PROJECT REPORT
CS -586
SOFTWARE SYSTEMS ARCHITECTURES

Pavithra Kuttiyandi Chandrakasu
A20353191

1. Introduction

This project is based on the coursework of CS586 Software System Architecture. This project includes design and implementation of two ACCOUNT components using Model-Driven Architecture.

Goal of the Project

The main goal 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 de-coupled from data and implementation details.

Overview of Project

It includes ACCOUNT system which performs different operations and actions based on the user's interest. There are two ACCOUNT components: ACCOUNT-1 and ACCOUNT-2:

ACCOUNT-1 component:
operations:
open (string p, string y, float a) // open an account where p is a pin, y is a user's identification #, and a is a balance
pin (string x) // provides pin #
deposit (float d); // deposit amount d
withdraw (float w); // withdraw amount w
balance (); // display the current balance
login (string y) // login where y is a client's identification #
logout () // logout from the account
lock (string x) // locks an account where x is a pin
unlock (string x) // unlocks an account where x is a pin

ACCOUNT-2 component:
operations:
OPEN (int p, int y, int a) // open an account where p is a pin, y is a 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
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.

2. Model Driven Architecture of ACCOUNT components

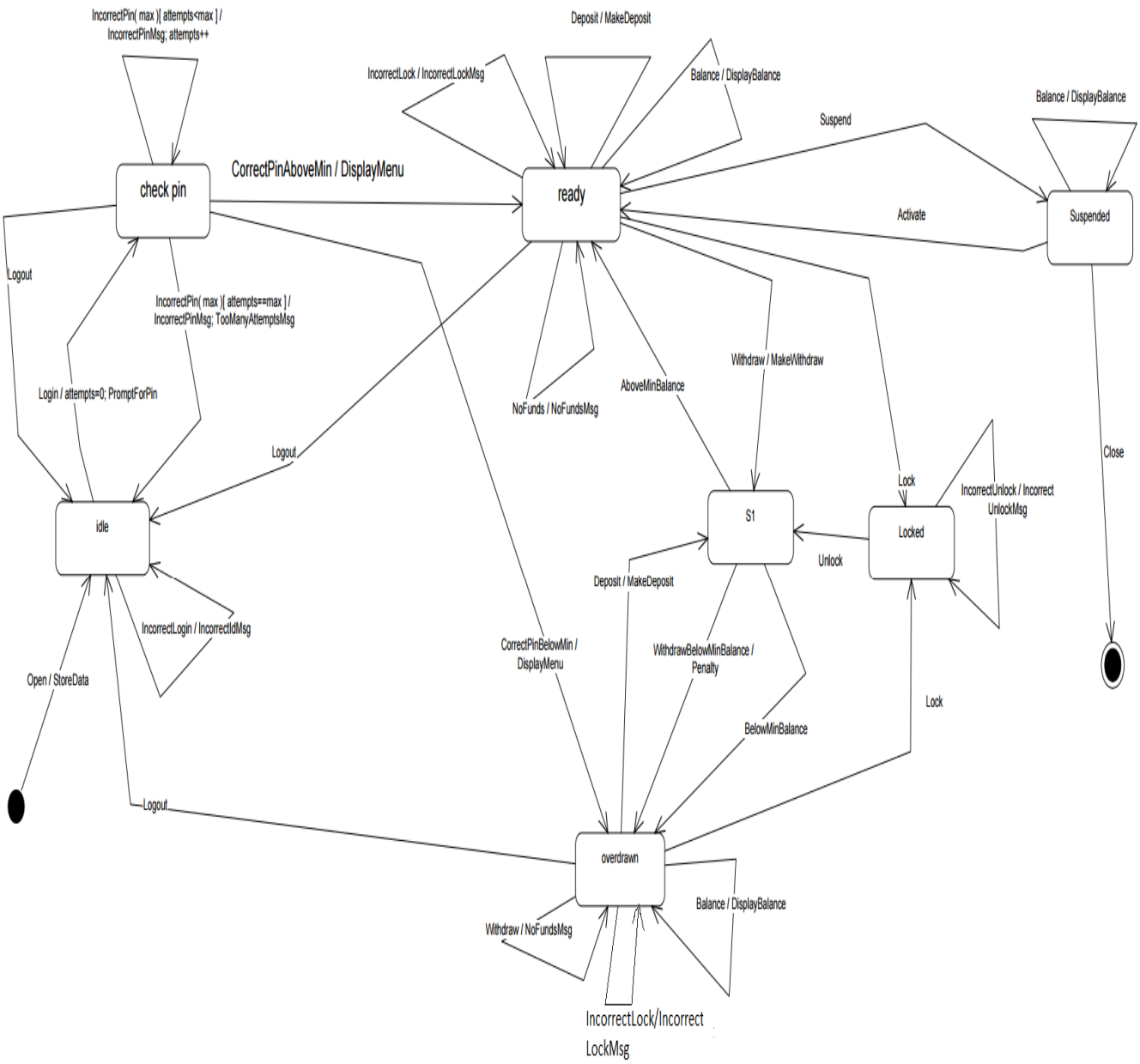
i) List of Events for the MDA-EFSM

Open ()
Login ()
IncorrectLogin ()
IncorectPin (int max)
CorrectPinBelowMin ()
CorrectPinAboveMin ()
Deposit ()
BelowMinBalance ()
AboveMinBalance ()
Logout ()
Balance ()
Withdraw ()
WithdrawBelowMinBalance ()
NoFunds ()
Lock ()
IncorrectLock ()
Unlock ()
IncorrectUnlock ()
Suspend ()
Activate ()
Close ()

ii) List of Actions for the MDA-EFSM

A1: StoreData () // stores pin from temporary data store to pin in data store
A2: IncorrectIdMsg () // displays incorrect ID message
A3: IncorrectPinMsg () // displays incorrect pin message
A4: TooManyAttemptsMsg () // display too many attempts message
A5: DisplayMenu () // display a menu with a list of transactions
A6: MakeDeposit () // makes deposit (increases balance by a value stored in temp. data store)
A7: DisplayBalance () // displays 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: IncorrectLock Msg () // displays incorrect lock message
A12: IncorrectUnlock Msg () // displays incorrect unlock message
A13: NoFundsMsg () // Displays no sufficient funds message

iii) A state diagram of the MDA-EFSM



iv) Pseudo-code of all operations of Input Processors of ACCOUNT-1 and ACCOUNT-2

Variables of the Input Processor (ACCOUNT-1)
m: is a pointer to the MDA-EFSM object
ds: is a pointer to the Data Store object
which contains the following data items:
• balance: contains the current balance

• pin: contains the correct pin #
• uid: contains the correct user ID
• temp_p, temp_y, temp_a, temp_d, temp_w is used to store values of parameters

Operations of the Input Processor (ACCOUNT-1)

open (string p, string y, float a)

```
// store p, y and a in temp data store
ds->temp_p=p;
ds->temp_y=y;
ds->temp_a=a;
m->Open ();
```

pin (string x)

```
if (x==ds->pin)

    if (d->balance > 500)
        m->CorrectPinAboveMin ();
    else
        m->CorrectPinBelowMin ();

else m->IncorrectPin (3)
```

deposit (float d)

```
    ds->temp_d=d;
    m->Deposit ();
    if (ds->balance>500)
        m->AboveMinBalance ();
    else m->BelowMinBalance ();
```

withdraw (float w)

```
    ds->temp_w=w;
    m->withdraw ();
    if ((ds->balance) >500))
        m->AboveMinBalance ();
    else m->WithdrawBelowMinBalance ();
```

balance ()

```
    m->balance ();
```

login (string y)

```
    if (y==ds->uid)
        m->Login ();
    else m->IncorrectLogin ();
```

logout ()

```
m->logout ();
```

lock (string x)

```
    if (ds->pin==x)
        m->Lock ();
    else m->IncorrectLock ();
```

unlock (string x)

```
    if (x==ds->pin)

        m->Unlock ();
        if (ds->balance > 500)
            m->AboveMinBalance ();
        else m->BelowMinBalance();

else m->IncorrectUnlock();
```

Variables of the Input Processor (ACCOUNT-2)

m: is a pointer to the MDA-EFSM object
ds: is a pointer to the Data Store object

which contains the following data items:
• balance: contains the current balance
• pin: contains the correct pin #
• uid: contains the correct user ID
• temp_p, temp_y, temp_a, temp_d, temp_w are used to store values of parameters

Operations of the Input Processor (ACCOUNT-2)

OPEN (int p, int y, int a)

```
// store p, y and a in temp data store
ds->temp_p=p;
ds->temp_y=y;
ds->temp_a=a;
m->Open ();
```

