# Online Marketplace

## Akhil Nayabu

## 2000075395

## Assignment #4

## Course Number: CSCI 50700 – 23706

## Course Name: Object-Oriented Design and Programming

### Overview

In Assignment #4 you will build upon and improve the existing framework that you have created in the first three assignments for our Marketplace App. For this particular assignment you will be examining the impact that **Concurrency** has on our application. As mentioned in lecture Java RMI provides the vague definition on the handling of concurrency (specifically threading):

*"3.2 Thread Usage in Remote Method Invocations*

*A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread. The RMI runtime makes no guarantees with respect to mapping remote object invocations to threads. Since remote method invocation on the same remote object may execute concurrently, a remote object implementation needs to make sure its implementation is thread-safe."*

### Our goal is to examine this definition within the scope of our application. The focus of this assignment will be to implement previously unimplemented functionality in your system as well as to explore, in your report, the consequences of concurrency in our Marketplace Application. In your report you should discuss this “guarantee” that Java RMI offers us with respect to concurrency and the impacts that it has on system design. For this assignment we will make use of the follow machines (listed on the next page) to demonstrate this across the network. 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. You should specify which machine you have selected to be your “server” in your README – failure to do so will result in a 0 on the program execution portion of the grading rubric. For this assignment we will make the assumption that there is only ever one server but many clients. As part of this assignment you should now fully implement the following three required functions: Purchase Item, Add Item, and Browse Item. 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. The key to this assignment is your report and discussion!

### **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.

### 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.

* No changes to domain model from assignment #3

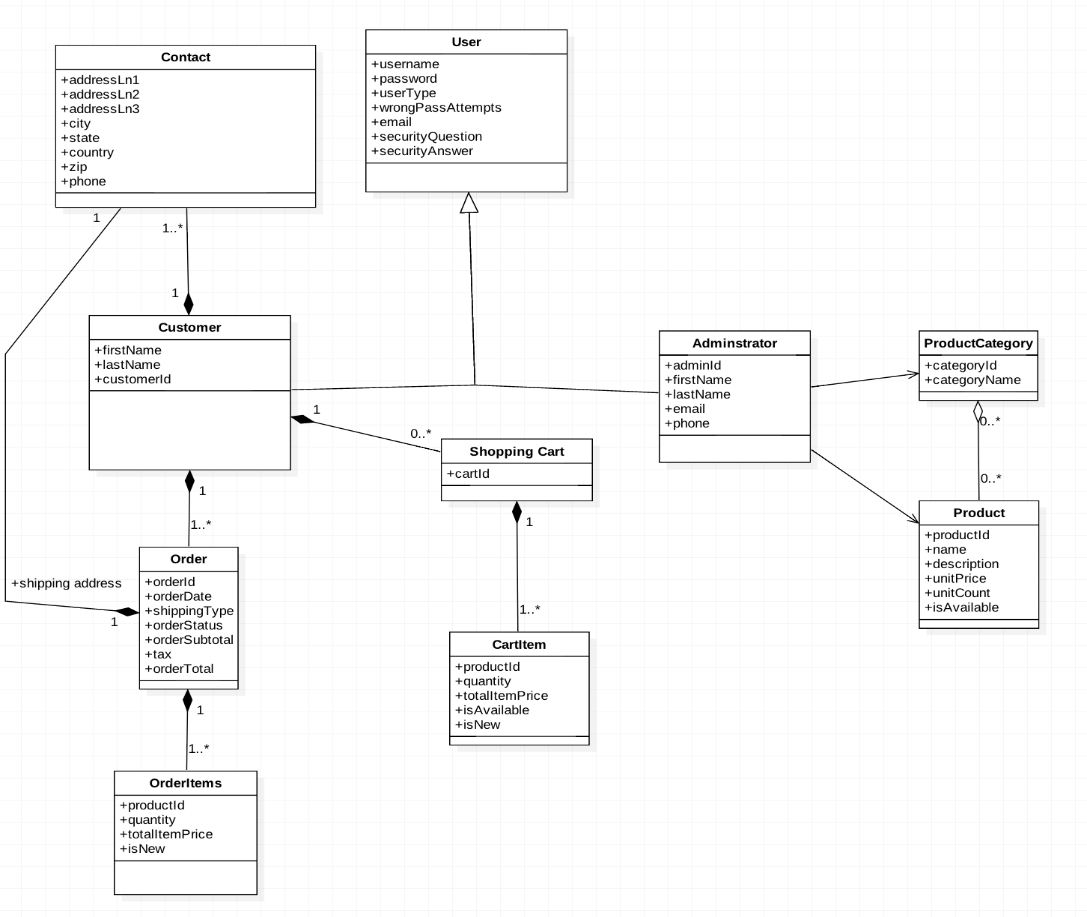


Fig 1: Domain model

### **Assignment #4**

**RMI Concurrency:**

JAVA RMI does not guarantee that they are going to be mutually exclusive and two calls on the same remote object may be executed concurrently unless you implement some synchronization. It is stated in RMI specification that: “The RMI runtime makes no guarantees with respect to mapping remote object invocations to threads.” As far as this assignment is concerned, I would say the system is concurrent as all changes

### **Functionalities Implemented:**

#### **Browse Items**

This view is common for both admin and client. 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.

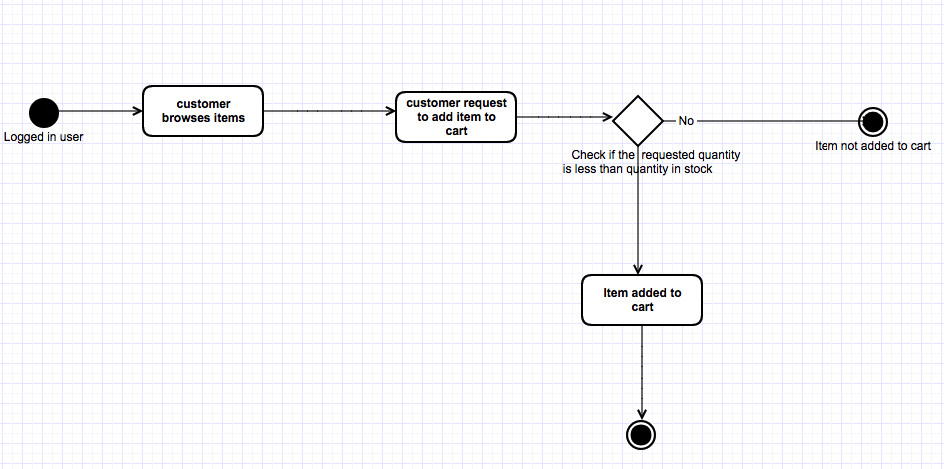
#### **Add Item**

This 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 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.

