Online Market Place Application

Assignment #3: Report

Implementation using Authorization Pattern, Proxy Pattern and Reflection Pattern

Course: CSCI 50700 - Object-Oriented Design and Programming

By
Mani Manjusha Kottala
IUPUI, Computer Information Science

Table of Contents

Introduction:

In previous assignment 1, an application is developed using (MVC) Model-View-Controller and Java RMI. In assignment 2, implemented the login functionality for user, admin and to use Front Controller pattern, Command pattern and Abstract Factory pattern. Now for this assignment, three more patterns Authorization, Proxy and Reflection are used to integrate role-based access to the application.

Assignment 2 – Fix:

In assignment2, I was asked to indent the code properly in one java source file - This comment is fixed. Another comment was if its ok to return null by getUserView in AdminFactory - In the AdminFactory class, the getUserView will never be used. When we extend the AbstractFactory class, its abstract methods should be implemented here. So, as it never be user I am returning null for the User Object.

Authorization Pattern, Proxy Pattern, Reflection Pattern:

A user/admin should be authenticated first to perform any action on the market application. Unauthorized access to any of the functionalities should be restricted. Authorization pattern is where it comes into play to support role-based access control for an application. Java annotations and reflections can be used to achieve such feature in our application. Two roles in my application are Admin and User. An admin can't act like a User and vice versa i.e., User can only browse and purchase items, but an admin can browse, update, delete and add items to the application.

Reflection pattern, one among architectural pattern is used to support the manipulation of function calls and type structures, hence supporting communication between layers. This helps the system to inspect source code to explore more by revealing its objects, methods, properties etc., One can perform an action on an unknown object or without even knowing information about the caller. All this newly known data can be put into a meta level and the logic into base level. Reflect package in java is better revealed when implemented along with Annotations.

Authorization Pattern, one among structural pattern is used to limit an user's functionality in a distributed system. Role based access control can be achieved through implementation of this pattern. Role based access means setting access rights based on role i.e., admin or user instead of individual rights for each user and admin. To apply this to my application, few modules like User session, Front controller, Dispatcher, Client controller should be modified using annotations and above reflection pattern.

Proxy pattern will assist our application by providing dynamic proxy feature which are used together with annotations. Below is the description of classes which implement all the above listed functionality:

- Session: This is used to create a session based on each user role passed to it. getRoleType() returns the assigned role. This session is now included in each functionality that can be performed either by a user or admin.
- RoleAnnotations: User defined annotations contain a string value which stores the annotation. This is used as @RoleAnnotation("user") or @RoleAnnotations("admin") for all the methods. This means that user can access only his methods same as admin has access to only his methods. This prevents hectic work of limiting each user of market app. Other annotations used are @Override, @Retention
- Invoker: Responsible for invoking user/admin commands

- AuthorizationInvocationHandler: Invocation handler uses reflection is responsible for reflection pattern implementation. This checks for annotation role value and compares this with the session role type value and performs tasks based on the output.
- AuthorizationException: Support for user defined exceptions. Used for implementation of distributed exception handling.
- MarketPlaceClientController: Contains dynamic proxy which is responsible for RMI interaction as well as proxy pattern implementation.
- FrontController: Handler for handling requests of the application. Responsible for User authentication and session creation for a successful user/admin login. This session is carried out till the user stays active with the application.
- Dispatcher: Dispatcher is responsible for redirecting user/admin to respective view along with created session.
- ClientEntryView: This is where either admin or user enters his details.
- Admin interface and User interface: Interfaces are now appended with session object
- MarketPlaceUserView and MarketPlaceAdminView: Both the views have methods which are now implemented in Server-side model.

Class Diagram [1]:

