
ONLINE MARKETPLACE APPLICATION

ASSIGNMENT-5

PROJECT REPORT

**IMPLEMENTING FULL APP FUNCTIONALITY & EXAMINING SYNCHRONIZATION
PATTERNS**

UNDER GUIDANCE OF

Dr. RYAN RYBARCZYK

BY

ADITHYA MORAMPUDI

Table of Contents

| | |
|------------------------------------|----------|
| ASSIGNMENT 5 REPORT | 3 |
| Assignment#4 Feedback | 3 |
| Introduction | 3 |
| Monitor Object Pattern | 3 |
| Future Pattern..... | 4 |
| Guarded Suspension | 4 |
| Scoped Locking..... | 4 |
| Thread Safe Interface | 4 |
| Java Synchronization | 5 |
| Handling Concurrent Requests | 6 |
| Newly Created Classes..... | 8 |
| Sample Runs | 8 |
| Conclusion | 13 |
| References | 13 |

ASSIGNMENT 5 REPORT

Assignment#4 Feedback

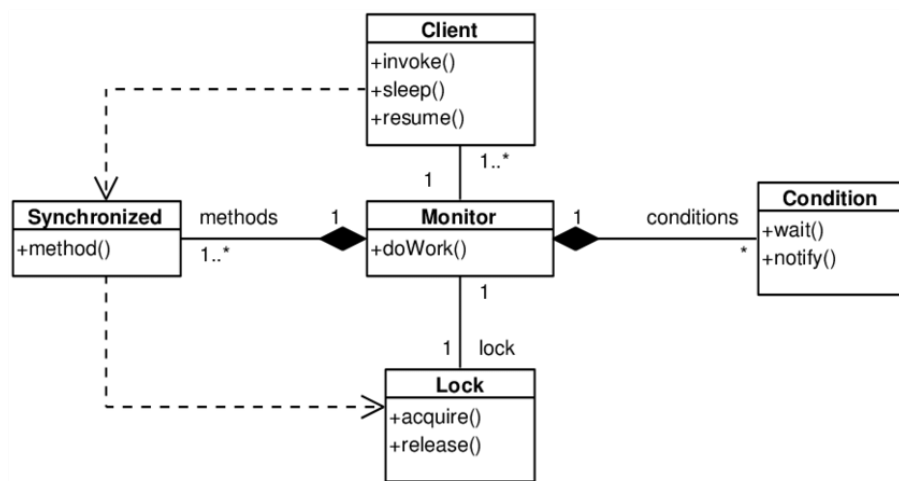
- Added the scopes to the variables, also removed redundant packages.

Introduction

In this Assignment, we are asked to fully implement the application functionality and examine how java implements synchronization and concurrency patterns, in this assignment I have fully implemented all the application functionality and made the system to efficiently handle concurrent requests at the same time

Monitor Object Pattern

The main concept behind Monitor object pattern is, a Monitor is a **Synchronization** construct that gives the flexibility of both Mutual Exclusion and the ability to wait in a Multithreaded Environment. This also gives the flexibility to notify other threads when a condition is met. Monitor object pattern essentially provides the mechanism to Monitor your object and make sure that only a method is being executed at any instance by acquiring locks to the critical section, which means when one thread has acquired this lock, no other thread can get the lock. This pattern provides a mechanism to acquire the lock and after the execution of the method or block, it should release the lock. So, when a condition is met, we need to acquire a lock on the critical section and after the execution we need to release it, in this assignment, I have implemented the monitor object pattern by **Synchronizing** the blocks of code, where I think the code is a critical section. I have implemented synchronization by using the inbuilt keyword **Synchronized** in java. Following is the pictorial representation of the pattern.



Fig[1]-Monitor Object Pattern.

Future Pattern

Future pattern essentially provides a virtual object when we are calling a method which is going to take a large amount of computational time in the program, so for example in this application whenever a user is trying to purchase items in a cart, that can take a significant amount of time, so at this point the **Synchronized** keyword provides us with a virtual object, so when a client tries to invoke field which is still under computation, Future blocks the client and makes it wait until the field is being calculated. Also with the help of virtual object which is returned to the client, the client can keep track of the execution with the help of virtual object and the client can use this value once it is computed. In my application, I have used the **Synchronized** keyword to implement this pattern, this keyword automatically provides with a virtual object if necessary and blocks the client till the execution is done.

