

## 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



# 如何有效测试软件？

测试用例：输入，执行条件，期望输出

手工输入？

人工观察输出？

人工统计测试结果？

程序可能不断被修改！

修改的程序需要不断被测试！

手工执行上述测试步骤可行吗？

大项目开发中的测试工作量？

自动化测试？

回归测试？

# JUnit之前的单元测试

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

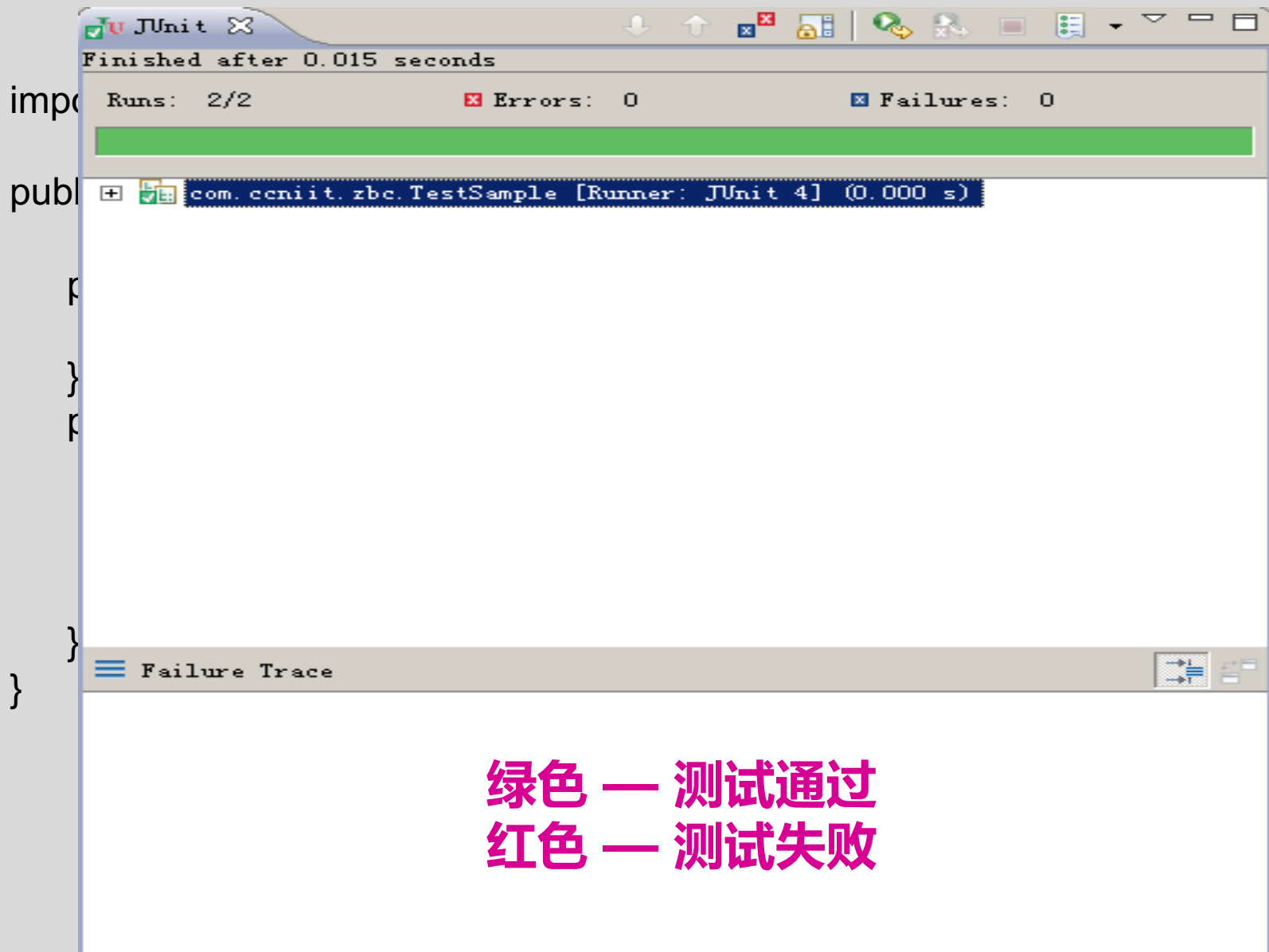
```
public class TestCar {  
    public static void main(String[] args) {  
        Car car = new Car();  
        if (4 == car.getWheels())  
            System.out.println("Ok!");  
        else  
            System.out.println("Error!");  
    }  
}
```

