CSCI 50700

*Marketplace Application*

*Assignment-6*

Siva Pinnamaneni,

2000087624

**Introduction**

The goal of this project is to develop a software application for Marketplace which in a way is similar to Amazon. This initial stage of the project would include developing the skeleton framework on which we develop the whole project. This assignment focuses on developing the application including command pattern, front control pattern and Abstract Factory Pattern.

**Marketplace**

This project focuses on developing an online Marketplace according to the demands of the customer. This Marketplace just like Amazon should cater to the users desire to browse around the plethora of products found and buy these products. The application system should present a view to the users through which the users can browse the products. This view for the customers is provided in the Client.java in our system. There will be Administrator who is responsible for managing the whole system. This Administrator is also responsible for adding or removing products and other people as Admins to the system. The system consists the following functions which handle the needs of both the users as well as the Admins:

• Registration(Customer)

• Login(Administrator/Customer)

• Browsing Items(Administrator/Customer)

• Updating Items(Administrator)

• Removing Items(Administrator)

• Adding Items(Administrator)

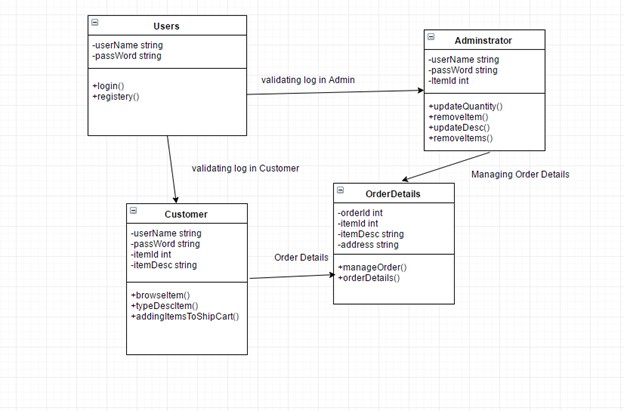
• Purchasing Items(Administrator)

**Remote Method Invocation (RMI):**

Java RMI is a true distributed computer application interface for java. Java RMI is based on Remote Procedure Calls(RPC). Java RMI provides us the option of accessing remote objects unlike RPC which only provides us the option of accessing remote procedures. To connect our system and gives us a true feel of an online shopping scenario we use Java RMI. With RMI we can connect as many users as we want at the same time to access the system. To use Java RMI for our system we utilize remote interface classes.

**Software Patterns**

Domain Model:



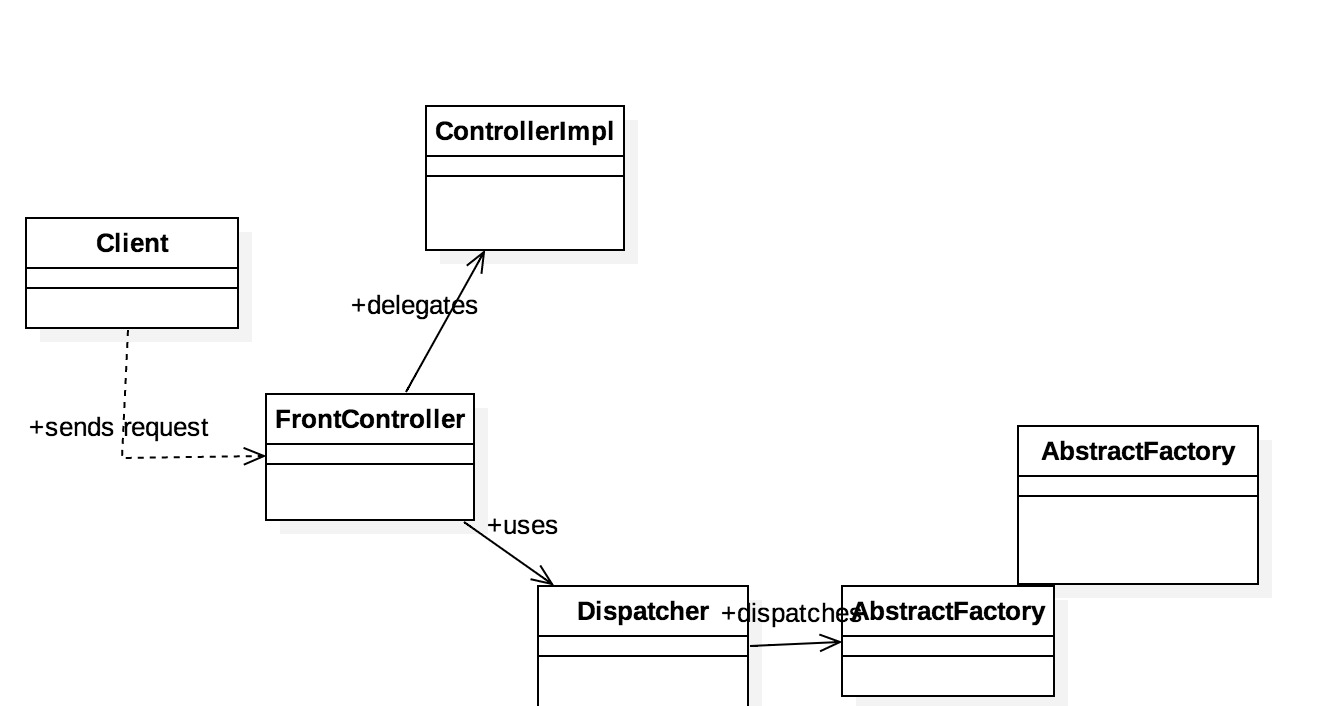
**Model-View-Controller (MVC):**

The model view controller is a software architectural pattern used to develop user interfaces. This architecture divides an application into three interconnected parts which is to separate internal representations of information from the ways information is presented to, and accepted from, the user.

In our system, we use the MVC pattern to segregate the user from the

Administrator. The view is provided in the Client program which is used to develop the GUI which the user will see and shop from. The server controller implementation is where the Admin makes any changes to the system. This class is the control part of the MVC. Any changes made by the Administrator will be to update the view. These changes will be visible to the user. The user can also suggest changes to which the system should adapt in order to make things easier for her. These changes will then be communicated to the Administrator who then manipulates the system which will be then visible to the user.

**Front Controller Pattern:**



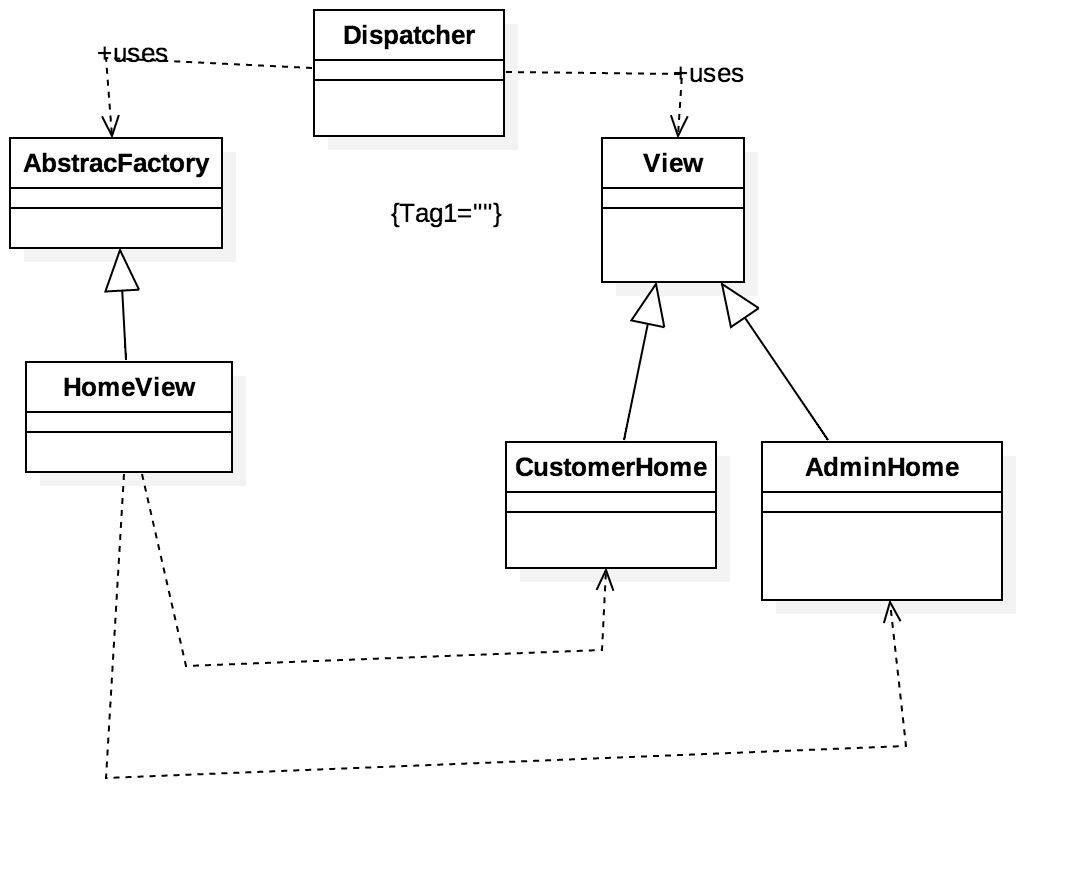
The front controller design pattern is used to provide a centralized request handling mechanism so that all requests will be handled by a single handler. This handler can do the authentication/ authorization/ logging or tracking of request and then pass the requests to corresponding handlers. Following are the entities of this type of design pattern.

