Model-View-Controller (MVC) Structure

Previous - Presentation-Model

Here the same calculator is organized according the the Model-View-Controller (MVC)pattern. The idea is to separate the user interface (the Presentation in the previous example) into a View (creates the display, calling the Model as necessary to get information), and Controller (responds to user requests, interacting with both the View and Controller as necessary). The literature on MVC leaves room for a number of variations, but they all follow this basic idea. This model is simple and can be used with simple method calls. If there are more complex interactions (eg, the Model is asynchronously updated), an Observer pattern (listeners) may be required.

An **excellent description of MVC** is Sun's article Java SE Application Design With MVC. Another description of MVC can be found at the beginning of the tutorial Applying the Model/View/Controller Design Paradigm in VisualAge for Java.

Main program

The main program initializes everything and ties everything together.

```
1  // structure/calc-mvc/CalcMVC.java -- Calculator in MVC pattern.
2  // Fred Swartz -- December 2004
3
4  import javax.swing.*;
5
6  public class CalcMVC {
7      //... Create model, view, and controller.  They are
8      //    created once here and passed to the parts that
9      //    need them so there is only one copy of each.
10     public static void main(String[] args) {
11
12         CalcModel      model      = new CalcModel();
13         CalcView       view       = new CalcView(model);
14         CalcController controller = new CalcController(model, view);
15
16         view.setVisible(true);
17     }
18 }
```

View

This View doesn't know about the Controller, except that it provides methods for registering a Controller's listeners. Other organizations are possible (eg, the Controller's listeners are non-private variables that can be referenced by the View, the View calls the Controller to get listeners, the View calls methods in the Controller to process actions, ...).

```
1  // structure/calc-mvc/CalcView.java - View component
2  //    Presentation only.  No user actions.
3  // Fred Swartz -- December 2004
4
5  import java.awt.*;
6  import javax.swing.*;
7  import java.awt.event.*;
8
9  class CalcView extends JFrame {
10     //... Constants
11     private static final String INITIAL_VALUE = "1";
12
```

```
13      //... Components
14      private JTextField m_userInputTf = new JTextField(5);
15      private JTextField m_totalTf     = new JTextField(20);
16      private JButton    m_multiplyBtn = new JButton("Multiply");
17      private JButton    m_clearBtn    = new JButton("Clear");
18
19      private CalcModel m_model;
20
21      //================================================== constructor
22      /** Constructor */
23      CalcView(CalcModel model) {
24          //... Set up the logic
25          m_model = model;
26          m_model.setValue(INITIAL_VALUE);
27
28          //... Initialize components
29          m_totalTf.setText(m_model.getValue());
30          m_totalTf.setEditable(false);
31
32          //... Layout the components.
33          JPanel content = new JPanel();
34          content.setLayout(new FlowLayout());
35          content.add(new JLabel("Input"));
36          content.add(m_userInputTf);
37          content.add(m_multiplyBtn);
38          content.add(new JLabel("Total"));
39          content.add(m_totalTf);
40          content.add(m_clearBtn);
41
42          //... finalize layout
43          this.setContentPane(content);
44          this.pack();
45
46          this.setTitle("Simple Calc - MVC");
47          // The window closing event should probably be passed to the
48          // Controller in a real program, but this is a short example.
49          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50      }
51
52      void reset() {
53          m_totalTf.setText(INITIAL_VALUE);
54      }
55
56      String getUserInput() {
57          return m_userInputTf.getText();
58      }
59
60      void setTotal(String newTotal) {
61          m_totalTf.setText(newTotal);
62      }
63
64      void showError(String errMessage) {
65          JOptionPane.showMessageDialog(this, errMessage);
66      }
67
68      void addMultiplyListener(ActionListener mal) {
69          m_multiplyBtn.addActionListener(mal);
70      }
71
72      void addClearListener(ActionListener cal) {
73          m_clearBtn.addActionListener(cal);
74      }
75  }
```

The Controller

The controller process the user requests. It is implemented here as an Observer pattern -- the Controller registers listeners that are called when the View detects a user interaction. Based on the user request, the Controller calls methods in the View and Model to accomplish the requested action.

```
1  // stucture/calc-mvc/CalcController.java - Controller
2  //    Handles user interaction with listeners.
3  //    Calls View and Model as needed.
4  // Fred Swartz -- December 2004
5
6  import java.awt.event.*;
7
8  public class CalcController {
9      //... The Controller needs to interact with both the Model and View.
10     private CalcModel m_model;
11     private CalcView  m_view;
12
13     //========================================================== constructor
14     /** Constructor */
15     CalcController(CalcModel model, CalcView view) {
16         m_model = model;
17         m_view  = view;
18
19         //... Add listeners to the view.
20         view.addMultiplyListener(new MultiplyListener());
21         view.addClearListener(new ClearListener());
22     }
23
24
25     ////////////////////////////////////////// inner class MultiplyListener
26     /** When a mulitplication is requested.
27      *  1. Get the user input number from the View.
28      *  2. Call the model to mulitply by this number.
29      *  3. Get the result from the Model.
30      *  4. Tell the View to display the result.
31      * If there was an error, tell the View to display it.
32      */
33     class MultiplyListener implements ActionListener {
34         public void actionPerformed(ActionEvent e) {
35             String userInput = "";
36             try {
37                 userInput = m_view.getUserInput();
38                 m_model.multiplyBy(userInput);
39                 m_view.setTotal(m_model.getValue());
40
41             } catch (NumberFormatException nfex) {
42                 m_view.showError("Bad input: '" + userInput + "'");
43             }
44         }
45     }//end inner class MultiplyListener
46
47
48     ////////////////////////////////////////// inner class ClearListener
49     /**  1. Reset model.
50      *   2. Reset View.
51      */
52     class ClearListener implements ActionListener {
53         public void actionPerformed(ActionEvent e) {
54             m_model.reset();
55             m_view.reset();
56         }
57     }// end inner class ClearListener
58 }
```

Model

The model is independent of the user interface. It doesn't know if it's being used from a text-based, graphical, or web interface. This is the same model used in the presentation example.

```
1  // structure/calc-mvc/CalcModel.java
2  // Fred Swartz - December 2004
3  // Model
4  //     This model is completely independent of the user interface.
5  //     It could as easily be used by a command line or web interface.
6
7  import java.math.BigInteger;
8
9  public class CalcModel {
10     //... Constants
11     private static final String INITIAL_VALUE = "0";
12
13     //... Member variable defining state of calculator.
14     private BigInteger m_total;  // The total current value state.
15
16     //================================================================ constructor
17     /** Constructor */
18     CalcModel() {
19         reset();
20     }
21
22     //====================================================================== reset
23     /** Reset to initial value. */
24     public void reset() {
25         m_total = new BigInteger(INITIAL_VALUE);
26     }
27
28     //================================================================ multiplyBy
29     /** Multiply current total by a number.
30     *@param operand Number (as string) to multiply total by.
31     */
32     public void multiplyBy(String operand) {
33         m_total = m_total.multiply(new BigInteger(operand));
34     }
35
36     //====================================================================== setValue
37     /** Set the total value.
38     *@param value New value that should be used for the calculator total.
39     */
40     public void setValue(String value) {
41         m_total = new BigInteger(value);
42     }
43
44     //====================================================================== getValue
45     /** Return current calculator total. */
46     public String getValue() {
47         return m_total.toString();
48     }
49 }
```

Copyleft 2004 Fred Swartz MIT License