CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2012
Doc 22 Value Object, MVC Patterns
April 24 2012

Copyright ©, All rights reserved. 2012 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.s.shtml) license defines the copyright on this document.
References

Pattern-Oriented Software Architecture: A System of Patterns v1, Buschman, Meunier, Rohnert, Sommerlad, Stal, 1996, pp 125-143

Patterns of Enterprise Application Architecture, Folwer, 2003, pp 330-386

Core J2EE Patterns: Best Practices and Design Strategies, 2nd, Alur, Crupi, Malks, 2003

Values in Object Systems, Baumer, Riehle,
Value Object
Values versus Objects

Values

- integers,
- real numbers,
- strings

Objects

- new Person("Roger")

No alterable state

- No side effects

Alterable state

- Side effects

One 5 is the same as all 5's

- Pointer equality
Values in Programs

social security numbers
credit card numbers
money
date
account numbers
width
height
weight
colors

Can use primitive types (ints, float) for value, but ...

Model abstractions from problem domain

Often

Measurements
Identifiers

Tuesday, April 24, 12
Money Example

int bankBalance = 5;

But what about
   Different currencies
   Rounding errors
Money Example

So make a Money class

But then have side effects
Value Object Pattern

For values in applications that need more than primitive types

Create a class for the abstraction

Make the objects immutable
MVC & Related Web Patterns
Model-View-Controller (MVC)

Context

Interactive application with human-computer interface

Forces

Same data may be displayed differently

Display & application must reflect data changes immediately

UI changes should be easy and even possible at runtime

Changing look & feel or port to other platforms should not affect core application code
Solution

Divide application into three parts:

Model (core application)
View (display, output)
Controller (user input)
**Model**

Core application code

Contains a list of observers (view or controller)

Has a broadcast mechanism to inform views of a change

Same mechanism as subject in Observer pattern
View

Displays information to user

Obtains data from model

Each view has a controller
Controller

Handles input from user as events

- Keystrokes
- Mouse clicks
- Mouse movements

Maps each event to proper action on model and/or view

Many people misinterpret what a controller does
Structure

Observer
update

View
myModel
myController
initialize(Model)
makeController
activate
display
update

Controller
myModel
myView
initialize(Model,View)
handleEvent
update
call service

Model
coreData
observers
attach(Observer)
detach(Observer)
notify

update
data
get
(service)

View + Controller

Make up the user interface

Some GUI frameworks combine these

VW Smalltalk contains both, but hides controller from programmer
### Some Existing Smalltalk Controllers & Views

<table>
<thead>
<tr>
<th>Controllers</th>
<th>Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationDialogController</td>
<td>ActionButtonView</td>
</tr>
<tr>
<td>BasicButtonController</td>
<td>AutoScrollingView</td>
</tr>
<tr>
<td>ClickWidgetController</td>
<td>BasicButtonView</td>
</tr>
<tr>
<td>ColoredAreaController</td>
<td>BooleanWidgetView</td>
</tr>
<tr>
<td>ComboBoxButtonController</td>
<td>CheckButtonView</td>
</tr>
<tr>
<td>ComboBoxInputBoxController</td>
<td>ClickWidget</td>
</tr>
<tr>
<td>ComboBoxListController</td>
<td>ComboBoxButtonView</td>
</tr>
<tr>
<td>ControllerWithMenu</td>
<td>ComboBoxInputFieldView</td>
</tr>
<tr>
<td>ControllerWithSelectMenu</td>
<td>ComboBoxListView</td>
</tr>
<tr>
<td>DataSetController</td>
<td>ComposedTextView</td>
</tr>
<tr>
<td>DataSetControllerProxy</td>
<td>DataSetView</td>
</tr>
<tr>
<td>DelayingWidgetController</td>
<td>DefaultLookCheckButtonView</td>
</tr>
<tr>
<td>DrawingController</td>
<td>DefaultLookRadioButtonView</td>
</tr>
<tr>
<td>DropDownListController</td>
<td>EmulationScrollBar</td>
</tr>
<tr>
<td>EmulatedDataSetController</td>
<td>GeneralSelectionTableView</td>
</tr>
<tr>
<td>EmulatedSequenceController</td>
<td>HierarchicalSequenceView</td>
</tr>
<tr>
<td>EmulationScrollBarController</td>
<td>HorizontalTabBarView</td>
</tr>
<tr>
<td>HierarchicalSequenceController</td>
<td>HorizontalTopTabBarView</td>
</tr>
<tr>
<td>InputBoxController</td>
<td>InputFieldView</td>
</tr>
</tbody>
</table>
Web related Patterns
The Patterns