PIN (int x)

```
if (x==ds->pin)
    m->CorrectPinAboveMin ();
else m->IncorrectPin (2)
```

DEPOSIT (int d)

```
ds->temp_d=d;
m->Deposit ();
```

WITHDRAW (int w)

```
ds->temp_w=w;
if (ds->balance>0)
    m->Withdraw ();
else m->NoFunds ();
```

BALANCE ()

```
m->Balance ();
```

LOGIN (int y)

```
if (y==ds->uid)
    m->Login ();
else
    m->IncorrectLogin ();
```

LOGOUT ()

```
m->Logout ();
```

suspend ()

```
m->Suspend ();
```

activate ()

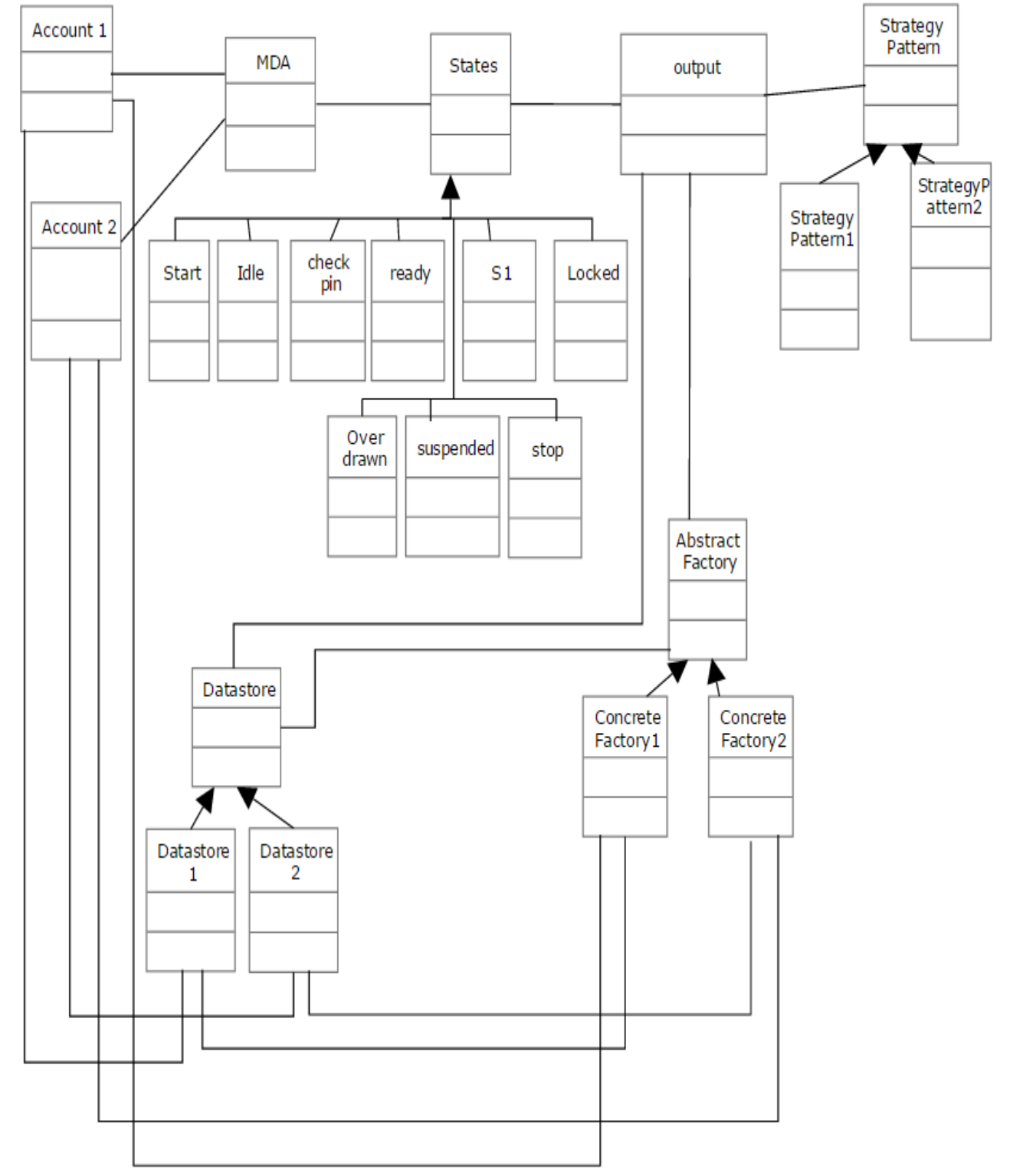
```
m->Activate ();
```

close ()

