JUnit: Java Testing Framework

Course of Software Engineering II
A.A. 2011/2012

Valerio Maggio, PhD Student
Prof. Marco Faella

Outline

- Brief Introduction to Junit 4.x
- Main differences with JUnit 3.x
- JUnit Examples in practice
- Further Insights
    - (Extensions and compatibilities)

JUnit Preliminaries

- **Q:** How many “types” of testing do you know?

  A: System Testing, Integration Testing, Unit Testing....

- **Q:** How many “testing techniques” do you know?

  A: Black Box and White Box Testing

  - Which is the difference?

- **Q:** What type and technique do you think JUnit covers?

JUnit: Java Unit Testing framework

- 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.
  - (source: http://junit.org)

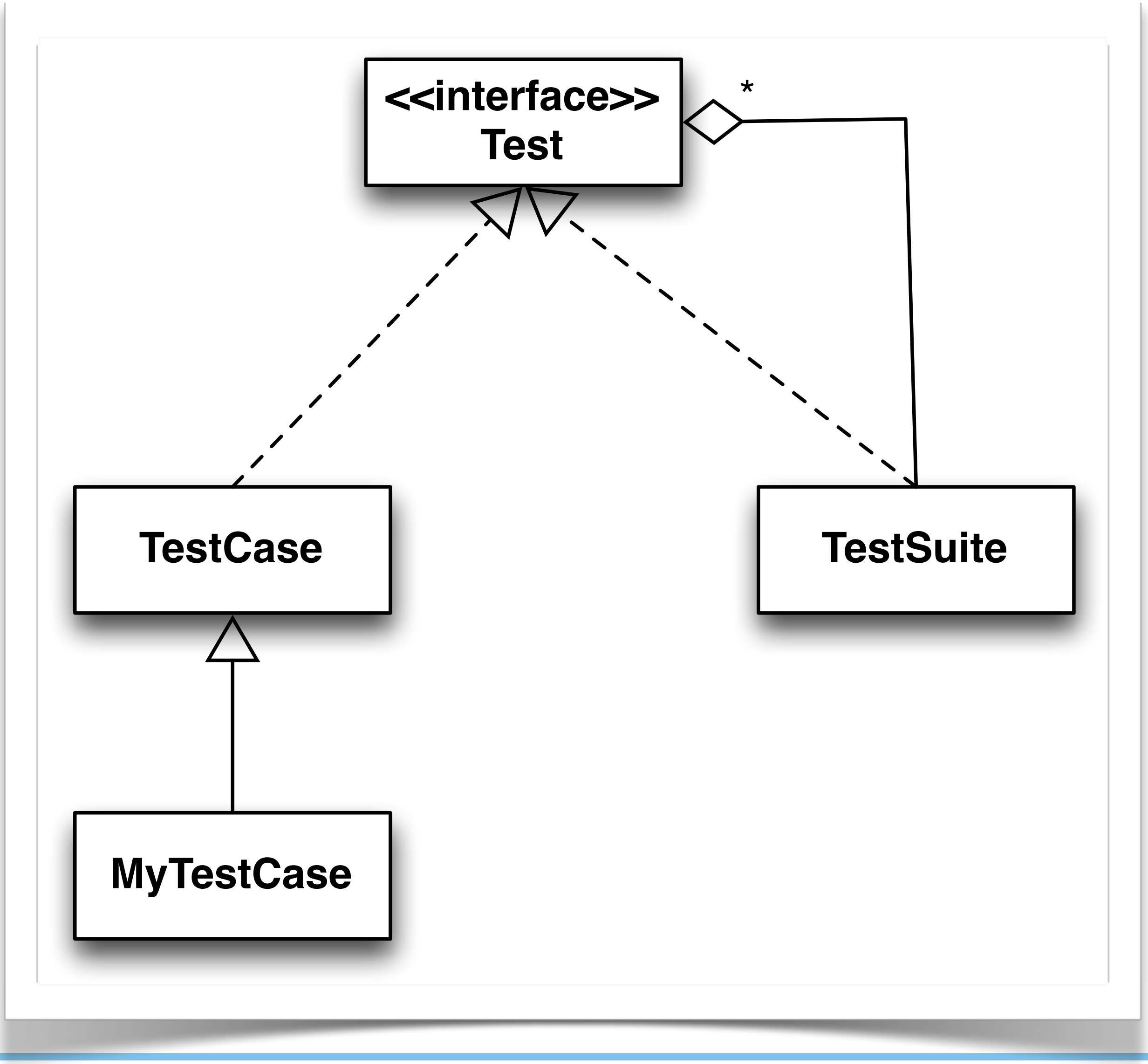
- JUnit features include:
  - Assertions for testing expected results
  - Test fixtures for sharing common test data
  - Test runners for running tests

- Originally written by Erich Gamma and Kent Beck.

xUnit Design

JUnit 3.x design was compliant with xUnit framework guidelines

- JUnit
- CppUnit
- PyUnit
- NUnit
- XMLUnit
- PHPUnit
- RUnit
- SUnit
- .....



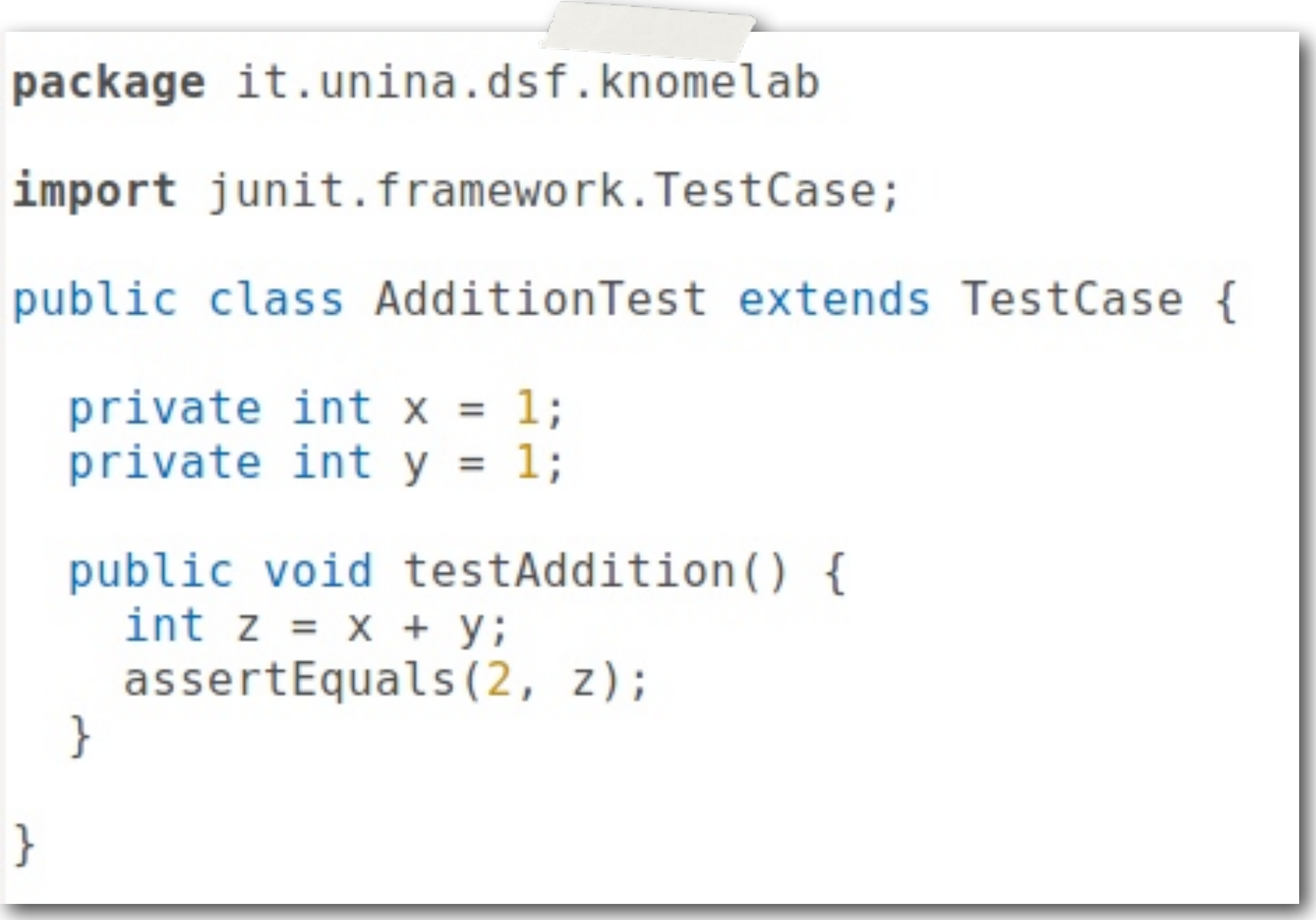
JUnit 3.x Design Rules

- All the Test classes **must** extend TestCase
  - Functionalities by inheritance

