

Java 应用与开发课程实验手册¹

王晓东

中国海洋大学

信息科学与工程学院计算机科学与技术系

2017 年 10 月 16 日

¹配套中国海洋大学信息科学与工程学院计算机科学与技术系《Java 应用与开发》课程使用。

目 录

1	Java 面向对象特性	1
1.1	实验目的	1
1.2	实验要求	1
1.3	问题分析	2
1.4	实验过程、步骤及原始记录	3
1.5	参考程序	6
2	模拟扑克牌游戏的洗牌	11
2.1	实验要求	11
2.2	实验过程、步骤及原始记录	12
2.3	参考程序	13
3	设计模式: 观察者 (Observer) 模式	16
3.1	实验目的	16
3.2	需求分析	16
3.3	问题分析	18
3.3.1	观察者模式定义	18
3.3.2	松耦合	19
3.4	实验过程、步骤及原始记录	19
3.5	参考程序	20
3.6	Java 自带的观察者模式	22
3.6.1	被观察者类	22
3.6.2	观察者类	22
3.6.3	示例代码	23

图目录

3.1	气象站应用的架构	17
3.2	气象站 UML 类图	18
3.3	观察者模式	18

表目录

序 言

本文档为中国海洋大学信息科学与工程学院计算机科学与技术系《J2EE 应用与开发》实验课程参考资料。

本文档参考了多方书籍、资料和源代码而编写，在此对原始资料的作者和编者表示感谢。但为保证实验授课质量，并没有随本文档公开原始参考资料的出处，望相关作者和编者谅解！

基于 GNU 通用公共许可证（General Public License）的条款，任何人有权复制、发布和 / 或修改本文。虽然由于参考资料的版权限制，本开放发布策略并不算非常合理。

第 1 章 Java 面向对象特性

1.1 实验目的

1. 熟悉 Java 开发工具 Eclipse 的安装和使用。
2. 熟练使用 Maven 创建和管理 Java 工程。
3. 理解面向对象编程的封装、多态、继承等特点。
4. 掌握面向对象技术的编程方法。

1.2 实验要求

实验一（必做）

下载并安装 Java 开发环境 Eclipse，熟练应用 Eclipse。

- 熟悉 Eclipse 的基本菜单项，各类视图。
- 操作创建 Java 项目。
- 操作创建 Java Class、Interface 等。
- 掌握 Eclipse 的程序调试方法。

实验二（必做）

设计一个银行账户类，其中包括：

- 账户信息，如帐号、姓名、开户时间、身份证号码等。
- 存款方法。
- 取款方法。
- 其他方法如“查询余额”和“显示账户信息”等。

用字符界面模拟存款和取款过程。

实验三（必做）

计算三角形面积和周长，要求如下：

- 设计 Point 类（点类）。
- 设计 Triangle 类，至少包括其构造方法，获取面积和周长的方法。
- 通过输入三点坐标来确定一个三角形，并输出三角形的面积、周长等内容。

实验四（选做）

编写程序求解一元多次方程的解，要求如下：

- 至少包括一元一次、一元二次、一元三次方程。
- 至少设计两个接口。
- 必须用到内部类和包。
- 写出详细的运行过程。

1.3 问题分析

实验一问题分析

无。

实验二问题分析

1. 根据题意，设计类 UserInfo 实现对银行账户的封装，其中类的成员变量包含帐号、余额、姓名、开户时间、身份证号等信息，而成员方法主要实现存款、取款、显示等功能，为方便对账户信息进行初始化，需增加类构造方法。
2. 为模拟字符操作界面，另外设计类 BankAccount，以实现对 UserInfo 对象的调用。

实验三问题分析

1. 采用海伦公式求解三角形面积，假定三角形三边为 a 、 b 、 c ，其面积 $S = \sqrt{p * (p - a) * (p - b) * (p - c)}$ ，其中 $p = (a + b + c) / 2$ 。
2. 由于题目中只给出了三角形的三点坐标，因此在应用海伦公式前还需要设计方法求出每条边的长度，假定任意两点的坐标为 (x_1, y_1) 、 (x_2, y_2) ，则其长度 $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ 。