Here is the activity diagram for this function:



#### **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.

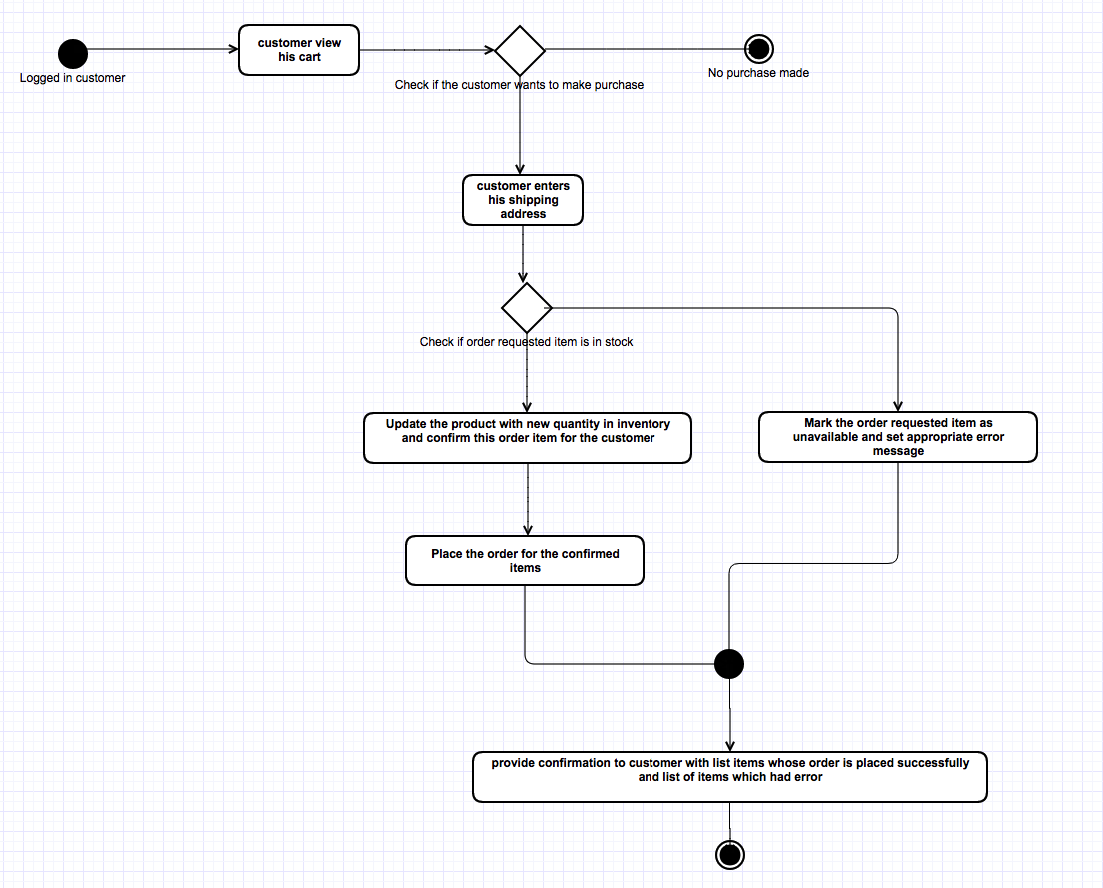
After validation and order has been placed, it redirects user to order confirmation view. Where details of the order is displayed. All the items which were placed and all those which were not placed. It plays order id, order date, order total, shipping address (item total + total tax).

The items which were placed and 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.

Activity Diagram for purchase:



#### **Order History View**

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

Screen shot is provided in sample run.

