

软件测试基础与实践

Software Testing: Foundations and Practices

第5讲 JUnit测试工具

教师：汪鹏 廖力

软件工程专业 主干课程



本讲内容

JUnit基础

概念、安装、运行

JUnit使用

用**JUnit**进行单元测试



1. Why JUnit?



测试—println或IDE Debugger是不够的

■ 编码内测试的不足

现象:

1. 难以确定复杂系统是否正常运转
2. 测试代码难以维护



测试—println或IDE Debugger是不够的

■ 编码内测试的不足

原因:

测试缺乏系统规划

编码不断变化

测试代码规范性

注释不足

无法回归测试

无法完成“日创建-日测试”



新型软件开发方法的要求

■ 敏捷软件开发

目的:

应对快速变化的软件需求

特点/价值观:

1. 人和交互**重于**过程和工具
2. 可工作的软件**重于**完善的文档
3. 客户协作**重于**合同谈判
4. 随时应对变化**重于**循规蹈矩



新型软件开发方法的要求

■ 敏捷软件开发

代表方法：

极限编程（XP, eXtreme Programming）

与测试有关的强调：

测试驱动开发技术

（先写测试代码，再进行编码）



JUnit

如果编码没有“一对一”的可执行测试用例来证明编码能按照原设计稳定运行，则这些程序代码都是暗箱执行、无法维护、向后延续的各阶段都是灾难性的...

JUnit Framework是一个已经被多数Java程序员采用和证实的优秀测试框架。开发人员只需要按照JUnit的约定编写测试代码，就可以对自己要测试的代码进行测试...

——《软件测试与JUnit实践》

2. What is JUnit?



JUnit简介

- 开源的基于Java的单元测试框架
- 适用于测试驱动开发的开发模型
- 高度评价
 - Best Java Performance Monitoring/Testing Tool
 - 最重要的Java第三方库之一
 - 版本质量稳定
 -



JUnit简介

■ JUnit的诞生

Erich Gamma



设计模式开创者之一(四人帮之一)
Eclipse Java总设计师

Kent Beck

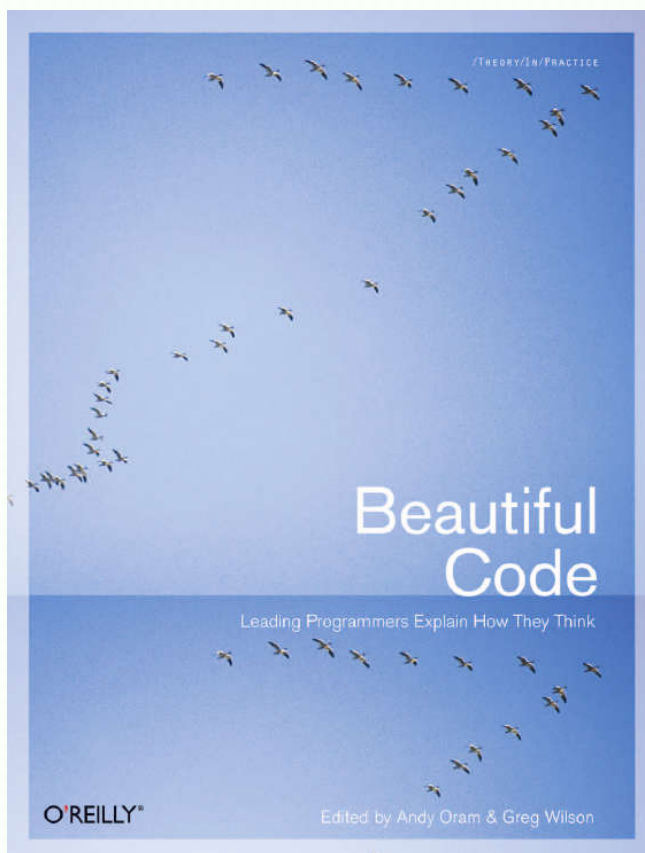


软件开发方法学大师
极限编程的创始人



JUnit简介

■ JUnit的诞生



- 7 BEAUTIFUL TESTS
by Alberto Savoia
That Pesky Binary Search
Introducing JUnit
Nailing Binary Search
Conclusion



JUnit简介

■设计目标

目标1:

提供一个测试框架，使开发人员方便地为程序写一个新测试，并避免付出重复努力

目标2:

提供一种管理测试用例的机制，使测试用例可以随时运行，并可以与后继测试用例一起对软件进行测试

目标3:

提供一种重用方式，使得测试框架能重用不同测试



JUnit简介




■ 版本

JUnit 4.x: 最新JUnit 4.9

JUnit 3.x: 经典JUnit 3.8

JUnit4.x之于JUnit3.x:
不是升级，而是框架的重新设计

<http://www.junit.org/>

junit » Downloads		
Download Packages		
 junit-4.9-SNAPSHOT-20100512-0041.jar	237KB · Uploaded May 12, 2010	246,386 downloads
 junit-dep-4.9-SNAPSHOT-20100512-0041.jar	219KB · Uploaded May 12, 2010	8,121 downloads
 junit-4.9-SNAPSHOT-20100512-0041-src.jar	129KB · Uploaded May 12, 2010	2,547 downloads
 junit4.9-SNAPSHOT-20100512-0041.zip	1.6MB · Uploaded May 12, 2010	9,999 downloads
 junit-4.8.2.jar	231KB · Uploaded April 09, 2010	106,356 downloads
 junit-dep-4.8.2.jar	213KB · Uploaded April 09, 2010	8,055 downloads
 junit-4.8.2-src.jar	127KB · Uploaded April 09, 2010	10,048 downloads
 junit4.8.2.zip	1.5MB · Uploaded April 09, 2010	32,907 downloads
 junit-4.8.1.jar	231KB · Uploaded December 08, 2009	5,845 downloads
 junit-dep-4.8.1.jar	212KB · Uploaded December 08, 2009	962 downloads



JUnit之前的单元测试

```
public class Car {  
    public int getWheels () {  
        return 4;  
    }  
}
```

