Patterns / J2EE Patterns

Overview

- A ***pattern*** is “A plan, diagram, or model to be followed in making things.”
  - Leonardo Da Vinci’s Notebook captures some of the patterns used in his time for civil engineering.
  - The inspiration for software design patterns stems from the documentation of architectural patterns used in Civil Engineering and Architecture in the classic work by Christopher Alexander, *A Pattern Language: Towns, Buildings, Construction* (Alexander, Ishikawa, and Silverstein 1977, Oxford Press).
- This book strongly influenced the book ***Design Patterns Elements of Reusable Object-Oriented Software*** (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1995, Addison-Wesley), by the so-called Gang of Four (GOF). The Gang of Four (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) are internationally recognized experts in the field of Object Oriented Technology.
- Dr. John Vlissides currently serves as the series editor for ***The Software Pattern Series*** **(**Addison-Wesley publications).

Defining a Pattern

- Patterns are about communicating problems and solutions.
  - Simply put, patterns enable us to document a known recurring problem and its solution in a particular context, and to communicate this knowledge to others.
  - One of the key elements in the previous statement is the word recurring, since the goal of the pattern is to foster conceptual reuse over time.

Categorizing Patterns

- Patterns, then, represent expert solutions to recurring problems in a context and thus have been captured at many levels of abstraction and in numerous domains.
- Numerous categories have been suggested for classifying software patterns, with some of the most common being:
  - Design patterns
  - Architectural patterns
  - Analysis patterns
  - Creational patterns
  - Structural patterns
  - Behavioral patterns

