一、实验目的

1. 验证所学理论，巩固所学知识并加深理解；
2. 培养学生分析单元模块，设计测试数据的的能力；
3. 熟悉白盒测试、黑盒测试用例的设计；
4. 熟悉使用 Junit 框架进行基于 java 语言的单元测试.

二、实验项目内容

1. 使用白盒测试用例设计方法为下面的程序设计测试用例：

• 程序要求：10 个铅球中有一个假球（比其他铅球的重量要轻），用天平三次称出假球。

• 程序设计思路：第一次使用天平分别称 5 个球，判断轻的一边有假球；拿出轻的 5 个球，取出其中 4 个第二次称，两边分别放 2 个球：如果两边同重，则剩下的球为假球；若两边不同重，拿出轻的两个球称第三次，轻的为假球。

2. 使用等价类划分法设计下面的测试用例：

输入三个整数作为边，分别满足一般三角形、等腰三角形和等边三角形。

三、实验过程或算法（源程序）

1. 用白盒测试对判断假球设计测试用例

1.1 构建 *Ball* 待测试类

```
 * 程序要求：10个铅球中有一个假球（比其他铅球的重量要轻），用天平三次称出假球。
 */
2 个用法
public class Ball {
    //定义weights来存储每个球的重量
    3 个用法
    private final int[] weights = new int[10];

    1 个用法
    public Ball() { }

    1 个用法
    public void setWeight(int[] a){
        System.arraycopy(a, 0, weights, 0, weights.length);
    }
    18 个用法
    private int sumOf(int left, int right){
        int sum = 0;
        for(int i = left - 1; i < right; i++){
            sum += weights[i];
        }
        return sum;
    }
```

