# Online Marketplace

## Akhil Nayabu

## 2000075395

## Assignment #5

## Course Number: CSCI 50700 – 23706

## Course Name: Object-Oriented Design and Programming

### Assignment Overview

In Assignment #5, you will be examining the impact that the concept that **Synchronization** has on our application. As we saw in Assignment #4, there are many significant challenges with respect to the construction of multi-threaded or concurrent systems. As part of this assignment you were tasked with examining some solutions to the challenges that you identified – often these challenges/problems were ones that could directly be solved through the use of Synchronization design patterns.

The focus of this assignment is to apply the information we have learned regarding the use of synchronization in the Java programming language to ensure that access to our shared resources are indeed thread-safe. Specifically, we will be examining how Java implements the following patterns: **Monitor Object**, **Future**, **Guarded Suspension**, **Scoped Locking**, and **Thread-Safe Interface**. Your job is to make use of the various Java defined constructs to accomplish this thread-safety within our application. As part of this assignment you will need to make sure that your application now makes use of the MySQL database that has been setup for us to use on: (in-csci-rrpc01.cs.iupui.edu). In addition, as this will be the last assignment for the semester, you are expected to fully complete all of the required functions as outlined in the project specifications. You should use good design practices, principles, and patterns, when applicable, to accomplish this. This final submission should be a polished product and something that should be proud of.

For this assignment, we will make use of the following machines (listed on this page) to demonstrate the role of synchronization and the functionality of your application in a concurrent environment. Your server should demonstrate the ability to handle multiple concurrent requests from different clients. These clients should be able to run on any of the given machines and should locate the server running on a given machine. For this assignment, we will, again, make the assumption that there is only ever one server but many clients. All of the other requirements are still valid. Any updates to your design should be reflected in your domain model, software architectural model, and a discussion of these new design decisions as part of your report. This final report should contain a complete overview and all proper documentation related to the creation of this application and the ongoing work we have done this semester.

Your code should be properly documented and should make use of common language standards and implementation practices. This code should be your own work – each student should submit their own project; failure to do so or any student caught cheating will be subject to the IUPUI guidelines on academic dishonesty as provided in the course syllabus.

### **Assignment #1 Feedback**

* **Canvas**: No comments provided
* **GitHub**: No feedback to address

### **Assignment #2 Feedback**

* **Canvas**: No comments are provided.
* **GitHub**: Comments are included each file was per the feedback provided on comments branch on GitHub

### **Assignment #3 Feedback**

* **Canvas**: No comments provided.
* **GitHub**: Comments included in each file.

### **Assignment #4 Feedback**

* **Canvas**: No comments provided.
* **GitHub**: Nothing provided in commits branch.

#### **Project Overview:**

Online Marketplace is a platform where an organization can sell their goods to customer who are geographically dispersed all around the world. This application allows the organization’s administrator to add items into the inventory, update a specific item or remove an item from the inventory. The customers can access this application from anywhere in world and try to purchase items from the platform. The customer has capability to go back the review his orders from past.

Below are the details of each functionality implemented in this project:

#### **Login:**

Login functionality allows the user (could be a admin or customer) to authenticate using their user name and password. Once the user is authenticated the users are redirected to their respective unique home pages. Customer and administrator have different home pages. This is a common functionality for both admin and customer.

#### **Browse Items**

This view is common for both admin and customer. It displays the product list i.e. displays all the products which are available in inventory on the server. A product list is retrieved from server which contains objects of each product. These products are iterated through and displayed here in browse view.

You will able to navigate to product view, from the browse when you select a product from the browse. This product view lists out all the details of the product from its object which was passed on from the browse view. From this view, you can add that product to cart if the user type is customer or can go back to home. The screenshots for these are shown in sample run.

#### **Update Product**

This is an admin specific function where admin will be able to update the product. In product update view, the admin is asked for product id and then he need to populate new product details for that product id. These new details are updated in database.

#### **Remove Product**

This is an admin specific function where admin will be able to remove the product. In this specific view, admin is asked for product id and entered product id is removed from the database.

#### **Add Item**

This is an admin specific functionality where admin can add an item to the inventory. It calls add item view where details for new product which is to be added are taken and is added to product list on the server. Screenshot is provided in sample runs.

#### **Add Item to cart**

This is only available to customer, if admin tries to add an item to cart, an exception is thrown showing not authorized to perform this task. You can add an item to the cart from product details view. And also you can see the cart details from the home view. All the cart details are preserved until purchase has been done. Cart is saved until it is been cleared out i.e. user can add the items and come back later to purchase them. When the item is added to cart it redirects customer to cart view.

Also, if the user enters quantity more than what is available then the product will not be added to the cart.

#### **Cart View:**

In this view it displays the user all the items which were added to cart by the user. It contains the product details and total cart price i.e. the total price of all the items in the cart. This is view is called when user adds an item to cart or from the home page. If the user has an associated cart with items attached to it already then it returns that cart or it displays that cart is empty. If the user doesn’t have a cart already mapped to him, then a cart is created for him. Each time the cart has been processed for purchased, it is emptied.

If the user is new, i.e. never added any item to cart, then he/she has no associated cart mapped to them, where it displays cart is empty. But when user adds item to cart for first time, a shopping cart is created and mapped to him/her.

Screenshots are provided in sample runs.

#### **Purchase Item**

You can purchase items from the cart view. If cart is not empty you can see purchase option for the list of items which are on cart. Upon purchase request, the system ask user for his shipping address and later the list of items in cart is passed to server for validation.

On server, it validates the cart items as follows:

Step 1: gets all the items from the cart  
Step 2: gets latest item details from the server for items which are on cart  
Step 3: checks if items are still available   
Step 4: if requested quantity is available for the item, item is placed and quantity for that product on server is reduced, and status for that item is set to processed  
Step 5: if requested quantity is not available, the item is not placed and status is set has not processed and current available quantity of that item is returned so that user is notified why the item was not placed.

After validation and order has been placed, it redirects user to order confirmation view. Where details of the order is displayed. All the items which were placed and all those which were not placed. It displays order id, order date, order total, shipping address (item total + total tax).

The items which were placed and those which were not placed are displayed based on the status message set for each item during validation of cart on the server.

Once the order has been placed the shopping cart is cleared.

Screenshots are provided in sample runs.

#### **Order History View**

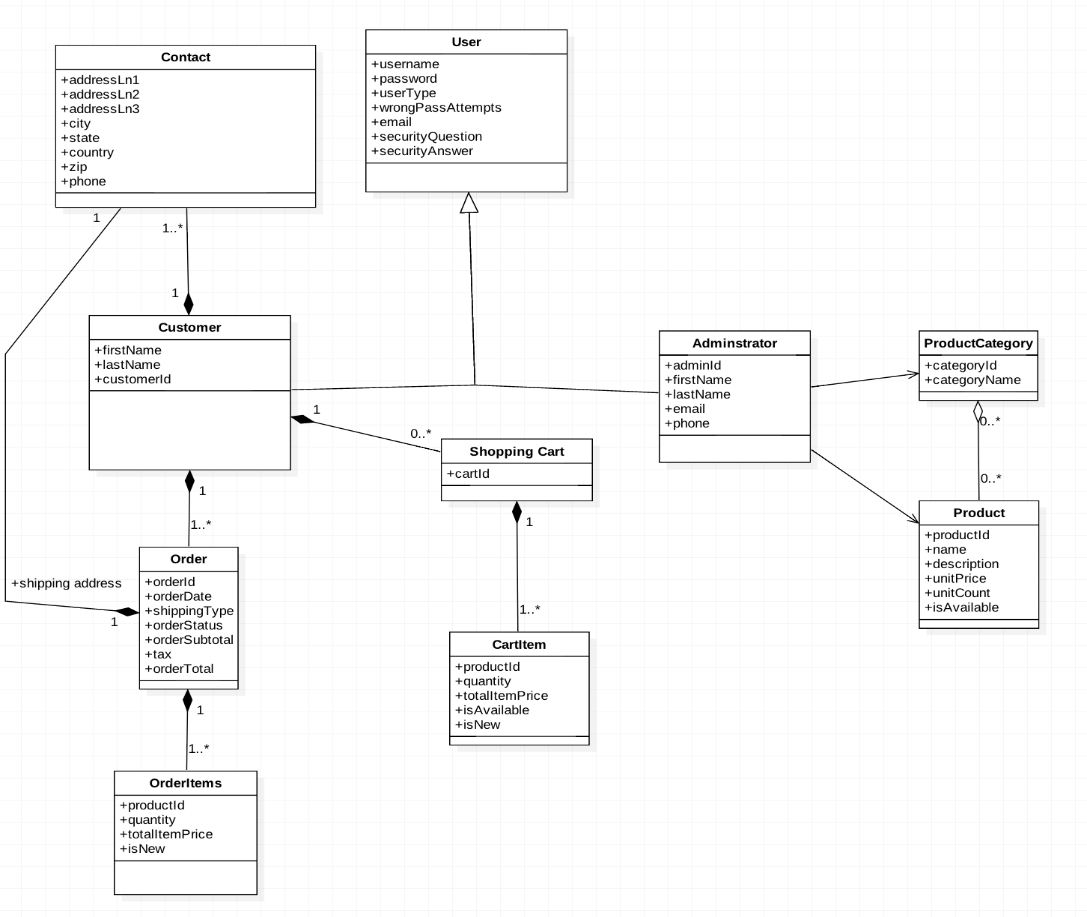
It is a customer specific view which displays all the orders which were successfully placed the user so far (till date).

