



Design Patterns

Overview and JSP/Servlet Application

Presentation Goals

- Review Good Habits
 - Layered applications
 - MVC
 - Describe Patterns
 - Refactoring
- Patterns in Servlets and JSPs
- Comparing Multiple Servlet vs. Single Servlet
- Repeatable Patterns
 - Overview and Description of components
 - Example

Reviewing good habits

Establish a good discipline and good habits will become second nature

- So what are the good habits?
 - Understanding application layering
 - Utilizing Patterns
 - GoF (Gamma, Helm, Johnson, Vlissides)
 - Creational
 - Structural
 - Behavioral
 - Model/View/Controller (MVC)
 - Refactoring

Layered Applications

Presentation(GUI,HTML,JSP)

**Controller/Mediator (Servlets,
JavaBeans)**

Domain (EJBs and JavaBeans)

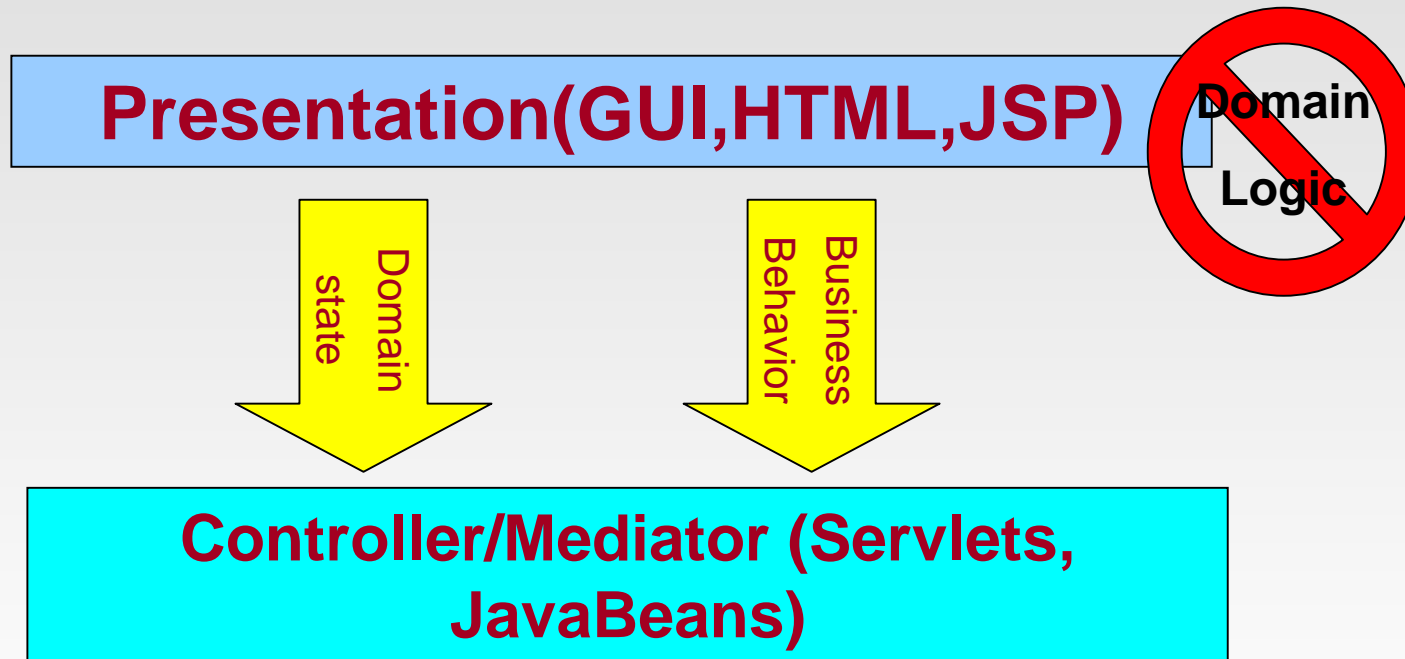
DataMapping

Data Source (JDBC, CICS, MQ ...)

