# DESIGN PATTERNS

Course 9

# PREVIOUS COURSE CONTENT

- Refactoring

# CURRENT CURSE CONTENT

- Applications split on levels
- J2EE Design Patterns
- Intercepting Filters

# APPLICATIONS SPLIT ON LEVELS

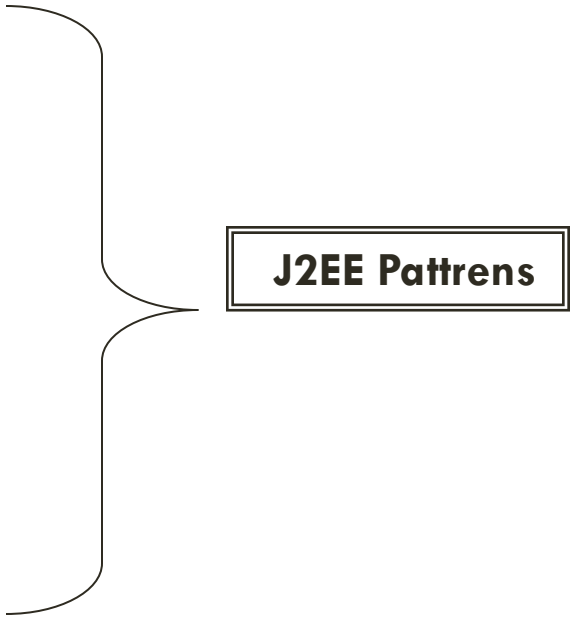
**Client Level**
Application clients, applets, others GUIs

**Prezentation level**
JSP, Servlets and others UI elements

**Business level**
EJB and others business resources

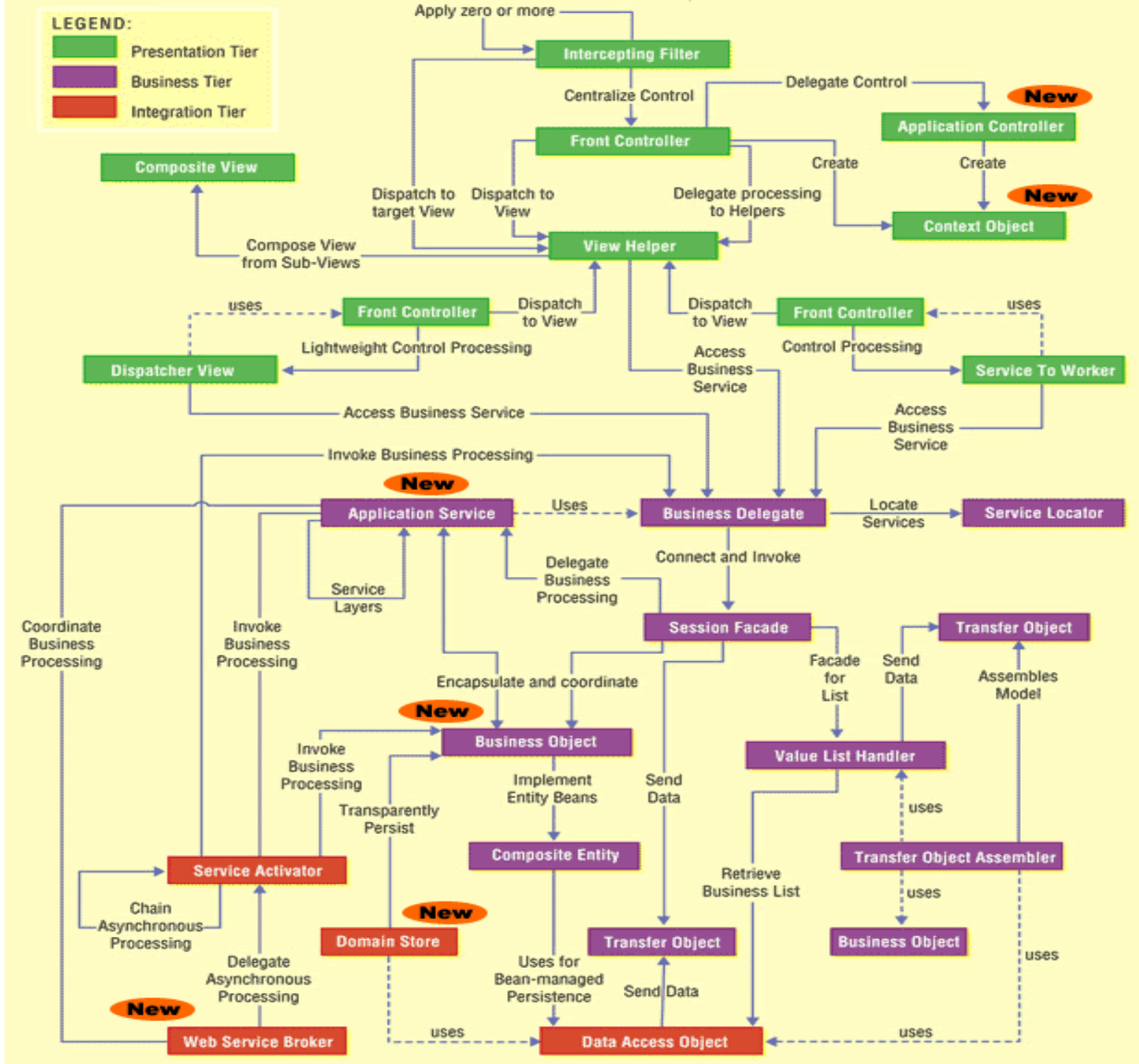
**Integration level**
JMS, JDBC, Connecters

