实验一 JUnit 测试基础（2 学时）

一、实验目的：

通过本次实验，熟悉 JUnit 测试环境，了解 JUnit 测试框架的使用，运行测试用例对 Java 程序测试。

二、实验准备：

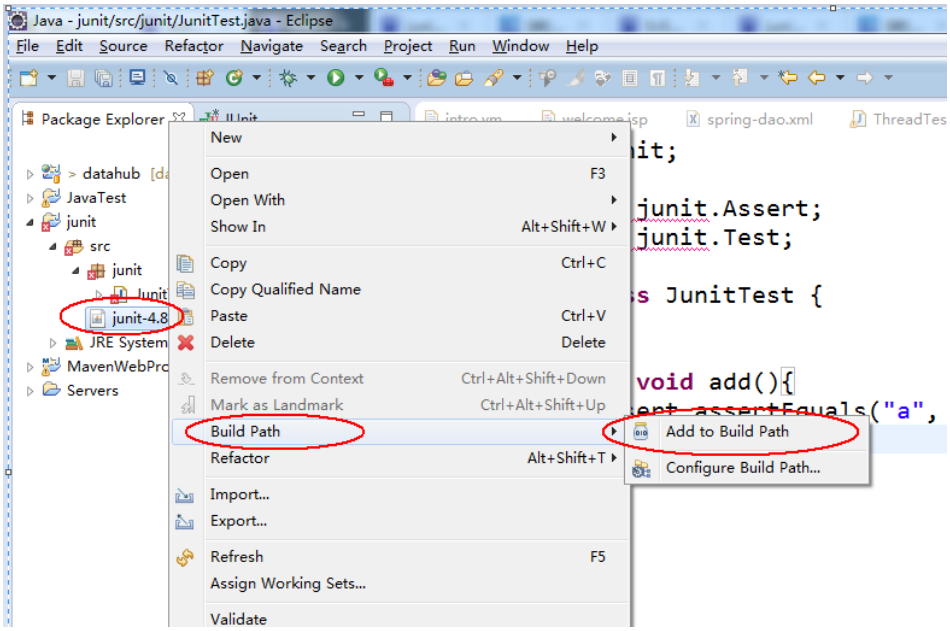
1、JUnit 简介

1) JUnit 是什么？

JUnit 是一个 Java 语言的单元测试框架。它由 Kent Beck 和 Erich Gamma 建立，逐渐成为源于 Kent Beck 的 sUnit 的 xUnit 家族中最为成功的一个 JUnit 有它自己的 JUnit 扩展生态圈。多数 Java 的开发环境都已经集成了 JUnit 作为单元测试的工具。

2) JUnit 能做什么？

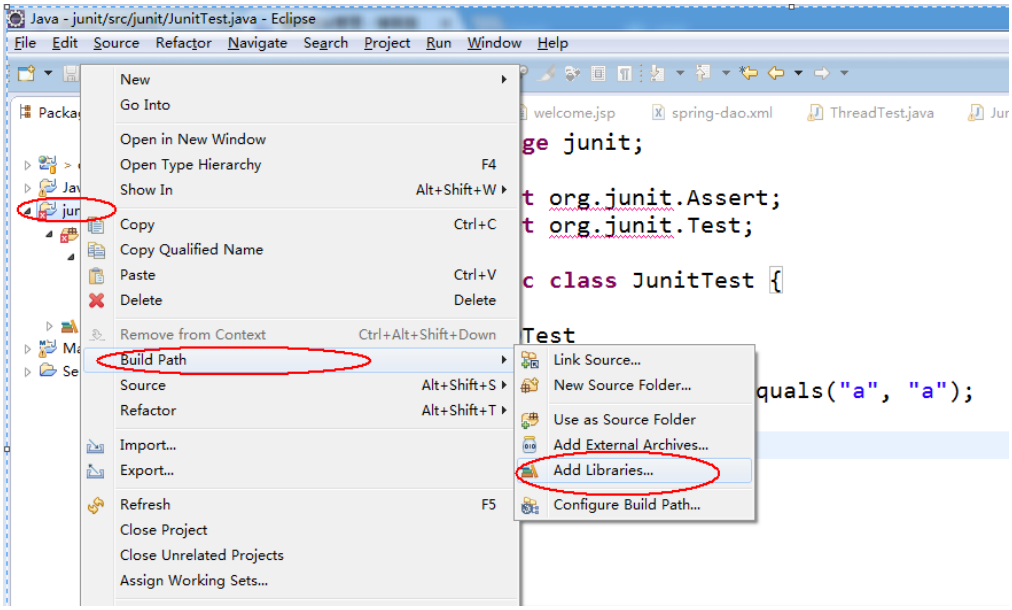
我们知道 JUnit 是一个单元测试框架，那么使用 JUnit 能让我们快速的完成单元测试。通常我们写完代码想要测试这段代码的正确性，那么必须新建一个类，然后创建一个 main() 方法，然后编写测试代码。如果需要测试的代码很多呢？那么要么就会建很多 main() 方法来测试，要么将其全部写在一个 main() 方法里面。这也会大大的增加测试的复杂度，降低程序员的测试积极性。而 JUnit 能很好的解决这个问题，简化单元测试，写一点测一点，在编写以后的代码中如果发现问题可以较快的追踪到问题的原因，减小回归错误的纠错难度。