Application Services

- Logging
- Exception Handling
- Start-up
- Shutdown

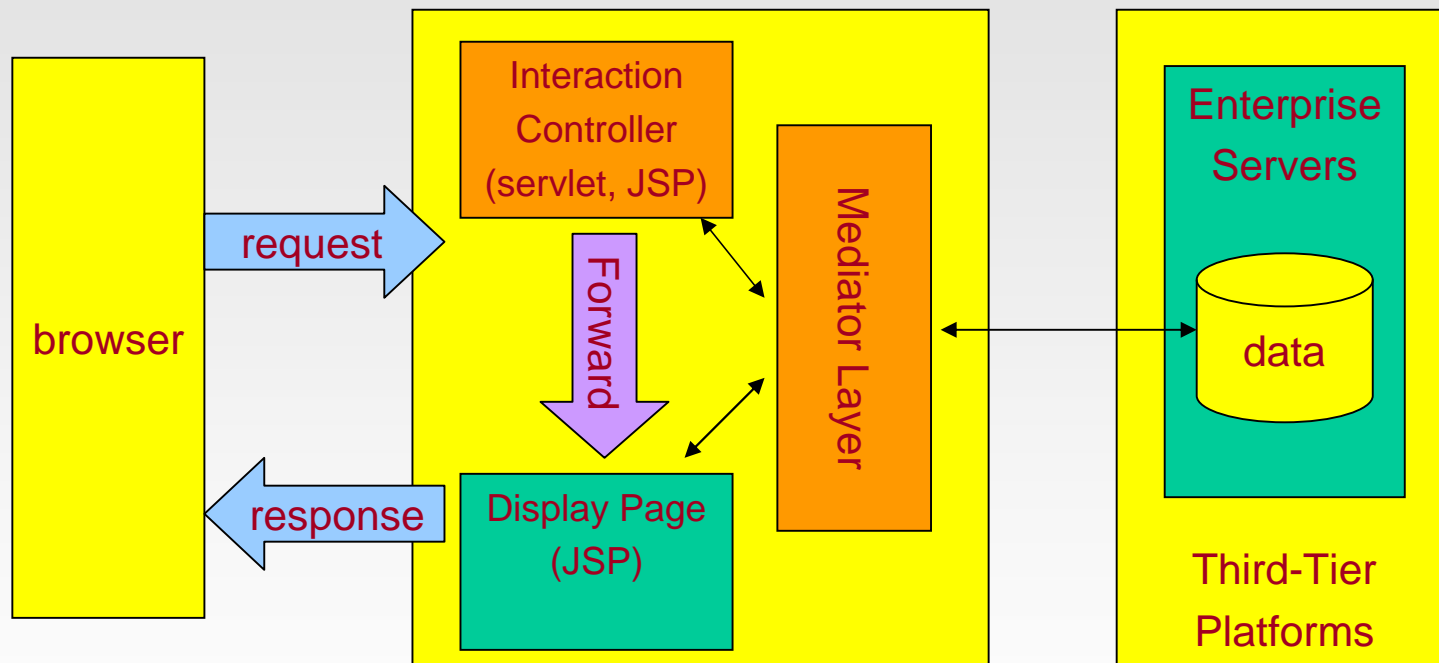
Requests for Domain State and Behavior



MVC (Model, View, Controller)

- Provides a conceptual framework for encapsulating components
 - Promotes reusability and extensibility by separating inter-dependencies between major components
 - Supports separation of development roles, e.g. GUI builders work on the View, etc.
- Servlets serve to control application flow.
- Beans encapsulate business logic, also possibly data access
- JSPs serve as the presentation medium.

MVC





Patterns

“They don’t build’em like they used to - anonymous”

What is a pattern?

- Design patterns are simple and elegant solutions to specific problems in object oriented software design.
- Patterns are key aspects of a common design structure useful for creating reusable object-oriented designs
- Design patterns capture solutions that have developed and evolved over time
- Patterns have 4 essential elements
 1. The pattern name
 2. The problem that the pattern is applied to
 3. The solution
 4. The consequences
- The basic concept of a pattern can also be seen as the basic concept of program design: *adding a layer of abstraction.*
- Separate things that change from things that stay the same.

The Gang of Four

- Authors of Design Patterns – Elements of Reusable Object-Oriented Software
- Defined a set of reusable patterns organized within categories that aid the developer in creating effective solutions
- Twenty three patterns
 - 5 Creational Patterns
 - 7 Structural Patterns
 - 11 Behavioral Patterns
- A bookshelf must!



Creational Patterns Used

- ***Abstract Factory**
 - Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- ***Factory Method**
 - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses
- ***Singleton**
 - Ensure a class only has one instance, and provide a global point of access to it.
- **Builder**
 - Separate the construction of a complex object from its representation so that the same construction process can create different representations
- **Prototype**
 - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

Sample

- A Singleton

```
final class Singleton {  
    private static Singleton s = new Singleton(15);  
    private int i;  
    private Singleton(int x) { i = x; }  
    public static Singleton getSingleton() { return s; }  
    public int getValue () { return i; }  
    public setValue(int x) { i = x; }  
}
```

Structural Patterns Used

- *Façade
 - Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use
- *Proxy
 - Provide a surrogate or placeholder for another object to control access to it.
- Adapter
 - Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces
- Bridge
 - Decouple an abstraction from its implementation so that the two can vary independently
- Composite
 - Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly

Structural Patterns (cont.)

- Decorator
 - Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality
- Flyweight
 - Use sharing to support large numbers of fine-grained objects efficiently

Behavioral Patterns

- *Command
 - Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations
- *Mediator
 - Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently
- *Memento
 - Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later
- *State
 - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class

Behavioral Patterns (cont.)

- Chain of Responsibility
 - Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
- Interpreter
 - Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language
- Iterator
 - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation
- Observer
 - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically

Behavioral Patterns (cont.)