Guarded Suspension

Guarded suspension states that to acquire a lock on an object, a precondition has to be satisfied before acquiring the lock and the operation can be executed, this pattern blocks the method, so we need to use this only when we know that a method is going to be blocked for a finite amount of time, else there will be a deadlock. If the precondition is not met, then client cannot access the method and it goes into wait() state. If after a point of time the precondition is met, then it notifies the thread and makes the first thread in the queue access this method. I have implemented this by using the **Synchronized** keyword.

Scoped Locking

The Scoped Locking ensures that a lock is acquired only when the control enters a scope, and lock is released if the control exits the scope of the block. I have implemented this kind of locking when accessing the executeQuery(String query) in my Database access layer, when a method is calling this method, I am only synchronizing this block in the method. When the control returns from the executeQuery(String query) method the lock is released. This way we don't necessarily block the entire code, by just blocking the required scope.

Thread Safe Interface

Thread safe interface design minimizes locking overhead. This ensures that intra-component method calls do not incur deadlock by trying to acquire a lock on the already acquired lock. For example, in my database layer, I have a function which is being called by several other methods, to make this thread safe, I am making the block synchronized at the calling level, which in our case is called the thread safe, by this what I mean is I am not blocking the code at the implementation level again, since I am already blocking the method at the interface level, where the client is calling the interface. I have implemented the Thread safe interface by using the **Synchronized** keyword at the interface level, by interface level I mean, I am blocking the client at the calling level rather than the implementation level. By this we can get better performance. If we can place the synchronized keyword at proper place then Java implements all these patterns itself, we don't have to do it manually.

Java Synchronization

In a Multithreaded environment, Threads communicate by sharing memory i.e they communicate by primarily sharing access to the fields and the object references fields refer to, with this form of communication the we make the programs extremely efficient, but only with the harm of having many errors. There may possibly be two kinds of errors thread interference and memory consistency errors. Which means two or more threads may try to access the same resource at the same time, which leads to inconsistency of the data which is being updated or inserted, in our case if two threads try to access the same object in the database, then there may be errors in updation of the data.

Synchronization is the keyword given by Java to avoid the above mentioned errors, but if we use the synchronization at all points without carefully given a thought, then we may run into deadlocks or livelocks, for example when two threads are going into a circular wait then we may end up getting nothing and essentially the system goes to deadlock and has to be terminated.

Java has Executor framework, which deals with the implementation of multithreaded programs and it deals with the patterns related to synchronization and concurrency in java.

With the help of Synchronization blocks or methods, java gives us a mechanism to lock the critical section and it only provides access to only one thread at a time, restricting access to other threads. The synchronized keyword provides us with different patterns such as guarded suspension, Future, Scoped Locking. But it also makes other threads into starvation, if there are many greedy threads in the system, then they may not allow access to the low priority threads, which makes the low priority threads to starve, we need to be careful in considering these conditions while working with synchronization in java.

This is a small piece of code from the oracle website to demonstrate how java provides synchronized blocks to restrict the access to multiple threads to a common shared resource.

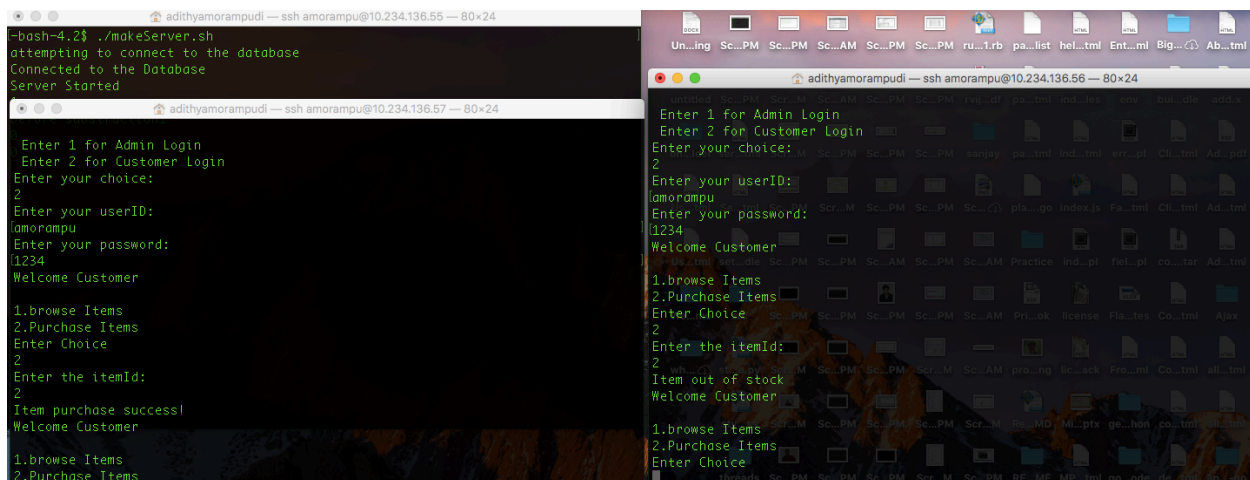
```
public void addName(String name) {  
    synchronized(this) {  
        lastName = name;  
        nameCount++;  
    }  
    nameList.add(name);  
}
```

