# 实验三 软件测试

## 一、实验目的

1. 掌握结构化分析与设计方法

2. 掌握 JUnit 的用法；

3. 掌握白盒测试中路径覆盖的测试用例设计；

4. 掌握黑盒测试中等价类方法；

## 二、实验内容

1. 某证券公司为了方便提供证券交易服务，欲开发一证券交易平台。该平台接收客户命令信息，可执行以下主要功能：

（1）登录。输入用户名、密码及校验码信息，登录证券交易平台。

（2）开户。如果是新客户，需先提交开户信息，进行开户，并将客户信息存入客户记录中，账户信息（余额等）存入账户记录中；

（3）存款。客户可以向其账户中存款，根据存款金额修改账户余额；

（4）取款。客户可以从其账户中取款，根据取款金额修改账户余额；

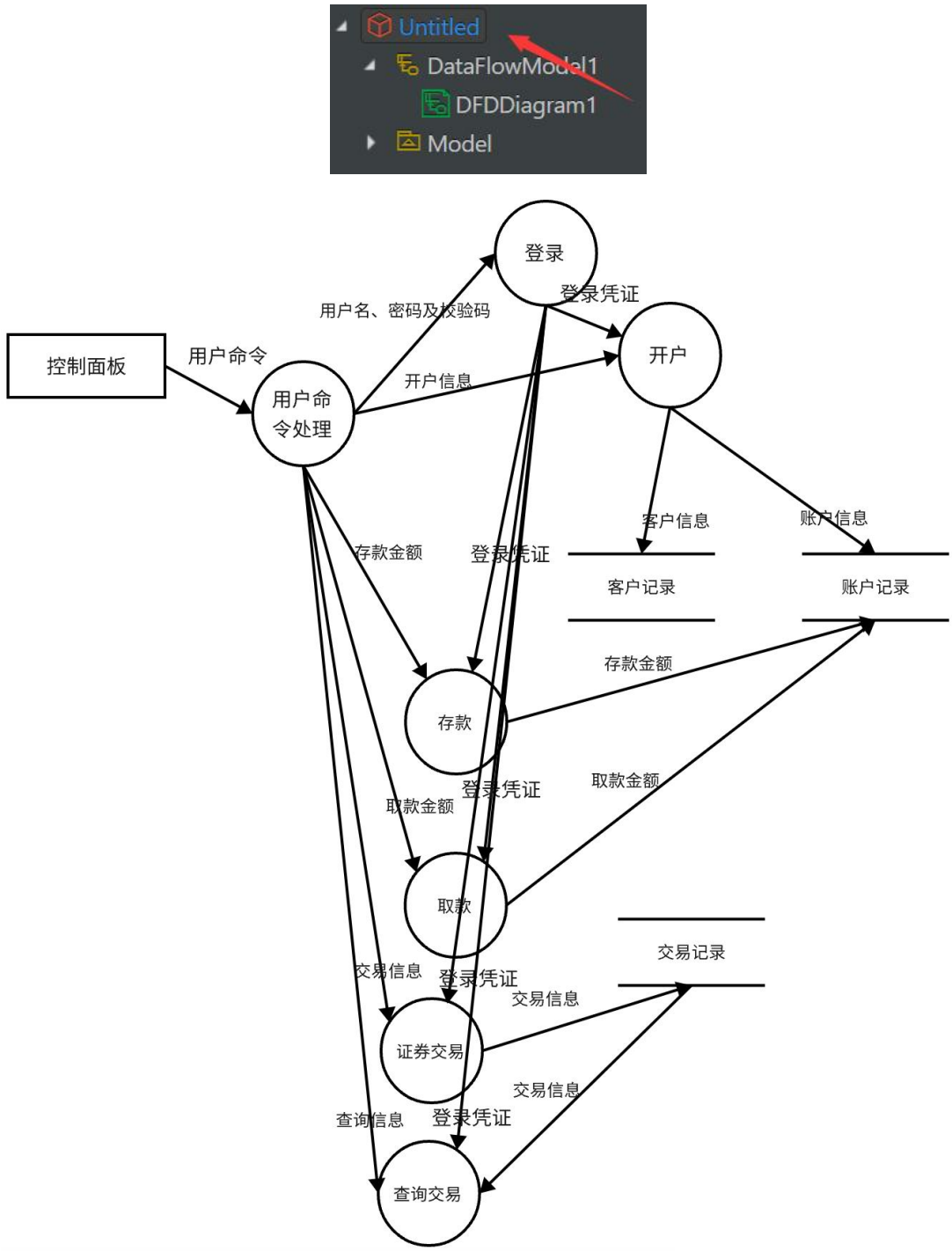
（5）证券交易。客户进行证券交易，将交易信息存入交易记录中；

（6）查询交易。平台从交易记录中读取交易信息，将交易明细返回给客户。。

