Market Place

Assignment Report 2

Virinchi Sainath Nalluri
vnalluri@iupui.edu

13th February, 2018

Table of Contents

Objective

The objective of this assignment is to implement Front-Controller Pattern, Command Pattern and Abstract Factory pattern. Also to implement the login functionality to the application for administrator and customer providing different views for both of them.

Assignment Discussion

Role of **Front Controller**

The front controller is the class which accepts all the requests from the client. Every communication with the server will pass through this controller. It is in this controller too that the lookup is done. As soon as the customer/administrator enters his user credentials, the details are passed on to the front controller for validation by the view. The front controller directs it to the application controller, for user validation. After validation, depending upon the validation, this login handler sends it to the dispatcher from processing the appropriate view.

In our project, front controller is the FrontController.java, application controller is ControllerImpl.java and the dispatcher is DispatcherImpl.java.

Role of **Command Pattern**

In Command Pattern, a command is designed in a way, that it tells the receiver to work on an Action(). And this command on executed using the execute() method, will ask the receiver to perform the action. The command's execute() method is said to invoke this method. The invoker will have a list of commands, and it will start executing them one by one. However, in this project, there is only one command that is the ValidateCustomerLogin or ValidateAdminLogin command. Executing the commands as a list functionality is provided for the future.

There is an interface Command.java which has an execute() method. The concrete command classes, ValidateCustomerLogin.java and ValidateAdminLogin.java implements the Command Interface. In order to create ValidateCustomerLogin and ValidateAdminLogin, their receivers have to be passed. The receiver will be stored in their instance variables. When the invoker starts executing the execute command of these objects, the execute() has code to make the receiver perform some action. In the project Invoker.class is the invoker.

The receiver in the project is CustomerService object and AdminService object. CustomerService.java and AdminService.java, will be having all the methods for Customer and

Administrator respectively. However, in this project, they have only one method, i.e., validateLoginDetails(). This is the action() for the receiver. Once the commands with their receivers are created, the invoker is passed this command and it sets this command. So, when invoker object call the invoker, it simply starts executing the execute() of these commands.

Role of **Abstract Factory**

In the Abstract Factory Pattern, we can create a group of factories which can be used to create related objects. In this project, the abstract factory is used to create views. The FactoryCreator.java has the code to create an object of AbstractFactory. The AbstactFactory is an interface which is implemented by concrete classes, HomeViewFactory.java and ErrorViewFactory.java. The HomeViewFactory.java depending upon the requirements, creates CustomerHomeView object or AdminHomeView object. The HomeViewFactory understands the requirement based on the parameter sent “viewDescription” while requesting a view. Similarly, ErrorViewFactory.java makes LoginErrorView object or PageNotFoundView object. These view have show method which have their display messages. As the appropriate view is received, the show method is executed.

UML Class Diagram

The UML diagram for the entire project is to big in one picture. So, three different diagrams for each pattern is sketched.