# 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 testgetWheels() {
        // Test Code goes here
        Car car = new Car();
        assertEquals(4, car.getWheels());
    }
}
```

# JUnit — TestSuite



# 本讲内容

**JUnit基础**

**概念、安装、运行**

**JUnit使用**

**用JUnit进行测试**

# 编码内测试的不足

## ■ 编码内测试存在问题

1. 难以确定复杂系统是否正常运转
2. 测试代码难以维护

## ■ 原因分析

- |                  |        |
|------------------|--------|
| 1. 测试缺乏系统规划      | 编码不断变化 |
| 2. 测试代码规范性       | 注释不足   |
| 3. 无法回归测试        |        |
| 4. 无法完成“日创建-日测试” |        |
| .....            |        |

# 新型软件开发方法

## ■ 敏捷软件开发

**目的：**应对快速变化的软件需求

**特点：**

- 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

## ■ 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，即先写测试代码，再进行编码。

# JUnit简介

**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》

# 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



Kent Beck



设计模式开创者之一  
Eclipse Java总设计师

软件开发方法学大师  
极限编程的创始人

# 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

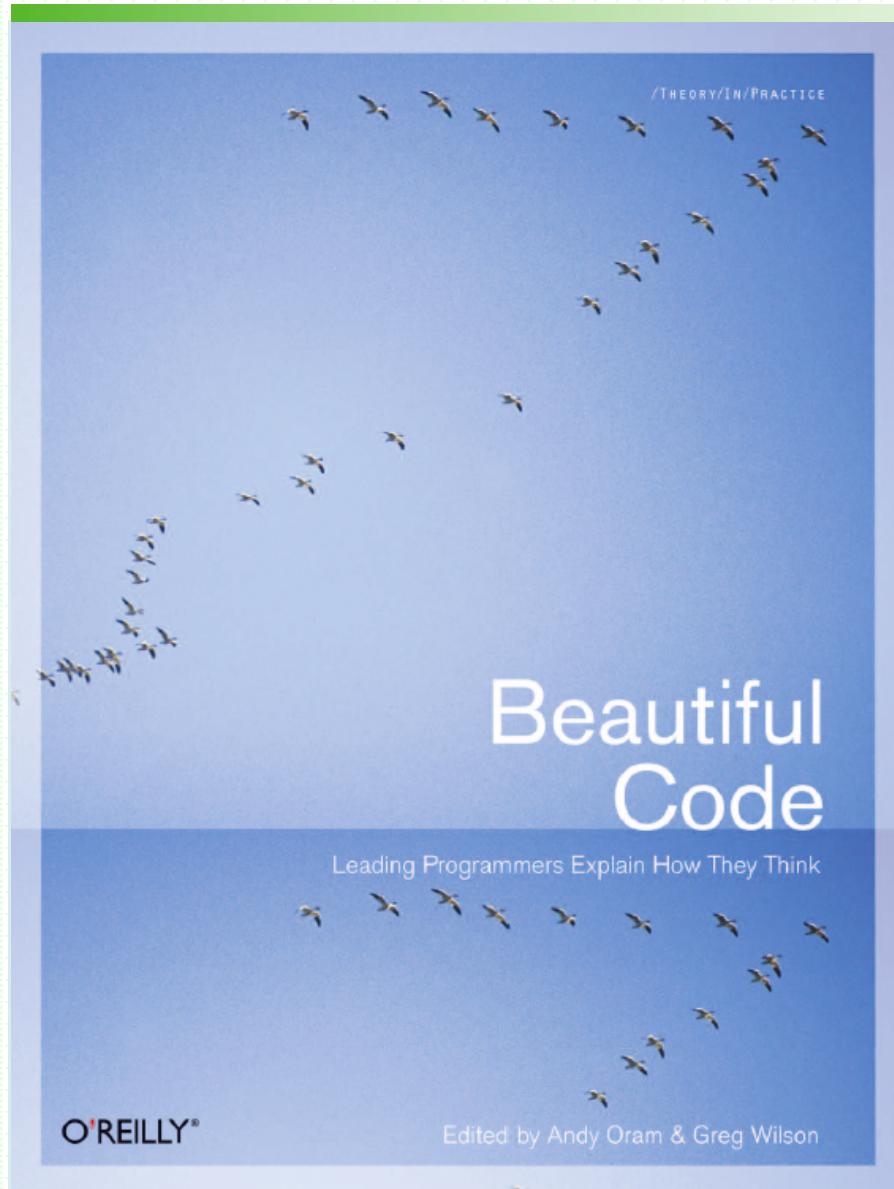
### 目标2：

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



## 7 BEAUTIFUL TESTS *by Alberto Savoia*

That Pesky Binary Search  
Introducing JUnit  
Nailing Binary Search  
Conclusion

# JUnit简介

**JUnit 4.x : 最新JUnit 4.10**

**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.集成测试
- 3.系统测试
- 4.性能测试
- 5.回归测试
- 6.面向对象测试

.....

# JUnit简介

## ■ JUnit安装与运行

获取JUnit:

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

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

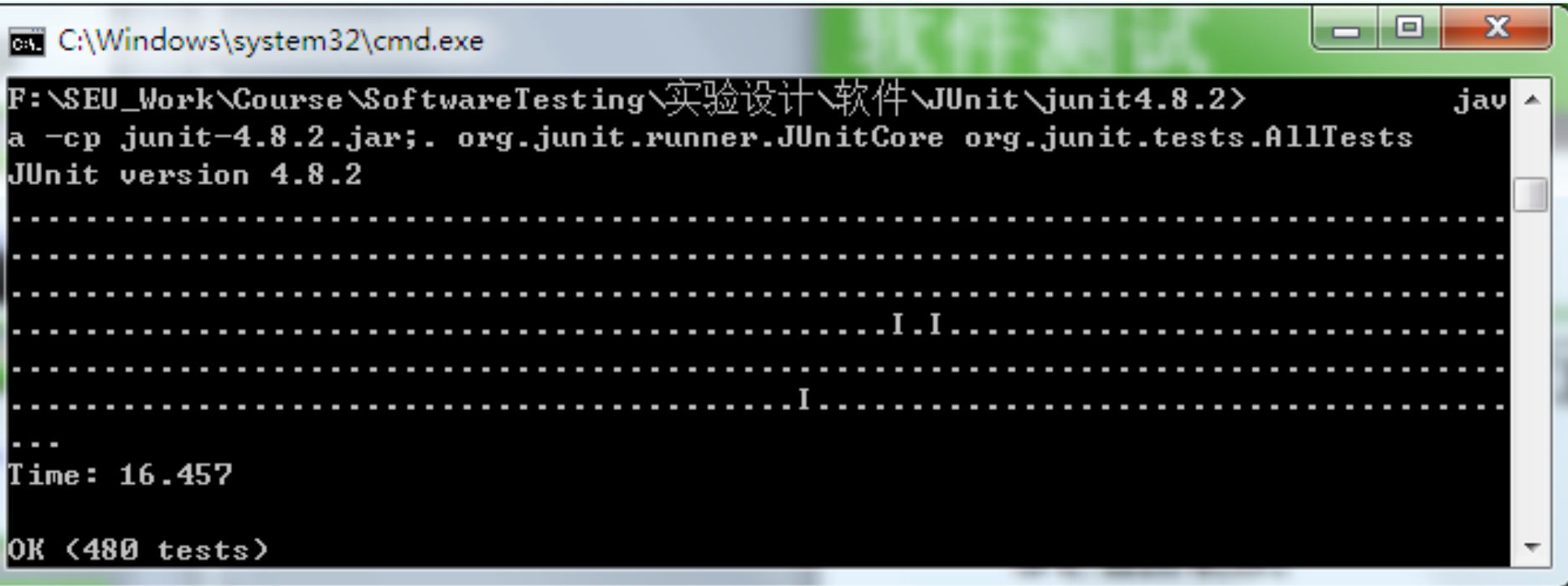
# JUnit简介

## ■ JUnit安装与运行

检验安装JUnit:

JUnit4.8:

`java -cp junit-4.8.2.jar;. org.junit.runner.JUnitCore org.junit.tests.AllTests`



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

F:\SEU_Work\Course\SoftwareTesting\实验设计\软件\JUnit\junit4.8.2> java
a -cp junit-4.8.2.jar;. org.junit.runner.JUnitCore org.junit.tests.AllTests
JUnit version 4.8.2
.....
.....I.I.....
.....I.....
...
Time: 16.457

OK (480 tests)
```