### Assignment#3

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

Fig 2: When user has access to the method

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

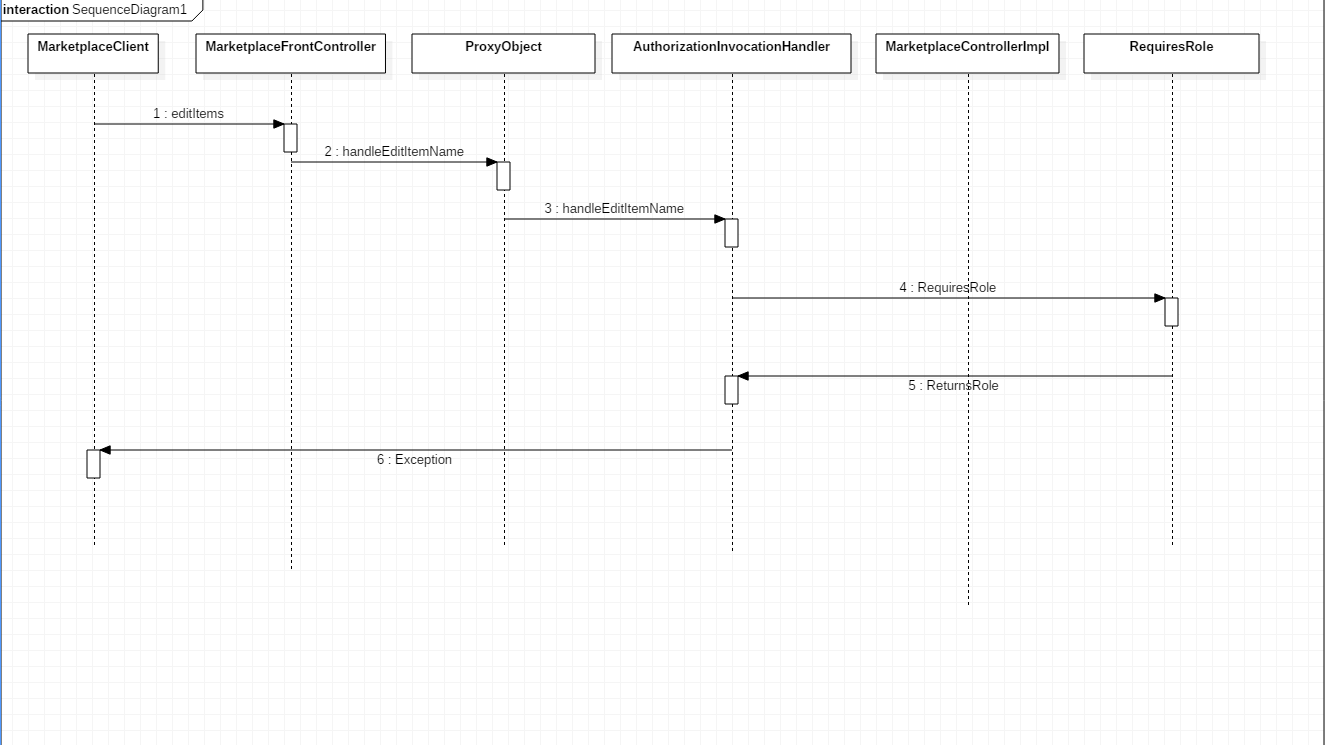


Fig 3: When user has no access the method

### Patterns from Assignment#2

### Front Controller Pattern

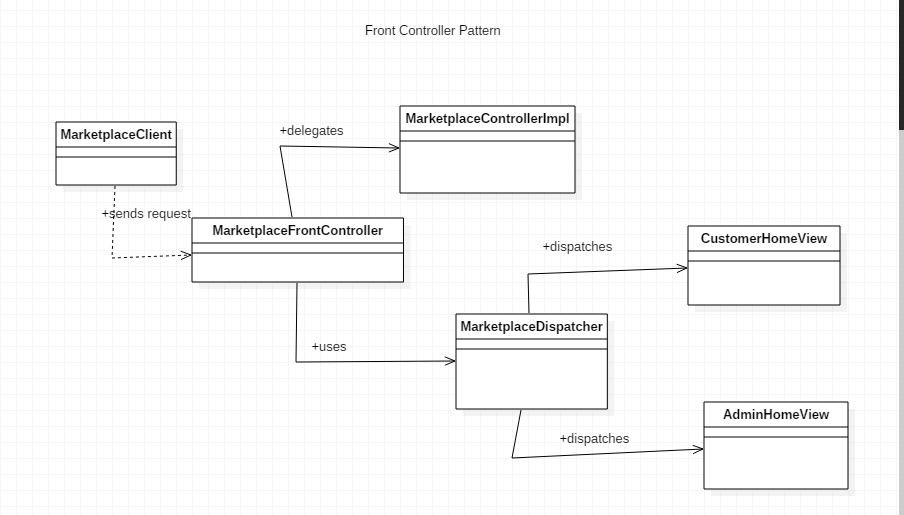


Fig 4: Front Controller

### Abstract Factory Pattern

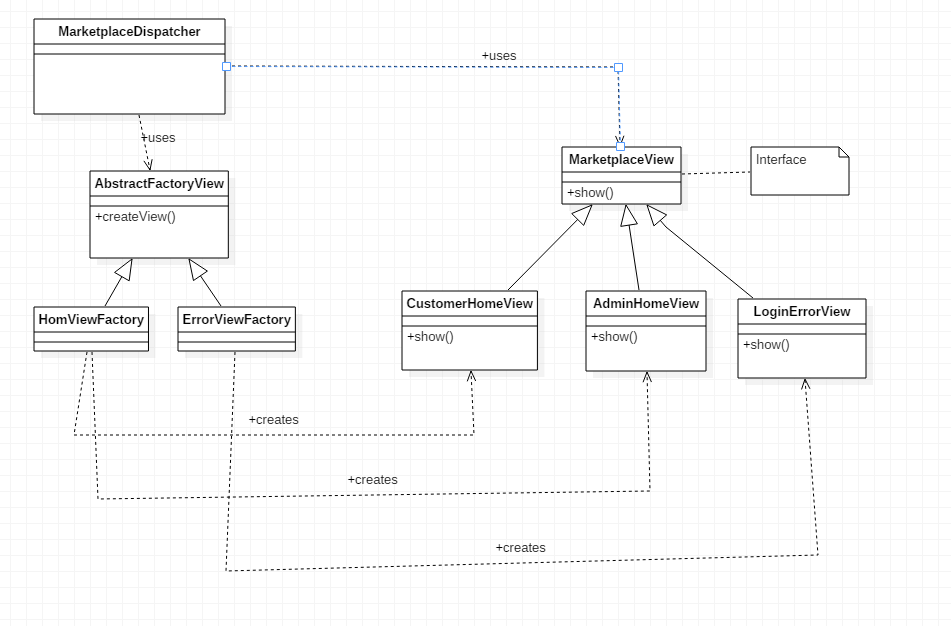


Fig 5: Abstract Factory

