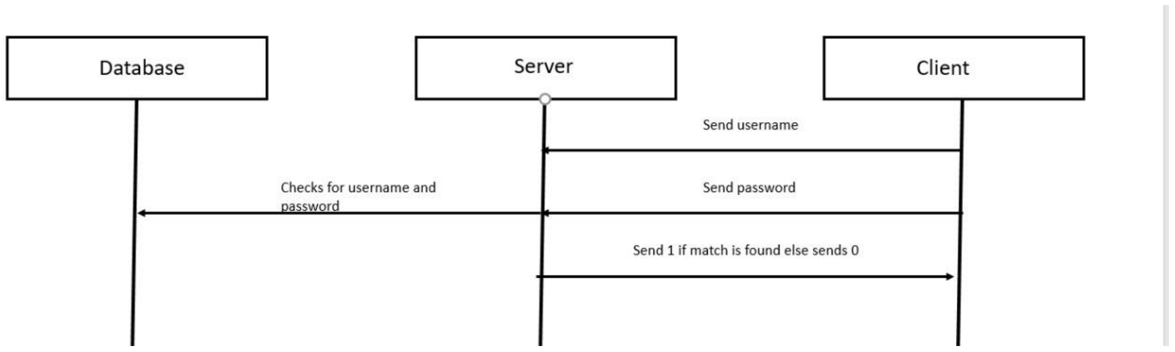
Contents

<u>**Question No. A**</u>

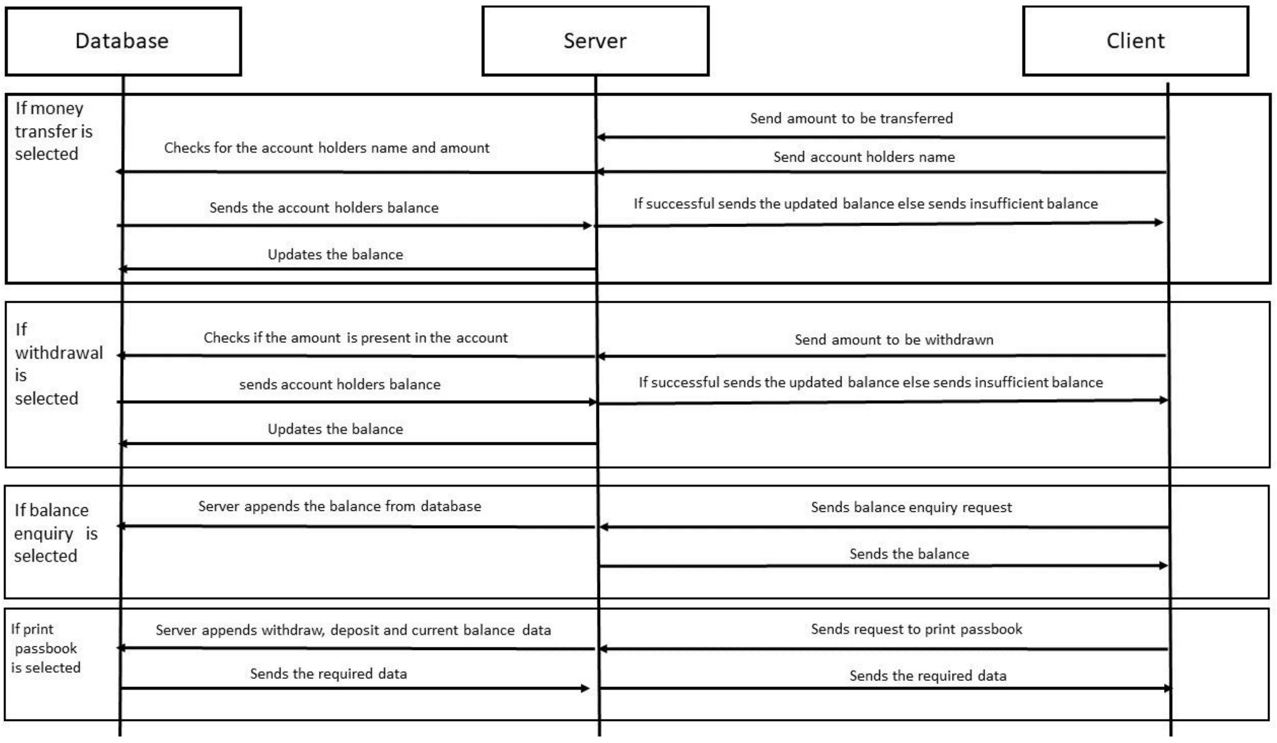
Scenario: Consider an online client-server based banking application, where customers perform transactions like balance enquiry, deposit and money transfer. A single server maintains all the accounts of customers. Multiple clients can access the server at the same time to invoke the above transactions. Design and implement 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.
Your report should emphasize on:

A1.1 Design of an effective solution to the above scenario and description of techniques used:

(i) For login part:


Database
Server
Client
Send username
Checks for username and password
Send password
Send 1 if match is found else sends 0

(ii) For logic part i.e. switch case part:

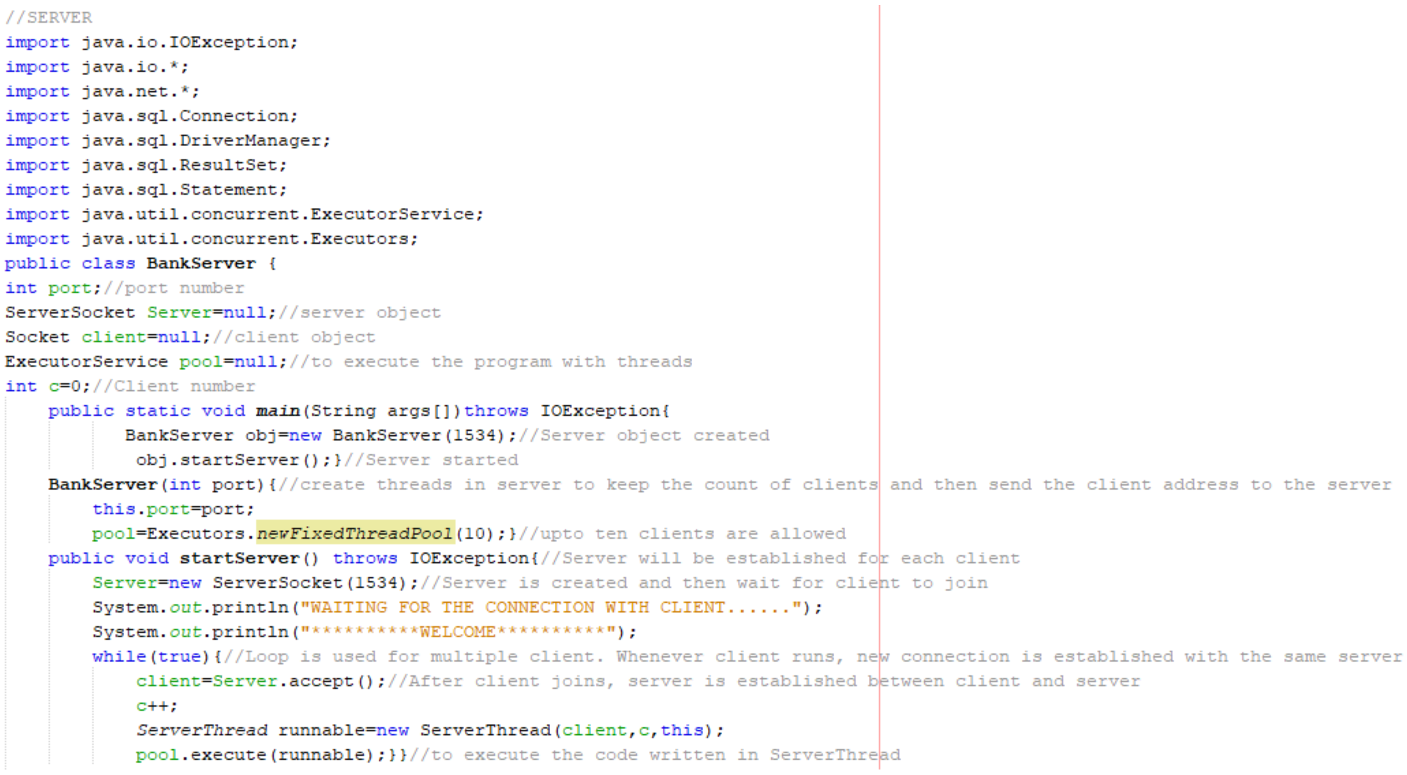

Database
Server
Client
If money transfer is selected
Send amount to be transferred
Checks for the account holders name and amount
Send account holders name
Sends the account holders balance
If successful sends the updated balance else sends insufficient balance
Updates the balance
If withdrawal is selected
Checks if the amount is present in the account
Send amount to be withdrawn
sends account holders balance
If successful sends the updated balance else sends insufficient balance
Updates the balance
If balance enquiry is selected
Server appends the balance from database
Sends balance enquiry request
Sends the balance
If print passbook is selected
Server appends withdraw, deposit and current balance data
Sends request to print passbook
Sends the required data
Sends the required data

Here I have made sequence to diagram to show how data will flow from client to server and server to client and also when data will be accessed by the database. So from in (i) part, it shows design for login. Client will send login details to server and server will verify it from the database. After server verifies is, it sends verification back to the client. If the client's login detail is incorrect then it'll break and program will terminate. If the login is successful then logic for handling the client's request will be performed as seen from part(ii). Based on the client's choice operation will b performed. After all the transaction is performed successfully, data will be committed and it will be updated in client's database. The main objective of this scenario is to handle multiple clients by single server. So here we have used socket programming and MySQL database which is handling the conflict pair. ACID(ATOMICITY, CONSISTENCY, ISOLATION, DURABILITY) property has to be satisfied by the program in order to not get conflict pair. Conflict pair occurs when two or more client at the same time tries to read and write or write and write in database concurrently. So inorder to handle multiple client at the same time I am using commit operation in my program.

