This project helped me to gain a thorough knowledge to design and synthesise reliable and secure distributed and cloud computing applications. Transactions, timing & synchronisation, coordination & consensus for developing distributed transactions and data bases are discussed in detail.

Considered an online client-server-based banking application, where customers perform transactions like balance enquiry, deposit and money transfer. A single server maintains all the accounts of customers. Multiple clients can access the server at the same time to invoke the above transactions. Designed and implemented the banking application so that all the concurrent transactions invoked by all the customers reflect correct and consistent states of account balance in all customers’ accounts.

My report contains:

**1.** Design of an effective solution to the above scenario and description of techniques used

**2.** Implementation of banking application to reflect correct and consistent states of account balance when multiple clients initiate transactions.

**3.** Testing of the developed banking application

**4.** Demonstration

**1. Design of banking application**

The objective is to build an online client-server based banking application, where customers are to perform transactions like balance enquiry, deposit, withdrawal and money transfer. We have used java as our programming language and MySQL as database to store the account details of the customers. We have created two different projects, one for the server and other for the clients and established socket type of connection between them. This helps in building an online client-server application which works on the request-reply mechanism.

The server project has the code for the server socket which keeps listening for new connection requests on the server port from the client objects, once they receive request, they create a new thread of their connection with the client. The communication and passing of information between the client and server is established by the input stream and output stream which are nothing but like a buffer which stores the passed information temporarily. The input stream of the server is the output stream of the client and vice versa. Server and client projects have pairing input and output streams to pass and receive the correct information. At first the clients will pass their username and password, server on the other side receives the data and creates the connection with the account details database and develops a result set from their user name and account no to access, modify or update their balance. Then the client pass their requests with arguments to the server, the server receives the request and arguments, the server process the client’s request and replies the client with output, server also updates to the database about the balance of the user after performing every requested transactions. We have used switch case to perform different transaction requests accordingly.

The client has two java class file, one for defining all the methods and another for the main function where the user can enter his transaction requests to the server. The java class file where we defined methods helps us to reuse the code, we have defined methods for connection to connect with the server using server port number and it also has methods for login, balance enquiry, deposit, withdrawal and transfer transaction which are used by the client objects in the main java class file, the client invokes the methods by calling it, which in turn send the requests to the server, the server receives the request and processed them using respective procedure and sends the output as reply, the client receives the reply and display the output balance after performing the every transaction. In the main java class file we create the objects of the clients and use the methods defined. As default the requests are processed serially as defined. For processing the request simultaneously of all the users we have used threads. For each user separate thread is allocated and in those threads user’s requests are specified, threads helps in sending the requests simultaneously to the server. We have used the synchronized method to clashing of the same requests from the user to the server when one is being processed, which blocks the request until the current user’s request is processed, then the latter user’s request is processed. We print the balance of the user after every transactions to keep track of the user’s balance and verify them at the end with the database to check for its correctness and consistency in maintaining the balance of all the users.

For the clients to perform transactions like balance enquiry, deposit, withdrawal and money transfer, we have created methods of them which is invoked by the clients from the main function and in the server side we have switch case to process the requested transaction. A single database server of MySQL is used to maintain all the accounts of customers. Multiple clients are allowed to access the server at the same time to invoke the above transactions by creating the objects of the clients we have used threads for concurrent transaction invoked by all the customers and, objects of clients, server connection threads and print statements of the balance after every transaction to reflect correct and consistent states of account balance in all customers’ accounts. This design of the application has met all the requirements.

**2. Implementation of banking application**

****

Fig 1: This Snippet is the Server project code with comments consisting of the main function of establishing connection with client’s objects.

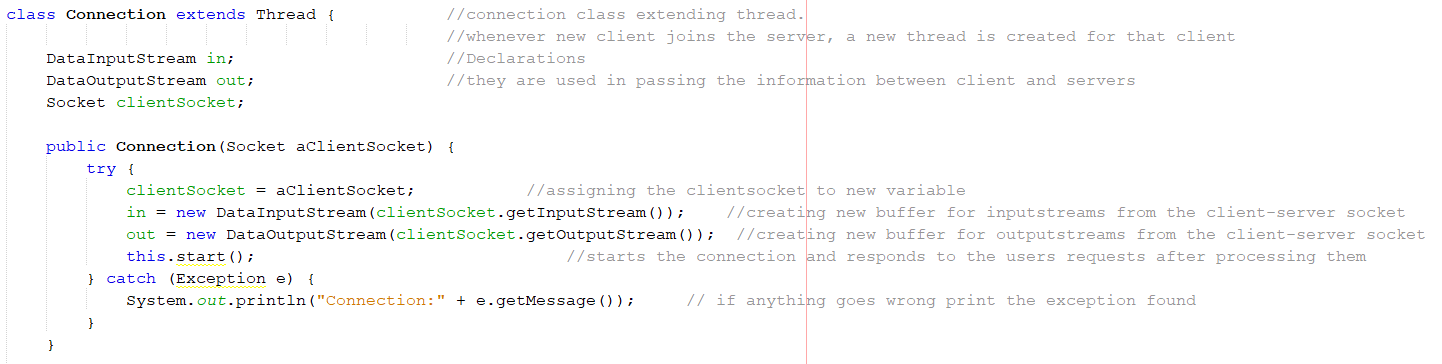
****

Fig 2: This Snippet is the Server project code with comments consisting of the connection class.

