**CSCI 50700 - Object-Oriented Design and Programming  
Assignment 6 Report**

Jingyi Hui

[hui@iu.edu](mailto:hui@iu.edu)

**Department of Computer and Information Science**

**IUPUI**

Table of Contents

[1. Assignment Overviews 4](#_Toc512606891)

[1) Assignment #1 Overviews 4](#_Toc512606892)

[2) Assignment #2 Overviews 4](#_Toc512606893)

[3) Assignment #3 Overviews 4](#_Toc512606894)

[4) Assignment #4 Overviews 5](#_Toc512606895)

[5) Assignment #5 Overviews 5](#_Toc512606896)

[2. Assignment #6 Discussion 6](#_Toc512606897)

[3. UML Diagrams 7](#_Toc512606898)

[1) Domain Model 7](#_Toc512606899)

[2) Software design diagram 7](#_Toc512606900)

[4. Sample Runs 8](#_Toc512606901)

[1) Setup server on rrpc01 8](#_Toc512606902)

[2) Setup client on rrpc02 8](#_Toc512606903)

[3) **Login function** for *admin* and *customer* roles 8](#_Toc512606904)

[**4)** **Choice 1: Browse Product Inventory** 9](#_Toc512606905)

[**5)** **Choice 2: Add Product into Shopping Cart** 9](#_Toc512606906)

[6) Choice 3: Remove Product from Shopping Cart 10](#_Toc512606907)

[7) Choice 4: Browse Shopping Cart under Username “Hui” 10](#_Toc512606908)

[8) Choice 5: Update Shopping Cart 11](#_Toc512606909)

[9) Choice 6: Add Product into Inventory 12](#_Toc512606910)

[10) Choice 7: Remove Product from Inventory 13](#_Toc512606911)

[11) Choice 8: Purchase 13](#_Toc512606912)

[12) Choice 9: Logout 14](#_Toc512606913)

[13) Choice 10: Update Product 14](#_Toc512606914)

[5. Final Conclusions 15](#_Toc512606915)

CSCI 50700 - Object-Oriented Design and Programming  
Assignment 6 Report

# Assignment Overviews

## Assignment #1 Overviews

To build the Marketplace application, we are in the situation that we need a user interface (front-end, View) and a persistent storage (back-end, Model). It’s desirable for us to separate the interface and system functionality since the UI changes more frequently than system functionality. As an architectural pattern, MVC pattern provides independency between UI and system functionality, so that there is no direct connection between UI and system data model, all the views are associated with one or more controllers that manipulate the model’s state. With this kind of separation, if we need to make changes to the UI, the evolution can be achieved without impacting other views or the model itself.

To create a distributed application, Java RMI is introduced to allow communication between remote JVMs. It allows an object running in one machine to invoke methods on an object running in machine. With this infrastructure, our server can provide service to our remote customers.

## Assignment #2 Overviews

2 different views based on different roles are implemented by Front Controller pattern together with the Abstract Factory pattern and Command Pattern. Front Controller provides centralized control for our application, it handles all the requests from users.

When user sees the LoginView, he/she will be required to input username and password. Based on the user’s information, user will be dispatched to either customer’s menu or admin’s menu created by the Abstract Factory. This is kind of a way to provide security to our application. Since different users will see different menus, which can prevent them have access to unauthorized operations.

A Request class is created to wrapped a user request under an object as command and passed to invoker (Action) object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

## Assignment #3 Overviews

We implemented an another way to provide access control – RABC, which is achieved by making use of Java Annotation. Instead of hiding invalid commands and showing different views based on roles, in this assignment, both of these 2 roles will see the same menu after login, and different roles have different authorized access to some commands.

In assignment 2, my Customer/Admin view didn’t work, since I didn’t retrieve user’s role after I validate their information. To fix the problem, a Session class is added for passing parameters (username, password, and role) for future use.

## Assignment #4 Overviews

In Assignment 4, 5 functions are implemented, and we are using 6 different machines to simulate a real distributed system. But I didn’t realize that there are some bugs (I store my inventory as an array list, and set product ID as a property of the product, but I muddled it with the array index.) in my implementation until I got my feedback.

We also explore the concurrency provided by Java RMI. I did some test runs to verify that RMI provides us concurrency. From some research I did, I got to know more about threading in RMI.

## Assignment #5 Overviews

Application functions are fully implemented in this assignment. Several bugs from Assignment 4 are fixed in this one.

MySQL database is used as the persistent storage in this assignment. We are no longer using *.txt* or *.property* files to store inventory or user information, which makes this project more like a real distributed application. *MySQL* class is added to establish the connection between my application and the database server, implement actions with MySQL queries, and works as a layer to shield my business logic from queries.

To achieve synchronization in our application, key word: synchronized is used for several methods like: Update Inventory, Purchase, Remove Inventory…to achieve thread safety. When several users are using our application, the system will apply some lock mechanism to ensure at a given time, only one thread can have the right to access that resource.

# Assignment #6 Discussion

*Database Access Layer Pattern*

I implemented the database access layer by creating a class (*MySQL.java*) of data access methods that directly reference a corresponding set of database stored procedures. This class works as an interface between my application logic and the database (hui\_db).