* **Front Controller** - Single handler for all kinds of requests coming to the application (either web based/ desktop based).
* **Dispatcher** - Front Controller may use a dispatcher object which can dispatch the request to corresponding specific handler.
* **View** - Views are the object for which the requests are made.

**Command Pattern:**

Command pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

**Abstract Factory Pattern:**

****

Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

**Java Annotation:**

In the Java, an annotation is a form of syntactic metadata that can be added to java source code. Classes, methods, variables, parameters and packages may be annotated. Unlike java doc tags, Java annotations can be reflective in that they can be embedded in class files generated by the compiler and may be retained by the Java VM to be made retrievable at run-time. It is possible to create meta-annotations out of the existing ones in Java.

**Proxy Pattern:**

In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern. In proxy pattern, we create object having original object to interface its functionality to outer world.

**Java Reflection:**

Sometimes it is necessary to issue requests to objects without knowing anything about the operation being requested or the receiver of the request. "The Command design pattern suggests encapsulating ("wrapping") in an object all (or some) of the following: an object, a method name, and some arguments. Java does not support "pointers to methods", but its reflection capability will do nicely. The "command" is a black box to the "client".

**Role Based Access Control:**

Role-based-access-control (RBAC) is a policy neutral access control mechanism defined around roles and privileges. The components of RBAC such as role-permissions, user-role and role-role relationships make it simple to perform user assignments. A study by NIST has demonstrated that RBAC addresses many needs of commercial and government organizations. RBAC can be used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions. Although RBAC is different from MAC and DAC access control frameworks, it can enforce these policies without any complication. Its popularity is evident from the fact that many products and businesses are using it directly or indirectly.

**RMI Concurrency:**

“ *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."*

RMI provides us with a basic Multi-threaded system. When a client connects to the server the connection can be a single thread or a multi thread. According to the text provided it is difficult to predict whether the connection will be single or multi-threaded. Irrespective of that RMI does provide concurrency. Within a single VM instance, invocations on an RMI Stub (client to an RMI server) will be internally synchronized. That is, the stub itself will prevent multiple threads from concurrently invoking a method on the remote server.

**SYNCHRONIZATION:**

By synchronization we mean providing access to multiple clients concurrently. This synchronization can be achieved by using the keyword synchronized provided by java, thread safe interface and guarded suspension.

**Singleton Pattern:**

It is a creational pattern which is used when an application needs one and only one instance of an object. The solution to the pattern is to ensure that the class has only one instance and a global point of access will be provided to it.

**Overview of work-flow**:

In our program the Client class takes the Username, password and usertype as input from the user. These details will be passed to the frontcontroller which has the validate login functionality. This request will be sent to the controllerImpl class which then validates the user and sends this back to the frontcontroller. The frontcontoller then sends it to the Dispatcher. Upon receiving the request from Frontcontroller the dispatcher checks the associated role for user session object and forwards the application onto the respective view. Each view is distinct in it’s own way. The two views we generate in our code are User and Admin. If it’s a user , they cannot access the option of Add Product and if it is an Admin member they cannot access the Buy option. Now there is an implementation of the functions Browse, Buy and Add products. These functions are again further implemented and additional features such as cart and synchronization are also implemented. For the Database Access layer implementation singleton pattern was used. By creating a class with single object and making it global.

