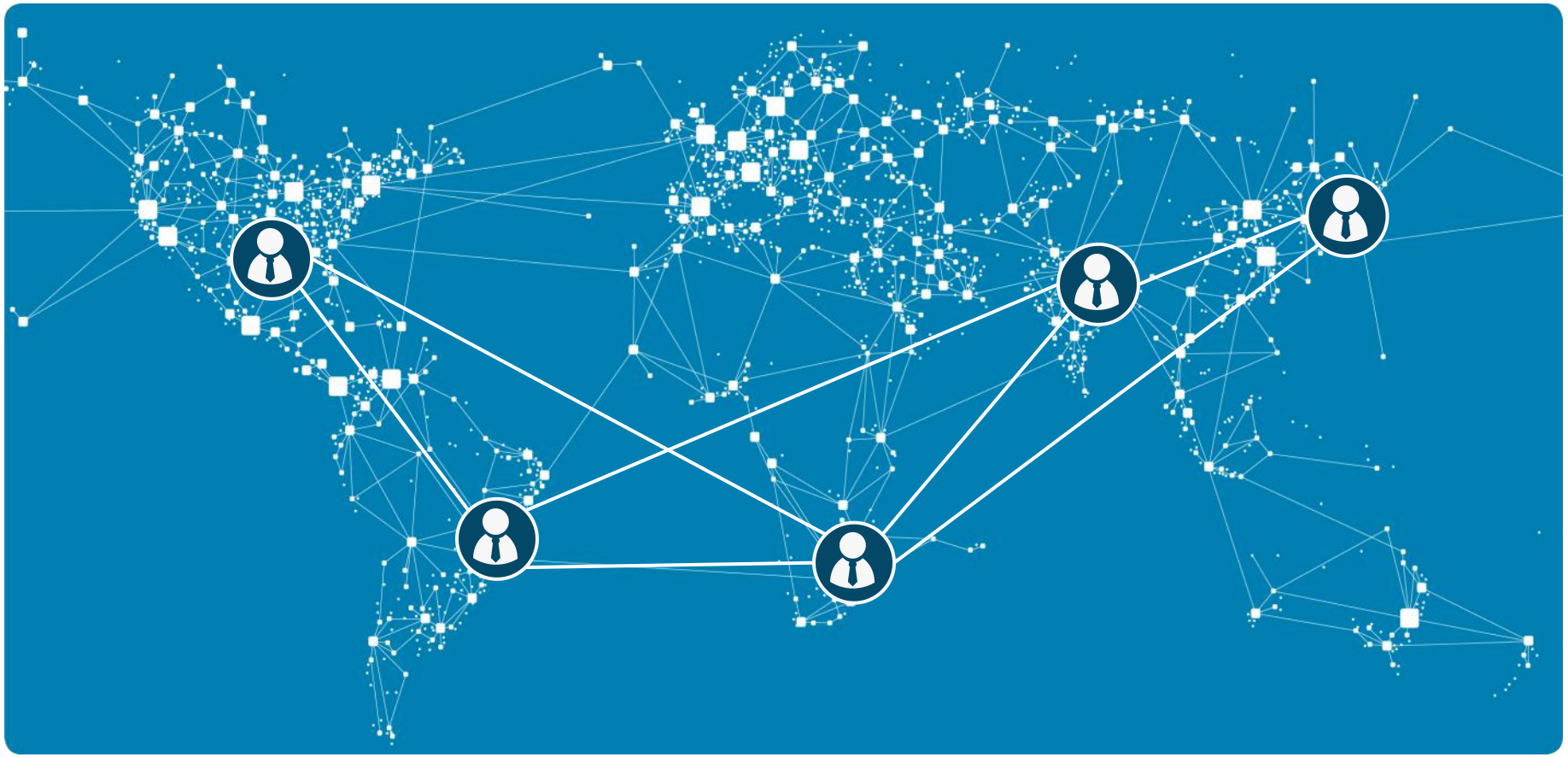


实验二：单元测试

单元测试

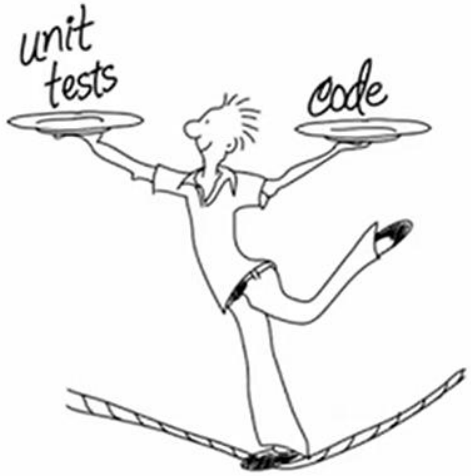
单元测试（Unit Testing）是对软件中的最小可测试单元进行检查和验证。

单元测试

程序员必须对自己的代码质量负责，单元测试是对自己代码质量的基本承诺。

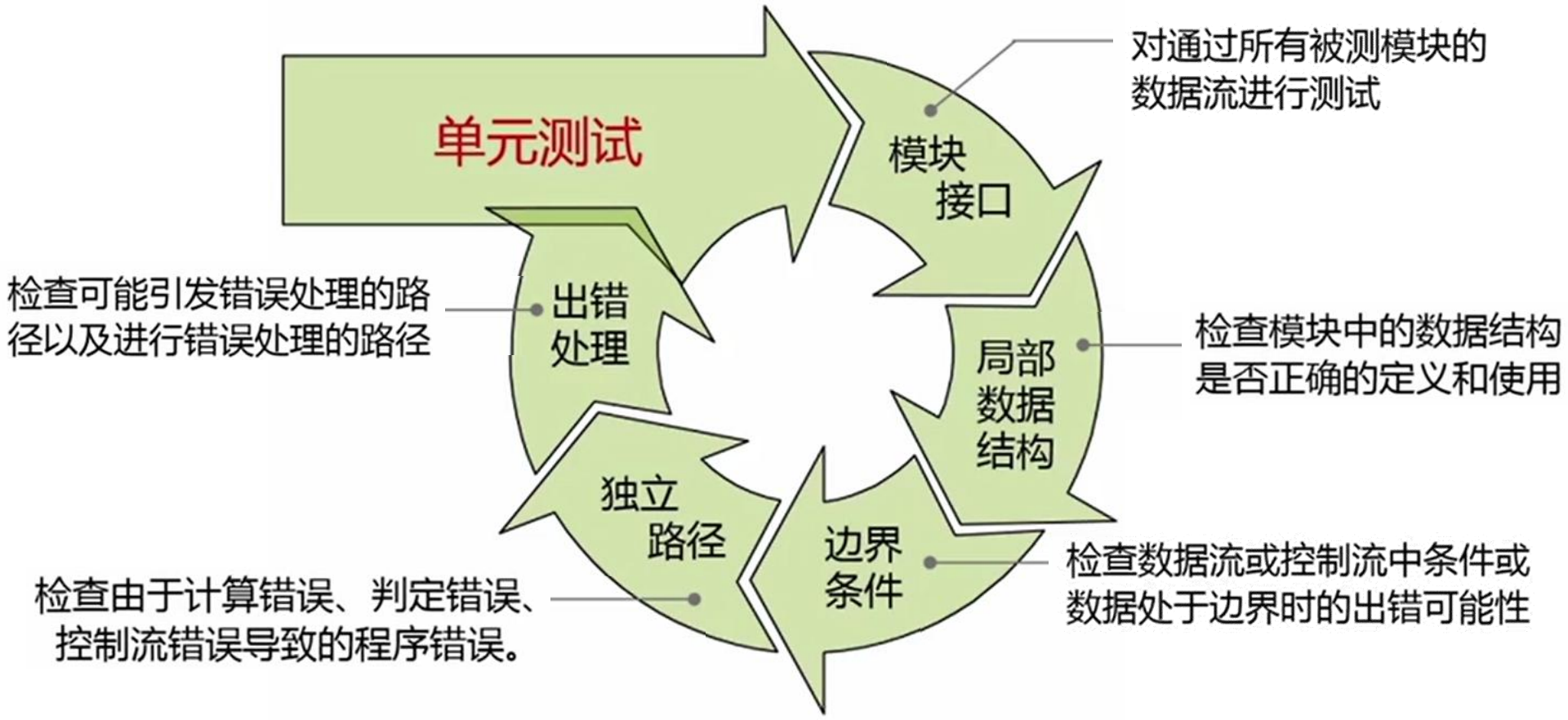
程序 = UT + CODE

测试人员有权利对没有做过UT的代码说No



在现实中，代码质量最好、开发速度最快的程序员是单元测试做得最好的。

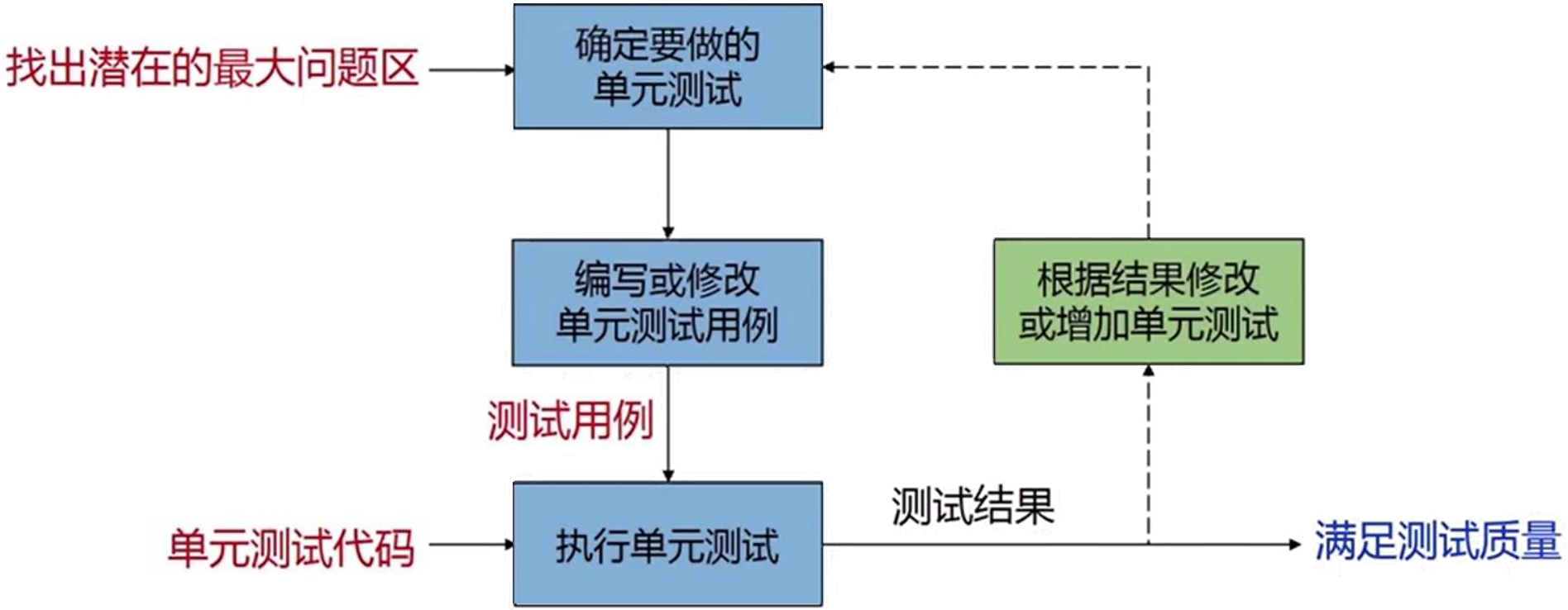
单元测试的内容


单元测试
模块接口
对通过所有被测模块的数据流进行测试
局部数据结构
检查模块中的数据结构是否正确的定义和使用
边界条件
检查数据流或控制流中条件或数据处于边界时的出错可能性
独立路径
检查由于计算错误、判定错误、控制流错误导致的程序错误。
出错处理
检查可能引发错误处理的路径以及进行错误处理的路径

单元测试原则

快速的	单元测试应能快速运行，如果运行缓慢，就不会愿意频繁运行它。
独立的	单元测试应相互独立，某个测试不应为下一个测试设定条件。当测试相互依赖时，一个没通过就会导致一连串的失败，难以定位问题。
可重复的	单元测试应该是可以重复执行的，并且结果是可以重现的。
