Online Market Place Application

Assignment #5: Report

Implementation of Complete Functionality, Discussion of Java Synchronization and its Patterns

Course: CSCI 50700 - Object-Oriented Design and Programming

By
Mani Manjusha Kottala
IUPUI, Computer Information Science

Table of Contents

Introduction:

In previous assignment 1, an application is developed using (MVC) Model-View-Controller and Java RMI. In assignment 2, implemented the login functionality for user, admin and to use Front Controller pattern, Command pattern and Abstract Factory pattern. In assignment 3, three more patterns Authorization, Proxy and Reflection are used to integrate role-based access to the application. In assignment 4, implemented the Purchase Item, Add Item and Browse Item functionalities and explored the usage of Java RMI usage for concurrency simulation in multiple clients. In this Assignment, the goal is to implement the complete functionality of the Online Market Place application and appropriate use of Java synchronization for ensuring access to shares resource is thread-safe.

Assignment 4 – Feedback:

Feedback provided on import of packages and scope associated with an object. I have fixed this by importing only the required packages and changing the visibility to private.

Java Synchronization and Patterns:

Synchronization in Java can be defined as the ability to control the access to any shared resources by multiple threads. In Java, Synchronization can be provided using the two ways i.e., 1. By using Synchronized method and 2. By using Synchronized statements.

Synchronized Methods:

Using the synchronized method, if a thread is accessing a synchronized method for an object, a lock will be acquired by that current thread and all other threads trying to access that method for the same object will be blocked until the execution of synchronized method is completed [2]. This also ensures that all the waiting threads are notified with any changes to the object after execution of the method.

Synchronized Statements:

A Synchronized block is used for this synchronizing some statements of the method instead of synchronizing the whole method itself. An object on which the lock should be acquired should be specified with the synchronized keyword.

Some of the Synchronization patterns are discussed below:

Monitor Object Pattern:

The synchronization in Java is implemented using the concept of monitors. Monitors in Java can be referred to as – a mechanism that monitors and controls the concurrent thread access on an object and its shared resources. This monitor ensures that only one thread can access the critical sections of the code. Each object is associated with a Monitor and this Monitor is responsible for scheduling thread execution sequences in critical section. This pattern provides the mechanism for mutual exclusion and waiting for the threads trying to access the critical section. A monitor lock is used for serialization of synchronized method invocation which can be acquired or released depending on the monitor condition. Monitor conditions (wait, notify and notify all) are used for providing the cooperative execution scheduling. The Monitor conditions are used for checking the queue for the thread execution. Monitor Object pattern is implemented in this assignment by synchronizing the critical sections of the code by synchronized block statements.

Future:

As we know that synchronization ensures access of only one method at a time, other methods should wait till the resource is available. To avoid this, wait and improve interaction among methods, future pattern is implemented in the market place app. So, if one thread keeps on executing making the other thread waiting may result in improper use of result i.e., result may not be available when the other client wishes to use it. Using a future promises the client by returning a virtual object which tracks the execution status and provide the desired result to the waiting client, hence acting as a virtual proxy.

Guarded Suspension:

In Guarded suspension patterns, a pre-condition must be satisfied and a lock to be acquired for the execution of a method. To avoid dead-lock situation, this pattern should only be used if we know that the method call will be suspended for a finite period. In Guarded suspension, when the pre-condition is not met the client goes into wait state instead of suspending. When the pre-condition is satisfied after a finite point of time, the client is notified and the thread execution resumes.

Scoped Locking:

When a programmer acquires and releases of locks explicitly, there might be situations when the programmer might miss releasing a lock or an unexpected abort of program can lead to failure in releasing the lock. The motivation for this pattern is to avoid the mentioned deadlock situation or problems. Scoped Locking states that a scope should be always defined for the critical section and a lock is acquired automatically when the control enters the defined scope. The lock release should also be automated whenever the control leaves the scoped block. This pattern helps in minimizing the error in execution of program due to problems in acquiring/releasing locks. The Monitor object pattern discussed above uses the concept of synchronized block for achieving synchronization. So, the Monitor Object pattern in Java is implemented using the scoped locking. In my Market Place application, the scoped locking is implicitly implemented when I used the Monitor Object pattern

Thread-Safe Interface:

Thread-safe interface pattern comes into action when there is a chance of intra-component method calls. In a distributed application, all the operations performed by users should be thread-safe. Writing the entire logic in a method containing its implementation requires locking for concurrent access. Self-deadlock can happen when two methods try to obtain lock in the same component. All these problems can be overruled by creating an application containing all the interface methods with public access and their logic or implementation in private methods. These private methods can only be called by its respective interfaces. Also, this interface has a right to obtain a lock on implementation and perform its action and allows other to execute after the lock is released. These protected or private implementation always believe in the interface methods that a proper lock is provided by them and don't involve in self-acquiring or releasing. Advantage is removing unnecessary locks, but in turn it increases overhead and method calls. In this assignment, a separate interface for all the public method is used.

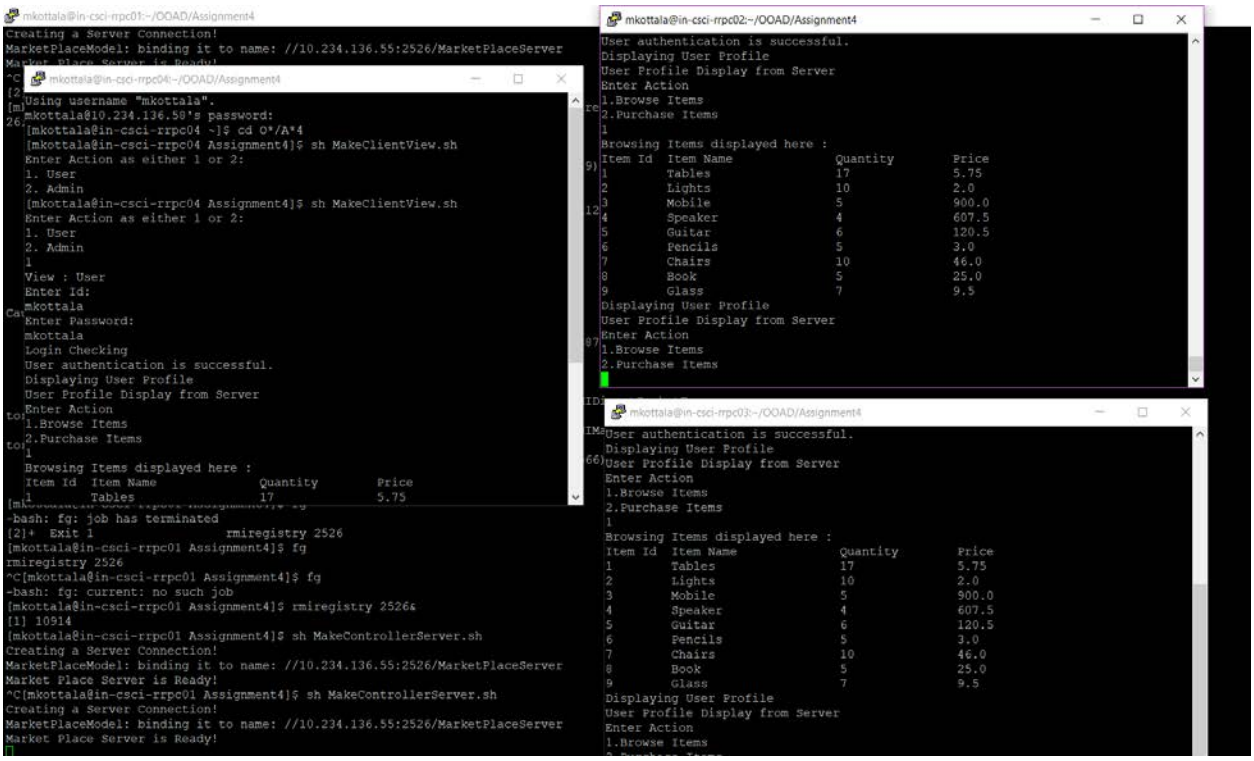
Java RMI Concurrency and Synchronization:

Concurrency can be referred as the ability to perform different tasks or parts of a task in parallel. Multi-threaded programs are typically used for achieving concurrency in Java where different threads can handle different tasks in a program. In the Multi-threaded programming context, the main issue is when different threads try to perform the same action or try to use the same resource. Java provides locking

mechanism to handle this situation where a lock is acquired by a thread on the shared resource and when another thread wants to access the resource, it must wait till the lock is released.