• 在 *Ball* 类中，设置 weights 数组用于记录十个铅球的重量；

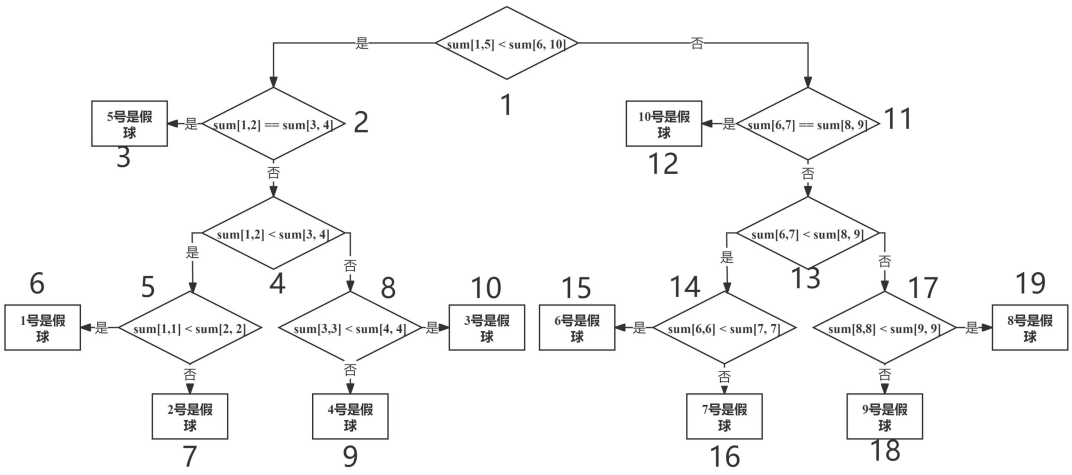
• *setWeight( )*函数用于每次重置测试用例中给定的铅球重量；

• *sumOf(int left, int right)*用于计算 weights 中从[left-1, right-1]连续编号的铅球重量之和。

```
/**
 * * 1. 第一次使用天平分别称5个球，判断轻的一边有假球；
 * * 2. 拿出轻的5个球，取出其中4个第二次称，两边分别放2个球：如果两边同重，则剩下的球为假球；
 * * 3. 若两边不同重，拿出轻的两个球称第三次，轻的为假球。
 * @return 出错球的编号1-10
 */
1 个用法
public int findFalseBall(){
    if(sumOf(1, 5) < sumOf(6, 10)) {
        if(sumOf(1, 2) == sumOf(3, 4))     return 5;
        else if(sumOf(1, 2) < sumOf(3, 4)) return (sumOf(1, 1) < sumOf(2, 2))? 1 : 2;
        else                               return (sumOf(3, 3) < sumOf(4, 4))? 3 : 4;
    }
    else {
        if(sumOf(6, 7) == sumOf(8, 9))     return 10;
        else if(sumOf(6, 7) < sumOf(8, 9)) return (sumOf(6, 6) < sumOf(7, 7))? 6 : 7;
        else                               return (sumOf(8, 8) < sumOf(9, 9))? 8 : 9;
    }
}
```

• *findFalseBall( )*用于该类对象逐步判断假球的位置，根据实验内容中的逻辑，可得以上程序设计，最终返回假球的编号(1-10).

1.2 构建测试流程图和设计测试用例



根据题目要求，得到以上流图，并设计测试用例：

测试用例 weights[ ]	预期结果(假球编号)	覆盖路径
[1, 2, 2, 2, 2, 2, 2, 2, 2, 2]	1	1-2-4-5-6
[2, 1, 2, 2, 2, 2, 2, 2, 2, 2]	2	1-2-4-5-7
[2, 2, 1, 2, 2, 2, 2, 2, 2, 2]	3	1-2-4-8-10
[2, 2, 2, 1, 2, 2, 2, 2, 2, 2]	4	1-2-4-8-9

[2, 2, 2, 2, 1, 2, 2, 2, 2, 2]	5	1-2-3
[2, 2, 2, 2, 2, 1, 2, 2, 2, 2]	6	1-11-13-14-15
[2, 2, 2, 2, 2, 2, 1, 2, 2, 2]	7	1-11-13-14-16
[2, 2, 2, 2, 2, 2, 2, 1, 2, 2]	8	1-11-13-17-19
[2, 2, 2, 2, 2, 2, 2, 2, 1, 2]	9	1-11-13-17-18
[2, 2, 2, 2, 2, 2, 2, 2, 2, 1]	10	1-11-12

1.3 构建 *BallTest* 白盒测试类

```
public class BallTest extends TestCase{
    4 个用法
    int []weight= new int[10];
    1 个用法
    public void initWeight(int i){
        Arrays.fill(weight, 2);
        weight[i] = 1;
        System.out.print("测试用例为: ");
        for (int k : weight) {
            System.out.print(k);
            System.out.print(" ");
        }
    }
```

- weight[]用于记录每个铅球的重量，*initWeight( )*用于初始化编号 1-10 ($i \in [0,9]$) 铅球的重量，并打印出每个测试用例中各个铅球对应的重量，假球的重量恒设置为 1，其余均设置为 2.

```
public void testFindFalseBall(){
    //测试10组
    final int rounds = 10;
    Ball ball = new Ball();
    for (int i = 0; i < rounds; i++){
        initWeight(i);
        ball.setWeight(weight);
        assertEquals("\n测试用例不通过", i + 1, ball.findFalseBall());
        System.out.println("PASS");
    }
}

public static void main(String[] args) {
    TestRunner.run(BallTest.class);
}
```

- *testFindFalseBall( )*用于构建测试用例，因有十个铅球，故设置测试用例个数 rounds 为 10；初始化重量后，使用 *assertEquals( )*判断结果是否与

预期一致，若一致则输出“PASS”，反之则触发中断“测试用例不通过”；最后使用 *testRunner.run( )*运行测试。

2. 用等价类划分法判断三角形类型

2.1 构建 *Triangle* 待测试类

```
/**
 * 使用等价类划分法设计下面的测试用例：
 * 输入三个整数作为边，分别满足一般三角形、等腰三角形和等边三角形。
 */
11 个用法
public class Triangle {
    8 个用法
    private static int a, b, c;
    //实例化参数
    11 个用法
    public Triangle(int a1, int b1, int c1){
        a = a1;
        b = b1;
        c = c1;
    }
```