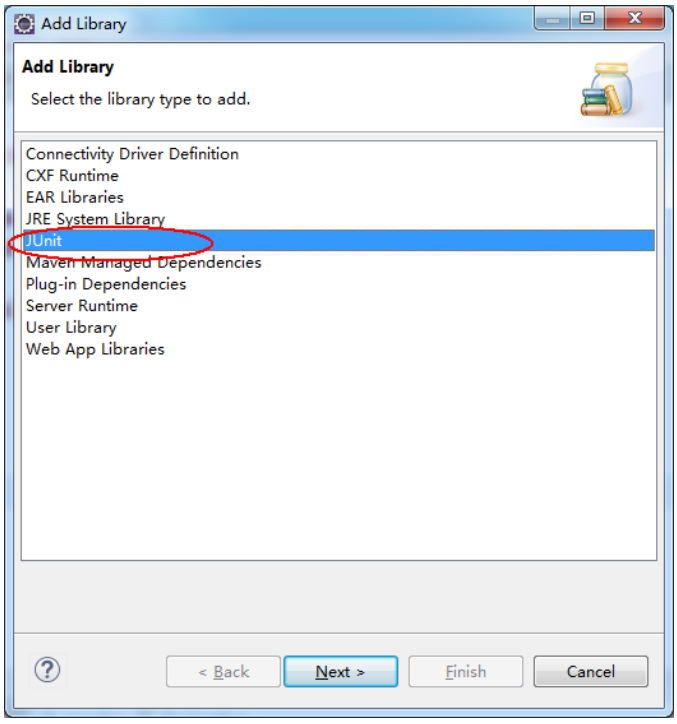
3) JUnit 的用法：

在项目中将 下载的 jar 包放进去，然后右键，Build--->Add to Build Path 即可，如上图所示。

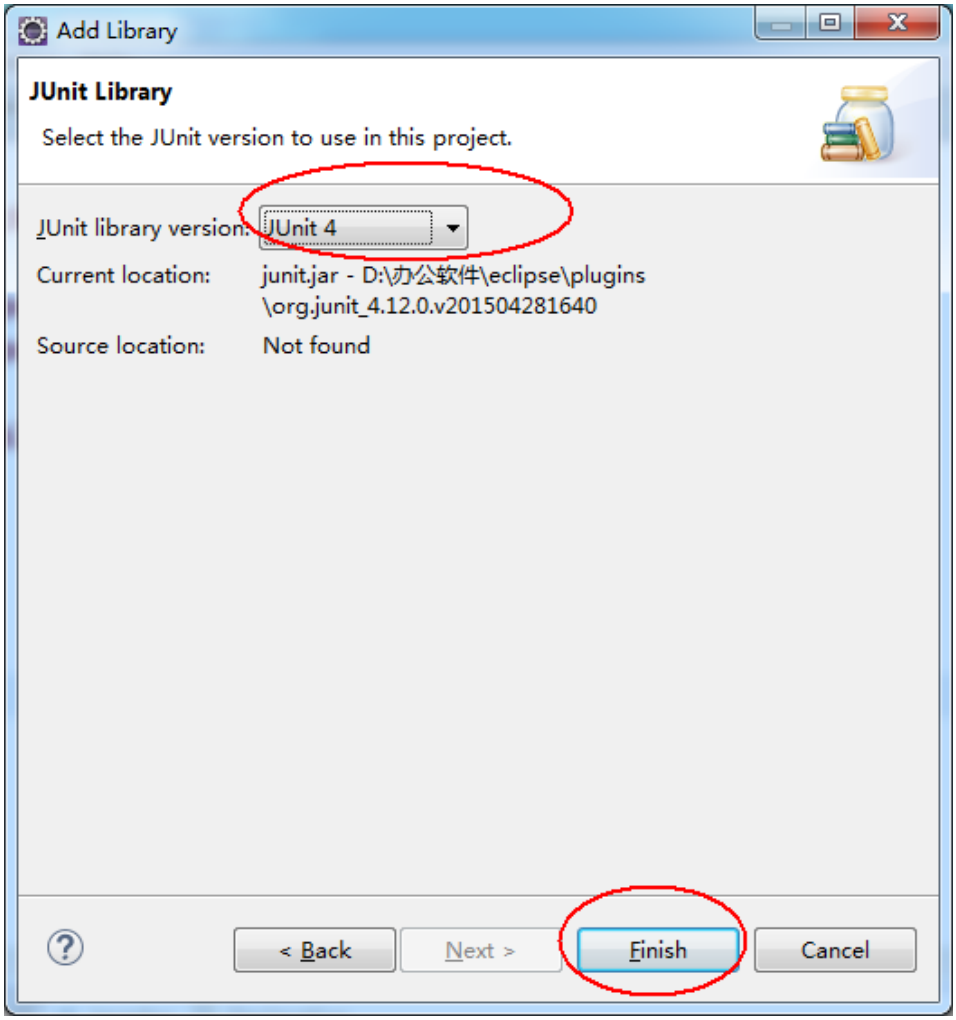
如果你是用 eclipse 开发，也可以不用下载那些 jar 包，eclipse 内部集成了，只需要引入即可。选中项目，右键 Build--->Add Library



②、弹出来的界面，选中 JUnit，点击 next



③、选中 JUnit 的版本，一般我们都用 4.0 以上的。点击 Finish



JUnit 优点：

1）可以使测试代码与产品代码分开。

2）针对某一个类的测试代码通过较少的改动便可以应用于另一个类的测试。

3）易于集成到测试人员的构建过程中，JUnit 和 Ant 的结合可以实施增量开发。

4）JUnit 是公开源代码的，可以进行二次开发。

5）可以方便地对 JUnit 进行扩展。

编写原则：

1）是简化测试的编写，这种简化包括测试框架的学习和实际测试单元的编写。

2）是使测试单元保持持久性。

3）是可以利用既有的测试来编写相关的测试。

JUnit 的特征

1）使用断言方法判断期望值和实际值差异，返回 boolean 值。

2）测试驱动设备使用共同的初始化变量或者实例。

3）测试包结构便于组织和集成运行。

4）支持图型交互模式和文本交互模式。

JUnit 框架组成

1）对测试目标进行测试的方法与过程集合，可称为测试用例(testcase)。

2）测试用例的集合，可容纳多个测试用例(testcase)，将其称作测试包(testsuite)。

3）测试结果的描述与记录。(testresult) 。

4）测试过程中的事件监听者(testlistener)。

5）每一个测试方法所发生的与预期不一致状况的描述，称其测试失败元素(testfailure)

6）JUnit Framework 中的出错异常（assertionfailederror）。

JUnit 框架是一个典型的 composite 模式：testsuite 可以容纳任何派生自 test 的对象；当调用 testsuite 对象的 run()方法是，会遍历自己容纳的对象，逐个调用它们的 run()方法。

2、JUnit 中常用的接口和类

test 接口——运行测试和收集测试结果

1）test 接口使用了 composite 设计模式，是单独测试用例 （testcase），聚合测试模式（testsuite）及测试扩展（testdecorator）的共同接口。

2）它的 public int counttestcases（）方法，它来统计这次测试有多少个 testcase，另外一个方法就是 public void run（ testresult ），testresult 是实例接受测试结果， run 方法执行本次测试。

testcase 抽象类——定义测试中固定方法

testcase 是 test 接口的抽象实现，（不能被实例化，只能被继承）其构造函数 testcase(string name)根据输入的测试名称 name 创建一个测试实例。由于每一个 testcase 在创建时都要有一个名称，若某测试失败了，便可识别出是哪个测试失败。

testcase 类中包含的 setup()、teardown()方法。setup()方法集中初始化测试所需的所有变量和实例，并且在依次调用测试类中的每个测试方法之**前**再次执行 setup()方法。teardown()方法则是在每个测试方法之**后**，释放测试程序方法中引用的变量和实例。

开发人员编写测试用例时，只需继承 testcase，来完成 run 方法即可，然后 JUnit 获得测试用例，执行它的 run 方法，把测试结果记录在 testresult 之中。

assert 静态类——一系列断言方法的集合

assert 包含了一组静态的测试方法，用于期望值和实际值比对是否正确，即测试失败，assert 类就会抛出一个 assertionfailederror 异常，JUnit 测试框架将这种错误归入 failes 并加以记录，同时标志为未通过测试。如果该类方法中指定一个 string 类型的传参则该参数将被做为 assertionfailederror 异常的标识信息，告诉测试人员改异常的详细信息。

JUnit 提供了 6 大类 31 组断言方法，包括基础断言、数字断言、字符断言、布尔断言、对象断言。

其中 assertequals（object expcted,object actual)内部逻辑判断使用 equals()方法，这表明断言两个实例的内部哈希值是否相等时，最好使用该方法对相应类实例的值进行比较。而 assertsame（object expected,object actual）内部逻辑判断使用了 java 运算符“==”，这表明该断言判断两个实例是否来自于同一个引用（reference），最好使用该方法对不同类的实例的值进行比对。

asserequals(string message,string expected,string actual)该方法对两个字符串进行逻辑比对，如果不匹配则显示着两个字符串有差异的地方。comparisonfailure 类提供两个字符串的比对，不匹配则给出详细的差异字符。

testsuite 测试包类——多个测试的组合

1）testsuite 类负责组装多个 test cases。待测得类中可能包括了对被测类的多个测试，而 testsuit 负责收集这些测试，使我们可以在一个测试中，完成全部的对被测类的多个测试。

