ONLINE MARKETPLACE APPLICATION

Shreyansh Mohnot
smohnot@iu.edu

APRIL 27, 2018

CSCI 507 - Object Oriented Design and Programming

DR. RYAN RYBARCZYK

Table of Contents

1. Assignment Overview

Online e-commerce is now flooded with various online shopping websites which present many unique features. Some e-commerce applications allow bidding on products, some have hourly deals, and some are community featured (only for specific communities).

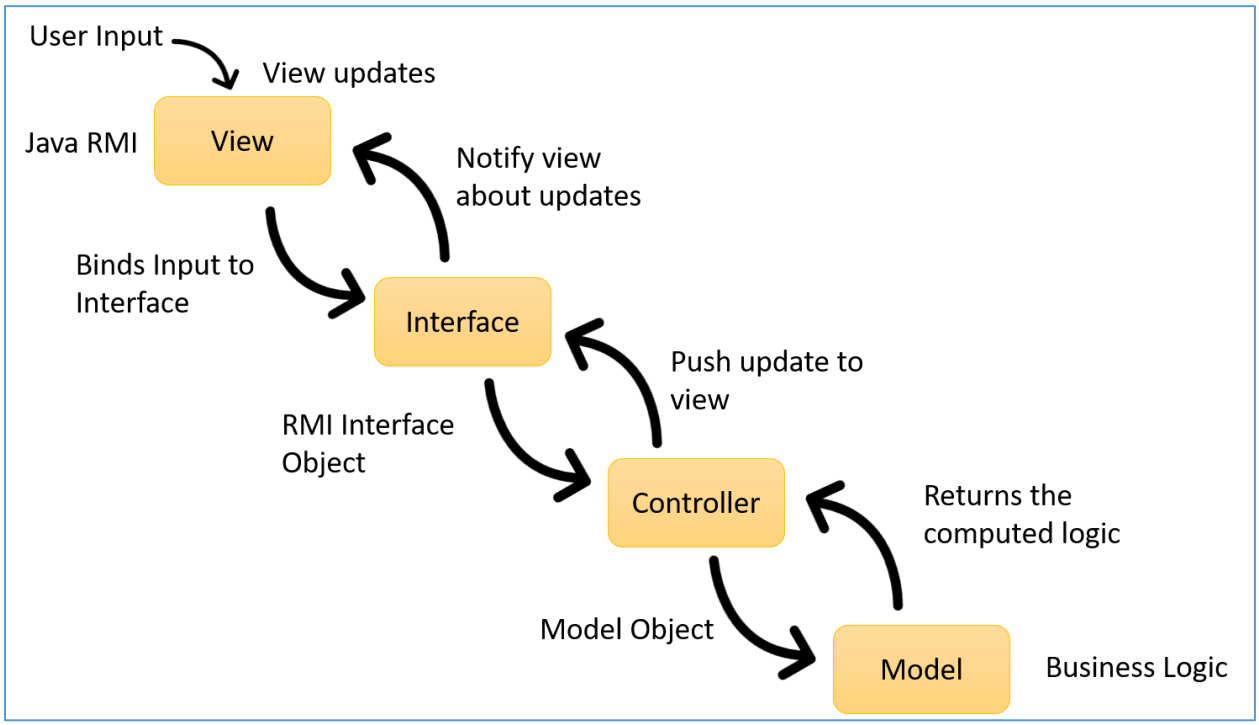
The goal of this project is to implement Online Marketplace application which has simple basic functionalities and on the other hand, also make it compatible to work in a distributed computing environment.

I. Assignment 1-

Java RMI (Remote Method Invocation), a system which allows java class objects to communicate between two different Java virtual machines remotely. Based on RPC (Remote Procedure Calls) structure. It uses interface to communicate between the server and client using two objects stub and skeleton.

The ***java.rmi.*;*** package contains all the necessary support files required for RMI API in a generalized manner.