#### **UML Diagrams:**

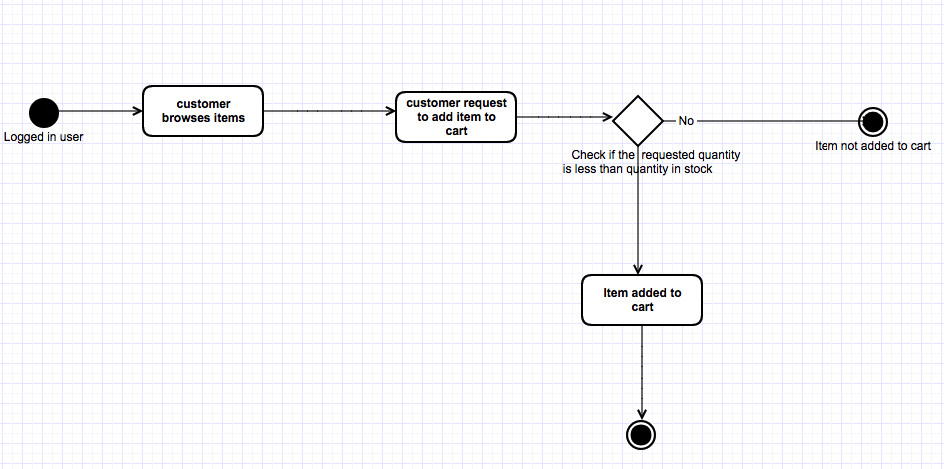
#### **Domain Model**

A domain model illustrates meaningful conceptual classes in problem domain. The following diagram illustrates the domain model for this assignment. It lists all the conceptual classes that will be used to build the framework for this assignment.

