Online Marketplace

Assignment 5



Samira Khorshidi

TAble of  
Contents

Assignment Discussion 1

Implementation overview 1

Synchronization 2

FiGures, Sample codes 5

UML Diagram 5

Sample Codes 6

Sample Runs 8

Screenshots 8

Discussion 15

Conclusion, References 17

Conclution 17

References 18

# Assignment Discussion

The purpose of this practice is designing Domain Model of online Marketplace using Java RMI and MVC design pattern and by leveraging useful and proper design patterns.

## OverView, Implementation

The goal of this assignment is having a synchronized distributed system using available tools and patterns. To achieve such a system, we first defined the synchronization as having order in execution of clients requests and define our critical blocks. In this implementation, the methods are well defined tasks with specific rule within entire system, so we decide to have each of them as a block that needs to be synchronized.

In this implementation we found **synchronized** keyword sufficient in order to achieve a desire system. As a test of synchronization, we put a function call in line 33 of FrontController which call server.concurrencyTestSync(); in a synchronized fashion.

To use database in our system we defined a specific class named “databaseManager” and the only classes that are using an object of this class are models, UserModel and ProductModel. So we separate our database layer from other parts of systems.

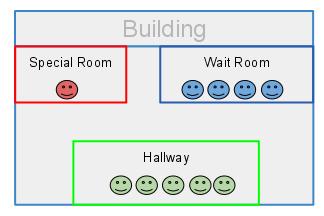
In following we will explain synchronization briefly:

## Synchronization and monitor concept

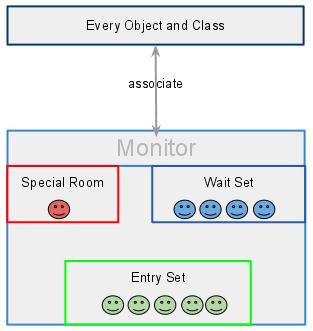
Monitor is an important concept of synchronization in operating systems. It is also used in Java synchronization, in following we will go through monitor concept and synchronization.

**1. What is a Monitor?**

A monitor can be considered as a building which contains a special room. The special room can be occupied by only one customer(thread) at a time. The room usually contains some data and code.



If a customer wants to occupy the special room, he has to enter the Hallway(Entry Set) to wait first. Scheduler will pick one based on some criteria(e.g. FIFO). If he is suspended for some reason, he will be sent to the wait room, and be scheduled to reenter the special room later. As it is shown in the diagram above, there are 3 rooms in this building.



In brief, a monitor is a facility which monitors the threads' access to the special room. It ensures that only one thread can access the protected data or code.

**2. How is it implemented in Java?**

In the Java virtual machine, every object and class is logically associated with a monitor. To implement the mutual exclusion capability of monitors, a lock (sometimes called a mutex) is associated with each object and class. This is called a semaphore in operating systems books, mutex is a binary semaphore.

If one thread owns a lock on some data, then no others can obtain that lock until the thread that owns the lock releases it. It would be not convenient if we need to write a semaphore all the time when we do multi-threading programming. Luckily, we don't need to since JVM does that for us automatically.

To claim a monitor region which means data not accessible by more than one thread, Java provide synchronized statements and synchronized methods. Once the code is embedded with synchronized keyword, it is a monitor region. The locks are implemented in the background automatically by JVM.

**3. In Java synchronization code, which part is monitor?**

We know that each object/class is associated with a Monitor. I think it is good to say that each object has a monitor, since each object could have its own critical section, and capable of monitoring the thread sequence.

To enable collaboration of different threads, Java provide wait() and notify() to suspend a thread and to wake up another thread that are waiting on the object respectively. In addition, there are 3 other versions:

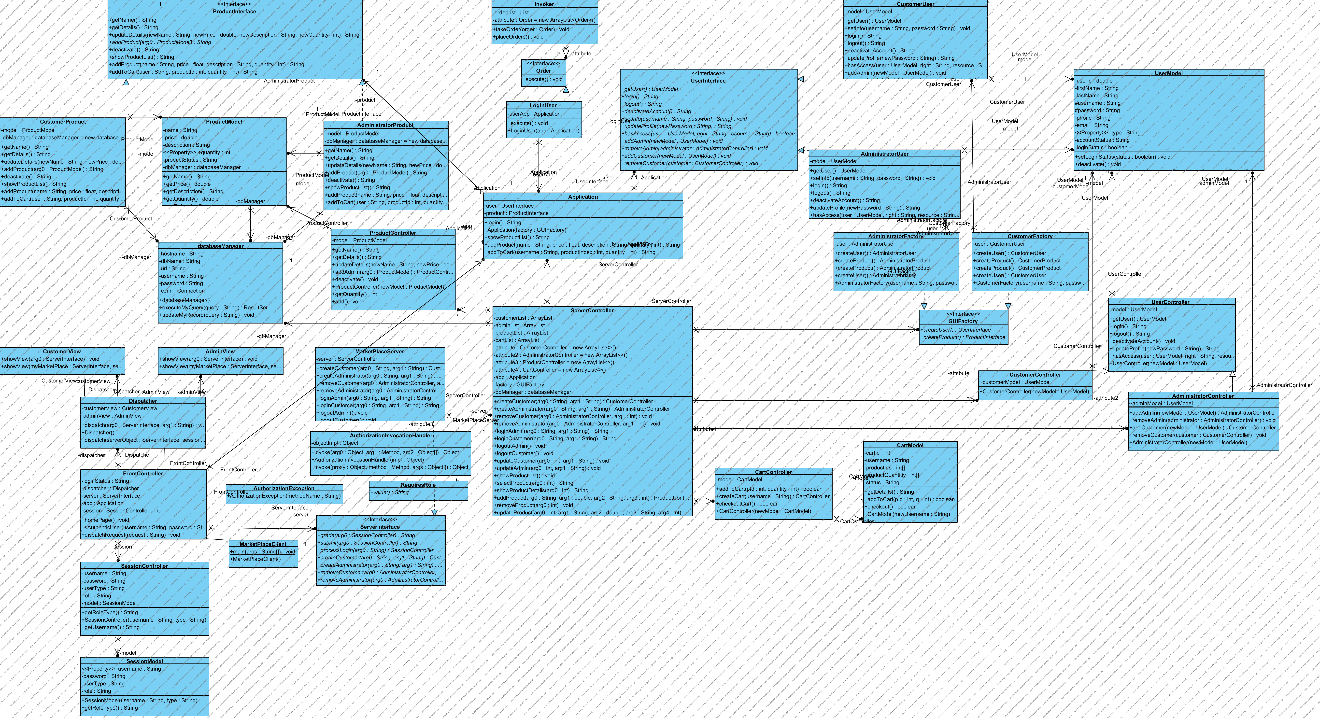
|  |
| --- |
| * wait(**long** timeout, **int** nanos) * wait(**long** timeout) notified by other threads or notified by timeout. * notify(all) |

Those methods can only be invoked within a synchronized statement or synchronized method. The reason is that if a method does not require mutual exclusion, there is no need to monitor or collaborate between threads, every thread can access that method freely.

# Figure, Sample codes

In following we provide you a Class Diagram of our system along with some sample codes.

## UML Diagram



The original UML diagram has been provided in Documentation directory. As you see, in our proposed architecture, we introduced a new class named SessionModel and also a temporary databaseManager class.

## Sample codes

In following we provide a sample code of the different new functionalities of the system:

Add a Administrator/Customer user:

public String addUser(String newUsername, String newPassword, String newType){

String commandStatus = null;

try{

String stm = String.format("INSERT INTO user(username, password, type, accountStatus) VALUES (' %s','%s','%s','active');",newUsername,newPassword, newType);

System.out.println(stm);

int result = dbManager.updateMyRecord(stm);

if(result>0)

commandStatus = "User has been added successfully!";

else

commandStatus = "Please try again later!";

}catch(Exception e){

commandStatus = "Please try again later!";

System.out.println("Database Exception" + e.getMessage());

}

return commandStatus;

}

* Remove product:

public String deactivate(int index) {

String commandStatus = null;

try{

String product\_stm = String.format("UPDATE product SET status = 'deactive' WHERE productId = %s",index);

int result = dbManager.updateMyRecord(product\_stm);

System.out.println(result);

if(result>0){

commandStatus = "Product has been removed successfully!";

} else{

commandStatus = "Please try again later!";

}

return commandStatus;

}catch(Exception e){

commandStatus = "Please try again later!";

System.out.println("Database Exception" + e.getMessage());

}

return commandStatus;