My application can store and retrieve data by calling an appropriate method (Figure 1) in my access layer, which decouples the core business logic and the database queries (Figure 2). If there is any change in database schema, I only need to modify this access layer.

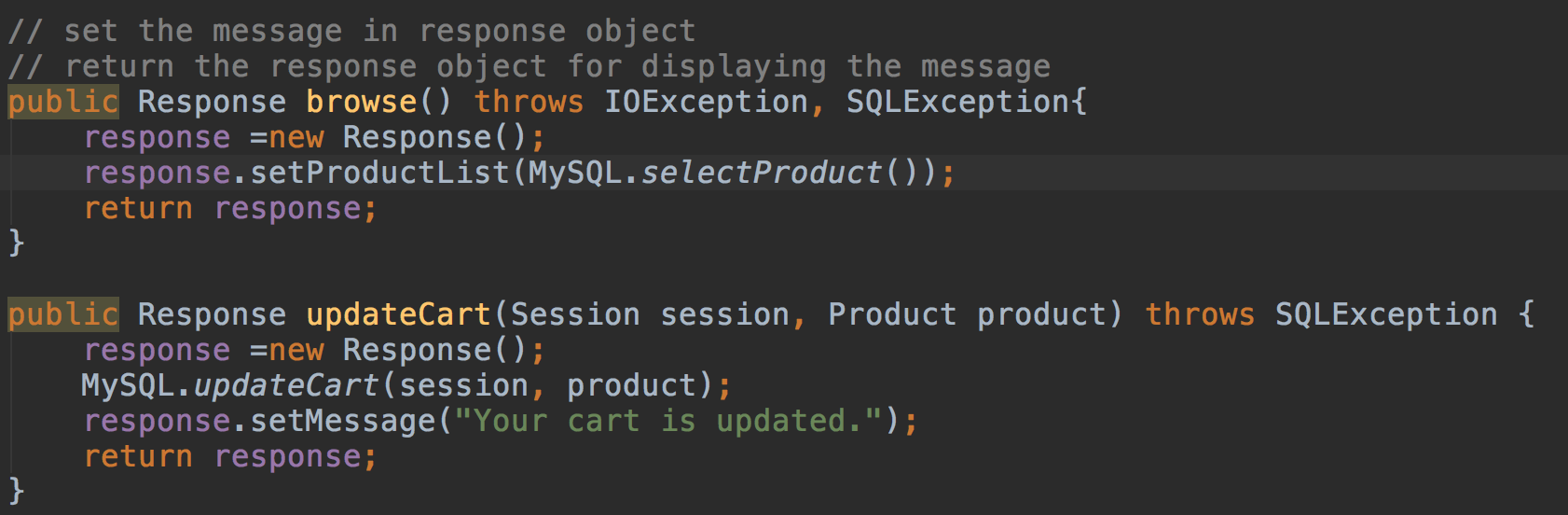


Figure 1 How the method is used in ActionServerModel class

