Online Market Place: Pattern-based Design

CSCI 50700 – Object-Oriented Design and Programming

Assignment-III

Authorization Pattern, Reflection and Proxy Pattern

Summary Report

Under the guidance of

Professor Ryan Rybarczyk

Computer & Information Science, IUPUI

By

Yashwanth Reddy Kuruganti

Computer & Information Science, IUPUI

Table of Contents

1. Introduction

This assignment aims at further integration of authorization and role-based access control over the previously constructed market place application using front controller, abstract factory and command pattern using a skeleton framework for Online Market place application, which is built in 1st assignment. Patterns that are implemented to achieve role-based control access are authorization, reflection and proxy. Each pattern has its own functionality that further improves the authentication mechanism in the market application

2. Assignment 2 Feedback

No comments were mentioned in the commentsA2 branch

3. Patterns implemented – Authorization, Reflection and Proxy

This assignment mainly focuses on creating a role-based access control to the application functional modules instead of the generic login page built in previous assignment. Through this a user with either of the roles, admin or a customer can login and access their specific views. Also, customer and admin can perform various commands like browse, update and delete etc., I have implemented these by understanding and referring InClass examples. [1]

I have implemented Authorization pattern using Java annotations along with two more patterns Reflection and Proxy using Java reflections. These patterns are integrated with Command pattern and Front controller pattern to implement the annotations interface and extend the functionalities such that customer can't access admin functionalities and admin may or may not access customer functionalities. So, role is given importance in this build instead of an individual user. Few new classes are introduced which are explained below along with the flow:

First step I have done is to implement a Session class and checked if it can be transferred over the application from the entry point. After successful creation, I have tried passing the session object using serialization. Next step is to provide user login through which a customer or admin can be granted a session after successful authentication. FrontController provides the feature of user authentication. Logic for this user authentication is written in OnlineMarketModel. When a user enters his/her details front controller takes care of this and then send a request to client-side controller(MarketViewController) and then to server-side controller(OnlineMarketController). Model contains the logic for user verification. This checks for authenticity and then send back response to client-side front controller through RMI.

Now, to provide role-based access control AnnotateInterface class is created with user defined annotations which contains a method for storing annotated value over a session. Various properties can be applied to this class to tell the annotation where to work on. This is implemented in the OnlineMarket interface over each method, which are further implemented on the server side. Annotations @Override is used to mention that superclass method must be overridden, @Retention tell when to apply the annotations and @Target tells where the annotation should work on etc., But to know the details about other classes or its object or its methods, Java reflection

should be used. This would support annotations to implement Authorization pattern. Next required steps is to convert the current RMI proxy to more dynamic proxy which can handle incoming requests in annotations environment. AuthenticateUserInvocationHandler contains code for session value and annotation value check and call respective methods based on the annotation value. If the values don't match then these errors should be caught, which is done by using a user defined exception class AuthenticateUserException. OnlineMarketController is changed in such a way that it can handle the requests using a Java dynamic proxy. Other classes like MarktetDispatcher, MarketFrontController and MarketViewController are modified to support session transfer and perform user actions in that given session.