实验四问题分析

1. 对于一元一次和一元二次方程的求解算法很简单，这里不赘述。

2. 一元三次方程的一种求解算法简单分析如下：

三次方程的一般形式为

$$ax^3 + bx^2 + cx + d = 0$$

两边除以 a 后设 $x = y - \frac{b}{3a}$ ，就可以转换成 $y^3 + py + q = 0$ 的形式，其中

$$p = \frac{c}{a} - \frac{b^2}{3a^2}$$

$$q = \frac{2b^3}{27a^3} - \frac{bc}{3a^2} + \frac{d}{a}$$

令

$$y = z - \frac{p}{3z}$$

得

$$z^3 - \frac{p^3}{27z^3} + q = 0$$

即

$$(z^3)^2 + qz^3 - \frac{p^3}{27} = 0$$

解这个关于 z^3 的二次方程，得

$$z^3 = -\frac{q}{2} + \sqrt{\left(\frac{p}{3}\right)^3 + \left(\frac{q}{2}\right)^2}$$

这样就将求解一个一元三次方程转化成求解一个一元二次方程的问题。

1.4 实验过程、步骤及原始记录

实验二

实验过程和代码如下：

```
1 .
2 .
3 .
4 .
5 .
6 .
7 .
8 .
9 .
10 .
```

输出结果参考如下：

output

请选择要进行的操作：1. 存款 2. 取款 3. 查询 4. 显示账户 5. 退出

1

请输入要存入的金额（整数）：

1000

存款成功！已存入 1000 元可用余额为 2234 元

请选择要进行的操作：1. 存款 2. 取款 3. 查询 4. 显示账户 5. 退出

2

请输入要取款金额（整数）

400

取款成功！已取出 400 元可用余额为 1834 元

请选择要进行的操作：1. 存款 2. 取款 3. 查询 4. 显示账户 5. 退出

4

帐号：1

姓名：Java

开户时间：Sun Feb 01 00:00:00 CST 2009

身份证号:1111

请选择要进行的操作：1. 存款 2. 取款 3. 查询 4. 显示账户 5. 退出

5

程序退出

实验四

实验过程和代码如下：

1
2
3
4
5
6
7
8
9
10

输出结果参考如下：

output

求解几次方程？ 1：一次 2：二次 3：三次

3

你选择的是一元三次方程

请输入形如一元三次方程 $mx^3+nx^2+tx+s=0$ 的四个系数

请输入系数 m：

2

请输入系数 n：

3

请输入系数 t:

4

请输入系数 s:

5

此方程有一个解:

$x = -1.3711343313073632$

你是否想继续: (y/n)

1.5 参考程序

实验二

File: BankAccount.java

```
1 package cn.edu.ouc.javase;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6 import java.util.Date;
7
8 class UserInfo {
9     private String id;
10    private String name;
11    private Date createDate;
12    private String idByCard;
13    private long money;
14
15    // 无参构造方法
16    public UserInfo() {
17    }
18
19    // 有参构造方法
20    public UserInfo(String _id, String _name, Date _createDate,
21        String _idByCard, long _money) {
22
23        this.id = _id;
24        this.name = _name;
25        this.createDate = _createDate;
26        this.idByCard = _idByCard;
27        this.money = _money;
28    }
29
30    public long AddMoney(long amount) {
31        money = amount + money;
32        return money;
33    }
34
35    public long DepositMoney(long amount) {
36        money = money - amount;
37        return money;
38    }
39
40    public long getMoney() {
41        return this.money;
42    }
43
44    // 打印人员信息
45    public void showUserInfo() {
```

```
46     System.out.println("帐号: " + id);
47     System.out.println("姓名: " + name);
48     System.out.println("开户时间: " + createDate);
49     System.out.println("身份证号:" + idByCard);
50 }
51 }

53 public class BankAccount {

55     public static void main(String[] args) {
56         UserInfo user = new UserInfo("1", "Java", new Date(109, 1, 1), "1111",
57             1234);
58         BankAccount b = new BankAccount();
59         b.operate(user);
60     }

62     public void operate(UserInfo user) {
63         long saveMoney = 0;
64         long takeMoney = 0;
65         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
66         String choose = "";

68         while (true) {
69             System.out.println("请选择要进行的操作: 1. 存款 2. 取款 3. 查询"
70                 + " 4. 显示账户 5. 退出");

72             try {
73                 choose = br.readLine();
74             } catch (IOException e) {
75                 e.printStackTrace();
76             }

78             // 存款
79             if (choose.compareTo("1") == 0) {
80                 while (true) {
81                     System.out.println("请输入要存入的金额 (整数): ");

83                     try {
84                         saveMoney = Long.parseLong(br.readLine());
85                     } catch (NumberFormatException e) {
86                         System.out.println("输入错误, 请输入整数");
87                         continue;
88                     } catch (IOException e) {
89                     }

91                     long balance = user.AddMoney(saveMoney);
92                     System.out.println("存款成功! 已存入" + saveMoney + "元" + "可用余额为"
93                         + balance + "元");
94                     break;
95                 }
96             }
```

```
98 // 取款
99 if (choose.compareTo("2") == 0) {
100     while (true) {
101         System.out.println("请输入要取款金额（整数）");
102
103         try {
104             takeMoney = Long.parseLong(br.readLine());
105
106         } catch (NumberFormatException e) {
107             System.out.println("输入错误，请输入整数");
108             continue;
109         } catch (IOException e) {
110
111         }
112
113         if (!(user.getMoney() < takeMoney)) {
114             long balance = user.DepositMoney(takeMoney);
115             System.out.println("取款成功！已取出" + takeMoney + "元"
116                 + "可用余额为" + balance + "元");
117             break;
118         } else {
119             System.out.println("余额不足，请重新输入");
120         }
121     }
122 }
123
124 // 查询
125 if (choose.compareTo("3") == 0) {
126     System.out.println("你的余额为" + user.getMoney() + "元");
127 }
128
129 // 显示账户
130 if (choose.compareTo("4") == 0) {
131     user.showUserInfo();
132 }
133 // 退出
134 if (choose.compareTo("5") == 0) {
135     System.out.println("程序退出");
136     System.exit(0);
137 }
138 }
139 }
140 }
```