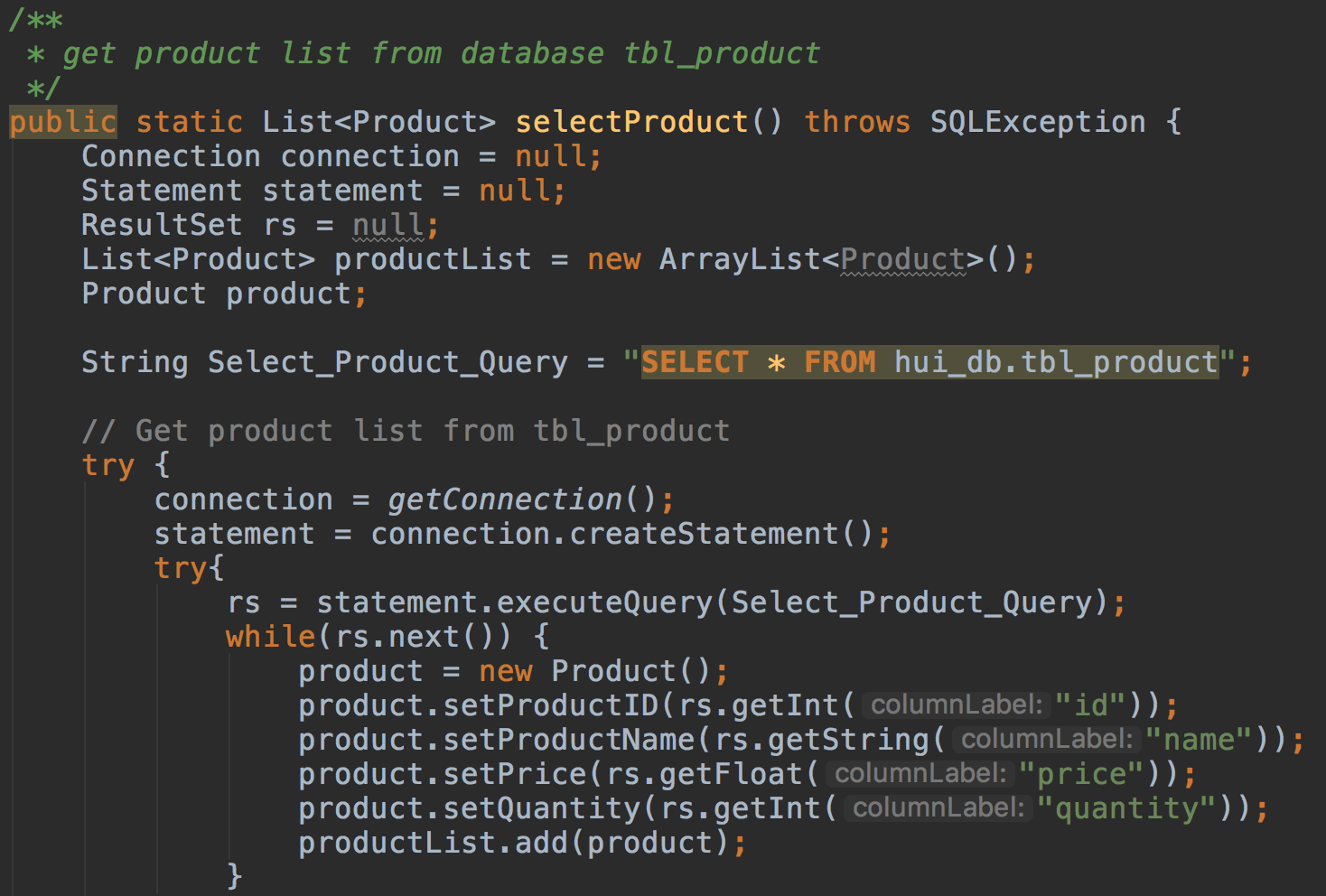


Figure 2 SQL used to retrieve product inventory in MySQL class

# UML Diagrams

## Domain Model

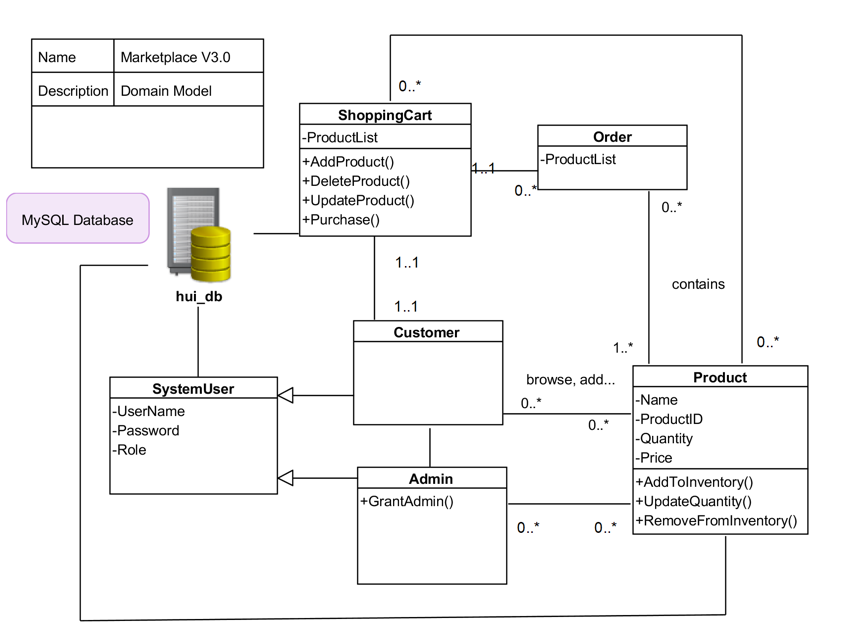


Figure 3 Marketplace Domain Model

## Software design diagram

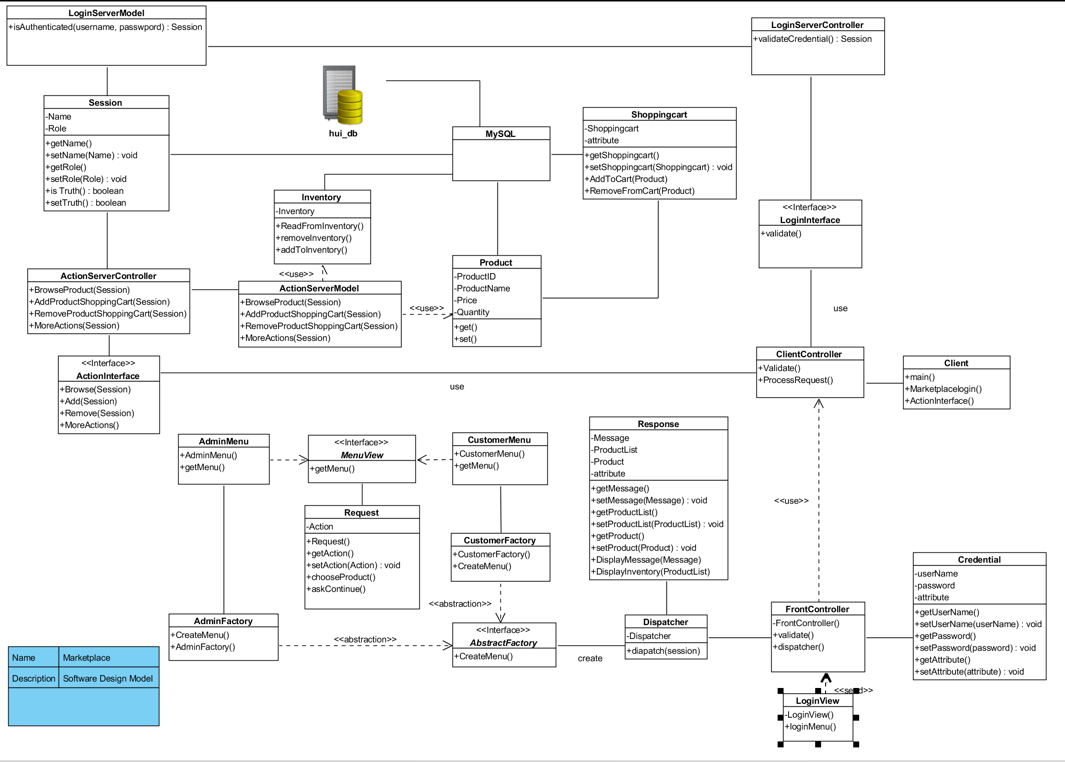


Figure 4 Marketplace Software Design Diagram