4. Class Diagram

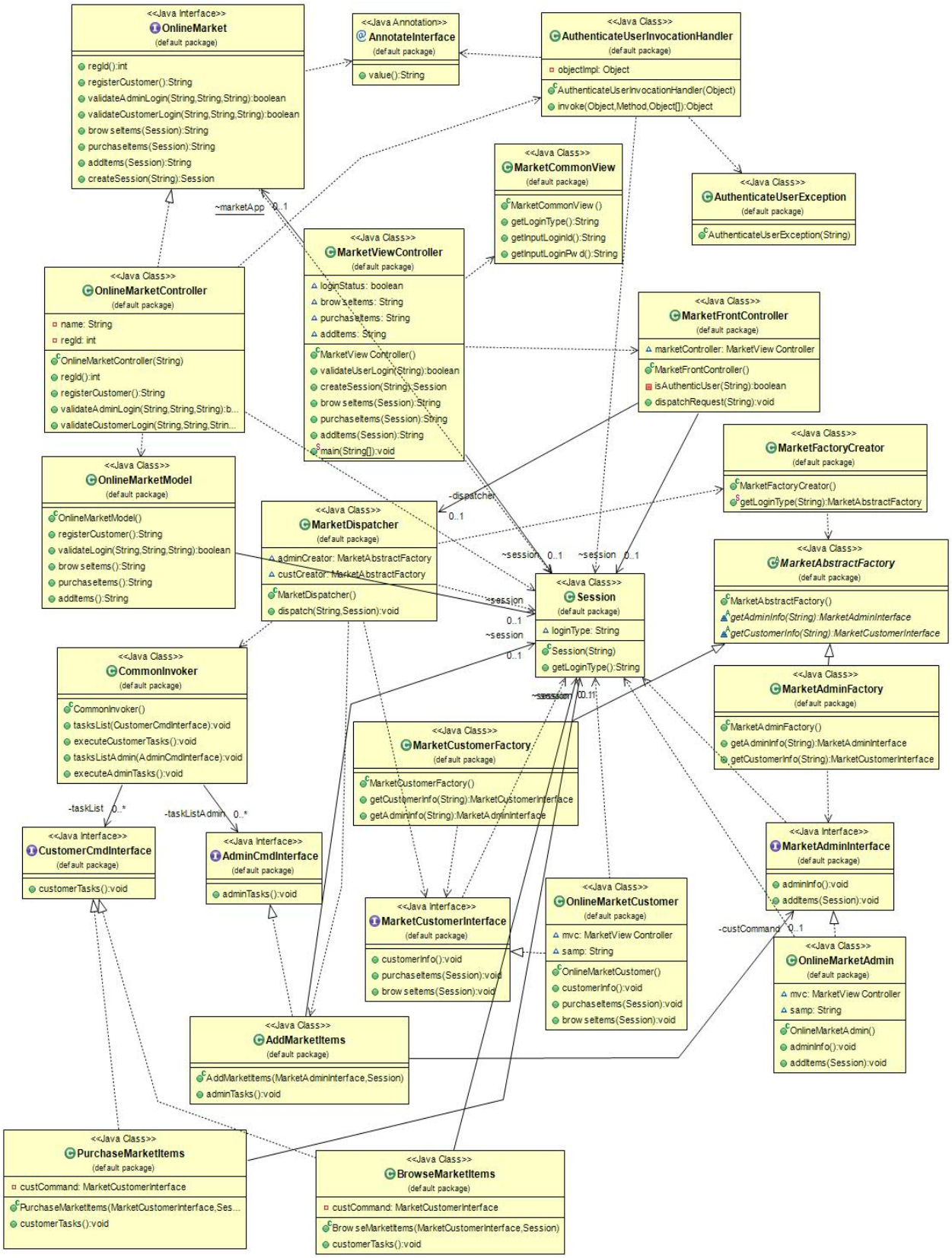
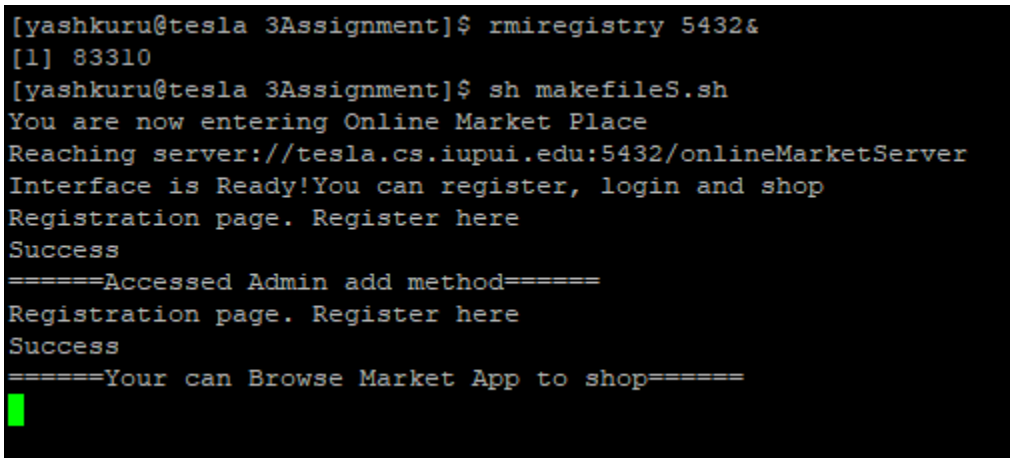