自我验证的	单元测试应该有布尔输出，无论是通过或失败，不应该查看日志文件或手工对比不同的文本文件来确认测试是否通过。
及时的	及时编写单元测试代码，应恰好在开发实际的单元代码之前。

单元测试过程


找出潜在的最大问题区
确定要做的
单元测试
编写或修改
单元测试用例
测试用例
单元测试代码
执行单元测试
测试结果
根据结果修改
或增加单元测试
满足测试质量

单元测试质量

测试
通过率

测试通过率是指在测试过程中执行通过的测试用例所占比例，单元测试通常要求测试用例通过率达到100%。

测试
覆盖率

测试覆盖率是用来度量测试完整性的一个手段，通过覆盖率数据，可以了解测试是否充分以及弱点在哪里。代码覆盖率是单元测试的一个衡量标准，但也不能一味地去追求覆盖率。

单元测试质量

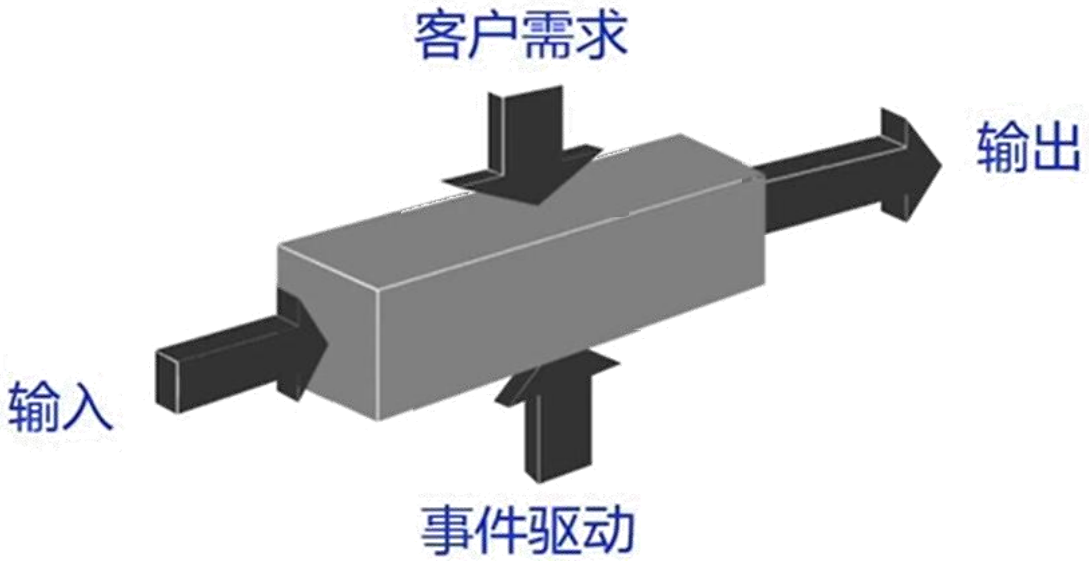
单元测试方法

静态测试：通过人工分析或程序正确性证明的方式来确认程序正确性。

动态测试：通过动态分析和程序测试等方法来检查和确认程序是否有问题。

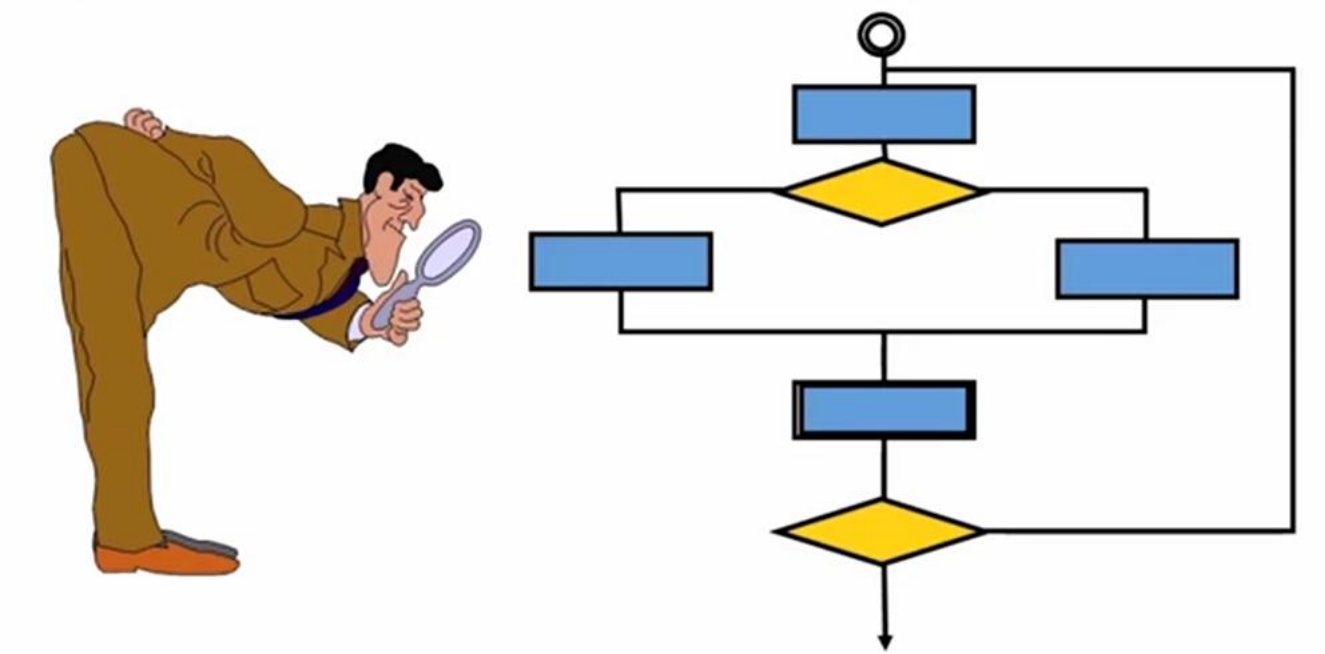
单元测试方法

黑盒测试（Black Box Testing）：又称功能测试，它将测试对象看做一个黑盒子，完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。