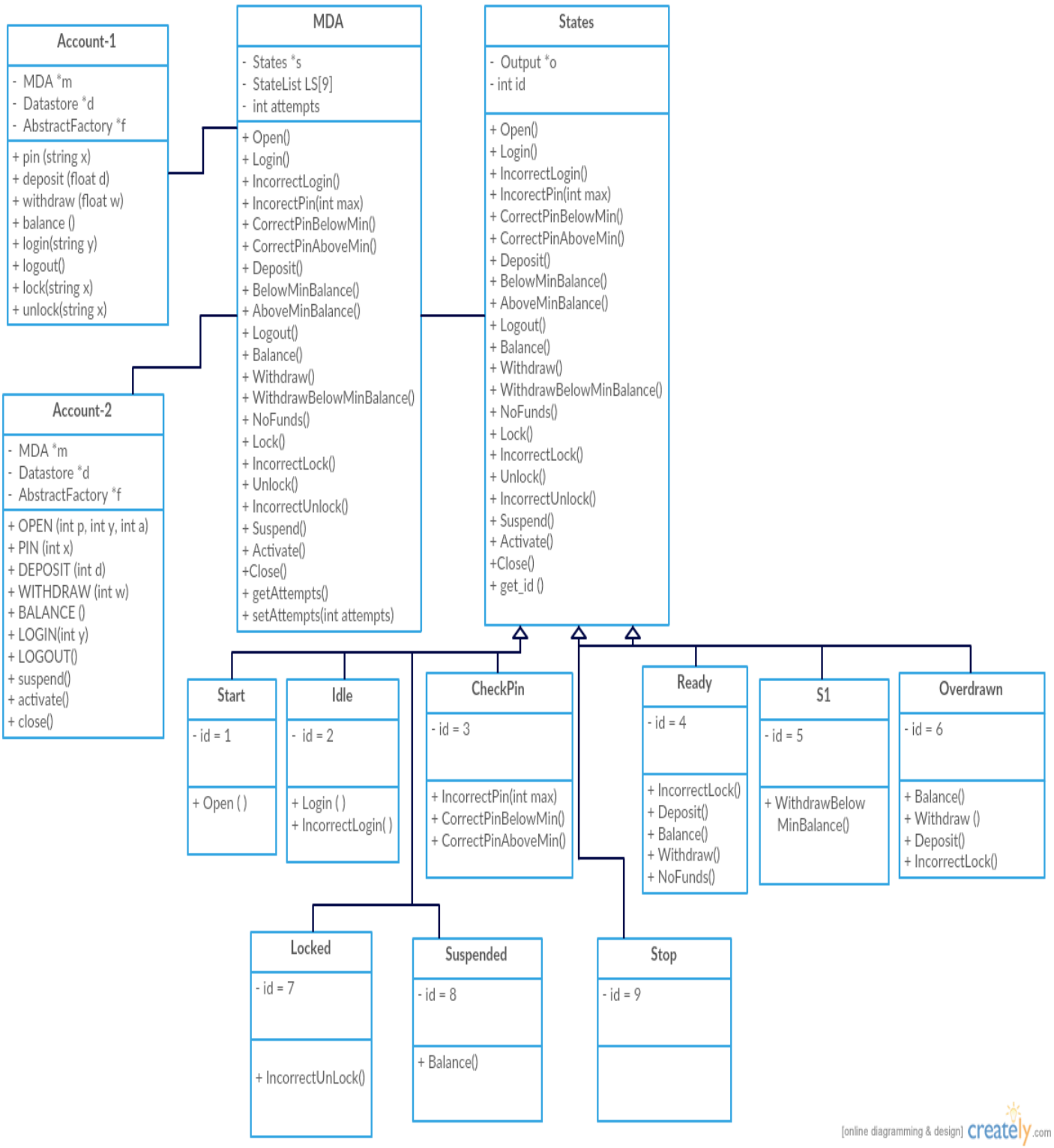
```
m->Close ();
```

3. Class diagram(s) of the MDA of the ACCOUNT components

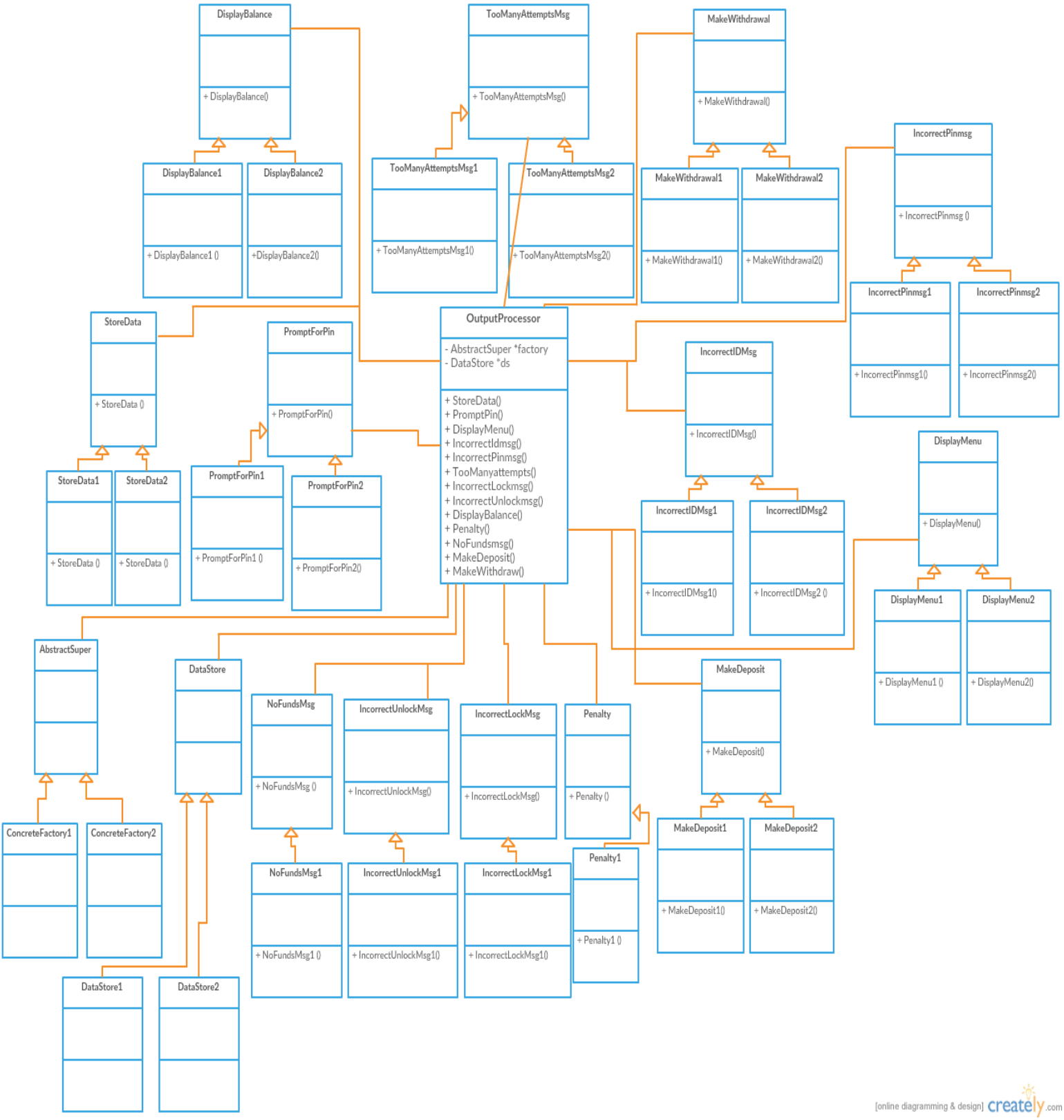
High Level Class Diagram



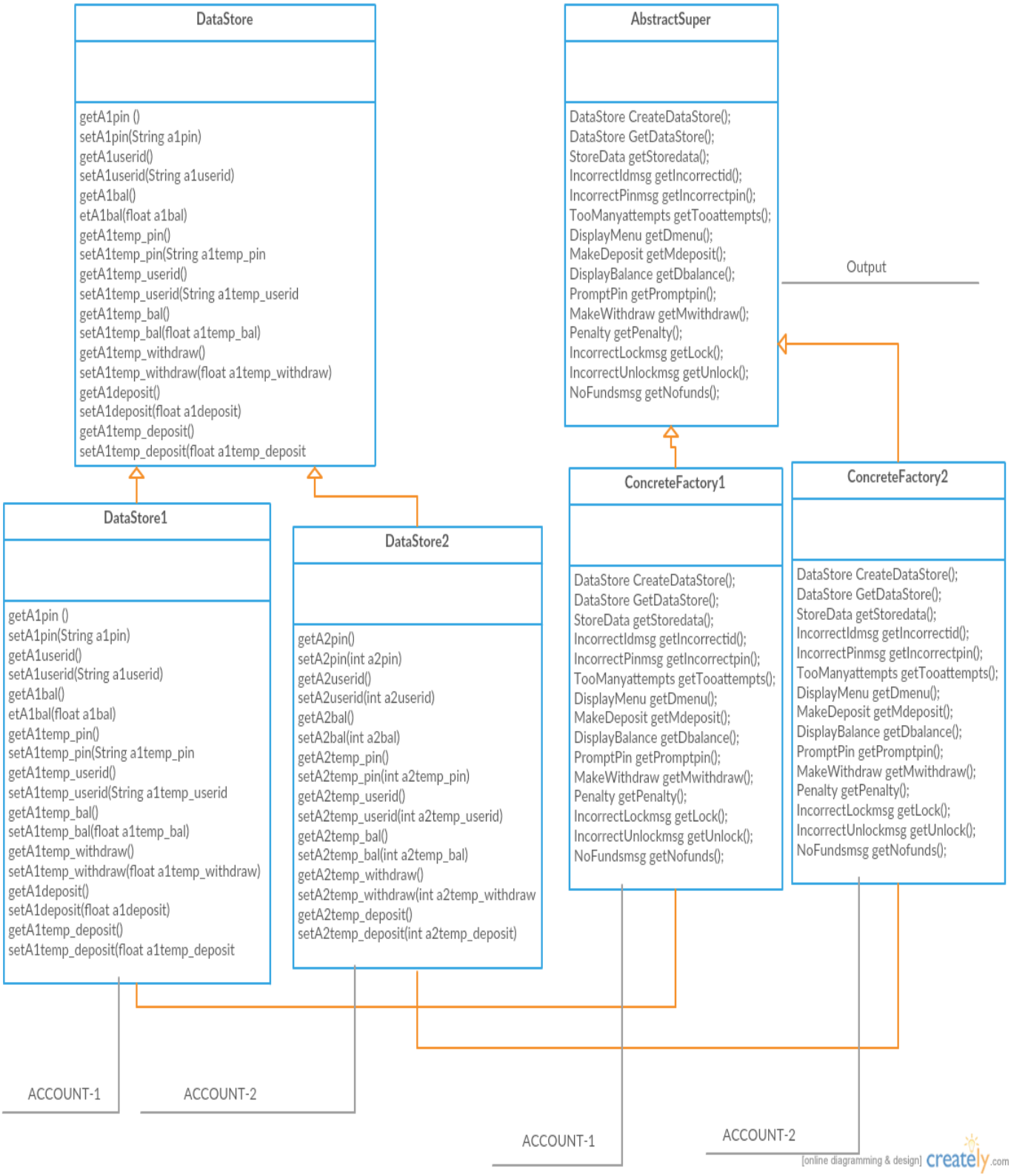
State Design Pattern:



Strategy Design Pattern



Abstract Factory Pattern



4. Operations, Attributes, Responsibilities

4.1 DataStore Class:

Purpose:
Store data needed by ACCOUNT component for storing temporary data like user inputs as well as permanent data needed by Strategy Pattern

Responsibility of Operations:

DataStore: all below operations are abstract operations.

