实验 3 JUnit 单元测试

四川大学软件学院 杨秋辉 yangqiuhui@scu.edu.cn

1 前言

本实验中，学生将熟悉如何在 eclipse 中使用 JUnit 建立一个 JUnit 测试项目来进行单元测试，并基于 javadoc 中描述的需求来创建单元测试用例。在这一部分中学生要根据各方法的需求描述创建单元测试用例，在完成测试用例编写以后，运行这些测试用例，并对测试结果进行收集和分析。

本次实验用 1 次实验课（2 学时）完成。

1.1 实验目的

本实验的主要目的是练习单元测试基本技术。JUnit 是目前最为广泛的应用于 Java 的单元测试工具。本实验要求学生学会使用 JUnit 来完成简单的单元测试用例编写。

1.2 测试工具简介

在本实验中使用的主要测试工具是 JUnit。JUnit 是一个广受欢迎的针对 java 代码的开源单元测试框架。JUnit 是 xUnit 家族的测试框架，由 Kent Beck 和 Erich Gamma 开发。它可以无缝集成在 Eclipse 开发环境中。JUnit 框架允许开发人员快速、轻松地创建单元测试和测试套件。要想了解更多关于 JUnit 的相关信息,可以查阅 http://www.junit.org。

另一个工具是 Javadoc。虽然 Javadoc 不是一个完全意义上的测试工具，在本实验中它将被用作存储需求规格说明的格式。Javadoc 允许开发者创建应用程序编程接口(API)文档，使源代码和执行代码一起整合在文档中。创建的文档和代码在同一个位置不仅能够提高开发人员、维护人员和测试人员之间的交流，并且可以使文档更新更加简单，同时也能够防止潜在的冗余和/或错误。要想了解更多关于 Javadoc 的信息,见 http://java.sun.com/j2se/javadoc。

1.3 被测系统

该实验的被测系统是 JFreeChart。JFreeChart 是一个开源的、基于 Java 的绘图框架，主要功能是图的计算、创建和显示。该框架支持多种（图形）图表类型，包括饼状图、柱状图、折线图、和其他图表类型。注意，本实验中所使用的 JFreeChart 与实际的版本是有区别的。为了更好地适应实验要求，我们对 JFreeChart 进行了相应的修改。

JfreeChart 的主要目的是提供一些接口，让 Java 程序开发者可以方便地在自己的系统中添加绘图功能。JfreeChart 提供的 API 接口非常简单。下图是使用 JfreeChart 绘制的 4 种典型的图。

虽然 JfreeChart 不是一个可以单独使用的绘图系统，但开发人员还是创建了几个可以执行的 demo，以展示其功能。这些类的名字以 Demo 结尾，后面的实验中会运行这些类。