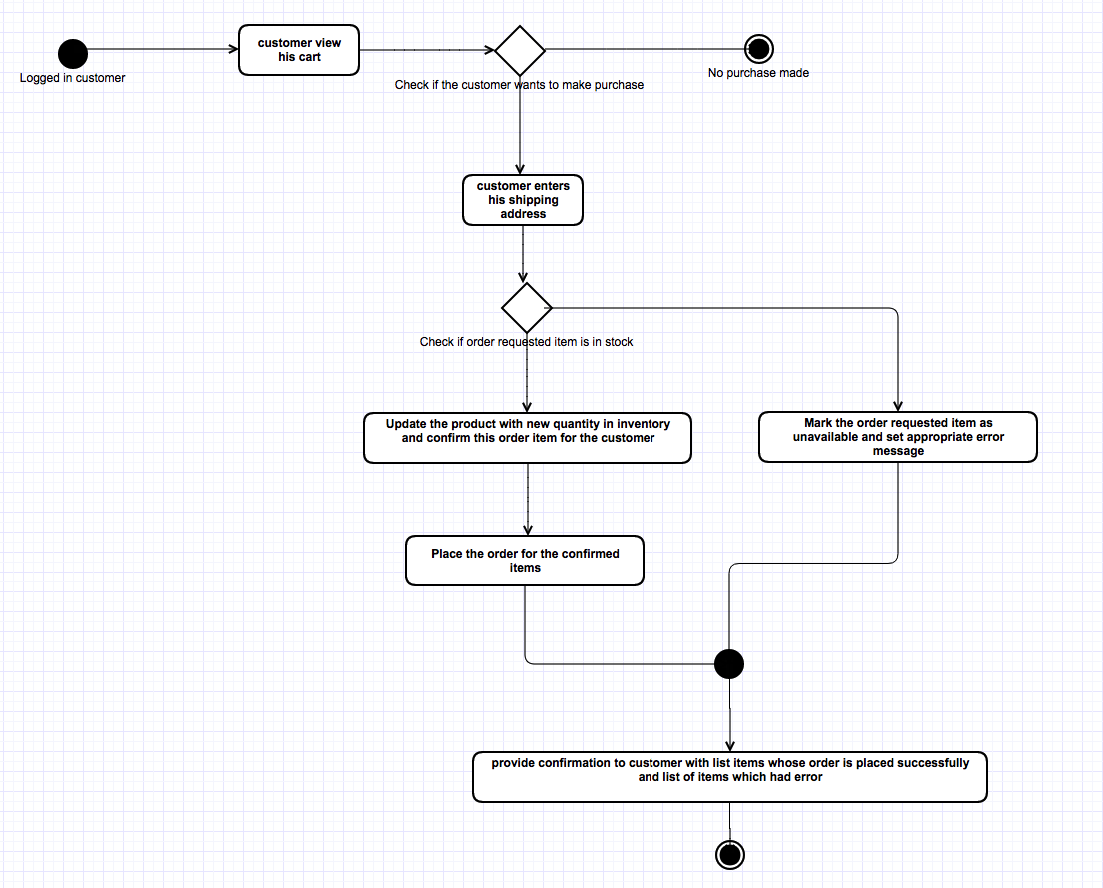
* Updated Domain Model: no changes

  
Fig 1: Domain model

#### **Activity Diagram for Add Item to cart:**



#### **Activity Diagram for purchase item:**



### 

### **Sequence flow when user has access to the method:**

Fig 2: When user has access to the method

#### **Sequence flow when user has no access to the method:**

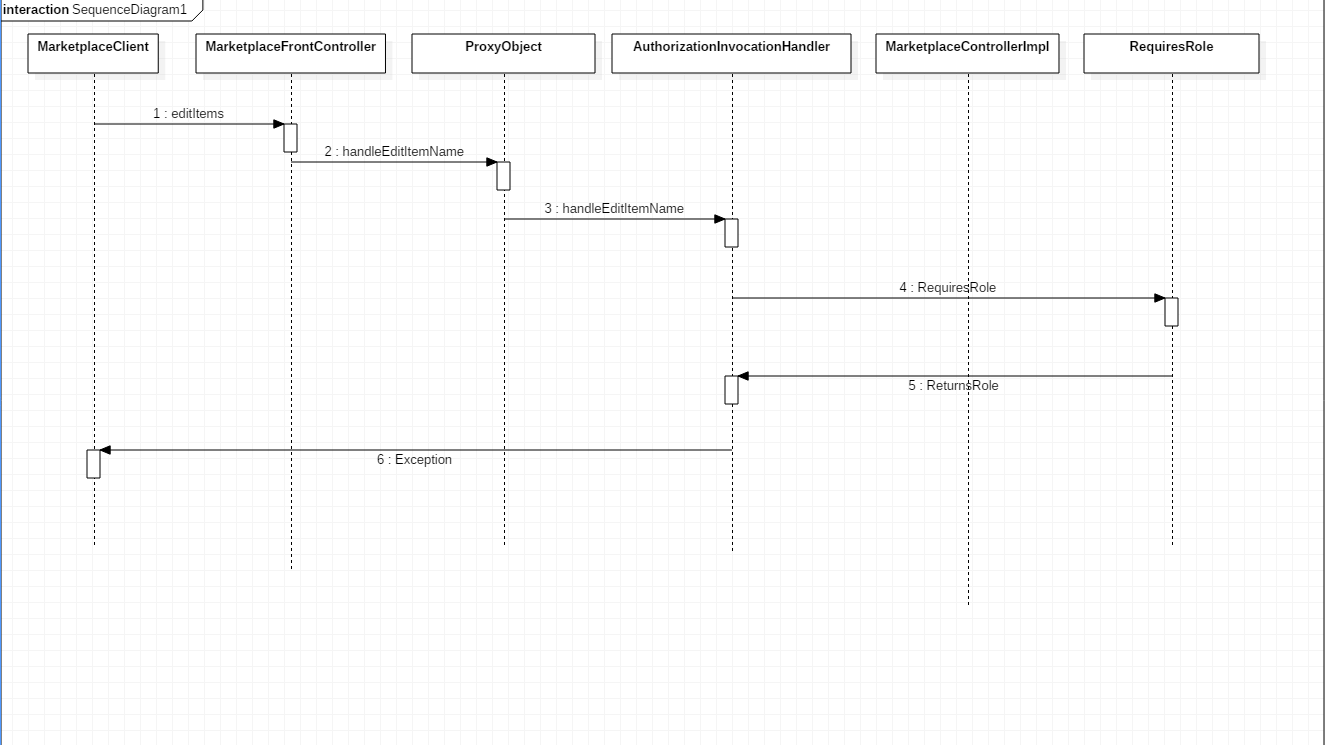


Fig 3: When user has no access the method

#### **Class diagram for Front Controller Pattern**

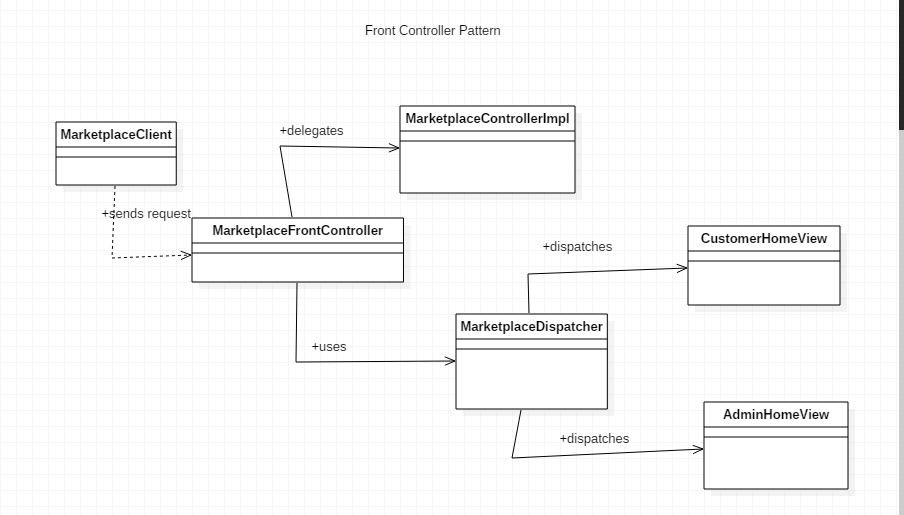


Fig 4: Front Controller

#### **Class diagram for Abstract Factory Pattern**

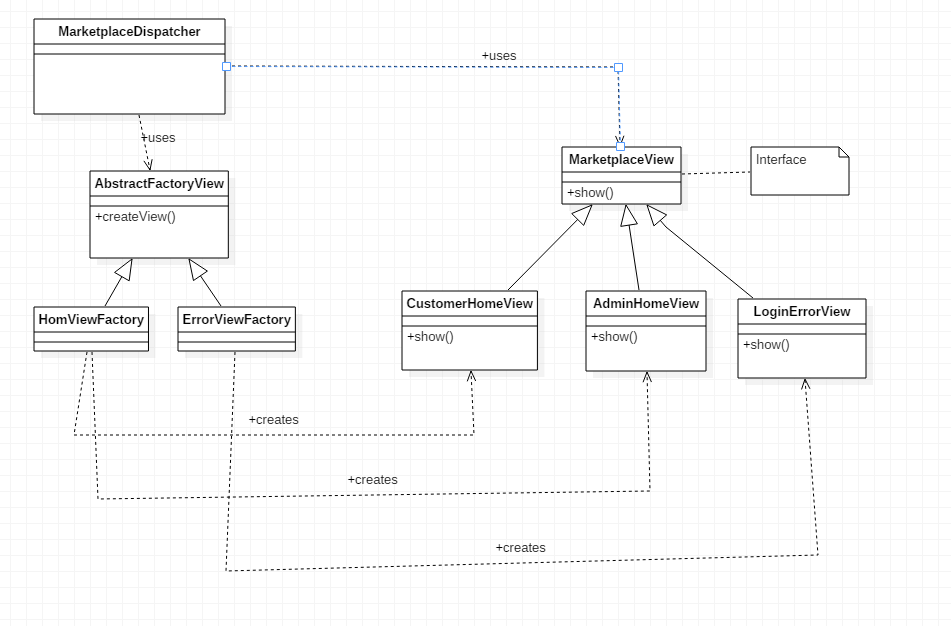


