Table of Contents

Introduction 2
Basic Requirements 2
Additional Requirements 3
Assumptions 3
System Requirements 5
Basic requirements 5
Additional Requirements 6
System Flow 6
Description 6
Requirements to Modelling 7
Entities 7
Events 7
Interactions 7
Finite State Processes Statements 8
Structure Diagram 9
UML Class Diagram 10
Modelling to Implementation 11
Evaluation and Testing 27
Depth of Discussion 30
Sections that met the requirements 30
Sections that didn't meet the requirements 32
Critical Appraisal 33
Limitations of developed system 33
Future Enhancements 33
Conclusion 34
References 34

Introduction

The client/server architecture is a widespread one. Modern web servers, the banking sector and many operating systems use a client server model. Most of these systems allow for multiple clients to interact with a server at once, and hence require the server to be safe when operating concurrently.

The software to be designed will have a customer console (keyboard and display) for interaction with the customer. The Console will communicate with the bank's computer over an appropriate communication link.

The console will service one customer at a time. A customer will be required to enter an Account number and personal identification number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions.

Basic Requirements

The console must be able to provide the following services to the customer:

- A customer must be able to login into the system with the help of a username and a password. Username would be the email-id of the customer and any password of more than 8 characters can be chosen by the user.

- A customer should also be able to create a login account by going to the registration by filling his particular. He has to fill his name, age, email-id, address, account number, account type etc. for the registration. After registration/creation of login account customer would be able to login into the system.

- A customer must be able to make a cash withdrawal from the account linked to the login, in multiples of Rs.100.00. Minimum balance of Rs.1000 is required to be maintained in case of saving account and Minimum balance of Rs.500 is required in case of current account. If a customer tries to overdraft his/her account proper message should be displayed and he/ she will charged with a penalty of Rs.100

- A customer must be able to make a balance inquiry of any account linked to the login.

- A customer must be able to abort a transaction in progress by pressing the Cancel key instead of responding to a request from the machine.

- If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to

successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back.

- If a transaction fails for any reason other than an invalid PIN, the console will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

- The console will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account ("to" account for transfers).
- The console will also maintain an internal log of transactions to facilitate resolving ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the console is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected), for the dispensing of cash, and for the receiving of an envelope. Log entries may contain card numbers and amounts, but for security will *never* contain a PIN.

The console will communicate each transaction to the bank and obtain verification that it was allowed by the bank. Ordinarily, a transaction will be considered complete by the bank once it has been approved.

Additional Requirements

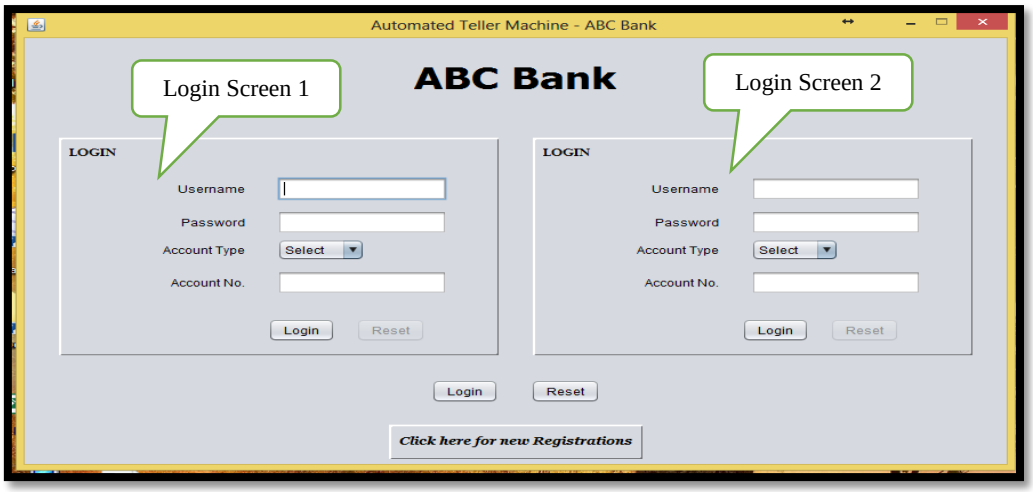
The overall task for this assignment is to build a client/server application to demonstrate your knowledge of the types of problems that are particular to concurrent programming.
While you are free to choose the domain of your problem, you must choose one which

- Requires that multiple clients access the server concurrently (i.e. your solution should use concurrent programming)
- Has the possibility of more than one client modifying a given resource at one time (i.e. your solution will need to be able to handle the issue of data corruption)
- Has the possibility of more than one client requiring locks on more than one resource at a given time (i.e. your solution will need to be able to handle the issue of deadlock)

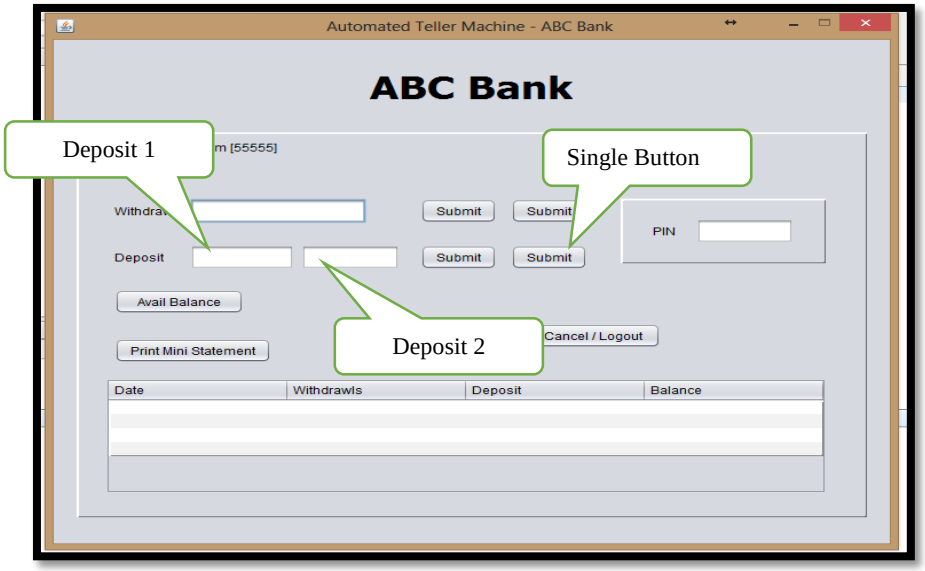
Assumptions

The various assumptions which were taken to make this system are:

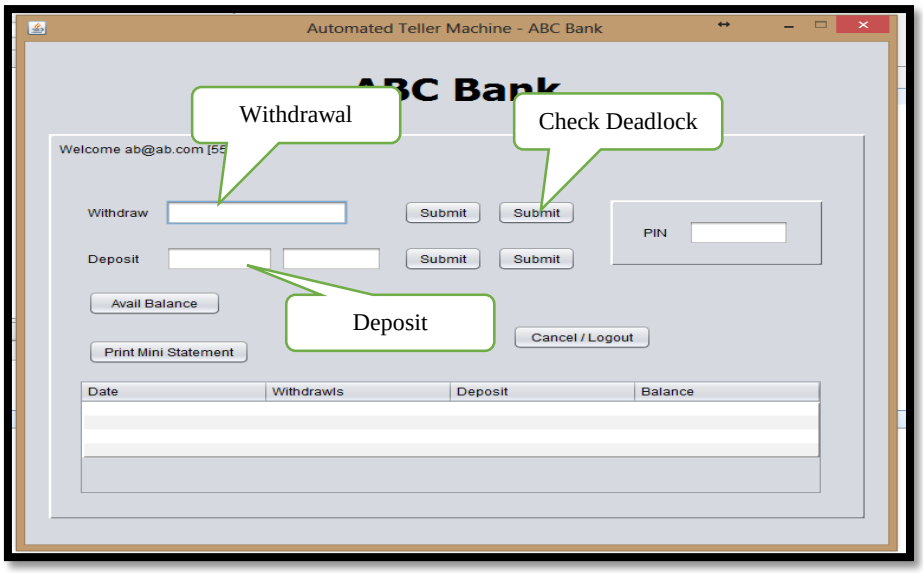
- The user logins through a registered email address, password and account number rather than the traditional ATM card insertion and PIN.
- To show concurrency, the Login screen contains the login panel for two users.



- To show that the system is free from data interference, on click of a single button two deposit actions is executed.



- To show that the system is free from deadlock, the following user interface is given:



- Though the system demands that a receipt be printed after every successful transactions. However, to achieve this the transaction details are saved to a txt file.