- String getA1pin ()
- void setA1pin(String a1pin)
- String getA1userid()
- void setA1userid(String a1userid)
- float getA1bal()
- void setA1bal(float a1bal)
- String getA1temp_pin()
- void setA1temp_pin(String a1temp_pin
- String getA1temp_userid()
- void setA1temp_userid(String a1temp_userid
- float getA1temp_bal()
- void setA1temp_bal(float a1temp_bal)
- float getA1temp_withdraw()
- void setA1temp_withdraw(float a1temp_withdraw)
- float getA1deposit()
- void setA1deposit(float a1deposit)
- float getA1temp_deposit()
- void setA1temp_deposit(float a1temp_deposit
- int getA2pin()
- void setA2pin(int a2pin)
- int getA2userid()
- void setA2userid(int a2userid)
- int getA2bal()
- void setA2bal(int a2bal)
- int getA2temp_pin()
- void setA2temp_pin(int a2temp_pin)
- int getA2temp_userid()
- void setA2temp_userid(int a2temp_userid)
- int getA2temp_bal()
- void setA2temp_bal(int a2temp_bal)
- int getA2temp_withdraw()
- void setA2temp_withdraw(int a2temp_withdraw
- int getA2temp_deposit()
- void setA2temp_deposit(int a2temp_deposit)

DataStore1:

- String getA1pin() // get the permanent pin
- void setA1pin(String a1pin) // set the permanent pin
- String getA1userid() // get the permanent user identification number
- void setA1userid(String a1userid) // set the permanent user identification number
- float getA1bal() // get the permanent balance
- void setA1bal(float a1bal) // set the permanent balance
- String getA1temp_pin() // get the permanent pin
- void setA1temp_pin(String a1temp_pin) // set the temporary pin
- String getA1temp_userid() // get the temporary pin
- void setA1temp_userid(String a1temp_userid) // set the temporary user identification number
- float getA1temp_bal() // get the temporary balance
- void setA1temp_bal(float a1temp_bal) // set the temporary balance
- float getA1temp_withdraw() // get the temporary withdraw
- void setA1temp_withdraw(float a1temp_withdraw) // set the temporary withdraw
- float getA1temp_deposit() // get the temporary deposit
- void setA1temp_deposit(float a1temp_deposit) // set the temporary deposit

DataStore2:

- int getA2pin() // get the permanent pin
- void setA2pin(int a1pin) // set the permanent pin
- int getA2userid() // get the permanent user identification number
- void setA2userid(int a1userid) // set the permanent user identification number
- int getA2bal() // get the permanent balance
- void setA2bal(int a1bal) // set the permanent balance
- int getA2temp_pin() // get the permanent pin
- void setA2temp_pin(int a1temp_pin) // set the temporary pin
- int getA2temp_userid() // get the temporary pin
- void setA2temp_userid(int a1temp_userid) // set the temporary user identification number
- int getA2temp_bal() // get the temporary balance
- void setA2temp_bal(int a1temp_bal) // set the temporary balance
- int getA2temp_withdraw() // get the temporary withdraw
- void setA2temp_withdraw(int a1temp_withdraw) // set the temporary withdraw
- getA2temp_deposit() // get the temporary deposit
- void setA2temp_deposit(int a2temp_deposit) // set the temporary deposit

Main Attributes of the Class:

DataStore: // abstract class

DataStore1:

```
/* Permanent Storage Variables */
String A1pin;    // user pin
String A1userid;          // user identification number
float A1bal;      // balance

/* Temporary Storage Variables */
```

```
String A1temp_pin;       //pin
String A1temp_userid;   // user identification no
float A1temp_bal;        // balance
float A1temp_withdraw;          // withdraw
float A1temp_deposit;   // deposit
```

DataStore2:

```
/* Permanent Storage Variables */
int A2pin;        // user pin
int A2userid;     // user identification number
int A2bal;        // balance

/* Temporary Storage Variables */
int A2temp_pin;          // pin
int A2temp_userid; // user id
int A2temp_bal;//balance
int A2temp_withdraw; //withdraw
int A2temp_deposit;   // deposit
```

4.2 AbstractSuper Class:

Purpose:
To group strategies for a particular ACCOUNT COMPONENT.

Responsibility of Operations:

AbstractSuper: all below operations are abstract operations.

- StoreData getStoredata ()
- IncorrectIdmsg getIncorrectid ()
- IncorrectPinmsg getIncorrectpin ()
- TooManyattempts getTooattempts ()
- DisplayMenu getDmenu ()
- MakeDeposit getMdeposit ()
- DisplayBalance getDbalance ()
- PromptPin getPromptpin ()
- MakeWithdraw getMwithdraw ()
- Penalty getPenalty ()
- IncorrectLockmsg getLock ()
- IncorrectUnlockmsg getUnlock ()
- NoFundsmsg getNofunds ()
- DataStore GetDataStore()

ConcreteFactory1:

- StoreData getStoredata() // returns object of Storedata1
- IncorrectIdmsg getIncorrectid() // returns object of Incorrectidmsg1
- IncorrectPinmsg getIncorrectpin() // returns object of Incorrectpinmsg1
- TooManyattempts getTooattempts() // returns object of TooManyattempts1
- DisplayMenu getDmenu() // returns object of DisplayMenu1
- MakeDeposit getMdeposit() // returns object of MakeDeposit1

- DisplayBalance getDbalance() // returns object of DisplayBalance1
- PromptPin getPromptpin() // returns object of PromptPin1
- MakeWithdraw getMwithdraw() // returns object of MakeWithdraw1
- Penalty getPenalty() // returns object of Penalty1
- IncorrectLockmsg getLock() // returns object of IncorrectLockMsg1
- IncorrectUnlockmsg getUnlock() // returns object of UnlockMsg1
- NoFundsmsg getNofunds() // returns object of NoFundsmsg1
- DataStore GetDataStore() // returns object of DataStore1

ConcreteFactory2:
- StoreData getStoredata() // returns object of Storedata2
- DataStore GetDataStore() // returns object of DataStore2
- IncorrectIdmsg getIncorrectid() // returns object of IncorrectIdmsg2
- IncorrectPinmsg getIncorrectpin() // returns object of IncorrectPinmsg2
- TooManyattempts getTooattempts() // returns object of TooManyattempts2
- DisplayMenu getDmenu() // returns object of DisplayMenu2
- MakeDeposit getMdeposit() // returns object of MakeDeposit2
- DisplayBalance getDbalance() // returns object of DisplayBalance2
- PromptPin getPromptpin() // returns object of PromptPin2
- MakeWithdraw getMwithdraw() // returns object of MakeWithdraw2
- NoFundsmsg getNofunds() // returns object of Nofundsmsg2

Main Attributes of the Class:

AbstractSuper:
ConcreteFactory1;
ConcreteFactory2:
No attributes exist for these classes.

4.3 MDA EFSM Class(Efsm.java)

Purpose:
Manages state machine for State pattern. Should be independent of different types of ATM Components. Captures platform independent behavior.

Responsibility of Operations:

public void Open() // store p, y and a in temp data store
public void Login() // set attempts to zero and calls Prompt for Pin message to do action in states
public void IncorrectLogin() // displays a incorrectId msg
public void IncorectPin(int max) // checks for maximum number of attempts and calls a operation to do action in states and performs state transition here
public void CorrectPinBelowMin() // Calls CorrectPinBelowMin to do action in respective state and changes to State Overdrawn
public void CorrectPinAboveMin() // Calls CorrectPinAboveMin to do action in respective state and changes to State to Ready
public void Deposit() // Calls Deposit to do action in states and performs respective state Transition

public void BelowMinBalance() // Calls BelowMinBalance to do action in states and changes to state overdrawn
public void AboveMinBalance() // Calls AboveMinBalance to do action in states and changes to state ready
public void Logout() // Calls Logout to do action in states and changes state respectively
public void Balance() // calls Balance to do action in states and performs state transition here
public void Withdraw() // Calls withdraw to do action in states and performs state transition here
public void WithdrawBelowMinBalance() // Calls WithdrawBelowMinBalance to do action in states and performs state transition here
public void NoFunds() // Calls NoFunds to do action in states and performs state transition here
public void Lock() // Calls Lock to do action in states and performs state transition here
public void IncorrectLock() // Calls IncorrectLock to do action in states and performs state transition here
public void Unlock() // Calls Unlock to do action in states and performs state transition here
public void IncorrectUnlock() // Calls IncorrectUnlock to do action in states and performs state transition here
public void Suspend() // Calls Suspend to do action in states and performs state transition here
public void Activate() // Calls Activate to do action in states and performs state transition here
public void Close() // Calls Close to do action in states and performs state transition here
public int get_attempts( ) // get the current attempts value
public void set_attempts(int attempts) // set the number of attempts

Main Attributes of the Class:

private StateSuper slist[] //Pointer to all states
StateSuper s //Pointer to current state
int attempts; // attempts variable
AbstractSuper factory =null; // initializes Abstract Factory
Output output = null; // initializes output

4.3 State Class

Purpose:
Manages states transitions and respective output actions.

Responsibility of Operations: all below operations are abstract operations.