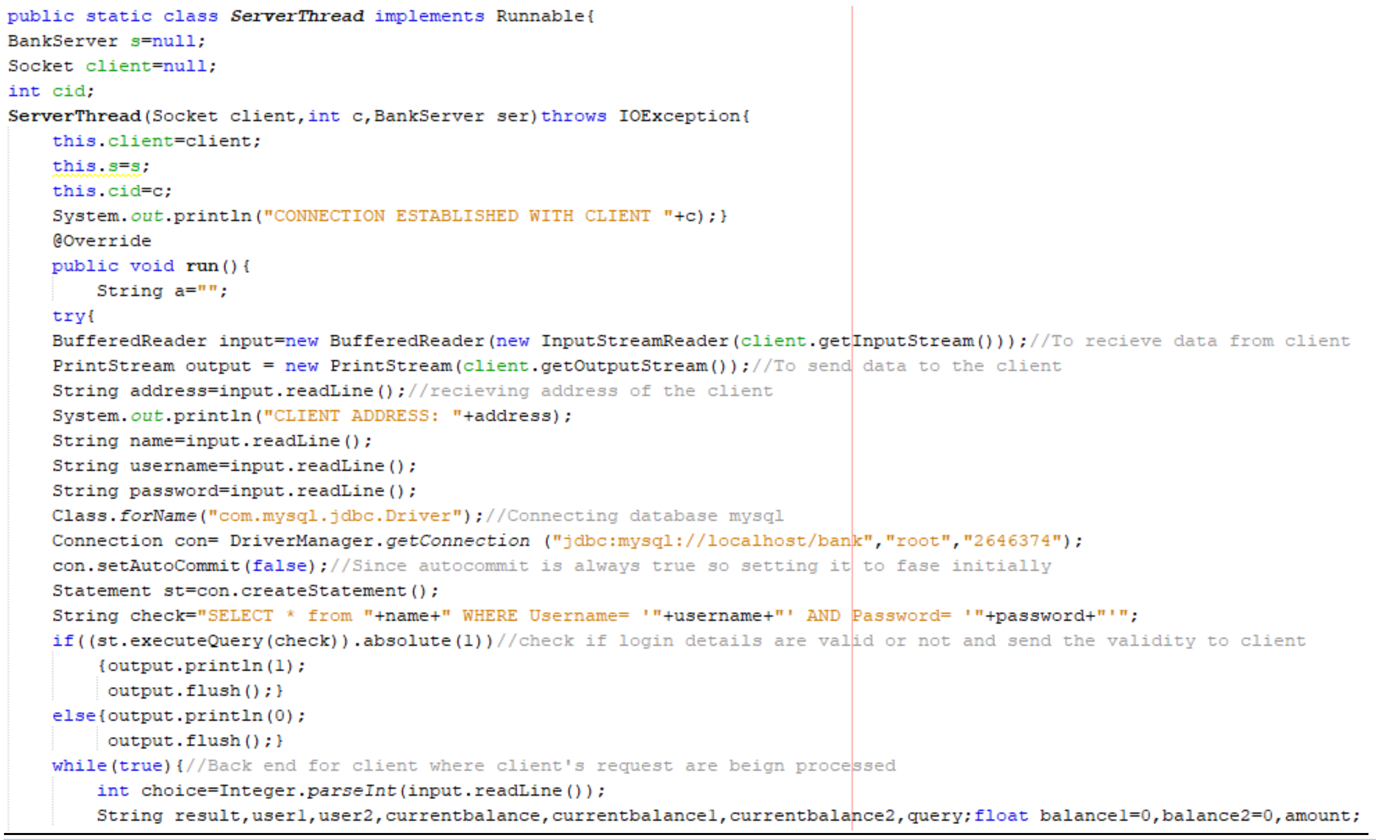
So using the above design I will create a socket programming in java for multiple client single server socket programming using multithreading.

A1.2 Implementation of banking application to reflect correct and consistent states of account balance when multiple clients initiate transactions:

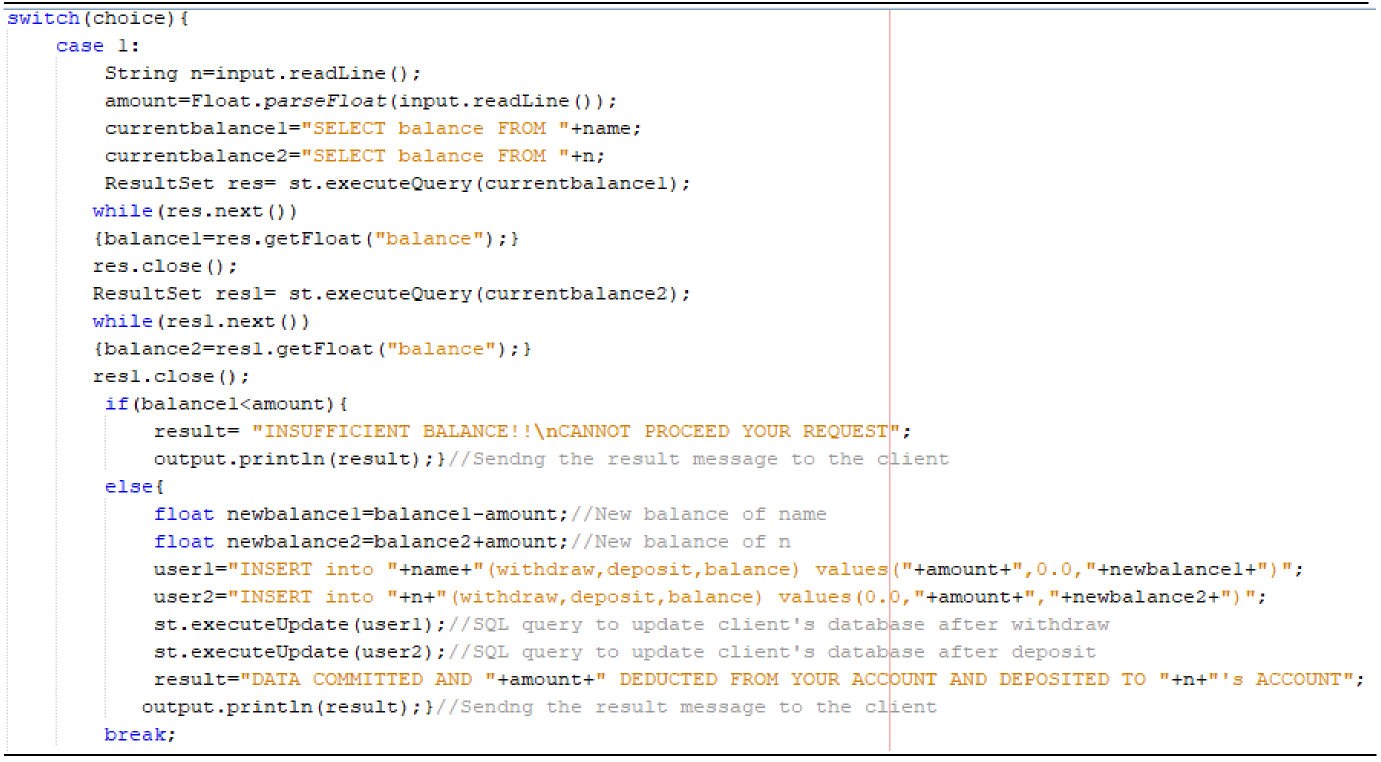
<u>SERVER PROGRAM</u> ➔



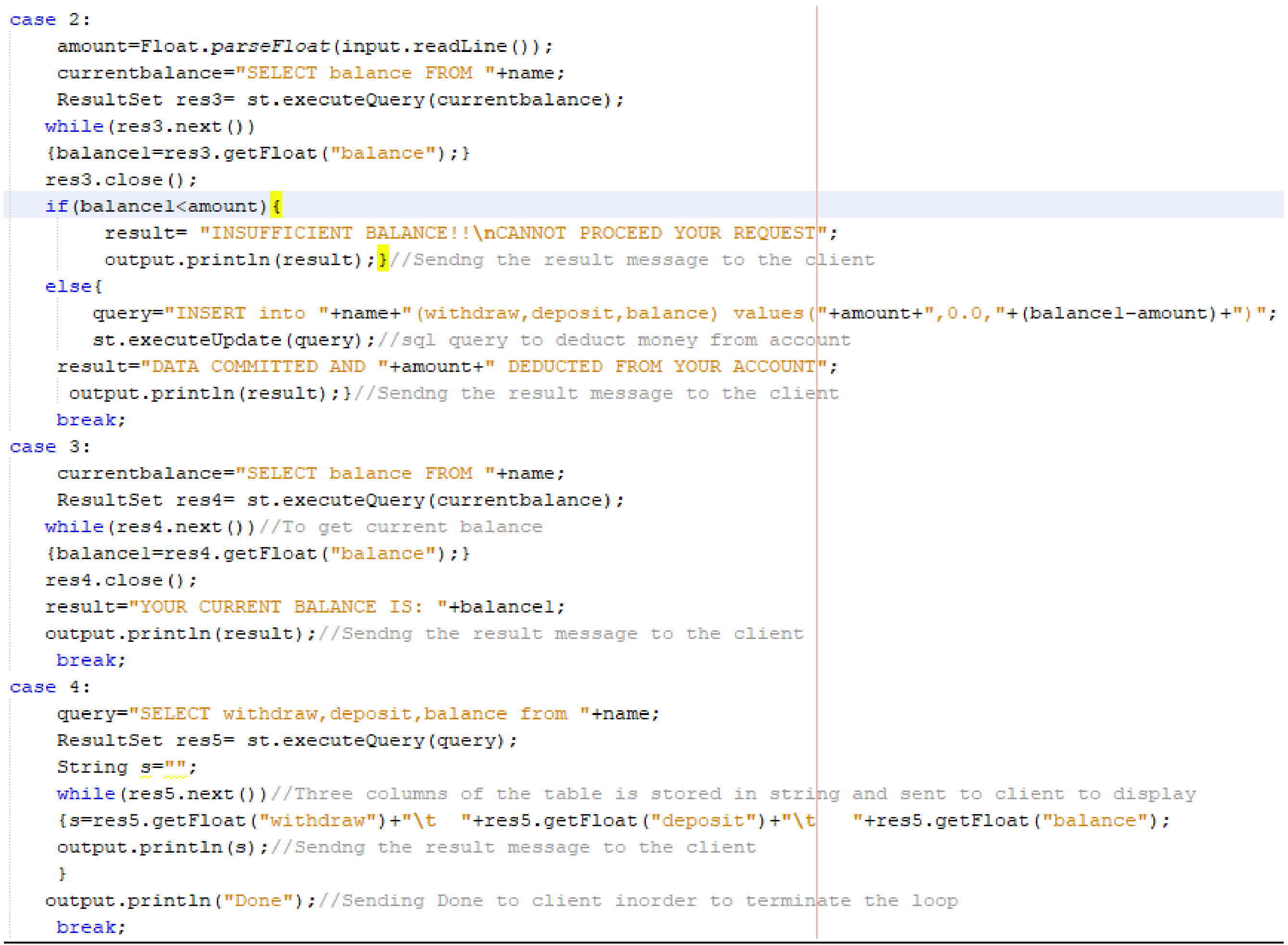
```
//SERVER
import java.io.IOException;
import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class BankServer {
int port;//port number
ServerSocket Server=null;//server object
Socket client=null;//client object
ExecutorService pool=null;//to execute the program with threads
int c=0;//Client number
    public static void main(String args[])throws IOException{
           BankServer obj=new BankServer(1534);//Server object created
            obj.startServer();}//Server started
    BankServer(int port){//create threads in server to keep the count of clients and then send the client address to the server
        this.port=port;
        pool=Executors.newFixedThreadPool(10);}//upto ten clients are allowed
    public void startServer() throws IOException{//Server will be established for each client
        Server=new ServerSocket(1534);//Server is created and then wait for client to join
        System.out.println("WAITING FOR THE CONNECTION WITH CLIENT......");
        System.out.println("**********WELCOME**********");
        while(true){//Loop is used for multiple client. Whenever client runs, new connection is established with the same server
            client=Server.accept();//After client joins, server is established between client and server
            c++;
            ServerThread runnable=new ServerThread(client,c,this);
            pool.execute(runnable);}}//to execute the code written in ServerThread
```



