

# Beyond Technology



## 单元测试

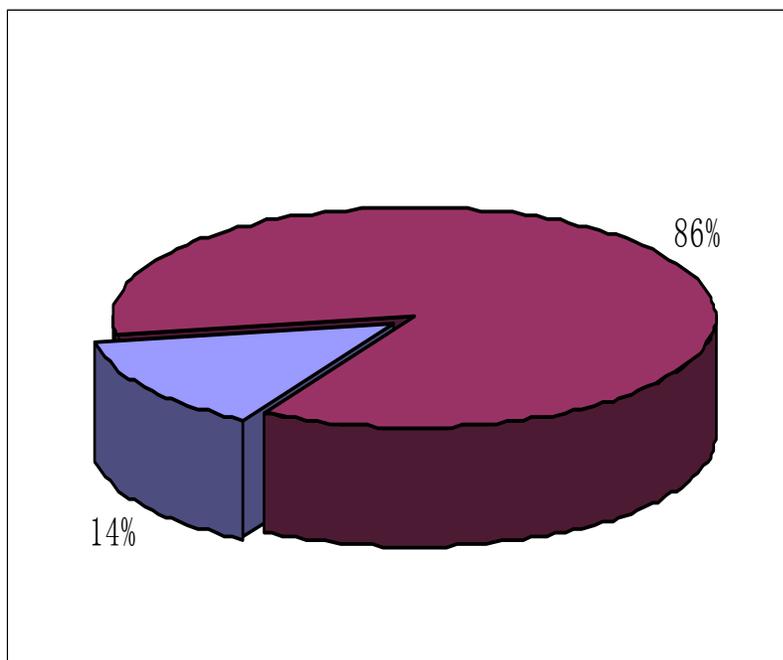
# 目的与目标

- 理解单元测试基本概念
- 学会编写单元测试代码
- 掌握执行单元测试的方法

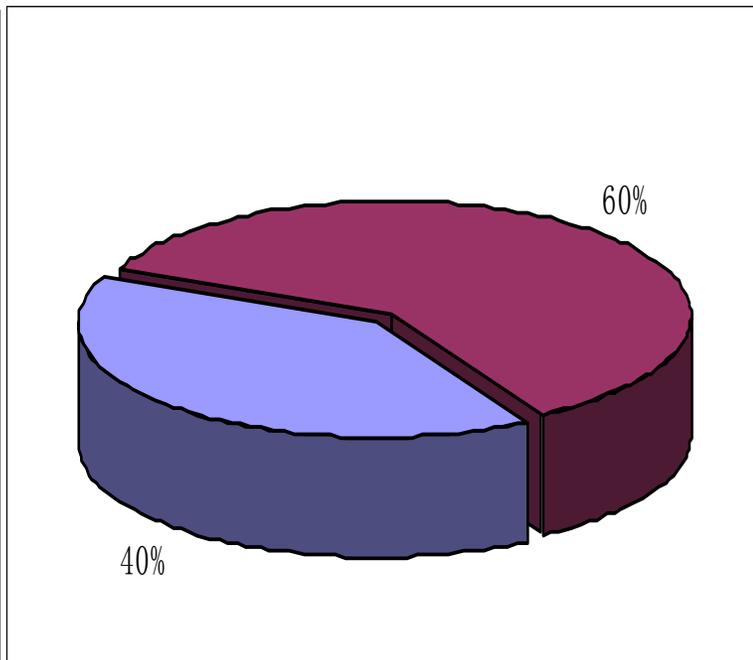
# 引言

- 错误分布

业界编码阶段错误数



国内某企业编码阶段错误数



# 单元测试基本概念

- 单元测试是什么？
- 单元测试什么时候做？
- 单元测试由谁来做？
- 单元测试测什么？
- 单元测试不测什么？

# 单元测试是什么？

- 单元测试
  - 是在软件开发过程中要进行的最低级别的测试活动，在单元测试活动中，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。
  - 是从程序员的角度编写的，它确保类的某个特定方法能成功执行一系列特定的任务。每个测试都确保只要给定输入，方法将输出预期的结果

# 单元测试什么时候做？

- Code a little, test a little
- Code all, test all

# 单元测试由谁来做？

- 单元测试是程序员的事
- 单元测试是编码的一部分
- 单元测试是程序员必备的一项基本素质

# 单元测试测什么？

- 过程语言：函数、过程或完成某一功能的程序块
- OO语言：类的成员函数

# 单元测试测什么？

- 核心方法及可能引入错误的地方
- 特定边界条件
- 复杂算法
- 复杂的业务逻辑
- 需求变动频繁之处

# 单元测试不测什么？

- 单元测试不测构造函数
- 单元测试不测setter()、getter()方法，如ActionForm的大部分方法
- 单元测试不测框架，如Struts框架

## 早期单元测试示例

- 测试目标: Factorial类
- `public class Factorial {`
- `public int eval(int num) {`
- `return num * eval(num - 1); // ???`
- `}`
- `public static void main(String[] args) {`
- `Factorial app=new Factorial();`
- `int x=app.eval(3);`
- `System.out.println(x==6);`
- `}`
- `}`