实验四

File: SolveEquation.java

```
1 package cn.edu.ouc.javase;
3 import java.io.IOException;
```

```
4 import java.util.Scanner;

6 public class SolveEquation {

8     void print() throws IOException {
9         char ch = 'y';
10        System.out.println("求解几次方程? 1: 一次 2: 二次 3: 三次");
11        Scanner sLine = new Scanner(System.in);
12        int pm = sLine.nextInt();

14        if (pm == 3) {
15            System.out.println("你选择的是一元三次方程");
16            SolveEquation.SolveCubicEquation fc = new SolveEquation().new SolveCubicEquation();
17            fc.SolveCubicEquation();
18        } else {
19            print();
20        }

22        System.out.println("你是否想继续: (y/n)");
23        ch = (char) System.in.read();
24        System.in.skip(2);

26        if (ch == 'y') {
27            print();
28        } else if (ch == 'n') {
29            System.out.println("Good luck!");
30        } else {
31            print();
32        }
33    }

35    public static void main(String[] args) throws IOException {
36        SolveEquation se = new SolveEquation();
37        se.print();
38    }

40    // 接口 规划求解一元三次方式必须实现的方法
41    interface ItCubicEquation {
42        void SolveCubicEquation();
43    }

45    class SolveCubicEquation implements ItCubicEquation {

47        @Override
48        public void SolveCubicEquation() {
49            System.out.println("请输入形如一元三次方程  $mx^3+nx^2+tx+s=0$  的四个系数");
50            Scanner sce = new Scanner(System.in);
51            System.out.println("请输入系数 m: ");
52            double m = sce.nextDouble();
53            System.out.println("请输入系数 n: ");
54            double n = sce.nextDouble();
```

```
55     System.out.println("请输入系数 t: ");
56     double t = sce.nextDouble();
57     System.out.println("请输入系数 s: ");
58     double s = sce.nextDouble();

60     if (m == 0) {
61         System.out.println("输入错误");
62     } else {
63         double a = n / m;
64         double b = t / m;
65         double c = s / m;
66         double q = (a * a - 3 * b) / 9;
67         double r = (2 * a * a * a - 9 * a * b + 27 * c) / 54;

69         if (r * r < q * q * q) {
70             System.out.println("此方程有三个解: ");
71             t = Math.acos(r / Math.sqrt(q * q * q));
72             double x1 = -2 * Math.sqrt(q) + Math.cos(t / 3) - a / 3;
73             double x2 = -2 * Math.sqrt(q)
74                 + Math.cos((t + 2 * Math.PI) / 3) - a / 3;
75             double x3 = -2 * Math.sqrt(q)
76                 + Math.cos((t - 2 * Math.PI) / 3) - a / 3;

78             System.out.println("x1 = " + x1 + ", x2 = " + x2
79                 + ", x3 = " + x3);
80         } else {
81             System.out.println("此方程有一个解: ");
82             int sgn = (r >= 0) ? 1 : -1;
83             double u = -sgn
84                 * Math.pow(
85                 (Math.abs(r) + Math.sqrt(r * r - q * q * q)),
86                 1.0 / 3);
87             double v = (u != 0) ? q / u : 0;
88             double x1 = u + v - a / 3;
89             System.out.println("x = " + x1);
90         }
91     }
92 }
93 }
94 }
```

