 *
 * CoffeeMaker object
 */
public class CoffeeMaker {
    /** Array of recipes in coffee maker */
    private Recipe [] recipeArray;
    /** Number of recipes in coffee maker */
    private final int NUM_RECIPES = 4;
    /** Array describing if the array is full */
    private boolean [] recipeFull;
    /** Inventory of the coffee maker */
    private Inventory inventory;

    /**
     * Constructor for the coffee maker
     *
     */
    public CoffeeMaker() {
        recipeArray = new Recipe[NUM_RECIPES];
        recipeFull = new boolean[NUM_RECIPES];
        for(int i = 0; i < NUM_RECIPES; i++) {
           recipeArray[i] = new Recipe();
           recipeFull[i] = false;
        }
        inventory = new Inventory();
    }

    /**
     * Returns true if a recipe is successfully added to the
     * coffee maker
```

- Create another package named ***com.hclt.coffeebeans.test.*** Use this package to write all your tests.
- Right click on the ***com.hclt.coffeebeans.test*** → Select New… Other.
- Select Java → JUnit → TestCase. Click Next.



- Create a JUnit TestCase. Name it as CoffeeMakerTest. Select setUp() and tearDown() methods. Click Finish.



Source Folder: junit_demo/src
Package: com.hclt.coffeebeans.test
Test case: CoffeeMakerTest
Test class:
Superclass: junit.framework.TestCase
Which method stubs would you like to create?
- [ ] public static void main(String[] args)
- [ ] Add TestRunner statement for: text ui
- [x] setUp()
- [x] tearDown()

CoffeeMakerTest

The TestCase named CoffeeMakerTest is explained below. This TestCase is generated by JUnit tool in NetWeaver Developer Studio.

```
package com.hclt.coffeebeans.test;


import junit.framework.TestCase;


import com.hclt.coffeebeans.CoffeeMaker;
import com.hclt.coffeebeans.Recipe;
public class CoffeeMakerTest extends TestCase {

  private CoffeeMaker cm;
  private Recipe r1;
```

```
/**
    * Constructor for CoffeeMakerTest.
    * @param arg0
    */

   public CoffeeMakerTest(String arg0) {
         super(arg0);
   }

   /**
    * @see TestCase#setUp()
    */

   protected void setUp() throws Exception {
         cm = new CoffeeMaker();
         i = cm.checkInventory();

         r1 = new Recipe();
         r1.setName("Coffee");
         r1.setPrice(50);
         r1.setAmtCoffee(6);
         r1.setAmtMilk(1);
         r1.setAmtSugar(1);
         r1.setAmtChocolate(0);
   }

   /**
    * test method to add recipe
    */

   public void testAddRecipe() {
         assertTrue(cm.addRecipe(r1));
         assertNotNull(cm);
   }
```

