**CS 591 P1 Assignment 6 Bank ATM design\_doc**

**Xueyan Xia (U82450191)**

**\*\*Content\*\***

**1 Assumption and Hypothesis**

**2 Inheritance Relationship**

**3 Introduction of Functions of Classes**

**4 GUI design**

**1. Assumption and Hypothesis**

1. There are two kinds of IDs: ManagerID and CustomerID.

The ManagerID is used by a manager to view the information and get today transaction report. In this project we offer one ManagerID, which could log in and do corresponding operation.

The customerID is the same as an account or a user in reality and its uniqueness is ensured by an attribute called index (the attribute ‘username’ also make sense). When we say creating an account in the login page, we mean to create a customerID. In each customerID, there are three types of accounts and information of user including his address, phone number, etc. Here we have 3 customerIDs for testing, you can also create your new one.

2. A customer could create unlimited accounts.

There are 3 types of accounts. Checking accounts and saving accounts could be created without any requirement. However, a user could create a loans account only when he has a collateral.

As for closing an account, any accounts could be closed but the default account, which is the first checking account created automatically when a user create a customerID. Thus, the 3 customerIDs for testing only have 1 checking account in the beginning.

If you want to delete or close this default checking account, you need to go to the login page and choose the function of ‘close this account’ after entering username and password.

3. Customers and managers need to log in by their username and password for next operations.

4. A customer could return loans on the page of loans account.

When he decides to return loans, he enters an amount. this amount and the interests would be deducted from the default checking account.

5. A customer could get the interests of saving money in a saving account by himself. Such a function belongs to a specific bank class, but use a Timer() and let it pay the interests automatically is inconvenient for testing. Thus, I choose to let the customer do it by himself by GUI.

**2. Inheritance Relationship**

public abstract class ID {

protected String userName;

protected String password;

protected int index;

}

public class ManagerID extends ID{

public ManagerID(String userName, String password,int index) {

super(userName, password,index);

}

}

public class CustomerID extends ID {

private ArrayList<Account> accounts;

private String collateral = "";

private int currentCheAccount;

private int currentSavAccount;

private int currentLoaAccount;

private String name = "";

private String address = "";

private String phone = "";

}

The three attributes current…Account are used for different situations when a GUI of Customer Window is instantiated. Since a caustomer doesn’t have loans accounts and saving accounts at the beginning, we could not display a combobox of accounts list directly. By setting the value of ‘currentSavAccount’ and ‘currentLoaAccount’ as -1, the GUI ‘s constructor would not appear null pointer exception. Besides, the value of ‘currentCheAccount’ is 0, which means this is a checking account which everyone has, and we could charge transaction fee from it.

The abstract class is designed for possible expansion. In the future if the managers want to add some functions like VIP customer, we could inherit this abstract ID class.

public class Account {

protected int type;

protected String accountNumber;

protected HashMap<Integer, Balance> balances;

protected boolean condition;

}

public class CheckingAccount extends Account {

public CheckingAccount() {

super();

this.type = Bank.CHECKING\_ACCOUNT;

}

}

public class SavingAccount extends Account{

public SavingAccount() {

super();

this.type = Bank.SAVING\_ACCOUNT;

}

}

public class LoansAccount extends Account{

public LoansAccount() {

super();

this.type = Bank.LOANS\_ACCOUNT;

}

}

In these class, there are no methods except for getter and setter, thus we would not go into details here.

public abstract class Bank {

}

public class SCBank extends Bank {

}

public class ATM {

private static ArrayList<Bank> banks;

private Bank currentBank;

private Date date;

private SimpleDateFormat dateFormat;

…

}

According to the requirements of our assignment, we need to manage a specific bank which is addict in making money by an ATM machine. In this case, the ATM also need to support other normal banks for users to handle their transactions. Thus, I call the bank in requirement “SC\_Bank” and its class should inherit an abstract class. Currently, this SC\_Bank is the only bank in our ATM. But it could be more in the future.

We would talk about the specific methods in 1.3. The abstract methods would be highlight.

**3. Introduction of Functions of Classes**

In this part, I would skip the classes mentioned in part 2.