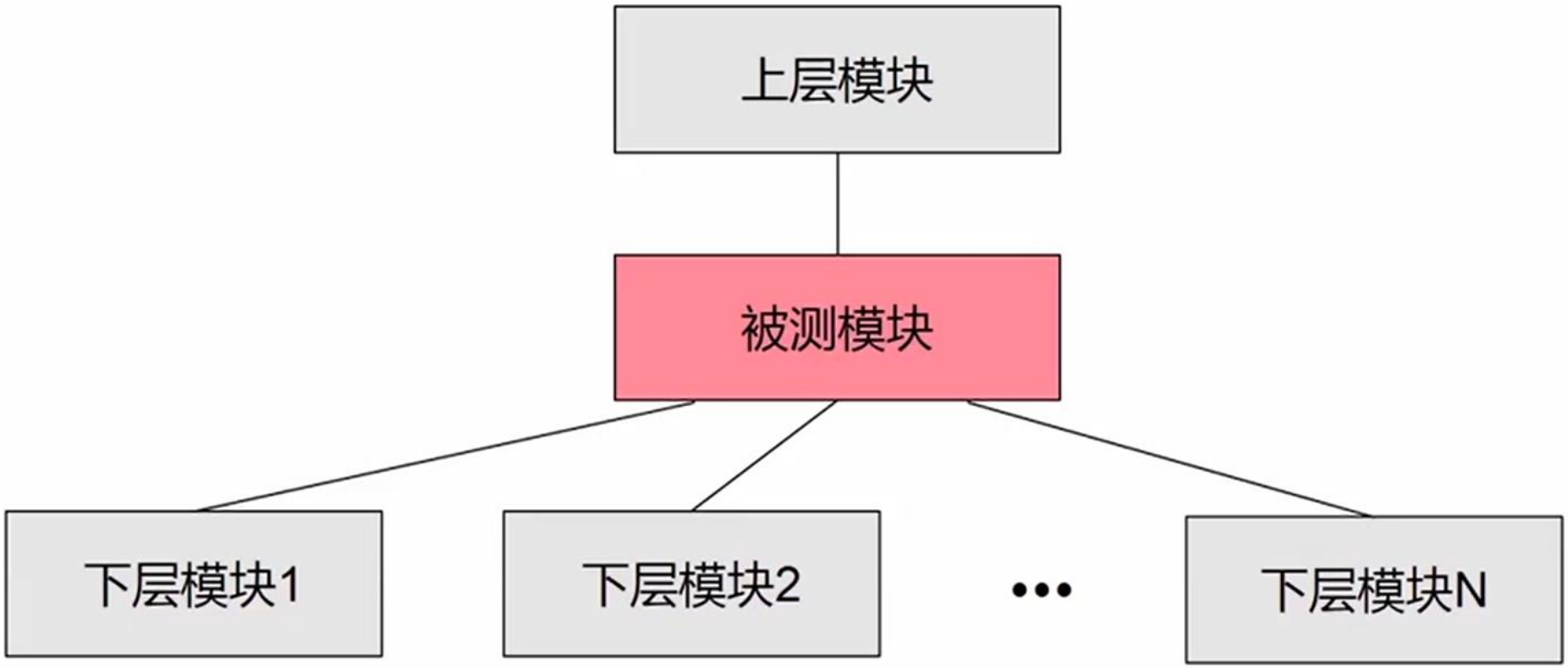


单元测试方法

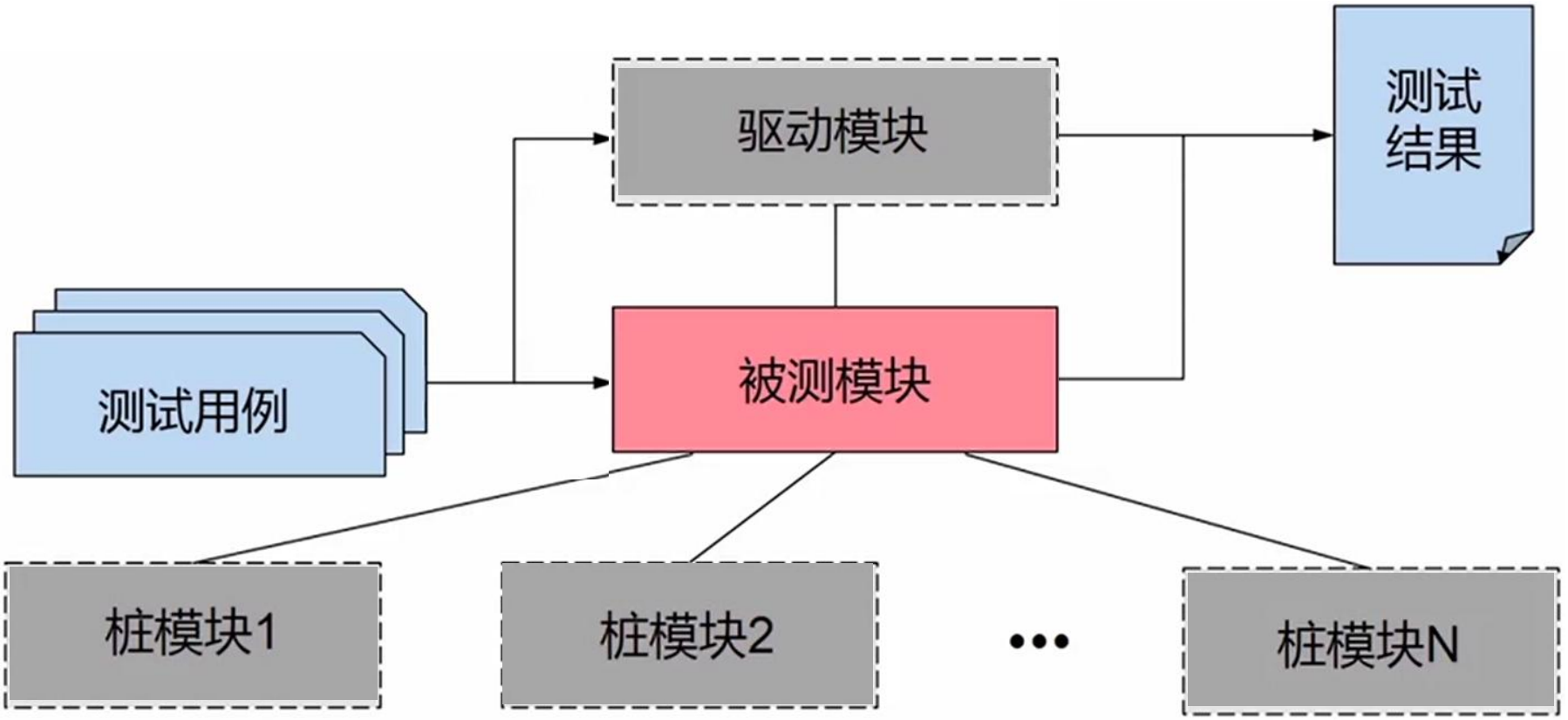
白盒测试（White Box Testing）：又称结构测试，它把测试对象看做一个透明的盒子，允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。



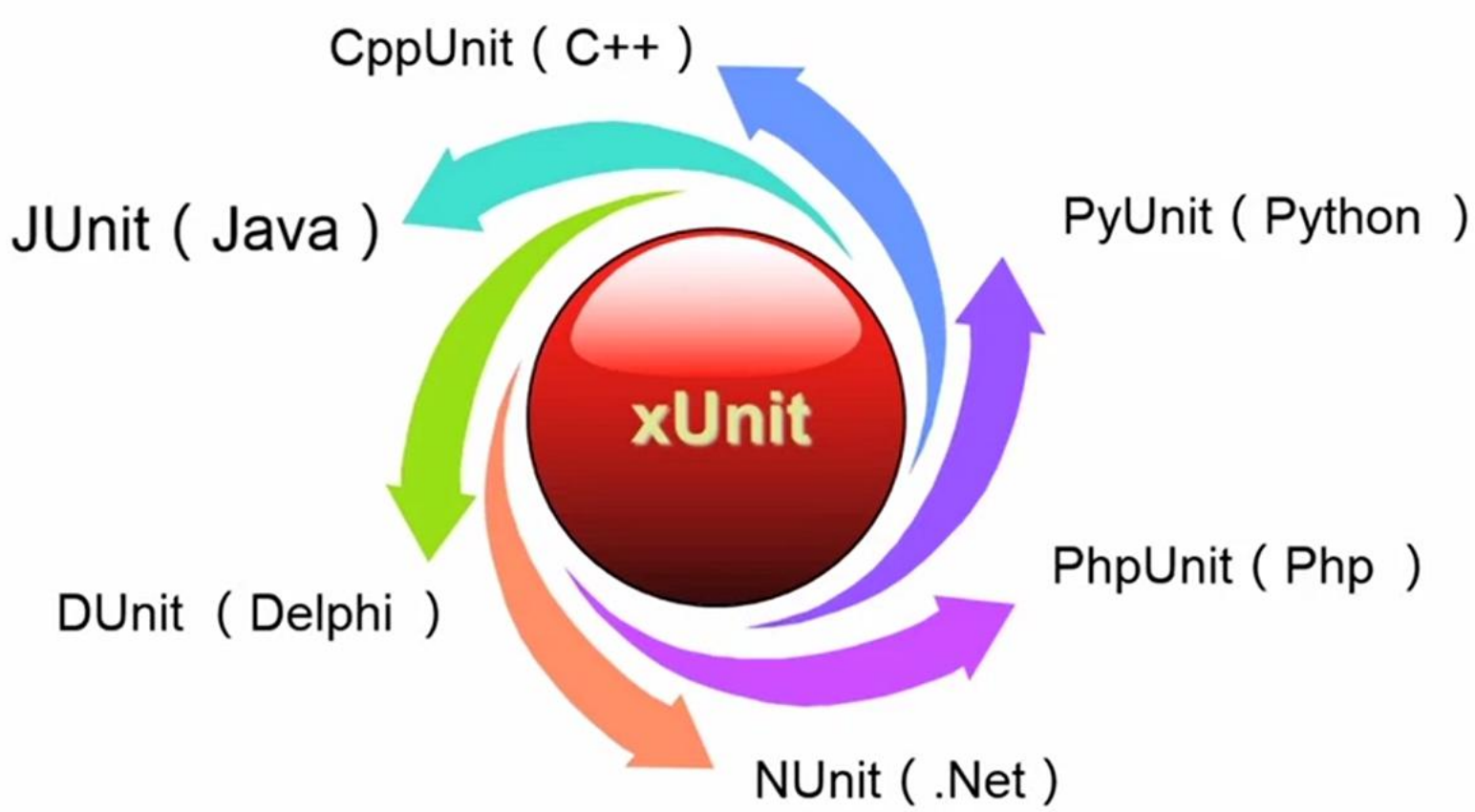
单元测试方法



单元测试方法



单元测试之xUnit


CppUnit (C++)
JUnit (Java)
PyUnit (Python)
xUnit
PhpUnit (Php)
DUnit (Delphi)
NUnit (.Net)