- All the test method's names **must** start with test to be executed by the framework
  - `TestSomething(...)`
  - `TestSomethingElse(...)`

- Let's do an example...

JUnit 3.x Typical Example



```java
package it.unina.dsf.knomelab

import junit.framework.TestCase;

public class AdditionTest extends TestCase {

  private int x = 1;
  private int y = 1;

  public void testAddition() {
    int z = x + y;
    assertEquals(2, z);
  }

}
```

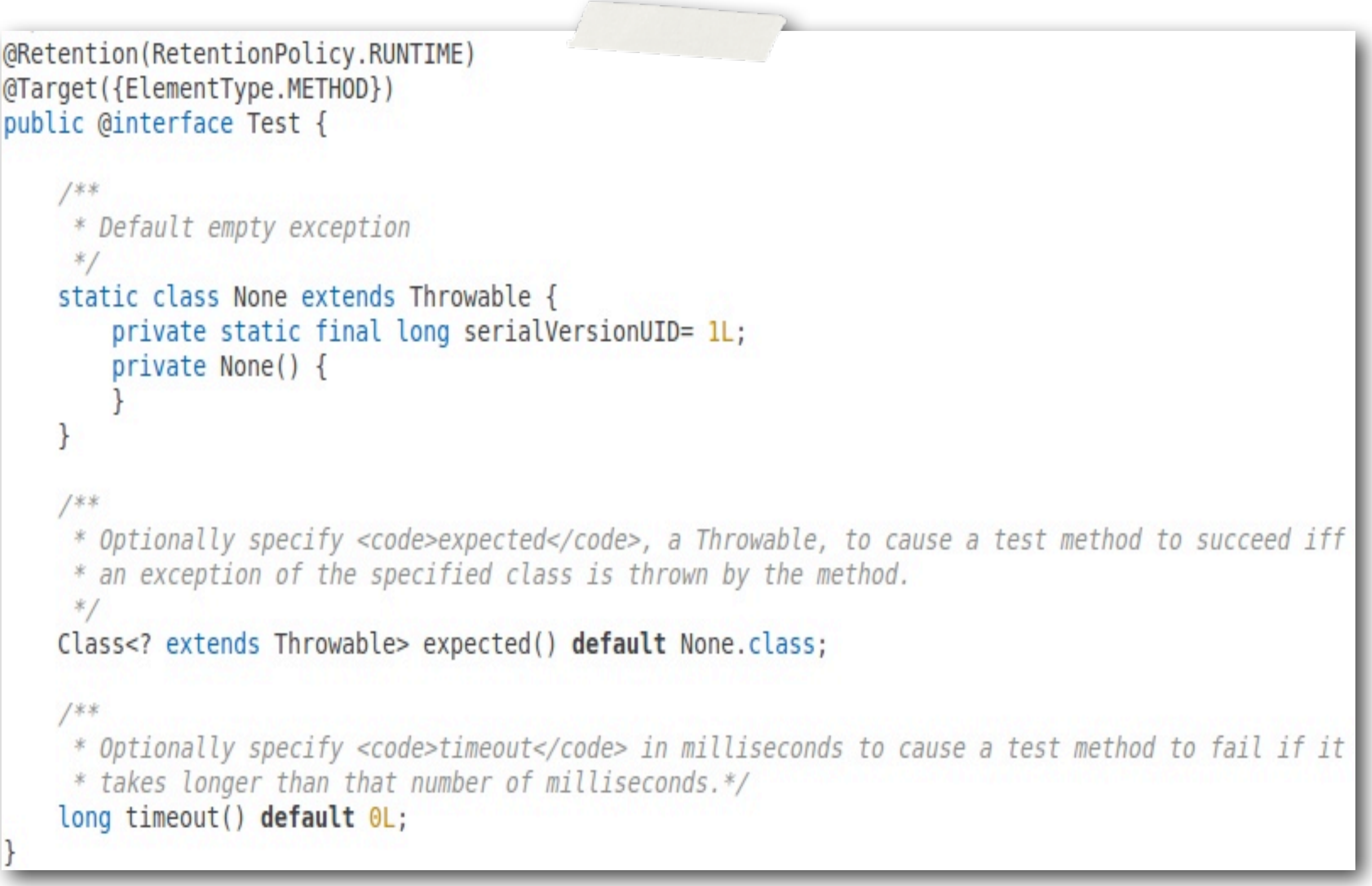
JUnit 4.x Design

- Main features inspired from other Java Unit Testing Frameworks
  - **TestNG**
- Test Method Annotations
  - Requires Java5+ instead of Java 1.2+
- Main Method Annotations
  - `@Before, @After`
  - `@Test, @Ignore`
  - `@SuiteClasses, @RunWith`

Java5 Annotations at glance

- Meta Data Tagging
  - `java.lang.annotation`
  - `java.lang.annotation.ElementType`
    - `FIELD`
    - `METHOD`
    - `CLASS`
    - …
- Target
  - Specify to which ElementType is applied
- Retention
  - Specify how long annotation should be available

JUnit Test Annotation



```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface Test {

    /**
     * Default empty exception
     */
    static class None extends Throwable {
        private static final long serialVersionUID= 1L;
        private None() {
        }
    }

    /**
     * Optionally specify <code>expected</code>, a Throwable, to cause a test method to succeed iff
     * an exception of the specified class is thrown by the method.
     */
    Class<? extends Throwable> expected() default None.class;

    /**
     * Optionally specify <code>timeout</code> in milliseconds to cause a test method to fail if it
     * takes longer than that number of milliseconds.*/
    long timeout() default 0L;
}
```

JUnit Testing Annotation (1)

- `@Test public void method()`
  - Annotation @Test identifies that this method is a test method.
- `@Before public void method()`
  - Will perform the method() before each test.
  - This method can prepare the test environment
  - E.g. read input data, initialize the class, ...
- `@After public void method()`

JUnit Testing Annotation (2)

- `@Ignore`
  - Will ignore the test method
  - E.g. Useful if the underlying code has been changed and the test has not yet been adapted.

- `@Test(expected=Exception.class)`
  - Tests if the method throws the named exception.