要求：

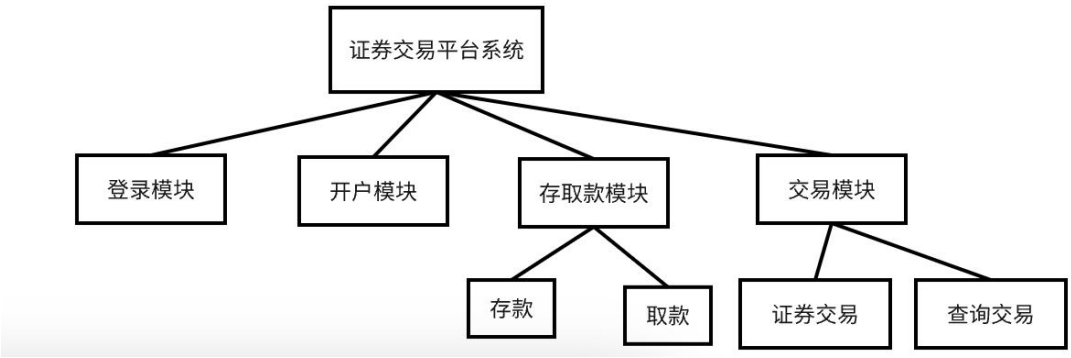
1）采用结构化分析方法，画出第 1 层的数据流图。

在 StarUML 中，可以右键点击模型（即下图 Untitled），选择“数据流图”。软件会生成一个数据流模型 DataFlowModel1，并自动创建一个数据流图 DFDDiagram1，在其上画图即可。




2）判断该信息流属于变换流还是事务流，并将该数据流图映射为程序结构，画出程序结构图。可以利用 Word 或 PowerPoint 的绘图功能绘制。

属于事务流




2. 进行测试驱动的开发

测试驱动开发，英文全称 Test-Driven Development，简称 TDD，是一种不同于传统软件开发流程的新型的开发方法。它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。