第 2 章 模拟扑克牌游戏的洗牌

2.1 实验要求

结合面向对象设计原则，分析和设计模拟扑克牌游戏的洗牌过程。

1. 编程定义一个表示扑克牌的类 **Poke**，用 **suit** 来表示扑克牌的花色，用 **face** 来表示扑克牌的牌面值，注意别忘了大小王，每副牌为 54 张。
2. 假设扑克牌游戏需要 2 付 (或 2 付以上) 的牌，请编程实现模拟洗牌，以及将这些扑克牌分给参加游戏的所有人，并将参加扑克牌游戏的所有人的扑克牌输出。
3. 实现 **Poke** 类中包含的 4 个静态方法，所实现的 **Poke** 类在 **PokeTest** 的 **main** 方法中进行测试，请阅读 **main** 方法的代码，并根据其中逻辑猜测 **Poke** 类中静态方法的行为并编写方法实现代码。

Poke.java

```
1 package ouc.cs.course.java.test.poke;
2 import java.util.Random;
3
4 public class Poke {
5     public static String[] createCard(int number) {
6     }
7
8     public static void display(String[] cards) {
9     }
10
11     public static void shuffle(String[] cards) {
12     }
13
14     public static void distribute(String[] cards,int player) {
15     }
16 }
```

PokeTest.java

```
1 package ouc.cs.course.java.test.poke;
2
3 import java.util.Scanner;
4 import ouc.cs.course.java.test.poke.Poke;
```



```
6 public class PokeTest {  
8     @SuppressWarnings("resource")  
9     public static void main(String[] args) {  
10         System.out.println("该扑克游戏需要几付扑克牌? ");  
11         Scanner sc = new Scanner(System.in);  
12         int num = sc.nextInt();  
13         String[] cards = Poke.createCard(num);  
14         System.out.println("该扑克游戏有几个玩家? ");  
15         Scanner sp = new Scanner(System.in);  
16         int per = sp.nextInt();  
17         System.out.println("\n显示所有的牌: ");  
18         Poke.display(cards);  
19         Poke.shuffle(cards);  
20         System.out.println("\n显示分配给每个人的牌: ");  
21         Poke.distribute(cards, per);  
23     }  
24 }
```

4. 要随机产生某个范围内的整数，可以用 `java.util.Random` 类的 `nextInt(int num)`。例如，输出 0 - 53 范围内的一个整数如下：

```
1 Random rand = new Random();  
2 int num = rand.nextInt(54);
```

2.2 实验过程、步骤及原始记录

实验过程和代码如下：

1
2
3
4
5
6
7
8
9
10

2.3 参考程序

File: Poke.java

```
1 package ouc.cs.course.java.test.poke;
2
3 import java.util.Random;
4
5 public class Poke {
6     public static String[] createCard(int number) {
7         if (number < 2)
8             number = 2;
9         int cardNumber = number * 52;
10        String[] cards = new String[cardNumber];
11        int suit;
12        int face;
13        for (int i = 0; i < cardNumber; i++) {
14            cards[i] = new String();
15            suit = i % 4;
16            switch (suit) {
17                case 0:
18                    cards[i] = "红桃";
19                    break;
20                case 1:
21                    cards[i] = "黑桃";
22                    break;
23                case 2:
24                    cards[i] = "方块";
25                    break;
26                case 3:
27                    cards[i] = "梅花";
28                    break;
29            }
30            face = i % 13;
31            switch (face) {
32                case 0:
33                    cards[i] = cards[i] + "A";
34                    break;
35                case 1:
36                    cards[i] = cards[i] + "2";
37                    break;
38                case 2:
39                    cards[i] = cards[i] + "3";
40                    break;
41                case 3:
42                    cards[i] = cards[i] + "4";
43                    break;
44                case 4:
45                    cards[i] = cards[i] + "5";
46                    break;
```

```
47     case 5:
48         cards[i] = cards[i] + "6";
49         break;
50     case 6:
51         cards[i] = cards[i] + "7";
52         break;
53     case 7:
54         cards[i] = cards[i] + "8";
55         break;
56     case 8:
57         cards[i] = cards[i] + "9";
58         break;
59     case 9:
60         cards[i] = cards[i] + "10";
61         break;
62     case 10:
63         cards[i] = cards[i] + "J";
64         break;
65     case 11:
66         cards[i] = cards[i] + "Q";
67         break;
68     case 12:
69         cards[i] = cards[i] + "K";
70         break;
71     }
72 }
73 return cards;
74 }

76 public static void display(String[] cards) {
77     for (int i = 0; i < cards.length; i++) {
78         System.out.printf("%-7s", cards[i]);
79         if ((i + 1) % 13 == 0)
80             System.out.println();
81     }
82 }

84 public static void shuffle(String[] cards) {
85     Random r = new Random();
86     String tmp = null;
87     for (int i = 0; i < cards.length; i++) {
88         int k1 = r.nextInt(cards.length);
89         int k2 = r.nextInt(cards.length);
90         tmp = cards[k1];
91         cards[k1] = cards[k2];
92         cards[k2] = tmp;
93     }
94 }

96 public static void distribute(String[] cards, int player) {
97     int num = cards.length / player;
```

```
98     for (int j = 0; j < cards.length; j++) {
99         System.out.printf("%-7s", cards[j]);
100         if ((j + 1) % 13 == 0)
101             System.out.println();
102         if ((j + 1) % num == 0) {
103             System.out.println("第" + (j + 1) / num + "个人的牌如上");
104             System.out.println();
105         }
106     }
107 }
108 }
```

