PROJECT REPORT

CS-586 Software Systems Architecture

Vikas Pathak

A20460927

vpathak1@hawk.iit.edu

Contents:

1. Introduction

Goal: The goal of this project is to design two different ACCOUNT components using a Model-Driven Architecture (MDA) and then implement these ACCOUNT components based on this design.

Description of the Project: There are two ACCOUNT components: ACCOUNT-1 and ACCOUNT-2:

ACCOUNT-1 component supports the following operations:

open (int p, int y, int a) // open an account where p is a pin, y is an user's identification #, and a is a balance

pin (int x) // provides pin #

deposit (int d); // deposit amount d

withdraw (int w); // withdraw amount w

balance (); // display the current balance

login(int y) // login where y is a client's identification #

logout() // logout from the account

lock(int x) // locks an account where x is a pin

unlock(int x) // unlocks an account where x is a pin

ACCOUNT-2 component supports the following operations:

OPEN (int p, int y, float a) // open an account where p is a pin, y is an user's identification #, and a is a balance

PIN (int x) // provides pin #

DEPOSIT (float d); // deposit amount d

WITHDRAW (float w); // withdraw amount w

BALANCE (); // display the current balance

LOGIN(int y) // login where y is a client's identification #

LOGOUT() // logout from the account

suspend() // suspends an account

activate() // activates a suspended account

close() // an account is closed

Both ACCOUNT components are state-based components and support three types of transactions:

withdrawal, deposit, and balance inquiry. Before any transaction can be performed, operation open(p, y, a) (or OPEN(p, y, a)) must be issued, where y is a client's identification #, p is a pin used to get permission to perform transactions and a is an initial balance in the account. It is assumed that open()/OPEN() operation is issued only once for a given account. Before any transaction can be performed, operation login(y) must be issued (where y is a client's identification #) followed by pin(x) (or PIN(x)) operation. The pin(x) (or PIN(x)) operation must contain the valid pin # that must be the same as the pin # provided in open(p, y, a) (or OPEN(p, y, a)) operation. There is a limit on the number of attempts with an invalid pin. The account can be overdrawn (below minimum balance), but a penalty may apply. If the balance is below the minimum balance then the withdrawal transaction cannot be performed. The account may become locked by lock operation or suspended by suspend operation. If the account is locked, withdrawal, deposit, logout and balance transactions cannot be performed. A locked account becomes unlocked by unlock operation. A suspended account can be activated by activate operation. In addition, a suspended account can be closed by close operation. The detailed behavior of both ACCOUNT components is specified using EFSM. The EFSM of Figure 1 shows the detail behavior of ACCOUNT-1, and the EFSM of Figure 2 shows the detailed behavior of

ACCOUNT-2. Notice that there are several differences between both ACCOUNTs.

Aspects that vary between two ACCOUNT components:

a. Maximum number of times incorrect pin can be entered

b. Minimum balance

c. Display menu(s)

d. Messages, e.g., error messages, etc.

e. Penalties

f. Operation names and signatures

g. Data types

h. etc.

The goal of this project is to design two ACCOUNT components using a Model-Driven Architecture (MDA) covered in the course. An executable meta-model, referred to as MDA-EFSM, of ACCOUNT components should capture the “generic behavior” of both ACCOUNT components and should be decoupled from data and implementation details. Notice that in your design there should be ONLY one MDA-EFSM for both ACCOUNT

components. The meta-model (MDA-EFSM) used in the ModelDriven architecture should be expressed as an EFSM (Extended Finite State Machine) model. Notice that the EFSMs shown in Figures 1 and Figure 2 are not acceptable as a meta-model (MDA-EFSM) for this model driven architecture.

2. EFSM model for the ACCOUNT components

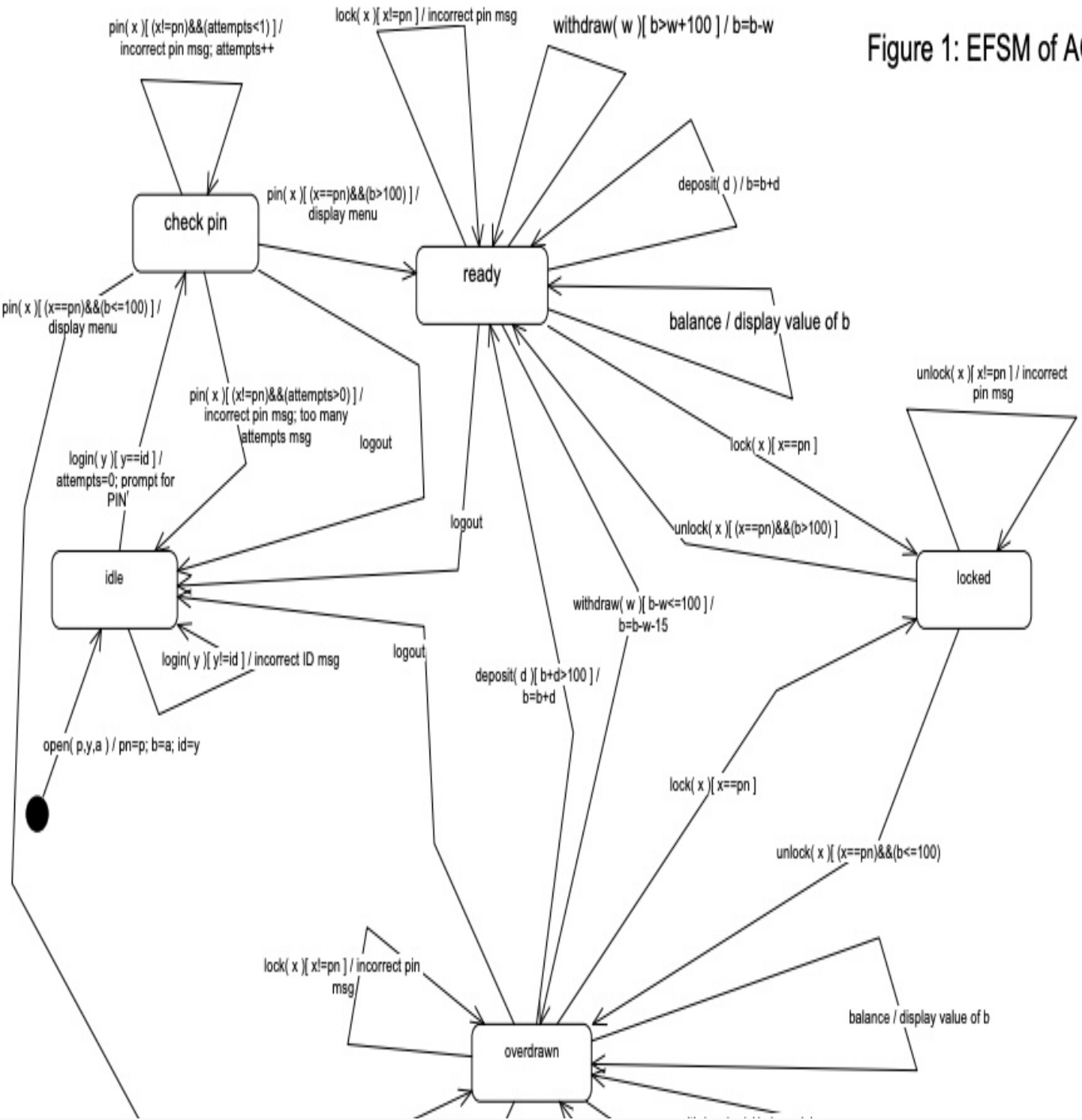
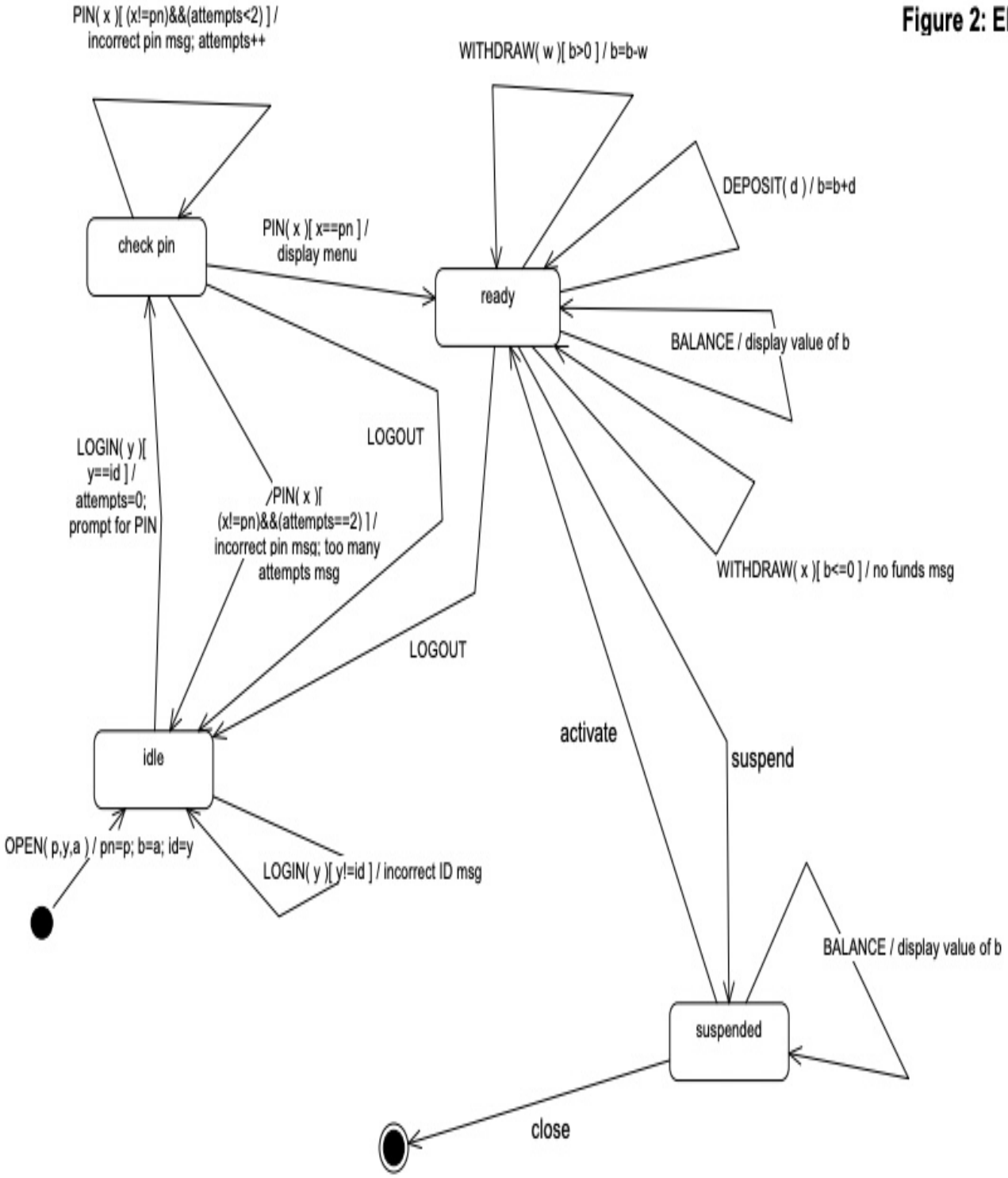
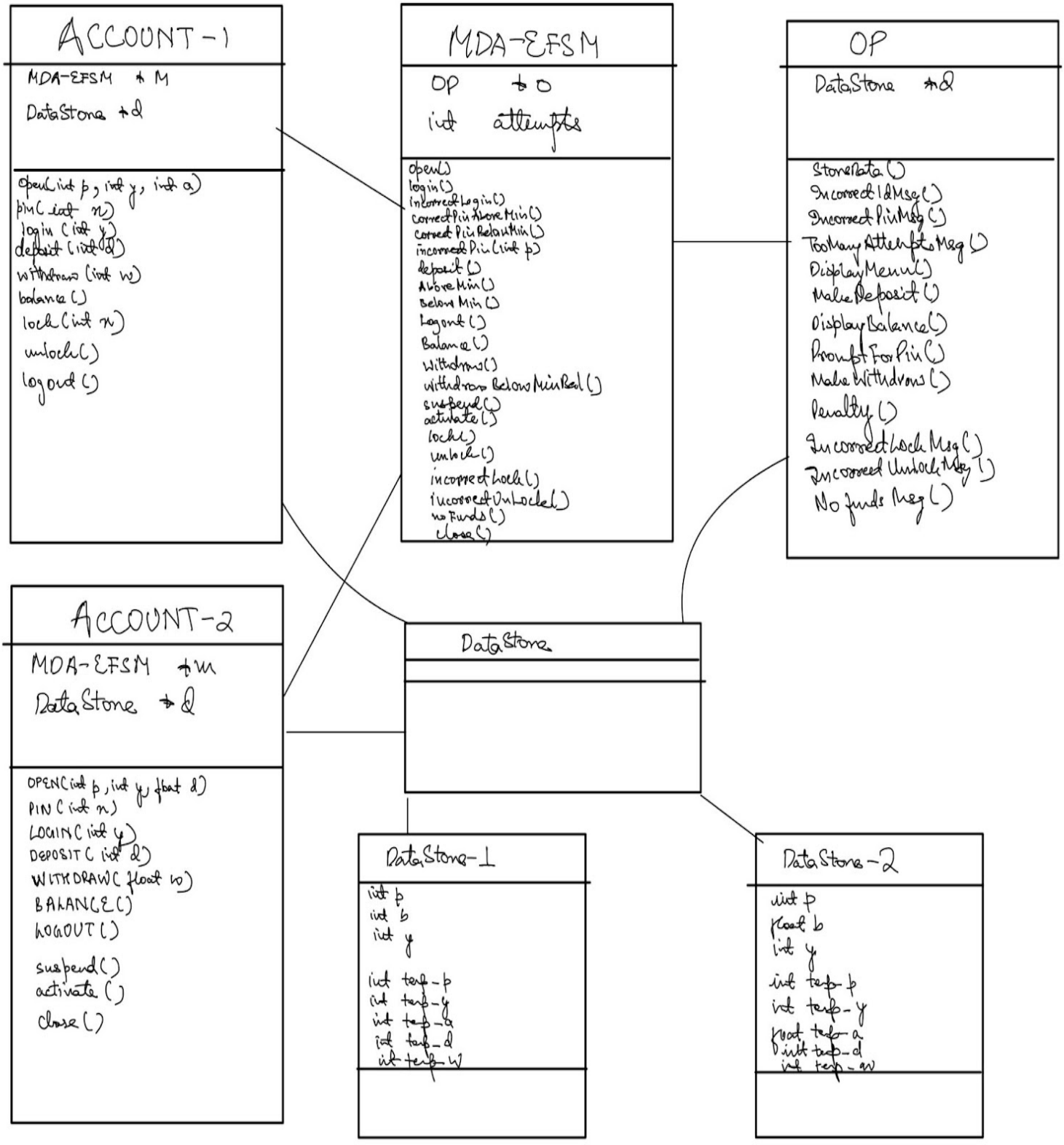