Template View
Page Controller
Front Controller
Intercepting Filter
Composite View
Transform View
Template View

Renders information into HTML by embedding markers in an HTML page

Server Pages
Java
  <html>
  <body>
  <%= int x = 1; %>
  <%= int y = 2; %>
  If we add <%= x %> to <%= y %> we will get <%= x + y %>
  </body>
  </html>

PHP, Smalltalk Server pages
Template View

Advantage

Graphic designers can generate view

Rapid development for small projects

Disadvantages

Poor module structure

Leads to mixing model, controller and view logic

Leads to repeated code in files

Many programming tools do not work on template files
Template View - Some common Issues

Conditional display

<p>Please pay your bill
<If user.isDeadBeat()><B> </If>
now.
<If use.isDeadBeat()><B> </If></p>

Iteration over collection

Given a list create a drop down menu

Use View Helper to separate out processing logic
Some Background
public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
Smalltalk Example

doGet: aRequest response: aResponse

    aResponse write: '<HTML><BODY>GET<BR>
                     Hello world</BODY></HTML>\n'.

doPost: aRequest response: aResponse

    aResponse write: '<HTML><BODY>POST<BR>
                    Hello world</BODY></HTML>\n'.

26

Tuesday, April 24, 12
Page & Front Controller
Page Controller

An object that handles a request for a specific page or action on a Web page.

Decodes URL

Extracts all form data and gets all data for the action

Create and invoke model objects, pass all relevant data to model

Determine which view should display the result page and forward model information to it

Each page or url on the site has a different page controller.
Front Controller

A controller that handles all requests for a Web site

Forces

Avoid duplicate control logic
Apply common logic to multiple requests
Separate system processing logic from view
Have a centralized controlled access point into system
How it works

All requests to a Web site are directed to the FrontController

The FrontController
- Examines the URL & form data
- Determines the correct command to handle the request
- Create the correct command
- Forwards the request to the command

Command is part of controller so it uses a separate view
Pros & Cons

Disadvantage

More complex than Page Controller

Advantages

Only one controller has to be configured into the web server

A command object handles only one request so command does not have to be thread-safe

Commands can be added dynamically (if controller uses reflection to create a command object)

Factor out common code from multiple Page Controllers
Intercepting Filter
Composite View
Transform View
Intercepting Filter

You want to manipulate a request and a response before and after the request is processed

Forces

You want

- Common processing across requests like
  - Logging
  - Compressing
  - Data encoding

Pre & post processing components loosely coupled with core request-handling services

Pre & post processing components independent of each other

Solution

Add a chain of decorators (filters) that end on the Front Controller
Composite View

Build a view from atomic components while managing content and layout independently

Forces

You want subview, such as headers, footers and tables reused in different pages

You want to avoid directly embedding and duplicating subviews in multiple pages

You have content in subviews that frequently change or are subject to access control

Solution

Use the composite pattern on views.

A page then is created as a composite object of views.
Transform View

A view that processes domain data elements by element and transforms them into HTML

Given a domain object, MusicAlbum, how to generate a web page for the object?

Use Template View
Convert object into html
Converting object into html

One could add toHtml to the object

    MusicAlbum ragas = new MusicAlbum.find("Passages");
    String html = ragas.toHtml();

But
    Domain object is coupled to view language
    Provides only one way to display object
Using Transforms

Use XML and XSLT

    Convert domain object to XML

    Use XSLT to convert XML into HTML

Now we can produce many different views of the object without changing the object

More complex that converting object to HTML
Transform View Verses Template View

Template View

More tools support

No language to learn

Transform View

Easier to avoid domain logic in view

Testing can be done without Web server

Easier to make global changes to Web site