In the above code we can see that we have two variables `lastName` and `nameCount` which are updated each time it is called, so if two threads are calling this method concurrently, then there is a high probability that the two threads might overwrite the `lastName` and `nameCount++` variable which is not desirable, to avoid this we are using the synchronized block at a scope which means the block of code which is synchronized is only accessible to a single thread at one time.

Handling Concurrent Requests

A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread.

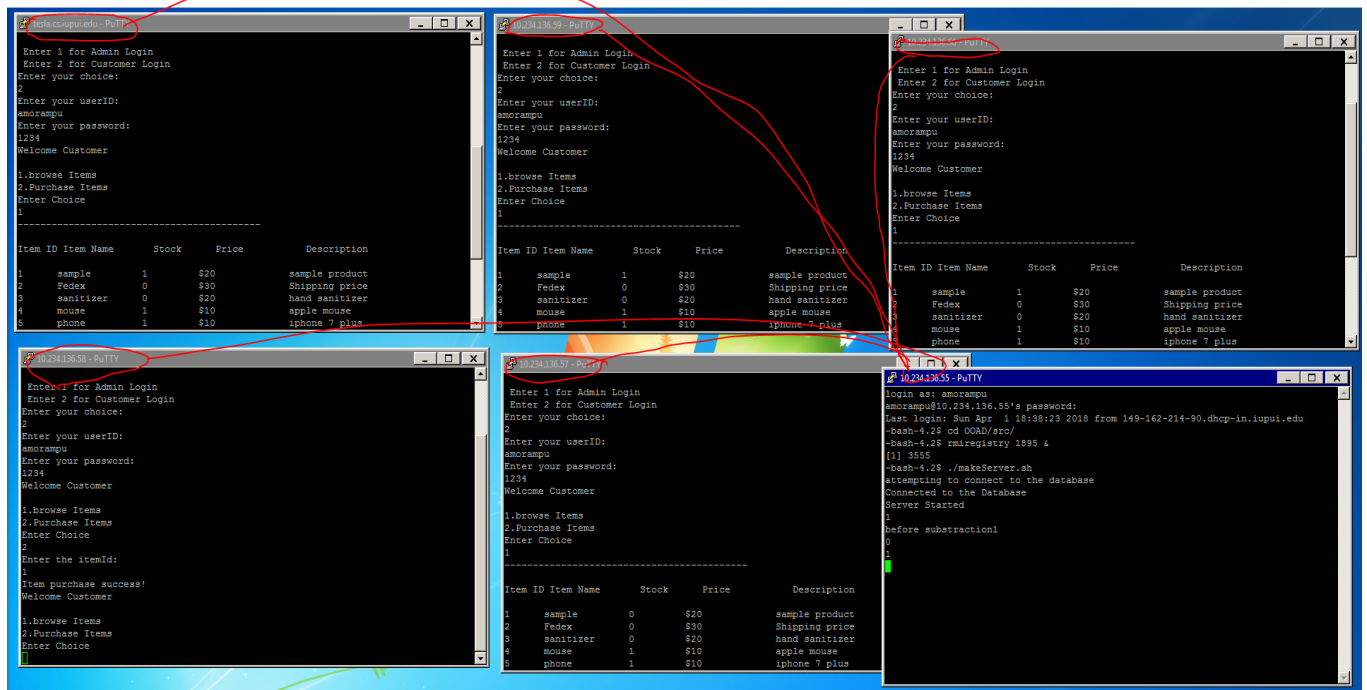
The main purpose of this assignment is to examine Thread usage in Remote Method Invocation within the scope of our application, for this purpose, I have implemented the functions to be thread safe by using the keyword synchronized, with the help of this keyword, the access to the function is synchronized, 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



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.