# 早期单元测试方式的不足

- **代码膨胀**

由于加入了测试，所以产品代码比所需要的要大。但我们不想交付测试代码，而只想交付产品。

- **测试不可靠**

既然 `main()` 是代码的一部分，`main()` 就对其他开发者通过类接口无法访问的私有成员和方法享有访问权。出于这个原因，这种测试方法很容易出错。

- **缺乏对自动测试的支持**

要进行自动测试，必须能够方便的收集测试结果以及对测试用例进行筛选，利用`main()`方法进行测试要做到这两点就比较麻烦。

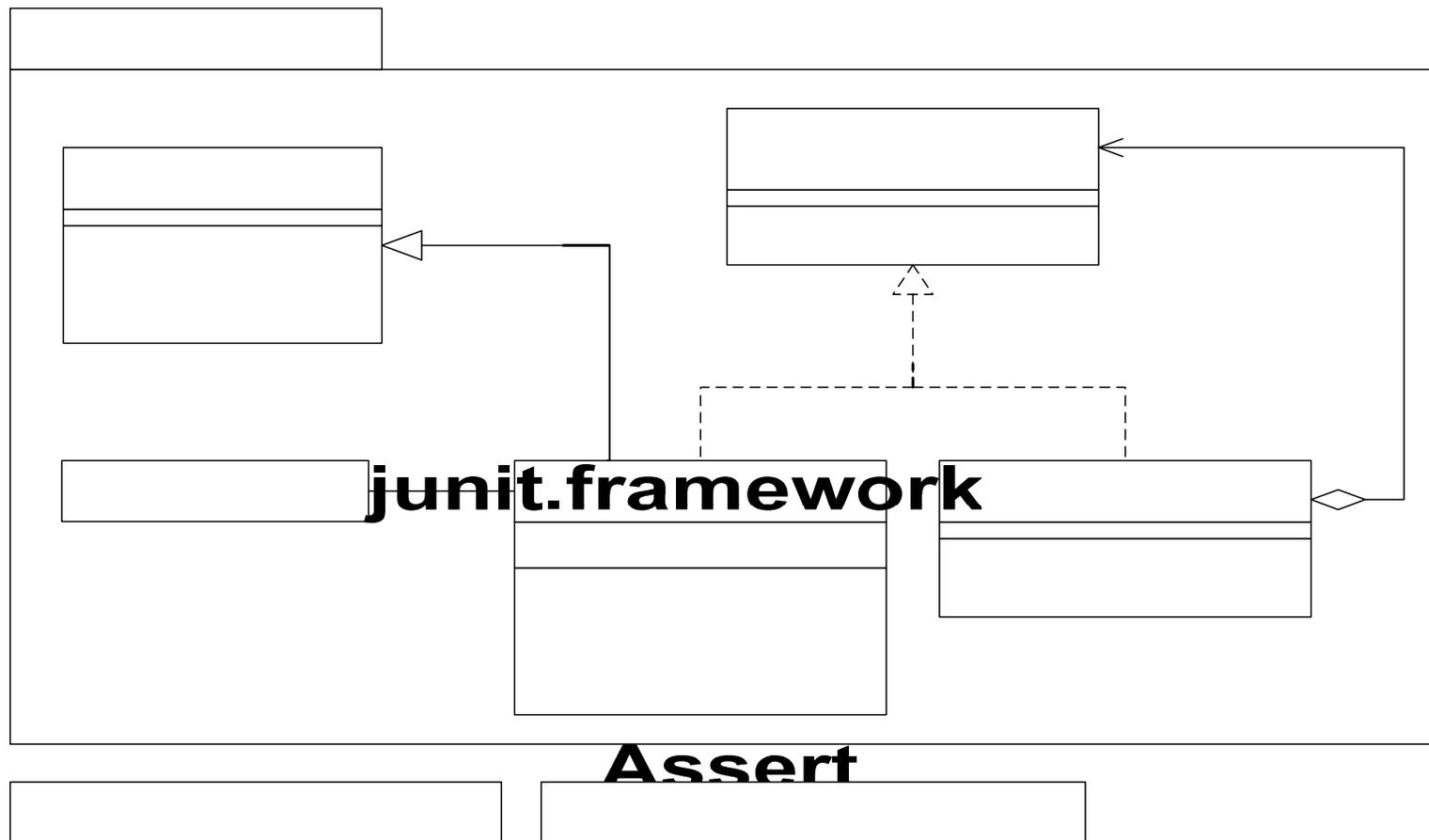
# JUnit简介

- 帮助开发人员测试Java代码的测试框架
- Open Source(<http://junit.sourceforge.net>)
- 由Erich Gamma 和 Kent Beck(极限编程创始人)设计并实现

# JUnit定位

	<b>JUnit is...</b>	<b>JUnit is not...</b>
作用范围	单元测试	集成测试
系统结构	框架,架构	完整系统
使用等级	工具	方法论
测试用例	手动产生	自动产生
测试实现代码	部分手写	全自动