Fig 5: Abstract Factory

#### **Class diagram for Command Pattern**

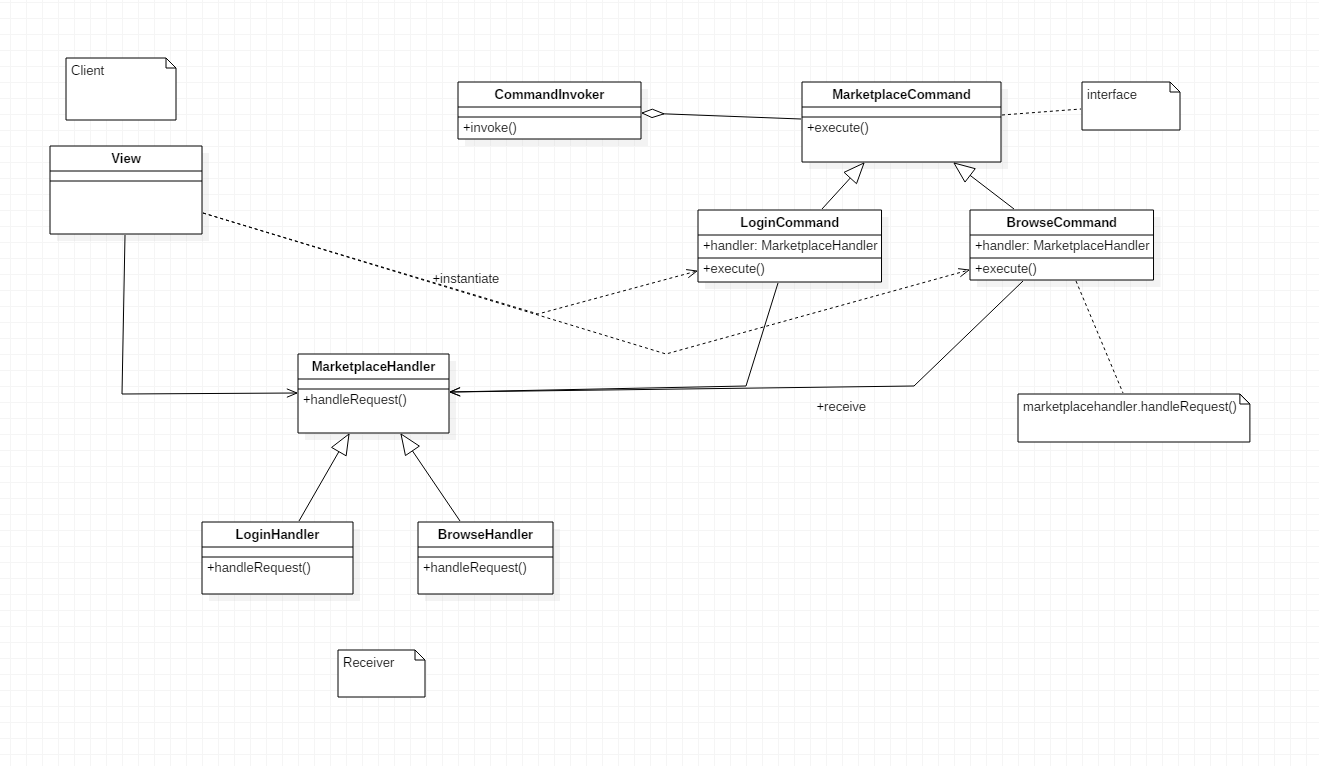


Fig 6: Command Pattern

#### **Synchronization Discussion (Assignment-5):**

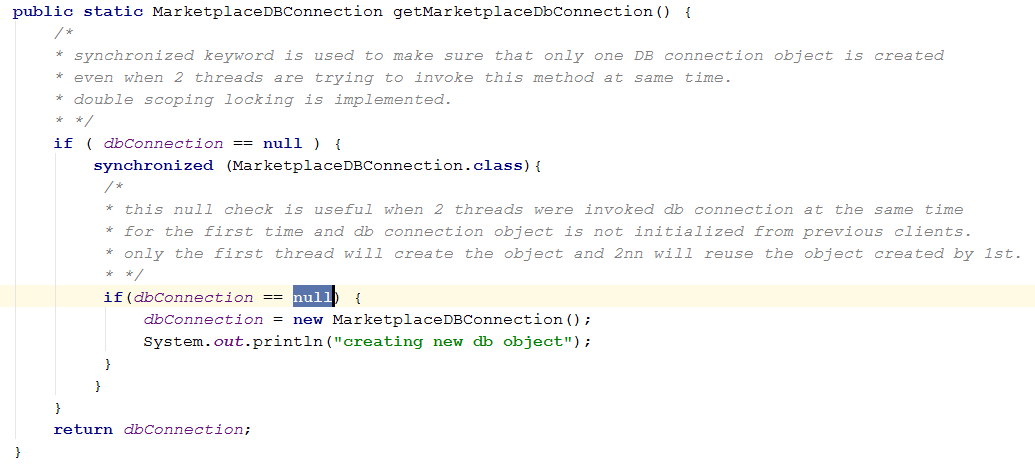
### **Database Connection through Singleton Pattern:**

This pattern is responsible for to create an object, and this is the only object which is created. This class will provide a way to access its only object which can be directly accessed without any need to instantiate the object of this class. We will try restrict this connection object to just one because they highly memory consuming and to centralize the process.