}

* Update Product:

public String updateProduct(int productIndex, String newName, float newPrice, String newDescription, int newQuantity){

String commandStatus = null;

try{

String product\_stm = String.format("UPDATE product SET name = '%s', price=%s, description='%s', quantity = %s WHERE productId = %s",newName, newPrice, newDescription, newQuantity, productIndex);

int result = dbManager.updateMyRecord(product\_stm);

System.out.println(result);

if(result>0){

commandStatus = "Product has been updated successfully!";

} else{

commandStatus = "Please try again later!";

}

return commandStatus;

}catch(Exception e){

commandStatus = "Please try again later!";

System.out.println("Database Exception" + e.getMessage());

}

return commandStatus;

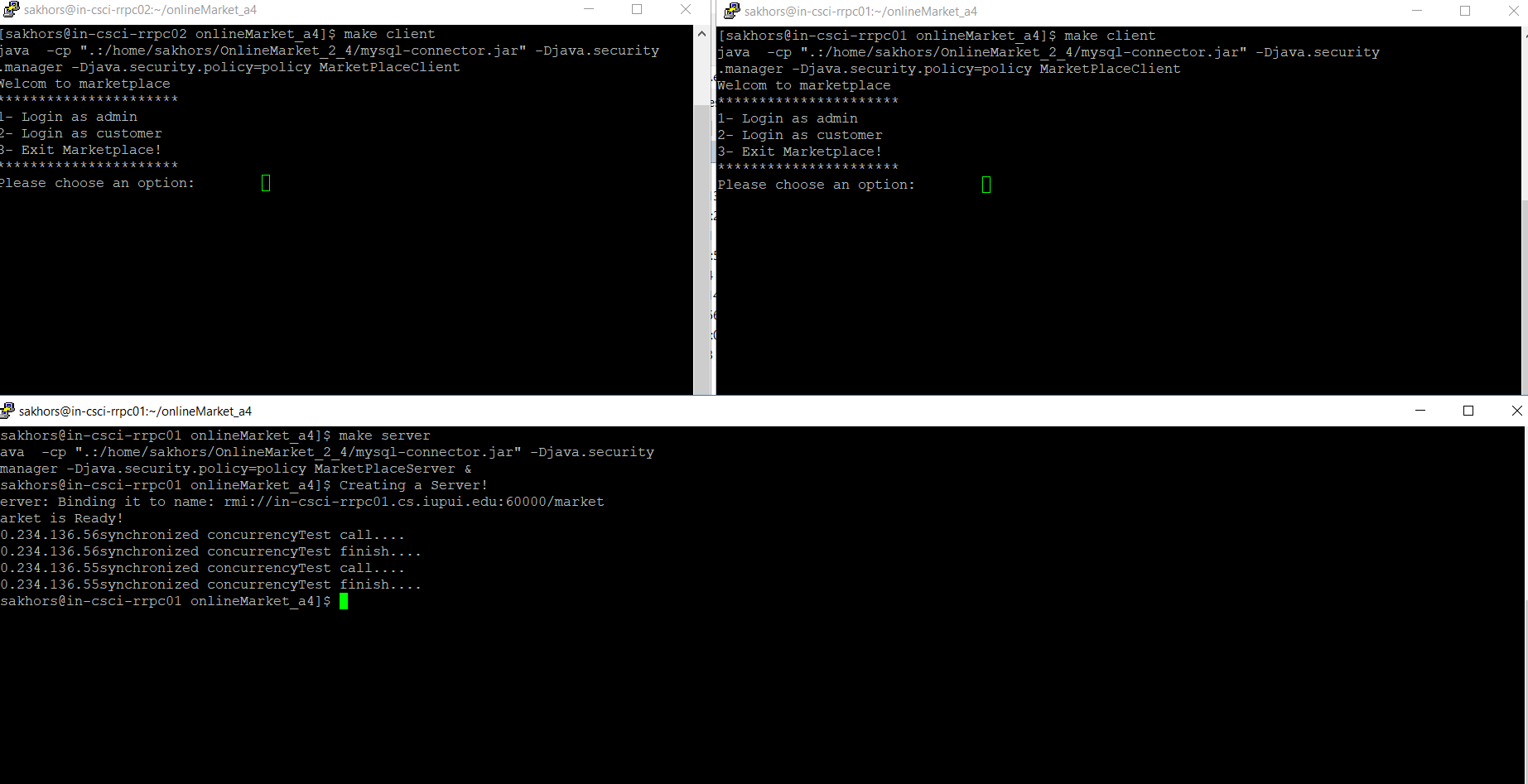
}

# Sample Runs

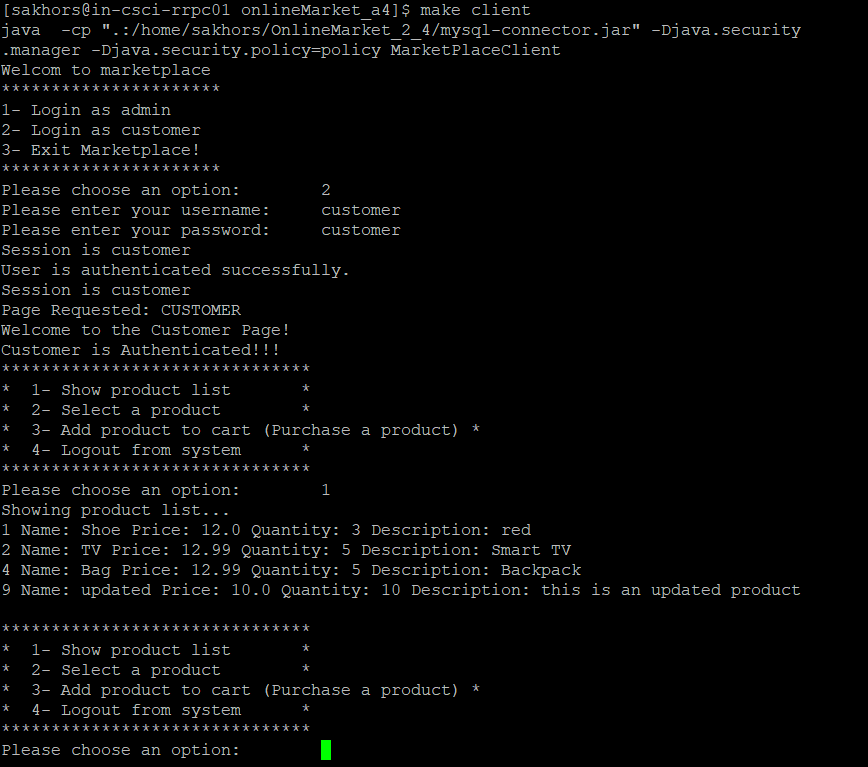
In this section you’ll see screenshots of sample runs. The system is minimal, so there is no GUI, instead every interaction happen using command prompt.