- Strategy
 - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
- Template Method
 - Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure
- Visitor
 - Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Types of patterns used in JSP/Servlets development

- Model/View/Controller
- Creational
 - Singletons
 - AbstractFactory
 - Factory Method – seen in servlet processing to create mediators
- Behavioral
 - Command – can be used to limit network traffic
 - Momento – seen in how HttpSession maintain state between pages
 - State – seen in business objects to enforce proper state management
 - Mediators – used in Servlets to decouple knowledge of domain
- Structural
 - Façade – used in providing an interface to EJB sessions
 - Proxy – used in EJBs stubs



Refactoring

“If it Stinks, change it” – [Grandma Beck] discussing child rearing

Refactoring

- Is a process of improving the overall design of an application by:
 - Adding levels of indirection
 - Adding abstraction layers
 - Moving methods that know too much about another class
 - And not changing the external behavior of the code
- It's a disciplined way to clean up code that minimizes the chance of introducing bugs
- Verifying changes by utilizing Unit Test Frameworks (like JUnit)



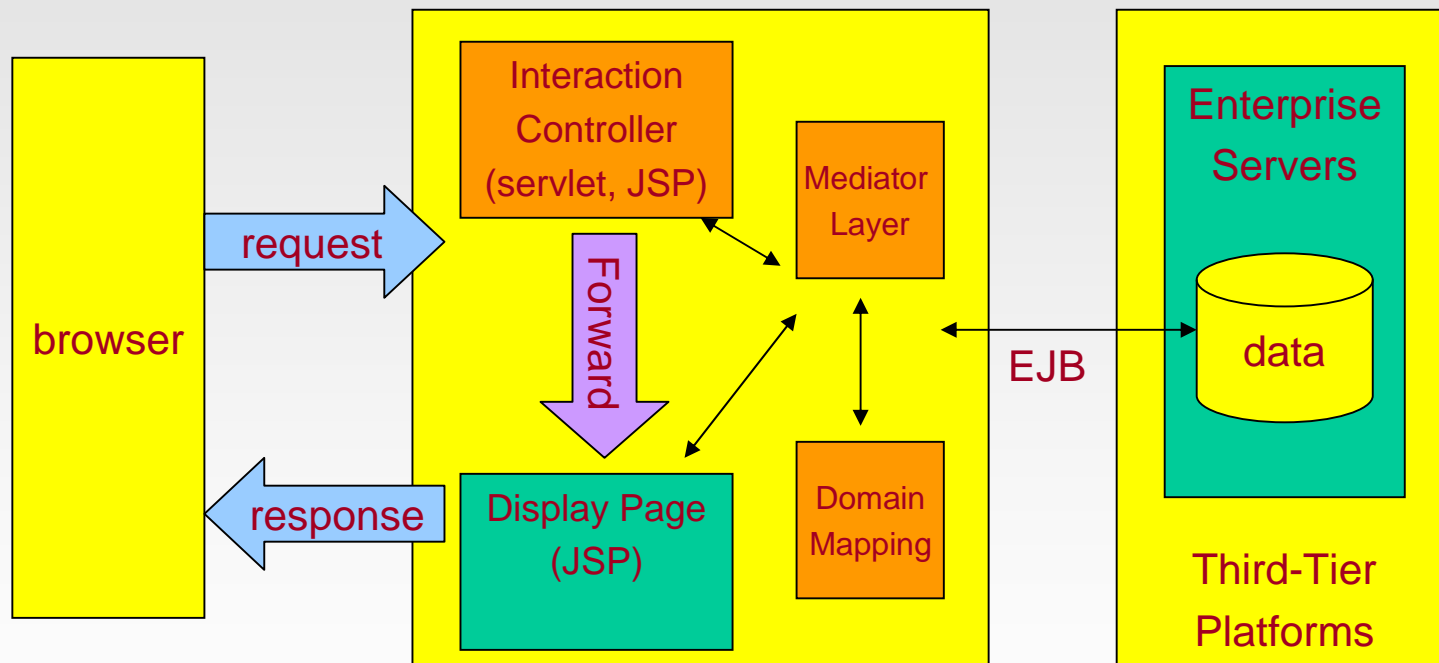
Patterns in JSPs/Servlets

“There is more than one way to skin a cat”

The Mediator Pattern built upon MVC

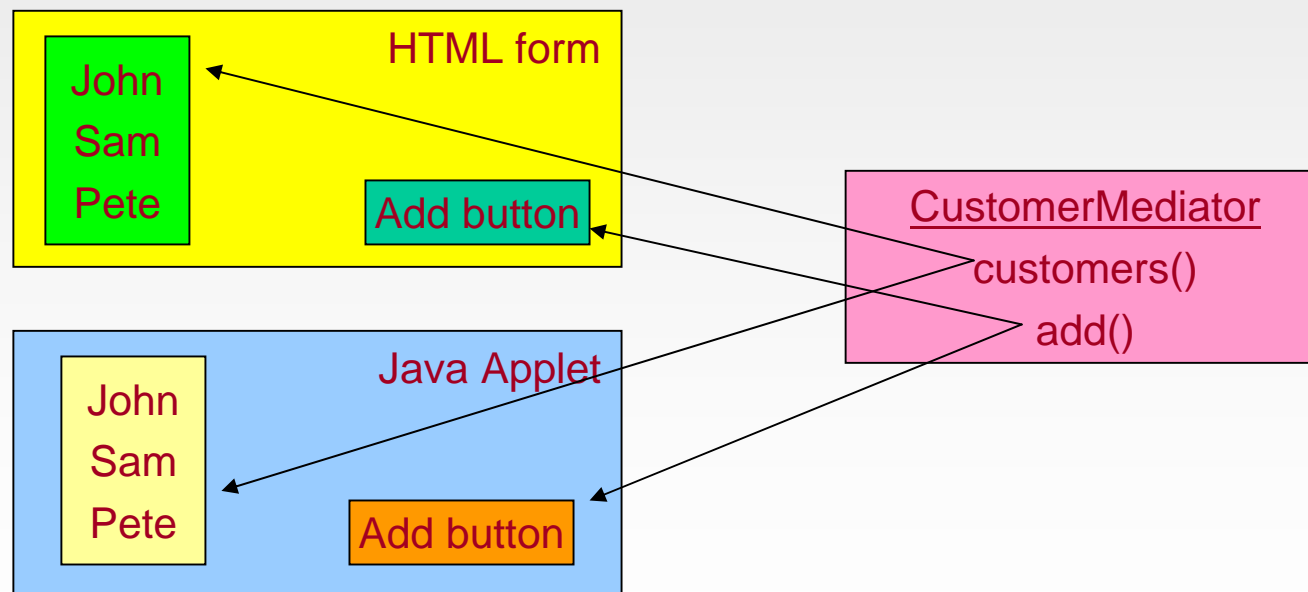
- Mediators are introduced to support the View by indirectly accessing the Model.
- Domain Mappers are introduced to add an additional layer of abstraction between the Data and the Model.
 - E.g., Domain mappers can point to EJBs, Data in Memory, Datasources
- Servlets still serve to control flow but is less involved in working with the Model.
- JSPs are less involved in working with the Model. JSP is treated solely as a presentation technology.
 - I.E. Mediator can help fetch for content for building GUIs by indirectly accessing Data.