We will create a static class object and a static method. This object will be initialized in the static method. We used this pattern in our database connection class. In this class, the static object is initialized first time, i.e. when it is null. From second time onwards it returns the same object since was already initialized.

This static method needs to be synchronized because there is chance that 2 threads might initialize this object since there is no restriction on how they are entering it, So 2 threads get access to this method at same time, then the first thread will create an instance which is later replaced by second thread, so to overcome this we will make this method synchronized.

But having this method synchronize has a drawback, i.e. it slows down the call, as each thread will wait for another thread to step out, so that it can get the object, which is not necessary as we only need to synchronize first creation of instance. So we will make use of synchronized block, so when 2 threads come in at same time for first time, then only one enters which will create an instance and since instance would not be null, 2nd thread we will reuse the created instance. This is also known as *double scope locking* ***[1]****.*



### **How synchronization is achieved in my application:**

#### **Identification of methods which needs to be synchronized: - Purchase order:**

#### No 2 customer should be able to buy same product at same time, that specific product should be processed one after the other. (If we allow them to purchase same product at same time, then both might end up placing the order even though there only 1 quantity available)

* But they should be able to buy at same time if, they are not buying the same product. Hence it can’t be synchronized on method, since we should allow multiple customers to place order at same time.
* Also, it should not process the order at same time when the same product is being updated/ removed by the administrator.

**- Update order:**

* No 2 admins should be able to update same product at same time.
* But they should be able to update products at same time, when the product they are update are not the same.
* Should not allow, customer to buy the product when it is being updated.

**- Remove order:**

* No 2 admins should be able to remove same product at same time.
* But they should be able to remove products at same time, when the product they are remove are not the same.
* Should not allow, customer to buy the product when it is being removed.

**Why can’t we synchronize on the methods and what is the need for synchronized block and how is thread-safe synchronization achieved?**

We cannot synchronize on methods as synchronizing methods will affect the performance, since no 2 customer threads will be able to access the method, even though they are trying to buy different products. We cannot stop a customer from buying a television, just because some other customer is purchasing a phone at the same time. We should only try to avoid situations where both of them are trying to purchase the same product. To allow customers to purchase concurrently when the products they are purchasing are not same, we shouldn’t synchronize on methods.

Since we can’t synchronize methods, we will use synchronized block. We will synchronize that block which has critical portion. We will synchronize this block on a monitor object which we would like to lock the block on. Hence we will trying to synchronize it on product id. By doing this we can implementing **monitor object, scoped-locking** which are required for this method. **Future pattern** can be used by displaying a notice for assuring a user that his/her process is being put forward and will be processed soon, this pattern can we used where these synchronized methods are being called, since one method cannot execute when a lock is placed by the other. Hence future pattern will be helpful here to do some useful work while that is being processed.

The **Thread-safe Interface** pattern ensures that intra-component method calls avoid self-deadlock and minimize locking overhead [2]. There are scenarios where one method will acquire a lock and then calls another method that tries to reacquire the same lock. This leads to a self-deadlock situation. To avoid this we have synchronize methods on outer boundaries on implementation and avoid synchronizing the inner methods. We achieved this by pushing acquiring and releasing of locking to outer-boundary methods. We made sure that no database read and write operations are inside the code where acquiring and releasing of lock is done.

As mentioned before since the monitor object is product id, the performance of the system is not degraded, since respective users can still buy/update at same time when product id is not same i.e. if they are not trying to make changed on same product.

#### **Monitor object: product id**

These are the methods which needs to be synchronized. So, these methods can be synchronized if product id is used as monitor object. It appropriate to lock on product id since, we only need to acquire and release locks only when:  
-> when 2 customers threads are trying to buy same product at same time  
-> when 2 admins threads are trying to update/remove same product at same time  
-> When a customer thread and a admin are trying to update and other trying to purchase same product at same time.

**Problem faced while trying synchronize on product id as monitor object [3]:**

We can only make 2 methods run mutually exclusively only when they are locked on the same monitor object. Since our application, product id is not from the same object in the methods we are trying to synchronize. Hence we had to figure out a way we could synchronize on same object. We used ConcurrentHashMap, this class has a method “putIfAbsent” which we will check if there is any object associated with existing key there is none it will put an object or if there exists a object it returns it, we used product id as key, so when same product id matches in the other method, it will get the object. Hence we were able to get the same object and were able to make these methods mutually exclusive.

#### **Sample runs:**

**Server: 10.234.136.55**

**Clients: 10.234.136.56  
 10.234.136.57  
 10.234.136.58  
 10.234.136.59  
 10.234.136.60**

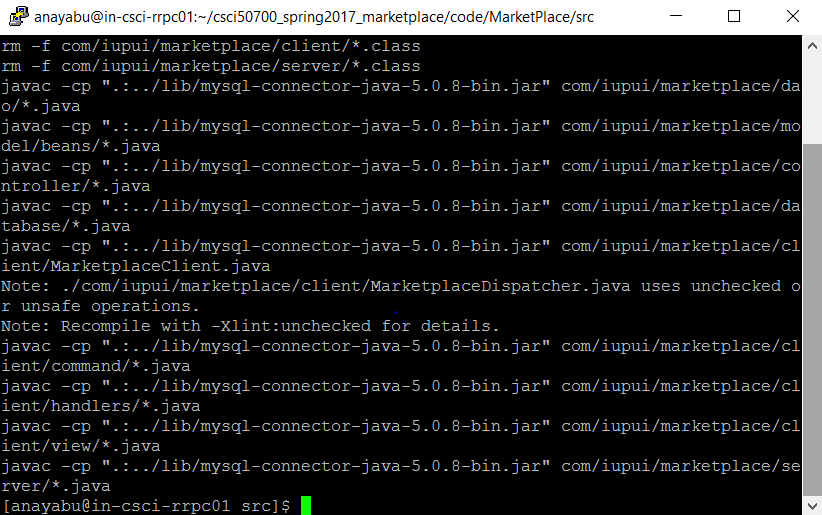
**Database:**

[**https://in-csci-rrpc01.cs.iupui.edu/phpmyadmin/**](https://in-csci-rrpc01.cs.iupui.edu/phpmyadmin/)

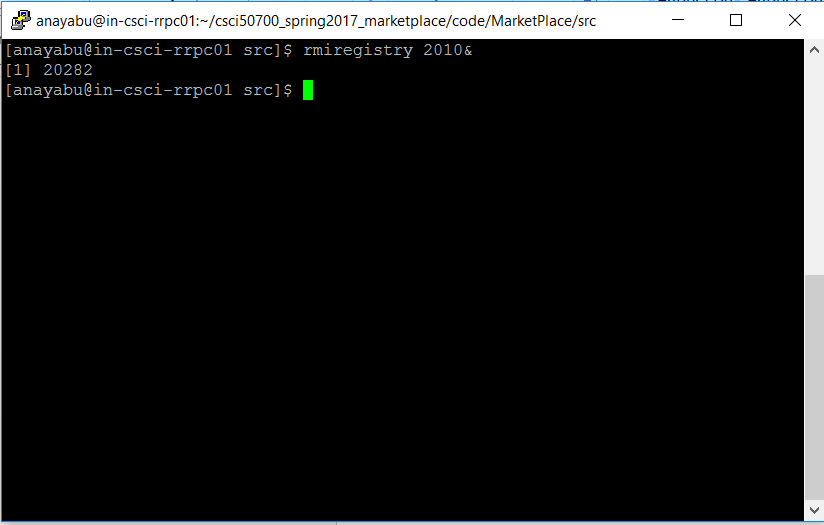
**Username: anayabu  
password: marketplace**

**Sample users: {username,password}  
admin : {admin,admin}**

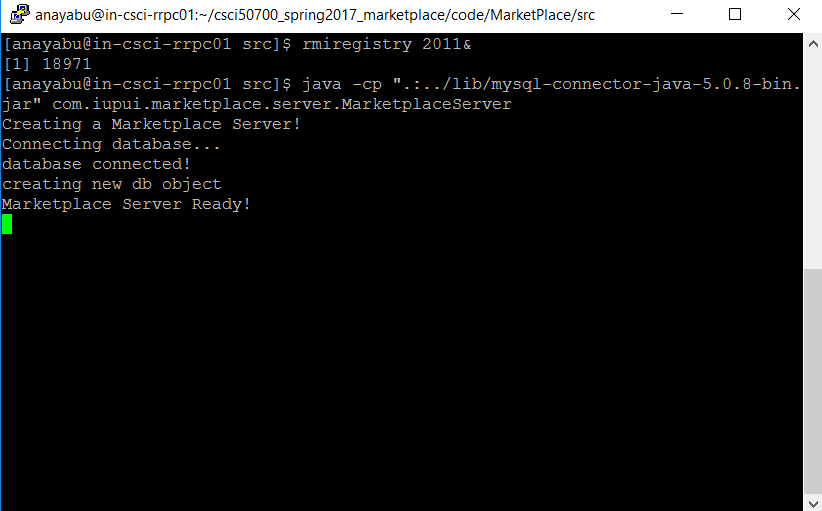
**Users: {user1:user, user2:user, user3:user}**