Figure 1: EFSM of ACCOUNT-1

Figure 2: EFSM of ACCOUNT-2



3. Class diagram of the MDA-EFSM MODEL



4. A list of meta events for the MDA-EFSM

e1: Open()
e2: login()
e3: IncorrectLogin()
e4: IncorrectPin(int p)
e5: CorrectPinBelowMin()
e6: CorrectPinAboveMin()
e7: Deposit()
e8: BelowMin()
e9: AboveMin()
e10: Logout()
e11: Balance()
e12: Withdraw()
e13: WithdrawBelowMinBal()
e14: NoFunds()
e15: Lock()
e16: Incorrectlock()
e17: UnLock()
e18: IncorrectUnlock()
e19: Suspend()
e20: Activate()
e21: close()

5. A list of meta actions for the MDA-EFSM

Meta Actions

A1: StoreData() // stores pin from temporary data store to "p" in dataStore.uid
// similarly balance from temporary dataStore to "b" & "y" in dataStore

A2: IncorrectIdMsg() // display Incorrect Id Message.

A3: IncorrectPinMsg() // display Incorrect Pin Message

A4: TooManyAttemptsMsg() // displays too many attempts Message

A5: DisplayMenu() // displays a menu with a list of transactions

A6: MakeDeposit() // makes deposit (increases balance by a value stored in temp. dataStore)

A7: DisplayBalance() // display the current value of the balance

A8: PromptForPin() // prompts to enter pin

A9: MakeWithdraw() // makes withdraw (decreases balance by a value stored in temp. data Store)

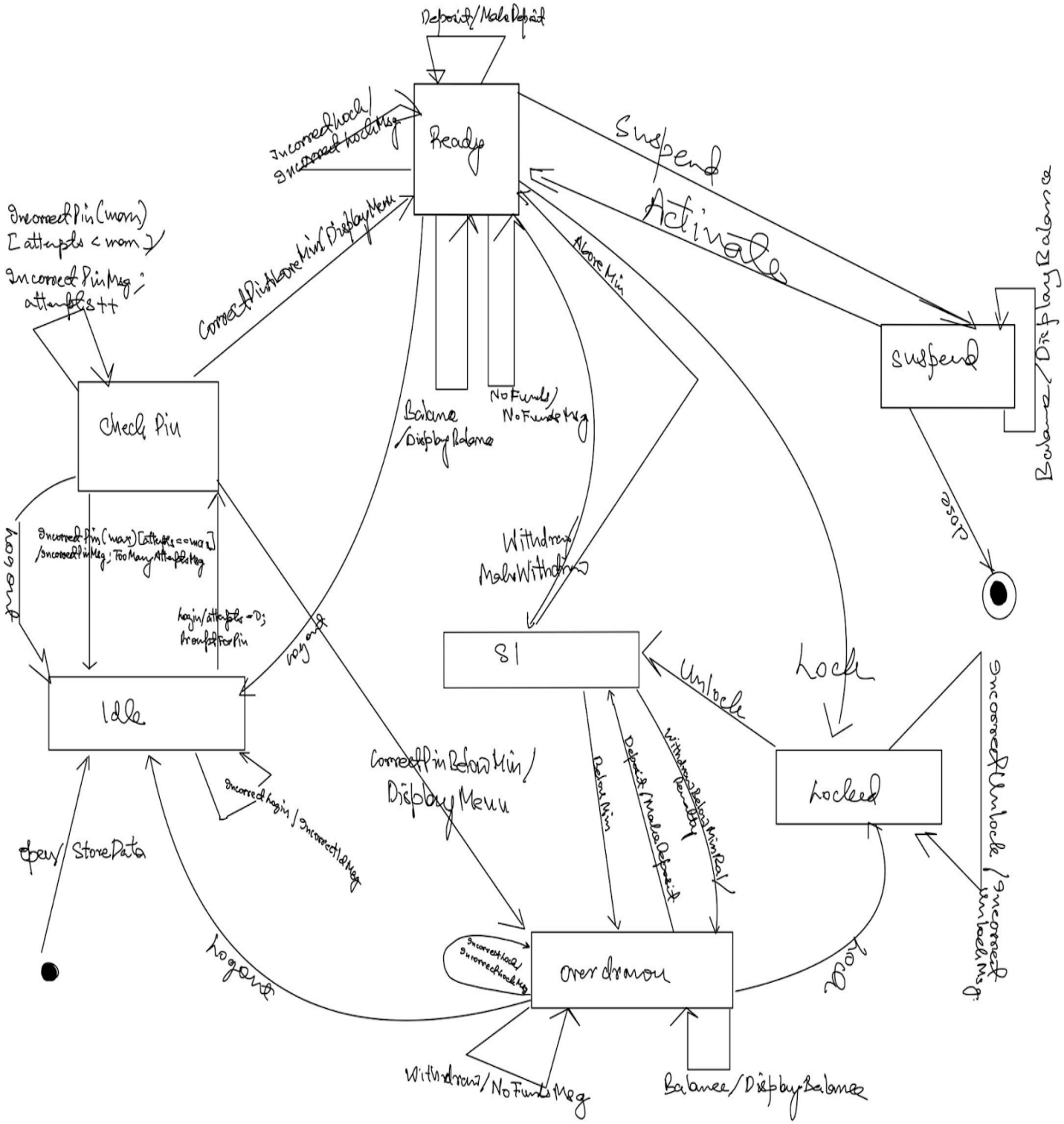
A10: Penalty() // applies penalty (decreases balance by the amount of penalty)

A11: IncorrectLockMsg() // displays Incorrect lock message

A12: IncorrectUnlockMsg() // displays Incorrect Unlock message

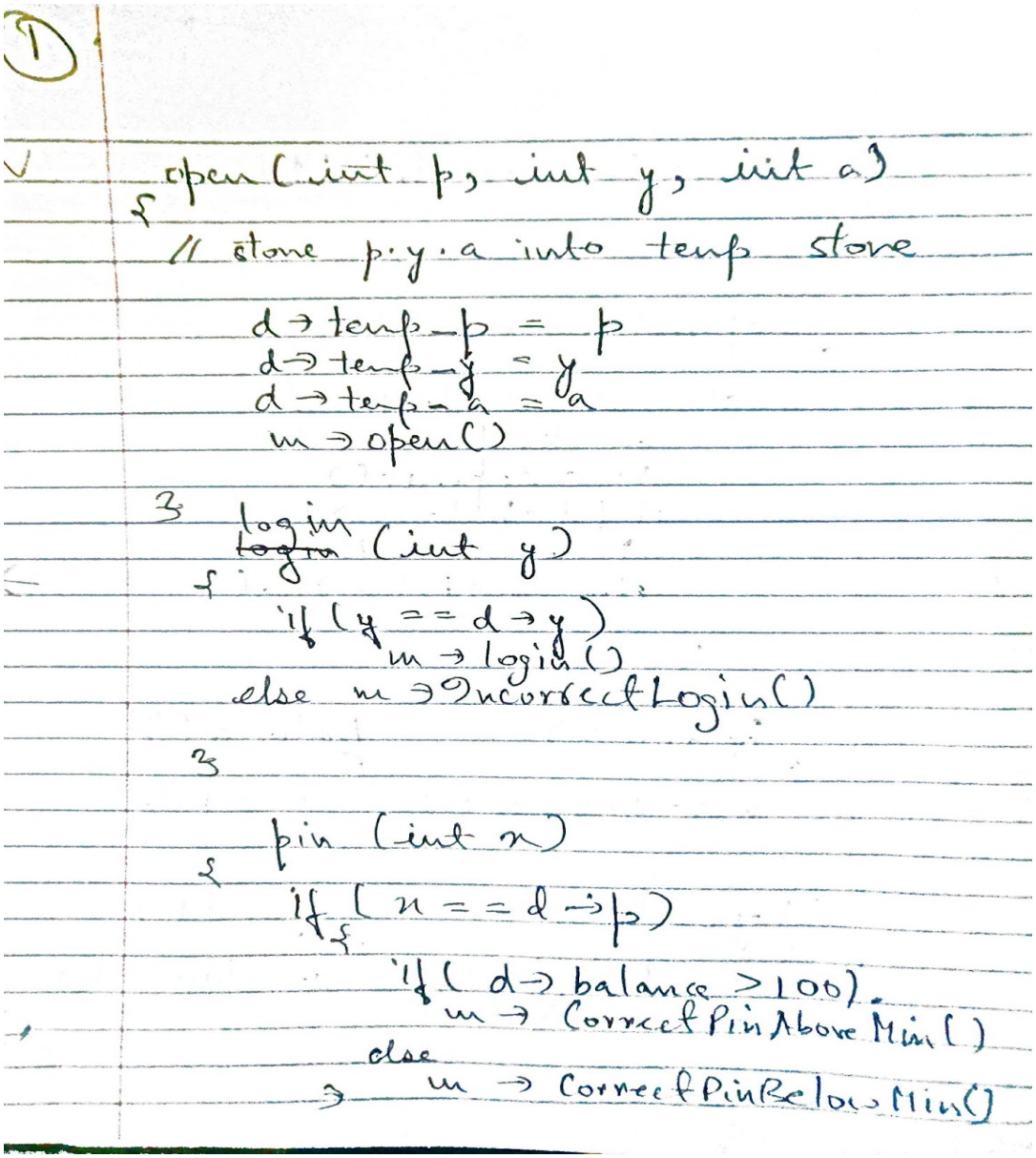
A13: NoFundsMsg() // displays no funds message

6. A state diagram/model of the MDA-EFSM



7. Pseudo-code of all operations of Input Processors of ACCOUNT-1 and ACCOUNT-2

Operations of the Input Processor (ACCOUNT-1):



```
①
open (int p, int y, int a)
{
// store p.y.a into temp store
    d → temp_p = p
    d → temp_y = y
    d → temp_a = a
    m → open()

3
login (int y)
{
    if (y == d → y)
        m → login()
    else m → IncorrectLogin()
}

pin (int n)
{
    if ( n == d → p)
        if ( d → balance > 100)
            m → CorrectPinAboveMin()
        else
            m → CorrectPinBelowMin()
```

```
m -> Incorrectpin(1)
}

deposit(int d)
{
  // store d int temp_d
     d -> temp_d = d;
     m -> deposit()
    if(d -> balance > 100)
        m -> AboveMin()
    else m -> BelowMin()

  }
```

```
balance()                    balance()
{                            {
  m -> balance()               m -> balance()
}                            }

logout()                     logout()
{                            {
  m -> logout()                m -> logout()
}                            }
```

```
withdraw ( int w) {
    d -> temp_w = w
    m -> withdraw()

    if (d -> balance > 100)
        m -> AboveMin()
    else m -> WithdrawBelowMinBal()

}

lock (int n)
{
    if (d -> pin == n) m -> Lock()
    else m -> IncorrectLock()

}

unlock (int n) {

    if (x == d -> pin) {
        m -> Unlock();
    if (d -> b > 100)
        m -> AboveMin()
    else m -> BelowMin()

    }
    else m -> IncorrectUnlock()
}
```

