Implementation of Medicine Sales by Using J2EE Design Patterns

Mya Myintzu Aye, Nan Si Kham
University of Computer Studies, Yangon
myintzzulay@gmail.com, nansikham@gmail.com

Abstract

Software design patterns describe solutions to specific software design problems. The usefulness of design patterns is simple to write down and catalogue common interactions between objects that programmer have frequently found useful. This system uses eight design patterns, in which Intercepting Filter pattern (log in), View Helper and Composite View patterns (helping to get rich interface), Front Controller pattern (customer, update, shopping cart, browse, payment) and Dispatcher pattern (use in dispatch to another page) are used in the Presentation tier. Transfer Object pattern (transferring data from one tier to another tier), Session Façade pattern (shopping view cart, login) are used in the Business tier to store data just in moment. Data Access Object (DAO) pattern (reduce redundancy coding to connect with database) is used in the Integration tier. The whole system is based on the model-view-controller patterns. The design objects are implemented the classes using Java programming language and SQL server.

Keywords: J2EE Design Patterns, Java 2 Enterprise Edition, Medicine Sales, E-commerce.

1. Introduction

J2EE (Java 2 Platform, Enterprise Edition) is designed to provide server-side and client-side support for developing distributed, multitier applications [4]. The advantages of multitier architecture are promoting software reusability, easier system maintenance and more effective use of data and networks. Most web-based enterprise applications are divided into three logical tiers, Presentation Tier, Business Logic Tier and Integration Tier.

Presentation tier encapsulates all presentation logic required to service the clients that access the system and which aims at presenting the business information to the user, is implemented using Servlets, JSPs, HTML and other pages.

Business tier provides the business services required by the application. The tier contains the business data and business logic. Typically, most business processing for the application is centralized into this tier.

Integration tier, which represents different kinds of legacy systems, database servers, etc are usually access through the JDBC API and other standard interfaces provided by the J2EE Connector Architecture [3].

In this paper, Section1 is the introduction of the system, and Section2 presents related work of the system. Section3 describes background theory. And then, Section4 describes Patterns and their related problem and solution. Section5 is the implementation of Design pattern classes. And then Section 6 is Overview of proposed system. Finally, Section7 is conclusion of the system.

2. Related work

Java EE has been the platform of choice across industries (banking, insurance, retail, hospitality, travel, and telecom, to name a few) for developing and deploying enterprise business applications. This is because Java EE provides a standard-based platform to build robust and highly scalable distributed applications that support everything from core banking operations to airline booking engines. But endeavours have failed as well. There are several reasons for such failures, of which the foremost is inadequate design and architecture. However, Java EE designers and architects have learned their lessons from both failures and successes by drawing up a list of useful design patterns.

Building applications using design patterns are easy to maintain, reuse, and extend [2]. In this system, Interfaces can be used to implement patterns such as Intercepting Filter, Front Controller, Dispatcher View, View Helper and Composite View is to be used. Business logic can be used to implemented patterns such as Session Façade and Transfer Object are to be used. Storing data can be used to implement patterns such as Data Access Object, etc are to be used.

The systems concerned with online application have been proposed and developed using J2EE

design patters techniques similar to our proposed system in this paper. Zhiguo Guo implemented with Design Patterns for J2EE Architecture and Patterns in Enterprise Systems [7]. A Pattern-Based J2EE Application Development Environment paper using framework and J2EE design patterns was implemented by Imed Hammouda and Kai Koskimies [3].

3. Background Theory

A design pattern is general reusable solution to commonly occurring problem in software design. The 21 Patterns are included in J2EE design patterns.

In which, Intercepting Filter Pattern, Front Controller Pattern, Context Object Pattern, Application Controller Pattern, View Helper Pattern, Composite View Pattern, Service to Worker Pattern, Dispatcher View Pattern are included in Presentation Tier. Business Delegate Pattern, Service Locator Pattern, Session Façade Pattern, Application Service Pattern, Business Object Pattern, Composite Entity Pattern, Transfer Object Pattern, Transfer Object Assembly Pattern, Value List Handle Pattern are included in Business Tier. And then, Data Access Object Pattern, Service Activator Pattern, Domain Store Pattern, Web Service Broker Pattern are included in Integration Tier.

4. J2EE Pattern: Problem and Solution

For any software, applying design patterns requires asking whether there are certain types of problems which occurred with regular frequency, which could be solved in the same way routinely.