**ATM.class**

This class is the entrance and an easier interface between user (GUI) and bank system(data). It offers data to GUI and check if user’s operations and input values are valid. If yes, connecting to the bank class to continue next process; if no, returning information to GUI and expecting user’s next operation.

public class ATM {

private static ArrayList<Bank> banks;

private Bank currentBank;

private Date date;

private SimpleDateFormat dateFormat;

public ATM() {…

banks.add(new SCBank("SC"));

currentBank = (SCBank) banks.get(0);

date = new Date();

…}

public static void main (String[] args) {

new LoginWin(new ATM());

}

// ATM functions, interface between user GUI and banks

public int loginToSCBank(String username, String pwd, int type) {…}

public void updateCustomerInfo(CustomerID cId) {…}

public void transaction(CustomerID cId, Account acc, int inOrOut, float amount, int type) {…}

public void transfer(CustomerID cId, Account acc, String acc2, float amount, int balanceType) {…}

public boolean ifAccount(String s) {…} // check if the transferred account is valid

public void createAccount(CustomerID cId, int type) {…}

public void stopAccount(CustomerID cId, Account curAccount) {…}

public void loans(CustomerID cId, Account acc, int inOrOut, float amount, int type) {…}

public void getInterest(CustomerID cId) {…}

public void closeAccount(int index) {…}

public void createCustomerID(String name, String userName, String pwd, String address, String phone,…}

// offer data to display by GUI

public String getManagerID(int index) {…}

public ArrayList<CustomerID> getCustomerIDs() {…}

public CustomerID getCustomerID(int index) {…}

public ArrayList<Transaction> getTransactions(int index) {…}

// jumping between windows in some specific situations

public void newCustomerWin(ATM atm, CustomerID cId, boolean b, int win) {…}

}

**Bank.class**

Here I skip a lot of static final variables for setting parameters.

public abstract class Bank {

public Bank(String name) {…}

public static int accountCount;

protected String name;

protected ArrayList<ManagerID> managerIDs;

protected ArrayList<CustomerID> customerIDs;

protected ArrayList<Transaction> transactions;

// check if login information is valid

public abstract int ifManagerAccount(String username, String passward);

public abstract int ifCustomerAccount(String username, String passward);

// open a customerID

public abstract void createCustomer(String name, String username, String pwd, String address, String phone,

String collateral);

// bank transaction

public static void createLoansAccount(CustomerID cId) {…}

public static void createSavingAccount(CustomerID cId) {…}

public static void createCheckingAccount(CustomerID cId) {…}

public void deposit(CustomerID cId, Account acc, float amount, int type) {…}

public void withdrawal(CustomerID cId, Account acc, float amount, int type) {…}

public void borrowLoans(CustomerID cId, Account acc, float amount, int type) {…}

public void returnLoans(CustomerID cId, Account acc, float amount, int type) {…}

public void stopAccount(CustomerID cId, Account curAccount) {…}

public void closeAccount(int index) {…}

// charge transaction fee

public static void charge(CustomerID cId, float price) {…}

// save transaction information

public void addTransaction(int index, String name, String accNum, String info, Balance b) {…}

**SCBank.class**

public class SCBank extends Bank {

public static final float SC\_CHECKING\_ACCOUNT\_OPEN\_FEE = 10;

public static final float SC\_CHECKING\_ACCOUNT\_CLOSE\_FEE = 10;

public static final float SC\_SAVING\_ACCOUNT\_OPEN\_FEE = 10;

public static final float SC\_SAVING\_ACCOUNT\_CLOSE\_FEE = 10;

public static final float SC\_LOANS\_ACCOUNT\_OPEN\_FEE = 10;

public static final float SC\_LOANS\_ACCOUNT\_CLOSE\_FEE = 10;

public static final float SC\_TRANSACTION\_FEE = 10;

public static final float SC\_WITHDRAWAL\_FEE = 10;

public static final double SC\_LOANS\_INTEREST = 0.001;

public static final double SC\_SAVING\_INTEREST = 0.002;

public static final double SC\_SAVING\_INTEREST\_THRESOLD = 5000;

public SCBank(String name) {…}

// set data for testing

private void createManagers() {…}

private void createCustomers() {…}

public int ifManagerAccount(String username, String passward) {…}

public int ifCustomerAccount(String username, String passward) {…}

public void createCustomer(String name, String userName, String pwd, String address, String phone,

String collateral) {…}

}

**Transaction.class**

Save information in case of querying by customers or manager.

public class Transaction {

private String name;

private String info;

private Balance balance;

private String accountNumber;

private int index;