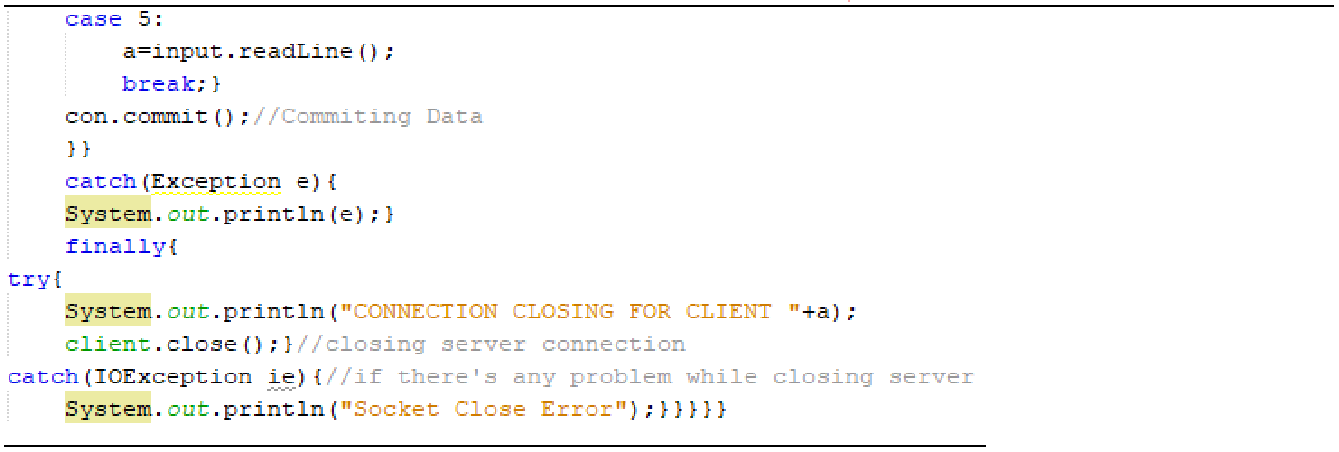
```java
public static class ServerThread implements Runnable{
BankServer s=null;
Socket client=null;
int cid;
ServerThread(Socket client,int c,BankServer ser)throws IOException{
    this.client=client;
    this.s=s;
    this.cid=c;
    System.out.println("CONNECTION ESTABLISHED WITH CLIENT "+c);}
    @Override
    public void run(){
        String a="";
    try{
    BufferedReader input=new BufferedReader(new InputStreamReader(client.getInputStream()));//To recieve data from client
    PrintStream output = new PrintStream(client.getOutputStream());//To send data to the client
    String address=input.readLine();//recieving address of the client
    System.out.println("CLIENT ADDRESS: "+address);
    String name=input.readLine();
    String username=input.readLine();
    String password=input.readLine();
    Class.forName("com.mysql.jdbc.Driver");//Connecting database mysql
    Connection con= DriverManager.getConnection ("jdbc:mysql://localhost/bank","root","2646374");
    con.setAutoCommit(false);//Since autocommit is always true so setting it to fase initially
    Statement st=con.createStatement();
    String check="SELECT * from "+name+" WHERE Username= '"+username+"' AND Password= '"+password+"'";
    if((st.executeQuery(check)).absolute(1))//check if login details are valid or not and send the validity to client
        {output.println(1);
         output.flush();}
    else{output.println(0);
         output.flush();}
    while(true){//Back end for client where client's request are beign processed
        int choice=Integer.parseInt(input.readLine());
        String result,user1,user2,currentbalance,currentbalance1,currentbalance2,query;float balance1=0,balance2=0,amount;
```



```java
switch(choice){
    case 1:
        String n=input.readLine();
        amount=Float.parseFloat(input.readLine());
        currentbalance1="SELECT balance FROM "+name;
        currentbalance2="SELECT balance FROM "+n;
        ResultSet res= st.executeQuery(currentbalance1);
       while(res.next())
       {balance1=res.getFloat("balance");}
       res.close();
       ResultSet res1= st.executeQuery(currentbalance2);
       while(res1.next())
       {balance2=res1.getFloat("balance");}
       res1.close();
        if(balance1<amount){
            result= "INSUFFICIENT BALANCE!!\nCANNOT PROCEED YOUR REQUEST";
            output.println(result);}//Sendng the result message to the client
        else{
            float newbalance1=balance1-amount;//New balance of name
            float newbalance2=balance2+amount;//New balance of n
            user1="INSERT into "+name+"(withdraw,deposit,balance) values("+amount+",0.0,"+newbalance1+")";
            user2="INSERT into "+n+"(withdraw,deposit,balance) values(0.0,"+amount+","+newbalance2+")";
            st.executeUpdate(user1);//SQL query to update client's database after withdraw
            st.executeUpdate(user2);//SQL query to update client's database after deposit
            result="DATA COMMITTED AND "+amount+" DEDUCTED FROM YOUR ACCOUNT AND DEPOSITED TO "+n+"'s ACCOUNT";
           output.println(result);}//Sendng the result message to the client
        break;
```

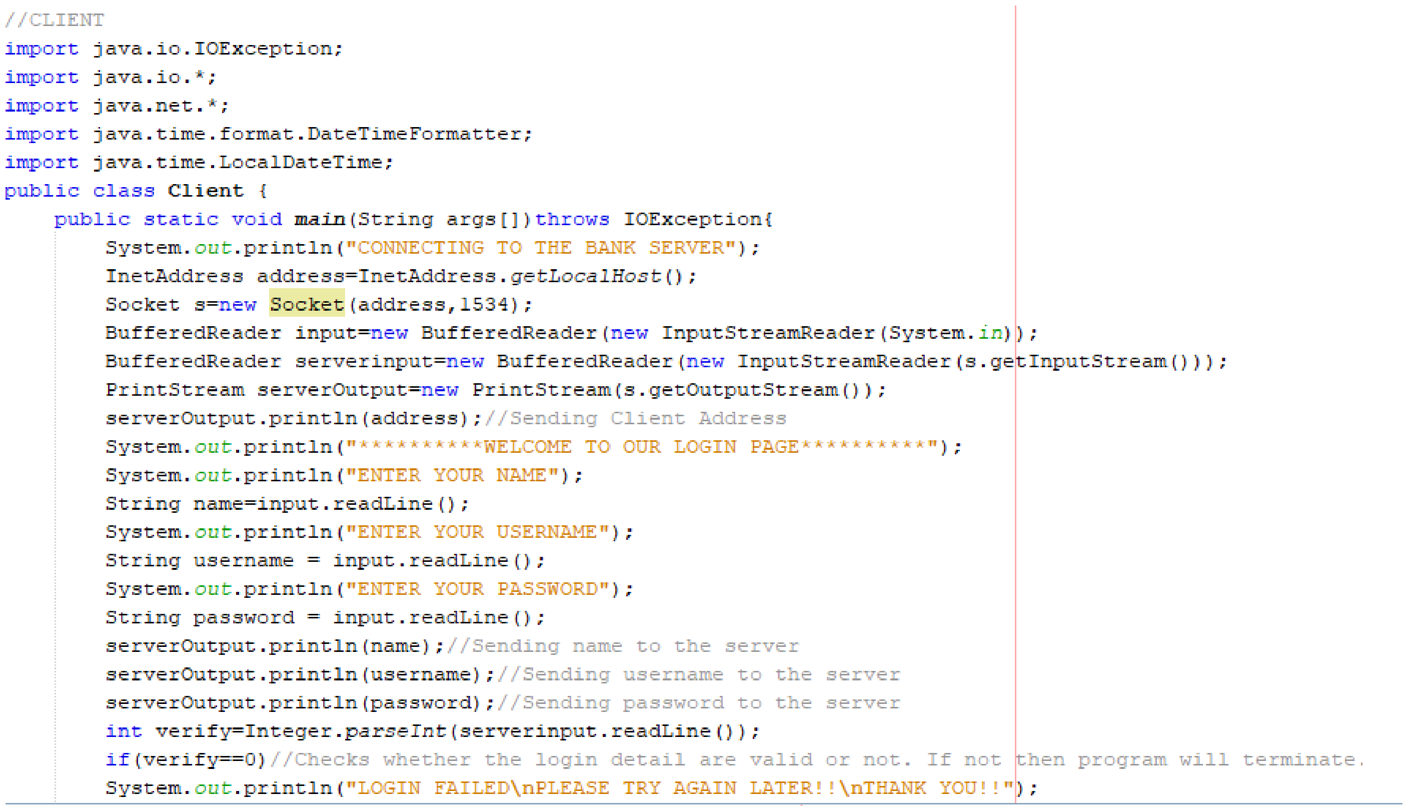


```
case 2:
     amount=Float.parseFloat(input.readLine());
     currentbalance="SELECT balance FROM "+name;
     ResultSet res3= st.executeQuery(currentbalance);
    while(res3.next())
    {balance1=res3.getFloat("balance");}
    res3.close();
    if(balance1<amount){
         result= "INSUFFICIENT BALANCE!!\nCANNOT PROCEED YOUR REQUEST";
         output.println(result);}//Sendng the result message to the client
    else{
        query="INSERT into "+name+"(withdraw,deposit,balance) values("+amount+",0.0,"+(balance1-amount)+")";
        st.executeUpdate(query);//sql query to deduct money from account
     result="DATA COMMITTED AND "+amount+" DEDUCTED FROM YOUR ACCOUNT";
      output.println(result);}//Sendng the result message to the client
     break;
case 3:
     currentbalance="SELECT balance FROM "+name;
     ResultSet res4= st.executeQuery(currentbalance);
    while(res4.next())//To get current balance
    {balance1=res4.getFloat("balance");}
    res4.close();
    result="YOUR CURRENT BALANCE IS: "+balance1;
    output.println(result);//Sendng the result message to the client
     break;
case 4:
     query="SELECT withdraw,deposit,balance from "+name;
     ResultSet res5= st.executeQuery(query);
     String s="";
     while(res5.next())//Three columns of the table is stored in string and sent to client to display
     {s=res5.getFloat("withdraw")+"\t  "+res5.getFloat("deposit")+"\t   "+res5.getFloat("balance");
     output.println(s);//Sendng the result message to the client
     }
    output.println("Done");//Sending Done to client inorder to terminate the loop
     break;
```

