Java

Model-View-Controller design pattern

The MVC pattern

- MVC stands for Model-View-Controller
- The Model is the actual internal representation
- The View (or a View) is a way of looking at or displaying the model
- The Controller provides for user input and modification

The Model

- Most programs are supposed to do work, not just be "another pretty face"
  - but there are some exceptions
  - useful programs existed long before GUIs
- The Model is the part that does the work
- The Model should be independent of the GUI
- Independence gives flexibility, robustness
The Controller

- A GUI lets the user control what work the program is doing
- The design of the GUI depends on the Model...
- ...but the Model should not depend on the GUI
- Unless the Model (what the program does) is the GUI, these can always be separated
- Java's controls are Buttons, TextFields, etc.

The View

- The user has to be able to see, or view, what the program is doing
- The Model should be independent of the View (but it can provide access methods)
- The View should not display what the Controller thinks is happening

Combining the Controller and View

- Sometimes the Controller and View are combined, especially in small programs
- Combining the Controller and View is appropriate if they are very interdependent
- The Model should still be independent
- *Never* mix Model code with GUI code!

Separation of concerns

- As always, you want code independence
- The Model should not be contaminated with control code or display code
- The View should represent the Model as it really is, not some remembered status
- The Controller should talk to the Model and View, not manipulate them
The Bouncing Ball Applet

- Each click of the Step button advances the ball a small amount
- The step number and ball position are displayed in the status line

The Ball Applet: Model

- The Ball Applet shows a ball bouncing in a window
- The Model controls the motion of the ball
- To know when to bounce, the Model must know the size of the window
- The Model doesn’t need to know anything else about the GUI

Sample CRC index card

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Set initial position</td>
<td>No collaborators...</td>
</tr>
<tr>
<td></td>
<td>Move one step</td>
<td>…but allow access from View</td>
</tr>
</tbody>
</table>
Model I

class Model {
    final int BALL_SIZE = 20;
    int xPosition = 0;
    int yPosition = 0;
    int xLimit, yLimit;
    int xDelta = 6;
    int yDelta = 4;
    // more...
}

Model II

void makeOneStep () {
    xPosition += xDelta;
    if (xPosition < 0) {
        xPosition = 0;
        xDelta = -xDelta;
    }
    // more...
}

Model III

if (xPosition >= xLimit) {
    xPosition = xLimit;
    xDelta = -xDelta;
} // still more...

Model IV

yPosition += yDelta;
if (yPosition < 0 || yPosition >= yLimit) {
    yDelta = -yDelta;
    yPosition += yDelta;
} // end of makeOneStep method
} // end of Model class
Model (repeated)

Model
• Set initial position
• Move one step
• No collaborators...
• …but allow access from View

The Ball Applet: View

• The View needs access to the ball’s state (in this case, it’s x-y location)
• For a static drawing, the View doesn’t need to know anything else

View

View
• Paint the ball
• Access Model

View 1

class View extends Canvas {
    Controller controller;
    Model model;
    int stepNumber = 0;
    // more...
public void paint(Graphics g) {
    g.setColor(Color.red);
    g.fillOval(model.xPosition, model.yPosition,
              model.BALL_SIZE, model.BALL_SIZE);
    controller.showStatus("Step " +
              (stepNumber++) +
              ", x = " + model.xPosition +
              ", y = " + model.yPosition);
} // end paint method

The Ball Applet: Controller

- The Controller tells the Model what to do
- The Controller tells the View when it needs to refresh the display
- The Controller doesn’t need to know the inner workings of the Model
- The Controller doesn’t need to know the inner workings of the View

Controller

- Create Model
- Create View
- Give View access to Model
- Tell Model to advance
- Tell View to repaint

View (repeated)

View

- Paint the ball
- Access Model
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Controller extends Applet {
    Panel buttonPanel = new Panel();
    Button stepButton = new Button("Step");

    Model model = new Model();
    View view = new View();
    // more...
}

Controller II

public void init () {
    // Lay out components
    setLayout (new BorderLayout ());
    buttonPanel.add (stepButton);
    this.add (BorderLayout.SOUTH, buttonPanel);
    this.add (BorderLayout.CENTER, view);
    // more...
}

Controller III

// Attach actions to components
stepButton.addActionListener
    (new ActionListener () {
        public void actionPerformed
            (ActionEvent event) {
            model.makeOneStep ();
            view.repaint ();
        }});
    // more...

Controller IV

// Tell the View about myself (Controller) and
// about the Model
view.model = model;
view.controller = this;
}
// end init method

// more...
public void start () {
    model.xLimit =
        view.getSize ().width - model.BALL_SIZE;
    model.yLimit =
        view.getSize ().height - model.BALL_SIZE;
    repaint ();
} // end of start method

} // end of Controller class

Controller (repeated)

Controller
• Create Model
• Create View
• Give View access to Model
• Tell Model to advance
• Tell View to repaint

Model
• View