}

**Balance.class**

It is a combination of currency and money amount.

public class Balance {

private float money;

private String currency;}

**Account.class**

public class Account {

protected int type; // 3 types of account

protected String accountNumber; // uniqueness

protected HashMap<Integer, Balance> balances; // different currencies

protected boolean condition; // if this account is closed

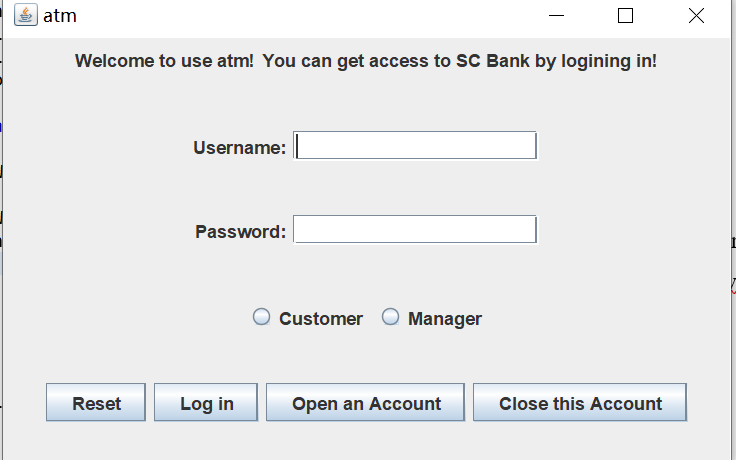
}

**4 GUI design**

In this part, I always use set…Win to instantiate the JComponent and use init…Win to set their position.

The temporary data would be underlined.

**LoginWin.class -- GridLayout**



This window is for customers and manager to login in and handle their transaction.

public class LoginWin {

…

private ATM atm;

public LoginWin(ATM atm) {

this.atm = atm;

setLoginWin();

initLoginWin();

}

private void setLoginWin() {…}

private void initLoginWin() {…}

class LoginListener implements ActionListener {

public void actionPerformed(ActionEvent e) {…}

}

**ManagerWin.class – GridBagLayout**

This window is for customers and manager to login in and handle their transaction.

public class ManagerWin {

…

private ArrayList<CustomerID> customers;

private String welcomeInfo;

private ATM atm;

public ManagerWin(ATM atm, int i) {

this.atm = atm;

welcomeInfo = "Hello " + this.atm.getManagerID(i) + "!";

setManagerWin();

initManagerWin();

}

private void setManagerWin() {…}

private void initManagerWin() {…}

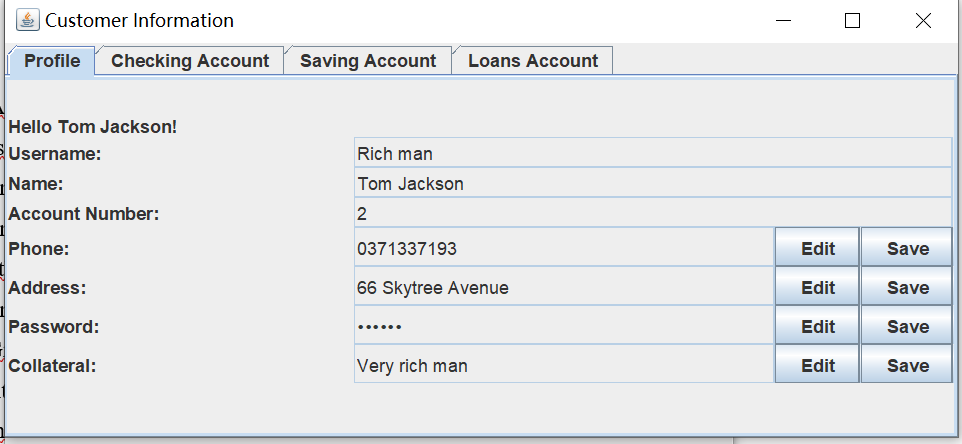
class FindCustomerListener implements ActionListener {…}