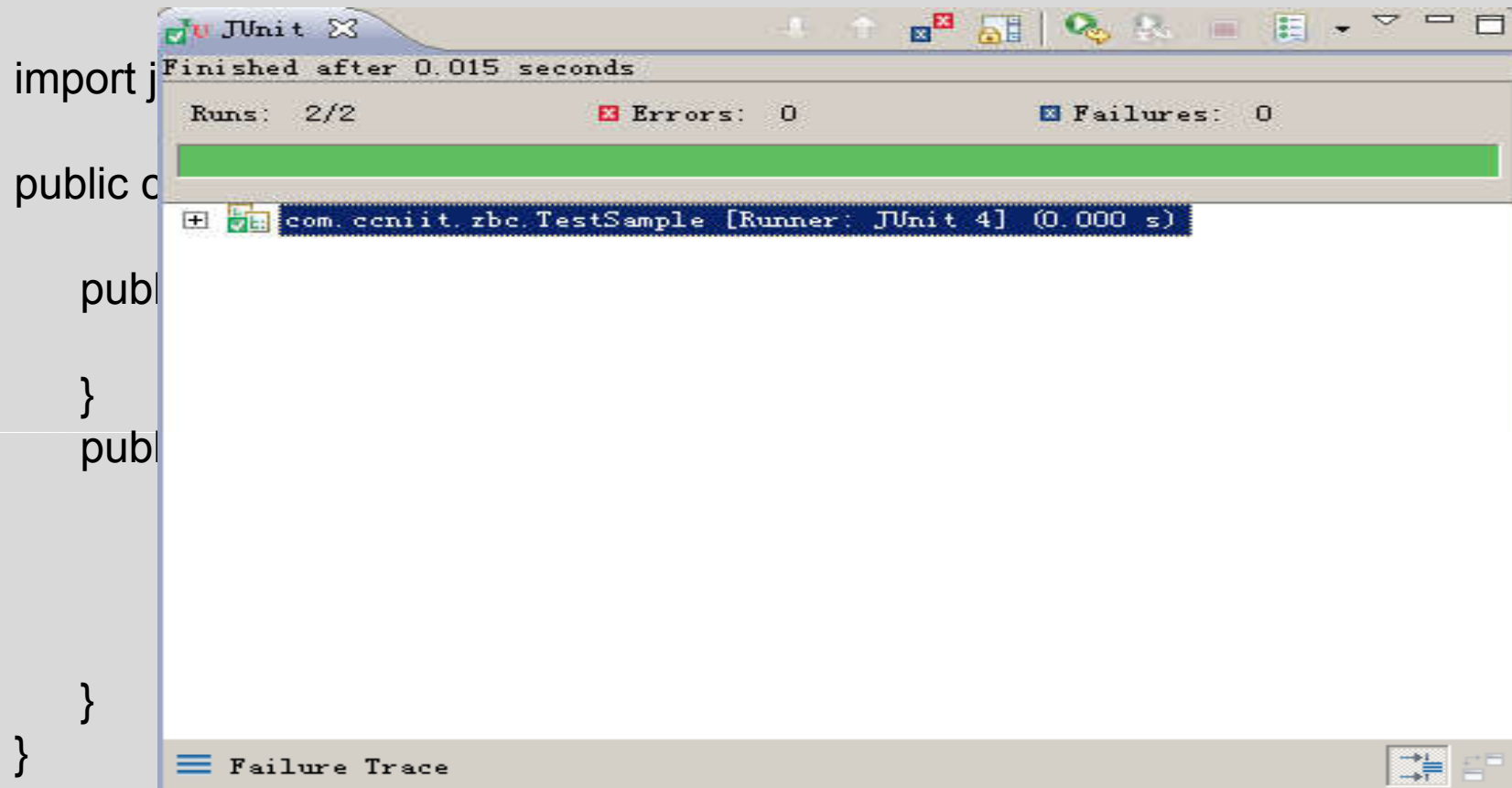
```
public class TestCar {  
    public static void main(String[] args) {  
        Car car = new Car();  
        if (4 == car.getWheels())  
            System.out.println("Ok!");  
        else  
            System.out.println("Error!");  
    }  
}
```

JUnit — TestCase

```
import junit.framework.TestCase;

public class Test extends TestCase {
    public Test(String name) {
        super(name);
    }
    public void setUp() {}
    public void tearDown(){}
    public void testHelloWorld() {
        // Test Code goes here
        String hello = "Hello";
        String world = "world";
        assertEquals("Hello world", hello+world);
    }
}
```


JUnit — TestSuite



绿色 — 测试通过
红色 — 测试失败

JUnit简介

■ JUnit益处

- 1.提高开发速度
- 2.提高代码质量
- 3.提升系统可信度
- 4.与其他开发框架结合: **Ant, JMock**
- 5.与主流IDE集成: **Eclipse, NetBeans**
- 6.测试代码与产品代码分开
- 7.提高测试代码重用率
- 8.方便对JUnit进行扩展和二次开发



JUnit简介

■ JUnit的应用

- 1.单元测试
- 2.面向对象测试
- 3.集成测试
- 4.功能测试
- 5.性能测试
-



3. Set up JUnit



JUnit简介

■ JUnit安装与运行

获取JUnit:

<http://www.junit.org/home>

JUnit文件组成:

- | | |
|------------|-------------------|
| 1.测试框架开发包: | junit*.jar |
| 2.源代码: | src.jar |
| 3.API文档: | \javadoc |
| 4.资料: | \doc |



JUnit简介

■ JUnit安装与运行

检验安装JUnit:

JUnit3.8:

`java -cp junit.jar;. junit.textui.TestRunner junit.samples.AllTests`

```
C:\WINDOWS\system32\cmd.exe

D:\DevWork\junit3.8.2>java -cp junit.jar;. junit.textui.TestRunner junit.samples
Class not found "junit.samples"

D:\DevWork\junit3.8.2>java -cp junit.jar;. junit.textui.TestRunner junit.samples.AllTests
.....
.....
.....
Time: 1.141
OK (131 tests)
```



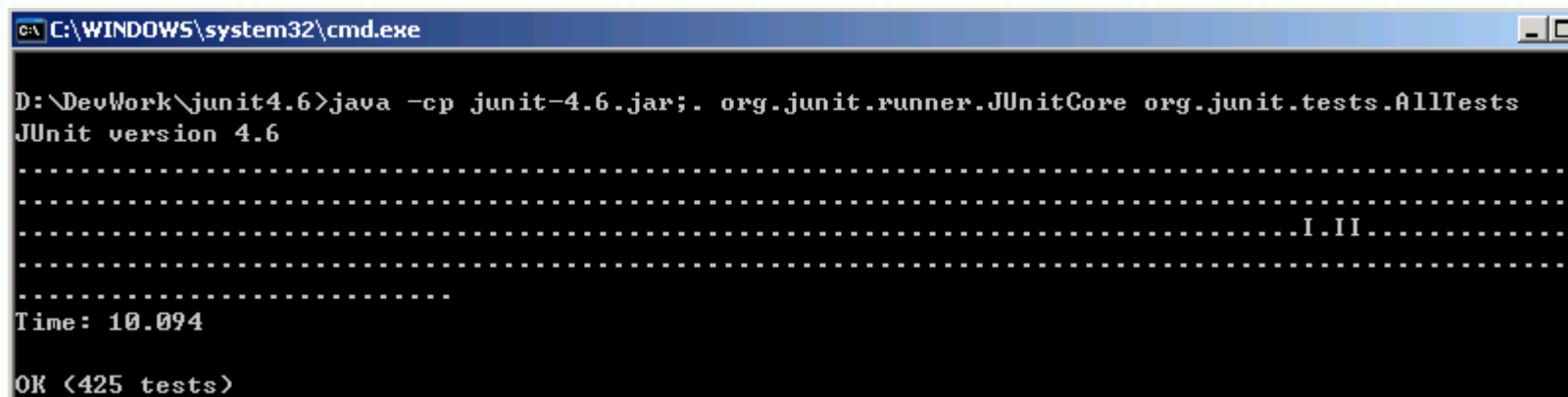
JUnit简介

■ JUnit安装与运行

检验安装JUnit:

JUnit4.6:

`java -cp junit-4.6.jar;. org.junit.runner.JUnitCore org.junit.tests.AllTests`



```
C:\WINDOWS\system32\cmd.exe

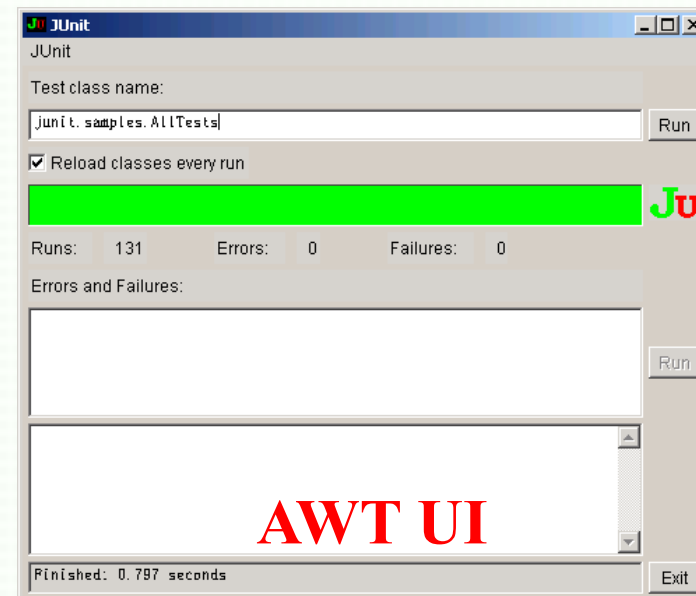
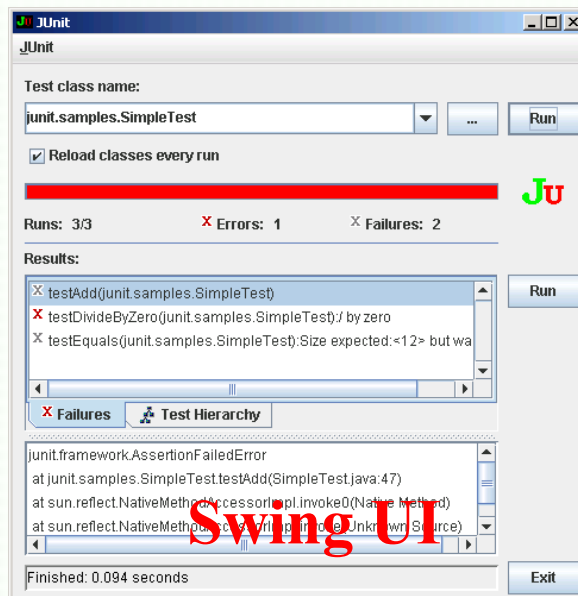
D:\DevWork\junit4.6>java -cp junit-4.6.jar;. org.junit.runner.JUnitCore org.junit.tests.AllTests
JUnit version 4.6
.....
.....I..II.....
.....
Time: 10.094

OK (425 tests)
```

JUnit简介

■ JUnit安装与运行 运行JUnit 3.x测试:

1. Text UI: `java -cp junit.jar;. Junit.textui.TestRunner <your_class>`
2. Swing UI: `java -cp junit.jar;. <your_classpath> junit.swingui.TestRunner`
3. AWT UI: `java -cp junit.jar;. <your_classpath> junit.awtui.TestRunner`



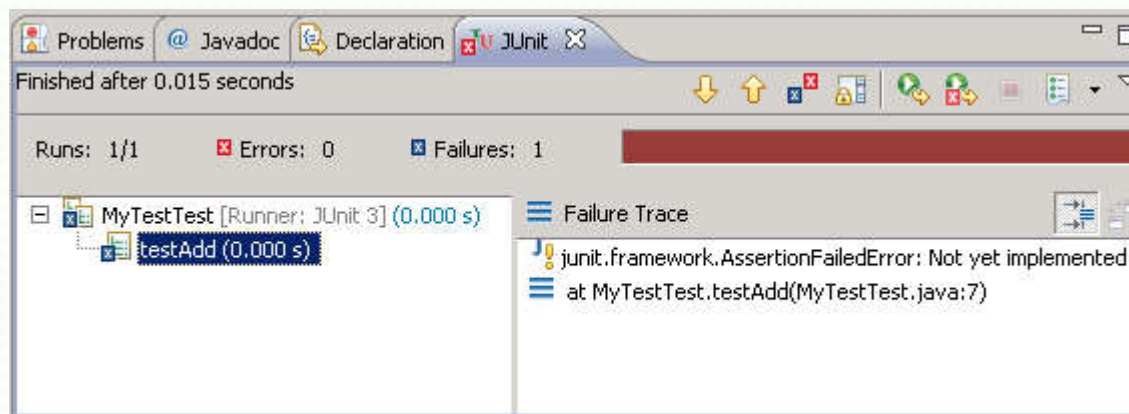
JUnit简介

■ JUnit安装与运行

运行JUnit 4.x测试:

1. Text UI: `java org.junit.runner.JUnitCore <your_class>`

JUnit的Eclipse插件:



4. How to use JUnit?

JUnit基础

■ JUnit 3.8的简单例子

JUnit3.8测试程序编写一般步骤:

Step1. 创建TestCase类的子类

Step2. 编写测试用例，每个测试用例是子类的方法

测试用例命名: **test+<TestCaseName>**

Public void test<TestCaseName>(){...}

Step3. 编写测试套件方法，将测试用例加入套件

Public static Test suite() {...}



JUnit基础

■ JUnit 3.8的简单例子

JUnit 3.8测试程序例子:

```
public class MyTestTest extends TestCase {  
    public void testMax() {  
        int x=Math.max(5,10);  
        assertTrue(x>=5 && x>=10);  
    }  
    public void testDivideByZero() {  
        int zero = 0;  
        int result=8/zero;  
    }  
    public static Test suite() {  
        return new TestSuite (MyTestTest.class);  
    }  
}
```

测试目标:

- 1.验证Java标准库函数
Math.max()方法
- 2.测试被零整除的结果



JUnit基础

■ JUnit 3.8的简单例子

JUnit 3.8测试程序例子：

程序要点：

- 1.测试类继承**TestCase**类
- 2.测试方法命名以**test**开头，**public**修饰，返回值**void**
- 3.**assertTrue(boolean)**帮助验证结果
- 4.测试套件方法**suite()**用**public static**修饰，返回值是**Test**类型
- 5.**TestSuite()**是把多个测试放入套件的方法之一



JUnit基础

■ JUnit 4.x的简单例子

JUnit 4.x测试程序例子:

```
public class MyTestTest {  
    @Test  
    public void maximum() {  
        int x=Math.max(5,10);  
        assertTrue(x>=5 && x>=10);  
    }  
    @Test  
    public void divideByZero() {  
        int zero = 0;  
        int result=8/zero;  
    }  
}
```

测试目标:

- 1.验证Java标准库函数 **Math.max()**方法
- 2.测试被零整除的结果



JUnit基础

■ JUnit 4.x的改进

- 1.重写了JUnit框架
- 2.利用Java1.5的Annotation特性简化测试用例编写
- 3.测试类不继承自TestCase
- 4.测试方法不必以test开头，只要以@Test描述