单元测试之xUnit

xUnit 通常适用于以下场景的测试

- 单个函数、一个类或者几个功能相关类的测试
- 尤其适用于纯函数测试或者接口级别的测试

xUnit 无法适用于复杂场景的测试

- 被测对象依赖关系复杂，甚至无法简单创建出这个对象
- 对于一些失败场景的测试
- 被测对象中涉及多线程合作
- 被测对象通过消息与外界交互的场景

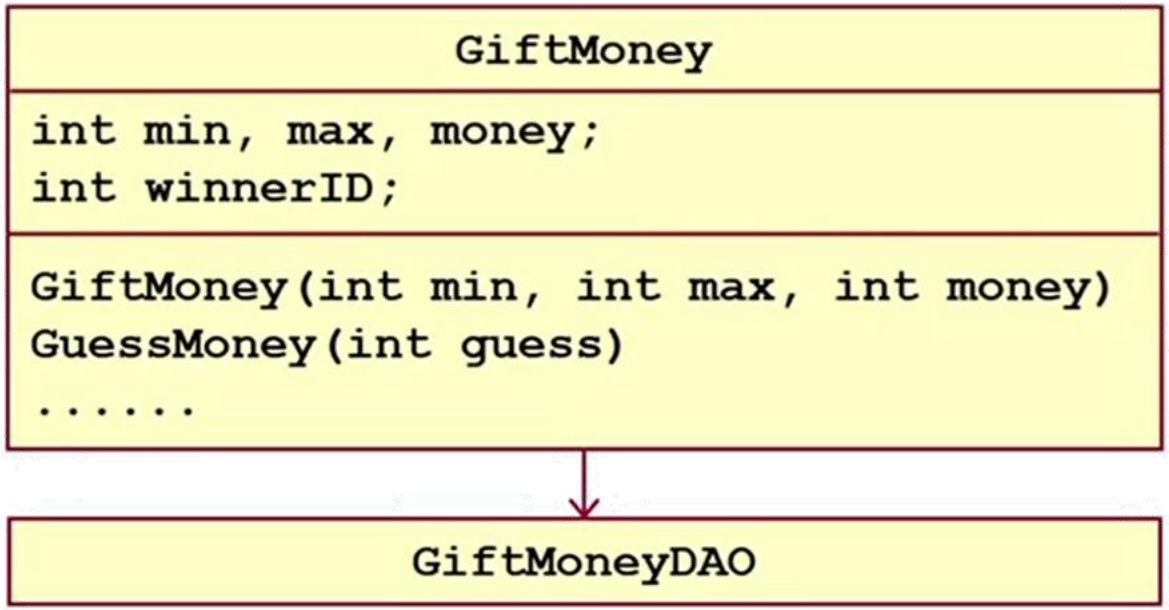
单元测试之Mock方法

Mock测试是在测试过程中对于某些不容易构造或者不容易获取的对象，用一个虚拟的对象（即**Mock**对象）来创建以便测试的方法。

- 真实对象具有不可确定的行为（产生不可预测的结果）
- 真实对象很难被创建（如具体的**Web**容器）
- 真实对象的某些行为很难触发（如网络错误）
- 真实情况令程序的运行速度很慢
- 真实对象有用户界面
- 测试需要询问真实对象它是如何被调用的
- 真实对象实际上并不存在

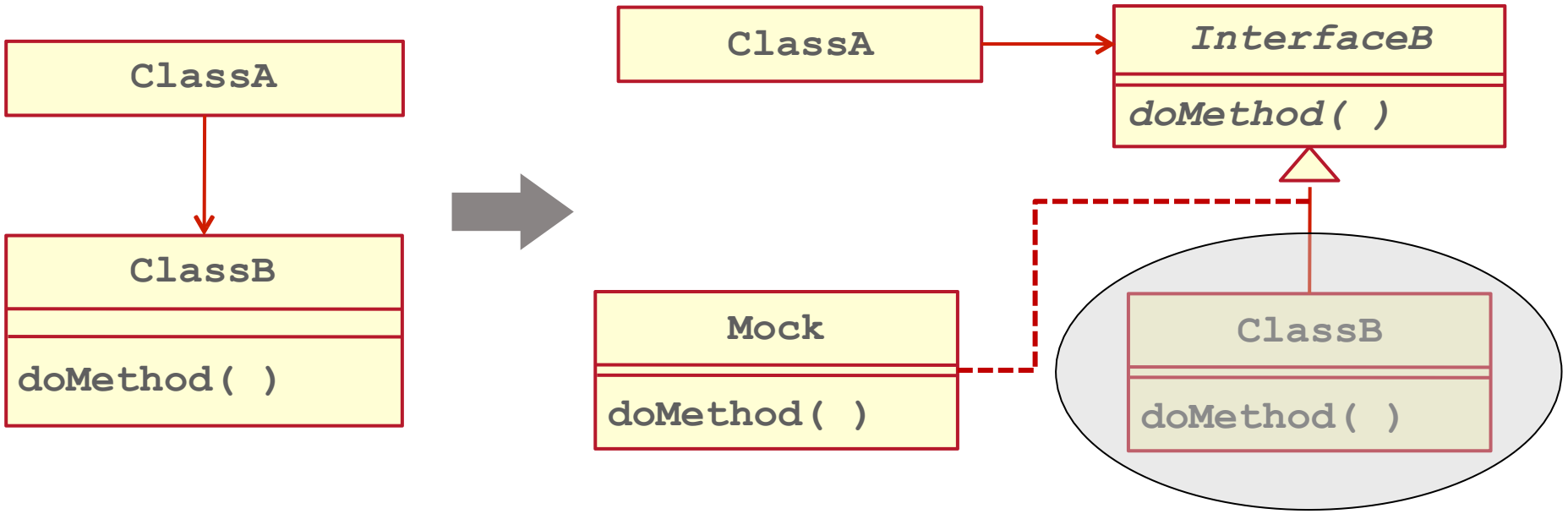
单元测试之Mock方法

举例：支付宝接龙红包通过猜金额的小游戏方式，实现朋友之间的互动并领取春节红包。这种情况应如何测试？



单元测试之Mock方法

关键：需要应用针对接口的编程技术，即被测试的代码通过接口来引用对象，再使用**Mock**对象模拟所引用的对象及其行为，因此被测试模块并不知道它所引用的究竟是真实对象还是**Mock**对象。



Eclipse中JUnit的使用

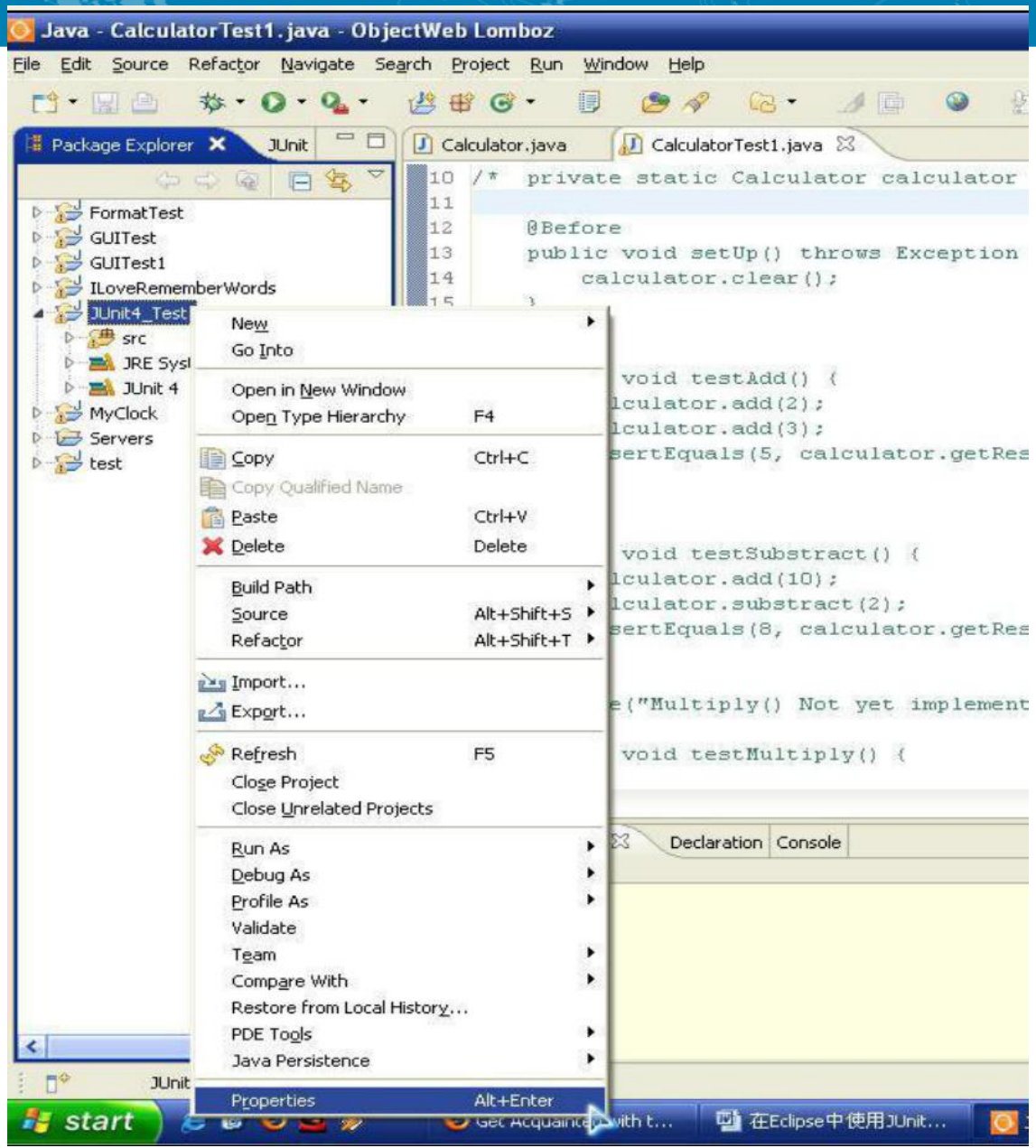
Eclipse集成了JUnit，可以非常方便地编写Test Case。Eclipse自带了一个JUnit插件，不用安装就可以在项目中测试相关的类，并且可以调试测试用例和被测类。