Thread usage in Java RMI can be discussed both in client side and server side. In client side, a request is marshalled and send to through remote invocation [1]. The requested thread waits till it gets the response back from the server, So the calling thread is in block state until it receives the response back from the server. So, when multiple clients try to access the remote objects, the client requester threads will be in blocked state until they receive a response back from the server. But in the server side as this statement states – *"A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread" [3]*, if concurrent calls are made from same client then there is no guarantee that they might execute on different threads [4]. The concurrent calls from different clients run on different threads. But for this concurrent access Java RMI doesn't provide any thread safety, so synchronized keyword can be used to make the Java RMI thread safe. The synchronization should be provided by examining the behavior of the objects to avoid unnecessary bottlenecks in concurrency or deadlocks.

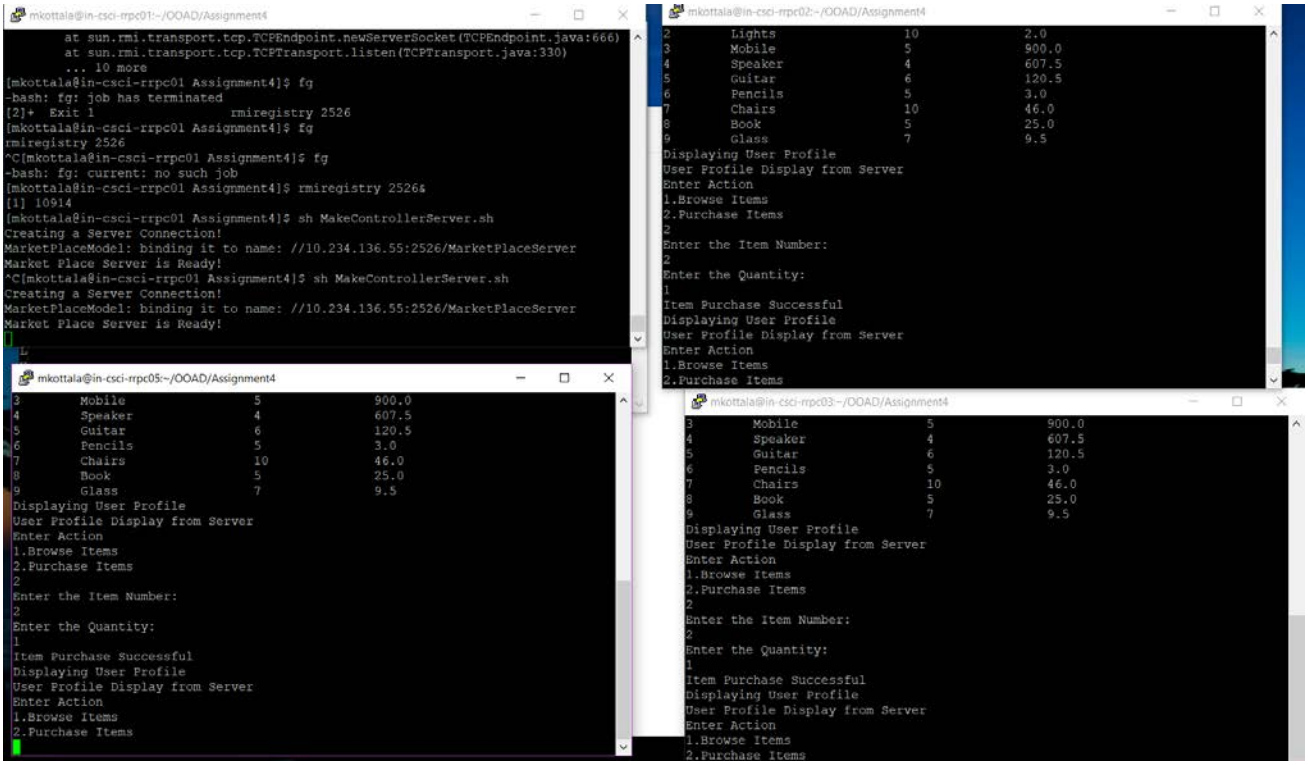
Java RMI in Market Place Application: In the current application, the Java RMI multi-threading is used. The functions for browse items, add items and purchase items can be accessed by multiple clients at the same time, for making the application thread safe these function use synchronizations. The application uses machine - **10.234.136.55** as the server as it has the database connection. By using the synchronized keywords, the access of the same function is limited to a single thread at any point of time. As Java RMI provides currency using multi-threading the application can run on two clients at the same time. For example, I have clients **10.234.136.56**, **10.234.136.57** and **10.234.136.58** accessing the same browse functionality. These three clients can access the same method concurrently as shown in the screenshots belows:

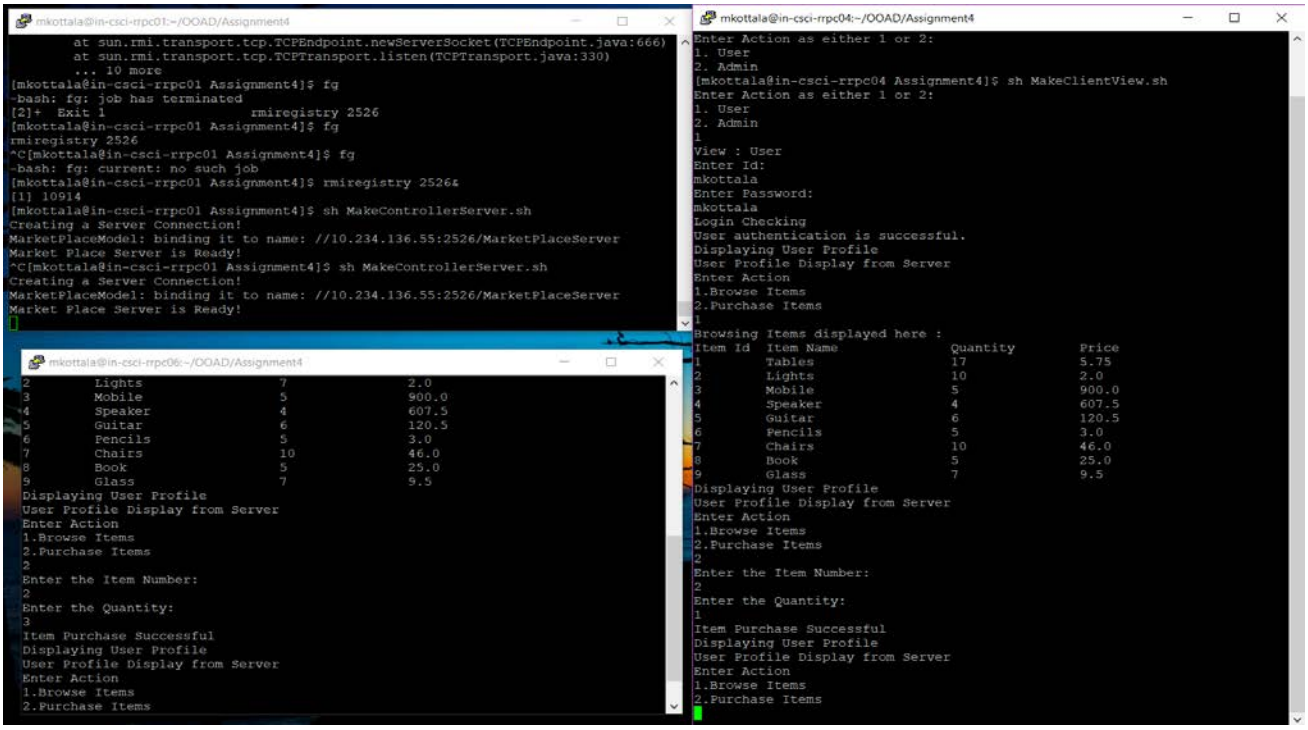


In absence of synchronized keyword, the function accessed can be concurrently but there may be cases where inconsistencies and errors may occur. If we consider the add purchase item and if two clients

are trying to purchase the same item at the same time, there can be inconsistencies in update to database as both the clients will try to update the same item's quantity. Here synchronized keyword helps to maintain consistence in the application while concurrent access. I have tried accessing the RMI without the synchronized key word for browse items as shown above and it shows that the Java RMI is multi-thread and allows concurrency.

As per the requirements the application must run on five clients, I have tried running the application on all the available six machines - **10.234.136.55** as server and other five machines as clients. We can see in the below screenshots that all the clients were able to communicate with server simultaneously.