Model-View-Controller (MVC) Architecture, is a software architectural pattern used in the development of user application. It provides separation between the application logic from the user interface and controller and decouples these components. This allows for efficient code reuse and parallel development. The model contains the business logic of the application, while the view is responsible for to render the contents of the model to the user. The model essentially has all the data and rules required for access to data. The view layer is a user interface representing model contents. View changes whenever the data in model changes. The controller processes all the user requests forwarded from the view. It converts all user interactions into actions with data model objects. The main functionality of controller includes – receiving user inputs, input validations and computing the business logic from the data model.



Review –

Established RMI and created an initial controller in the model. This made my client lightweight and had limited view functionality. MVC provided the flexibility to change the design patterns such that it does not affect the client side. The controller is responsible for data model objects and to communicate with the model and receive data.

II. ASSIGNMENT 2-

Front Controller, is a pattern which channels all user requests through a single controller. This controller is the single point of entry, for the users of the application and for views in MVC architecture. This controller makes a centralized point for each control and management of requests. In my implementation of the front controller, it uses application controller, dispatcher, and a command pattern to realize this functionality in it. The dispatcher manages views and navigation between the views.

Command Pattern, encapsulates requests into object commands and passes them to command invoker. At the invoker, it goes to the method designated which can handle that requests and passes that command to the corresponding object. This pattern decouples action to be taken with from where the request came and then performs the action. Command pattern used to separate all the functionalities and make them a single point of execution. Each concrete command is sent to application controller which calls execute().

Abstract Factory Pattern creates a factory of factory patterns. These factories are abstract and do not care about the concrete types of objects. It allows developers to create a more generic design which is highly decoupled to objects of the same class and follow a required pattern. It has a general architecture where we have Abstract Factory, Concrete Factory. In the end, we are returned with a factory of related objects without knowing their class types by the factory creator. Here factory creator returns us the objects of views requests by the controller after user login into the system and separated as admin or customer or browse functionality as requested by the respective user role.

Review –

Provided Separation of concerns with the correct architecture of Model-View-Controller. Numerous design patterns implemented in this part. Also added login functionality to the application.

III. ASSIGNMENT 3-

Role Based Access Controls with Authorization Pattern, controls the usage of various functionalities within the system. Provide the benefit of configuring how and when each user would be able to access. This helped in providing a secure client distributed environment with multiple clients and servers. I implemented in interface design since it separated the view and model at that point in the application. Two roles are specified – Admin and Customer using Java Annotation.

Proxy Pattern, creates a dynamic proxy class which implements and creates a proxy instance. The proxy has three instances such as Interface, List of Interface which proxy to apply and the invocation handler to redirect the proxy method. Each proxy instance has an associated invocation handler object.

A method invocation on a proxy instance gets dispatched to the invoke method of the instance's invocation handler. It passes the proxy instance object identifying the method that was invoked and an array of type Object containing the arguments.

Reflection Pattern provides us the capabilities to reverse engineer source code from class files. It gives the power to extract information given in a class and implement the useful information using this pattern. It modifies runtime applications and examines the behavior of code at runtime without knowing actual implementation. Proxy pattern uses Reflection pattern and implements the functions whose methods are called and then returns the values back to calling instance.

Review –

With the part of the assignment, implemented controller at model as well at view side with their function of providing a login to customers and administrators. Highle decoupled application logic created and provided an update to the design patterns. Each request from the view is now authorized by the role-based access before going to the model. Implemented a session object to keep track of users on both the model and view.

IV. ASSIGNMENT 4-

Concurrency, is a technique to invoke multiple tasks at a single time in a distributed application. It plays a major role in terms of data manipulation and retrieval. Concurrency is achieved using serialization of Objects, synchronization of methods and thus the remote objects are passed using serialization in RMI.

We learned about RMI being multi-threaded, allowing your servers to exploit Java threads for better concurrent processing of client requests. But makes no guarantee that having thread safe implementation of methods running concurrently. To achieve concurrency, we implement synchronization at the method level such that same object method when executed would result in thread safe implementation of the methods called while RMI threads are invoked. Java RMI facilities concurrent application and provides an interface to run multiple clients and communicate with those different hosts over the network.

