# Online Marketplace

## Akhil Nayabu

## 2000075395

## Assignment #3

## Course Number: CSCI 50700 – 23706

## Course Name: Object-Oriented Design and Programming

### Overview

The main motive of assignment #3 is to build upon and improve the existing framework that we have created in assignment#1 and assignment#2. We will be implementing Authorization pattern through Role-Based Access Control which will make use of java annotations to achieve this we will make use of java proxy and reflection pattern. For this Assignment we have two user types: customer and admin, for this assignment authorization and access roles of each user are differentiated using the above stated patterns.

### 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

### 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 #2

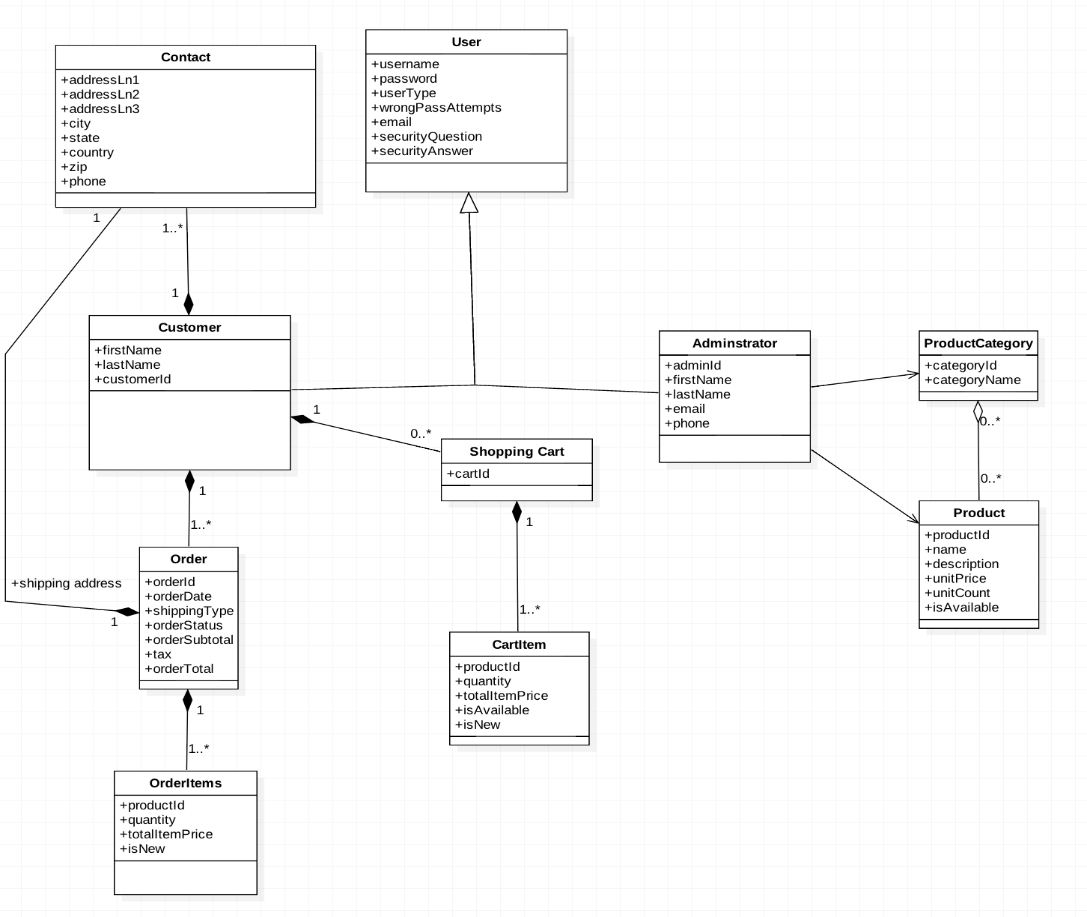


Fig 1: Domain model

### Assignment#3

**Java Annotations:**

We are using java annotations to check what role is assigned to the method which user is trying to invoke. When no annotation is mentioned, method will be invoked. They are very useful when we want to give access based upon the role the current user.

For our assignment annotations are available while runtime, through java reflection by which we will check for the role of the user and methods will only be invoked if role of the user is appropriate.

