JUnit 4

Charles Sharp
Object Computing, Inc.

St. Louis Java Users Group
April 10, 2008

Best Laid Plans...

- Conversion from JUnit 3.8.x to 4.4
- Runners
- New Features

First Things

- Must use Java 5 or later:
  - Annotations
  - Generics
  - Autoboxing
- *And* Must use JUnit 4.4 jar

Conversion

The 4.4 Package contains the 3.8 packages which permits phased conversion of existing tests. (mostly)

Work on one test, get it running, go to the next.

This works best in an IDE.

As Simple As...

1. Modify and add imports

2. Remove

   2.1. extends TestCase

   2.2. the super() invocations

   2.3. the constructor argument

   2.4. (modify) protecteds to public

3. Add

   3.1.Annotations to support and tests

Import conversion

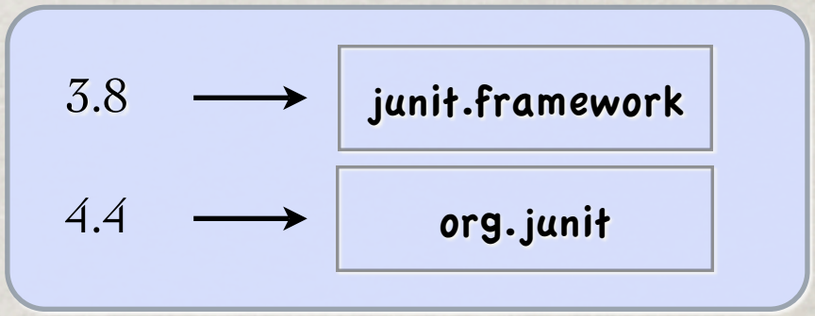
JUnit 3.8

```
import junit.framework.*
public class BlargleTest extends TestCase {
```

JUnit 4.4

```
import org.junit.*;
import static org.junit.Assert.*;
public class BlargleTest {
```

- New package structure



- 3.8.x TestCase class contains Assert definitions
- "import static" used to fetch the Asserts in 4.4

Code Conversion

- remove “extends TestCase” and associated “super()” calls
- decorate setUp() with @Before
- decorate tearDown() with @After
- decorate the test<method>s with @Test

Runner Conversion

- JUnit 4.x is out of the UI business
  - No more swingui, awtui, or textui
  - JUnit developer's assumption is most tests are run from an IDE or ant.
    - Supporting UIs took too much time
- Use org.junit.runner.JUnitCore <Test...> for console output

@Test

- Replaces naming convention of test<Meaningful> (that ***doesn't*** mean...)
- Used before each test method to be run
- @Test is a multi-value annotation:
  - expected=<some exception class> (don't forget the .class!)
  - timeout= <millisecond timeout value>

org/junit/Test.java

@Before & @After

- Methods are public, not protected
- Can be multiple occurrences of either
- Order of execution for multiple methods in the same class is ***not*** guaranteed
- Order of execution for parent classes ***is*** guaranteed

org/junit/After.java
org/junit/Before.java

@BeforeClass
@AfterClass

- Annotation appears before a public static method
- Superclass @BeforeClass is executed before class
- Superclass @AfterClass is executed after class

org/junit/BeforeClass.java

@Ignore

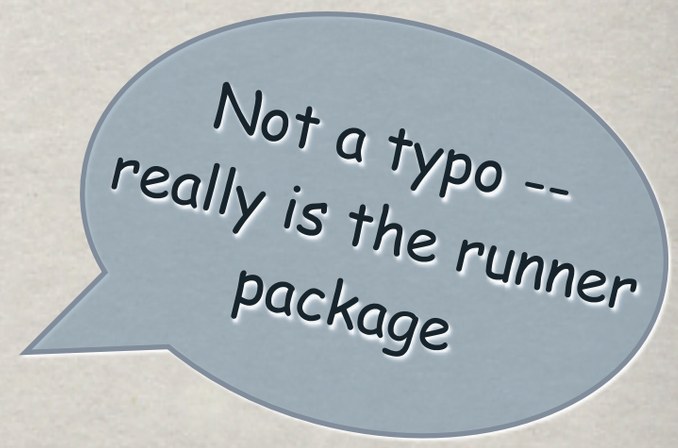
- Inserted before an @Test to ignore a test. Honest.
- A single-element annotation taking an optional String used as an explanation for ignoring the test.