- void Open()
- void Login()
- void IncorrectLogin()
- void IncorectPin(int max)
- void CorrectPinBelowMin()
- void CorrectPinAboveMin()
- void Deposit()
- void BelowMinBalance()
- void AboveMinBalance()
- void Logout()

- void Balance()
- void Withdraw()
- void WithdrawBelowMinBalance()
- void NoFunds()
- void Lock()
- void IncorrectLock()
- void Unlock()
- void IncorrectUnlock()
- void Suspend()
- void Activate()
- void Close()
- initializeOutput(Output output)
- int get_id()

Main Attributes of the Class:

Efsm *mda; //Pointer to EFSM
Output *output; //Pointer to Output
Int id; // state id

StartState Class

Purpose:
Creates the start state and initializes all the values respective to the ACCOUNT components

Responsibility of Operations:

```
public void Open(){
        // calls output action to store p, y and a in temp data store
        output.StoreData();
}
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =1; // state id

IdleState Class

Purpose:
Based on the stored pin values in takes care of Login operation of the ACCOUNT component.

Responsibility of Operations:

```
@Override
    public void Login()
    {
        // intializes number of attempts to zero and calls output action PromptPin
        mda.set_attempts(0);
```

```
		output.PromptPin();

	}

	@Override
	public void IncorrectLogin()
	{
		// Calls output action IncorrectID Message
		output.IncorrectIdmsg();
	}
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =2; // state id

CheckPinState Class

Purpose:
Checks the correctness of the user input pin and checks for maximum number of attempts to do the transaction menu items

Responsibility of Operations:

```
@Override
	public void IncorectPin(int max)
	{
		/* calls output action incorrectPin msg and calls action TooManyAttempts if number of attempts
		 * exceeds maximum
		 */

		if ( mda.get_attempts() < max)
		{

			output.IncorrectPinmsg();
			mda.set_attempts(mda.get_attempts()+1);
			//System.out.println("inside if"+ max+ mda.get_attempts());
		}
		else if( max == mda.get_attempts())
		{
			output.IncorrectPinmsg();
			output.TooManyattempts();
			mda.set_attempts(mda.get_attempts()+1);
		}

	}
```

```
@Override
public void CorrectPinBelowMin()
{
        // Calls output action Display Menu to display the Menu details
        output.DisplayMenu();
}
@Override
public void CorrectPinAboveMin()
{
        // Calls output action Display Menu to display the Menu details
        output.DisplayMenu();
}
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =3; // state id

ReadyState Class

Purpose:
From this State, we can perform all the transaction menu items like Deposit, Withdraw, Balance, Lock.

Responsibility of Operations:

```
@Override
public void Deposit()
{
        // Calls MakeDeposit output action
        output.MakeDeposit();
}
@Override
public void Balance()
{
        // Calls DisplayBalance to do output action
        output.DisplayBalance(); // Display balance
}
@Override
public void Withdraw()
{
        // Calls MakeWithdraw to do output action
        output.MakeWithdraw(); //withdraw action
}
@Override
public void NoFunds()
{
        // Calls NoFundsmsg to do respective output action
        output.NoFundsmsg(); //displays the no funds message
}
```

```
@Override
public void IncorrectLock(){
        // Calls IncorrectLockmsg to do output action
        output.IncorrectLockmsg();
}
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =4; // state id

S1State Class

Purpose:
This State is used as intermediate state while performing withdraw and lock operations from ready and overdrawn states.

Responsibility of Operations:

```
        @Override
public void WithdrawBelowMinBalance()
{
        // Calls Penalty to do output action with respect to Penalty
        output.Penalty();
}
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =5; // state id

LockedState Class

Purpose:
This State is used as intermediate state while performing withdraw and lock operations from ready and overdrawn states.

Responsibility of Operations:

```
@Override
public void IncorrectUnlock()
{
        // Calls Output action IncorrectUnlock message
        output.IncorrectUnlockmsg();
}
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =7; // state id

OverdrawnState Class

Purpose:
State where the user will be able to make only few operations like Deposit to get into Ready to perform all operation of transaction menu items.

Responsibility of Operations:

```
@Override
        public void IncorrectUnlock()
        {
                // Calls Output action IncorrectUnlock message
                output.IncorrectUnlockmsg();
        }
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =7; // state id

StopState Class

Purpose:
State where the user account will be closed and no longer available to perform any transactions
Responsibility of Operations:
// no output action available for this state.
Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =8; // state id

SuspendedState Class

Purpose:
State where the user account will be closed and no longer available to perform any transactions
Responsibility of Operations:

```
@Override
        public void Balance()
        {
                // Calls DisplayBalance to do output action
                output.DisplayBalance(); // display available balance
        }
```

Main Attributes of the Class: (initialized in constructor)

Efsm *mda; //Pointer to EFSM
Id =8; // state id

4.4 Output Processor Classes

(output.java)

Purpose:

This class will perform output actions for respective events called from input processor.

Responsibility of Operations:

```
public Output(AbstractSuper factory,DataStore dataStore)
        {
                // Initializes respective factory and datastore of ACCOUNT Components
     this.factory = factory;
     this.dataStore = dataStore;
  }
        public void StoreData(){
                // Initializes StoreData object and Calls output action StoreData
                storedata = factory.getStoredata();
                storedata.StoreData(dataStore);
        }
        public void PromptPin(){
                // Initializes PromptPin object and Calls output action PromptPin
                promptpin = factory.getPromptpin();
                promptpin.PromptPin();
        }
        public void DisplayMenu(){
                // Initializes DisplayMenu object and Calls output action DisplayMenu
                dmenu = factory.getDmenu();
                dmenu.DisplayMenu();
        }
        public void IncorrectIdmsg() {
                // Initializes IncorrectIdmsg object and Calls output action IncorrectIdmsg
                incorrectid = factory.getIncorrectid();
                incorrectid.IncorrectIdmsg();
        }
        public void IncorrectPinmsg() {
                // Initializes IncorrectPinmsg object and Calls output action IncorrectPinmsg
                incorrectpin = factory.getIncorrectpin();
                incorrectpin.IncorrectPinmsg();
        }
        public void TooManyattempts() {
                // Initializes TooManyattempts object and Calls output action TooManyattempts
                tooattempts = factory.getTooattempts();
                tooattempts.TooManyattempts();
        }
        public void IncorrectLockmsg(){
                // Initializes IncorrectLockmsg object and Calls output action IncorrectLockmsg
```

```
            lock = factory.getLock();
            lock.IncorrectLockmsg();

    }
    public void IncorrectUnlockmsg(){
            // Initializes IncorrectLockmsg object and Calls output action IncorrectLockmsg
            unlock = factory.getUnlock();
            unlock.IncorrectUnlockmsg();
    }
    public void DisplayBalance(){
            // Initializes DisplayBalance object and Calls output action DisplayBalance
            dbalance = factory.getDbalance();
            dbalance.DisplayBalance(dataStore);
    }
    public void Penalty(){
            // Initializes Penalty object and Calls output action Penalty
            penalty = factory.getPenalty();
            penalty.Penalty(dataStore);
    }
    public void NoFundsmsg(){
            // Initializes NoFundsmsg object and Calls output action NoFundsmsg
            nofunds = factory.getNofunds();
            nofunds.NoFundsmsg();
    }
    public void MakeDeposit()
    {
            // Initializes MakeDeposit object and Calls output action MakeDeposit
            mdeposit = factory.getMdeposit();
            mdeposit.MakeDeposit(dataStore);
    }
    public void MakeWithdraw()
    {
            // Initializes MakeWithdraw object and Calls output action MakeWithdraw
            mwithdraw = factory.getMwithdraw();
            mwithdraw.MakeWithdraw(dataStore);
    }
```

Main Attributes of the Class: (initialized in constructor)

```
    // Initialized pointers to all Output actions object
    private DataStore data;
    private StoreData storedata;
    private IncorrectIdmsg incorrectid;
    private IncorrectPinmsg incorrectpin;
    private TooManyattempts tooattempts;
    private DisplayMenu dmenu;
    private MakeDeposit mdeposit;
    private DisplayBalance dbalance;
    private PromptPin promptpin;
    private MakeWithdraw mwithdraw;
    private Penalty penalty;
    private IncorrectLockmsg lock;
    private IncorrectUnlockmsg unlock;
```

