Corporate Banking System Java project

**SOURCE CODE**

import javax.swing.\*;

import javax.swing.event.\*; // for ChangeListener of JTabbedPane.

import java.awt.event.\*;

import java.awt.\*;

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.text.\*;

import java.sql.\*;

public class Server extends JFrame implements ActionListener, ChangeListener, Runnable {

// a list of the clients that are logged currently:

private static Vector clients = new Vector();

// number of ports to receive messages from clients:

private static int client\_port;

//object to AccessServer class

AccessServer lastClient;

AccessDbase aDbase;

// keeps server's address:

private static InetAddress localHost = null;

// GUI's

AdminEntryLevel adminEntry;

AdminCreateAcc adminCreate;

AdminDeleteAcc adminDelete;

AdminEditAcc adminEdit;

AdminMainMenu adminMain;

AdminUpdateAcc adminUpdate;

AdminViewAccount adminViewAcct;

AdminViewReport adminView;

// AdminMainMenu adminMain;

// AdminCreateAcc adminCreate;

//server sockets for receiving connection from clients:

ServerSocket socketForClient = null;

// visual interface:

String Dattee;

// char dt = new char[50] ;

// button for termination of a client:

JButton btnTerminate = new JButton("Terminate ");

//button for administrator previlages

JButton btnAdministrator = new JButton("Administrator ");

JLabel lblRunning; // label to show how many clients are logged.

JLabel timeRunning; // label to show the updated time.

/\*

JLabel lblWaiting = new JLabel("lblWaiting"); // expected connection status.

JLabel lblWaitAddress = new JLabel("lblWaitAddress"); // waiting player host.

JLabel lblWaitPort = new JLabel("lblWaitPort"); // waiting player port.

JButton btnCancelWait = new JButton("Cancel Waiting"); // button to cancel waiting.

\*/

long acctno,balance;

ImageIcon icon = new ImageIcon("pic.gif"); // picture for tabbedPane.

JTextArea txtInfo = new JTextArea(); // text area for information.

JTabbedPane tabbedPane = new JTabbedPane(); // tabs to select clients.

JPanel pInnerTab = new JPanel (new BorderLayout()); // panel inside the tabbedPane.

JLabel lblDateRunning ;

// thread that cares about accepting new clients:

Thread thClientAccept = null;

// thread that cares about updating the client info:

Thread thUpdateClientInfo = null;

// thread that cares about updating the date:

Thread clockThread = null;

Thread dateThread = null;

String dtString = new String("");;

String currentTime = new String("");

/\*\*

\* Default Constructor.

\*/

public Server() {

super ("The Server"); // set window caption.

// set server sockets:

try {

socketForClient = new ServerSocket(client\_port);

} catch (IOException ioe) {

System.err.println("Cannot open server socket: " + ioe);

System.exit(0);

}

adminEntry= new AdminEntryLevel(this);

adminCreate = new AdminCreateAcc(this);

adminDelete = new AdminDeleteAcc(this);

adminEdit = new AdminEditAcc(this);

adminMain = new AdminMainMenu(this);

adminUpdate = new AdminUpdateAcc(this);

adminViewAcct = new AdminViewAccount(this);

setDisplay();

// adminView = new AdminViewReport(this);

// look & feel setup:

try {

UIManager.setLookAndFeel(

UIManager.getSystemLookAndFeelClassName() );

} catch (Exception e) {

System.err.println("Couldn't use the system "

+ "look and feel: " + e);

}

// update look & feel for those components created in

// declaration (if required):

btnTerminate.updateUI();

btnAdministrator.updateUI();

txtInfo.updateUI();

tabbedPane.updateUI();

pInnerTab.updateUI();

/\* The default value is: HIDE\_ON\_CLOSE,

we need to ask user "Are you sure?" when there are clients logged.

\*/

setDefaultCloseOperation(WindowConstants.DO\_NOTHING\_ON\_CLOSE);

// processing window events:

WindowListener L= new WindowAdapter () {

public void windowClosing(WindowEvent e) {

closeApplication();

}

};

addWindowListener(L);

// prepare the layout:

JPanel pMain = new JPanel (new BorderLayout()); // main layout.

JPanel pUpper = new JPanel (new GridLayout(1,2)); // upper layout.

JPanel pLLBL = new JPanel (new GridLayout(5,1)); // for labels at left.

JPanel pRLBL = new JPanel (new GridLayout(1,1)); // for labels at right.

lblRunning = new JLabel(" Currently logged : 0 client(s).");

lblDateRunning = new JLabel("");

pLLBL.add(new JLabel(" "));

pLLBL.add(new JLabel(" "));

pLLBL.add(lblDateRunning);

pLLBL.add(new JLabel(" Server is running on host: " + localHost));

pLLBL.add(lblRunning);

timeRunning = new JLabel(" ");

// add labels for displaying the time:

pRLBL.add(timeRunning);

// upper part conatins server state labels on one side,

// and current connection state on the other:

pUpper.add(pLLBL);

pUpper.add(pRLBL);

JPanel pBtns = new JPanel (new FlowLayout()); // for "Terminate" button.

btnTerminate.addActionListener(this);

pBtns.add(btnTerminate);

// shown inside the tabbedPane:

pInnerTab.add(txtInfo, BorderLayout.CENTER);

pInnerTab.add(pBtns, BorderLayout.SOUTH);

pMain.add(pUpper, BorderLayout.NORTH);

tabbedPane.addChangeListener(this);

pMain.add(tabbedPane, BorderLayout.CENTER);

JPanel adminBtns = new JPanel (new BorderLayout());

JPanel adminDisplay = new JPanel (new FlowLayout());

adminDisplay.add(new JLabel("Login as : "));

btnAdministrator.addActionListener(this);

adminDisplay.add(btnAdministrator);

adminBtns.add(adminDisplay,BorderLayout.SOUTH);

pMain.add(adminBtns,BorderLayout.SOUTH);

// set content pane:

setContentPane (pMain);

// it doesn't work with our JTabbedPane !!! ---> pack();

setSize(600, 500);

setBounds(100,20,600,500);

// show the window:

setVisible(true);

setResizable(false);

aDbase= new AccessDbase();

aDbase.connectionDb();

pause(2000);

adminView = new AdminViewReport(this);

// start threads:

thClientAccept = new Thread(this);

thUpdateClientInfo = new Thread(this);

clockThread = new Thread(this);

dateThread = new Thread(this);

thClientAccept.start(); // start accepting new clients.

thUpdateClientInfo.start(); // start to care about updating the info.

clockThread.start();

dateThread.start(); // start to care about the time.