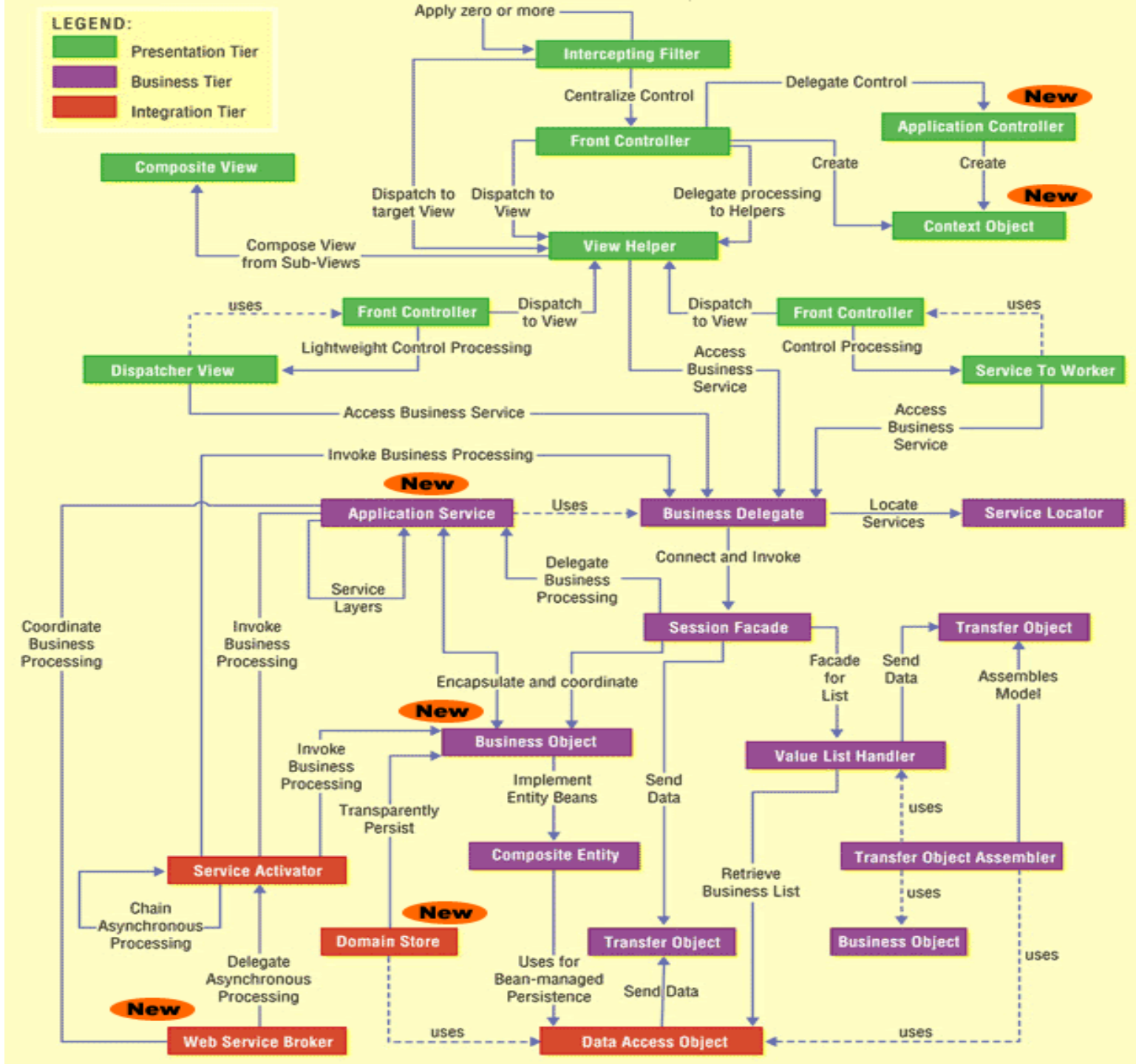
**Resource level**
Data bases, external systems, resources