Mediator Pattern builds on MVC



Mediator Pattern – the Mediator

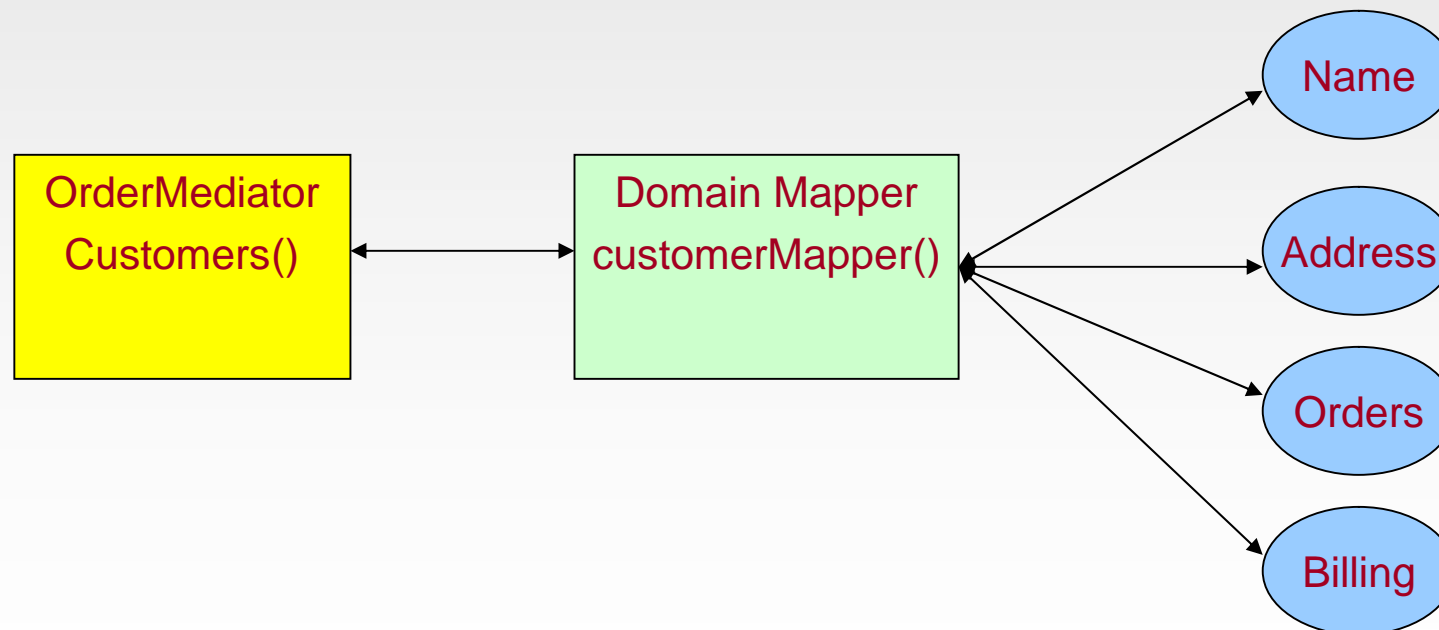
- The Mediator decouples and captures the requirements of the view for the model.
 - Mediator generalizes the interaction between the view and model.
 - In effect, the Mediator can provide support for behaviors used in different types of presentation technologies, i.e. an Applet form and an HTML form.
 - JSPs only have knowledge of and uses Mediators (which indirectly accesses the Model) for accessing data necessary to build the view.



Mediators (cont.)

The Mediator decouples persistence behavior and knowledge from the Model.