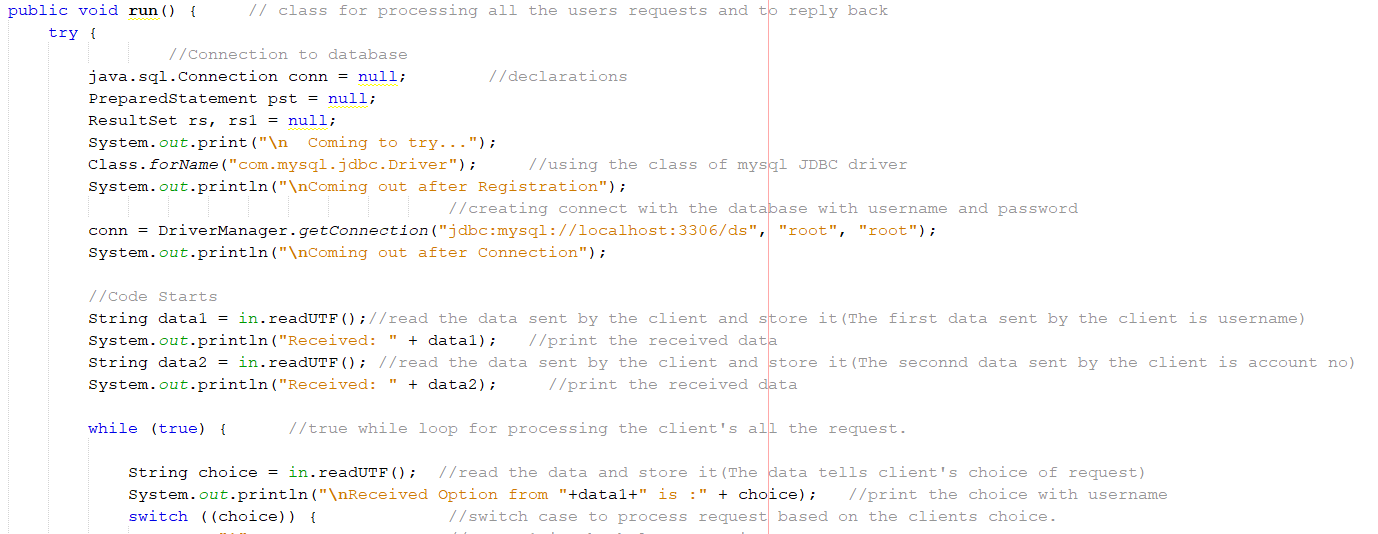
****

Fig 3: This Snippet is the Server project code with comments consisting of run function where the connection with the database is established, also the receiving of username and password from the client side.

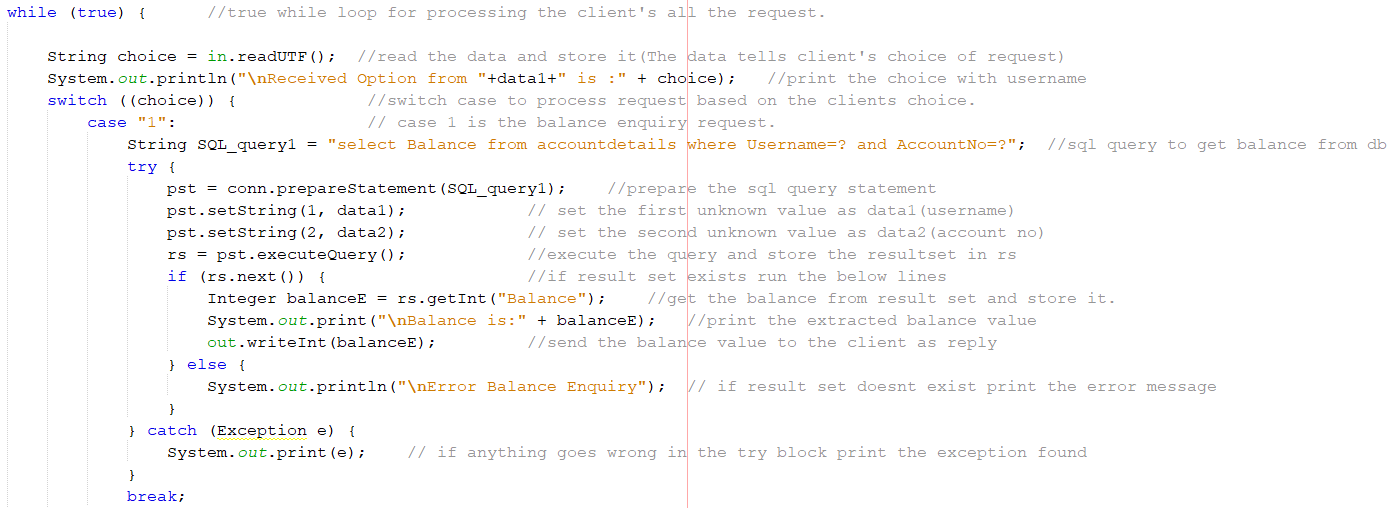
****

Fig 4: This Snippet is the sever project code with comments consisting of the switch case within a while loop. It contains case 1 which is balance enquiry.

