Online Marketplace

Assignment 4



Samira Khorshidi

TAble of  
Contents

Assignment Discussion 1

Java RMI Concurrency 1

FiGures, Sample codes 4

UML Diagram 4

Sample Codes 5

Sample Runs 8

Screenshots 7

Discussion 12

Conclusion, References 13

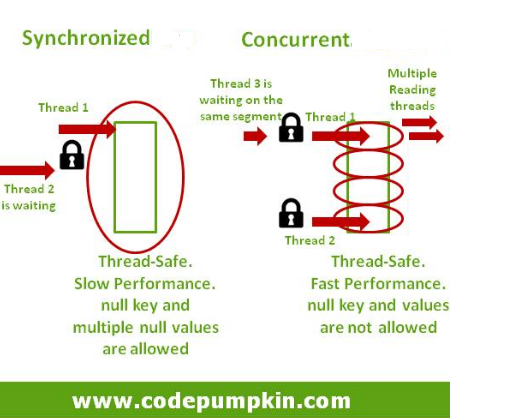
Conclution 13

References 13

# 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.

## Java RMI Concurrency



Above image give us a broad view of concurrent system vs synchronized system. Concurrent programming has two main benefits: First, it allows natural solutions to software design problems that are inherently parallel or distributed. Second, concurrent programs offer better efficiency than sequential programs. However, concurrent programming poses many challenges that do not exist in the sequential setting. For instance, the processes in the system may livelock, diverge, or even deadlock. The Java Remote Method Invocation (RMI) package facilitates the implementation of concurrent applications in which, for instance, the processes reside on different hosts and communicate over the internet. More precisely, it hides

the details of network communication. Unfortunately, it does not relieve the programmer from the potential pitfalls of controlling the concurrent access to remote objects. Consequently, RMI applications are prone to the same problems as concurrent programs in general.  
  
 In our application, since the system is distributed, logically, blocking in a remote object is simple. Suppose that client A calls a synchronized method of a remote object. To make access to remote objects look always exactly the same as to local objects, it would be necessary to block A in the client-side stub that implements the object's interface and to which A has direct access. Likewise, another client on a different machine would need to be blocked locally as well before its request can be sent to the server. The consequence is that we need to synchronize different clients at different machines.   
An alternative approach would be to allow blocking only at the server. In principle, this works fine, but problems arise when a client crashes while its invocation is being handled by the server. We may require relatively sophisticated protocols to handle this situation, and which that may significantly affect the overall performance of remote method invocations.