# JUnit简介

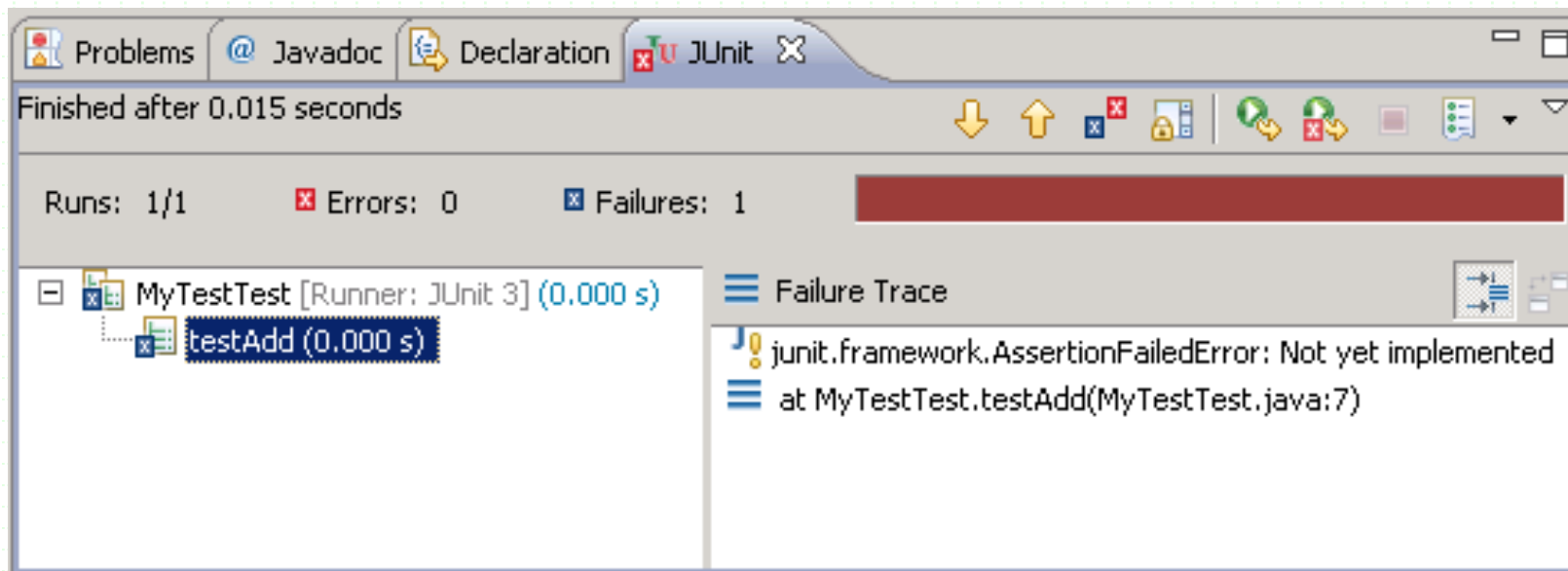
## ■ JUnit安装与运行

### 运行JUnit 4.x测试:

#### 1. 命令行方式:

java **org.junit.runner.JUnitCore** <your\_class>

#### 2. JUnit的Eclipse插件:



# JUnit基础

## 一个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.测试被零整除结果

# JUnit基础

## 观察JUnit 4.x测试程序特点：

1. 利用Java1.5的Annotation特性（元数据）简化测试用例编写
2. 测试类不继承自TestCase
3. 测试方法不必以test开头，只要以@Test描述
4. 测试方法命名以public修饰，返回值void
5. assertTrue(boolean)帮助验证结果