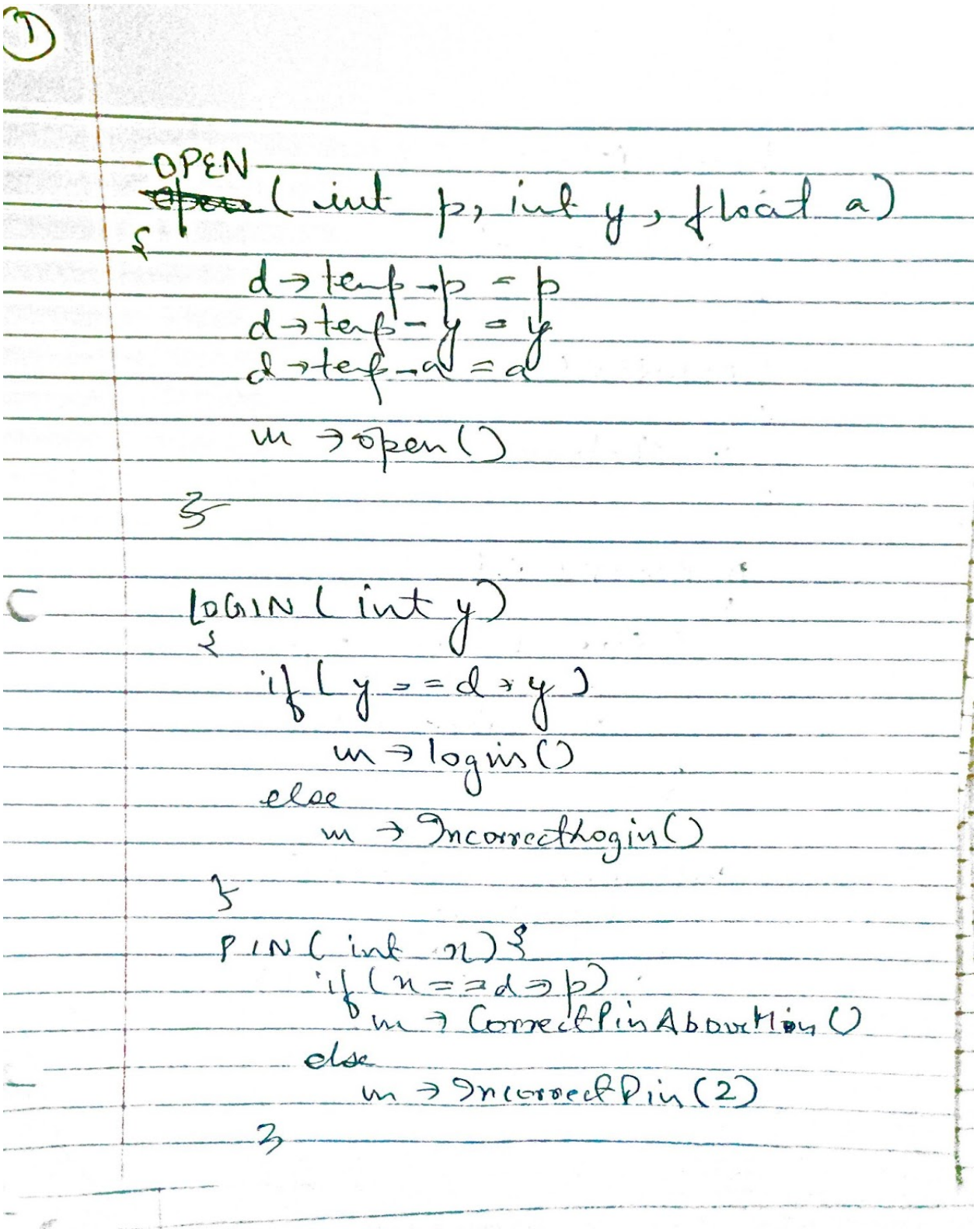
Note :

m : is a pointer to the MDA-EFSM object

d : is a pointer to the DataStore object which contains the following data items.

- b : contains the current balance
- p : contains the correct pin
- y : contains the correct authorization ID
- temp_p, temp_a, temp_y, temp_d, temp_w are used to store values of temporary parameters.

Operations of the Input Processor (ACCOUNT-2):



```
OPEN
(int p, int y, float a)
{
    d->temp->p = p
    d->temp->y = y
    d->temp->a = a

    m->open()

}

LOGIN(int y)
{
    if(y==d->y)
        m->login()
    else
        m->IncorrectLogin()

}

PIN(int n){
    if(n==d->p)
        m->CorrectPinAbovetion()
    else
        m->IncorrectPin(2)
}
```

```
logout()
{
    m->logout()
}
BALANCE()
{
  m->balance()
}
DEPOSIT(int d)
{
    // store d into temp_d
        d->temp_d = d
        m->deposit()
}
WITHDRAW(int w){
      d->temp_w = w
    if(d->b > 0)
        m->Withdraw()
    else m->NoFunds()
}
```

suspend() { m->Suspend(); }

activate() { m->Activate(); }

close() { m->Close(); }

8. Model Driven Architecture of ACCOUNT COMPONENTS

OVERVIEW

The project has a ATM ACCOUNT COMPONENTS MODEL with two seperate designs. Each of these designs have specified functional requirements. The main goal is to seperate the platform specific portion of the system from the platform independent portion.

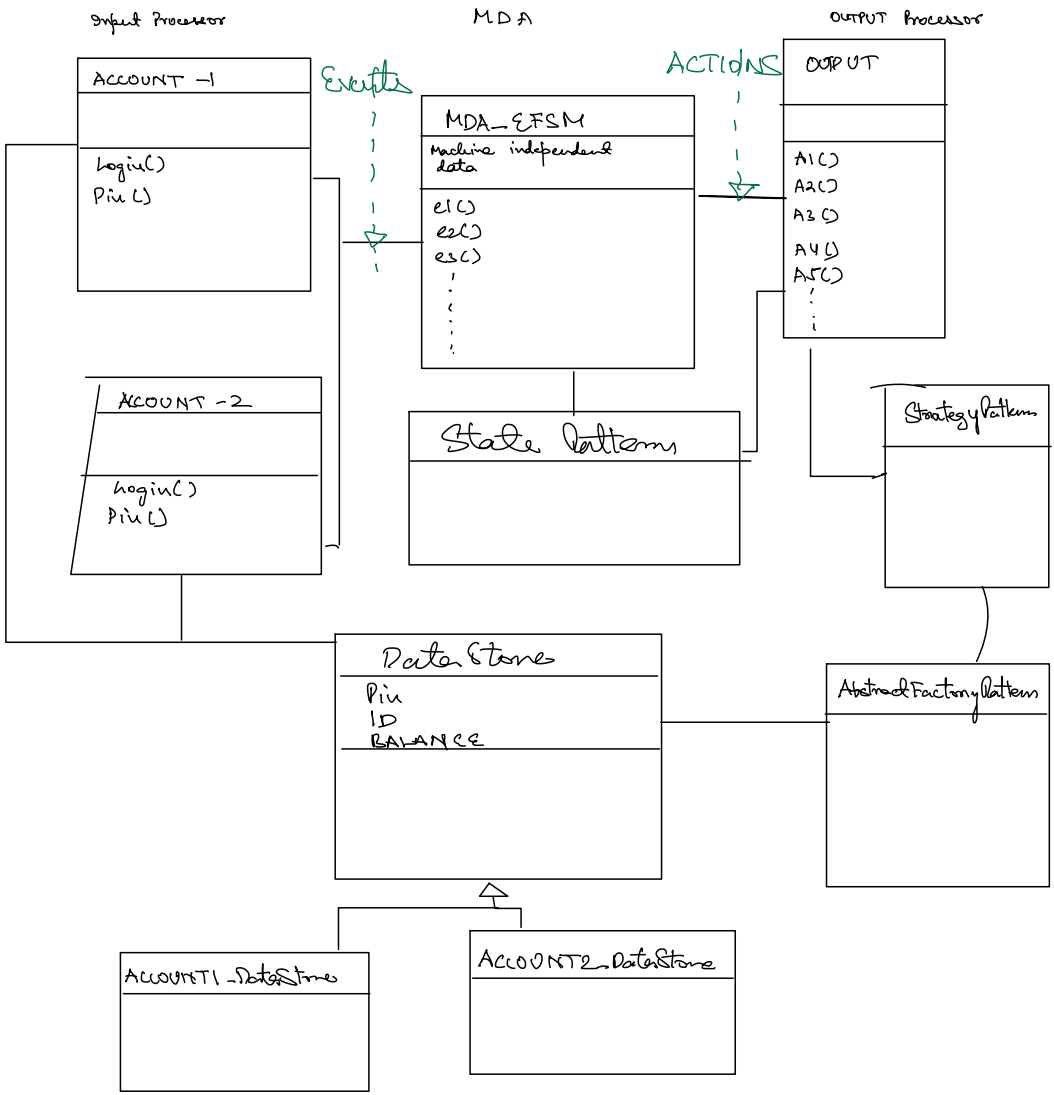
The project uses following three design strategies:

- State Design Pattern
- Abstract Factory Design Pattern
- Strategy Design Pattern

The ACCOUNT components are as follows:

- ACCOUNT-1
- ACCOUNT-2

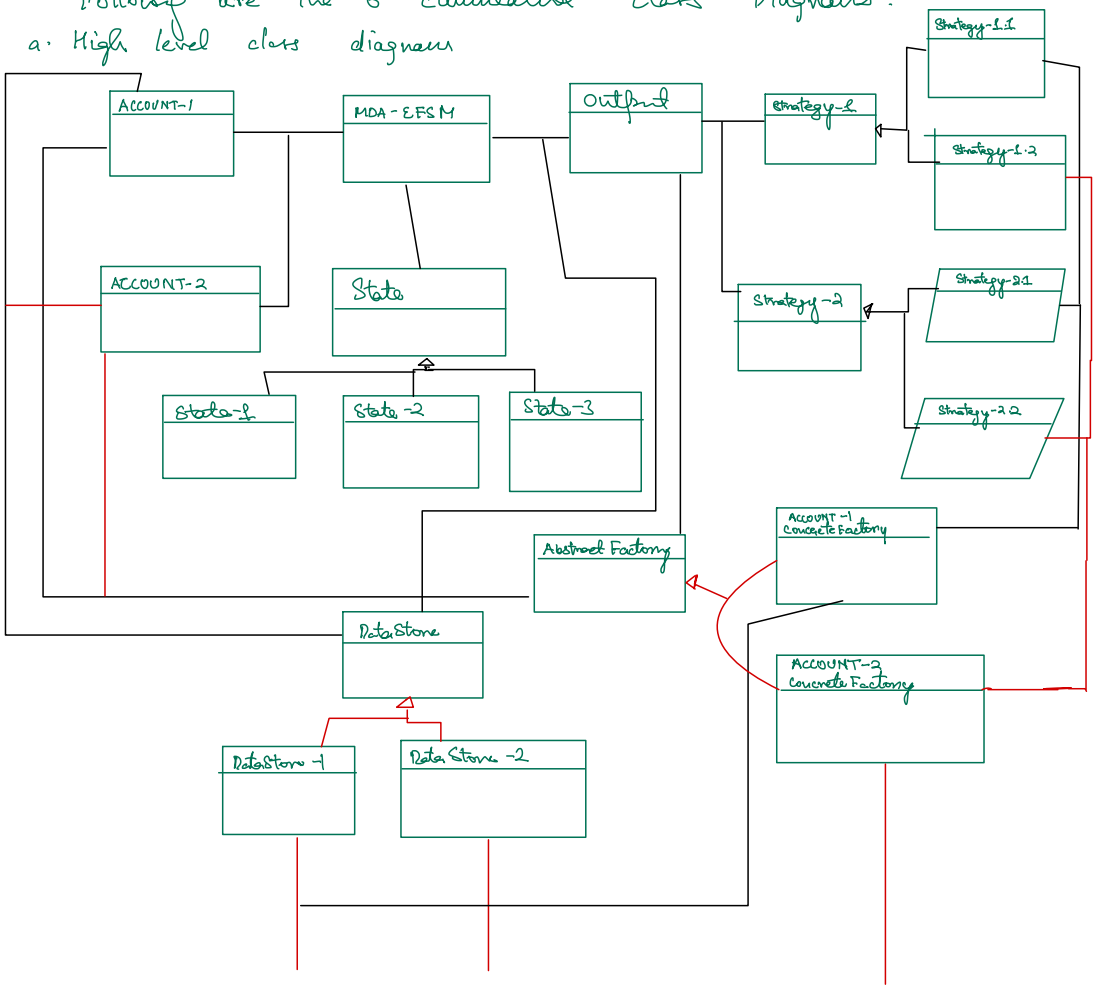
The project implements the OUTPUT Processor using Strategy pattern, and uses Abstract Factory Pattern to select and initialise a list of output actions for each ACCOUNT


Input Processor
MDA
OUTPUT Processor
ACCOUNT -1
Login()
Pin()
Events
MDA_EFSM
machine independent data
e1()
e2()
e3()
ACTIONS
OUTPUT
A1()
A2()
A3()
A4()
A5()
ACCOUNT -2
login()
Pin()
State Patterns
Strategy Pattern
Data Store
Pin
ID
BALANCE
Abstract Factory Pattern
ACCOUNT1_DataStore
ACCOUNT2_DataStore