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.

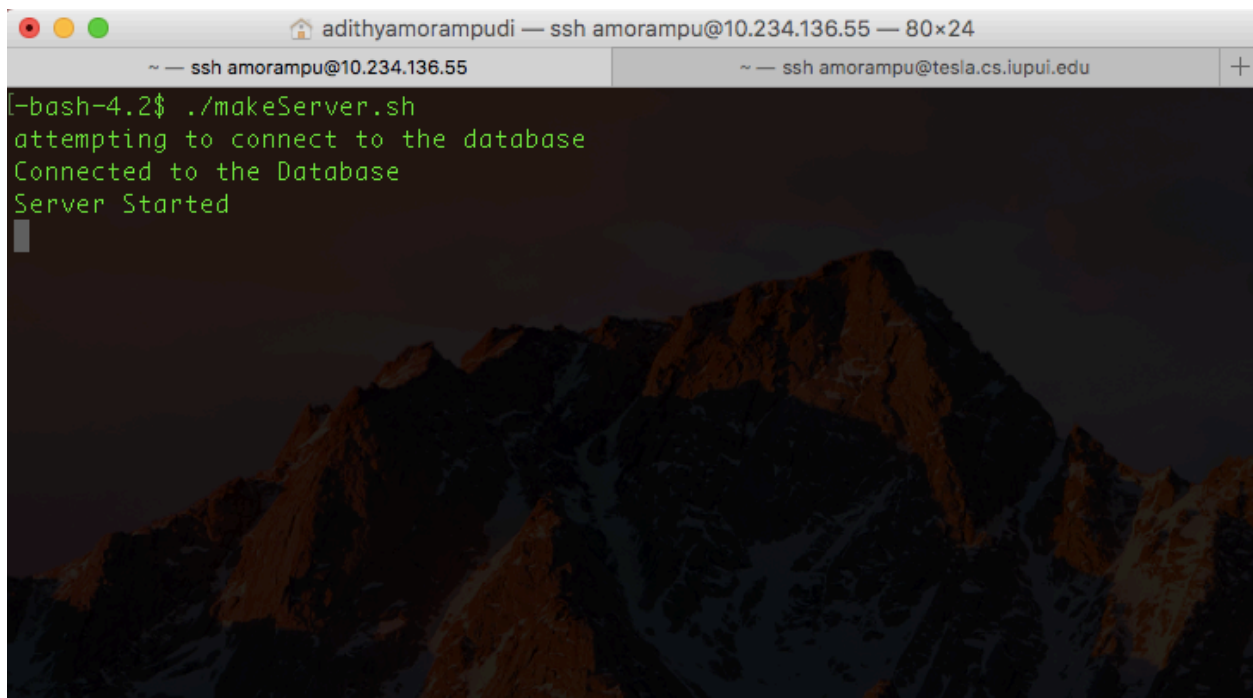
So, from all the perspectives after running the application, I came to a conclusion that java RMI is multithreaded and allows concurrent access to resources, but the point is we need to be aware of the underlying concepts behind how to allow concurrent access to the application, we need to use the synchronized 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

Newly Created Classes

There are many newly created classes in this assignment, I have created a new concrete factory to produce the view when a new user is registered, I have created concrete commands to implement the complete application functionality. I have implemented the complete functionality of the application, you can find the screenshots below.

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.

A screenshot of a terminal window with a dark background and a mountain landscape wallpaper. The window title bar shows 'adithyamorampudi — ssh amorampu@10.234.136.55 — 80x24'. The terminal content shows the execution of a script: '[~bash-4.2\$./makeServer.sh', followed by 'attempting to connect to the database' in green, 'Connected to the Database' in green, and 'Server Started' in green. The prompt returns to '[~bash-4.2\$'.

