Market Place

Assignment Report 5

Virinchi Sainath Nalluri
vnalluri@iupui.edu
20th April, 2018

Table of Contents

Objective

The objective of this assignment is mainly to understand synchronization in Java Programming Language and ensure shared resources used different clients are being consistent with each other without any lag or performance issues. And also to discuss patterns such as Monitor Object, Future Object, Guarded Suspension, Scoped locking and thread safe interface.

Assignment Discussion

Java Synchronization Discussion

In any distributed scenarios, synchronization of code is important to maintain consistency among all systems. The synchronization in java can be achieved in many ways. One popular way to synchronize is using the *synchronized* keyword. Synchronized keyword can be used in method signature or for blocks. Synchronized methods is not recommended because, when one thread is using a synchronized method an intrinsic lock is made on the entire class. That means, if another thread is accessing another method (synchronized or not) has to wait until the first thread releases the lock on the class. As you can see why synchronized methods are less preferred way and they are blocking accessing to other methods of the same class.

Synchronized blocks are better because the code wrapped around as synchronzied block is only locked, not the entire class unlike synchronized methods. Using the synchronized block you can only put lock on an individual object. For synchronized methods and blocks, the lock is intrinsic. However, there are reentrant locks, which implements the Lock interface of java.util.concurrent.locks. The advantage is we can provide fairness parameter to these locks. That means it has a queue of threads waiting. The first to enter the queue will be given the lock next. Similarly we can make unfair locks, those that randomly picks a thread among waiting threads. Using synchronized automatically implies the unfair parameter. Moreover, using reentrantlock gives extra functionalities, for example getWaitingThreads() which returns a collection of waiting threads. Hence, reentrant locks are better than using synchronized methods and blocks.

Design Patterns

In the project, Database connection class follows a singleton pattern. Singleton class is a class of which only one instance can exist. There is one database and it has to be given only one database connection to make changes to it. Hence it is restricted to having only one instance. This

restriction can be achieved by making the constructor private and by giving it the power to create its one class. When another class depending on this class requests for an object using getInstance method, if first checks if an instance is created. If not it will create and return an instance. Here, in the getDatabaseConnector() (getInstance method), scope locking is applied. If the second thread enters the if block (when null) and waits for the synchronized block to release lock, meanwhile the first thread creates an instance, then second thread when it turn comes it doesn't know that the first thread has created an instance. So another if clause has to be placed. This is **scope locking**. Without scope locking what would happen is, two threads needing DatabaseConnection instance would request and both of them would create two different instances causing problems. Thereby by making it **Thread Safe** also. **Monitor Pattern** is when you want to synchronize a piece of code/object using that code, instead of synchronizing the entire method, (as it will put an intrinsic lock on the class). This can be achieved using reentrant locks, where the object is the lock itself, and it locks the code between lock.lock() and lock.unlock() in DAO layer service classes.. In **Guarded Suspension design pattern** can be useful in scenarios where the thread has to wait until a pre condition is met. Lock will be given to this thread only when this precondition is met. Until then it has to be waiting and only proceed when the pre condition is met. The use synchronzied keyword in DatabaseConnection or the locks in ItemDaoService, CustomerDaoService, AdminDaoService classes reflect this pattern automatically. When you are using locks the precondition it is waiting for is access to the lock. **Future pattern** is implemented when the thread return value can be used later in the future. It follows the policy, fire and forget. We simple can fire the thread to do a job and the return will come back to you later rather than wait for computation by keeping other threads waiting. For example,

```
Callable<Boolean> callableTask = new Callable<Boolean>() {

    @Override
    public Boolean call() throws Exception {
        return customerDaoService.purchaseItemForCustomer(customerId, i.getId());
    }
};
Future<Boolean> future = executorService.submit(callableTask);
try {
    if (future.get()){ //try-catch blocks for future.get()
        receiptMap.put(i, true);
        price = price + i.getPrice();
    }
    else {
        receiptMap.put(i, false);
    }
} catch (InterruptedException e) {
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
}
```

In the above code, I am making threads using executors and making a callable task who job is to purchase an Item (one) by the customer's id. Here as soon as it is submitted (fired), the future return boolean is saved into future of type Future<Boolean> and was forgotten. Later when it is needed to get that return value, we use future.get() and do the following computation. The above code is used for purchasing products (each thread for each product for speed) of a customer.