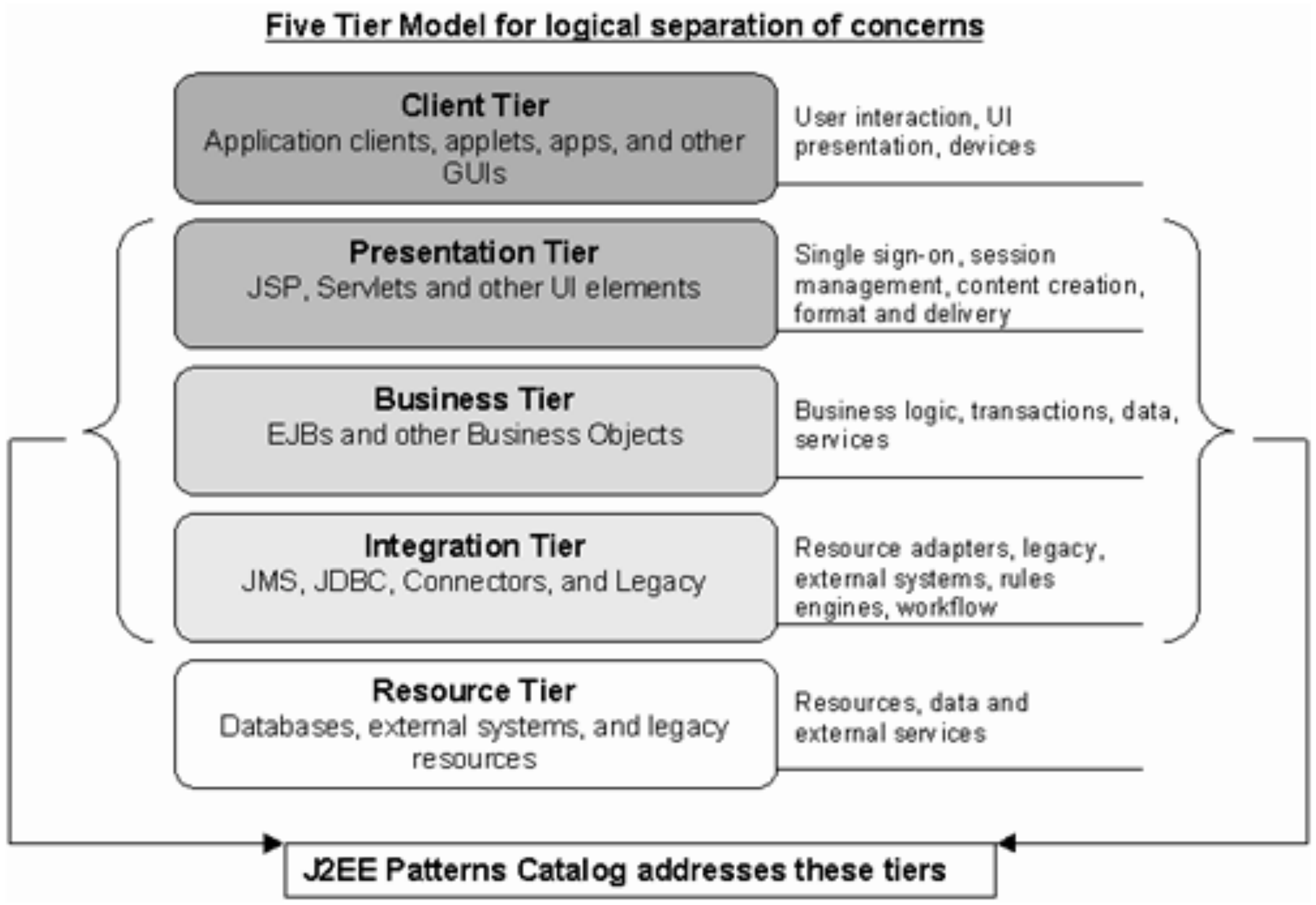
J2EE patterns

- Each J2EE pattern hovers somewhere between **a design pattern and an architectural pattern**, while the strategies document portions of each pattern point to a lower level of abstraction.

- A simple scheme in J2EE is to classify each pattern within one of the following logical architectural tiers:

Core J2EE™ Patterns: Best Practices and Design Strategies
By Deepak Alur, John Crupi, Dan Malks

Tier Model


Five Tier Model for logical separation of concerns
Client Tier
Application clients, applets, apps, and other GUIs
User interaction, UI presentation, devices
Presentation Tier
JSP, Servlets and other UI elements
Single sign-on, session management, content creation, format and delivery
Business Tier
EJBs and other Business Objects
Business logic, transactions, data, services
Integration Tier
JMS, JDBC, Connectors, and Legacy
Resource adapters, legacy, external systems, rules engines, workflow
Resource Tier
Databases, external systems, and legacy resources
Resources, data and external services
J2EE Patterns Catalog addresses these tiers

Pattern Organization

- The patterns are usually organized into the following sections:
  - **Problem -** This section describes the problem that the pattern addresses. For most patterns, the problem is introduced in terms of a concrete example. After presenting the problem in the example, the Context section suggests a design solution to the problem.
  - **Forces** —Describes the considerations you need to take into account while documenting the pattern. These considerations include environmental, linguistic, organizational, and platform issues. Recognizing forces that cause the problem is an extremely complex process.
  - **Implementation** —Deals with the solution to the problem in context.
  - **Strategies** —Describes the various collaborations that can be implemented in using the pattern apart from the regular way.
  - **Results** —Includes any issues that still need to be resolved after the pattern is applied.
  - **Sample code** —Describes through a skeleton sample code of how to implement the pattern or sometimes the class diagram is presented on how the pattern can be modeled or designed.
  - **Related patterns** —Describes other patterns that are related to this pattern. Sometimes you can use a combination of patterns to resolve a recurring problem. You might even form a new pattern.

Front Controller Pattern

- **Problem**
  - You want a **centralized access point** for presentation-tier request handling.
  - Without a central access point, **control code** that is common across multiple requests is **duplicated** in numerous places, such as within multiple views.
  - **When control code is intermingled** with view-creation code, **the application is less modular and cohesive**.
  - Additionally, having control code in numerous places is **difficult to maintain** and a single code change might require changes be made in numerous places.