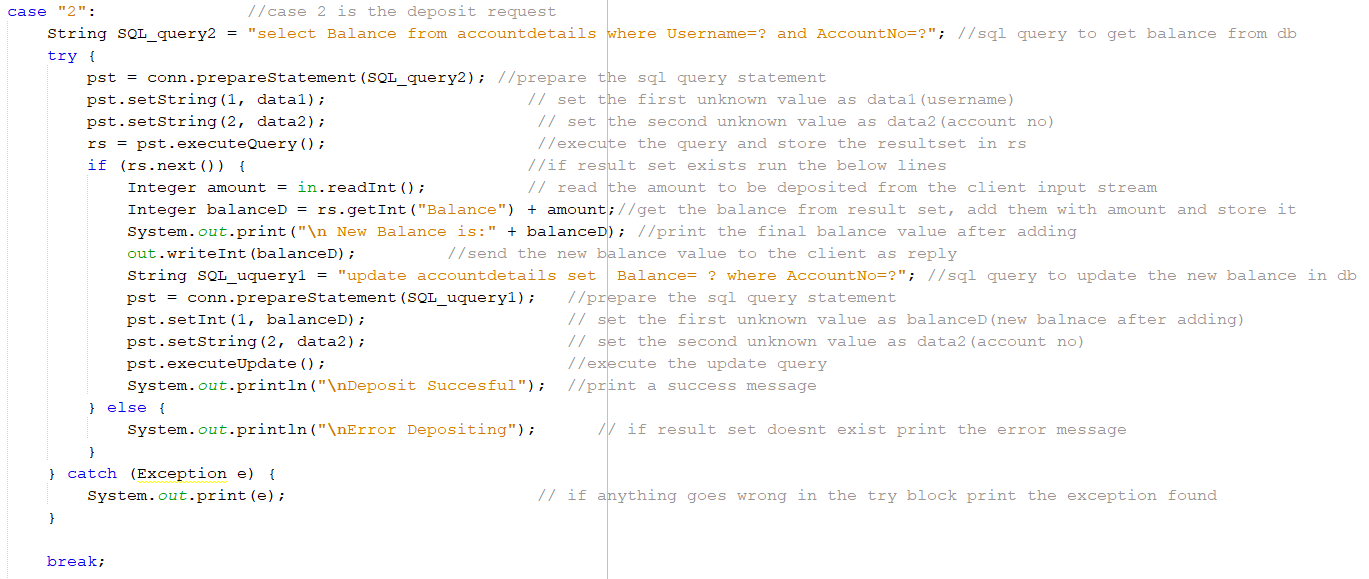
****

Fig 5: This Snippet is the sever project code with comments consisting case 2 of the switch case, which is deposit transaction

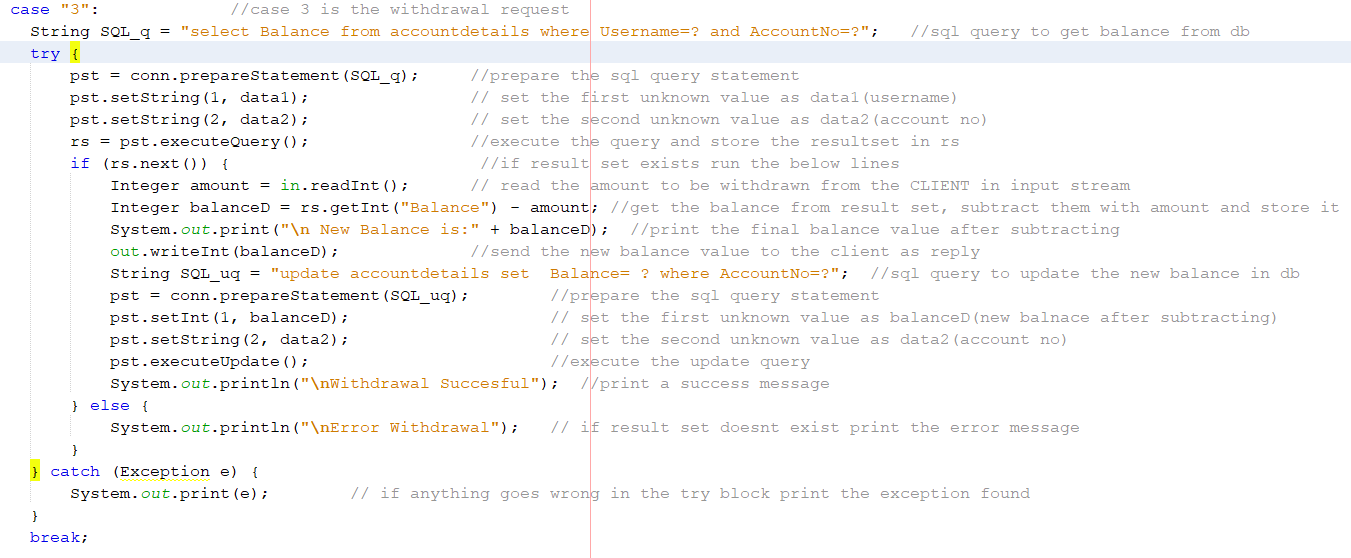
****

Fig 6: This Snippet is the sever project code with comments consisting case 3 of the switch case, which is withdrawal transaction

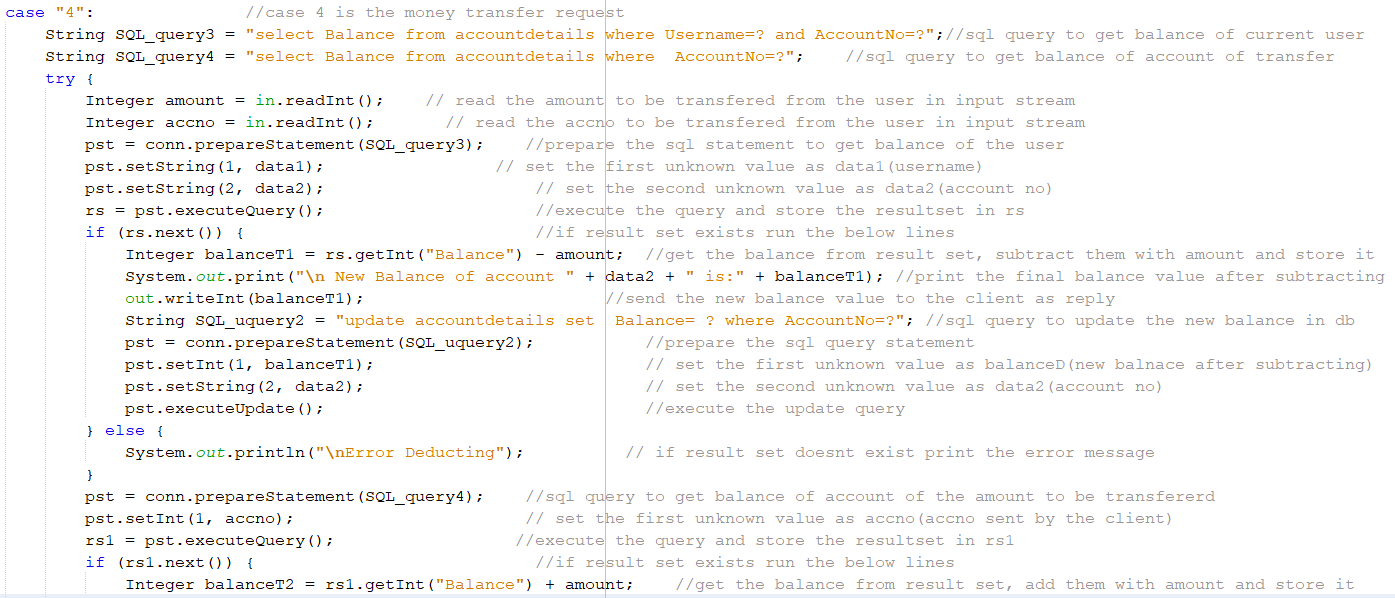
****

Fig 7: This Snippet is the sever project code with comments consisting case 4 of the switch case which is transfer transaction