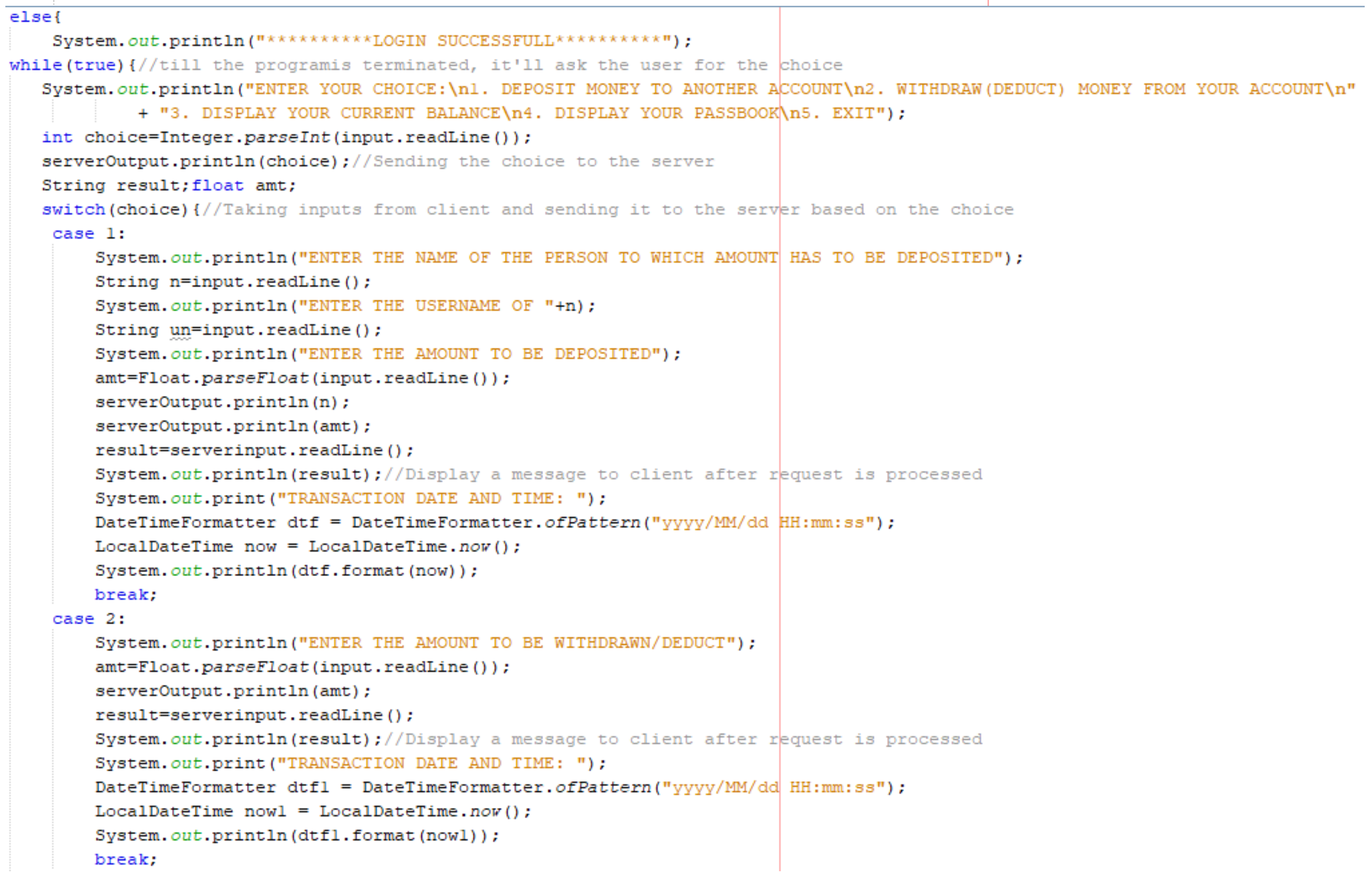


```
     case 5:
         a=input.readLine();
         break;}
     con.commit();//Commiting Data
     }}
     catch(Exception e){
     System.out.println(e);}
     finally{
try{
     System.out.println("CONNECTION CLOSING FOR CLIENT "+a);
     client.close();}//closing server connection
catch(IOException ie){//if there's any problem while closing server
     System.out.println("Socket Close Error");}}}}}
```

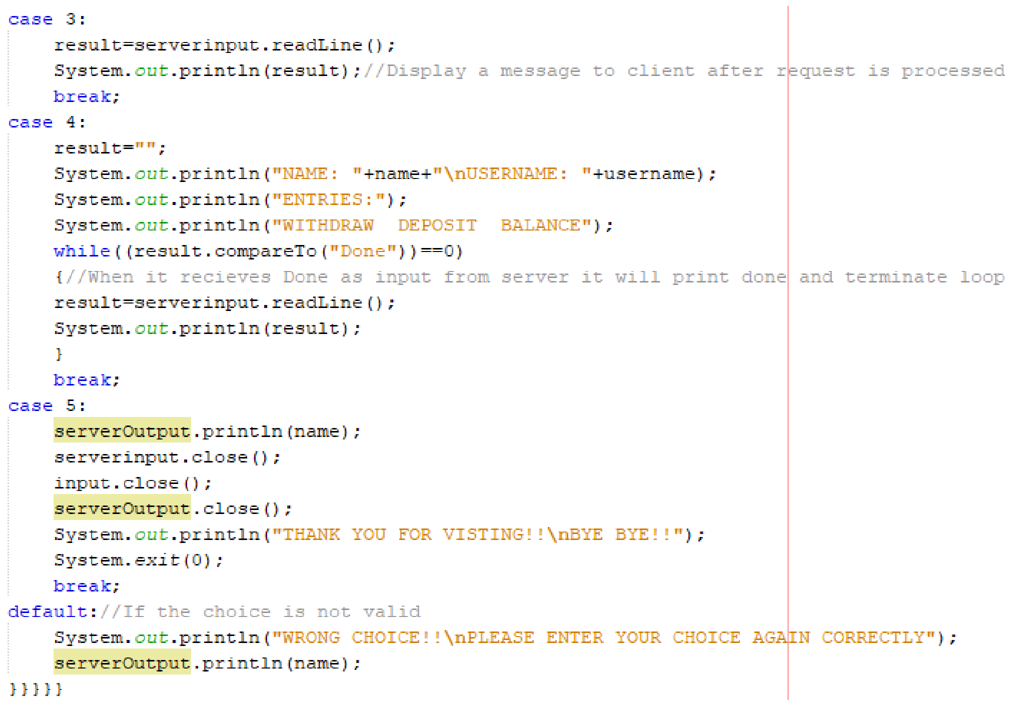
CLIENT PROGRAM ➔



```java
//CLIENT
import java.io.IOException;
import java.io.*;
import java.net.*;
import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;
public class Client {
    public static void main(String args[])throws IOException{
        System.out.println("CONNECTING TO THE BANK SERVER");
        InetAddress address=InetAddress.getLocalHost();
        Socket s=new Socket(address,1534);
        BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
        BufferedReader serverinput=new BufferedReader(new InputStreamReader(s.getInputStream()));
        PrintStream serverOutput=new PrintStream(s.getOutputStream());
        serverOutput.println(address);//Sending Client Address
        System.out.println("**********WELCOME TO OUR LOGIN PAGE**********");
        System.out.println("ENTER YOUR NAME");
        String name=input.readLine();
        System.out.println("ENTER YOUR USERNAME");
        String username = input.readLine();
        System.out.println("ENTER YOUR PASSWORD");
        String password = input.readLine();
        serverOutput.println(name);//Sending name to the server
        serverOutput.println(username);//Sending username to the server
        serverOutput.println(password);//Sending password to the server
        int verify=Integer.parseInt(serverinput.readLine());
        if(verify==0)//Checks whether the login detail are valid or not. If not then program will terminate.
        System.out.println("LOGIN FAILED\nPLEASE TRY AGAIN LATER!!\nTHANK YOU!!");
```



```java
else{
    System.out.println("**********LOGIN SUCCESSFULL**********");
while(true){//till the programis terminated, it'll ask the user for the choice
   System.out.println("ENTER YOUR CHOICE:\n1. DEPOSIT MONEY TO ANOTHER ACCOUNT\n2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT\n"
           + "3. DISPLAY YOUR CURRENT BALANCE\n4. DISPLAY YOUR PASSBOOK\n5. EXIT");
   int choice=Integer.parseInt(input.readLine());
   serverOutput.println(choice);//Sending the choice to the server
   String result;float amt;
   switch(choice){//Taking inputs from client and sending it to the server based on the choice
    case 1:
        System.out.println("ENTER THE NAME OF THE PERSON TO WHICH AMOUNT HAS TO BE DEPOSITED");
        String n=input.readLine();
        System.out.println("ENTER THE USERNAME OF "+n);
        String un=input.readLine();
        System.out.println("ENTER THE AMOUNT TO BE DEPOSITED");
        amt=Float.parseFloat(input.readLine());
        serverOutput.println(n);
        serverOutput.println(amt);
        result=serverinput.readLine();
        System.out.println(result);//Display a message to client after request is processed
        System.out.print("TRANSACTION DATE AND TIME: ");
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
        LocalDateTime now = LocalDateTime.now();
        System.out.println(dtf.format(now));
        break;
    case 2:
        System.out.println("ENTER THE AMOUNT TO BE WITHDRAWN/DEDUCT");
        amt=Float.parseFloat(input.readLine());
        serverOutput.println(amt);
        result=serverinput.readLine();
        System.out.println(result);//Display a message to client after request is processed
        System.out.print("TRANSACTION DATE AND TIME: ");
        DateTimeFormatter dtf1 = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
        LocalDateTime now1 = LocalDateTime.now();
        System.out.println(dtf1.format(now1));
        break;
```



```
case 3:
    result=serverinput.readLine();
    System.out.println(result);//Display a message to client after request is processed
    break;
case 4:
    result="";
    System.out.println("NAME: "+name+"\nUSERNAME: "+username);
    System.out.println("ENTRIES:");
    System.out.println("WITHDRAW  DEPOSIT  BALANCE");
    while((result.compareTo("Done"))==0)
    {//When it recieves Done as input from server it will print done and terminate loop
    result=serverinput.readLine();
    System.out.println(result);
    }
    break;
case 5:
    serverOutput.println(name);
    serverinput.close();
    input.close();
    serverOutput.close();
    System.out.println("THANK YOU FOR VISTING!!\nBYE BYE!!");
    System.exit(0);
    break;
default://If the choice is not valid
    System.out.println("WRONG CHOICE!!\nPLEASE ENTER YOUR CHOICE AGAIN CORRECTLY");
    serverOutput.println(name);
}}}}}
```

EXPLANATION ➔

In the scenario it was asked to make an application for banking system through distributed system. To make this application we have a bank end as server for bank where all the details for client will be stored and also all the operation that client wants to perform will be stored at server. Client will just give the request for some operation to be performed and server will perform the operation and display the message to the client for the confirmation of the operation. Here I have assumed that client already has an account in the bank so here server will not ask the client to make an account. Client will directly login into the bank server and server will check if the user is present in his bank or not. If the entered name username and password is not valid then a message will be displayed for login failure and program will be terminate. If the login is successful then user will be asked for his/her choice as which operation to be performed- deposit money to another account i.e. transfer money to another account, withdraw money from his account, display current balance, display passbook. These are the operation that can be performed by the user. To make a client server program I have used socket programming in java for multiple client single server socket programming using multithreading. There are many other ways to make client server model such as java RMI, RPC in C, socket programming in c, etc. Here I have used multithreading in order to make single server multiple client program. Program works as- at server side socket is created in with port number as 1534. A socket object is created as server which will be used to accept the client. We have put accept method which allows the client to connect to the server, inside while loop and inside loop thread object is created and using that object thread is created. Whenever a client is connected to the server, a thread will be created. In my program I've given the value of thread as 10 which means that up to 10 clients can join the server. After that thread server class is called and server starts and then overriding occurs which means that a subclass run is provided to perform implementation of the logic.

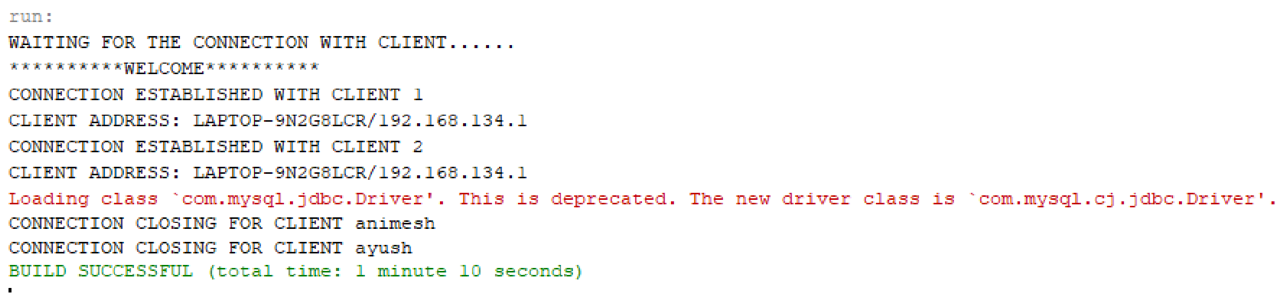
At client side, INET address is defined gives the address of the system from which client is currently working on and also the port number is defined. Socket object is created and passed to the server. Port number is then checked with server's port number and if it matches then client is connected. After the client and server are connected using buffered read, at client side object is created to take input from user and server and at the server side ibject is created to take input from client. name.getInputStream() is used to take input from server or client. And to send data from client to server or server to client PrinterWriter is used and its object is created and name.getOutputStream() is used to accept data sent by client or server from client or server. There are other methods also to read and write data from client and server but I have used input and output stream because here we want the data to be in serialized way. Serialization is used to convert the state of an object into a byte stream which can be sent over a network. It means that in serials client will send the data and server will accept the data. It will be in an order.