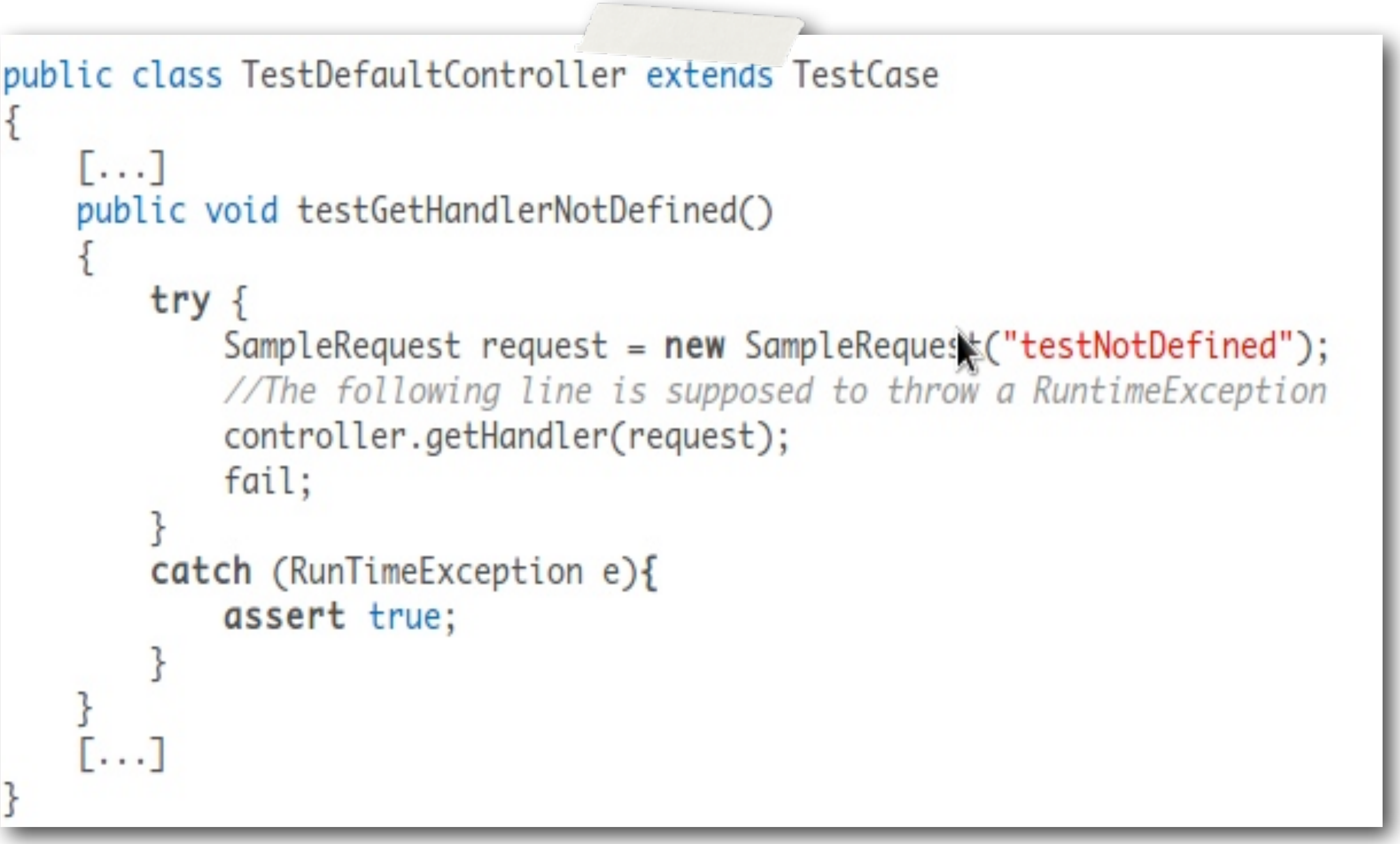
- `@Test(timeout=100)`
  - Fails if the method takes longer than 100 milliseconds.

JUnit Assert Statements

- `assertNotNull([message], object)`
  - Test passes if Object is not null.
- `assertNull([message], object)`
  - Test passes if Object is null.
- `assertEquals([message],expected, actual)`
  - Asserts equality of two values
- `assertTrue(true|false)`
  - Test passes if condition is True
- `assertNotSame([message], expected, actual)`
  - Test passes if the two Objects are not the same Object
- `assertSame([message], expected, actual)`
  - Test passes if the two Objects are the same Object

Testing Exception Handling

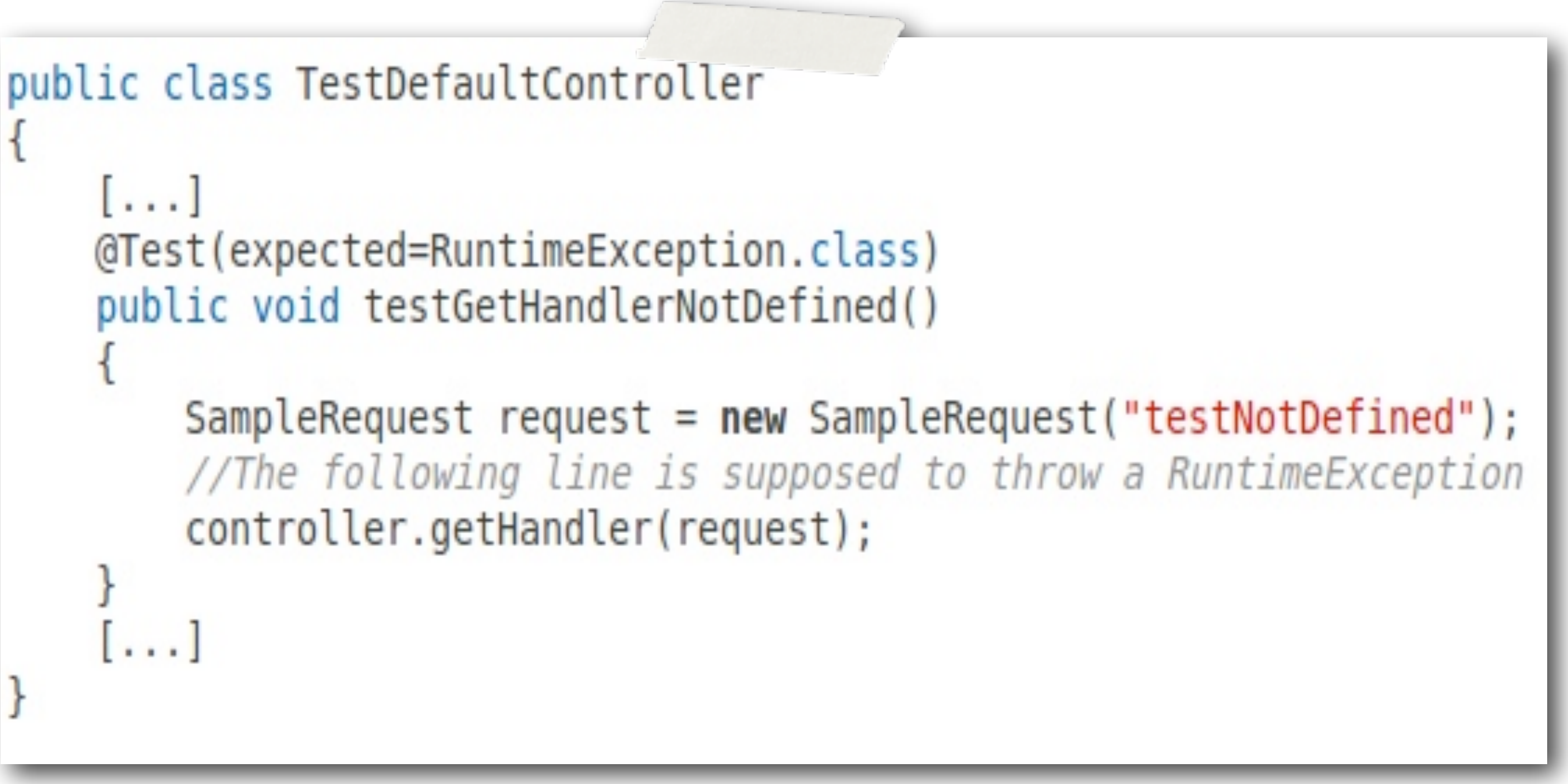
Test anything that could possibly fail



```
public class TestDefaultController extends TestCase
{
    [...]
    public void testGetHandlerNotDefined()
    {
        try {
            SampleRequest request = new SampleRequest("testNotDefined");
            //The following line is supposed to throw a RuntimeException
            controller.getHandler(request);
            fail;
        }
        catch (RunTimeException e){
            assert true;
        }
    }
    [...]
}
```