# JUnit基础

## ■ JUnit 4.x的主要知识点

- 元数据(MetaData)
- 断言
- 测试套件(TestSuite)
- 测试固件

# JUnit基础

## ■ JUnit 4.x的元数据(MetaData)

### 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



# JUnit基础

## ■ JUnit 4.x的元数据(MetaData)

5. **@ignore**

测试方法会被忽略

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

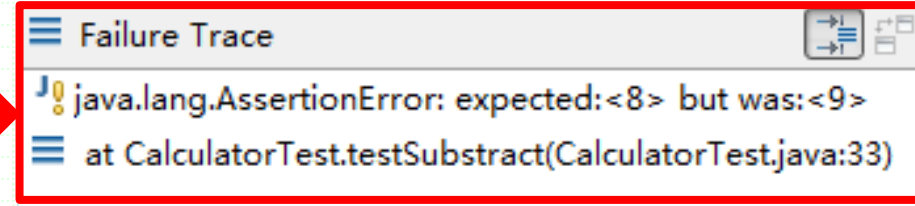
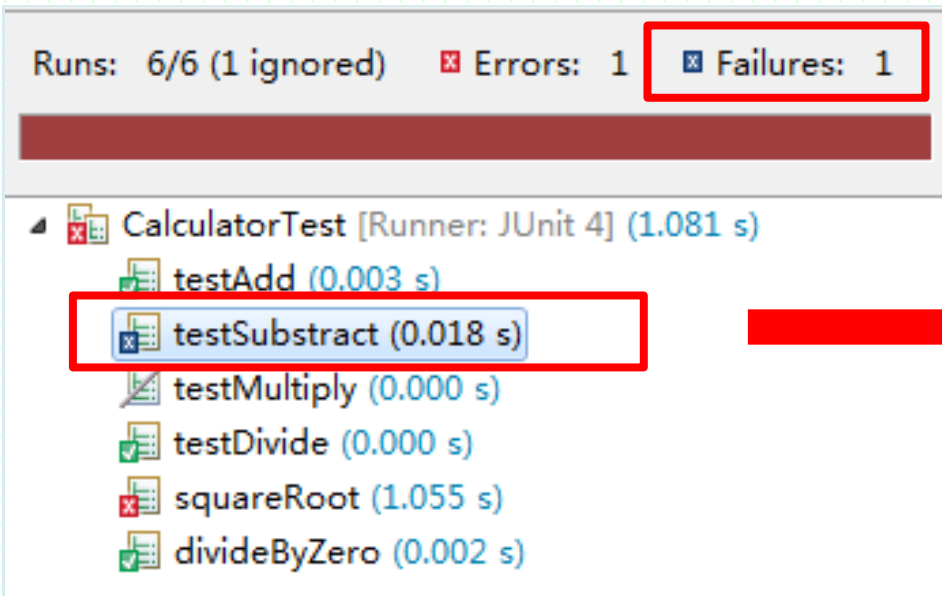
1. 基础断言: `assertTrue()`      `assertFalse()`
2. 数值断言: `assertEquals()`
3. 字符断言: `assertEquals()`
4. 布尔断言: `assertEquals()`
5. 比特断言: `assertEquals()`
6. 对象断言: `assertEquals()`      `assertNotNull`  
                  `assertSame()`  
                  `assertNotSame()`
7. 其它断言:  
                  `assertArrayEquals()`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

# JUnit基础

## ■失败(Failure)与错误(Error)

**Failure:**

断言失败-->测试用例没通过--> 被测编码有问题

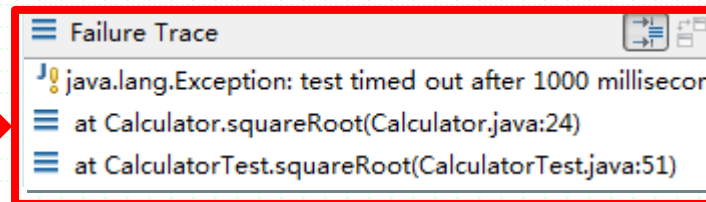
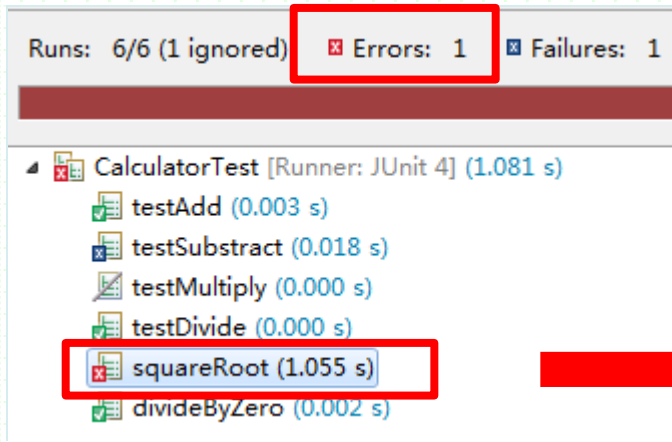


# JUnit基础

## ■失败(Failure)与错误(Error)

**Error:**

不曾预料到的失败条件  
发生未被预测到的异常  
测试代码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: 控制测试用例执行: 选择、次序、次数...**

**不定义TestSuite时, JUnit将自动生成默认TestSuite()**

```
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;  
}
```

**只用于JUnit3.x**

# JUnit基础

## ■ 测试套件(TestSuite)

**JUnit4.x没有TestSuite**

**JUnit4.x如何组织测试用例?**

1. **@Ignore**
2. **打包测试**

# JUnit基础

## ■ 测试固件(Fixture)

**目的：重用**

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(Fixture)

```
public class VectorTest {  
    protected Vector fEmpty, fFull;  
  
    @Before  
    public void setUp() throws Exception {  
        fEmpty = new Vector();  
        fFull = new Vector();  
        fFull.addElement(new Integer(1));  
        fFull.addElement(new Integer(2));  
    }  
  
    @After  
    public 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测试

## ■ 测试环境

**JUnit版本：** JUnit 4.8

**Java 版本：** Java 1.6

**IDE 环境：** Eclipse 3.6

# JUnit测试

## ■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;  
    }  
}
```