After choice is given by client it will be sent to server. According to the choice operation will be performed and database will be updated and after operation is performed, message will be displayed by the server to client for the confirmation of operation performed. I have used SQL database in my program. The order in which client or server sends the data, in that order only server or client will receive the data respectively. So if data is sent in a loop by client then server side also it will accept the data in loop. I have created tables in my database with client's name with attribute as- username, password, withdraw, deposit, balance. So using client's name I will access the database and then operation will be performed and database will be updated. I have used localtimedate method in my program to get the transaction date and time. So after every transaction, date and time of transaction will also be displayed.

In banking transaction, there can be chances that the program has conflict pair and it doesn't satisfy the condition of ACID(ATOMICITY, CONSISTENCY, ISOLATION, DURABILITY). Conflict pair occurs when two or more client at the same time tries to read and write or write and write in database concurrently. So inorder to handle multiple client at the same time I am using commit operation in my program. What it does is that it will permanently store the values of database after client is done with the transaction. If I will put it after every operation, then it will commit the database and later if any changes is made, it will not happen since the database is already committed. If the any of the operation is not performed properly then it will rollback to the starting of the transaction and perform it again. Initially commit is always true so for small program we don't actually need it but for larger program, we will have to commit the data so that the data is not corrupted. To use commit initially we will have to set it false and then use. In my program I've used it outside while loop as it can bee seen in server program. So that after client ends all the data that has been changed is stored into the database and doesn't get corrupted. After client is terminated then other client if wants to transfer money to that client then it will not show any error and data will be updated properly.

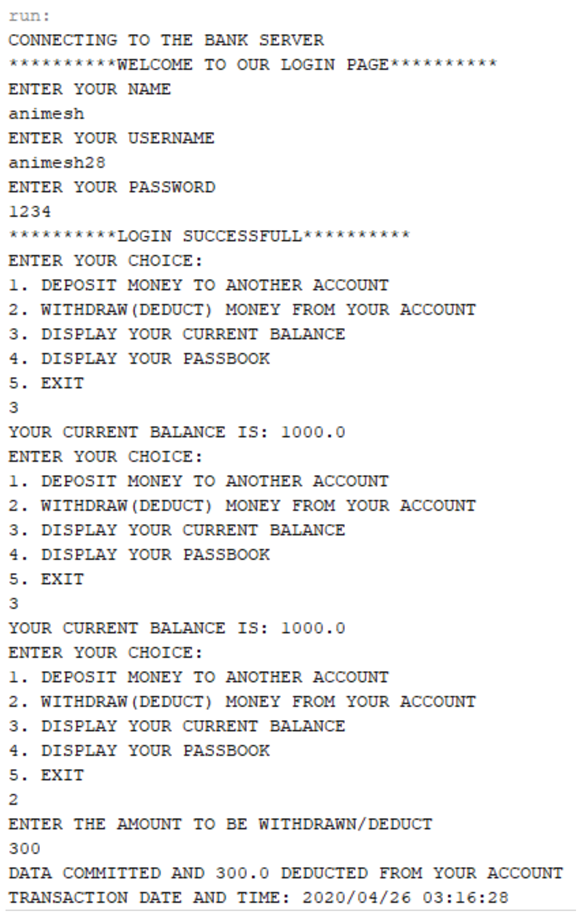
A1.3 Testing of the developed banking application:

(i) **OUTPUT OF SERVER HANDLING TWO CLIENTS CONCURRENTLY➔**

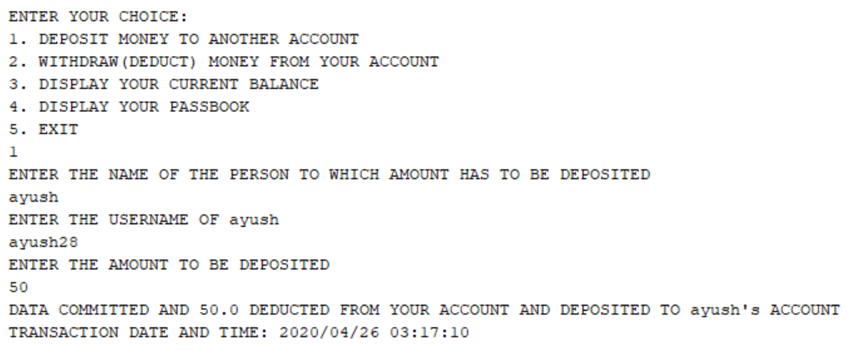


```
run:
WAITING FOR THE CONNECTION WITH CLIENT......
**********WELCOME**********
CONNECTION ESTABLISHED WITH CLIENT 1
CLIENT ADDRESS: LAPTOP-9N2G8LCR/192.168.134.1
CONNECTION ESTABLISHED WITH CLIENT 2
CLIENT ADDRESS: LAPTOP-9N2G8LCR/192.168.134.1
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'.
CONNECTION CLOSING FOR CLIENT animesh
CONNECTION CLOSING FOR CLIENT ayush
BUILD SUCCESSFUL (total time: 1 minute 10 seconds)
```

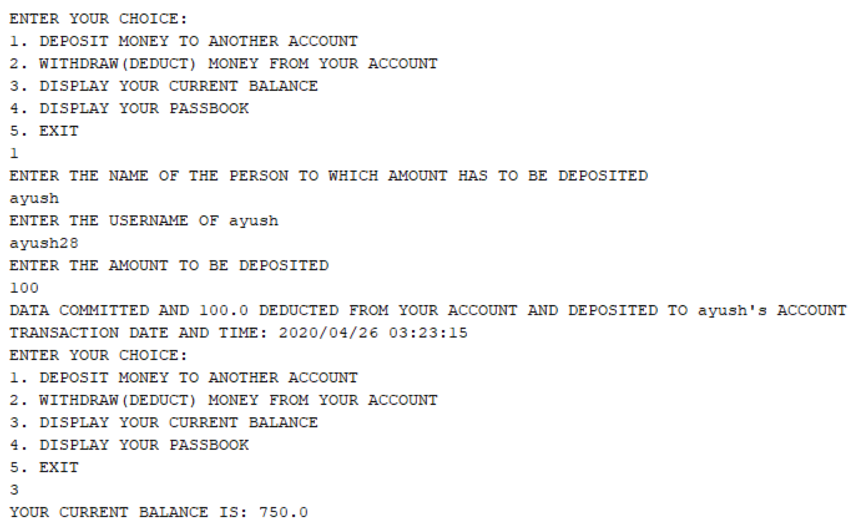
(ii) **OUTPUT FOR CLIENT 1 animesh ➔**



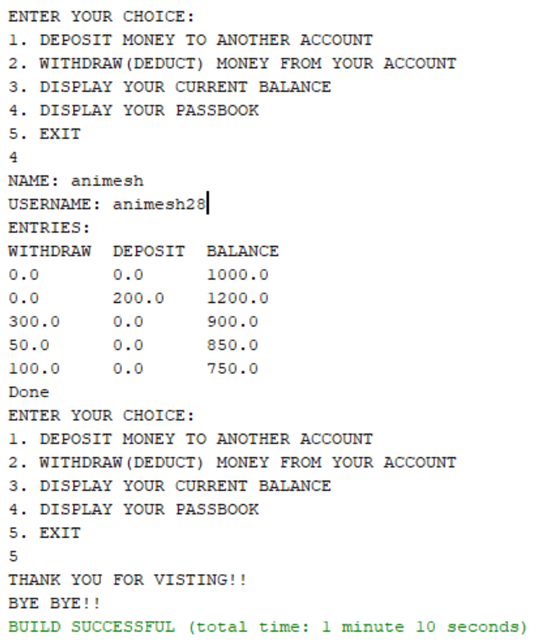
```
run:
CONNECTING TO THE BANK SERVER
**********WELCOME TO OUR LOGIN PAGE**********
ENTER YOUR NAME
animesh
ENTER YOUR USERNAME
animesh28
ENTER YOUR PASSWORD
1234
**********LOGIN SUCCESSFULL**********
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
3
YOUR CURRENT BALANCE IS: 1000.0
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
3
YOUR CURRENT BALANCE IS: 1000.0
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
2
ENTER THE AMOUNT TO BE WITHDRAWN/DEDUCT
300
DATA COMMITTED AND 300.0 DEDUCTED FROM YOUR ACCOUNT
TRANSACTION DATE AND TIME: 2020/04/26 03:16:28
```



```
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
1
ENTER THE NAME OF THE PERSON TO WHICH AMOUNT HAS TO BE DEPOSITED
ayush
ENTER THE USERNAME OF ayush
ayush28
ENTER THE AMOUNT TO BE DEPOSITED
50
DATA COMMITTED AND 50.0 DEDUCTED FROM YOUR ACCOUNT AND DEPOSITED TO ayush's ACCOUNT
TRANSACTION DATE AND TIME: 2020/04/26 03:17:10
```