To know the ideal behavior of the synchronized keyword, the functions should be accessed exactly at the same time from two different machines. So, the Java RMI provides multi-threading with concurrency and the application can be made thread safe using the synchronized keyword. But the usage of synchronized should be determined on the behavior of the functions on which it is applied to avoid any unnecessary deadlocks.

Functionalities Implementation:

One of the requirements in this assignment is complete implementation of the following functions of Admin and User:

- **Add Items to Cart**: This method can be accessed by Customer. Customer can add items to the shopping cart using this method. A remote method addItemsToCart() is called which takes item_id and quantity as parameters and returns a string to display the success or failure in adding to the database.
- **Display cart**: This method can only be accessed by Customer. The remote method diaplayCart() is called which returns an ArrayList object which has the list of objects.
- **Checkout**: This method can only be accessed by Customer. This method is used for purchase of all the items in the shopping cart. A remote method purchase() is called which returns an ArrayList containing the each of the cart items purchase status. The items which are in stock are purchased and their quantities in the items table are updated.
- **Delete Items**: This method can only be accessed by Admin. This method is used by admin to delete the items from the items table. A remote method call deleteItems() is made with itemId as parameter passed. This method returns a string with the status of deletion.
- **Update Items**: This method can only be accessed by Admin. This method is used by admin for updating – Quantity, Descriprion or price of an item. A remote method updateItems() with

parameters item id, item field (to specify which field to update) and itemUpdate (updated value) is called. This method returns a string with the status of successful update or failure.

- **Add Admin/Customer**: These methods can only be accessed by Admin. Respective remote methods addAdmin() and addCustomer() are called with the parameter String array which contains their details. These methods return a string with the status of successful insertion or failure
- **Remove Customer**: This method can only be accessed by Admin. A remote method removeUser() with parameter user id is called for deleting the customer from the customers table.

New Classes created:

- **AddAdminConcrete**: This is a concrete class for adding admins which implements the Actions interface for admin. It implements the execute() method which calls the user purchase function.
- **AddUserConcrete**: This is a concrete class for adding customers which implements the Actions interface for admin. It implements the execute() method which calls the user purchase function.
- **RemoveUser**: This is a concrete class for removing customers which implements the Actions interface for admin. It implements the execute() method which calls the user purchase function.
- **AddItemsToCart**: This is a concrete class for adding items to cart which implements the UserActions interface. It implements the execute() method which calls the user purchase function.
- **DisplayCart**: This is a concrete class for displaying items of cart which implements the UserActions interface. It implements the execute() method which calls the user purchase function.

Class Diagram[6]:

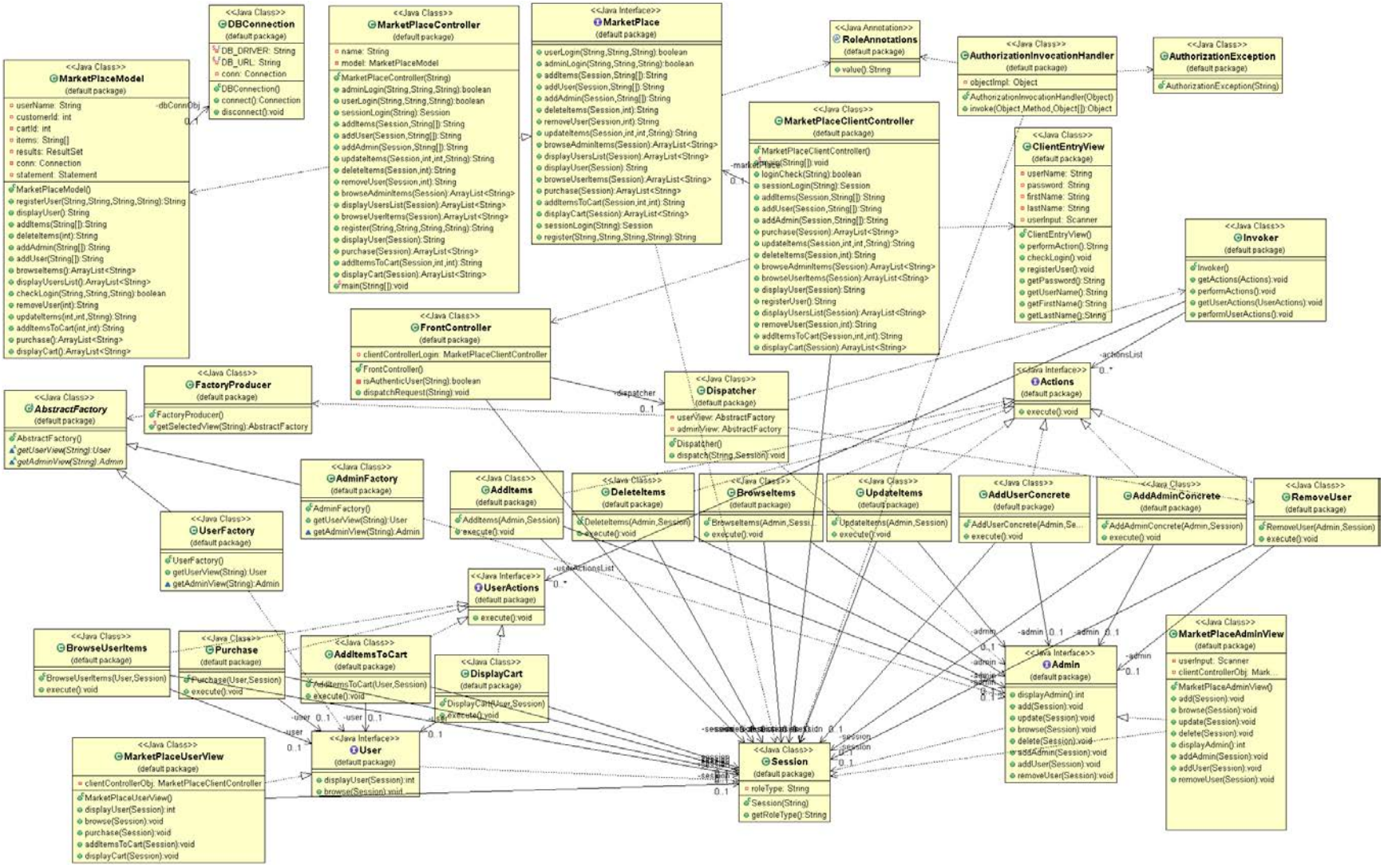


Fig 2: Class Diagram for MarketPlace Application

Sample runs:

1. Starting the RMI registry and Running the server application on Machine - **10.234.136.55**

```
[mkottala@in-csci-rrpc01 Assignment4]$ rmiregistry 2526&
[1] 12119
[mkottala@in-csci-rrpc01 Assignment4]$ javac -cp "/home/mkottala/OOAD/Assignment4/mysql-connector.jar" *.java
[mkottala@in-csci-rrpc01 Assignment4]$ java -cp ".:/home/mkottala/OOAD/Assignment4/mysql-connector.jar" -Djava
.security.policy=policy MarketPlaceController
Creating a Server Connection!
MarketPlaceModel: binding it to name: //10.234.136.55:2526/MarketPlaceServer
Market Place Server is Ready!
```

Or

Using makefile: MakeControllerServer.sh

```
[mkottala@in-csci-rrpc01 Assignment4]$ rmiregistry 2526&
[1] 12495
[mkottala@in-csci-rrpc01 Assignment4]$ sh MakeControllerServer.sh
Creating a Server Connection!
MarketPlaceModel: binding it to name: //10.234.136.55:2526/MarketPlaceServer
Market Place Server is Ready!
```

2. Running the Client application on any of the machines - 10.234.136.56, 10.234.136.57, 10.234.136.58, 10.234.136.59 and 10.234.136.60

```
Enter Action as either 1, 2 or 3:
1. User
2. Admin
3. Customer Registration
3
```

