# http://leepoint.net/notes-java/GUI/structure/40mvc.html

# Model-View-Controller (MVC) Structure

[Previous - Presentation-Model](http://leepoint.net/notes-java/GUI/structure/30presentation-model.html)

SimpleCalc-mvc

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](http://java.sun.com/developer/technicalArticles/javase/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](http://www.javadude.com/articles/vaddmvc1/mvc1.htm).

### Main program

The main program initializes everything and ties everything together.

// structure/calc-mvc/CalcMVC.java -- Calculator in MVC pattern.

// Fred Swartz -- December 2004

import javax.swing.\*;

public class CalcMVC {

//... Create model, view, and controller. They are

// created once here and passed to the parts that

// need them so there is only one copy of each.

public static void main(String[] args) {

CalcModel model = new CalcModel();

CalcView view = new CalcView(model);

CalcController controller = new CalcController(model, view);

view.setVisible(true);

}

}

## 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, ...).

// structure/calc-mvc/CalcView.java - View component

// Presentation only. No user actions.

// Fred Swartz -- December 2004

import java.awt.\*;

import javax.swing.\*;

import java.awt.event.\*;

class CalcView extends JFrame {

//... Constants

private static final String INITIAL\_VALUE = "1";

//... Components

private JTextField m\_userInputTf = new JTextField(5);

private JTextField m\_totalTf = new JTextField(20);

private JButton m\_multiplyBtn = new JButton("Multiply");

private JButton m\_clearBtn = new JButton("Clear");

private CalcModel m\_model;

//==================================== constructor

/\*\* Constructor \*/

}

public CalcView(CalcModel model) {

//... Set up the logic

m\_model = model;

m\_model.setValue(INITIAL\_VALUE);

//... Initialize components

m\_totalTf.setText(m\_model.getValue());

m\_totalTf.setEditable(false);

//... Layout the components.

JPanel content = new JPanel();

content.setLayout(new FlowLayout());

content.add(new JLabel("Input"));

content.add(m\_userInputTf);

content.add(m\_multiplyBtn);

content.add(new JLabel("Total"));

content.add(m\_totalTf);

content.add(m\_clearBtn);

//... finalize layout

this.setContentPane(content);

this.pack();

this.setTitle("Simple Calc - MVC");

// The window closing event should probably be passed to the

// Controller in a real program, but this is a short example.

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

void reset() {

m\_totalTf.setText(INITIAL\_VALUE);

}

String getUserInput() {

return m\_userInputTf.getText();

}

void setTotal(String newTotal) {

m\_totalTf.setText(newTotal);

}

void showError(String errMessage) {

JOptionPane.showMessageDialog(this, errMessage);

}

void addMultiplyListener(ActionListener mal) {

m\_multiplyBtn.addActionListener(mal);

}

void addClearListener(ActionListener cal) {

m\_clearBtn.addActionListener(cal);

}

}

## 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.

// stucture/calc-mvc/CalcController.java - Controller

// Handles user interaction with listeners.

// Calls View and Model as needed.

// Fred Swartz -- December 2004

import java.awt.event.\*;

public class CalcController {

//... The Controller needs to interact with both the Model and View.

private CalcModel m\_model;

private CalcView m\_view;

//============================================= constructor

/\*\* Constructor \*/

CalcController(CalcModel model, CalcView view) {

m\_model = model;

m\_view = view;

//... Add listeners to the view.

view.addMultiplyListener(new MultiplyListener());

view.addClearListener(new ClearListener());

}

///////////////////////////////// inner class MultiplyListener

/\*\* When a mulitplication is requested.

\* 1. Get the user input number from the View.

\* 2. Call the model to mulitply by this number.

\* 3. Get the result from the Model.

\* 4. Tell the View to display the result.

\* If there was an error, tell the View to display it.

\*/

class MultiplyListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String userInput = "";

try {

userInput = m\_view.getUserInput();

m\_model.multiplyBy(userInput);

m\_view.setTotal(m\_model.getValue());

} catch (NumberFormatException nfex) {

m\_view.showError("Bad input: '" + userInput + "'");

}

}

}//end inner class MultiplyListener

////////////////////////////////// inner class ClearListener

/\*\* 1. Reset model.

\* 2. Reset View.

\*/

class ClearListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

m\_model.reset();

m\_view.reset();

}

}// end inner class ClearListener

}

## 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.

// structure/calc-mvc/CalcModel.java

// Fred Swartz - December 2004

// Model

// This model is completely independent of the user interface.

// It could as easily be used by a command line or web interface.

import java.math.BigInteger;

public class CalcModel {

//... Constants

private static final String INITIAL\_VALUE = "0";

//... Member variable defining state of calculator.

private BigInteger m\_total; // The total current value state.

//=================================================== constructor

/\*\* Constructor \*/

CalcModel() {

reset();

}

//========================================================== reset

/\*\* Reset to initial value. \*/

public void reset() {

m\_total = new BigInteger(INITIAL\_VALUE);

}

//=================================================== multiplyBy

/\*\* Multiply current total by a number.

\*@param operand Number (as string) to multiply total by.

\*/

public void multiplyBy(String operand) {

m\_total = m\_total.multiply(new BigInteger(operand));

}

//======================================================== setValue

/\*\* Set the total value.

\*@param value New value that should be used for the calculator total.

\*/

public void setValue(String value) {

m\_total = new BigInteger(value);

}

//============================================== getValue

/\*\* Return current calculator total. \*/

public String getValue() {

return m\_total.toString();

}

}