## 软件系统分析与设计

## Software System Analysis and Design

第一次作业

### 一、题目重述

一般地，面向对象分析与设计中存在三种基本事件处理的机制，除了普通的方法调用外，常常也会用到回调函数，而J2EE中还提供了一种基于监听方式的事件处理机制，请查阅资料，对Action 以及ActionListener的机制进行分析，完成一个分析示例。

同时，请将这三种方法或其它更多的事件处理方法在代码实现过程中的优劣进行比较和分析，并形成详细的分析总报告。

### 二、基于监听方式的事件处理机制

利用ActionListener进行监听的事件处理步骤一般如下：

首先需要实现一个实现了ActionListener接口的类，并实现其中的public void actionPerformed(ActionEvent e)方法，指定这一监听器在监听到事件后的行为。这个类可以单独定义，也可以是内部类或匿名类。

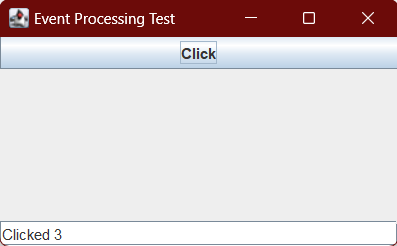
其次应创建自己实现的监听器类的对象，并将该对象添加到事件源，让事件源清楚事件对象应该交由谁处理。

如此操作之后，当用户执行某一操作，事件源将生成一个事件对象，并通知监听器对此事件进行处理。

以下是基于监听方式的事件处理的一个实例：

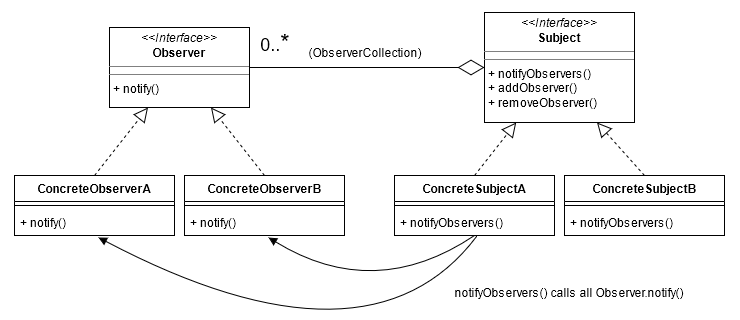
1. public class MyFrame extends JFrame {
2. private JButton myButton = new JButton("Click");
3. private JTextField myTextField = new JTextField("");
4. private ButtonListener myListener = new ButtonListener();
5. int click = 0;
6. public MyFrame(){
7. Container cp = this.getContentPane();
8. cp.setLayout(new BorderLayout());
9. cp.add(myButton,BorderLayout.NORTH);
10. cp.add(myTextField,BorderLayout.SOUTH);
11. myButton.addActionListener(myListener);
12. pack();
13. setTitle("Event Processing Test");
14. setVisible(true);
15. }
16. private class ButtonListener implements ActionListener{
17. @Override
18. public void actionPerformed(ActionEvent e) {
19. click++;
20. myTextField.setText("Clicked "+click);
21. myTextField.updateUI();
22. }
23. }
24. }

这个实例实现了一个按钮和一个文本框，当按下按钮时，文本框内将显示Clicked及当前点击的总次数。运行效果如下：



这个实例实现了一个内部类ButtonListener作为监听器，好处是可以直接使用JTextField对象而无需更多操作。实例中一个ButtonListener对象通过addActionListener方法注册到myButton，从而按钮被点击时，myButton产生一个Action并交由actionPerformed方法进行处理。

这种事件处理机制是观察者（Observer）模式的一种实现。观察者模式是一种软件设计模式，它的基本结构一般包括抽象目标（Subject），具体目标（ConcreteSubject），抽象观察者（Observer），具体观察者（ConcreteObserver）。在此种模式中，一个目标对象管理所有相依于它的观察者对象，并且在它本身的状态改变时主动发出通知。下图是观察者模式的关系图。



图中，抽象/具体观察者分别对应本例中的ActionListener和ButtonListener，其notify方法对应本例中的actionPerformed。抽象/具体目标对应本例中的JButton，addObserver、removeObserver对应本例中的addActionListener、removeActionListener。事件发生时，Subject通过notifyObservers通知观察者调用notify方法。

### 三、三种基本事件处理的机制

#### 1. 普通的方法调用

普通的方法调用的例子可见实例代码中对addActionListener的调用。这种方法一般是将调用位置的信息存入堆栈，转而执行被调用的方法，等待该方法执行完成后再继续执行原来的方法。

#### 2. 方法回调

方法回调是指，被调用的Class2中的方法又反过来调用Class1中的方法。这种方法中，Class2中的可以根据自己的需要对Class1进行操作，给予了功能实现更高的自由度。事实上，方法回调就是通过两次或以上的连续方法调用来进行的。方法回调中，Class1对象常常需要将自己的引用或接口引用传给Class2的方法，从而Class2才能实现对Class1中方法的回调。为了保证安全性，推荐创建一个包含必要方法的接口并在方法参数中使用接口，这样可以避免不必要数据或方法的泄露。

#### 3. 三种基本事件处理的机制的比较

监听方式的事件处理已经在上文有所讲述。通过生产-销售的具体实例，我们可以直观看到三种事件处理机制的异同。

假若某品牌现需要生产10000件商品并售卖，若采取普通的方法调用方式，它将需要等待10000件商品制作完成后再进入销售流程。若采取方法回调，它可以提供一个sell方法，让工厂自行根据仓库、资金等因素进行销售。工厂可以在制作任意件商品后调用sell方法将已制作的商品售出。若采取事件监听，我们可以雇佣数名监听/销售员，他们受工厂管理，当工厂生产一定数量的产品或达到某些条件，它将通知这些监听/销售员，监听/销售员利用sell方法卖掉产品。

相比之下，直接调用、方法回调、事件监听这三种方式实现难度递增，其中直接调用是最简单最基础的方式，后两者的实现还需要借助它。直接调用会造成线程的阻塞，而后两者可以用于并行处理。因此，复杂的事件处理经常采取后两种方式，即它们的适用范围逐渐增大。

### 参考文献

[1] 宋淼,袁兆山,陈刚等.Java事件处理机制中设计模式的分析[J/OL].合肥工业大学学报(自然科学版),2004(11):1383-1386.

[2] johnson\_moon. Java 事件监听原理及Demo实现[EB/OL] https://blog.csdn.net/johnson\_moon/article/details/53574470

[3] 下雨天不带伞boy. 基于监听和基于回调的事件处理机制[EB/OL] https://www.jianshu.com/p/79f9f8795a5d

[4] 永恒之魂. 观察者模式（Observer模式）详解[EB/OL] https://zhuanlan.zhihu.com/p/77275289

[5] 维基百科. 观察者模式 [DB/OL] https://zh.wikipedia.org/wiki/观察者模式