# PATTERNS CLASSIFICATION

- Patterns applicable on presentation level
- Patterns applicable on business level
- Patterns applicable on integration level


LEGEND:
Presentation Tier
Business Tier
Integration Tier
Apply zero or more
Intercepting Filter
Centralize Control
Delegate Control
New
Application Controller
Front Controller
Create
Create
New
Context Object
Composite View
Dispatch to target View
Dispatch to View
Delegate processing to Helpers
Compose View from Sub-Views
View Helper
uses
Front Controller
Dispatch to View
Dispatch to View
Front Controller
uses
Lightweight Control Processing
Control Processing
Dispatcher View
Access Business Service
Service To Worker
Access Business Service
Access Business Service
Invoke Business Processing
New
Application Service
Uses
Business Delegate
Locate Services
Service Locator
Service Layers
Delegate Business Processing
Connect and Invoke
Coordinate Business Processing
Invoke Business Processing
Session Facade
Transfer Object
Facade for List
Send Data
Assembles Model
Encapsulate and coordinate
New
Business Object
Value List Handler
Invoke Business Processing
Implement Entity Beans
Send Data
uses
Transparently Persist
Composite Entity
Transfer Object Assembler
Service Activator
Retrieve Business List
Chain Asynchronous Processing
New
Domain Store
Transfer Object
Business Object
uses
Delegate Asynchronous Processing
Uses for Bean-managed Persistence
Send Data
uses
New
Web Service Broker
uses
Data Access Object
uses


LEGEND:
Presentation Tier
Business Tier
Integration Tier
Apply zero or more
Intercepting Filter
Centralize Control
Delegate Control
New
Application Controller
Front Controller
Create
Create
New
Context Object
Composite View
Dispatch to target View
Dispatch to View
Delegate processing to Helpers
Compose View from Sub-Views
View Helper
uses
Front Controller
Dispatch to View
Dispatch to View
Front Controller
uses
Lightweight Control Processing
Control Processing
Dispatcher View
Access Business Service
Service To Worker
Access Business Service
Access Business Service
Invoke Business Processing
New
Application Service
Uses
Business Delegate
Locate Services
Service Locator
Connect and Invoke
Delegate Business Processing
Service Layers
Coordinate Business Processing
Invoke Business Processing
Session Facade
Transfer Object
Facade for List
Send Data
Assembles Model
Encapsulate and coordinate
New
Business Object
Value List Handler
Invoke Business Processing
Implement Entity Beans
Send Data
Transparently Persist
uses
Composite Entity
Transfer Object Assembler
Service Activator
Retrieve Business List
Chain Asynchronous Processing
uses
New
Domain Store
Transfer Object
Business Object
Delegate Asynchronous Processing
Uses for Bean-managed Persistence
Send Data
uses
New
Web Service Broker
uses
Data Access Object
uses