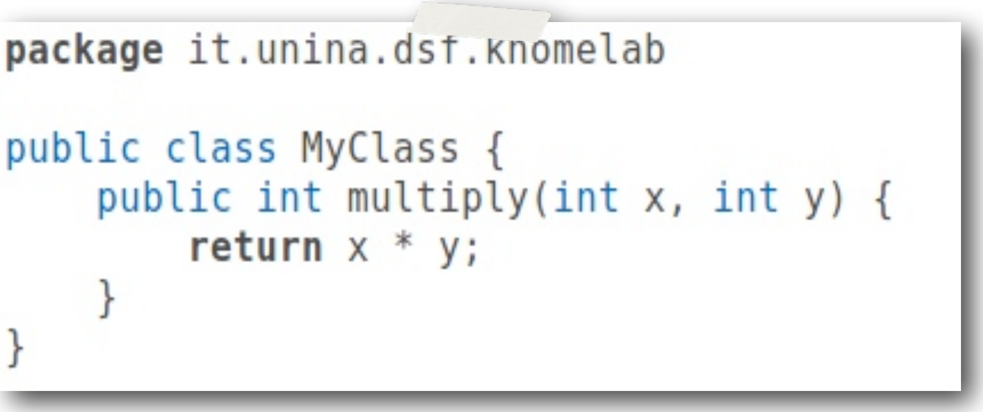
New way of Testing exception handling

Test anything that could possibly fail

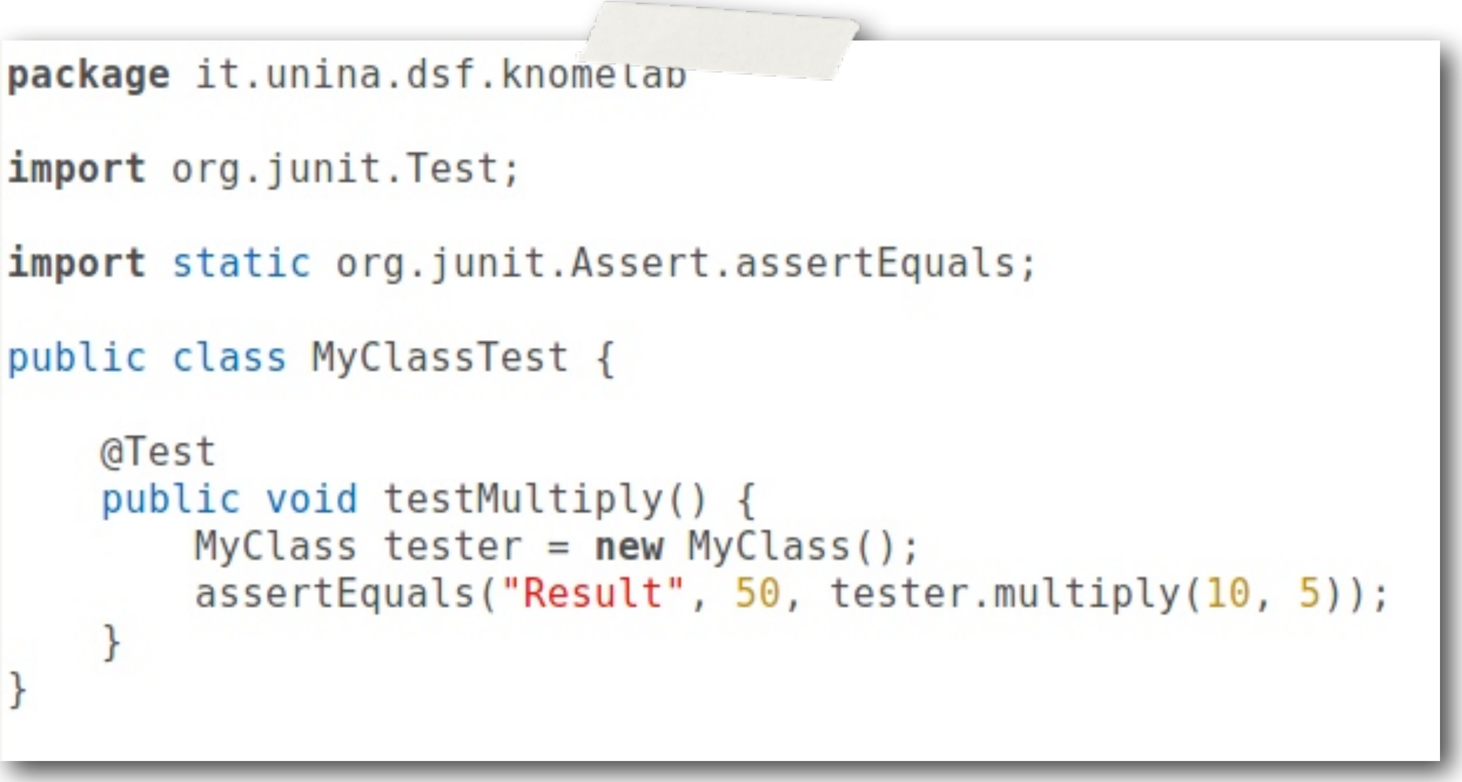


```
public class TestDefaultController
{
    [...]
    @Test(expected=RuntimeException.class)
    public void testGetHandlerNotDefined()
    {
        SampleRequest request = new SampleRequest("testNotDefined");
        //The following line is supposed to throw a RuntimeException
        controller.getHandler(request);
    }
    [...]
}
```

JUnit Example: TestCase and ClassUnderTest

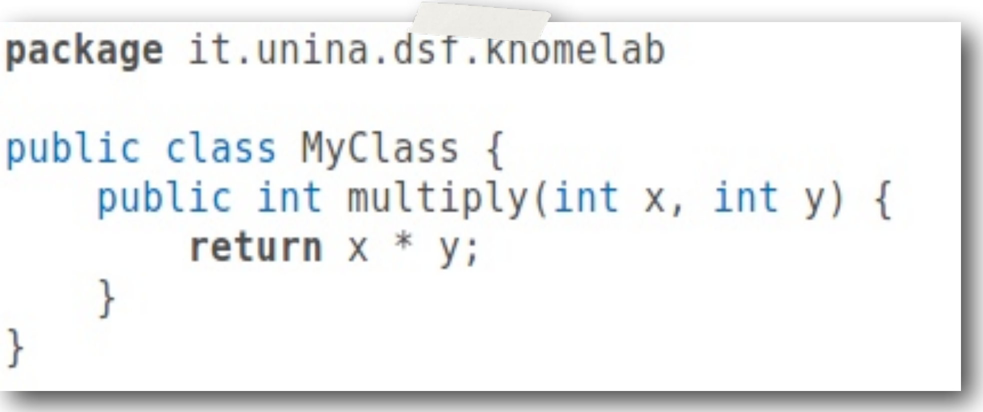


```java
package it.unina.dsf.knomelab

public class MyClass {
    public int multiply(int x, int y) {
        return x * y;
    }
}
```