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.


```

adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 80x24
~ — ssh amorampu@10.234.136.55
^C-bash-4.2$ ./makeClient.sh
welcome to RMI Username: amorampu Password: Port: Quickconnect
Enter 1 for Admin Login
Enter 2 for Customer Login
Enter your choice: 2
Retrieving directory listing of /home/amorampu/OOAD/src/.git/logs/refs/heads/master
File transfer successful, transferred 41 bytes in 1 second
File transfer successful, transferred 1,066 bytes in 1 second
Enter your userID:
[amorampu]adithyamorampudi/Documents/Courses/OOAD/Assignments/A1/
Enter your password:
1234
Welcome Customer
csci50700_spring2017_marketplace-Assignment-3
csci50700_spring2017_marketplace-Assignment-4
csci50700_spring2017_marketplace-master
1.browse Items
2.Purchase Items
Enter Choice
1
----- Server -----
Software Quality Assurance
Item ID Item Name Stock Filetype Price Last modified Description
1 sample 2 Directory $20 04/01/2018 17:3... sample product
2 Fedex 1 Directory $30 04/01/2018 17:3... Shipping price
3 sanitizer 0 Directory $20 03/06/2018 16:0... 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.

```

adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 80x24
~ — ssh amorampu@10.234.136.55
1.browse Items
2.Purchase Items
Enter Choice
^C-bash-4.2$ ./makeClient.sh
[-bash-4.2$ ./makeClient.sh
welcome to RMI Username: amorampu Password: Port: Quickconnect
Enter 1 for Admin Login
Enter 2 for Customer Login
Enter your choice:
2
Enter your userID:
[amorampu]adithyamorampudi/Documents/Courses/OOAD/Assignments/A1/
Enter your password:
1234
Welcome Customer
csci50700_spring2017_marketplace-Assignment-3
csci50700_spring2017_marketplace-Assignment-4
csci50700_spring2017_marketplace-master
1.browse Items
2.Purchase Items
Enter Choice
2
Enter the itemId:
1
Item purchase success!
Filesize Filetype Last modified
Directory 04/01/2018 17:3...
Directory 04/01/2018 17:3...
Directory 03/06/2018 16:0...
Directory 02/12/2018 22:3...

```

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 — 80x24
~ — ssh amorampu@10.234.136.55
Enter your userID:
admin@tesla.cs.iupui. Username: amorampu Password: Port: Quickconnect
Enter your password:
12345
Starting upload of /home/amorampu/Documents/Courses/OOAD/Assignments/A1/src/.git/logs/refs/heads/master
Listing directory listing of "/home/amorampu/OOAD/src/.git/logs/refs/heads"...
File transfer successful, transferred 41 bytes in 1 second
File transfer successful, transferred 1,066 bytes in 1 second
Welcome Admin
1.browseItems
2.Update items
3.removeItems
4.add Items
5.Purchase Items, This Method wont work in ADMIN View(RBAC Demo)
Enter Choice
4
csci50700_spring2017_marketplace-Assignment-4
Enter the details of the items to be added!
Design_Patterns
Lecture_slides
javaapplication16
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.

```

Below screenshot shows New User registration.

```

adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@10.234.136.55
pu/OOAD/src/.git/logs/refs/heads ...
.git/logs/refs/heads
-bash-4.2$ sh makeClient.sh
welcome to MarketPlace Application
s/Enters/1 for Admin Login
Enter 2 for Customer Login
Enter 3 for Customer Registration
Enter your choice:
3
Enter the First Name:
Mike
Enter the Last Name:
Tyson
Enter the User Name:
mtyson
Enter the password:
1234
success
Welcome new User