}

/\*\*

\* The run method is used by all threads.

\*/

public void run( ) {

Thread thisThread = Thread.currentThread();

// thread that cares about accepting new clients:

while (thClientAccept == thisThread) {

try {

// wait for client to connect, and then validate connection:

// use temporary pointer for new connection:

Socket clientConnection = socketForClient.accept();

System.out.println("Client accepted!");

// register a new client:

addNewTab();

lastClient = new AccessServer(this);

clients.addElement(lastClient);

lblRunning.setText("Currently logged :" + clients.size() + " client(s)" );

// connect client to AccessServer:

lastClient.connectClient(clientConnection);

}catch (Exception e) {

if (thClientAccept != null) // not shutting down yet?

System.err.println("Accept client -> " + e);

}

}

// thread that cares about updating the client info:

// automatic update of info for selected client:

while (thUpdateClientInfo == thisThread) {

if (clients.size() > 0) // no clients?

showClientData();

thUpdateClientInfo.yield(); // this thread isn't so important, think of others.

pause(1200); // update every 1.2 of a second.

}

while (clockThread == thisThread) {

iterateTime();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.err.println("clock thread -> " + e);

// the VM doesn't want us to sleep anymore,

// so get back to work

}

}

while (dateThread == thisThread) {

iterateDate();

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

System.err.println("clock thread -> " + e);

// the VM doesn't want us to sleep anymore,

// so get back to work

}

}

} // end of run().

private void iterateTime() {

// get the time and convert it to a date

Calendar cal = Calendar.getInstance();

java.util.Date date = cal.getTime();

//System.out.println("Display time : " + date.toString());

//char dts[] = date.toString().toCharArray() ;

// Dattee ;

//System.out.println("Display time : " + dts);

//Dattee.valueOf(dts,12,9);

//System.out.println("Display time : " + Dattee);

// format it and display it

DateFormat dateFormatter = DateFormat.getTimeInstance();

SimpleDateFormat dateFormatterH = new SimpleDateFormat("HH:mm:ss");

currentTime = dateFormatterH.format(date);

timeRunning.setText(" Time : " + dateFormatter.format(date));

}

private void iterateDate() {

// get the time and convert it to a date

// format it and display it

Calendar cal = Calendar.getInstance();

java.util.Date date = cal.getTime();

SimpleDateFormat dt = new SimpleDateFormat ("dd MMM yyyy '(' EE ')' ");

//java.util.Date currTime = new java.util.Date();

dtString = dt.format(date);

lblDateRunning.setText(" Date : " + dtString);

}

/\*\*

\* Makes current thread to pause.

\*

\* @param time miliseconds to sleep.

\*/

public void pause(int time) {

try {

Thread.sleep(time);

}

catch (InterruptedException e) {

System.err.println(e.getMessage());

}

}

/\*\*

\* Processes clicks on buttons.

\*

\* @param e the ActionEvent object.

\*/

public void actionPerformed(ActionEvent e){

// JButton tempBtn = (JButton)e.getSource();

JButton src = (JButton)e.getSource();

if (src==btnTerminate) { // terminate the client

try {

AccessServer t; // temporary pointer.

t = (AccessServer) clients.get(tabbedPane.getSelectedIndex());

t.sendToClient("TERMINATED");

removeClient(t);

}

catch (ArrayIndexOutOfBoundsException ae) {

txtInfo.setText( "No client with index: " + tabbedPane.getSelectedIndex());

}

}

else if (src == btnAdministrator){

btnAdministrator.setEnabled(false);

adminEntry.setClear();

adminEntry.setVisible(true);

}

else if(src == adminEntry.logIn)

{

String s=new String(adminEntry.pField.getPassword());

if((adminEntry.txtID.getText().equalsIgnoreCase("Administrator"))&&

(s.equals("Admin") ))

{

System.out.println("AdminEntryLevel: Logged In");

adminEntry.setVisible(false);

adminMain.setClear();

adminMain.setVisible(true);

}

else {

System.out.println("AdminEntryLevel:Log In Failed");

JOptionPane.showMessageDialog(adminEntry,

"Log In Failed",

"Admin Entry Level",

JOptionPane.ERROR\_MESSAGE

);

}

}

else if(src==adminEntry.cancelLogIn)

{

adminEntry.setVisible(false);

btnAdministrator.setEnabled(true);

}

else if(src==adminMain.btnCreate)

{

adminMain.setVisible(false);

adminCreate.setClear();

adminCreate.setVisible(true);

System.out.println("Admin Create Acc");

}

else if(src==adminMain.btnDelete)

{

adminMain.setVisible(false);

adminDelete.setClear();

adminDelete.setVisible(true);

System.out.println("Admin Delete Acc");

}

else if(src==adminMain.btnEdit )

{

adminMain.setVisible(false);

adminEdit.setClear();

adminEdit.setVisible(true);

System.out.println("Admin Edit Acc");

}

else if(src==adminMain.btnViewAcct)

{

adminMain.setVisible(false);

adminViewAcct.setClear();

adminViewAcct.setVisible(true);

}

else if(src == adminMain.btnViewReport)

{

adminMain.setVisible(false);

adminView.setActionCmd();

//adminView.setMaximizedBounds(new Rectangle(0,0,Integer.MAX\_VALUE,Integer.MAX\_VALUE));

adminView.setVisible(true);

}

else if(src==adminMain.btnLogout)

{

adminEntry.setClear();

adminMain.setVisible(false);

adminEntry.setVisible(true);

}

else if(src==adminCreate.btnSubmit)

{

try { if((adminCreate.fields[1].getText().trim().equals(""))||(adminCreate.fields[2].getText().trim().equals(""))||(adminCreate.fields[3].getText().trim().equals(""))||(adminCreate.fields[4].getText().trim().equals(""))||(adminCreate.fields[5].getText().trim().equals(""))||(adminCreate.fields[6].getText().trim().equals(""))||(adminCreate.fields[7].getText().trim().equals("")))