举例：建立一个基于JUnit4的测试项目，对Calculator类当中的多个方法进行单元测试。

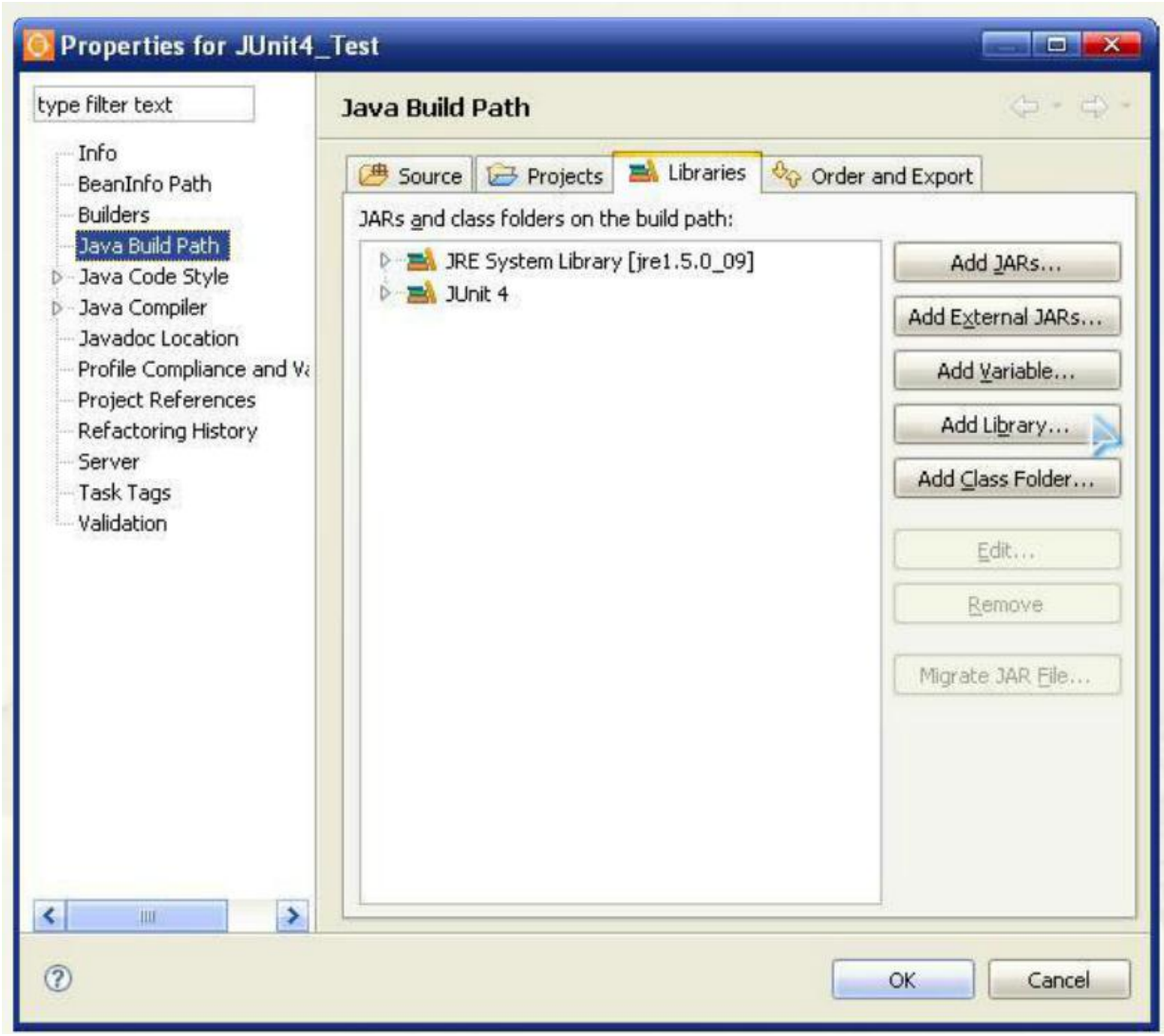
（1）新建一个名为JUnitTest的项目，在其中编写如下的Calculator类。

```
public class Calculator{
    private static int result; //静态变量，用于存储运行结果
    public void add(int n) {   result=result+n;   }
    public void substract(int n)
    {  result=result-1; }  //Bug：正确的应该是result=result-n;
    public void multiply(int n){ }   //此方法尚未写好
    public void divide(int n)  {   result=result/n; }
    public void square(int n)  {   result=n*n;}
    public void squareRoot(int n)
    {   for(;;);  }  //Bug:死循环
    public void clear()
    {    result=0; } //将结果清零
     public int getResult()
    {    return result; }
}
```