```
	/**
	 * @see TestCase#tearDown()
	 */

	protected void tearDown() throws Exception {
		super.tearDown();
	}

    }
```

setUp() - instantiates a new CoffeeMaker and a Recipe and adds ingredients to Recipe.

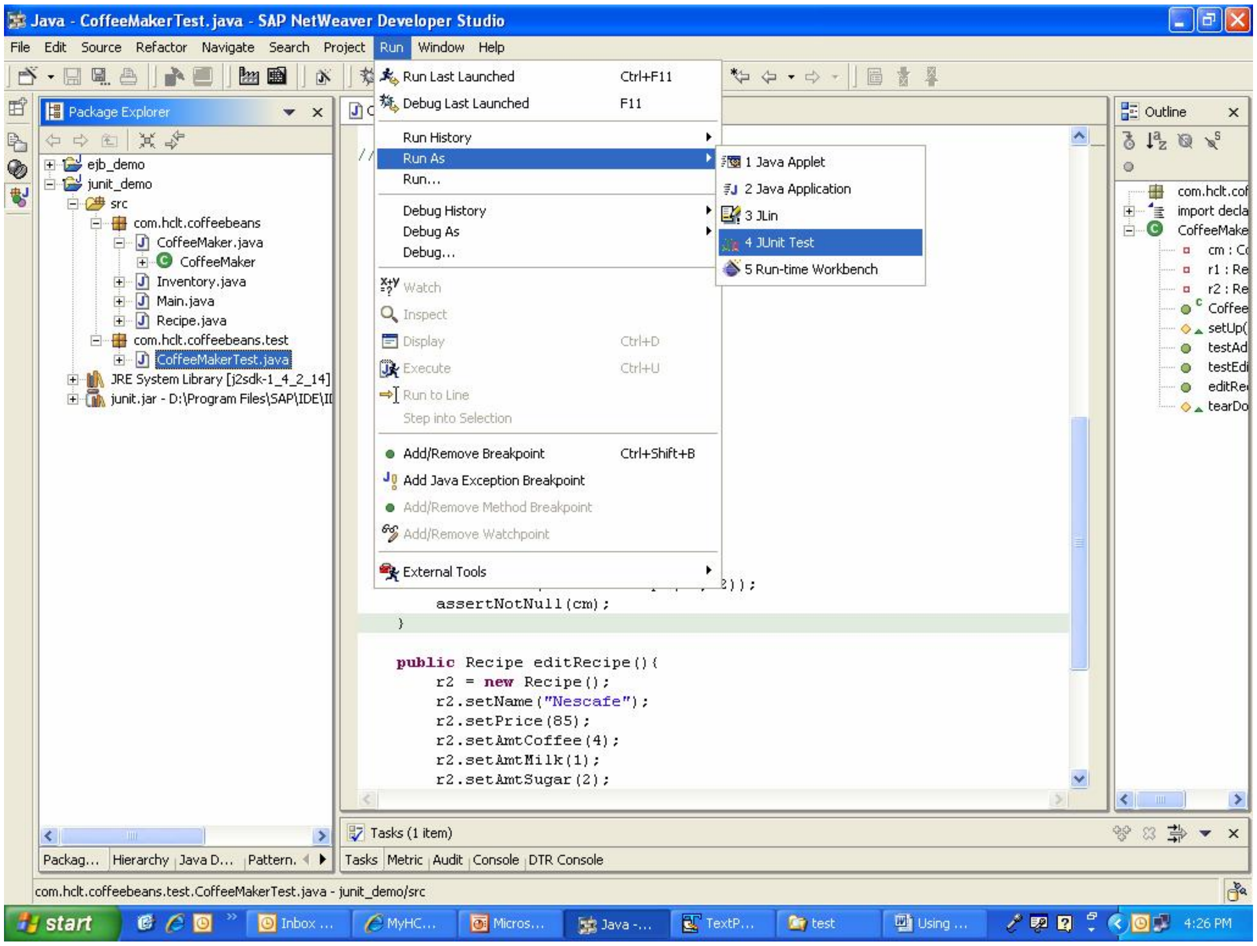
testAddRecipe() – asserts true that recipe r1 is added to coffeemaker cm. It also asserts that coffeemaker cm is not a null object as r1 is already added to it.

We can keep on adding tests to this TestCase viz. we are going to add testEditRecipe() to the TestCase to create a new Recipe.

```
	public void testEditReceipe() {
		r2 = editRecipe();
		assertTrue(cm.addRecipe(r2));
		assertTrue(cm.editRecipe(r1,r2));
		assertNotNull(cm);
	}

