

# 重 庆 交 通 大 学

## 学生实验报告

实验课程名称：《软件测试》

开 课 实 验 室：软件实验室（南岸）

学 院：信息学院

专 业：计算机科学与技术

班 级：2014 级 一 班

学 号：631406010109

学 生 姓 名：郭文浩

指 导 教 师：何 伟

开 课 时 间：2016 至 2017 学 年 第 2 学 期



## 一、实验目的

- 1、掌握结构性测试技术，并能应用结构性测试技术设计测试用例。
- 2、掌握白盒测试的六种逻辑覆盖方法。

## 二、实验内容及原理

### 1、实验内容

题目一：使用逻辑覆盖测试方法测试以下程序段

```
void DoWork (int x,int y,int z)
{
1  int k=0,  j=0;
2  if ( (x>3)&&(z<10) )
3  {
4      k=x*y-1;
5      j=sqrt(k);
6  }
7  if((x==4)|| (y>5))
8      j=x*y+10;
9      j=j%3;
10 }
```

说明：程序段中每行开头的数字（1~10）是对每条语句的编号。

（1）画出程序的控制流图（用题中给出的语句编号表示）。

（2）分别以语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、组合覆盖和路径覆盖方法设计测试用例，并写出每个测试用例的执行路径（用题中给出的语句编号表示）。

题目二：三角形问题

在三角形计算中，要求输入三角型的三个边长：A、B 和 C。当三边不可能构成三角形时提示错误，可构成三角形时计算三角形周长。若是等腰三角形打印“等腰三角形”，若是等边三角形，则提示“等边三角形”。画出程序流程图、控制流程图、计算圈复杂度  $V(g)$ ，找出基本测试路径。

## 2、实验原理

### 原理一：逻辑覆盖

结构性测试力求提高测试覆盖率。逻辑覆盖是对一系列测试过程的总称，它是在使用白盒测试法时，选用测试用例执行程序逻辑路径的方法。

逻辑覆盖按覆盖程度由低到高大致分为以下几类：

(1) 语句覆盖：设计若干测试用例，使程序中每一可执行语句至少执行一次；

(2) 判断覆盖：设计用例，使程序中的每个逻辑判断的取真取假分支至少经历一次；

(3) 条件覆盖：设计用例，使判断中的每个条件的可能取值至少满足一次；

(4) 判断/条件覆盖：设计用例，使得判断中的每个条件的所有可能结果至少出现一次，而且判断本身所有可能结果也至少出现一次；

(5) 条件组合覆盖。设计用例，使得每个判断表达式中条件的各种可能组合都至少出现一次；显然，满足⑤的测试用例也一定是满足②、③、④的测试用例。

(6) 路径覆盖。设计足够的测试用例，使程序的每条可能路径都至少执行一次。

如果把路径覆盖和条件组合覆盖结合起来，可以设计出检错能力更强的测试数据用例。

### 原理二：基本路径测试

如果把覆盖的路径数压缩到一定限度内，例如，程序中的循环体只执行零次和一次，就成为基本路径测试。它是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。

设计出的测试用例要保证在测试中，程序的每一个可执行语句至少要执行一次。

## 三、测试代码、测试方法及测试用例

### 1、测试代码

题目一中已经自带代码，题目二（三角形问题）的代码（自己的）如下：

```

package com.gwh.bhcs;
import java.util.Scanner;
/**
 * 三角形类 用于练习白盒测试
 *
 * @author guowenhao
 * @version 1.0
 */
public class Triangle {
    public static void main(String[] args) {
        // 接收器初始化
        Scanner sc = new Scanner(System.in);
        // 三角形三边初始化
        double edge1 = 0;
        double edge2 = 0;
        double edge3 = 0;
        // 接收三角形三边
        System.out.println("即将录入三角形的边长.....");
        edge1 = receiveEdge(sc, "一");
        edge2 = receiveEdge(sc, "二");
        edge3 = receiveEdge(sc, "三");
        // 判断是否能够成三角形
        if (IsTriangle(edge1, edge2, edge3)) {
            System.out.println("周长为" + (edge1 + edge2 + edge3));
            // 判断能否构成等边三角形
            if (IsEquilateralTriangle(edge1, edge2, edge3)) {
                System.out.println("等边三角形");
            }
            // 判断能否构成等腰三角形
            else if (IsIsoscelesTriangle(edge1, edge2, edge3)) {
                System.out.println("等腰三角形");
            } else {
                //
            }
        } else {
            System.out.println("此三边无法构成三角形！程序结束！");
        }
        sc.close();
    }

    /**
     * 判断是否为等边三角形
     *
     * @param edge1

