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.
JUnit 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...
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

```java
@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

```java
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

```java
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;
    }
}

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;
    }
}

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: Execution

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

Diagram:
- Ajax Testing
- Presentation Layer Testing
- JUnit Framework
- JSF Apps Testing
- Server side Java Testing
JUnitMatchers: Hamcrest

- JUnit 4.4+ introduces matchers
  - Imported from Hamcrest project

- Matchers improve testing code refactoring
  - Writing more and more tests assertion became hard to read
  - Remember:
    - Documentation purposes

- Let's do an example ...
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");
    }
    @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

JUnit in Action, 2nd Ed.
Massol et al., Manning Pubs 2009
Unit Test Frameworks
Tools for High-Quality Software Development
Paul Hamill, O'Reilly Media 2004

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