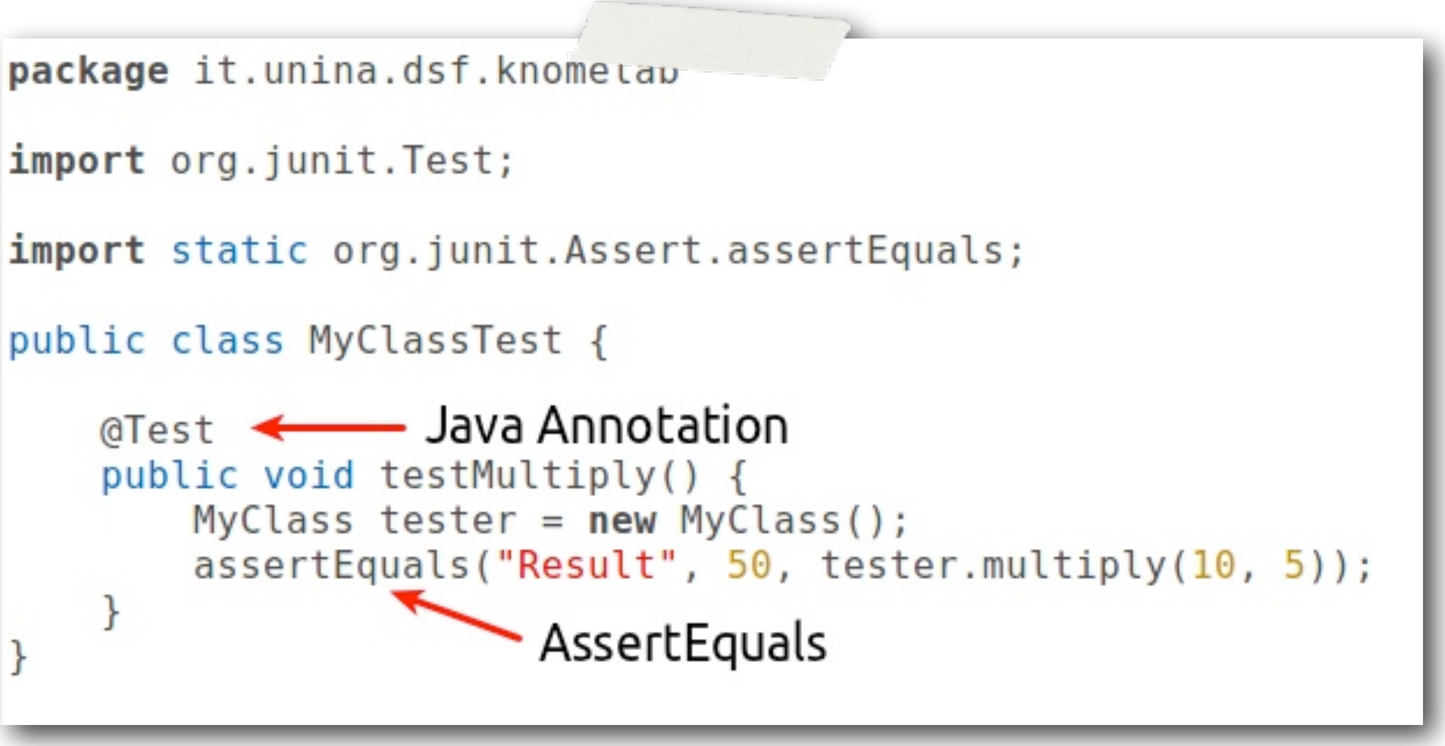


```java
package it.unina.dsf.knomelab

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MyClassTest {

    @Test
    public void testMultiply() {
        MyClass tester = new MyClass();
        assertEquals("Result", 50, tester.multiply(10, 5));
    }
}
```

JUnit Example: TestCase and ClassUnderTest



```java
package it.unina.dsf.knomelab

public class MyClass {
    public int multiply(int x, int y) {
        return x * y;
    }
}
```

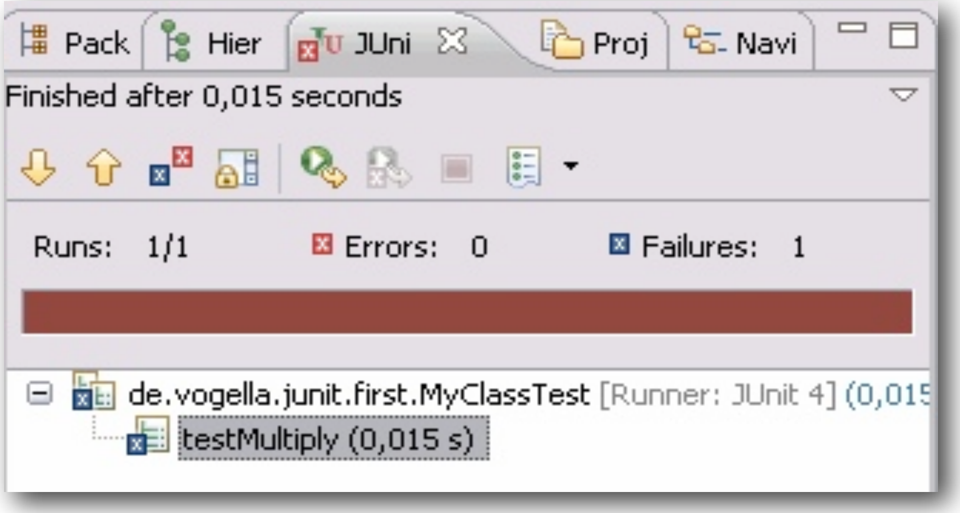


```java
package it.unina.dsf.knomelab

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MyClassTest {

    @Test
    public void testMultiply() {
        MyClass tester = new MyClass();
        assertEquals("Result", 50, tester.multiply(10, 5));
    }
}
```

Java Annotation

AssertEquals

JUnit Example: Execution

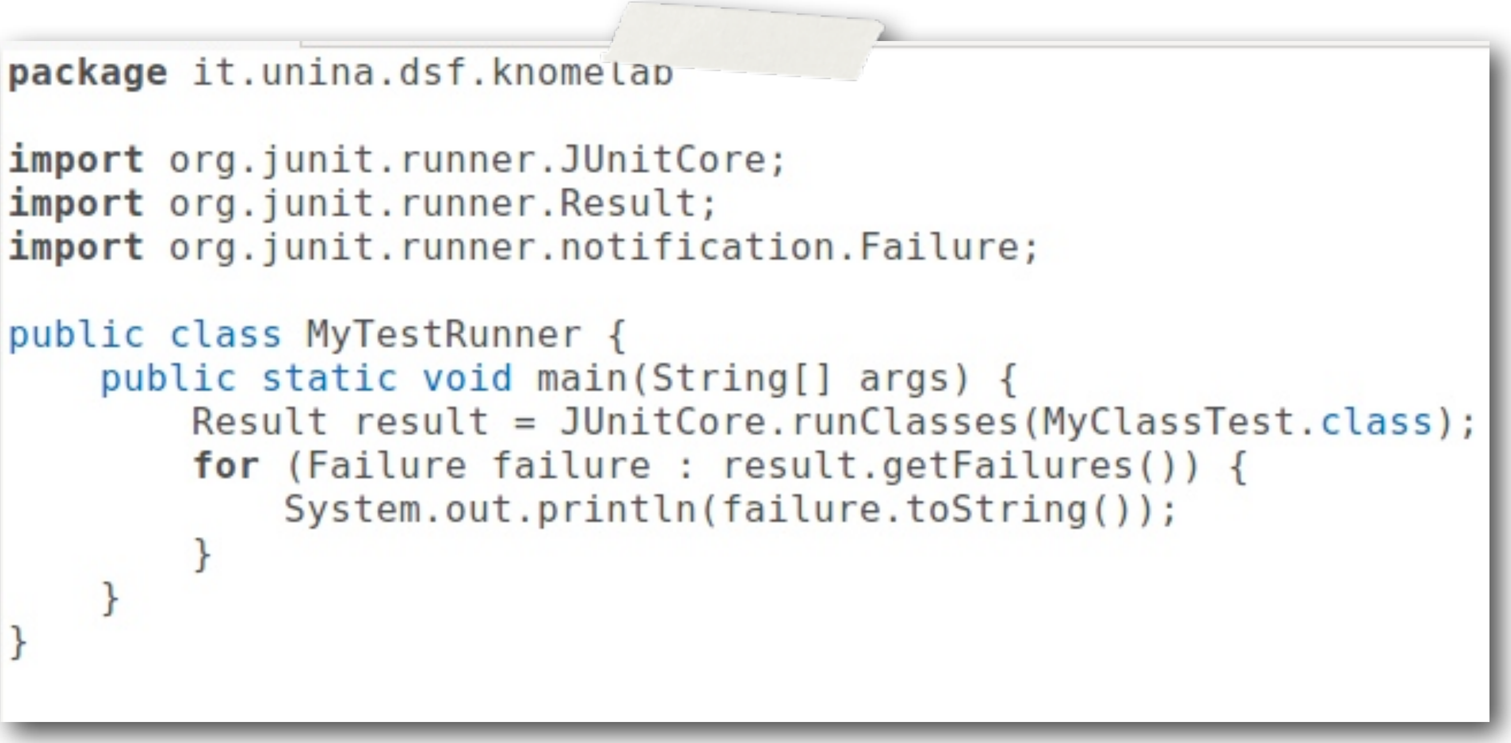


Finished after 0,015 seconds

Runs: 1/1 ☒ Errors: 0 ☒ Failures: 1

⊟ de.vogella.junit.first.MyClassTest [Runner: JUnit 4] (0,015

testMultiply (0,015 s)