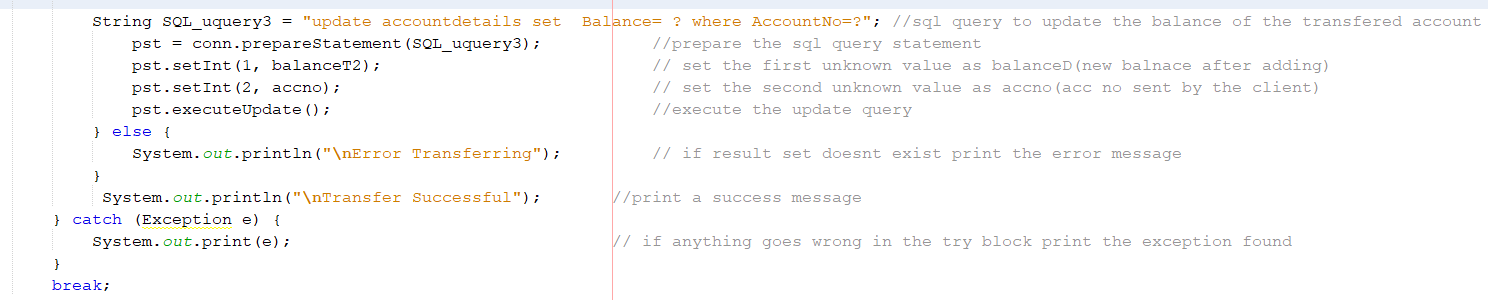
****

Fig 8: This Snippet is the sever project code with comments consisting case 4 of the switch case which is transfer transaction of updating the balance to the database.

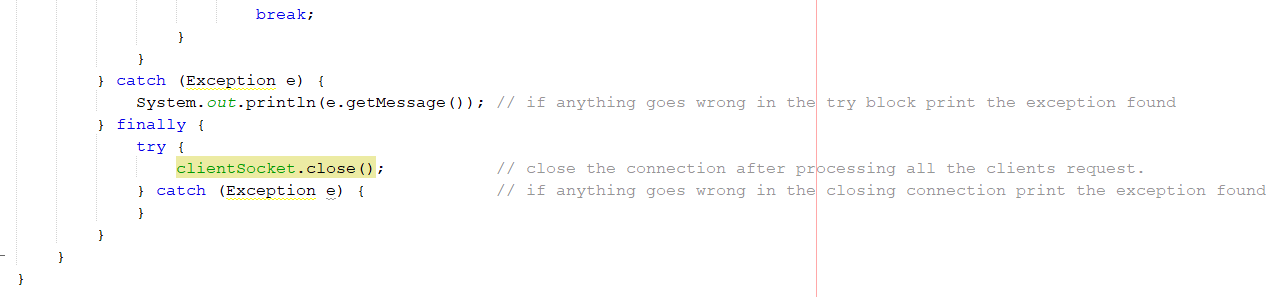
****

Fig 9: This Snippet is the sever project code with comments consisting of the catch blocks of the code at the end

****

Fig 10: This Snippet is the client project code of the java class file consisting of the client side connection request code with comments

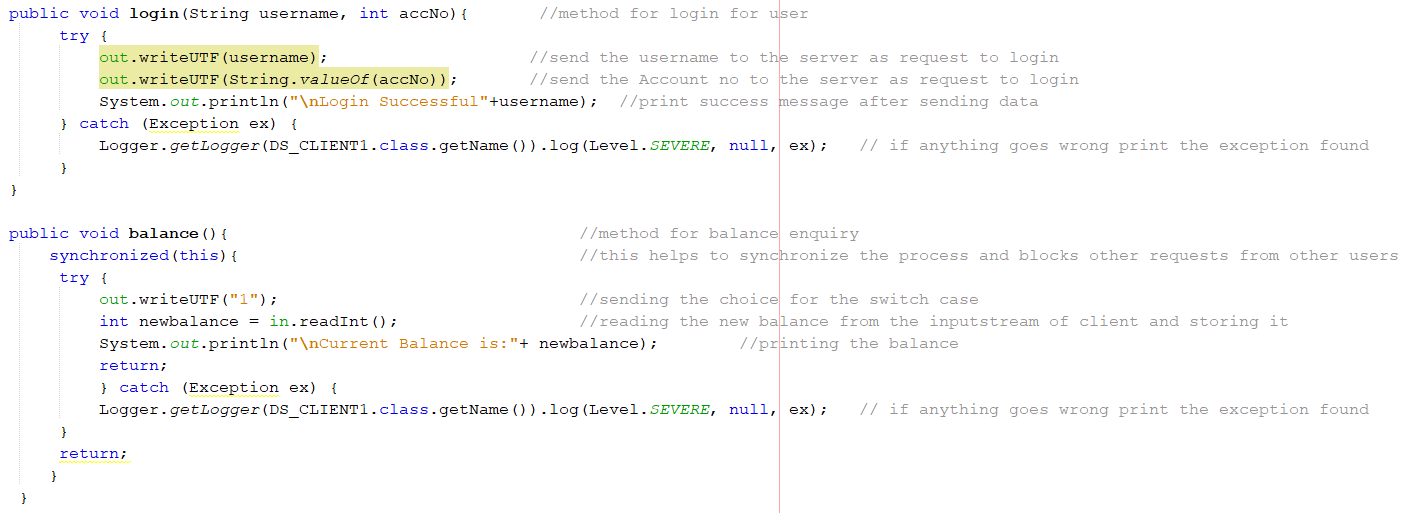
****

Fig 11: This Snippet is the client project code of the java class file consisting of the login and balance enquiry methods with comments