Database Schema

Customer Table : customer_id, firstname, lastname, username, password

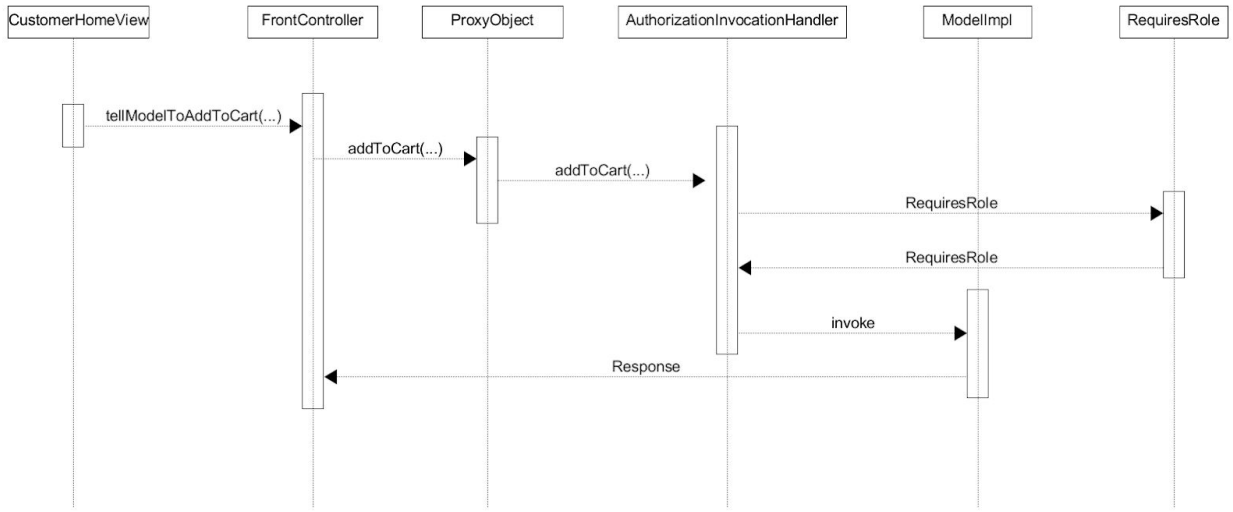
Admin Table: admin_id, firstname, lastname, username, password

Item Table: item_id, type, description, price, quantity

Cart Table: id, customer_id, item_id

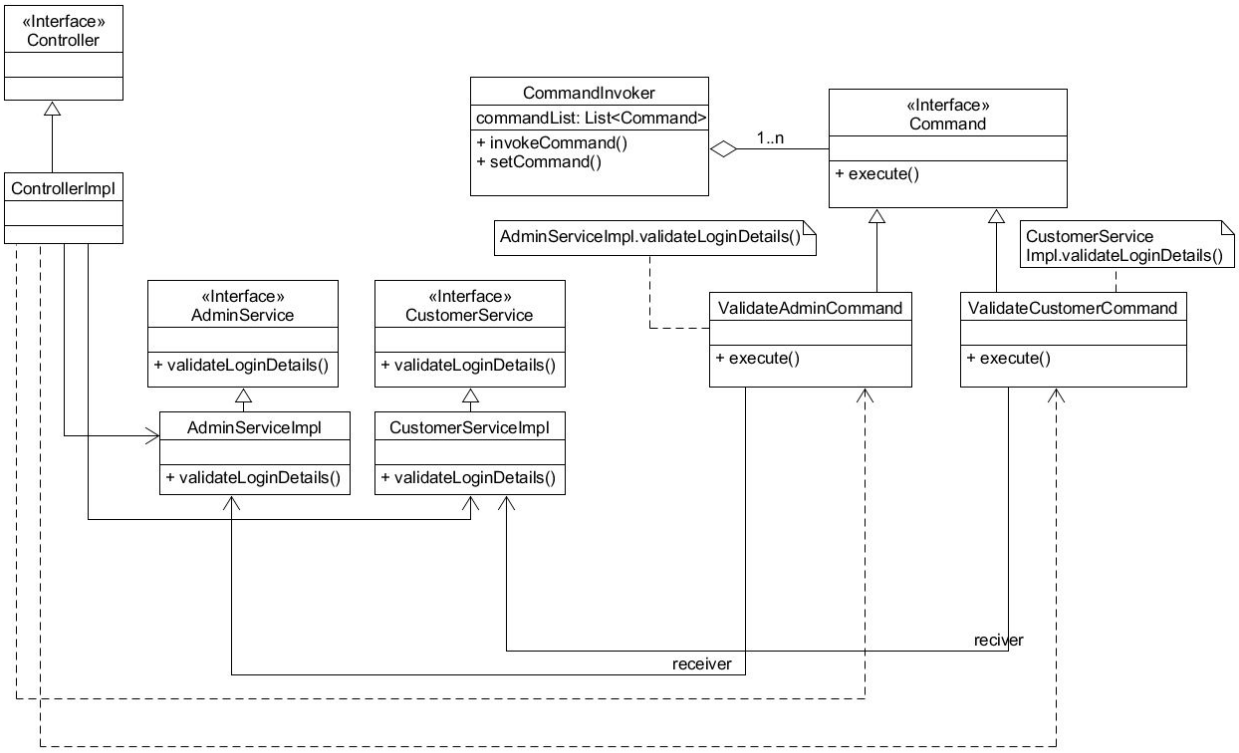
Here, the cart table has only customer_id and item_id, as everytime, the customer logs in, the cart of the customer is computed from this table. For computing the cart, find the number of customer_id, item_id pairs and add the item_id pairs to the cart. Everytime there is a change in the cart, the changes are persisted in this table.