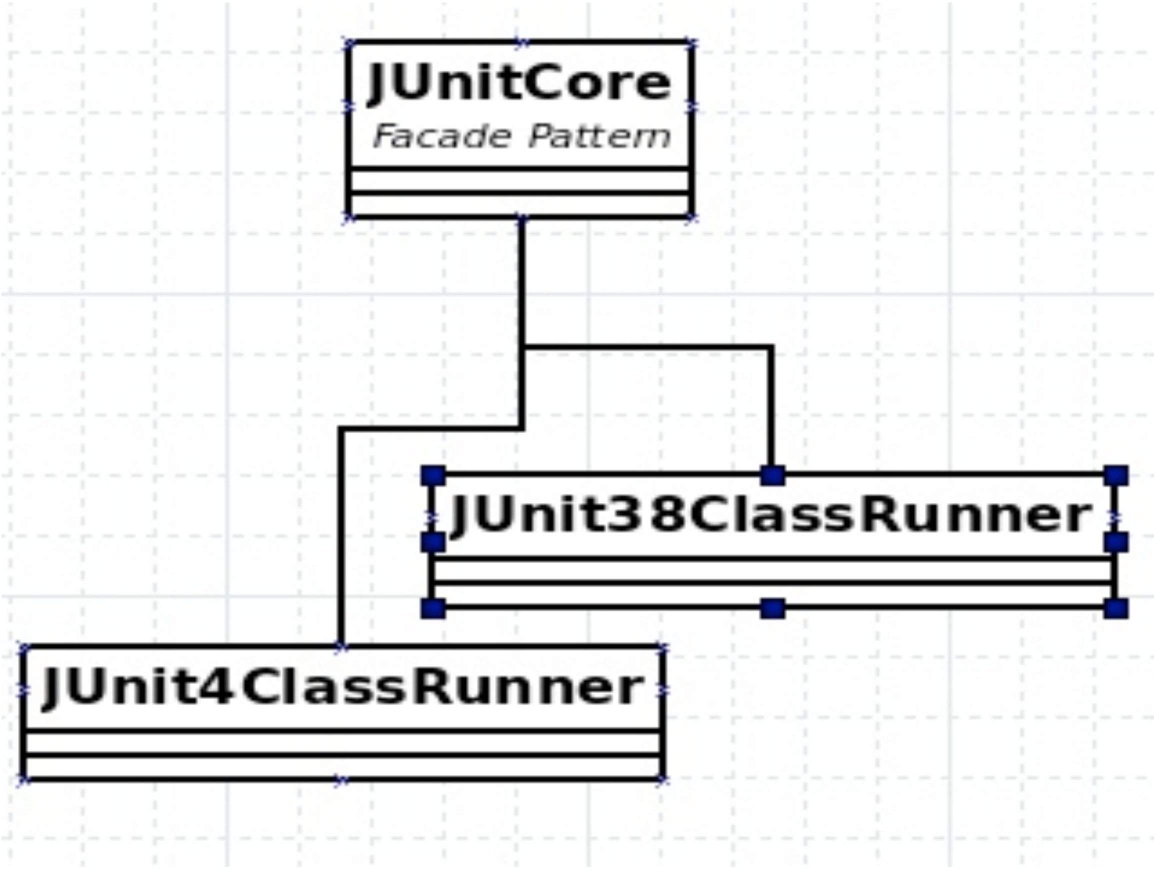


```
package it.unina.dsf.knomelab

import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class MyTestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(MyClassTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
    }
}
```

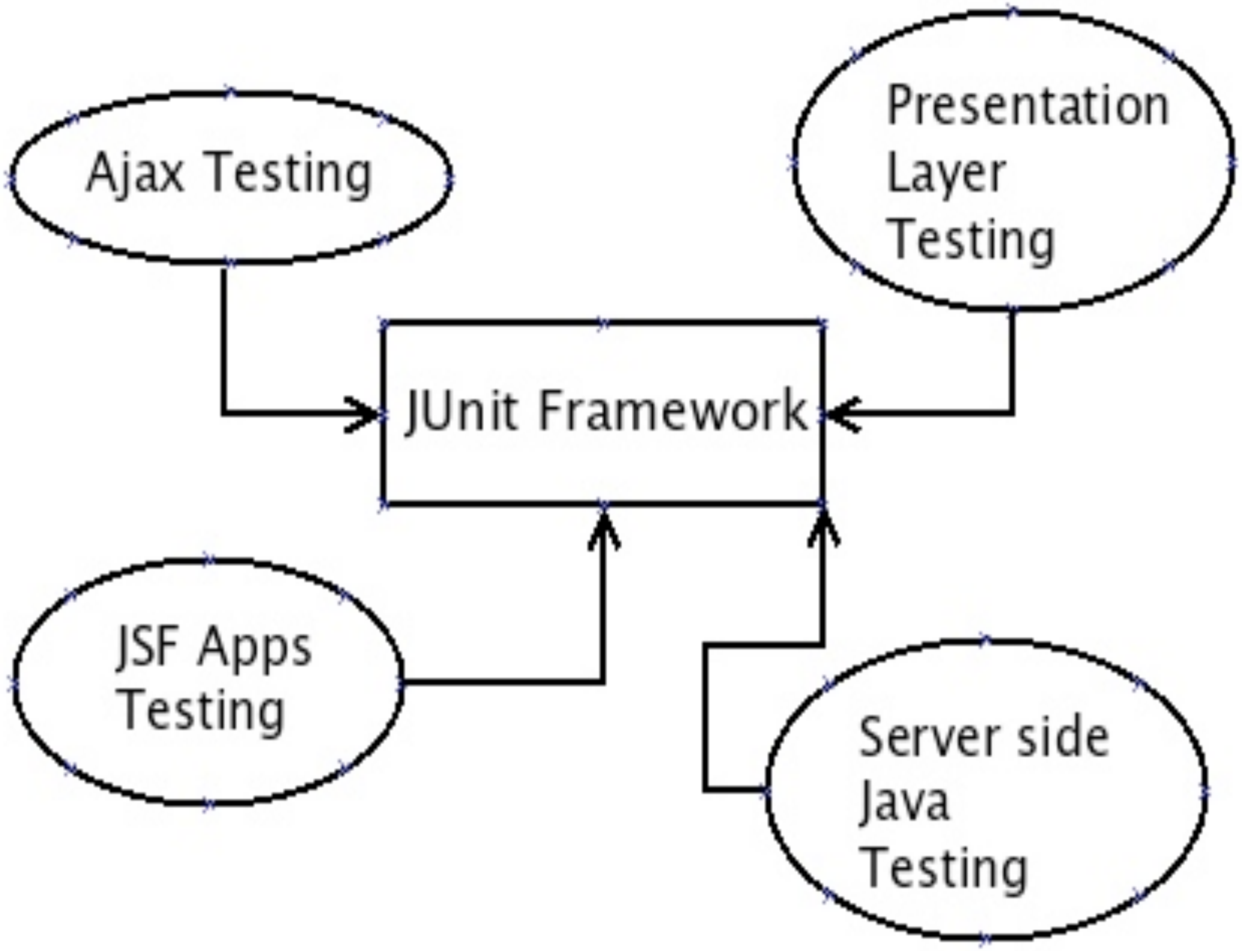
Further Insights

JUnit 4.x backward compatibility

- JUnit provides a façade class which operates with any of the test runners.
  - org.junit.runner.JUnitCore