Review –

Browse Item, utilizes database and works as every browse functionality to provide a network call to the product table in the database. It returns a list of products from the database which is then send to the application controller. ***Add Item***, facilities addition of items to the database. This could be only done by the administrator and thus the method to add an item is synchronized and get updated in the product table of database. ***Purchase Item***, provides purchasing option to customer to purchase current shopping cart items. It finalizes the quantity and decrements them in the product table. If it is "out of stock" then it displays a message. To purchase an item from database requires extensive checks and validations in the database table. It also provides synchronous database access and updating.

V. ASSIGNMENT 5-

Synchronization, is a terminology where updating a data objects concurrently updates every other same object in the execution. This synchronizes the implementation. Synchronization is provided using "synchronized" keyword in Java. It utilizes monitor lock associated with the objects class and gives exclusive access to the objects state and invoking a **happens-before** relationship. This preserves the exclusivity of the thread and has thread safe implementation. Lock is release when the method returns or throws an exception. I used synchronization keywords where database access is given to modify certain tables and updating the values in it. Access to database is done synchronously and parallel modification

takes place in the database. Database also has its own read and write lock levels which further adds to synchronization in the application.

Monitors, are data structures responsible for providing synchronization in java. Monitor objects allows modification to shared resources in multithreading environment when multiple threads try to modify single resource, it blocks those. It releases them the lock after the scope of the invoked instance method is returned or completed. These locks are reentrant i.e. this means a lock already owned by a thread could be acquire once again by the same thread without running into deadlock. In my implementation the underlying architecture has monitor locks and each synchronized method block uses them. After acquiring the lock, it enters critical section of the code and blocks other thread trying to acquire the same lock on that method until it returns from the critical section is completed. Hence atomicity is achieved with this.

Thread Safety, is not guaranteed by the Java itself and their behaviors are unpredictable leading to various issues such as starvation, deadlock and race conditions. To ensure thread safety it is done by thread safe variables and keyword implementing thread safe. Threads access is consistent with execution and obeys a synchronization order consistency. My usage of volatile array list of products has updated the values at any given time and helped in synchronization of product list. Thus, the product list becomes thread safe meaning it could be used by multiple threads at a given time.

Guarded Suspension, ensure threads do not violate the data safety and ensure consistent and concurrent execution of threads inside a system. Threads perform context switching and follow an order of execution before proceeding to the next step. This ensure critical section is not achieved until the shared resources are released by the thread or are ready to be acquired by a thread. Whenever the inside the critical section, the object invokes wait() on the thread then it would suspend the execution of the thread and release the lock. It then has to wait in the future to acquire the same lock and invoke notfiyAll(). This creates an alarm that something occurred in the thread.

In my application, guarded suspension was not used explicitly. There is an internal mechanism of synchronized word which underlines the principle of guarded suspension such that is another thread tries to access a synchronized block, it is blocked until the thread which has acquired the lock release it. Thus, enabling a guard sought of mechanism on the block of method.

Future, represents on how a time-consuming computation will yield a result that will eventually appear after its processing is complete. Future implemented using Future interface. This enables us to execute other process while waiting on the expensive tasks yet to be completed and yield in future. While waiting, it provides a client virtual data object keeping a track of running state computation and once value is computed, provides its value to client in the future. In my application scope of future is when synchronize tells the JVM to synchronize the block and method automatically providing a java virtual object and notify the client thread when the result is available.

Scope Locking, mechanisms is needed when we enter a critical section and must get a proper assignment of lock for that mechanism but due to some exception behaviors as the application terminates without releasing the lock. This results in starvation of other threads in the pool waiting for the lock. Hence scope locking avoids this by automating the release of lock when control leaves the scope via any exit path. Thus, reducing the starvation problem. It is achieved using the same synchronized keyword which handles the monitor locks by corresponding instructions monitorexit and monitorenter. This allows handling of locks automatically by the JVM.

Review –

The application runs synchronization keyword in blocks which have concurrent access to the data. Hence, we can see that wherever the synchronization keyword is used thread safety, future, guarded suspension, monitor objects and synchronization are implemented successfully.