测试驱动开发过程可分为：1）分析需求；2）设计测试用例；3）开发测试代码；4）开发功能代码；5）运行测试，据此修改代码，直到全部通过。

本实验要求按照测试驱动的开发的过程进行一个类的开发。要求编写一个类 Circle，通过圆心点与半径定义一个圆。要求该类具备：

1）无参构造方法，圆心点默认为(0,0)，半径默认为 1。

2）两参数构造方法，接收 x,y 参数作为圆心坐标。半径默认为 1。

3）三参数构造方法，接收 x,y 参数作为圆心坐标，radius 参数为半径。

4）能够计算圆形面积与周长。

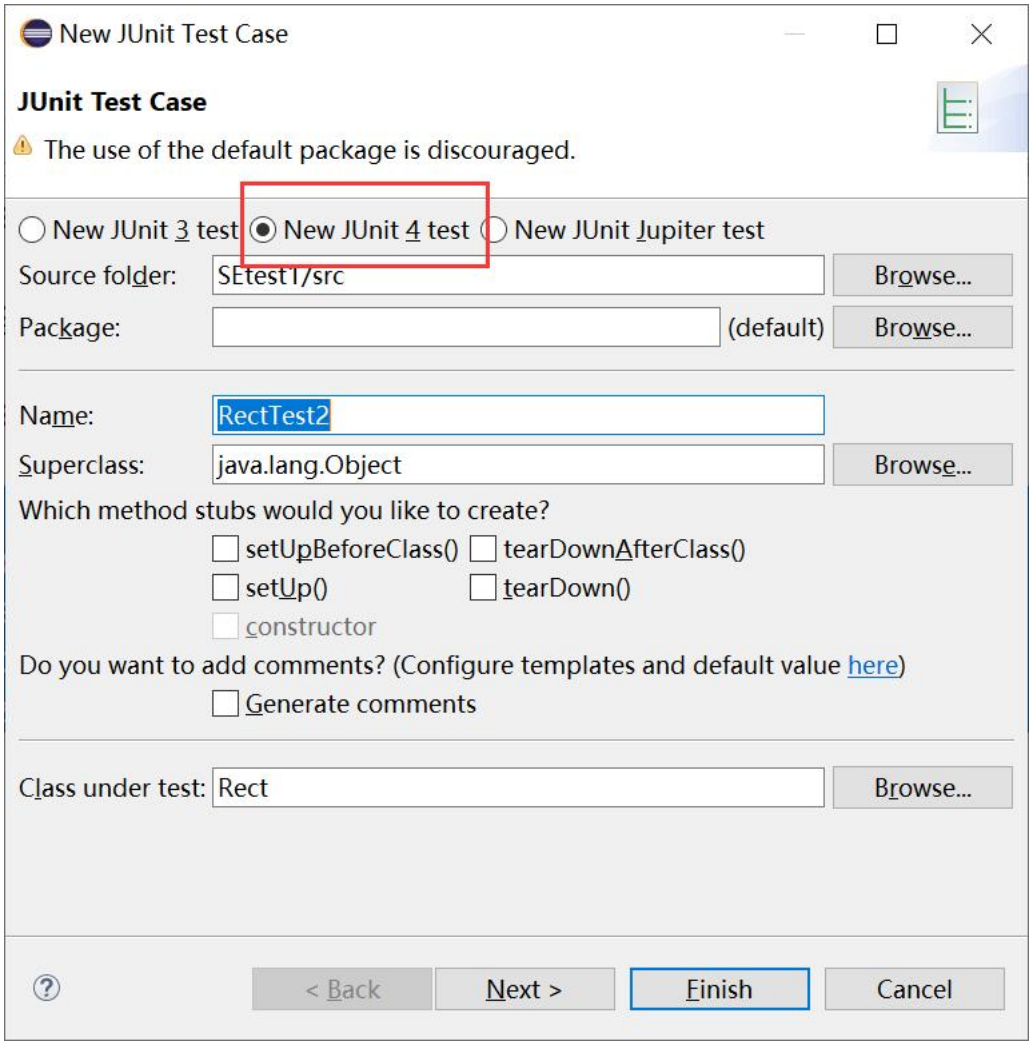
5）能够判断某个点是位于圆周内（返回 1）、在圆周上（返回 0，与圆周距离小于某个正值，该值可以参数形式提供）、还是在其外（返回-1）。

以下以 Java 语言为例，可按照以下步骤进行开发。（可以采用其它语言和相应的单元测试包，如 PyUnit，CppUnit）

1）建立一个空的 Rect 类；

2）右键点击 Rect 类，选择 New→Junit Test Case，在对话模式选择：

(如果 JDK 版本超过 9，可以选择 New Junit Jupiter test，即 Junit5 测试，只需注意 module path 的设置，eclipse 能够为你推荐正确的设置方法，只须照做即可。Junit 4 不需要进行设置)




建立一个测试用例 RectCreationTest，用于测试 Rect 的构造方法。

3）第一个测试可写成：

```
@Test
public void test() {
    Circle r = new Circle();
```

assertNotEquals(p, null, “构造函数出错”);

assertEquals(r.getX(), 0.0 , 1.0e-6, “无参构造函数圆心出错”);

assertEquals(r.getY(), 0.0 , 1.0e-6, “无参构造函数圆心出错”);

assertEquals(r.getRadius(), 1.0 , 1.0e-6, “无参构造函数半径出错”);