# Sample Runs

## Setup server on rrpc01

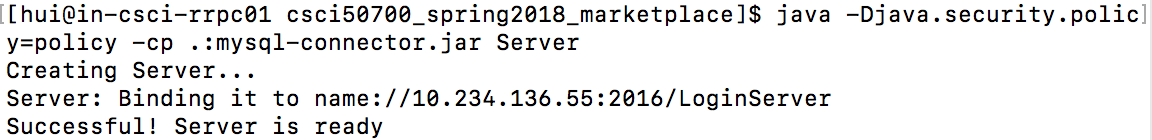


Figure 5 Setup Server

## Setup client on rrpc02

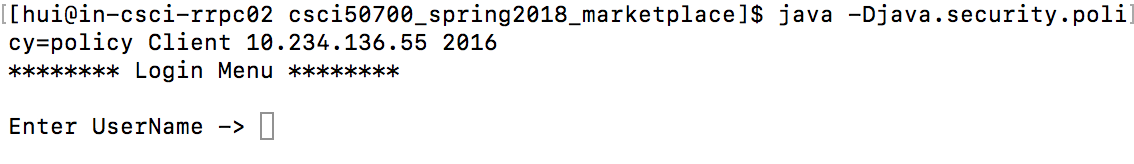


Figure 6 Setup Client

## **Login function** for *admin* and *customer* roles

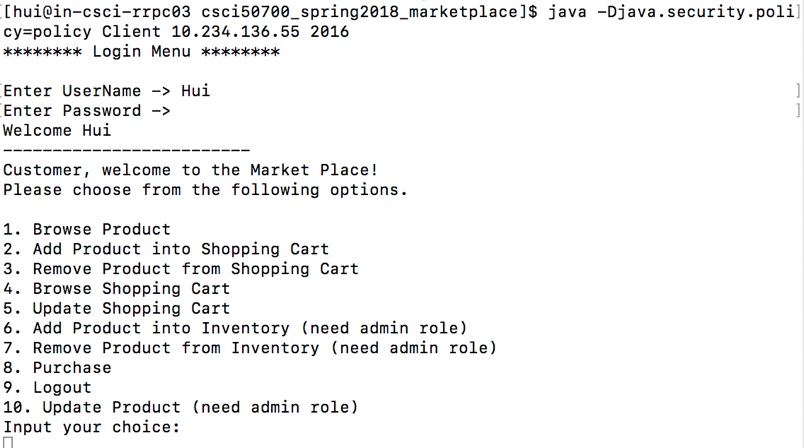
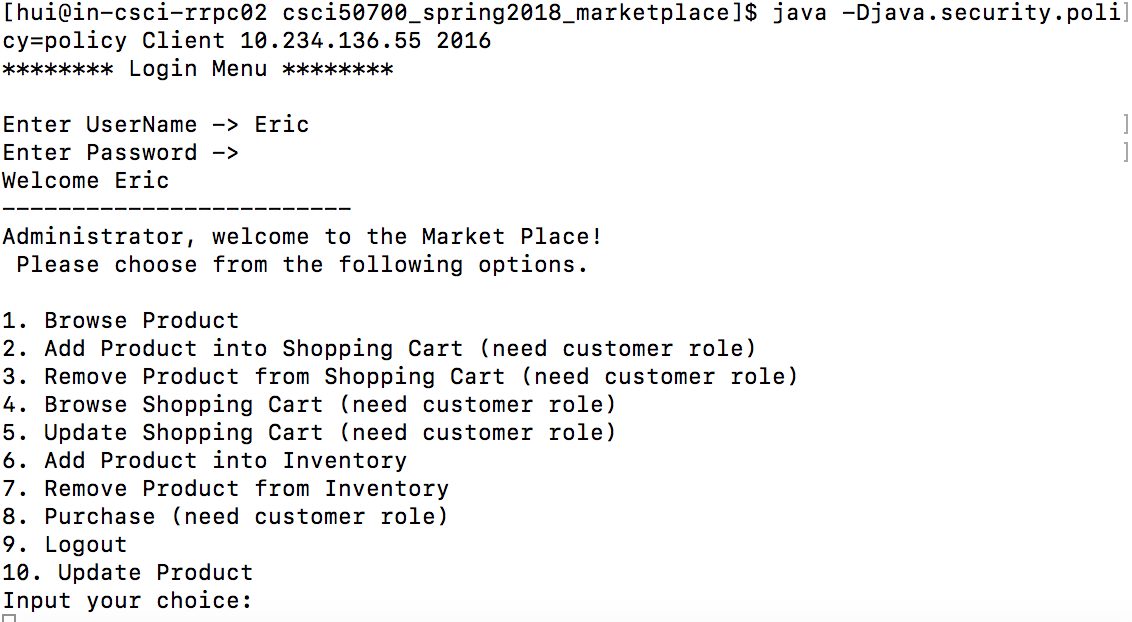


