ONLINE MARKETPLACE APPLICATION

ASSIGNMENT-4

PROJECT REPORT

IMPLEMENTING ADD, BROWSE, PURCHASE FUNCTIONALITY AND CONCURRENCY SIMULATION IN JAVA RMI

UNDER GUIDANCE OF

Dr. RYAN RYBARCZYK

BY
ADITHYA MORAMPUDI

Table of Contents

ASSIGNMENT 4 REPORT

Assignment#3 Feedback

- Added the comments and honor pledge in the files that were missing it.

Introduction

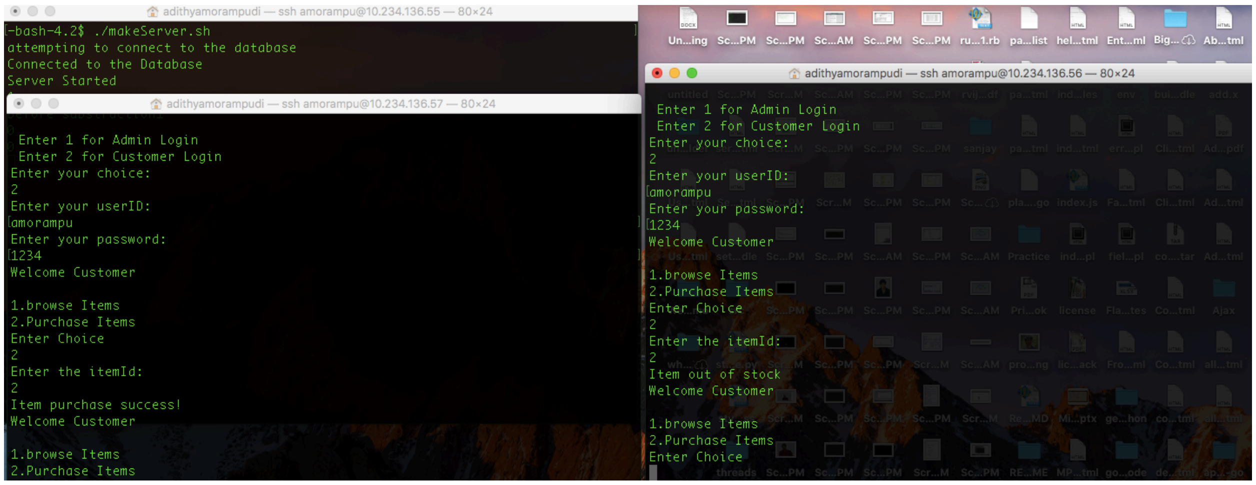
In this Assignment, we are asked to fully implement the Add items, Browse Items, Purchase Items functionality, and we are asked to simulate the java RMI concurrency feature using all the 6 network machines provided to us, for this Assignment, I have changed my server machine to be 10.234.136.55, which was previously 10.234.136.57, the reason behind changing the server machine is that, I have added database functionality to this assignment.

Application Flow

On top of the previously built application using Java RMI which was the implementation of login process and RBAC using Java Annotations to protect the application from unauthorized access, in this assignment, I have improved the functionality or features of the application by implementing concurrency, to handle multiple clients by using the keyword 'synchronized', I made the application thread safe as java RMI handles multiple clients or it is multithreaded, the keyword synchronized in Java helps us to make the functions thread safe, by providing sequential access to the method. Coming to the application flow, whenever a user or client tries to browse the items, the request is sent to the server and then the server controller, pulls the information from the database and then it sends this information to the client via Remote call. This browse functionality is common to both Admin and client. The admin can also add the products to the database using the add items method, whenever the admin adds the product it first goes to the server controller and the controller calls the DbConnection class which is responsible for making the interactions between the database and the server using JDBC driver. The client can also purchase the items, this also goes through the same procedure, once the client wants to purchase an item, he has to select the item id and then send this item id to the server, the server checks if this item is available by pulling the item stock from the database, the purchase feature can be completed only if the available stock is greater than 0, once the purchase is made, an update is made to the stock available by reducing it by 1, for now the feature of purchase is limited to only one item per transaction, so the stock is reduced only by 1 each time the purchase item method is executed. I have added a new class in this assignment, which is used to make interactions with the MySQL database, for now I have a method which is used to establish the connection between the database and the server once the server is started, only if the connection is established we can make interactions with the database, if not we cannot. In this assignment, I am using the auto commit feature of JDBC, if required in further assignment or if the application demands, then I will use transactions, which can be more helpful to make the database and the transactions more robust. Let's get into the software design without much delay. last but not the least, no software designs were implemented in this assignment, apart from the ones already existing in the application.