Front Controller Pattern - 2

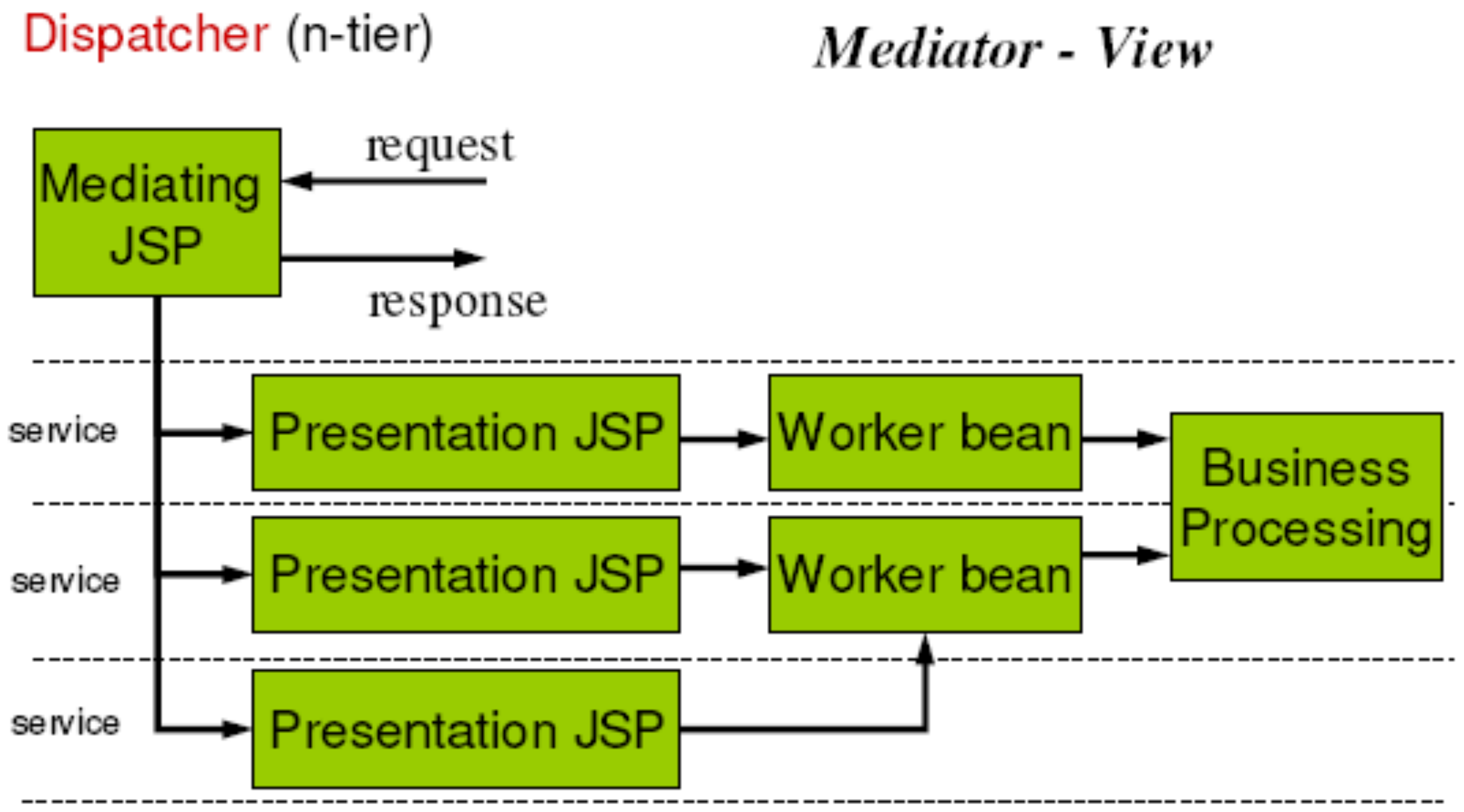
Forces

- You want to **avoid duplicate control logic.**
- You want **to apply common logic** to multiple requests.
- You want to **separate system processing logic from the view**.
- You want to **centralize controlled access points** into your system.

Front Controller Pattern - 3

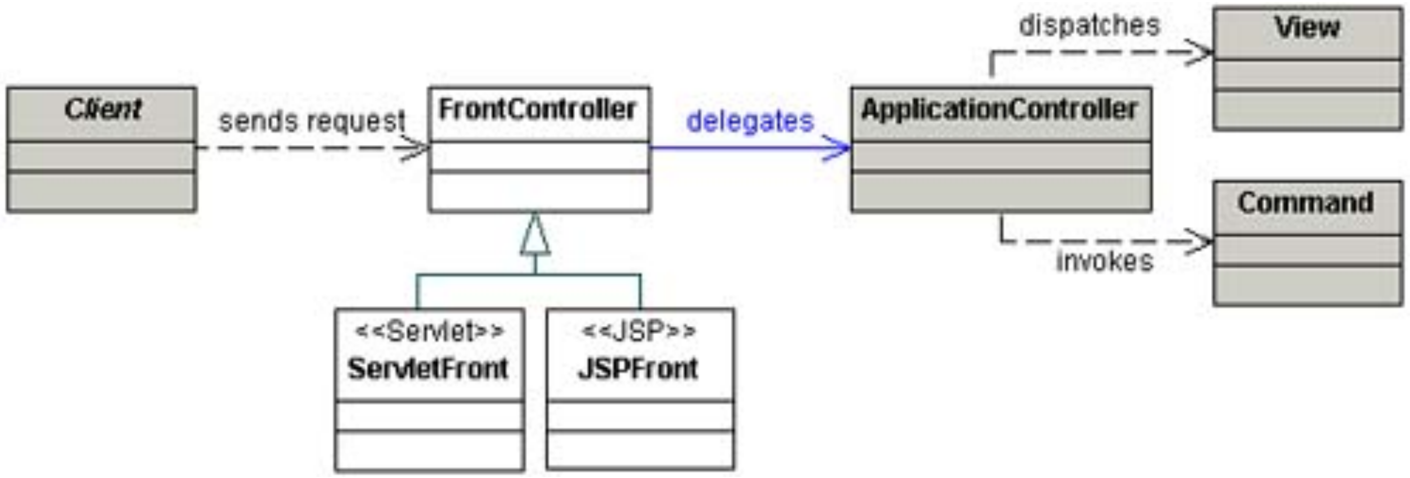
- **Solution**
  - Use a Front Controller as the initial point of contact for handling all related requests.
  - The Front Controller provides a centralized entry point for handling requests.
  - This pattern is similar to the Intercepting Filter because they both factor out and consolidate common presentation-tier control logic.
  - A Front Controller typically uses an Application Controller , which is responsible for action and view management.
    - **Action management** refers to locating and routing to the specific actions that will service a request.
    - **View management** refers to finding and dispatching to the appropriate view. While this behavior can be folded into the Front Controller, partitioning it into separate classes as part of an Application Controller improves modularity, maintainability, and reusability.