Figure 7 Login-admin view Figure 8 Login-customer view

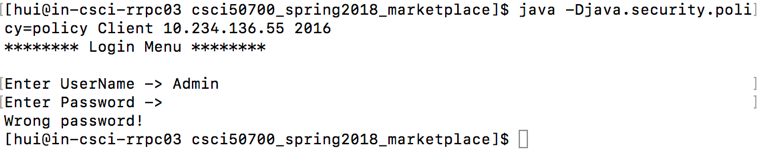
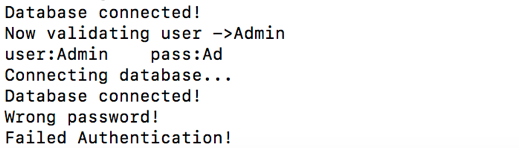


Figure 9 Login with Incorrect Password (Server Side) Figure 10 Login with Incorrect Password (Client Side)

## **Choice 1: Browse Product Inventory**

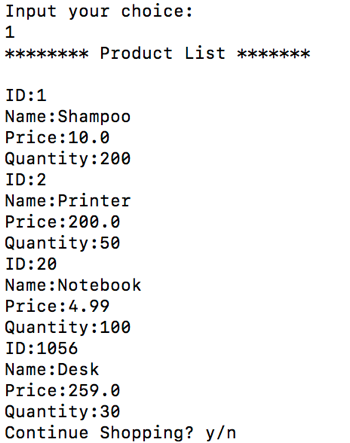


Figure 11 Browse Product Inventory

## **Choice 2: Add Product into Shopping Cart**

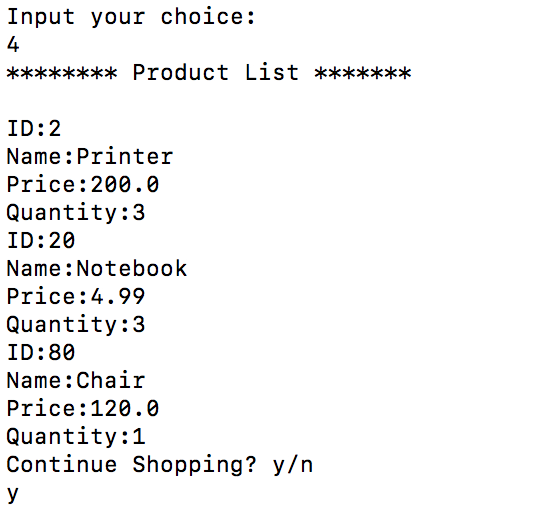
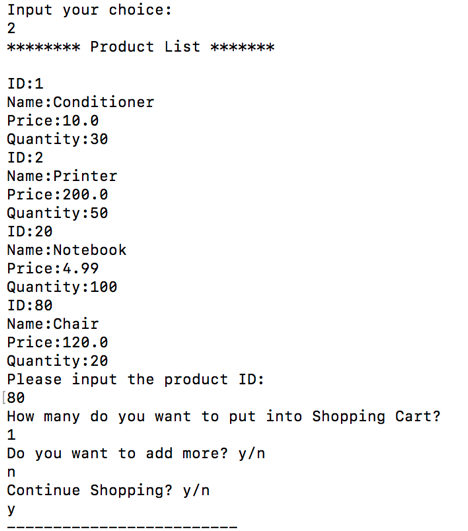