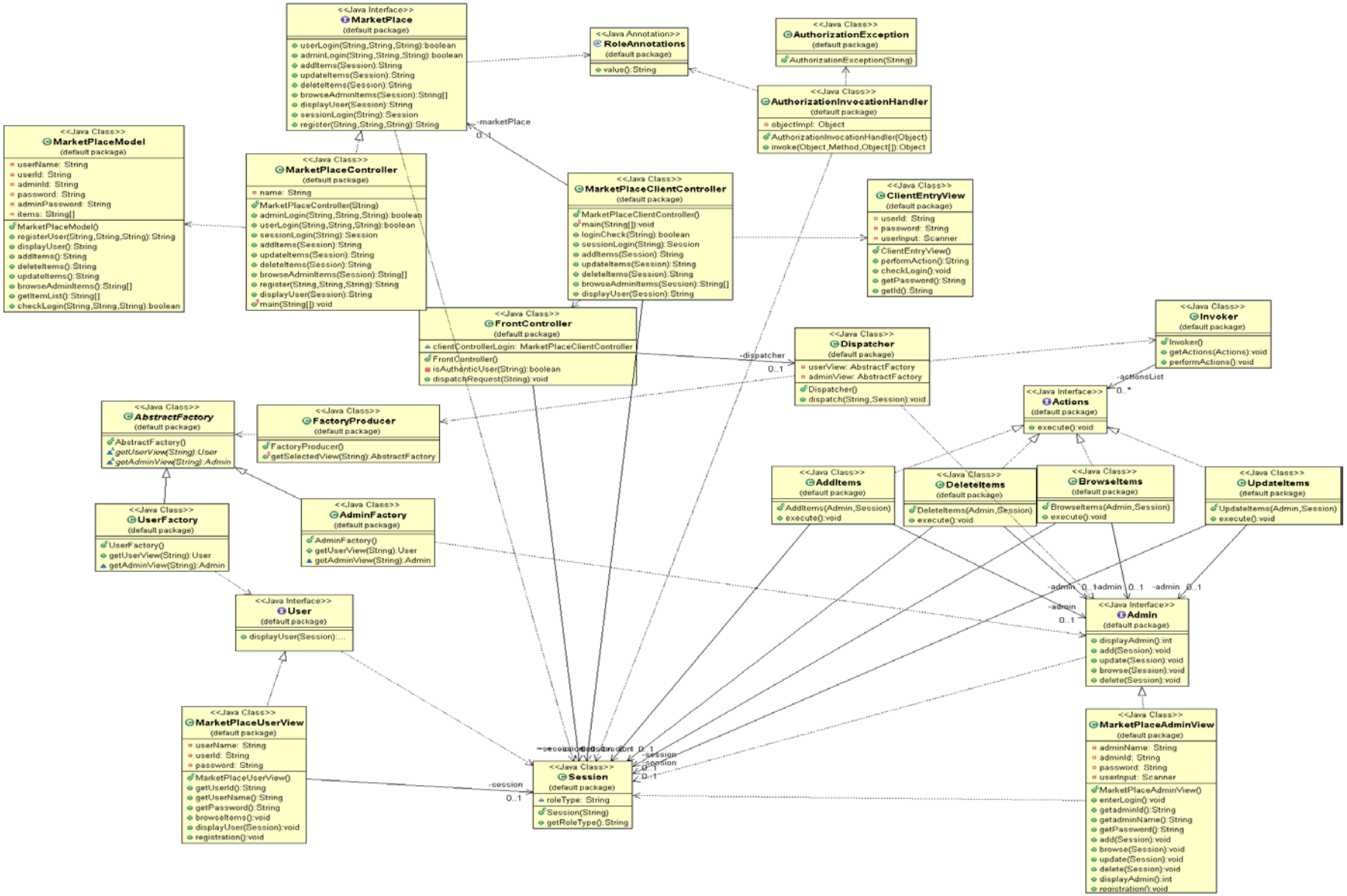

Fig 2: Class Diagram for MarketPlace Application

Sample runs:

1. Starting the RMI registry and Running the server application

```
[mkottala@tesla Assignment3]$ rmiregistry 2525&
[1] 74482
[mkottala@tesla Assignment3]$ javac *.java
[mkottala@tesla Assignment3]$ java -Djava.security.policy=policy MarketPlaceController
Creating a Server Connection!
MarketPlaceModel: binding it to name: //tesla.cs.iupui.edu:2525/MarketPlaceServer
Market Place Server is Ready!
```

Or

Using makefile: MakeControllerServer.sh

```
[mkottala@tesla Assignment3]$ rmiregistry 2525&
[1] 74032
[mkottala@tesla Assignment3]$ sh MakeControllerServer.sh
Creating a Server Connection!
MarketPlaceModel: binding it to name: //tesla.cs.iupui.edu:2525/MarketPlaceServer
Market Place Server is Ready!
```

2. Running the Client application

```
[mkottala@tesla Assignment3]$ java -Djava.security.policy=policy  MarketPlaceClientController
Enter Action as either 1 or 2:
1. User
2. Admin
```

Or

Using MakeFile: MakeClientView.sh

```
[mkottala@tesla Assignment3]$ sh MakeClientView.sh
Enter Action as either 1 or 2:
1. User
2. Admin
1
View : User
Enter Id:
mkottala
Enter Password:
mkottala
Login Checking
User authentication is successful.
Displaying User Profile
User Profile Display from Server
```

```
[mkottala@tesla Assignment3]$ sh MakeClientView.sh
Enter Action as either 1 or 2:
1. User
2. Admin
2
View : Admin
Enter Id:
manju
Enter Password:
manju
Login Checking
Admin authentication is successful.
Displaying Admin Profile
Enter Action
1.Add Items
2.Delete Items
3.Update Items
4.Browse Items
4
Browsing Items displayed here
Book
Pen
Cycle
Camera
```

3. Terminating RMI registry:

```
[mkottala@tesla Assignment3]$ fg
rmiregistry 2525
^C[mkottala@tesla Assignment3]$
```

Conclusion:

In this assignment, I have gained knowledge on implementation of Authorization pattern over previously built market application modules. Also learnt a new technology, Java annotations which can be used to build a role-based access application. On top of role-based access, I have also learnt the use of proxy and reflection pattern which supplement the functionality of role-based access.

References:

[1] https://dzone.com/articles/uml2-class-diagram-java

[2] Examples discussed in class