	public Recipe editRecipe(){
		r2 = new Recipe();
		r2.setName("Nescafe");
		r2.setPrice(85);
		r2.setAmtCoffee(4);
		r2.setAmtMilk(1);
		r2.setAmtSugar(2);
		r2.setAmtChocolate(0);
		return r2;
	}
```

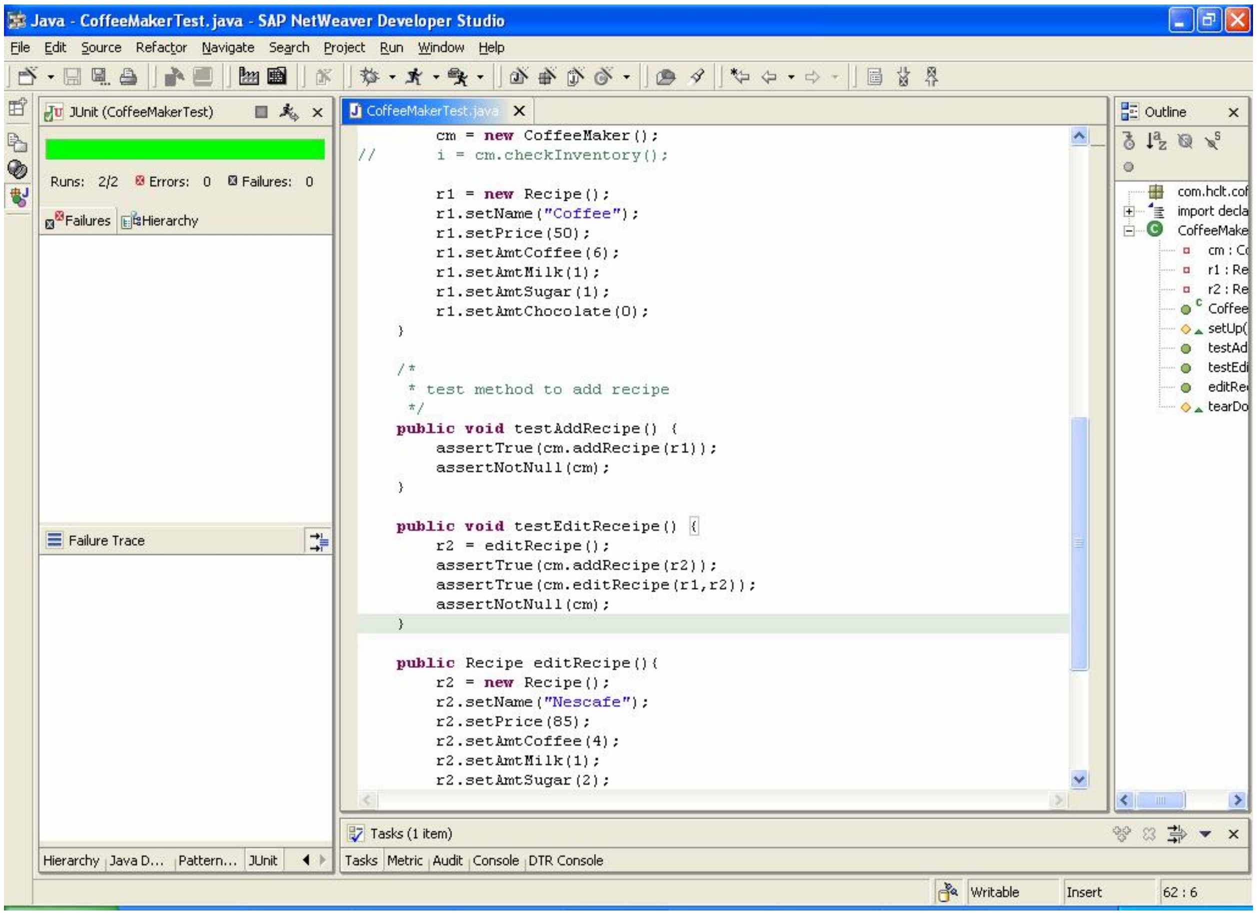
Running a Successful Test

- Steps to be followed to Run a Test.