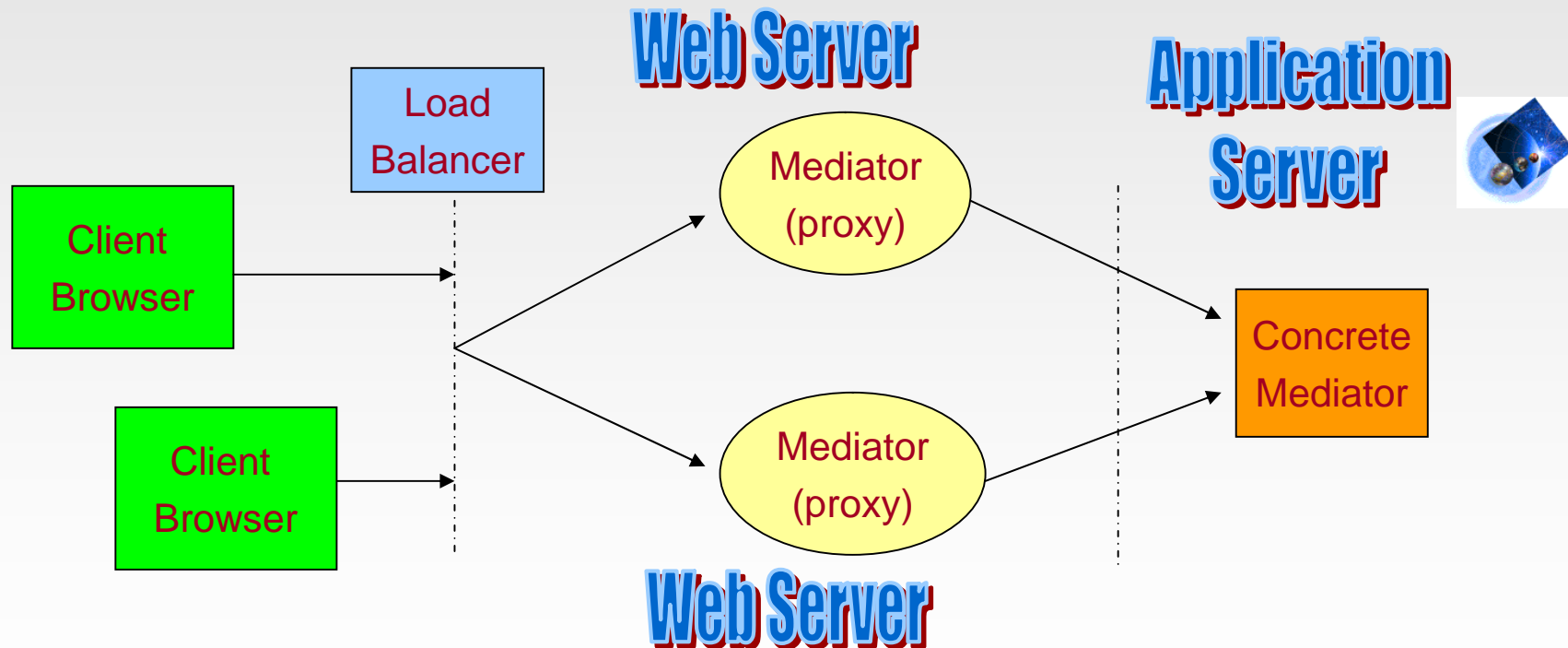
- Mediators, with the help of Domain mapping classes



Distributed Mediator Topology

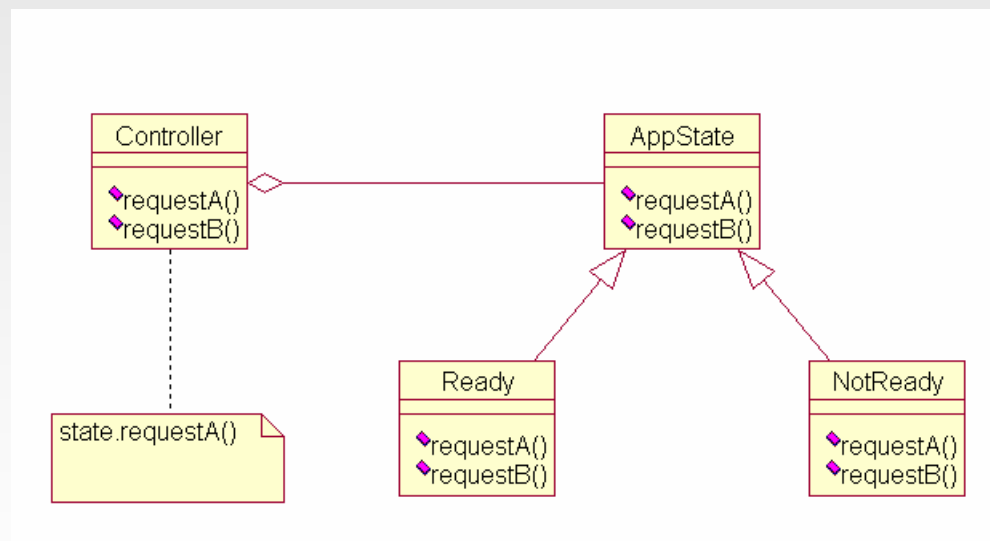
Making the Mediator a distributed object provides for a thin client.

- Helps keep the session lightweight.