```
private NoFundsmsg nofunds;
AbstractSuper factory =null;
DataStore dataStore = null;
```

1) DisplayBalance Class

Purpose:
State where the output action of displaying account current balance is taken care

Responsibility of Operations:
public void DisplayBalance(DataStore ds) // abstract operation

Main Attributes of the Class: no attributes exist in this class

DisplayBalance1 Class

Purpose:
State where the output action of displaying ACCOUNT1 current balance is taken care

Responsibility of Operations:
public void DisplayBalance(DataStore ds) // displays current balance of account1

Main Attributes of the Class: no attributes exist in this class

DisplayBalance2 Class

Purpose:
State where the output action of displaying ACCOUNT2 current balance is taken care

Responsibility of Operations:
public void DisplayBalance(DataStore ds) // displays current balance of account2

Main Attributes of the Class: no attributes exist in this class

2) DisplayMenu Class

Purpose:
State where the output action of displays Transaction Menu Details available for ACCOUNT components

Responsibility of Operations:
public void DisplayMenu() // abstract operation

Main Attributes of the Class: no attributes exist in this class

DisplayMenu1 Class

Purpose:
State where the output action of displays Transaction Menu Details available for ACCOUNT1

Responsibility of Operations:
public void DisplayMenu() // displays transaction menu of account1

Main Attributes of the Class: no attributes exist in this class

DisplayMenu2 Class

Purpose:
State where the output action of displays Transaction Menu Details available for ACCOUNT2

Responsibility of Operations:
public void DisplayMenu() // displays transaction menu of account2

Main Attributes of the Class: no attributes exist in this class

3) IncorrectIdmsg Class:

Purpose:
State where the output action of displays Incorrect User ID Message for ACCOUNT components

Responsibility of Operations:
public void IncorrectIdmsg () // abstract operation

Main Attributes of the Class: no attributes exist in this class

IncorrectIdmsg1 Class

Purpose:
State where the output action of displays Incorrect User ID Message for ACCOUNT1

Responsibility of Operations:
public void IncorrectIdmsg () // displays Incorrect User ID Message for account1

Main Attributes of the Class: no attributes exist in this class

IncorrectIdmsg2 Class

Purpose:
State where the output action of displays Incorrect User ID Message for ACCOUNT2

Responsibility of Operations:
public void IncorrectIdmsg () // displays Incorrect User ID Message for account2

Main Attributes of the Class: no attributes exist in this class

4) IncorrectLockmsg Class:

Purpose:
State where the output action of displays Incorrect Lock Message for ACCOUNT components

Responsibility of Operations:
public void IncorrectLockmsg () // abstract operation

Main Attributes of the Class: no attributes exist in this class

IncorrectLockmsg1 Class

Purpose:
State where the output action of displays Incorrect Lock Message for ACCOUNT

Responsibility of Operations:
public void IncorrectLockmsg () // displays Incorrect Lock Message

Main Attributes of the Class: no attributes exist in this class

5) IncorrectPinmsg Class:

Purpose:
State where the output action of displays Incorrect Pin Message for ACCOUNT components

Responsibility of Operations:
public void IncorrectPinmsg () // abstract operation

Main Attributes of the Class: no attributes exist in this class

IncorrectPinmsg1 Class

Purpose:
State where the output action of displays Incorrect Pin Message for ACCOUNT1

Responsibility of Operations:
public void IncorrectPinmsg () // displays Incorrect Pin Message

Main Attributes of the Class: no attributes exist in this class

IncorrectPinmsg2 Class

Purpose:
State where the output action of displays Incorrect Pin Message for ACCOUNT2

Responsibility of Operations:

public void IncorrectPinmsg () // displays Incorrect Pin Message

Main Attributes of the Class: no attributes exist in this class

6) IncorrectUnlockmsg Class:

Purpose:

State where the output action of displays Incorrect UnlockMessage for ACCOUNT components

Responsibility of Operations:

public void IncorrectUnlockmsg () // abstract operation

Main Attributes of the Class: no attributes exist in this class

IncorrectUnlockmsg1 Class

Purpose:

State where the output action of displays Incorrect Unlock Message for ACCOUNT1

Responsibility of Operations:

public void IncorrectUnlockmsg () // displays Incorrect Unlock Message

Main Attributes of the Class: no attributes exist in this class

7) MakeDeposit Class

Purpose:

State where the output action of adding deposit amount to the current balance is done

Responsibility of Operations:

public void MakeDeposit(DataStore ds) // abstract operation

Main Attributes of the Class: no attributes exist in this class

MakeDeposit1 Class

Purpose:

State where the output action of adding deposit amount to the current balance for ACCOUNT1 is taken care

Responsibility of Operations:

public void MakeDeposit (DataStore ds) // adds deposit amount to the current balance and sets current balance again in DataStore1

Main Attributes of the Class: no attributes exist in this class

MakeDeposit2 Class

Purpose:
State where the output action of adding deposit amount to the current balance for ACCOUNT2 is taken care

Responsibility of Operations:
public void MakeDeposit (DataStore ds) // adds deposit amount to the current balance and sets current balance again in DataStore2

Main Attributes of the Class: no attributes exist in this class

8) MakeWithdraw Class

Purpose:
State where the output action of deducting withdraw amount from the current balance is done

Responsibility of Operations:
public void MakeWithdraw(DataStore ds) // abstract operation

Main Attributes of the Class: no attributes exist in this class

MakeWithdraw1 Class

Purpose:
State where the output action of deducting withdraw amount from the current balance for ACCOUNT1 is taken care

Responsibility of Operations:
public void MakeWithdraw (DataStore ds) // deducts withdraw amount from the current balance and sets current balance again in DataStore1

Main Attributes of the Class: no attributes exist in this class

MakeWithdraw2 Class

Purpose:
State where the output action of deducting withdraw amount from the current balance for ACCOUNT2 is taken care

Responsibility of Operations:
public void MakeWithdraw (DataStore ds) // deducts withdraw amount from the current balance and sets current balance again in DataStore2

Main Attributes of the Class: no attributes exist in this class

9) NoFundsmsg Class:

Purpose:
State where the output action of displays NoFunds Message for ACCOUNT components

Responsibility of Operations:
public void NoFundsmsg () // abstract operation

Main Attributes of the Class: no attributes exist in this class

NoFundsmsg1 Class

Purpose:
State where the output action of displays NoFunds Message for ACCOUNT1

Responsibility of Operations:
public void NoFundsmsg () // displays NoFunds Message

Main Attributes of the Class: no attributes exist in this class

10) Penalty Class:

Purpose:
State where the output action of applying penalty operation for ACCOUNT components

Responsibility of Operations:
public void Penalty () // abstract operation

Main Attributes of the Class: no attributes exist in this class

Penalty1 Class

Purpose:
State where the output action of applying penalty operation for ACCOUNT1 is done and current balance is set to the amount after applied penalty

Responsibility of Operations:
public void Penalty () // Penalty amount is applied to the current balance and set the new balance in DataStore1

Main Attributes of the Class: no attributes exist in this class

11) PromptPin Class

Purpose:
State where the output action which display prompting pin message

Responsibility of Operations:
public void PromptPin() // abstract operation

Main Attributes of the Class: no attributes exist in this class

PromptPin1 Class

Purpose:
State where the output action of displays PromptPin Message for ACCOUNT1 component

Responsibility of Operations:
public void PromptPin () // displays PromptPin Message for account1

Main Attributes of the Class: no attributes exist in this class

PromptPin2 Class

Purpose:
State where the output action of displays PromptPin Message for ACCOUNT2 component

Responsibility of Operations:
public void PromptPin () // displays PromptPin Message for account2

Main Attributes of the Class: no attributes exist in this class

12) StoreData Class

Purpose:
State where the output action of storing pin, userid, balance from temporary variables of DataStore to permanent variables respectively

Responsibility of Operations:
public void StoreData(DataStore ds) // abstract operation

Main Attributes of the Class: no attributes exist in this class

StoreData1 Class

Purpose:
State where the output action of storing pin, userid, balance from temporary variables of DataStore to permanent variables respectively for Account1

Responsibility of Operations:

public void StoreData (DataStore ds) // sets pin,userid, balance from temporary to permanent variables in DataStore1

Main Attributes of the Class: no attributes exist in this class

StoreData2 Class

Purpose:

State where the output action of storing pin, userid, balance from temporary variables of DataStore to permanent variables respectively for Account2

Responsibility of Operations:

public void StoreData(DataStore ds) // sets pin,userid, balance from temporary to permanent variables in DataStore2

Main Attributes of the Class: no attributes exist in this class

13) TooManyAttemptsmsg Class:

Purpose:

State where the output action of displays TooManyattempts Message for ACCOUNT components

Responsibility of Operations:

public void TooManyattempts () // abstract operation

Main Attributes of the Class: no attributes exist in this class

TooManyAttemptsmsg1 Class

Purpose:

State where the output action of displays TooManyattempts Message for ACCOUNT1

Responsibility of Operations:

public void TooManyattempts () // displays TooManyattempts Message

Main Attributes of the Class: no attributes exist in this class

TooManyAttemptsmsg2 Class

Purpose:

State where the output action of displays TooManyattempts Message for ACCOUNT2

Responsibility of Operations:

public void TooManyattempts () // displays TooManyattempts Message

Main Attributes of the Class: no attributes exist in this class

4 . Source – Code And Patterns

DATASTORE CLASSES:

//DataStore.java – All opeartions in this class are abstract operations