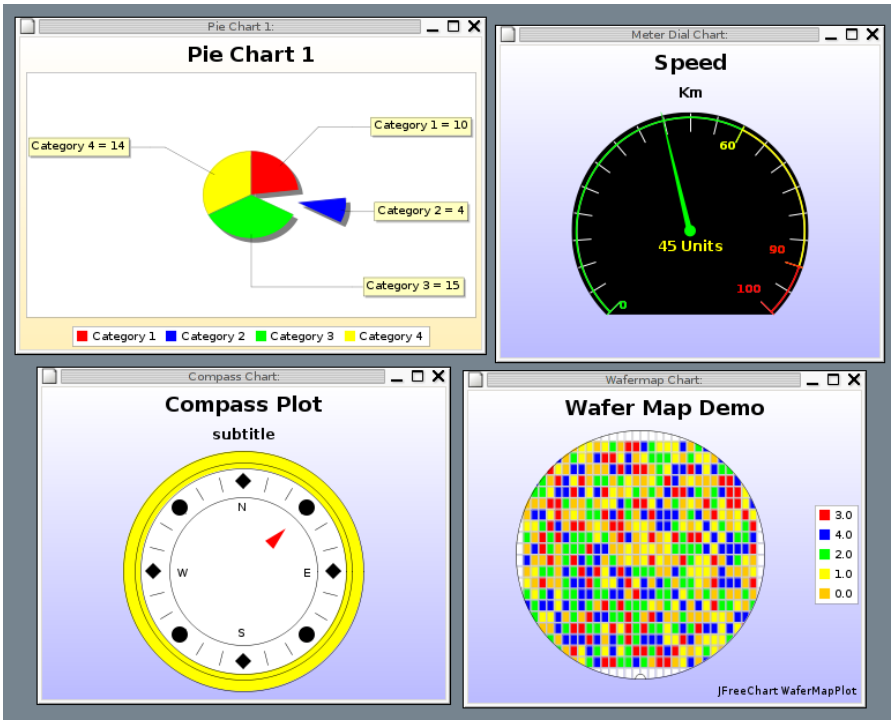

图 1 – 使用 JFreeChart 绘制的 4 种不同类型的图

2 熟悉工具

本小节的目的是进一步熟悉 JUnit 的使用，下面将详细说明如何进行测试用例的构建。

2.1 熟悉阶段

1. 在教师提供的资料中找到“JFreeChart v1.0.zip”。

2. 将.zip 文件中的内容解压到自己创建的目录下。

2.1.1 创建一个eclipse项目

3. 打开 eclipse（3.3 或以上版本），新建一个 Java Project，取名为 JFreeChart。注意：在新建项目的最后一步的时候，在 Java Settings 对话框顶部有 4 个选项卡：Source、Projects、Libraries 和 Order and Export。选择 Libraries 选项卡，然后点击 Add External JARs…按钮。

4. 在上述 JFreeChart v1.0.zip 文件解压的目录下选择添加 jfreechart.jar 文件。再一次点击 Add External JARs…，将 JFreeChart v1.0.zip 解压后的 lib 文件夹下的所有.jar 文件添加进去。这时 Java Settings 对话框的状态应该如下图所示。

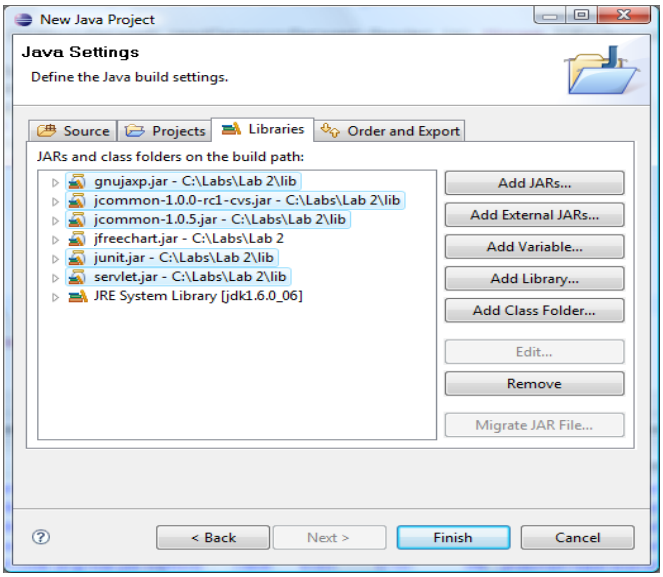

图2 加入所需文件后的Java Settings对话框

5. 点击 finish。这时被测试系统的项目就建立了，为测试做好了准备。下面就是运行示例类。在 package explorer 下找到新创建的 JFreeChart 项目。其下有一个 Referenced Libraries 目录，在目录下找到 JFreeChart.jar，在其上点击鼠标右键，并选择 Run As -> Java Application (如下图 3 所示)。

