Adv. Java means DURGA SIR..

ADV.JAVA

With

SCWCD / OCWCD

Design Patterns Material

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students
for last 14 years across INDIA

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Java EE Patterns

1) DTO
2) MVC
3) Intercepting Filter
4) Front-Controller
5) Business Delegate
6) Service Locator
7) DAO
8) DAOFactory

What is Design Pattern?

A software design pattern is repeatable solution for commonly occuring software problems , it provides a standard process to design application.

1) DTO (Data Transfer Object) Design Pattern

Context: In Distributed applications client & server's are locating at remote locations and they have to communicate via network.

Problem: Every call between client & server is a remote method call if client application calls individual getter & setter methods to retrieve and update values then number of remote calls is required as many as number of attributes and impacts performance of the System.

Solution:

Create an object to encapsulation all attribute values that are required by the client application, This object is called Data Transfer Object.

When the client request data from the server , server side components getters data values and construct Transfer objects by setting its data values, this object send to the client , client uses this object to query all the required data values hence the transfer object act as a proxy for the properties of remote object.

Consequences & Implications:

- DTO design patterns reduces network traffic between client & server .
- It minimizes coupling (dependencies)
- The main advantage of design pattern is we can prevent network overhead problems and we can improve performance of the system.
- The main limitation of DTO design pattern is Update information may not be available to the client side.

Key-Words:

- Reduces network traffic
- Improve responsibility
- Transfer data across Tiers
- Group information

2) MVC (Model View Controller) Design Pattern

Context:

- In the System involving the user interfaces the following situations typically arises.
- The System has to accept data from the users.
- Update Database and return the same data to the user at later point of time, there are several ways data can be accepted and several data can be presented to the user.

Problem:

If the system deploys a single component that interacts with a user and maintains Database then a requirement to support a new type of display or view will requires redesign of entire component.

Solution:

The solution is separate data presentation from data maintenance and have 3rd component that co-ordinate first 2 components and these 3 components are called Model, View , Controller.
It is responsible to keeping the data or state of the application; it also manages storage and retrieval of data from Database.

View:

It Contains presentation logic, it displays data from the model to the end-use. It also allows the user interact with the System.

Controller:

It manages whole show , It initiate the model & view and associates view with model dependency on application requirements , It may initiates multiple views for the same model.

Consequences OR Limitations:

- **To provides job separation hence without effecting remaining any components.**
- **To improves code maintainability.**
- **It follows loose coupling, high cohesion.**
- **It promotes re-usability of the model components for the same model we can provide multiple views.**
- **The main limitation of design pattern is it increases overall complexity of the application. And it is not suitable for small scale applications.**

Key-Words:

- **Separation concerns.**
- **Provide services to different clients, web clients, remote clients.**
- **Multiple Views such as HTML, XML, JSP, XHTML etc.,**

3) Business Delegate Design Pattern

Problem:

Client applications are directly having a code to interact with Business component and to call business methods of Business components, If Business component details are change and to call Business methods of Business Components , if Business component details are change or communication details are change we need to disturb the code of client application.

Solution:

Work with Business Delegate class , This is java class that separate the code of interacting with Business components from client applications , having 2 benefits . In feature if business components details are modify we just need to modify business delegates class. There is no need to modify all the client application code. The logic of Business Delegate class can be used by multiple client applications having re-usuability to interact with Business Components.

Business Delegate Features :

- **Business Delegate class act as a proxy implementing remote interfaces.**
- **Initiates communicates with remote service .**
- **Handlers communication details and exceptions.**
- **Receives request from controller components .**
- **Translate the request forward to the business service(via stub).**
- **Translate the response and returns to the controller components.**

Business Delegate Principles :

- **Business Delegate based hidding complexity.**
- **Coding to interfaces.**
- **Loose coupling.**
- **Separation concerns.**

  - **Minimizes the impact of web-tier when changes occurs in Business Tier.**
  - **Reducing the complexity between the Tier or Layers.**
  - **Add a layer to the application which increases complexity.**

Distributed web-application:

Distributed applications are location transparency because take the support of registry software location transparency means client application can dynamically recognizes the location classes of server applications registry software can manage object & object reference having lias name (or) nick name.

In distributed web-applications client always tells about JNDI registry to get Business Component reference and it uses that Business Component reference to call the Business method of Business object/component.