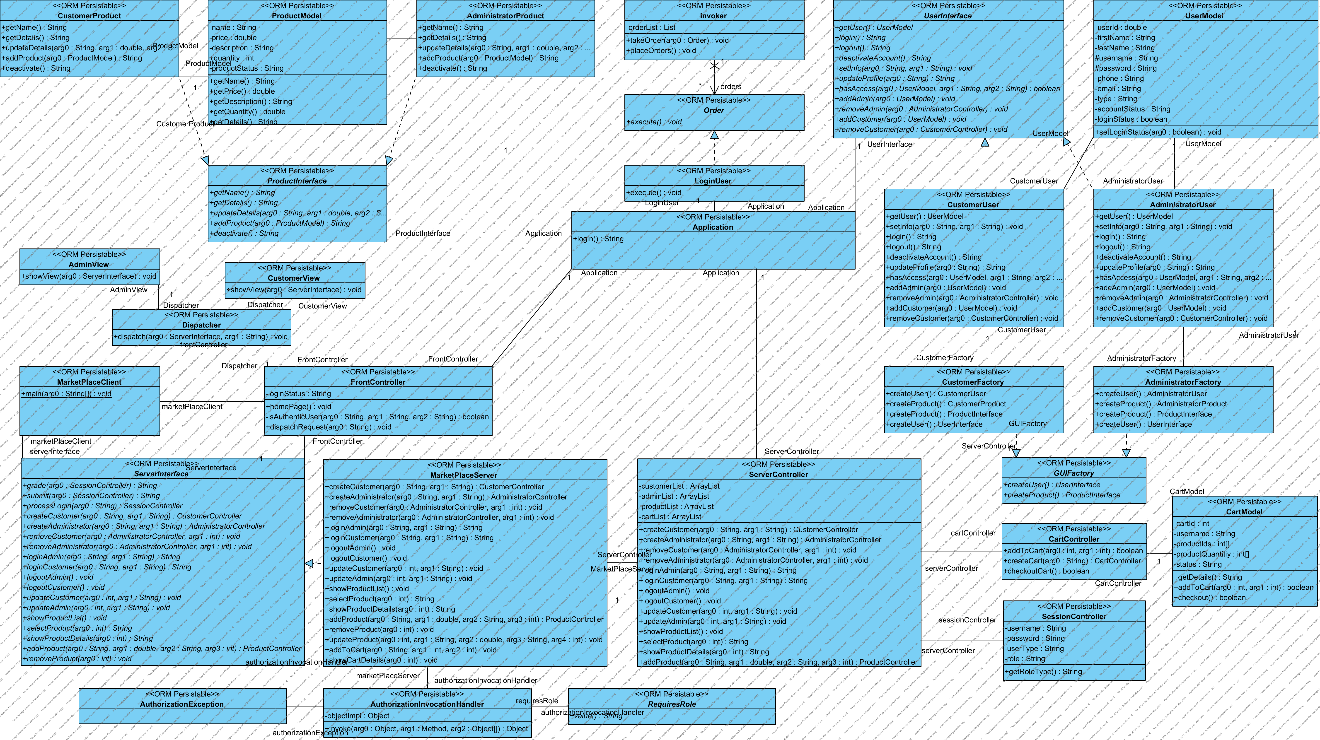
Therefore, the designers of Java RMI have chosen to restrict blocking on remote objects only to the proxies (Wollrath et al., 1996). This means that threads in the same process will be prevented from concurrently accessing the same remote object, but threads in different processes will not. Obviously, these synchronization semantics are tricky: at the syntactic level (ie, when reading source code) we may see a nice, clean design. Only when the distributed application is actually executed, unanticipated behavior may be observed that should have been dealt with at design time.

If I implement my RMI remote methods as synchronized, RMI does not provide such guarantee that they are mutually exclusive on its own (unlike EJB) and two calls on the same remote object may be executed concurrently unless you implement some synchronization. Also, if I have a method that the server executes periodically. It is used to do cleanups. I have to make sure that this particular method does not execute when there is any RMI method being run/used by remote clients. If the cleanup job is in another class, I will need to define a lock that you will share between the remote object and the cleanup job. In the remote object, define an instance variable that I will use as a lock.

# 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 some of the different new functionalities of the system:   
Add a product to customer’s cart:

public String addToCart(String user, int productId, int quantity) {

String commandStatus = null;

try{

String product\_stm = String.format("UPDATE product SET quantity = (quantity - %s) WHERE productId = %s",quantity,productId);

dbManager.updateMyRecord(product\_stm);

String cart\_stm = String.format("INSERT INTO cart(username, productId, quantity) VALUES ('%s',%s,%s)", user, quantity,productId);

dbManager.updateMyRecord(cart\_stm);

commandStatus = "Product has been added to your cart successfully!";

}catch(Exception e){

commandStatus = "Please try again later!";

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

}

return commandStatus;

}

* Browse products:

public String showProductList(){

StringBuilder str = new StringBuilder();

str.append("");

try{

ResultSet rs = dbManager.executeMyQuery("SELECT \* FROM `product` WHERE `status` = 'active'");

while (rs.next()) {

int id = rs.getInt("productId");

String productName = rs.getString("name");

str.append(id);

str.append(" ");

str.append(productName);

str.append("\n");

System.out.println("ID: " + id + "\t" + "Name: " + productName);

}

* Concurrency simulating:

**public void concurrencyTestSync(String host){**

**try{**

**System.out.println(host + "synchronized concurrencyTest call...."); Thread.sleep(6000);**

**System.out.println( host + "synchronized concurrencyTest finish....");**

**} catch(Exception e){**

**System.out.println("Exception is :" + e.getMessage());**

**}**

**} }**

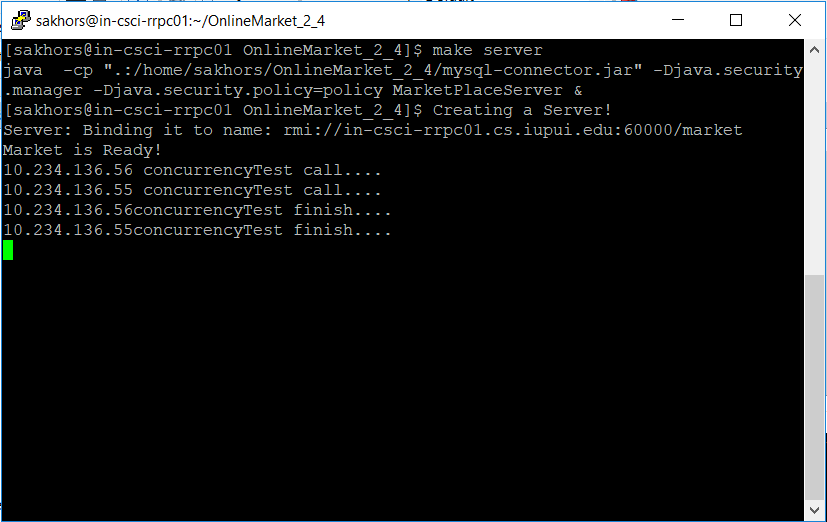
**}**

# Sample Runs

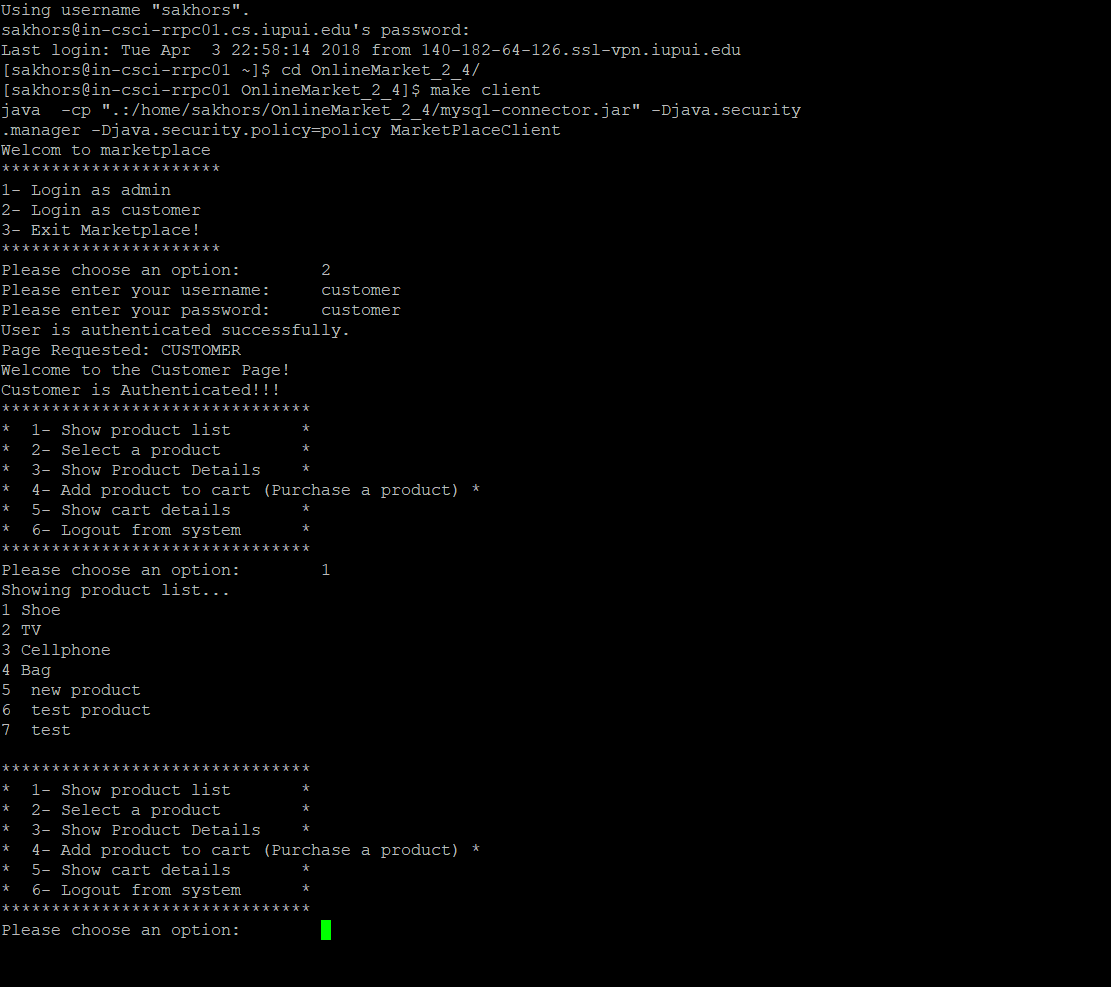
In this section you’ll see screenshots of sample runs. The system is minimal, so there is no GUI, instead every interaction happen using command prompt.