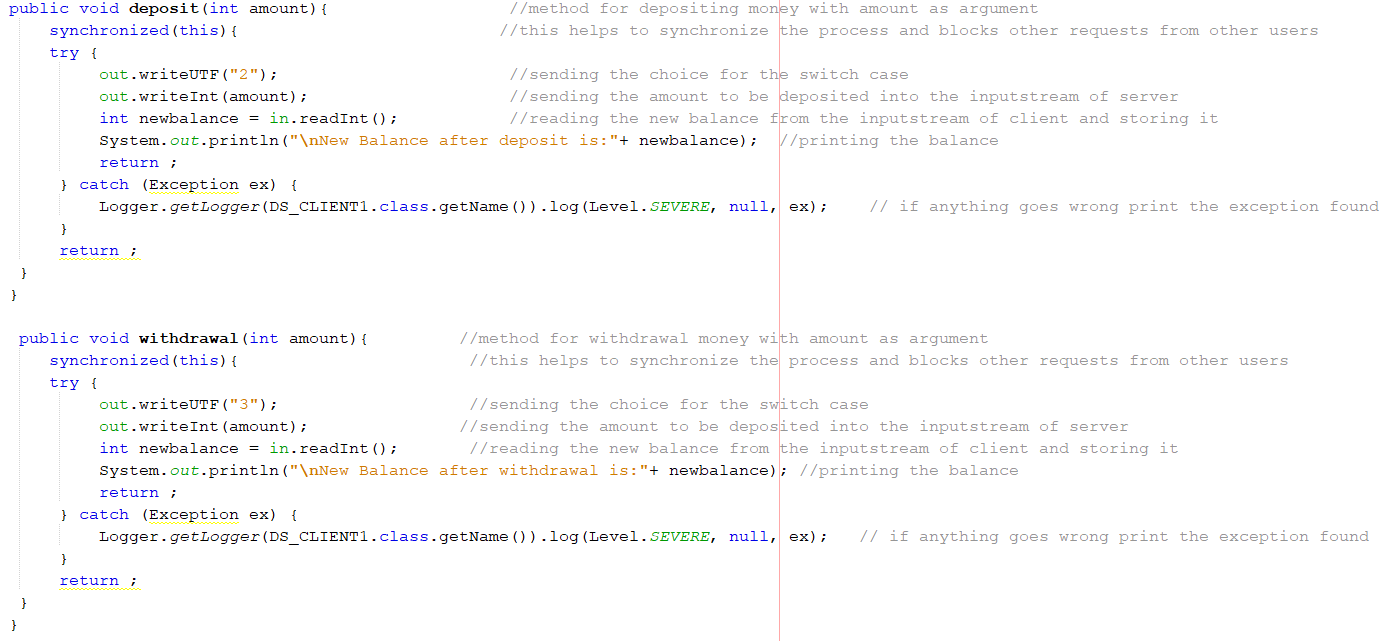
****

Fig 12: This Snippet is the client project code of the java class file consisting of deposit and withdrawal methods with comments

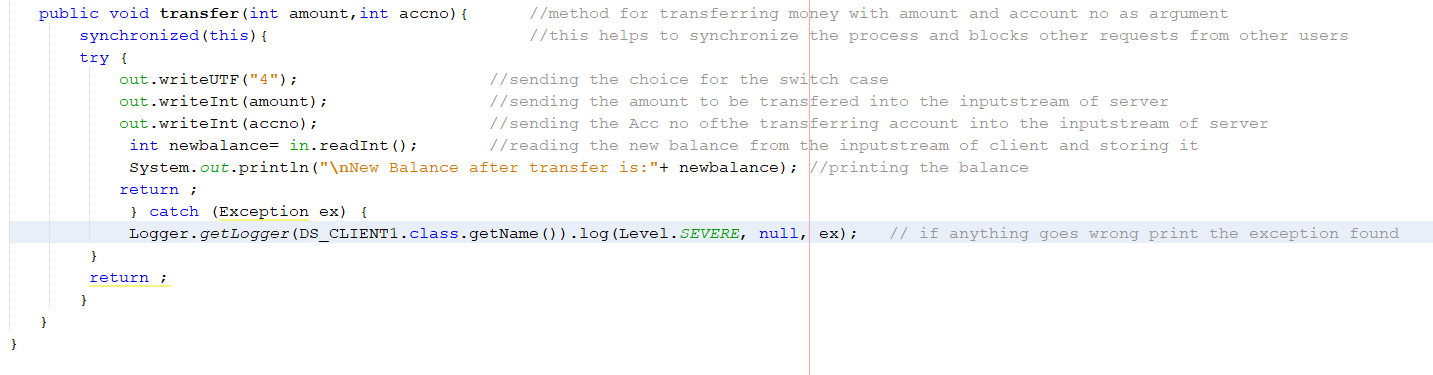
****

Fig 13: This Snippet is the client project code of the java class file consisting of transfer method with comments