System Requirements

Basic requirements

The ATM console must be able to provide the following basic functionalities

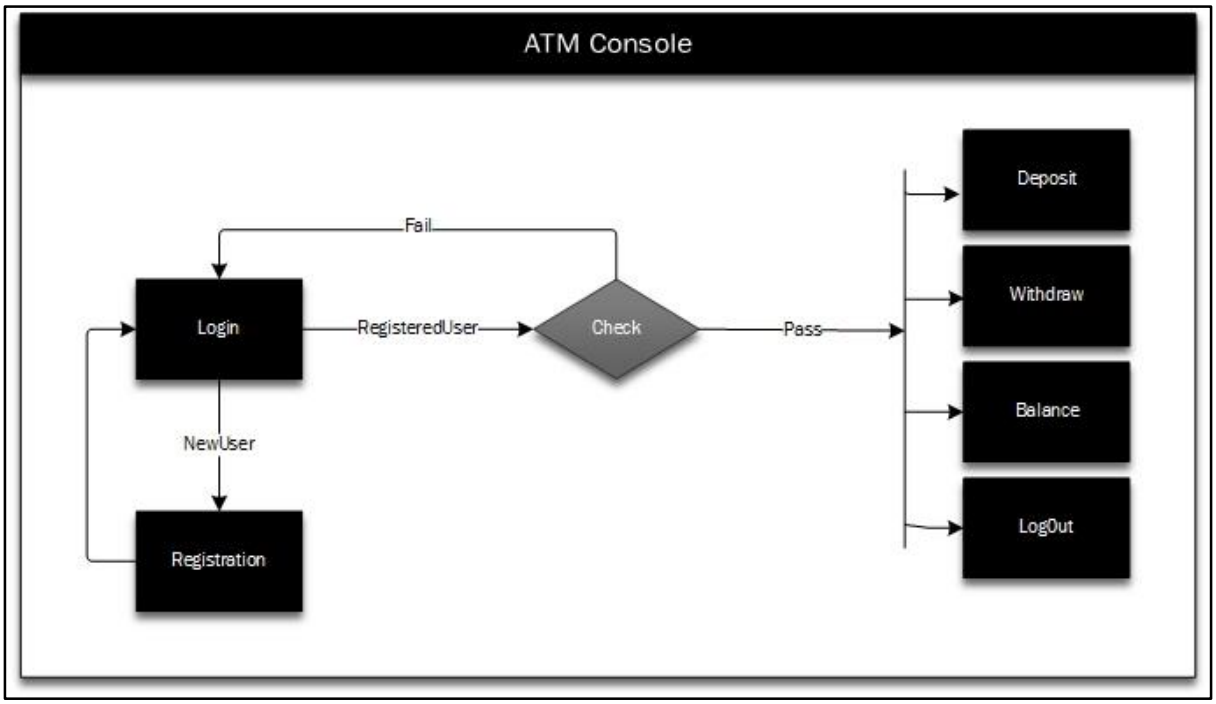
S No.	**Requirements**
1.	The customer should be able to login with the registered email-id and password [more than 8 characters]. Also, the users account type and account number would also be required during login.
2	For new users, they should be able to create a login account with details such as name, email-id, password, address, dob, mobile, account type, account number and pin.
3.	On successful login, customer can withdraw money in the multiples of 100. Minimum balance of 1000 and 500 to be maintained in the saving and current accounts respectively. On overdraft limit, Rs.100 to be deducted per transaction.
4.	Customers can check for the balance of their accounts
5.	A PIN would be asked for before every transaction. On entry of wrong PIN for 3 times will block the account.
6.	Appropriate message to be shown if transaction fails.
7.	Receipt to be printed after each successful transaction showing date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account.
8.	Transactions can be aborted at any time by the user
9.	An internal log file to be maintained.

Additional Requirements

The ATM console must be able to achieve the following additional functionalities:

S No.	**Requirements**
1.	To show concurrency : multiple access to clients
2.	To show how to handle issue of data corruption
3.	To show how to handle issue of deadlock

System Flow



Description

From the above diagram, it can be seen that the first screen of the application is the Login screen. If the user is already registered, they can fill in their credentials, and if the credentials are correct they would have an option to perform Deposit, Withdrawal, Check balance and/or Logout. If the user is already not registered, they can get themselves registered and on successful registration they are forwarded to the Login screen, and the whole process is repeated.

Requirements to Modelling

Entities

Entities	**Description**
Login	Registered users use email, password , account type and number to login to the system
.Users	The main users of the system
Account	For withdrawal and deposits of the attached accounts
Receipt	Print receipt of general transactions

Events

Events	**Description**	**Entities Involved**
Login	Registered users use email, password , account type and number to login to the system to use its functionalities	Users
Register	New users can get themselves registered with correct details	Users
Deposit	Users can deposit money to their accounts and print receipts	Users, Account, Receipt
Withdrawal	Users can withdraw money from their accounts and print receipts	Users, Account, Receipt
Balance	Users can check for balances in their accounts	Users, Account, Receipt
Logout	Users can logout of their accounts at any time	Users

Interactions

S No.	**Users using ATM**
1.	The console will service one User at a time
2.	Registered users can login into the system
3.	New users can get themselves registered
4.	User can check their balance
5.	Users can deposit money
6.	Users can withdraw money
7.	User need to enter a PIN before every transaction. If PIN is entered wrong more than 3 times, account is blocked
8.	Users can cancel any transactions

Finite State Processes Statements

S No.	**Requirements**
1.	The customer should be able to login with the registered email-id and password [more than 8 characters]. Also, the users account type and account number would also be required during login.
FSP	

```
USER=(enter_email->enter_pswd->enter_acctype->enter_accno->VALIDATELOGIN),
VALIDATELOGIN=(invalid_login->USER|valid_login->do_transactions->logout->USER)
||MULTI_USER=({a,b}:USER).
```

2.	For new users, they should be able to create a login account with details such as name, email-id, password, address, dob, mobile, account type, account number and pin.
FSP	

```
REGISTER=(name->NAME),
NAME=(email->EMAIL),
EMAIL=(pswrd->PSWRD),
PSWRD=(invalidpswd->EMAIL|cpswrd->CPSWRD),
CPSWRD=(nomatch->PSWRD|address->acctype->ACCTYP),
ACCTYP=(acctno->ACCTNO),
ACCTNO=(notuniq->ACCTYP|pin->PIN),
PIN=(invalidpin->ACCTNO|registered->STOP).
```

3.	On successful login, customer can withdraw money in the multiples of 100. Minimum balance of 1000 and 500 to be maintained in the saving and current accounts respectively. On overdraft limit, Rs.100 to be deducted per transaction.
FSP	

```
WITHDRAW=(login->accno->pin->WD),
WD=(amt->(not100->error->WD
          |multiple100->(sacct->(less1000->fine->FINE|
                                more1000->withdraw->STOP)
                        |cacct->(less500->fine->FINE
                                |more500->withdraw->STOP)))),
FINE=(penalty->STOP).
```

4.	Customers can check for the balance of their accounts
FSP	

```
BAL=(login->accno->checkBAL->VALIDATE),
VALIDATE=(correctPIN->dispbalance->BAL
          |incorrectPIN->errormassage->VALIDATE).
```

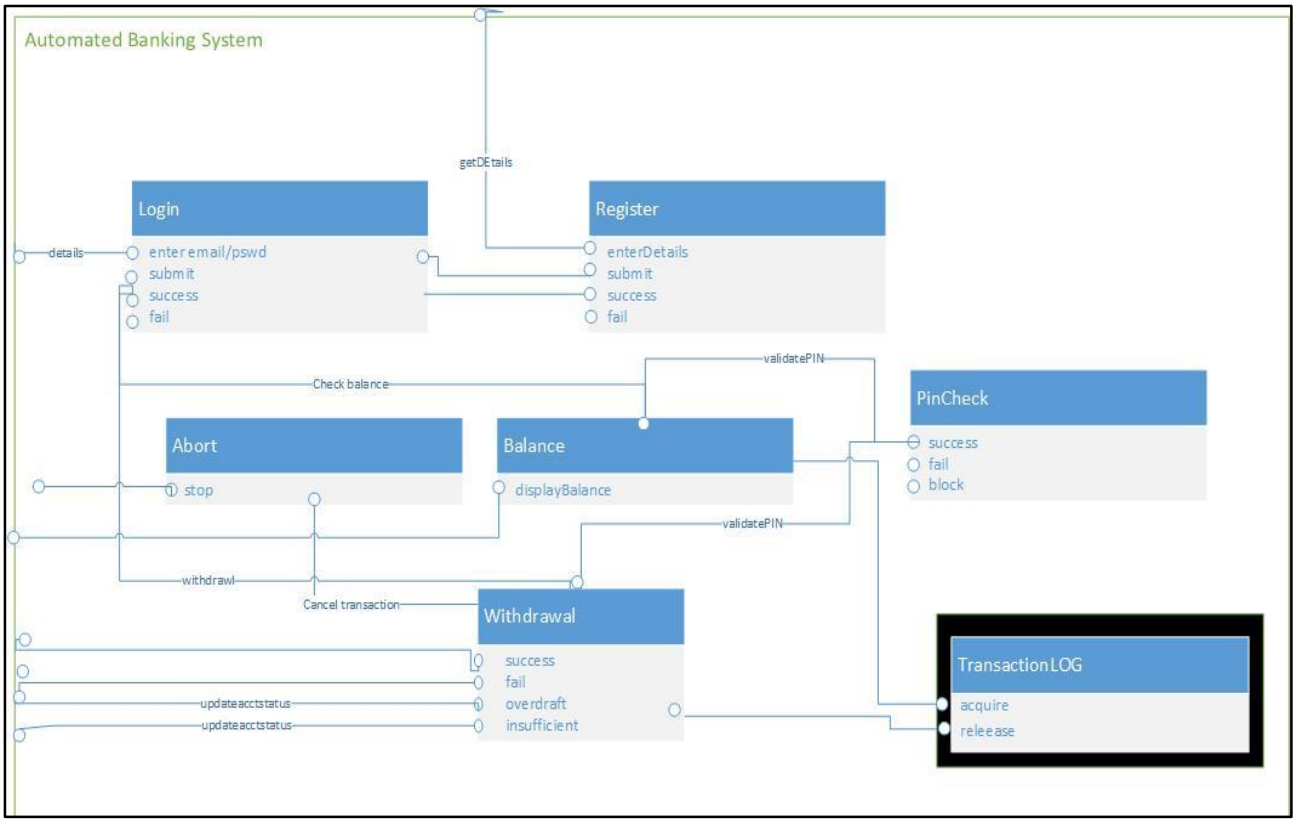
5.	A PIN would be asked for before every transaction. On entry of wrong PIN for 3 times will block the account.
FSP	

```
range T=0..2
range R=0..1
PIN =PIN[0],
PIN[i:T]=(pinval->STOP|when(i<3)pininval->PININC[i]),
PININC[i:R]=(triesleft->PININC[i+1]|pinval->PIN),
PININC[2]=(acctblocked->STOP).
```