Firstly, the Command Pattern,

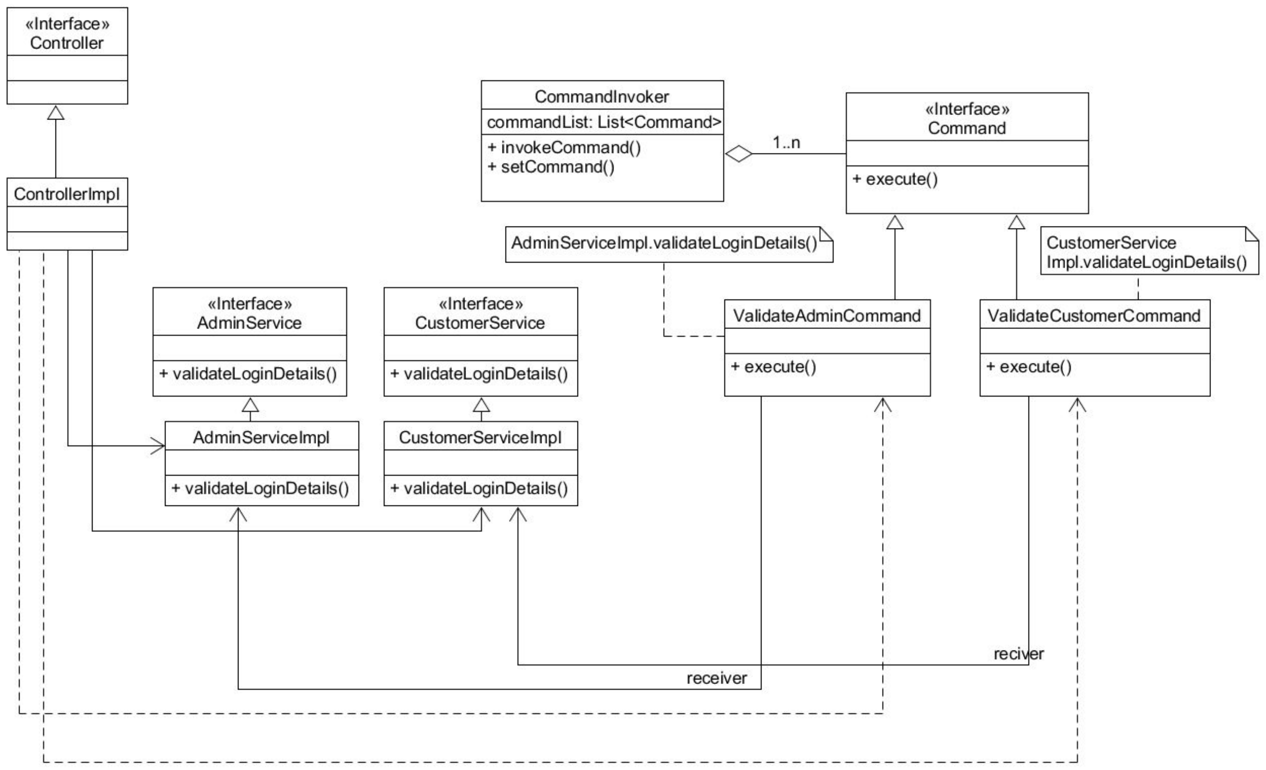


Fig1 Command Pattern

In the controller class, a command is created using the receiver (AdminServiceImpl/CustomerServiceImpl). The reciver has the implemenation code for validating. The command's execute methods has the code AdminServiceImpl.validateLoginDetails() or CustomerServiceImpl.validateLoginDetails(). The invoker is set this command and when on invoking the invoker, is validation is done.

