Coffee Maker Unit Test Framework

The source code for the very simple Java unit test framework can be downloaded as a zip file. Or you can just cut and paste from here.

The first part is the TestGUI class. This class is the applet that runs the tests and reports the results. You will probably notice at this point that the formatting is different from what you are used to seeing. This is an experiment.

```java
package unittest.framework;

import java.awt.*;
import java.applet.*;

public class TestGUI extends Applet
   {Label scoreLabel;
   Button runTestsButton;
   List listOfTests;
   Test tests [];

   public void init()
      {initializeTests();
      setLayout(new BorderLayout());
      add("North", scoreLabel());
      add("Center", listOfTests());
      add("South", runButton());
      scoreLabel.setBackground(Color.lightGray);}

   public void initializeTests()
      {String testSuiteName = getParameter("TestSuite");
      tests = testSuiteNamed(testSuiteName).tests;}

   private TestSuite testSuiteNamed(String aClassName)
      {try
         {return (TestSuite) Class.forName(aClassName).newInstance();}
      catch (Exception exception)
         {return new TestSuite();};}

   void runTests()
      {for (int each = 0; each < tests.length; each++)
         {runTest(each);
         showResults();};}

   void runTest(int anIndex)
      {tests[anIndex].setUp();
      tests[anIndex].run();
      tests[anIndex].tearDown();}

   private void showResults()
      {for (int each = 0; each < tests.length; each++)
         {listOfTests.replaceItem(tests[each].result, each);}
         showScore();}

   private void showScore()
      {int passed = numberPassed();
      float total = (float) tests.length;
      int score = (int)(passed / total * 100);
      scoreLabel.setText(new Integer(score).toString() + "%");
      showPassFail(score);}

   private int numberPassed ()
      {int passed = 0;
      for (int each = 0; each < tests.length; each++)
         {if (tests[each].success) passed++;}
      return passed;}

   private void showPassFail (int aScore)
      {scoreLabel.setBackground((aScore == 100) ? Color.green : Color.red);}

   private Label scoreLabel()
      {return scoreLabel = new Label("Not Run", Label.CENTER);}

   private List listOfTests ()
      {listOfTests = new List(tests.length, false);
      for (int each = 0; each < tests.length; each++)
         {listOfTests.addItem(tests[each].result);};
      return listOfTests;}

   private Button runButton()
      {runTestsButton = new Button("Run Tests");
      return runTestsButton;}

   public boolean action(Event anEvent, Object anObject)
      {if(wasRunTestsPressed(anEvent))
         {runTests();
         return true;}
      else
         {return false;};}

   private boolean wasRunTestsPressed(Event anEvent)
      {return anEvent.target == runTestsButton;};}
```

The next portion is the Test class. This class will be the super class of any tests we will be creating.

```java
package unittest.framework;

/*
* This is the class to extend for each test you need to run.
* One new class for each test. JUnit allows multiple tests
* per test class and is becoming the standard. Use JUnit instead.
* setUp() is called before the test is run and can be used to
* initialize your test. runTest() is called to actually run
* your test. Override it and send the message should(boolean, String)
* to check if the test has passed or failed.
* tearDown() is called after your test is run and can be used
* to clean up.
*/

public class Test
   {public boolean success;
   public String result;

   public Test()
      {super();
      this.initialize();}

   private void initialize()
      {this.testFailed("not run");}

   public void setUp()
      {}

   protected void runTest()throws Exception
      {}

   public void tearDown()
      {}

   protected void should (boolean aTestPassed, String aMessage)
      {if (!aTestPassed)
         {throw new TestFailedException(aMessage);};}

   public void run()
      {runAndCaptureAborts();}

   private void runAndCaptureAborts()
      {try
         {runAndCaptureFailures();}
      catch (Exception exception)
         {testFailed("Aborted : " + exception.getMessage());};}

   private void runAndCaptureFailures()throws Exception
      {try
         {runAndAllowExceptions();}
      catch (TestFailedException exception)
         {testFailed("Failed : " + exception.getMessage());};}

   private void runAndAllowExceptions()throws TestFailedException, Exception
      {runTest();
      testPassed();}

   private void testPassed()
      {success = true;
      result = message("Passed");}
   private void testFailed(String aMessage)
      {success = false;
      result = message(aMessage);}

   private String message(String aString)
      {return getClass().getName() + " : " + aString;};}
```

The third and last portion of the framework is the TestSuite class. This simple class holds a set of tests together.

```
package unittest.framework;

/*
* Extend this class to create sets of tests to be run together.
* Override the creation method and initialize the tests variable to
* contain an array of Test subclasses. One instance for each
* test that needs to be run. This superclass also provides a
* default empty suite of tests.
*/

public class TestSuite
   {public Test tests [];

   public TestSuite()
      {tests = new Test[0];};}
```

It helps to see some examples of how to actually use this framework. Let's create a sample TestSuite with 3 tests. One each for pass, fail, and abort results.

The most instructional class is presented last. The `AbortTest` is more like a test we would create. An expression followed by what we are testing is passed to the should method.

```
package unittest.framework;

public class FrameworkTests extends TestSuite
   {public FrameworkTests()
      {tests = new Test[3];
      tests[0] = new unittest.framework.GoodTest();
      tests[1] = new unittest.framework.FailTest();
      tests[2] = new unittest.framework.AbortTest();};}

public class GoodTest extends Test
   {protected void runTest()
      {should (true, "This test always succeeds");};}

public class FailTest extends Test
   {protected void runTest()
      {should(false, "this test always fails");};}

public class AbortTest extends Test
   {private int number[] = {0,1,2,3};

   protected void runTest()
      {should(number[1] / number[0] == 0, "test divide by zero");};}
```