JUnit基础

■ JUnit 4.x的元数据

1. @Before

每个测试方法执行前都要执行

2. @After

每个测试方法执行后都要执行

3. @Test(expected=*.class)

测试方法应该抛出一个异常

4. @Test(timeout=xxx)

测试方法在给定时间内应完成

5. @ignore

测试方法会被忽略



JUnit基础

■ JUnit 4.x的元数据

6. @BeforeClass

在所有测试方法执行前执行一次

7. @AfterClass

在所有测试方法执行后执行一次



JUnit基础

■ JUnit的断言

断言用于判断测试是否通过

```
@Test
public void add() {
    Double result = 2+3;
    assertTrue (result==6);
}

@Test
public void equal() {
    double a=6, b=7;
    assertTrue (a==b);
}
}
```



JUnit基础

■ JUnit的断言

- 1.基础断言: `assertTrue()` `assertFalse()`
- 2.数值断言: `assertEquals()`
- 3.字符断言: `assertEquals()`
- 4.布尔断言: `assertEquals()`
- 5.比特断言: `assertEquals()`
- 6.对象断言: `assertEquals()` `assertNotNull`
 `assertSame()` `assertNotSame()`
- 7.其它断言: `assertArrayEquals()` `assertThat()` ...



JUnit基础

■ 失败(Failure)与错误(Error)

Failure:

断言失败，即预期的失败条件
被测编码有问题

Error:

不曾预料到的失败条件
发生其它异常，该异常未被预测到
测试本身或测试环境有问题

测试要求：先解决错误，再解决失败



JUnit基础

■ 测试套件(TestSuite)

作用:

控制测试用例的执行

注意:

不定义TestSuite时, JUnit将自动生成默认
TestSuite

只用于JUnit3.x



JUnit基础

■ 测试套件(TestSuite)

自动加入测试套件和手动加入测试套件:

```
public static Test suite() {  
    return new TestSuite (MyTestTest.class);  
}
```

```
public static Test suite() {  
    TestSuite suite = new TestSuite("MyTest");  
    suite.addTest(new MyTestTest(){protected void runTest(){testMax();}});  
    suite.addTest(new MyTestTest(){protected void runTest(){testDivideByZero();}});  
    return suite;  
}
```

JUnit基础

■ 测试套件(TestSuite)

JUnit4.x没有TestSuite

JUnit4.x通过@Ignore和打包测试组织测试用例

JUnit基础

■ 测试固件(Fixture)

目的:

将多个测试都能用到的操作统一管理，避免代码冗余，便于维护

JUnit3.x 两种固件: setUp()和tearDown()

JUnit4.x 固件: @Before/@After



JUnit基础

■ 测试固件

```
public class VectorTest extends TestCase {  
    protected Vector fEmpty, fFull;  
  
    protected void setUp() throws Exception {  
        fEmpty = new Vector();  
        fFull = new Vector();  
        fFull.addElement(new Integer(1));  
        fFull.addElement(new Integer(2));  
    }  
    protected void tearDown() throws Exception {  
        fEmpty = null;  
        fFull = null;  
    }  
    ...  
}
```

```
public void testCapacity() {  
    int size = fFull.size();  
    for (int i=0;i<100;i++)  
        fFull.addElement(new Integer(i));  
    assertTrue(fFull.size()==100+size);  
}  
public void testContains(){  
    assertTrue(fFull.contains(new Integer(1)));  
    assertTrue(!fEmpty.contains(new Integer(1)));  
}
```

JUnit 3.x VS JUnit 4.x

	JUnit 3.x	JUnit 4.x
1.框架	——	不是对3.x的改进，而是重新设计
2. package	junit.Framework.*	org.junit.*, 为兼容，发行两种 package
3.继承	测试类扩展 junit.framework.TestCase	不继承，但测试方法用@Test标注
4.断言	——	增加了两个新断言：比较数组对象
5. Fixture	setUp tearDown	@Before @After
6.测试方法命名	test+<TestCaseName>	不用test前缀，但要用@Test标注



JUnit 3.x VS JUnit 4.x

	JUnit 3.x	JUnit 4.x
7.忽略一个测试	注释, 改名	@Ignore标注
8.运行测试UI	text, AWT, SWing	text
9.测试集组织	suite()方法	@RunWith 和 @Suite标注一个空类
10.运行器	——	@RunWith
11.高级测试		预设环境@BeforeClass/@AfterClass 限时测试 参数化测试



5. Case study



JUnit4.x单元测试实例

■ 测试环境

JUnit版本: **JUnit 4.6**

Java 版本: **Java 1.6**

IDE 环境: **Eclipse 3.5.1**



JUnit4.x单元测试实例

■ 被测程序：Calculator类

```
public class Calculator {  
    private static int result; //静态变量  
    /*加函数*/  
    public void add(int n) {  
        result = result + n;  
    }  
    /*减函数*/  
    public void subtract(int n) {  
        result = result - 1; //Bug: 正确的应该是 result =result-n  
    }  
    /*尚未实现的方法*/  
    public void multiply(int n) {  
    }  
    /*除函数*/  
    public void divide(int n) {  
        result = result / n;  
    }  
}
```



JUnit4.x单元测试实例

■ 被测程序：Calculator类

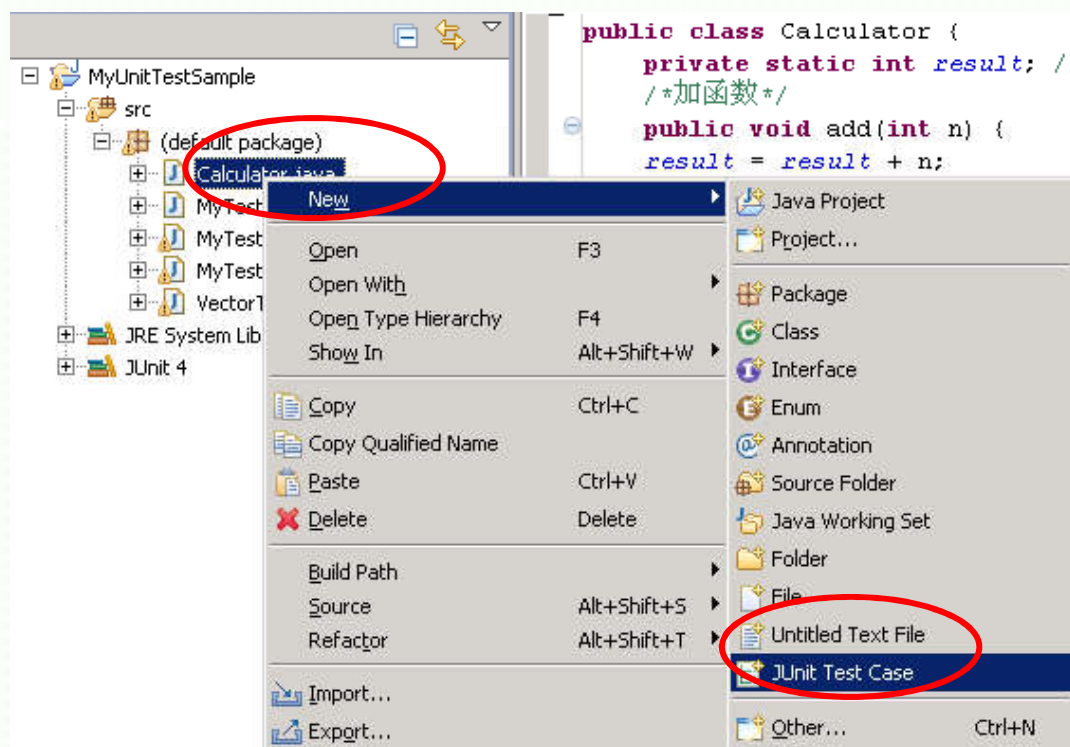
```
/*平方函数*/  
public void square(int n) {  
    result = n * n;  
}  
/*死循环*/  
public void squareRoot(int n) {  
    for (;;);  
}  
/*结果清零*/  
public void clear() {  
    result = 0;  
}  
/*返回结果*/  
public int getResult() {  
    return result;  
}  
}
```