**Compilation**:

**RMIRegistry:**

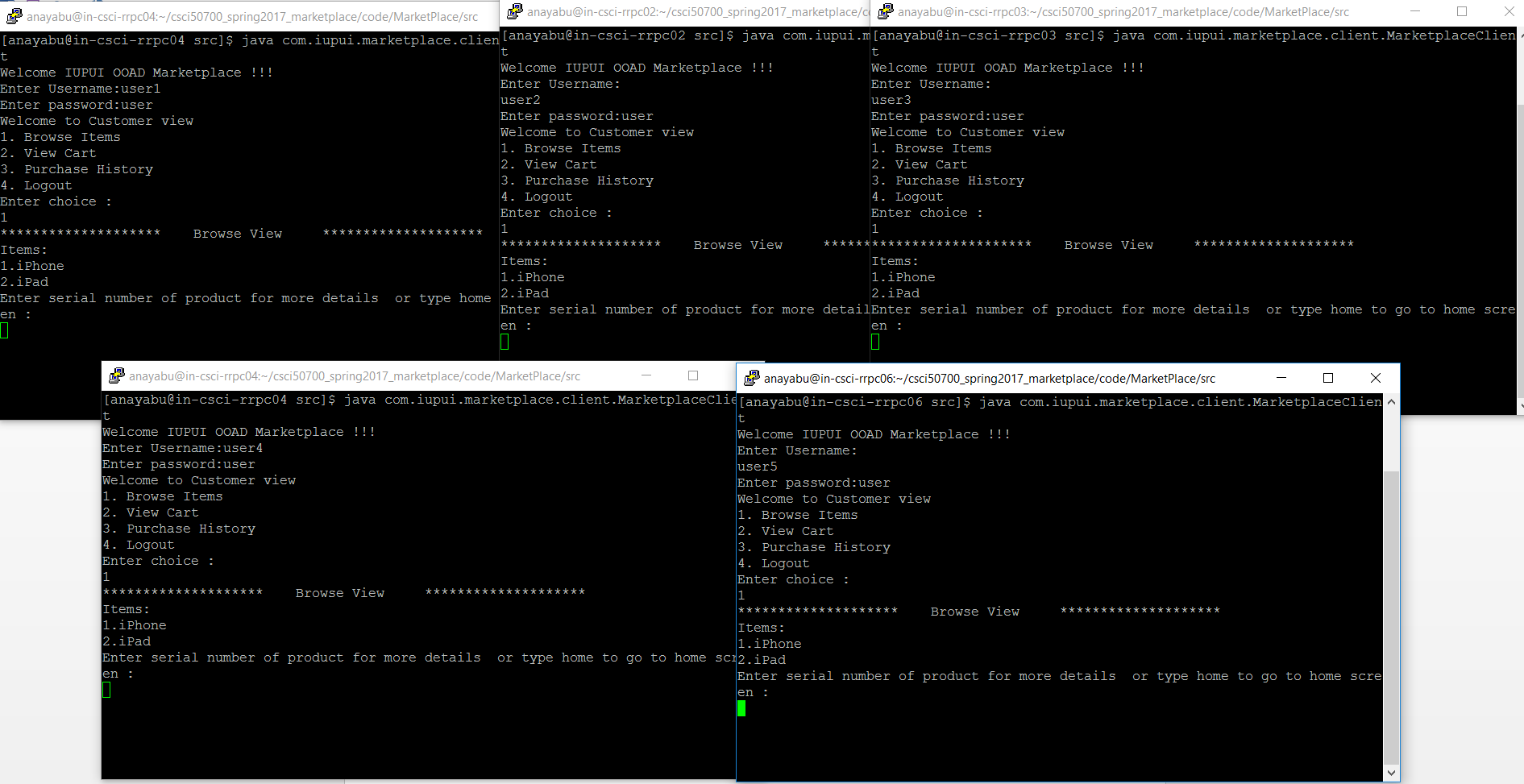


**Server:**



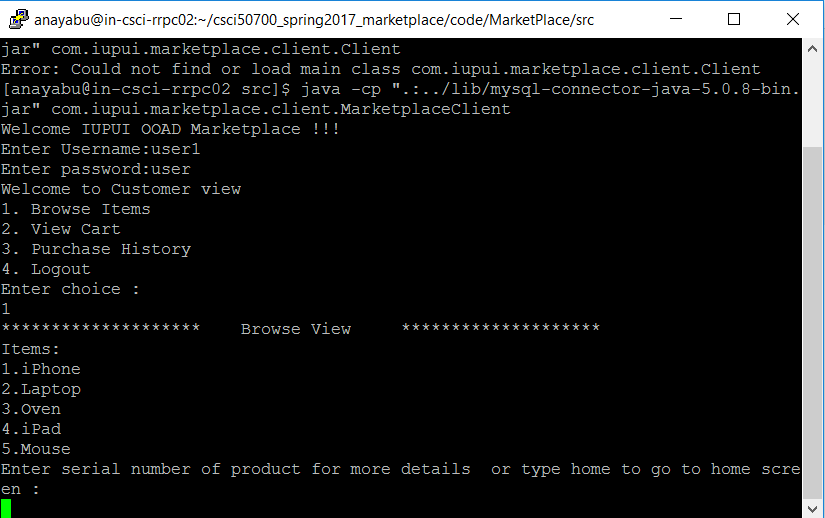
**Client:**

**On 5 different clients**

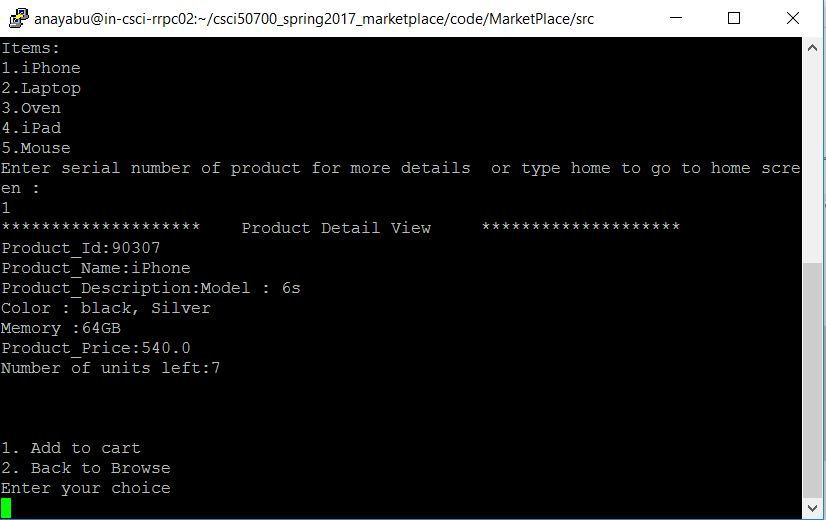


**For Customers:**

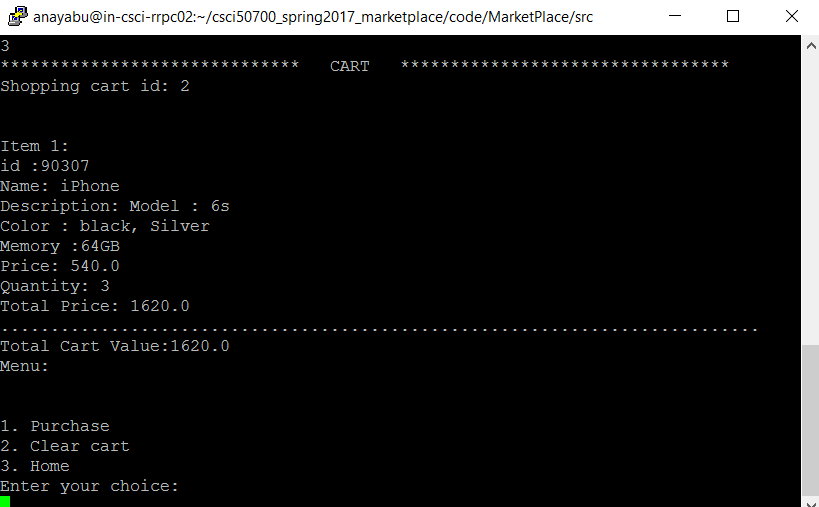
**Browse View:**



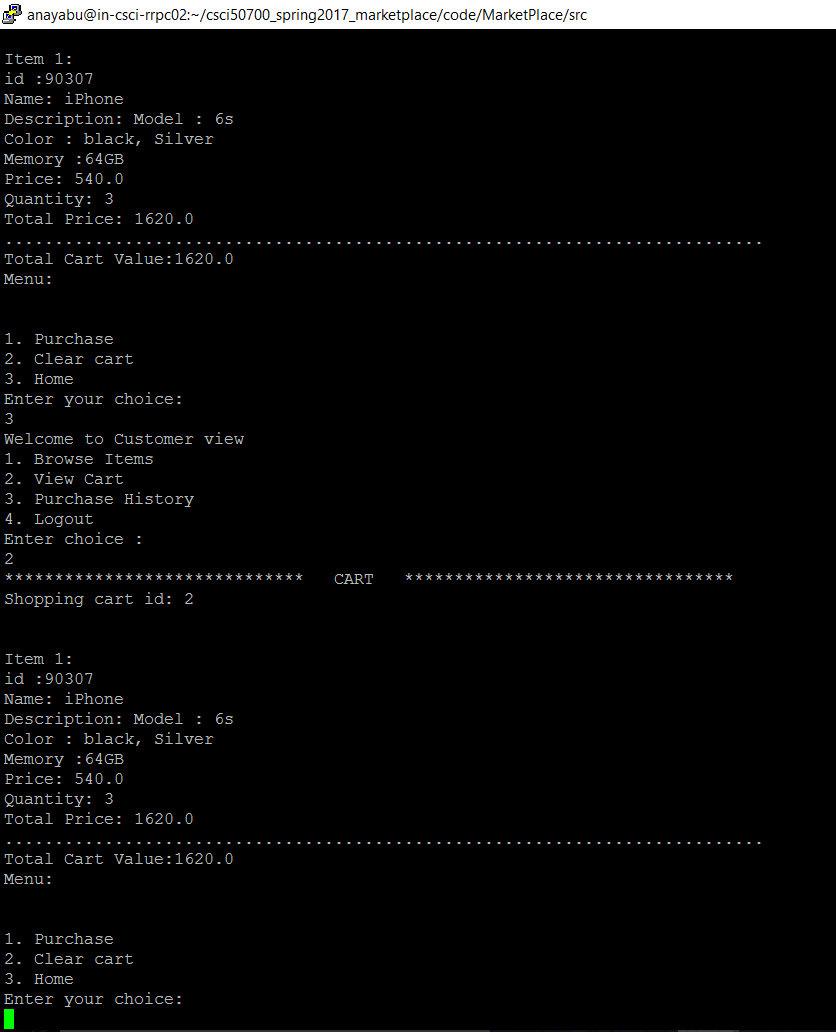
**Product View:**



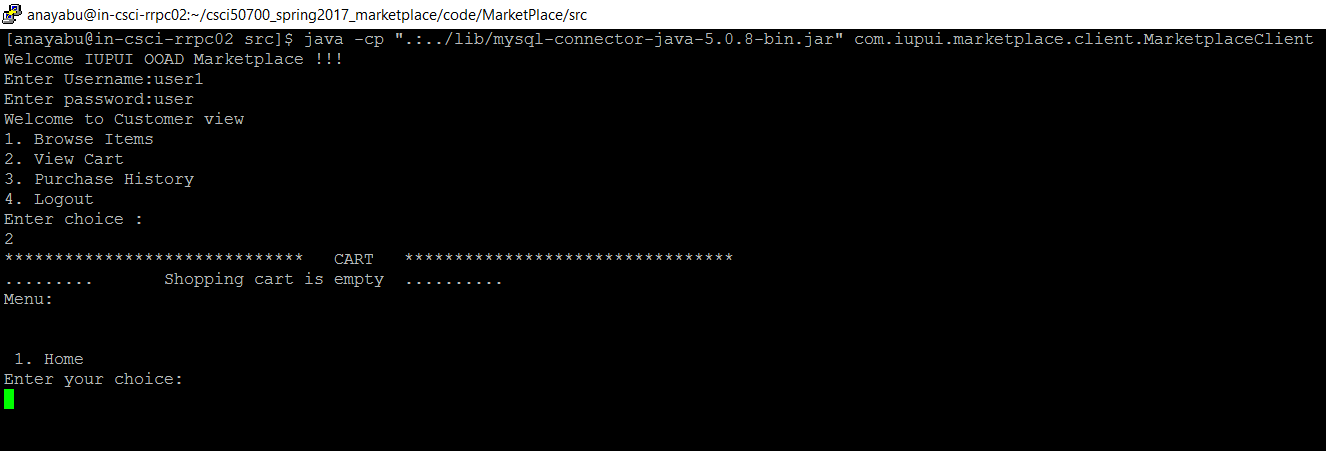
**Cart View:   
when user added 3 iPhones**