```
        assertNotNull(cm);
    }

    public Recipe editRecipe(){
        r2 = new Recipe();
        r2.setName("Nescafe");
        r2.setPrice(85);
        r2.setAmtCoffee(4);
        r2.setAmtMilk(1);
        r2.setAmtSugar(2);
```

  - JUnit tool displays a green bar on successful Test run.

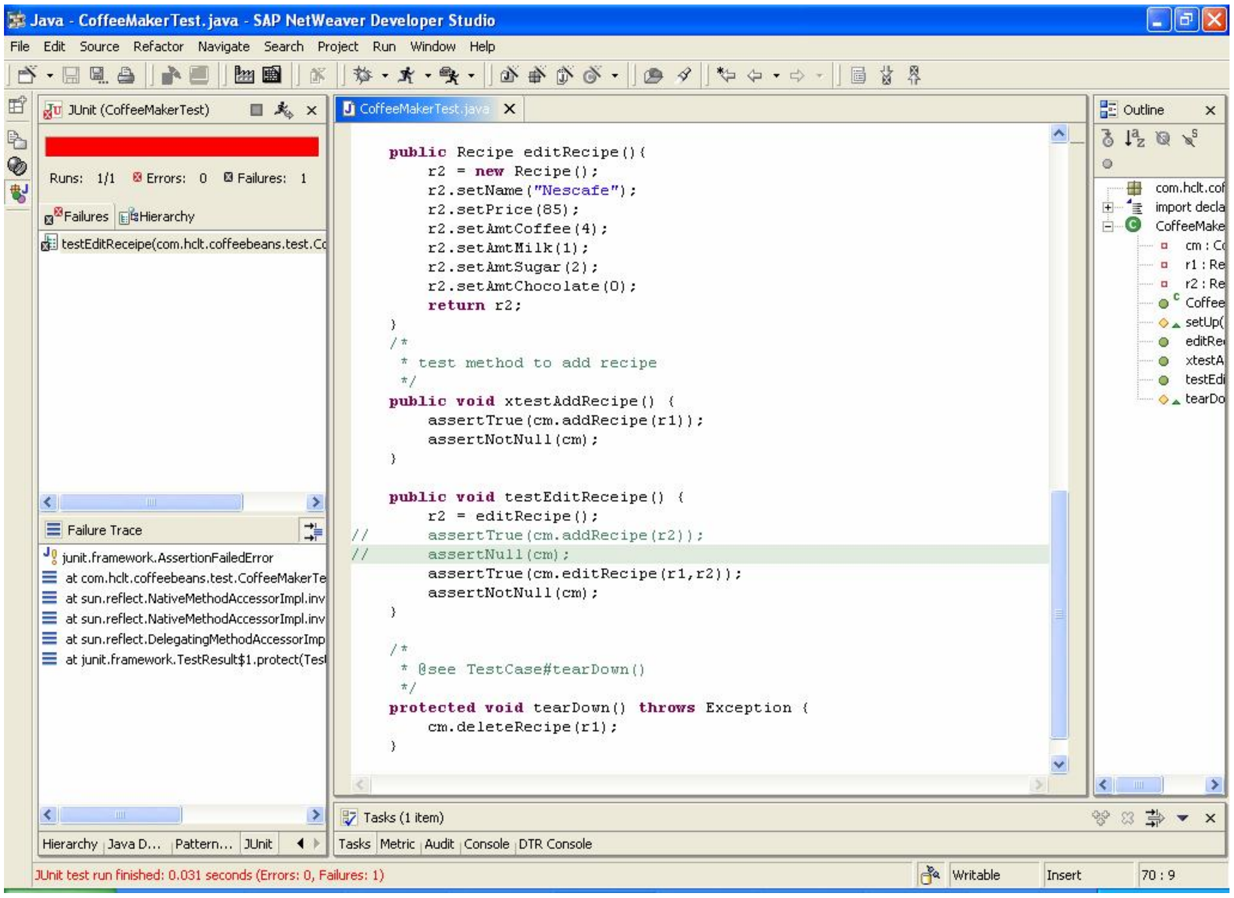

Java - CoffeeMakerTest.java - SAP NetWeaver Developer Studio
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit (CoffeeMakerTest)
Runs: 2/2 Errors: 0 Failures: 0
Failures Hierarchy
Failure Trace
Hierarchy Java D... Pattern... JUnit
CoffeeMakerTest.java
Outline
Tasks (1 item)
Tasks Metric Audit Console DTR Console
Writable Insert 62 : 6

```
        cm = new CoffeeMaker();
//      i = cm.checkInventory();

        r1 = new Recipe();
        r1.setName("Coffee");
        r1.setPrice(50);
        r1.setAmtCoffee(6);
        r1.setAmtMilk(1);
        r1.setAmtSugar(1);
        r1.setAmtChocolate(0);
    }

    /*
     * test method to add recipe
     */
    public void testAddRecipe() {
        assertTrue(cm.addRecipe(r1));
        assertNotNull(cm);
    }

    public void testEditReceipe() {
        r2 = editRecipe();
        assertTrue(cm.addRecipe(r2));
        assertTrue(cm.editRecipe(r1,r2));
        assertNotNull(cm);
    }

    public Recipe editRecipe(){
        r2 = new Recipe();
        r2.setName("Nescafe");
        r2.setPrice(85);
        r2.setAmtCoffee(4);
        r2.setAmtMilk(1);
        r2.setAmtSugar(2);
```

Test Failure

The whole idea behind running tests is to write tests that will fail. Fix the defects of your programs and the test will be successful. So a code like this will fail.