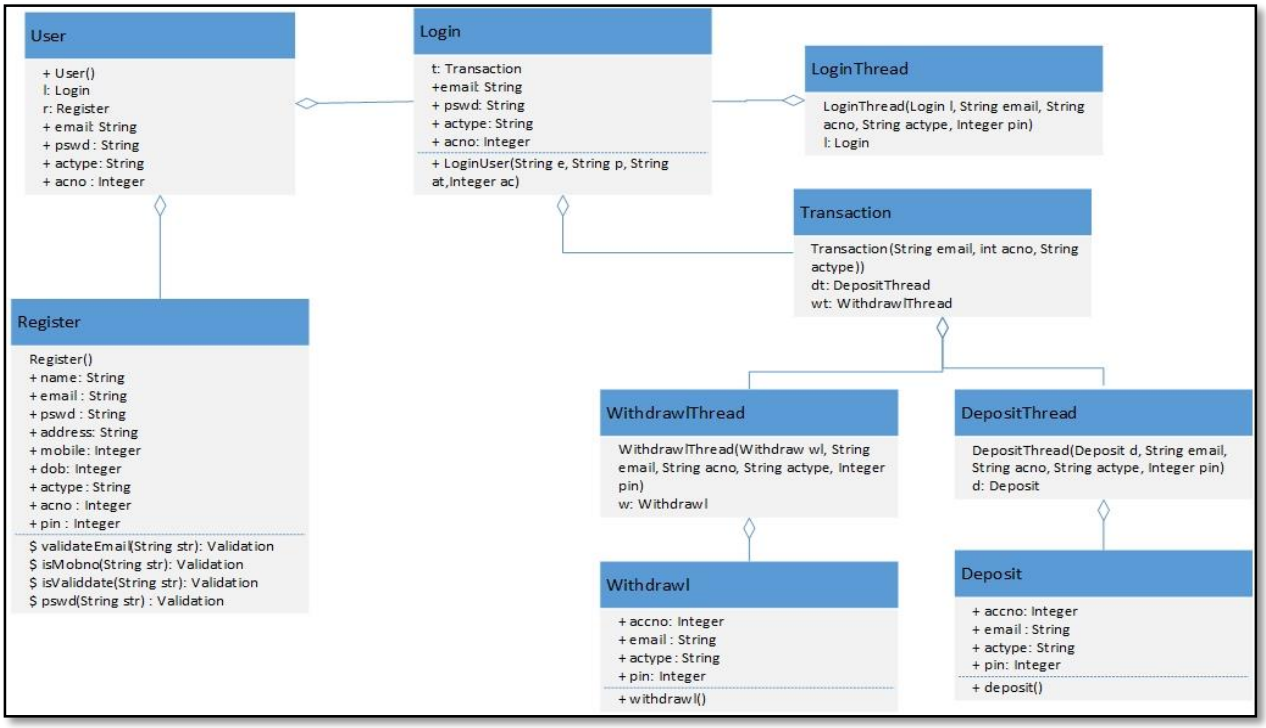
7.	Receipt to be printed after each successful transaction showing date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account.

FSP	`RECEIPT=(transaction->(success->RECEIPT)` `|fail->STOP).`
8.	Transactions can be aborted at any time by the user
FSP	`TRANSACTION=(transaction->cancel->TRANSACTION` `|transaction->inprocess->STOP).`

Structure Diagram



UML Class Diagram



Description: In the above UML Class diagram, the USER class has objects of the Login and Register classes. The LoginThread class which extends Thread, has the object of Login class and is used to create multiple threads for Login class through its method LoginUser(). The classes WithdrawlThread and DepositThread have the object of Withdrawal and Deposit classes. And these two classes in turn are the objects in Transaction class.

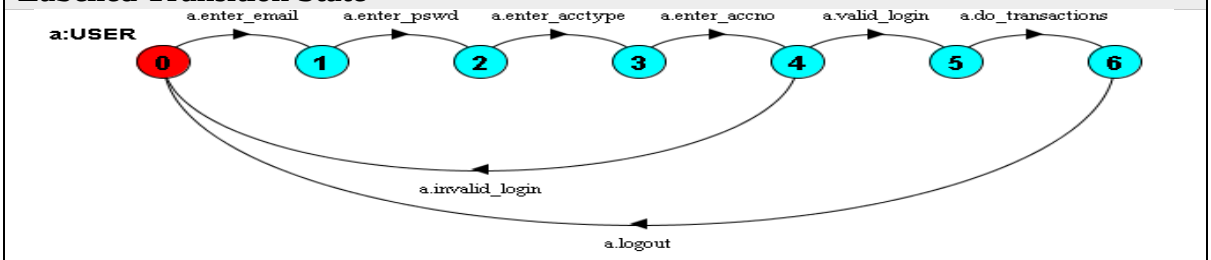
Modelling to Implementation

Task 1 : LOGIN

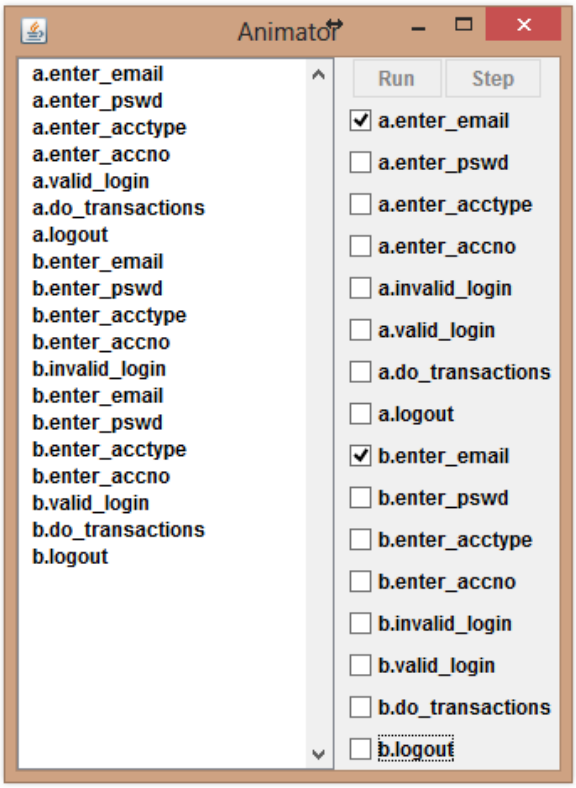
Finite State Process

```
USER=(enter_email->enter_pswd->enter_acctype->enter_accno->VALIDATELOGIN),
VALIDATELOGIN=(invalid_login->USER|valid_login->do_transactions->logout->USER)
||MULTI_USER=({a,b}:USER).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
            while (res.next()) {
            u1 = res.getString("email");
            p1 = res.getString("password");
            name = res.getString("name");
    }
            if (at==1)
            {
                res1 = st.executeQuery("SELECT * from SACCT where email='"+u+"' and accno='"+an1+"'");
                while (res1.next()) {
                    an=res1.getString("accno");
                    st1=res1.getString("status");
                }
            }
            else if (at==2){
                res2 = st.executeQuery("SELECT * from CACCT where email='"+u+"' and accno='"+an1+"'");
                 while (res2.next()) {
                    an=res2.getString("accno");
                    st1=res2.getString("status");
                }
            }
            else
                System.out.println( "Please select Account type \n Please enter Account Number");

    if(u.equals(u1) && p.equals(p1) && an1.equals(an)&&st1.equals("ACTIVE")) {
  // JOptionPane.showMessageDialog(null,"Welcome " +name.toUpperCase()+ " !!!");
    Transaction t=new Transaction(u,an,at);
     t.setVisible(true);
     t.setLocationRelativeTo(null);
    }
```

Description

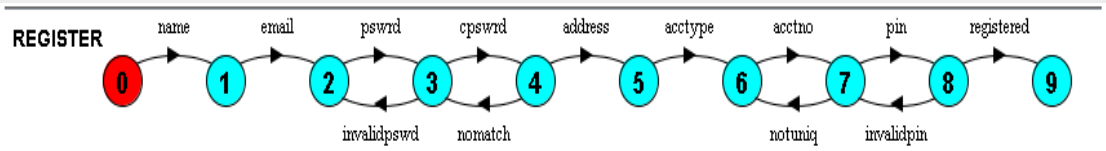
In login, the user needs to enter email, password, account type and account number. If these credentials matches, then the user is let into the system. After login, the user can perform various operations like withdrawal, deposit, balance enquiry or can logout.

Task 2 : REGISTRATION

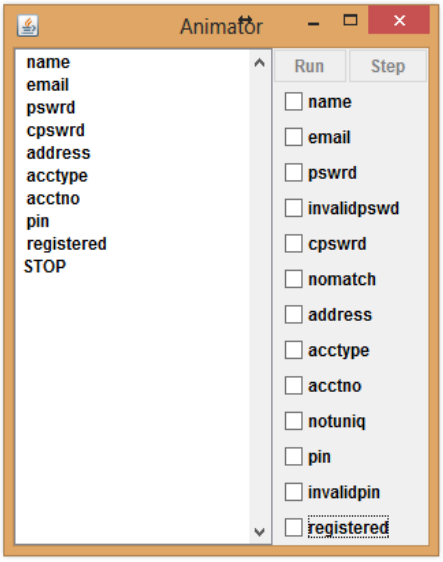
Finite State Process

```
REGISTER=(name->NAME),
NAME=(email->EMAIL),
EMAIL=(pswrd->PSWRD),
PSWRD=(invalidpswd->EMAIL|cpswrd->CPSWRD),
CPSWRD=(nomatch->PSWRD|address->acctype->ACCTYP),
ACCTYP=(acctno->ACCTNO),
ACCTNO=(notuniq->ACCTYP|pin->PIN),
PIN=(invalidpin->ACCTNO|registered->STOP).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
                if (k==0)
                   JOptionPane.showMessageDialog(null,"Sorry, account cannot be created !!!");
                else {

if (value8.equals("Savings")){
                st.executeQuery("insert into SBAL (accno,email,dep,bal,date1,abal) values('"+jTextField5.getText()+"','"-
                st.executeQuery("insert into SACCT (accno,email,status,pin) values('"+ jTextField5.getText()+"','"+ valu
                JOptionPane.showMessageDialog(null,"Account successfully created for "+value1.toUpperCase()+" \n Please 
                setVisible(false);
                 User u=new User(value2,b,time.toString());
                 u.setVisible(true);
                 u.setLocationRelativeTo(null);
}
else if (value8.equals("Current")){
                 st.executeQuery("insert into CBAL (accno,email,dep,bal,date1,abal) values('"+jTextField5.getText()+"','
                 st.executeQuery("insert into CACCT (accno,email,status,pin) values('"+jTextField5.getText()+"','"+ valu
                 JOptionPane.showMessageDialog(null,"Account successfully created for "+value1.toUpperCase()+" \n Please
                 setVisible(false);
                 User u=new User(value2,c,time.toString());
                 u.setVisible(true);
                 u.setLocationRelativeTo(null);
}

ResultSet rs = st.executeQuery("SELECT name,email,mobile,crdate from CUST_DETAILS");
                bw = new BufferedWriter(new FileWriter("AccountHolders.TXT"));
                StringBuilder  receipt = new StringBuilder();
                receipt.append("Name").append("       ").append("Email-ID").append("          ").append("Mobile").append
                receipt.append("\n");
```