```

```

*          第一条边的边长
* @param edge2
*          第二条边的边长
* @param edge3
*          第三条边的边长
* @return 是等边三角形返回true 不是等边三角形返回false
*/
private static boolean IsEquilateralTriangle(double edge1, double edge2, double
edge3) {
    if (edge1 == edge2 && edge1 == edge3)
        return true;
    else
        return false;
}

/**
* 判断是否为等腰三角形
*
* @param edge1
*          第一条边的边长
* @param edge2
*          第二条边的边长
* @param edge3
*          第三条边的边长
* @return 是等腰三角形返回true 不是等腰三角形返回false
*/
private static boolean IsIsoscelesTriangle(double edge1, double edge2, double
edge3) {
    if (edge1 == edge2 || edge1 == edge3 || edge2 == edge3)
        return true;
    else
        return false;
}

/**
* 判断输入的边长是否合法，合法返回输入，不合法则重新输入
*
* @param sc
*          接收器
* @param index
*          第几条边
* @return 合法的边长
*/
private static double receiveEdge(Scanner sc, String index) {

```

```

        double edge = 0;
        while (true) {
            try {
                System.out.println("请输入第" + index + "条边：");
                String input = sc.nextLine();
                edge = Double.parseDouble(input);
                if (edge > 0)
                    break;
                else {
                    System.out.println("第" + index + "条边输入有误，边长应为正数，请重新输入！");
                    continue;
                }
            } catch (NumberFormatException e) {
                System.out.println("第" + index + "条边输入有误，边长应为数字，请重新输入！");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return edge;
    }

    /**
     * 通过输入的三边确定是否能构成三角形
     *
     * @param edge1
     *      第一条边的边长
     * @param edge2
     *      第二条边的边长
     * @param edge3
     *      第三条边的边长
     * @return 能构成三角形返回true 不能构成三角形返回false
     */
    private static boolean IsTriangle(double edge1, double edge2, double edge3) {
        if (edge1 < edge2 + edge3 && edge2 < edge1 + edge3 && edge3 < edge1 + edge2
            && edge1 > Math.abs(edge2 - edge3)
            && edge2 > Math.abs(edge1 - edge3) && edge3 > Math.abs(edge1 - edge2))
            return true;
        else
            return false;
    }
}