## Screenshots

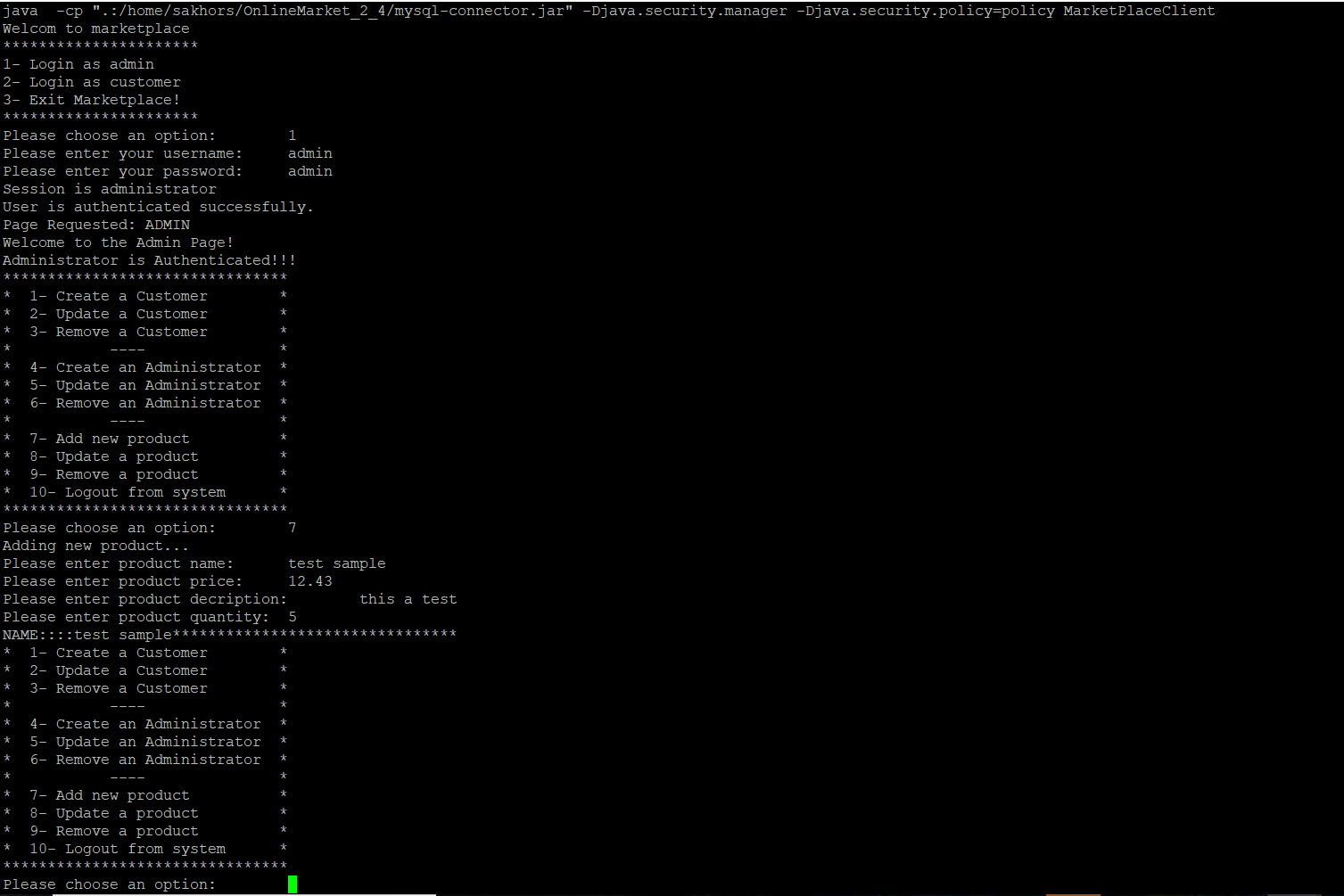
All required functionalities has been implemented and here is a sample run of an action from two different machine in synchronized way:



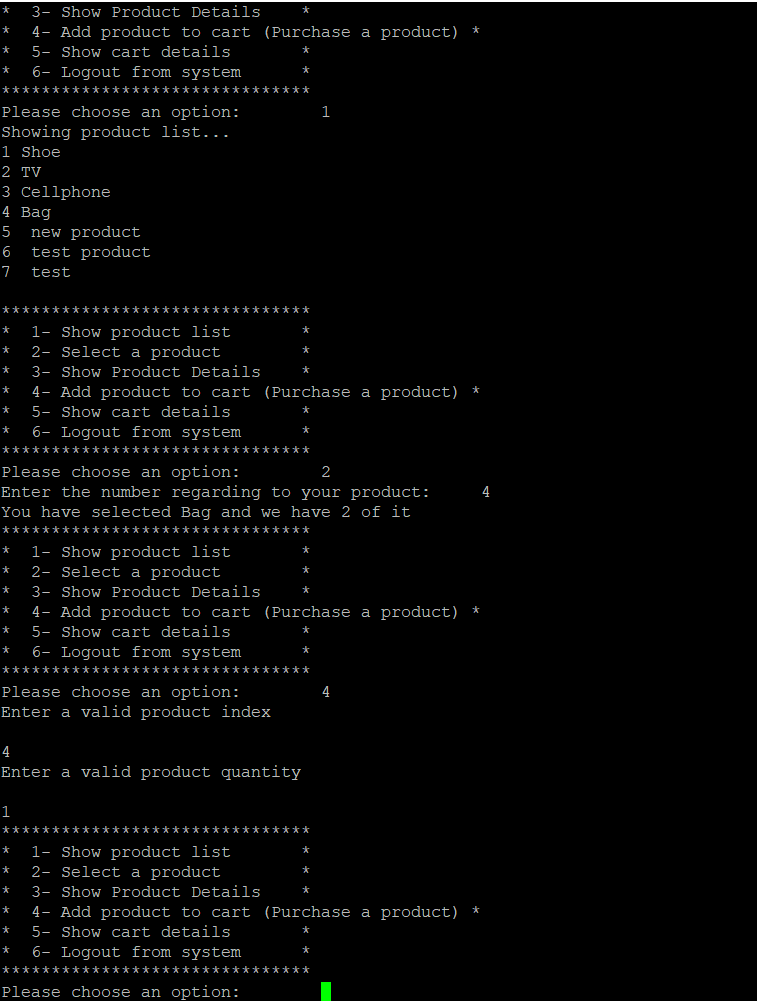
This is a sample run of browsing system products from database:



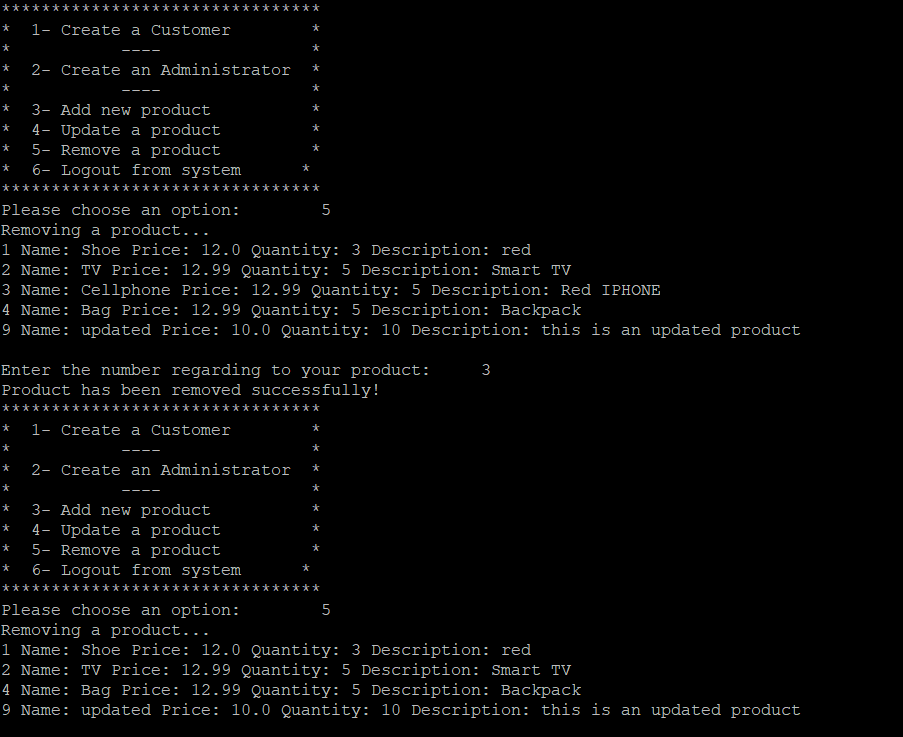
This a sample run of adding a new product by admin user:



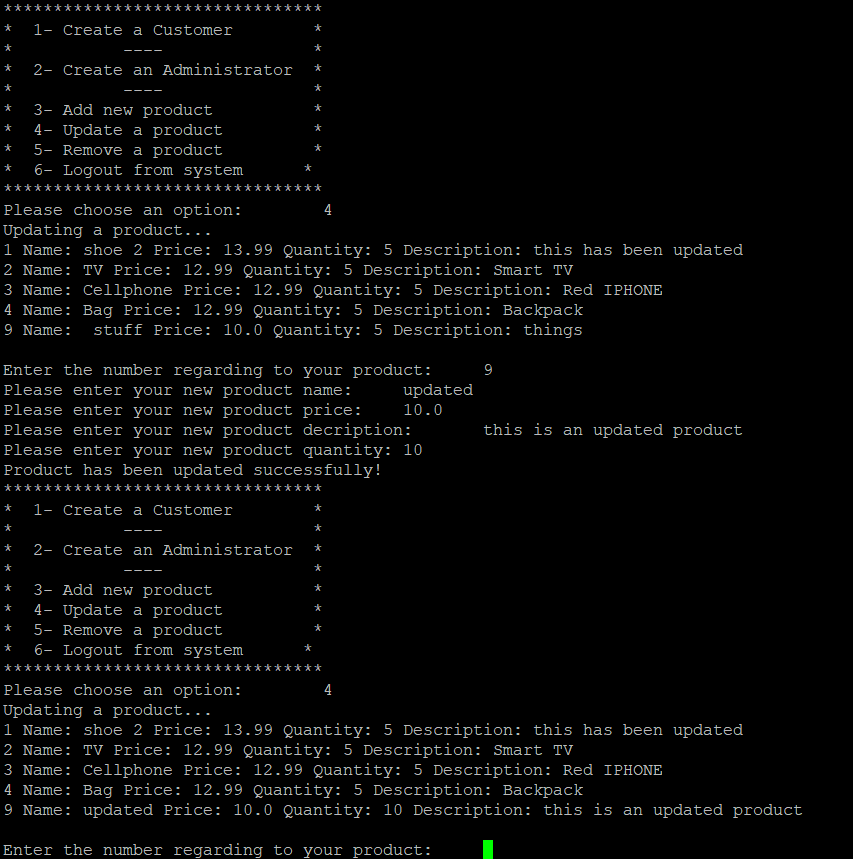
And this is a sample run of adding a product to the user shopping cart (Purchase):