JUnit4.x单元测试实例

■ 自动生成测试框架

Calculator.java → 右键菜单 → New → JUnit Test Case



JUnit4.x单元测试实例

■ 自动生成测试框架

JUnit Test Case

The use of the default package is discouraged.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: MyUnitTestSample/src Browse...

Package: (default) Browse...

Name: CalculatorTest

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☒ setUp() ☒ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Class under test: Calculator Browse...

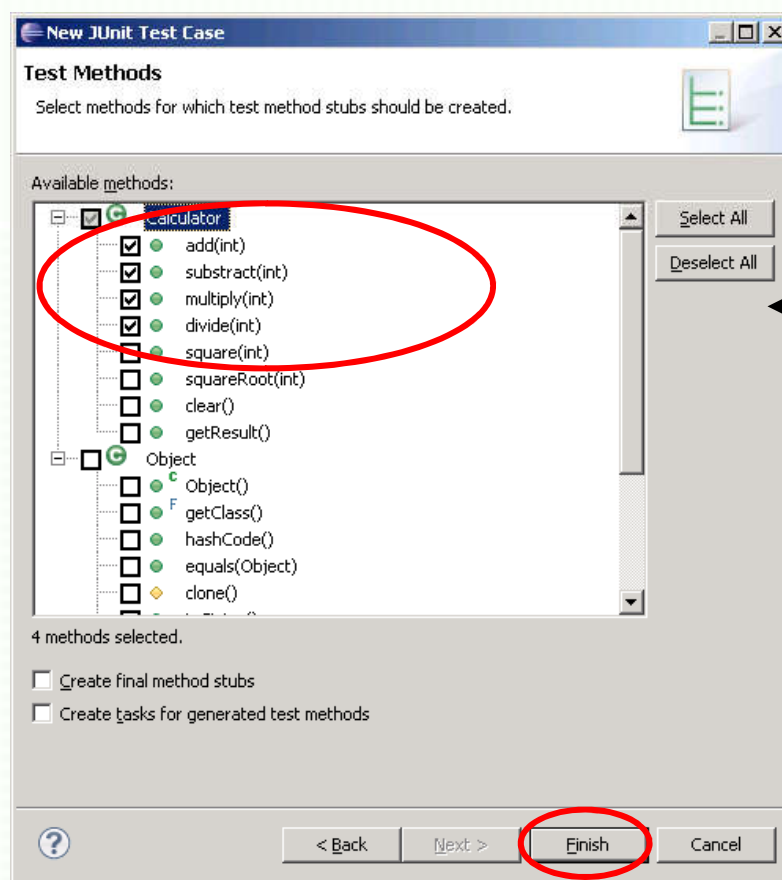
< Back Next > Finish Cancel

测试代码类名

选择固件方法

JUnit4.x单元测试实例

■ 自动生成测试框架



选择待测方法

JUnit4.x单元测试实例

■ 自动生成CalculatorTest类

```
public class CalculatorTest {  
    @Before  
    public void setUp() throws Exception {  
    }  
    @After  
    public void tearDown() throws Exception {  
    }  
    @Test  
    public void testAdd() {  
        fail("Not yet implemented");  
    }  
    .....  
}
```



JUnit4.x单元测试实例

■ 自动生成CalculatorTest类

```
@Test
public void testSubtract() {
    fail("Not yet implemented");
}

@Test
public void testMultiply() {
    fail("Not yet implemented");
}

@Test
public void testDivide() {
    fail("Not yet implemented");
}
}
```



JUnit4.x单元测试实例

■完善CalculatorTest类

```
public class CalculatorTest {  
    private static Calculator calculator = new Calculator(); //被测类实例  
    private static int nCount = 0; //测试方法统计  
  
    @Before  
    public void setUp() throws Exception {  
        calculator.clear(); //计算器归零  
    }  
  
    @After  
    public void tearDown() throws Exception {  
        nCount++; //计数，并显示  
        System.out.println("Test Done:"+nCount);  
    }  
}
```



JUnit4.x单元测试实例

■完善CalculatorTest类

```
@Test
public void testAdd() {
    /*验证2+3=5*/
    calculator.add(2);
    calculator.add(3);
    assertEquals(5, calculator.getResult());
}
```

```
@Test
public void testSubtract() {
    /*验证10-2=8*/
    calculator.add(10);
    calculator.subtract(2);
    assertEquals(8, calculator.getResult());
}
```



JUnit4.x单元测试实例

■完善CalculatorTest类

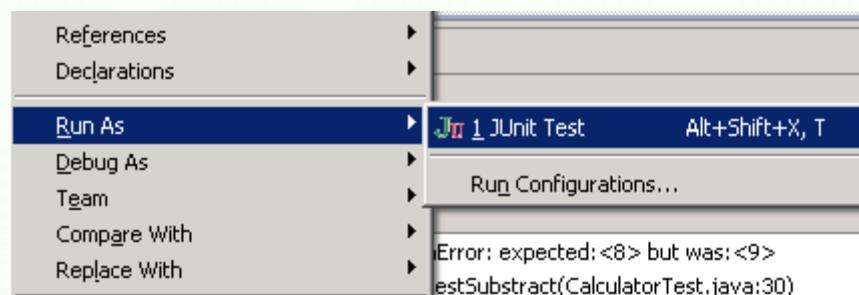
```
/*标记乘法未实现*/  
@Ignore("Multiply() Not yet implemented")  
@Test  
public void testMultiply() {  
    fail("Not yet implemented");  
}  
  
/*验证8/2=4*/  
@Test  
public void testDivide() {  
    calculator.add(8);  
    calculator.divide(2);  
    assertEquals(4, calculator.getResult());  
}
```