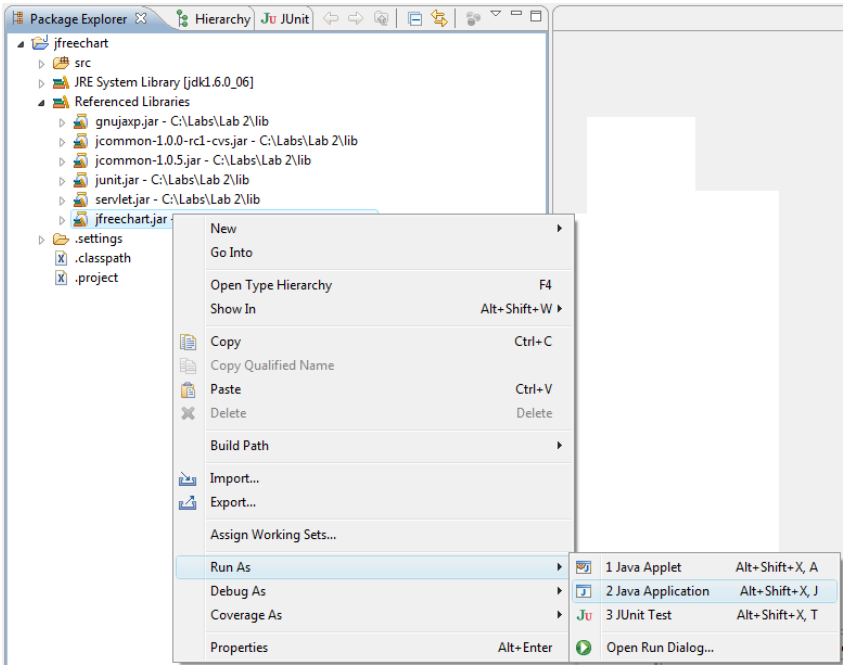

图3 运行JFreeChart

6. 在 Select Java Application 对话框中选择 4 个 Demo 程序中的任意一个后点击运行（例如选择 CompositeDemo），如下图。可以看到 JFreeChart 能够绘制的几种典型的图的展示。

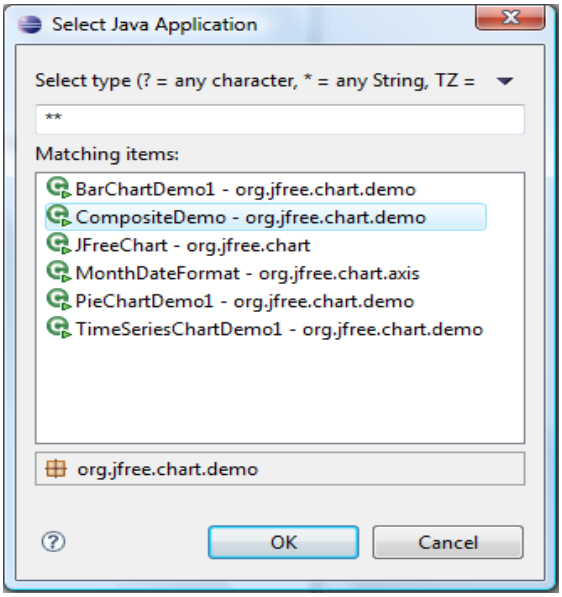

图4 Select Java Application对话框

2.1.2 一个简单的JUnit 单元测试

使用 JUnit 创建一个简单的测试套，其中只包含一个单元测试用例，其步骤如下：

7. 在 package explorer 中，展开 Referenced Libraries 目录，会看到项目相关的所有文件。

8. 展开 JFreeChart.jar，将显示其中包含的所有包。

9. 展开该文件夹下的 org.jfree.data 包，将看到该包下的所有.class 文件。

10. 最后展开 Range.class，显示其中包含的类和类中的方法。

11. 右击 Range 类（有一个绿色的 C 标记，表明这是一个类），选择 New -> JUnit Test Case。在 Package 域中填写 org.jfree.data.test，以便为测试用例创建一个新的包；Name 域中填写 RangeTest。Superclass 域中空着。在需要创建的桩函数（Which method studs would you like to create?）栏中选中 setUp()、tearDown()、constructor()。