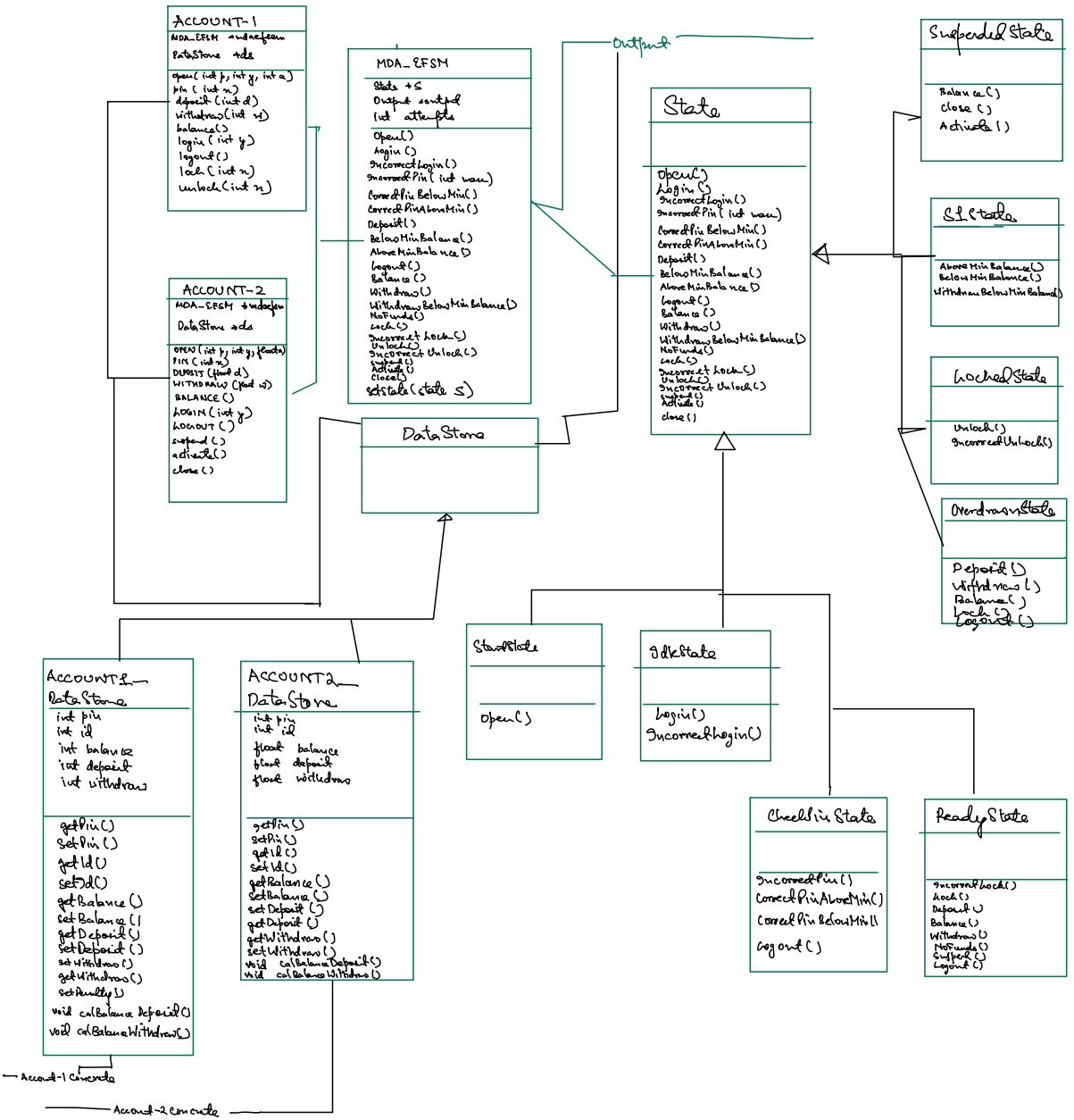
3. Class DIAGRAMS of the MDA of the ACCOUNT Components

Following are the 6 cumulative class Diagrams:

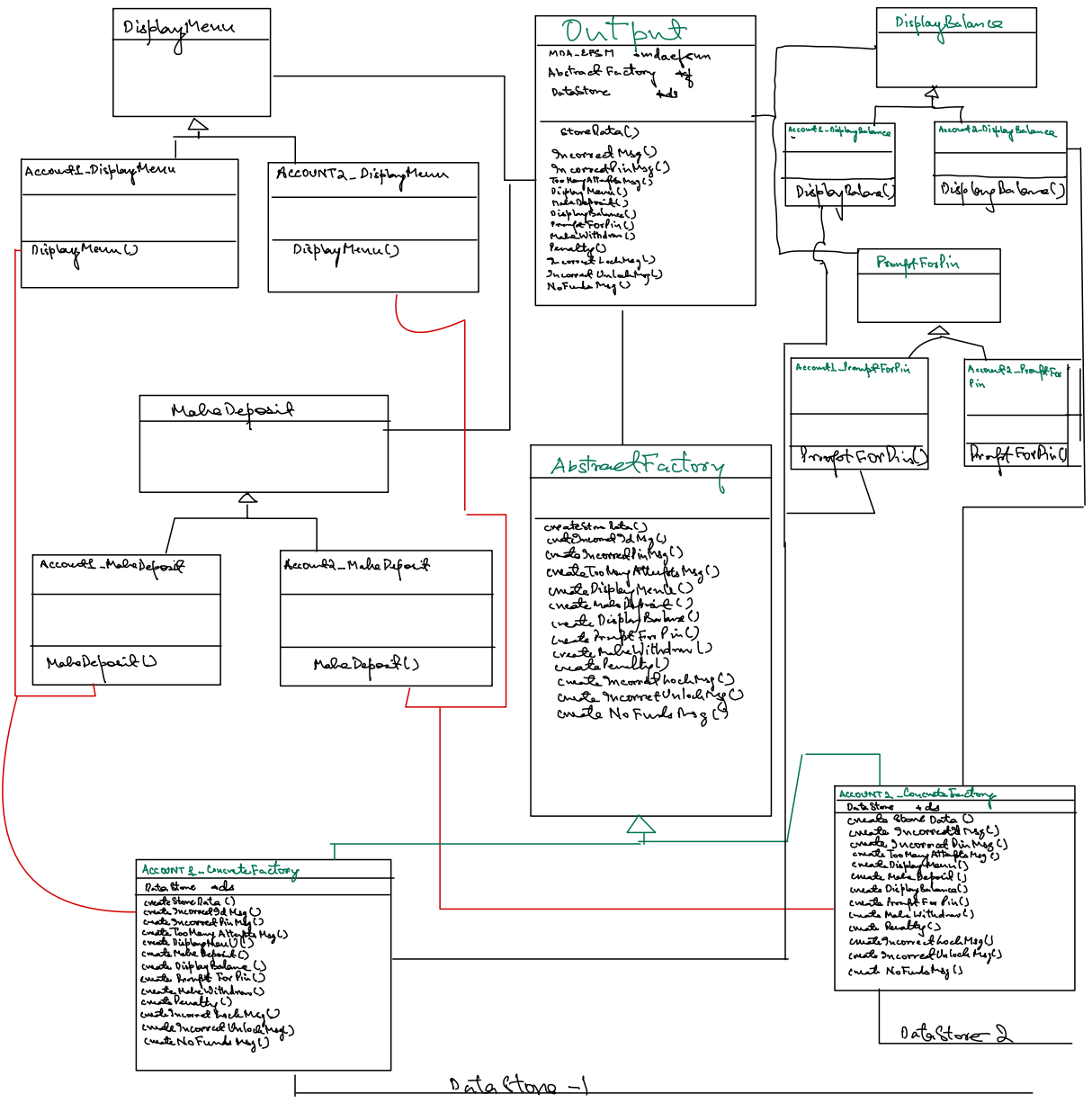
a. High level class diagram



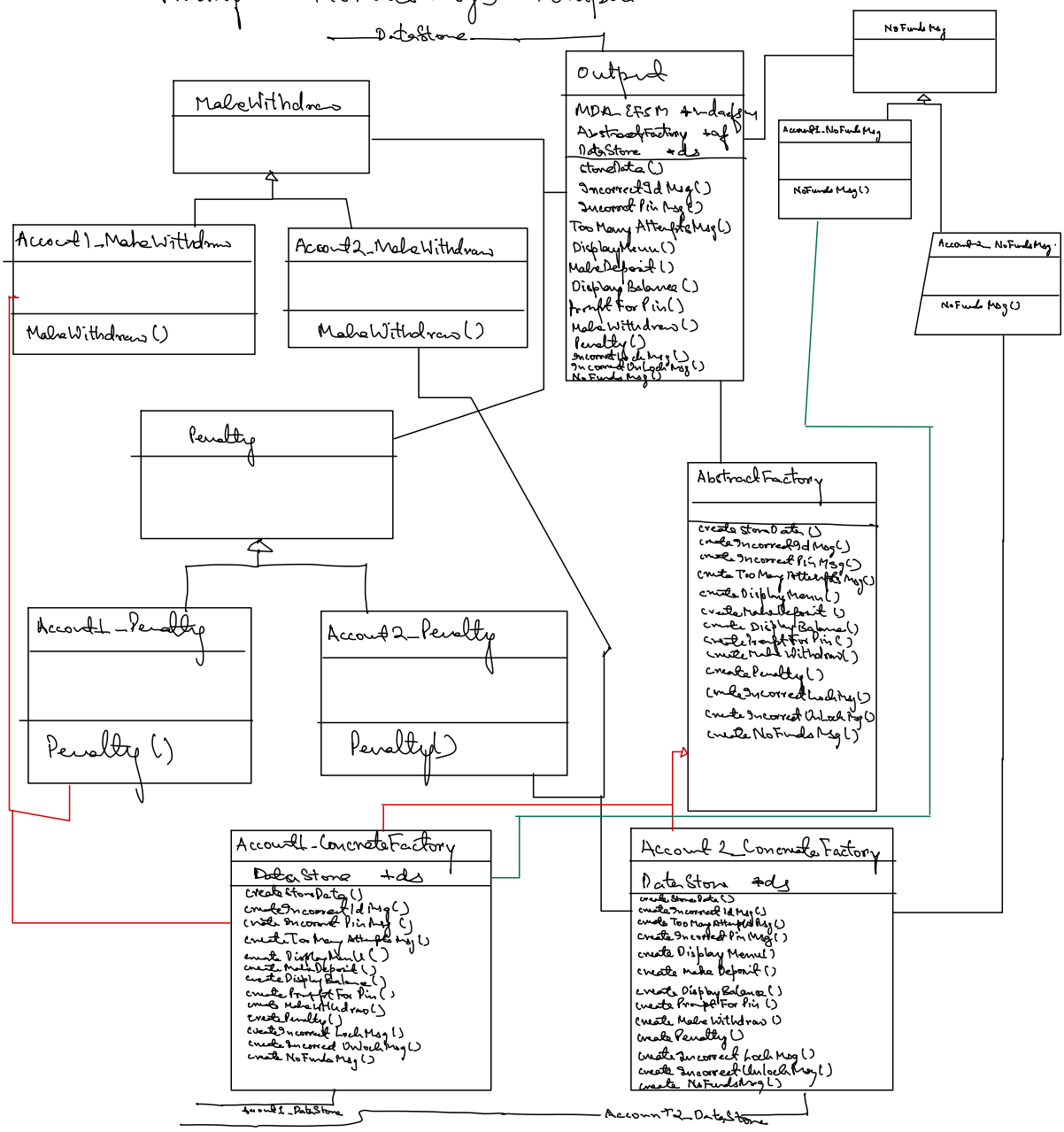
(b) Following class diagram models state pattern. It represents the associations of MDA_EFSM and State, DataStore, ACCOUNT-1 & ACCOUNT-2



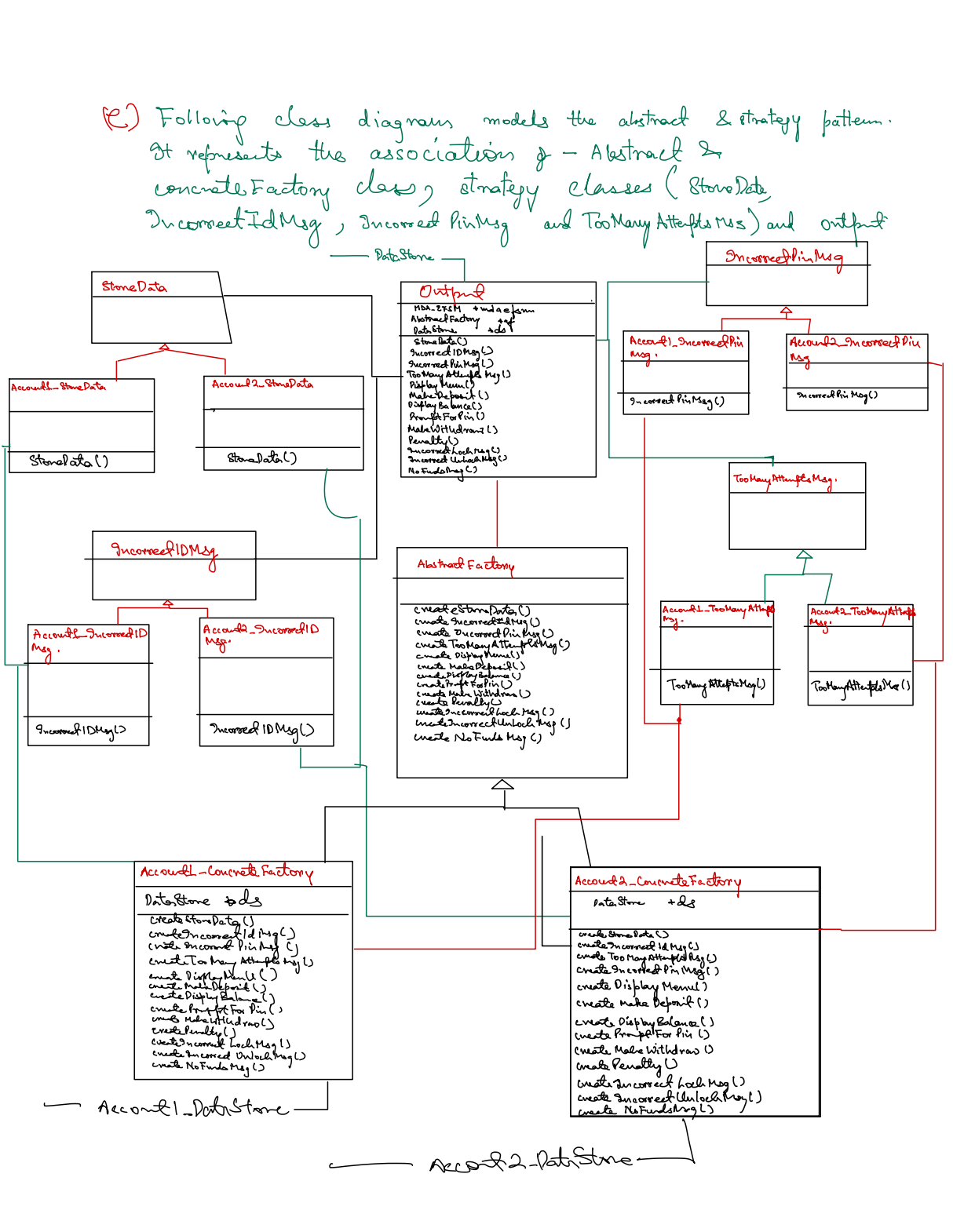
(c) Following class diagram models the abstract & strategy pattern. It represents the Association of Abstract & Concrete Factory classes, Strategy classes (DisplayMenu, MakeDeposit, DisplayBalance & PromptForPin) & output