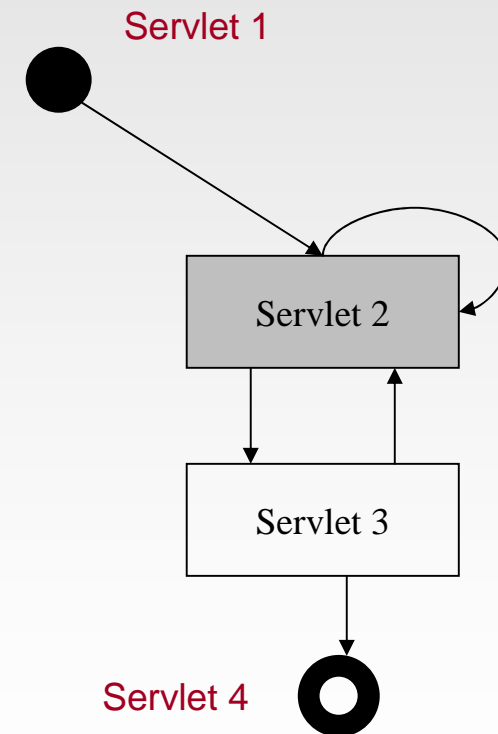
The State Pattern

- Used to determine appropriate behavior of an incoming client request
- Managed by the controller



State Transitions

- Once you view a page as a directed graph, you see how it can be viewed as a State Transition Diagram.
- Coding State Diagrams is manageable
 - Gamma et. al.'s *State Pattern*
 - Table Representations
- At each service request determine if a transition is valid.



Simplest Representation

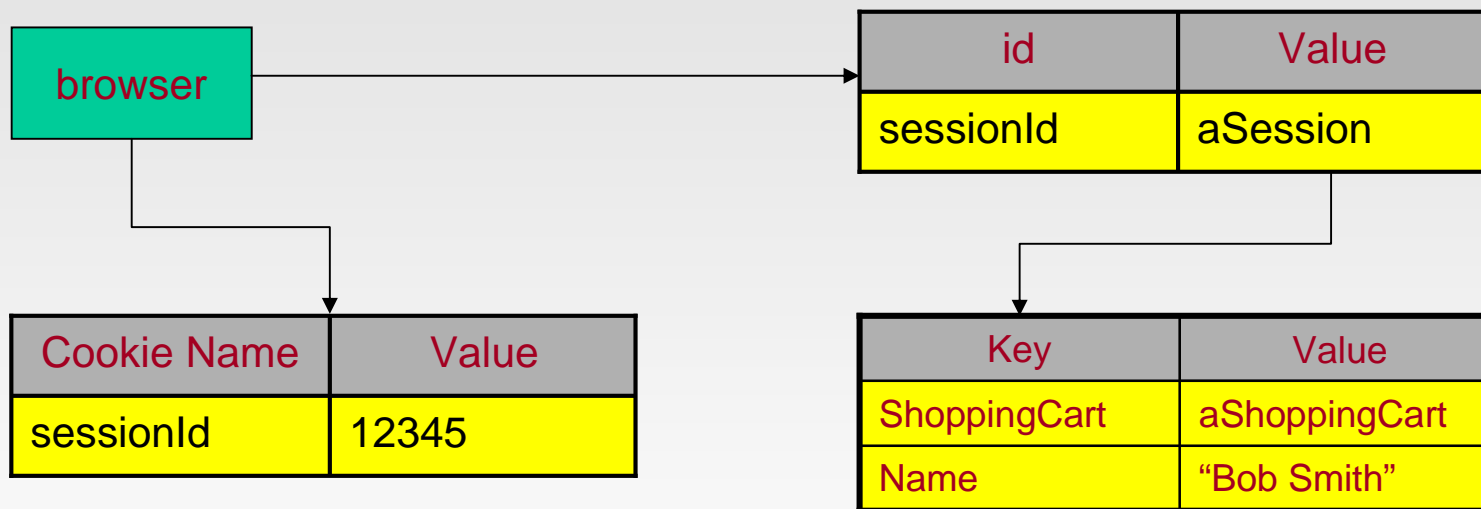
- Simplest state representation is a 2-d array
 - Use booleans to represent if a transition is valid or not.
 - Keep track of the “current” node in a Session.
 - Lookup the “From” and “to” nodes on a request.
 - If the transition is invalid, show an error page

	Servlet 1	Servlet 2	Servlet 3
Servlet 1	true	true	true
Servlet 2	false	true	true
Servlet 3	false	false	true

Using Momentos

- When traversing backwards through a series of forms you want to redisplay information already entered.
 - If you are not caching, the information previously entered is gone.
- Therefore, you want to store the entered information on the server (in a Session)
 - The HttpSession class utilizes the Momento Design Pattern
 - Have a servlet “refill” the form with stored information.

HttpSession lookup





Comparing Multiple Servlets vs. Single Servlet

“Are two heads better than one?”