# PRESENTATION PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

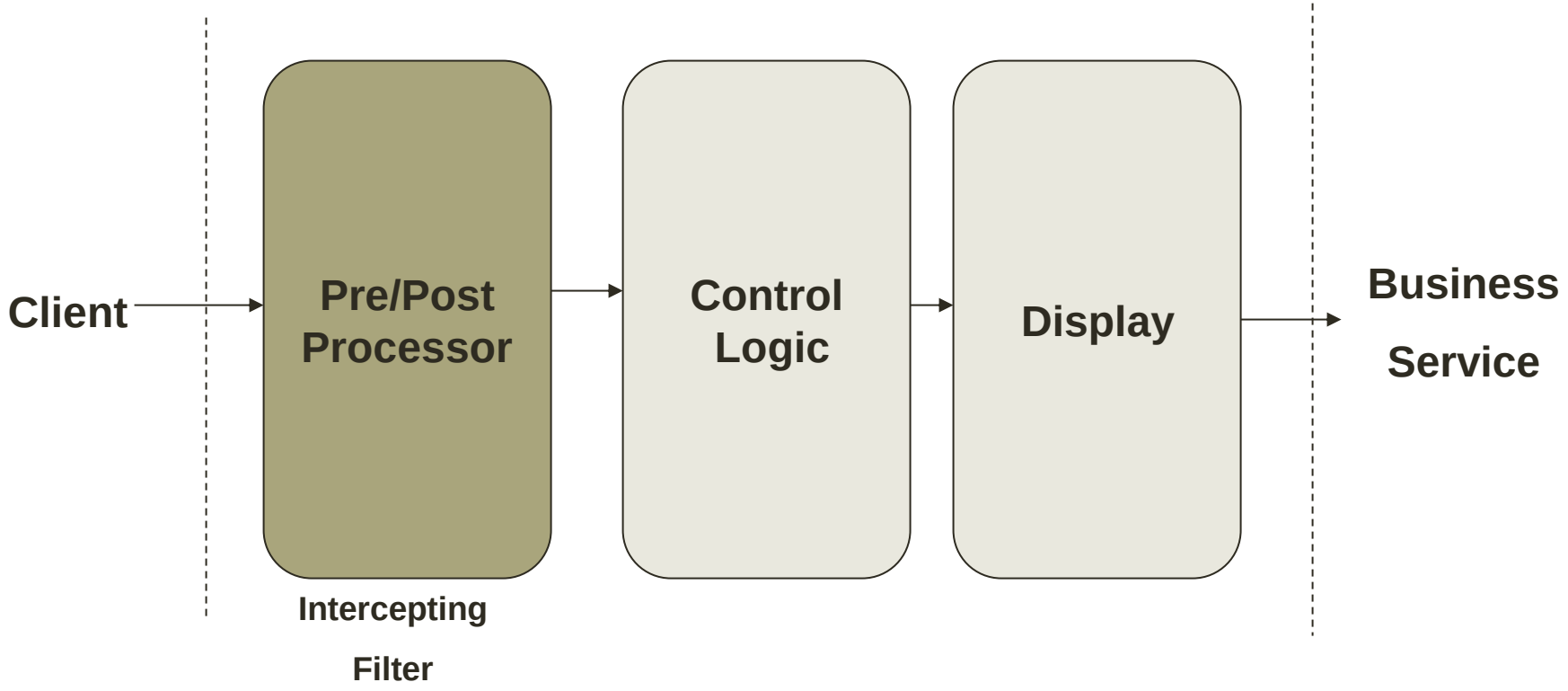
Dispatcher View

**Facilitates pre/post request/response processing**

Useful for security check, caching, packaging

Independent chain filtering

# PRESENTATION PATTERNS. INTERCEPTING FILTER



# PRESENTATION PATTERNS. INTERCEPTING FILTER

- Problem
  - You want to intercept and manipulate a request and a response before and after the request is processed.

```
<%
        if (session.getAttribute("user") == null) {
            //redirect to a login page
    }
%>
```

- Examples
  - Has the client a valid session?
  - The request satisfies all the constraints?
  - What encoding is used to pass date?
  - Is the request stream encoded or compressed?
  - From which browser came the request?

# PRESENTATION PATTERNS. INTERCEPTING FILTER

Forces

- You want centralized, common processing across requests, such as checking the data-encoding scheme of each request, logging information about each request, or compressing an outgoing response.

- You want pre and postprocessing components loosely coupled with core request-handling services to facilitate unobtrusive addition and removal.

- You want pre and postprocessing components independent of each other and self contained to facilitate reuse.