}

即用于测试无参构造方法所得对象，圆心点默认为(0,0)，半径默认为 1。

4）运行测试，Run As→Junit Test，不通过。

5）开发功能代码，即写上无参构造方法。

6）再次运行测试，直至通过。至此可保证无参构造方法已编写完毕。

按照上述过程，将上面要求的其余 4 项功能按照测试驱动开发的方式编写完成。将整个项目打包上传。

Test_circle.py

```python
import unittest
from circle import Circle


class TestCircle(unittest.TestCase):
    def setUp(self):
        self.default_circle = Circle()
        self.two_param_circle = Circle(3, 4)
        self.three_param_circle = Circle(1, 2, 3)

    def test_default_constructor(self):
        self.assertAlmostEqual(self.default_circle.x, 0.0, places=6)
        self.assertAlmostEqual(self.default_circle.y, 0.0, places=6)
        self.assertAlmostEqual(self.default_circle.radius, 1.0, places=6)

    def test_two_param_constructor(self):
        self.assertAlmostEqual(self.two_param_circle.x, 3.0, places=6)
        self.assertAlmostEqual(self.two_param_circle.y, 4.0, places=6)
        self.assertAlmostEqual(self.two_param_circle.radius, 1.0, places=6)

    def test_three_param_constructor(self):
        self.assertAlmostEqual(self.three_param_circle.x, 1.0, places=6)
        self.assertAlmostEqual(self.three_param_circle.y, 2.0, places=6)
        self.assertAlmostEqual(self.three_param_circle.radius, 3.0, places=6)

    def test_area(self):
        self.assertAlmostEqual(self.default_circle.area(), 3.141592653589793,
        places=6)
```

```
        self.assertAlmostEqual(
            self.three_param_circle.area(), 28.274333882308138, places=6
        )

    def test_perimeter(self):
        self.assertAlmostEqual(
            self.default_circle.perimeter(), 6.283185307179586, places=6
        )
        self.assertAlmostEqual(
            self.three_param_circle.perimeter(), 18.84955592153876, places=6
        )

    def test_point_inside(self):
        self.assertEqual(self.default_circle.point_inside(0, 0), 1)
        self.assertEqual(self.default_circle.point_inside(1, 0), 0)
        self.assertEqual(self.default_circle.point_inside(2, 0), -1)


if __name__ == "__main__":
    unittest.main()
```

Circle.py

```
import math


class Circle:
    def __init__(self, x=0, y=0, radius=1):
        self.x = x
        self.y = y
        self.radius = radius

    def area(self):
        return math.pi * self.radius**2

    def perimeter(self):
        return 2 * math.pi * self.radius

    def point_inside(self, px, py, tolerance=1e-6):
        distance = math.sqrt((px - self.x) ** 2 + (py - self.y) ** 2)
        if distance < self.radius - tolerance:
            return 1
        elif abs(distance - self.radius) < tolerance:
            return 0
        else:
            return -1

```

```
(base) nanmener@Haotians-MacBook-Pro 实验3 % python3 test_circle.py
......
----------------------------------------------------------------------
Ran 6 tests in 0.000s

OK
```

3. 设计一个静态方法，接收一个字符串，判断该字符串是否为一个格式正确的电话号码。所谓正确格式，有以下规则：

1）如果含 3 位数值，首位需为 1；（110 等）

2）如果含 8 位数值，首位不为 0；（本地座机号码）

3）如果含 11 位数值，首位需为 1（手机号码）或 0（北京等地座机号码）；

4）如果含 12 位数值，首位需为 0；（杭州等地座机号码，第 5 位不检测）

（国外号码等情况不考虑）

请结合等价类方法给出该方法的测试用例，并基于 JUnit，写出测试类。

方法：

1）等价类分为：

- 有效等价类：是指对于程序的规格说明来说，是合理的，有意义的输入数据构成的集合。
- 无效等价类：是指对于程序的规格说明来说，是不合理的，无意义的输入数据构成的集合。

2）确立等价类划分原则

原则 1：若规定了取值范围，或输入值的个数，则可以确立一个有效等价类和两个无效等价类。

例：程序对输入条件的要求是：…输入数是从 1 到 999…

则　有效等价类是：1≦输入数≦999；

两个无效等价类是：输入数〈1 或 输入数>999

原则 2：如果规定了输入数据的一组值，而且程序要对每种输入数据分别处理，则可为每种输入值确立一个有效等价类，此外针对这组值确立一个无效等价类，它是所有不允许的输入值的集合。

例：教师上岗方案中规定对教授、副教授、讲师和助教分别计算分数，做相应的处理。因此可以确定 4 个有效等价类为教授、副教授、讲师和助教，一个无效等价类，它是所有不符合以上身分的人员的输入值的集合。