2）testsuite 类实现了 test 接口，且可以包含其它的 testsuites。它可以处理加入 test 时的所有抛出的异常。

3）testsuite 处理测试用例有 6 个规约（否则会被拒绝执行测试）

a 测试用例必须是公有类（public）

b 测试用例必须继承与 testcase 类

c 测试用例的测试方法必须是公有的（ public ）

d 测试用例的测试方法必须被声明为 void

e 测试用例中测试方法的前置名词必须是 test

f 测试用例中测试方法误任何传递参数

testresult 结果类和其它类与接口

testresult 结果类集合了任意测试累加结果，通过 testresult 实例传递个每个测试的 run（）方法。testresult 在执行 testcase 是如果失败会异常抛出

testlistener 接口是个事件监听规约，可供 testrunner 类使用。它通知 listener 的对象相关事件，方法包括测试开始 starttest(test test)，测试结束 endtest(test test),错误，增加异常 adderror(test test,throwable t)和增加失败 addfailure(test test,assertionfailederror t)

testfailure 失败类是个“失败”状况的收集类，解释每次测试执行过程中出现的异常情况。其 tostring()方法返回“失败”状况的简要描述。

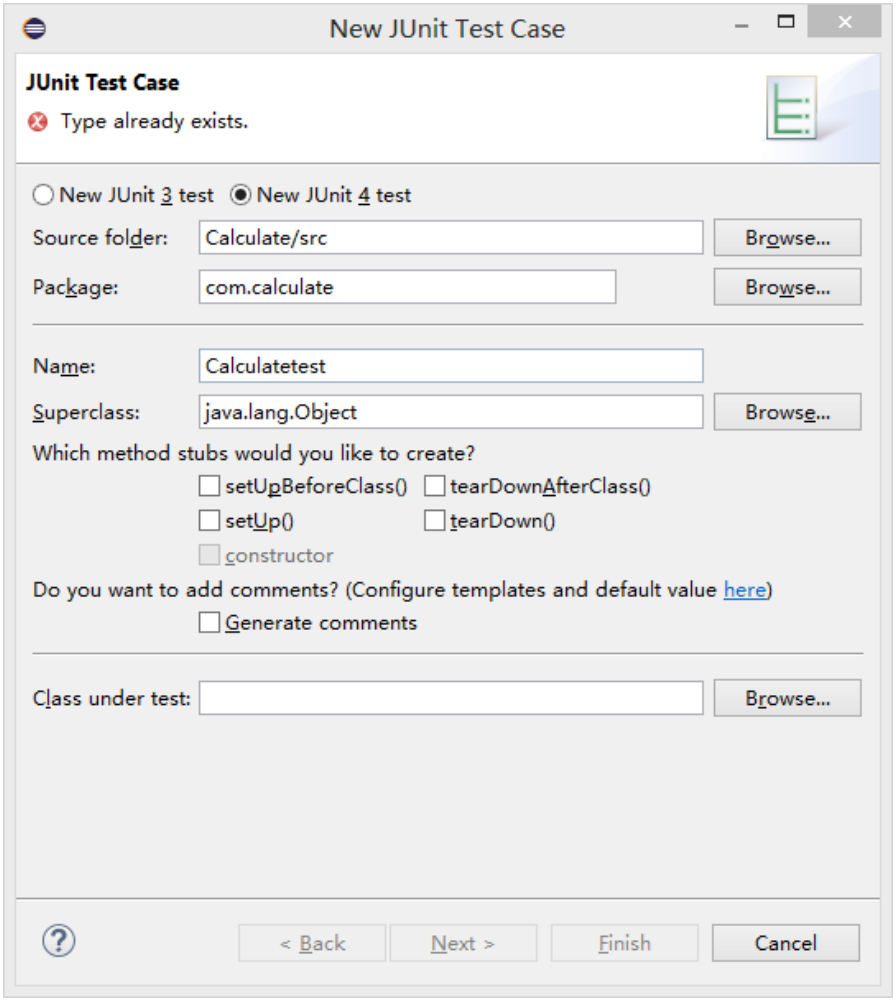
3、实验内容

用 java 语言编写一个计算器类，求实现加、减、乘、除、求平方根、求绝对值、求倒数 1/x, 方法，并用 JUnit 进行对象类的单元测试。参阅帮助文档。（说明，设计求除法、求倒数的方法，可在方法中不检测 x 是否为 0, 测试用例用 y/0 去测试、求平方根可不检测 x>0, 用负数测试），联系使用 JUnit 来对此程序测试。

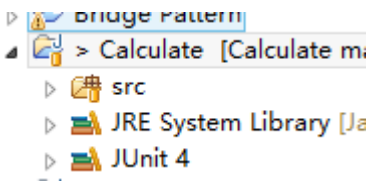
实验步骤：

1、eclipse 创建 Java project，并建立相应的包、类（calculate.java）

2、在 package explorer 中右键 new JUnit Test Case,关联相应的工程



3、选择 Junit4 则会自动引入相应的包



4、编写 calculate 类中的加减乘除函数

```
package com.test.junit3;
public class Calculator {
        public static void main(String[] agrs)
        {
        }
        public int add(int a,int b)
        {
                return a + b;
        }
        public int minus(int a,int b)
        {
                return a - b;
        }
        public int multi(int a,int b)
        {
                return a * b;
        }
        public int divd(int a,int b) throws Exception
        {
                if(b==0)
                {
                        throw new Exception("除数不能为0!");
                }
                return a - b;//为了演示效果，此处构造错误的除法算法
        }
}
```

5、主函数 main

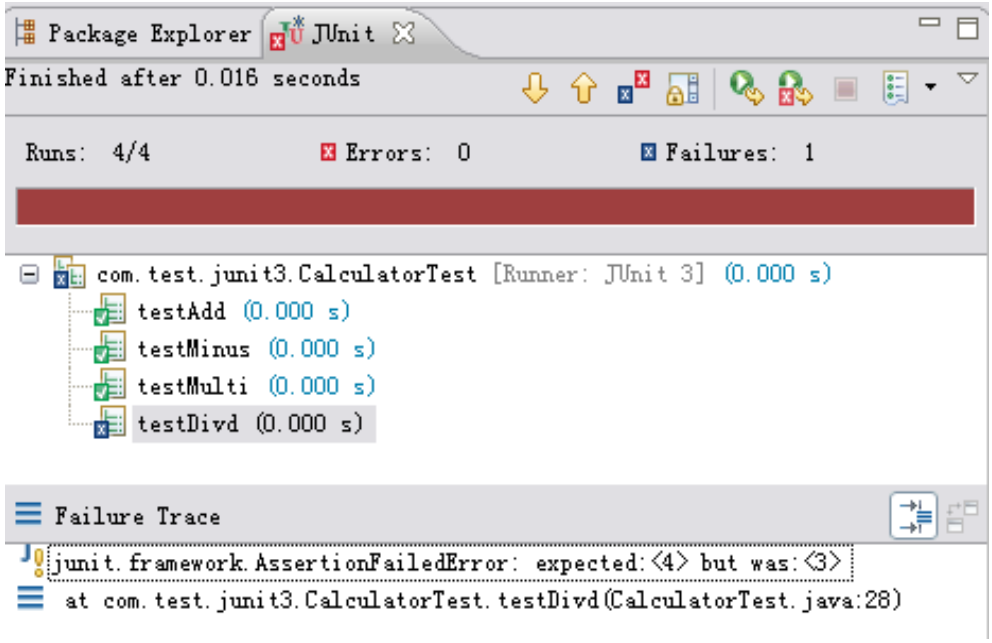
```
package com.calculate;


public class test {
    public static void main(String[] args)
    {
        Calculate test=new Calculate();
```

```
        System.out.println("加法运算："+test.add(12, 13));
        System.out.println("减法运算："+test.subtract(18, 2));
        System.out.println("乘法运算："+test.multiply(2, 6));
        System.out.println("除法运算："+test.divide(4, 4));
    }
}
```