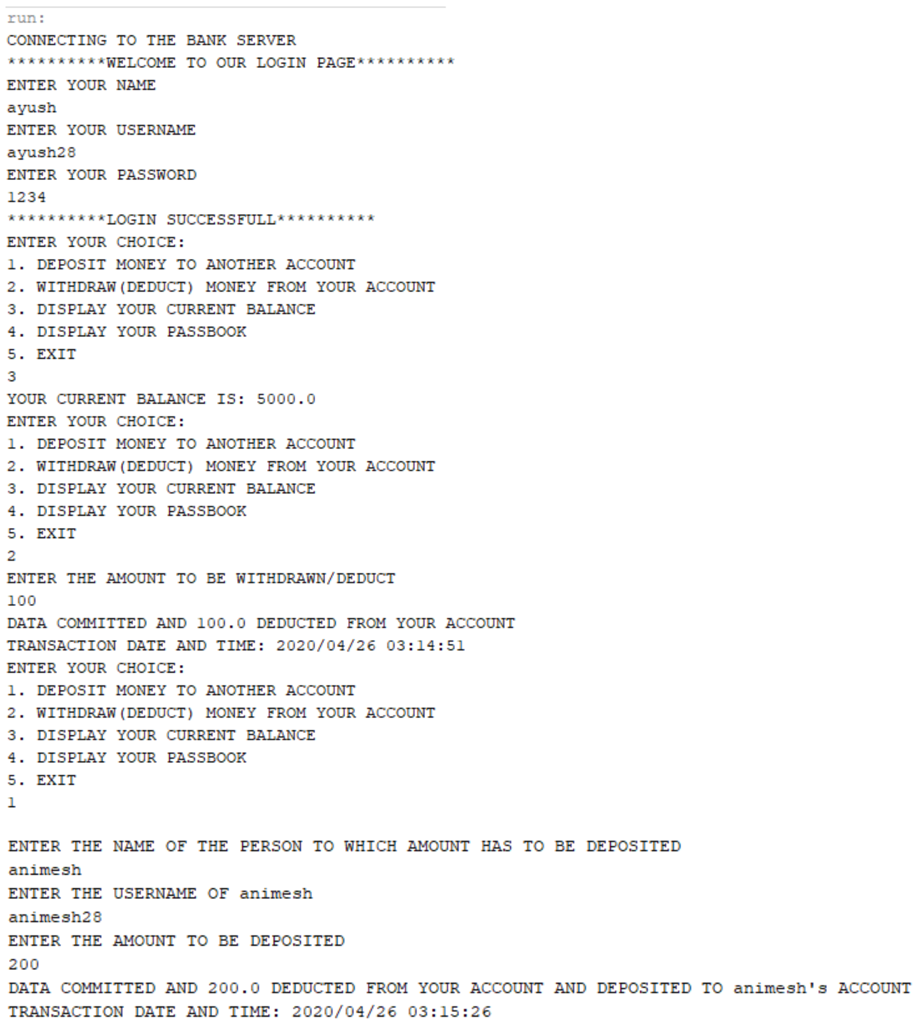


```
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
1
ENTER THE NAME OF THE PERSON TO WHICH AMOUNT HAS TO BE DEPOSITED
ayush
ENTER THE USERNAME OF ayush
ayush28
ENTER THE AMOUNT TO BE DEPOSITED
100
DATA COMMITTED AND 100.0 DEDUCTED FROM YOUR ACCOUNT AND DEPOSITED TO ayush's ACCOUNT
TRANSACTION DATE AND TIME: 2020/04/26 03:23:15
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
3
YOUR CURRENT BALANCE IS: 750.0
```



```
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
4
NAME: animesh
USERNAME: animesh28
ENTRIES:
WITHDRAW  DEPOSIT  BALANCE
0.0       0.0      1000.0
0.0       200.0    1200.0
300.0     0.0      900.0
50.0      0.0      850.0
100.0     0.0      750.0
Done
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
5
THANK YOU FOR VISTING!!
BYE BYE!!
BUILD SUCCESSFUL (total time: 1 minute 10 seconds)
```

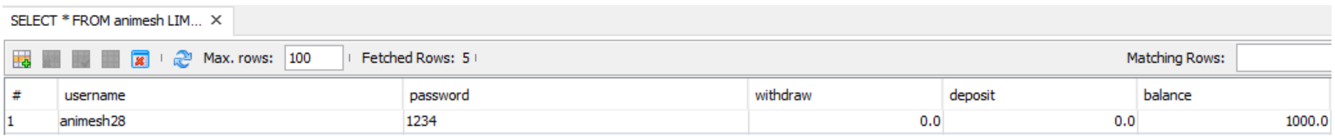
(iii) **OUTPUT FOR CLIENT 2 ayush ➔**



```
run:
CONNECTING TO THE BANK SERVER
**********WELCOME TO OUR LOGIN PAGE**********
ENTER YOUR NAME
ayush
ENTER YOUR USERNAME
ayush28
ENTER YOUR PASSWORD
1234
**********LOGIN SUCCESSFULL**********
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
3
YOUR CURRENT BALANCE IS: 5000.0
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
2
ENTER THE AMOUNT TO BE WITHDRAWN/DEDUCT
100
DATA COMMITTED AND 100.0 DEDUCTED FROM YOUR ACCOUNT
TRANSACTION DATE AND TIME: 2020/04/26 03:14:51
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
1

ENTER THE NAME OF THE PERSON TO WHICH AMOUNT HAS TO BE DEPOSITED
animesh
ENTER THE USERNAME OF animesh
animesh28
ENTER THE AMOUNT TO BE DEPOSITED
200
DATA COMMITTED AND 200.0 DEDUCTED FROM YOUR ACCOUNT AND DEPOSITED TO animesh's ACCOUNT
TRANSACTION DATE AND TIME: 2020/04/26 03:15:26
```

```
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
3
YOUR CURRENT BALANCE IS: 4850.0
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
4
NAME: ayush
USERNAME: ayush28
ENTRIES:
WITHDRAW  DEPOSIT  BALANCE
0.0       0.0      5000.0
100.0     0.0      4900.0
200.0     0.0      4700.0
0.0       50.0     4750.0
0.0       100.0    4850.0
Done
ENTER YOUR CHOICE:
1. DEPOSIT MONEY TO ANOTHER ACCOUNT
2. WITHDRAW(DEDUCT) MONEY FROM YOUR ACCOUNT
3. DISPLAY YOUR CURRENT BALANCE
4. DISPLAY YOUR PASSBOOK
5. EXIT
5
THANK YOU FOR VISTING!!
BYE BYE!!
BUILD SUCCESSFUL (total time: 1 minute 8 seconds)
```

(iv) **INITIAL DATABASE FOR CLIENT 1 animesh→**

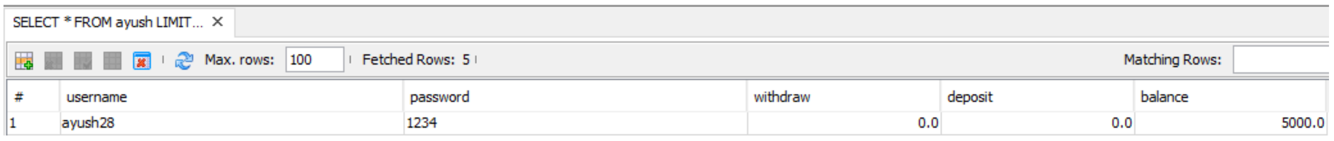


SELECT * FROM animesh LIM... ×

Max. rows: 100 | Fetched Rows: 5 | Matching Rows:

#	username	password	withdraw	deposit	balance
1	animesh28	1234	0.0	0.0	1000.0

(v) **INITIAL DATABASE FOR CLIENT 2 ayush→**

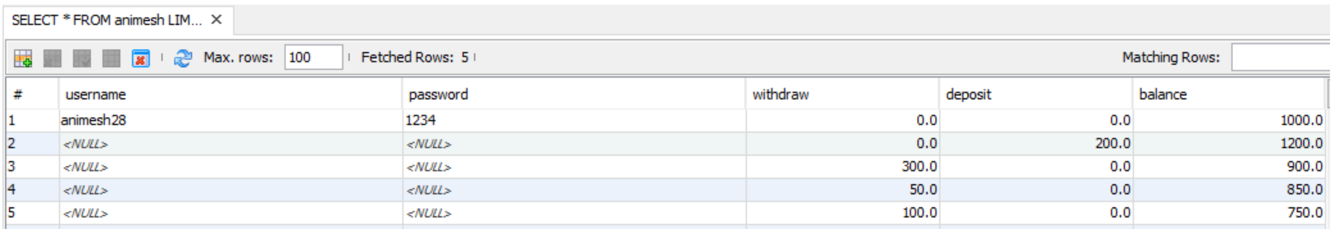


SELECT * FROM ayush LIMIT... ×

Max. rows: 100 | Fetched Rows: 5 | Matching Rows:

#	username	password	withdraw	deposit	balance
1	ayush28	1234	0.0	0.0	5000.0

(vi) **FINAL DATABASE FOR CLIENT 1 animesh→**

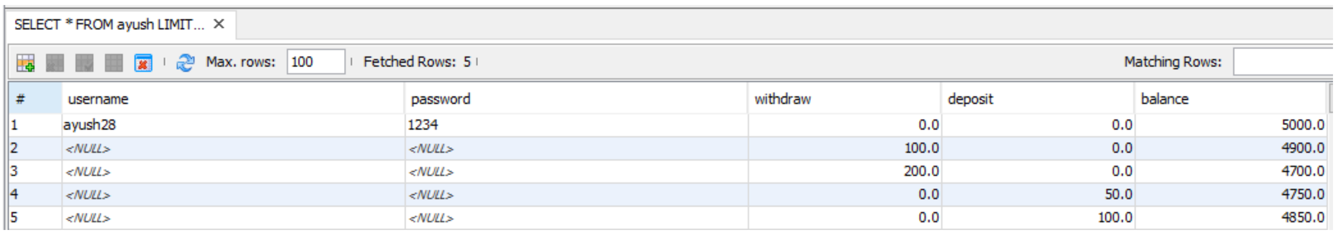


SELECT * FROM animesh LIM... ×

Max. rows: 100 | Fetched Rows: 5 | Matching Rows:

#	username	password	withdraw	deposit	balance
1	animesh28	1234	0.0	0.0	1000.0
2	<NULL>	<NULL>	0.0	200.0	1200.0
3	<NULL>	<NULL>	300.0	0.0	900.0
4	<NULL>	<NULL>	50.0	0.0	850.0
5	<NULL>	<NULL>	100.0	0.0	750.0

(vii) **FINAL DATABASE FOR CLIENT 2 ayush→**

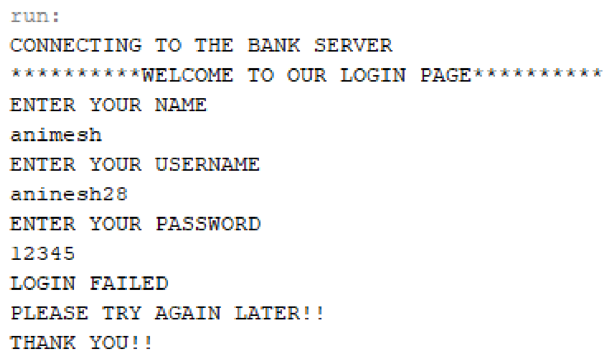


SELECT * FROM ayush LIMIT... ×

Max. rows: 100 | Fetched Rows: 5 | Matching Rows:

#	username	password	withdraw	deposit	balance
1	ayush28	1234	0.0	0.0	5000.0
2	<NULL>	<NULL>	100.0	0.0	4900.0
3	<NULL>	<NULL>	200.0	0.0	4700.0
4	<NULL>	<NULL>	0.0	50.0	4750.0
5	<NULL>	<NULL>	0.0	100.0	4850.0

(viii) **WHEN CLIENT USERNAME OR PASSWORD IS WRONG**➔



```
run:
CONNECTING TO THE BANK SERVER
**********WELCOME TO OUR LOGIN PAGE**********
ENTER YOUR NAME
animesh
ENTER YOUR USERNAME
aninesh28
ENTER YOUR PASSWORD
12345
LOGIN FAILED
PLEASE TRY AGAIN LATER!!
THANK YOU!!
```

EXPLANATION OF THE ABOVE RESULTS AND TESTING➔

After implementing the program, we have tested our program by giving different outputs. From part (iv) and (v) initial database of client is shown. Here I am assuming that client already exists in my bank server. So here I am considering only two client and showing the output. First server is ran and then both the client class is ran. Whenever client class is run at server side a message will be displayed that client is connected. Now after clients are connected to the server- name, username and password are taken input from the clients and sent to the server and server will then validate the username and password. If the entered username or password is wrong then it will terminate the program as it can be seen in part (viii). If it is correct then it will ask for the choice. According to the choice operation will be performed.

The main objective of this program is to not have conflict pairs in transactions and should satisfy the property of ACID. So for that I have give input in such a way that read-read, read-write, write-read and write-write operations are performed by client 1 and client 2 simultaneously. Part (ii) shows output for client 1 i.e. animesh and part (iii) shows output for client 2 i.e. ayush.