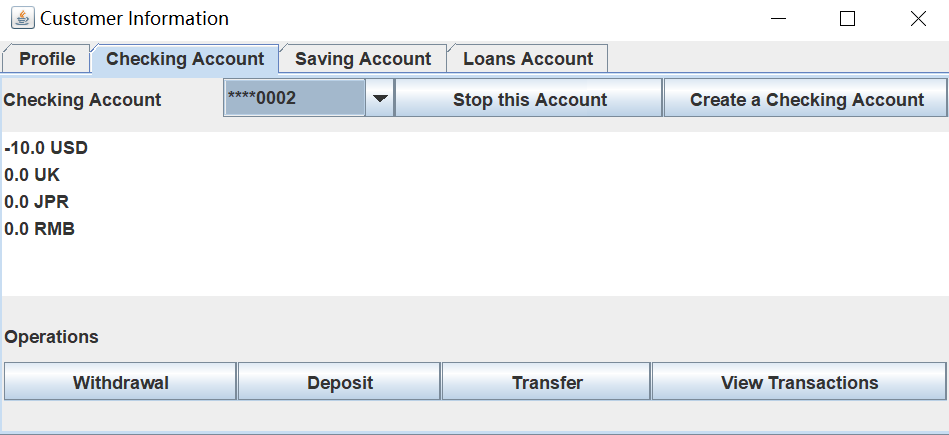
class FindTransactionListener implements ActionListener {…}

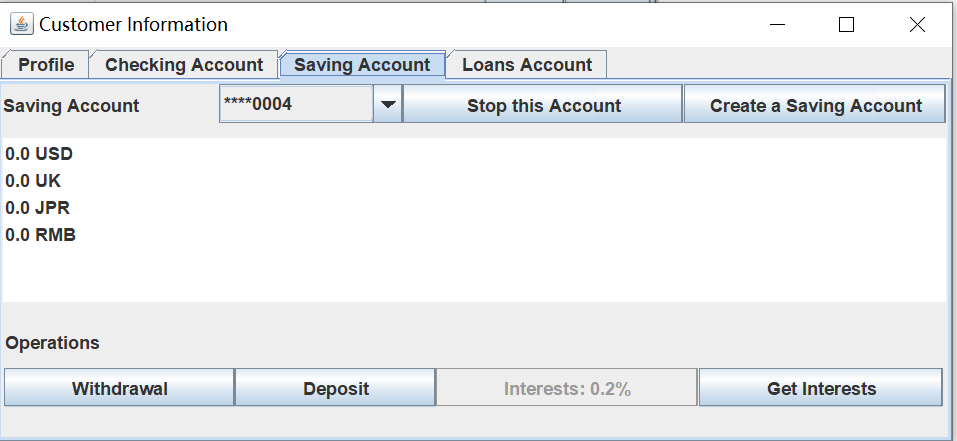
class SharedListSelectionHandler implements ListSelectionListener {…}

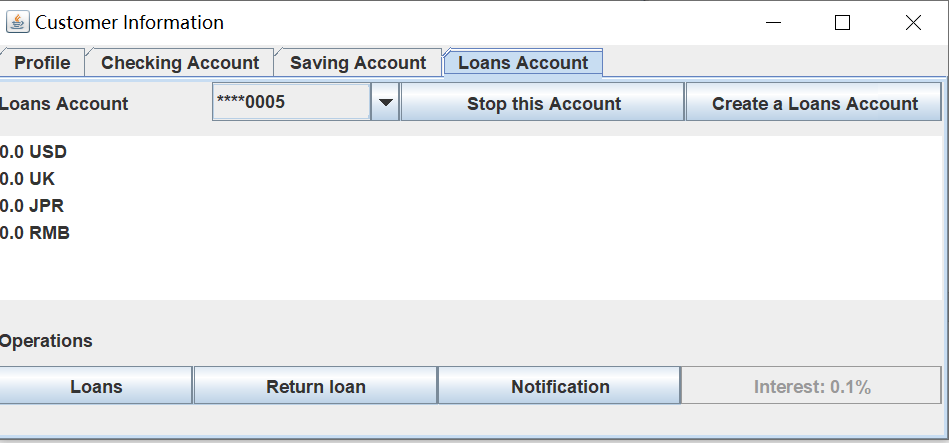
}

**CustomerWin.class – GridBagLayout**









This window can be used by both managers and customers.

When this window is visited by a manager, all of the buttons are unenabled and the text fields are uneditable. This is because a manager could only view the information and could not update or modify it.

When this window is visited by a customer, he can do operations based on his ID ‘s condition.