6、编写测试用例

```
package com.test.junit4;
import junit.framework.Assert;
import junit.framework.TestCase;
publicclass CalculatorTest extends TestCase{
        publicvoid testAdd()
        {
                Calculator cal=newCalculator();
                int result=cal.add(1, 2);
                Assert.assertEquals(3, result);
        }
        publicvoid testMinus()
        {
                Calculator cal=newCalculator();
                int result=cal.minus(1, 2);
                Assert.assertEquals(-1, result);
        }
        publicvoidtestMulti()
        {
                Calculator cal=newCalculator();
                int result=cal.multi(1, 2);
                Assert.assertEquals(2, result);
        }
        publicvoid testDivd() throws Exception
        {
                Calculator cal=newCalculator();
                int result=cal.divd(4, 1);
                Assert.assertEquals(4, result);
        }
}
```

7、右键工程->Run as->JUnit Test,如果有错误则滚动条会出现红色，否则为绿色。



从上述案例可以看出，除法功能测试失败，期望结果是 4，但实际结果为 3，查看代码，原因是除法功能代码错误，“a/b”错误写成“a-b”。执行单元测试时，仅关注每个函数或类单元的输入输出，根据预期结果与实际结果的对比，判断被测对象的正确与否。

扩展训练：请在 calculate 类中的添加平方和开方函数，并分别用数据 4，0，-7 进行单元测试。

三、实验总结

通过 JUnit 测试实验，熟悉了 JUnit 测试过程，.....

实验二 JUnit 测试框架的使用（二）（2 学时）

一、实验目的：

通过本次实验，掌握 Junit 测试框架的使用，会编写测试驱动程序，运行测试用例对 Java 程序测试。

二、实验内容：

假设某高校规定在校学生创新学分的成绩评定规则如表 10-1 所示。要求编写 JAVA 代码实现如下功能：输入高校在校学生创新学分的单项最高分和总分，将按规则输出该生的创新学分成绩（优秀、良好、中等、及格、不及格）。

表 10-1 创新学分成绩评定规则

单项最高分	总分	成绩
大于等于 5 分		优秀
	大于等于 7 分	优秀
大于等于 4 分		良好
	大于等于 6 分	良好
大于等于 3 分	大于等于 5 分	中等
	大于等于 4 分	及格
	小于 4 分	不及格

参考程序代码如图 10 所示。程序代码包括两个 Java 类：Chuang 类和 Snippet 类。Chuang 类用于实现创新学分成绩的计算，Snippet 类实现根据输入的创新学分输出创新学分成绩。

程序运行结果如图 10 所示。假设为该程序写了 100 个白盒测试用例，执行这些测试用例时则需要运行 100 次程序。这种完全手工的测试方法是非常费时费力的。

Snippet.java | Chuang.java

```java
public class Chuang {
    double single_Max; //单项最高分
    double sum; //总分
    public double getSignle_Max() {return single_Max;} // Getter
    public void setSignle_Max(double singlemax) {this.single_Max=singlemax;}// Setter
    public double getSum() {return sum;}
    public void setSum(double sum) {this.sum=sum;}
    public Chuang (double sm, double s)//参数初始化
    {
        single_Max=sm;      sum=s;
    }
    public String getGrade() {   //根据单项最高分和总分判断成绩
        String result="";
        if (single_Max>=5 || sum>=7)
            result = "优秀";
        else if (single_Max>=4 || sum>=6)
            result = "良好";
        else if ((single_Max>=3 && sum>=4) || sum>=5)
            result = "中等";
        else if (sum>=4)
            result = "及格";
        else
            result = "不及格";
        return result;
    }
}
```

Snippet.java | *Chuang.java

```java
package snippet;

import java.util.Scanner;

public class Snippet {
    @SuppressWarnings("resource")
    public static void main(String[] args)
    {
        //方案1
        Scanner reader= new Scanner(System.in);
        double sm=0.0; double s=0.0;
        System.out.println("请输入单项最高分和总分，以等号结束;");
        while (reader.hasNextDouble()) {
            sm=reader.nextDouble();
            s=reader.nextDouble();
        }
        Chuang testObj= new Chuang(sm,s);

        String result =testObj.getGrade();
        String output="单项最高分"+sm+",总分"+s+",成绩是"+result;
        System.out.println(output);
    }
}
```

Problems | Javadoc | Declaration | Console | Progress

```
<terminated> Snippet [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (2022年6月11日
请输入单项最高分和总分，以等号结束;
6 8
=
单项最高分6.0,总分8.0,成绩是优秀
```

1、 在 Main 函数中编写测试脚本实现测试

为了提高测试效率，可以编写测试脚本实现：自动获取输入和校验实际输出，并自动判断测试结果；如果出现 Bug，自动记录。参考程序代码如图 10 所示。

*Snippet.java Chuang.java

```
18          String result ="";
19          String expected="";
20          String output="";
21          //测试用例1
22          Chuang testObj= new Chuang(6.0,8.0);
23          result =testObj.getGrade();
24          expected="优秀"; //预期执行结果；
25          if (result==expected) {  //校验执行结果；
26              output="pass";
27          }
28          else
29              output="Fail, 单项最高分 6,总分8,成绩应该是"+expected;
30
31          System.out.println(output);
32
33          //测试用例2
34          testObj= new Chuang(4.0,6.0);
35          result =testObj.getGrade();
36          expected="良好"; //预期执行结果；
37          if (result==expected) {  //校验执行结果；
38              output="pass";
39          }
40          else
41              output="Fail, 单项最高分4,总分6,成绩应该是"+expected;
```

这样的测试实现可以节省部分手工测试工作，但是仍然存在如下问题：

1、测试脚本在 Main 函数中，测试代码和产品代码混合在一起。但是在提交产品代码时，并不需要测试代码。

2、不满足一次编写，多次执行的目的。

3、存在大量重复代码。

4、测试过程记录不完整。

2、 在 Test 类中编写测试脚本实现测试

为了克服上述测试方案中的问题，从以下几个方面做改进：

1、把测试代码独立出来，放在 test 文件夹中；

2、主动记录执行过程和测试结果。

参考程序代码如图所示。

```
public class Test {
    Chuang chuangObj; //
    public void createTestObj(double sm, double s) {chuangObj = new Chuang(sm, s);}
    public void freeTestObj() { chuangObj = null; }
    public boolean verify(String expected, String actual) {
    public String record(double sm, double s, String expected, String actual, boolean testResult) {
        String output = "";
        if(testResult){
            output += "Pass. 单项最高分:" + sm + ", 总分:" + s;
        }else{
            output += "Fail. 单项最高分:" + sm + ", 总分:" + s +
                    ", Expected:" + expected + ", Actual:" +  actual;
        }
        return output;
    }
    public void testGetGrade1() {
        createTestObj(5.0,8.0);
        String actual = chuangObj.getGrade();
        boolean testResult= verify("优秀",actual);
        System.out.println(record(5.0,8.0,"优秀",actual,testResult));
        freeTestObj();
    }
    public static void main(String[] args) {
        Test test= new Test();
        test.testGetGrade1();
    }
}
```