# JUnit测试

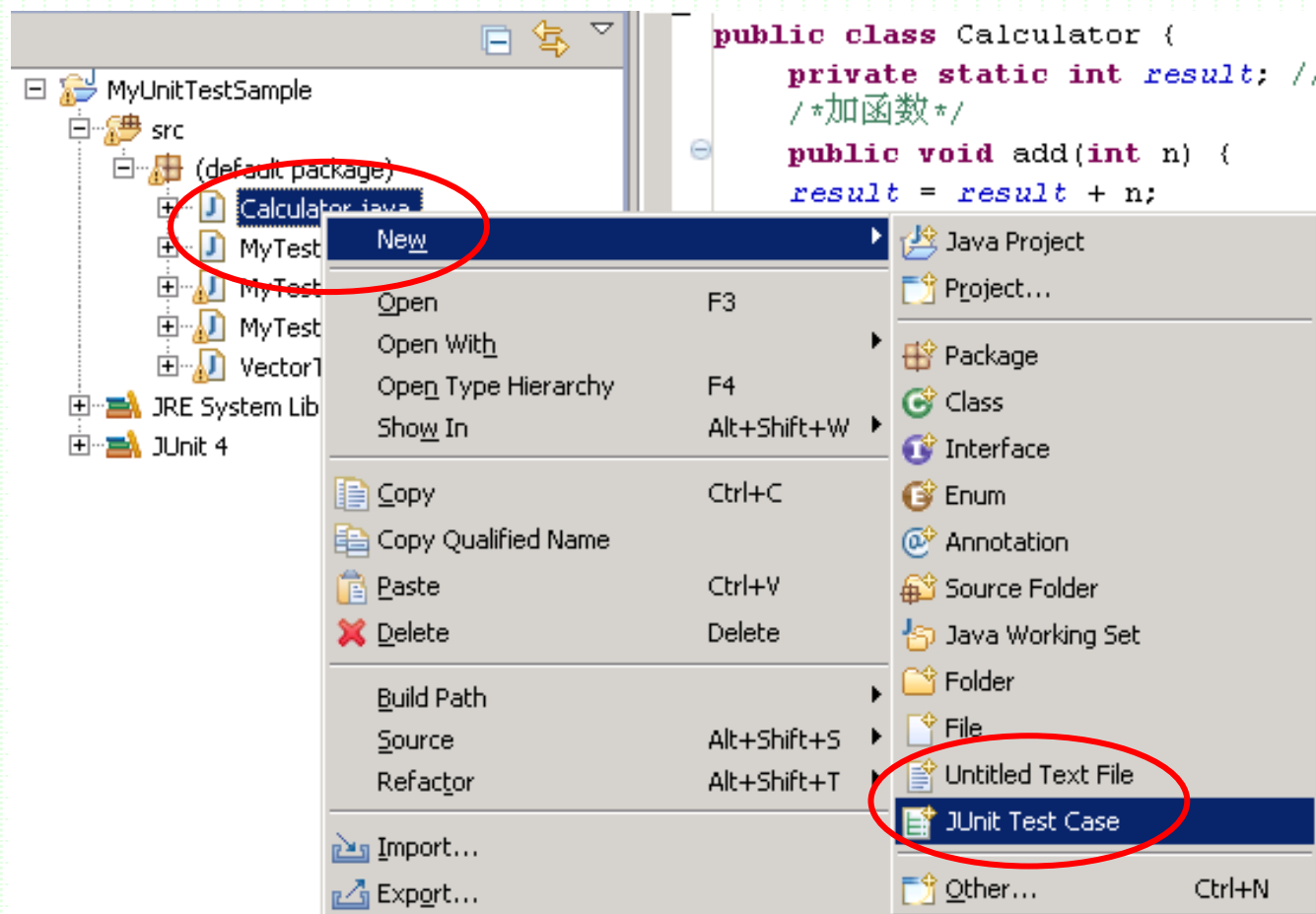
## ■被测程序：Calculator类

```
/*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;  
}  
}
```