{

JOptionPane.showMessageDialog(adminCreate,

"Fields are incomplete.\n Status :Error" ,

"Admin Create Account",

JOptionPane.ERROR\_MESSAGE

);

}

else

{

aDbase.stmt.executeUpdate("INSERT INTO ClientInfo" + " VALUES( " + adminCreate.fields[0].getText() + ",'" + adminCreate.fields[1].getText() + "','" + adminCreate.fields[2].getText() + "','" + adminCreate.fields[3].getText() + "','" + adminCreate.fields[4].getText() + "','" + adminCreate.fields[5].getText() + "','" + adminCreate.fields[6].getText() + "'," + "'" + adminCreate.fields[7].getText() + "',True," + adminCreate.fields[8].getText() + " )" );

aDbase.stmt.executeUpdate("INSERT INTO ClientAccStatus" + " VALUES( " + adminCreate.fields[0].getText() + ",'" + adminCreate.fields[1].getText() + "',500, No)");

System.out.println("Dbase Created");

JOptionPane.showMessageDialog(adminCreate,

"Account No : " + adminCreate.fields[0].getText() +"\nName : " + adminCreate.fields[1].getText() + "\nPassword : " + adminCreate.fields[2].getText() + "\nPin : " + adminCreate.fields[8].getText() + "\nStatus : Account Created" ,

"Admin Create Account",

JOptionPane.INFORMATION\_MESSAGE

);

adminCreate.setVisible(false);

adminMain.setVisible(true);

}

}

catch(SQLException sqle)

{

System.out.println("Error:"+sqle);

}

}

else if(src==adminCreate.btnCancel)

{

adminCreate.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminDelete.btnDelete)

{

try{

String s = adminDelete.txtAcctNo.getText();

String updateQuery = "UPDATE ClientInfo SET Validity = False WHERE AccountNo = " + s ;

aDbase.stmt.executeUpdate(updateQuery);

aDbase.uprs = aDbase.stmt.executeQuery("SELECT Name FROM ClientInfo WHERE AccountNo = " + s );

aDbase.uprs.next();

String name = aDbase.uprs.getString(1);

JOptionPane.showMessageDialog(adminDelete,

"Account No : " + s +"\nName : " + name + " \nStatus : Deleted" ,

"Admin Delete Account",

JOptionPane.INFORMATION\_MESSAGE

);

System.out.println("Dbase Deleted");

adminDelete.setVisible(false);

adminMain.setVisible(true);

}

catch(SQLException sqle)

{

JOptionPane.showMessageDialog(adminDelete,

"Invalid Account Number",

"Admin Delete Account",

JOptionPane.ERROR\_MESSAGE

);

System.out.println("Error:"+sqle);

}

}

else if(src==adminDelete.btnCancel)

{

adminDelete.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminEdit.btnEdit)

{

try {

String s = adminEdit.txtAcctNo.getText();

aDbase.uprs = aDbase.stmt.executeQuery("SELECT \* FROM ClientInfo WHERE AccountNo = " + s );

aDbase.uprs.next();

adminUpdate.fields[0].setText(s);

adminUpdate.fields[0].setEditable(false);

adminUpdate.fields[1].setText(aDbase.uprs.getString(2));

adminUpdate.fields[2].setText(aDbase.uprs.getString(3));

adminUpdate.fields[3].setText(aDbase.uprs.getString(4));

adminUpdate.fields[4].setText(aDbase.uprs.getString(5));

adminUpdate.fields[5].setText(aDbase.uprs.getString(6));

adminUpdate.fields[6].setText(aDbase.uprs.getString(7));

adminUpdate.fields[7].setText(aDbase.uprs.getString(8));

adminEdit.setVisible(false);

adminUpdate.setVisible(true);

}

catch(SQLException sqle)

{

JOptionPane.showMessageDialog(adminEdit,

"Invalid Account Number",

"Admin Edit Account",

JOptionPane.ERROR\_MESSAGE

);

}

}

else if(src==adminEdit.btnCancel)

{

adminEdit.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminUpdate.btnUpdate)

{

try {

String s= adminEdit.txtAcctNo.getText();

String updateQuery = "UPDATE ClientInfo SET Name = '" + adminUpdate.fields[1].getText() + "', Password = '" + adminUpdate.fields[2].getText() + "', AddressLine1 = '" + adminUpdate.fields[3].getText() + "', AddressLine2 = '" + adminUpdate.fields[4].getText() + "',City = '" + adminUpdate.fields[5].getText() + "', State = '" + adminUpdate.fields[6].getText() + "', Phone = '" + adminUpdate.fields[7].getText() + "', Validity = True WHERE AccountNo = " + s ;

aDbase.stmt.executeUpdate(updateQuery);

aDbase.uprs = aDbase.stmt.executeQuery("SELECT Name,Password FROM ClientInfo WHERE AccountNo = " + s );

aDbase.uprs.next();

String name = aDbase.uprs.getString(1);

String pword = aDbase.uprs.getString(2);

JOptionPane.showMessageDialog(adminUpdate,

"Account No : " + s +"\nName : " + name + "\nPassword : " + pword + "\nStatus : Updated" ,

"Admin Update Account",

JOptionPane.INFORMATION\_MESSAGE

);

adminUpdate.setVisible(false);

adminMain.setVisible(true);

}

catch(SQLException sqle)

{

System.out.println("Error" + sqle);

}

}

else if(src==adminUpdate.btnCancel)

{

adminUpdate.setVisible(false);

adminMain.setVisible(true);

}

else if(src==adminViewAcct.btnDbBegin)

{

try{

String query1 = " SELECT \* FROM ClientInfo ";

String query2 = " SELECT Balance FROM ClientAccStatus ";

aDbase.tmpuprs = aDbase.tmpStmt.executeQuery(query1);

aDbase.tmpuprs.first();

acctno = aDbase.uprs.getLong(1);

String AcNo = Long.toString(acctno);

boolean val = aDbase.uprs.getBoolean(9);