JSP patterns: Front Controller



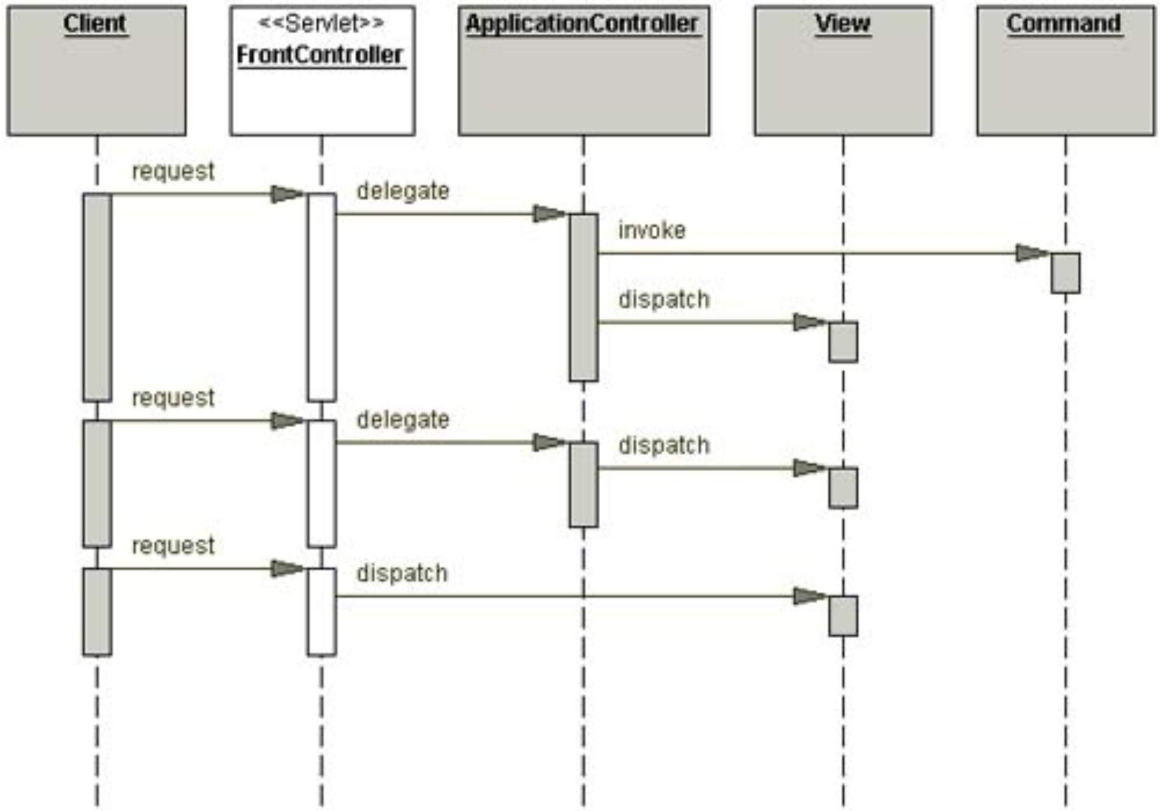
Front Controller Pattern - 4

- Front Controller class diagram



Front Controller Pattern - 5

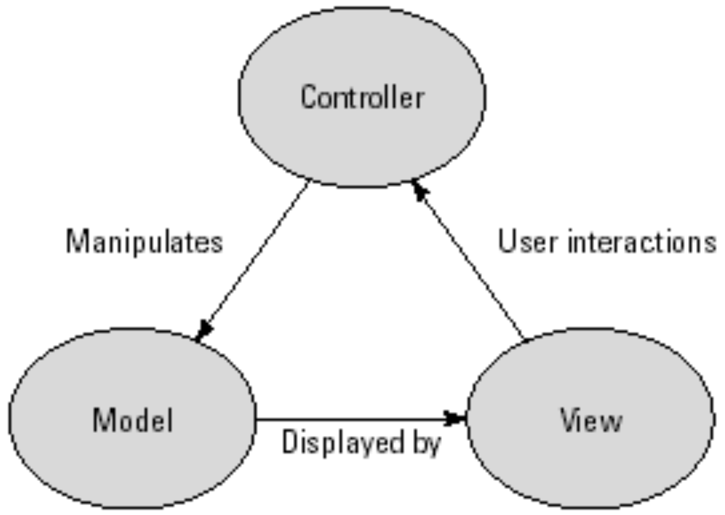
- Sequence Diagram for Front Controller



JSP Model

Model-View-Controller Pattern

- The MVC paradigm provides a pattern for separating the presentation logic (view), business logic (control), and data objects (model).



- J2EE’s architecture maps onto the MVC nicely.

The Model

- The *M* in MVC refers to the data object model.
  - For example, in an airline ticketing service you may have the concept of a booking, which in the real world is represented by a paper ticket.
- The model deals with issues such as how the required data is represented within the software system, where it is persisted, and how it is accessed.
  - For example, the booking may be held within a relational database within a table named Bookings with the fields PassengerName, DepartureCity, DestinationCity, TravelDate, and DepartureTime. This data may be accessed via JDBC using a Web component (Java Program, Beans, EJB, …)

The View

- The *V* in MVC is responsible for presentation issues.
- It handles how the client will see the application, and so HTML issues are usually dealt with here. However, other markup languages such as Wireless Markup Language (WML) and Extensible Markup Language (XML) are increasingly being used to support more varied types of clients.
  - The Booking example may be displayed in various ways. For example, on a wireless device only the most relevant information might be displayed due to the limited screen size.
  - In fact, the term *view* may be misleading, implying that it is meant for visual display only; the view may also be used to present the model via an audio interface if desired.
- The method in which the model is presented is abstracted from the underlying data.

The Control

- The *C* in MVC refers to the control part of the paradigm and deals with the business logic of the application.
- It handles how and when a client interacting with the view is able to access the model.
- The control layer usually interacts with authorization and authentication services, other J2EE services, and external systems to enforce the business rules to be applied to the application.
  - In our Booking example, the control would determine whether the view can actually display the model (loggin, validation, booking …)