Description

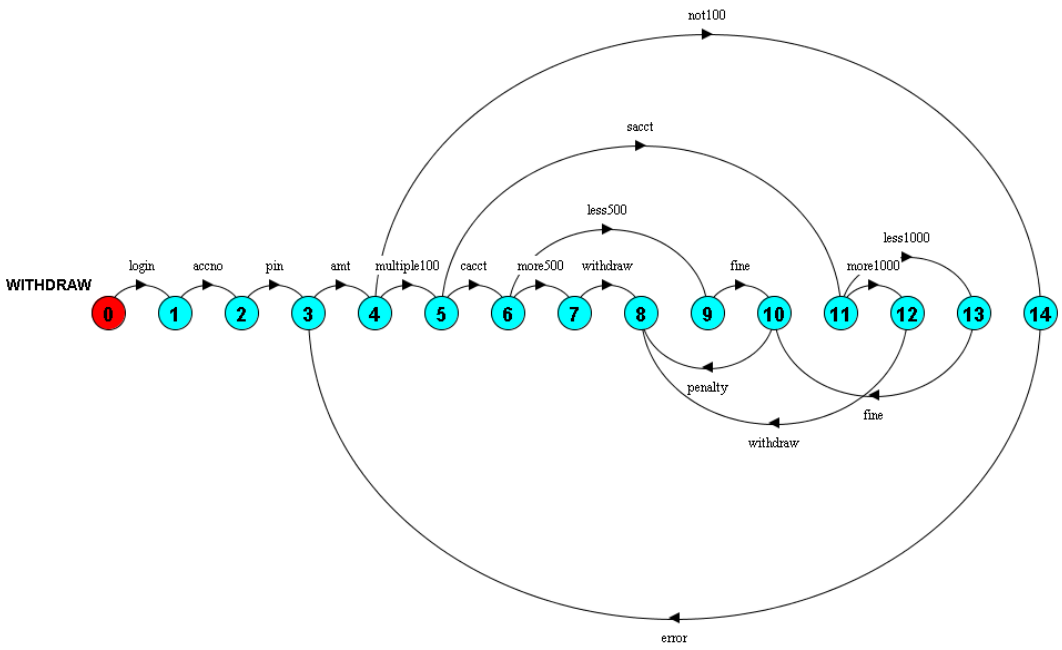
The user needs to enter the name, valid email id, password, address, dob, account type, account number and pin, If all details are correct , then the account is created,

Task 3 : WITHDRAWAL

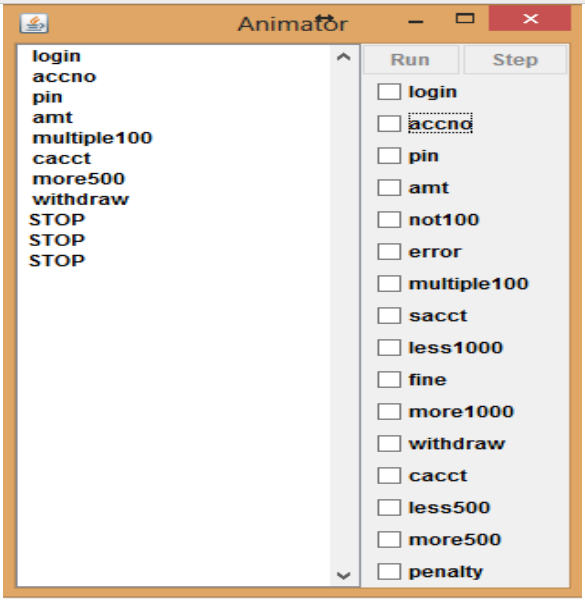
Finite State Process

```
WITHDRAW=(login->accno->pin->WD),
WD=(amt->(not100->error->WD
          |multiple100->(sacct->(less1000->fine->FINE|
                                 more1000->withdraw->STOP)
                         |cacct->(less500->fine->FINE
                                  |more500->withdraw->STOP)))),
FINE=(penalty->STOP).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```java
public class Withdrawl {
    int i=3;
    synchronized void withdraw(String name,int wd1,String val,String an,int at,String pin1) throws IOException
{
    int d,b=0,ab,p = 0;
    BufferedWriter bw = new BufferedWriter(new FileWriter("ReceiptWithdrawl.TXT"));
    StringBuilder  receipt = new StringBuilder();
     java.util.Date time = Calendar.getInstance().getTime();
    try{
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "CCSD","1234");
                Statement st = con.createStatement();
                if (at==1)
                        {
                ResultSet rs=st.executeQuery("SELECT * from SBAL where email='"+val+"' and accno='"+an+"'");
                while (rs.next()) {
                d=rs.getInt("wd");
                b=rs.getInt("abal");
                }
                ResultSet rs1=st.executeQuery("SELECT * from SACCT where email='"+val+"' and accno='"+an+"'");
                while (rs1.next()) {
                p=rs1.getInt("pin");
                }
                if(p==Integer.parseInt(pin1)){
                int abal=b-wd1;
                if (abal>=1000){
               st.executeQuery("insert into SBAL (accno,email,wd,bal,date1,abal) values('"+an+"','"+val+"','"+wd1+"','"+abal+"','"+t:
                st.executeUpdate("Update SBAL SET abal='"+abal+"' where email='"+val+"' and accno='"+an+"'");
                receipt.append(time).append("   ").append("Panipat").append("  ").append("Withdrawl").append("   ").append("Saving A
                else if (abal>=0)
                {
                abal=abal-100;
                System.out.println("Overdraft limit reached : Rs.100 Penalty");
                st.executeQuery("insert into SBAL (accno,email,wd,bal,date1,abal) values('"+an+"','"+val+"','"+wd1+"','"+abal+"'
                st.executeUpdate("Update SBAL SET abal='"+abal+"' where email='"+val+"' and accno='"+an+"'");
                receipt.append(time).append("   ").append("Panipat").append("  ").append("Withdrawl").append("   ").append("Savi
                }
                else
                    System.out.println("Insufficient funds");
                }
                else if(i>0){
                    i--;
                    System.out.println("Wrong pin entered !!! " +i+" Attempts left ");
                }
                else
                {
                    String status1="Blocked : Wrong PIN";
            st.executeUpdate("Update SACCT SET status='"+status1+"' where email='"+val+"'");
                    User u=new User();
                    u.setVisible(true);
                   u.setLocationRelativeTo(null);

                }
                        }
                else
                    {
                        ResultSet rs=st.executeQuery("SELECT * from CBAL where email='"+val+"' and accno='"+an+"'");
                while (rs.next()) {
```

Description

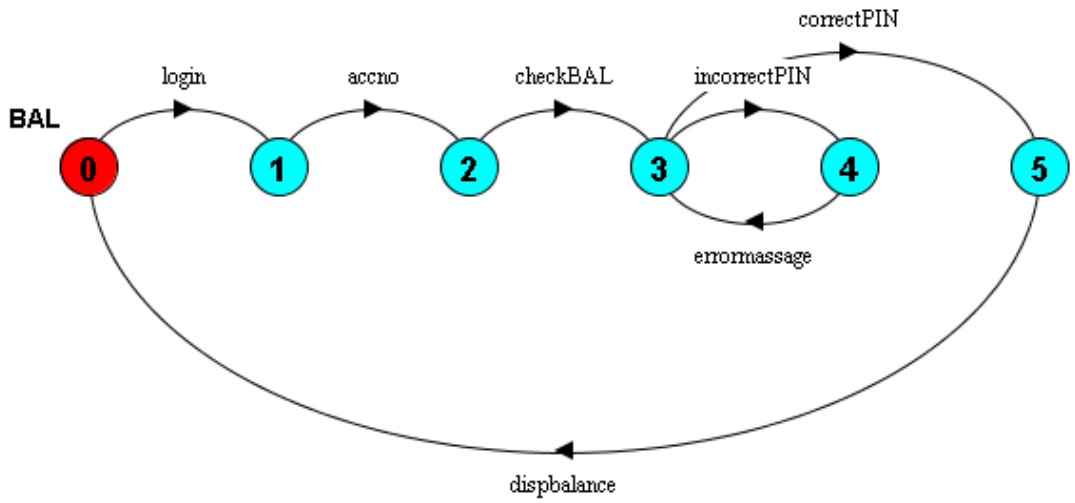
In this process, the user logins into the system, and can opt to withdraw from the account. The user needs to input the account number pin and amount. If the amount is not a multiple of 100 then an error message is shown. If the amount is a multiple of 100, then the account type is checked. If the account type is savings then the minimum balance of Rs.1000 needs to be maintained and in case of current account it is Rs.500. If the withdrawal leads to balance to go below the minimum balance , then overdraft message is shown along with a penalty of Rs.100.