Figure 12 Add Product (ID: 80, quantity: 1) into Cart Figure 13 Updated Cart

## Choice 3: Remove Product from Shopping Cart

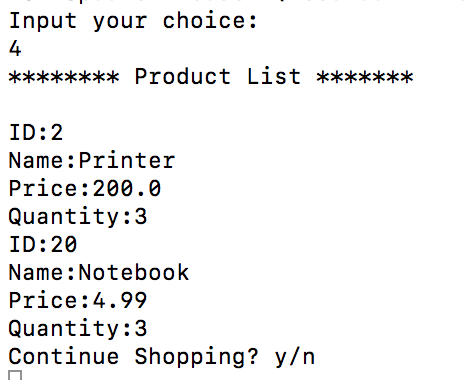
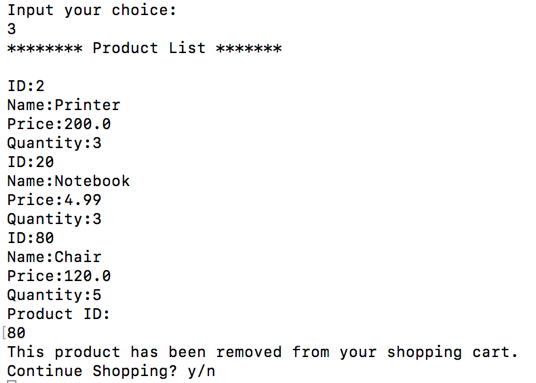


Figure 14 Before Remove Product (ID: 80) Figure 15 After Removal

## Choice 4: Browse Shopping Cart under Username “Hui”

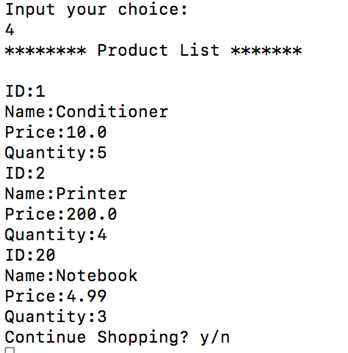


Figure 16 Browse Shopping Cart

under Username “Hui”

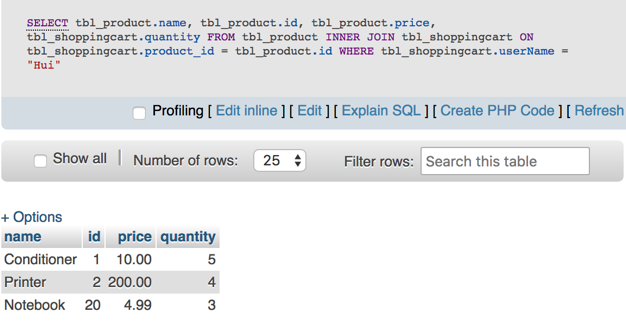


Figure 17 Browse Shopping Cart in Database under Username “Hui”

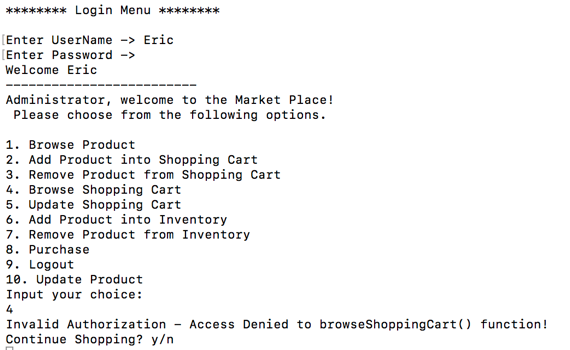


Figure 18 Browse Shopping Cart Denied under Admin Account

## Choice 5: Update Shopping Cart

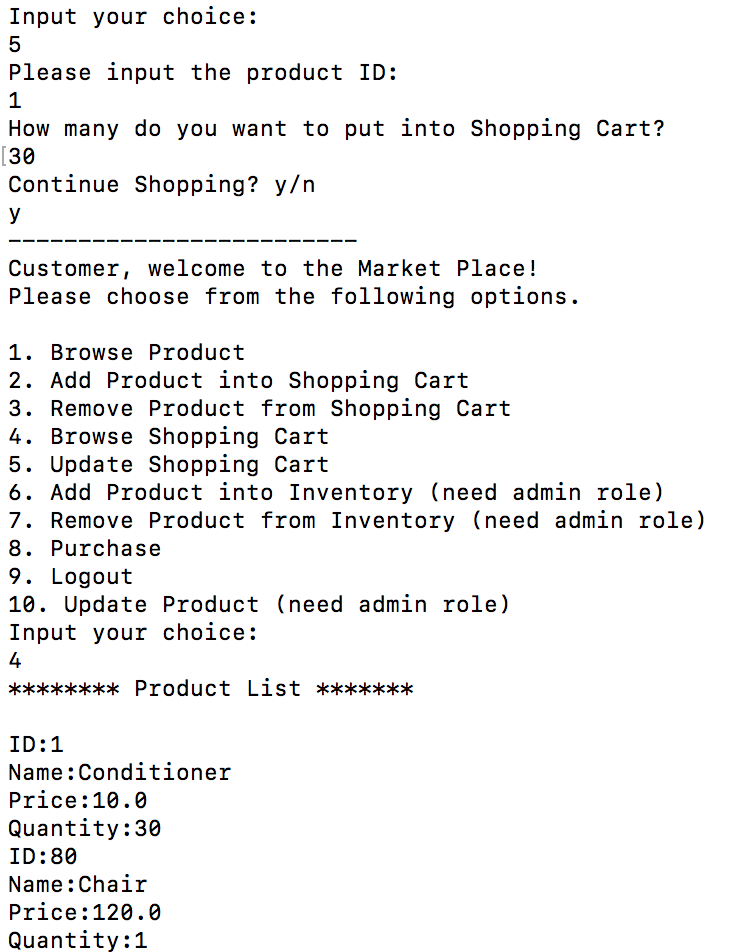


Figure 19 Update Product Quantity in Shopping Cart (ID:1, Quantity:30)