UML Sequence Diagram

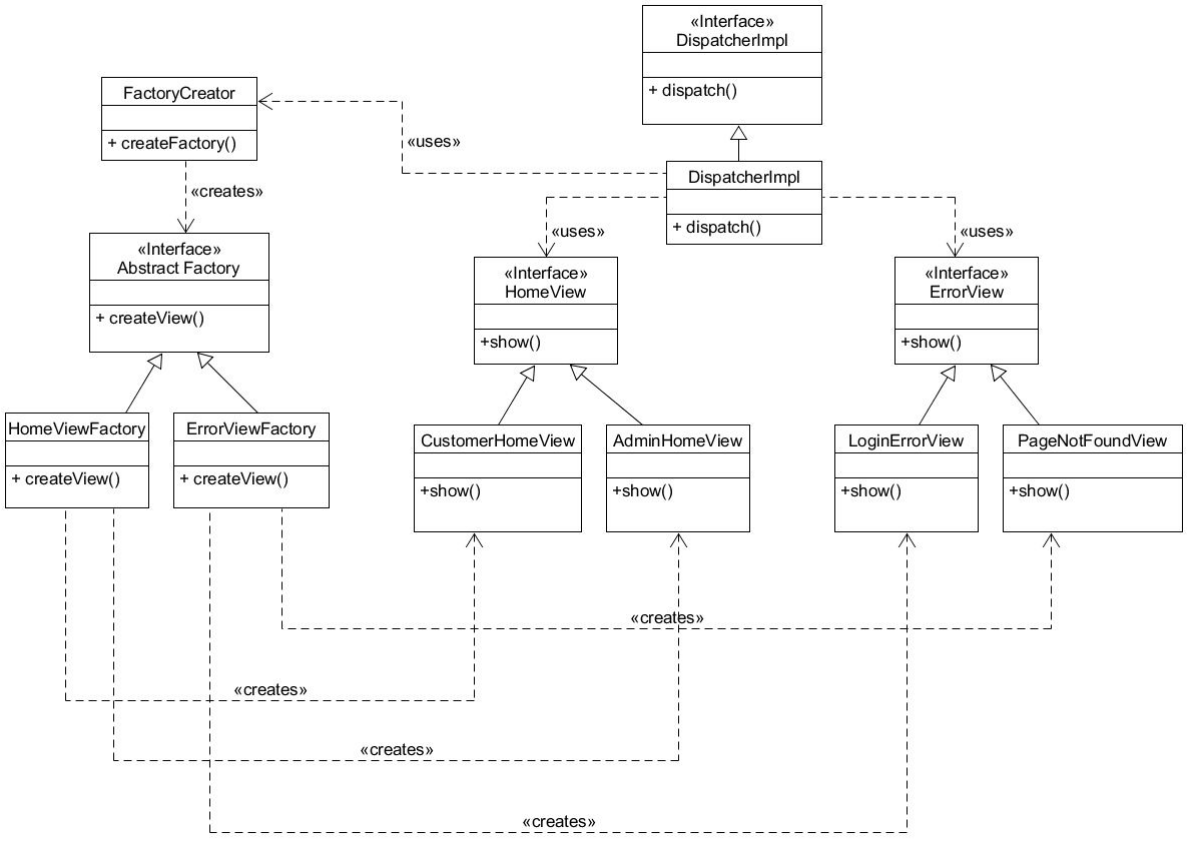


Other diagrams such as class diagram and domain model diagram are still valid.