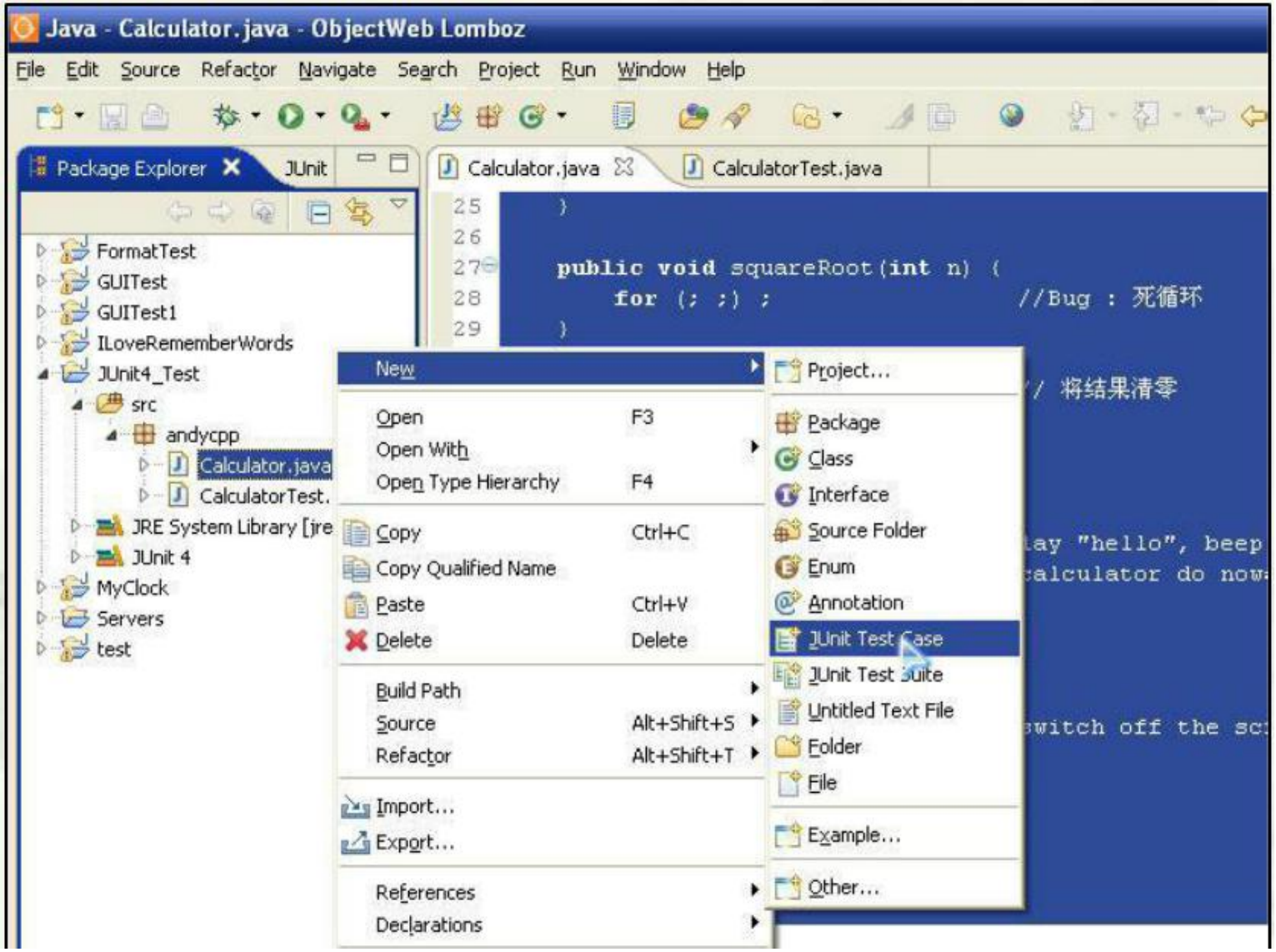
(2) 将JUnit4单元测试包引入该项目



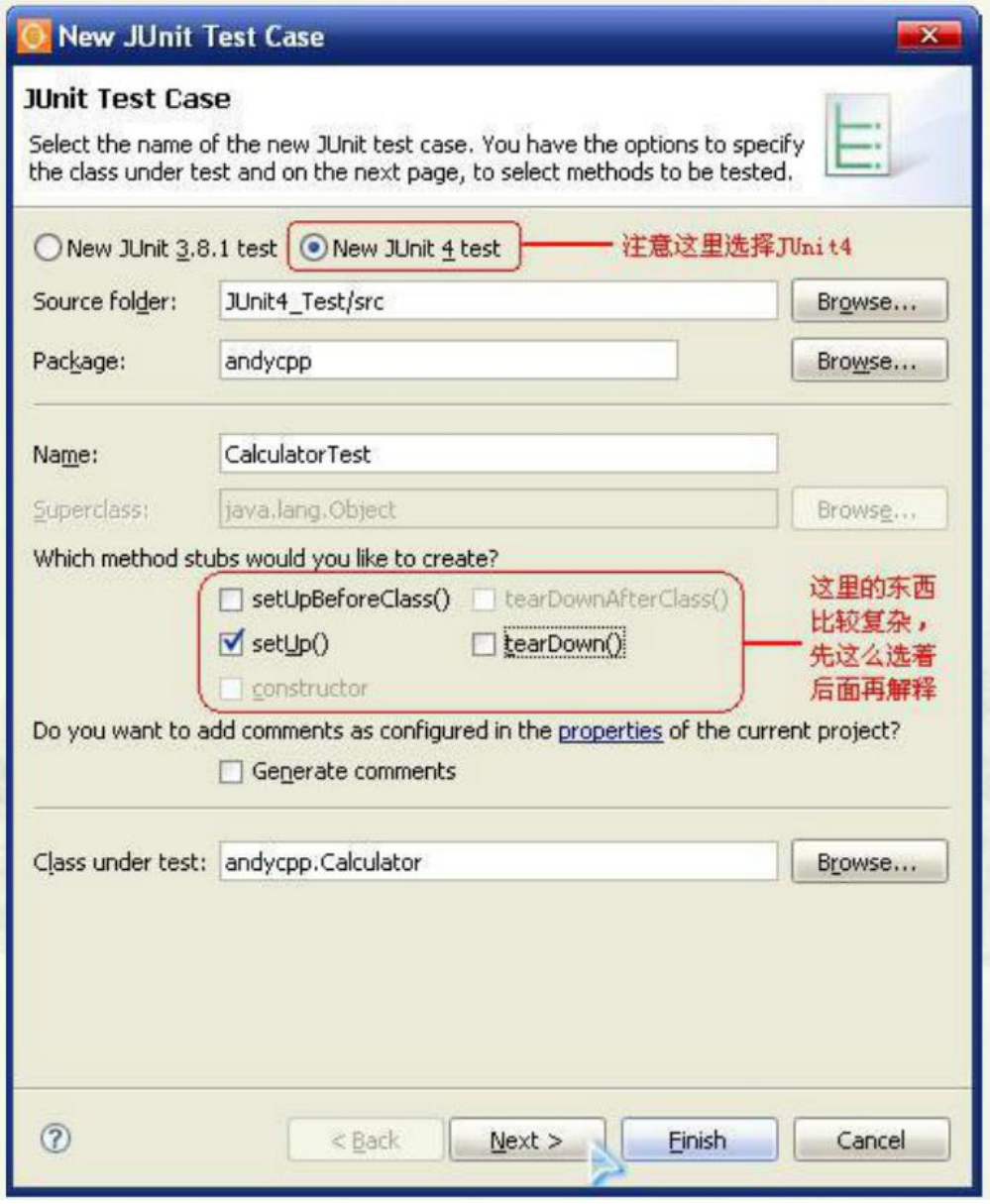
(2) 将JUnit4单元测试包引入该项目



(3) 生成JUnit测试框架



（3）生成JUnit测试框架



Eclipse

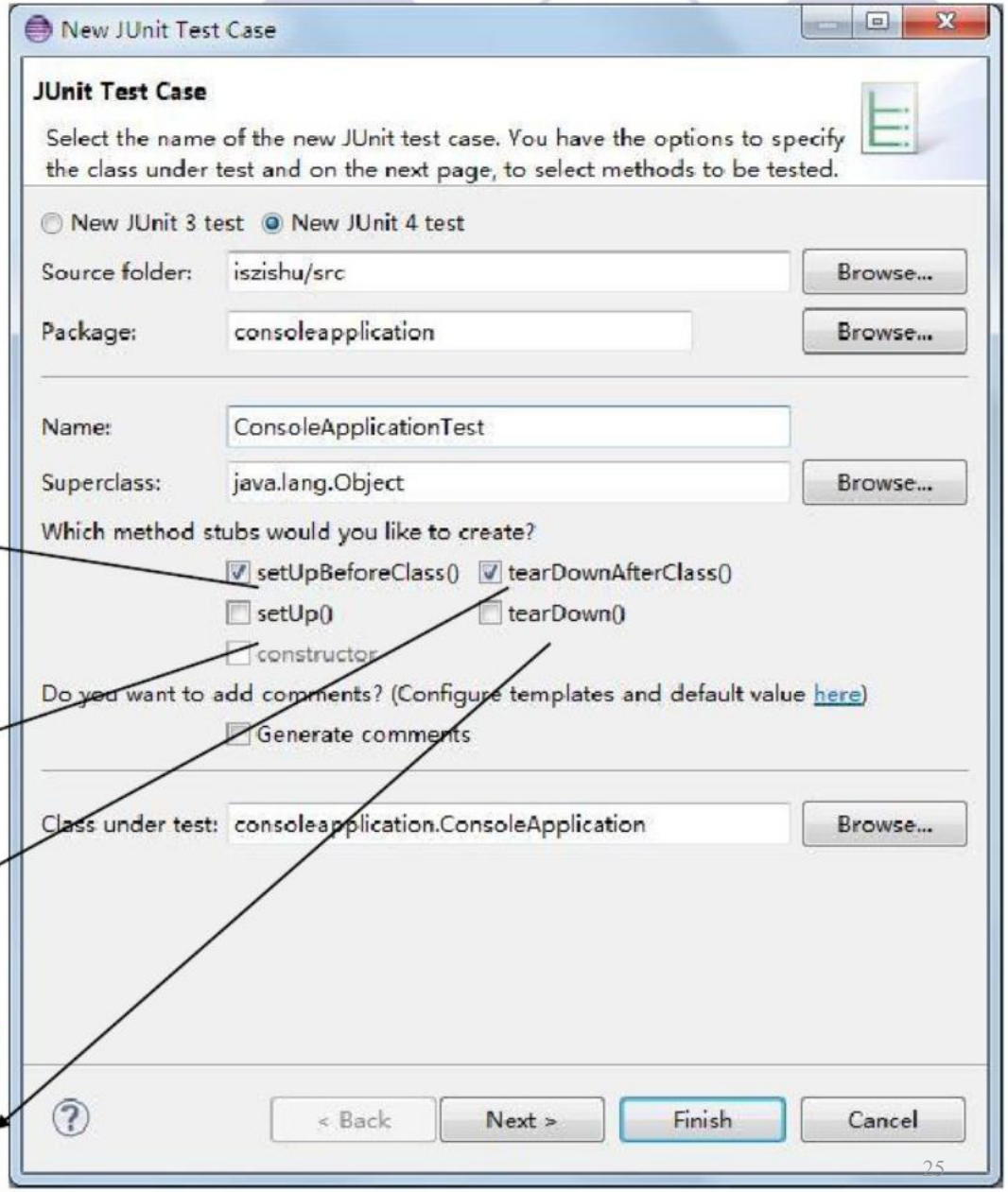
- 如右图填写
- 单击Next

方法在整个类初始化后调用，一般作测试的准备工作。

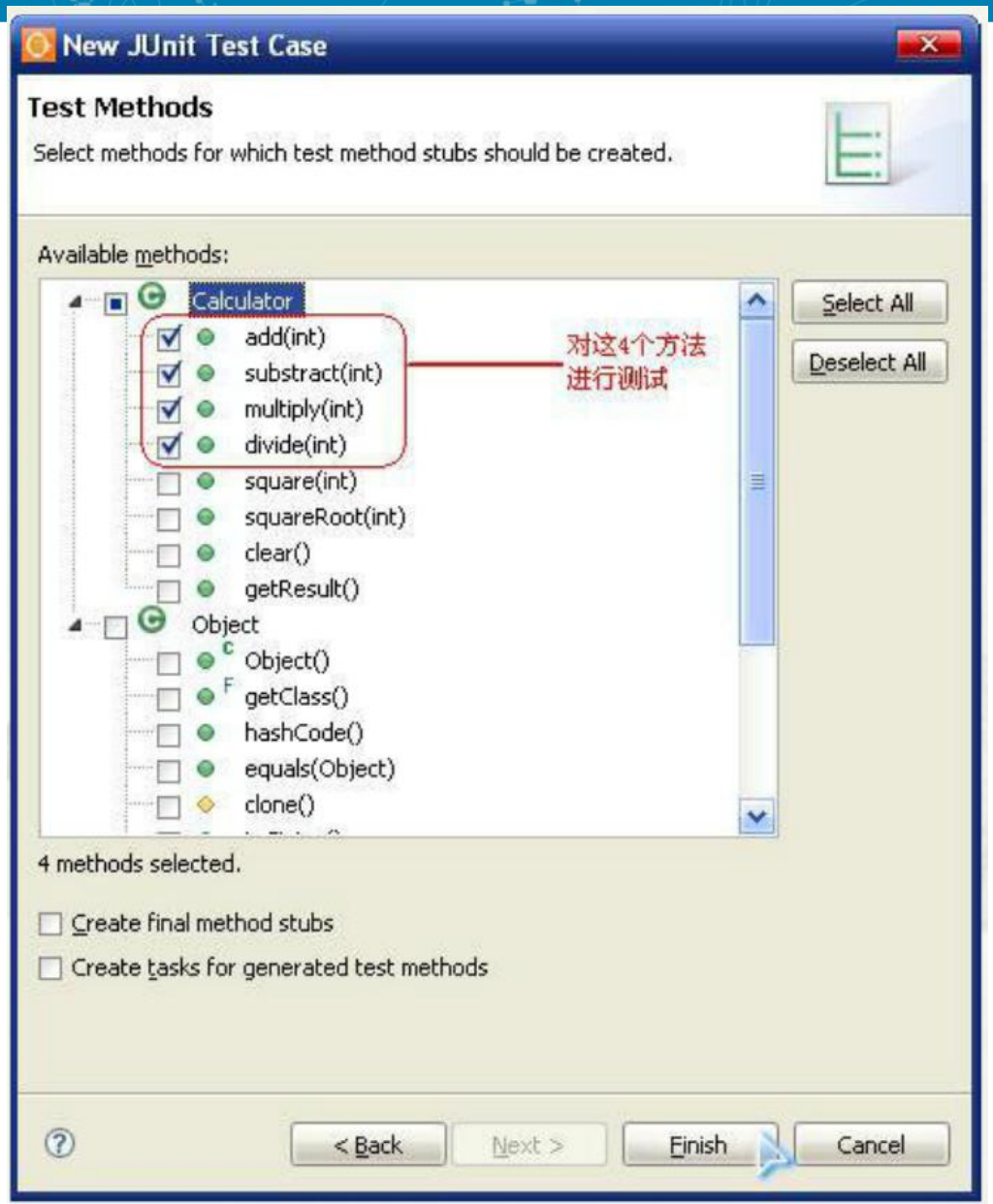
在测试方法前调用，一般用来做测试准备工作。

方法在整个类结束之前调用，一般作测试的清理工作。

在测试方法后调用，一般作测试的清理工作。



(3) 生成JUnit测试框架



CalculatorTest类

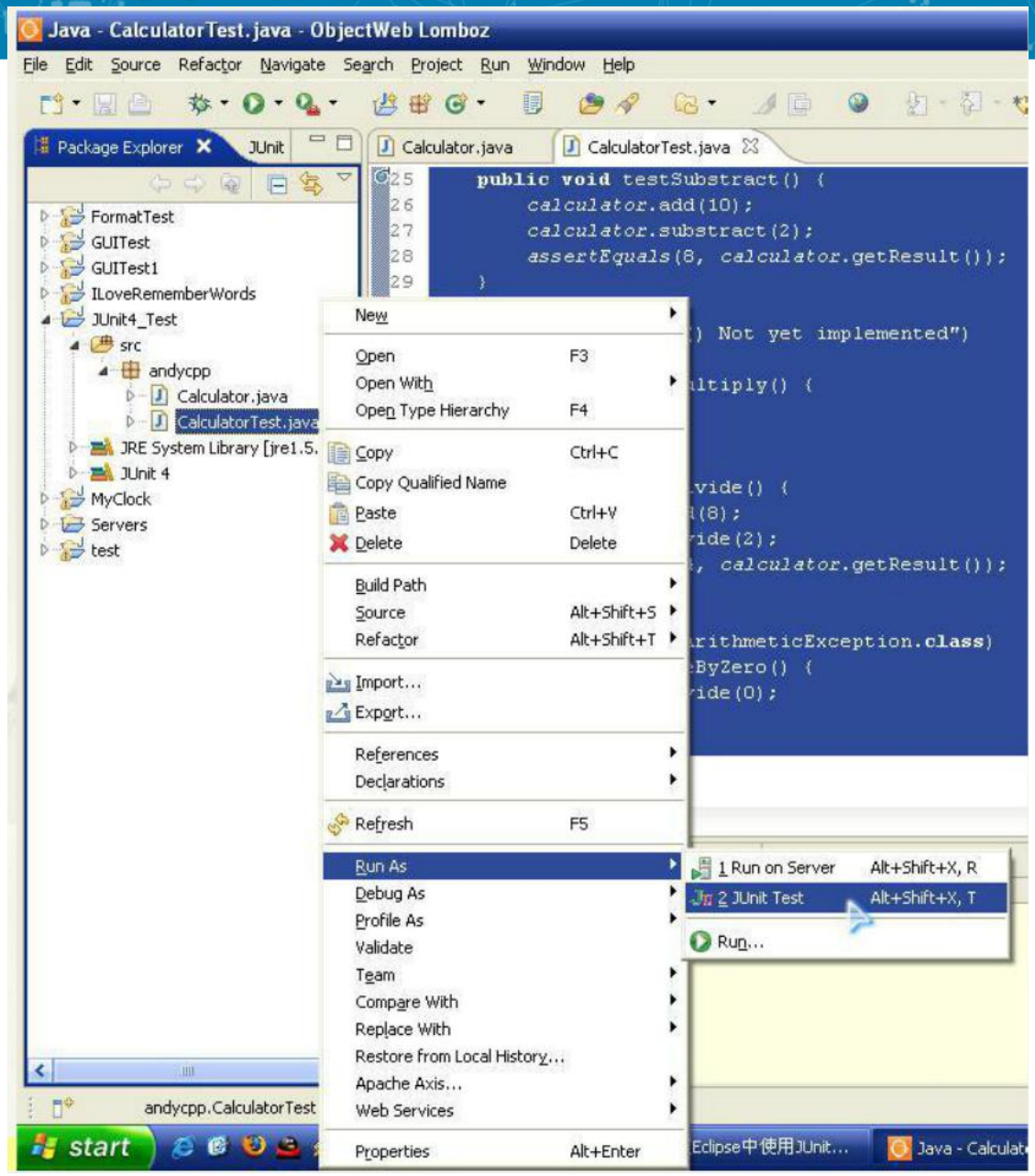
```
package andycpp;
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import
org.junit.internal.runners.TestClassRunner;
import org.junit.runner.RunWith;

@RunWith(TestClassRunner.class)
public class CalculatorTest {


    @Before
    public void setUp() throws Exception {

    }
    @After
    public void tearDown() throws Exception
{
    }
```

```
    @Test
        public void testAdd() {



        }
     @Test
     public void testSubstract() {



        }
     @Ignore("Multiply() Not yet implemented")
     @Test
     public void testMultiply() {
     }
     @Test
        public void testDivide() {



        }
}
```