# JUnit测试

## ■ 自动生成测试框架

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



# JUnit测试

## ■ 自动生成测试框架

New JUnit Test Case

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

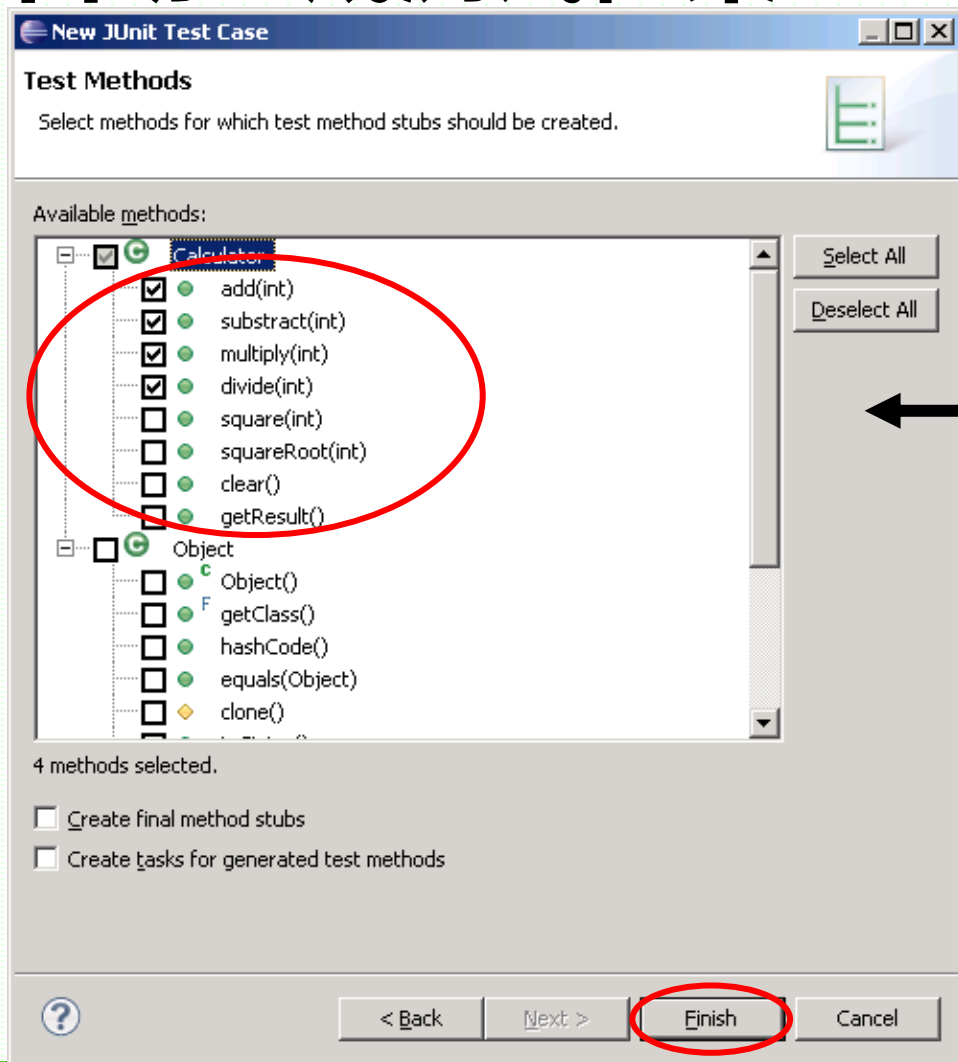
Class under test: Calculator Browse...

测试代码类名

选择固件方法

# JUnit测试

## ■ 自动生成测试框架



选择待测方法

# JUnit测试

## ■ 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");  
    }  
    .....  
}
```



# JUnit测试

## ■ 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");
}
}
```

# JUnit测试

## ■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);  
    }  
}
```

# JUnit测试

## ■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());  
}
```

# JUnit测试

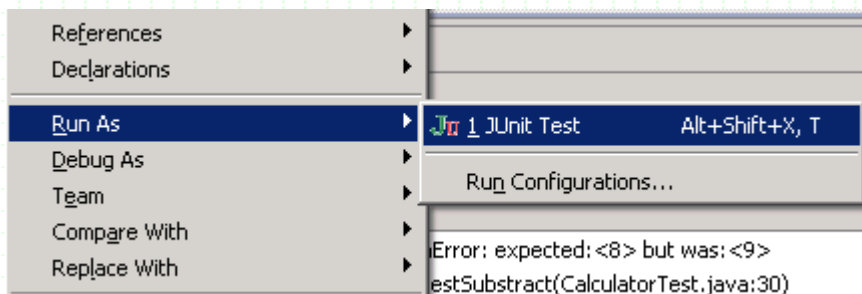
## ■完善CalculatorTest类

```
/*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());  
}
```