String valid;

if (val)

valid = "Yes";

else

valid = "No";

adminViewAcct.fields[0].setText(" " + AcNo);

adminViewAcct.fields[1].setText(" " + aDbase.uprs.getString(2));

adminViewAcct.fields[3].setText(" " + valid);

adminViewAcct.fields[4].setText(" " + aDbase.uprs.getString(4));

adminViewAcct.fields[5].setText(" " + aDbase.uprs.getString(5));

adminViewAcct.fields[6].setText(" " + aDbase.uprs.getString(6));

adminViewAcct.fields[7].setText(" " + aDbase.uprs.getString(7));

adminViewAcct.fields[8].setText(" " + aDbase.uprs.getString(8));

balance = aDbase.tmpuprs.getLong(1);

String Bal = Long.toString(balance);

adminViewAcct.fields[2].setText(" Rs " + Bal + "\\-");

System.out.println("Admin View Acct |<");

}

catch(SQLException sqle)

{

System.out.println("Error :" + sqle);

}

}

else if(src==adminViewAcct.btnDbBwd)

{

//long acct = Long.parseLong(adminViewAcct.fields[0].getText()) ;

try{

if(!aDbase.uprs.isFirst())

{

aDbase.uprs.previous();

aDbase.tmpuprs.previous();

acctno = aDbase.uprs.getLong(1);

String AcNo = Long.toString(acctno);

boolean val = aDbase.uprs.getBoolean(9);

String valid;

if (val)

valid = "Yes";

else

valid = "No";

adminViewAcct.fields[0].setText(" " + AcNo);

adminViewAcct.fields[1].setText(" " + aDbase.uprs.getString(2));

adminViewAcct.fields[3].setText(" " + valid);

adminViewAcct.fields[4].setText(" " + aDbase.uprs.getString(4));

adminViewAcct.fields[5].setText(" " + aDbase.uprs.getString(5));

adminViewAcct.fields[6].setText(" " + aDbase.uprs.getString(6));

adminViewAcct.fields[7].setText(" " + aDbase.uprs.getString(7));

adminViewAcct.fields[8].setText(" " + aDbase.uprs.getString(8));

balance = aDbase.tmpuprs.getLong(1);

String Bal = Long.toString(balance);

adminViewAcct.fields[2].setText(" Rs " + Bal + "\\-");

System.out.println("Admin View Acct <<");

}

}

catch(SQLException sqle)

{

System.out.println("Error :" + sqle);

}

}

else if(src==adminViewAcct.btnDbFwd)

{

try{

if(!aDbase.uprs.isLast())

{

//acct = acct -1 ;

//String Actno = Long.toString(acct) ;

//String query1 = " SELECT \* FROM ClientInfo WHERE AccountNo = " + Actno ;

//String query2 = " SELECT Balance FROM ClientAccStatus WHERE AccountNo = " + Actno;

//aDbase.uprs = aDbase.tmpStmt.executeQuery(query1);

aDbase.uprs.next();

aDbase.tmpuprs.next();

acctno = aDbase.uprs.getLong(1);

String AcNo = Long.toString(acctno);

boolean val = aDbase.uprs.getBoolean(9);

String valid;

if (val)

valid = "Yes";

else

valid = "No";

adminViewAcct.fields[0].setText(" " + AcNo);

adminViewAcct.fields[1].setText(" " + aDbase.uprs.getString(2));

adminViewAcct.fields[3].setText(" " + valid);

adminViewAcct.fields[4].setText(" " + aDbase.uprs.getString(4));

adminViewAcct.fields[5].setText(" " + aDbase.uprs.getString(5));

adminViewAcct.fields[6].setText(" " + aDbase.uprs.getString(6));

adminViewAcct.fields[7].setText(" " + aDbase.uprs.getString(7));

adminViewAcct.fields[8].setText(" " + aDbase.uprs.getString(8));

balance = aDbase.tmpuprs.getLong(1);

String Bal = Long.toString(balance);

adminViewAcct.fields[2].setText(" Rs " + Bal + "\\-");

System.out.println("Admin View Acct >>");

}

}

catch(SQLException sqle)

{

System.out.println("Error :" + sqle);

}

}

}

/\*\*

\* processes events for the JTabbedPane.

\*

\* @param e the ChangeEvent object.

\*/

public void stateChanged(ChangeEvent e) {

Object src=e.getSource();

if (src == tabbedPane) { // click on a tab

showClientData();

}

}

/\*\*

\* Adds new tab to tabbedPane,

\* the same component is displayed for all tabs,

\* so if there are not tabs the pInnerTab is added.

\*/

private void addNewTab() {

try {

int curTabs = tabbedPane.getTabCount();

if (curTabs == 0) { // no tabs in tabbedPane?

tabbedPane.addTab("Client 1", icon, pInnerTab);

tabbedPane.setSelectedIndex(0);

}

else {

// add empty tab, component from Tab#0 will be used:

tabbedPane.addTab("Client " + (curTabs+1), icon, null);

// activate last tab (newly added):

tabbedPane.setSelectedIndex(curTabs);

}

}

catch (Exception e) {

System.err.println("addNewTab() -> " + e);

}

}

/\*\*

\* Removes the last tab from tabbedPane,

\* (used when game is terminated).

\*/

private void removeLastTab() {

try {

int curTabs = tabbedPane.getTabCount();

tabbedPane.removeTabAt(curTabs-1);//check for the correction..PENDING......

// activate first tab (if any):

if (curTabs > 1)

tabbedPane.setSelectedIndex(0);

}

catch (Exception e) {

System.err.println("removeLastTab() -> " + e);

}

}

/\*\*

\* Updates the text box that shows the information for

\* currectly selected client (by the tabbedPane).

\*/

private void showClientData() {

try {

AccessServer temp; // temporary pointer.

temp = (AccessServer) clients.get(tabbedPane.getSelectedIndex());

String sInfo = temp.getInfo();

if (!sInfo.equals( txtInfo.getText() )) // update text only when required.

txtInfo.setText(sInfo);

}

catch (ArrayIndexOutOfBoundsException ae) {

txtInfo.setText( "No client with index: " + tabbedPane.getSelectedIndex());

}

}

/\*\*

\* Main function, from where Server starts.

\*/

public static void main(String args[]) {

// validate parameter count:

if (args.length!=1) {

System.err.println("Wrong parameters! Usage:");

System.err.println("java Server <client\_port> ");

System.exit(1);

}

// process parameters:

try {

client\_port = Integer.parseInt(args[0]);

}

catch (NumberFormatException e) {

System.err.println("Wrong number for a port -> " + e);

System.exit(1);

}

// get address of the server:

try {

localHost = InetAddress.getLocalHost();

} catch (UnknownHostException e) {

System.out.println("Unknown host - probably localhost with no IP!");

// no exit, since can work on "localhost" without internet.