# PRESENTATION PATTERNS. INTERCEPTING FILTER

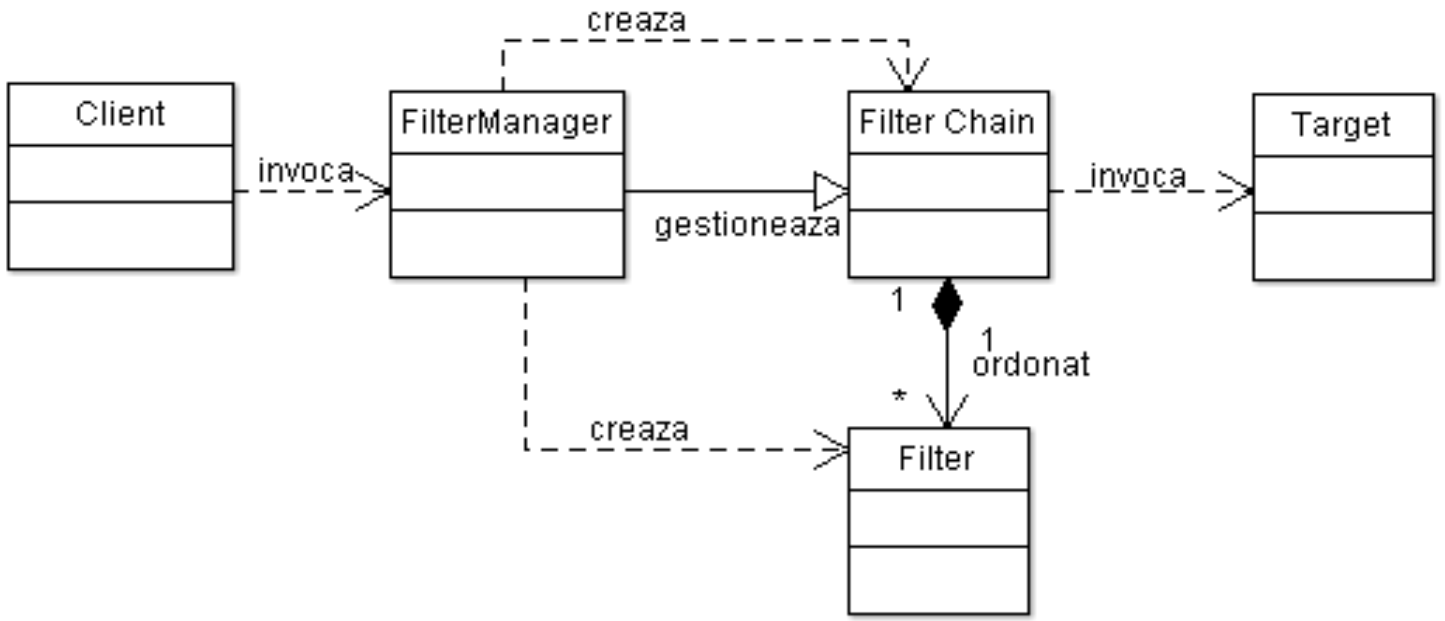
Solution

- Use an Intercepting Filter as a pluggable filter to pre and postprocess requests and responses. A filter manager combines loosely coupled filters in a chain, delegating control to the appropriate filter. In this way, you can add, remove, and combine these filters in various ways without changing existing code.

