Chapter: Compound Patterns - Introduction

- Text initial example
  - Iteratively develops an implementation for the duck class
  - By end of development, have a duck interface, can count how many times ducks quack, can manage flocks of ducks.
  - Patterns incorporated:
    1. Adapter, so can deal with geese too.
    2. Decorator, to count number of quacks.
    3. Abstract Factory, to insure all ducks are decorated.
    4. Composite, to manage flocks.
    5. Observer, so bird watchers can monitor individual birds.

- Example demonstrates
  - How a number of patterns can be incorporated into a single implementation
  - That is OK to vary from a strict pattern definition in order to fit a particular problem

- The above example is not a compound pattern!
  - Rather, it is just a set of patterns working together

- Compound Pattern defined:
  
  A set of patterns working together that can be applied to many problems.

- A compound pattern is a general solution to a problem

- Note that the definition is the same as that for Design Pattern, but applied to several patterns working together
Chapter: Compound Patterns - Model-View-Controller (MVC) Compound Pattern

- Consider a system composed of
  
  1. A system that represents the main set of methods and objects with which the user wants to interact (the Model)
     - Holds data, state, application logic
  2. A display of some sort (the View)
     - Provides a representation of the model to the user
  3. A component/behavior that allows communication between the View and Model (the Controller)

- The flow of control is modeled as
  
  1. The user uses the View to perform some action on the Model
  2. The action is relayed to the Controller
  3. The Controller makes a request to the Model to perform the action
  4. The Model informs the View when its state has changed as a result of performing the action

- Flow of control:
• Note that
  – The Controller can ask the View to update its display
  – The View can directly query the Model for its state

• Such a system can be implemented using

  1. Strategy Pattern
     – In terms of the View and Controller
     – The Controller represents the behavior of the View

  2. Observer Pattern
     – In terms of the View and Model
     – The Model is the observable, the View is the observer
     – The Controller can be an observer also

• While could have View communicate directly with Model (i.e., eliminate Controller), do not want to because

  – This would give View two responsibilities, violating Single Responsibility Principle
  – Do not want such tight coupling between View and Model
    * This would preclude using a different Model
public interface ModelInterface {
    ... action1(...);
    ... action2(...);
    ...
    void initialize (...);
    ...
    void registerObserver(ModelObserver o);
    void removeObserver(ModelObserver o);
    ...
}

public class Model implements ModelInterface {
    ArrayList modelObservers = new ArrayList();
    State curState; // abstraction of what View is interested in
    ...
    ...
    ... action1(...) { // Methods that implement Model logic
    ...
    }
    ...
    ...
    ...
    ...
    ...
    void initialize (...) {
        // initialize model
    }
    ...
    void registerObserver (ModelObserver m) { // Observer implementation
        modelObservers.add(m);
    }
    void removeObserver (ModelObserver m) {
        int i = modelObservers.indexOf(m);
        if (i >= 0)
            modelObservers.remove(i);
    }
    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer = (Observer)modelObservers.get(i);
            observer.update(curState);
        }
    }
    public void measurementsChanged () {
        notifyObservers();
    }
    public void setState(...) {
        ...
        measurementsChanged();
    }
    }
}
public interface ModelObserver {
    public void updateState(...);
}

public class View implements ModelObserver {
    ModelInterface model;
    ControllerInterface controller;
    State modelState;

    public View (ControllerInterface controller, ModelInterface model) {
        this.controller = controller;
        this.model = model;
        model.registerObserver (this);
    }

    public void updateDisplay (...) { //Display state of Model
        // create/update user interface
    }

    public void createControls (...) { //Display control devices
        // create/update controls accessible to user
    }

    // various enabling and disabling methods for controls

    public void performAction (...) { // Relays requests to controller
        if // action1()
            controller.doAction1(...);
        else if // action2()
            controller.doAction2(...);
    }

    public void update (State state) { // State pushed by Model
        modelState = state;
        ...  
        processPushedState();
    }

    public ... processPushedState() {
        // process model state
        updateDisplay();
    }

    ...
public interface ControllerInterface {
    void start();
    void stop();
    ... doAction1(...);
    ... doAction2(...);
    ... doAction3(...);
    ...
}

public class Controller implements ControllerInterface {
    Model model;
    View view;
    ...

    public Controller (ModelInterface model) {
        this.model = model;
        view = new View(this, model);
        view.updateDisplay();
        view.createControls();
        ...
        model.initialize(...);
    }

    public void start () {
        ...
    }

    public void stop () {
        ...
    }

    public ... doAction1 (...) {
        ...
        model.action1(...);
    }

    public ... doAction2 (...) {
        ...
        model.action2(...);
    }
}
Chapter: Compound Patterns - MVC Sample Code: Driver

```java
public class MVCDriver {
    public static void main (String[] args) {
        ModelInterface model = new Model();
        ControllerInterface interface = new Controller(model);
    }
}
```
Chapter: Compound Patterns - MVC Model 2

- Model 2 is MVC adapted for use with the Internet
- It implements the client/server model
- Flow of control:
  1. Browser makes an HTTP request to a servlet
     - Data usually sent as a form
  2. Servlet processes request, which usually results in creation of a JavaBean
     - JavaBean is sent to the model - usually a database
  3. The servlet forwards control to a JSP
     - This generates a page that represents the view of the database
     - This view is provided by the JavaBean
  4. The page is sent to the browser, where it is displayed

- Comparing this to the MVC Pattern
  1. The servlet is the Controller
  2. The database is the Model (nothing really different here)
  3. The JSP is the View
Chapter: Compound Patterns - MVC m2 Example

- There are three steps needed to implement MVC m2

  1. Adapt the *Model* for use with MVC m2
     - In the standard MVC, both *View* and *Controller* are transparent from the *Model*
     - Consequently, the *Model* needs no refactoring

  2. Create a servlet
     - The servlet receives an HTTP request and translates it into actions to be applied to the *Model*

  3. Create a JSP
     - The JSP receives a JavaBean from the servlet
     - This JavaBean represents the *model*
     - The JSP extracts what it needs
public class ViewServlet extends HttpServlet {

    public void init () throws ServletException {
        Model model = new Model();
        model.initialize();
        getServletContext().setAttribute("model", model);
    }

    //doPost method

    public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        Model model = (Model) getServletContext().getAttribute("model");

        ... param = request.getParameter("XXX"); // See if there is a value for this tag
        if (param != null) {
            model.XXXmethod(...); // Call model method assoc'd with this tag
        }

        ... //Other checks for tags

        request.setAttribute("model", model); //Send JSP a bean with Model in it

        RequestDispatcher dispatcher = request.getRequestDispatcher("/jsp/View.jsp");
        dispatcher.forward(request, response);
    }
}
Chapter: Compound Patterns - MVC m2 Example: JSP

<jsp:useBean id="model" scope="request" class="Model"/>

<html>
<head>
<title>Model</title>
</head>
<body>
...
XXX = <jsp:getProperty name="model" property = XXX" />
//Extract property to be displayed
...
<form method="post" action="model/servlet/Model">
<input type="submit" name="XXX" value="YYY"> //Actions available to user
...
</form>

</body>
</html>
Chapter: Compound Patterns - MVC m2 v Standard MVC

- View not explicitly registered as an observer of the Model
  - Is informed of state changes by Controller
  - Receives a Bean from which the Model state can be extracted
  - Only needs to be informed when an HTTP response is generated

- Strategy not explicit
  - Servlet (Controller) represents the Strategy object
  - Not composed with View as in standard MVC
  - It still represents the behavior of the View, and can be swapped out