Task 4 : BALANCE ENQUIRY

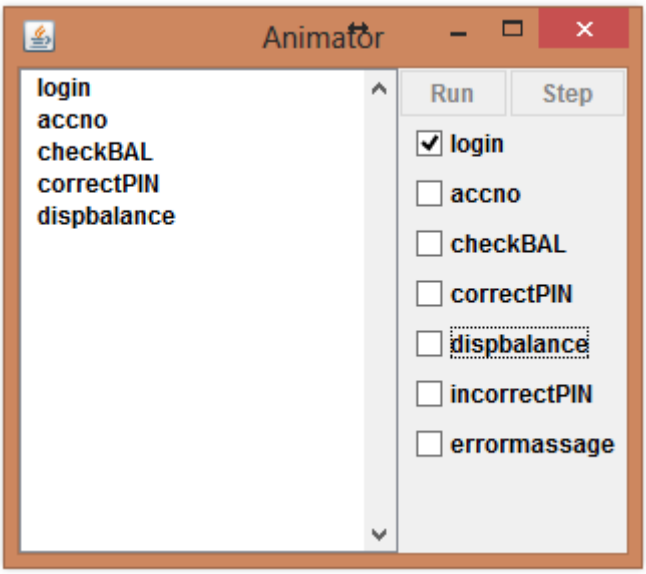
Finite State Process

```
BAL=(login->accno->checkBAL->VALIDATE),
VALIDATE=(correctPIN->dispbalance->BAL
          |incorrectPIN->errormassage->VALIDATE).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "CCSD","1234");
                Statement st = con.createStatement();
                if(at==1)
                {
                ResultSet rs=st.executeQuery("SELECT * from SBAL where email='"+val+"' and accno='"+an+"'");
                while (rs.next()) {
                 avb=rs.getInt("abal");
                }
               jLabel5.setText(Integer.toString(avb));
                   jLabel5.setVisible(true);
                   String log=val+" balance enquiry Rs. "+avb;
            java.util.Date time = Calendar.getInstance().getTime();
            st.executeQuery("insert into LOG values('"+log+"','"+ time.toString()+"')");
                }
                else
                {
                   ResultSet rs=st.executeQuery("SELECT * from CBAL where email='"+val+"' and accno='"+an+"'");
                while (rs.next()) {
                 avb=rs.getInt("abal");
                }
               jLabel5.setText(Integer.toString(avb));
                   jLabel5.setVisible(true);
                   String log=val+" balance enquiry Rs. "+avb;
            java.util.Date time = Calendar.getInstance().getTime();
            st.executeQuery("insert into LOG values('"+log+"','"+ time.toString()+"')");
                }
        } catch(ClassNotFoundException | SQLException e)
                        {
                            System.out.println(e);
                        }
```

Description

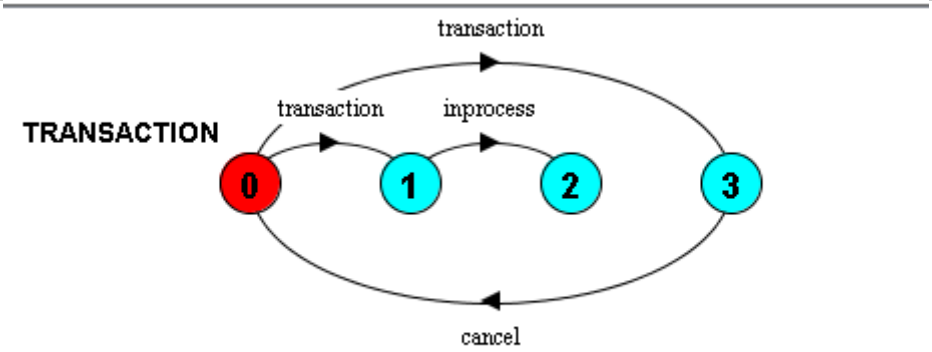
In this process, the user needs to login to the system. On successful login the user needs to click check balance button. Then a valid PIN needs to be inserted and the account number. Then the balance of the linked account is shown to the user.

Task 5 : CANCEL TRANSACTION

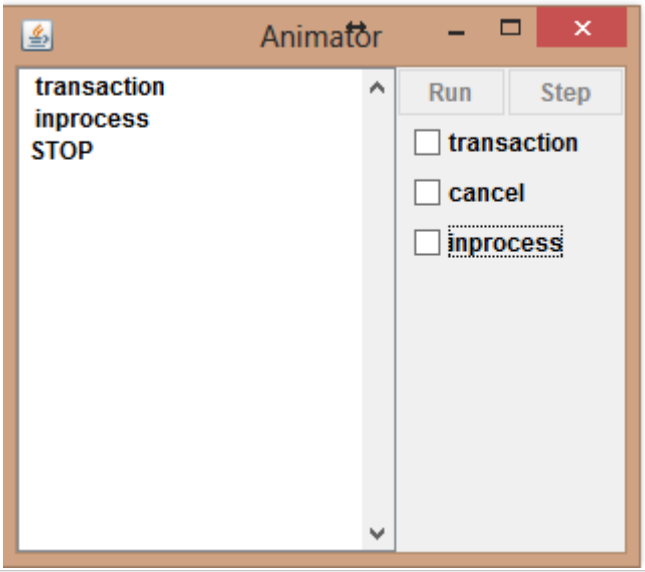
Finite State Process

```
TRANSACTION=(transaction->cancel->TRANSACTION
            |transaction->inprocess->STOP).
```

Labelled Transition State



Labelled Transition State Trace



Description

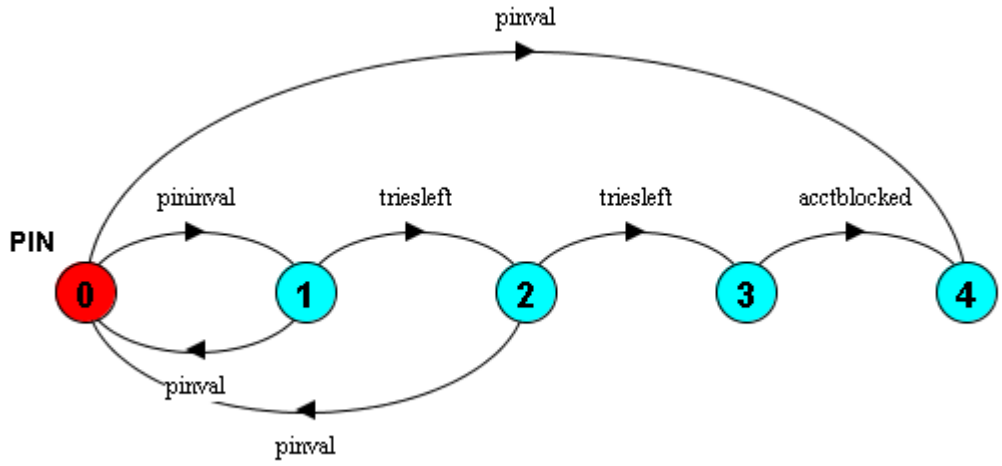
Transactions can be cancelled at any time of processing.

Task 6 : PIN

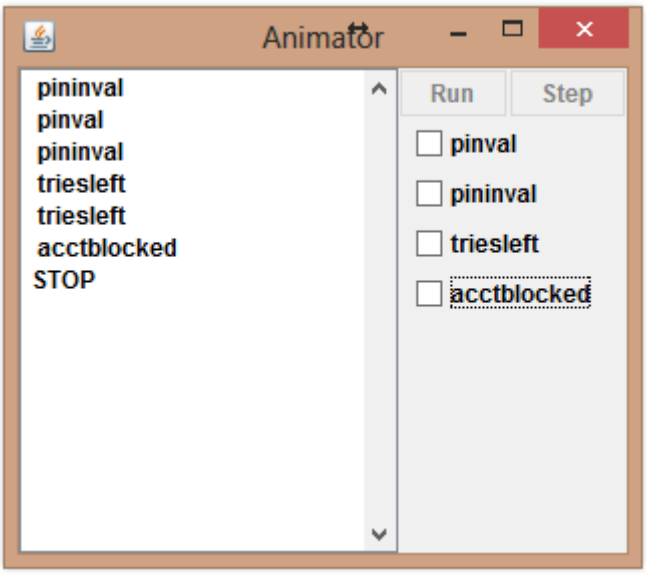
Finite State Process

```
range T=0..2
range R=0..1
PIN =PIN[0],
PIN[i:T]=(pinval->STOP|when(i<3)pininval->PININC[i]),
PININC[i:R]=(triesleft->PININC[i+1]|pinval->PIN),
PININC[2]=(acctblocked->STOP).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
                while (rs1.next()) {
                p=rs1.getInt("pin");
                }
                if(p==Integer.parseInt(pin1)){
                int abal=b-wd1;
        System.out.println("Insufficient funds");
    }
    else if(i>0){
        i--;
        System.out.println("Wrong pin entered !!! " +i+" Attempts left ");
    }
    else
    {
        String status1="Blocked : Wrong PIN";
executeUpdate("Update SACCT SET status='"+status1+"' where email='"+val+"'");
        User u=new User();
        u.setVisible(true);
       u.setLocationRelativeTo(null);

    }
            }
    else
        ,
```

Description

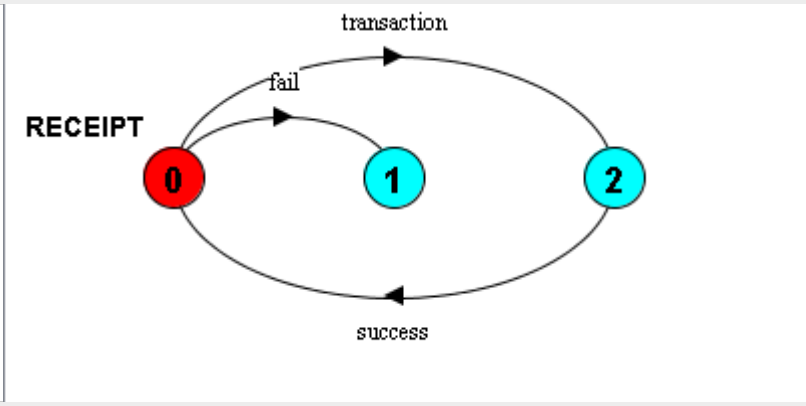
In this process the user has to enter a transaction pin in order to successfully complete the transaction. If the user enters wrong PIN, they can re-enter the PIN again, however only three chances for re-entering are available, after which the card is retained with the bank and the account is blocked.