Java RMI Concurrency Discussion

The main purpose of this assignment is to examine Thread usage in Remote Method Invocation within the scope of our application, for this purpose, I have implemented the functions to be thread safe by using the keyword synchronized, with the help of this keyword, the access to the function is synchronized, meaning that no two clients can access the same function at the same time, let's first identify the concurrency provided by java RMI, now I am running two different clients to connect to the same server, the server is running on 10.234.136.55 and the two clients are running on 10.234.136.57 and 10.234.136.56, we can add up to 6 clients since we have 6 machines in our network these can be handled at the same time by the server. Let us look at a screen shot of the different clients running at the same time


adithyamorampudi — ssh amorampu@10.234.136.55 — 80×24
adithyamorampudi — ssh amorampu@10.234.136.57 — 80×24
adithyamorampudi — ssh amorampu@10.234.136.56 — 80×24

```
-bash-4.2$ ./makeServer.sh
attempting to connect to the database
Connected to the Database
Server Started
```

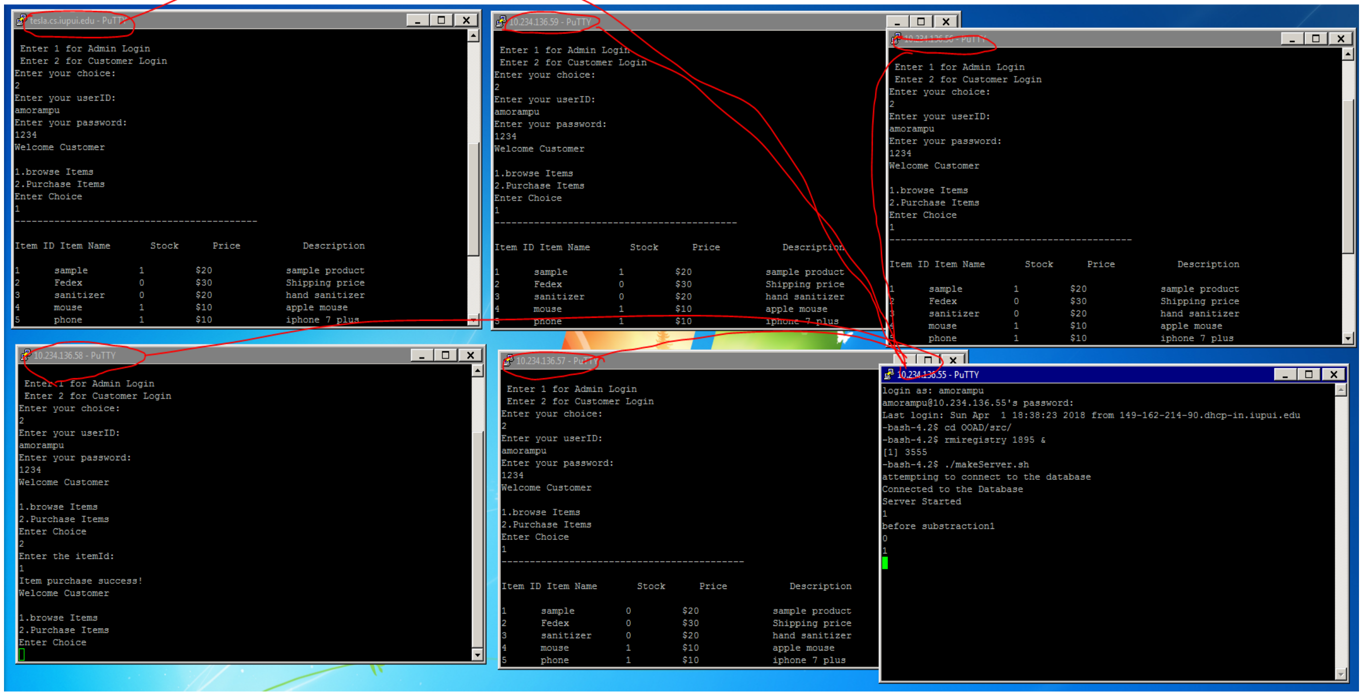
```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
2
Enter the itemId:
2
Item purchase success!
Welcome Customer

1.browse Items
2.Purchase Items
```

```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
2
Enter the itemId:
2
Item out of stock
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
```

in the above figure, two different clients are connected to the same server and accessing the server at the same time, they both are trying to access the purchase items function at the same time and only one client can make the purchase as the item goes out of stock for the other client and the error message is displayed to him saying the item is out of stock.