第3章 设计模式: 观察者（Observer）模式

3.1 实验目的

1. 理解设计模式之观察者（Observer）模式。
2. 理解模式中所包含的 Java 设计原则。
3. 基于观察者（Observer）模式编写 Java 程序，加深对观察者模式的理解。
4. 学习使用 Java 自带的观察者模式实现。

3.2 需求分析

假设我们团队获得了一项合约，是需求方 Weather-O-Rama 公司委托我们开发一个气象站应用。首先来看一下需求方 Weather-O-Rama 公司对气象站应用的需求概况（如图 3.1）。其中，WeatherData 对象用于从气象站获取温度、湿度和气压数据，并驱动多个显示装置显示。合约中要求至少包括目前状况、气象统计、天气预报的显示布告板。同时，需求方所给的 WeatherData 类参考设计如下。

Class: WeatherData

```
1 public class WeatherData {  
2     public float getTemperature() {  
3         // 从气象站获取温度数据，由需求方提供方法  
4     }  
5     public float getHumidity() {  
6         // 从气象站获取湿度数据，由需求方提供方法  
7     }  
8     public float getPressure() {  
9         // 从气象站获取气压数据，由需求方提供方法  
10    }  
11    public void measurementsChanged() {  
12        // 当气象数据值发生改变时，该方法被期望调用  
13    }  
14 }
```