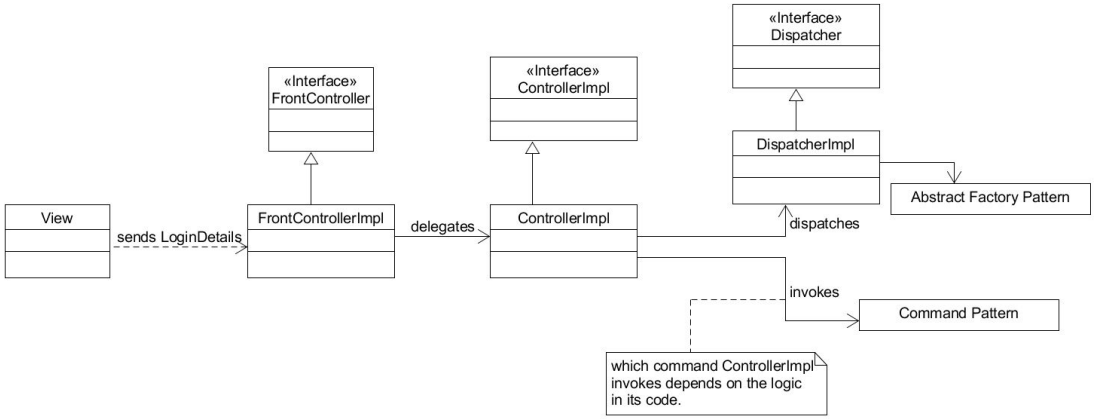
UML Class Diagram for Command Pattern



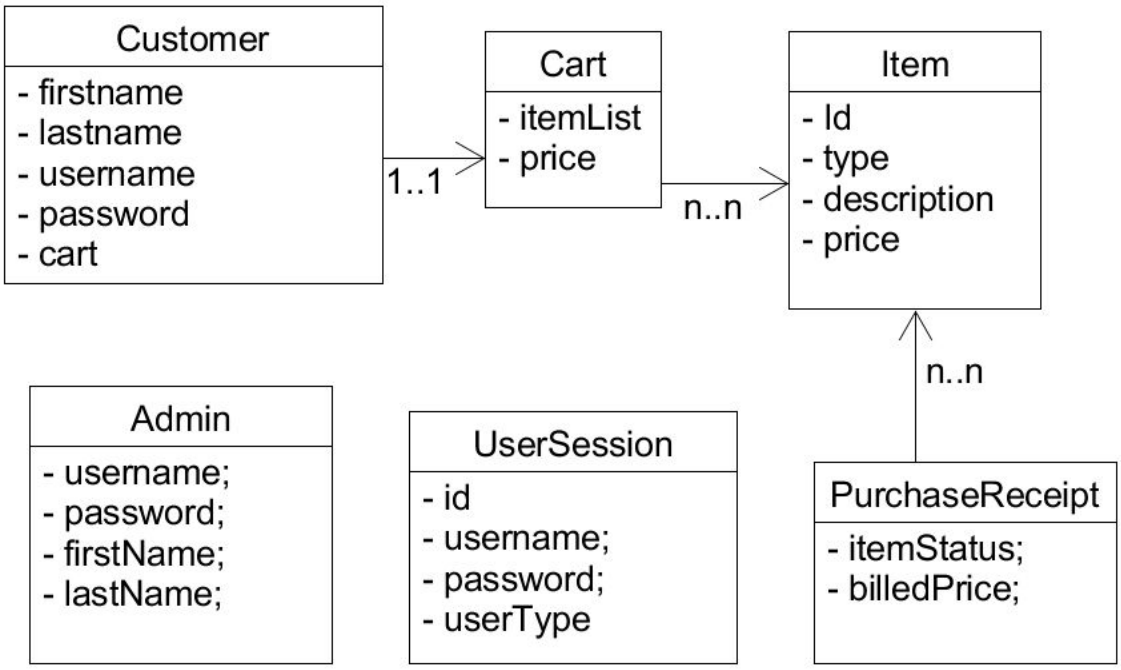
UML Diagram for Abstract Factory



UML Class Diagram for Front Controller



UML Domain Model (Updated)



Transition From Assignment 4 to Assignment 5

New Classes Created

DatabaseConnector: Has the code for connecting to MySQL and serving requests to whoever needs an instance, in a synchronized way.

BaseDaoService: Has the basic code for getting an instance of DatabaseConnection and initializing common variables.

CustomerDaoService: Extends BaseDaoService and has the MySQL JDBC code to make CRUD on customer, customer_item table in database.

AdminDaoService: Extends BaseDaoService and has the MySQL JDBC code to make CRUD on admin table in database.

ItemDaoService: Extends BaseDaoService and has the MySQL JDBC code to make CRUD on item table in database.

Removed

Product and Order from Entities. Removed NewProductView as I am taking new product details from the admin view itself.

Comments from Assignment 4

1. Scope: Made changes to code according giving correct scope.

Addressed in the code.

2. *"As a customer, I was unable to purchase an item I added as an admin. I got a message that I didn't have permission Everything else worked fine."*

Now it fixed with full implementation.

3. Missing A3 branch

Created Assignment 3 branch.

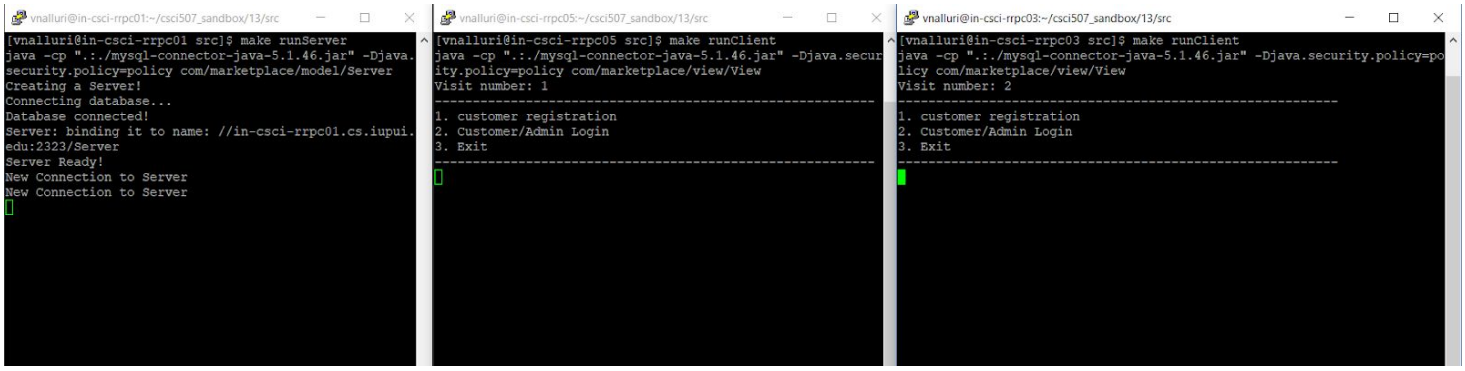
4. What is this guarantee telling us about Java Threads? About Java RMI?