For this assignment we have used 2 annotations which specifies roles of user, one role being “CUSTOMER” and the other “ADMIN”. Methods with customer role cannot invoked by admin and methods with admin role cannot be invoked by customer.

**Java Proxy:**

It is a design pattern which is used to create and use proxy objects when we want to add or modify some functionality of an already existing class. The proxy object will be used instead of original class object. Proxy will have all the methods that the actual class have. When the proxy object is invoked instead of the original object the proxy decides whether it has to invoke the original method or some handler, in our case authorization handler.

**Role Based Access Control:**

It is regulation of access to user to specific methods based on the role with which they are trying to access it. A specific role is assigned to users by which they can only access the methods to which are they have access to. It is a realization of authorization pattern. We are implementing this through java annotations.

### 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:

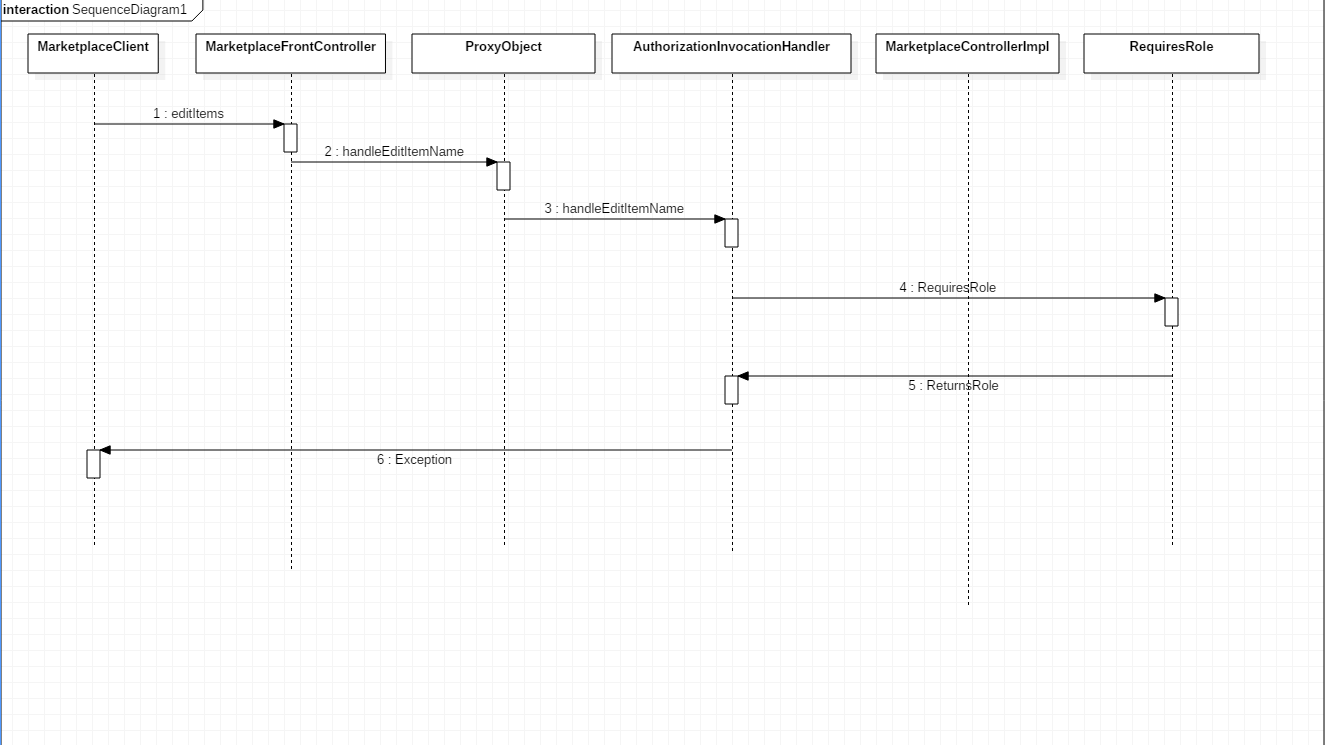


Fig 3: When user has no access the method

### Patterns from Assignment#2

### Front Controller Pattern

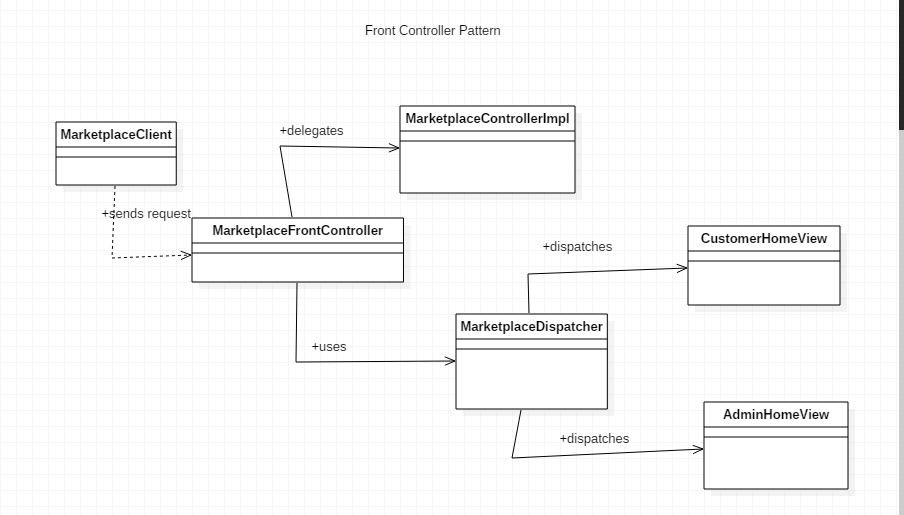


Fig 4: Front Controller

### Abstract Factory Pattern

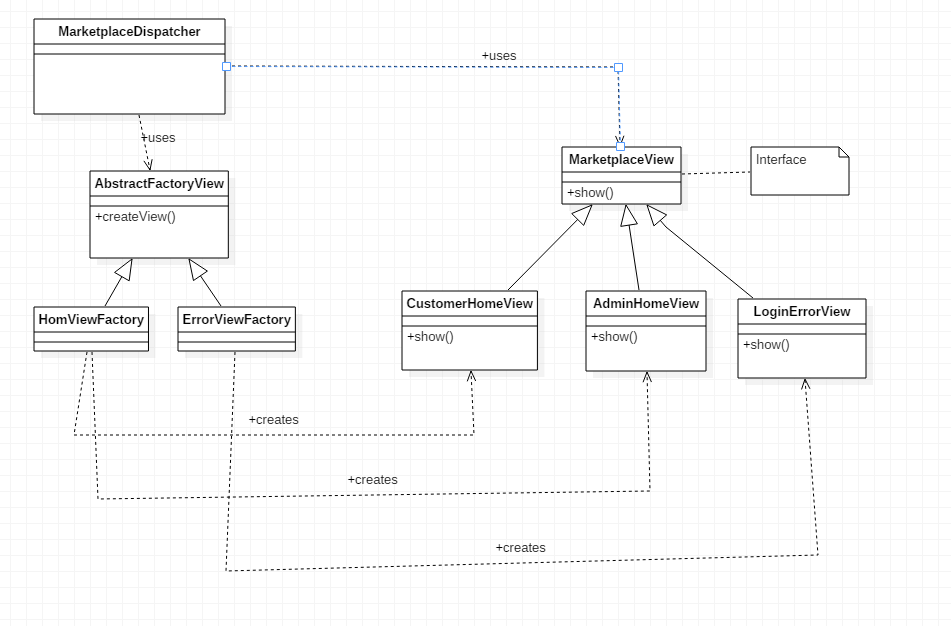


Fig 5: Abstract Factory

### Command Pattern

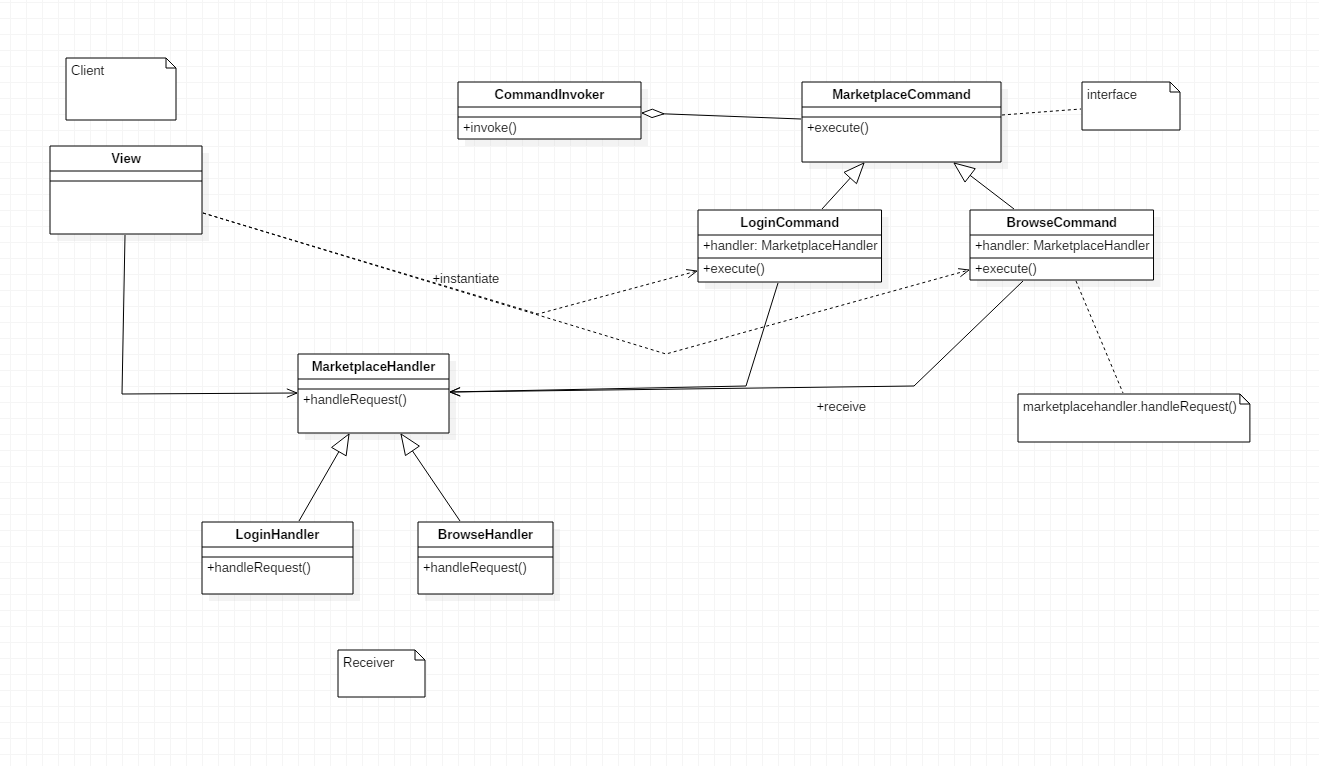
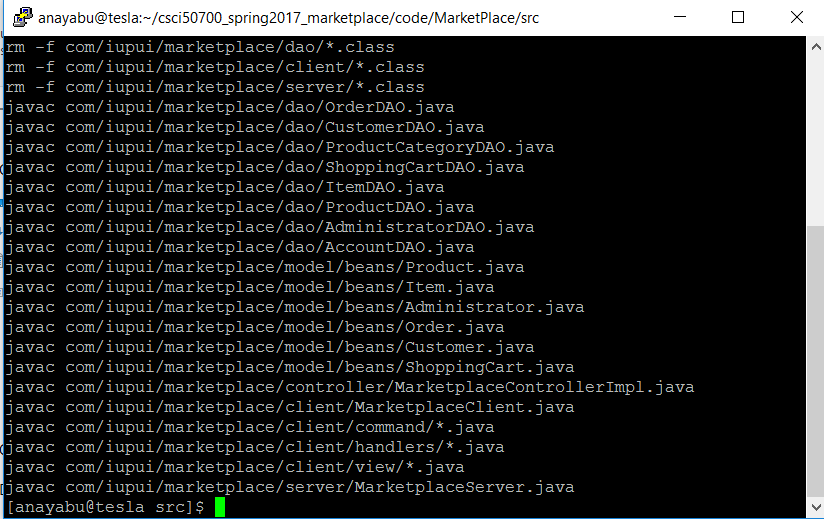