- 在 *Triangle* 类中，设置 a,b,c 用于记录三角形的三边长度；
- Triangle*( )*构造函数用于初始化 a,b,c；

```
public String isTriangle(){
    if(a > 0 && b > 0 && c > 0){
        if(a + b > c && b + c > a && a + c > b){
            if((a-b)*(b-c)*(a-c)!=0)    return "一般三角形";
            else if(b == c && a == c)   return "等边三角形";
            else                        return "等腰三角形";
        }
        else return "非三角形";
    }
    else return "非三角形";
}
```

- *Triangle( )*函数用于判断是否为三角形以及三角形的类型：若有两边之和小于或等于第三边或某条边长度为负数，则返回“非三角形”。在满足为三角形条件下，若三条边两两之差的乘积不为零，则返回“一般三角形”；若三条边均相等，则返回“等边三角形”；否则，则返回“等腰三角形”。

2.2 设计测试用例

(1) 划分等价类

基于以上需求，设计以下等价类：

测试项	有效等价类	无效等价类
三角形的三边	$a>0$ (1) $b>0$ (2) $c>0$ (3) $a+b>c$ (4) $a+c>b$ (5) $b+c>a$ (6)	$a\le 0$ (11) $b\le 0$ (12) $c\le 0$ (13) $a+b\le c$ (14) $a+c\le b$ (15) $b+c\le a$ (16)
等腰三角形	$a=b$，$a!=c$ (7) $b=c$，$a!=b$ (8) $a=c$，$a!=b$ (9)	$(a-b)(b-c)(a-c)!=0$ (17)
等边三角形	$a=b=c$ (10)	$a!=b \| b!=c \| a!=c$ (18)

(2) 设计具体用例

a	*b*	*c*	覆盖等价类	预期输出
3	4	5	(1)(2)(3)(4)(5)(6) (17)(8)(19)(20)	一般三角形
0	4	5	(11)	非三角形
3	0	5	(12)	
3	4	0	(13)	
2	3	5	(14)	
2	5	3	(15)	
5	2	3	(16)	
3	3	5	(7)	等腰三角形
5	3	3	(8)	
3	5	3	(9)	
3	3	3	(11)	等边三角形

2.3 构建 *TriangleTest* 黑盒测试类

```java
public class TriangleTest extends TestCase{
    //@Test
    public void testIsTriangle(){
        //等价类1,2,3,4,5,6,17,18,19,20
        assertEquals("测试用例1不通过", "一般三角形", new Triangle(3, 4, 5).isTriangle());
        //等价类11
        assertEquals("测试用例2不通过", "非三角形", new Triangle(0, 4, 5).isTriangle());
        //等价类12
        assertEquals("测试用例3不通过", "非三角形", new Triangle(3, 0, 5).isTriangle());
        //等价类13
        assertEquals("测试用例4不通过", "非三角形", new Triangle(3, 4, 0).isTriangle());
        //等价类14
        assertEquals("测试用例5不通过", "非三角形", new Triangle(2, 3, 5).isTriangle());
        //等价类15
        assertEquals("测试用例6不通过", "非三角形", new Triangle(2, 5, 3).isTriangle());
        //等价类16
        assertEquals("测试用例7不通过", "非三角形", new Triangle(5, 2, 3).isTriangle());
        //等价类7
        assertEquals("测试用例8不通过", "等腰三角形", new Triangle(3, 3, 5).isTriangle());
        //等价类8
        assertEquals("测试用例9不通过", "等腰三角形", new Triangle(5, 3, 3).isTriangle());
        //等价类9
        assertEquals("测试用例10不通过", "等腰三角形", new Triangle(3, 5, 3).isTriangle());
        //等价类11
        assertEquals("测试用例11不通过", "等边三角形", new Triangle(3, 3, 3).isTriangle());
    }

    public static void main(String[] args) {
        TestRunner.run(TriangleTest.class);
    }
```