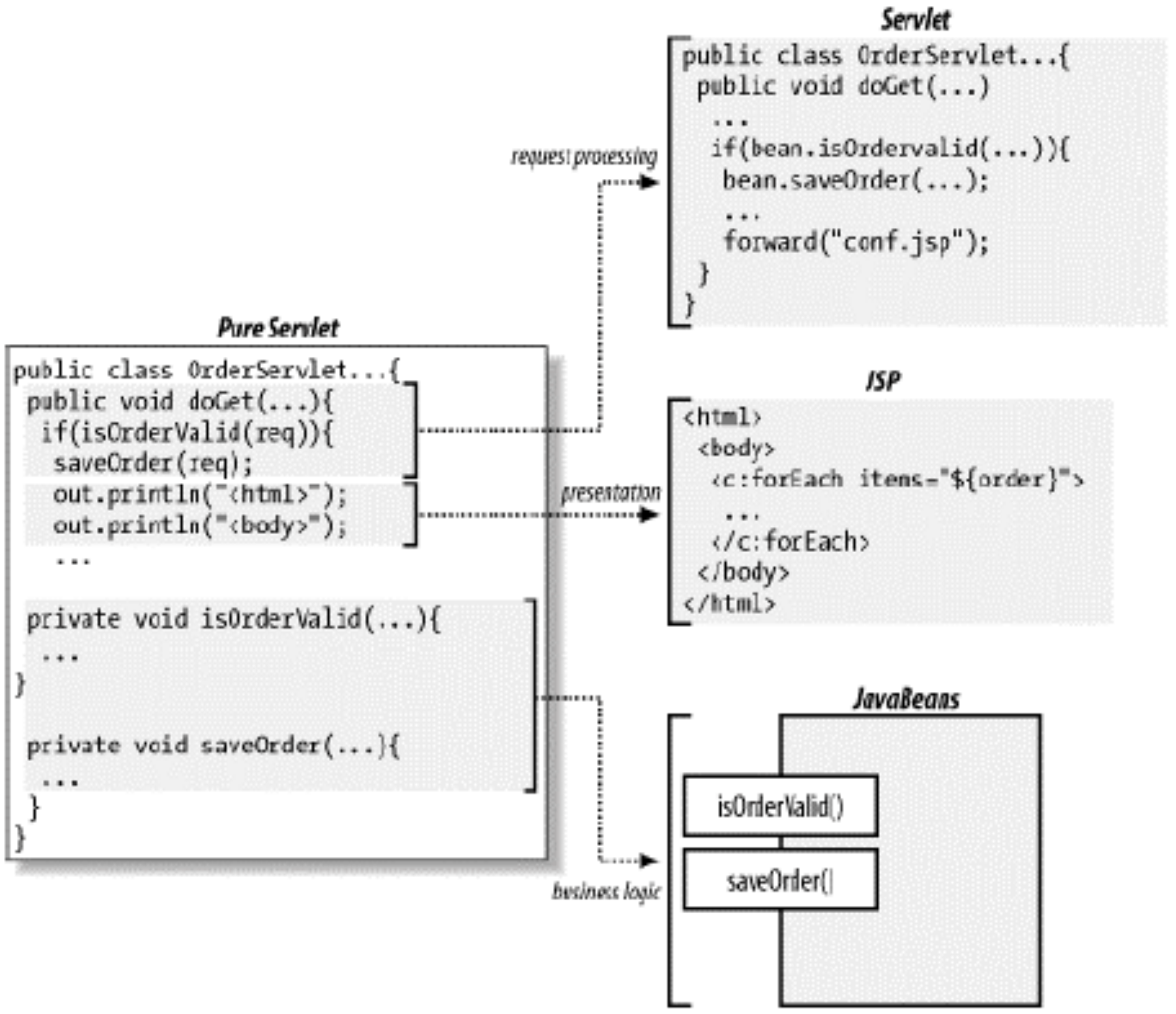
Examining MVC and JSP

- For Web applications the MVC pattern can work as follows:
  - The model portion of an application can be developed with any Java (and non-Java) classes such as normal Java programs, JavaBeans, EJB, and other.
    It can also provide data storage using a database-management system, flat files, and so on.
  - The view portion is implemented by means of JSP, HTML, and JavaScript
  - The controller is a navigation object that redirects control to appropriate classes based on the user’s choices or some other events that may happen in the system. For example, the user makes a selection on a Web page that has to be processed by a Java class or JSP. A Java servlet is a good candidate for this role.

JSP model

- MVC pattern:
  - **Control**: The request processing logic can remain the domain of the servlet
  - **View**: instead of embedding HTML in the code (servlet), you place all static HTML in a JSP page, just as in a regular web page, and add a few JSP elements to generate the dynamic parts of the page.
  - **Model**: the business data (and related business logic) can be handled and stored in JavaBeans and/or EJB

Separation of request processing, business logic, and presentation



Pure Servlet

```
public class OrderServlet...{
 public void doGet(...){
  if(isOrderValid(req)){
   saveOrder(req);
   out.println("<html>");
   out.println("<body>");
   ...

 private void isOrderValid(...){
  ...
 }

 private void saveOrder(...){
  ...
 }
}
```

Servlet

```
public class OrderServlet...{
 public void doGet(...)
  ...
  if(bean.isOrdervalid(...)){
   bean.saveOrder(...);
   ...
   forward("conf.jsp");
  }
}
```

JSP

```
<html>
 <body>
  <c:forEach items="${order}">
   ...
  </c:forEach>
 </body>
</html>
```

JSP Constructs

Simple App

↓

Complex App

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combination (MVC)
- MVC with JSP expression language
- Custom tags
- MVC with beans, custom tags, and a framework like Struts or JSF

Pattern-Based Framework

- MVC pattern
  - Struts: MVC pattern
  - Java Server Faces