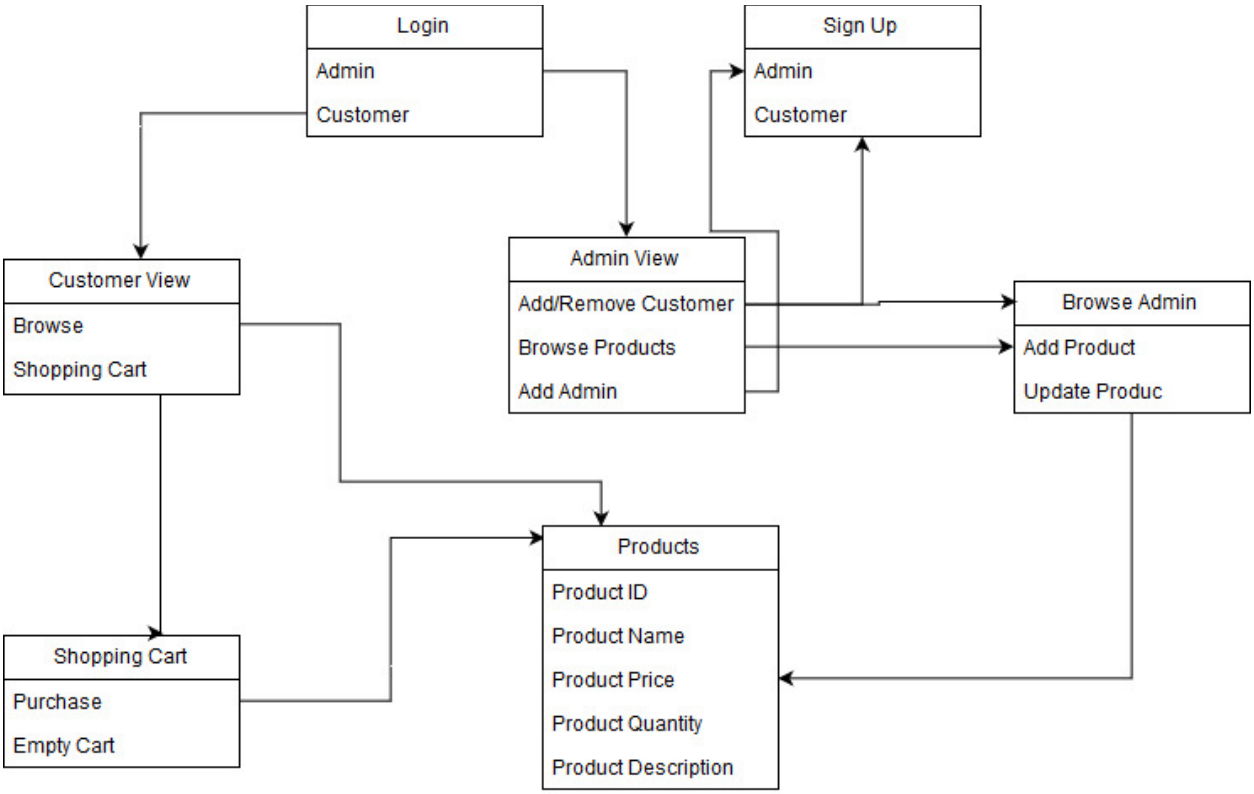
2. CURRENT ASSIGNMENT DISCUSSION

In current assignment I have separated the database layer from the server model. It was implemented such that any changes to database could easily be incorporated in the whole application and thus providing low coupling and high cohesion. I implemented a static final object connection of the database which ensures that database connection is made once and after the connection is established it could not be made once after. Each server would have its own database connection object which would be final too. It showcases a singleton pattern where instance of a class can be created once only and for reusability it would use that same instance again and again.