}

// print out the info (the same info is also shown on the server's

// GUI window).

System.out.println("Server is running on host: " + localHost);

System.out.println("Waiting clients on port: " + client\_port);

// create & start server GUI engine:

new Server();

} // end of main().

/\*\*

\* Closes the server, cares about closing all sockets,

\* and informing the clients are running

\* currenlty about the shutdown, and terminating clients.

\*/

private void closeApplication() {

// ask user if he/she is sure to shut down the server when

// there are clients running:

if (clients.size() > 0) {

int result;

result = JOptionPane.showConfirmDialog(this,

"Are you sure you want to shut down the SERVER?\n" +

"All clients will be terminated!",

"Close anyway?",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.WARNING\_MESSAGE);

if (result != 0) // no, cancel.

return;

// otherwise - yes, close.

// send termination message to all clients:

for (int i=clients.size()-1; i>=0; i--) {

AccessServer temp;

temp = (AccessServer) clients.get(i);

temp.sendToClient("TERMINATED");

removeClient(temp);

}

}

// stop the server's threads:

thClientAccept = null;

// close sockets:

try {

socketForClient.close();

}catch (IOException e) {

System.err.println("On close -> " + e);

}

// close everything:

System.exit(0);

}

public void setDisplay(){

adminEntry.setVisible(false);

adminCreate.setVisible(false);

adminDelete.setVisible(false);

adminEdit.setVisible(false);

adminMain.setVisible(false);

adminUpdate.setVisible(false);

adminViewAcct.setVisible(false);

//adminView.setVisible(false);

}

/\*\*

\* Terminates the client.

\*

\* @param clientToDelete the pointer to client to be deleted.

\*/

public void removeClient(AccessServer clientToDelete) {

if (clients.contains(clientToDelete)) { // check if not removed already.

// close sockets, streams, stop threads:

pause(1500);

clientToDelete.closeEverything();

clients.remove(clientToDelete); // remove from vector.

lblRunning.setText("Currently logged: " + clients.size() + " client(s).");

removeLastTab();

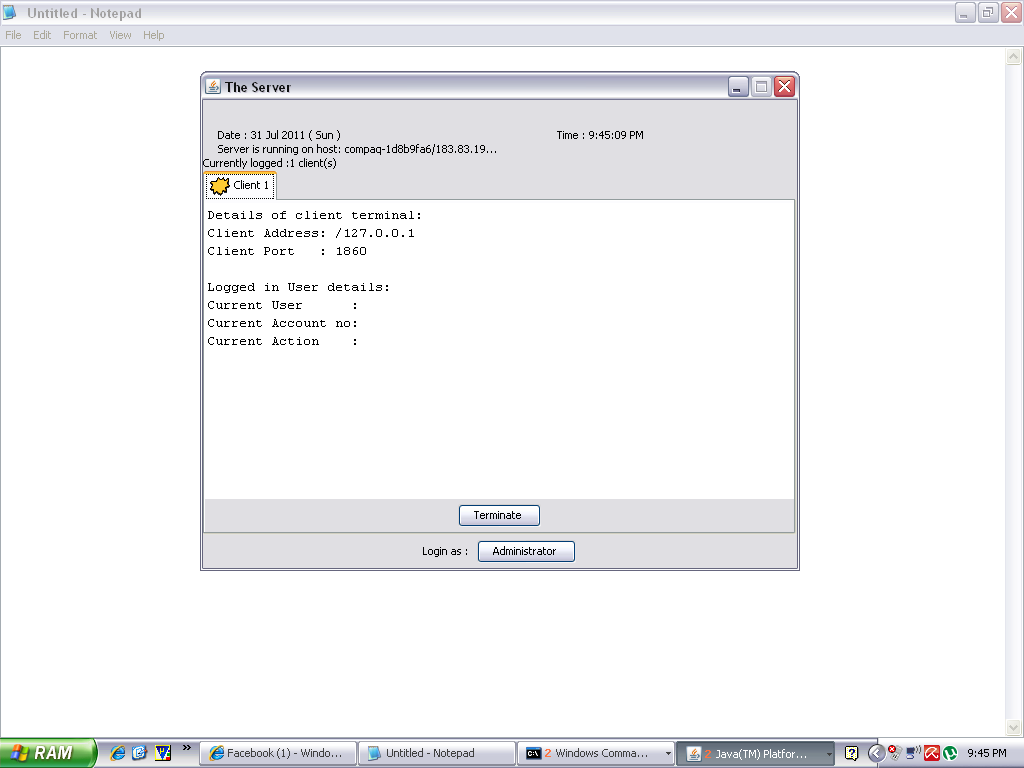
}

}

} //end of server class.