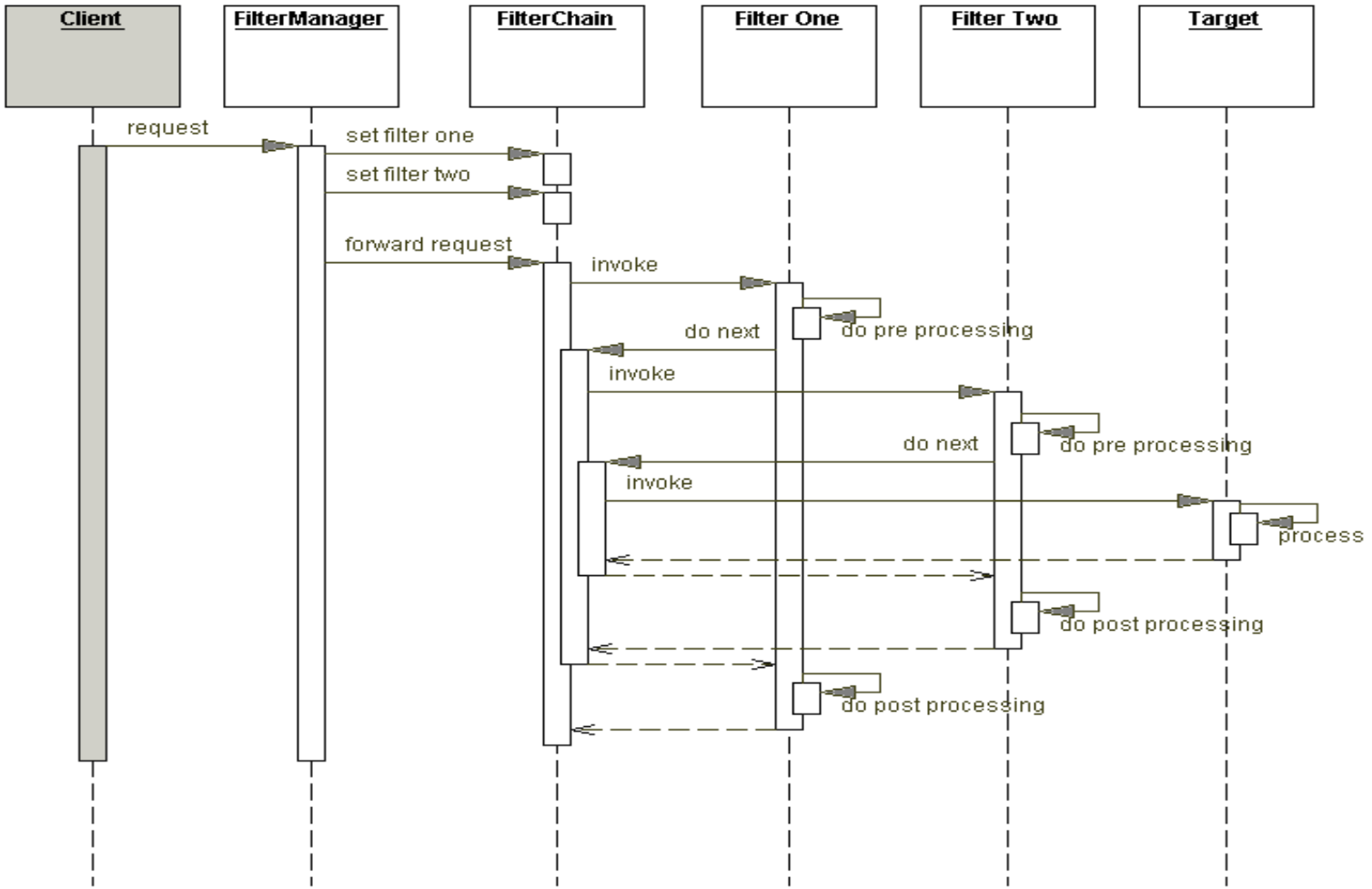
Examples

- Servlets Filter for HTTP request/responce
- Message Handles for SOAP request/responce

# PRESENTATION PATTERNS. INTERCEPTING FILTER



# PRESENTATION PATTERNS. INTERCEPTING FILTER



# PRESENTATION PATTERNS. INTERCEPTING FILTER

Strategies

- Standard filters (Servlet 2.3)
  - Components  used in deployment descriptor
- Personalized filters
- Base filters
- Filters based on templates

## Standard Filter Example

```
public class HelloWorldFilter implements Filter {
public FilterConfig filterCfg;
   @Override
    public void destroy() { }
    @Override
    public void init(FilterConfig filterCfg) throws ServletException {
      this.filterCfg = filterCfg;
}
  public void doFilter(final ServletRequest request, final
 ServletResponse response, FilterChain chain)
      throws java.io.IOException,    javax.servlet.ServletException {
       System.out.println("Intrare Filter");
       request.setAttribute("hello", "Hello World!");
       chain.doFilter(request, response);
       System.out.println("Iesire HelloWorldFilter");
  } }
```

## Deployment Descriptor

```
<filter>
    <filter-name>helloWorld</filter-name>
    <filter-class>
 web.filter.baseStrategy.HelloWorldFilter
     </filter-class>
</filter>

<filter-mapping>
    <filter-name>helloWorld</filter-name>
    <url-pattern>/jsp/filter.jsp</url-pattern>
</filter-mapping>
```

# TEMPLATE FILTERS
# FOCUS ON PRE/POST PROCESSING

```
public abstract class TemplateFilter implements Filter {
    private FilterConfig filterCfg;

    public void init(FilterConfig filterCfg) throws ServletException {
        this.filterCfg = filterCfg;
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
ServletException {
            doPreProcessing(request, response);
            chain.doFilter(request, response);
            doPostProcessing(request, response);
    }

    public abstract void doPostProcessing(ServletRequest request, ServletResponse response);

     public abstract void doPreProcessing(ServletRequest request, ServletResponse response);

     public void destroy() { }
}
```

# PRESENTATION PATTERNS. INTERCEPTING FILTER

Consequences

- Centralizes control with loosely coupled handlers
- Improves reusability
- Declarative and flexible configuration
- Information sharing is inefficient

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

**Offers a centerlasid controller for request management**

Enter point in system. It does not have to be too large (it delegates attributes to Application Controller

Request processing:

1. Protocol handeling and context transformations
2. Navigation and routing
3. Request pocessing
4. Control transfer

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

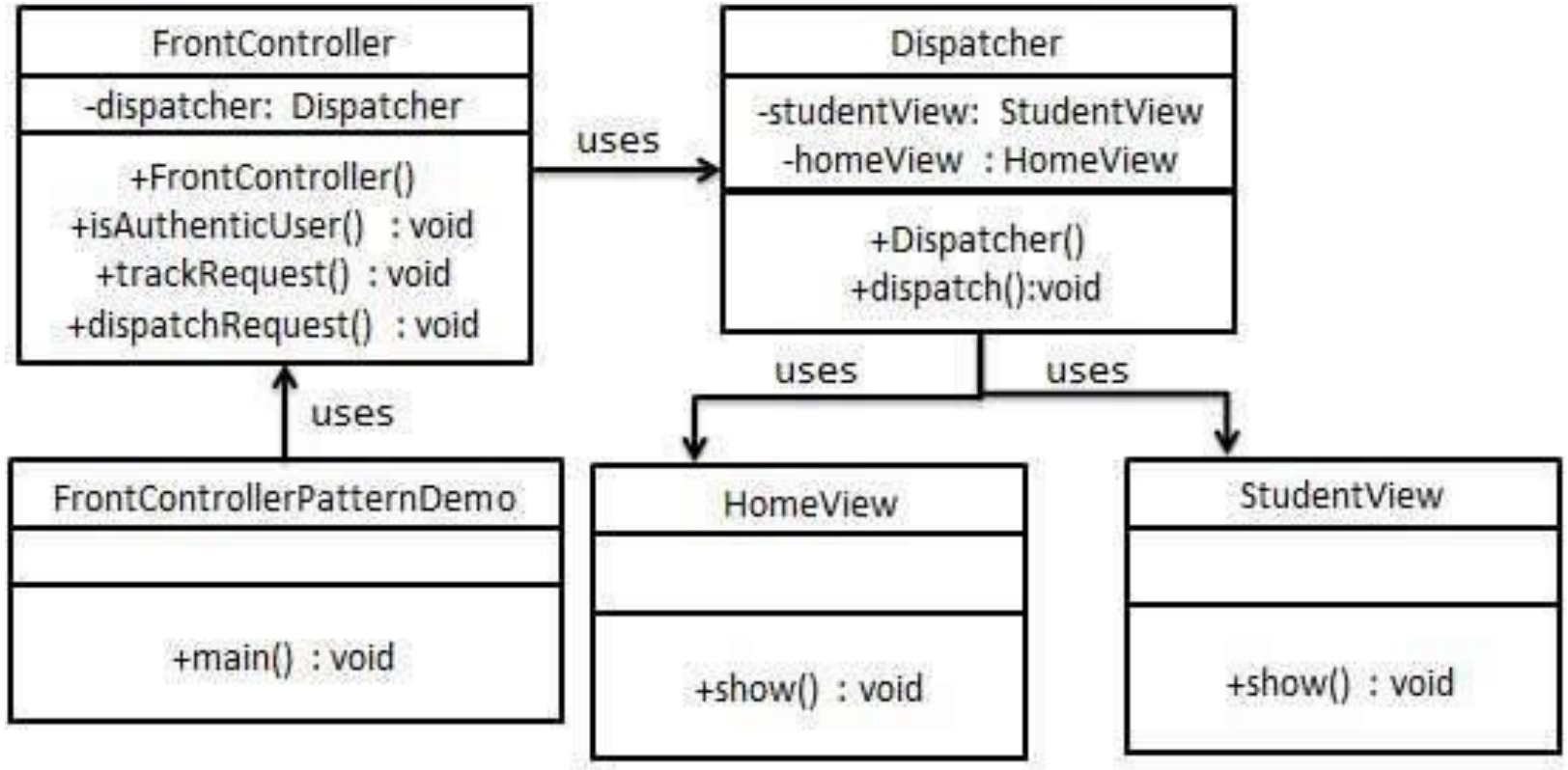
Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View



# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

**Incapsulates the state in order to avoid using protocol-specific system information outside of its relevant contexte**

The application componesnts does not have to know HTTP protocol. They must call methods getXX  on an object of the context.

Struts - ActionForm

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

*You want to centralize and modularize action and view management.*

- ❑ You want to reuse action and view-management code.
- ❑ You want to improve request-handling extensibility, such as adding use case functionality to an application incrementally.
- ❑ You want to improve code modularity and maintainability, making it easier to extend the application and easier to test discrete parts of your request-handling code independent of a web container.

Struts

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

*You want to separate a view from its processing logic.*

- ❑ You want to use template-based views, such as JSP.
- ❑ You want to avoid embedding program logic in the view.
- ❑ You want to separate programming logic from the view to facilitate division of labor between software developers and web page designers.