```

2、测试方法与测试用例：

题目一：

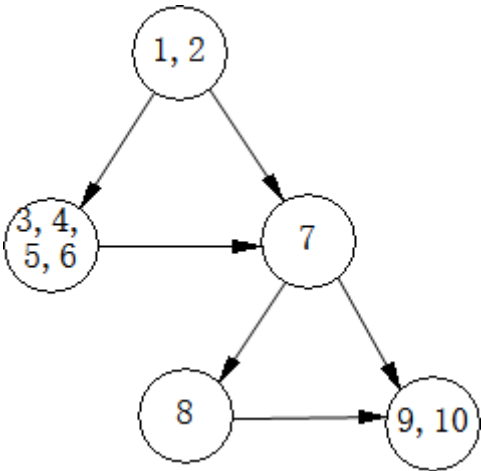


图 1. DoWork 代码段控制流图

覆盖方法	测试用例	执行路径	预测结果	实验结果
语句覆盖	x=4; y=6; z=9;	1-2-3-4-5-6-7-8-9-10	k=23; j=1;	k=23; j=1;
判定覆盖	x=4; y=6; z=9;	1-2-3-4-5-6-7-8-9-10	k=23; j=1;	k=23; j=1;
	x=2; y=3; z=9;	1-2-7-9-10	k=0; j=0;	k=0; j=0;
条件覆盖	x=4; y=6; z=9;	1-2-3-4-5-6-7-8-9-10	k=23; j=1;	k=23; j=1;
	x=2; y=3; z=11;	1-2-7-9-10	k=0; j=0;	k=0; j=0;
判定/条件覆盖	x=4; y=6; z=9;	1-2-3-4-5-6-7-8-9-10	k=23; j=1;	k=23; j=1;
	x=2; y=3; z=11;	1-2-7-9-10	k=0; j=0;	k=0; j=0;
组合覆盖	x=4; y=6; z=9;	1-2-3-4-5-6-7-8-9-10	k=23; j=1;	k=23; j=1;
	x=2; y=3; z=11;	1-2-7-9-10	k=0; j=0;	k=0; j=0;
	x=4; y=3; z=11;	1-2-7-8-9-10	k=0; j=1;	k=0; j=1;
	x=2; y=6; z=9;	1-2-7-8-9-10	k=0; j=1;	k=0; j=1;
路径覆盖	x=4; y=6; z=9;	1-2-3-4-5-6-7-8-9-10	k=23; j=1;	k=23; j=1;
	x=2; y=3; z=11;	1-2-7-9-10	k=0; j=0;	k=0; j=0;
	x=2; y=6; z=9;	1-2-7-8-9-10	k=0; j=1;	k=0; j=1;
	x=5; y=4; z=9;	1-2-3-4-5-6-7-9-10	k=19; j=1;	k=19; j=1;