Secondly, Abstract Pattern

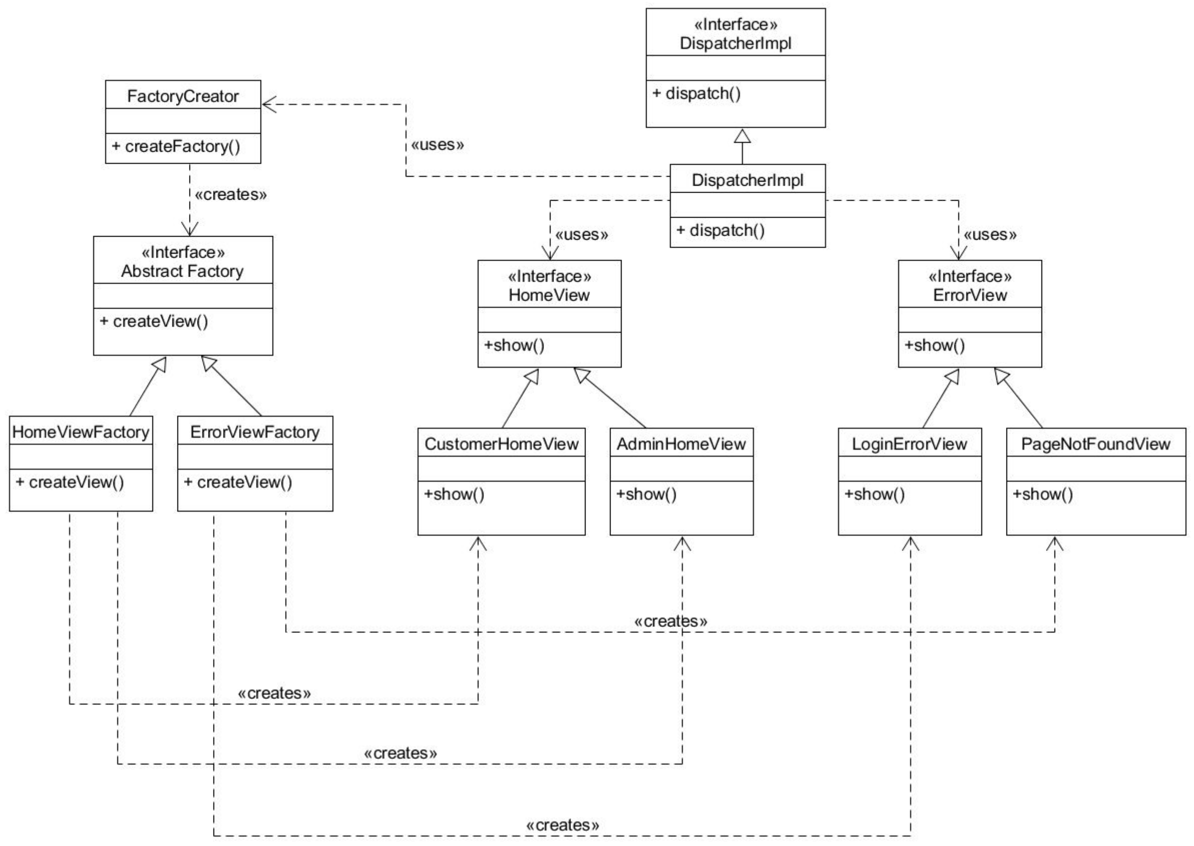


Fig2 Abstract Factory Pattern

The Factory Creator creates the Abstract Factory. Which abstract factory is created by the factory creator is dependent on the parameter passed while creating. HomeViewFactory is used to create HomeViews such as CustomerHomeView or AdminHomeView. Similarly, ErrorViewFactory is used to creae ErrorViews such as LoginErrorView and PageNotFoundView. Which ever view is called, show method is implmented on for that created View which is displayed on the client side.

Thirdly, Front Controller,

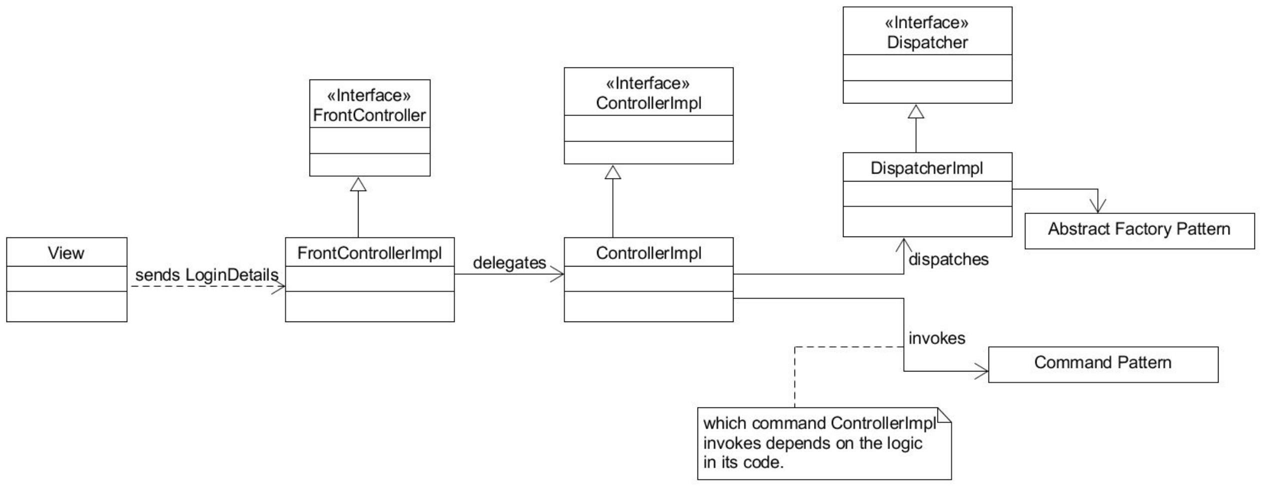


Fig3 Front Controller

The view sends LoginDetails to the FrontController which decides whether it is a customer or admin. The application controller does the validation and sends the job of creating appropiate view to dispatcher, which creates the views accordingly. The Dispatcher uses the Abstract Factory Pattern for creating of factories and the application controller depends upon the Command pattern for validation.

Transition From Assignment 1 to Assignment 2

New Classes Created

Related to Command Pattern: Command.java, ValidateCustomerDetails.java, ValidateAdminDetials.java, CommandInvoker.java, CustomerService.java, CustomerServiceImpl.java, AdminService.java, AdminServiceImpl.java and CommandInvoker.java.