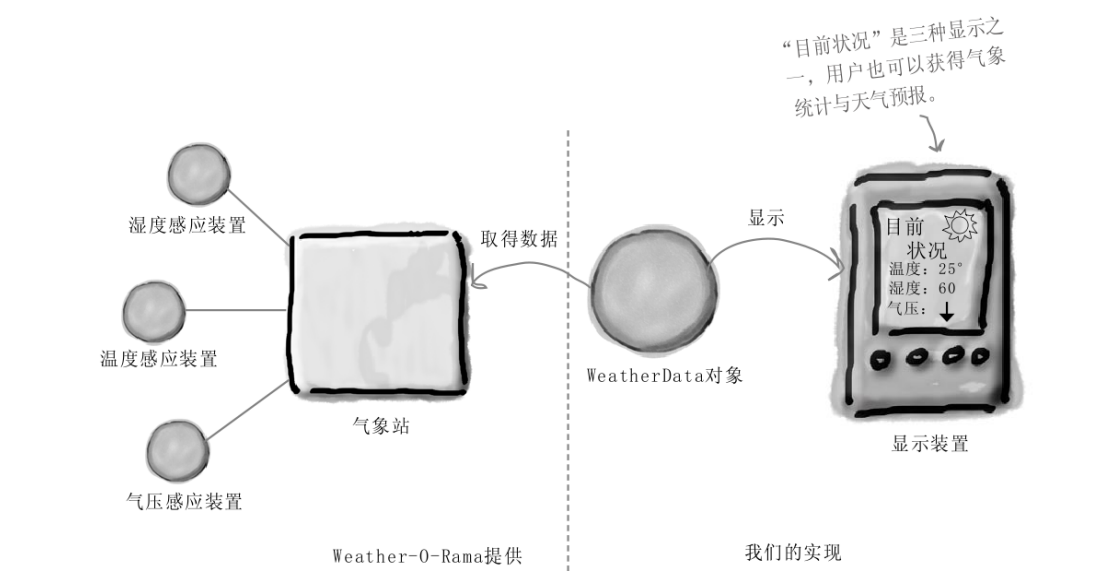


图 3.1: 气象站应用的架构

根据任务合约中的内容和需求方提供的 WeatherData 类参考，你一般会这样设计这个气象观测站：在 measurementsChanged 方法中添加了功能代码，实现了对三个布告板的数据更新，如下所示。

```
1 public class WeatherData {
2     // 实例变量声明
3
4     public void measurementsChanged() {
5         float temp = getTemperature(); // 调用取得三种数据的方法
6         float humidity = getHumidity();
7         float pressure = getPressure();
8
9         currentConditionsDisplay.update(temp, humidity, pressure);
10        statisticsDisplay.update(temp, humidity, pressure);
11        forecastDisplay.update(temp, humidity, pressure);
12    }
13    // 这里是其他的 WeatherData 方法
14 }
```

上述实现方案非常不佳，我们来分析一下并从如下选项中找出上述实现所存在的问题：

- 针对具体实现编程，而非针对接口编程 ☐
- 对于每添加一个布告板，都得修改代码 ☐
- 无法在运行时动态增加（删除）布告板 ☐
- 布告板没有实现一个共同的接口 ☐
- 尚未封装改变的部分 ☐