# JUnit架构



```
assertTrue()  
assertEquals()
```

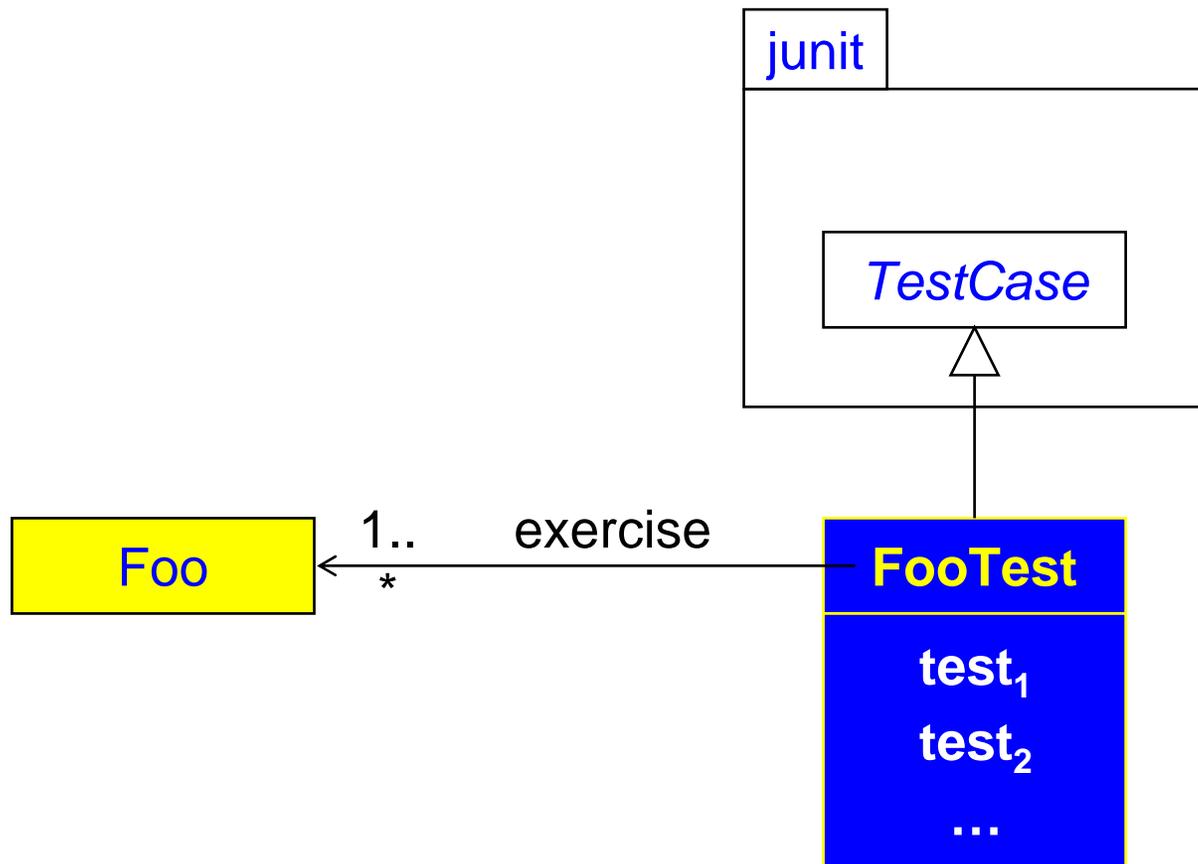
# 学会编写单元测试代码

- JUnit基础篇
- JUnit进阶篇

# JUnit 基础篇

- 自动产生测试框架
- 最简单的完整测试示例

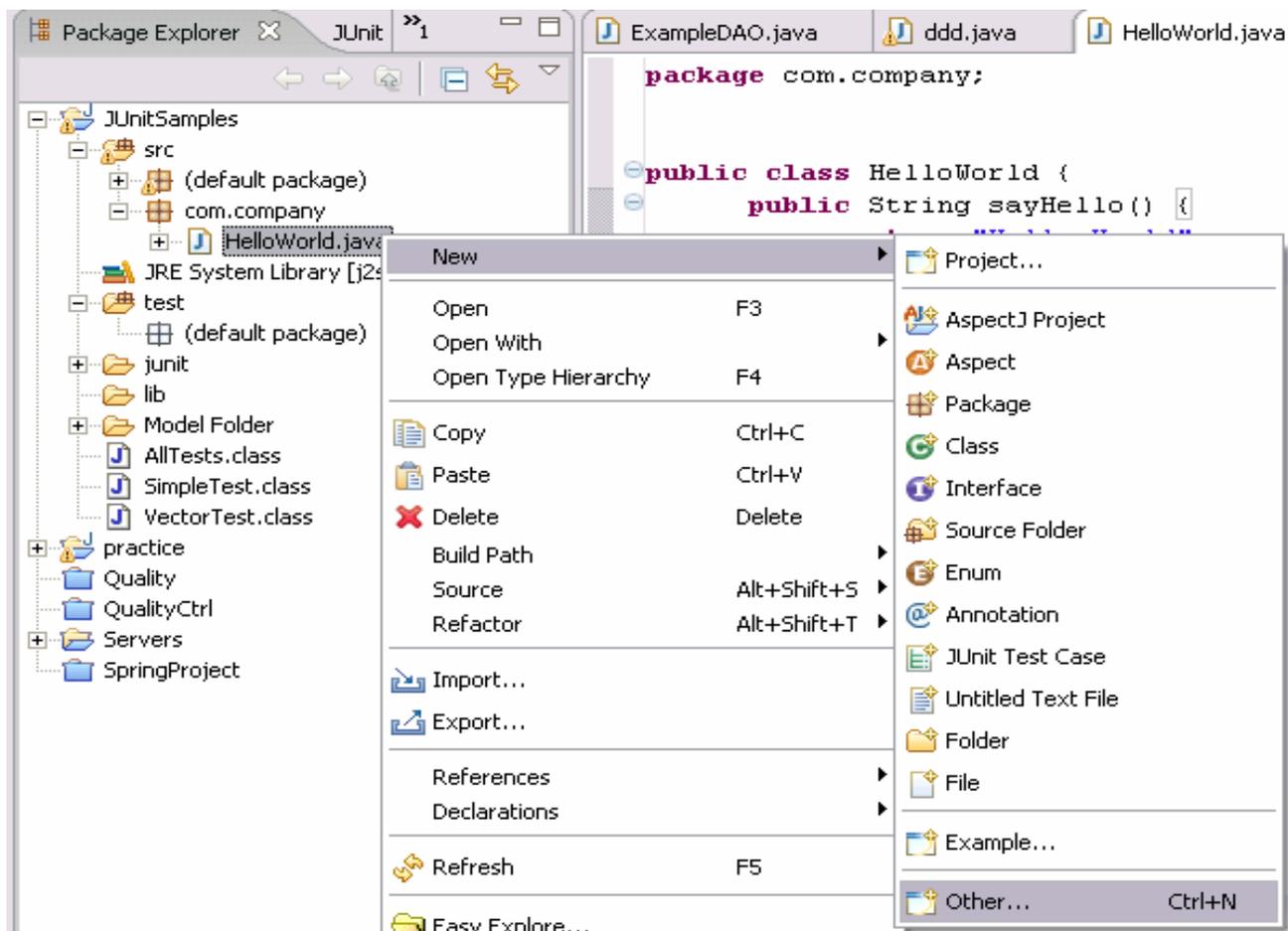
# 示例1



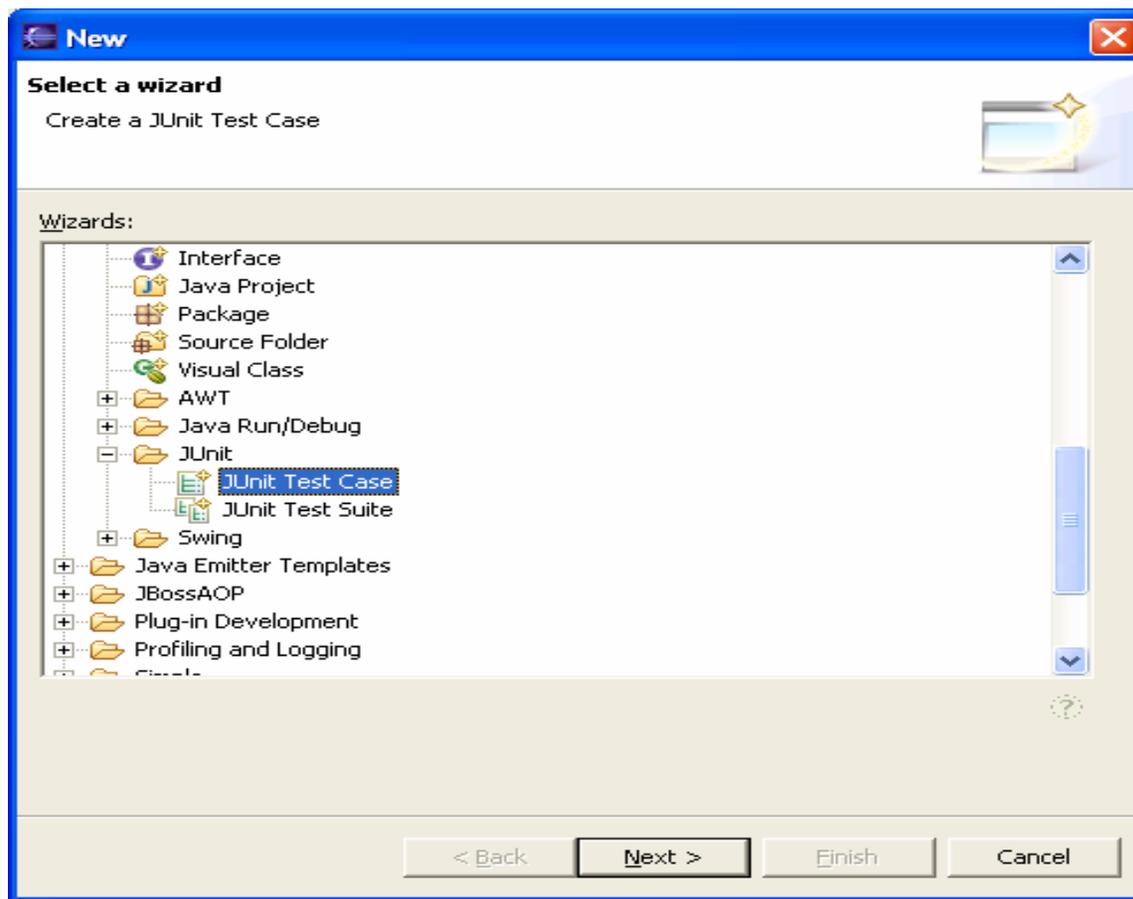
# 示例1 待测代码

```
package com.company;  
public class HelloWorld  
{  
    public String sayHello()  
    {  
        return "Hello World";  
    }  
}
```