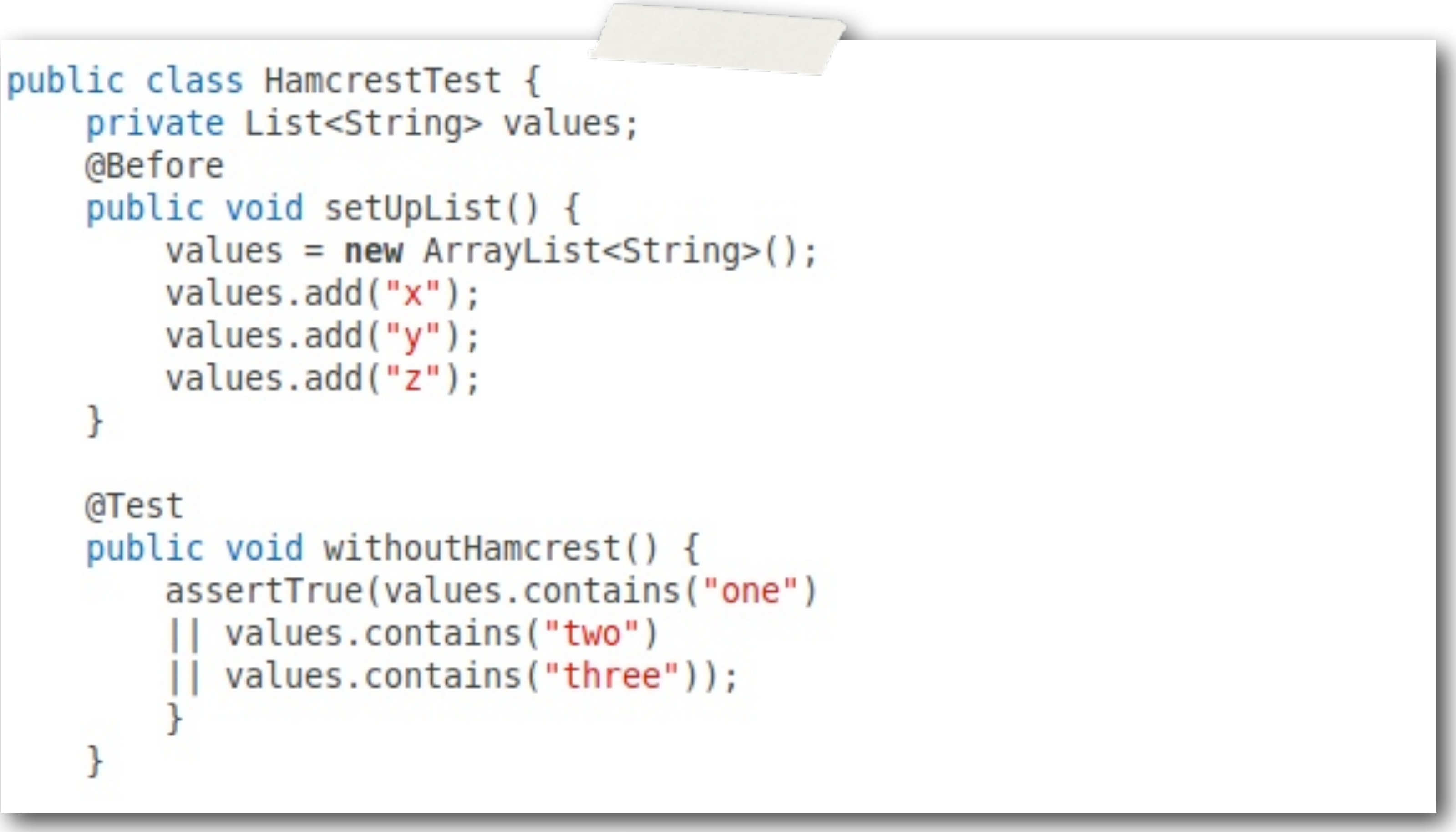
JUnit 4.x Extensions



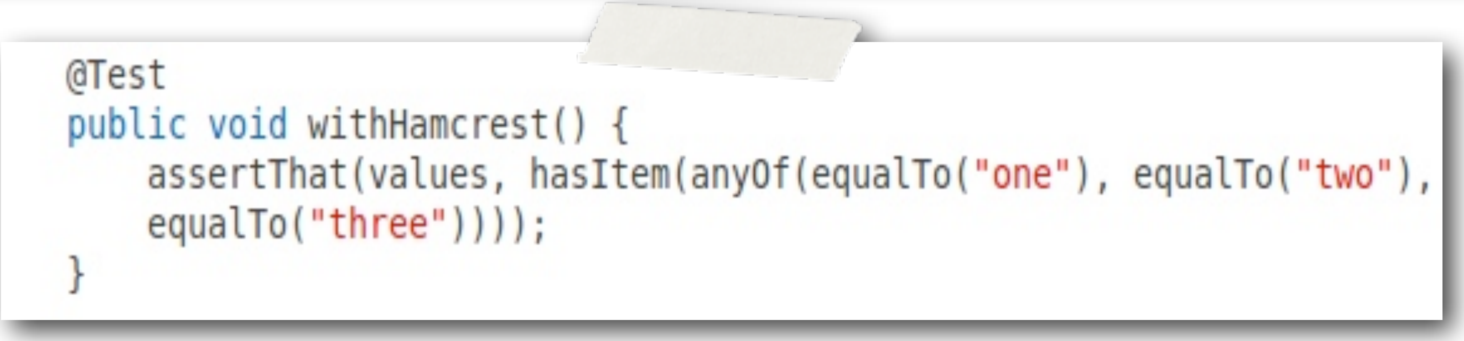
JUnit Matchers: Hamcrest

- Junit 4.4+ introduces matchers
  - Imported from Hamcrest project
  - http://code.google.com/p/hamcrest/
- Matchers improve testing code refactoring
  - Writing more and more tests assertion became hard to read
  - Remember:
    - Documentation purposes
- Let's do an example …

Matchers Example



```java
public class HamcrestTest {
    private List<String> values;
    @Before
    public void setUpList() {
        values = new ArrayList<String>();
        values.add("x");
        values.add("y");
        values.add("z");
    }

    @Test
    public void withoutHamcrest() {
        assertTrue(values.contains("one")
        || values.contains("two")
        || values.contains("three"));
        }
    }
```