In this window, we have a JTabbedPane and 4 subpages. (Profile page, 3 types of accounts page)

Since the last three pages have similar layout, we create ArrayLists of Buttons, Labels and use a function *settingComponents()* to set the position of each component, avoiding too many similar code in this class. The use of gridbaglayout and JTabbedPane helps me build a more beautiful and concise GUI here.

public class CustomerWin {

…

private ArrayList<Account> cheAccounts;

private ArrayList<Account> savAccounts;

private ArrayList<Account> loaAccounts;

private Account curAccount;

private ATM atm;

public CustomerWin(ATM atm, int index, boolean flag, int win){}

private void setCustomerWin(boolean flag) {}

// init comboboxes of 3 types of account numbers

private void initComboBoxes() {}

// get comboboxes content

private String[] getAccountStr(ArrayList<Account> a) {}

// set some components unuseable, for managers

private void setUserRight(boolean flag) {}

private void initCustomerWin(int win) {}

private void setPanel1(){}

private void setPanel2(){}

private void setPanel3(){}

private void setPanel4(){}

private void settingComponents (int accountType, JComboBox<String> cb, JList<String> bList, ArrayList<JPanel> ps, ArrayList<JLabel> ls, ArrayList<JButton> bs, ArrayList<GridBagLayout> gbs, ArrayList<GridBagConstraints> gbcs)

private void initPanel3(){}

private void initPanel2() {}

private String[] getBalalceList(int i, int accountType) {}

private void initPanel1() {}

private void add(GridBagLayout gb, Component c, GridBagConstraints gbc, int gridwidth, int gridheight, int weightx, int weighty) {}

class ProfileButtonListener implements ActionListener {}

class CheSavAccButtonListener implements ActionListener {}

class AccOnOffButtonListener implements ActionListener {}

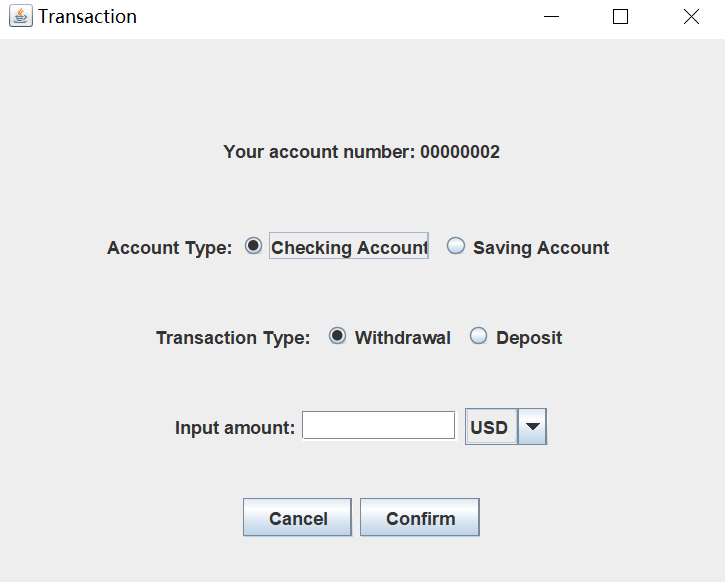
class TransferButtonListener implements ActionListener {}

class LoaButtonListener implements ActionListener {}

class ComboBoxListener implements ActionListener {}

}

**TransactionWin.class -- GridLayout**



This window is for customers to withdrawal or deposit money by checking account or saving account.

To make a more user-friendly GUI, the default setting of these radio buttons depends on the parameter from last window. In this case, the user could omit some operations. The user could also change their choice (checking account/saving account, deposit/withdrawal) in this window.

public class TransactionWin {

…

private CustomerID cId;

private Account acc;

private int inOrOut;

private ATM atm;

public TransactionWin(ATM atm, int inOrOut, CustomerID cId, Account acc) {

this.acc = acc;

this.cId = cId;

this.atm = atm;

setTransactionWin();

initTransactionWin(acc.getType(), inOrOut);