**SCREENSHOTS**

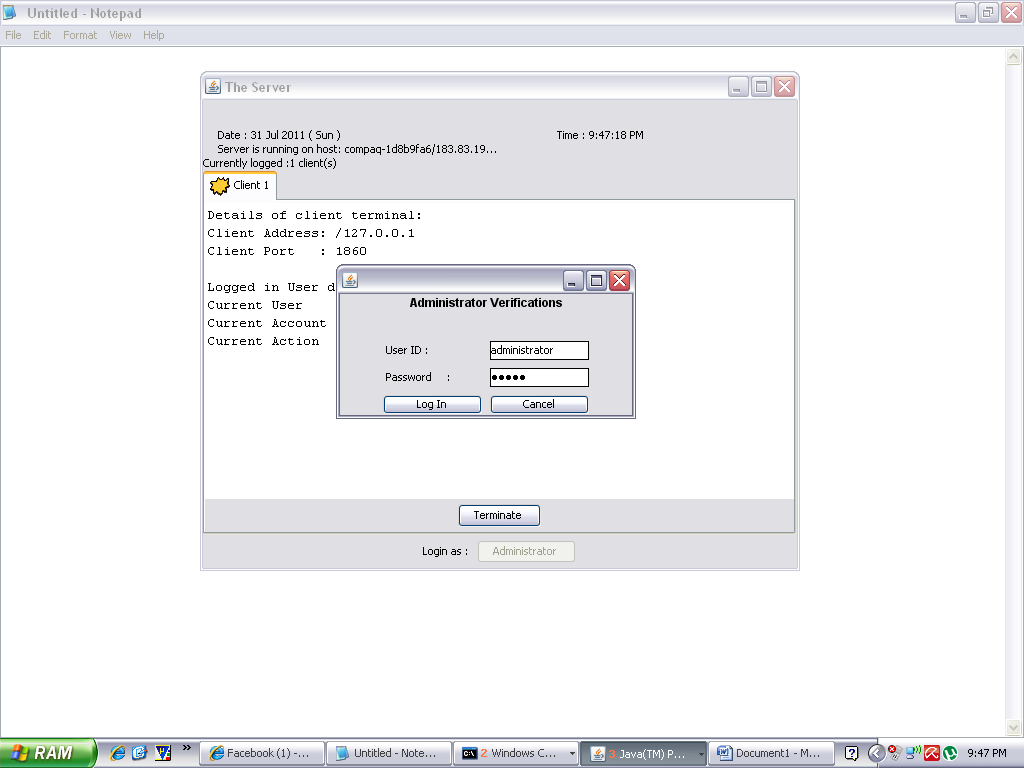
* **SCREENSHOT FOR ADMIN PAGE:**

****

**Fig: 7.3.1 screenshot for admin page.**

The screen is to enter in to the login page as after clicking the button administrator we will enter in to the login page.

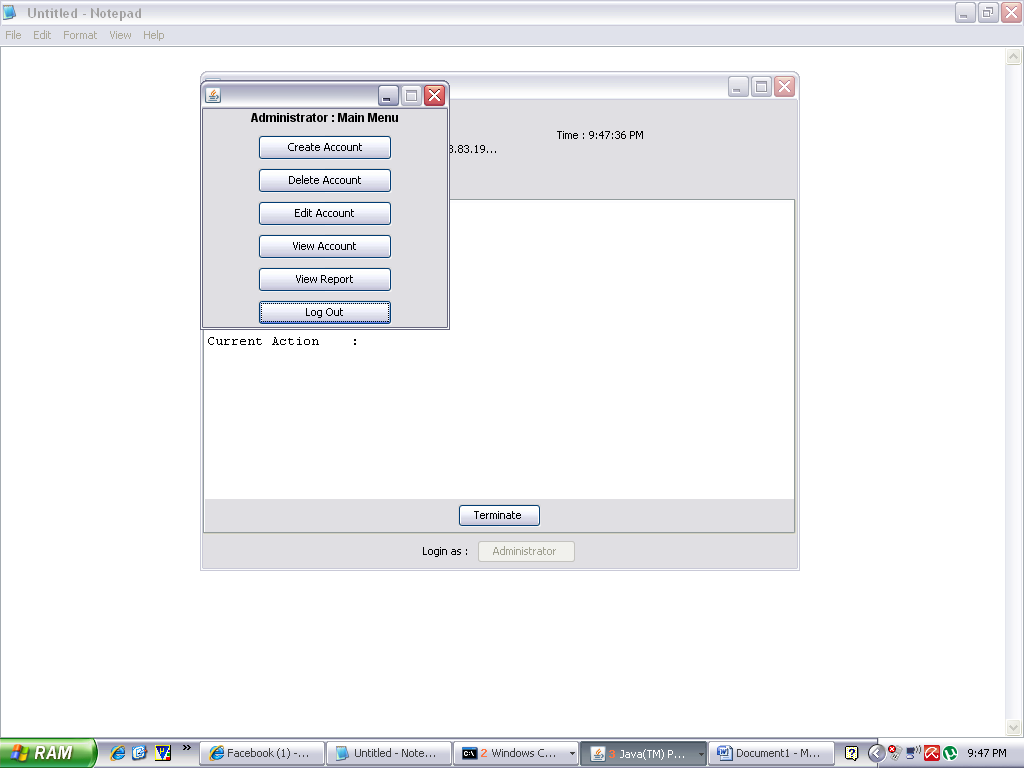
* **SCREENSHOT OF LONGIN PAGE.**

****

**Fig: 7.3.2 Screenshot of login page.**

This page is the login page as we clicked on administrator button on the admin page we entered into the login page as we can see the screen. After giving the user ID and PASSWORD admin can be login to the server side.

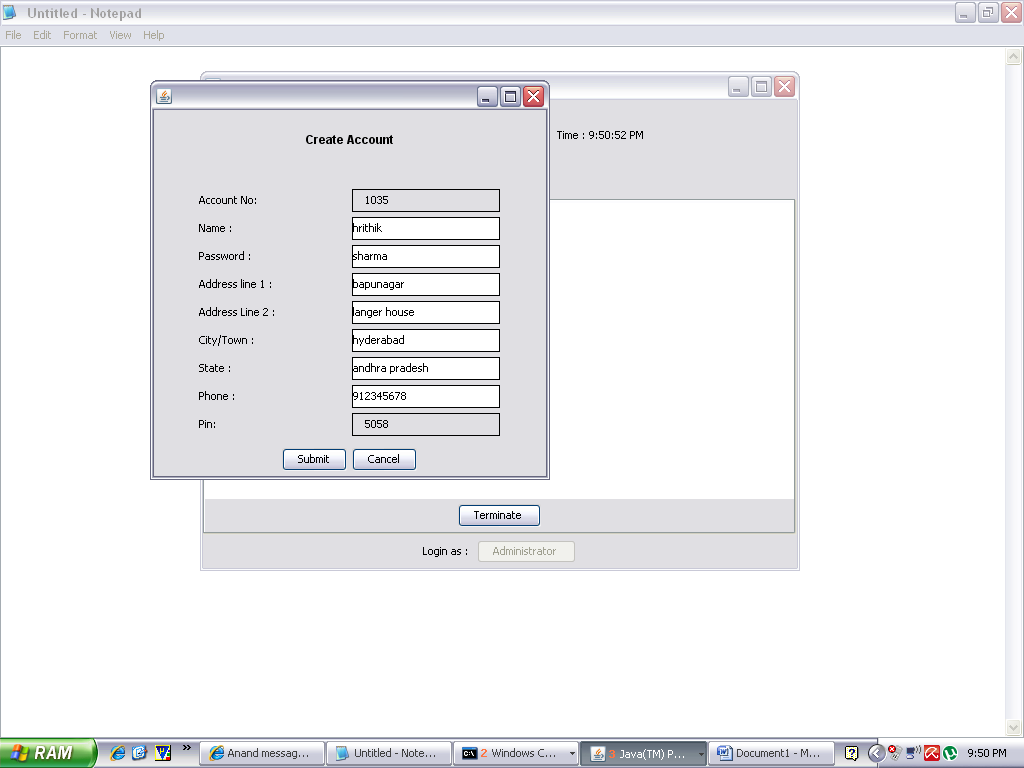
* **SCREENSHOT FOR MAIN MENU .**

****

**Fig 7.3.3 Screenshot for main menu**

This screenshot is for the main page main menu of admin where after login we get all the details of various steps performed by the admin like creating account, editing the account, deleting account etc.