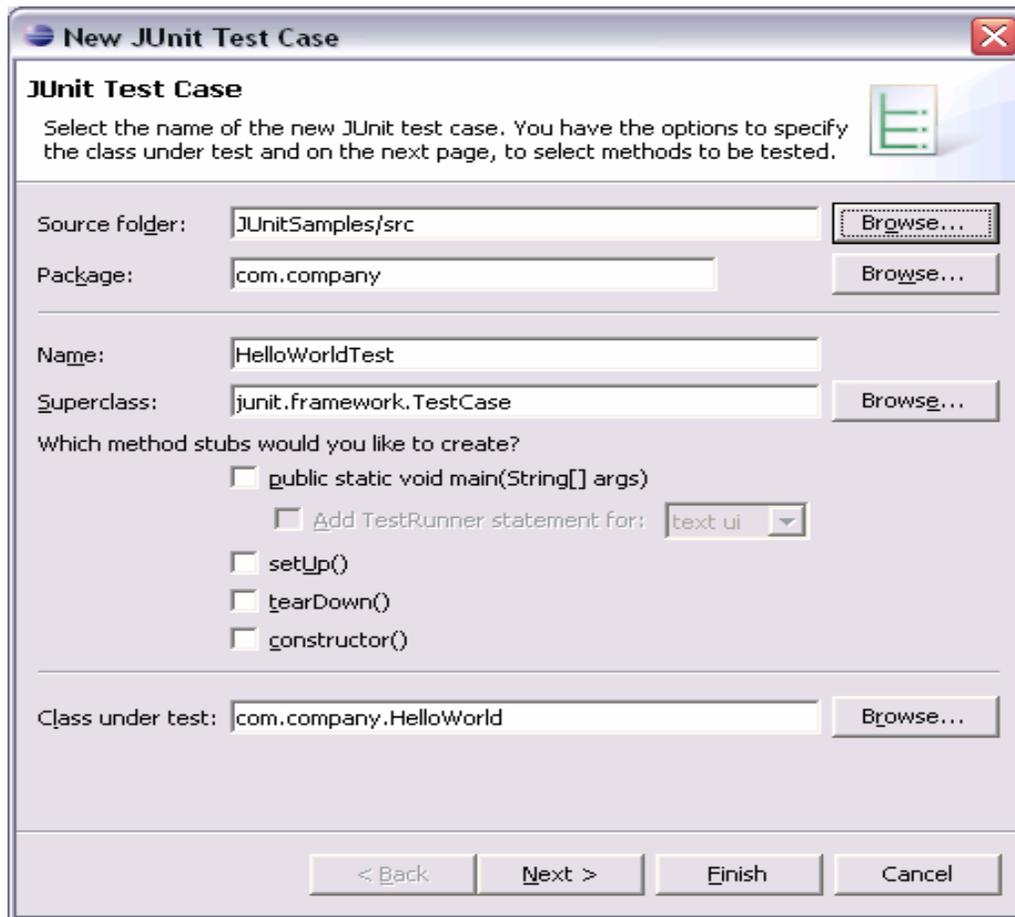
# 产生测试用例



# 产生测试用例



# 产生测试用例



**New JUnit Test Case**

**JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

Source folder: JUnitSamples/src

Package: com.company

Name: HelloWorldTest

Superclass: junit.framework.TestCase

Which method stubs would you like to create?

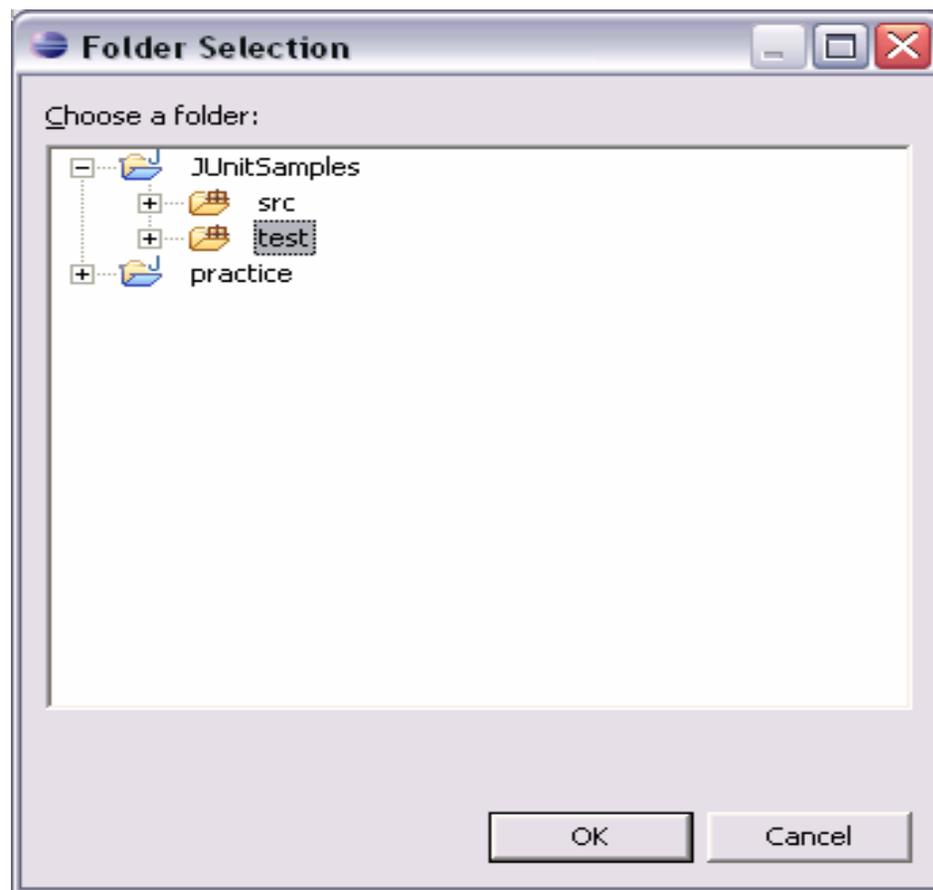
public static void main(String[] args)  
 Add TestRunner statement for: text ui

setUp()  
 tearDown()  
 constructor()

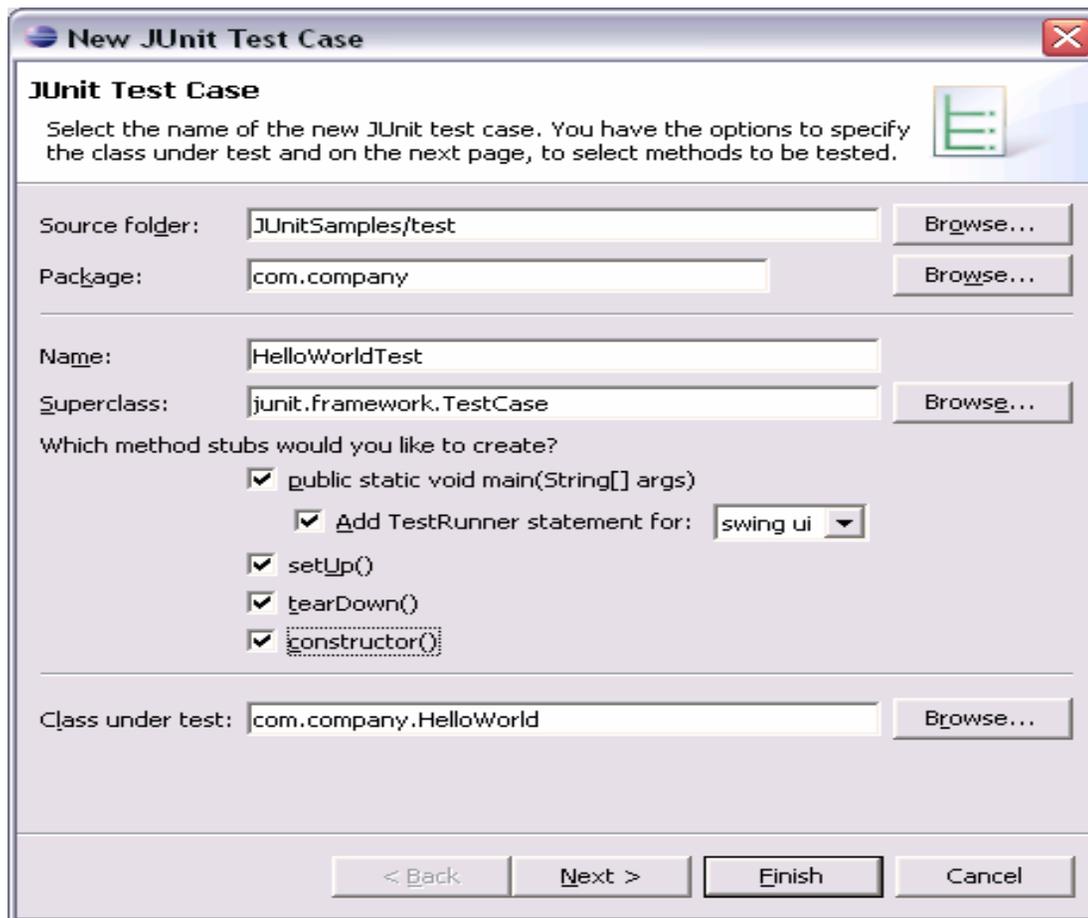
Class under test: com.company.HelloWorld

< Back   Next >   Finish   Cancel

# 产生测试用例



# 产生测试用例



**New JUnit Test Case**

**JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

Source folder:

Package:

Name:

Superclass:

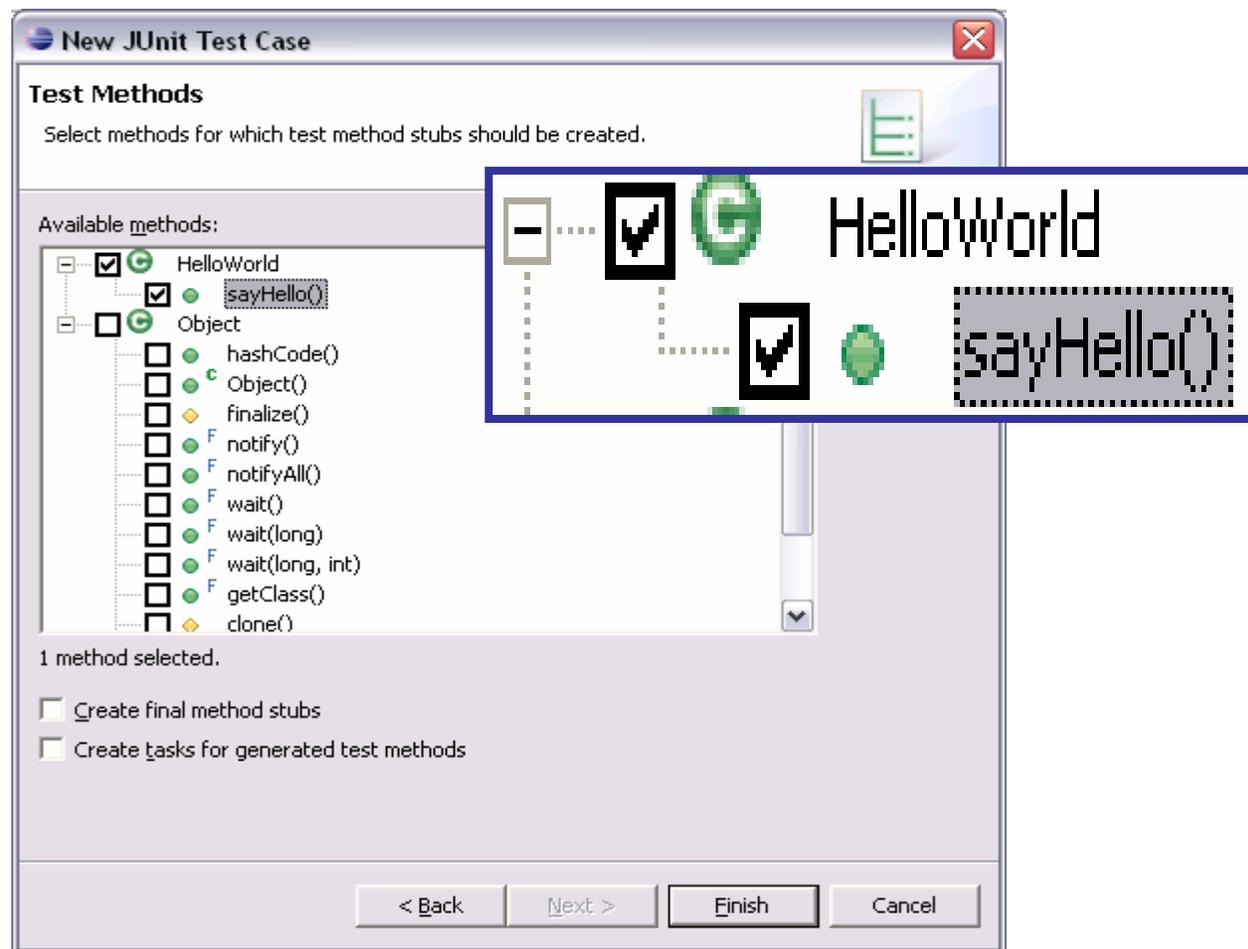
Which method stubs would you like to create?

- public static void main(String[] args)
  - Add TestRunner statement for:
- setUp()
- tearDown()
- constructor()

Class under test:

< Back   Next >   Finish   Cancel

# 产生测试用例



# 产生测试用例

```
package com.company;

import junit.framework.TestCase;

public class HelloWorldTest extends TestCase {

    public static void main(String[] args) {
        junit.swingui.TestRunner.run(HelloWorldTest.class);
    }

    public HelloWorldTest(String name) {
        super(name);
    }

    protected void setUp() throws Exception {
        super.setUp();
    }

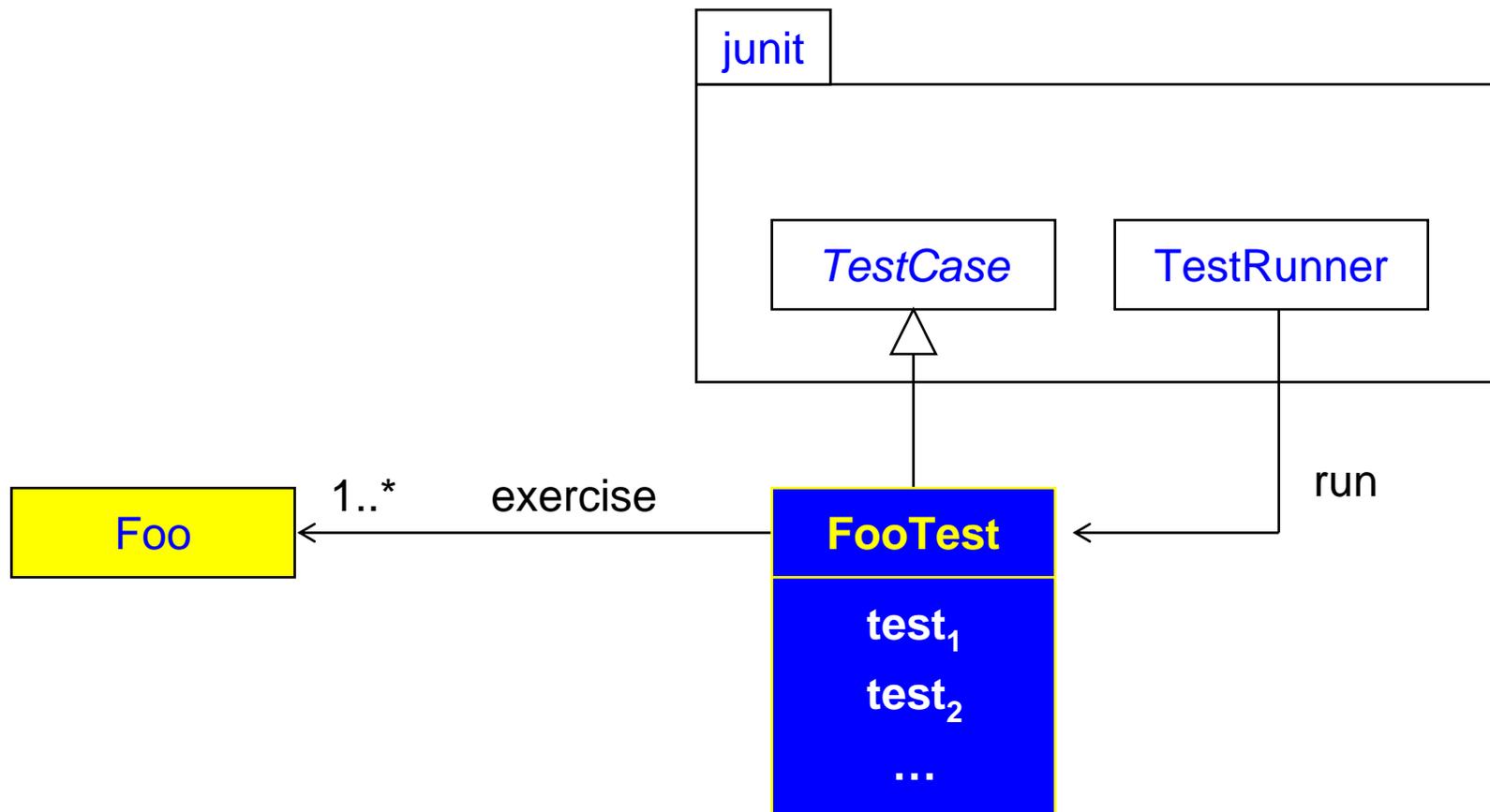
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /*
     * Test method for 'com.company.HelloWorld.sayHello()'
     */
    public void testSayHello() {
    }
}
```