This proves that java RMI is indeed multiThreaded, but to be much clear of this notion, I have started all the 5 clients to connect to the server, and then, I was able to connect all the 5 clients to the server without any problem and everything worked fine, to demonstrate the use of synchronized keyword, I ran the clients at the same time on the same method and tried to browse the items, it worked fine and similarly I did this for all the other methods and everything worked fine, I also removed the synchronized keyword and tried to run the application, but I couldn't find any noticeable use of the synchronized keyword as the calling of functions by me might not have occurred at the exact fraction of time.

To test all the 5 clients at once, I ran the code on all the 6 machines, 10.234.136.55 being the server and the clients are all the 5 different machines, you can see in the below figure that I have successfully made the 5 clients communicate with the server without any problem, in the below figure each client is performing same and also different functions.



```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
1
-------------------------------------------

Item ID Item Name       Stock      Price           Description

1       sample         1          $20             sample product
2       Fedex          0          $30             Shipping price
3       sanitizer      0          $20             hand sanitizer
4       mouse          1          $10             apple mouse
5       phone          1          $10             iphone 7 plus
```

```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
1
-------------------------------------------

Item ID Item Name       Stock      Price           Description

1       sample         1          $20             sample product
2       Fedex          0          $30             Shipping price
3       sanitizer      0          $20             hand sanitizer
4       mouse          1          $10             apple mouse
5       phone          1          $10             iphone 7 plus
```

```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
1
-------------------------------------------

Item ID Item Name       Stock      Price           Description

1       sample         1          $20             sample product
2       Fedex          0          $30             Shipping price
3       sanitizer      0          $20             hand sanitizer
4       mouse          1          $10             apple mouse
5       phone          1          $10             iphone 7 plus
```

```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
2
Enter the itemId:
1
Item purchase success!
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
```

```
 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
1
-------------------------------------------

Item ID Item Name       Stock      Price           Description

1       sample         0          $20             sample product
2       Fedex          0          $30             Shipping price
3       sanitizer      0          $20             hand sanitizer
4       mouse          1          $10             apple mouse
5       phone          1          $10             iphone 7 plus
```

```
login as: amorampu
amorampu@10.234.136.55's password:
Last login: Sun Apr  1 18:38:23 2018 from 149-162-214-90.dhcp-in.iupui.edu
-bash-4.2$ cd OOAD/src/
-bash-4.2$ rmiregistry 1895 &
[1] 3555
-bash-4.2$ ./makeServer.sh
attempting to connect to the database
Connected to the Database
Server Started
1
before substraction1
0
1
```

Testing the application without synchronized keyword:

I have also tested the application by removing the synchronized keyword and tried to simulate the purchase item in two different clients, on the same product which has a stock of 1, the application was indeed working same, as when it had the synchronize keyword, I feel this is because the time when I am hitting the enter button to execute something is different in both the applications, if I were to successfully hit the enter button at exactly the same fraction of time, then my application would've allowed for both the clients to get the product since both of them are accessing the database at the same fraction of time and we would've potentially allowed both clients to get the hold of the product even though there was only one product.

I also observed that when I used synchronized keyword, within the methods which needs to be thread safe, I ran the application thrice and in three times I found three different results, which helped me found out that Java RMI is multithreaded but threads are not managed as to which thread has higher priority over the other, this behavior is evident from the fact that, each time I ran the application, a different client was able to get the product over other and sometimes the same client got the product.

So, from all the perspectives after running the application, I came to a conclusion that java RMI is multithreaded and allows concurrent access to resources, but the point is we need to be aware of the underlying concepts behind how to allow concurrent access to the application, we need to use the synchronized keyword by taking into consideration all the behaviors of deadlocks and synchronization and also the access to the critical section. With all these results, I can conclude that RMI is multithreaded and provides concurrent access to resources and it is our responsibility to take care of properly implementing the concurrent access.

Software Design

In this assignment, I made modifications only to the required methods, which are:

- Browse Item
- Add Items
- Purchase Items

In these 3 methods, the issues of concurrency have been addressed as we see in the above screenshots, all the three methods have been synchronized and the issues of deadlocks have been taken care of. Let's consider each of these one by one.