3. User Functions:

User can Browse the items and purchase the items. Below are the screenshots for user functionalities:

```
Displaying User Profile
User Profile Display from Server
Enter Action
1.Browse Items
2.Checkout (Purchase cart Items)
3.Add Item to Cart
4.Display cart
3
**************************************** ITEMS DISPLAYED HERE ****************************************

Item Id  Item Name              Description             Quantity        Price
7        Tables                Tables                  25              50.0
8        Speaker               Bose Speaker            2               55.0
9        Laptop                Dell Laptop             49              50.0
10       Clock                 Clock                   14              24.0
11       Light Stand           Mainstays               14              46.0
14       Shutters              Mainstays               14              25.0
15       PenDrive              Sony                    13              19.0
******************************************************************************************************************
**************************************** ADD ITEMS TO CART ****************************************
Enter the item Id to add item to cart:
11
Enter item Quantity:
5
Added item Successfully
```

```
Displaying User Profile
User Profile Display from Server
Enter Action
1.Browse Items
2.Checkout (Purchase cart Items)
3.Add Item to Cart
4.Display cart
4
****************************************** DISPLAYING CART ************************************
Cart Items displayed here :
CartId   ItemId   Quantity
17        7        5
17        11       5
***********************************************************************************************
Displaying User Profile
User Profile Display from Server
Enter Action
1.Browse Items
2.Checkout (Purchase cart Items)
3.Add Item to Cart
4.Display cart

2
************************************* CHECK OUT ***********************************************
Item Id : 7 Successfully Purchased
Item Id : 11 Successfully Purchased
***********************************************************************************************
```

4. Admin Functionalities:

In this assignment, Add Items and browse items functionalities are implemented for the admin. Screenshot for them are below:

```
Enter Action as either 1, 2 or 3:
1. User
2. Admin
3. Customer Registration
2
View : Admin
Enter username:
manju
Enter Password:
manju
Login Checking
Admin authentication is successful.
Displaying Admin Profile
For exit enter anything other than 1,2,3,4,5,6,7
1.Add Items
2.Delete Items
3.Update Items
4.Browse Items
5.Add Admin
6.Add Customer
7.Remove Customer
```

```
1
****************************************** ADDING ITEMS ********************************
Enter Item Name:
Computer
Enter Item Description:
Dell
Enter Item Price:
550
Enter Item Quantity:
60
Item has been added
************************************************************************************************
Displaying Admin Profile
```

```
For exit enter anything other than 1,2,3,4,5,6,7
1.Add Items
2.Delete Items
3.Update Items
4.Browse Items
5.Add Admin
6.Add Customer
7.Remove Customer
3
************************************************************************************************
ITEM ID  ITEM NAME              DESCRIPTION                 QUANTITY        PRICE
7        Tables                 Tables                      20              50.0
8        Speaker                Bose Speaker                2               55.0
9        Laptop                 Dell Laptop                 49              50.0
10       Clock                  Clock                       14              24.0
11       Light Stand            Mainstays                   9               46.0
14       Shutters               Mainstays                   14              25.0
15       PenDrive               Sony                        13              19.0
Enter Item Id from above list:
11
************************************************************************************************
Enter update action 1, 2 or 3
1. Update item Description
2. Update item Price
3. Update item Quantity
2
Enter Update Value :
75
Updated Item Description
************************************************************************************************
```

```
6
****************************************** ADD USER ************************************
Adding User
Enter First Name:
Chris
Enter Last Name:
Antony
Enter username:
chris
Enter password:
chris
Customer has been added
************************************************************************************************
```

```
**********************************************************************************************************
Displaying Admin Profile
For exit enter anything other than 1,2,3,4,5,6,7
1.Add Items
2.Delete Items
3.Update Items
4.Browse Items
5.Add Admin
6.Add Customer
7.Remove Customer
7
****************************************** REMOVE USER ************************************
Id       User Name
2        mkottala
25       rahul
26       blair
27       tessa
29       stone
30       swapna
31       barma
34       srikar
35       ramesh
36       chris
Enter Customer Id to delete from above list:
36
Customer is deleted
**********************************************************************************************************
```

5. Terminating RMI registry:

```
[mkottala@in-csci-rrpc01 Assignment4]$ fg
rmiregistry 2526
^C[mkottala@in-csci-rrpc01 Assignment4]$
```

Conclusion:

In this assignment, I have understood how synchronization and various design patterns will help a distributed application to function in a thread-safe manner along with providing concurrent features. Also implemented full functionalities in the market application based on given requirements.

References:

[1] https://www.safaribooksonline.com/library/view/java-rmi/1565924525/ch11s06.html

[2] https://www.javatpoint.com/synchronization-in-java

[3] https://docs.oracle.com/javase/6/docs/platform/rmi/spec/rmi-arch3.html

[4] https://stackoverflow.com/questions/1300145/race-condition-in-rmi-java-2-client-1-server

[5] http://www.informit.com/articles/article.aspx?p=26251&seqNum=3

[6] https://dzone.com/articles/uml2-class-diagram-java

[7] class slides