Task 8 : RECEIPT

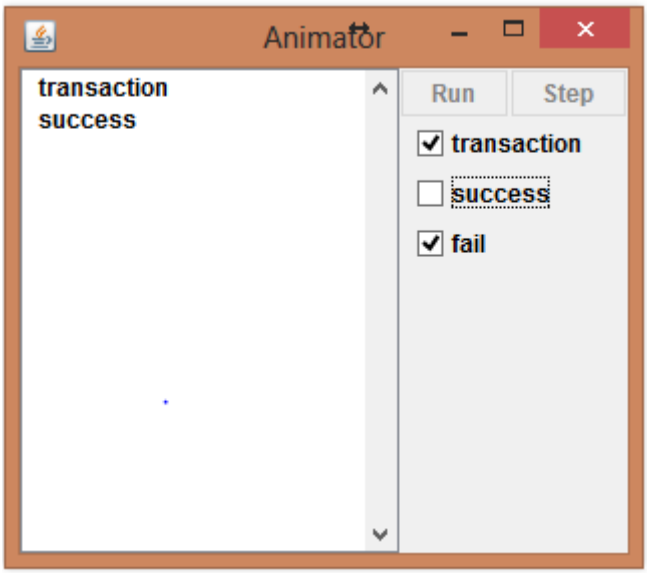
Finite State Process

```
RECEIPT=(transaction->(success->RECEIPT)
                      |fail->STOP).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
String log=val+" deposited Rs. "+d;
            String log2=val+" deposited Rs. "+d2;
    java.util.Date time = Calendar.getInstance().getTime();
    st.executeQuery("insert into LOG values('"+log+"','"+ time.toString()+"')");
    st.executeQuery("insert into LOG values('"+log2+"','"+ time.toString()+"')");
try{
    dt.join();
    dt1.join();
```

Description

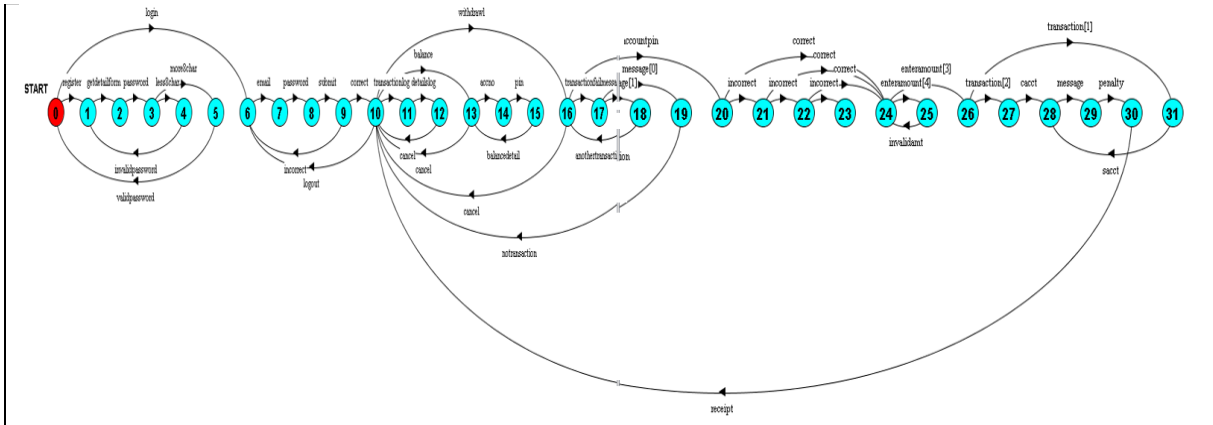
A receipt would be printed after every successful transaction.

Task 10 : ATM

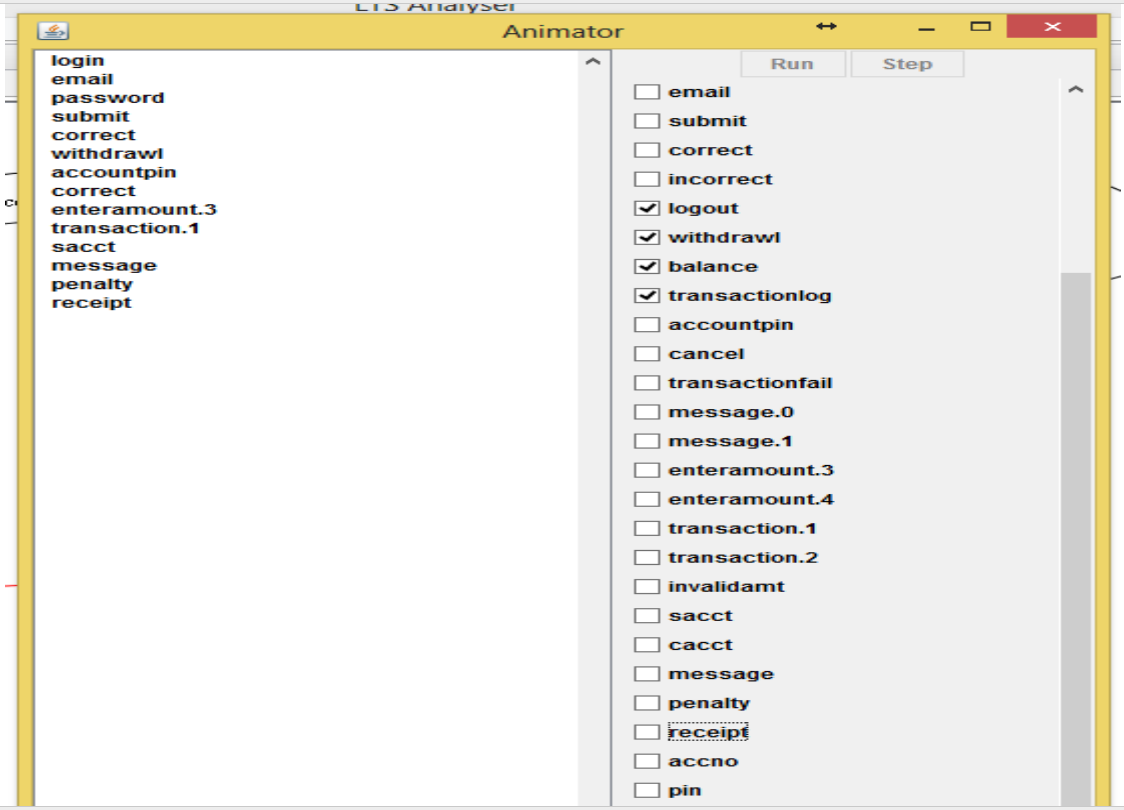
Finite State Process

```
const MBC=500
range YN=0..1
range N=1..3
range BAL=1..2000
START= (login->SIGNIN
            |registration->REGISTRATION),
REGISTRATION=(getdetailform->password->HOLD),
HOLD=(morethan8character->validpassword->START
    |lessthan8character->invalidpassword->REGISTRATION),
SIGNIN=(email_id->password->submit->VALIDATE),
VALIDATE=(correct->HOME
        |incorrect->SIGNIN),
HOME=(logout->SIGNIN
        |withdrawl->WITHDRAWL
        |balanceenquir->BALANCEENQUIRY
        |transactionlog->TRANSACTIONLOG),

WITHDRAWL=(accountpin->PIN
                |cancel->HOME
                |transactionfail->message[t:YN]->TM[t]),
PIN =NEW[1],
NEW[i:N]=(correct->AMOUNT
            |when (i<3)incorrect->NEW[i+1]
            |when(i==3)incorrect->STOP),
AMOUNT=(enteramount[b:I]->AMOUNTCHECK[b]),

AMOUNTCHECK[i:I]=(when(i==3)transaction[j:AT]->WITHDRAWALAMOUNT[j]
                    |when(i==4)messageinvalidammount->AMOUNT),
WITHDRAWALAMOUNT[k:AT]=(when(k==1)savingaccount->WITHDRAWLMESSAGE
                      | when(k==2)currentaccount->WITHDRAWLMESSAGE),
WITHDRAWLMESSAGE=(message->penalty->printbill->HOME),
BALANCEENQUIRY=(accnt_numb->pin->balancedetail->BALANCEENQUIRY
                |cancel->HOME),
TM[h:YN]=(when(h==1)anothertransaction->WITHDRAWL
        |when(h==0)notransaction->HOME),
TRANSACTIONLOG=(detailslog->cancel->HOME).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

NA

Description

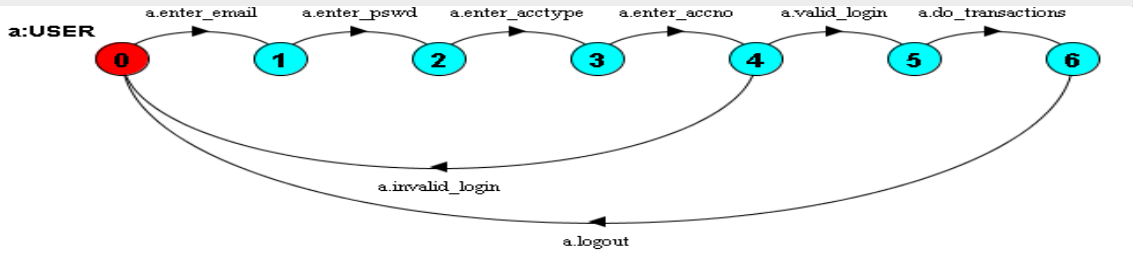
NA

Task 11 : Concurrency [multiple Logins]

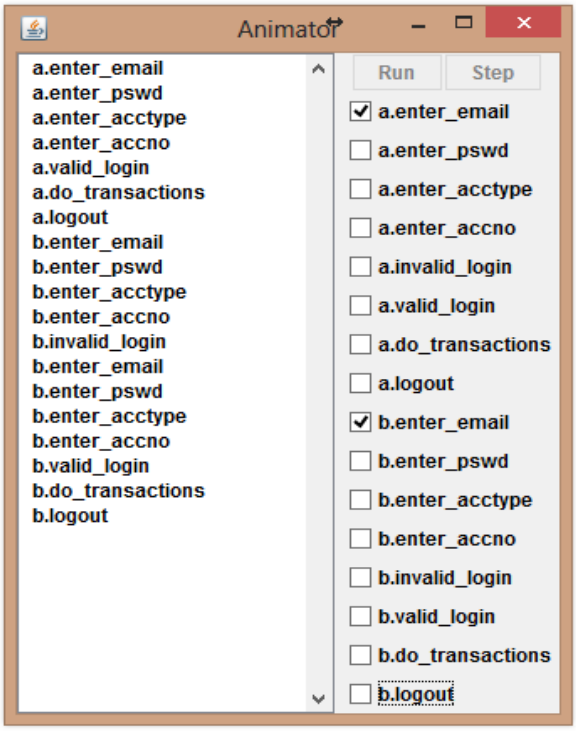
Finite State Process