- 侵犯了 WeatherData 类的封装 □

所以，本实验将采取观察者（Observer）模式来实现气象站应用。根据 UML 类图 3.2，实现气象站，要求能够避免上述分析中旧设计所存在的问题。

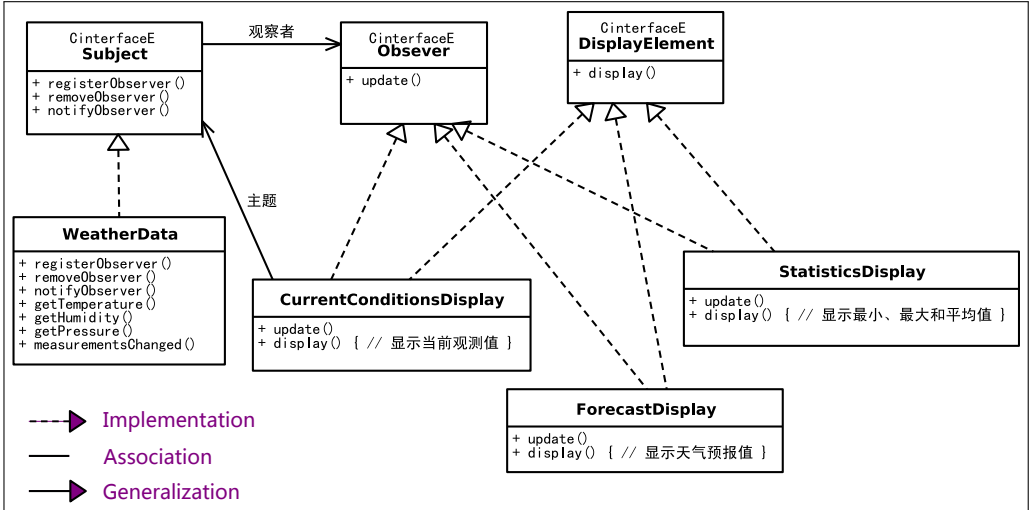


图 3.2: 气象站 UML 类图

3.3 问题分析

3.3.1 观察者模式定义

观察者模式定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新，如图 3.3。

在现实世界中，我们可以用报纸订阅服务来类比观察者模式。

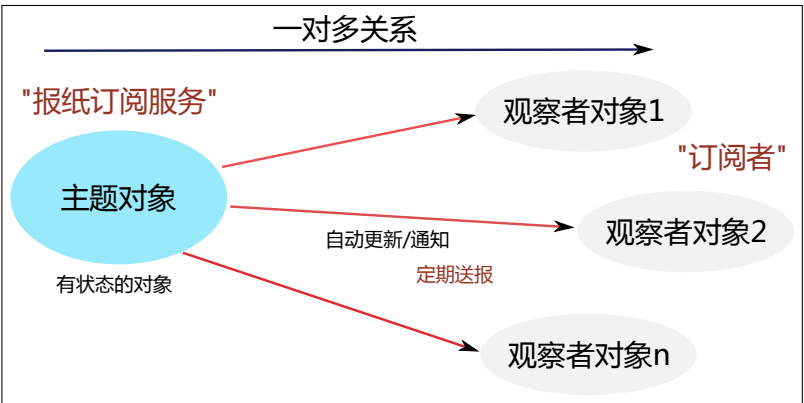


图 3.3: 观察者模式

3.3.2 松耦合

设计原则

为了交互对象之间的松耦合设计而努力。

两个对象之间松耦合，它们依然可以交互，但是不太清楚彼此的细节。观察者模式提供了一种对象设计，让主题和观察者之间松耦合。

- 关于观察者的一切，主题只知道观察者实现了某个接口（也就是 **Observer** 接口）。主题不需要知道观察者的具体类是谁、做了些什么或其他任何细节。
- 任何时候我们都可以增加新的观察者。因为主题唯一依赖的东西是一个实现 **Observer** 接口的对象列表，所以我们可以随时增加观察者。事实上，在运行时我们可以用新的观察者取代现有的观察者，主题不会受到任何影响。同样的，也可以在任何时候删除某些观察者。
- 有新类型的观察者出现时，主题的代码不需要修改。假如我们有个新的具体类需要当观察者，我们不需要为了兼容新类型而修改主题的代码，所有要做的就是新的类里实现此观察者接口，然后注册为观察者即可。主题不在乎别的，它只会发送通知给所有实现了观察者接口的对象。
- 我们可以独立地复用主题或观察者。如果我们在其他地方需要使用主题或观察者，可以轻易地复用，因为二者并非紧耦合。
- 改变主题或观察者其中一方，并不会影响另一方。因为两者是松耦合的，所以只要他们之间的接口仍被遵守，我们就可以自由地改变他们。

3.4 实验过程、步骤及原始记录

实验过程和代码如下：

```
1 .
2 .
3 .
4 .
5 .
6 .
```

输出结果参考如下（仅实现了当前气象状况布告板，另外的布告板请自行添加并完成实验报告）：

output

Current conditions: 80.0F degrees and 65.0% humidity
Current conditions: 78.0F degrees and 90.0% humidity

3.5 参考程序

本实验为学习验证性实验，直接给出参考程序，希望大家认真学习理解观察者模式的使用以及其中包含的设计哲学，并尝试增加其他类型的气象布告板，数据人工模拟即可。

File: Subject.java

```
1 package ouc.cs.java.desingpattern.observer;
2
3 public interface Subject {
4     public void registerObserver(Observer o);
5     public void removeObserver(Observer o);
6     public void notifyObservers();
7 }
```

File: Observer.java

```
1 package ouc.cs.java.desingpattern.observer;
2
3 public interface Observer {
4     public void update(float temperature, float humidity, float pressure);
5 }
```

File: DisplayElement.java

```
1 package ouc.cs.java.desingpattern.observer;
2
3 public interface DisplayElement {
4     public void display();
5 }
```

File: WeatherData.java

```
1 package ouc.cs.java.desingpattern.observer;
2
3 import java.util.ArrayList;
4
5 public class WeatherData implements Subject {
6
7     private ArrayList observers;
8
9     private float temperature;
10    private float humidity;
11    private float pressure;
12
13    public WeatherData() {
14        observers = new ArrayList();
15    }
16
17    @Override
18    public void registerObserver(Observer o) {
19        observers.add(o);
```

```
20 }  
  
22 @Override  
23 public void removeObserver(Observer o) {  
24     int i = observers.indexOf(o);  
25     if (i >= 0) {  
26         observers.remove(i);  
27     }  
28 }  
  
30 @Override  
31 public void notifyObservers() {  
32     for (int i = 0; i < observers.size(); i++) {  
33         Observer observer = (Observer) observers.get(i);  
34         observer.update(temperature, humidity, pressure);  
35     }  
36 }  
  
38 public void measurementsChanged() {  
39     notifyObservers();  
40 }  
  
42 public void setMeasurements(float temperature, float humidity, float pressure) {  
43     this.temperature = temperature;  
44     this.humidity = humidity;  
45     this.pressure = pressure;  
46     measurementsChanged();  
47 }  
48 }
```