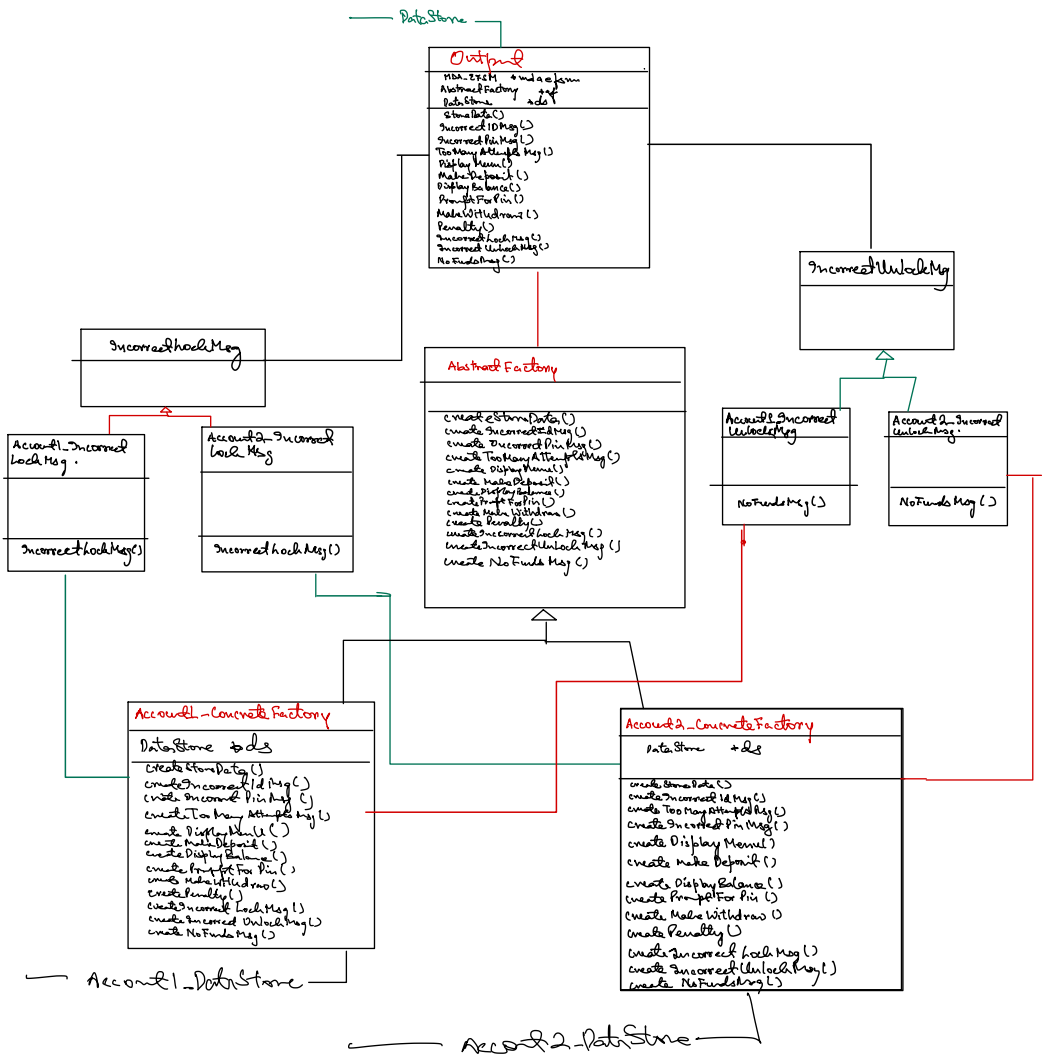
(d) Following class Diagram models the abstract & strategy pattern. It depicts the association of Abstract & Concrete Factory classes, Strategy classes (MakeWithdraw, Penalty & NoFundsMsg) & output



(c) Following class diagram models the abstract & strategy pattern. It represents the association of - Abstract & concrete Factory class, strategy classes (StoreData, IncorrectIdMsg, IncorrectPinMsg and TooManyAttemptsMsg) and output



(f) Following class diagram models the abstract & strategy patterns. It represents the association of - Abstract & concrete Factory classes, Strategy classes (IncorrectLockMsg & IncorrectUnlockMsg) & and Output



10. Purpose & Main Attributes of class

a). DataStone class

It is an abstract class for ACCOUNT1_DataStone & ACCOUNT2_DataStone classes.
They contain all the functions responsible for storing as well as retrieving temporary & permanent values from DataStone.

Attr: ["Temp Variables"]

int tmpBalance

int tmpPin

int tmpID

int tmpDeposit

int tmpWithdraw

["Permanent Variables"]

int balance

int pin

int id

int deposit

int withdraw

b) MDA_EFSM class

This class is responsible for change of states. It performs a variety of functions that result in a state change. After then, states perform meta actions. Then it captures platform independent behaviours & manages state machine for State pattern.

Attr:

State s // pointer to current state

public int attempts;

AbstractFactory af = null;

// initialise abstract factory pattern object

Output output = null; // initilize output object

c) State class

This is an interface class which implements different state classes responsible for performing various actions.

These state classes are:

1) StartState class – responsible for Open() action
2) IdleState class – responsible for login() & IncorrectLogin() actions.
3) CheckPinState class – responsible for IncorrectPin(), CorrectPinAboveMin(), CorrectPinBelowMin() & logout() actions
4) ReadyState class – Responsible for IncorrectPin(), Deposit(), Balance(), Withdraw(), NoFunds(), Suspend() & logout() actions
5) LockedState class – Responsible for Unlock() & IncorrectUnlock() actions
6) SuspendedState class – Responsible for Balance(), Close() & Activate() actions
7) SIState class – Responsible for AboveMinBalance(), BelowMinBalance() & WithdrawBelowMinBalance() actions
8) OverdrawnState class – Responsible for Deposit(), Balance(), Withdraw(), Lock() & logout() actions

d) AbstractFactory class

This is an abstract class for ACCOUNT1_ConcreteFactory & ACCOUNT2_ConcreteFactory classes. It is responsible for creating objects of the data stores & actions associated with ACCOUNT1 & ACCOUNT2

e) Output class

This class is responsible for handling all actions.

Attr:

DataStore *ds // pointer to data store object

AbstractFactory *af // pointer to abstract factor object

(f) ACCOUNT1 class

We initiate this class when user selects ACCOUNT1 from the menu listed by Driver.

Attr:

MDA_EFSM *mdaefsm // pointer to MDA_EFSM object
DataStone *ds // pointer to data Stone object

(e) ACCOUNT2 class

We initiate this class when user selects ACCOUNT2 from the menu listed by Driver.

Attr:

MDA_EFSM *mdaefsm // pointer to MDA_EFSM object
DataStone *ds // pointer to data Stone object

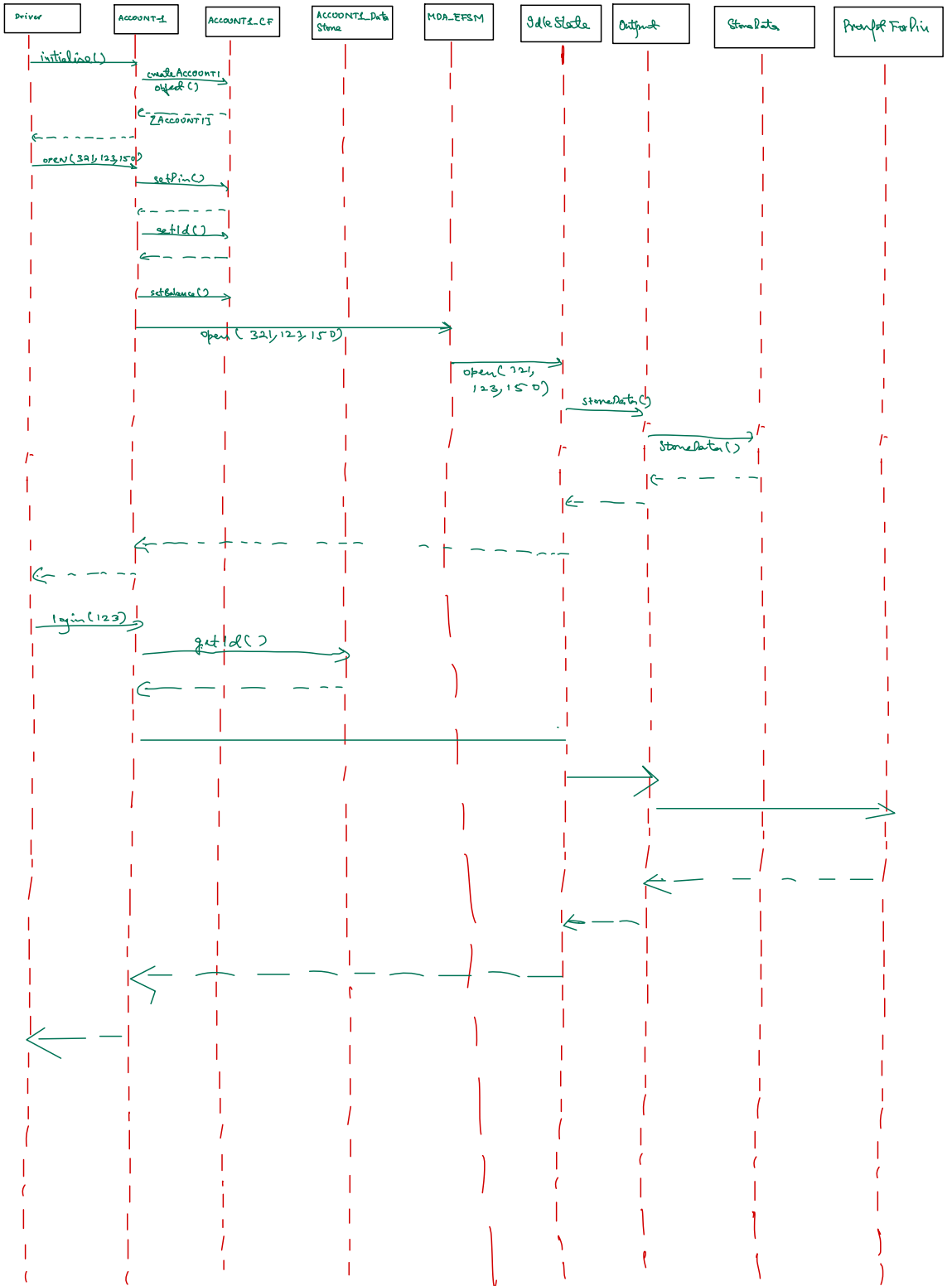
II DYNAMICS:

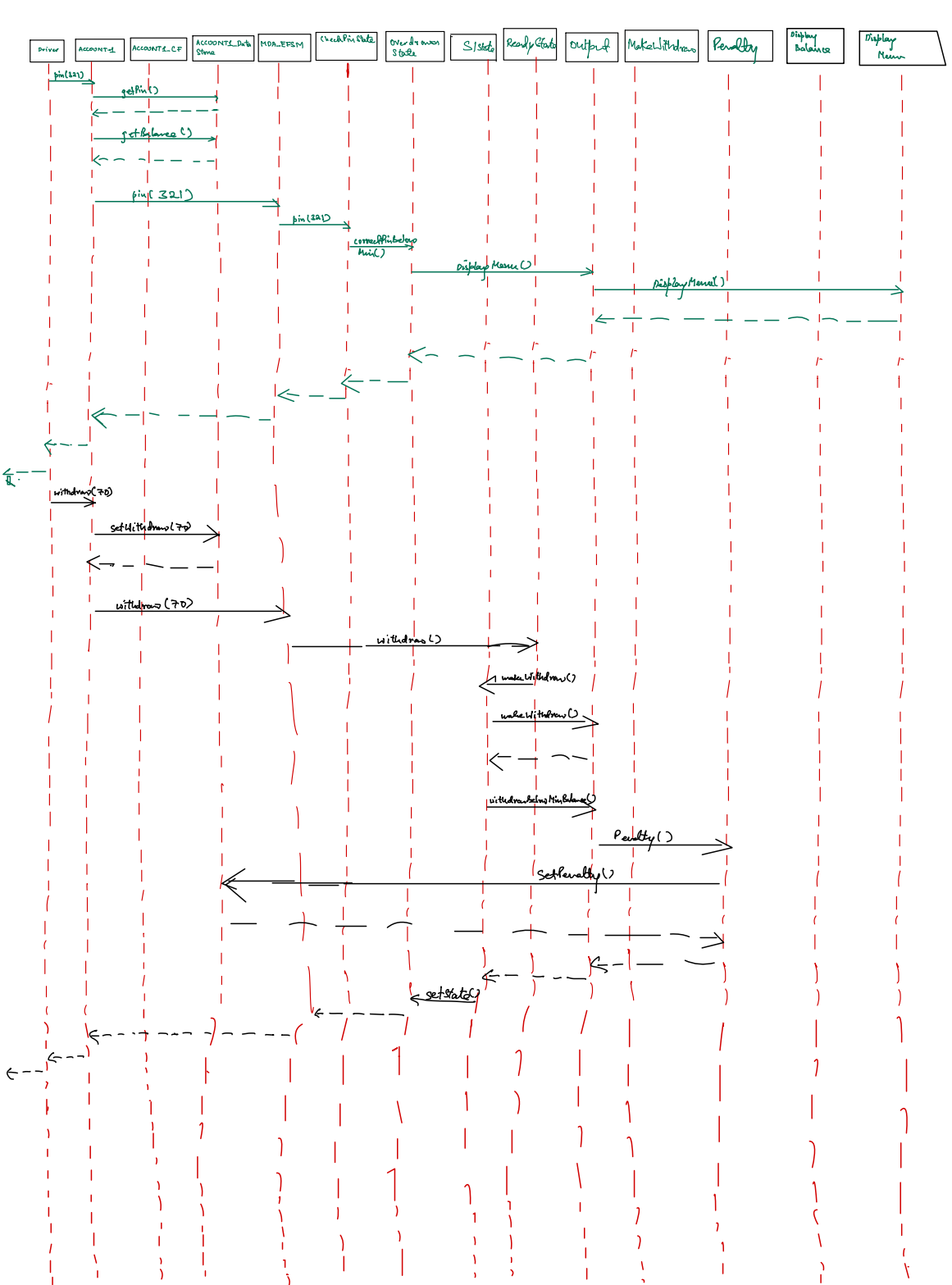
SEQUENCE DIAGRAM

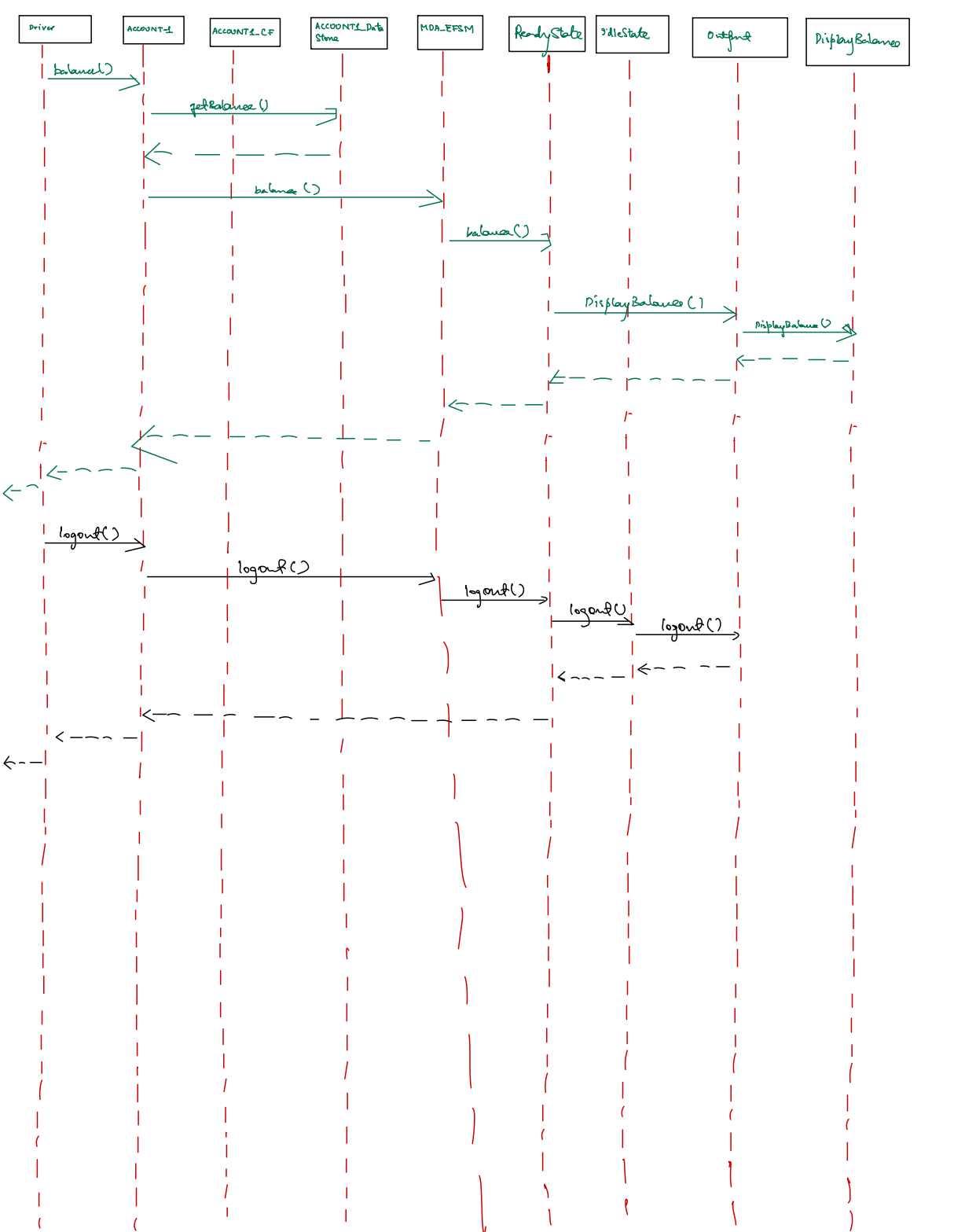
SEQUENCE DIAGRAM are created for the following scenario:

Open(321, 123, 150), login(123), pin(321), withdraw(70), balance(), logout()

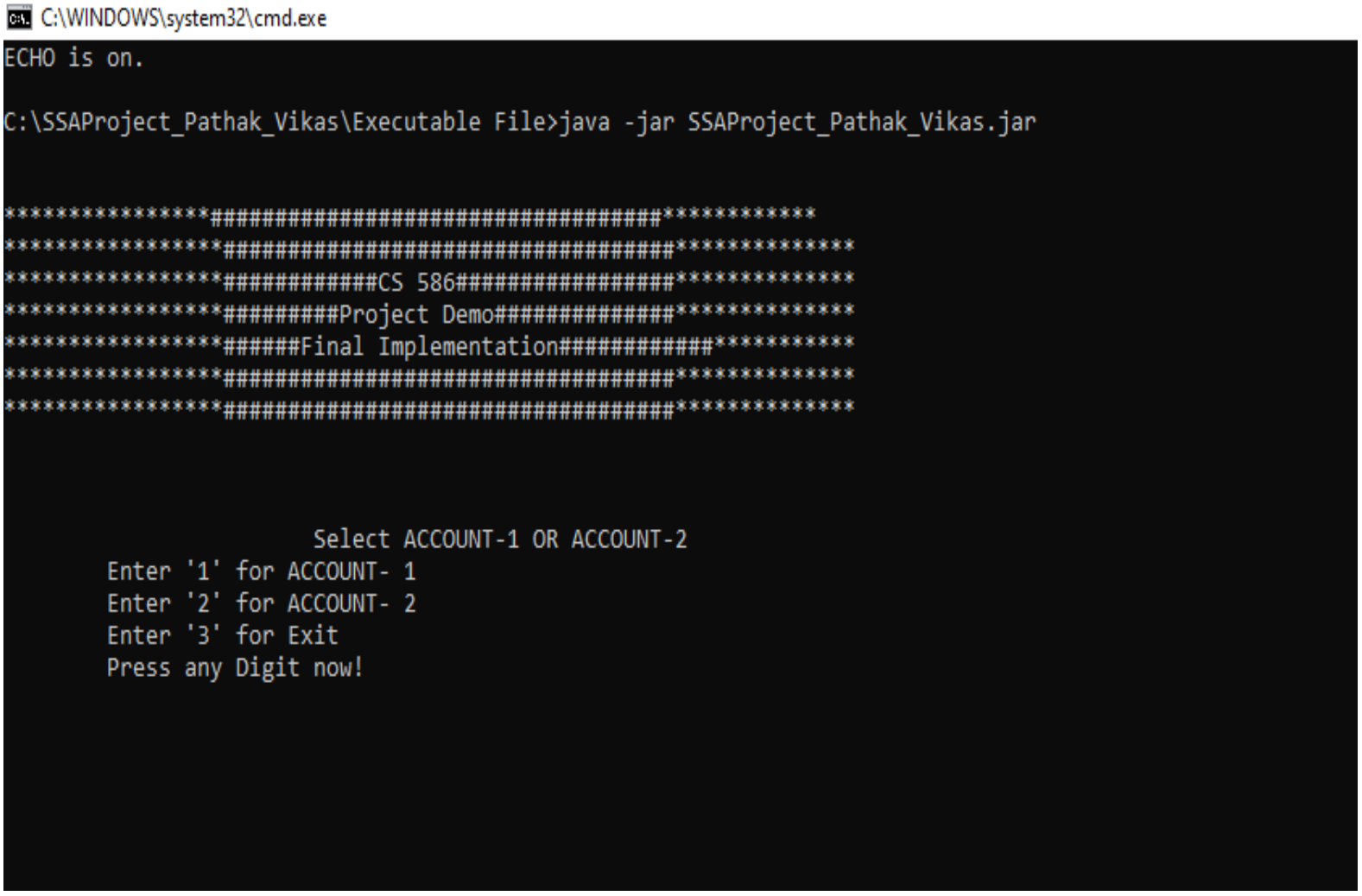
FOR ACCOUNT1


Driver
ACCOUNT_1
ACCOUNT1_CF
ACCOUNT1_Data Store
MDA_EFSM
Idle State
Output
StoreData
Prompt For Pin
initialise()
create ACCOUNT1 object()
[ACCOUNT1]
open(321,123,150)
setPin()
setId()
setBalance()
open(321,123,150)
open(321, 123, 150)
storeData()
storeData()
login(123)
getId()

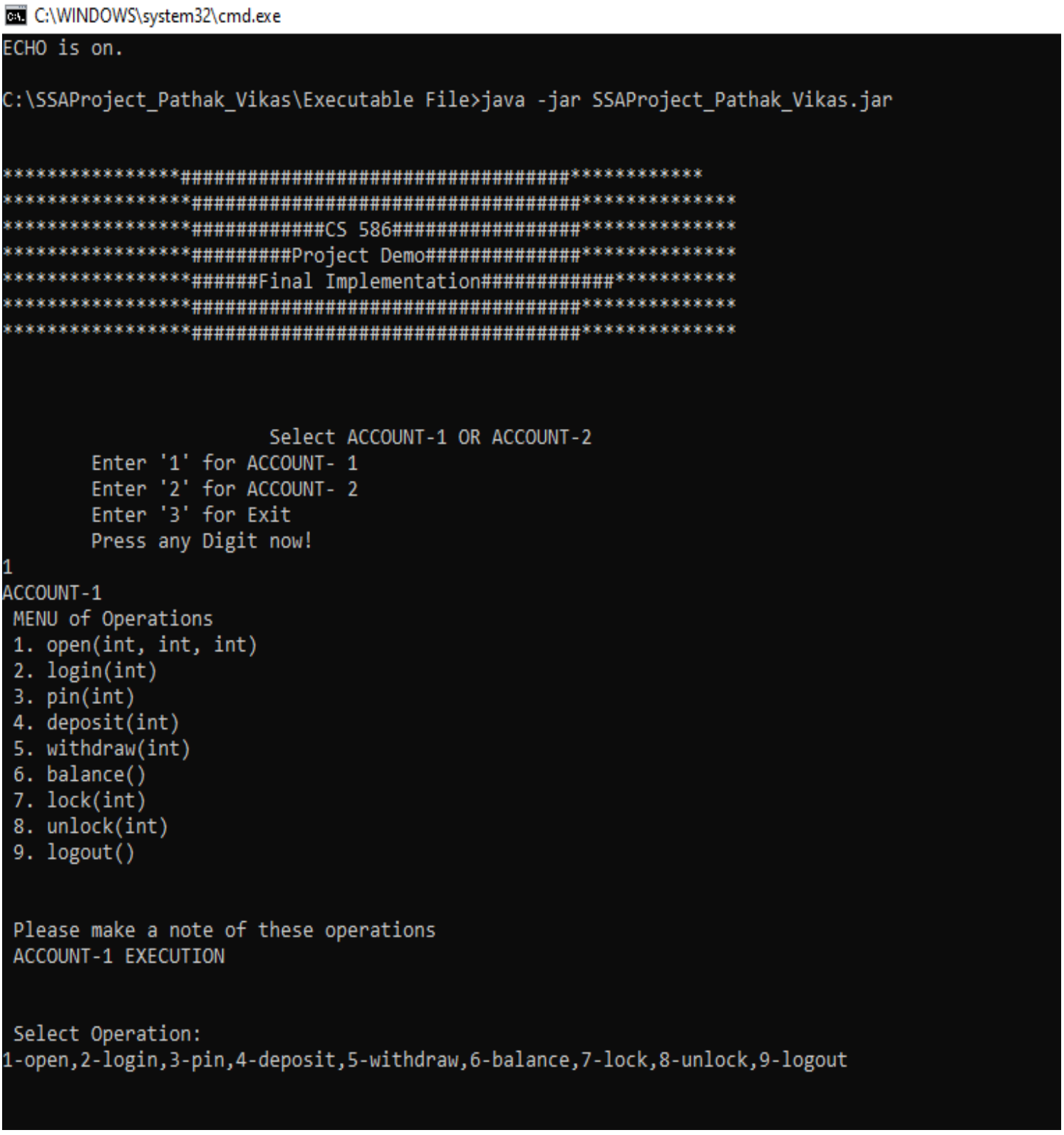

Driver
Account_I
Account_CF
Account_Data Store
MDA_EFSM
CheckPinState
Overdrawn State
S_State
ReadyState
Output
MakeWithdraw
Penalty
Display Balance
Display Menu
pin(321)
getPin()
getBalance()
pin(321)
pin(321)
correctPinBalance()
DisplayMenu()
DisplayMenu()
withdraw(70)
setWithdraw(70)
withdraw(70)
withdraw()
makeWithdraw()
makeWithdraw()
withdrawBelowMinBalance()
Penalty()
SetPenalty()
setState()