```
USER=(enter_email->enter_pswd->enter_acctype->enter_accno->VALIDATELOGIN),
VALIDATELOGIN=(invalid_login->USER|valid_login->do_transactions->logout->USER)
||MULTI_USER=({a,b}:USER).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
        value1=jTextField1.getText();
        value2=jPasswordField1.getText();
        value3=jTextField2.getText();
        value4=jComboBox1.getSelectedIndex();
        value8=jComboBox2.getSelectedIndex();
        value5=jTextField3.getText();
        value6=jPasswordField2.getText();
        value7=jTextField4.getText();
    LoginThread u1=new LoginThread(1, "User1", value1,value2,value4,value3);
    LoginThread u2=new LoginThread(1, "User2", value5,value6,value8,value7);
    try{
        u1.join();
        u2.join();
    }
    catch(InterruptedException ex){
        System.out.println(ex);
    }
    setVisible(false);
```

Description

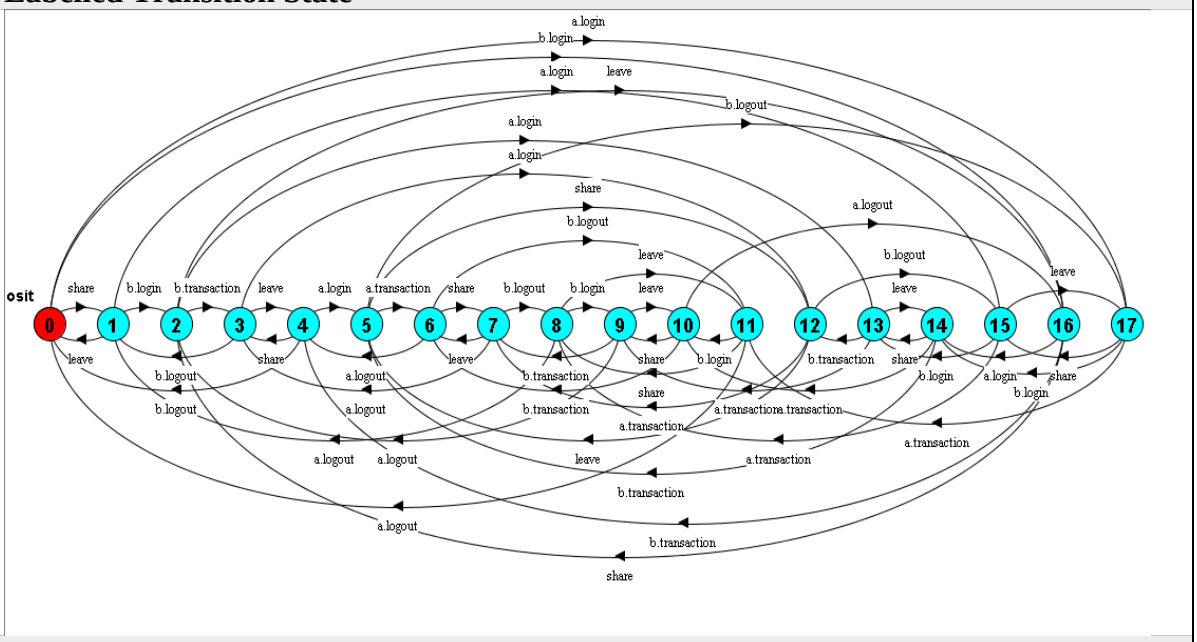
In the above program, we are displaying two panels on the same frame. First panel indicates User1 and second one User2. Two threads are started simultaneously and login happens concurrently for the two users.

Task 12 : Data Interference

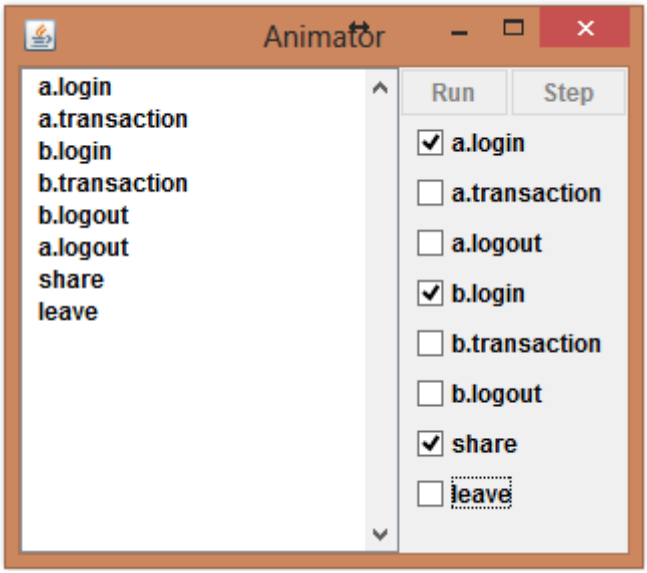
Finite State Process

```
DEPOSIT = (login->transaction->logout->DEPOSIT).
DB = (share->leave->DB).
||SYNC_Deposit = (a:DEPOSIT || b:DEPOSIT || DB).
```

Labelled Transition State



Labelled Transition State Trace



Code Snippet

```
        // TODO add your handling code here:

        int d=Integer.parseInt(jTextField2.getText());
        int d2=Integer.parseInt(jTextField3.getText());
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");

                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "CCSD","1234");
                Statement st = con.createStatement();
               if (d%100==0 && d2%100==0){
                   JOptionPane.showConfirmDialog(null, "Do you want to continue ?");
        DepositThread dt=new DepositThread(d1,"Deposit1",d, val, an,at,jPasswordField1.getText());
        DepositThread dt1=new DepositThread(d1,"Deposit2",d2, val, an,at,jPasswordField1.getText());
        JOptionPane.showMessageDialog(null, "Deposit succesful !!! ");
        String log=val+" deposited Rs. "+d;
                    String log2=val+" deposited Rs. "+d2;
            java.util.Date time = Calendar.getInstance().getTime();
            st.executeQuery("insert into LOG values('"+log+"','"+ time.toString()+"')");
            st.executeQuery("insert into LOG values('"+log2+"','"+ time.toString()+"')");
       try{
            dt.join();
            dt1.join();
        }
        catch(InterruptedException ex){
            System.out.println(ex+"error");
        }
               }else
        {
            JOptionPane.showMessageDialog(null,"Please enter in multiples of 100","Error",JOptionPane.ERROR_MESSAGE);
```

Description

Task 13 : Deadlock [balance and deposit]

Finite State Process

```
RW_LOCK = RW[0][False][0][False],
RW[readers:0..Nread][writing:Bool][waitingW:0..Nwrite][readersturn:Bool] =
    (when (!writing && (waitingW==0||readersturn))
       acquireRead -> RW[readers+1][writing][waitingW][readersturn]
    |releaseRead -> RW[readers-1][writing][waitingW][False]
    |when (readers==0 && !writing)
       acquireWrite ->RW[readers][True][waitingW-1][readersturn]
    |releaseWrite ->RW[readers][False][waitingW][True]
    |requestWrite ->RW[readers][writing][waitingW+1][readersturn]
    ).
```

Code Snippet

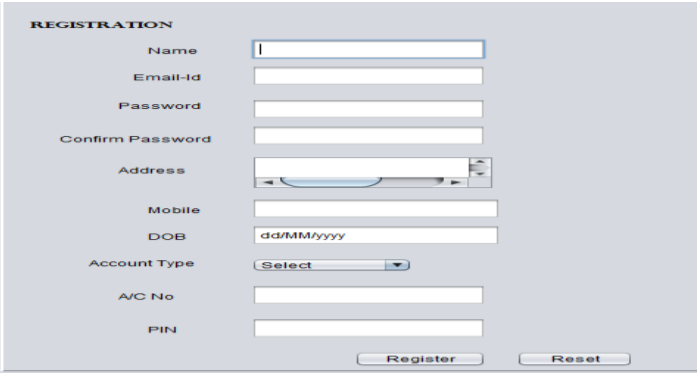
```
        String pin2=jPasswordField1.getText();
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");

                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "CCSD","1234");
                Statement st = con.createStatement();
        int i;
   rwmonitor rw[ ]=new rwmonitor[2];

   for(i=0;i<1;i++)
    rw[i]=new rwmonitor(i,"balance",val,an,at);
   for(i=1;i<2;i++)
    rw[i]=new rwmonitor(i,"deposit",d,val,an,at,pin2);
   for(i=1;i>=0;i--)
    rw[i].start();

                    String log=val+" deposited Rs. "+d;

            java.util.Date time = Calendar.getInstance().getTime();
            st.executeQuery("insert into LOG values('"+log+"','"+ time.toString()+"')");