原则 3：若规定了输入值的集合，或者是规定了“必须如何”的条件，则可确立一个有效等价类和一个无效等价类。

例：某种语言对变量标识符规定必须“以字母打头”，则所有以字母打头的

构成有效等价类，而不以字母打头的归于无效等价类。

原则 4：如果规定输入数据为整型，则可划分出正整数、零和负整数三个有效类，其他数据为无效类。

原则 5：如果程序处理对象是表格，则应使用空表、含一项和多项的表。

3）在确立了等价类之后，建立等价类表，列出所有划分出的等价类，每个等价类分配一个唯一的编号。

| 输入条件 | 有效等价类 | 无效等价类 |
| --- | --- | --- |
| …… | …… | …… |
| …… | …… | …… |

例如：

| 输入等价类 | 合理等价类 | 不合理等价类 |
| --- | --- | --- |
| 报表日期的类型及长度 | ① 6位数字字符 | ② 有非数字字符<br>③ 少于6位数字字符<br>④ 多于6位数字字符 |
| 年份范围 | ⑤在1990～1999之间 | ⑥ 小于1990<br>⑦ 大于1999 |
| 月份范围 | ⑧ 在1～12之间 | ⑨ 等于0<br>⑩ 大于12 |

**4）设计测试用例，覆盖所有的有效等价类组合，和所有的无效等价类组合。**

设计一个新的测试用例，使它能够尽量覆盖尚未覆盖的有效等价类。重复这个步骤，直到所有的有效等价类均被测试用例所覆盖。

设计一个新的测试用例，使它仅覆盖一个尚未覆盖的无效等价类。重复这一步骤，直到所有的无效等价类均被测试用例所覆盖。

| 序号 | 测试用例描述 | 输入参数 | | | 期望输出 | 覆盖范围 |
| --- | --- | --- | --- | --- | --- | --- |
| | | 位数 | 首位 | 用例 | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 有效等价类 | 3 | 1 | 110 | true | 1，5，8 |
| 2 | 无效等价类 | 3 | 2 | 222 | false | 2 |
| | …… | | | | | 3 |
| | | | | | | |

请按照上述介绍，将“3）等价类表”和“4）测试用例”写在下方：

| 输入等价类 | 有效等价类 | 无效等价类 |
|---|---|---|
| 字符串位数 | （1）3 位，首位为 1<br>（2）8 位，首位不为 0<br>（3）11 位，首位为 1<br>（4）11 位，首位为 0<br>（5）12 位，首位为 0 | （6）3 位，首位不为 1<br>（7）8 位，首位为 0<br>（8）11 位，首位不为 1 且不为 0<br>（9）12 位，首位不为 0<br>（10）非 3、8、11、12 位数值 |
| 字符 | (11)ASCII 码属于 48-57 | (12)ASCII 码小于 48<br>(13)ASCII 码大于 57 |