这样的测试实现虽然可以自动运行测试用例，自动校验测试结果，自动报告缺陷，自动记录测试过程，自动统计测试情况，但还存在如下问题：

1、测试逻辑复杂。

2、测试代码不灵活，不能很好地适应测试设计的变化。

3、测试代码重用度不高。

3、 使用 JUnit 工具进行简单测试

使用 JUnit 工具进行测试的步骤很简单，一般分为以下四个步骤：

（1）写测试类

一般一个类对应一个测试类，如 Tool.java 和 ToolTest.java。测试类与被测试类最好是放到同一个包中，测试类的名字为被测试类的名字加 Test 后缀。

（2）写测试方法。

一般一个方法对应一个单元测试方法。测试方法的名字为 test 前缀加被测试方法的名字，如 testAddPerson()。单元测试方法上面要加上@Test 注解

（org.junit.Test）！

单元测试方法不能有参数，也不能有返回值（返回 void）！测试的方法不能是静态的方法。

（3）运行测试。

选中方法名 --> 右键 --> Run '测试方法名' 运行选中的测试方法；选中测试类类名 --> 右键 --> Run '测试类类名' 运行测试类中所有测试方法；选中模块名 --> 右键 --> Run 'All Tests' 运行模块中的所有测试类的所有测试方法。

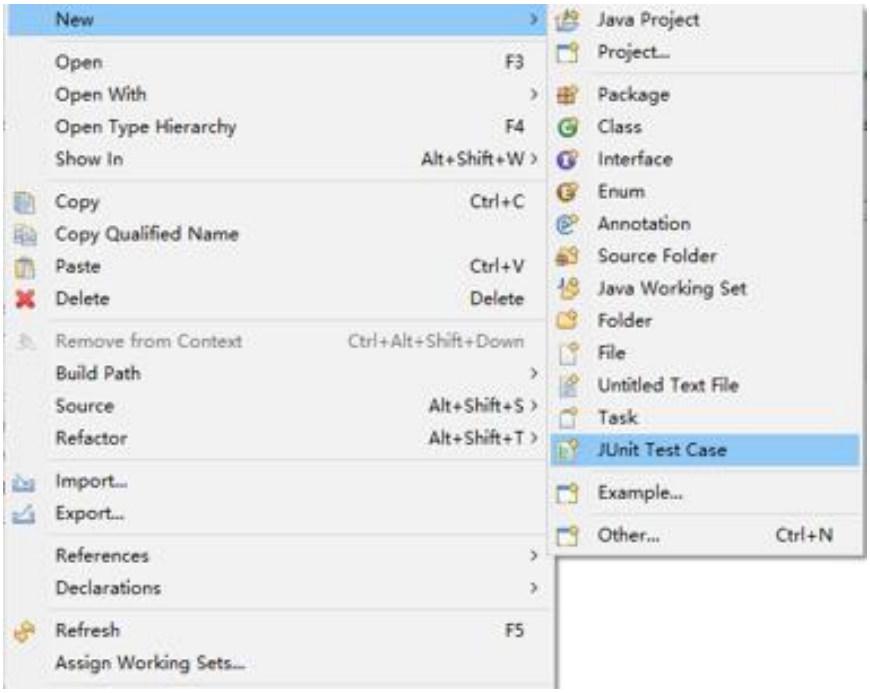
（4）执行测试。

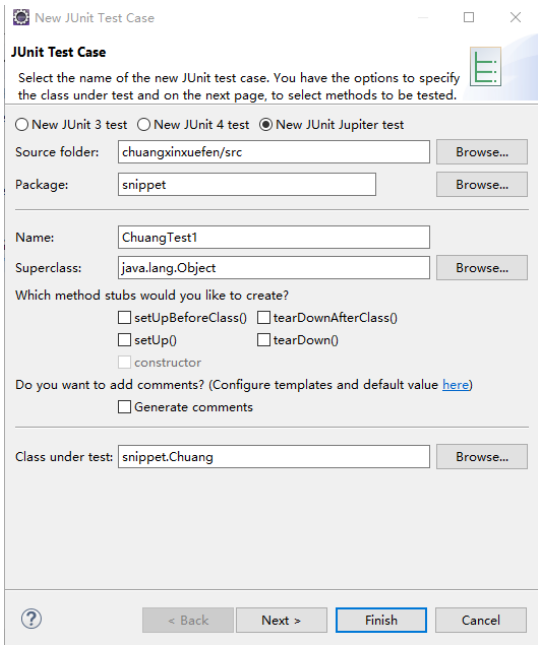
测试那个方法，加上@Test 注解，就把光标放在那个方法里，run as junit test；注意一定要在测试完毕后删除；因为这个测试注解可能会影响正常的代码执行。