- Browse Items: This is method which is responsible for displaying the products present in the database, with the help of this method both the admin and customer can see the available products and there is no separate implementation for both admin and user, this is common to both. This method uses the DbConnection class to get the hold of database and then it can perform the necessary task. Coming to the coding implementation of this method, this method returns a List<String> each tuple in the database is retrieved and stored in the list as a string, once this information is returned to the client, the client then performs the String split operation and displays the information to the user

- Purchase Item: using this method the customer can purchase the product, by selecting its Id, once a user requests to buy a product, the application asks the user to enter the Id of the required product, which he can see by browsing the items, in the future implementation, I will add the feature to purchase the product while browsing, but for now, the user has to enter the Id of the product which he wants to buy, after this the purchase item method on the server side checks if the product is available, it does so by getting the stock of the item with the help of the getUniqueItem method in the DbConnection class, once the method retrieves the stock and checks that the stock is indeed greater than 0, this method is executed completely with the help of purchase method in the DbConnection class, which essentially reduces the stock by 1 and sends a status message of success to the user. If the stock is not available it sends and error message saying that the stock is not available. This method only provides access to the customer. Admin cannot access this method. The purchase item just takes a session object and a integer variable to access the server side method, the value is sent to the server and purchase is made if product is available.

- Add Items: This is the method which allows the admin to add products into the database, once the admin has authenticated to the system, he can call the add items method to add products into the database and Admin need not enter the Id of the product, since this field is auto populated by the database itself, in this assignment, I haven't implemented any transactions, I am using the auto commit feature of JDBC. This method returns a Boolean, when an item is added into the database then method returns true, if not then it returns false, also the information from the client is sent to the server using a String Array, this way the entire tuple can be sent at once, the server will use the index of the array to retrieve the necessary information from the array and send this to the database and returns a Boolean value.

Class Diagram

class diagram, also there is a separate Image in the source directory(ClassDiagram.jpg)