        }catch(ClassNotFoundException | SQLException e){
            System.out.println(e);}
    }
```

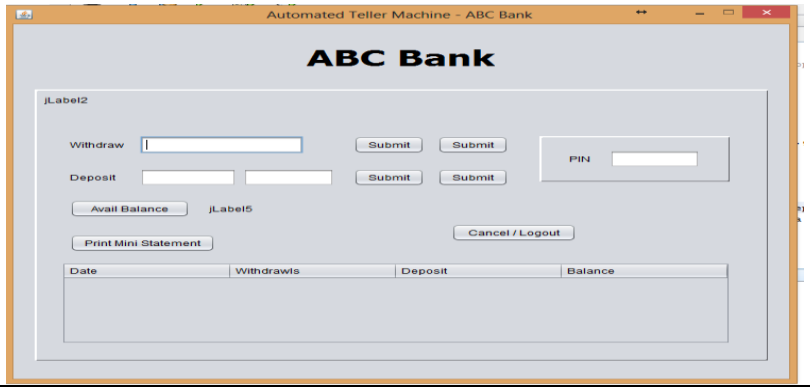
Description

T show that the system is deadlock free, deposit and balance were called simultaneously. The process deposit was executed first then the balance. Using the reader writer's implementation we were able to make the system deadlock free.

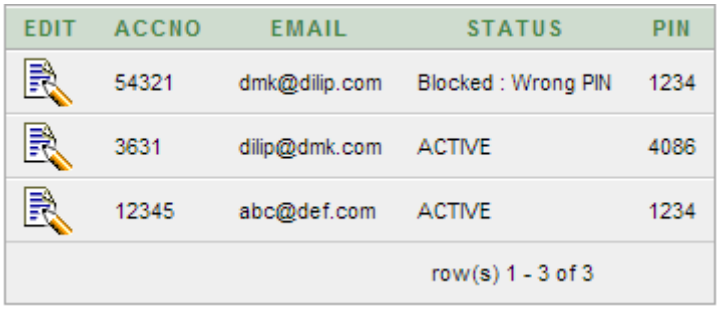
Evaluation and Testing

Project : **Concurrent System Design for an Automated banking system**		Date : 25/03/2014
Test Name : Login Module		**Test ID** : ATM01
Description : In this module user logs in using username, password, account type and account number		
Step	**Result Expected**	**Outputs**
The user is to input the email-id, password, account type and account number	If all the credentials are correct, the transaction form should be shown	Transaction form was shown
Result : Pass	**Corrective Actions:** None Required	
Conclusion: The user was able to login in to the account using the email id, password, account type and account number		
Screenshot :		

Project : **Concurrent System Design for an Automated banking system**		Date : 26/03/2014
Test Name : Registration Module		**Test ID** : ATM02
Description : In this module user can create a new login		
Step	**Result Expected**	**Outputs**
The user is to input the desired email-id, password, and address, mobile, dob, account type, pin and account number. Email and password length to be validated	If all the credentials are entered correct, the user would be registered. Email and password length should be validated	Message box confirming the registration was shown. Also, proper validations were there.
Result : Pass	**Corrective Actions:** None Required	
Conclusion: The user was able to create a new account with the desired credentials		
Screenshot : 		

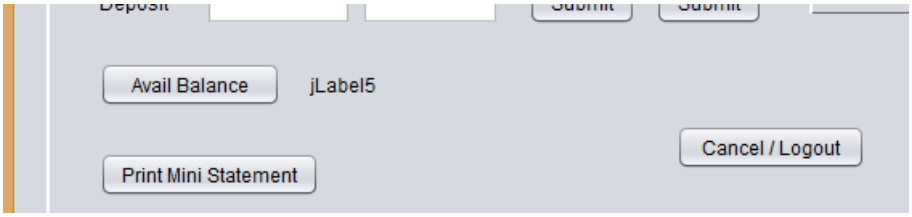
Project : **Concurrent System Design for an Automated banking system**		Date : 25/03/2014
Test Name : Withdrawal		**Test ID** : ATM03
Description : In this module after logging into system, the user should be able to withdraw money from the account		
Step	**Result Expected**	**Outputs**
The user is to input the withdrawal amount and the correct transaction pin	If the PIN is correct, money should be withdrawn	Money was deducted from the account
Result : Pass	**Corrective Actions:** None Required	
Conclusion: The user was able to withdraw money from the account after inputting the correct PIN		
Screenshot : 		

Project : **Concurrent System Design for an Automated banking system**		Date : 26/03/2014
Test Name : Correct PIN		**Test ID** : ATM04
Description: To perform a transaction, a PIN is required and if this PIN is entered wrong for more than 3 times, the account will be blocked.		
Step	**Result Expected**	**Outputs**
The user is to input the correct transaction pin before any transaction	After entering PIN 3 times wrong should block the account	The account status was changed from ACTIVE to BLOCKED
Result : Pass	**Corrective Actions:** None Required	
Conclusion: The user account was blocked after the wrong PIN was entered thrice.		
Screenshot :		



EDIT	ACCNO	EMAIL	STATUS	PIN
	54321	dmk@dilip.com	Blocked : Wrong PIN	1234
	3631	dilip@dmk.com	ACTIVE	4086
	12345	abc@def.com	ACTIVE	1234
			row(s) 1 - 3 of 3	

Project : **Concurrent System Design for an Automated banking system**		Date : 29/03/2014
Test Name : Balance Enquiry		**Test ID** : ATM05
Description: To perform a balance enquiry of a linked account.		
Step	**Result Expected**	**Outputs**
The user is to click the balance button	Available balance should be shown	Available balance was shown
Result : Pass	**Corrective Actions:** None Required	
Conclusion: The balance shown was correct.		
Screenshot :		



Depth of Discussion

Sections that met the requirements

S No.	**Basic Requirements**	**Achieved**	**Difficulty**
1.	The customer should be able to login with the registered email-id and password [more than 8 characters]. Also, the users account type and account number would also be required during login.	✓	Low
2	For new users, they should be able to create a login account with details such as name, email-id, password, address, dob, mobile, account type, account number and pin.	✓	Medium
3.	On successful login, customer can withdraw money in the multiples of 100. Minimum balance of 1000 and 500 to be maintained in the saving and current accounts respectively. On overdraft limit, Rs.100 to be deducted per transaction.	✓	Medium
4.	Customers can check for the balance of their accounts	✓	Low
5.	A PIN would be asked for before every transaction. On entry of wrong PIN for 3 times will block the account.	✓	Medium
6.	Appropriate message to be shown if transaction fails.	✓	Low
7.	Receipt to be printed after each successful transaction showing date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account.	✓	Medium
8.	Transactions can be aborted at any time by the user	✓	Medium
9.	An internal log file to be maintained.	✓	High

S No.	**Additional Requirements**	**Achieved**	**Difficulty**
1.	To show concurrency : multiple access to clients	✓	Medium
2.	To show how to handle issue of data corruption	✓	Medium
3.	To show how to handle issue of deadlock	✓	High

Easiest Code/Section: Balance Enquiry

```
        // TODO add your handling code here:
        int avb = 0;
        try{
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "CCSD","1234");
                Statement st = con.createStatement();
                if(at==1)
                {
                ResultSet rs=st.executeQuery("SELECT * from SBAL where email='"+val+"' and accno='"+an+"'");
                while (rs.next()) {
                 avb=rs.getInt("abal");
                }
               jLabel5.setText(Integer.toString(avb));
                   jLabel5.setVisible(true);
                }
                else
                {
                   ResultSet rs=st.executeQuery("SELECT * from CBAL where email='"+val+"' and accno='"+an+"'");
                while (rs.next()) {
                 avb=rs.getInt("abal");
                }
               jLabel5.setText(Integer.toString(avb));
                   jLabel5.setVisible(true);
                }
         } catch(ClassNotFoundException | SQLException e)
                        {
                            System.out.println(e);
                        }
    }
```

Hardest Code/Section: Login through Concurrency

```
public class LoginThread extends Thread {
    Login l;
    String un,pswd,an;
    int at;
    public LoginThread(Login l,String a,String un, String pswd,int at, String an)
    {
        super(a);
        this.l=l;
        this.an=an;
        this.at=at;
        this.un=un;
        this.pswd=pswd;
        start();
    }
    @Override
    public void run()
    {
        l.LoginUser(Thread.currentThread().getName(), un, pswd, at, an);

    }
}
```

```java
public class Login {
   synchronized void LoginUser(String a,String u, String p, int at, String an1)
{
    String name="",u1="",p1="",an="",st1="";
    ResultSet res1 = null,res,res2 = null;
    int i=3;
    System.out.print(a);
        try{
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "CCSD","1234");
                Statement st = con.createStatement();
                res = st.executeQuery("SELECT * from CUST_DETAILS where email='"+u+"' and password='"+p+"'");

                while (res.next()) {
                u1 = res.getString("email");
                p1 = res.getString("password");
                name = res.getString("name");
        }
                if (at==1)
                {
                    res1 = st.executeQuery("SELECT * from SACCT where email='"+u+"' and accno='"+an1+"'");
                    while (res1.next()) {
                        an=res1.getString("accno");
                        st1=res1.getString("status");
                    }
                }
                else if (at==2){
                    res2 = st.executeQuery("SELECT * from CACCT where email='"+u+"' and accno='"+an1+"'");
                     while (res2.next()) {
                        an=res2.getString("accno");
```

Sections that didn't meet the requirements

Though the developer to his full capacity has tried to complete this system and its requirements. Though proper validations have been given, however the corresponding messages are not being shown in message boxes or labels and are only being displayed on the console.

Critical Appraisal

Database concurrency control permits users to access a database in a multi-programmed fashion while preserving the delusion that each user is executing alone on a dedicated system. The main difficulty in achieving this goal is to prevent database updates performed by one user from interfering with database retrievals and updates performed by another. The system has to concurrent, free from data corruption and deadlock.

To understand state changes, Finite State Processes were drawn which are simple algebraic notations to model process models.

Corresponding to the FSP descriptions, Labelled Transition System designs are created with all the possible traces.

Structure diagram for the whole system and for some shared resource process have been made which gives a clear idea of resources.

Class Diagram, depicting all the classes along with its operations and attributes and links with other classes shows a rough idea of how system is been made.

Code implementation includes all necessary explanation of the codes required by the system, concurrency, multithreading, safety and liveliness checks for various processes states that the system is appropriate though the requirement which cannot be made are thoroughly explained.

Test cases used for testing all validation fields in the system and also check whether or not a process is working.

Limitations of developed system

The following are the limitations of the system developed:

- To show concurrency, the console shows two user windows to work on.
- No proper verification of the email and mobile number
- No security measures have been given

Future Enhancements

The following feature can be included:

- Email-id verification through verification code being sent to the registered mail
- Registering mobile phone number by using the verification code
- Sending transaction details to registered email and mobile
- This system can be made web-based to utilize the full functionality of concurrency.
- Virtual keyboard can be given to aid security

Conclusion

The developer to his best capacity has tried to implement concurrent programming design methods to this project.

The various phases of development of this project were:

- Identifying and understanding the requirements of the system
- Mapping those requirements into a model. For this purpose, Class diagrams, Structure Diagrams, Finite State process diagrams and Labelled Transition Systems diagram were made.
- The created models were then used for creating the corresponding Java codes
- The developer had to make himself familiar with the common concepts of Java like inheritance, composition, SWINGS/AWT, multi-threading, interface, etc.
- Database concepts were also revived for this project

References

- Jeff M & Jeff K (2006) , Concurrency – State Models & Java Programming , England, 1st Edition – John Wiley & sons, Chapter-1 page 1-16, 2 page 1-20

- Jean B (1998). , Concurrent Systems – Operating Systems, Database and Distributed Systems: An Integrated Approach, Harlow 2nd Edition, Addison-Wesley page 167