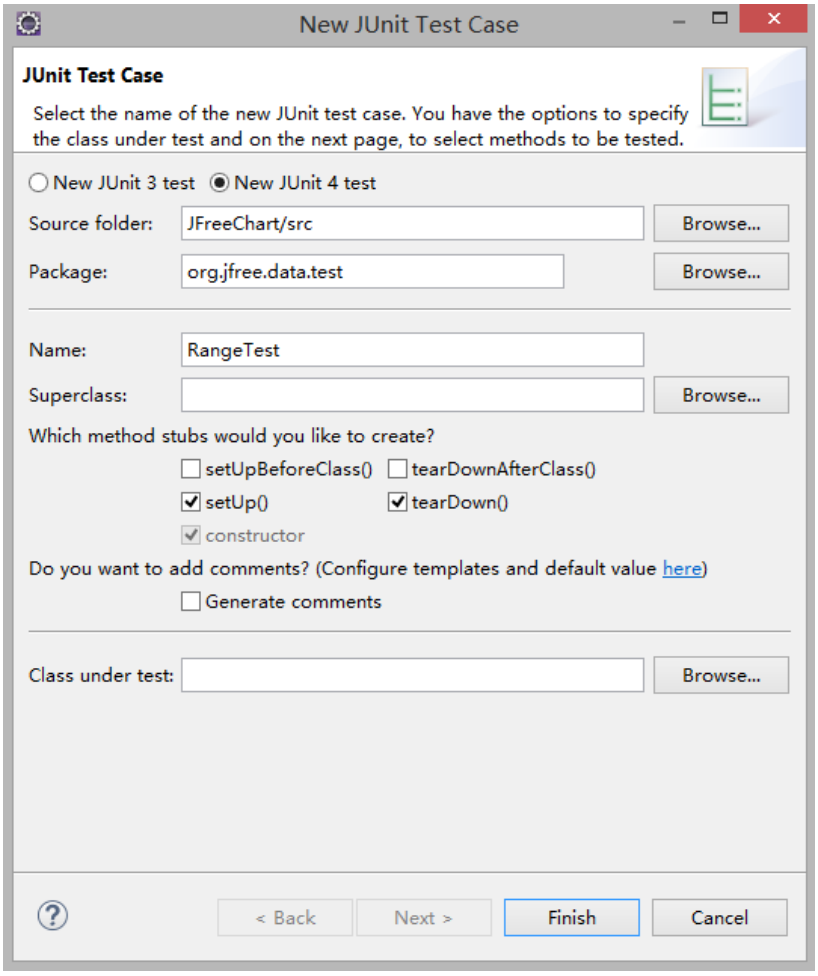

图5 创建一个新的JUnit测试用例

12. 点击 Finish。在 Eclipse 编辑窗口中会出现系统自动生成的 JUnit 单元测试代码。

RangeTest.java

```
1  package org.jfree.data.test;
2
3  import static org.junit.Assert.*;
8
9  public class RangeTest {
10
11     public RangeTest() {
12     }
13
14     @Before
15     public void setUp() throws Exception {
16     }
17
18     @After
19     public void tearDown() throws Exception {
20     }
21
22     @Test
23     public void test() {
24         fail("Not yet implemented");
25     }
26
27 }
```

图6 系统自动生成的 Junit 代码

13. 新创建的测试类（RangeTest）继承于 TestCase，TestCase 类是在 JUnit 中定义的所有测试用例的超类。TestCase 类为测试用例和测试套提供了统一的接口。

当需要测试某个类中相似的一组方法时，相应的测试用例有一些共性，即需要在测试之前和之后有一些共同的工作：初始化这个类的实例、以及另外一些为测试需要的准备工作、测试完成后删除创建的实例等等。在 JUnit 中用@Before：setUp()和@After：tearDown()方法实现这些共同的工作。