4) Service Locator Design Pattern

Problem:

If multiple applications of a project wants to interact with same business Component of distributed application then they need to interact with JNDI registry to get same Business object reference , in the process each client application should interact with registry s/w separately to get same business object reference which impact performance due to network traffic s/w registry and client applications. Here all the 3 client applications are interacting with registry s/w over the n/w getting the Business object reference through n/w .

Solution:

Develop Service Locator class having a load buffer to maintain Business object reference gathering from registry and supply business object reference to client applications ,these reduces n/w traffic between client applications of a same project and registry s/w .
With respect to the diagram client application makes Service Locator to get Business object reference from JNDI registry and keeps that business object reference in local buffer of Service Locator where as in remaining client applications directly gather Business object reference from local buffer of Service Locator . To make all clients working with single buffering Service Locator should be taken as singleton java class. The buffer in Service Locator is nothing but collection framework data structure like HashMap having capability to store Business object reference along with nick name or alias name.

Service Locator Features :

- **Obtain InitialContext object.**
- **Perform registry LookUp.**
- **Handles communication details and exceptions.**
- **Can improve performance by cache previously obtained reference.**
- **Works with a variety of registry softwares such as JNDI , RMI, UDDI , COS .**

Service Locator Principles :

- **The Service Locator is based on hidding complexity.**
- **Separation of concerns.**
- **Minimizes the impact on web-tier when remote component location change .**
- **Reduces coupling between the Tiers.**

5) Front Controller Design Pattern

Context:

The Presentation Tier request handling mechanism must control and co-ordinate processing of each user across multiple requests such control mechanism may be managed either in centralized or decentralized manner.

Problem:

When a user accessing view component directly without goin through a centralized mechanism each view is required to provide it's own system services which may result duplicate code.

Solution:

Use Front Controller design pattern to gather common redundent request processing code into a single component, this allows the applicationto be more cohesive , less complex.

- **If servlet act as Front Controller then it is called FrontControllerServlet.**
- **If Jsp act as Front Controller then it is called FrontControllerJsp.**
- **FrontControllerServlet , FrontControllerJsp must be configured in web.xml file by using either direct match URL (or) extension match URL pattern.**
  Eg1 : <url-pattern> *.do </url-pattern>
  Eg2 : <url-pattern> /raja/abc/test/* </url-pattern>

Consequences and Implications:

- **Centralized Controller of request processing.**
- **Improves maintanability**
- **Promotes Re-usuability**

6) Intercepting Filter Design pattern

Context:

Client request may have many different processing units and System/application must able to meet them .
The system has to receive the request by multiple protocols like HTTP, FTP , SMTP etc.,
The system must authorize and authenticate each request .
The needs to add or remove information before request further processing .

Problem:

Request and response needs to be pre and post process in simple organized manner , this pre-processing needs to be perform without effecting rest of the system i.e., it should be plugable .

Solution:

Use Filters to pre-process the request and post-processing the response.
Configuring or removing Filters takes place without effecting rest of the system.

Consequences or Implications:

- Intercepting Filter is based on Cohesion , loose Coupling , increasing declarative control.
- Declarative Control allows Filters can be easily to be implemented either on temparary bases or permanently bases .
- Declarative Control allows the sequences of invocation to be easily Updatable .

Key-Words:

- Central object for pre-processing request and post-processing response.
- Authentication of each request.
- Authorization of each request.
- Compression of the response .
- Encryption of the response.
- Altering request and response information (Wrappers)

Filter life cycle can be managed by Web-Container.

7) DAO (Data Access Object) Design Pattern

Problem:

In mvc2 and other projects , if persistence logic is mixed with other logic of the applications a specially (business logic OR persistence logic) persistence logic will not become flexible to modify that means modification done in persistence logic may disturb other logic of the application.

Solution:

Work with DAO , the java class that separates persistence logic from other logics of the application to make persistence logic as a flexible logic to modify such type of classes are nothing but DAO classes.

DAO class containes the following logics.

Logic to establish the connection
Logic to release the connection
Logic to perform persistence operations like insert , update , delete , select and etc.,
It also contains transaction managemant code and etc.,

Note: It is always recommended to write separate DAO class for each business component.

Design Pattern	Application Tier
DTO	Business Tier
Front Controller	Presentation Tier
Intercepting Filter	Presentation Tier
Business Delegate	Business Tier
Service Locator	Business Tier
MVC	Presentation Tier & Business Tier
DAO	Business Tier