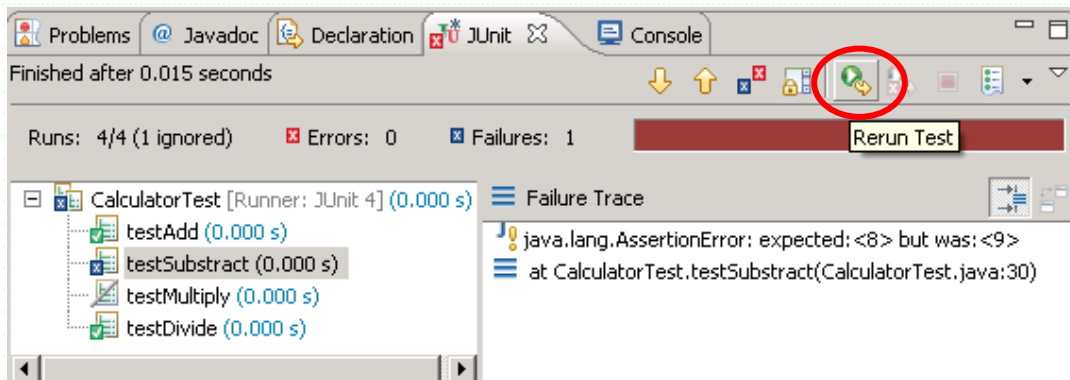
# JUnit测试

## ■ 执行CalculatorTest进行测试

方法1：右键菜单→Run As→JUnit Test

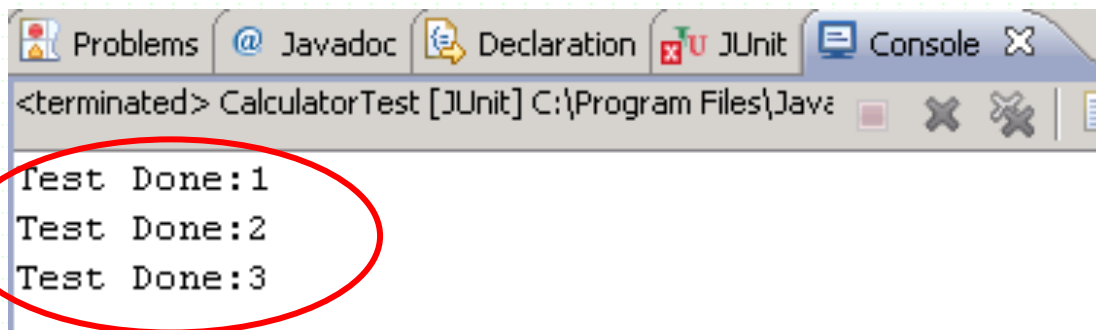
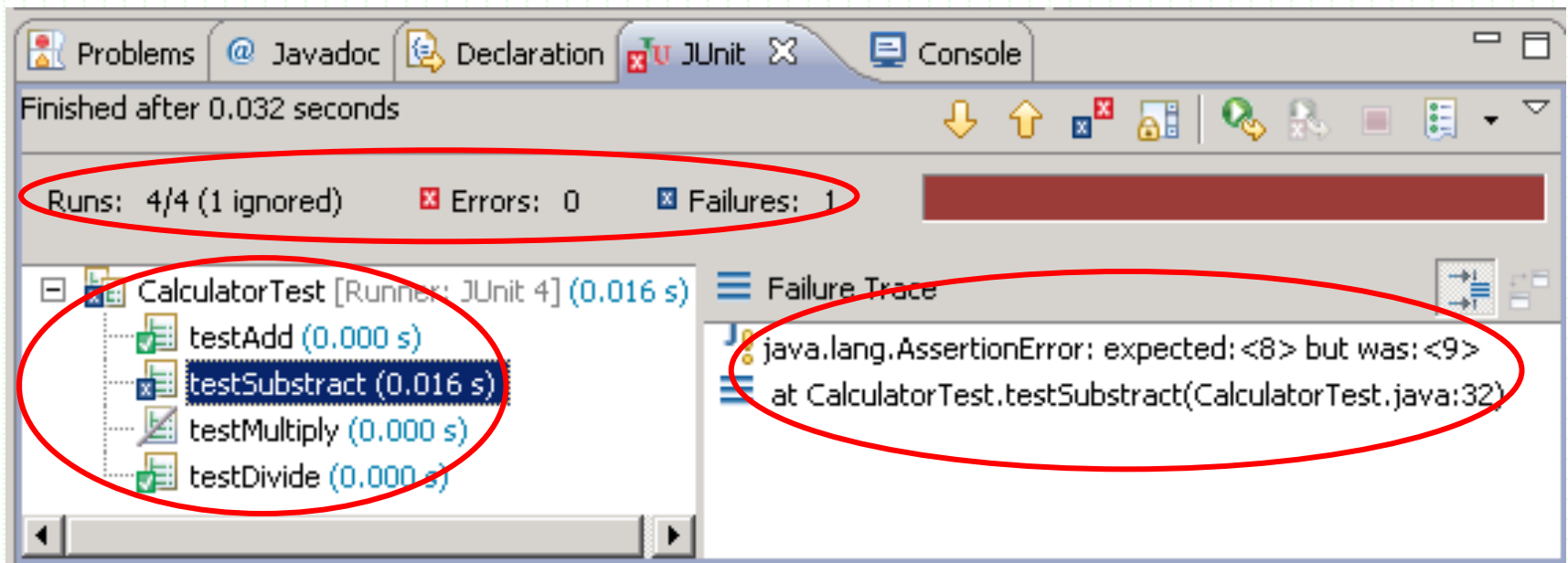


方法2：JUnit插件



# JUnit测试

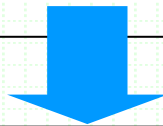
## ■ CalculatorTest测试结果



# JUnit测试

## ■增加限时测试

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

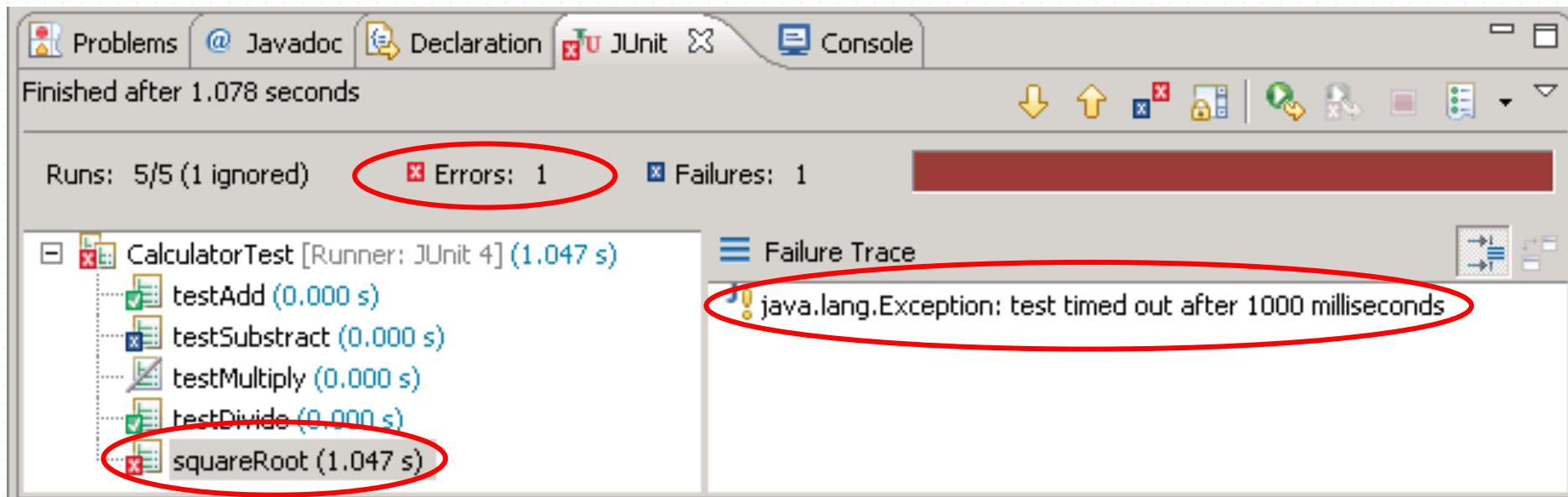


```
@Test(timeout = 1000)  
public void squareRoot() {  
    calculator.squareRoot(4);  
    assertEquals(2, calculator.getResult());  
}
```