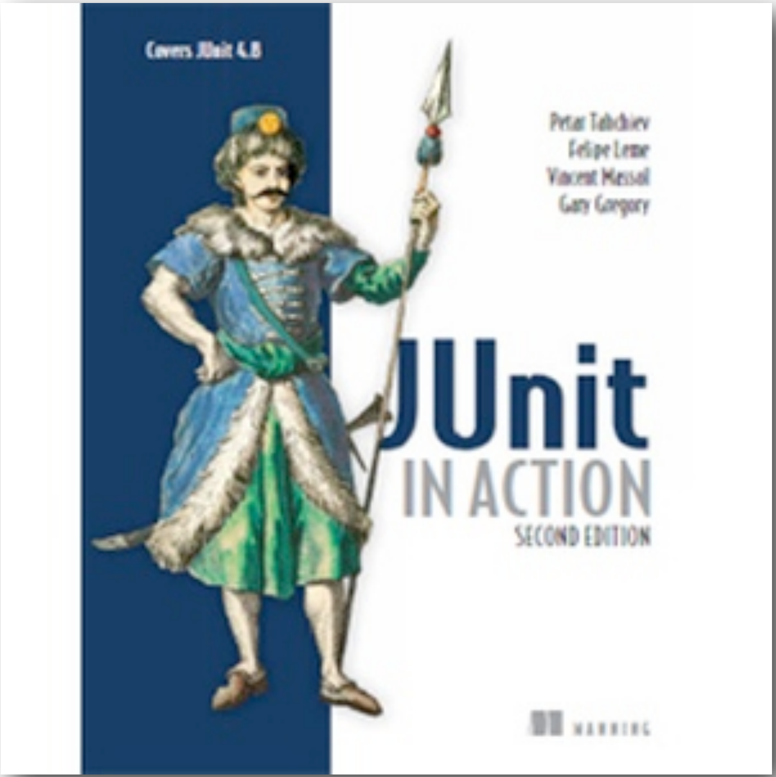


```java
@Test
public void withHamcrest() {
    assertThat(values, hasItem(anyOf(equalTo("one"), equalTo("two"),
    equalTo("three"))));
}
```

References 1/2

Professional Java JDK 5 Edition
Richardson et. al.,Wrox Publications 2006

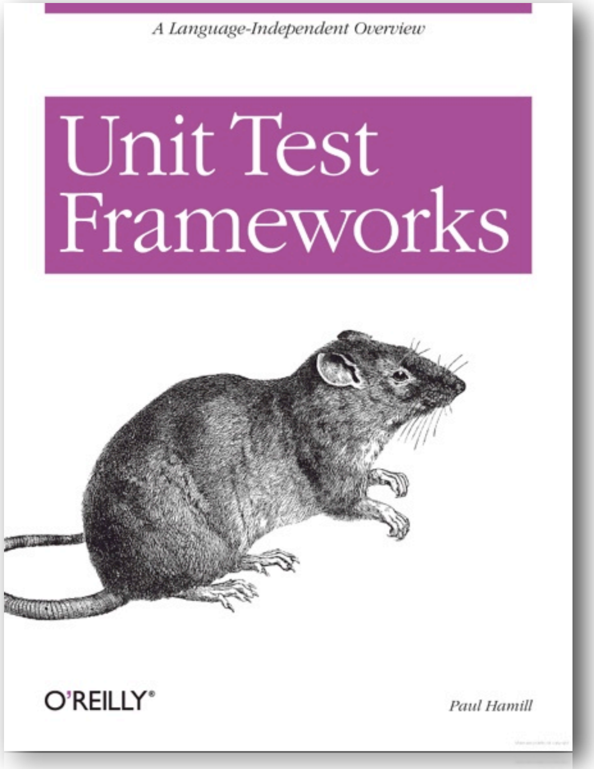
Programmer to Programmer™
Professional
Java™
JDK™ 5 Edition
W. Clay Richardson, Donald Avondolio, Joe Vitale, Scot Schrager, Mark W. Mitchell, Jeff Scanlon
wrox
Updates, source code, and Wrox technical support at www.wrox.com

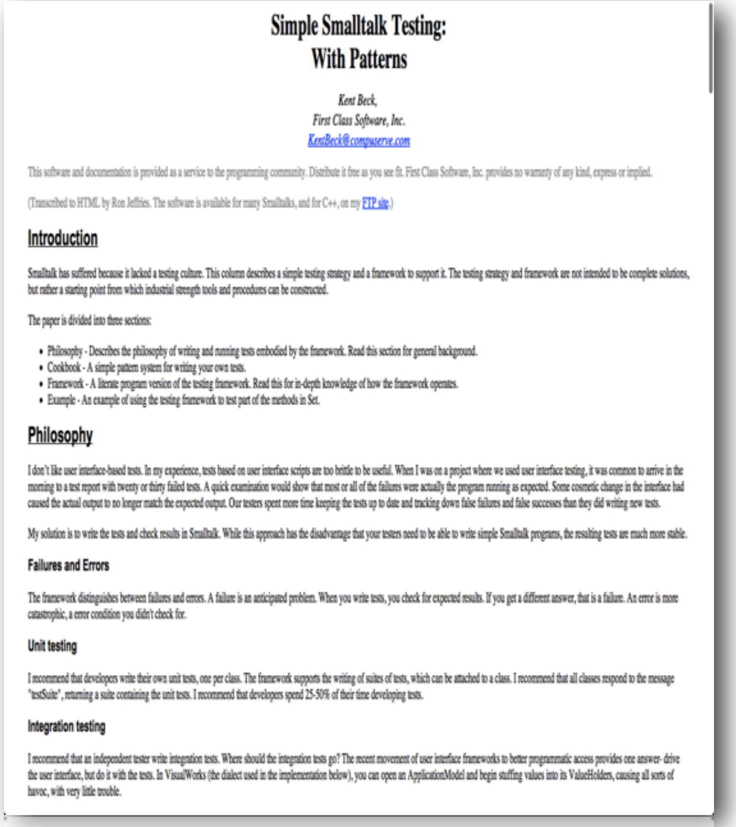


JUnit in Action, 2nd Ed.
Massol et al. , Manning Pubs 2009

References 2/2

Unit Test Frameworks
Tools for High-Quality Software Development
Paul Hamill, O'Reilly Media 2004





Simple Smalltalk Testing: With Patterns

Kent Beck,
First Class Software, Inc.
KentBeck@compuserve.com

This software and documentation is provided as a service to the programming community. Distribute it free as you see fit. First Class Software, Inc. provides no warranty of any kind, express or implied.

(Transcribed to HTML by Ron Jeffies. The software is available for many Smalltalks, and for C++, on my FTP site.)

Introduction

Smalltalk has suffered because it lacked a testing culture. This column describes a simple testing strategy and a framework to support it. The testing strategy and framework are not intended to be complete solutions, but rather a starting point from which industrial strength tools and procedures can be constructed.

The paper is divided into three sections:

- Philosophy - Describes the philosophy of writing and running tests embodied by the framework. Read this section for general background.
- Cookbook - A simple pattern system for writing your own tests.
- Framework - A literate program version of the testing framework. Read this for in-depth knowledge of how the framework operates.
- Example - An example of using the testing framework to test part of the methods in Set.

Philosophy

I don't like user interface-based tests. In my experience, tests based on user interface scripts are too brittle to be useful. When I was on a project where we used user interface testing, it was common to arrive in the morning to a test report with twenty or thirty failed tests. A quick examination would show that most or all of the failures were actually the program running as expected. Some cosmetic change in the interface had caused the actual output to no longer match the expected output. Our testers spent more time keeping the tests up to date and tracking down false failures and false successes than they did writing new tests.

My solution is to write the tests and check results in Smalltalk. While this approach has the disadvantage that your testers need to be able to write simple Smalltalk programs, the resulting tests are much more stable.

Failures and Errors

The framework distinguishes between failures and errors. A failure is an anticipated problem. When you write tests, you check for expected results. If you get a different answer, that is a failure. An error is more catastrophic, a error condition you didn't check for.

Unit testing

I recommend that developers write their own unit tests, one per class. The framework supports the writing of suites of tests, which can be attached to a class. I recommend that all classes respond to the message "testSuite", returning a suite containing the unit tests. I recommend that developers spend 25-50% of their time developing tests.

Integration testing

I recommend that an independent tester write integration tests. Where should the integration tests go? The recent movement of user interface frameworks to better programmatic access provides one answer- drive the user interface, but do it with the tests. In VisualWorks (the dialect used in the implementation below), you can open an ApplicationModel and begin stuffing values into its ValueHolders, causing all sorts of havoc, with very little trouble.

Kent Beck's Original Testing Framework Paper
http://www.xprogramming.com/testfram.htm