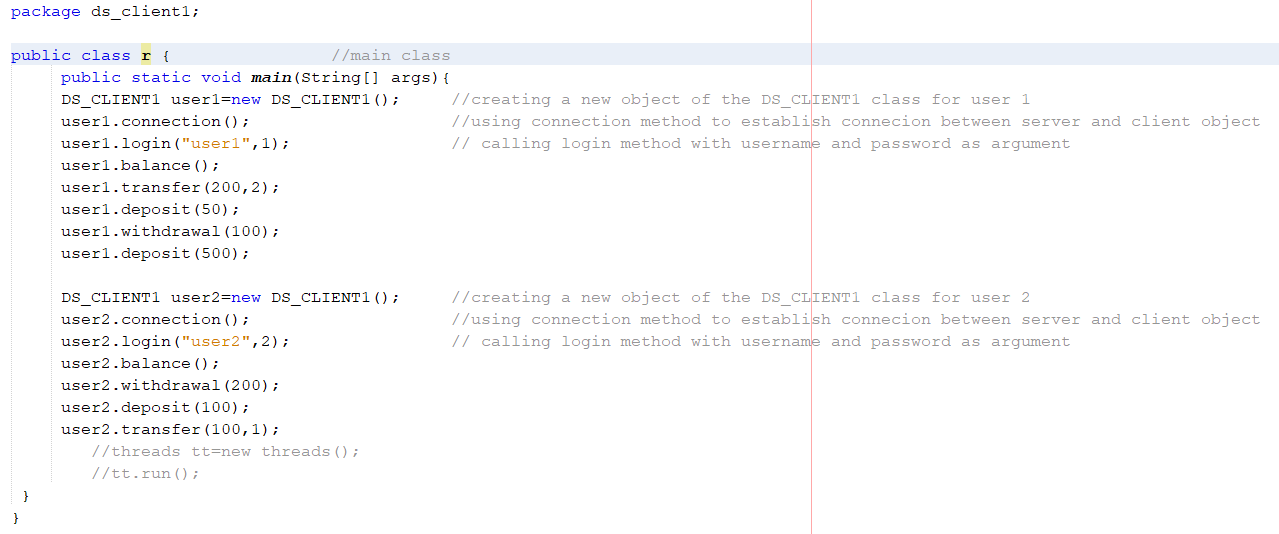
****

Fig 14: This Snippet is the client project code of the java class file consisting of main function for serial execution of the transactions with comments

From the figure 14: There are two users namely user1 and user2 using our banking application with the account no 1 and 2 respectively. User 1‘s requests are processed first then the requests of user 2. Balance of all the users is set to 1000 as default. The User 1 first checks the balance, transfers 200 to account no 2, then deposits 50, withdraws 100 from his account and finally deposits 500 into his account. User 2 also check his balance first, withdraws 200 from his account, deposits 100 and finally transfers 100 to the account no 1.

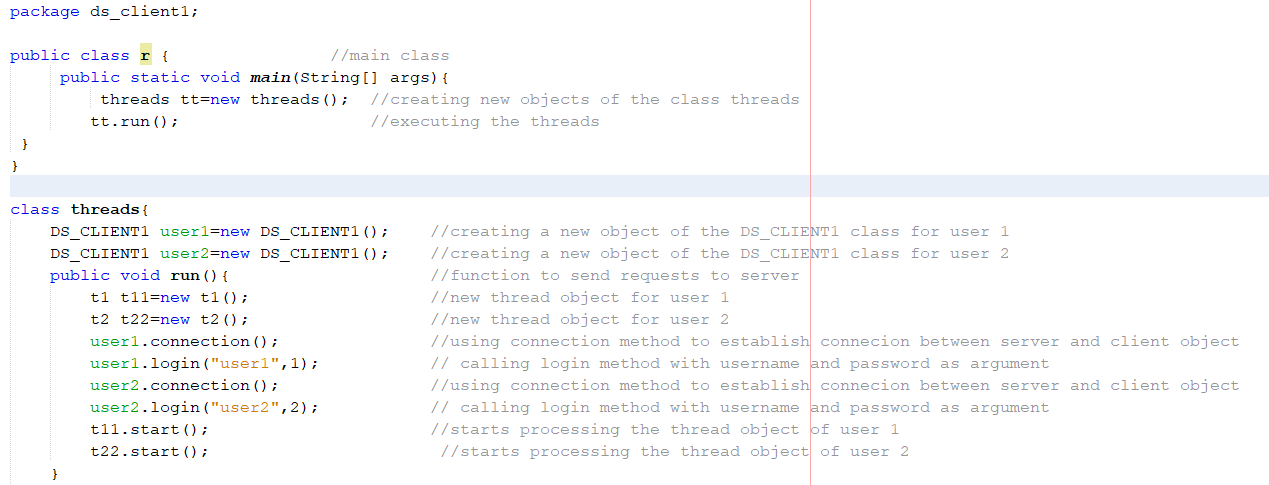
****

Fig 15: This Snippet is the client project code of the java class file consisting of main for concurrent execution of the transactions function with comments

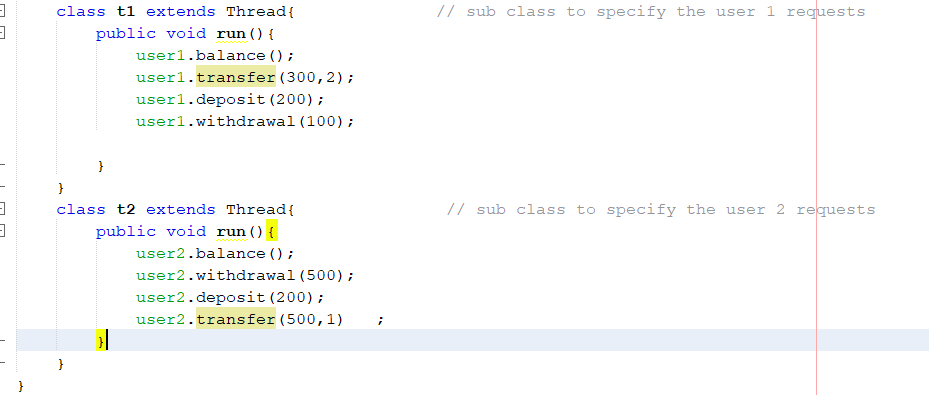
****

Fig 16: This Snippet is the client project code of the java class file consisting of main for concurrent execution of the transactions function threads with comments

From the figure 15 & 16: There are two users namely user1 and user2 using our banking application with the account no 1 and 2 respectively. Both the users’ requests are processed concurrently. Balance of all the users is set to 1000 as default. The User 1 first checks the balance, transfers 300 to account no 2, then deposits 200 and finally withdraws 100 from his account. User 2 also check his balance first, withdraws 500 from his account, deposits 200 and finally transfers 500 to the account no 1.