This system uses eight design patterns, in which Intercepting Filter Pattern, Front Controller Pattern, View Helper Pattern, Composite View Pattern and Dispatcher View Pattern are used in the Presentation Tier. Transfer Object Pattern and Session Façade Pattern are used in the Business Tier. Data Access Object (DAO) Pattern is used in the Integration Tier [5].

These are the problems before using the design patterns and to overcome these problems after using these design patterns solution.

4.1 Presentation Tier Patterns

4.1.1 Intercepting Filter

Request and responses need sometimes to be processed before being passed to handlers and other clients. An example of request processing is form validation, user authentication. The solution is to create pluggable filter to process common logic without requiring changes to core request processing which improves code reusability and decouples request handlers.

4.1.2 Front Controller

The system requires a centralized access point for request handling. Having control code in numerous places is difficult to maintain and a single code change might require changes be made in numerous places. Use this pattern as the initial point of contact for handling all related requests. The Front Controller centralizes control logic that might otherwise be duplicated, and manages the key request handling activities [1].

4.1.3 Dispatcher View

The system needs to handle the flow of request and the navigation between views. In particular, the system needs to know which view to dispatch next based on the request. Use this pattern encapsulates page selection. In its simplest form, the dispatcher takes some parameters from the request and uses them to select actions and a view [6]. Business processing, if necessary in limited form, is managed by the views.

4.1.4 View Helper

View is used for content presentation, which may require the processing of dynamic business data. Business logic should not be placed within views. Use this pattern, views to encapsulate view-processing logic (e.g. JSP tags) and helpers to encapsulate formatting business logic code. It makes the application more modular and promotes code reuse.

4.1.5 Composite View

The problem is to keep multiple independent views increases problem of maintainability and reusability. Use composite views that are composed of multiple atomic sub-views. Each sub-view can be included dynamically in the whole, and the layout of the page can be managed independently of the content.

4.2 Business Tier Patterns

4.2.1 Session Façade

The communication between presentation layer and business layer in distributed business applications often leads to tight coupling between clients and the business tier. The interaction could get so complex that maintain the system become difficult. The solution to this problem is to provide a simpler interface that reduces the number of business objects exposed to the client over the network and encapsulates the complexity of this interaction.

4.2.2 Transfer Object

The client needs to transfer multiple data elements over a tier. The solution is to use a Transfer Object to carry multiple data across a tier with a single method call is used to send and retrieve the needed data.

4.3 Integration Tier Pattern

4.3.1 Data Access Object (DAO)

Many real-worlds applications need to use persistent data at some point. For many applications,

persistent storage is implemented with different persistent storage mechanism. Other applications may need to access data that resides on separate system. The solution is to use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with data source to obtain and store data [1].

5. The Design Patterns Classes which are used in Medicine Sales Application

In this session, we present eight design patterns classes that are implemented in Medicine Sales business application.

5.1 Intercepting Filter Class

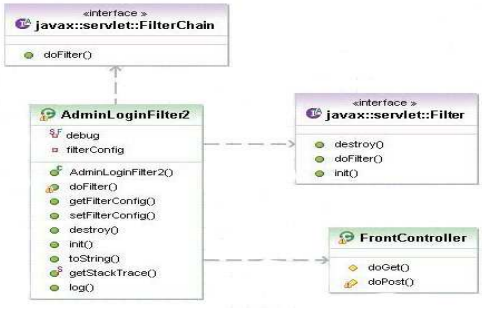

Figure 1. Intercepting Filter Pattern Implementation class

Figure 1 shows the implementation of Intercepting Filter Class. By implementing the method of Filter Chain and Filter interface, we get the AdminLoginFilter2 class to control and filter the user request. To be complete login process, the user request needs to have user name and password field. So AdminLoginFilter2 class validate the request whether to pass the parameter, if not reply the error message to the client side then go to target (Front Controller) to be complete login process.