Guarantee: There is a need or no need for Java RMI to create a new thread for every customers' every method invocation. It may or may not use the already existing threads to do these method invocations. The Java RMI doesn't "guarantee" that a new thread is created or how the mapping of thread to method invocation is done. However, it does say that if the remote method implementation is made thread safe, then every client can execute his remote method Invocation.

Java RMI: The server class which has methods which are to be remotely accessed is first bound to a string and added to the registry. The client then looks up for this string from registry and get this remote object. The client then accesses this remote object as if it is any other object. However, (internally which even the client even doesn't have to know) when the object's method is called a stub in invoked which wires this method to the method in the server. The stub sends the arguments and gets back the return object (must be serializable to be sent over to the client side). This is how remote method invocation works in Java RMI.

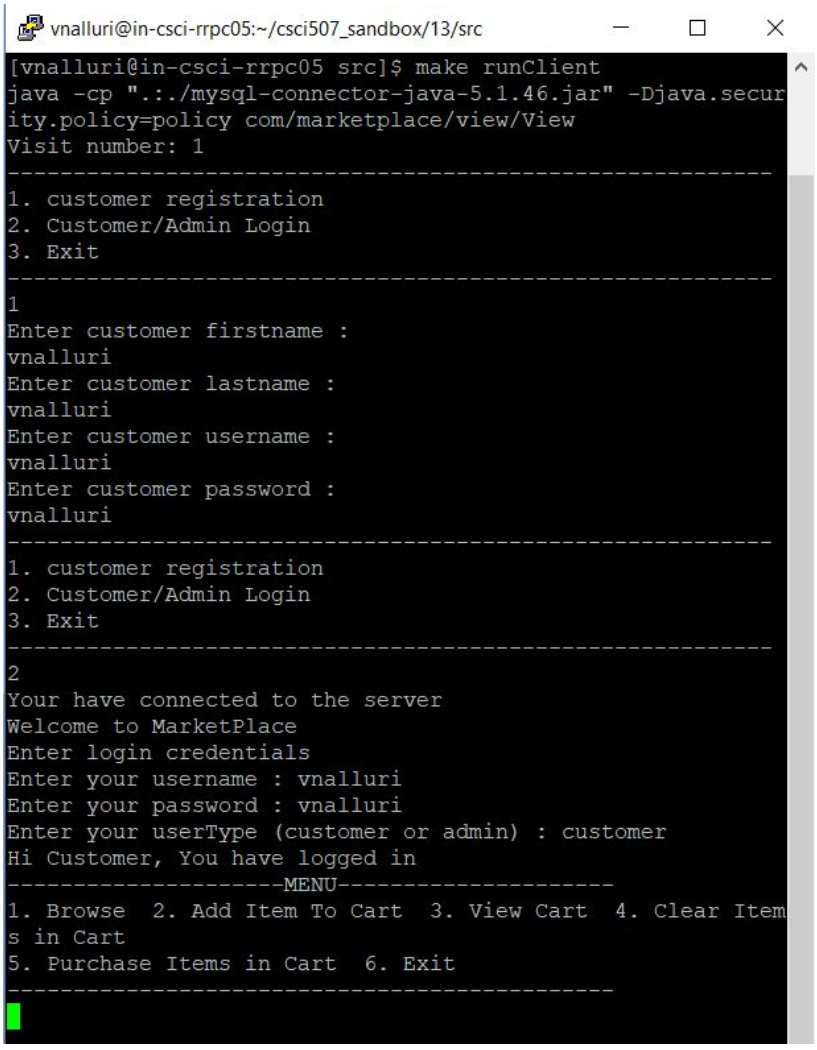
Sample Runs

Initially, running server and two clients

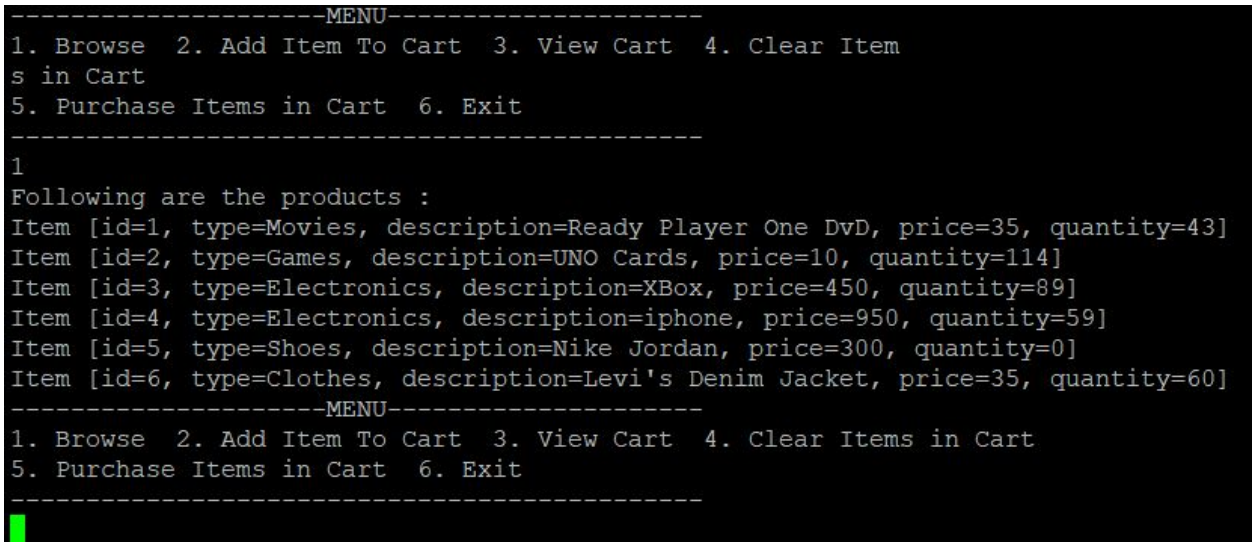