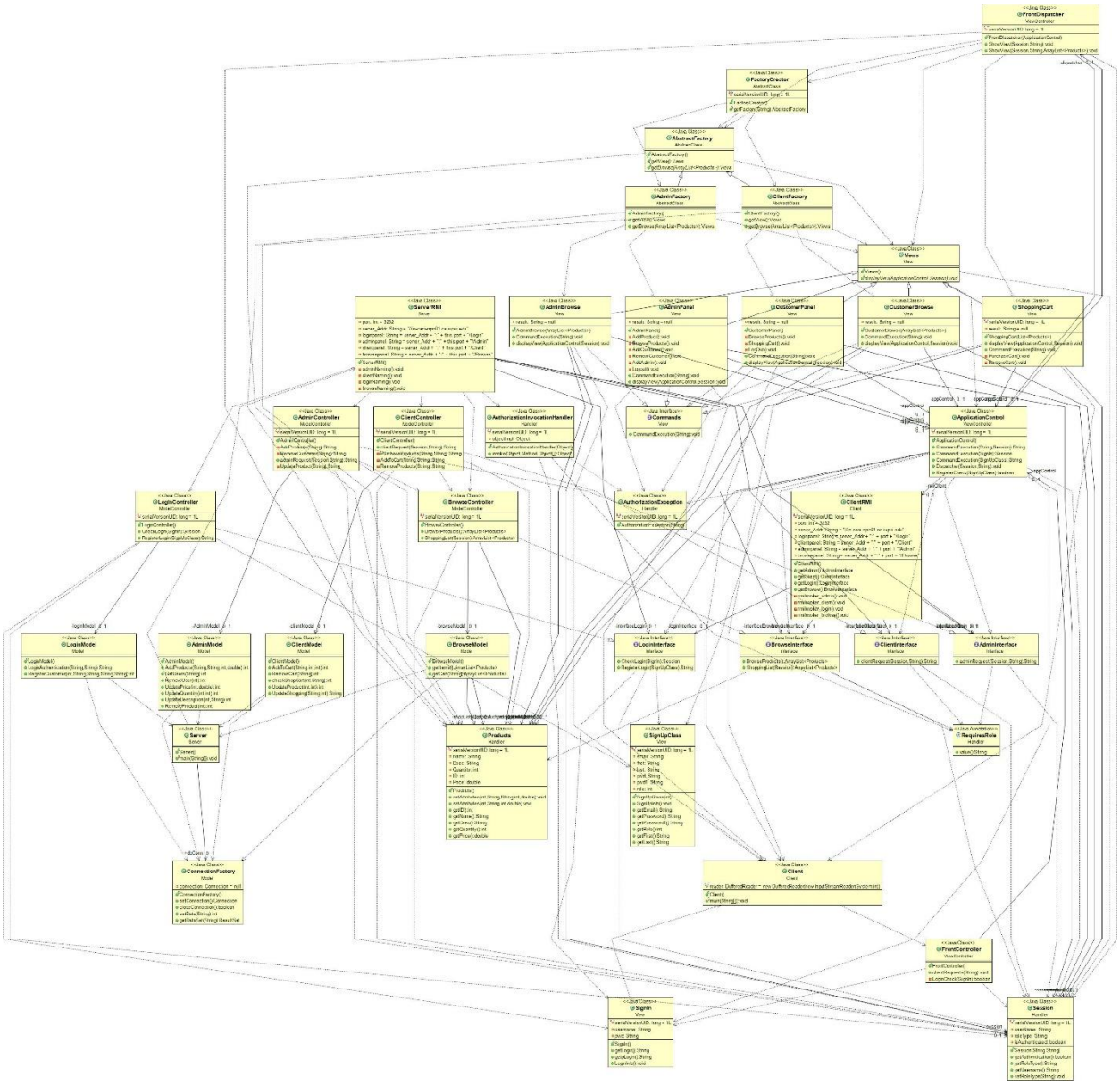
I have also added some synchronization variables in the MySQL database where if a product is removed from the product list by the Administrator. It also removed from all the other tables and shopping cart tables. This ensures that product is not be able to be purchased by any customer. There has been a feedback where my browse is not synchronized. I have tried to fix that too in this update.