系统自动生成的 setUp()和 tearDown()方法不能满足测试需求，将 RangeTest 类中的 setUp()和 tearDown()方法改写，如下图 7 所示。

```
private Range testRange;

public RangeTest() {
}
@Before
public void setUp() throws Exception {
    testRange = new Range(-1,1);
}

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

图7 简单的 setUP（）和 tearDown（）函数

上面的 setUp()方法建立一个新的实例，用于后面的测试；tearDown()方法删除创建的实例。

```
注：关于 JUnit 中的注解
  @BeforeClass : 针对所有测试，只执行一次，且必须为 static void
  @Before : 初始化方法
  @Test : 测试方法，在这里可以测试期望异常和超时时间
  @After : 释放资源
  @AfterClass : 针对所有测试，只执行一次，且必须为 static void
  @Ignore : 忽略的测试方法
一个单元测试类执行顺序为：
  @BeforeClass -> @Before -> @Test -> @After -> @AfterClass
    每一个测试方法的调用顺序为：
    @Before -> @Test -> @After
```

14. JUnit 中的每个测试用例都是独立的方法，前面通常都需要加上“test”前缀。例如，testCentralValue()。这些测试用例可以执行任意数量的步骤，但都应该遵循 4 个测试阶段（setup，exercise，verify and teardown）。在 JUnit 中，测试用例的结果判断可以使用多种不同的方法实现，断言（Assertions）是应用最广泛的一种方法。

一个断言的例子：

assertTrue((“example string”).length() == 14);

这条语句判断字符串“example string”的长度是否等于 14，很明显，这条语句将总是能够执行成功（如果 length（）方法是正确的）。上面的断言的语法在 JUnit 中的定义是：

public static void assertTrue([java.lang.String message,] boolean condition)

该方法用于判断其后的条件是否为真。如果不是，就抛出 AssertionFailedError 异常。当一个测试用例被执行的时候，其中的断言也会被执行。如果其后面跟的条件不为真，则导致测试执行失败（即不符合预期结果）。附录中列举了 JUnit 中可以使用的所有断言。

15. 一个简单的练习：

为 Range 类中的 getCentralValue()方法编写一个简单的测试用例。assertEquals 的语法格式如下：

public static void assertEquals(java.lang.String message, double expected, double actual, double delta)

assertEquals 用于判断两个值是否相等。其中的 delta 参数，是为了在浮点数比较时允许存在误差。如果两个值不相等，则会抛出异常 AssertionFailedError。

下图 8 是相应的测试用例：

```
private Range testRange;

public RangeTest() {
}
@Before
public void setUp() throws Exception {
    testRange = new Range(-1,1);
}