## 题目二：

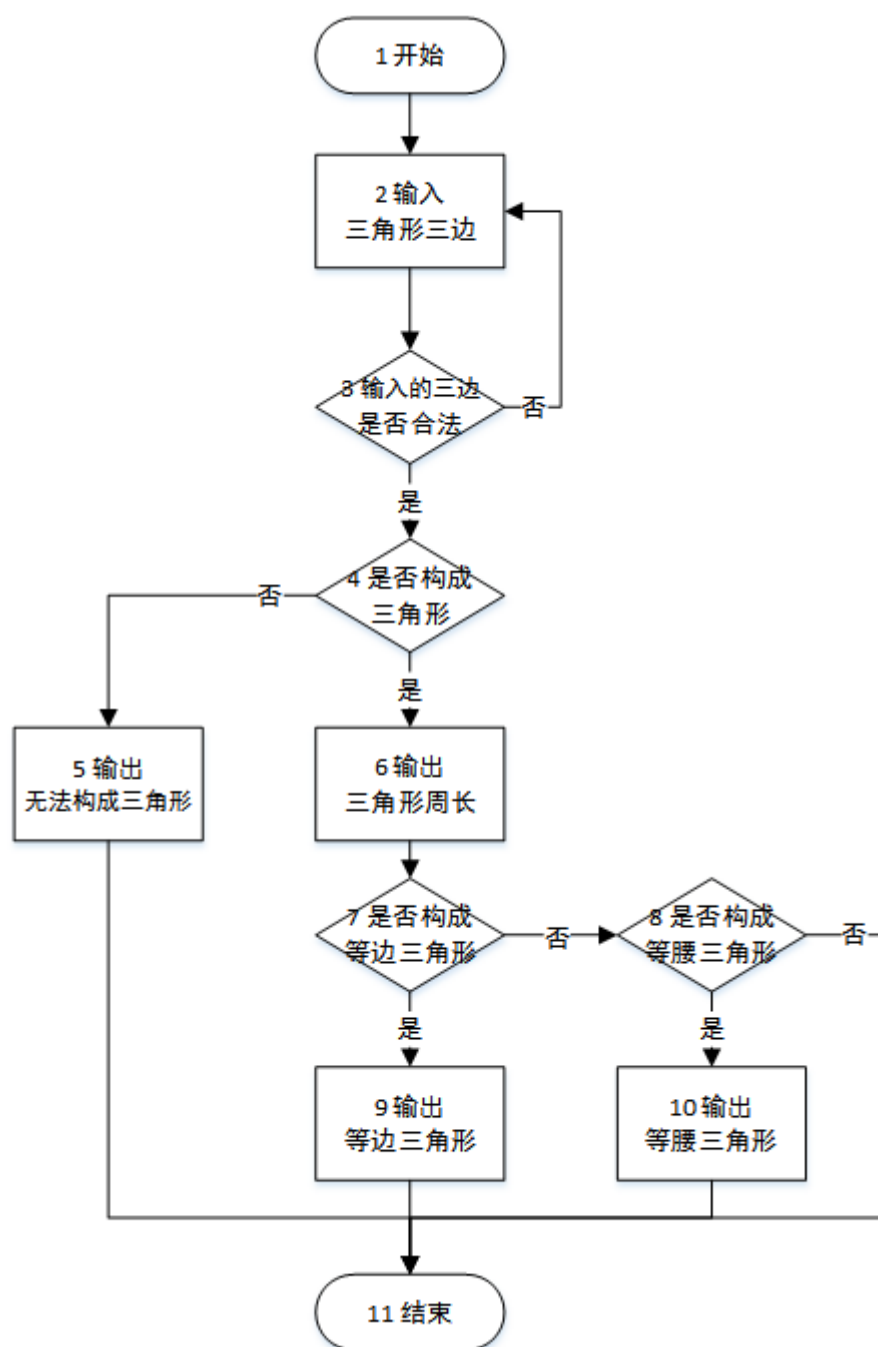


图 2. 三角形问题流程图

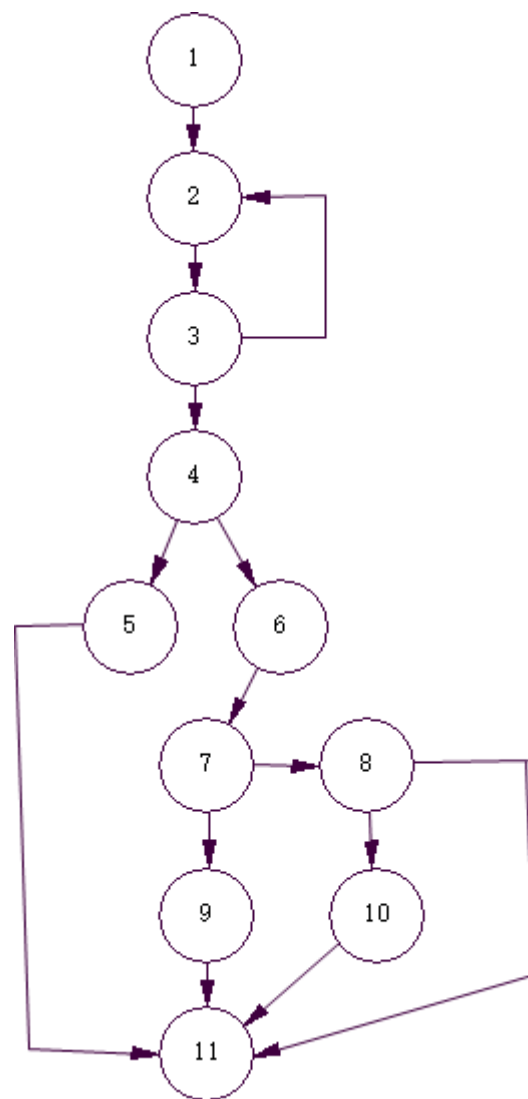


图 3. 三角形问题控制流图

圈复杂度：

$$V(g)=14-11+2=5;$$

基本测试路径：

- path1: 1-2-3-4-5-11
- path2: 1-2-3-4-6-7-9-11
- path3: 1-2-3-4-6-7-8-11
- path4: 1-2-3-4-6-7-8-10-11
- path5: 1-2-3-2-3-4-6-7-8-10-11

测试用例

- edge1=1; edge2=1; edge3=100;
- edge1=6; edge2=6; edge3=6;
- edge1=6; edge2=7; edge3=8;
- edge1=6; edge2=7; edge3=7;
- edge1=a;6; edge2=7; edge3=7;

## 四、发现程序缺陷及修改方案

### 1、程序缺陷

题目一的缺陷比较严重，由于  $k$ 、 $j$  两个参数定义时均为整型，而第 5 行代码用了 `sqrt()` 这个函数，开平方不能保证每次都能刚好除尽，所以必然会产生精度丢失问题。