# Lesson Learned

- Code a little, test a little

# 示例2



## 示例2 测试代码

```
package com.company;
import junit.framework.*;
public class HelloWorldTest extends TestCase {
    public HelloWorldTest (String name) { super(name); }
    public void testSayHello()
    {
        String expectedReturn = "Hello World";
        String actualReturn = helloWorld.sayHello();
        assertEquals("return value", expectedReturn,
            actualReturn );
    }
}
```

命名规则:xxxTest  
(类名称后加"Test")

继承:TestCase

命名规则:testXXX  
(测试方法前加"test")

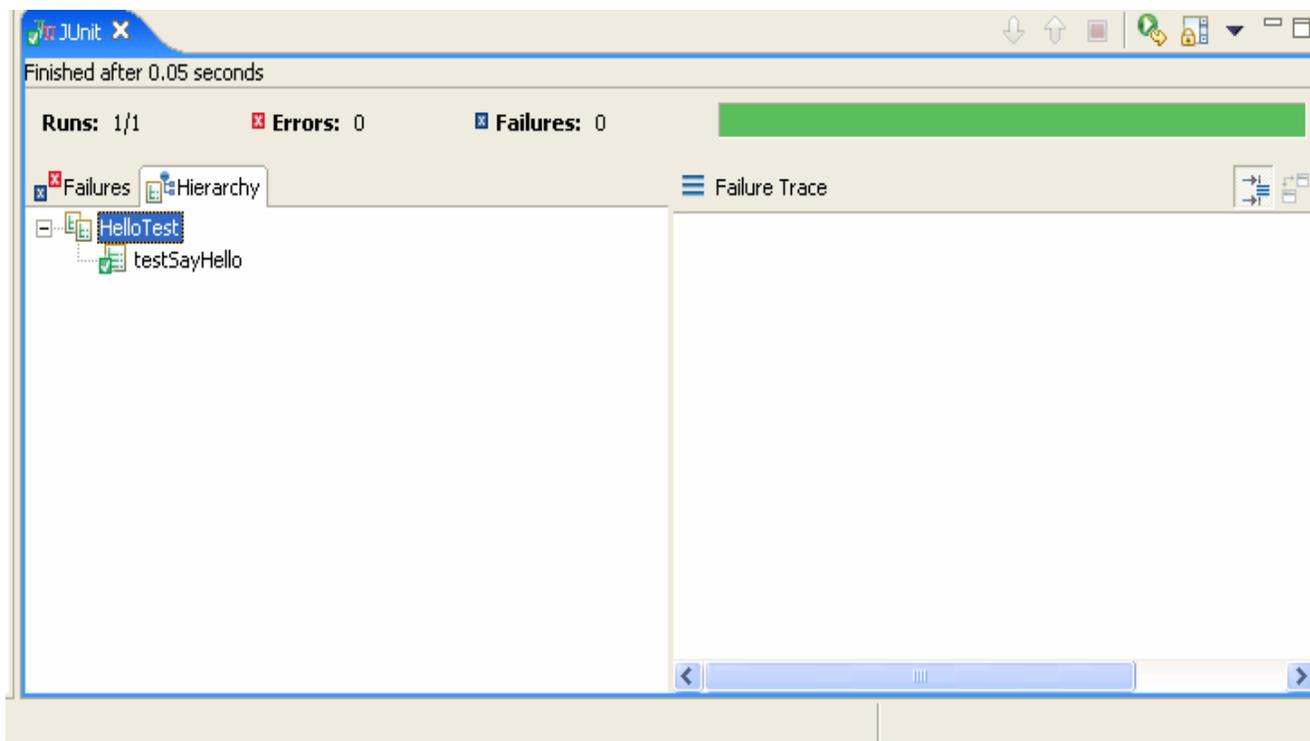
断言(判断类实例的  
特性是否和你期待  
的一致)

## 示例2 测试代码 [续]

```
public static void main(String[] args) {  
  
    junit.swingui.TestRunner.run(HelloWorldTest.cl  
ass);  
}  
}
```