```
package dataStore;

abstract public class DataStore {

	public String getA1pin() {
		return null;
	}
	public void setA1pin(String a1pin) {
	}
	public String getA1userid() {
		return null;
	}
	public void setA1userid(String a1userid) {
	}
	public float getA1bal() {
		return 0;
	}
	public void setA1bal(float a1bal) {
	}
	public String getA1temp_pin() {
		return null;
	}
	public void setA1temp_pin(String a1temp_pin) {
	}
	public String getA1temp_userid() {
		return null;
	}
	public void setA1temp_userid(String a1temp_userid) {
	}
	public float getA1temp_bal() {
		return 0;
	}
	public void setA1temp_bal(float a1temp_bal) {
	}
	public float getA1temp_withdraw() {
		return 0;
	}
	public void setA1temp_withdraw(float a1temp_withdraw) {
	}
	public float getA1deposit() {
		return 0;
	}
	public void setA1deposit(float a1deposit) {
	}
```

```
	public float getA1temp_deposit() {
		return 0;
	}
	public void setA1temp_deposit(float a1temp_deposit) {
	}
	public int getA2pin() {
		return 0;
	}
	public void setA2pin(int a2pin) {
	}
	public int getA2userid() {
		return 0;
	}
	public void setA2userid(int a2userid) {
	}
	public int getA2bal() {
		return 0;
	}
	public void setA2bal(int a2bal) {
	}
	public int getA2temp_pin() {
		return 0;
	}
	public void setA2temp_pin(int a2temp_pin) {
	}
	public int getA2temp_userid() {
		return 0;
	}
	public void setA2temp_userid(int a2temp_userid) {
	}
	public int getA2temp_bal() {
		return 0;
	}
	public void setA2temp_bal(int a2temp_bal) {
	}
	public int getA2temp_withdraw() {
		return 0;
	}
	public void setA2temp_withdraw(int a2temp_withdraw) {
	}
	public int getA2temp_deposit() {
		return 0;
	}
	public void setA2temp_deposit(int a2temp_deposit) {
	}

}
```

// DataStore1.java

```
package dataStore;
```

```
public class DataStore1 extends DataStore {

	private String A1pin;
	private String A1userid;
	private float A1bal;
	private String A1temp_pin;
	private String A1temp_userid;
	private float A1temp_bal;
	private float A1temp_withdraw;
	private float A1temp_deposit;

	public String getA1pin() {
		return A1pin;
	}
	public void setA1pin(String a1pin) {
		A1pin = a1pin;
	}
	public String getA1userid() {
		return A1userid;
	}
	public void setA1userid(String a1userid) {
		A1userid = a1userid;
	}
	public float getA1bal() {
		return A1bal;
	}
	public void setA1bal(float a1bal) {
		A1bal = a1bal;
	}
	public String getA1temp_pin() {
		return A1temp_pin;
	}
	public void setA1temp_pin(String a1temp_pin) {
		A1temp_pin = a1temp_pin;
	}
	public String getA1temp_userid() {
		return A1temp_userid;
	}
	public void setA1temp_userid(String a1temp_userid) {
		A1temp_userid = a1temp_userid;
	}
	public float getA1temp_bal() {
		return A1temp_bal;
	}
	public void setA1temp_bal(float a1temp_bal) {
		A1temp_bal = a1temp_bal;
	}
	public float getA1temp_withdraw() {
		return A1temp_withdraw;
	}
```

```
	public void setA1temp_withdraw(float a1temp_withdraw) {
		A1temp_withdraw = a1temp_withdraw;
	}
	public float getA1temp_deposit() {
		return A1temp_deposit;
	}
	public void setA1temp_deposit(float a1temp_deposit) {
		A1temp_deposit = a1temp_deposit;
	}



}
```

// DataStore2.java

```
package dataStore;

public class DataStore2 extends DataStore   {
	private int A2pin;
	private int A2userid;
	private int A2bal;
	private int A2temp_pin;
	private int A2temp_userid;
	private int A2temp_bal;
	private int A2temp_withdraw;
	private int A2temp_deposit;

	public int getA2temp_deposit() {
		return A2temp_deposit;
	}
	public void setA2temp_deposit(int a2temp_deposit) {
		A2temp_deposit = a2temp_deposit;
	}

	public int getA2pin() {
		return A2pin;
	}
	public void setA2pin(int a2pin) {
		A2pin = a2pin;
	}
	public int getA2userid() {
		return A2userid;
	}
	public void setA2userid(int a2userid) {
		A2userid = a2userid;
	}
	public int getA2bal() {
		return A2bal;
	}
	public void setA2bal(int a2bal) {
```

```
		A2bal = a2bal;
	}
	public int getA2temp_pin() {
		return A2temp_pin;
	}
	public void setA2temp_pin(int a2temp_pin) {
		A2temp_pin = a2temp_pin;
	}
	public int getA2temp_userid() {
		return A2temp_userid;
	}
	public void setA2temp_userid(int a2temp_userid) {
		A2temp_userid = a2temp_userid;
	}
	public int getA2temp_bal() {
		return A2temp_bal;
	}
	public void setA2temp_bal(int a2temp_bal) {
		A2temp_bal = a2temp_bal;
	}
	public int getA2temp_withdraw() {
		return A2temp_withdraw;
	}
	public void setA2temp_withdraw(int a2temp_withdraw) {
		A2temp_withdraw = a2temp_withdraw;
	}

}
```

ABSTRACT FACTORY PATTERN

// AbstractSuper.java - Operations in this class are abstract operations

```
package abstractFactory;

import dataStore.*;
import strategy.*;

public class AbstractSuper {

	public DataStore CreateDataStore()
	{
		return null;
	}
	public DataStore GetDataStore()
	{
		return null;
	}

	public StoreData getStoredata() {
		return null;
```

```
	}

	public IncorrectIdmsg getIncorrectid() {
		return null;
	}
	public IncorrectPinmsg getIncorrectpin() {
		return null;
	}
	public TooManyattempts getTooattempts() {
		return null;
	}
	public DisplayMenu getDmenu() {
		return null;
	}

	public MakeDeposit getMdeposit() {
		return null;
	}

	public DisplayBalance getDbalance() {
		return null;
	}

	public PromptPin getPromptpin() {
		return null;
	}

	public MakeWithdraw getMwithdraw() {
		return null;
	}

	public Penalty getPenalty() {
		return null;
	}

	public IncorrectLockmsg getLock() {
		return null;
	}

	public IncorrectUnlockmsg getUnlock() {
		return null;
	}

	public NoFundsmsg getNofunds() {
		return null;
	}

}
```

// ConcreteFactory1.java

```
package abstractFactory;

import dataStore.DataStore;
import strategy.*;
import dataStore.DataStore1;

public class ConcreteFactory1 extends AbstractSuper {

	DataStore dataStore = new DataStore1();

	public DataStore CreateDataStore()
	{
		// creates DataStore 1 object
		return(this.dataStore);
	}

	public DataStore GetDataStore()
	{
		// returns object of DataStore1
		return this.dataStore;
	}

	public StoreData getStoredata() {
		// returns object of StoreData1
		return new StoreData1();
	}

	public IncorrectIdmsg getIncorrectid() {
		// returns object of IncorrectIdmsg1
		return new IncorrectIdmsg1();
	}
	public IncorrectPinmsg getIncorrectpin() {
		// returns object of IncorrectPinmsg1
		return new IncorrectPinmsg1();
	}
	public TooManyattempts getTooattempts() {
		// returns object of TooManyattempts1
		return new TooManyattempts1();
	}
	public DisplayMenu getDmenu() {
		// returns object of DisplayMenu1
		return new DisplayMenu1();
	}

	public MakeDeposit getMdeposit() {
```

```
		// returns object of MakeDeposit1
		return new MakeDeposit1();
	}

	public DisplayBalance getDbalance() {
		// returns object of DisplayBalance1
		return new DisplayBalance1();
	}

	public PromptPin getPromptpin() {
		// returns object of PromptPin1
		return new PromptPin1();
	}

	public MakeWithdraw getMwithdraw() {
		// returns object of MakeWithdraw1
		return new MakeWithdraw1();
	}

	public Penalty getPenalty() {
		// returns object of Penalty1
		return new Penalty1();
	}

	public IncorrectLockmsg getLock() {
		// returns object of IncorrectLockmsg1
		return new IncorrectLockmsg1();
	}

	public IncorrectUnlockmsg getUnlock() {
		// returns object of IncorrectUnlockmsg1
		return new IncorrectUnlockmsg1();
	}

	public NoFundsmsg getNofunds() {
		// returns object of NoFundsmsg1
		return new NoFundsmsg1();
	}