Role of the Servlet

- Servlets are responsible for taking parameters from the presentation layer
- Contacting the appropriate business logic classes
- Routing the user to the next appropriate screen.
- Servlets act as filters, data comes in, its processed and it goes out to a different form

Servlets act as filters

Benefits of multiple servlets

- Each servlets knows how to act as a particular “filter” and can process the parameters appropriately.
- Security can be set at a servlet level within certain application servers.
 - Prevents unauthorized access to servlets
- Separation of control logic layer
- Tools can evaluate hit counts on servlets

Single Servlet

- All Servlets follow a basic process
 - Take a request made over HTTP
 - Extract passed arguments
 - Initiate a process to handle the requests
 - Provide dynamic results based on the processing
 - Return HTML directly
 - Forward to a JSP
 - Redirect to a new page

Benefits of a Single Servlet

- Take advantage of Polymorphism by:
 - Utilizing the AbstractFactory Pattern to create instances of controller objects
 - Refactoring common methods into a base class which all subclasses inherit
- Allows for easier modifications
- Reduced duplication between servlets
- Servlet is coded once
- Less compilation of servlets
- No Application Server reconfigurations
- Servlet can remain running as new user interfaces are added
- Reduces thread safety issues



Repeatable Patterns

“Knit one Pearl two”

What you find in the repeatable patterns

- Our Best Practices have been developed within this JSP/Servlet Pattern that include:
 - Development Management
 - Setting Standards
 - Adhering to Change Management practices
 - Follow a Development process (UML, XP ...)
 - Establish division of labor between Web Designers and Java Developers
 - Deployment/Tooling
 - Target today's browsers (ie 5.0+, NS 4.7+)
 - Tool Utilization (VisualAge, WebSphere Studio, Dreamweaver...)
 - Limited Javascript, rules in business object/layer

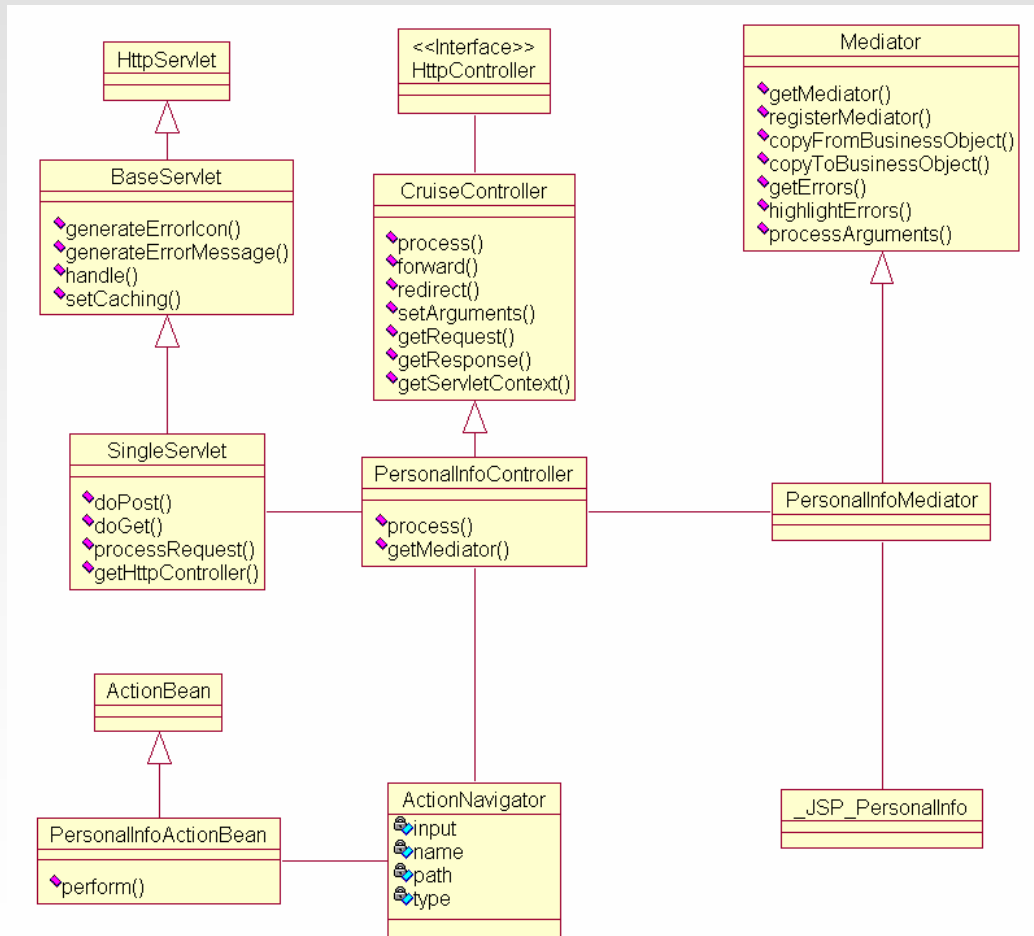
What you find in the repeatable patterns (cont.)

- Development Processes
 - Implement a Single Servlet process
 - Effectively use Mediator/Javabeans
 - Use Polymorphism
 - Develop using layers
 - Refactoring
 - Session management and persistence (State patterns)
 - Follow an Exception Handling Strategy
 - Utilize a Logging strategy
 - Always build Unit Tests Suites
 - Use of a translator/formatter
 - Static members for consistency
 - Common error display pattern

Model/View/Controller implementation

- Clean separation of responsibility and maintenance of elements
 - Using JSP primarily as “Display Pages”
- Utilize AbstractFactory pattern to instantiate controller logic
 - limit what each controller knows and does
- Single Servlet
 - Uses the Mediator to extract Form information and process business transaction
 - perform process()
 - performs copyToBusinessObject()
 - performs copyFromBusinessObject()
 - performs save()
 - Instantiates unique controller objects to manage each page
 - Utilizes ActionNavigator object to manage next forward/redirect