## Screenshots

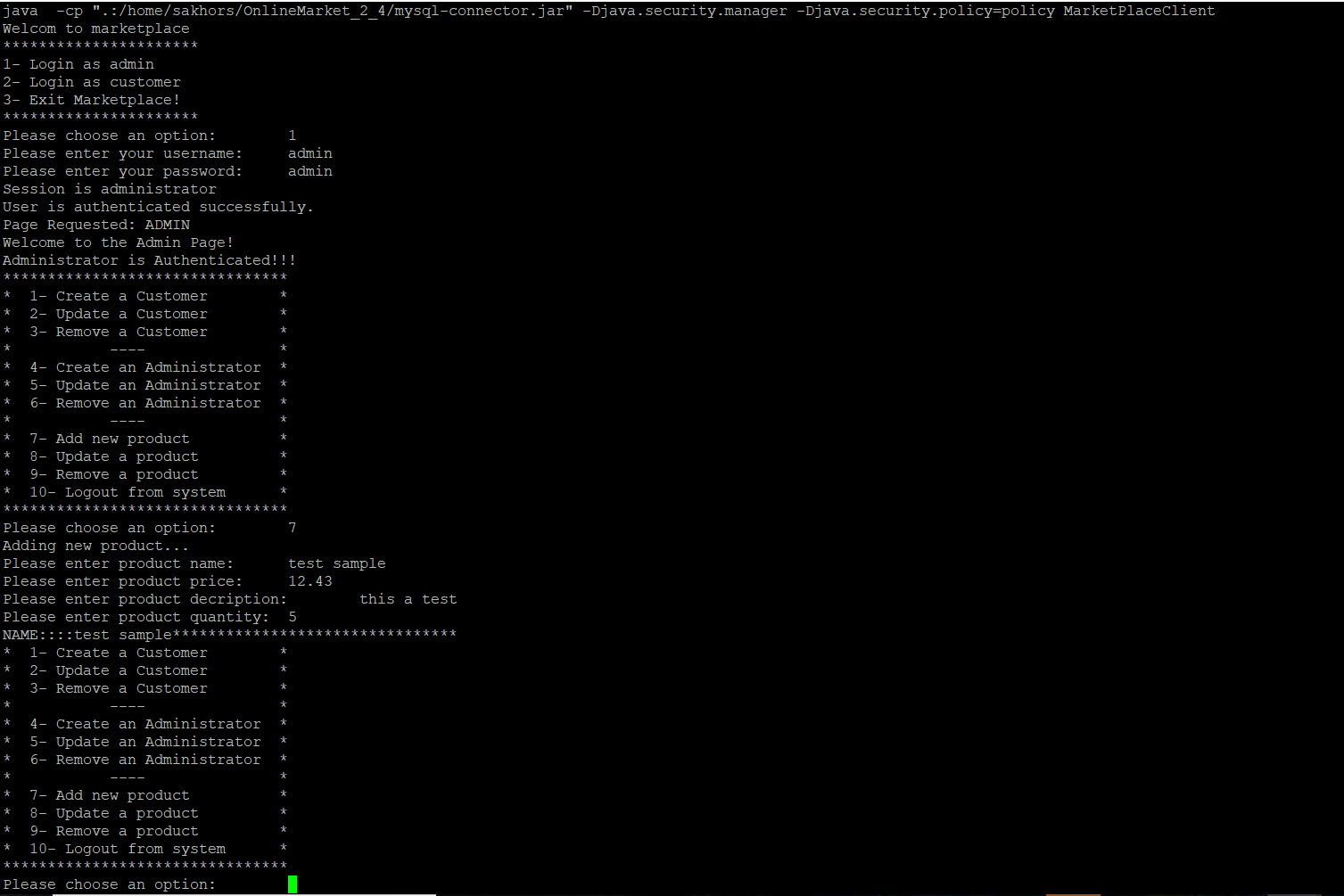
Here is a sample run of simulating concurrency using two client from two different machine:



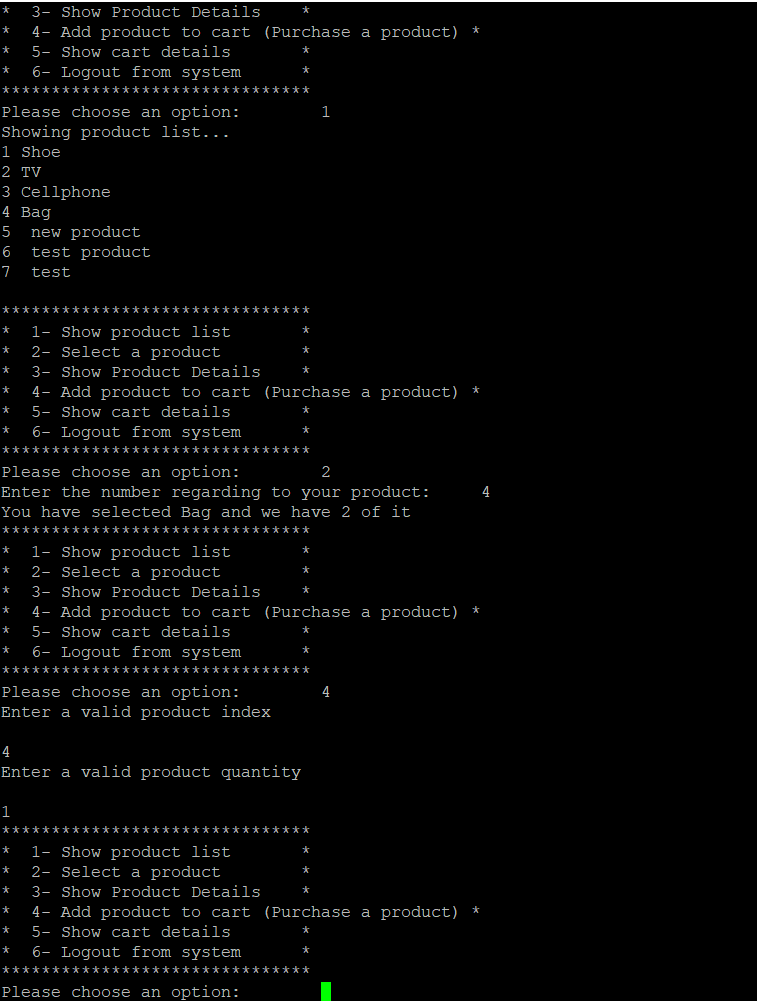
This is a sample run of browsing system products from database:



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):



## Discussion

During this assignment, we investigated the concurrency in java RMI and also implemented three different functionalities including Browse, add products and purchase 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.  
About the concurrency, we found that each client is separate than others and has its own threads.

# Conclution References

## Conclution

In conclusions, we found that java RMI has no guarantee in term of keeping our system thread safe, but we found that each client has its own thread pool and we just need to keep thread safe system for each client of RMI server.

## References

* <https://pdos.csail.mit.edu/6.824/papers/waldo-rmi.pdf>
* <https://pdfs.semanticscholar.org/9f05/b720ada20b6910fe2b1b20bd6650f93e8784.pdf>
* Java and developers forums
* Lecture slides