JUnit 4.0 - A Quick Start Guide by Jens Uwe Pipka

Migrating existing JUnit 3.x unit tests to JUnit 4.0 as well as writing new unit tests with JUnit 4.0 is much easier at it seems. Most important, only the JUnit framework that is used for test implementation has been changed; you can still execute the unit tests in your used test runner including the integrated Eclipse test runner or the JUnit 3.0 Swing test runner. As JUnit 4.0 now comes with annotations, it is required to use Java 5.0 or above. In the following, we describe the necessary steps to get warm with JUnit 4.0 for both, migrating existing unit tests as well as writing new ones.

Migrating existing Unit Tests

```
~~import junit.framework.TestCase;~~

import org.junit.*;
import junit.framework.JUnit4TestAdapter;

public class SimpleUnitTest
  ~~extends TestCase~~ {

  String value;

  @Before public void setUp() {
    value="start";
  }

  @Test public void testValue() {

    ~~this~~Assert.assertEquals("start", value);
    value += " 1";
    ~~this~~Assert.assertEquals("start 1", value);
  }

  @After public void tearDown() {
    value="";
  }

  public static junit.framework.Test suite() {
    return new
     JUnit4TestAdapter(SimpleUnitTest.class);
  }

}
```

1. Check that Java 5.0 is used.
2. Within class path, replace junit.jar from Junit 3.x with the updated junit.jar from Junit 4.0. Your old unit test should still work with the new JUnit version.
3. Change import Statement: Replace
   `import junit.framework.TestCase;`
   with
   `import org.junit.*;`
   `import junit.framework.JUnit4TestAdapter;`
4. Introduce Annotations:
   1. Insert `@Test` in front of any `test..()` method, i.e. so called test fixture.
   2. Insert `@Before` in front of the original `setUp()` method
   3. Insert `@After` in front of the original `tearDown()` method
5. The test case should not inherit from Test Case any more, thus delete `extends TestCase`. As a consequence, `this.assertEquals` is not accessible any longer. Therefore, replace `this.assertEquals(...)` with `Assert.assertEquals(...)`
6. To execute an unit test based on Junit 4.0 with an existing test runner, add the following method:
   ```
   public static junit.framework.Test suite() {
      return new Junit4TestAdapter(<classname>.class);
   }
   ```
7. That's all! Now, you can execute this unit test by calling SimpleUnitTest within your favourite test runner, including the Eclipse built-in test runner.

Writing new unit tests

1. Check that Java 5.0 is used.

2. Insert junit.jar from JUnit 4.0 in your class path.

3. Do **not** extend the new class from
   `junit.framework.TestCase`

4. Import the JUnit 4.0 relevant classes:
   ```
   import org.junit.*;
   import junit.framework.JUnit4TestAdapter;
   ```

5. Annotate new test methods with `@Test`

6. Annotate the test fixture with `@Before` and `@After` for specific initialization.

7. Use `Assert.assertEquals(...)` to check the actual results with the expected ones.

8. Insert the following method to run the unit test with an existing test runner:
   ```
   public static
     junit.framework.Test suite() {
       return new
         Junit4TestAdapter(<classname>.class);
   }
   ```

```
import org.junit.*;
import junit.framework.JUnit4TestAdapter;
import java.util.Vector;

public class SimpleUnitTest {

  String value;

  @Before public void setUp() {
    value="start";
  }

  @Test public void testValue() {
    Assert.assertEquals("start", value);
    value += " 1";
    Assert.assertEquals("start 1", value);
  }

  @Ignore public void testDoNothing() {
    Assert.assertTrue(true);
  }

  @Test(expected=
    ArrayIndexOutOfBoundsException.class)
    public void testException() {
    Vector test = new Vector();
    Assert.assertEquals(0, test.elementAt(42));
  }

  @After public void tearDown() {
    value="";
  }

  public static junit.framework.Test suite() {
   return new
     JUnit4TestAdapter(SimpleUnitTest.class);
  }

}
```

Some Advanced Features:

One time setup:

Use `@BeforeClass` and `@AfterClass` to initialize specific resources that should be initialized only once during the run of a certain test case.

Exception handling:

You can set a parameter for the `@Test` annotation which exceptions should be thrown:
`@Test(expected=<classname>.class)`

Ignoring a Test:

You can ignore an test by using the `@Ignore` annotation.