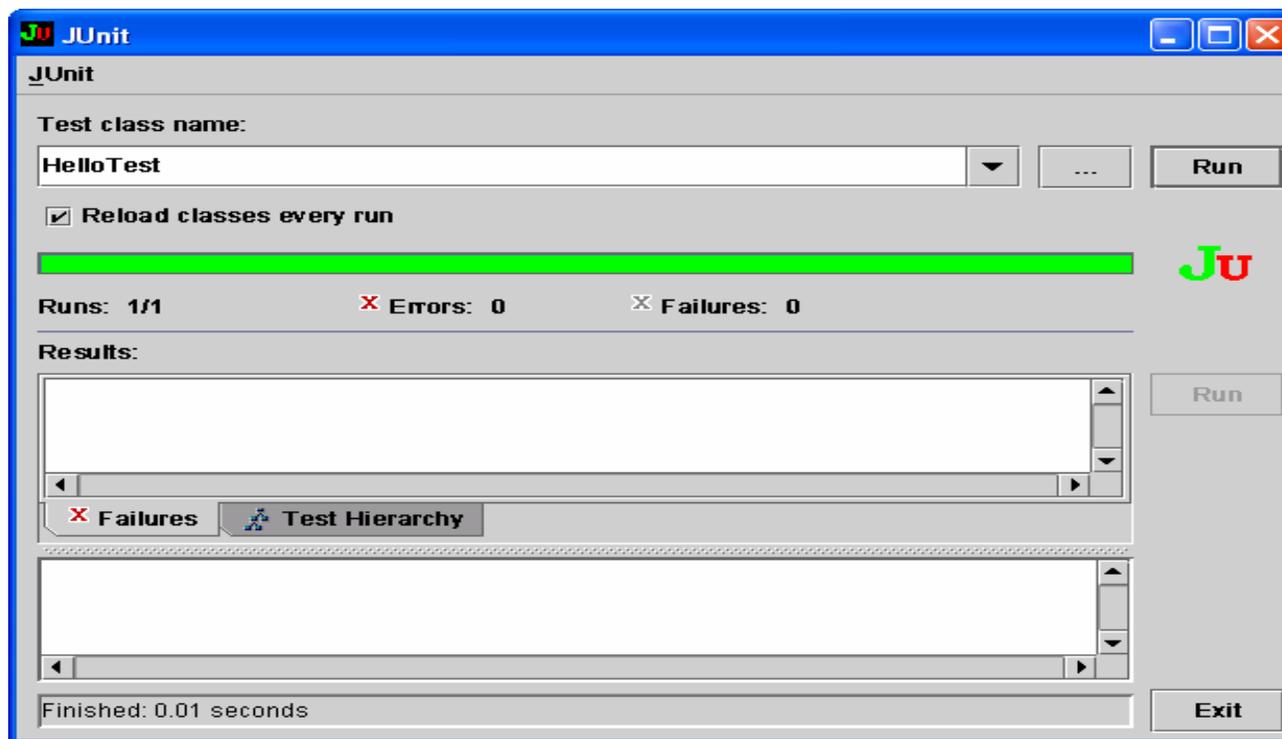
# 选择一个 Test Runner

- eclipse Test Runner



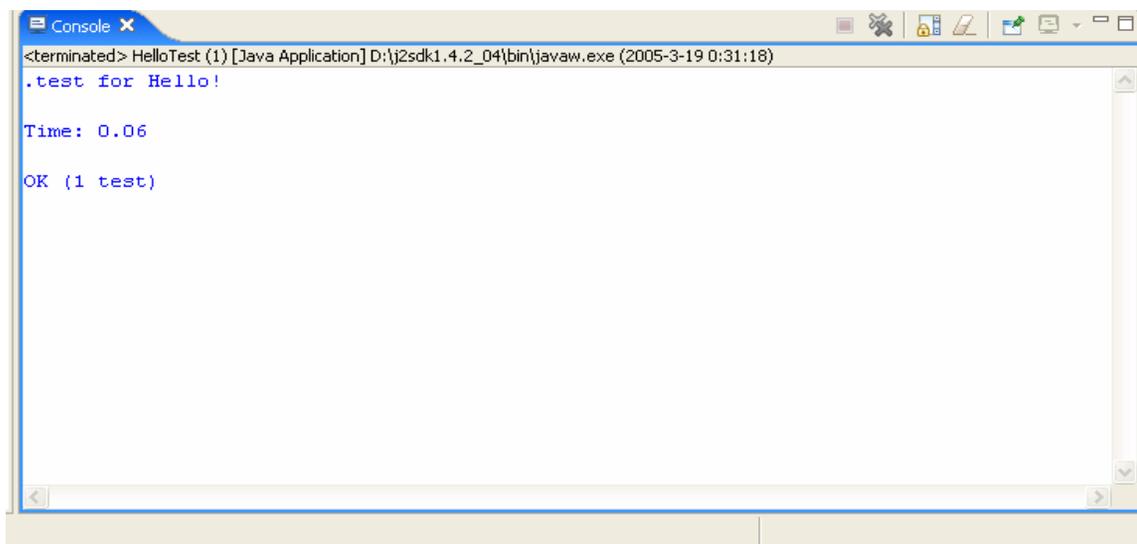
# SwingUI Test Runner

- `junit.swingui.TestRunner.run(HelloTest.class)`



# TextUI Test Runner

- `junit.textui.TestRunner.run(HelloTest.class)`  
不推荐使用



```
<terminated> HelloTest (1) [Java Application] D:\jdk1.4.2_04\bin\javaw.exe (2005-3-19 0:31:18)
.test for Hello!

Time: 0.06

OK (1 test)
```

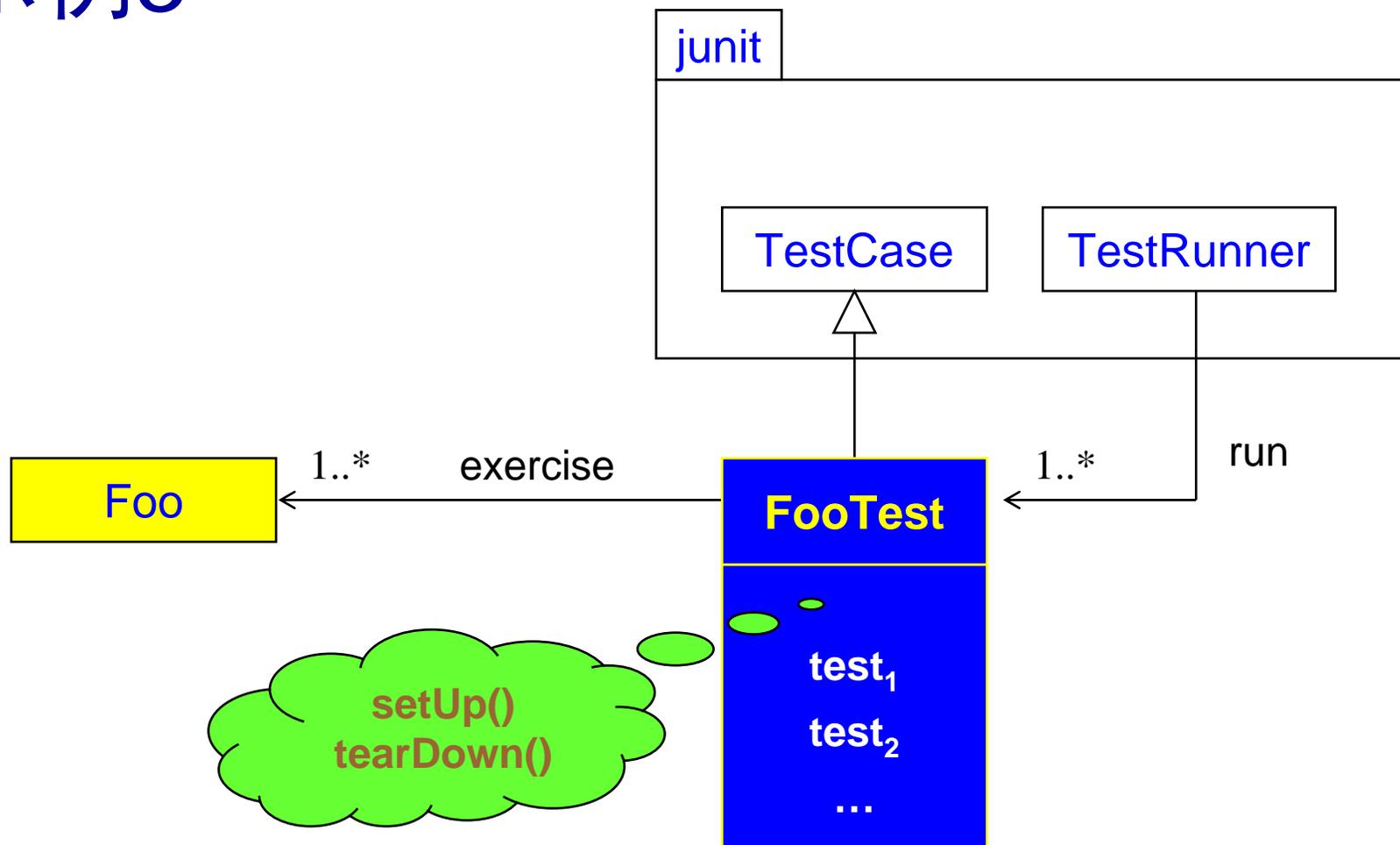
# Lesson Learned

- Generate & exercise test case(s)
  - Write **test...()** method(s)
- Verify
  - Choose a **Assert.assert...()**
- Choose a **TestRunner**
  - **junit.textui.TestRunner**
  - **junit.swingui.TestRunner**
- Failure vs. error

# JUnit进阶篇

- Fixture/context 管理
- Test suite
- 其他特性
- 断言类别
- 测试代码编写步骤
- 最佳实践

# 示例3



## 示例3 待测代码

```
public class Money implements Cloneable {  
    private int fAmount;  
    private String fCurrency;  
  
    public Money(int amount, String currency) { /*...*/ }  
  
    public Money add(Money m) {  
        if (m == null) return (Money) clone();  
        return new Money(amount() + m.amount(), currency());  
    }  
  
    public Object clone() { /*...*/ }  
    public boolean equals(Object obj) { /*...*/ }  
    public int amount() { /*...*/ }  
    public String currency() { /*...*/ }  
}
```

# Fixture

- setUp():
  - 用来初始化对象为某种特定状态、准备数据等。
- tearDown():
  - 用来撤销初始化、销毁数据。

## 示例3 测试代码

```
public class MoneyTest extends TestCase {  
    private Money money1, money2, money3;  
  
    public MoneyTest(String s)    { /*...*/ }  
  
    protected void setUp() {  
        String curr = "RMB";  
        int val1 = 2002;  
        int val2 = 345;  
  
        money1 = new Money(val1, curr);  
        money2 = new Money(val2, curr);  
        money3 = new Money(val1 + val2, curr);  
    }  
}
```