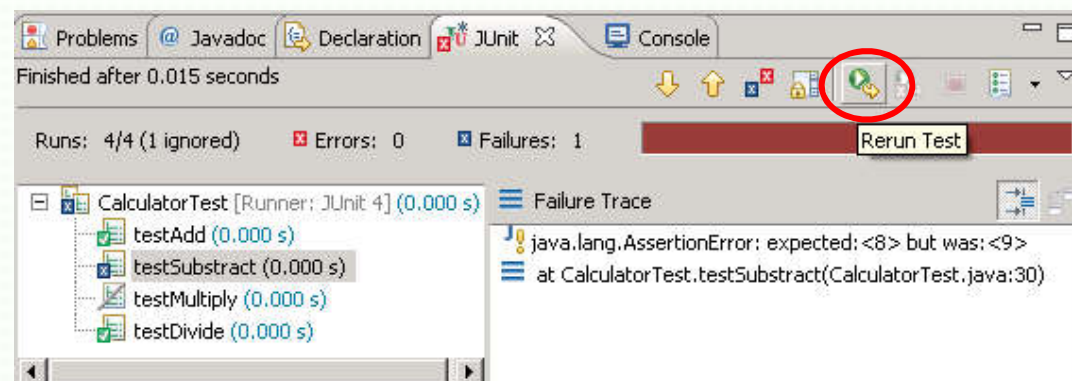
JUnit4.x单元测试实例

■ 执行CalculatorTest测试

方法1：右键菜单→Run As→JUnit Test

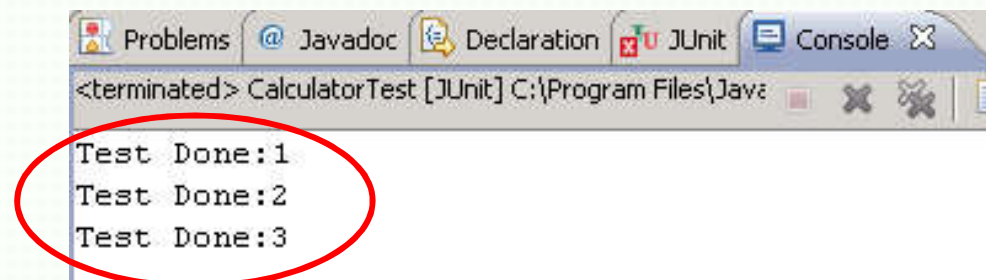
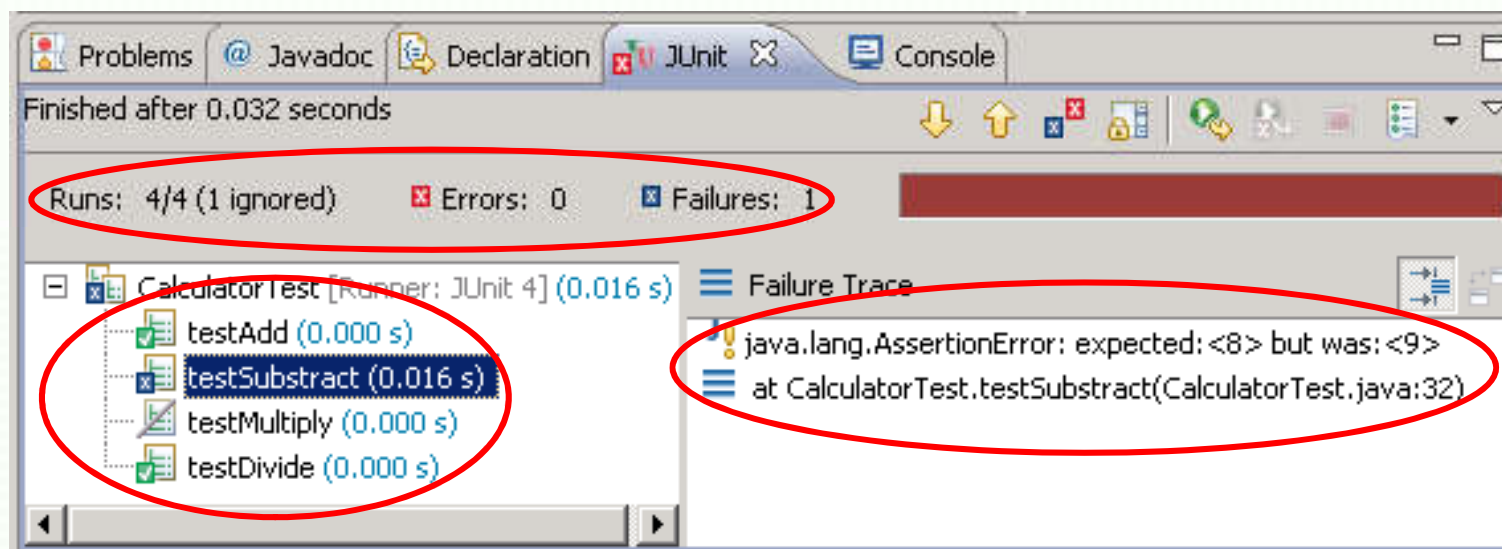


方法2：JUnit插件



JUnit4.x单元测试实例

■ CalculatorTest测试结果



JUnit4.x单元测试实例

■ CalculatorTest中增加限时测试

```
/*死循环*/  
public void squareRoot(int n) {  
    for (;;) ;  
}
```



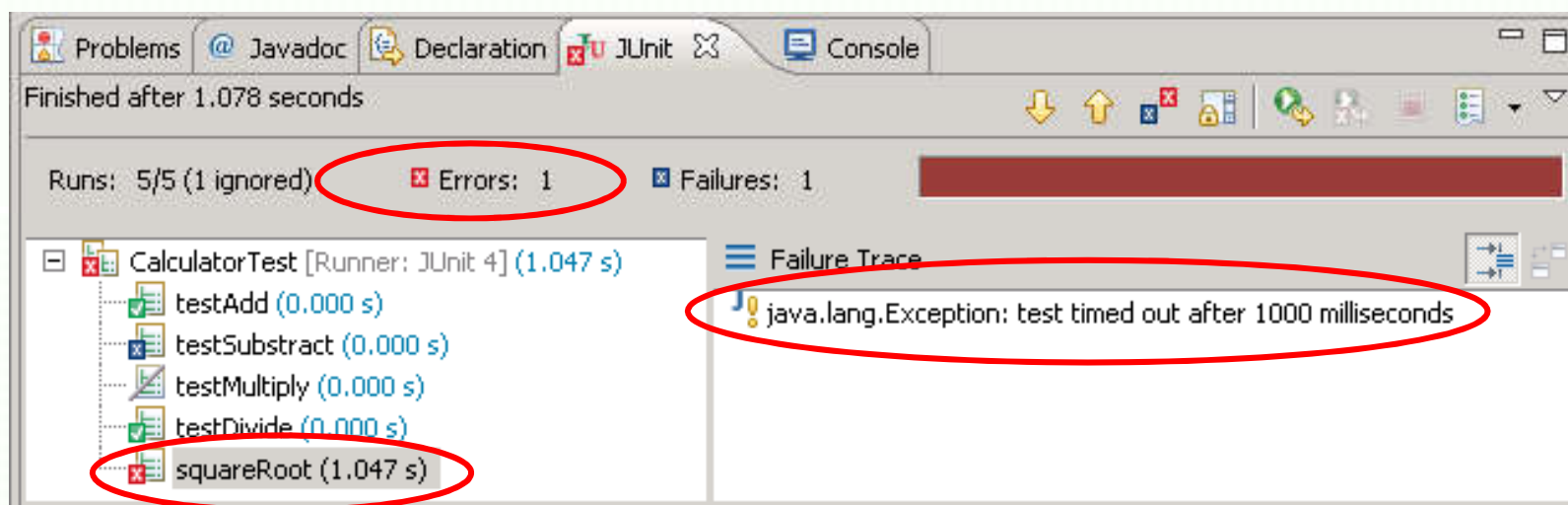
```
@Test(timeout = 1000)  
public void squareRoot() {  
    calculator.squareRoot(4);  
    assertEquals(2, calculator.getResult());  
}
```

用途：死循环预防；跳出复杂代码...