As we can see that first client 1 is reading and client 2 is also reading the database and displaying the current balance. Then client 1 is reading and client 2 is writing i.e. displaying current balance and depositing money to another account (i.e. transferring money) respectively. Then client 1 is performing write and client 2 is performing read operation. Then client 1 is writing and client 2 is also writing. Then at last read operation is performed by both the clients i.e. passbook is getting printed for both the client. Then client has selected an option to terminate the program so the program is terminated and at the server side as it can be seen, an output is displayed saying that “connection closing for client animesh” and connection “closing for client ayush”. After the all the operations are performed by the server that was requested by client, at server side data will be committed and values will be stored to the database without any connection. If there is any error in the connection or error in the sql connection or error in committing data then catch block will run and that exception will be printed. Here I am showing output for only two clients but in this program more than two clients can also access the server at the same time.

So from the above figures we saw that multiple client accessed the server at the same time and server handled each client and processed the request of each client without any error. Hence the program is successfully executed and multiple clients are being handled properly by single server when accessed at the same time without any error.

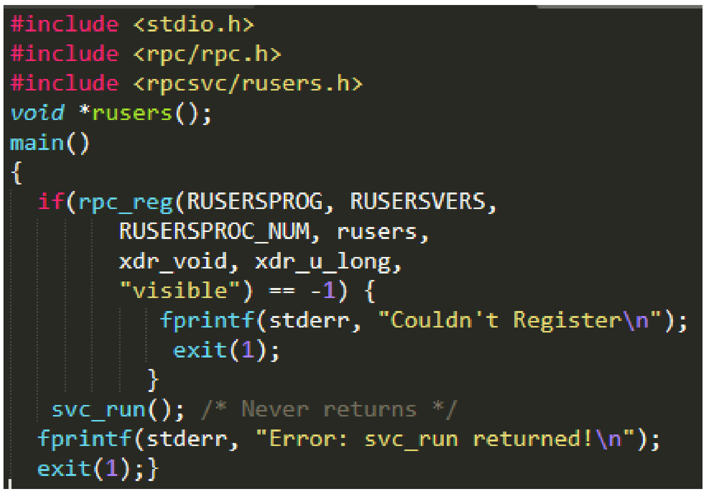
Solution to Question No. B:

B1.1 RPC, the need for it and two sample programs:

RPC stands for Remote Procedural Calls. By the name itself we can tell that RPC is used by procedural languages. Remote Procedural calls are a form of inter-process communication which has different processes with different address spaces. RPC uses client server model in which one program can request a service(which will act as a client here) from a program located in another computer on a different network(Server which is receiving the request and proving service to the client). Subroutine call and functional call are same as procedural call. In distributed computing, RPC is used when program calls procedure or subroutine to execute the program which has different address space. The applications can run on different machines also so for that we will need RPC method in order to perform it. RPC will define a network protocol where message passing will happen. Serialization and deserialization is also provided by RPC and also code abstraction. In RPC, calling arguments are passed to the remote procedure and the caller will then wait for a response to be returned from the remote procedure. In client server model, client will make a procedural call which will send the request to the server and waits. RPC is used because it makes it easier to develop an application that includes multiple programs which are distributed in a network. Two sample programs for RPC are as below:-

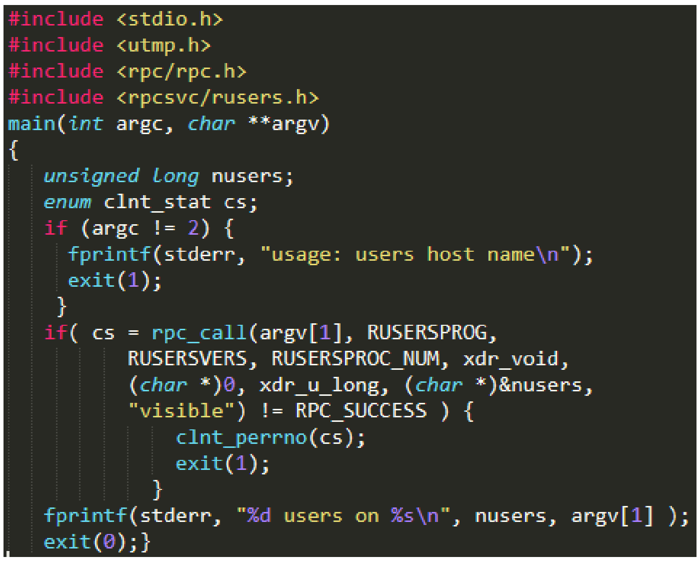
a) Client Server Program using RPC: Program is basically just request reply protocol where client is sending message through procedural calls and waits for the reply from the server.

(i) Server Side:- It calls rpc_reg() to register the procedure to be called and then it calls svc_run() which dispatches the RPC library.



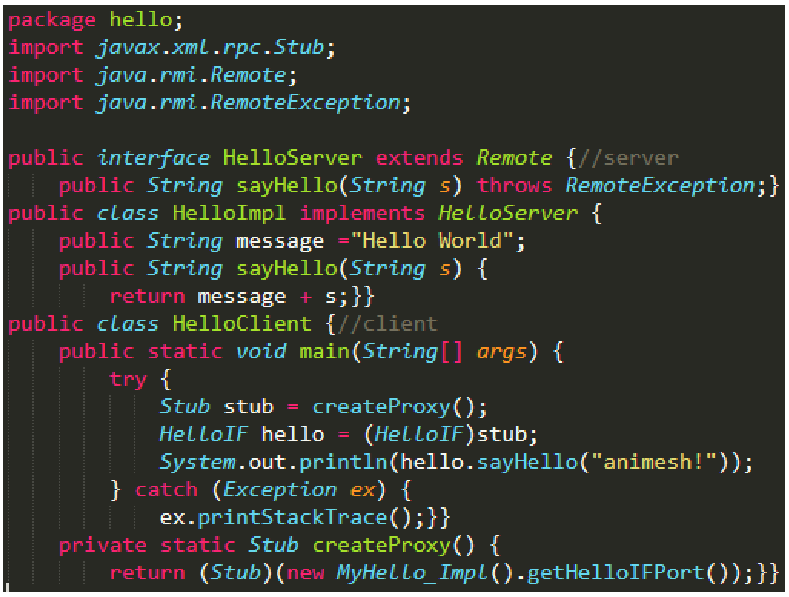
```
#include <stdio.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>
void *rusers();
main()
{
  if(rpc_reg(RUSERSPROG, RUSERSVERS,
        RUSERSPROC_NUM, rusers,
        xdr_void, xdr_u_long,
        "visible") == -1) {
           fprintf(stderr, "Couldn't Register\n");
            exit(1);
          }
   svc_run(); /* Never returns */
  fprintf(stderr, "Error: svc_run returned!\n");
  exit(1);}
```

(ii) Client Side:- It has only one function call rpc_call(). After this function is called it will be blocked until message is received. After server receives, it returns RPC_SUCCESS with value 0. If it’s not success then it’ll not return 0.



```
#include <stdio.h>
#include <utmp.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>
main(int argc, char **argv)
{
   unsigned long nusers;
   enum clnt_stat cs;
   if (argc != 2) {
     fprintf(stderr, "usage: users host name\n");
     exit(1);
    }
   if( cs = rpc_call(argv[1], RUSERSPROG,
          RUSERSVERS, RUSERSPROC_NUM, xdr_void,
          (char *)0, xdr_u_long, (char *)&nusers,
          "visible") != RPC_SUCCESS ) {
              clnt_perrno(cs);
              exit(1);
            }
   fprintf(stderr, "%d users on %s\n", nusers, argv[1] );
   exit(0);}
```

b) A simple Hello world program using RPC: Here a java program is made using RPC concept where message Hello World is printed. A stub method is invoked by client which will create connections with the server and though procedural calls that is by passing address space, message will be sent and wait till the server replies and once the server replies, it will receive the message by server and display it to client.



```
package hello;
import javax.xml.rpc.Stub;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloServer extends Remote {//server
    public String sayHello(String s) throws RemoteException;}
public class HelloImpl implements HelloServer {
    public String message ="Hello World";
    public String sayHello(String s) {
        return message + s;}}
public class HelloClient {//client
    public static void main(String[] args) {
        try {
            Stub stub = createProxy();
            HelloIF hello = (HelloIF)stub;
            System.out.println(hello.sayHello("animesh!"));
        } catch (Exception ex) {
            ex.printStackTrace();}}
    private static Stub createProxy() {
        return (Stub)(new MyHello_Impl().getHelloIFPort());}}
```

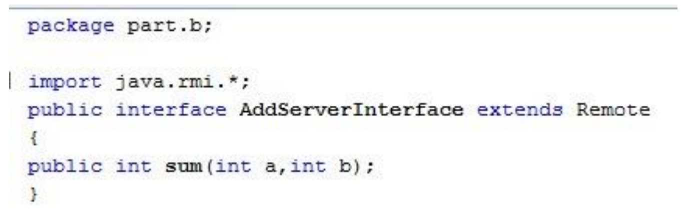
B1.2 RMI, the need for it and two sample programs:

RMI stands for Remote Method Invocation which works with object oriented programming language. JAVA RMI is one of the RMI where this method is used. The method is to allow the object of one system that is currently running on one java virtual machine to invoke or access an object that is running on another java virtual machine due to which it provides remote communication between java programs. Stub and skeleton are two objects that provides remote communication between the applications. A remote object is an object whose method can be invoked from another JVM. A stub acts as a gateway for the client side.

Whenever there is a connection with remote JVM then it writes and transmits the parameter to JVM and waits for the result. Then at the server side the client request is read and then after processing the request it returns the value to the client. Writing and transmitting of data at the client side is called marshaling and reading the data at the server side is called unmarshalling. Skeleton is a an object that acts as a gateway at the server side. Whenever a request is received by the skeleton, it reads the parameter for the remote method and then invokes it on the actual remote object. After performing operation it writes and transmits i.e. marshals the result to the client. There are many application for the remote method invocation. Following are the two sample programs:

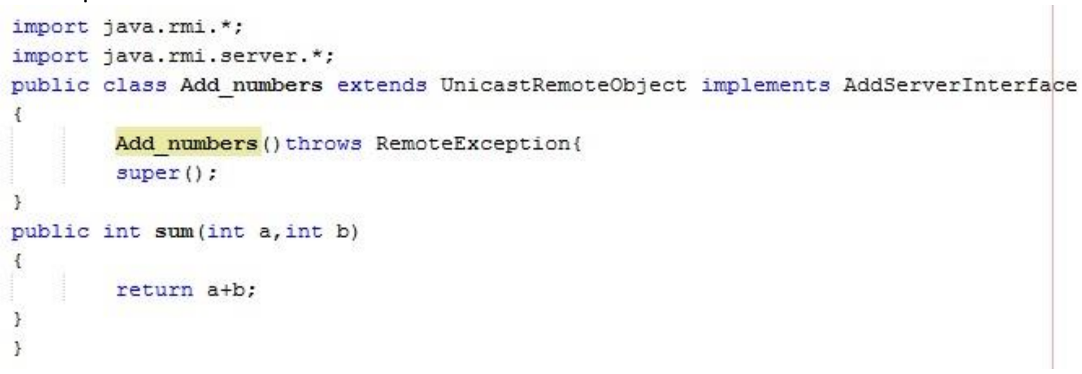
a) Below is a program that is has normal client server connection. Following are the steps with screenshots which shows the method to create a JAVA RMI program.