## 示例3 测试代码 [续]

```
protected void tearDown()    { }

public void testAddForNull() {
    Money money10 = money1.add(null); // null value
    assertEquals(money1, money10);
}

public void testAddForSimple() {
    Money money10 = money1.add(money2);
    assertEquals(money3, money10);
}
}
```

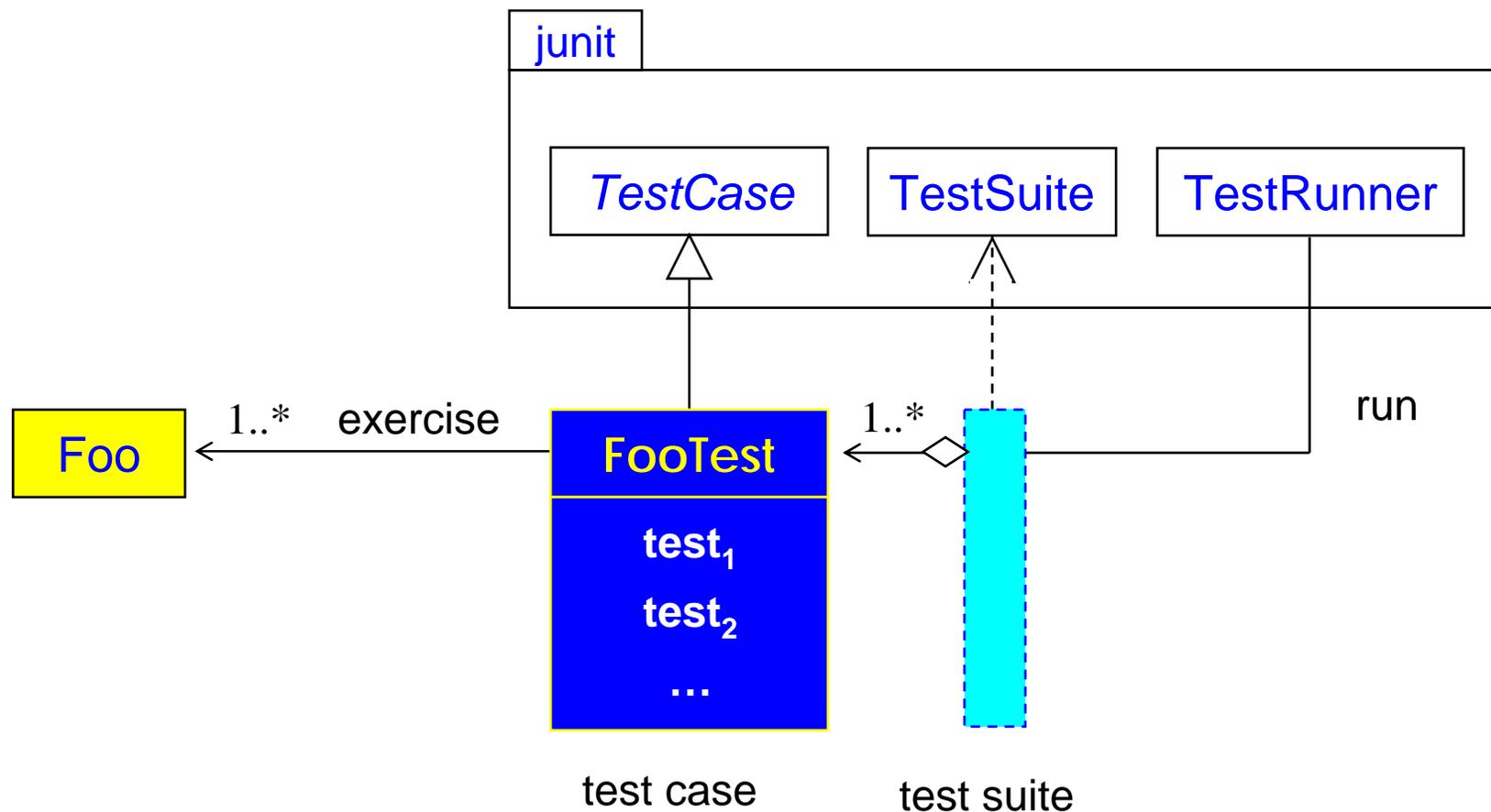
# Lesson Learned

- Fixture
  - Overrides `setUp()`
  - Overrides `tearDown()`
- Each `test...()` method is isolated
- TestCase lifecycle
  1. `setUp`
  2. `testXXX()`
  3. `tearDown()`
  4. Repeats 1 through 3 for each `testXXX` method...

# 再论Factorial

- 练习：使用JUnit实现Factorial的单体测试代码

# 示例4



# TestSuite

- 包含TestCase的容器。
- 加入多个测试的类
- 加入不同类的多个测试方法。
- 将一个包下的TestCase包含到一个suite中。

## 示例4 TestSuite

```
package com.company;
import junit.framework.*;
public class AllTests {
    public static Test suite() {
        TestSuite suite = new TestSuite("Test for
com.company");
        suite.addTestSuite>HelloWorldTest.class);
        suite.addTestSuite(CarTest.class);
        return suite;
    }
}
```

# Lessons Learned

- Test case suite
  - Write a **suite()** as the interactive mode entry point
  - Write a **main()** as the batch mode entry point

# 重复测试

- 问题：需要重复运行测试。
- 方案：使用junit.extensions.RepeatedTest类。

## 重复测试 [续]

- 举例:
- `import junit.framework.*;`
- `import junit.extensions.*;`
- `public class MyRepeatedTest`
- `{`
- `public static Test suite()`
- `{`
- `TestSuite suite = new TestSuite();`
- `suite.addTest(new RepeatedTest(new`
- `TestSuite(MoneyTest.class), 10));`
- `return suite;`
- `}`
- `}`

# 断言类别

断言方法	描述
<code>assertEquals</code>	比较两件事物是否相等（基本类型或对象）
<code>assertTrue</code>	对布尔值求值，看它是否为 <code>true</code>
<code>assertFalse</code>	对布尔值求值，看它是否为 <code>false</code>
<code>assertNull</code>	检查对象是否为 <code>null</code>
<code>assertNotNull</code>	检查对象是否不为 <code>null</code>
<code>assertSame</code>	检查两个对象是否为同一实例
<code>assertNotSame</code>	检查两个对象是否不为同一实例

# 测试代码编写步骤

1. 创建并初始化被测试对象
2. 调用被测方法，取得方法的返回值或系统状态变化
3. 创建“期待结果”
4. 调用适合的assert方法

# JUnit Best Practices

- Separate production and test code
- Compile into separate trees, allowing deployment without tests
- Don't forget OO techniques, base classing

# 练习

- 商店促销
- 根据性别取得赠品； 男性赠送剃须刀， 女性赠送护手霜。
- 设计：

```
String gainPresent(String sex){  
    if sex等于“男性” { return “剃须刀” ;}  
    else { return “护手霜” ;}  
}
```

# 参考实现代码

```
/* ver1 存在bug */  
package com.company;  
public class ShopAssistant {  
    public String gainPresent(String sex) {  
        if (sex == "男性") {  
            return "剃须刀";  
        }  
        else  
        {  
            return "护手霜";  
        }  
    }  
}
```

# 讲解练习

参看示例代码

# 总结

- 掌握单元测试的概念
- 编写单元测试代码
- 执行单元测试

# Thank you

# 谢谢

Neusoft Group Ltd.

Neusoft Group Ltd.