```

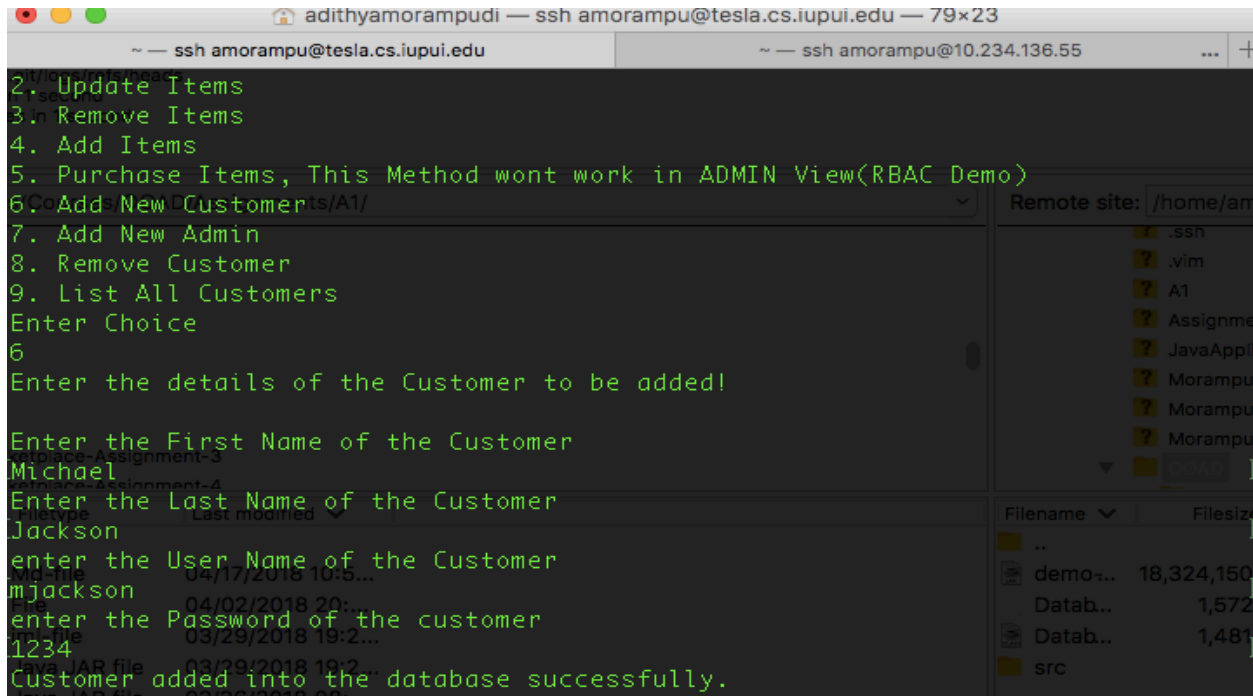
Adding items to the cart.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
~/GOAD/src/git/logs/refs/heads...
git/logs/refs/heads
7 in 1 second
phone                2                $10                iphone 7 plus
=====
8 Courses/IdACardignments/A2                $10                crimson cards
=====
9 earphones          2                $5                apple earphones
=====
10 Monitors          1                $20                sfadd
=====
12 Diamond          16                $2030             fasdf
=====
Filetype             Last modified
Enter the Id of the product to be added to the cart from above list
12
Enter the quantity of the product.
3
Item added to the cart successfully.
```

Purchasing the user cart, the user can remove the items from the cart as well.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
~/logs/refs/heads
1 second
Items present in the cart are:
ItemId  quantity
12      10
Welcome CMTYSON! Enjoy your customized portal
1. browse Items
2. Purchase Items
3. List Items in the cart
4. Add Items to cart
5. Delete From cart
Enter Choice
2
Purchased
Welcome MTYSON! Enjoy your customized portal
1. browse Items
2. Purchase Items
3. List Items in the cart
4. Add Items to cart
5. Delete From cart
Enter Choice
```

Admin adding a new customer.