```
public void xtestAddRecipe() {
      assertTrue(cm.addRecipe(r1));
      assertNotNull(cm);
}

public void testEditReceipe() {
      r2 = editRecipe();
      assertTrue(cm.editRecipe(r1,r2));
      assertNotNull(cm);
}

/*
 * @see TestCase#tearDown()
 */
protected void tearDown() throws Exception {
      cm.deleteRecipe(r1);
}
```


Java - CoffeeMakerTest.java - SAP NetWeaver Developer Studio
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit (CoffeeMakerTest)
Runs: 1/1 Errors: 0 Failures: 1
Failures Hierarchy
testEditReceipe(com.hclt.coffeebeans.test.Co
Failure Trace
CoffeeMakerTest.java
Outline
Tasks (1 item)
Tasks Metric Audit Console DTR Console
Hierarchy Java D... Pattern... JUnit
Writable Insert 70 : 9

```
junit.framework.AssertionFailedError
at com.hclt.coffeebeans.test.CoffeeMakerTe
at sun.reflect.NativeMethodAccessorImpl.inv
at sun.reflect.NativeMethodAccessorImpl.inv
at sun.reflect.DelegatingMethodAccessorImp
at junit.framework.TestResult$1.protect(Tes
```

```
    public Recipe editRecipe(){
        r2 = new Recipe();
        r2.setName("Nescafe");
        r2.setPrice(85);
        r2.setAmtCoffee(4);
        r2.setAmtMilk(1);
        r2.setAmtSugar(2);
        r2.setAmtChocolate(0);
        return r2;
    }
    /*
     * test method to add recipe
     */
    public void xtestAddRecipe() {
        assertTrue(cm.addRecipe(r1));
        assertNotNull(cm);
    }

    public void testEditReceipe() {
        r2 = editRecipe();
//      assertTrue(cm.addRecipe(r2));
//      assertNull(cm);
        assertTrue(cm.editRecipe(r1,r2));
        assertNull(cm);
    }