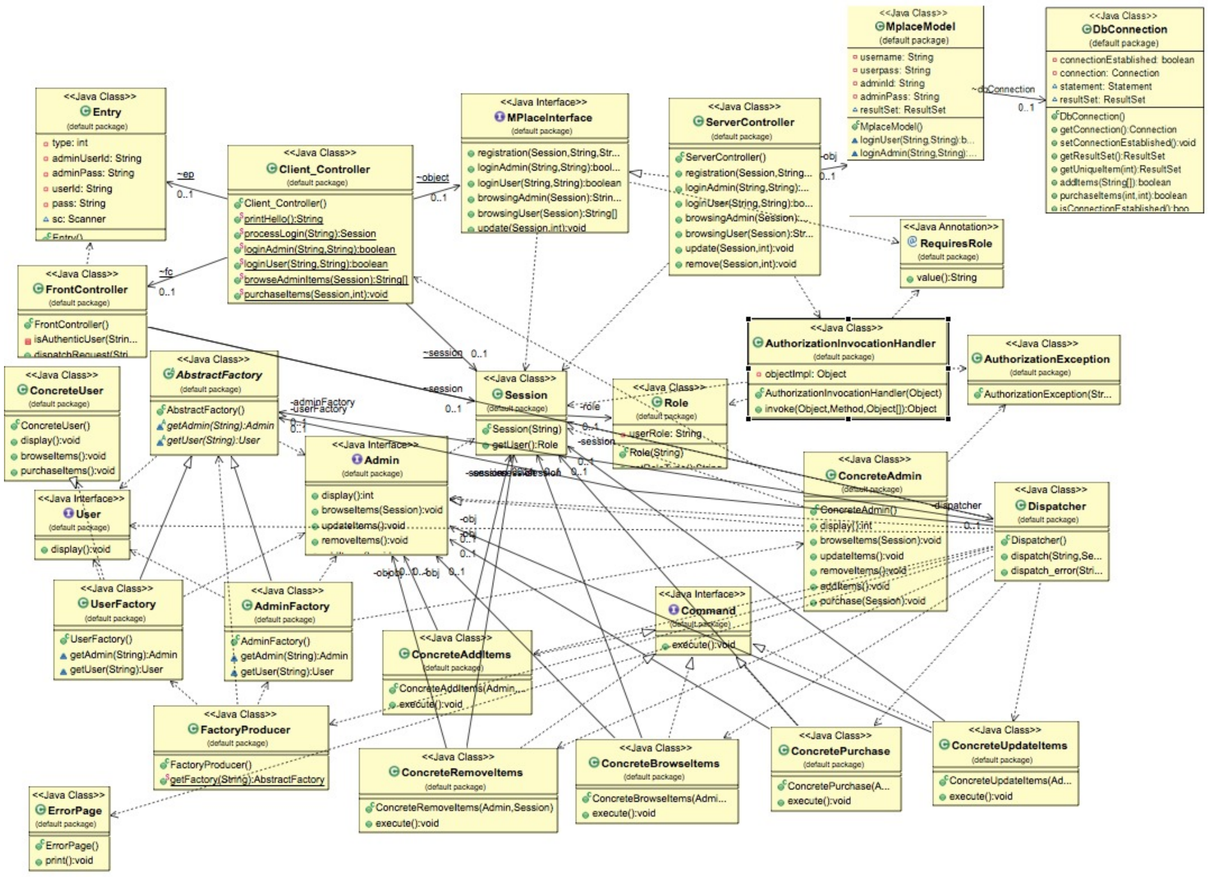


Fig [4]- Class Diagram

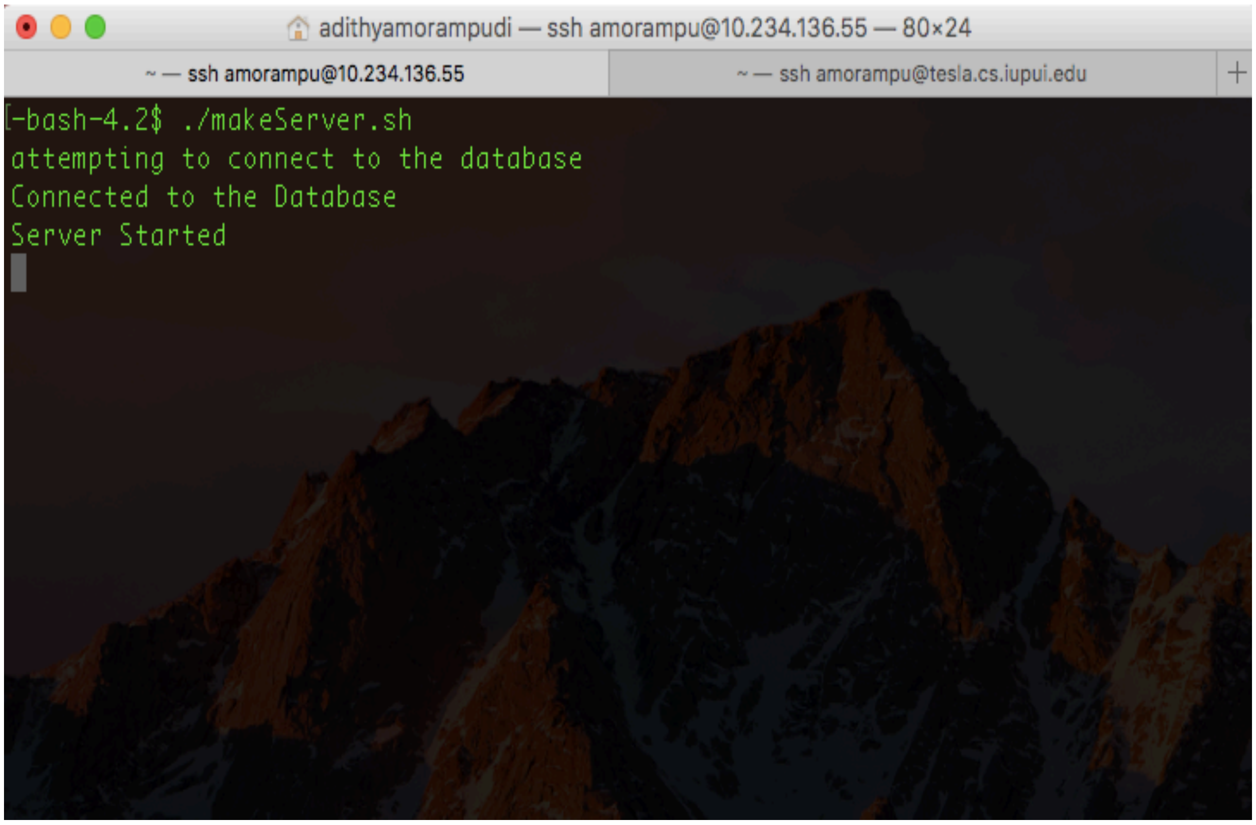
Newly Created Classes

- DbConnection.java

  This class helps us in establishing the connection to the database and helps us in making our queries work after connecting to the database, for now I have 5 different methods in this class, which help us in establishing the connection, getting the result set which helps the browse items method, purchase items which helps the user to purchase the items and change the stock value to stock – 1, and also the getUniqueItem method which helps us in determining the quantity of the product, if the quantity is less than 1, then it doesn't allow the purchase items method to execute, if not the purchase item method is executed successfully. I could've included all this logic in the model of the application, but for essential separation of concerns and as a good practice, I have separated this class out and implemented all the database functionality in this class.