5.2 Front Controller Class

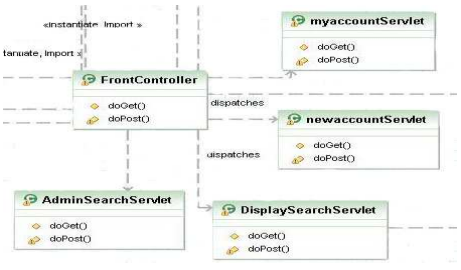

Figure 2. Front Controller Pattern Implementation Class

Figure 2 shows the implementation of Front Controller Class. Every request from the client side must through the FrontController class to be centralized system. So the FrontController class is the main control of the every classes in server side. It makes the system to check the log easily and to add the new rule for security easily. FrontController class as a Servlet to handle the request from the client to control code that is common across multiple requests is duplicated in numerous places, such as within multiple views. The FrontController Class also helps reduce the amount of programming logic embedded directly in the views.

5.3 Dispatcher View Class

By implementing the RequestDispatcher interface, we get Dispatcher View Class. According to the application, carrying the parameters through the one server side class to another, we have to use forward method of Dispatcher class.

5.4 View Helper Class

To be View Helper design pattern, needs simple java class and java Bean(POJO) as helper to support the view(JSP). It makes to improve maintainability and better reusable code. Views are separated from the processing logic. In this, views (JSP) don't contain business and system logic. It makes to improve maintainability and better code reuse.

5.5 Composite View Class

Developing and maintaining dynamic views are challenging, since there are often aspects of both the view content and the view layout that are common across multiple views. When content and layout are intertwined, it is harder to maintain and extend the views. Reuse and modularity also suffer when common code is duplicated across views. So,each sub view(JSP) class of the overall template can be included dynamically in the whole, and the layout of the page can be managed independently of the content.

5.6 Session Façade Class

The main role of web application development is using Session tracking. By implementing the HttpSession interface, we get the Session Façade class to store the data in memory for traffic and complexity saving. By storing the parameter to session class, we can get the stored parameter from everywhere of the system until removing from the session. It hides the real implementation of the business logic and exposes only the required interface to the client.

5.7 Transfer Object Class

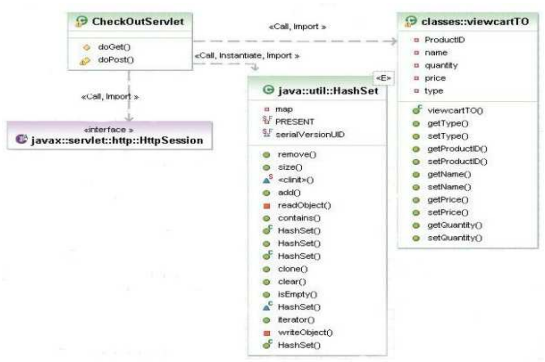

Figure 3. Transfer Object Pattern Implementation Class

Figure 3 shows the implementation of Transfer Object Class.Transfer Object class to encapsulate the business data transferred between the client and the business components. Instead of invoking multiple getters and setters for every field, a single method call Transfer Object class is used to send and retrieve the needed data.

5.8 Data Access Object (DAO) Class

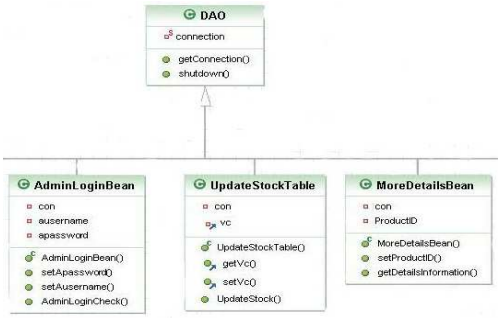

Figure 4. Data Access Object Pattern Implementation Class

Figure 4 shows the implementation of Data Access Object Class. DAO enables loose coupling between the business and resource tiers. DAO encapsulates all the data access logic to create, retrieve, delete and update data from a persistent store. DAO uses Transfer Object to send and receive data.