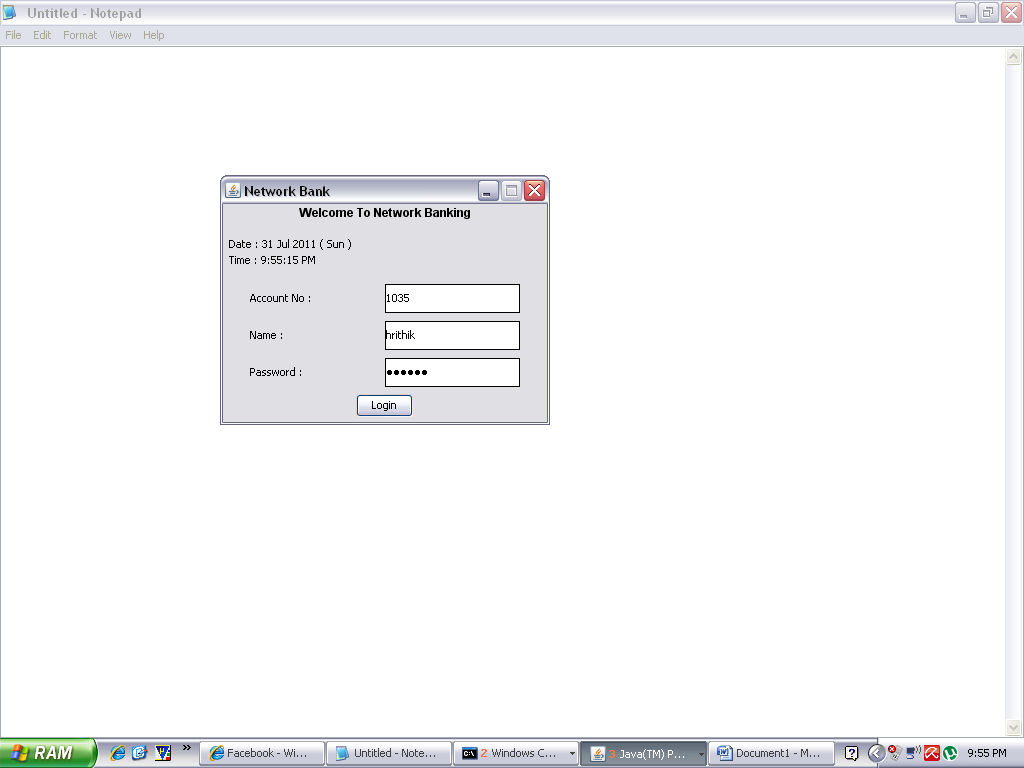
* **SCREENSHOT FOR CREATE ACCOUNT .**

****

**Fig 7.3.4 Screenshot for create account.**

Admin will create an account for the client by filling the above form and admin will provide user id and password for the client so he/she can only access their account. Admin can edit account, delete account, perform this operations by using client account number.

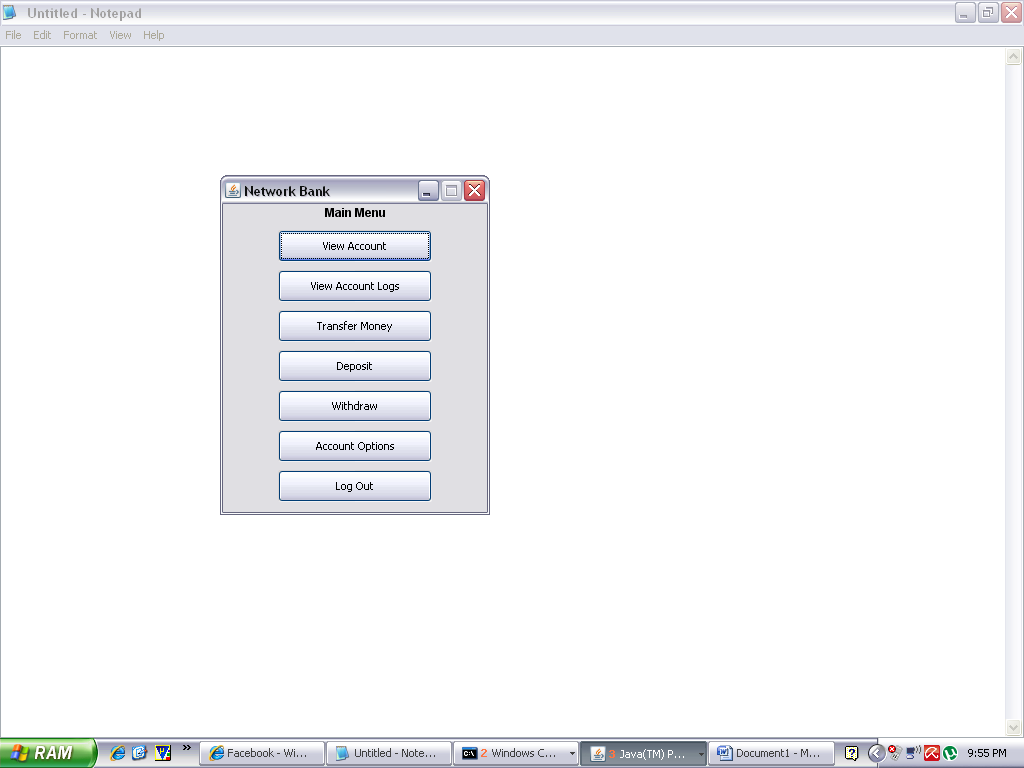
* **SCREENSHOT FOR LOGIN PAGE OF CLIENT .**

****

**Fig 7.3.5 Screenshot for login page of client.**

This screenshot is for the login page of the client and client can log in with user id and password and account number giving by the admin.