org/junit/Ignore.java

@SuiteClasses

- Annotation is defined in the Suite class
- Specifies the classes to be run when a class is annotated with @RunWith(Suite.class)
- Used to enumerate the classes/methods to be grouped together for a single test. Analogous to the suite() usage in 3.8

org/junit/runners/Suite.java
org/junit/runner/RunWith.java



Missing

- Errors -- JUnit 3.x has failures, errors, and successes. JUnit 4.4 has failures, successes, and ignored tests.
- TestCase class had getTest() method -- often used in setUp/tearDown methods for logging

Asserts(1)

- Quote from JUnit Release Notes for 4.4

  "The old assert methods are never, ever, going away. Developers may continue using the old assertEquals, assertTrue, and so on."

- Now handles array tests with

  `assertArrayEquals(Object[] expecteds, Object[] actuals);`

  autoboxing removed the need for specifying types.

org/junit/Assert.java

Asserts(2)

- Asserts now compare all numbers using their native implementations, for example:

```
assertEquals(new Integer(1), new Long(1));
```

now fails. It fails in 3.8 but passes in 4.3.

- JUnit 4 is compatible with the Java assert keyword. An assert in the code that fails is marked as a failure.
(must use jvm arg, -ea)


NEW
improved

Matchers

static <T> Matcher<T>
allOf(java.lang.Iterable<Matcher<? extends T>> matchers)
Evaluates to true only if ALL of the passed in matchers evaluate to true.

static <T> Matcher<T>
allOf(Matcher<? extends T>... matchers)
Evaluates to true only if ALL of the passed in matchers evaluate to true.

org/hamcrest/CoreMatchers.java

Matchers

static <T> Matcher<T>
any(java.lang.Class<T> type)
This matcher always evaluates to true.

static <T> Matcher<T>
anyOf(java.lang.Iterable<Matcher<? extends T>> matchers)
Evaluates to true if ANY of the passed in matchers evaluate to true.

static <T> Matcher<T>
anyOf(Matcher<? extends T>... matchers)
Evaluates to true if ANY of the passed in matchers evaluate to true.

org/hamcrest/CoreMatchers.java

Matchers

static <T> Matcher<T>
anything()
This matcher always evaluates to true.

static <T> Matcher<T>
anything(java.lang.String description)
This matcher always evaluates to true.

org/hamcrest/CoreMatchers.java

Matchers

static <T> Matcher<T>
describedAs(java.lang.String description, Matcher<T> matcher, java.lang.Object... values)
Wraps an existing matcher and overrides the description when it fails.

static <T> Matcher<T>
equalTo(T operand)
Is the value equal to another value, as tested by the Object.equals(java.lang.Object) invokedMethod?

org/hamcrest/CoreMatchers.java

Matchers

static Matcher<java.lang.Object>
instanceOf(java.lang.Class<?> type)
Is the value an instance of a particular type?

static <T> Matcher<T>
sameInstance(T object)
Creates a new instance of IsSame24

org/hamcrest/CoreMatchers.java

Matchers

static Matcher<java.lang.Object>
is(java.lang.Class<?> type)
This is a shortcut to the frequently used is(instanceOf(SomeClass.class)).

static <T> Matcher<T>
is(Matcher<T> matcher)
Decorates another Matcher, retaining the behavior but allowing tests to be slightly more expressive.

static <T> Matcher<T>
is(T value)
This is a shortcut to the frequently used is(equalTo(x)).

org/hamcrest/CoreMatchers.java

Matchers

static <T> Matcher<T>
not(Matcher<T> matcher)
Inverts the rule.

static <T> Matcher<T>
not(T value)
This is a shortcut to the frequently used not(equalTo(x)).

org/hamcrest/CoreMatchers.java

Matchers

static <T> Matcher<T>
notNullValue()
Matches if value is not null.

static <T> Matcher<T>
notNullValue(java.lang.Class<T> type)
Matches if value is not null.

static <T> Matcher<T>
nullValue()
Matches if value is null.

static <T> Matcher<T>
nullValue(java.lang.Class<T> type)
Matches if value is null.

org/hamcrest/CoreMatchers.java

assertThat

assertThat([value], [matcher statement]);