6. Overview of Proposed System Design

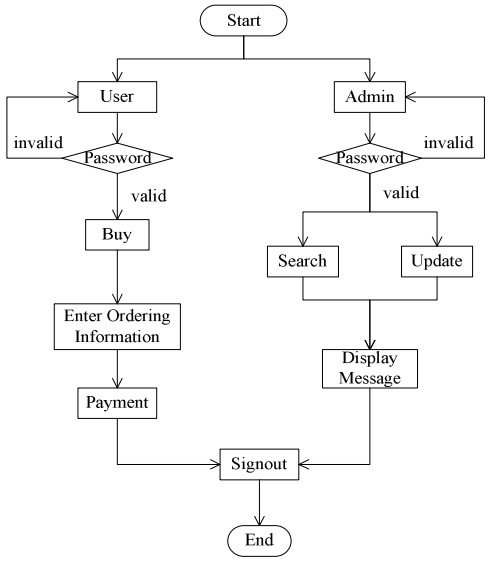

Figure 5. Proposed System Design

Figure 5 shows the overview design of proposed sysetem. In this system, there are two parts, User and Administrator. In User part, User can know about the medicine information. If user wants to buy some medicines, user will need to login. If user didn't login, user would need to signup. After that, fill about the ordering information and valid credit card number. Finally, user needs to be sign out. In Administrator part, Admin can add new medicine and update about the medicine. Before add or update the medicine, admin must need to login with valid password. After adding or updating about the medicine, admin needs to be sign out.

6.1 J2EE Design Pattern Relationship in this System

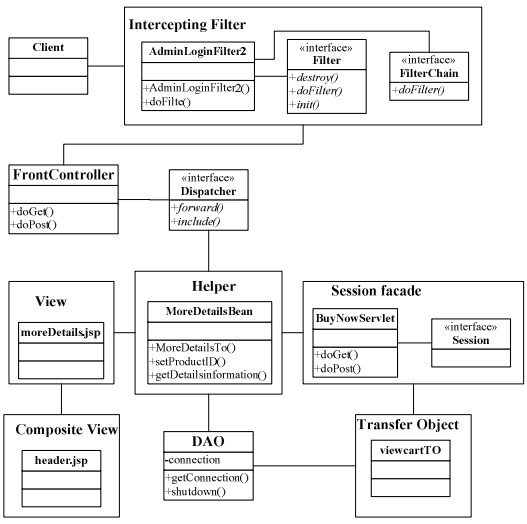

Figure 6.J2EE Patterns relationship in this system

Figure 6 shows the J2EE Patterns relationship in this system. AdminLoinFilter2 intercepts incoming requests and outgoing responses and applies a filter. For incoming request, Front Controller is used not only to control the desire view but also to hold the common processing logic that occurs within the presentation tier and that may otherwise be erroneously placed in a View. A Front Controller handles requests and manages content retrieval, security, view management and navigation, and then delegating to a Dispatcher component to dispatch to a View Helper. View component encapsulates the presentation formatting. Helper component encapsulates business logic. View may be composed of multiple sub-components to create its template using Composite View.

Session Façade encapsulates the complexities of the business object interaction. The Session façade may also use other patterns to provide its services: Transfer Object, Business Delegate and Service to Worker, etc. The Transfer Object provides best techniques and strategies to exchange data across tier and then attempts to reduce the network overhead by minimizing the number of network calls to get data from the business tier.

The Data Access Object pattern provides loose coupling between the business and resource tiers. The Data Access Object is intercepted from all access for the resource tier, making the implementation details of the resource tiers transparent to the clients. The data in the resource tier can reside in database systems, proprietary systems, other external systems and services. By using these patterns, user can build applications these are more flexible and portable [8].

7. Conclusion

This application was divided into three tiers (presentation tier, business tier and integration tier) according to the distributed responsibilities; Web and container were required. With the development of enterprise software, the current J2EE architecture and patterns will be much widely applied or adapted to build robust enterprise systems. This pattern-based system can significantly ease the development process of J2EE applications and reduce the designing time. The advantages of applying design patterns in this application will be solved the recurring problem, reusability, improved performance and increase maintainability, etc.

8. References

[1] Deepak Alur, John Crupi and Dan Malks "*Core J2EE™ Patterns: Best Practices and Design Strategies*", Second Edition

[2] Dhrubojyoti Kayal, "*Pro Java EE Spring Patterns, Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework*", Apress

[3] Imed Hammouda and Kai Koskimies, "*A Pattern-Based J2EE Application Development Environment*" paper

[4] Inderjeet Singh, Beth Stearns, Mark Johnson, and the Enterprise Team, "*Designing Enterprise Applications with the J2EE Platform, Second Edition*", Addison-Wesley

[5] "*J2EE Patterns Overview"* downloaded PDF file

[6] William Crawford and Jonathan Kaplan, "*J2EE Design Patterns*"

[7] Zhiguo Guo, "*J2EE Architecture and Patterns in Enterprise Systems"*, Master's Thesis, 2004

[8] http:// www.dibsden.com (Online document)