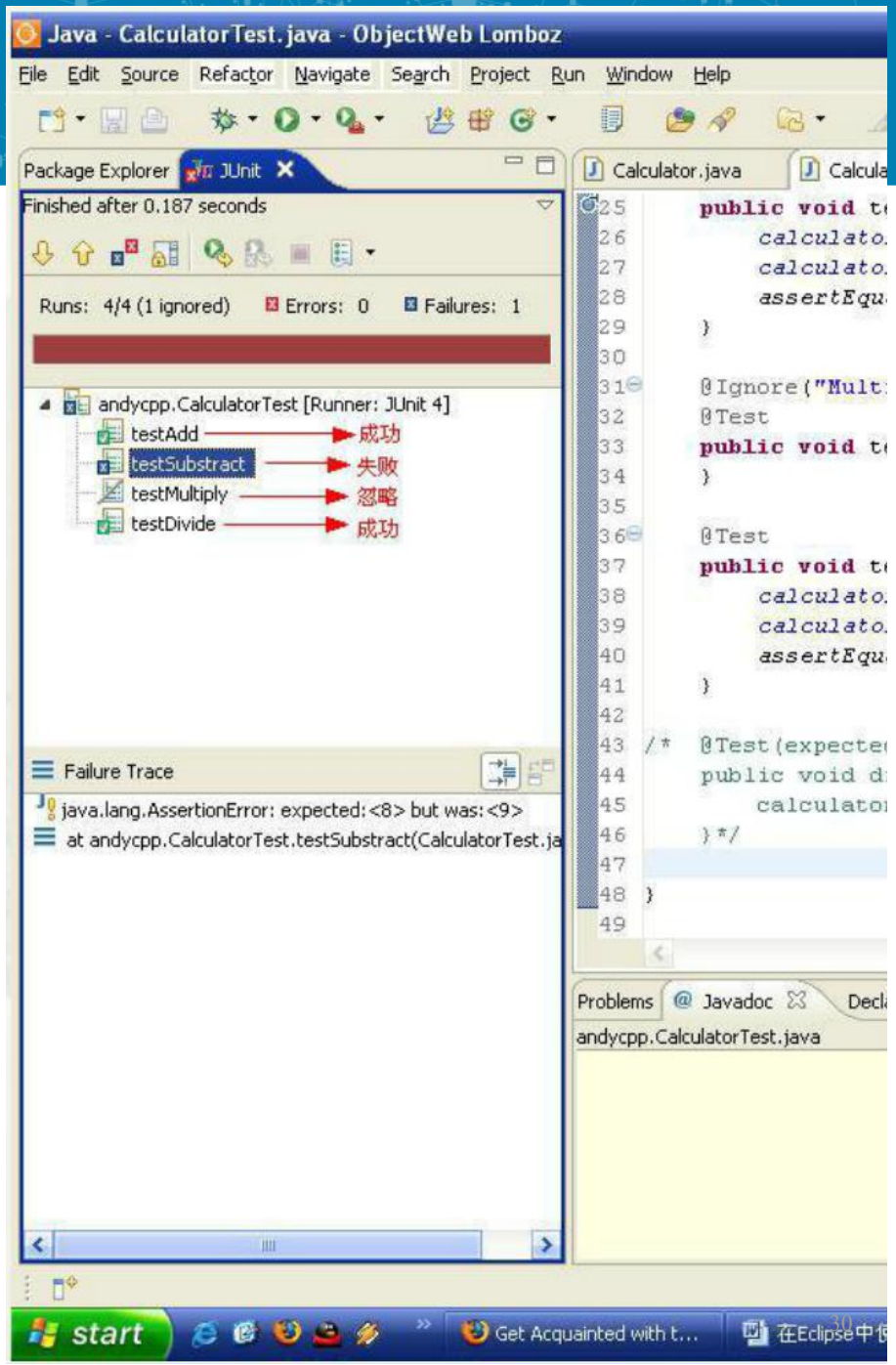
CalculatorTest类

```
package andycpp;
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import
org.junit.internal.runners.TestClassRunner;
import org.junit.runner.RunWith;

@RunWith(TestClassRunner.class)
public class CalculatorTest {
private static Calculator calculator=new
Calculator();
    @Before
    public void setUp() throws Exception {
        calculator.clear();
    }
    @After
    public void tearDown() throws Exception
{
    }
    @Test(timeout=1000)
        public void testAdd() {
         calculator.add(2);
         calculator.add(3);
         assertEquals(5, calculator.getResult());
        }
     @Test
     public void testSubstract() {
         calculator.add(10);
         calculator.substract(2);
         assertEquals(8,calculator.getResult());
        }
     @Ignore("Multiply() Not yet implemented")
     @Test
     public void testMultiply() {
     }
     @Test(expected =ArithmeticException.class)
        public void testDivide() {
         calculator.add(8);
         calculator.divide(0);
         assertEquals(4,calculator.getResult());
        }
}
```

（4）运行测试代码


Java - CalculatorTest.java - ObjectWeb Lomboz
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Calculator.java
CalculatorTest.java
FormatTest
GUITest
GUITest1
ILoveRememberWords
JUnit4_Test
src
andycpp
Calculator.java
CalculatorTest.java
JRE System Library [jre1.5.
JUnit 4
MyClock
Servers
test
public void testSubstract() {
calculator.add(10);
calculator.substract(2);
assertEquals(8, calculator.getResult());
}
) Not yet implemented")
ltiply() {
vide() {
(8);
ide(2);
, calculator.getResult());
rithmeticException.class)
ByZero() {
ide(0);
New
Open F3
Open With
Open Type Hierarchy F4
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Run As
Debug As
Profile As
Validate
Team
Compare With
Replace With
Restore from Local History...
Apache Axis...
Web Services
Properties Alt+Enter
1 Run on Server Alt+Shift+X, R
2 JUnit Test Alt+Shift+X, T
Run...
andycpp.CalculatorTest
start
Eclipse中使用JUnit...
Java - Calculat

（4）运行测试结果


Java - CalculatorTest.java - ObjectWeb Lomboz
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
Finished after 0.187 seconds
Runs: 4/4 (1 ignored)
Errors: 0
Failures: 1
andycpp.CalculatorTest [Runner: JUnit 4]
testAdd
成功
testSubstract
失败
testMultiply
忽略
testDivide
成功
Failure Trace
java.lang.AssertionError: expected:<8> but was:<9>
at andycpp.CalculatorTest.testSubstract(CalculatorTest.ja
Calculator.java
Problems
Javadoc
andycpp.CalculatorTest.java
start
Get Acquainted with t...

测试框架细节

API for JUnit:

http://junit.sourceforge.net/javadoc/

Assert class:

http://junit.sourceforge.net/javadoc/org/junit/Assert.html

一、包含必要地Package

在测试类中用到了JUnit4框架，自然要把相应地Package包含进来。最主要地一个Package就是org.junit.*。把它包含进来之后，绝大部分功能就有了。还有一句话也非常地重要“import static org.junit.Assert.*;”，我们在测试的时候使用的一系列assertEquals方法就来自这个包。大家注意一下，这是一个静态包含（static），是JDK5中新增添的一个功能。也就是说，assertEquals是Assert类中的一系列的静态方法，一般的使用方式是Assert.assertEquals()，但是使用了静态包含后，前面的类名就可以省略了，使用起来更加的方便。

二、测试类的声明

大家注意到，我们的测试类是一个独立的类，没有任何父类。测试类的名字也可以任意命名，没有任何局限性。所以我们不能通过类的声明来判断它是不是一个测试类，它与普通类的区别在于它内部的方法的声明，我们接着会讲到。

测试框架细节

三、创建一个待测试的对象。

你要测试哪个类，那么你首先就要创建一个该类的对象。正如上一篇文章中的代码：

```
private static Calculator calculator = new Calculator();
```

为了测试Calculator类，我们必须创建一个calculator对象。

四、测试方法的声明

在测试类中，并不是每一个方法都是用于测试的，你必须使用“标注”来明确表明哪些是测试方法。“标注”也是JDK5的一个新特性，用在此处非常恰当。我们可以看到，在某些方法的前有@Before、@Test、@Ignore等字样，这些就是标注，以一个“@”作为开头。这些标注都是JUnit4自定义的，熟练掌握这些标注的含义非常重要。

测试框架细节

五、编写一个简单的测试方法。

首先，你要在方法的前面使用@Test标注，以表明这是一个测试方法。对于方法的声明也有如下要求：名字可以随便取，没有任何限制，但是返回值必须为void，而且不能有任何参数。如果违反这些规定，会在运行时抛出一个异常。至于方法内该写些什么，那就要看你需要测试些什么了。比如：

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

我们想测试一下“加法”功能时候正确，就在测试方法中调用几次add函数，初始值为0，先加2，再加3，我们期待的结果应该是5。如果最终实际结果也是5，则说明add方法是正确的，反之说明它是错的。assertEquals(5, calculator.getResult());就是来判断期待结果和实际结果是否相等，第一个参数填写期待结果，第二个参数填写实际结果，也就是通过计算得到的结果。这样写好之后，JUnit会自动进行测试并把测试结果反馈给用户。

测试框架细节

六、忽略测试某些尚未完成的方法。

如果你在写程序前做了很好的规划，那么哪些方法是什么功能都应该实现定下来。因此，即使该方法尚未完成，他的具体功能也是确定的，这也就意味着你可以为他编写测试用例。但是，如果你已经把该方法的测试用例写完，但该方法尚未完成，那么测试的时候一定是“失败”。这种失败和真正的失败是有区别的，因此JUnit提供了一种方法来区别他们，那就是在这种测试函数的前面加上@Ignore标注，这个标注的含义就是“某些方法尚未完成，暂不参与此次测试”。这样的话测试结果就会提示你有几个测试被忽略，而不是失败。一旦你完成了相应函数，只需要把@Ignore标注删去，就可以进行正常的测试。

测试框架细节

七、Fixture（暂且翻译为“固定代码段”）

Fixture的含义就是“在某些阶段必然被调用的代码”。比如我们上面的测试，由于只声明了一个Calculator对象，他的初始值是0，但是测试完加法操作后，他的值就不是0了；接下来测试减法操作，就必然要考虑上次加法操作的结果。这绝对是一个很糟糕的设计！我们非常希望每一个测试都是独立的，相互之间没有任何耦合度。因此，我们就很有必要在执行每一个测试之前，对Calculator对象进行一个“复原”操作，以消除其他测试造成的影响。因此，“在任何一个测试执行之前必须执行的代码”就是一个Fixture，我们用@Before来标注它，如前面例子所示：

```
@Before

public void setUp() throws Exception ...{

    calculator.clear();

}
```

这里不在需要@Test标注，因为这不是一个test，而是一个Fixture。同理，如果“在任何测试执行之后需要进行的收尾工作”也是一个Fixture，使用@After来标注。由于本例比较简单，没有用到此功能。

实验内容

- 实验目的
  - 掌握单元测试的方法；
  - 学习JUnit测试原理及框架；
  - 掌握在Eclipse环境中加载JUnit及JUnit测试方法和过程。
- 实验要求
  - 利用JUnit对“实验一”中的各个类，进行单元测试。

Java单元测试(Junit+Mock+代码覆盖率)：

http://www.cnblogs.com/AloneSword/p/4109407.html