Customer Registration and Login

First I registered as vnalluri and logged in as vnalluri and got success message.

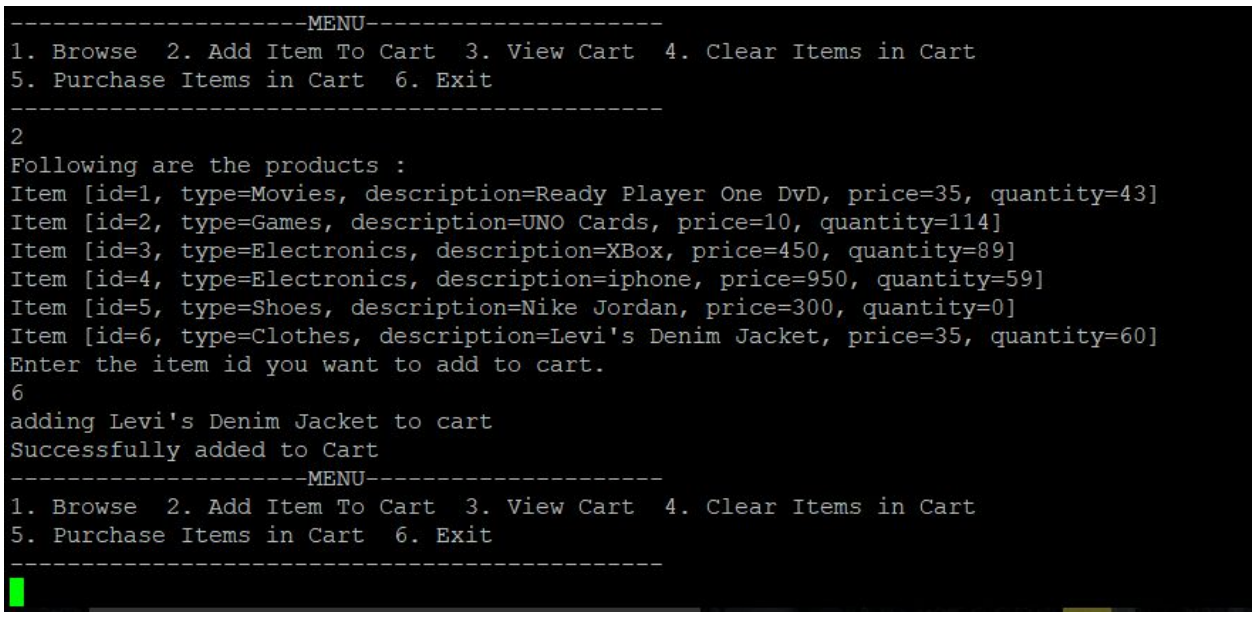


Customer Browse



Customer Add Item To Cart

On selecting add item to cart (2) , a list is printed to pick id. On picking 6, it is added to cart.



Customer View Cart (after adding the above item - 6 to cart)

```
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
3
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Total Price of Cart = 35
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
3
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Total Price of Cart = 35
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
```

Clear Cart

First on view cart (3), shows two items, after clearing cart (4), no items in cart.

```
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
3
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=89]
Total Price of Cart = 485
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
4
Cleared Cart
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
3
No items in cart
Total Price of Cart = 0
--------------------MENU--------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
----------------------------------------------
```