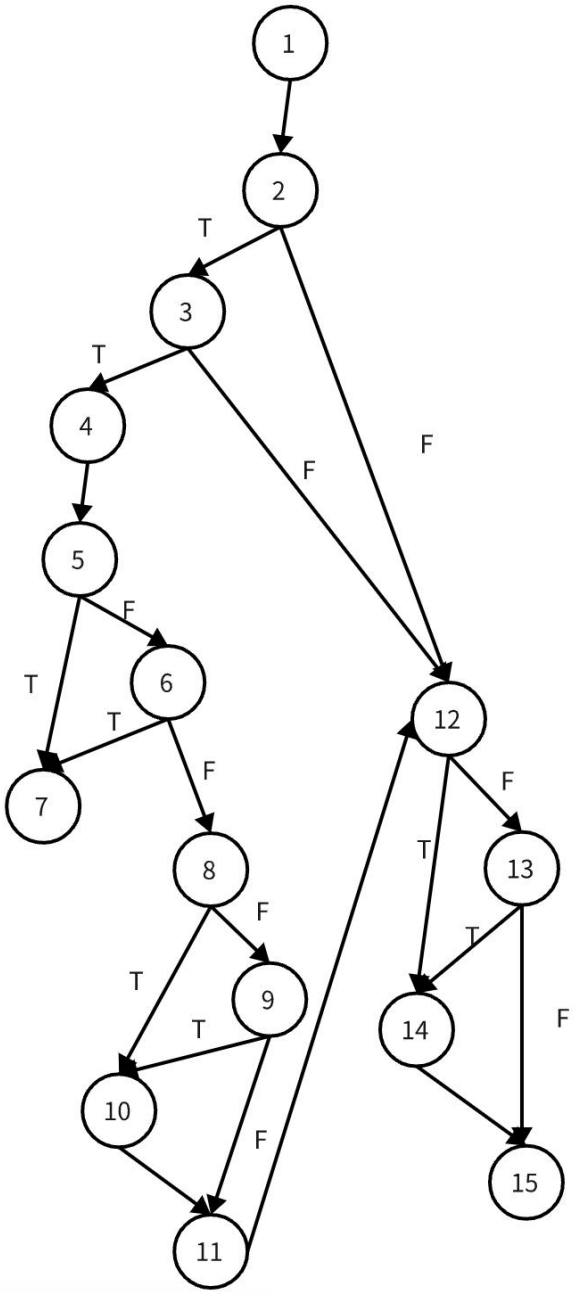
| 序号 | 测试用例描述 | 输入参数 | | | 期望输出 | 覆盖范围 |
|---|---|---|---|---|---|---|
| | | 位数 | 首位 | 用例 | | |
| 1 | 有效等价类 | 3 | 1 | 110 | true | 1,11 |
| 2 | 有效等价类 | 8 | 8 | 81234567 | true | 2,11 |
| 3 | 有效等价类 | 11 | 1 | 13912341234 | true | 3,11 |
| 4 | 有效等价类 | 11 | 0 | 01212341234 | true | 4,11 |
| 5 | 有效等价类 | 12 | 0 | 012345678912 | true | 5,11 |
| 6 | 无效等价类 | 3 | 0 | 010 | false | 6 |
| 7 | 无效等价类 | 8 | 0 | 01234567 | false | 7 |
| 8 | 无效等价类 | 11 | 2 | 21212341234 | false | 8 |
| 9 | 无效等价类 | 12 | 1 | 112341234123 | false | 9 |
| 10 | 无效等价类 | 1 | 1 | 1 | false | 10 |
| 11 | 无效等价类 | 3 | 1 | 10! | false | 12 |
| 12 | 无效等价类 | 3 | 1 | 10A | false | 13 |

4. 有以下程序（其中的操作并没有什么具体意义，仅用于演示）

基于 JUnit，针对以下程序设计测试用例，满足路径覆盖标准。

```
import java.awt.Point;
public class Path {

    public static Point Coverage(int a, int b, int c)
    {
        Point p = new Point();
        int x = 5; //1
        int y = 7;
        if(a>b && b>c) //2,3     {
            a += 1;1   //4
            x += 6;

            if(a==10 || b>20)//5,6
            {
                b += 1;    //7
                x = y+4;
            }

            if(a<10 || c==20)//8,9
            {
                b += 2;    //10
                y = 4;
            }

            a = a + b + 1;    //11
            y = x + y;
        }

        if(a>5 || c<10)   //12,13
        {
            b = c + 5; //14
            x += 1;
        }

        p.x = x;   //15
        p.y = y;
        return p;
    }
}
```

画出程序流图，如教材 P329。计算独立路径数量，进而设计基本路径测试所需的测试用例：




| 序号 | 测试用例描述 | 输入参数 | | | 期望输出 |
|---|---|---|---|---|---|
| | | a | b | c | |
| 1 | 基本路径测试 1 | 3 | 4 | 5 | 6，7 |
| 2 | 基本路径测试 2 | 7 | 6 | 5 | 12，7 |
| 3 | 基本路径测试 3 | 9 | 6 | 5 | 15，11 |
| 4 | 基本路径测试 4 | 8 | 21 | 5 | 18，11 |
| 5 | 基本路径测试 5 | 7 | 6 | 20 | 12，8 |
| 6 | 基本路径测试 6 | 9 | 21 | 20 | 18，8 |
| 7 | 基本路径测试 7 | 6 | 7 | 11 | 6，7 |
| 8 | 基本路径测试 8 | 4 | 6 | 9 | 6，7 |