在上述例子中，我们执行下面的步骤进行简单地单元测试：

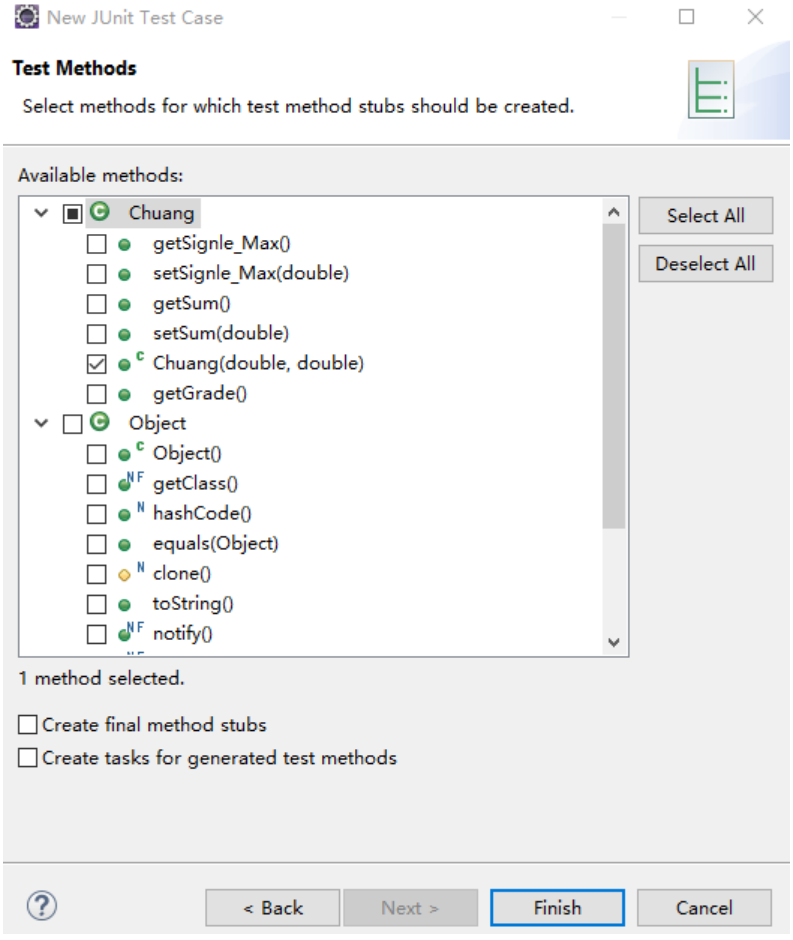
（1）

选中被测试类 Chuang，在右键菜单中选择“New”-》“Junit Test Case”，如图所示；





（2）点击“Next”按钮，在出现的窗口中选中函数“Chuang(double, double)”作为被测试对象，如图所示；



（3）点击“Finish”按钮，出现如图所示代码窗口；

Snippet.java | Chuang.java | Test.java | ChuangTest.java | ChuangTest1.java

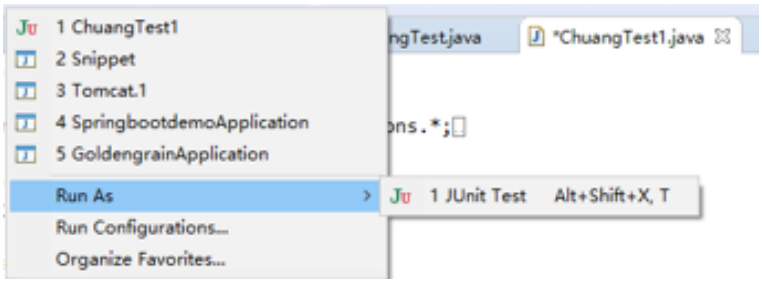
```
1  package snippet;
2
3  import static org.junit.jupiter.api.Assertions.*;
6
7  class ChuangTest1 {
8
9      @Test
10     void testChuang() {
11         fail("Not yet implemented");
12     }
13
14 }
15
```

（4）编写简单的测试代码，结果如图所示；

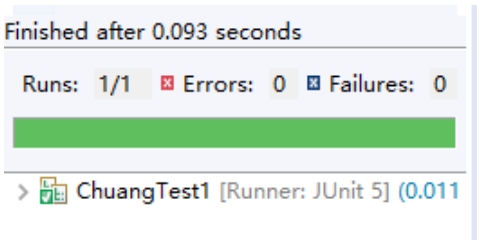
Snippet.java | Chuang.java | Test.java | ChuangTest.java | *ChuangTest1.java

```
1  package snippet;
2
3  import static org.junit.jupiter.api.Assertions.*;
6
7  class ChuangTest1 {
8      Chuang testObj; //被测类
9      @Test
10     void testChuang() {
11         testObj = new Chuang (5.0,8.0);
12         String expected = "优秀";
13         assertTrue(testObj.getGrade()==expected);
14         //fail("Not yet implemented");
15     }
16
17 }
18
```

（5）选择如图所示的执行菜单选项；



（6）结果如图所示，绿色代表测试通过。



思考：上述例子中仅实现了“优秀”成绩的测试，如何进一步实现“良好”、“中等”、“及格”、“不及格”成绩的测试？

三、实验总结

通过 JUnit 测试实验，熟悉了 JUnit 测试过程，.....

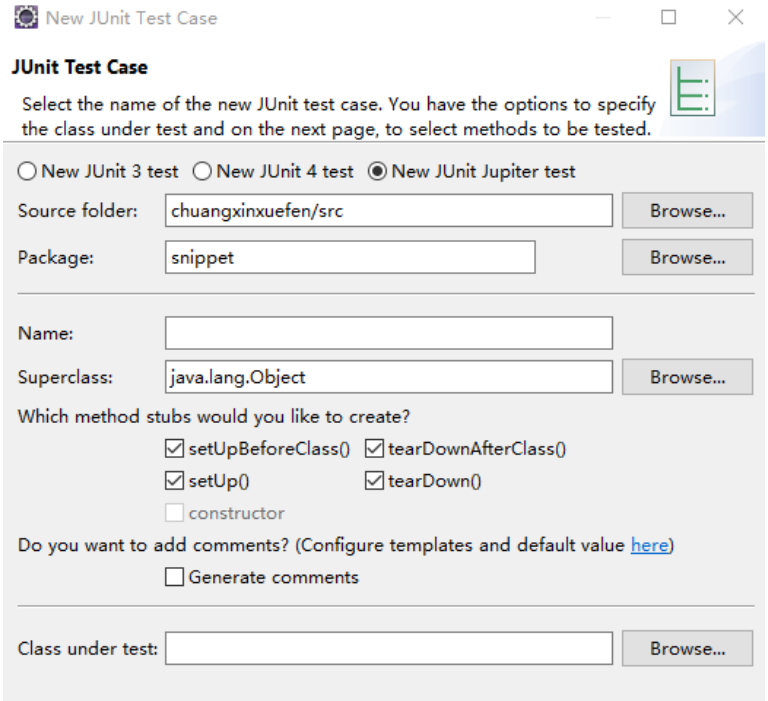
实验三 JUnit 测试框架的使用（三）（2 学时）

一、实验目的：

通过本次实验，掌握 Junit 测试框架@Before 和@After 注解的应用，会编写测试驱动程序，运行测试用例对 Java 程序测试。

二、实验内容：

上一小节例子中，要完成对所有创新学分成绩的测试，需要进一步补充代码。下图所示窗口右边是一种代码补充方案，但从窗口左边的运行情况可以看出，testChuangPass()方法的测试失败了。这是因为尽管几个测试方法在代码窗口中有前后顺序，但在 JUnit 中的执行顺序却是随机的。观察代码可以看出，被测类对象 testObj 只在 testChuangGreat()方法中被创建，在其他方法中都是通过修改对象 testObj 的属性值来实现不同数据的测试。如果 testChuangGreat()方法不是第一个被运行方法，那么将会出现对象 testObj 没有被创建就被调用的错误，就是下图所示的结果。



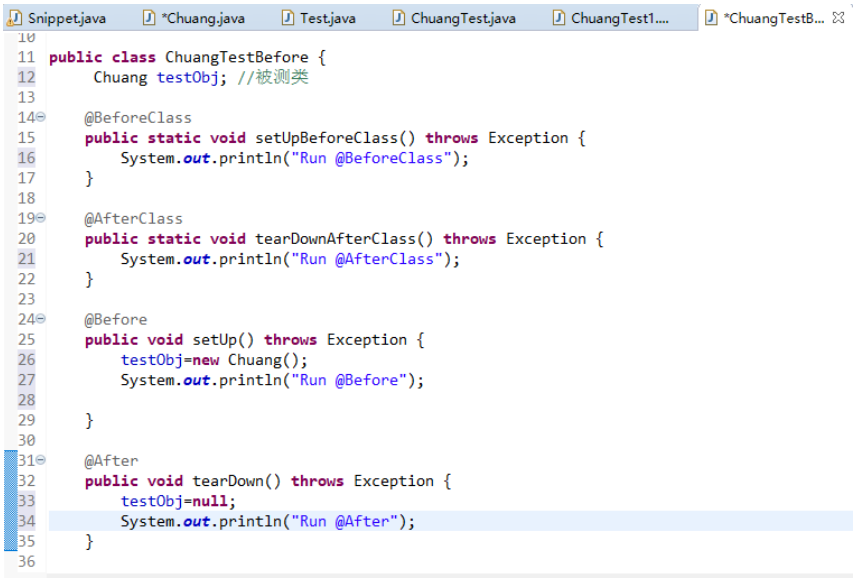
为了解决这类问题，JUnit 提供了@Before 和@After 注解。其中@Before 注解在每个测试用例执行之前执行一次，@After 注解在每个测试用例执行之后执行一次。这为测试前创建类对象和测试后释放类对象提供了方便。除此之外，JUnit 还提供了@BeforeClass 和@AfterClass 注解，分别在测试类的所有测试用例