- Advantages (from the horse's mouth):
  - More readable and typeable: this syntax allows you to think in terms of subject, verb, object (assert "x is 3") rather than assertEquals, which uses verb, object, subject (assert "equals 3 x")
  - Combinations: any matcher statement s can be negated (not(s)), combined (either(s).or(t)), mapped to a collection (each(s)), or used in custom combinations (afterFiveSeconds(s))
  - Custom Matchers. By implementing the Matcher interface yourself, you can get all of the above benefits for your own custom assertions.

http://junit.sourceforge.net/doc/ReleaseNotes4.4.html

assertThat(2)

- Advantages(continued):
  - Readable failure messages. Compare:

```
assertTrue(responseString.contains("color") || responseString.contains("colour"));
```

which produces the failure message:

```
java.lang.AssertionError:
```

```
assertThat(responseString, anyOf(containsString("color"), containsString("colour")));
```

which produces the failure message:

```
java.lang.AssertionError:

    Expected: (a string containing "color" or a string containing "colour")

    got: "Please choose a font"
```

http://joe.truemesh.com/blog/000511.html

Assume

The possibility of saying anything about a thing rests on the assumption that it preserves its identity, or continues to be the same thing in the respect described, that it will behave in future situations as it has in past. -- Frank Knight, economist (1885 - 1972)

- `assumeThat(T value, Matcher<T> assumption);`
- `assumeNotNull(java.lang.Object... objects);`
- `assumeTrue(boolean b);`
- `assumeNoException(java.lang.throwable t);`

org/junit/Assume.java
org/hamcrest/CoreMatchers.java

Parameterized Tests

- @RunWith(Parameterized.class)on the Test Class

- A public static method that returns a Collection containing test data. Each element of the collection must be an Array of the parameters needed by the constructor.

- A public constructor that uses the parameter values in a Collection element.

org/junit/Parameterized.java

@Theory
@DataPoint

- A Theory is:
  - a general statement about a set of values
  - a template for generating tests by instantiating with particular values
- A test is a single statement about a single set of values

org/junit/experimental/theories/Theory.java
org/junit/experimental/theories/DataPoint.java

JUnit Sourceforge Members

Mike Clark
Matthias Schmidt
David Saff
Erich Gamma
Erik G. H. Meade
Kent Beck
Vladimir Ritz Bossicard

More Information

JUnit 4:

FAQ: http://junit.sourceforge.net/doc/faq/faq.htm

Release notes: http://junit.sourceforge.net/doc/ReleaseNotes4.4.html

javadoc: http://junit.sourceforge.net/javadoc_40/index.html

(Note two things, the javadoc at the website is for 4.0 and the javadoc tab at the website is for the 3.8 javadoc)

Theories:
http://shareandenjoy.saff.net/2006/12/new-paper-practice-of-theories.html

Java Testing with JUnit

Goals
Testing is a vital component of software development because it has a proportional relationship to code quality - the more complete the testing suite the better the quality of the application code. This course is meant to introduce students to many different aspects of testing, their application in a wide variety of software settings, and the open source tools that are available for executing Java software testing. Students attending the class will gain a firm understanding of software testing and test writing strategies for unit, integration, and acceptance tests including design tips that promote testability as well as the tools that are used to run all styles of tests.

Audience
Software developers with some Java experience, wishing to learn about testing Java applications

Duration
2 days

Prerequisites
"Intermediate Java Programming" or equivalent experience

Contents
- Testing overview
- The JUnit Framework
- Writing unit tests
- Mock objects
- Improving testability through refactoring
- Test automation and coverage tools
- Integration and acceptance testing
- Performance testing

http://www.ociweb.com/education/services/descrip/ESJA16-01.html

Agley

- Transform 3.8 to 4.4 (soon to be 4.5)
- Taste of the new features of 4.4

Here's the way testing goes:

```
becomeTimidAndTestEverything
while writingTheSameThingOverAndOverAgain
    becomeMoreAggressive
    writeFewerTests
    writeTestsForMoreInterestingCases
    if getBurnedByStupidDefect
        feelStupid
        becomeTimidAndTestEverything
    end
end
```

The loop, as you can see, never terminates.

J.B.Rainsberger

taken from http://junit.sourceforge.net/doc/faq/faq.htm#best_3

Go. Test Hard.