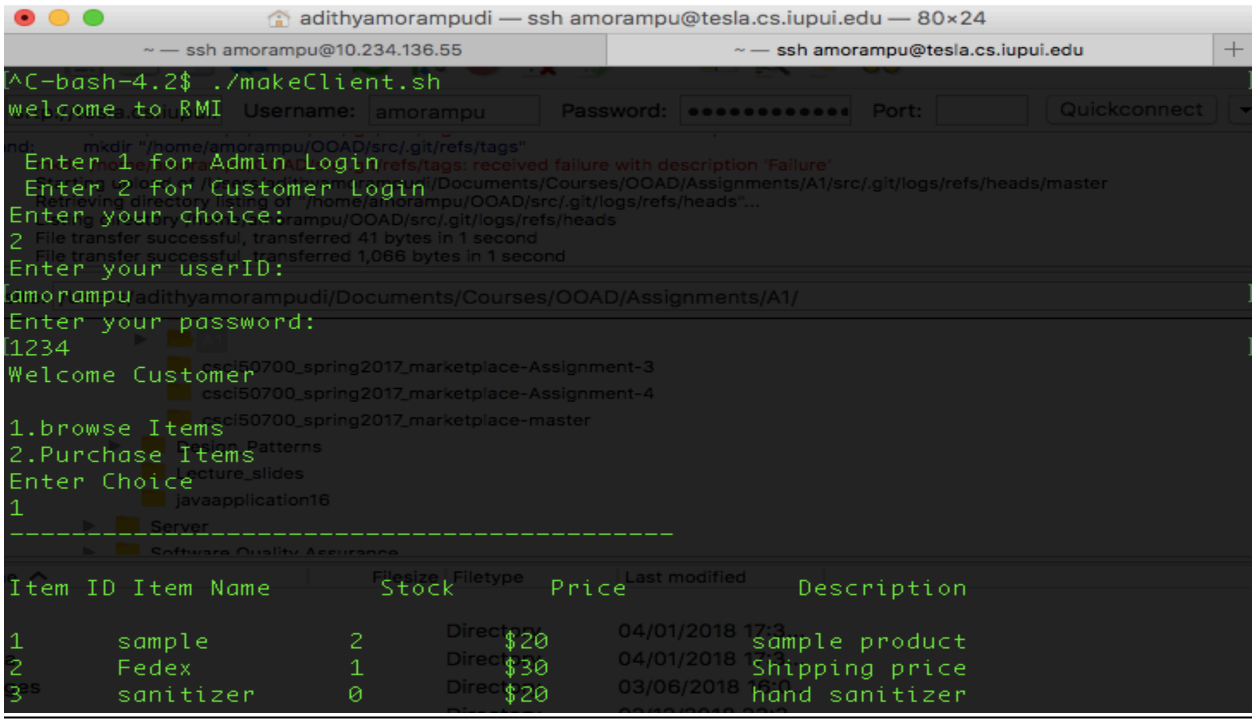
Sample Runs

Screen Shot to show the running Instance of the Server, please note, the server should always be run on 10.234.136.55 only. As soon as the server is started it first connects to the database. As you can see from the below figure. Now we can run different clients on different machines and our server can handle concurrent requests, without any trouble.


adithyamorampudi — ssh amorampu@10.234.136.55 — 80×24
~ — ssh amorampu@10.234.136.55
~ — ssh amorampu@tesla.cs.iupui.edu