执行前和执行后执行。也就是说，在测试用例执行时，首先执行@BeforeClass 所注解的方法，接下来，针对每一个测试用例，先执行由@Before 所注解的方法，接着执行由@Test 所注解的测试方法，再执行由@After 所注解的方法，在所有测试用例执行完成后，最后执行@AfterClass 所注解的方法。

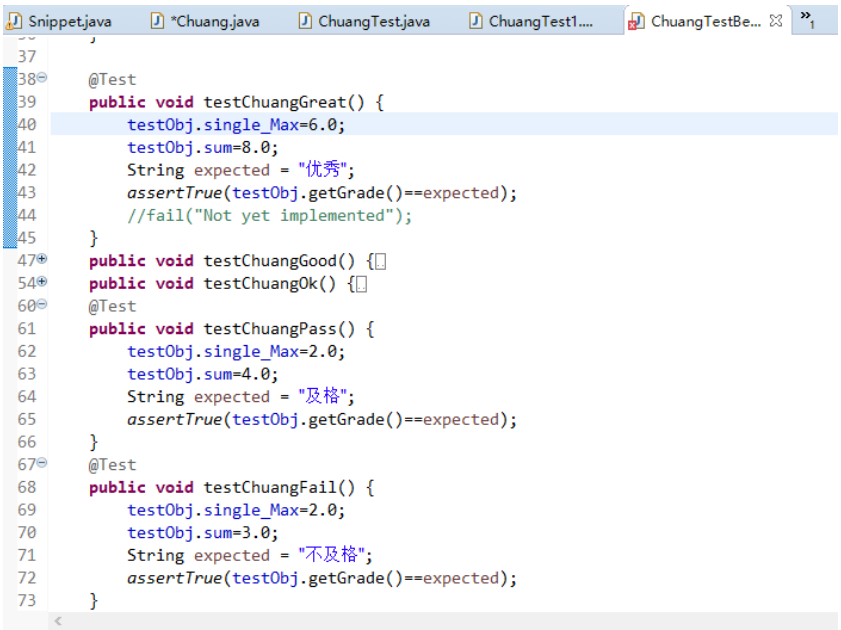
接下来，我们使用@Before 和@After 注解来实现上述测试。

1、新建测试类 ChuangTestBefore，并在出现的创建窗口中选择@Before 和@After、@BeforeClass 和@AfterClass，如上图所示；

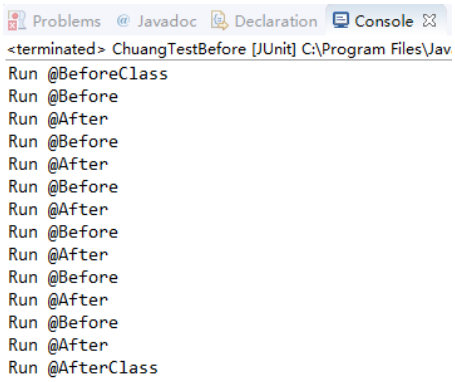
2、**点击确定**，产生如图所示窗口；



3、在带@Before 和@After、@BeforeClass 和@AfterClass 注解的方法中添加代码，如图所示；



4、执行程序，显示测试通过，输出结果如图所示。



三、实验总结

通过 JUnit 测试实验，熟悉了 JUnit 测试过程，.....

实验四 JUnit 测试框架的使用（四）（2 学时）

一、实验目的：

通过本次实验，掌握 Junit 测试框架参数化运行器的应用，会编写测试驱动程序，运行测试用例对 Java 程序测试。

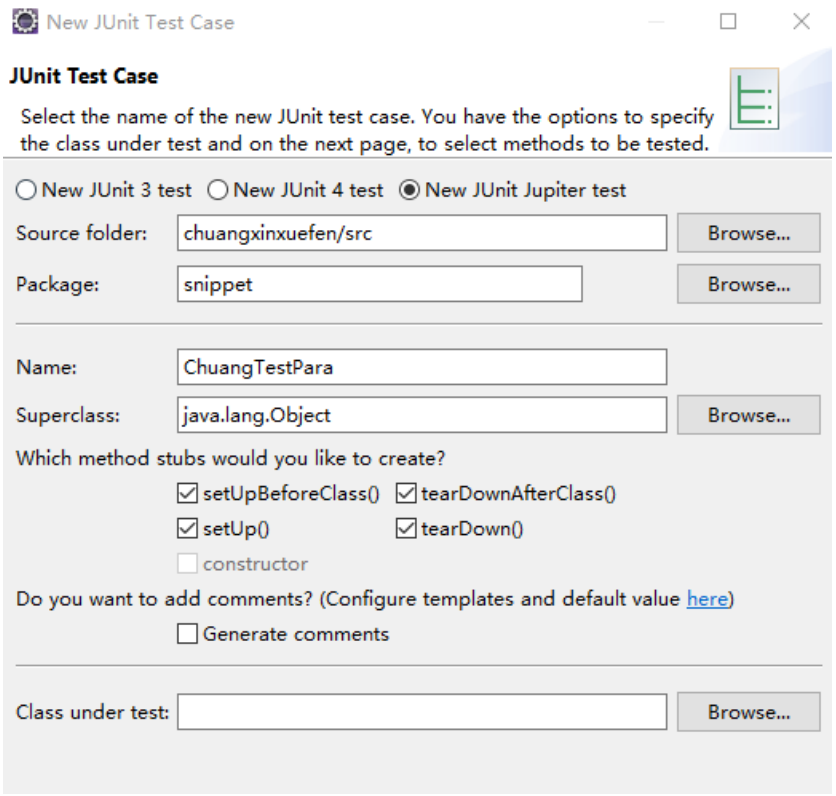
二、实验内容：

实验三中的 5 个测试用例方法，发现代码有很大的相似性。能不能进一步精简这些测试用例方法呢？

JUnit 提供的参数化运行器可以很好地解决这个问题。参数化运行器提供了两种测试数据准备方法：构造器注入和属性注入。构造器注入通过带参数的构造函数获取数据集，属性注入通过属性指定获取数据集。

1、构造器注入

首先，新建测试类 ChuangTestPara，并在出现的创建窗口中选择@Before 和@After、@BeforeClass 和@AfterClass，如下图所示。



接下来，导入 import org.junit.runners.Parameterized 包，引入注解

@RunWith(Parameterized.class)指定参数化运行器；为测试用例输入和预期输出定义私有属性，并编写构造函数实现属性的初始化；引入注解@Parameterized.Parameters，并编写方法实现测试数据集的准备，具体代码如图所示。