Expresion Language, JSLT

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

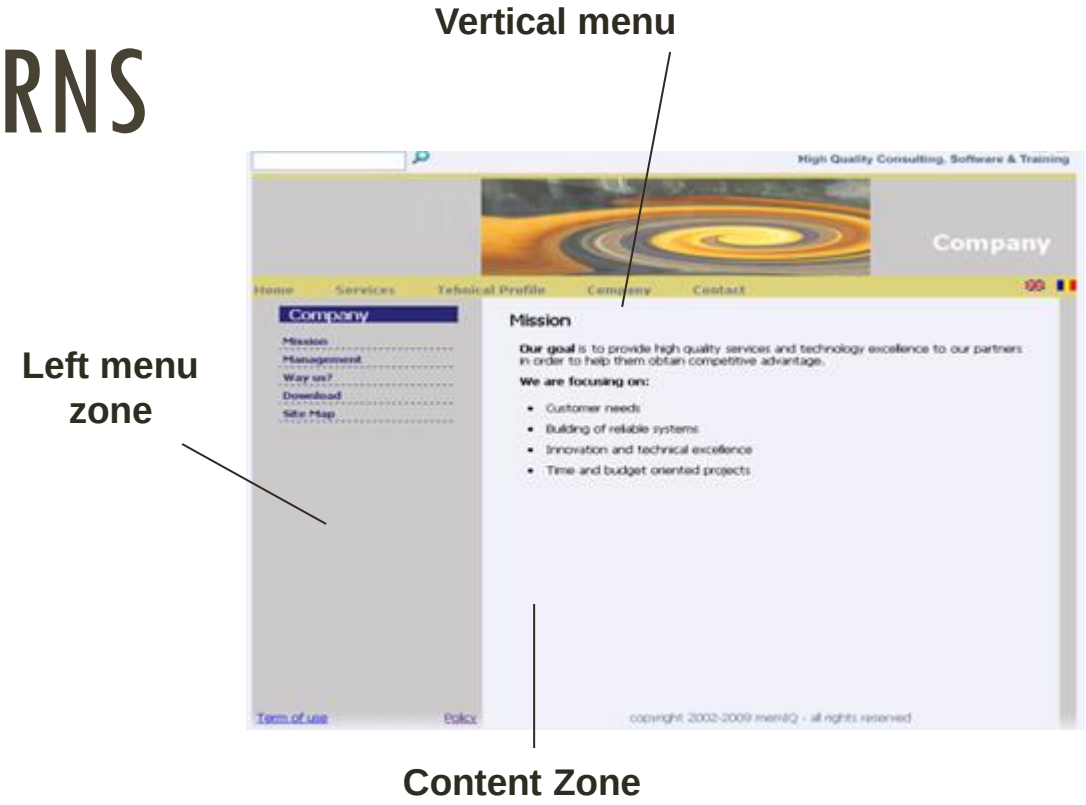
Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View




**Crează şi agregă vederi din componente atomice**

Tiles

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

*You want to perform core request handling and invoke business logic before control is passed to the view*

Use Service to Worker to centralize control and request handling to retrieve a presentation model before turning control over to the view. The view generates a dynamic response based on the presentation model.

# PRESENTATIONS PATTERNS

Intercepting Filter

Front Controller

Context Object

Application Controller

View Helper

Composite View

Service to Worker

Dispatcher View

*You want a view to handle a request and generate a response, while managing limited amounts of business processing.*

Use Dispatcher View with views as the initial access point for a request. Business processing, if necessary in limited form, is managed by the views.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You want to hide clients from the complexity of remote communication with business service components.*

- ❑ You want to access the business-tier components from your presentation-tier components and clients, such as devices, web services, and rich clients.
- ❑ You want to minimize coupling between clients and the business services, thus hiding the underlying implementation details of the service, such as lookup and access.
- ❑ You want to avoid unnecessary invocation of remote services.
- ❑ You want to translate network exceptions into application or user exceptions.
- ❑ You want to hide the details of service creation, reconfiguration, and invocation retries from the clients.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You want to transparently locate business components and services in a uniform manner.*

Use a Service Locator to implement and encapsulate service and component lookup.

A Service Locator hides the implementation details of the lookup mechanism and encapsulates related dependencies.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*ou want to expose business components and services to remote clients.*

Use a *Session Façade* to encapsulate business-tier components and expose a coarse-grained service to remote clients.

Clients access a *Session Façade* instead of accessing business components directly.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You want to centralize business logic across several business-tier components and services.*

Use an Application Service to centralize and aggregate behavior to provide a uniform service la

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You have a conceptual domain model with business logic and relationship.*

Use Business Objects to separate business data and logic using an object model.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You want to use entity beans to implement your conceptual domain model.*

Use a Composite Entity to implement persistent Business Objects using local entity beans and POJOs.

Composite Entity aggregates a set of related Business Objects into coarse-grained entity bean implementations.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You want to transfer multiple data elements over a tier.*

Use a Transfer Object to carry multiple data elements across a tier.

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You want to obtain an application model that aggregates transfer objects from several business components.*

# BUSINESS PATTERNS

Business Delegate

Service Locator

Session Facade

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler

*You have a remote client that wants to iterate over a large results list.*

Use a *Value List Handler* to search, cache the results, and allow the client to traverse and select items from the results

# BUSINESS PATTERNS

**Business Delegate**

**Service Locator**

**Session Facade**

Application Service

Business Object

Composite Entity

Transfer Object

Transfer Object Assembler

Value List Handler