MVC Implementation

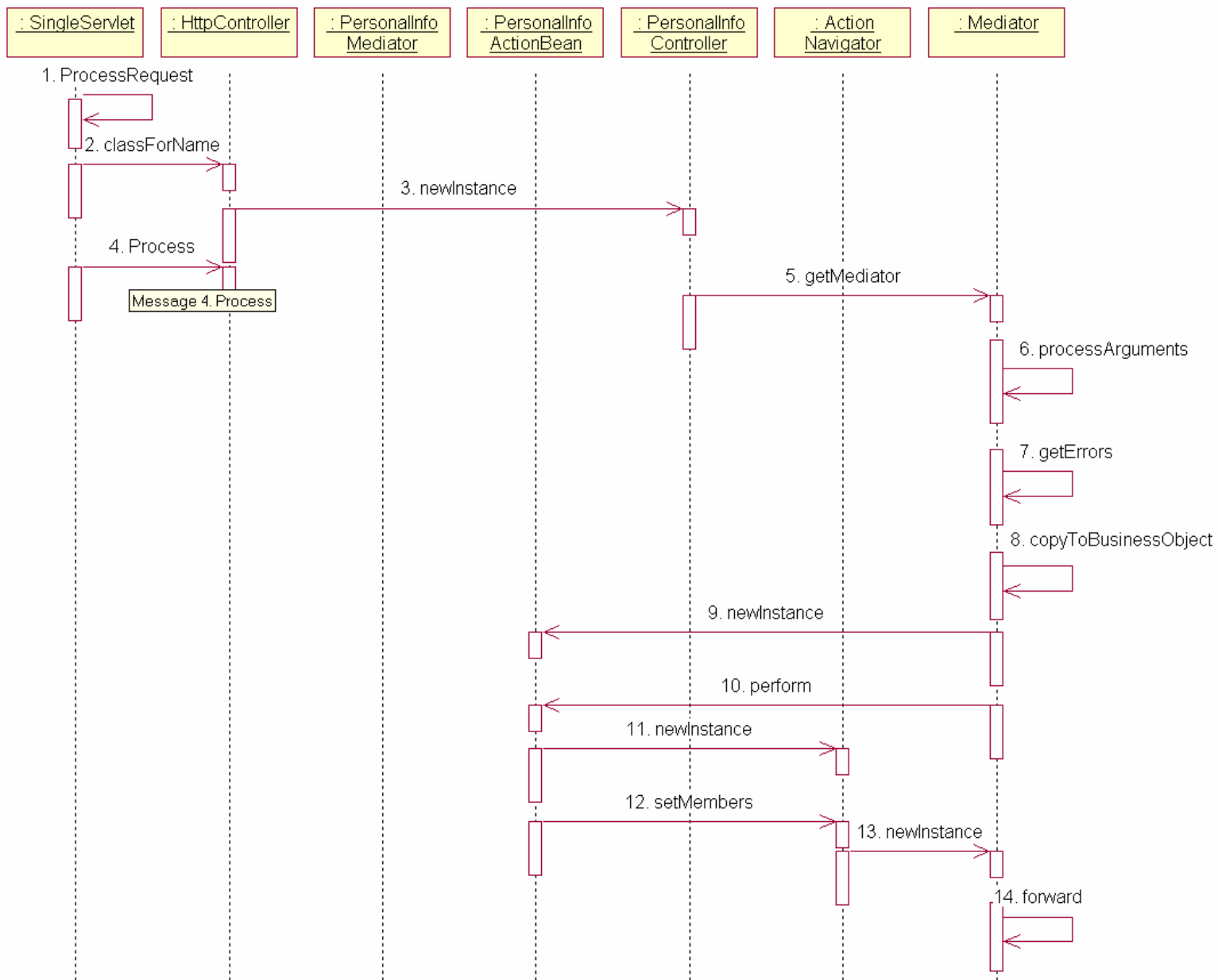


ServletController CRC

- Class Description
 - A servlet controller class manages navigation between servlets and JSPs
- Responsibility
 - Acts as the controller
 - based on action parameter
 - Instantiates appropriate ActionBean object
 - Sends messages to Mediator for application processing
 - eg. copyToBusinessObject(), process(), getErrors() ...
 - Instantiates ActionBean to get next Page
 - Ensures thread safety with synchronization
 - Ensures proper navigation and recovery
- Collaborators
 - Mediator
 - ActionNavigator
 - ActionBean

Mediator CRC

- Class Description
 - The Mediator is a supporting java bean which is utilized as a standard in our JSP development. Its properties contain the values of the elements displayed and/or captured from a HTML form. It is instantiated in a servlet and passed via the HttpSession object to the JSP. It is referenced in the JSP using <UseBean>
- Responsibility
 - Takes form parameters and sets into bean
 - Takes business model information and sets into bean
 - Delivers HTML elements populated with model information
 - Performs getErrors() processing
 - Performs highlightErrors() to signal problems to user
- Collaborators
 - JSP
 - ServletController



Resources and contacts

- Web Sites
 - Struts Framework at www.apache.org
 - WebSphere Developers Domain www.ibm.software/software/ad/wsdd
- Books
 - Design Patterns – Gamma et al.
 - Refactoring – Martin Fowler
 - WebSphere Application Server Bible
- Contact me for questions:
Bryon Kataoka – Commerce Solutions
www.commercesolutions.com
707.773.1198
bkataoka@commercesolutions.com