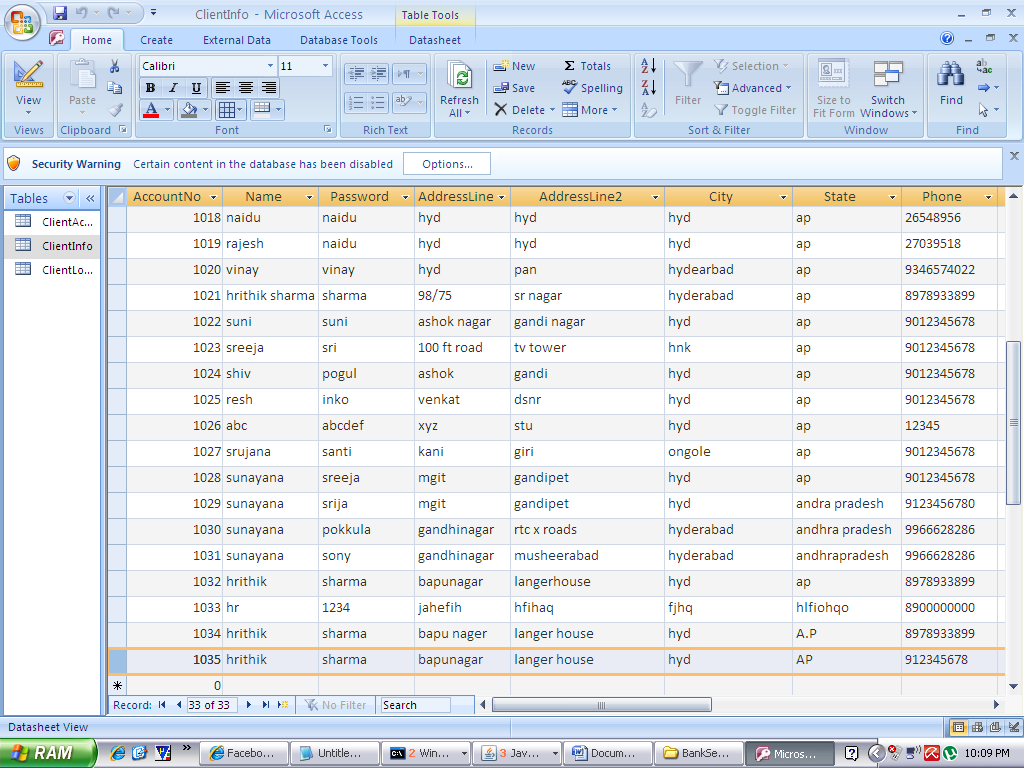
* **SCREENSHOT FOR MAIN MENU FOR THE CLIENT.**

****

**Fig 7.3.6 Screenshot for the main menu of the client.**

This screenshot for client after he/she login their account. Client can view their account by view account, transfer money, withdraw, account option, log out. Client can perform the action by using pin number giving by admin.

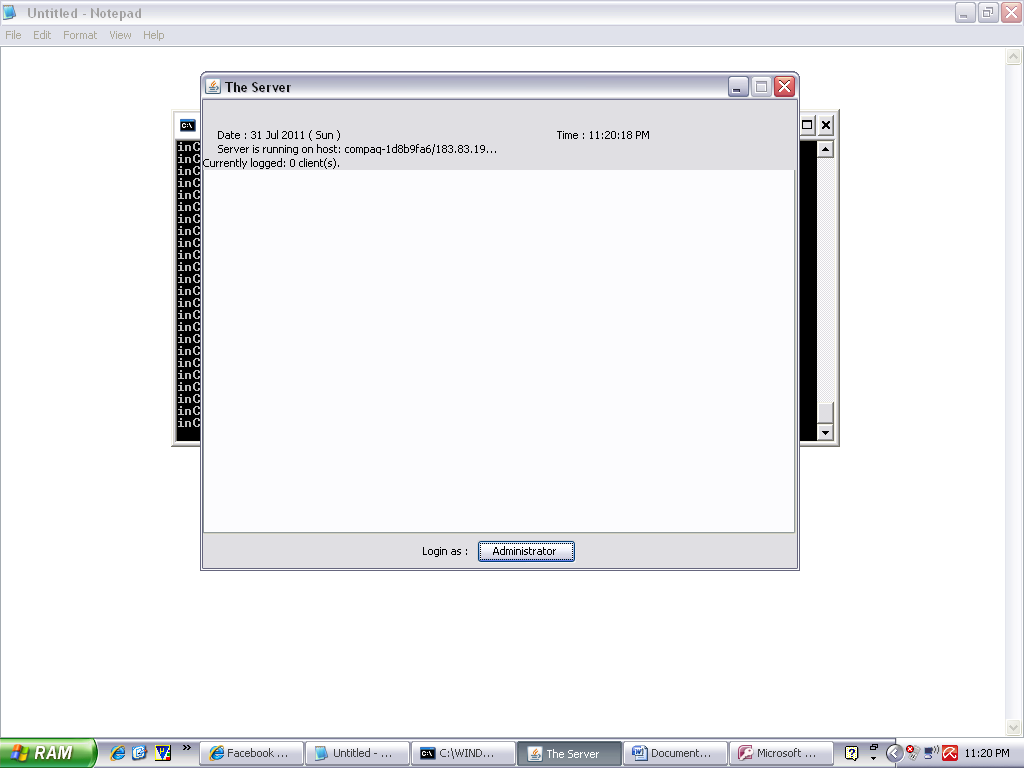
* **SCREENSHOT FOR VIEW OF CLIENT ACCOUNT TO THE ADMIN.**

****

**Fig 7.3.7 Screenshot for view of client account to the admin.**

This screenshot is for the administrator to view client account status, client account info, client log. Administrator can view all the client information. Only admin can view the above information.

* **SCREENSHOT FOR ADMINISTRATOR LOG OUT.**

****

**Fig 7.3.8 Screenshot for administrator logout page.**

This screenshot is for administrator logout page here admin is logout from the server. As the admin is logout automatically client will also logout from their account. After the logout of the client admin should logout from the server.