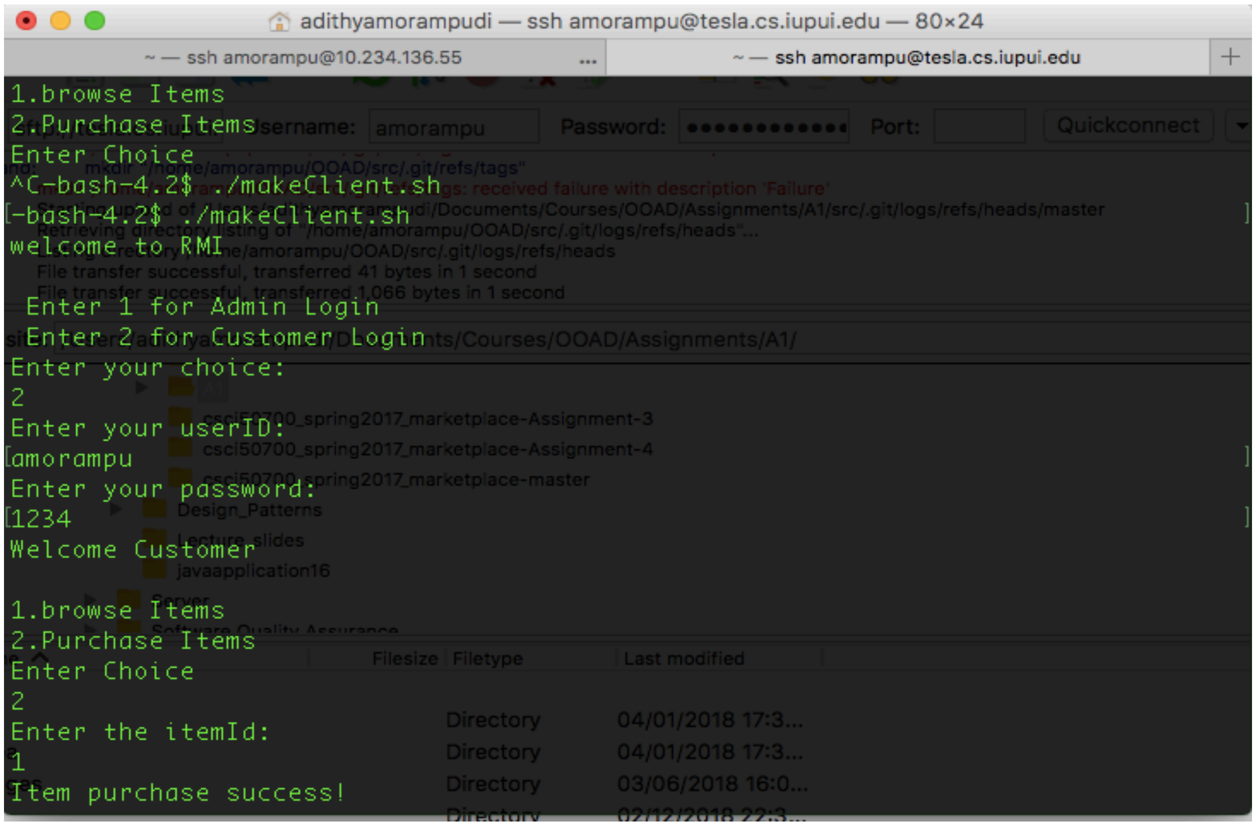
```
[-bash-4.2$ ./makeServer.sh
attempting to connect to the database
Connected to the Database
Server Started
```

A running Instance of the client in the figure below, which is being executing and sending requests to the server, we can see that the client can get the items from the **database** and display it to the user. Now, the user can select the other available options based on the output shown to him. The client side of the application displays the following, the entry screen will display a view which asks the user to select if it is a customer login or Admin login, based on the user input, the respective view is rendered and displayed to the user, once the view is displayed, the user is prompted to enter credentials and if they are valid then a number of options are available to him/her, based on the choices available the user can select the required one and can start accessing the application, all the options are displayed in the below screenshots.