题目二的缺陷它本身不是个缺陷，但是当用户不断地输入非整型值时他会不断地提醒用户重新输入，我觉得这样会使用户产生反感，用户觉得不好的就是软件缺陷。

### 2、修改方案

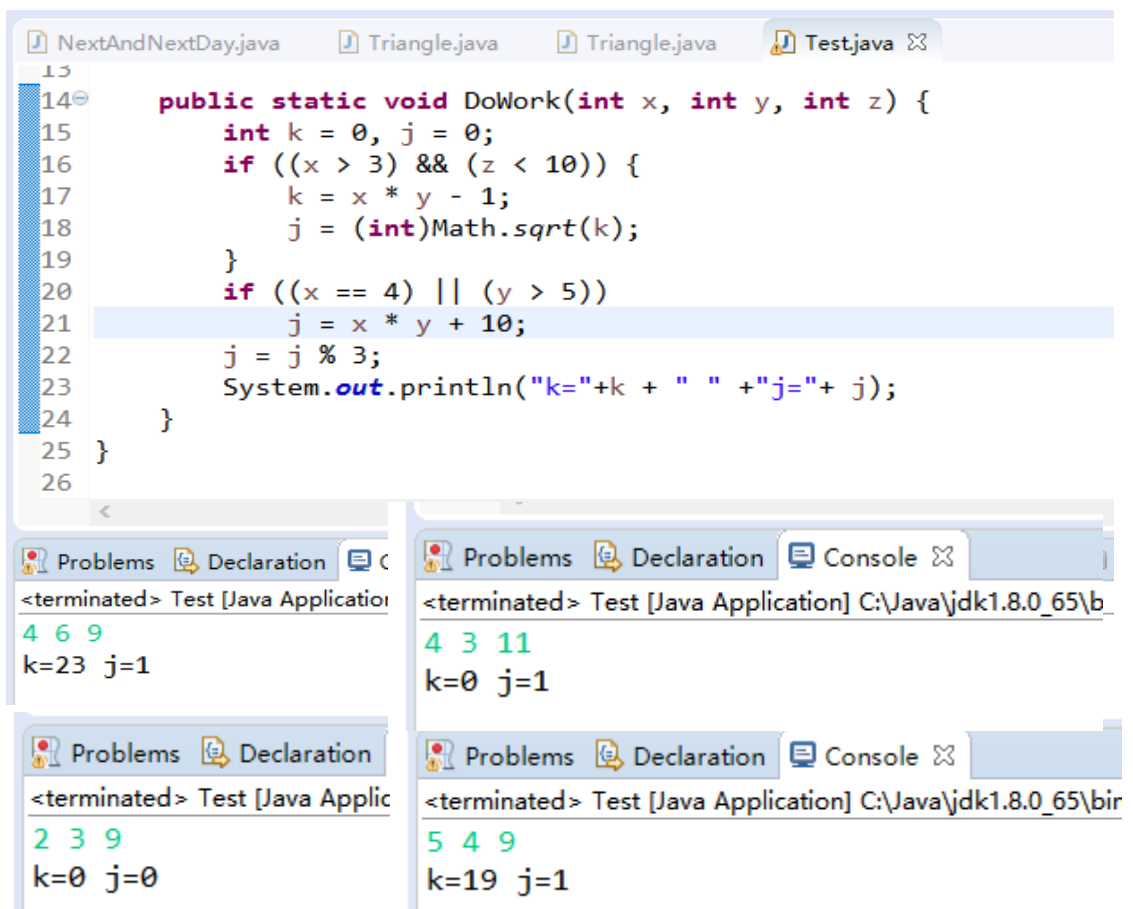
题目一将  $k$ 、 $j$  定义时不要使用 `int` 型，直接定义为 `double` 即可。

题目二增加一个计数功能，当用户累计输入错误三次后自动退出。

## 五、测试结果及分析

### 1、测试结果及分析

题目一：



```
13
14 public static void DoWork(int x, int y, int z) {
15     int k = 0, j = 0;
16     if ((x > 3) && (z < 10)) {
17         k = x * y - 1;
18         j = (int)Math.sqrt(k);
19     }
20     if ((x == 4) || (y > 5))
21         j = x * y + 10;
22     j = j % 3;
23     System.out.println("k="+k + " " + "j="+ j);
24 }
25 }
26
```

Problems Declaration Console

<terminated> Test [Java Application]

4 6 9  
k=23 j=1

Problems Declaration Console

<terminated> Test [Java Application] C:\Java\jdk1.8.0\_65\b\_

4 3 11  
k=0 j=1

Problems Declaration Console

<terminated> Test [Java Application]