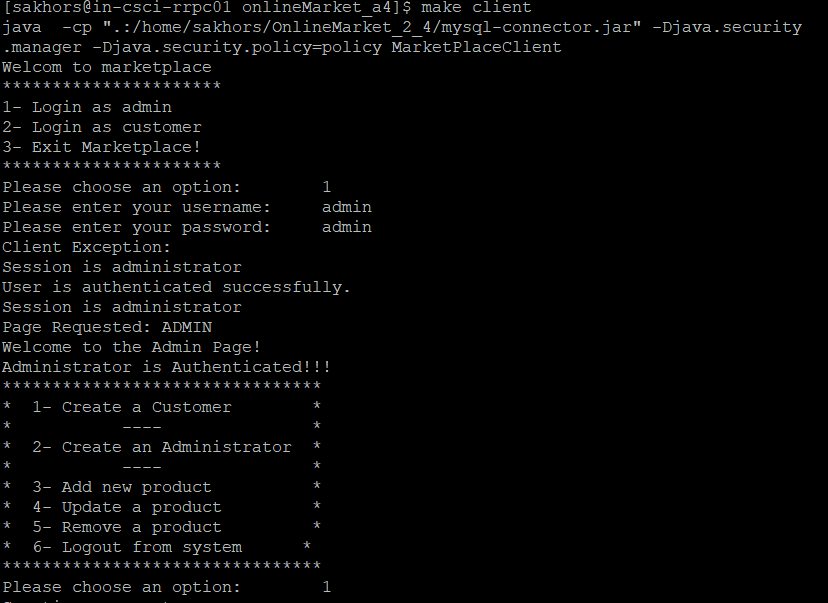
Here is remove product sample run:

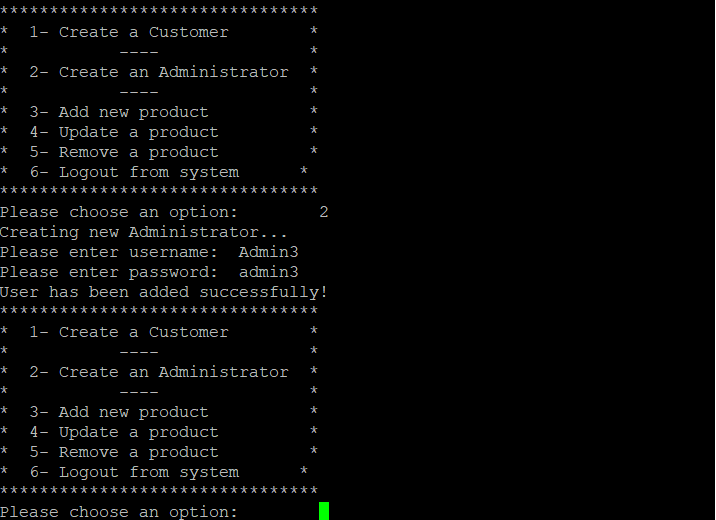
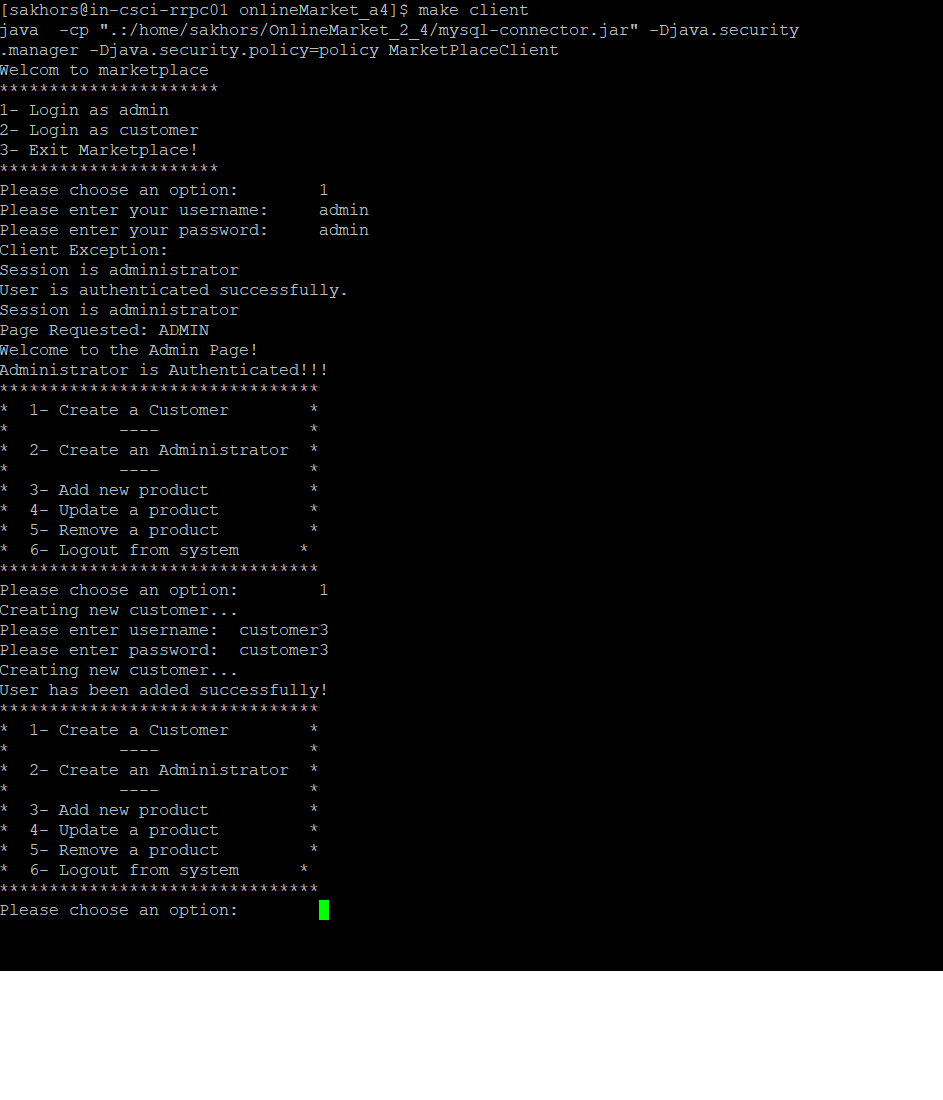


Update product:



In term of user functionalities, you can find login in following:



Add an Admin user:Add a customer user:

## Discussion

During this assignment, we focused on applying the information we have learned regarding the use of synchronization in the Java to ensure that access to our shared resources are indeed thread-safe and also we complete our system functionalities by implementing different functionalities including login, update Product, add users (customer and administrator) and remove a product. Based on the concept behind purchase, browse and add products, and having multiple clients, we would like to have them in a synchronized way rather than concurrent way.

# Conclution References

## Conclution

In conclusions, we found that in a way that we used java JVM, RMI and synchronized keyword, even if they give us no guarantee in term of keeping our system thread safe, but we found that each client has its own thread pool and we achieved synchronized trade-safe system.

## References

* <https://dzone.com/articles/java-callable-future-understanding>
* <https://link.springer.com/chapter/10.1007/978-3-540-45209-6_63>
* <http://winterbe.com/posts/2015/04/30/java8-concurrency-tutorial-synchronized-locks-examples/>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/syncrgb.html>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html>
* <http://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/>
* <https://www.programcreek.com/2011/12/monitors-java-synchronization-mechanism/>
* <http://blog.e-zest.com/java-monitor-pattern/>
* Java and developer’s forums
* Lecture slides