**3. Testing of the developed banking application**

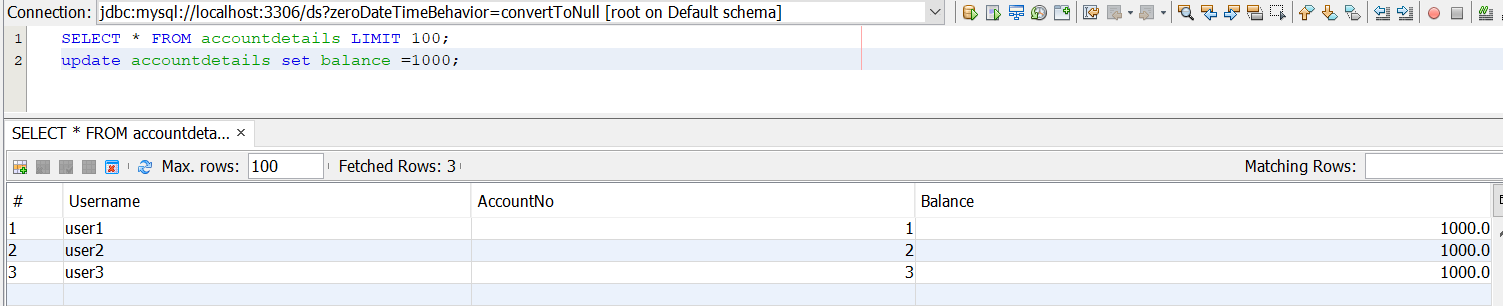


Fig 17: This Snippet is database table of account details consisting of the users’ username, account no and their balance.

All users balance is set to 1000 as default before execution of the user’s request both serially and concurrently.

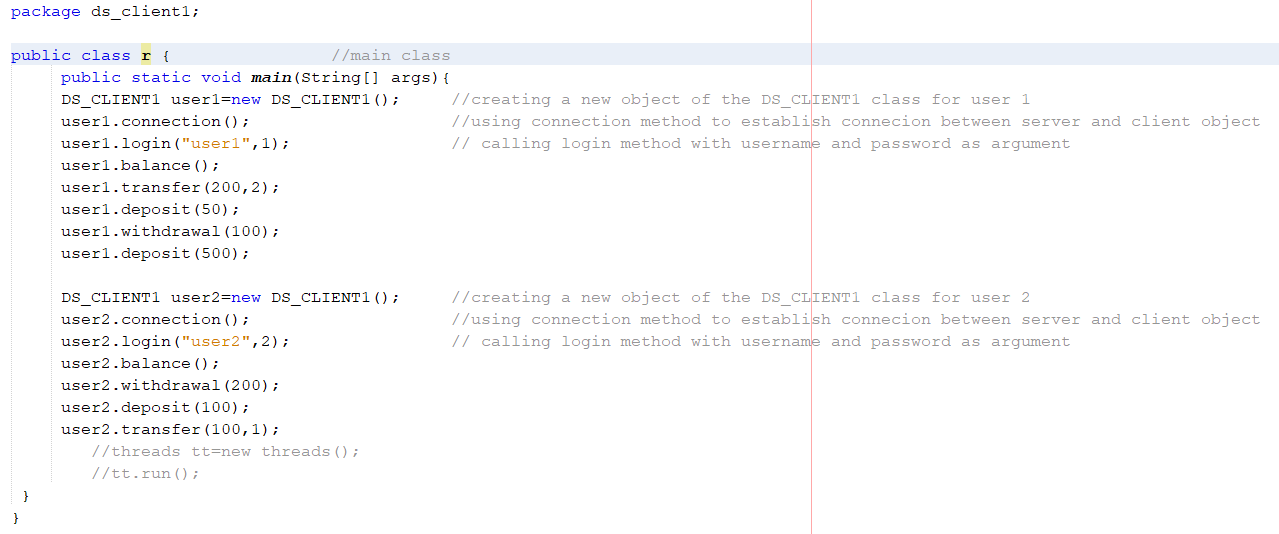
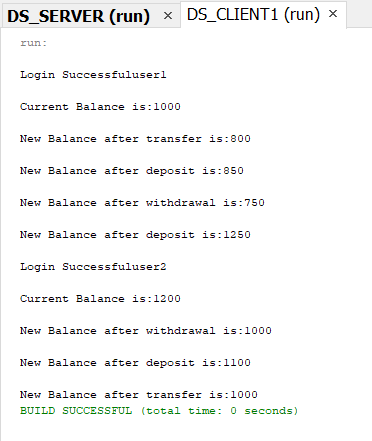
****

Fig 18: This Snippet is output of the client side serial execution of users’ request.

From the figure 14 there were two users namely user1 and user2 using our banking application with the account no 1 and 2 respectively. User 1‘s requests are processed first then the requests of user 2. Balance of all the users is set to 1000 as default. The User 1 first checks the balance [1000], transfers 200 to account no 2 [800], then deposits 50 [850], withdraws 100 [750] from his account and finally deposits 500 [1250] into his account and then User 2 login checks his balance [1200] (1000 + transferred amount 200=1200), withdraws 200 [1000] from his account, deposits 100 [1100] and finally transfers 100 [1000] to the account no 1 is displayed in the figure 18.

**Note: [ ] indicates the balance of the user after the transactions**

**BALANCES:**

* User 1: 1000 -200(transfer) +50(deposit) -100(withdrawal) +500(deposit) **=1250** + (100 transfer from user 2) **=1350** (final balance).
* User 2: 1000 +(200 transfer from user 1) =1200(current balance) -200(withdrawal) +100(deposit) -100(transfer) **=1000**

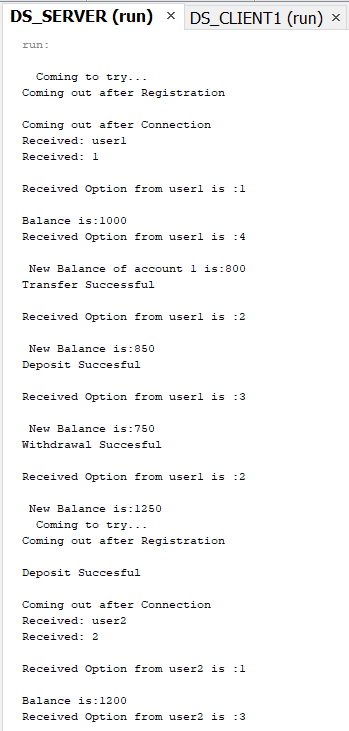
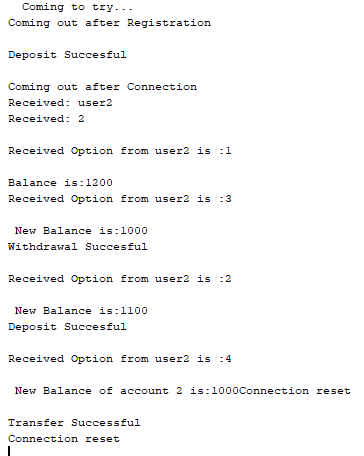


Fig 19: This Snippet is output of the server side serial execution of users’ request. This snippet is the detailed version of the client side output as how users’ requests are processed, their order of processing and their output.