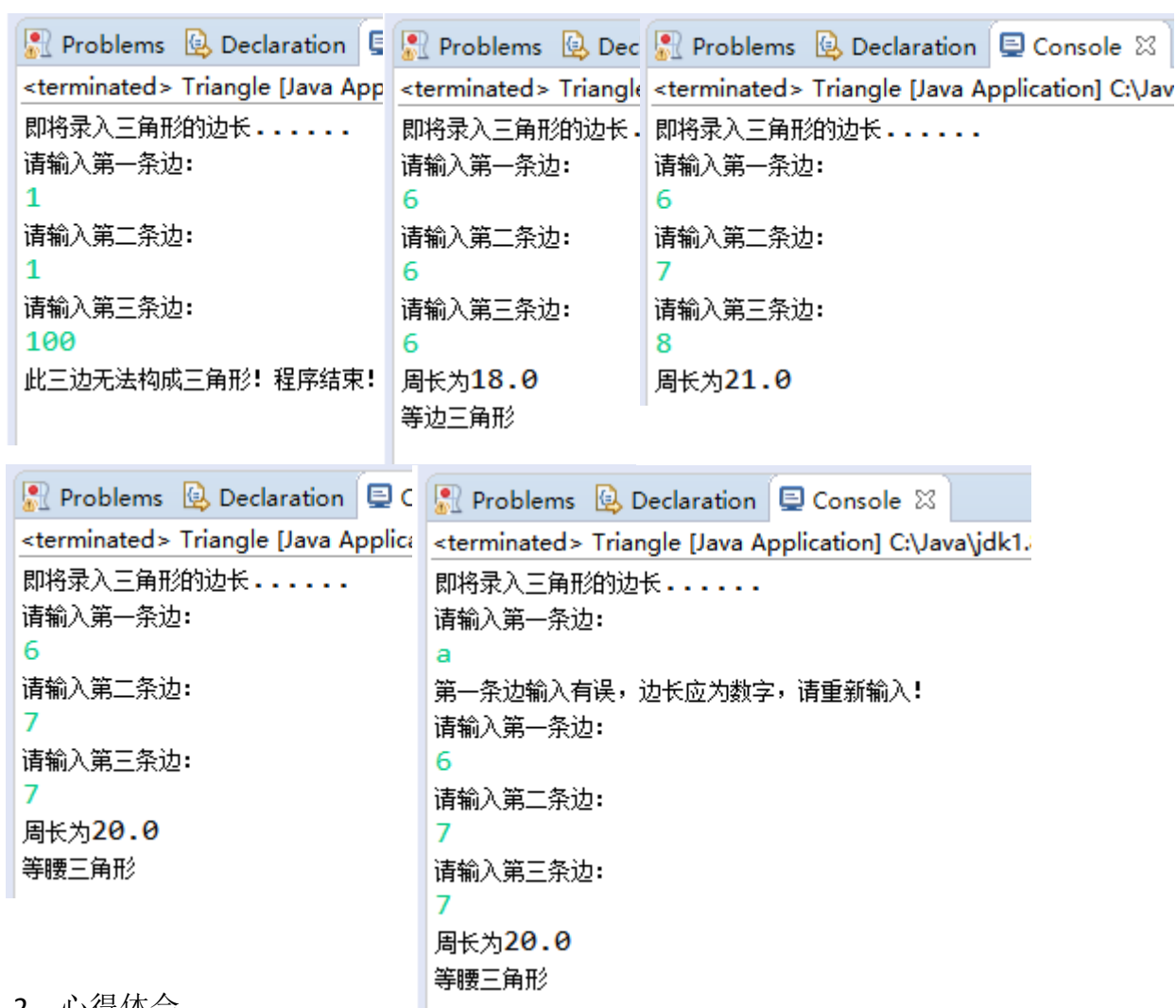
2 3 9  
k=0 j=0

Problems Declaration Console

<terminated> Test [Java Application] C:\Java\jdk1.8.0\_65\bir

5 4 9  
k=19 j=1

## 题目二：



## 2、心得体会

此次实验练习了白盒测试的两个测试方法，其中一个为逻辑覆盖，另外一个为基本路径覆盖。通过对这两个方法的使用，个人感觉基本路径覆盖更高效一些，它是根据圈复杂度确定测试用例个数，然后在利用路径覆盖对程序进行测试，相比较逻辑覆盖，基本路径覆盖的覆盖率很轻松就能达到 100%，使用起来很方便。

之前写程序都是只注重功能的实现，并没有进行这些测试，导致程序真正用的时候会出现很多 bug，其实这些 bug 都是一些很小的问题，只要稍微测试就可以发现它们。今后要多注重测试方面的工作，使自己的程序更加健壮。