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


@Test
public void testCentraValue() {
    assertEquals("The central value of -1 and 1 should be 0",
            0, testRange.getCentralValue(), 0.0000001d);
}
```

图8 增加了测试用例后的RangeTest类

16. 现在我们已经完成了一个测试用例，下面运行测试用例。在 package explorer 中右击 RangeTest 类，然后选择 Run As -> JUnit Test。

17. 视图将转换到 JUnit 视图，并运行 Rangetest 类中的所有测试，如下图所示。刚刚编写的测试应该能够成功执行。

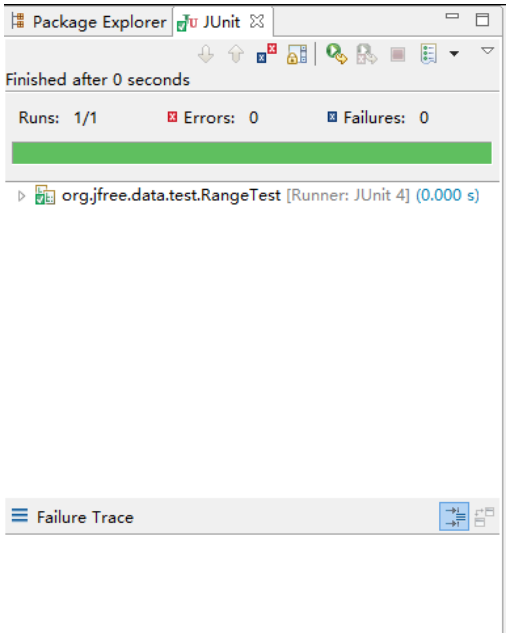

图9 测试执行后的JUnit视图

在 JUnit 中，failure 和 error 很相似，但又有细微的差别。Error 意味着你所测试的方法没有像预期的那样执行（例如产生了一个没有被捕获的异常）。而 failure 是指被测试的方法按照预期执行了，但是断言失败了。

2.1.3 参数化测试

你可能遇到过这样的函数，它的参数有许多特殊值，或者说它的参数分为很多个区域。比如，一个对考试分数进行评价的函数，返回值分别为“优秀，良好，一般，及格，不及格”，因此你在编写测试的时候，至少要写 5 个测试，把这 5 中情况都包含了。考虑测试以下计算器类的“求平方”函数，我们将输入暂且分三类：正数、0、负数。

被测试类：

```
3  public class Calculator {
4
5      private int result;
6
7      public void clear() {
8          result = 0;
9      }
10     public void square(int r) {
11         result = r * r;
12     }
13
14     public int getResult() {
15         return result;
16     }
17 }
```

图10 待测试类源码

对三类输入进行测试的测试代码如下：

```
10 public class SquareTest {
11
12     private static Calculator calculator = new Calculator();
13
14     @Before
15     public void clearCalculator() {
16         calculator.clear();
17     }
18     @Test
19     public void testSquare1() {
20         calculator.square(2);
21         assertEquals(4, calculator.getResult());
22     }
23     @Test
24     public void testSquare2() {
25         calculator.square(0);
26         assertEquals(0, calculator.getResult());
27     }
28     @Test
29     public void testSquare3() {
30         calculator.square(-3);
31         assertEquals(9, calculator.getResult());
32     }
```

图11 未使用参数化测试的测试代码

为了简化类似的测试，JUnit4 提出了“参数化测试”的概念，只写一个测试函数，把这若干种情况作为参数传递进去，一次性的完成测试。

使用参数化测试的测试代码：