Related to Abstract Factory Pattern: FactoryCreator.java, AbstractFactory.java, HomeViewFactory.java, ErrorViewFactory.java, HomeView.java, ErrorView.java, CustomerHomeView.java, AdminHomeView.java, LoginErroView.java, PageNotFountView.java, ViewInterface.java and LoginView.java

Related to FrontController Pattern: FrontContoller.java, FrontControllerImpl.java, Dispatcher.java, DispatcherImpl.java.

Where each class fits into the architecture has been described using UML class diagram.

Comments from Assignment 1

The data related to Controller component has been placed in View. The data has been placed in its appropriate position, currently in the FrontController.java.

Also there was comment, regarding the domain model containing reference types, it has been modified and also an extra class, LoginDetails has been added in the current Assignment.

A new class LoginDetails is added. This class is used to carry details username, password, userType and if these details have been authenticated. This object will be passed from one class to others.

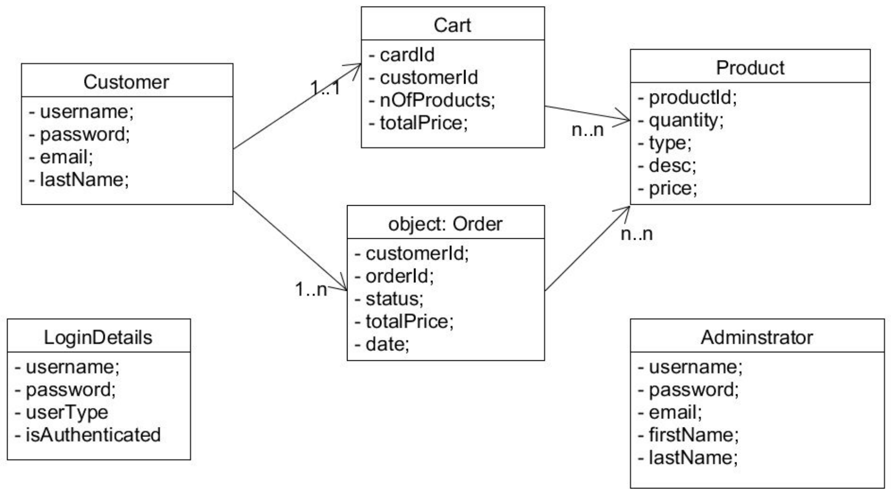
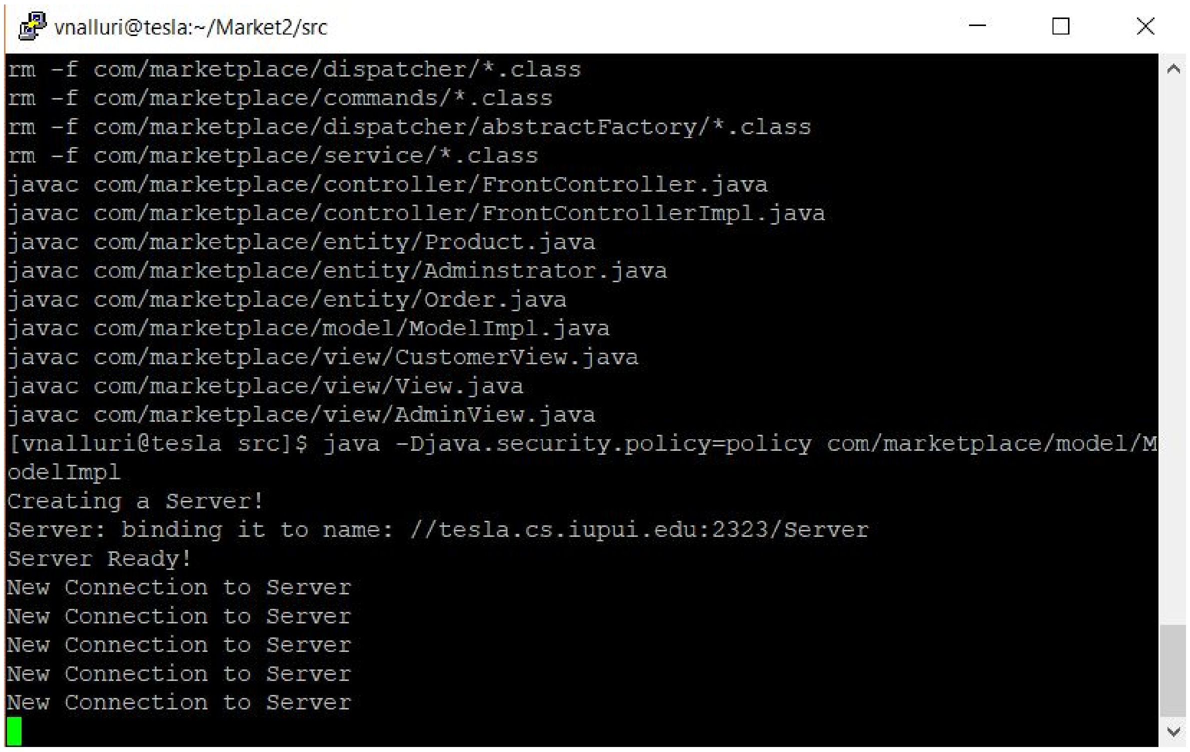