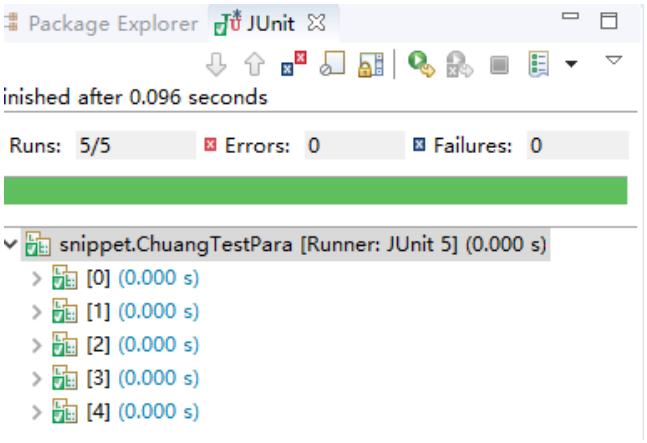
```
14 import org.junit.runners.Parameterized;
15 //1.指定参数化运行器
16 @RunWith(Parameterized.class)
17 public class ChuangTestPara {
18      Chuang testObj; //被测类
19     //定义属性
20     private double singlemax;
21     private double sum;
22     private String expected;
23     //定义构造方法
24 public ChuangTestPara(double sm, double sum, String exp) {
25     this.singlemax=sm;
26     this.sum=sum;
27     this.expected=exp;
28 }
29 //2.准备测试数据
30 @Parameterized.Parameters
31 public static Collection testDataset() {
32
33     return Arrays.asList(new Object[][] {
34         {6.0,8.0,"优秀"},
35         {4.0,6.0,"良好"},
36         {3.0,5.0,"中等"},
37         {2.0,4.0,"及格"},
38         {2.0,3.0,"不及格"}
39     });
40 }
```

接着，在@Before 和@After 注解的方法中添加代码，编写测试用例执行方法，如图所示。

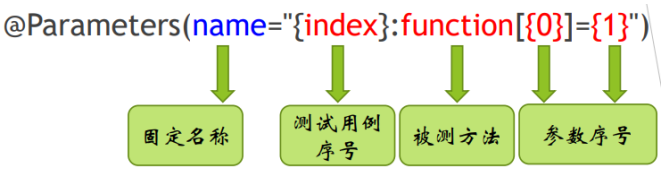
```
@Before
public void setUp() throws Exception {
    testObj=new Chuang(singlemax, sum);
}

@After
public void tearDown() throws Exception {
    testObj=null;
}

@Test
public void test() {
    assertTrue(testObj.getGrade()==expected);
    //fail("Not yet implemented");
}
```

需要注意的是，与实验三中例子不同，此例中@Before 注解方法调用的是带参的构造函数，测试用例方法只包含一行断言语句即可实现不同测试用例的测试。

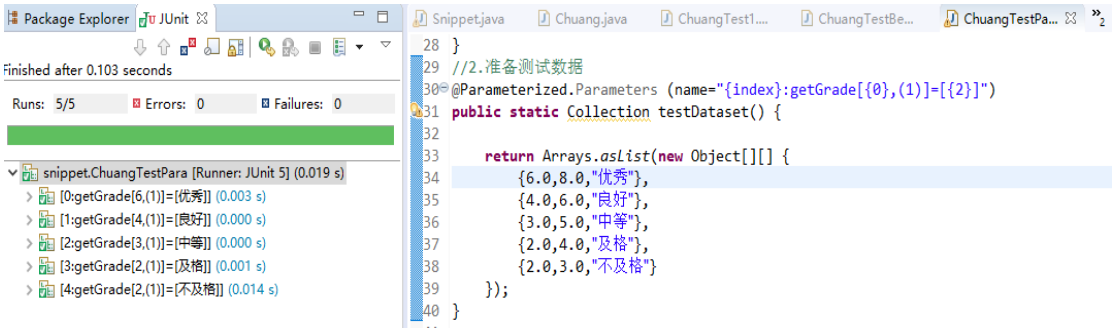
运行结果如图所示，可以看出共执行了 5 个测试用例。



为了提高脚本的可读性，在准备测试数据集时，修改注解“@Parameterized.Parameters”为“@Parameterized.Parameters (name="{index}:getGrade[{0},(1)]=[{2}]")”，含义如图所示。



再次运行测试程序，结果如图所示，可读性得到了极大提高。



三、实验总结

通过 JUnit 测试实验，熟悉了 JUnit 测试过程，.....

实验五 JUnit 测试框架的使用（白盒测试）（2 学时）

一、实验目的：

通过本次实验，掌握白盒测试用例设计方法，并在 Junit 测试框架运行测试用例完成对 Java 程序测试。

二、实验内容：

程序有三个输入变量 month、day、year（month、day 和 year 均为整数值，并且满足：1≤month≤12、1≤day≤31、1900≤ year ≤2050。），分别作为输入日期的月份、日、年份，通过程序可以输出该输入日期在日历上下一天的日期。例如，输入为 2005 年 11 月 29 日，则该程序的输出为 2005 年 11 月 30 日。

主要程序代码如下：

```
public String getNextDay(int year, int month, int day)
 {
      //闰年、平年每月的天数
      int[][] days={{31,28,31,30,31,30,31,31,30,31,30,31},
                          {31,29,31,30,31,30,31,31,30,31,30,31}};
     //index=0:平年，index=1:闰年
     int index = 0;
     if(year%4==0 && year400!=0 || year@100==0)
           index = 1;
     //判断输入日期是否正确
     if(month<1 || month>12 || day<1 || day>days[index][month-1])
          return "输入日期不正确";
     //小于当前月份的天数
      if(day < days[index][month-1])
          day++;
     else
     {
           if(month==12)
          {
                    year++;
```

```
                month=1;
            }
            else
            {
                    month++;
                day=1;
            }
        }
        return year + "-" + month + "-" + day;
}
```

要求：

a.针对上面 getNextDay 函数，使用**路径覆盖**设计测试用例。

b.将程序补充完整，并用设计的测试用例在 Junit 测试框架下测试程序，记录测试结果。

三、实验总结

通过白盒测试实验，学习白盒测试中的判定/条件覆盖、路径覆盖测试方法.....

实验六 JUnit 测试框架的使用（功能测试）（2 学时）

一、实验目的：

通过本次实验，掌握功能测试用例设计方法，并在 Junit 测试框架运行测试用例完成对 Java 程序测试。

二、实验内容：

实验三中的 getNextDay 函数说明如下：有三个输入变量 month、day、year（month、day 和 year 均为整数值，并且满足：1≤month≤12、1≤day≤31、1900≤ year ≤2050。），分别作为输入日期的月份、日、年份，通过程序可以输出该输入日期在日历上下一天的日期。例如，输入为 2005 年 11 月 29 日，则该程序的输出为 2005 年 11 月 30 日。

根据上述函数情况说明，结合等价类划分、判定表为此函数编写测试用例，并使用这些测试用例测试实验五中的实现的 getNextDay 函数，编写测试报告。

三、实验总结

通过功能测试实验，学习功能测试中的等价类划分、判定表等测试方法.....