### Command Pattern

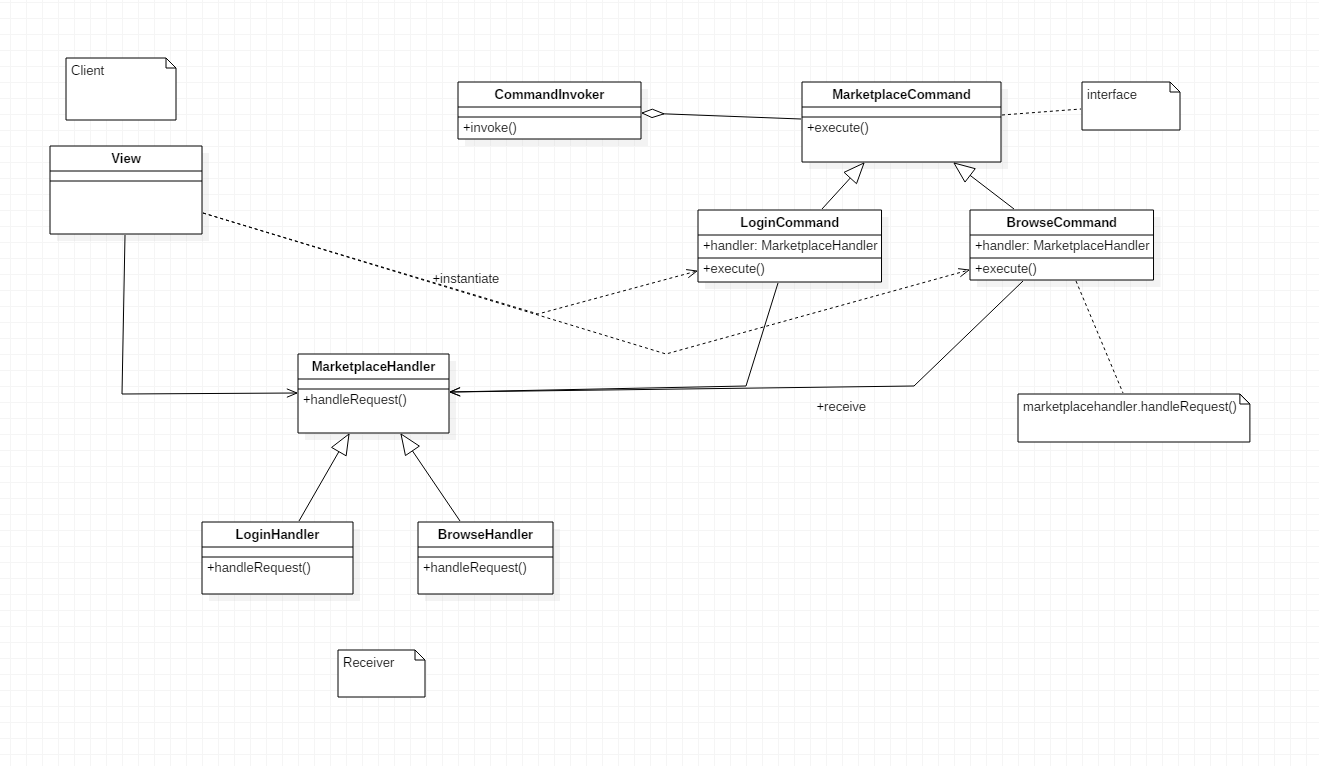


Fig 6: Command Pattern

# 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**

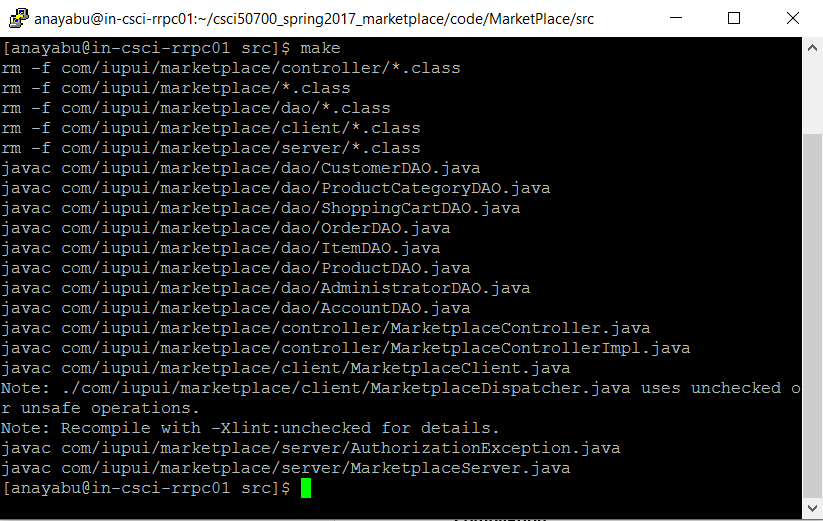
**Users:**

**Admin: username: admin  
 password: admin**

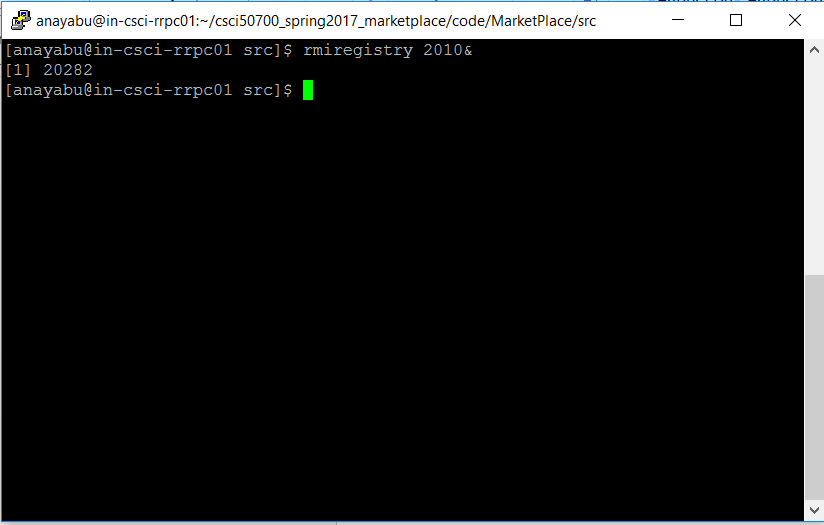
**Customers: username: user1  
 password: user**

**{Username: password} = {user2: user, user3: user, user4: user, user5: user}**

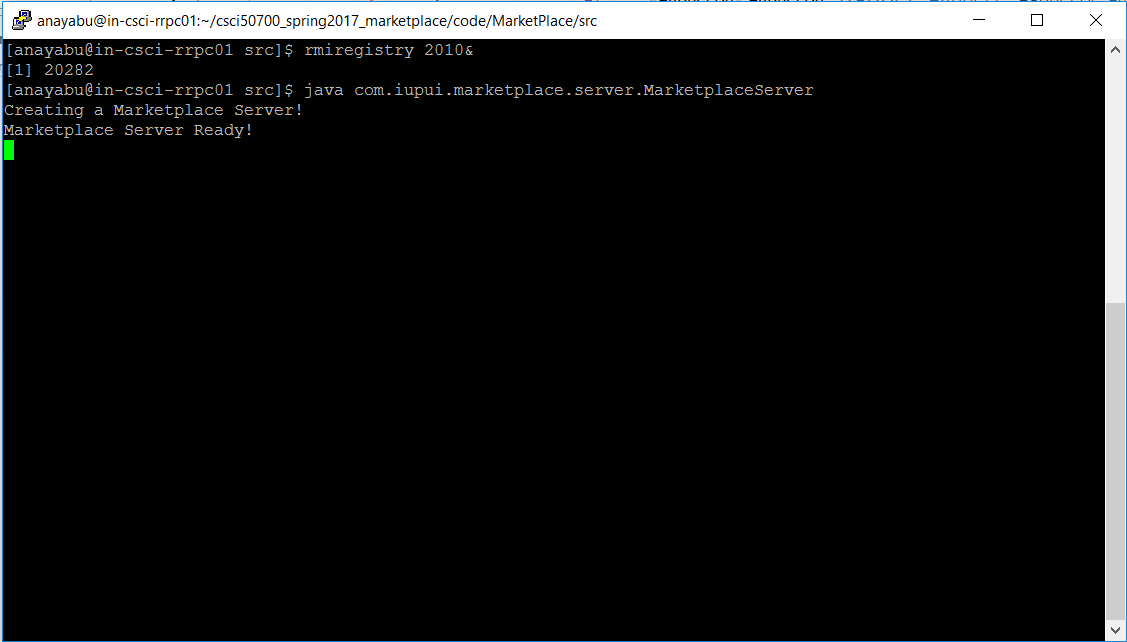
**Compilation**:



**RMIRegistry:**

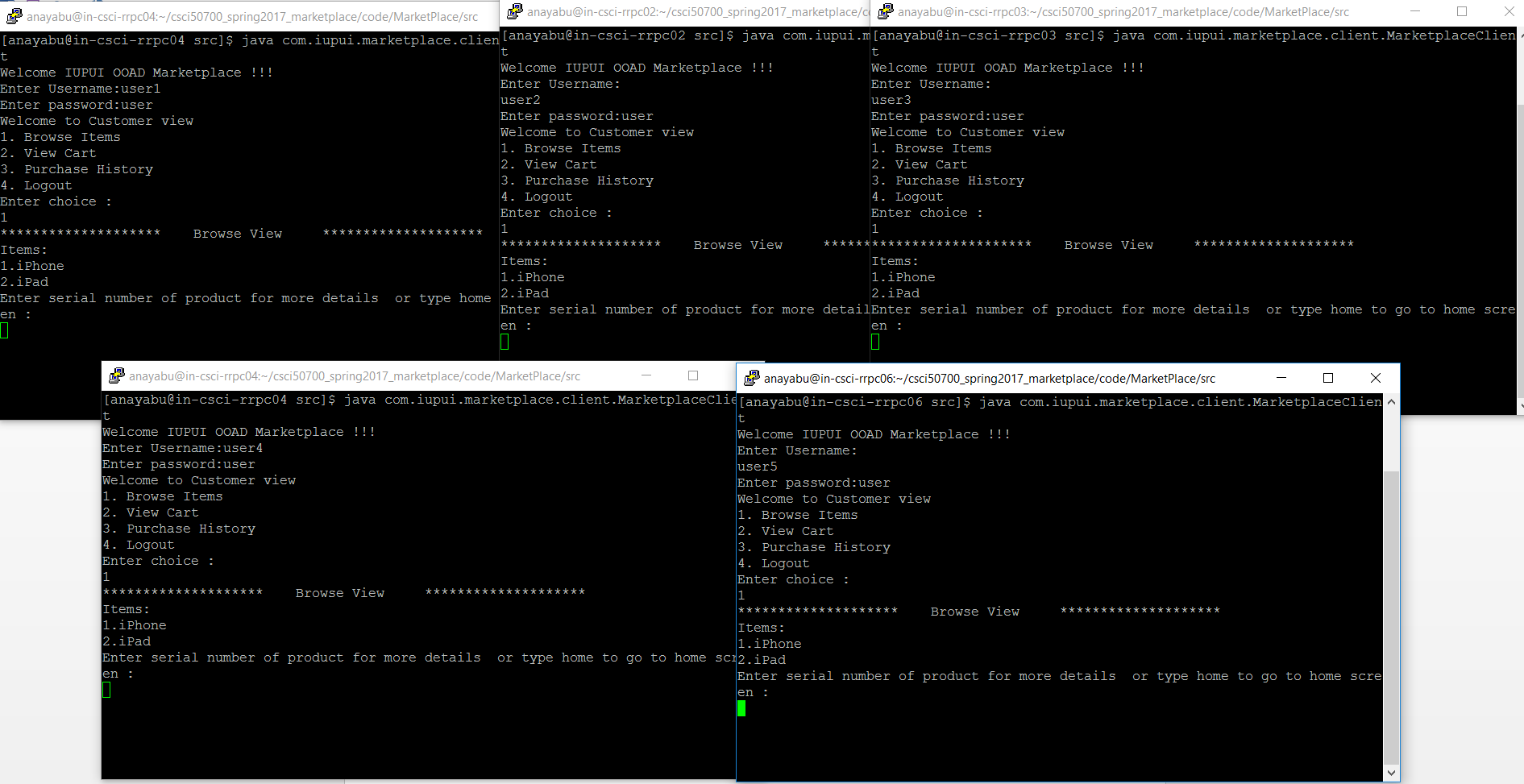


**Server:**



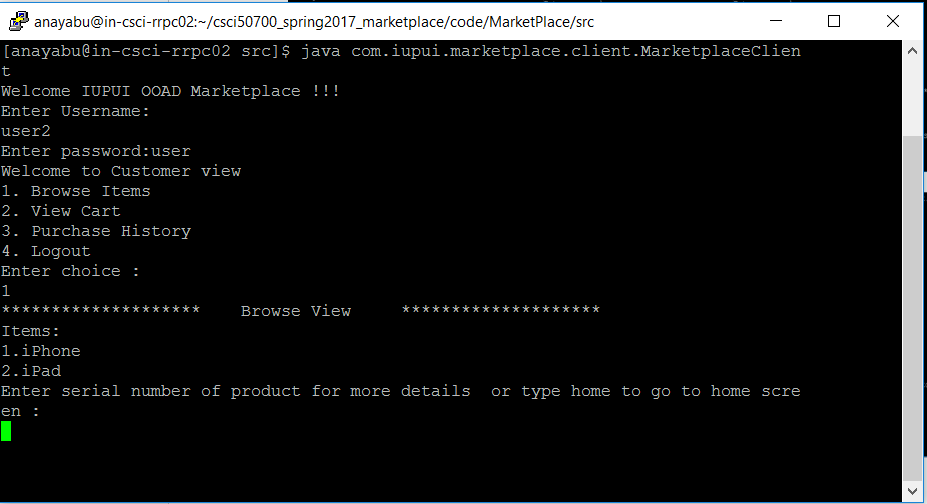
**Client:**

**On 5 different clients**

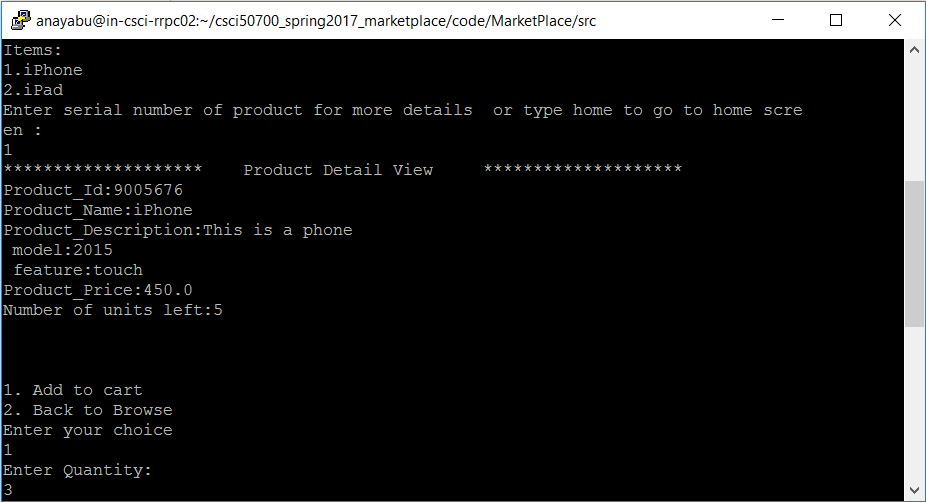