**用途：死循环预防；跳出复杂代码...**

# JUnit测试

## ■ 增加限时测试





# JUnit测试

## ■增加测试异常

测试方法：**@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

# JUnit测试

## ■ 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());
}
```

# JUnit测试

## ■ 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}});  
}
```

```
.....
```

# JUnit测试

## ■ 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());  
}  
}
```

# JUnit测试

## ■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

# JUnit测试

## ■打包测试

### 将多个测试类进行打包测试

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

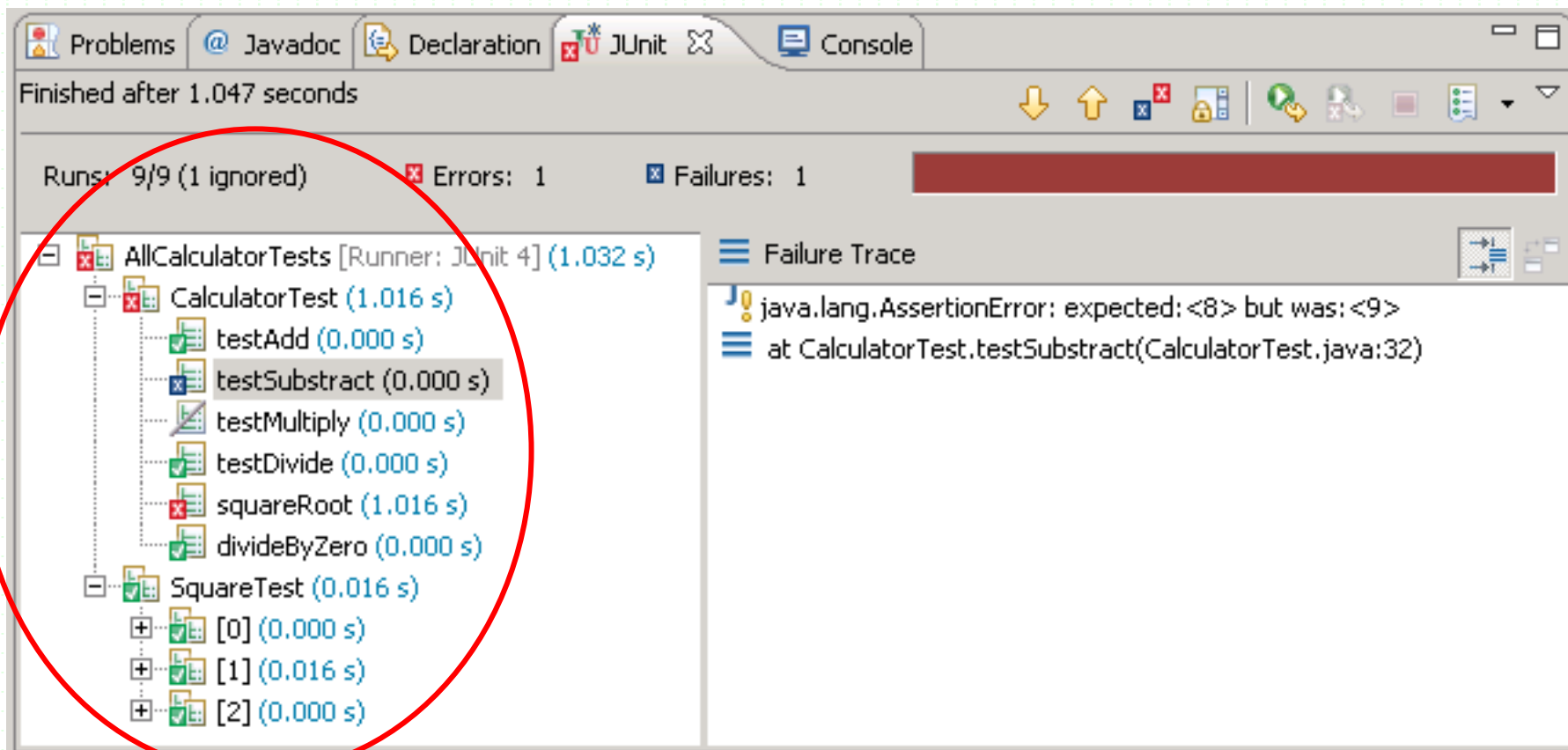
```
@RunWith(Suite.class)  
@Suite.SuiteClasses({  
    CalculatorTest.class,  
    SquareTest.class  
})
```

```
public class AllCalculatorTests {  
}
```

# JUnit测试

## ■打包测试

将多个测试类进行打包测试



# JUnit测试

## ■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



# JUnit测试

## ■打包测试

### 打包测试+@Ignore:

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

# JUnit3.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

# JUnit3.x VS JUnit 4.x

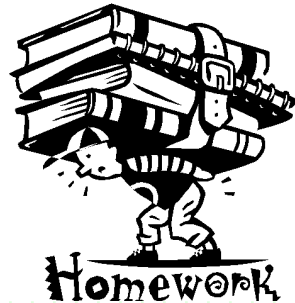
	JUnit 3.x	JUnit 4.x
7.忽略一个测试	注释 , 改名	@Ignore标注
8.运行测试UI	text, AWT, SWing	text
9.测试集组织	suite()方法	@RunWith 和 @Suite标注一个空类
10.运行器	——	@RunWith
11.高级测试		预设环境 @BeforeClass/@AfterClass 限时测试 参数化测试

# 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/](http://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](mailto:pwang@seu.edu.cn)