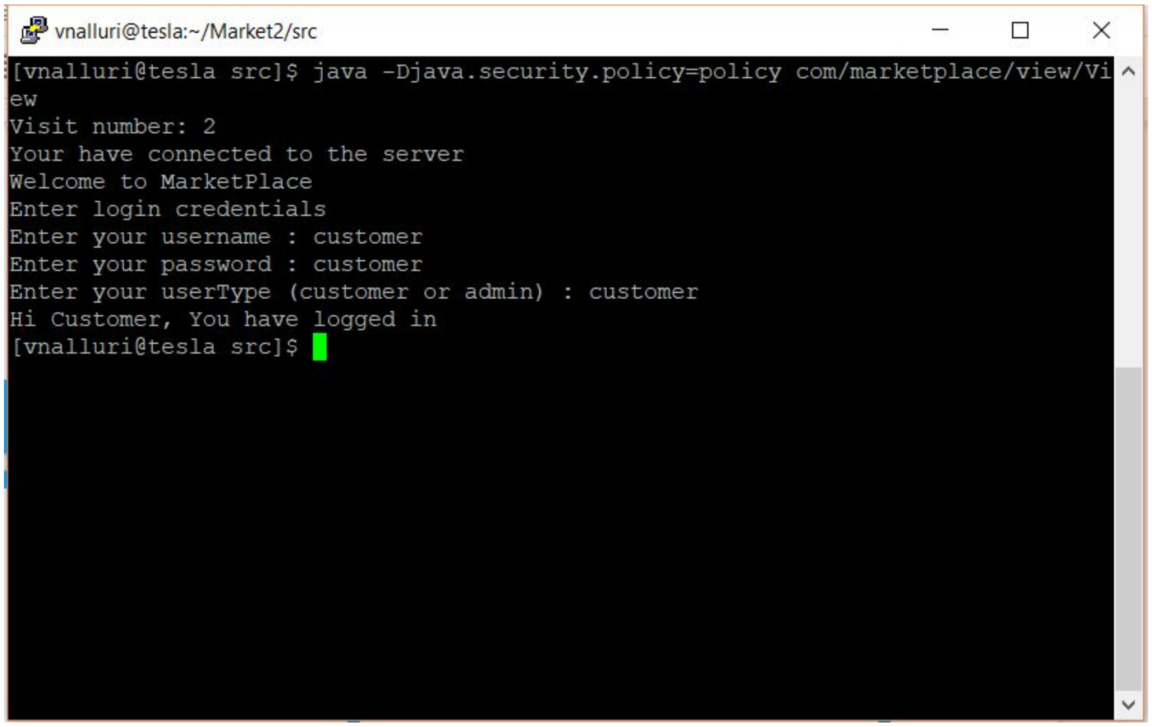
Fig4 Domain Model

Sample Runs



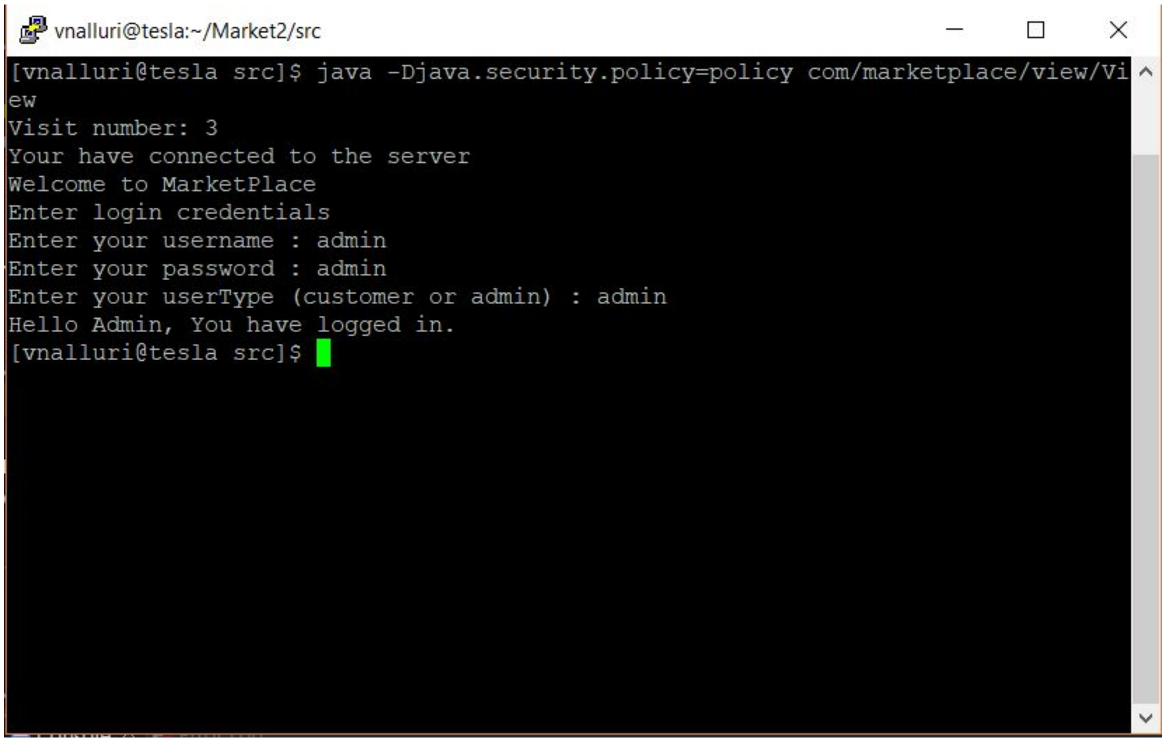
```
rm -f com/marketplace/dispatcher/*.class
rm -f com/marketplace/commands/*.class
rm -f com/marketplace/dispatcher/abstractFactory/*.class
rm -f com/marketplace/service/*.class
javac com/marketplace/controller/FrontController.java
javac com/marketplace/controller/FrontControllerImpl.java
javac com/marketplace/entity/Product.java
javac com/marketplace/entity/Adminstrator.java
javac com/marketplace/entity/Order.java
javac com/marketplace/model/ModelImpl.java
javac com/marketplace/view/CustomerView.java
javac com/marketplace/view/View.java
javac com/marketplace/view/AdminView.java
[vnalluri@tesla src]$ java -Djava.security.policy=policy com/marketplace/model/M
odelImpl
Creating a Server!
Server: binding it to name: //tesla.cs.iupui.edu:2323/Server
Server Ready!
New Connection to Server
New Connection to Server
New Connection to Server
New Connection to Server
New Connection to Server
```

Fig 5 On the server side



```
[vnalluri@tesla src]$ java -Djava.security.policy=policy com/marketplace/view/Vi
ew
Visit number: 2
Your have connected to the server
Welcome to MarketPlace
Enter login credentials
Enter your username : customer
Enter your password : customer
Enter your userType (customer or admin) : customer
Hi Customer, You have logged in
[vnalluri@tesla src]$
```