**For Customers:**

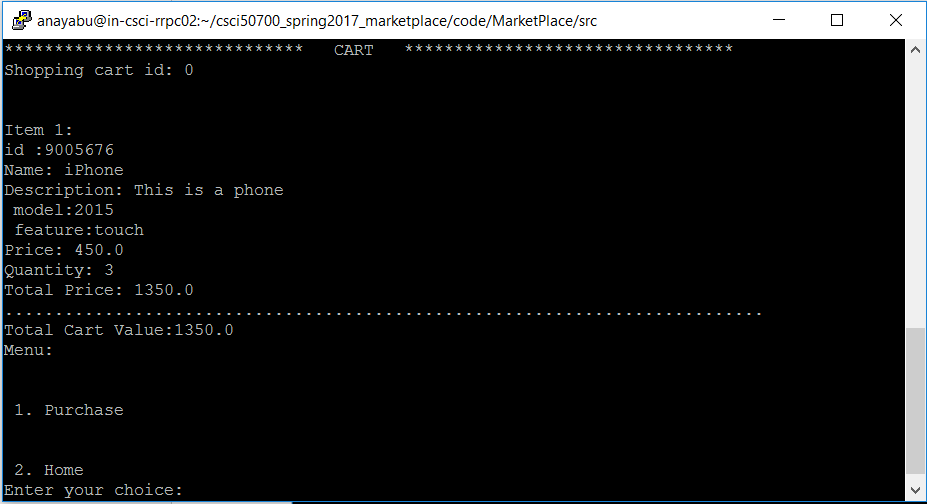
**Browse View:**



**Product View:**



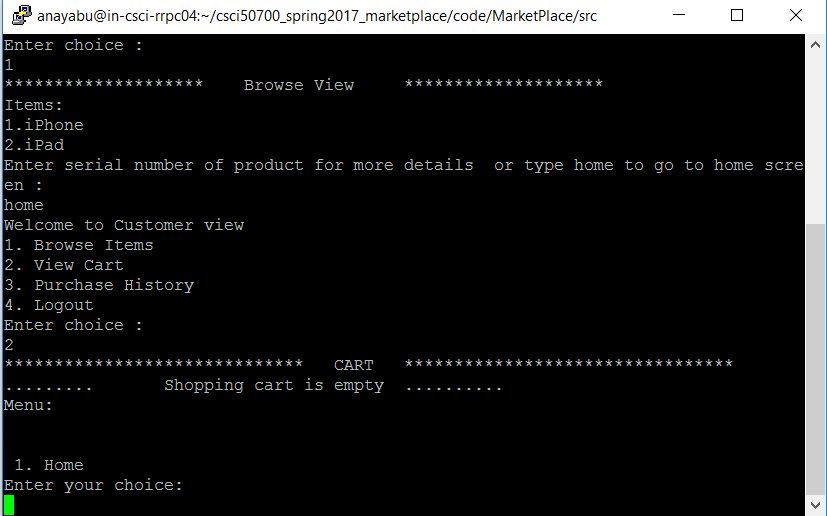
**Cart View:   
when user added 3 iPhones**