JUnit4.x单元测试实例

■ CalculatorTest中增加限时测试



JUnit4.x单元测试实例

■ CalculatorTest中增加测试异常

方法：@Test的expected属性

```
/*测试异常抛出*/  
@Test(expected = ArithmeticException.class)  
public void divideByZero() {  
    calculator.divide(0);  
}
```

用途：测试函数能正常抛出异常



JUnit4.x单元测试实例

■ 运行器Runner

Q: JUnit4.x框架如何运行测试代码?

A: Runner

未指定Runner时: JUnit用默认Runner来运行测试代码

指定一个Runner: 需要用@RunWith标注, 并把指定的Runner作为参数

@RunWith用来修饰类



JUnit4.x单元测试实例

■ 运行器Runner

```
import org.junit.internal.runners.TestClassRunner;  
import org.junit.runner.RunWith;  
@RunWith(TestClassRunner.class)  
  
public class CalculatorTest {}
```



JUnit4.x单元测试实例

■ 参数化测试

例:测试 x^2 ,分3种情况: $x>0$, $x=0$ 和 $x<0$

```
@Test
public void square1() {
    calculator.square(2);
    assertEquals(4, calculator.getResult());
}
@Test
public void square2() {
    calculator.square(0);
    assertEquals(0, calculator.getResult());
}
@Test
public void square3() {
    calculator.square(-3);
    assertEquals(9, calculator.getResult());
}
```



JUnit4.x单元测试实例

■ 参数化测试

利用参数化测试简化测试代码:

```
@RunWith(Parameterized.class)
```

```
public class SquareTest {  
    private static Calculator calculator = new Calculator();  
    private int param;  
    private int result;
```

```
@Parameters
```

```
public static Collection data() {  
    return Arrays.asList(new Object[][]{{2, 4},{0, 0},{-3, 9}});  
}
```

```
.....
```



JUnit4.x单元测试实例

■ 参数化测试

利用参数化测试简化测试代码:

```
/*构造函数*/  
public SquareTest(int param, int result) {  
    this.param = param;  
    this.result = result;  
}  
  
@Test  
public void square() {  
    calculator.square(param);  
    assertEquals(result, calculator.getResult());  
}  
}
```



JUnit4.x单元测试实例

■参数化测试

特点:

1. 为参数化测试专门生成一个类
2. 为该测试类指定Runner:

@RunWith(Parameterized.class)

3. 指定参数和预期结果，用**@Parameters**修饰



JUnit4.x单元测试实例

■打包测试

将两个测试类进行打包测试：

```
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;
```

```
@RunWith(Suite.class)  
@Suite.SuiteClasses({  
    CalculatorTest.class,  
    SquareTest.class  
})
```

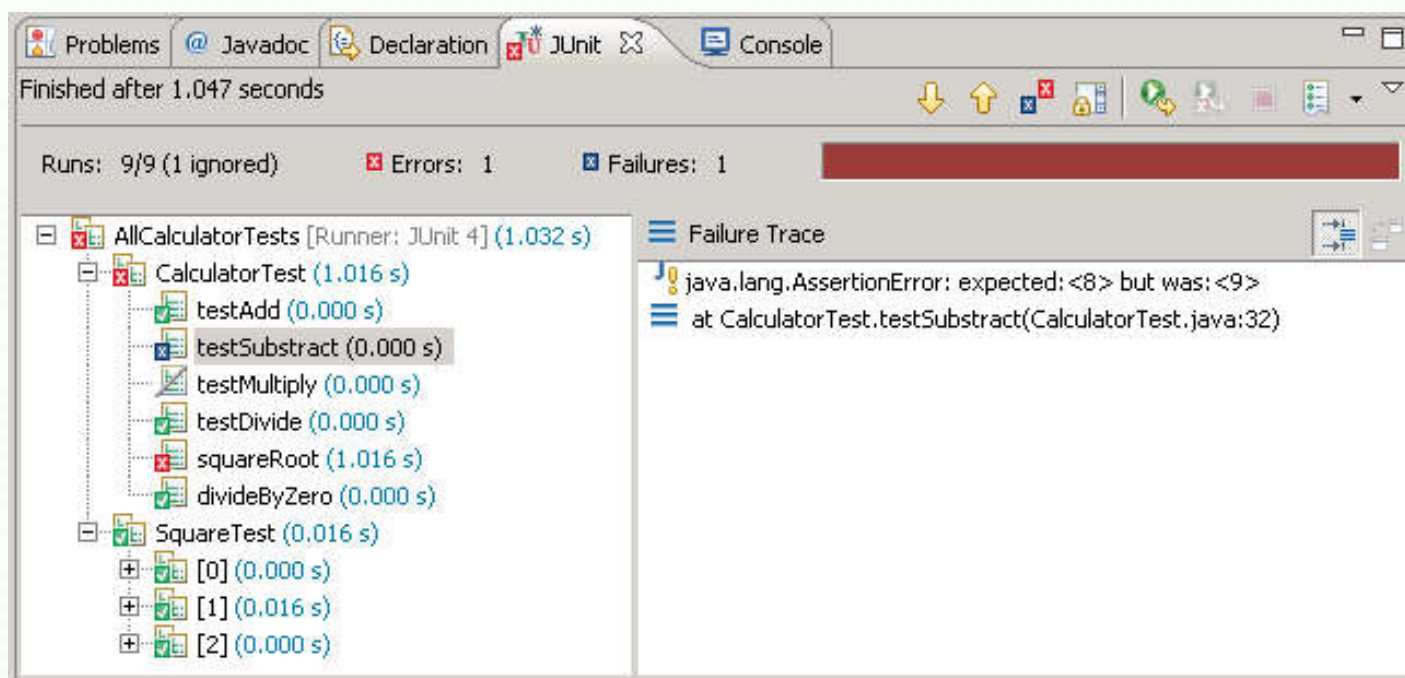
```
public class AllCalculatorTests {  
}
```



JUnit4.x单元测试实例

■打包测试

将两个测试类进行打包测试：



JUnit4.x单元测试实例

■打包测试

要点:

1. **@RunWith**标注传递参数**Suite.class**
2. **@Suite.SuiteClasses**标注该类是打包测试类
3. 具体类实现可为空



JUnit4.x单元测试实例

■打包测试

目的:

1. 将测试类按顺序进行组织
2. 对测试进行集中规划，提高测试效率



6. 模仿对象测试



模仿对象测试(Mock Object Testing)

■ 单元测试中的现实问题

单元测试本质：

在特定时间段测试一段代码特性，精确定位测试中潜在问题

单元测试要求：

测试代码尽可能简单和清晰

单元测试困难：

代码粘连执行问题？

在所需外部资源、类或服务未实现前，实现对当前代码的测试？



模仿对象测试(Mock Object Testing)