对每个测试用例，直接使用构造函数 *new Triangle( )*将进行构造验证，调用 *TestRunner.run* 运行测试。

四、实验结果及分析和（或）源程序调试过程

1. 用白盒测试对判断假球测试结果分析

1.1 所有结果与预期输出一致

```
D:\Java\jdk1.8.0_191\bin\java.exe ...
.测试用例为: 1 2 2 2 2 2 2 2 2 2 PASS
测试用例为: 2 1 2 2 2 2 2 2 2 2 PASS
测试用例为: 2 2 1 2 2 2 2 2 2 2 PASS
测试用例为: 2 2 2 1 2 2 2 2 2 2 PASS
测试用例为: 2 2 2 2 1 2 2 2 2 2 PASS
测试用例为: 2 2 2 2 2 1 2 2 2 2 PASS
测试用例为: 2 2 2 2 2 2 1 2 2 2 PASS
测试用例为: 2 2 2 2 2 2 2 1 2 2 PASS
测试用例为: 2 2 2 2 2 2 2 2 1 2 PASS
测试用例为: 2 2 2 2 2 2 2 2 2 1 PASS

Time: 0.011

OK (1 test)
```

在保持预期输出与实际输出一致的情况下运行测试可观察到，10 个测试用例均通过了测试，并打印出每个测试用例具体情况，说明程序设计正

确。

1.2 更改某个测试用例与实际不符

```
    if(i == 6) assertEquals("\n测试用例不通过", i, ball.findFalseBall());
    else assertEquals("\n测试用例不通过", i + 1, ball.findFalseBall());
    System.out.println("PASS");
}
```

将第 7 个测试用例更改为与实际不符，如图所示，再运行测试，得到：

```
.测试用例为: 1 2 2 2 2 2 2 2 2 2 PASS
测试用例为: 2 1 2 2 2 2 2 2 2 2 PASS
测试用例为: 2 2 1 2 2 2 2 2 2 2 PASS
测试用例为: 2 2 2 1 2 2 2 2 2 2 PASS
测试用例为: 2 2 2 2 1 2 2 2 2 2 PASS
测试用例为: 2 2 2 2 2 1 2 2 2 2 PASS
测试用例为: 2 2 2 2 2 2 1 2 2 2 F
Time: 0.003
There was 1 failure:
1) testFindFalseBall(org.example.BallTest)junit.framework.AssertionFailedError:
测试用例不通过 expected:<6> but was:<7>
    at org.example.BallTest.testFindFalseBall(BallTest.java:26) <3 个内部行>
    at org.example.BallTest.main(BallTest.java:33)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

可以看到，前 6 个测试用例均通过测试，但第 7 个发生了错误并指出了错误点，同时中断了剩余的测试用例，由此验证了测试用例的合理性和可靠性。

2. 用黑盒测试判断三角形类型结果分析

2.1 所有结果与预期输出一致

```
D:\Java\jdk1.8.0_191\bin\java.exe ...
.
Time: 0.009

OK (1 test)


进程已结束,退出代码0
```

在保持预期输出与实际输出一致的情况下运行测试可观察到，11 个测试用例均通过了测试，说明程序设计正确。

2.2 更改某个测试用例与实际不符

```
    //等价类8
    assertEquals("测试用例9不通过", "等边三角形", new Triangle(5, 3, 3).isTriangle());
    //等价类9
     //等价类11
     assertEquals("测试用例11不通过", "等腰三角形", new Triangle(3, 3, 3).isTriangle());
}
```

将第 9 个测试用例更改为“等边三角形”(实际为等腰三角形)，第 11 个测试用例更改为“等腰三角形”(实际为等边三角形)，其余保持不变，再运行测试：

```
.F
Time: 0.003
There was 1 failure:
1) testIsTriangle(org.example.TriangleTest)junit.framework.ComparisonFailure: 测试用例9不通过 expected:<等[边]三角形> but was:<等[腰]三角形>
    at org.example.TriangleTest.testIsTriangle(TriangleTest.java:27) <3 个内部行>
    at org.example.TriangleTest.main(TriangleTest.java:35)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

可以看到，前 7 个测试用例均通过测试，但第 8 个发生了错误并指出了错误点，给出了预期输出和实际输出的差异点，同时中断了剩余的测试用例，即第 11 个测试用例的错误没有检测到。由此验证了测试用例的合理性和可靠性。