Purchase Items

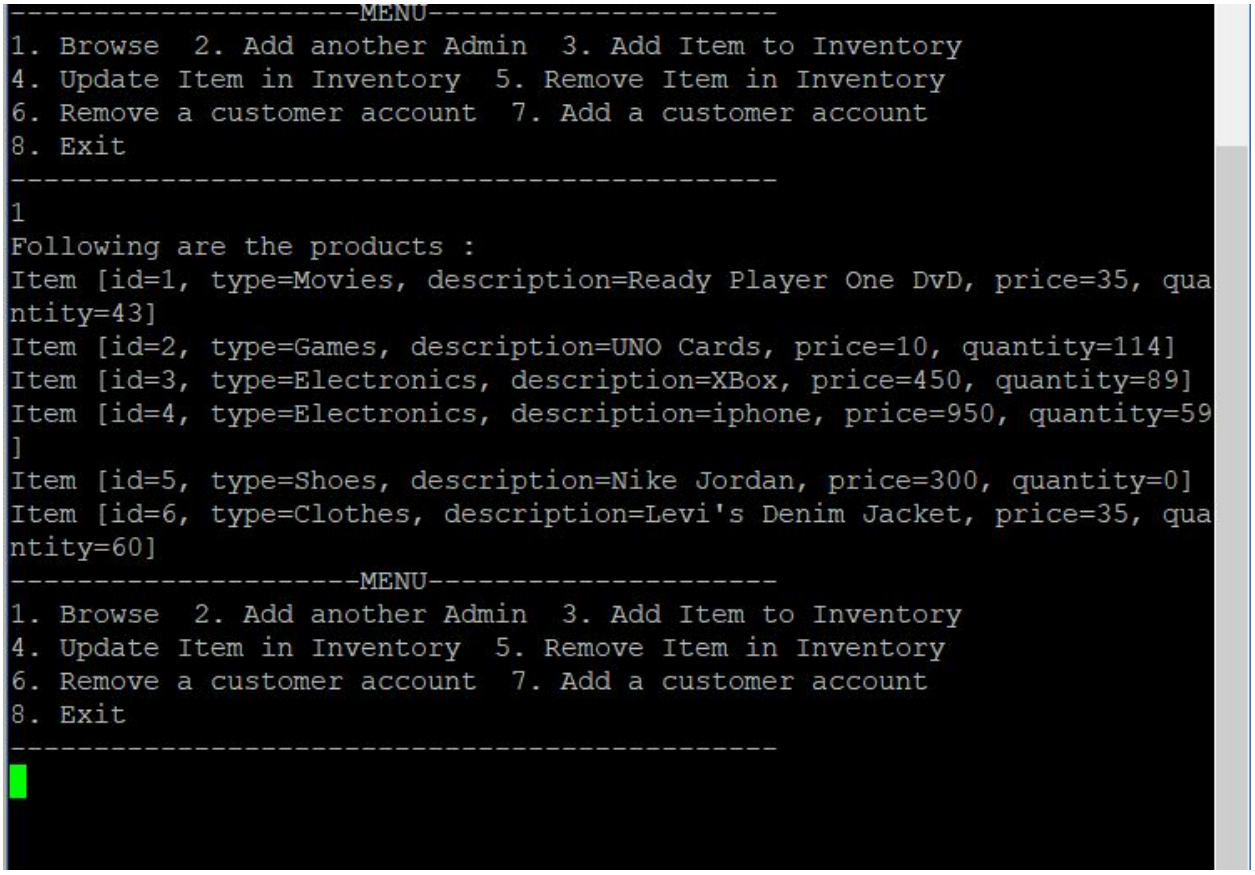
Showing failed and successfully purchased items.

```
----------------------MENU----------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
------------------------------------------------
3
Item [id=3, type=Electronics, description=XBox, price=450, quantity=89]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=1]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=1]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=114]
Total Price of Cart = 1060
----------------------MENU----------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
------------------------------------------------
5
Your receipt:
Item: UNO Cards - Successful
Item: Nike Jordan - Failed
Item: Nike Jordan - Successful
Item: XBox - Successful
Price Billed = 760
----------------------MENU----------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
------------------------------------------------
```

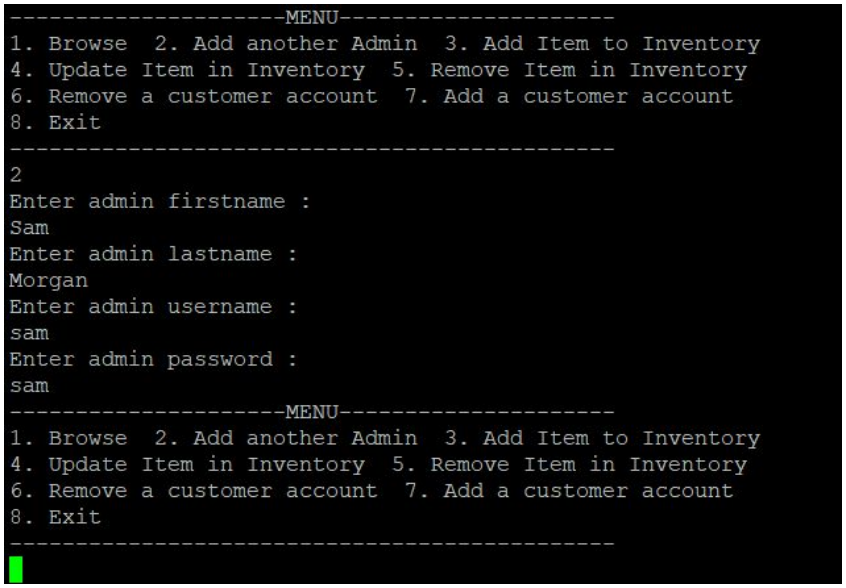
Trying to add item whose quantity is 0

```
----------------------MENU----------------------
1. Browse  2. Add Item To Cart  3. View Cart  4. Clear Items in Cart
5. Purchase Items in Cart  6. Exit
------------------------------------------------
2
Following are the products :
Item [id=1, type=Movies, description=Ready Player One DvD, price=35, quantity=43]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=114]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=89]
Item [id=4, type=Electronics, description=iphone, price=950, quantity=59]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=0]
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Enter the item id you want to add to cart.
5
Add to Cart - Not Successful
```