Driver
Account_1
Account_1_CF
Account_1_Data Store
MDA_EFSM
ReadyState
IdleState
Output
DisplayBalance
balance()
getBalance()
balance()
balance()
DisplayBalance()
DisplayBalance()
logout()
logout()
logout()
logout()
logout()

11. Project Demo ScreenShots



If the user selects ACCOUNT1, Menu displays as below:



If the user selects ACCOUNT2, Menu displays as below:

```
ECHO is on.

C:\SSAProject_Pathak_Vikas\Executable File>java -jar SSAProject_Pathak_Vikas.jar


**********###########################################**********
**********#####################################**********
**********############CS 586#################**********
**********#########Project Demo##############**********
**********######Final Implementation###########**********
**********#####################################**********
**********#####################################**********



                    Select ACCOUNT-1 OR ACCOUNT-2
       Enter '1' for ACCOUNT- 1
       Enter '2' for ACCOUNT- 2
       Enter '3' for Exit
       Press any Digit now!
2
 ACCOUNT-2
 MENU of Operations
 1. OPEN(int, int, float)
 2. LOGIN(int)
 3. PIN(int)
 4. DEPOSIT(float)
 5. WITHDRAW(float)
 6. BALANCE()
 7. suspend()
 8. activate()
 9. LOGOUT()
 10. close()


 Please make a note of these operations
 ACCOUNT-2 EXECUTION


 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

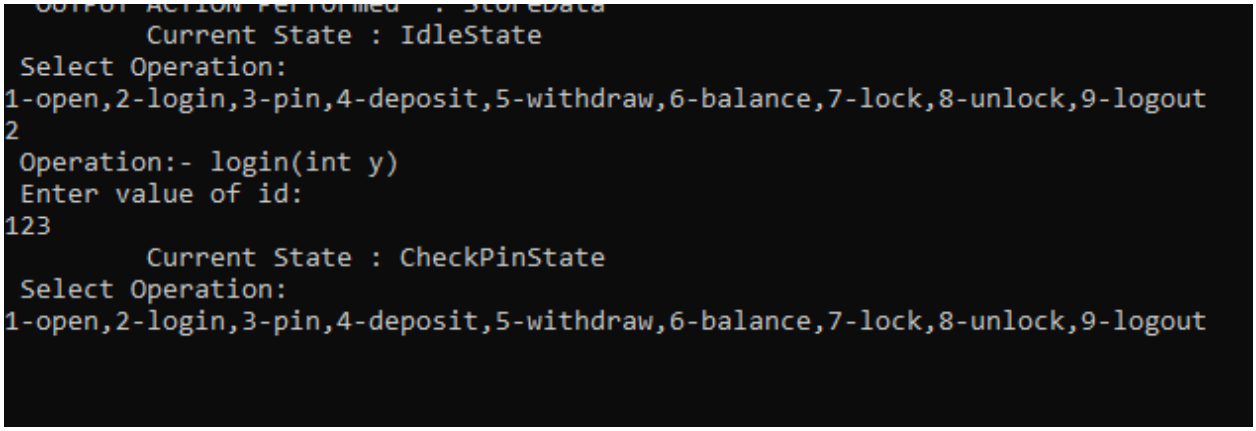
Sample Output for ACCOUNT1 as per the sequence diagram:

open(321,123,150), login(123), pin(321), withdraw(70), balance(), logout()

```
ECHO is on.

C:\SSAProject_Pathak_Vikas\Executable File>java -jar SSAProject_Pathak_Vikas.jar


**********##########################################**********
**********##########################################**********
**********############CS 586#######################**********
**********#########Project Demo#####################**********
**********######Final Implementation###############**********
**********##########################################**********
**********##########################################**********



                      Select ACCOUNT-1 OR ACCOUNT-2
        Enter '1' for ACCOUNT- 1
        Enter '2' for ACCOUNT- 2
        Enter '3' for Exit
        Press any Digit now!
1
ACCOUNT-1
 MENU of Operations
 1. open(int, int, int)
 2. login(int)
 3. pin(int)
 4. deposit(int)
 5. withdraw(int)
 6. balance()
 7. lock(int)
 8. unlock(int)
 9. logout()


 Please make a note of these operations
 ACCOUNT-1 EXECUTION


 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
1

 Operation:- open(int pin, int id, int balance)
 Enter value of the parameter pin:
321
 Enter value of the parameter id:
123
 Enter value of the parameter balance:
150

  OUTPUT ACTION Performed  : StoreData
         Current State : IdleState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

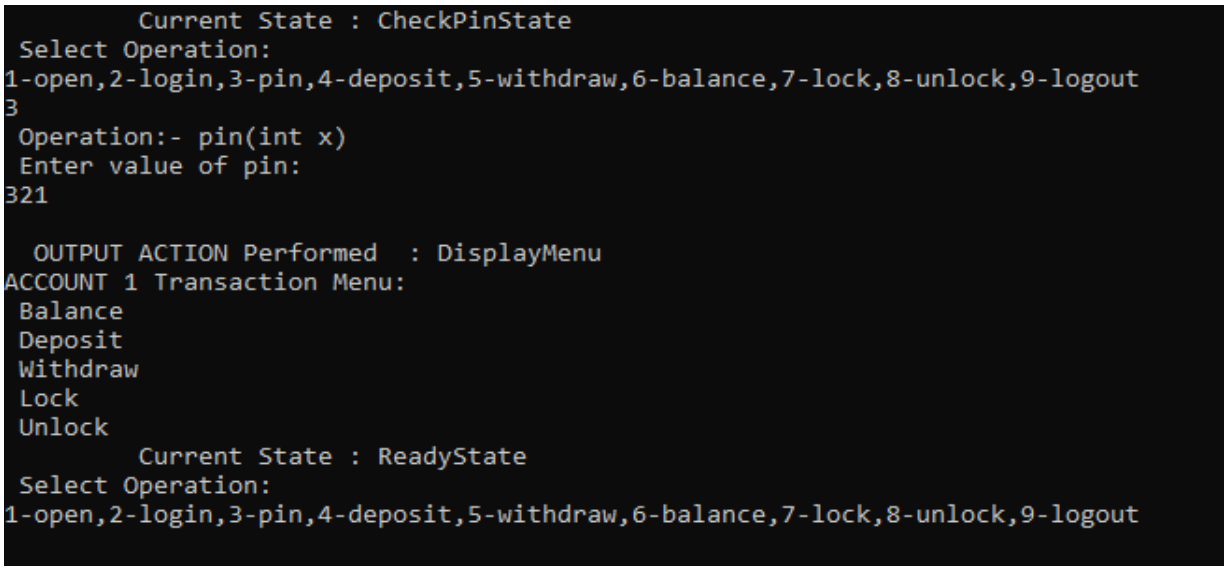
As you can see in above screenshot, account1 open is chosen and open(321,123,150) operation is performed on account of which first data is stored temporarily and then the state changes to IdleState. [Which is the desired state changes]

Now , we will perform login operation:



The state changes to CheckPinState

Now I will perform pin operation:



Now the system goes to ReadyState on successful login as per the screenshot and menu for account1 is displayed.

Now I will perform withdraw(70) operation:

```
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
5
 Operation:- withdraw(int w)
 Enter value of the parameter Withdraw:
70

 OUTPUT ACTION Performed : MakeWithdraw
        Current State : S1State

 OUTPUT ACTION Performed : Penalty
 Penality Applied : 15$
 Reason : Minimum required balance is $100.

        Current State : OverdrawnState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Since the balance requirement of ACCOUNT1 is minimum 100

When we withdraw 70 the balance goes to overdrawn value and then penalty of 15$ is added to balance so the final balance is 65$ (80 - 15) $

Now the system is officially in overdrawn state.

Now I will perform balance operation

```
        Current State : OverdrawnState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()

 OUTPUT ACTION Performed  : DisplayBalance

 The Current Balance in the Account is: 65

        Current State : OverdrawnState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

The state stays the same , operation - balance() , and the action performed is displaybalance

Now, I will perform logout() operation

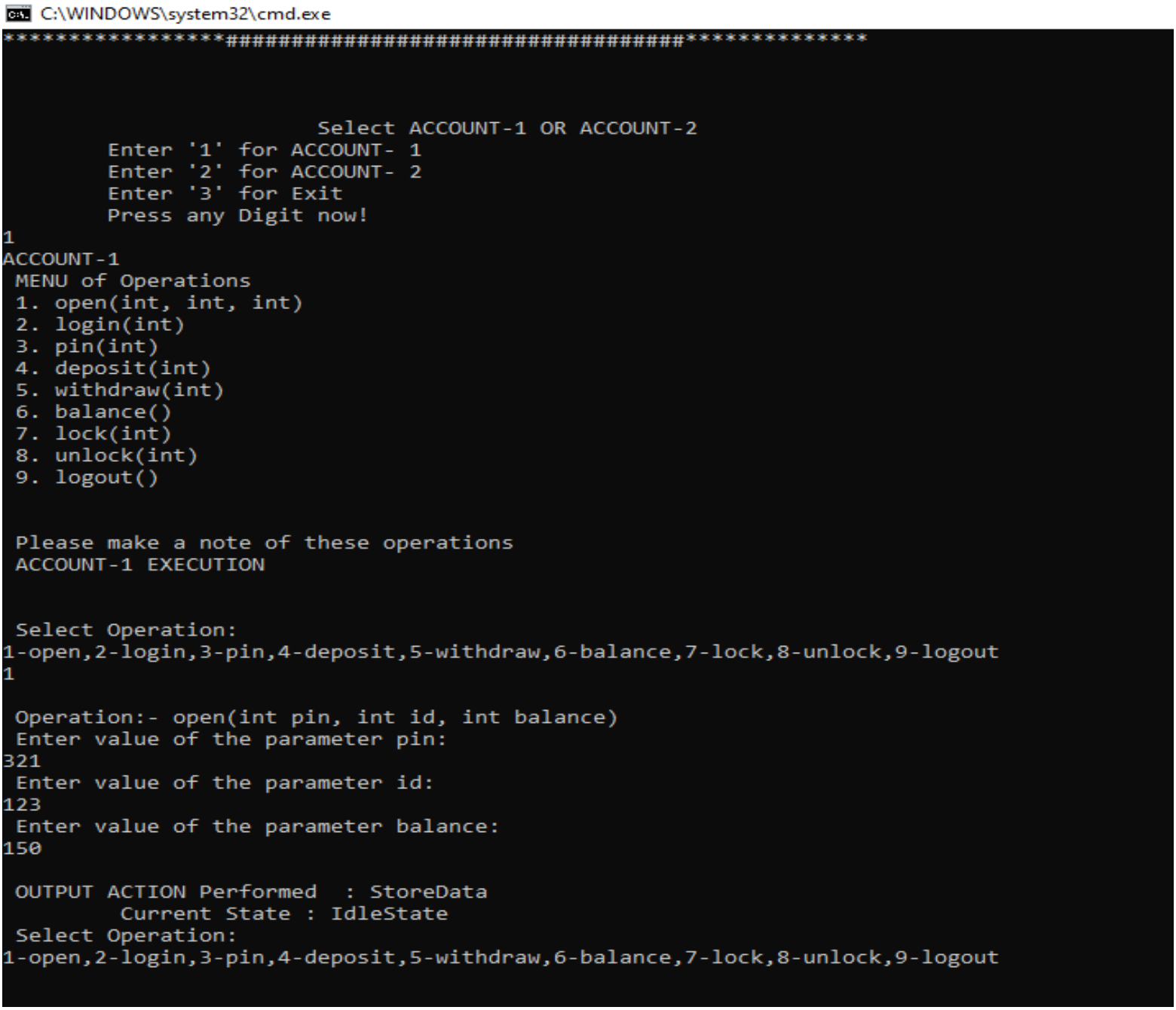
```
        Current State : OverdrawnState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
9
 Operation:- logout()
        Current State : IdleState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

After I perform logout the system will logout current account and goes to idlestate which is desired state change based on state diagram

Sample Output for ACCOUNT1 as per desired state diagram:

open(321,123,150), login(123), open(1,1,1000), pin(1), pin(321), balance(), deposit(150), balance(), lock(321), withdraw(70), balance(), unlock(321), balance(), withdraw(250), balance(), deposit(200), lock(321), deposit(50), unlock(6), unlock(321), balance(), logout()

Now performing the open operation like done last time



Now I will perform login(123) operation:

```
 OUTPUT ACTION Performed  : StoreData
         Current State : IdleState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
2
 Operation:- login(int y)
 Enter value of id:
123
         Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Now I will perform open(1,1,1000) again the system is in checkPinstate and will remain in check pinSate and pin id balance should not change ideally.

```
         Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
1

 Operation:- open(int pin, int id, int balance)
 Enter value of the parameter pin:
1
 Enter value of the parameter id:
1
 Enter value of the parameter balance:
1000
         Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Now let’s test the new pin(1) : It should show incorrect pin msg

```
        Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
3
 Operation:- pin(int x)
 Enter value of pin:
1

 OUTPUT ACTION Performed : IncorrectPinMsg
Incorrect Pin!!
        Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Now I will try with correct old pin (321) :

```
 OUTPUT ACTION Performed : IncorrectPinMsg
Incorrect Pin!!
        Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
3
 Operation:- pin(int x)
 Enter value of pin:
321

 OUTPUT ACTION Performed  : DisplayMenu
ACCOUNT 1 Transaction Menu:
 Balance
 Deposit
 Withdraw
 Lock
 Unlock
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Correct pin plus system goes to ready state

Now performing balance operation : balance()

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()

 OUTPUT ACTION Performed  : DisplayBalance

 The Current Balance in the Account is: 150

        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

As we can see that latest open operation couldn't not affect the balance value and balance is still 150 only not 1000. Proving the correct working of account1 operations.

Now performing deposit(150) operation :

```
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
4
 Operation:- deposit(int d)
 Enter value of the parameter Deposit:
150

 OUTPUT ACTION Performed : MakeDeposit
        Current State : ReadyState
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Deposit made : makeDeposit called

Now let's check balance again: balance()

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()

 OUTPUT ACTION Performed  : DisplayBalance

 The Current Balance in the Account is: 300

        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

The balance is 300 (150 + 150) now, which should be true.