■ 单元测试中的现实问题

在所需外部资源、类或服务未实现前，实现对当前代码的测试？

■ 隔离测试

隔离其它方法或环境对被测程序进行单元测试

■ 模仿对象(Mock Object)

实现隔离测试的一种技术

模仿对象用来代替被测程序所需的对象



模仿对象测试(Mock Object Testing)

■ Mock Object实例

```
public class Purchase {  
    ...  
    public double getTotalPrice(DBAccess dbAccess)  
    {  
        dbAccess.getPriceFromDB();  
    }  
    ...  
}
```

被测类Purchase

```
class PurchaseWithRealDBAccessTest {  
    @Test  
    public void testGetTotalPrice() {  
        DBAccess dbAccess=new DBAccess();  
        assertTrue(180, purchase.getTotalPrice(dbAccess);  
    }  
    ...  
}
```

用真实DBAccess来
测试Purchase

模仿对象测试(Mock Object Testing)

■ Mock Object实例

```
class MockDBAccess extends DBAccess {  
    public double getPriceFromDB() {  
        double price=180;  
        return price;  
    }  
}
```

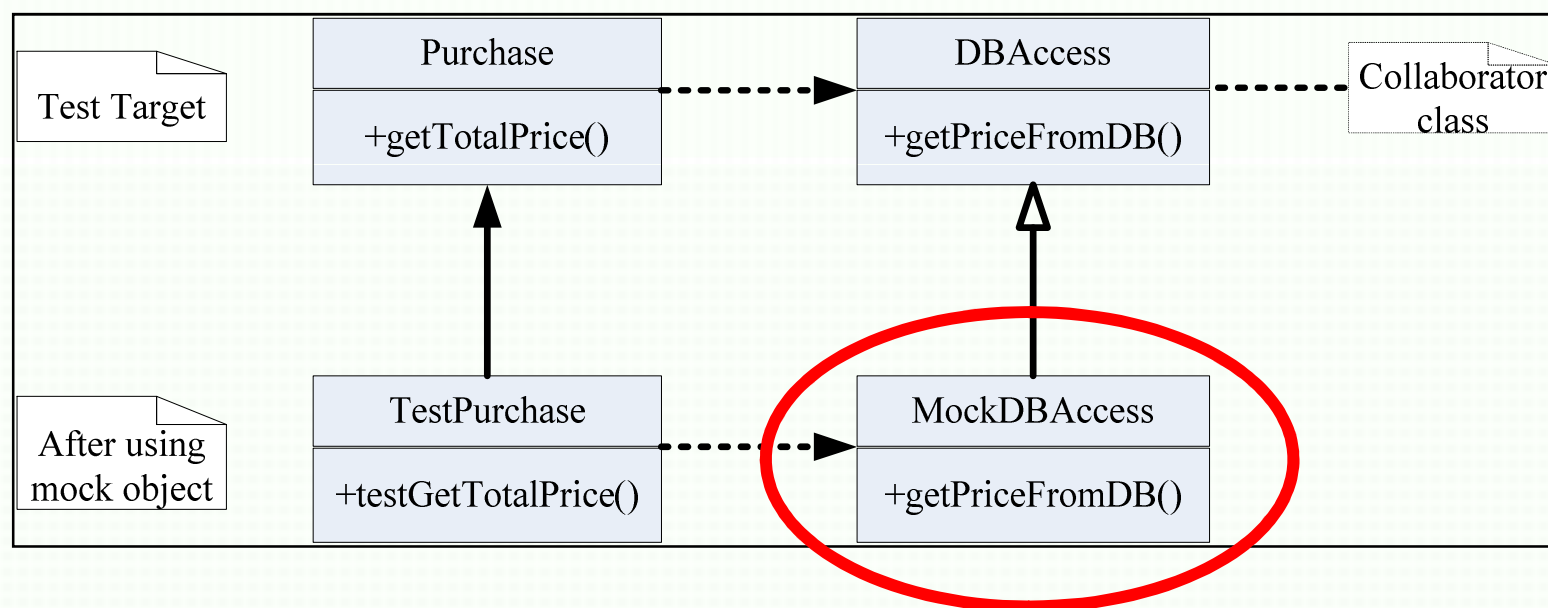
**DBAccess的Mock
Object**

```
class PurchaseWithRealDBAccessTest {  
    @Test  
    public void testGetTotalPrice() {  
        MockDBAccess mockDBAccess=new MockDBAccess();  
        assertTrue(180, purchase.getTotalPrice(mockDBAccess);  
    }  
    ...  
}
```

**用Mock Object来
测试Purchase**

模仿对象测试(Mock Object Testing)

■ Mock Object实例



模仿对象测试(Mock Object Testing)

■ Mock Object特点

1. 模仿对象有与被测对象的“合作者”完全一致的接口
2. 模仿对象在测试中由被测程序调用
3. 模仿对象根据测试意图改变某些状态或返回期望结果，以检查被测程序是否按照期望的逻辑进行工作

Mock Object是其它被测试代码行为的映射或者中介物的实现替代，它应该比真实代码更简单，而不是复制它的所有实现。

Mock Object实现的重点：简单化而非完整化



模仿对象测试(Mock Object Testing)

■ Mock Object测试步骤

Step1. 创建模仿对象的实例

Step2. 设置模仿对象中的状态和期望值

Step3. 将模仿对象作为参数来调用被测代码

Step4. 验证模仿对象中的一致性和被测对象的返回值或状态



模仿对象测试(Mock Object Testing)

■ Mock Object作用

1. 隔离了被测代码之间，被测代码和测试代码之间的关联程度
2. 简化了测试结构，避免被测代码因测试环境而出现污染



模仿对象测试(Mock Object Testing)

■ Mock Object的专用工具

工具:

JMock和EasyMock

作用:

快速创建模仿对象，定义交互过程中的约束条件

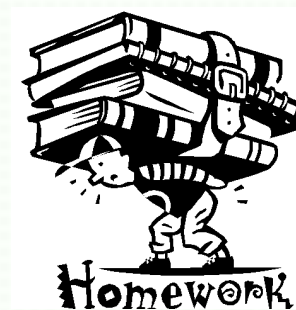
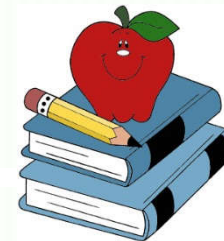
参考文献

1. 郁莲, 软件测试方法与实践, 清华大学出版社, p103-p146, 2008
2. 王东刚, 软件测试与JUnit实践, 人民邮电出版社, 2004
3. JUnit Cookbook, <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>
4. www.junit.org/
5. JUnit 4.x Quick Tutorial, <http://code.google.com/p/t2framework/wiki/JUnitQuickTutorial>
6. 在Eclipse中使用JUnit4进行单元测试,
<http://developer.51cto.com/art/200906/127656.htm>
7. Vincent Massol, JUnit in Action, Manning Publications, 2003



课后习题

结合实验进行



本讲总结

■ JUnit测试工具

JUnit基础

JUnit3.x

JUnit4.x



内容预告

- 系统测试

- 确认测试



衷心感谢各位老师莅临指导！
欢迎各位老师同学批评指正！

Email: pwang@seu.edu.cn