Admin Browse



Admin Add Another Admin



Add Item To Inventory

```
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
3
Enter product type :
Sports
Enter product description :
Basketball
Enter product price :
19
Enter product quantity :
50
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
1
Following are the products :
Item [id=1, type=Movies, description=Ready Player One DvD, price=35, quantity=43]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=113]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=88]
Item [id=4, type=Electronics, description=iphone, price=950, quantity=59]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=0]
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Item [id=7, type=Sports, description=Basketball, price=19, quantity=50]
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
```

Update item in inventory (updating the just added item)

```
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
4
Following are the products in the inventory:
Item [id=1, type=Movies, description=Ready Player One DvD, price=35, quantity=43]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=113]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=88]
Item [id=4, type=Electronics, description=iphone, price=950, quantity=59]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=0]
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Item [id=7, type=Sports, description=Basketball, price=19, quantity=50]
Enter the id of the item you want to update
7
Enter the updated item's type
Outdoor Sports
Enter the updated item's description
Basketball and Ring
Enter the updated item's price
40
Enter the updated item's quantity
35
Product successfully added to Inventory
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
1
Following are the products :
Item [id=1, type=Movies, description=Ready Player One DvD, price=35, quantity=43]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=113]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=88]
Item [id=4, type=Electronics, description=iphone, price=950, quantity=59]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=0]
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Item [id=7, type=Outdoor Sports, description=Basketball and Ring, price=40, quantity=35]
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
```

Remove Item in Inventory (removing the just updated item)

```
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
5
Following are the products in the inventory:
Item [id=1, type=Movies, description=Ready Player One DvD, price=35, quantity=43]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=113]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=88]
Item [id=4, type=Electronics, description=iphone, price=950, quantity=59]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=0]
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
Item [id=7, type=Outdoor Sports, description=Basketball and Ring, price=40, quantity=35]
Enter the id of the item you want to remove
7
Successfully removed item
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
1
Following are the products :
Item [id=1, type=Movies, description=Ready Player One DvD, price=35, quantity=43]
Item [id=2, type=Games, description=UNO Cards, price=10, quantity=113]
Item [id=3, type=Electronics, description=XBox, price=450, quantity=88]
Item [id=4, type=Electronics, description=iphone, price=950, quantity=59]
Item [id=5, type=Shoes, description=Nike Jordan, price=300, quantity=0]
Item [id=6, type=Clothes, description=Levi's Denim Jacket, price=35, quantity=60]
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
```

Remove customer account

```
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
6
Enter the username of the customer to delete
bill
successfully removed customer
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
```

Add a customer account

```
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
7
Enter customer firstname :
Bill
Enter customer lastname :
Freeman
Enter customer username :
bill
Enter customer password :
bill
customer account created
----------------------MENU----------------------
1. Browse  2. Add another Admin  3. Add Item to Inventory
4. Update Item in Inventory  5. Remove Item in Inventory
6. Remove a customer account  7. Add a customer account
8. Exit
------------------------------------------------
```

Conclusion And Analysis

Discussion is done on synchronization in Java and how design patterns such as Monitor Object, Future, Guarded Suspension, Scope Locking and Thread-Safe Interface. Also how sample runs of different funcitonality of the application in both the customer and admin's perspective.

References

1. For writing make file : https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
2. Java RMI : Bank example in Canvas
3. MVC Pattern : Calculator Example in Canvas
4. Front Controller Pattern : Front Pattern Example in Canvas
5. Abstract Factory Pattern : Abstract Pattern Bank Example in slides.
6. Command Pattern: Command Pattern example in Canvas.
7. Reflection Pattern, Proxy Pattern : example from Canvas.
8. Authorization Pattern, RequiresRole annotation, AuthorizationInvocationHandler : from example in Canvas.
9. Thread-Safe, Scoped Locking — http://www.cs.wustl.edu/~schmidt/PDF/locking-patterns.pdf
10. Future — https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/Future.html
11. Guarded Suspension— https://en.wikipedia.org/wiki/Guarded_suspension
12. Technologies Used : Eclipse IDE, WinScp, PUTTY, Tesla, Java 8