JUnit4 primer

Introduction

This page is a Junit 4 primer for those who are writing new tests in the project, or who are transforming old JUnit 3.8 tests to JUnit 4.4

JUnit 4.4 does not bring a hell of new features, but at least one is very interesting : the possibility to have class wide initialization (a global setup and teardown per test class). This will save us a lot of time when running integration tests where the server was launched for each single test.

Howto

Writing a JUnit4 testcase is very simple. The first thing to do is to enable JUnit4 in eclipse, which should be the default in Eclipse-3.3.1.

As we are now using the JUnit-4.4.jar in all our projects, you don't have anything to do with old tests (JUnit 3.8 tests run well with this new version). Let's see how we define a new test.

Creating the TestCase

First, create a new TestCase, with comments. No need to extends TestCase anymore. You don't have to postfix the class with 'Test', but it would be better to do so, for clarity.

```
package org.apache.directory.shared.ldap.common;

/**
 * A test for StringValue
 *
 * @author <a href="mailto:dev@directory.apache.org">Apache Directory Project</a>
 * @version $Rev$, $Date$
 */
public class StringValueTest
{
...
```

A first test

Create your first test by adding an @Test annotation in front of the test :

```
import org.junit.Test;
import static org.junit.Assert.assertNull;

...
    /**
     * Test a null value
     */
    @Test public void testNullValue()
    {
        StringValue value = new StringValue();

        assertNull( value.getValue() );
    }
...
```

As a result, a new Import should be created : import org.junit.Test; Also note that the test name don't have to start with 'test', but this is better to do so, still for clarity.As you are using an assertNull assertion, you will get an error if you don't add a static import : import static org.junit.Assert.assertNull;(Static imports are for importing static methods directly without having to prefix the method with Assert here. This is a novelty brought by Java 5) This is it. You can now run your first test !

One step further

Now, let's suppose you need to initialize some data and clear them up at the end of each test. This is very easy. Just define your setup and teardown methods (they can have other names, but keep it simple) :

```
...
    @Before
    public void setUp()
    {
        // Init code here ...
    }
...
    @After
    public void tearDown()
    {
        // teardown code here ...
    }
....
```

These two methods will be called by every single test.

Of course, you will have to add some new imports :

```
...
import org.junit.Before;
import org.junit.After;
...
```

"One small step for JUnit, a giant leap for testers..."

JUnit4 brings a very long waited new feature : you can launch a single setup and teardown for the whole test class. Again, it's really easy :

```
...
import org.junit.BeforeClass;
import org.junit.AfterClass;

...
    @BeforeClass
    public static void globalSetUp()
    {
        // Init code here ...
    }
...
    @AfterClass
    public static void globalTearDown()
    {
        // teardown code here ...
    }
....
```

The methods name are not important, as soon as you have the annotations.

Last, not least

You can also tests failing tests more easily, by using parameters for the @test annotation :

```
    @Test ( expected = IndexOutOfBoundsException.class ) public void outOfBounds()
    {
        new ArrayList<Object>().get(1);
    }
```

or

```
@Test ( timeout = 1000 ) public void infinity()
{
    for (;;)
    {
    }
}
```

To disable a test without commenting it, just add the **@Ignore** annotation in front of the test :

```
@Ignore("Emmanuel has committed a stupid infinite loop in this test ...") @Test public void infinity()
{
    for (;;)
    {
    }
}
```