```
15 @RunWith(Parameterized.class)
16 public class ParameterizedSquareTest {
17
18     private static Calculator calculator = new Calculator();
19     private int param;
20     private int result;
21
22     @Parameters
23     public static Collection data() {
24         return Arrays.asList(new Object[][]{
25             {2, 4},
26             {0, 0},
27             {-3, 9},
28             {1,1}
29         });
30     }
31     // 使用构造函数对变量进行初始化
32     public ParameterizedSquareTest(int param, int result) {
33         this.param = param;
34         this.result = result;
35     }
36     @Test
37     public void testSquare() {
38         calculator.square(param);
39         assertEquals(result, calculator.getResult());
40     }
41
42 }
```

图12 使用参数化测试的测试代码

创建一个参数化测试一般需要如下步骤：

首先，为这种测试专门生成一个新的类，并且为这个类指定一个 Runner，而不能使用默认的 Runner 了，因为特殊的功能要用特殊的 Runner。

第二步，定义一个待测试的类实例，并且定义两个变量，一个用于存放参数，一个用于存放期待的结果。接下来，定义测试数据的集合，也就是上述的 data()方法，必须使用@Parameters 标注进行修饰。这个方法的框架是一个二维数组，每组中的这两个数据，一个是参数，一个是预期的结果。

之后是构造函数，其功能就是对先前定义的两个参数进行初始化。

最后就是写一个简单的测试例了，和前面介绍过的写法完全一样，在此就不多说。

2.1.4 使用Javadoc查看被测系统的需求说明

下面的 2.2 节中要求你根据被测系统的需求编写单元测试用例，这些需求描述在 Javadoc 中可以找到。

18. 解压教师所给资料中的 JFreeChart–ModifiedJavadoc.zip 文件，打开 index.html 文件，其中就是被测系统的 Javadoc 描述。附录中描述了 Javadoc 中不同部分显示的内容是什么。

19. 在左上角的 packages 列表中，找到 org.jfree.data 包并点击，在左下角的 class 列表中，将只显示这个包中包含的类。

20. 在类列表中，点击 Range，右侧主窗口中将显示 Range 的 API 说明。最前面是关于类的说明，然后是嵌套的类、属性、方法、继承的方法等，最后是对每个方法的详细说明。

21. 认真阅读其中的 Method Summary 和 Method Detail 中的说明，后面的测试用例主要是根据这些说明来编写。

2.2 创建测试用例

22. 为Range类（org.jfree.data.Range）中的方法编写测试用例，Range类在org.jfree.data包中，其中有15个方法，要求从中选至少**5个**编写测试用例。注意，对于每一个被测方法，需要写多个不同的测试用例，以便测试多种不同输入情况（建议每个方法至少测试3种不同输入情况）。

23. 在 org.jfree.data.function中，分别为LineFunction2D类和PowerFunction2D中的getValue()方法编写测试用例；为NormalDistributionFunction2D类中的getMean(),getStandardDeviation(),getValue()方法编写测试用例。要求为这**5个**方法使用**参数化测试。**

请阅读Javadoc中相关类的说明，以便帮助你正确写出测试用例。

24. 执行你编写的测试用例，并记录测试结果，在实验报告中报告你所发现的缺陷。

3 总结

通过完成这个实验，学生应该对使用 JUnit 框架进行基于单元需求的单元测试有了一定的了解。注意，单元测试和 JUnit 都是非常复杂的，要想精通这些需要花费很长的时间和大量的精力，不要寄希望于完成本实验就能够精通 Junit。如果你想在这一领域有所建树，需要对这一框架进行更加详尽的学习。

4 提交的内容及评分标准

4.1 实验报告 (70%)

在实验报告中按照要求描述你的测试用例，每个方法的测试用例占 6 分。

请根据教师给出的“实验 3-JUnit 单元测试-实验报告模板.doc”撰写实验报告，你可以根据自己的情况，修改其中的部分标题内容。提交实验报告的文件格式为 doc 文件，命名规定：“学号-姓名-JUnit 单元测试-实验报告.doc”。

4.2 测试用例 (30%)

将你的测试用例打包提交。根据测试用例的清晰性（代码格式规范，可读性好）、与需求一致性（只测试需求中描述的功能）、正确性（测试了真正需要测试的属性）格式规范性、测试完整性（测试了需求中描述的所有内容）进行评分。

附录 A：JUNIT 中的断言（ASSERTIONS）

- assertEquals(expected, actual)
- assertEquals(message, expected, actual)
- assertEquals(expected, actual, delta) - used on doubles or floats, where delta is the allowable difference in precision
- assertEquals(message, expected, actual, delta) - used on doubles or floats, wheredelta is the allowable difference in precision
- assertFalse(condition)
- assertFalse(message, condition)
- assertNotNull(object)
- assertNotNull(message, object)
- assertNotSame(expected, actual)
- assertNotSame(message, expected, actual)
- assertNull(object)
- assertNull(message, object)
- assertSame(expected, actual)
- assertSame(message, expected, actual)
- assertTrue(condition)
- assertTrue(message, condition)
- assertThat(message, actual, expected)
- fail() - Fails a test with no message.
- fail(message) - Fails a test with the given message.
- failNotEquals(message, expected, actual)
- failNotSame(message, expected, actual)
- failSame(message)

附录 B：JAVADOC 布局