    /*
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception {
        cm.deleteRecipe(r1);
    }
```

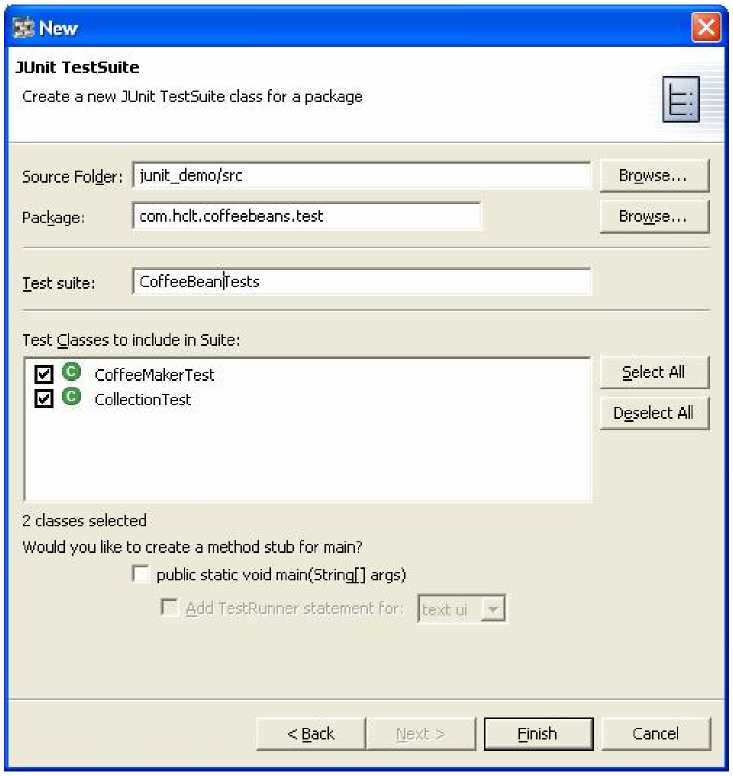
```
JUnit test run finished: 0.031 seconds (Errors: 0, Failures: 1)
```

The test results are displayed and the root for the failure can be traced. Making logical corrections to test assures that the programs on which the test is being run are correct in nature unless there is some incident occurs like the data has been deleted and so on. Tests are used to track and minimize very common exceptions in programs related to Null Pointer and Data Availability.

Note: Add x in front of test method name to make the method inactive i.e. this method will not be taken into consideration when the test runs.

Creating Test Suite

- Select New → Project → Other → Java → JUnit → TestSuite
- Enter TestSuite name.
- Select the classes required to be included in the Suite. Click Finish.



New

JUnit TestSuite

Create a new JUnit TestSuite class for a package

Source Folder: junit_demo/src

Package: com.hclt.coffeebeans.test

Test suite: CoffeeBeanTests

Test Classes to include in Suite:

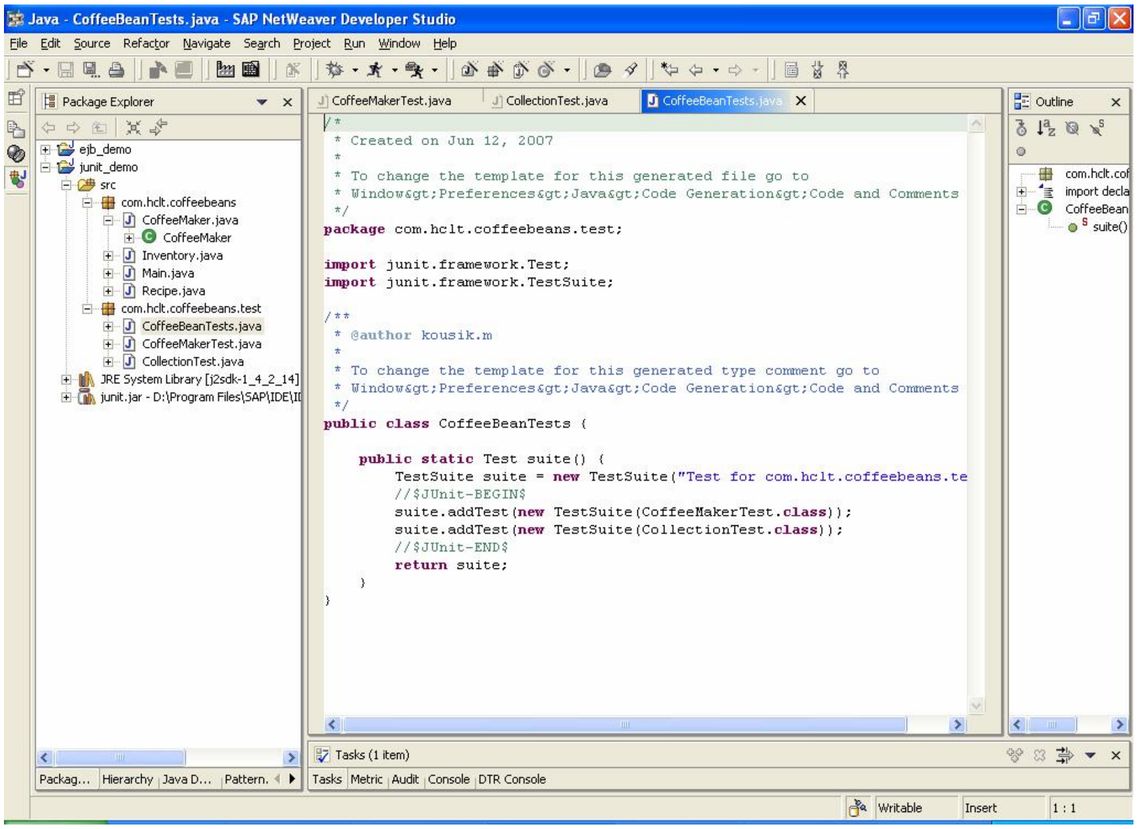
- [✓] CoffeeMakerTest
- [✓] CollectionTest

2 classes selected

Would you like to create a method stub for main?

- [ ] public static void main(String[] args)
- [ ] Add TestRunner statement for: text ui

- Code snippet from our TestSuite



Java - CoffeeBeanTests.java - SAP NetWeaver Developer Studio

```
/*
 * Created on Jun 12, 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package com.hclt.coffeebeans.test;

import junit.framework.Test;
import junit.framework.TestSuite;

/**
 * @author kousik.m
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class CoffeeBeanTests {

    public static Test suite() {
        TestSuite suite = new TestSuite("Test for com.hclt.coffeebeans.te
        //$JUnit-BEGIN$
        suite.addTest(new TestSuite(CoffeeMakerTest.class));
        suite.addTest(new TestSuite(CollectionTest.class));
        //$JUnit-END$
        return suite;
    }
}
```

- Select CoffeeBeanTests. Click Run… Run As… JUnit Test.

Related Content

The Philosophy of Unit Tests

Developing J2EE Applications

Integration Unit Testing on SAP NetWeaver Application Server

Disclaimer and Liability Notice