3. UPDATED FIGURES AND DOCUMENTATION

Domain Model –

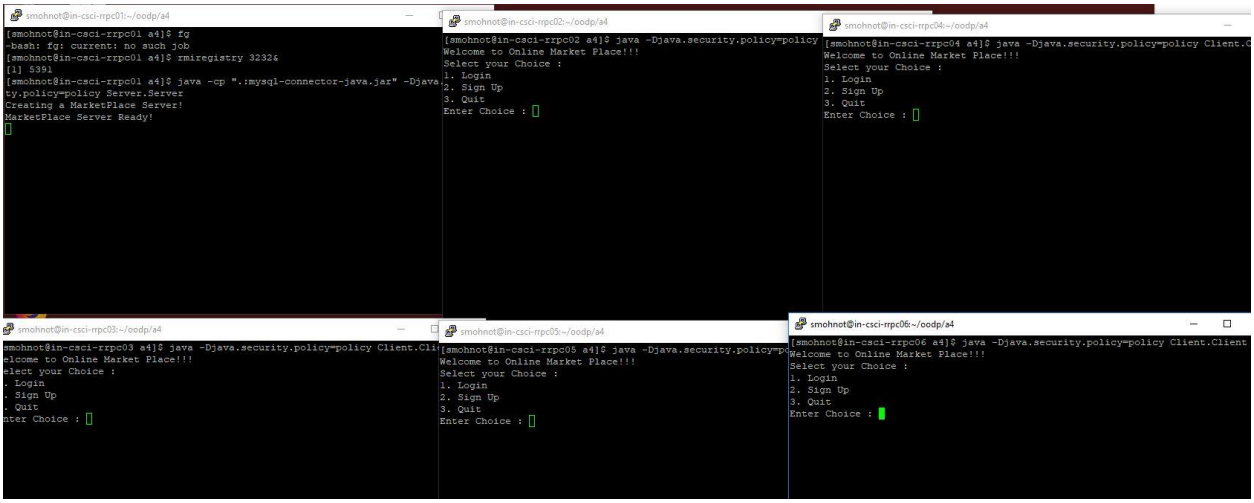


UML Diagram –



Sample Runs –

Initial Home-



Admin View -

```
javac -cp ".:mysql-connector-java.jar" -g Handler/AuthorizationInvocationHandler.java
javac -cp ".:mysql-connector-java.jar" -g Model/AdminModel.java
[smohnot@in-csci-rrpc02 a5]$ 1
bash: 1: command not found...
[smohnot@in-csci-rrpc02 a5]$ java -Djava.security.policy=policy Client.Client
Welcome to Online Market Place!!!
Select your Choice :
1. Login
2. Sign Up
3. Quit
Enter Choice : 1
Enter Login Information :
Username/Email : admin
Password: admin
Welcome Administrator :- admin !!
1. Browse Products
2. Add Product
3. Add Admin
4. Add Customer
5. Remove Customer
6. Sign Out
1
```

Customer View –

```
2. Shopping Cart
3. Sign Out
^C[smohnot@in-csci-rrpc05 a5]$ make clean
rm -f */*.class
[smohnot@in-csci-rrpc05 a5]$ java -Djava.security.policy=policy Client.Client
Welcome to Online Market Place!!!
Select your Choice :
1. Login
2. Sign Up
3. Quit
Enter Choice : 1
Enter Login Information :
Username/Email : shreyansh
Password: shreyansh
Welcome to Online Martket Place :- shreyansh !!!
1. Browse
2. Shopping Cart
3. Sign Out
1
Product ID     Product Name      Product Description      Product Price
 1              Iphone            Mobile                   10.00
 2              iPad              tablet                   400.00
 7              tv                television               300.00
 8              LCD               monitor                  250.00
```

These are the respective views for the administrator and customer. Currently, it works with same logins or multiple logins with same username.

Customer Browse and Add to Cart –

```
Username/Email : shreyansh
Password: shreyansh
Welcome to Online Martket Place :- shreyansh !!!
1. Browse
2. Shopping Cart
3. Sign Out
1
Product ID     Product Name      Product Description      Product Price
 1              Iphone            Mobile                   10.00
 2              iPad              tablet                   400.00
 7              tv                television               300.00
 8              LCD               monitor                  250.00
 9              MacBook           Laptop                   750.00
 10             Stuff             things                   4.99
 11             Table             wooden                   3.99
 12             Bottle            water                    1.99
Enter Product ID to Add to Cart OR Press 0 to return..
1
Cannot be added to cart. Out of Stock
Progress : Done
1. Browse
2. Shopping Cart
3. Sign Out
1
```