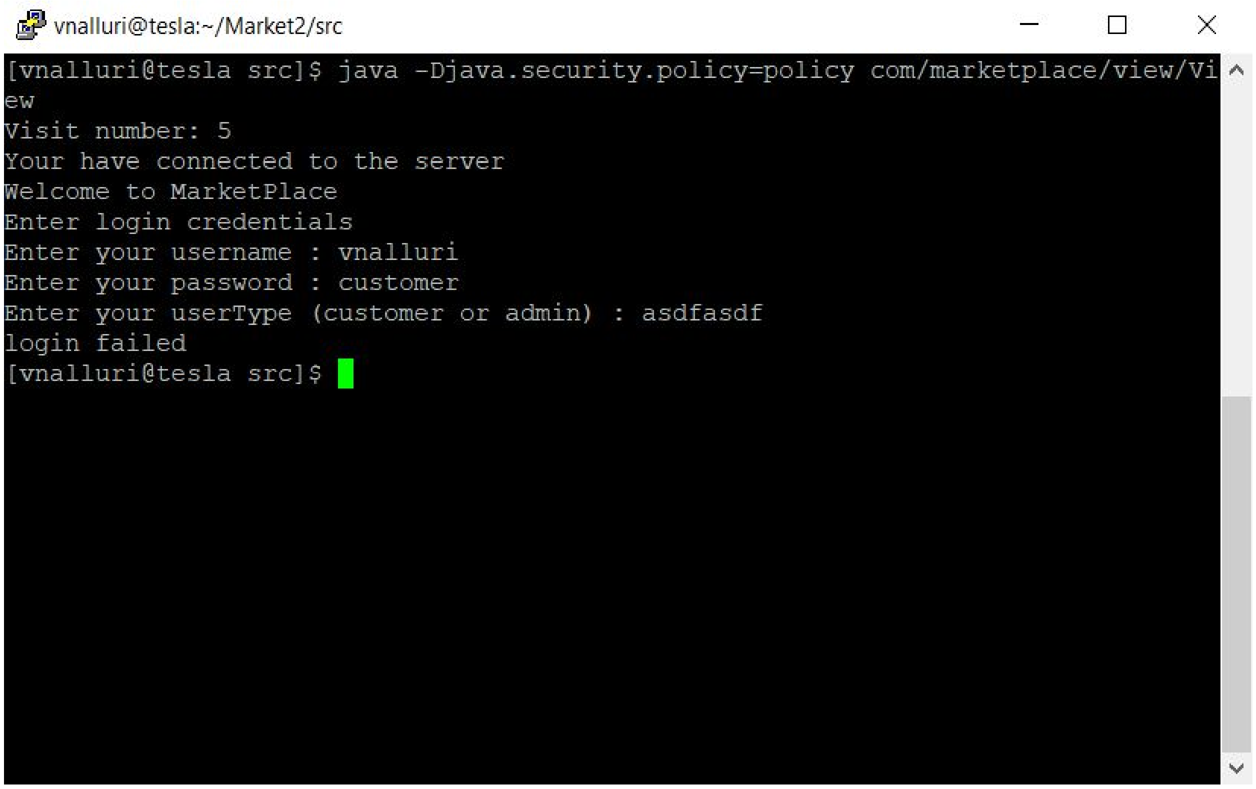
Fig 6 On the client side, when customer enters his correct user credentials.



vnalluri@tesla:~/Market2/src

```
[vnalluri@tesla src]$ java -Djava.security.policy=policy com/marketplace/view/Vi
ew
Visit number: 3
Your have connected to the server
Welcome to MarketPlace
Enter login credentials
Enter your username : admin
Enter your password : admin
Enter your userType (customer or admin) : admin
Hello Admin, You have logged in.
[vnalluri@tesla src]$
```

Fig 7 On the client side, when admin enters correct user credentials.



vnalluri@tesla:~/Market2/src

```
[vnalluri@tesla src]$ java -Djava.security.policy=policy com/marketplace/view/Vi
ew
Visit number: 5
Your have connected to the server
Welcome to MarketPlace
Enter login credentials
Enter your username : vnalluri
Enter your password : customer
Enter your userType (customer or admin) : asdfasdf
login failed
[vnalluri@tesla src]$
```

Fig 8 On the client side, when user enters wrong credentials.

Conclusion And Analysis

With this assignment, using design patterns such as command pattern, abstract factory pattern and front-controller pattern, implementation has been for login of the customer and administrator. Both customer and administrator have different views.

References

1. For writing make file : https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
2. Java RMI : Bank example in Canvas
3. MVC Pattern : Calculator Example in Canvas
4. Front Controller Pattern : Front Pattern Example in Canvas
5. Abstract Factory Pattern : Abstract Pattern Bank Example in slides.
6. Command Pattern: Command Pattern example in Canvas.
7. Technologies Used : Eclipse IDE, WinScp, PUTTY, Tesla, Java