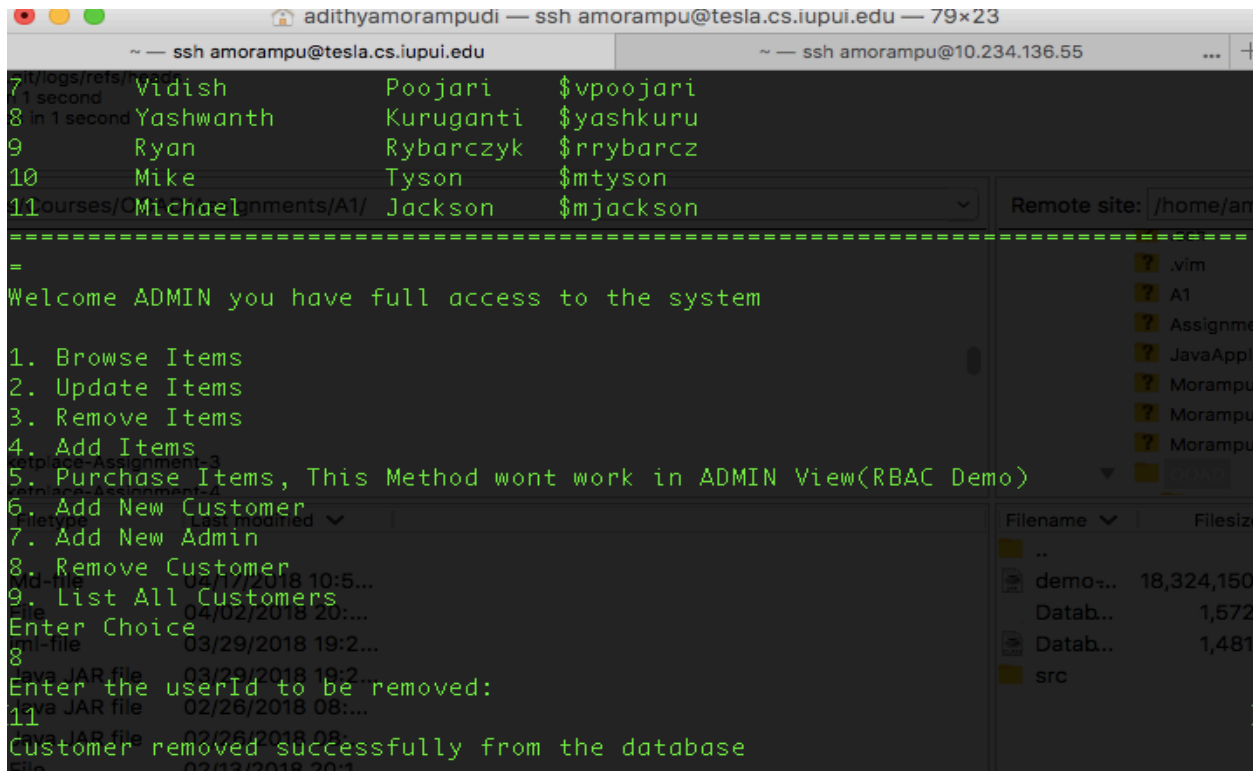


The terminal window shows a menu-driven interface for an application. The user has selected option 6, 'Add New Customer'. The interface prompts for the first name, last name, user name, and password. The user enters 'Michael', 'Jackson', 'mjackson', and '1234' respectively. A confirmation message states 'Customer added into the database successfully.' The right sidebar shows a file explorer for the remote site /home/am, listing files like .ssh, .vim, A1, Assignme, JavaAppli, Morampu, and Morampu.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
2. Update Items
3. Remove Items
4. Add Items
5. Purchase Items, This Method wont work in ADMIN View(RBAC Demo)
6. Add New Customer
7. Add New Admin
8. Remove Customer
9. List All Customers
Enter Choice
6
Enter the details of the Customer to be added!

Enter the First Name of the Customer
Michael
Enter the Last Name of the Customer
Jackson
Enter the User Name of the Customer
mjackson
Enter the Password of the customer
1234
Customer added into the database successfully.
```

Removing a customer from the database



The terminal window shows the same menu-driven interface. The user has selected option 8, 'Remove Customer'. The interface prompts for the user ID to be removed. The user enters '11'. A confirmation message states 'Customer removed successfully from the database'. The right sidebar shows the same file explorer as the previous screenshot.

```
adithyamorampudi — ssh amorampu@tesla.cs.iupui.edu — 79x23
~ — ssh amorampu@tesla.cs.iupui.edu
7. Add New Customer
8. Remove Customer
9. List All Customers
Enter Choice
8
Enter the user id to be removed:
11
Customer removed successfully from the database
```

Conclusion

After doing this assignment, I have understood the pattern related to concurrency such as Monitor object, Future, Guarded Suspension, Scoped Locking and Thread Safe Interface, I have fully implemented the application functionality.

References

- [1]. <https://stackoverflow.com/questions/3507253/java-rmi-and-synchronized-methods>
- [2]. <https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html>
- [3]. <http://www.cs.wustl.edu/~schmidt/PDF/ScopedLocking.pdf>
- [4]. <https://docs.oracle.com/javase/tutorial/rmi/overview.html>
- [5]. <https://stackoverflow.com/questions/27759566/concurrent-access-to-a-remote-object->
- [6]. <https://wiki.hsr.ch/PnProg/files/ThreadSafeInterface.pdf>
- [7]. <https://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html>