Fig 1. Class Diagram

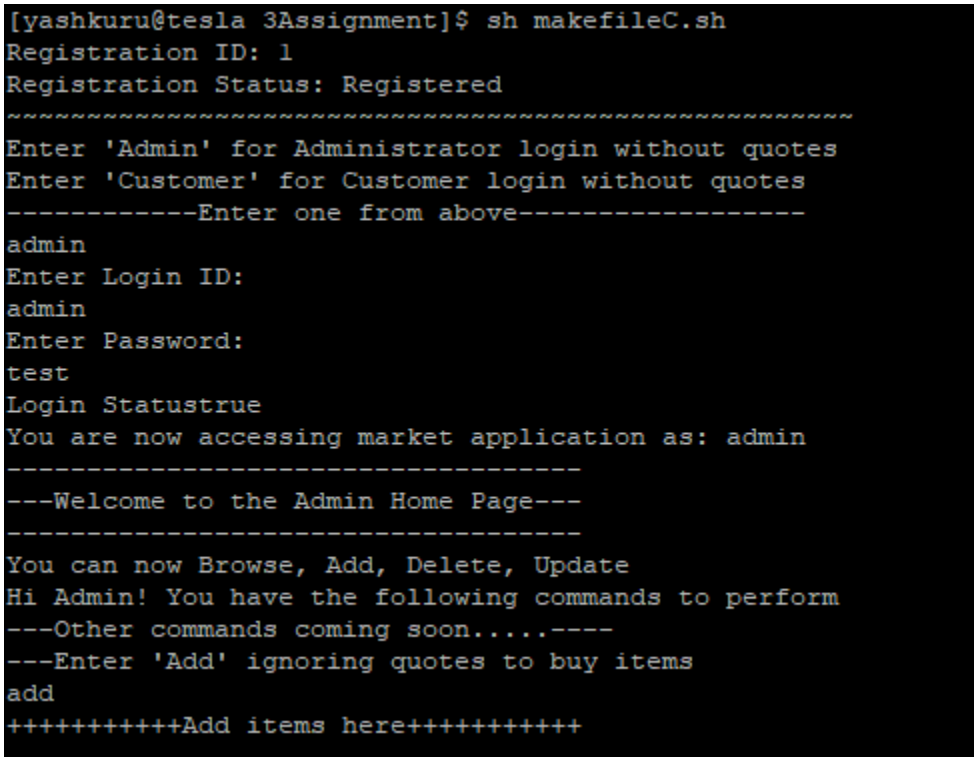
5. Sample Run

Steps on running this software is given in Readme file. Simple way of executing is to run makefiles.

- Running RMI registry and server makefile

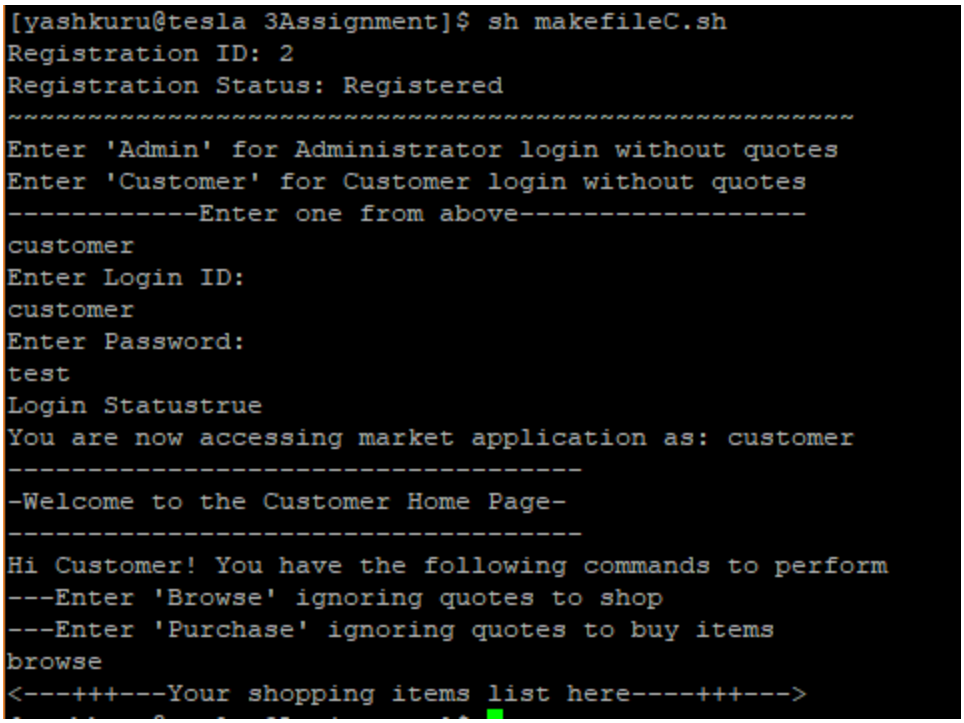


- Running client side make file



- User defined distributed exception handling example

```
[yashkuru@tesla 3Assignment]$ sh makefileC.sh
Registration ID: 1
Registration Status: Registered
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter 'Admin' for Administrator login without quotes
Enter 'Customer' for Customer login without quotes
------------Enter one from above------------------
adtga
Enter Login ID:
asdf
Enter Password:
asdf
>>>>>Input should be only from either of the above roles<<<<<<
Authorization denied for user type: adtga
[yashkuru@tesla 3Assignment]$ sh makefileC.sh
Registration ID: 2
Registration Status: Registered
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter 'Admin' for Administrator login without quotes
Enter 'Customer' for Customer login without quotes
------------Enter one from above------------------
admin
Enter Login ID:
admin
Enter Password:
test
Login Status:true
You are now accessing market application as: admin
-----------------------------------
---Welcome to the Admin Home Page---
-----------------------------------
You can now Browse, Add, Delete, Update
Hi Admin! You have the following commands to perform
---Other commands coming soon.....----
---Enter 'Add' ignoring quotes to buy items
purchase
Invalid command input
```



6. Conclusion

From this assignment, I have learnt and understood how various design patterns like Authorization, Reflection and Proxy pattern can be used to provide role based accessing to a distributed application instead of assigning rights to each user. Also gained knowledge on creating and transferring a session through the network and maintaining its life when a user is still acting on the application. Java dynamic proxy instead of RMI made the communication more robust.

7. References

[1] In-Class examples