**Cart View from home:**

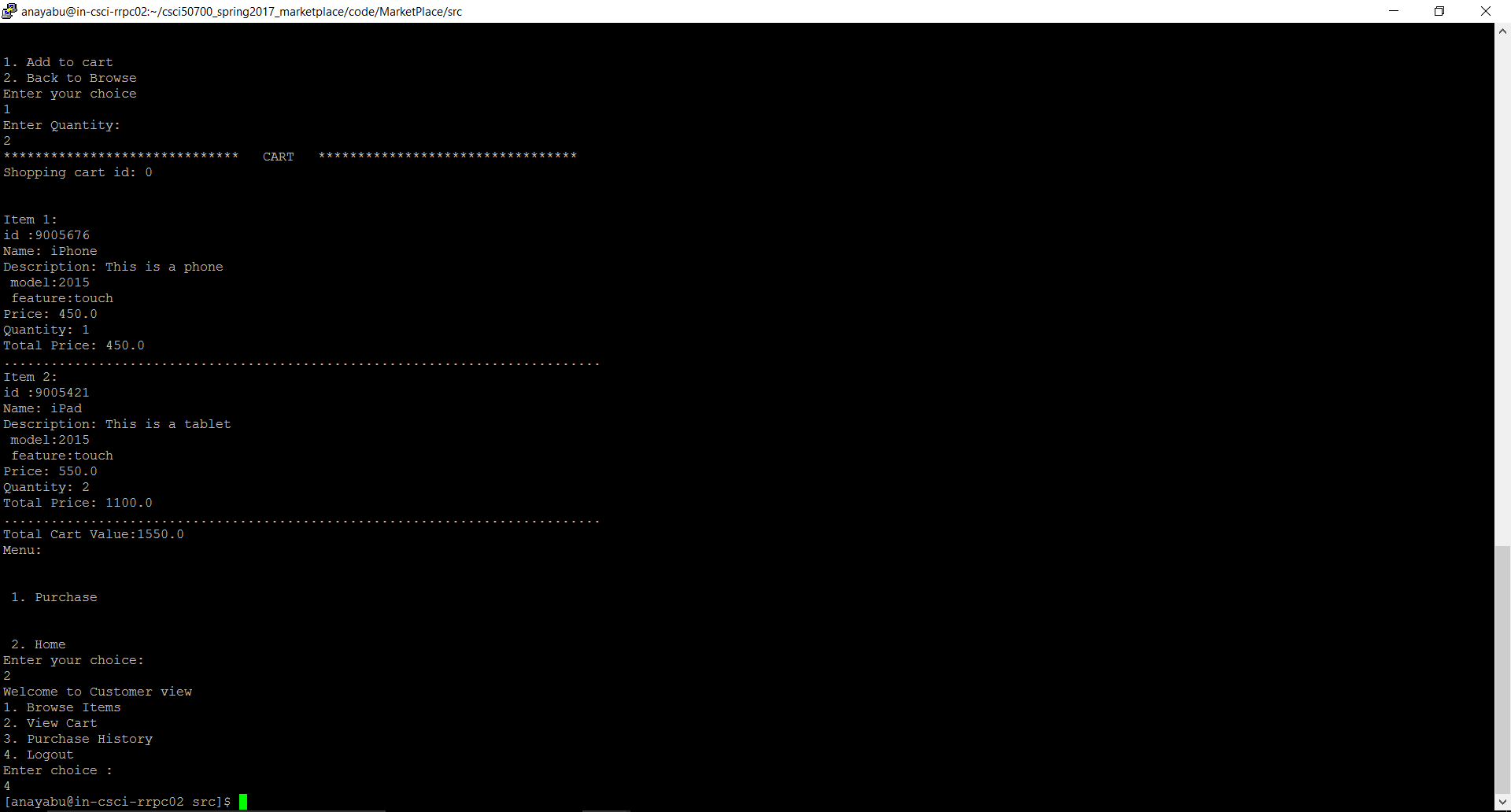


**If user has no items in cart:**

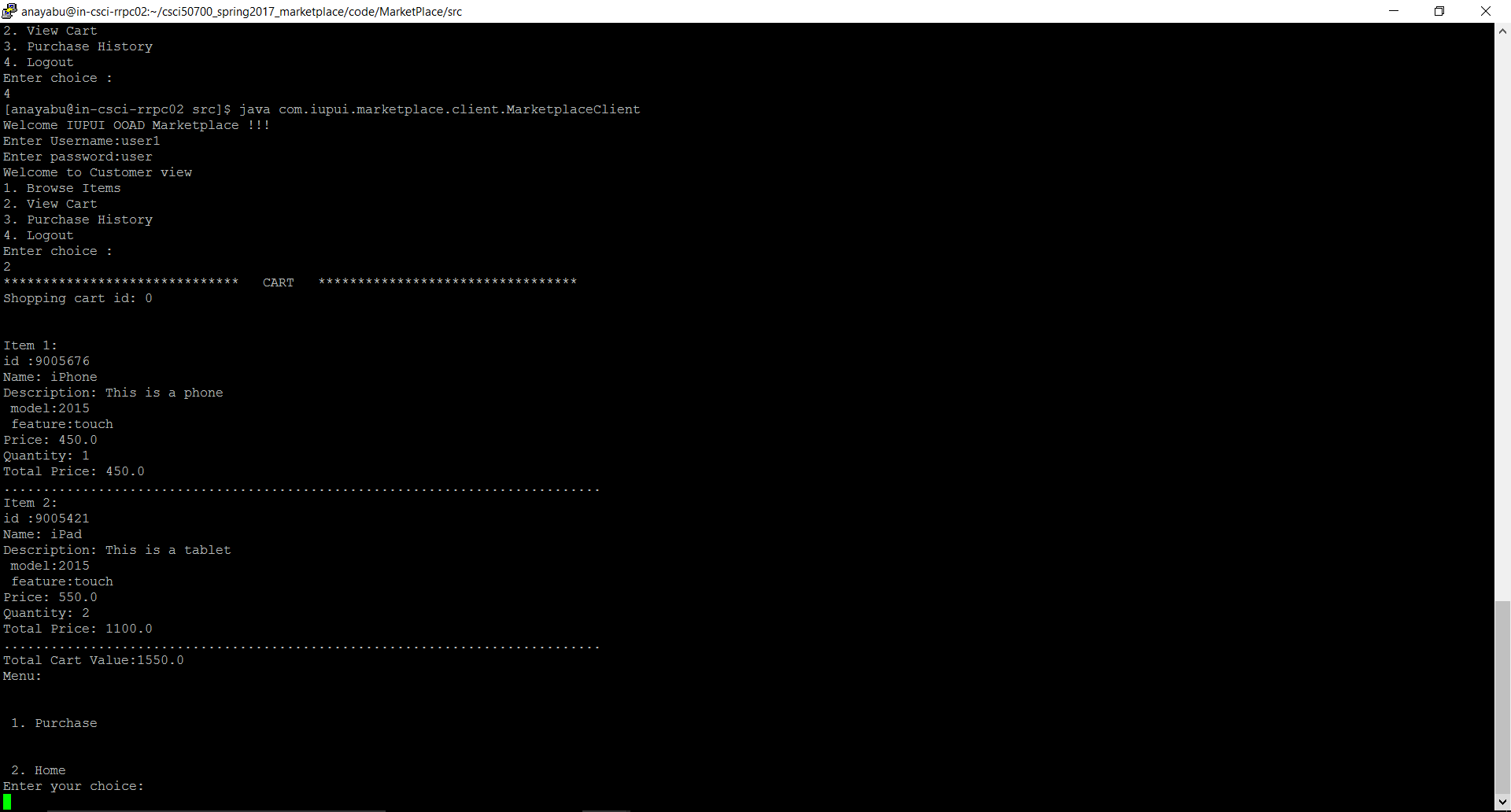


**When user adds item to cart, logs-out and comes back to see his cart of previous