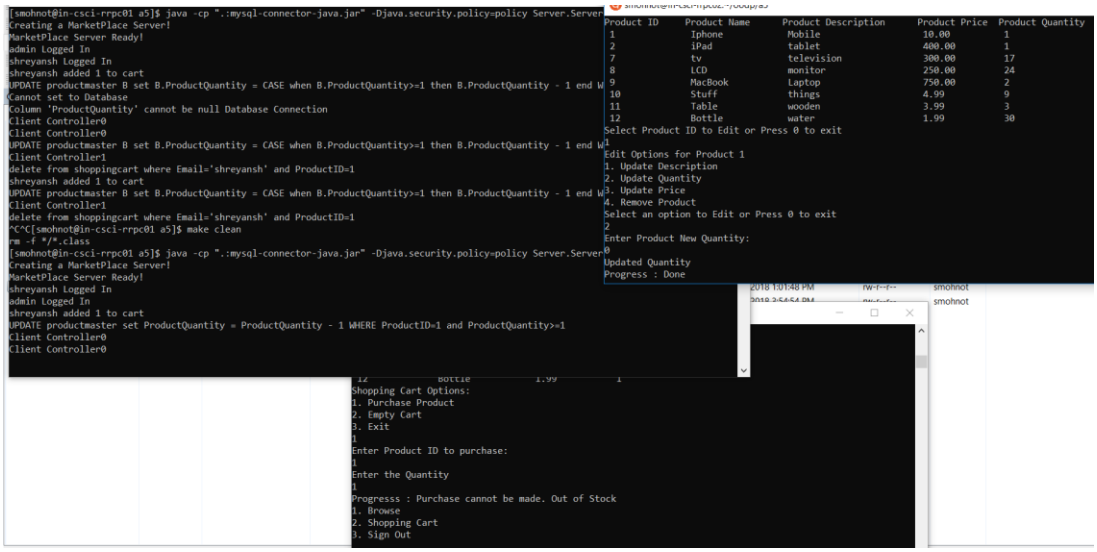
Customer Shopping Cart and Purchase –

```
 12             Bottle            water                    1.99
Enter Product ID to Add to Cart OR Press 0 to return..
1
Product Added
Progress : Done
1. Browse
2. Shopping Cart
3. Sign Out
2
Product ID     Product Name      Product Price  Product Quantity
 1              Iphone            10.00          1
 2              iPad              400.00         1
 12             Bottle            1.99           1
Shopping Cart Options:
1. Purchase Product
2. Empty Cart
3. Exit
1
Enter Product ID to purchase:
1
Enter the Quantity
1
Progresss : Purchase Complete
1. Browse
```

Admin Browse and Update –

```
Welcome Administrator :- admin !!
1. Browse Products
2. Add Product
3. Add Admin
4. Add Customer
5. Remove Customer
6. Sign Out
1
Product ID     Product Name      Product Description      Product Price  Product Quantity
 1              Iphone            Mobile                   10.00          0
 2              iPad              tablet                   400.00         1
 7              tv                television               300.00         17
 8              LCD               monitor                  250.00         24
 9              MacBook           Laptop                   750.00         2
 10             Stuff             things                   4.99           9
 11             Table             wooden                   3.99           3
 12             Bottle            water                    1.99           30
Select Product ID to Edit or Press 0 to exit
1
Edit Options for Product 1
1. Update Description
2. Update Quantity
3. Update Price
4. Remove Product
Select an option to Edit or Press 0 to exit
2
Enter Product New Quantity:
1
Updated Quantity
```

Concurrent –



4. FINAL CONCLUSIONS

This assignment was a comprehensive analysis of various design patterns learned throughout the course structure and implement them in real time. This made it clearer to see how every pattern could/could not be accommodated in the design of a system. This assignment class term project made me think twice before programming and designing. What I like to do during the assignment phase was to design where and how functionalities would be implemented and how would they work. Surely, I was not considering the distributed aspect until concurrency popped up.

My main dislike about the assignment was the use of RMI. Can't we use socket is would be much more simpler and thread programming would be a lot easier to implement and would not face any concurrent issues in socket. Also, I would like to change how I implemented the view controller part. Because I made the client side do so much heavy lifting which would affect the performance of the application.

5. REFERENCES

I. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html

II. https://docs.oracle.com/javase/specs/jls/se8/html/jls-17.html