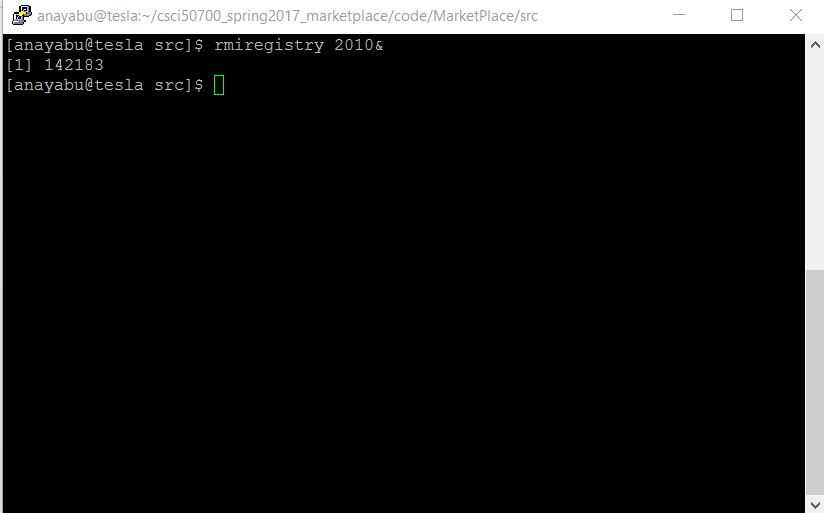
Fig 6: Command Pattern

# Sample runs:

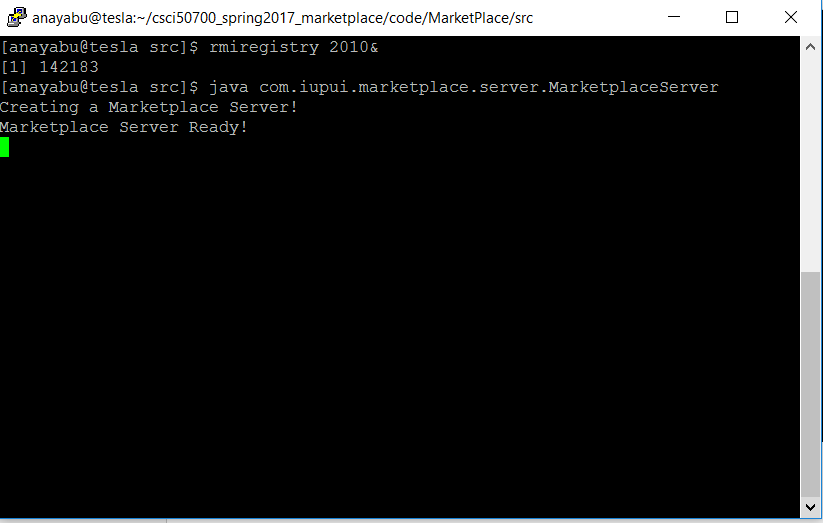
**Compilation**:



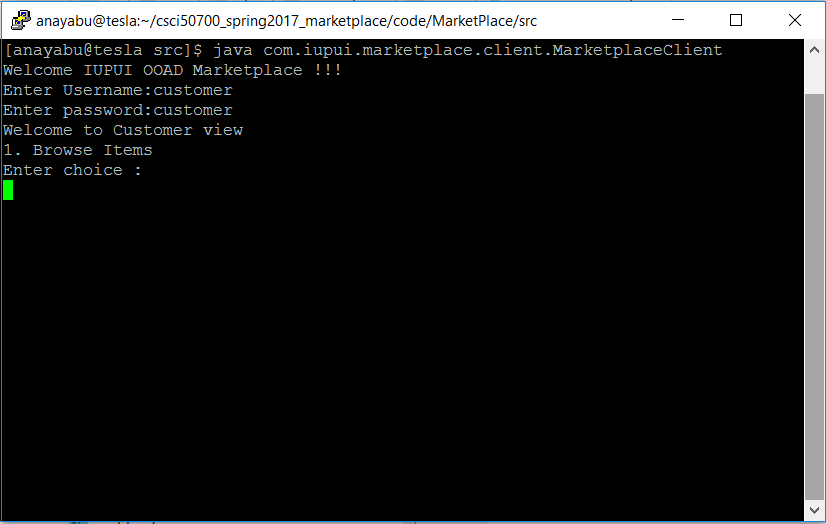
**RMI REGISTRY:**



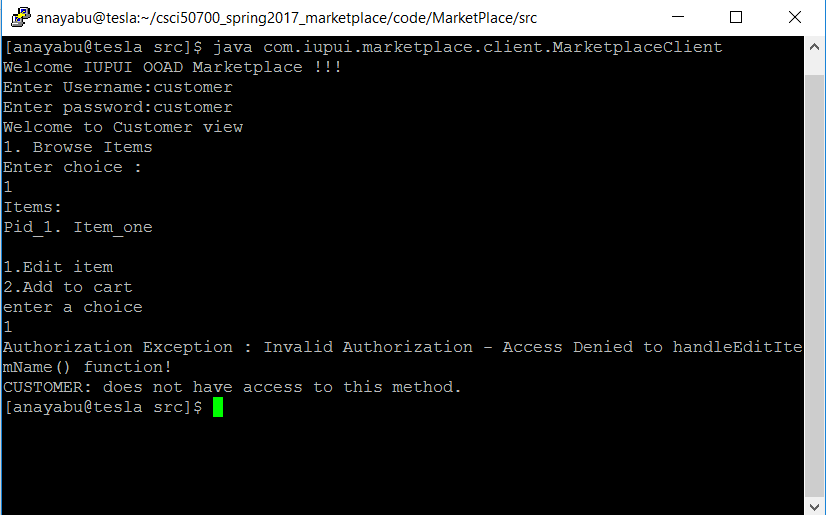
**Server:**



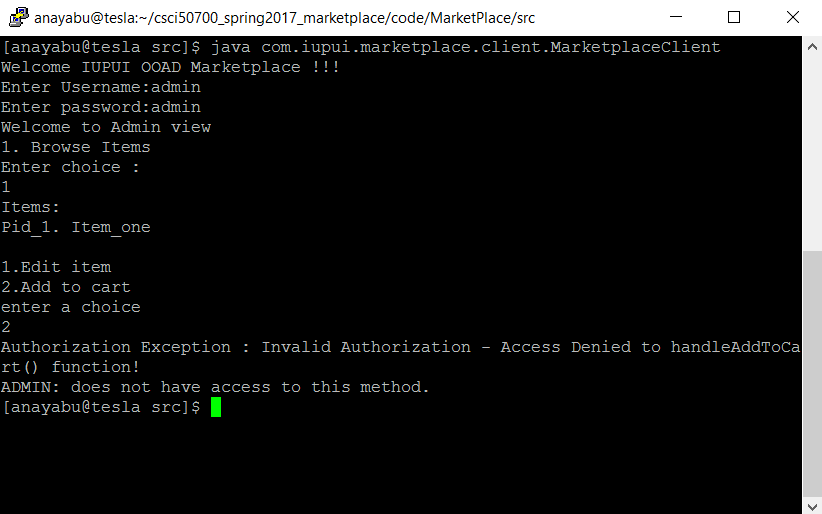
**Client:**



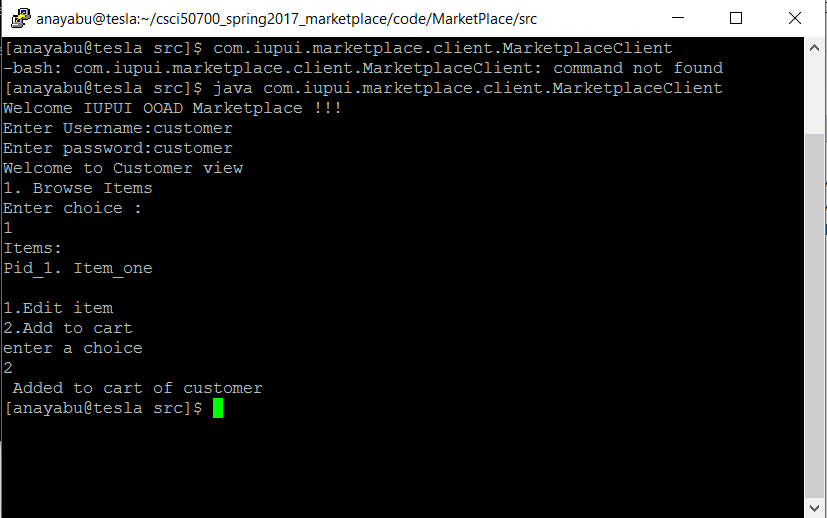
**When customer tries to edit an Item:**



**When admin tries to add item to cart:**



**Customer can add items to cart:**



**Admin can edit items:**