**Cart View from home:**

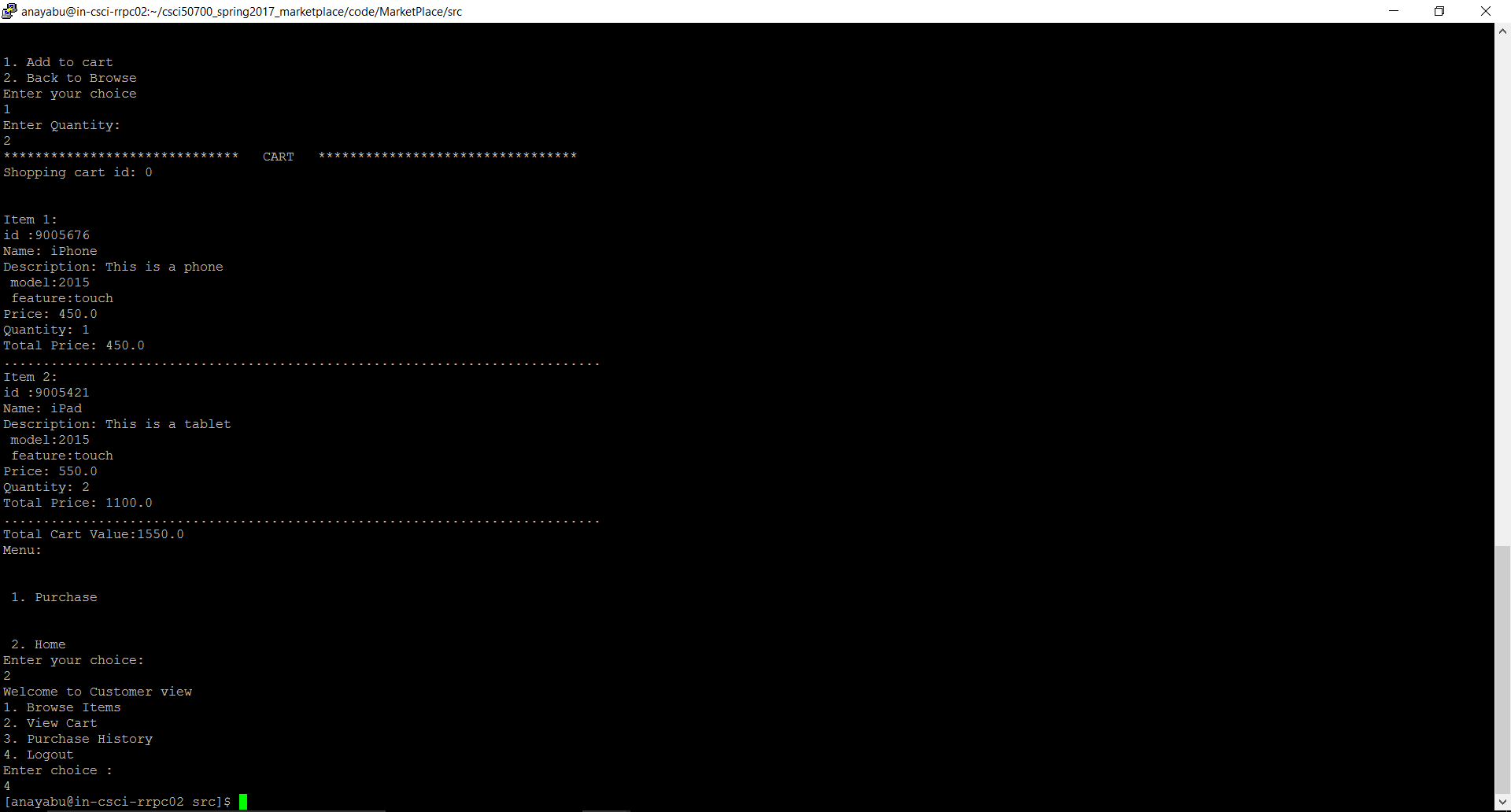


**If user has no items in cart:**

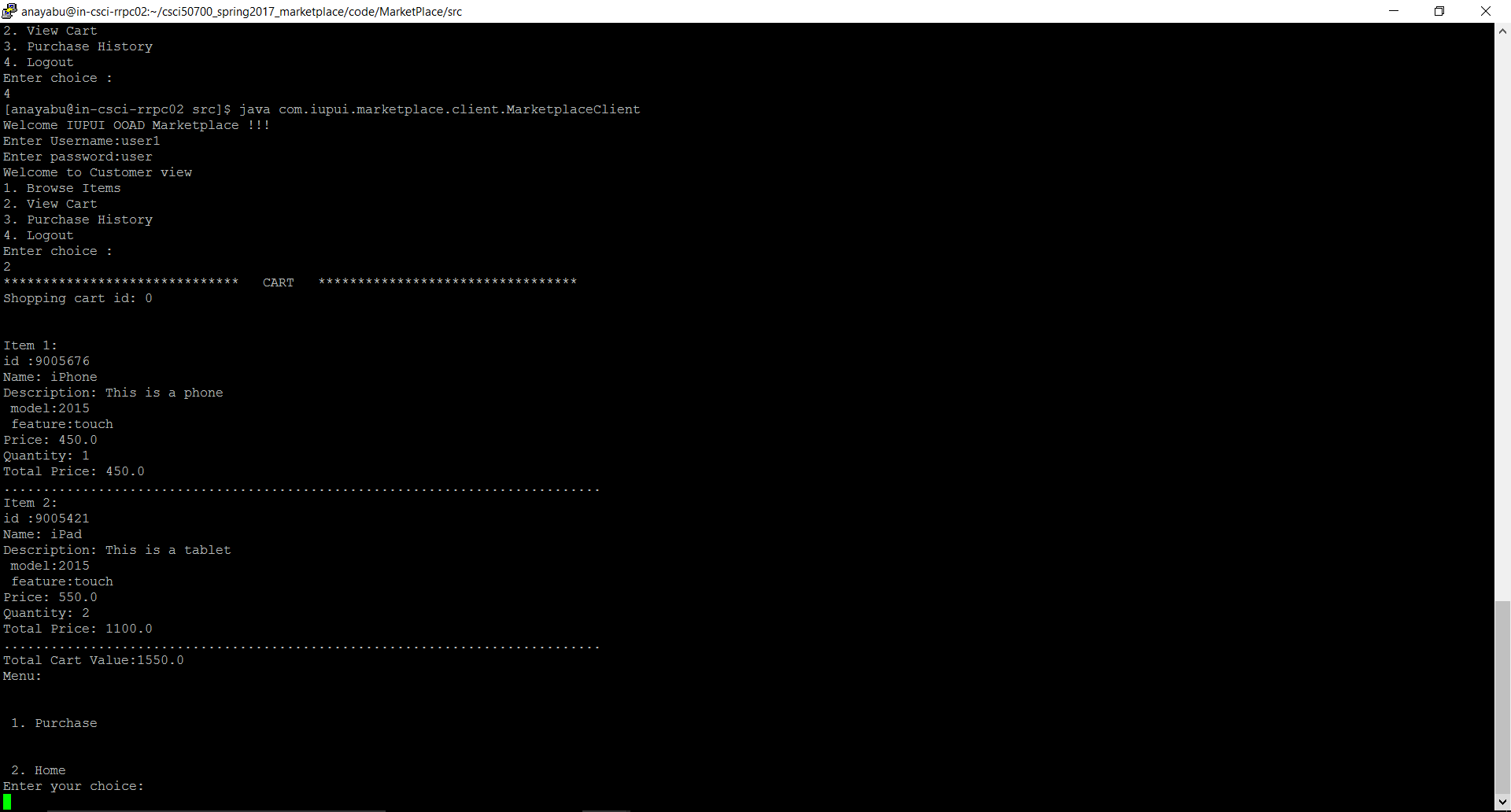


**When user adds item to cart, logs-out and comes back to see his cart of previous session:**

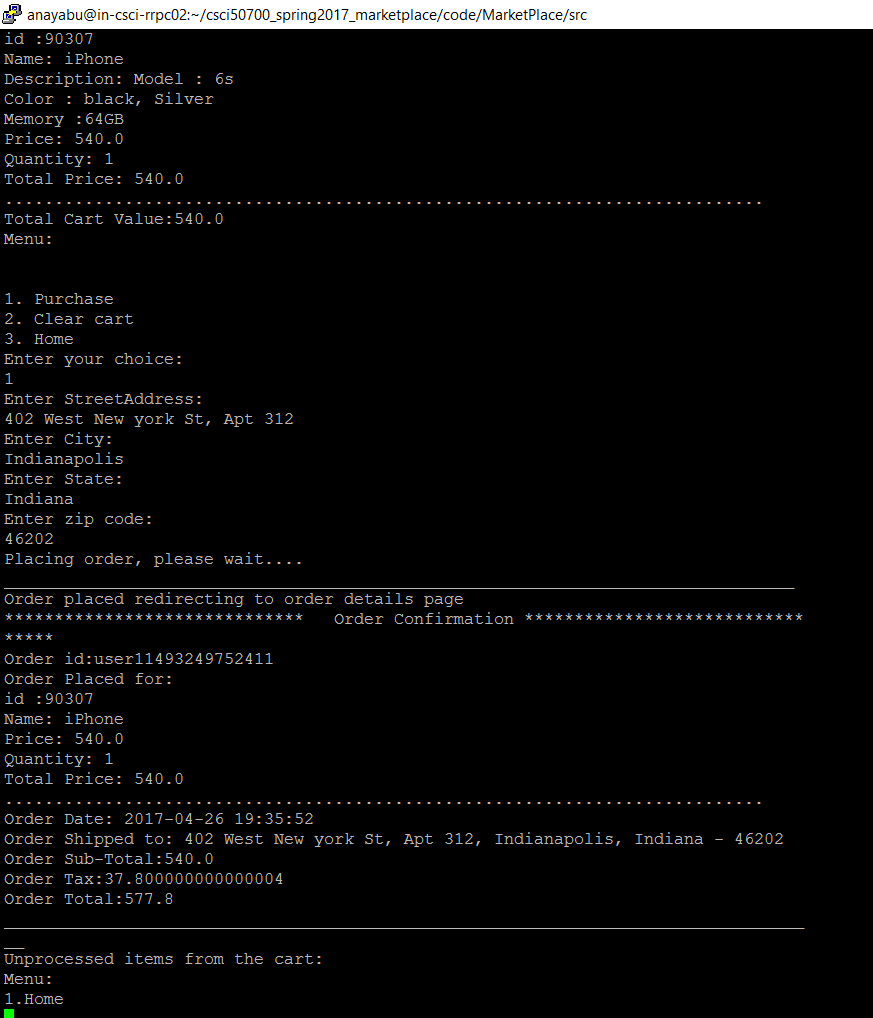
1. **User logs out after adding:**



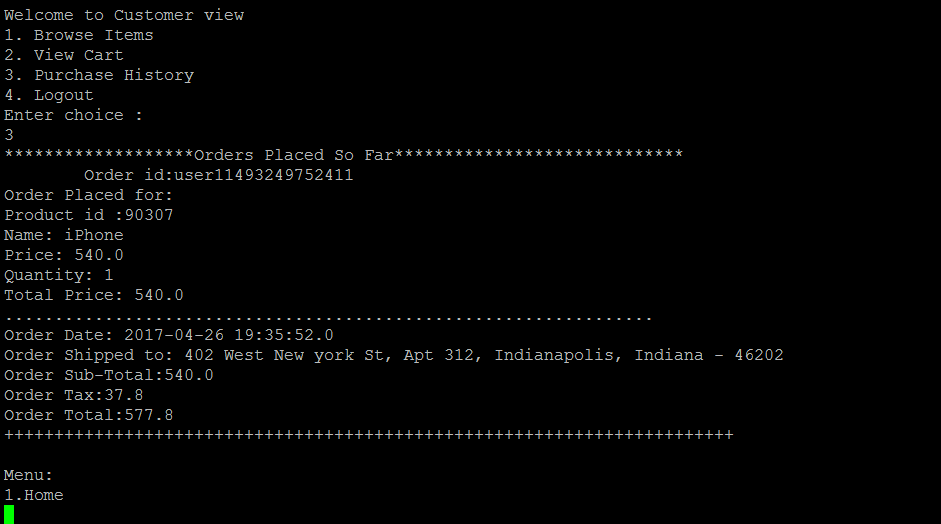
1. **Cart when he comes back**



**Purchase:**

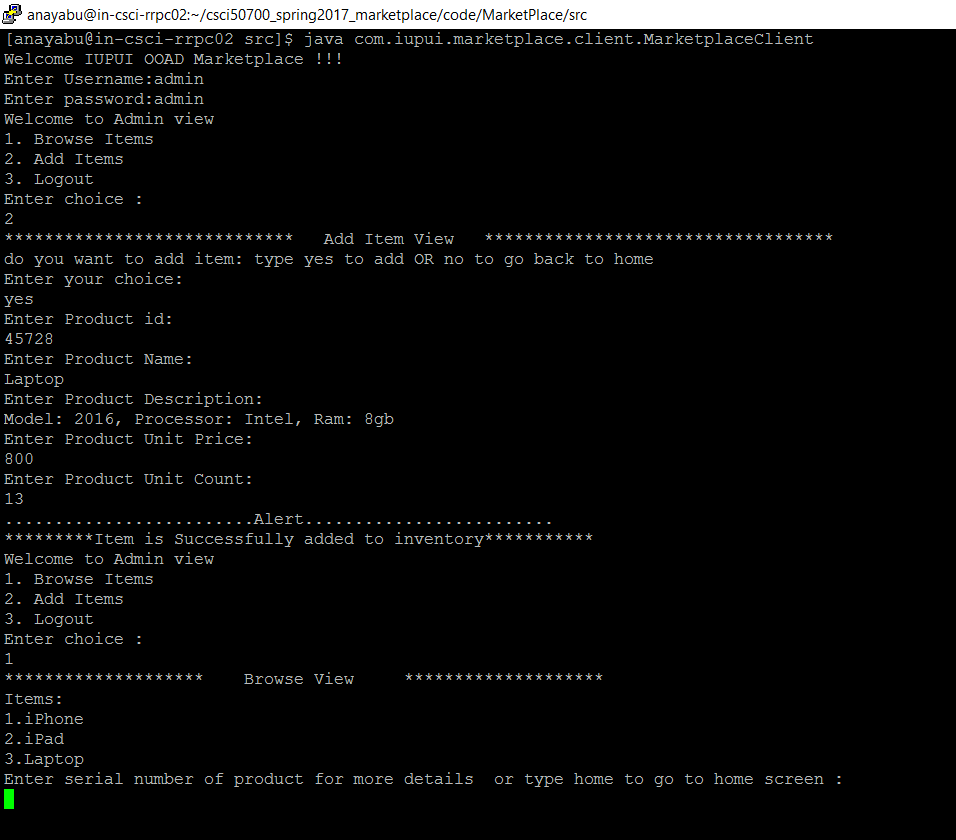


**Order History View:**

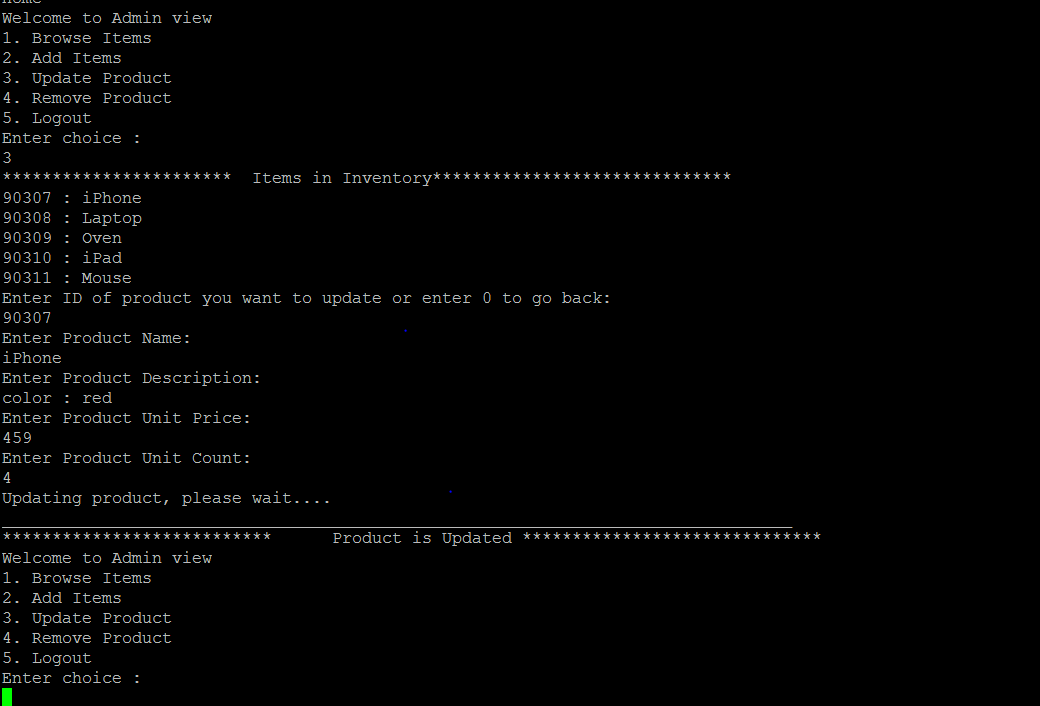


**For admin:**

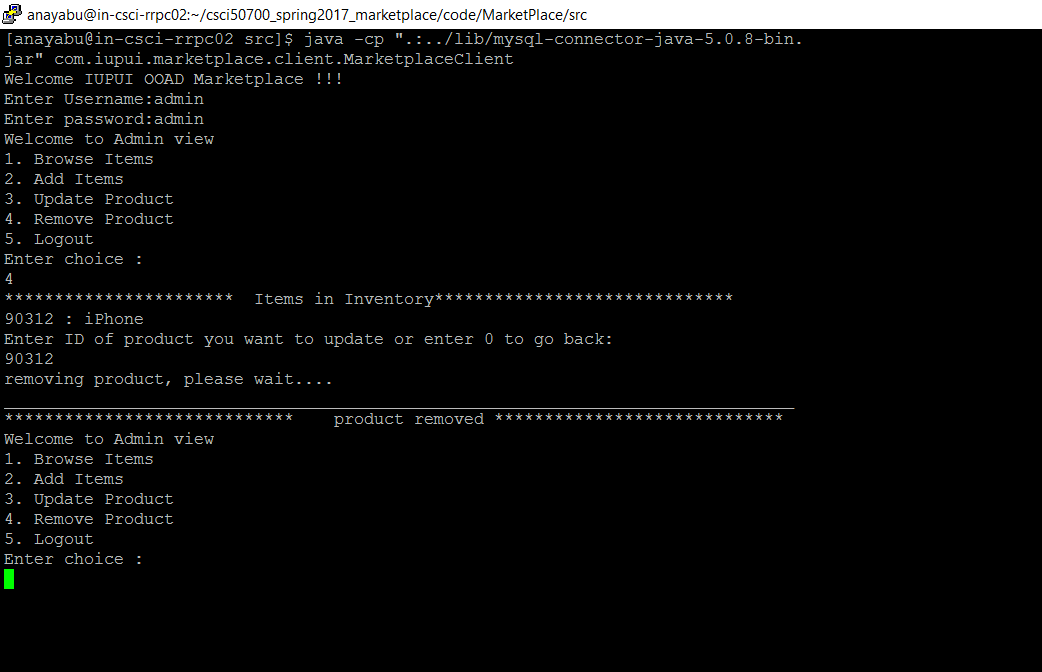
**Add item:**



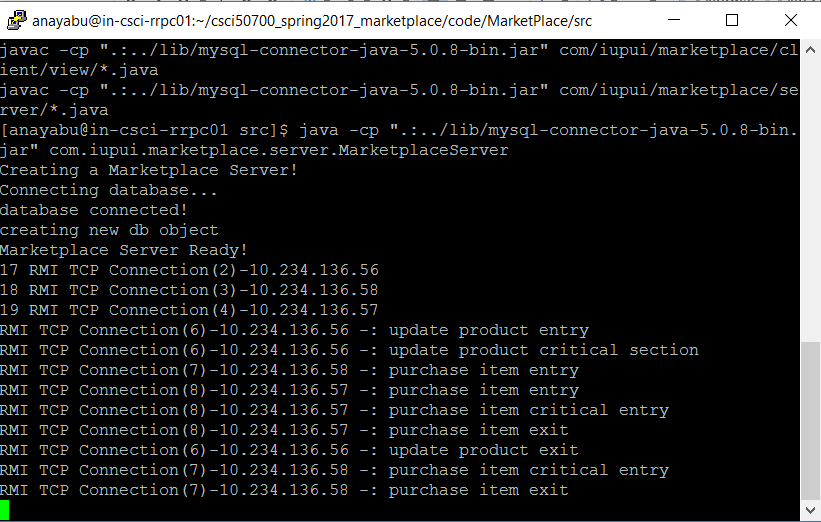
**Update item:**



**Remove item:**



**One of synchronization scenario:**



**Scenario:**

2 customer threads: buying 2 different products with product id: 90313, 90314  
1 admin thread trying to update product with product id: 90314

So, if the update was 1st to invoke then, update thread which is updating 90314 and customer thread which is purchasing 90313 can be executed concurrently and that’s what happened.

And purchase of 90314 can only happen after update process has been done. If we see the above screen shot you will notice that what was happened.

#### **References**

*[1]* - <https://www.youtube.com/watch?v=zUYLY8kYavs>

*[2] -* Strategized Locking, Thread-safe Interface, and Scoped Locking - Patterns and Idioms for Simplifying Multi-threaded C++ Components, Douglas C. Schmidt, Department of Computer Science, Washington University. <http://www.cs.wustl.edu/~schmidt/PDF/locking-patterns.pdf>

*[3] -* [*http://stackoverflow.com/questions/6616141/java-threads-locking-on-a-specific-object*](http://stackoverflow.com/questions/6616141/java-threads-locking-on-a-specific-object)