Now performing lock() operation :

```
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
7
 Operation:- lock(int pin)
 Enter value of the parameter pin:
321
        Current State : LockedState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

System moved into lockedstate when provided correct pin

Now performing withdraw(70) operation:

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
5
 Operation:- withdraw(int w)
 Enter value of the parameter Withdraw:
70
        Current State : LockedState
        Current State : LockedState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Since its in locked state the state doesn't changes after withdraw operation From Locked to Locked.

Now performing balance operation to verify balance :

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()
        Current State : LockedState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Now performing unlock() operation:

```
        Current State : LockedState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
3
 Operation:- unlock(int pin)
 Enter value of the parameter pin:
321
        Current State : S1State
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

By providing correct pin system goes to first S1 state then to Ready state after checking AboveMinBalance()

Now checking balance :

```
        Current State : ReadyState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()

 OUTPUT ACTION Performed  : DisplayBalance

 The Current Balance in the Account is: 300
```

Now withdraw(250) operation is performed :

```
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
5
 Operation:- withdraw(int w)
 Enter value of the parameter Withdraw:
250

 OUTPUT ACTION Performed : MakeWithdraw
        Current State : S1State

 OUTPUT ACTION Performed : Penalty
 Penality Applied : 15$
 Reason : Minimum required balance is $100.

        Current State : OverdrawnState
```

Now checking balance():

```
        Current State : OverdrawnState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()

 OUTPUT ACTION Performed  : DisplayBalance

 The Current Balance in the Account is: 35

        Current State : OverdrawnState
```

Now deposit(200) operation is performed :

```
        Current State : OverdrawnState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
4
 Operation:- deposit(int d)
 Enter value of the parameter Deposit:
200

 OUTPUT ACTION Performed : MakeDeposit
        Current State : S1State
        Current State : ReadyState
```

The system goes from overdrawnState to readyState as expected: balanceAboveMin()

Now locking again the account : lock()

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
7
 Operation:- lock(int pin)
 Enter value of the parameter pin:
321
        Current State : LockedState
```

Now deposit(50) operation is performed:

```
        Current State : LockedState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
4
 Operation:- deposit(int d)
 Enter value of the parameter Deposit:
50
        Current State : LockedState
        Current State : LockedState
 Select Operation:
```

No DEPOSIT PERFORMED SYSTEM STILL IN LOCKEDSTATE!

Now unlocking with incorrect pin : unlock(6)

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
8
 Operation:- unlock(int pin)
 Enter value of the parameter pin:
6

 OUTPUT ACTION Performed  : IncorrectUnlockMsg
Incorrect Unlock!!
        Current State : LockedState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

INCORRECT UNLOCK MSG SHOWN

STILL IN LOCKED STATE

Now unlocking with correct pin : unlock(321)

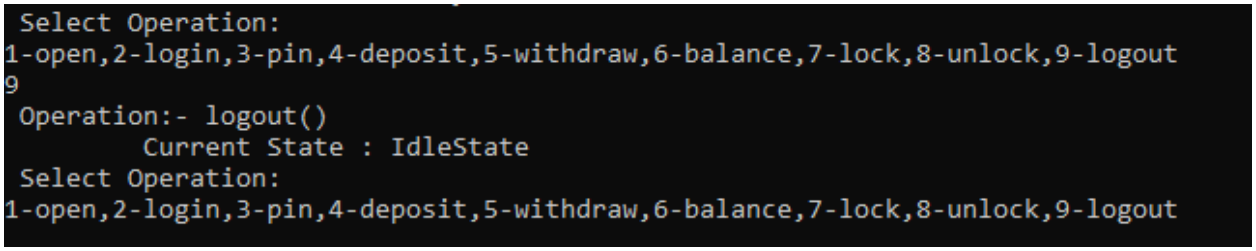
```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
8
 Operation:- unlock(int pin)
 Enter value of the parameter pin:
321
        Current State : S1State
        Current State : ReadyState
```

System goes to readyState as expected

Now checking balance() again:

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
6
 Operation:- balance()

 OUTPUT ACTION Performed  : DisplayBalance

 The Current Balance in the Account is: 235

        Current State : ReadyState
```

Now logout operation:



Sample Output for ACCOUNT1 as per desired state diagram:

open(321,123,150), login(1), [trying for incorrect login attempt in this sequence]

```
                        Select ACCOUNT-1 OR ACCOUNT-2
        Enter '1' for ACCOUNT- 1
        Enter '2' for ACCOUNT- 2
        Enter '3' for Exit
        Press any Digit now!
1
ACCOUNT-1
 MENU of Operations
 1. open(int, int, int)
 2. login(int)
 3. pin(int)
 4. deposit(int)
 5. withdraw(int)
 6. balance()
 7. lock(int)
 8. unlock(int)
 9. logout()


 Please make a note of these operations
 ACCOUNT-1 EXECUTION


 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
1

 Operation:- open(int pin, int id, int balance)
 Enter value of the parameter pin:
321
 Enter value of the parameter id:
123
 Enter value of the parameter balance:
150

 OUTPUT ACTION Performed  : StoreData
         Current State : IdleState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Now login(1) operation:

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
2
 Operation:- login(int y)
 Enter value of id:
1

 OUTPUT ACTION Performed  : IncorrectIdMsg
Incorrect Login Id!!
         Current State : IdleState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Since the ID was incorrect : incorrectIdMsg message shown!

Sample Output for ACCOUNT1 as per desired state diagram:

open(321,123,150), login(123), pin(2), pin(3), pin(3)

[trying for too many incorrect pin attempts in this sequence]

```
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
2
 Operation:- login(int y)
 Enter value of id:
123
         Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
3
 Operation:- pin(int x)
 Enter value of pin:
2

 OUTPUT ACTION Performed : IncorrectPinMsg
Incorrect Pin!!
         Current State : CheckPinState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
3
 Operation:- pin(int x)
 Enter value of pin:
3

 OUTPUT ACTION Performed : IncorrectPinMsg
Incorrect Pin!!

 OUTPUT ACTION Performed  : TooManyAttemptsMsg
Too Many Attempts!!
         Current State : IdleState
 Select Operation:
1-open,2-login,3-pin,4-deposit,5-withdraw,6-balance,7-lock,8-unlock,9-logout
```

Sample Output for ACCOUNT2 as per desired state diagram:

OPEN(321,123,150), BALANCE(), LOGIN(123), PIN(321), WITHDRAW(70), BALANCE(), suspend(), close()

OPEN(321,123,150) operation :

C:\WINDOWS\system32\cmd.exe

```
ECHO is on.

C:\SSAProject_Pathak_Vikas\Executable File>java -jar SSAProject_Pathak_Vikas.jar


**********#########################################**********
**********#########################################**********
**********############CS 586#####################**********
**********#########Project Demo###################**********
**********######Final Implementation#############**********
**********#########################################**********
**********#########################################**********



                      Select ACCOUNT-1 OR ACCOUNT-2
        Enter '1' for ACCOUNT- 1
        Enter '2' for ACCOUNT- 2
        Enter '3' for Exit
        Press any Digit now!
2
 ACCOUNT-2
 MENU of Operations
 1. OPEN(int, int, float)
 2. LOGIN(int)
 3. PIN(int)
 4. DEPOSIT(float)
 5. WITHDRAW(float)
 6. BALANCE()
 7. suspend()
 8. activate()
 9. LOGOUT()
 10. close()


 Please make a note of these operations
 ACCOUNT-2 EXECUTION


 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
1

 Operation:- OPEN(int pin, int id, float balance)
 Enter value of the parameter pin:
321
 Enter value of the parameter id:
123
 Enter value of the parameter balance:
150

 OUTPUT ACTION Performed  : StoreData
         Current State : IdleState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

BALANCE()

```
 OUTPUT ACTION Performed  : StoreData
        Current State : IdleState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
6
 Operation:- BALANCE()
        Current State : IdleState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

LOGIN(123)

```
        Current State : IdleState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
2
 Operation:- LOGIN(int x)
 Enter value of LOGIN:
123
        Current State : CheckPinState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

PIN(321)

```
        Current State : CheckPinState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
3
 Operation:- PIN(int x)
 Enter value of pin:
321

 OUTPUT ACTION Performed  : DisplayMenu
ACCOUNT 2 Transaction Menu:
 BALANCE
 DEPOSIT
 WITHDRAW
 LOCK
 UNLOCK
        Current State : ReadyState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

WITHDRAW(70)

```
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
5
 Operation:- WITHDRAW(float w)
 Enter value of the parameter Withdraw:
70

 OUTPUT ACTION Performed : MakeWithdraw
        Current State : S1State
        Current State : ReadyState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

BALANCE()

```
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
6
 Operation:- BALANCE()

 OUTPUT ACTION Performed  : DisplayBalance
Balance is: 80.0
        Current State : ReadyState
 Select Operation:
```

suspend()

```
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
7
 Operation:- suspend()
        Current State : SuspendedState
 Select Operation:
```

close()

```
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
10
 Operation:- close()
        Current State : StartState
 Select Operation:
1-OPEN,2-LOGIN,3-PIN,4-DEPOSIT,5-WITHDRAW,6-BALANCE,7-suspend,8-activate,9-LOGOUT,10-close
```

12. Patterns in SourceCode

[This is Driver class which enables us to access both ACCOUNT COMPONENTS execution]

Driver.java

Design Patterns starts here:

Source Code (or class files) which are responsible for the implementation of the three required design patterns are structured as follows.

- **STATE PATTERN**
  1. MDAEFSM.java
  2. Output.java
  3. State.java

     3.1 StartState.java

     3.2 S1State.java

     3.3 SuspendedState.java

     3.4 ReadyState.java

     3.5 OverdrawnState.java

     3.6 LockedState.java

     3.7 IdleState.java

     3.8 CheckPinState.java

- **STRATEGY PATTERN**
    1. DisplayMenu.java
        a. ACCOUNT1_DisplayMenu.java
        b. ACCOUNT2_DisplayMenu.java
    2. DisplayBalance.java
        a. ACCOUNT1_DisplayBalance.java
        b. ACCOUNT2_DisplayBalance.java
    3. IncorrectIdMsg.java
        a. ACCOUNT1_IncorrectIdMsg.java
        b. ACCOUNT2_IncorrectIdMsg.java
    4. IncorrectLockMsg.java
        a. ACCOUNT1_IncorrectLockMsg.java
        b. ACCOUNT2_IncorrectLockMsg.java
    5. IncorrectPinMsg.java
        a. ACCOUNT1_IncorrectPinMsg.java
        b. ACOOUNT2_IncorrectPinMsg.java
    6. IncorrectUnlockMsg.java
        a. ACCOUNT1_IncorrectUnlockMsg.java
        b. ACCOUNT2_IncorrectUnlockMsg.java
    7. MakeDeposit.java
        a. ACCOUNT1_MakeDeposit.java
        b. ACCOUNT2_MakeDeposit.java
    8. MakeWithdraw.java
        a. ACCOUNT1_MakeWithdraw.java
        b. ACCOUNT2_MakeWithdraw.java
    9. NoFundsMsg.java
        a. ACCOUNT1_NoFundsMsg.java

b. ACCOUNT2_NoFundsMsg.java

10. Penalty.java
    a. ACCOUNT1_Penalty.java
    b. ACCOUNT2_Penalty.java
11. PromptForPin.java
    a. ACCOUNT1_PromptForPin.java
    b. ACCOUNT2_PromptForPin.java
12. TooManyAttemptsMsg.java
    a. ACCOUNT1_TooManyAttemptsMsg.java
    b. ACCOUNT2_TooManyAttemptsMsg.java
13. StoreData.java
    a. ACCOUNT1_StoreData.java
    b. ACCOUNT2_StoreData.java

- **ABSTRACT FACTORY PATTERN**
    1. AbstractFactory.java
        a. ACCOUNT1_ConcreteFactory.java
        b. ACCOUNT2_ConcreteFactory.java
    2. ACCOUNT1.java
    3. ACCOUNT2.java
    4. DataStore.java
        a. ACCOUNT1_DataStore.java
        b. ACCOUNT2_DataStore.java