**Feedback from Assignment1:**

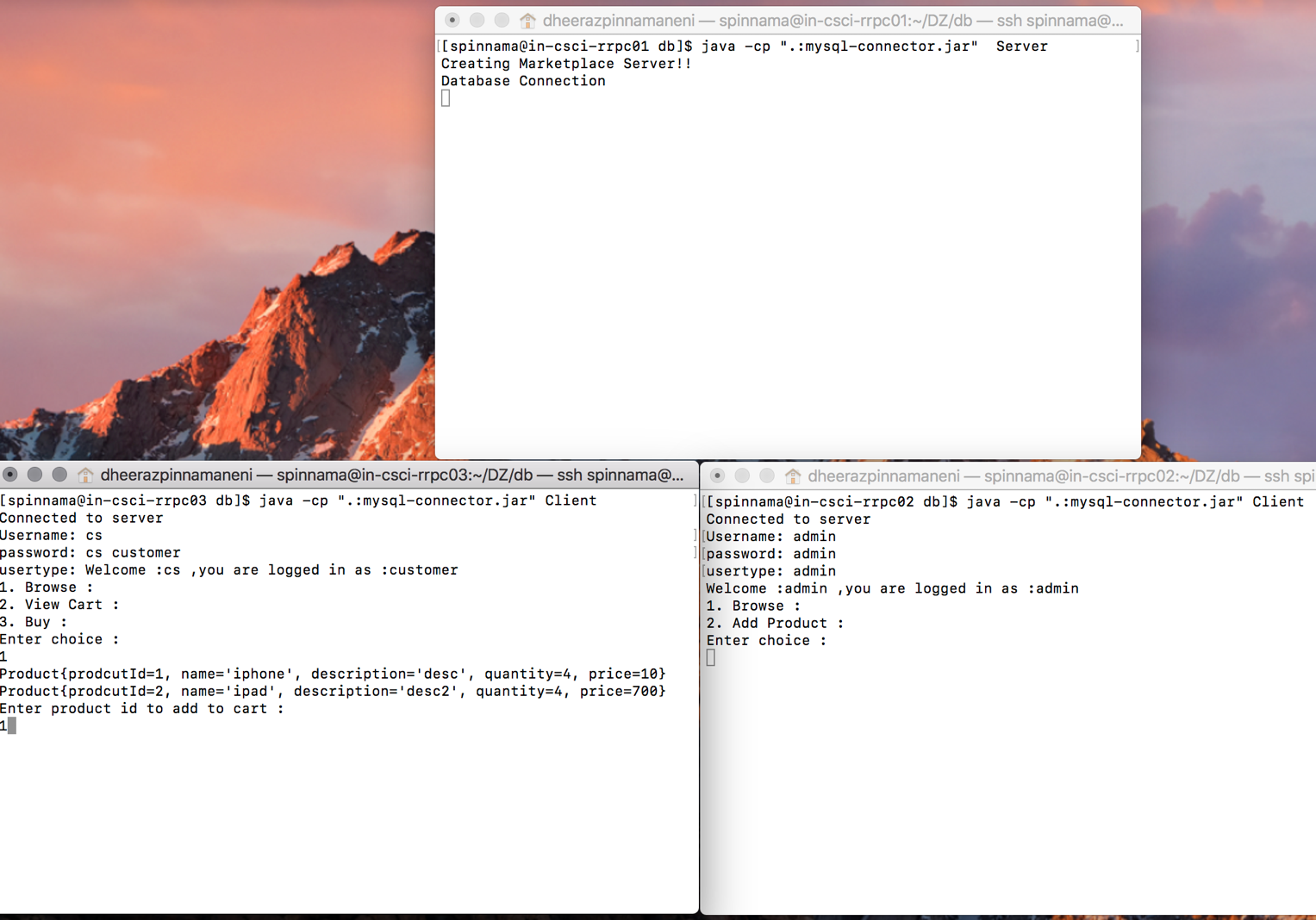
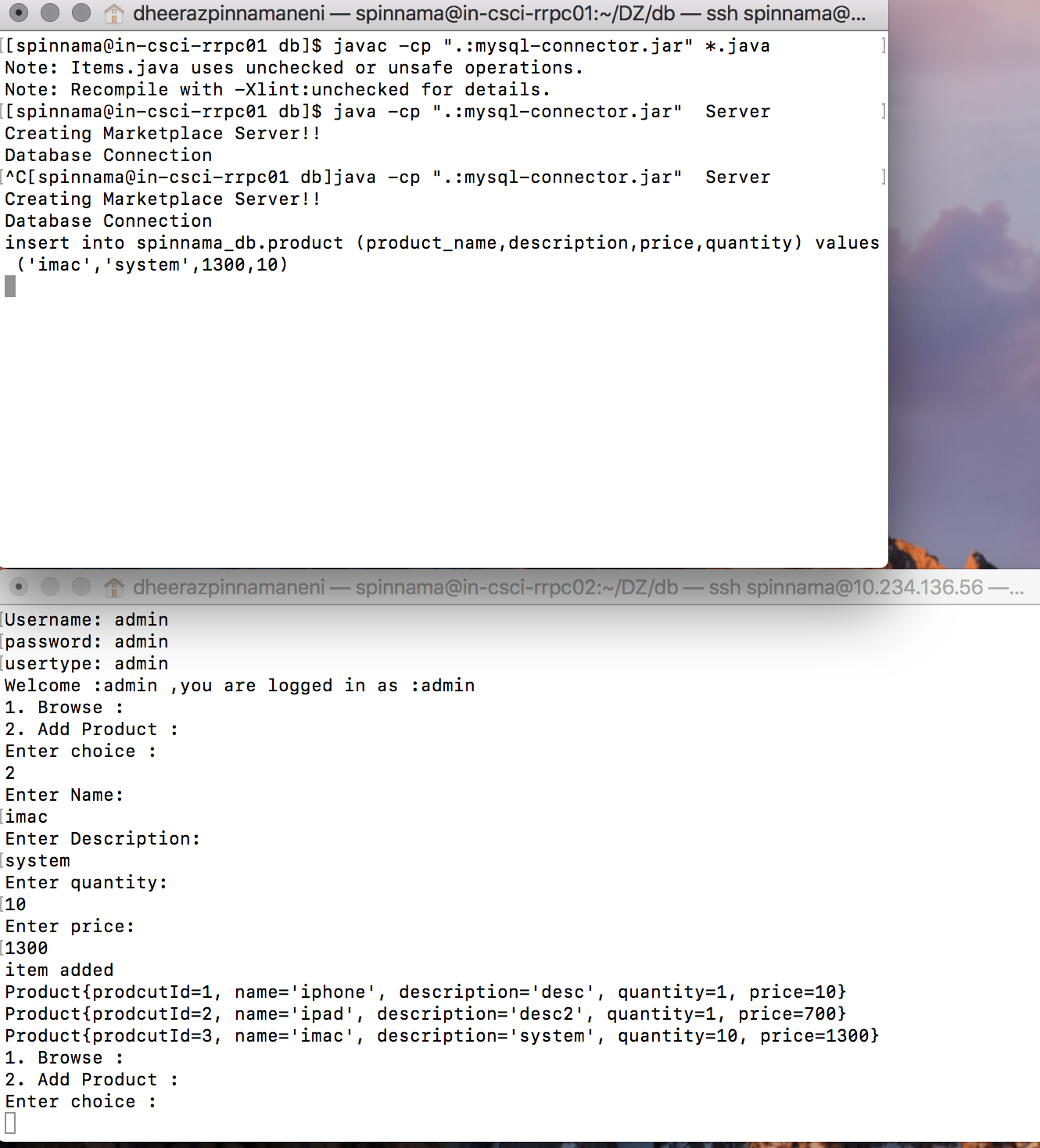
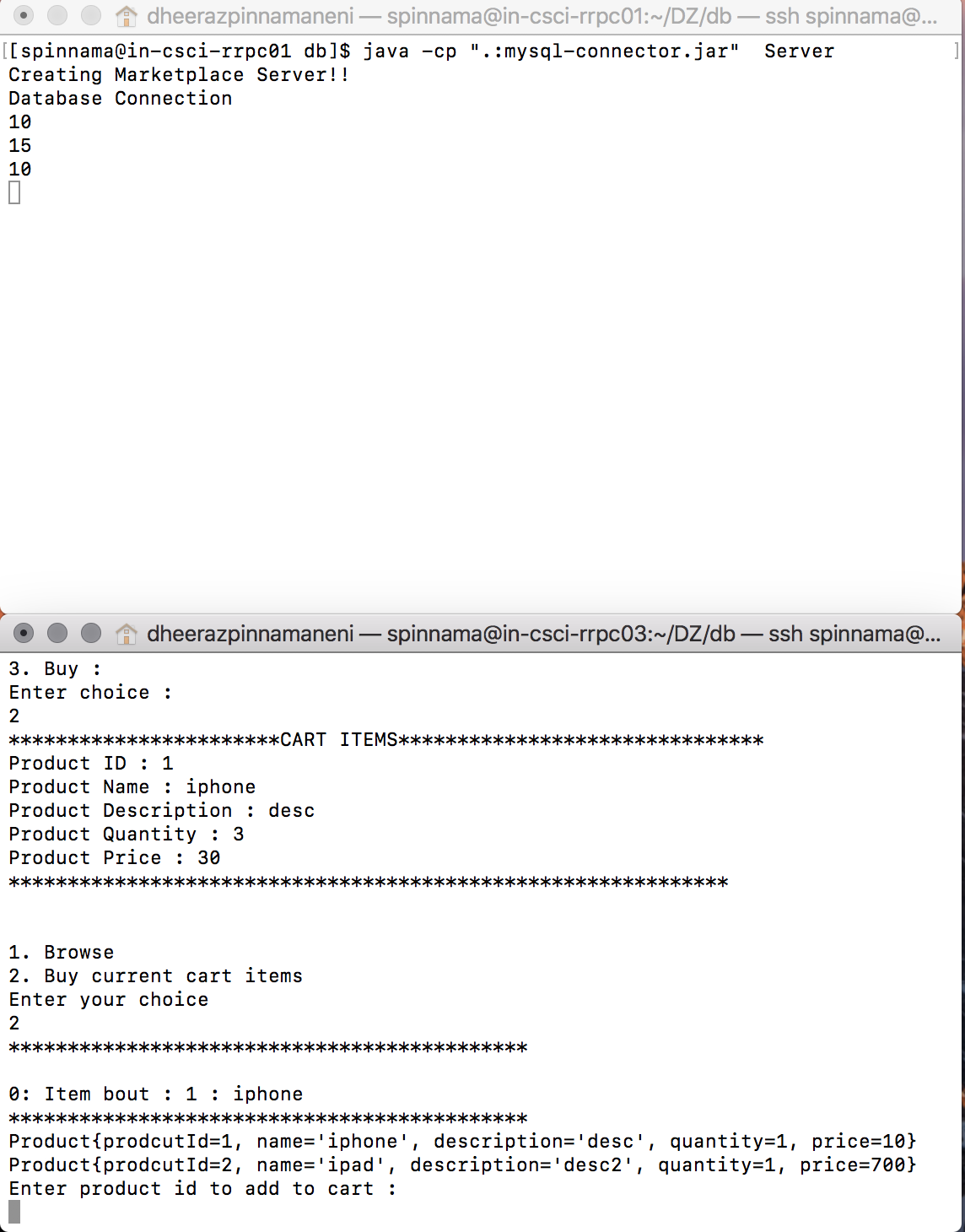
**Assignment 1:** Included the report discussing the requirements.

**Assignment 2:** Improved the Read-Me file to better help executing the program. Included the Honor Pledge. Included the screenshots (sample runs) inside the report rather than separately on GitHub.

**Assignment 3:** Changed the title of the Report and assigned the appropriate title.

**Assignment 4:** Included the honor pledge in all the files. Included comments where necessary. Included the screenshots which show the implementation of concurrency.

**Screenshots:**



**Project Feedback:**

What did you like about the assignment:

Patterns! By implementing those patterns we provide abstraction. We hide the details from the user which if visible would not make for a good application. Debugging becomes easy too, it was of great help that I could find the error without changing the code as a whole.

What did you dislike about the assignment?

Learning and implementing the patterns took time.

What would you change about your design if you could go back? Why?

I would definitely try implementing the application using a GUI as I have a better idea of patterns now.