session:**

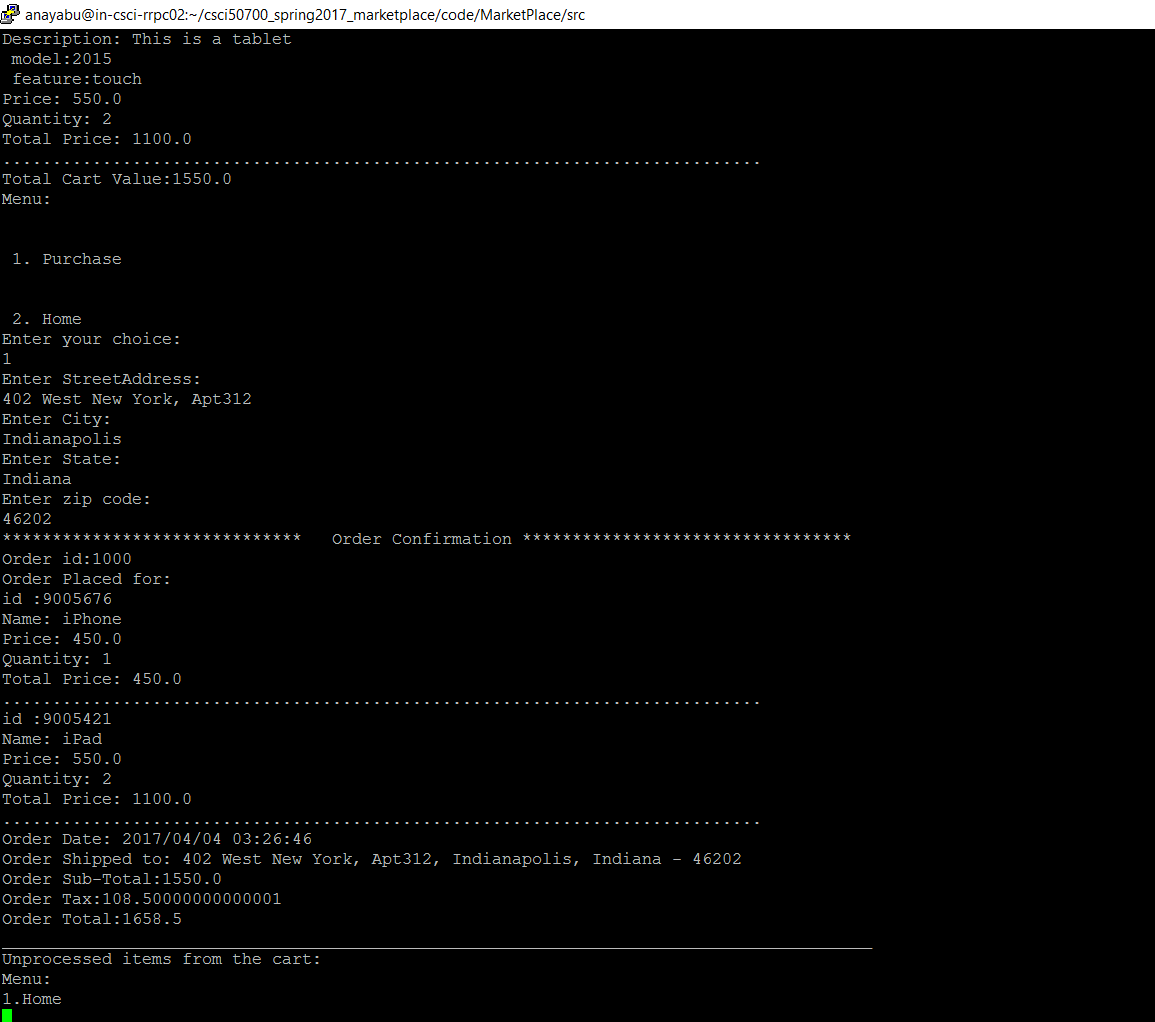
1. **User logs out after adding:**



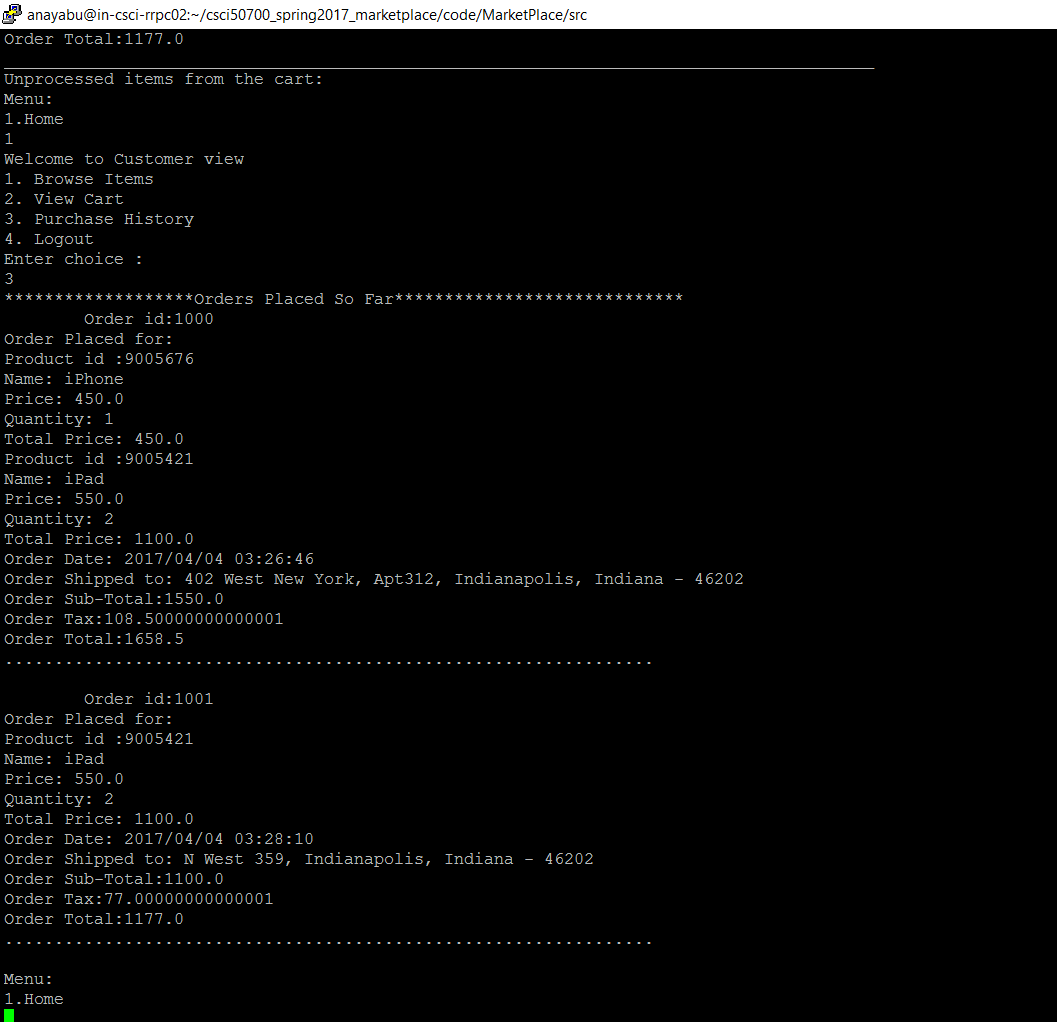
1. **Cart when he comes back**



**Purchase:**



**Order History View:**



**For admin:**

**Add item:**