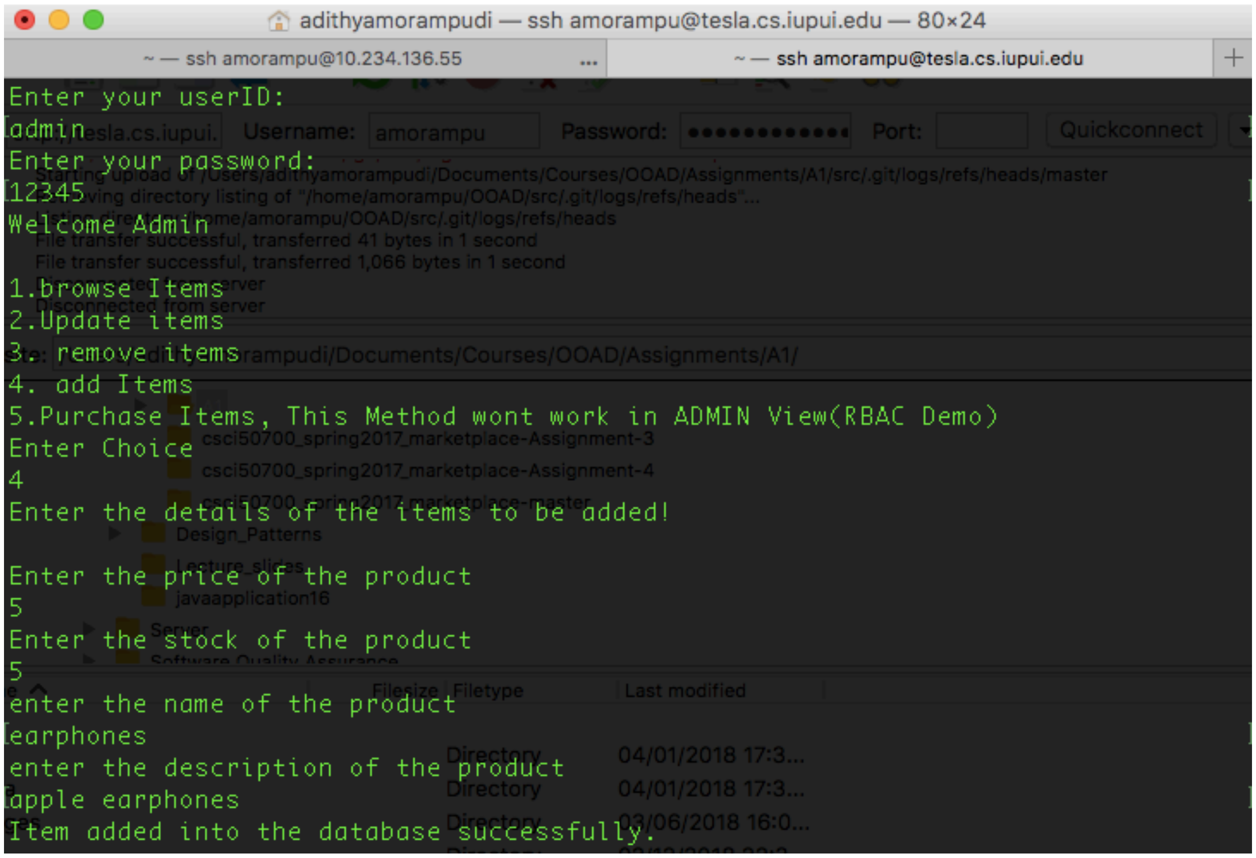
```
^C-bash-4.2$ ./makeClient.sh
welcome to RMI

 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
1
-------------------------------------------

Item ID Item Name           Stock     Price          Description

1       sample          2         $20         sample product
2       Fedex           1         $30         Shipping price
3       sanitizer       0         $20         hand sanitizer
```

Screenshot to show the **purchase** functionality, when a user tries to purchase an item he needs to display enter the id related to the product and if the stock is available then user can purchase it, otherwise an error message is displayed saying the product is out of stock.



```
1.browse Items
2.Purchase Items
Enter Choice
^C-bash-4.2$ ./makeClient.sh
-bash-4.2$ ./makeClient.sh
welcome to RMI

 Enter 1 for Admin Login
 Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
amorampu
Enter your password:
1234
Welcome Customer

1.browse Items
2.Purchase Items
Enter Choice
2
Enter the itemId:
1
Item purchase success!
```

Screenshot to show the add items functionality, this function is restricted to the administrator of the application, once the admin is logged in, he or she can then access the add item function and add the items into the database, please note that the id of the item is auto generated and auto populated, so the admin only needs to enter the remaining fields.


adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 80×24
~ — ssh amorampu@10.234.136.55
~ — ssh amorampu@tesla.cs.iupui.edu

```
Enter your userID:
admin
Enter your password:
12345
Welcome Admin

1.browse Items
2.Update items
3. remove items
4. add Items
5.Purchase Items, This Method wont work in ADMIN View(RBAC Demo)
Enter Choice
4
Enter the details of the items to be added!

Enter the price of the product
5
Enter the stock of the product
5
enter the name of the product
earphones
enter the description of the product
apple earphones
Item added into the database successfully.
```

Conclusion

After Implementing this assignment, I have understood what exactly RMI Concurrency is, and how threads in java RMI work, I understood, how concurrent access to the same resource can be protected and how multiple can access the same resource at the same time.

References

[1]. https://stackoverflow.com/questions/3507253/java-rmi-and-synchronized-methods
[2]. https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html
[3]. Class slides