File: CurrentConditionsDisplay.java

```
1 package ouc.cs.java.designpattern.observer;  
  
3 public class CurrentConditionsDisplay implements Observer, DisplayElement {  
  
5     private float temperature;  
6     private float humidity;  
7     private Subject weatherData;  
  
9     public CurrentConditionsDisplay(Subject weatherData) {  
10         this.weatherData = weatherData;  
11         weatherData.registerObserver(this);  
12     }  
  
14     @Override  
15     public void update(float temperature, float humidity, float pressure) {  
16         this.temperature = temperature;  
17         this.humidity = humidity;  
18         display();  
19     }  
}
```

```
21 @Override
22 public void display() {
23     System.out.println("Current conditions: " + temperature
24         + "F degrees and " + humidity + "% humidity");
25 }
26 }
```

File: WeatherStation.java

```
1 package ouc.cs.java.designpattern.observer;
2
3 public class WeatherStation {
4
5     public static void main(String[] args) {
6         WeatherData weatherData = new WeatherData();
7         CurrentConditionsDisplay cd = new CurrentConditionsDisplay(weatherData);
8
9         weatherData.setMeasurements(80, 65, 30.4f);
10        weatherData.setMeasurements(78, 90, 29.2f);
11    }
12 }
```

3.6 Java 自带的观察者模式

由于观察者模式的广泛应用性，Java API 自带了观察者模式的实现。其中，Observable 对象是被观察者，Observer 对象是观察者。使用 Java 自带的 API 实现观察者模式非常简单：

- 创建被观察者类（即 Subject），它继承自 java.util.Observable 类；
- 创建观察者类，它实现 java.util.Observer 接口；

3.6.1 被观察者类

使用被观察者类的 addObserver() 方法把观察者对象添加到观察者对象列表中。当被观察者中的事件发生变化时执行 setChanged() 和 notifyObservers() 方法。

- setChange() 方法用来设置一个内部标志位注明数据发生了变化；
- notifyObservers() 方法会去调用观察者对象列表中所有的 Observer 的 update() 方法，通知它们数据发生了变化。

注意：只有在 setChange() 被调用后，notifyObservers() 才会去调用 update()。

3.6.2 观察者类

观察者类需要实现 Observer 接口的唯一方法 update()。

```
1 void update(Observable o, Object arg);
```

形参 Object arg 对应一个由 notifyObservers(Object arg) 传递来的参数，当执行的是 notifyObservers() 时，arg 为 null。

3.6.3 示例代码

被观察者 ServerManager

```
1 package ouc.cs.java.test.observer;
2
3 import java.util.Observable;
4
5 /**
6  * 被观察者
7  *
8  */
9 public class ServerManager extends Observable {
10
11     private int data = 0;
12
13     public int getData() {
14         return data;
15     }
16
17     public void setData(int i) {
18
19         /*
20          * 只有在setChanged()被调用后，notifyObservers()才会去调用update()，否则什么都不干
21          */
22         if (this.data != i) {
23             this.data = i;
24             setChanged();
25         }
26         notifyObservers();
27     }
28 }
```

观察者 ObserverA

```
1 package ouc.cs.java.test.observer;
2
3 import java.util.Observable;
4 import java.util.Observer;
5
6 /**
7  * 观察者A
8  *
9  */
10 public class ObserverA implements Observer {
```

```
12     public ObserverA(ServerManager sm) {
13         super();
14         sm.addObserver(this);
15     }
17     public void update(Observable arg0, Object arg1) {
18         System.out.println("ObserverA receive: Data has changed to " + ((ServerManager) arg0).getData());
19     }
20 }
```

观察者 ObserverB

```
1     package ouc.cs.java.test.observer;
3     import java.util.Observable;
4     import java.util.Observer;
6     /**
7      * 观察者A
8      *
9      */
10    public class ObserverB implements Observer {
12        public ObserverB(ServerManager sm) {
13            super();
14            sm.addObserver(this);
15        }
17        public void update(Observable arg0, Object arg1) {
18            System.out.println("ObserverB receive: Data has changed to " + ((ServerManager) arg0).getData());
19        }
20    }
```

测试主程序 TestJavaObserver

```
1     package ouc.cs.java.test.observer;
3     public class TestJavaObserver {
5         public static void main(String[] args) {
7             ServerManager sm = new ServerManager();
8             ObserverA a = new ObserverA(sm);
9             ObserverB b = new ObserverB(sm);
10            sm.setData(5);
11            sm.setData(8);
13            /*
14             * 注销观察者，以后被观察者有数据变化就不再通知这个已注销的观察者
15             */
16        }
```

```
16     sm.deleteObserver(a);  
17     sm.setData(10);  
18 }  
19 }
```