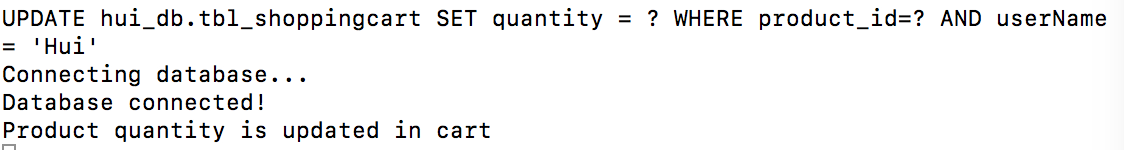


Figure 20 Update Product Quantity in Shopping Cart (Server Side Message)

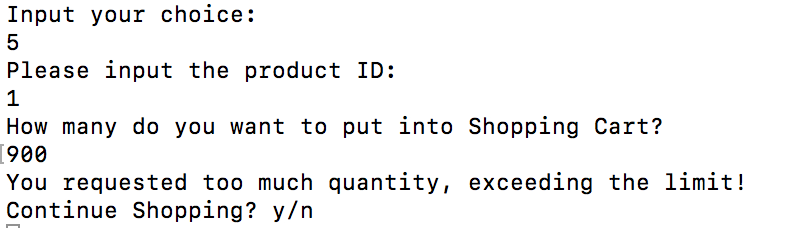


Figure 21 Request Quantity Exceed Inventory

## Choice 6: Add Product into Inventory

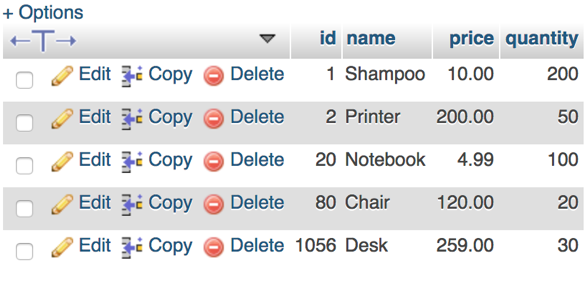
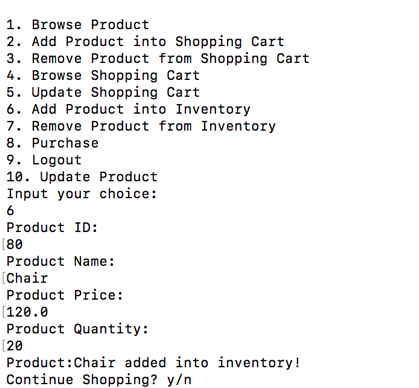


Figure 22 Chair Added into Database Figure 23 Add Chair (ID:80) into Inventory

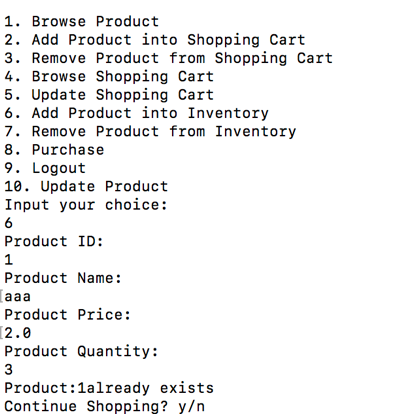


Figure 24 Add Product Failed (ID: 1 Already Exist)