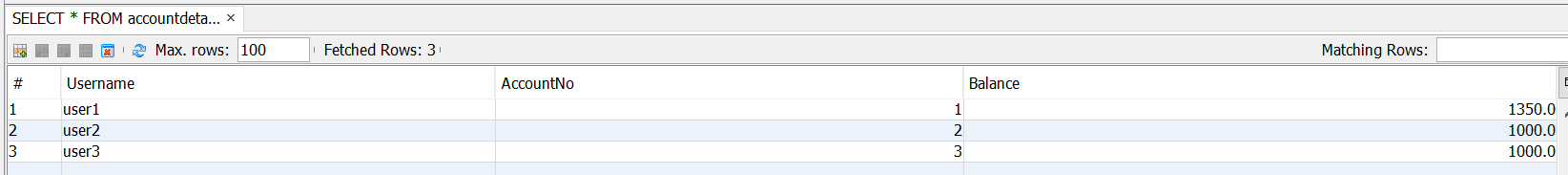


Fig 20: This Snippet database table after serial execution of the user’s request. The balance of user 1 was 1250 but later user 2 transferred 100 from his account which results to 1350 as user 1 balance and user 2 balance was 1000 at the end after transfer transaction which was processed serially.

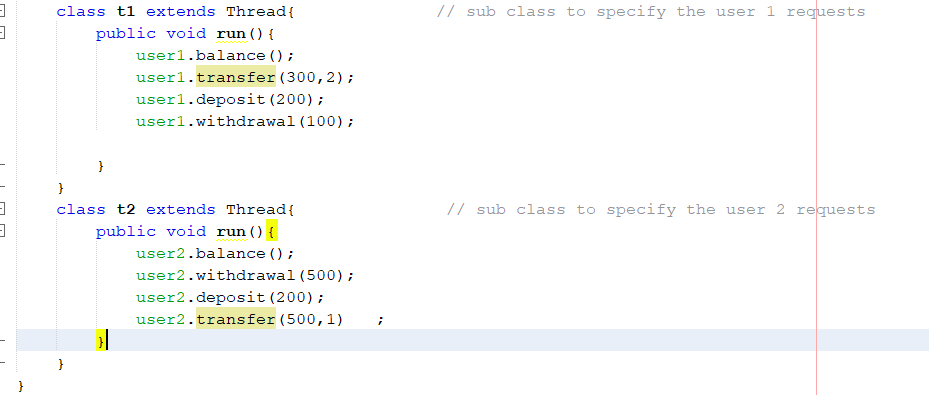
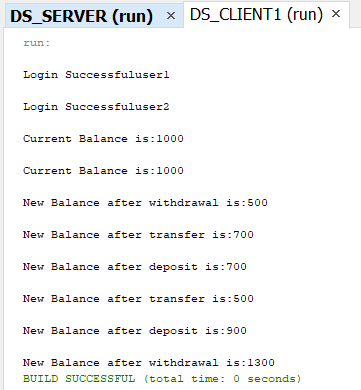
****

Fig 21: This Snippet is output of the client side serial execution of users’ request.

From the figure 15 & 16 there are two users namely user1 and user2 using our banking application with the account no 1 and 2 respectively. Both the users’ requests are processed concurrently using threads. Balance of all the users is set to 1000 as default. User 1 has logged in first then the user 2. Both the users balance is displayed. User 2’s withdrawal of 500 [500] is processed, then user 1’s transfer of 300[700] to the user 2 (800), then user 2’s deposit of 200 [700] (should be 1000 but delay in updating), then the user2’s transfer of 500 [500] (reflected correctly as 500 at the end of transaction) is processed, then the user 1’s deposit of 200[900](should be 1400 but delay in updating database) finally user 1’s withdrawal of 100 [1300] (Reflected correctly as 1300 at the end of all transaction) is processed, this all transaction are displayed in the figure 21 as in mentioned order.

**Note: [ ] indicates the balance of the user after the transactions**

**BALANCES:**

* User 1: 1000 -300(transfer) +200(deposit) -100(withdrawal) = 800 + 500(transfer from user 2) **=1300**
* User 2: 1000 -500(withdrawal) +200(deposit) -500(transfer) = 200 +300(transfer from user 1) **=500**

As they run concurrently, accessing the balance at different times caused few troubles in displaying the correct balance at that instant of time but at the end all the users balance are reflected correctly after processing of all transaction. User 1’s balance of 1300 and user 2’s balance of 500.

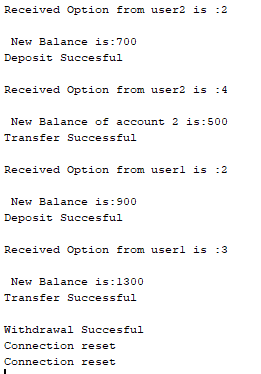


Fig 22: This Snippet is output of the server side concurrent execution of users’ request. This snippet is the detailed version of the client side output as how users’ requests are processed, their order of processing and their output.



Fig 23: This Snippet database table after concurrent execution of the user’s request. The balance of user 1 is 1300 and user 2 is 500 at the end after transfer transaction which was processed concurrently.

**4. Demonstration.**

For actual demonstration of the client server based banking application, you will have to run the source files in the DS\_SERVER and DS\_CLIENT1 folder. First run the server.java and then the ds\_client1.java. You will be able to witness the Transactions, timing & synchronisation, coordination & consensus in the distributed transaction.

Note: Make sure to include the java-sql-connection-5.0.8.jar file in your libraries.