}

private void setTransactionWin() {…}

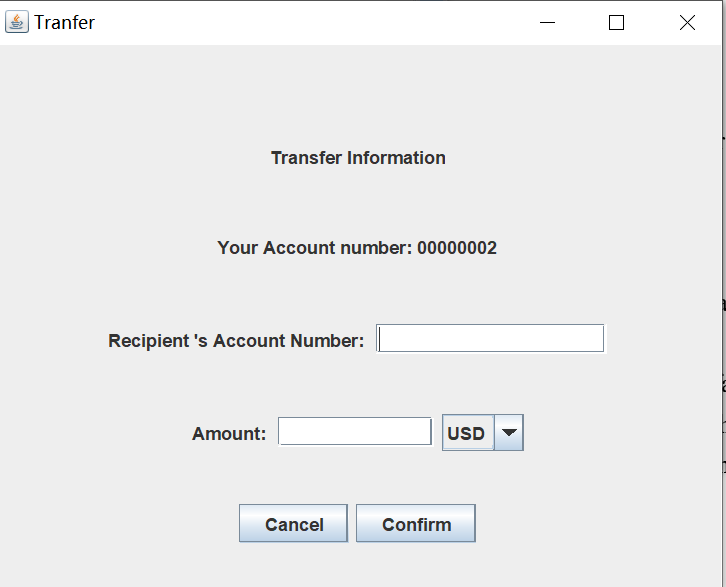
private void initTransactionWin(int accountType, int inOrOut) {…}

class ButtonListener implements ActionListener {…}

public boolean ifAvailable(Account acc, float amount, int type) {…}

}

**TransferWin.class – GridLayout**



This window is for customers to transfer money to another account.

In ATM.class, this method is designed for transferring between any two accounts.

public class TransferWin {

private CustomerID cId;

private Account acc;

private int balType;

private ATM atm;

public TransferWin(ATM atm,CustomerID cId, Account acc) {

this.atm = atm;

this.acc = acc;

this.cId = cId;

balType = Bank.USD;

setTransferWin();

initTransferWin();

}

private void setTransferWin() {…}

private void initTransferWin() {…}

class ButtonListener implements ActionListener {…}

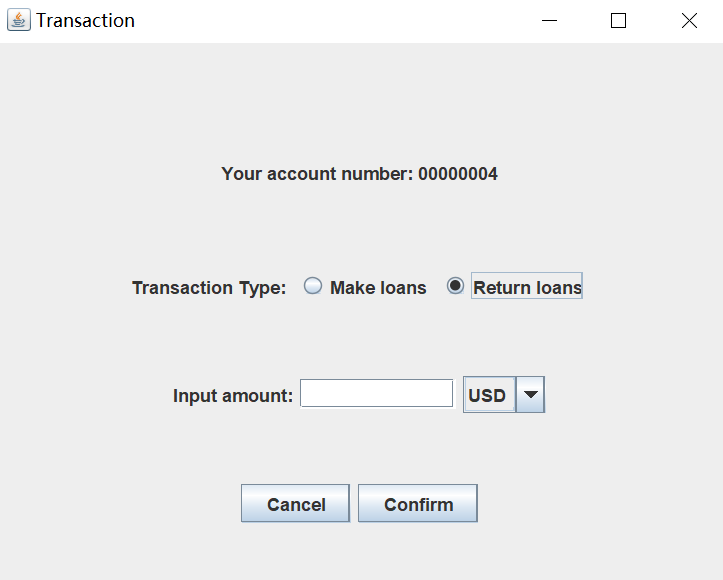
class ComboBoxListener implements ItemListener {…}

// if this account have enough amount to transfer to another account

public boolean ifAvailable(Account acc, float amount, int type) {…}

}

**LoansWin.class -- GridLayout**



This window is for customers to make loans or return loans.

public class LoansWin {

…

private CustomerID cId;

private Account acc;

private int inOrOut;

private ATM atm;

public LoansWin(ATM atm, int inOrOut, CustomerID cId, Account acc) {

this.acc = acc;

this.cId = cId;

this.atm = atm;

setLoansWin();

initLoansWin(inOrOut);

}

private void setLoansWin() {…}

private void initLoansWin(int inOrOut) {…}

class ButtonListener implements ActionListener {…}

// if the default checking account has enough amount to return this loan

public boolean ifAvailable(CustomerID cId, float amount, int type) {…}

}

**ViewTransactionWin.class – GridBagLayout**

This window is for customers and manager to view a list of their transaction.

public class ViewTransactionWin {

…

private ArrayList<Transaction> transactions;

private ATM atm;

public ViewTransactionWin(ATM atm, int index) {…}

private void initViewTransactionWin() {…}

private void setViewTransactionWin() {…}

}