## Choice 7: Remove Product from Inventory

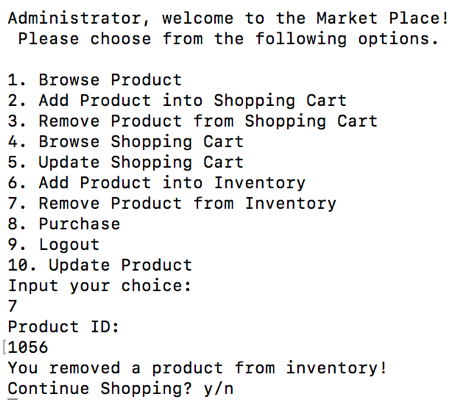


Figure 25 Remove Product (ID: 1056) from Inventory

## Choice 8: Purchase

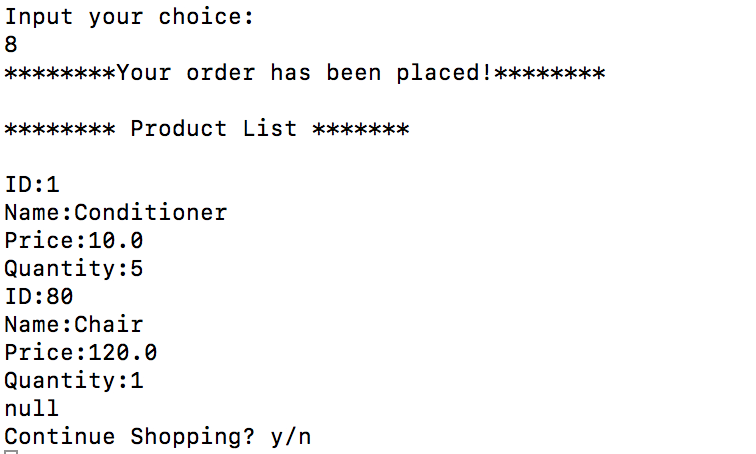


Figure 26 Purchase Shopping Cart

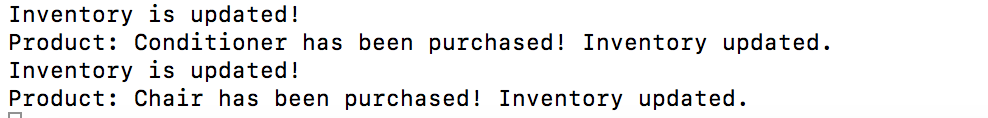


Figure 27 Purchase Shopping Cart (Server Side Message)

## Choice 9: Logout

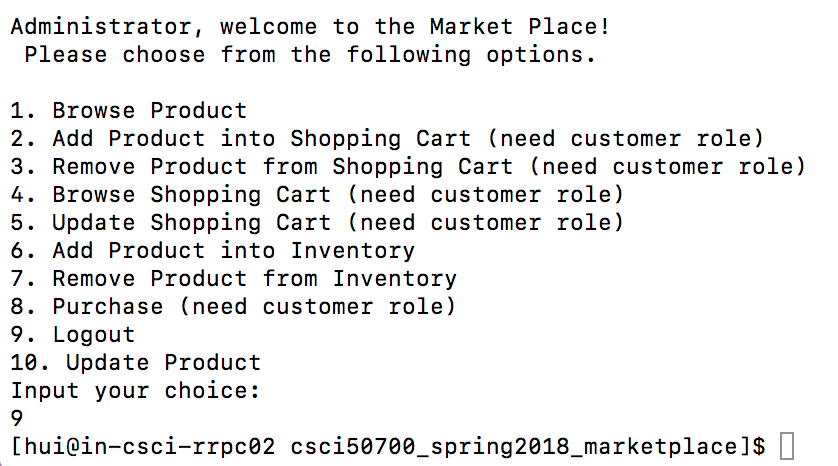


Figure 28 Exit Application

## Choice 10: Update Product

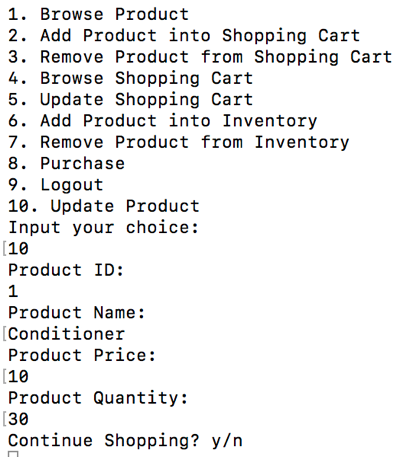


Figure 29 Update Product (ID:1)

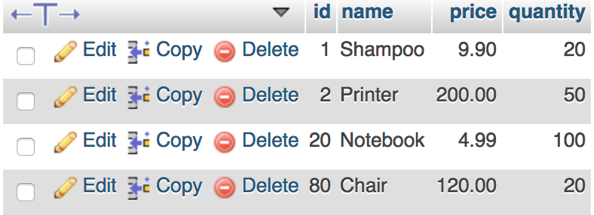
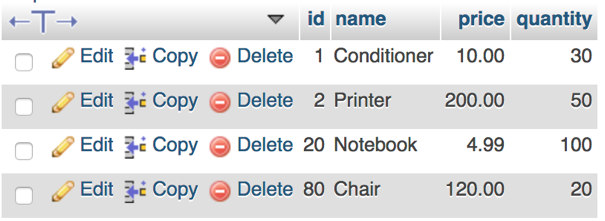
 

Figure 24 Before Update Figure 30 Updated (name, price, quantity)

# Final Conclusions

1. What do I like about the assignment?

My favorite thing about this assignment is that I got a chance to apply several design patterns in one product. This helps me get a better understanding of how these patterns cooperate to achieve some goal.

I used to know little about operation system, For the second half of the project, together with the lecture, I learnt quite a lot about process, thread, concurrency and synchronization. I believe this can be a good start before I take an OS course.

1. What do I dislike about the assignment?

The project is built upon all the previous work. This is a good thing to enhance the knowledge we learnt, but is also a pain to modify our design again and again. In the beginning, I have little experience in design patterns, and it takes so many nights to learn a new pattern and implement it in our application. Once I got feedback that I need to deal with problems in my design, I would be extra stressed. : P

1. What would you change about your design if I could go back? Why?

I will probably change my design of RMI interface. I think my design used RMI in a wrong way especially after listening to others’ presentations. Currently, I have the RMI interface specially for login, and all the other user actions are registered in another interface. There must be a better way to reorganize the structure, maybe I should declare all the methods (user actions) under the remote interface.