   (i) Defining remote interface:



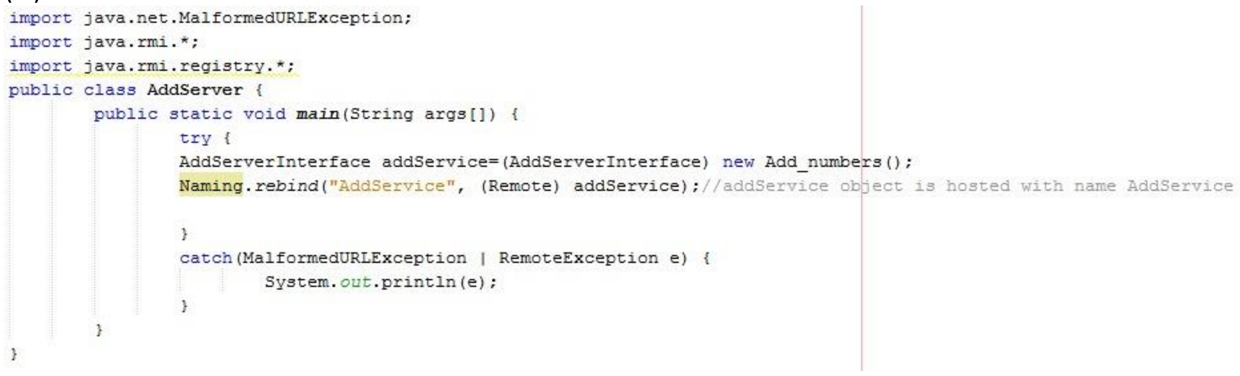
```
package part.b;

import java.rmi.*;
public interface AddServerInterface extends Remote
{
public int sum(int a,int b);
}
```

   (ii) Implementation of the remote interface:



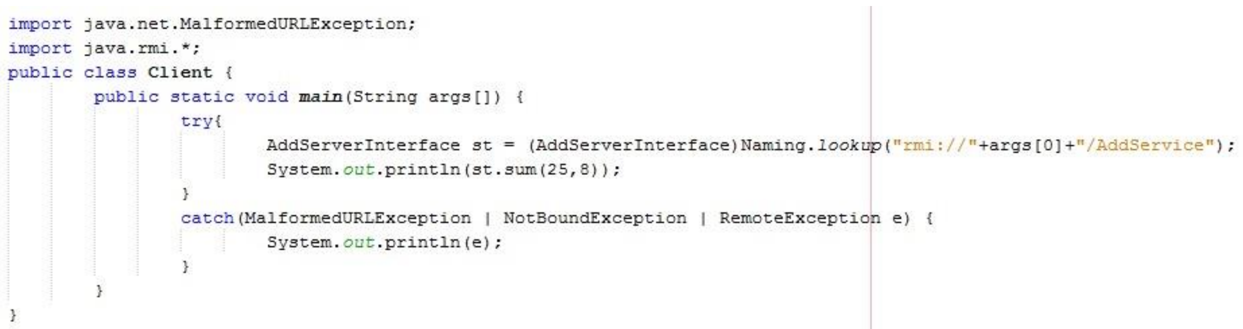
```
import java.rmi.*;
import java.rmi.server.*;
public class Add_numbers extends UnicastRemoteObject implements AddServerInterface
{
        Add_numbers()throws RemoteException{
        super();
}
public int sum(int a,int b)
{
        return a+b;
}
}
```

   (iii) Create AddServer and host RMI service:



```
import java.net.MalformedURLException;
import java.rmi.*;
import java.rmi.registry.*;
public class AddServer {
        public static void main(String args[]) {
                try {
                AddServerInterface addService=(AddServerInterface) new Add_numbers();
                Naming.rebind("AddService", (Remote) addService);//addService object is hosted with name AddService

                }
                catch(MalformedURLException | RemoteException e) {
                        System.out.println(e);
                }
        }
}
```

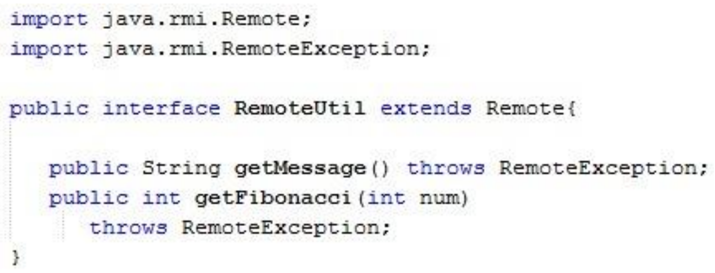
   (iv) Now create client application:



```
import java.net.MalformedURLException;
import java.rmi.*;
public class Client {
        public static void main(String args[]) {
                try{
                        AddServerInterface st = (AddServerInterface)Naming.lookup("rmi://"+args[0]+"/AddService");
                        System.out.println(st.sum(25,8));
                }
                catch(MalformedURLException | NotBoundException | RemoteException e) {
                        System.out.println(e);
                }
        }
}
```

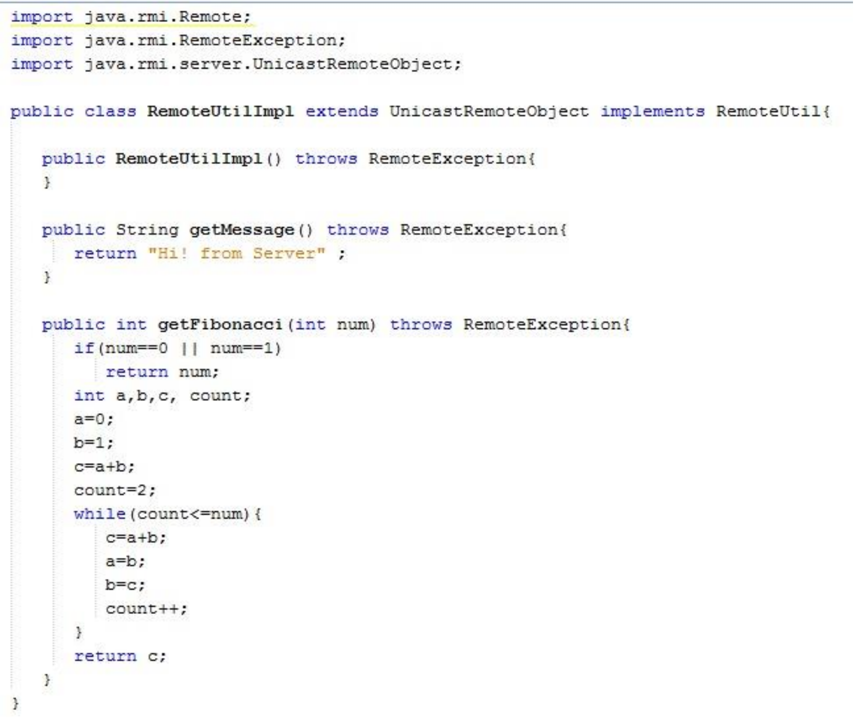
b) Following is a sample program for Fibonacci series using java RMI. In this also same steps what we did for client server, we will do it here also.

(i) Creating remote interface:



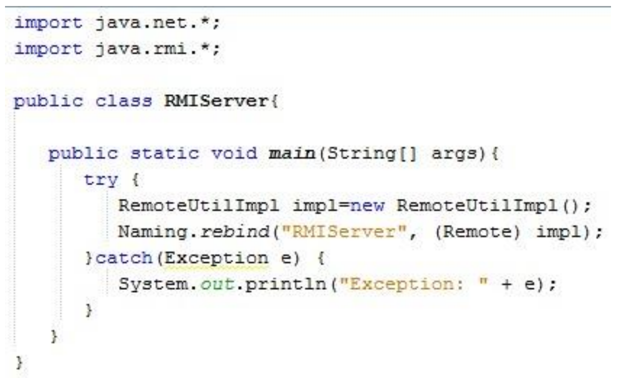
```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RemoteUtil extends Remote{

   public String getMessage() throws RemoteException;
   public int getFibonacci(int num)
      throws RemoteException;
}
```

(ii) Implementing a remote interface:



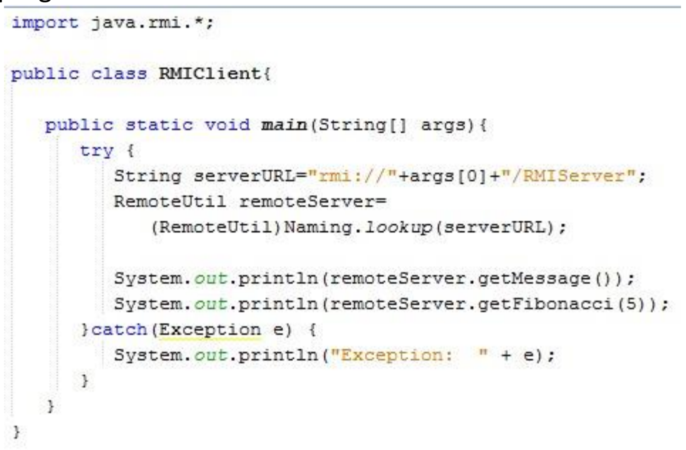
```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RemoteUtilImpl extends UnicastRemoteObject implements RemoteUtil{

   public RemoteUtilImpl() throws RemoteException{
   }

   public String getMessage() throws RemoteException{
      return "Hi! from Server" ;
   }

   public int getFibonacci(int num) throws RemoteException{
      if(num==0 || num==1)
         return num;
      int a,b,c, count;
      a=0;
      b=1;
      c=a+b;
      count=2;
      while(count<=num){
         c=a+b;
         a=b;
         b=c;
         count++;
      }
      return c;
   }
}
```

(iii) Creating a server program:



```
import java.net.*;
import java.rmi.*;

public class RMIServer{

   public static void main(String[] args){
      try {
         RemoteUtilImpl impl=new RemoteUtilImpl();
         Naming.rebind("RMIServer", (Remote) impl);
      }catch(Exception e) {
         System.out.println("Exception: " + e);
      }
   }
}
```

(iv) Creating a client program:



```
import java.rmi.*;

public class RMIClient{

   public static void main(String[] args){
      try {
         String serverURL="rmi://"+args[0]+"/RMIServer";
         RemoteUtil remoteServer=
            (RemoteUtil)Naming.lookup(serverURL);

         System.out.println(remoteServer.getMessage());
         System.out.println(remoteServer.getFibonacci(5));
      }catch(Exception e) {
         System.out.println("Exception:  " + e);
      }
   }
}
```

So from the above examples we can see that first a remote interface is defined and then implemented and after doing that server and client program is made. Using RMI registry method is invoked and client server program interacts with each other.

B1.3 Similarities and differences between RPC and RMI:

Similarities between RMI and RPC is that both of them supports programming with interface. RPC and RMI both of them works similar to request reply protocol which is similar to client server model. Also both RMI and RPC has same level of transparency i.e. local and remote calls has same syntax. In both RMI and RPC, client sends proxy to the server but the only difference in sending the proxy is that RMI invokes method and RPC invokes functions to send proxy from client.

Difference between RMI and RPC is that RMI supports object oriented programming language whereas RPC supports procedural programming language. Thus it makes RMI based on java platform and RPC is OS and library dependent platform. RMI is more efficient than RPC since RMI is the new updated and better version of RPC. In RMI we pass objects as parameters and in RPC we pass normal data or message as a parameter. Since RMI is object oriented, thus it provides robustness to the client whereas RPC doesn’t provide security to the client’s data or information.