}
```

// ConcreteFactory2.java

```
package abstractFactory;


import dataStore.DataStore;
import dataStore.DataStore2;
import strategy.*;


public class ConcreteFactory2 extends AbstractSuper {
	DataStore dataStore = new DataStore2();

	public StoreData getStoredata() {
		// returns object of StoreData2
		return new StoreData2();
	}
	public DataStore CreateDataStore()
	{
		// creates DataStore2 object
		return(this.dataStore);
	}
	public DataStore GetDataStore()
	{
		// returns object of DataStore2
		return this.dataStore;
	}
	public IncorrectIdmsg getIncorrectid() {
		// returns object of IncorrectIdmsg2
		return new IncorrectIdmsg2();
	}
	public IncorrectPinmsg getIncorrectpin() {
		// returns object of IncorrectPinmsg2
		return new IncorrectPinmsg2();
	}
	public TooManyattempts getTooattempts() {
		// returns object of TooManyattempts2
		return new TooManyattempts2();
	}
	public DisplayMenu getDmenu() {
		// returns object of DisplayMenu2
		return new DisplayMenu2();
	}

	public MakeDeposit getMdeposit() {
		// returns object of MakeDeposit2
		return new MakeDeposit2();
	}


	public DisplayBalance getDbalance() {
		// returns object of DisplayBalance2
		return new DisplayBalance2();
```

```
	}

	public PromptPin getPromptpin() {
		// returns object of PromptPin2
		return new PromptPin2();
	}

	public MakeWithdraw getMwithdraw() {
		// returns object of MakeWithdraw2
		return new MakeWithdraw2();
	}

	public NoFundsmsg getNofunds() {
		// returns object of NoFundsmsg1
		return new NoFundsmsg1();
	}

}
```

MDA – EFSM and STATE DESIGN PATTERN

// Efsm.java

```
package mda_Efsm;

import mda_States.*;
import outputProcessor.Output;
import abstractFactory.*;;

public class Efsm {

	// pointer to all States
	private StateSuper slist[] = { new StartState(this), new IdleState(this), new CheckpinState(this),
			new ReadyState(this), new S1State(this), new OverdrawnState(this), new
LockedState(this), new SuspendedState(this),
			new StopState(this)};
	// Pointer to current State
	public StateSuper s = slist[0];
	// attempts variable for checking pin entries
	public int attempts;
	AbstractSuper factory =null; // initializes abstract factory
	Output output = null; // initializes output processer

	public void Open()
	{
		// store p, y and a in temp data store
		s.Open();
		switch (s.get_id()) {
		case 1: s=slist[1]; //State change from Start to Idle State
		System.out.println("State changed to Idle");
		break;
```

```
		default:
			System.out.println("Operation not permitted in this state");
			break;
	}
	}
	public void Login()
	{
		// set attempts to zero and Display Prompt for Pin message
		s.Login();
		switch (s.get_id()) {
		case 2: s=slist[2]; //State change from Idle to Check Pin State
		System.out.println("State changed to CheckPin");
		break;
		default:
			System.out.println("Operation not permitted in this state");
			break;
		}
	}
	public void IncorrectLogin()
	{
		// displays a incorrectId msg
		s.IncorrectLogin();
		// No change of state
	}
	public void IncorectPin(int max)
	{
		// checks for maximum number of attempts and displays a TooManyAttemptsMsg if
exceeds maxximum
		s.IncorectPin(max);
		switch (s.get_id()) {
		case 3:
		if(attempts  == max+1 )
		{
		s=slist[1];
		System.out.println("State changed to IDLE");
		}
		else if(attempts  < max+1)
		{	//No change of State
		System.out.println("State - CHECKPIN");
		}
		break;
		}
	}
	public void CorrectPinBelowMin()
	{
		// Calls CorrectPinBelowMin to do action in respective state and changes to State
Overdrawn
		s.CorrectPinBelowMin();
		switch (s.get_id()) {
		case 3: s = slist[5];
		System.out.println("State changed to OVERDRAWN");
```

```
		break;
		default:
			System.out.println("Operation not permitted in this state");
			break;
		}
	}
	public void CorrectPinAboveMin()
	{
		/**
		 * Calls CorrectPinAboveMin to do action in respective state and changes to State to
Ready
		 */
		s.CorrectPinAboveMin();
		switch (s.get_id()) {
		case 3: s = slist[3];
		System.out.println("State changed to READY");
		break;
		default:
			System.out.println("Operation not permitted in this state");
			break;
		}
	}
	public void Deposit()
	{
		/**
		 * Calls Deposit to do action in respective state and performs respective state Transition
		 */
		s.Deposit();
		switch (s.get_id()) {
		case 4: s = slist[3];
		System.out.println("State - READY");
		break;
		case 6: s = slist[4];
		System.out.println("State changed to S1");
		break;
		default:
			System.out.println("Operation not permitted in this state");
			break;
		}
	}
	public void BelowMinBalance()
	{
		/**
		 * Calls BelowMinBalance to do action in states and changes to state overdrawn
		 */
		s.BelowMinBalance();
		switch (s.get_id()) {
		case 5: s = slist[5];
		System.out.println("State changed to OVERDRAWN");
		break;
		}
```

```
}
public void AboveMinBalance()
{
	/**
	 * Calls AboveMinBalance to do action in states and changes to state ready
	 */
	s.AboveMinBalance();
	switch (s.get_id()) {
	case 5: s = slist[3];
	System.out.println("State changed to READY");
	break;
	}
}
public void Logout()
{
	/**
	 * Calls Logout to do action in states and changes state respectively
	 */
	s.Logout();
	switch (s.get_id()) {
	case 3: s = slist[1];
	System.out.println("State changed to IDLE");
	break;
	case 4: s = slist[1];
	System.out.println("State changed to IDLE");
	break;
	case 6: s = slist[1];
	System.out.println("State changed to IDLE");
	break;
	default:
		System.out.println("Operation not permitted in this state");
		break;
}
}
public void Balance()
{
	/**
	 * Calls Balance to do action in states and performs state transition here
	 */
	s.Balance();
	// No state Change
	switch (s.get_id()) {
	case 4: s = slist[3];
	System.out.println("State - READY");
	break;
	case 8: s = slist[7];
	System.out.println("State - Suspended");
	break;
	case 6: s = slist[5];
	System.out.println("State - Overdrawn");
	break;
```

```
		default:
			System.out.println("Operation not permitted in this state");
			break;
		}
	}
	public void Withdraw()
	{
		/**
		 * Calls withdraw to do action in states and performs state transition here
		 */
		s.Withdraw();
		switch (s.get_id()) {
		case 4: s = slist[4];
		System.out.println("State changed to S1");
		break;
		case 6: s = slist[5];
		System.out.println("State - Overdrawn"); // no change of state
		break;
		default:
			System.out.println("Operation not permitted in this state");
			break;

		}
	}
	public void WithdrawBelowMinBalance()
	{
		/**
		 * Calls WithdrawBelowMinBalance to do action in states and performs state transition
here
		 */
		s.WithdrawBelowMinBalance();
		switch (s.get_id()) {
		case 5: s = slist[5];
		System.out.println("State changed to OVERDRAWN");
		break;

		}
	}
	public void NoFunds()
	{
		/**
		 * Calls NoFunds to do action in states and performs state transition here
		 */
		s.NoFunds();
		switch (s.get_id()) {
		case 4: s = slist[3];
		System.out.println("State - READY");
		break;

		}
	}
```

```
    public void Lock()
    {
// Calls Lock to do action in states and performs state transition here
            s.Lock();
            switch (s.get_id()) {
            case 4: s = slist[6];
            System.out.println("State changed to LOCKED");
            break;
            case 6: s = slist[6];
            System.out.println("State changed to LOCKED");
            break;
            default:
                    System.out.println("Operation not permitted in this state");
                    break;

            }
    }
    public void IncorrectLock(){

            // Calls IncorrectLock to do action in states and performs state transition here

            s.IncorrectLock();
            switch (s.get_id()) {
            case 4: s = slist[3];
            // no state change
            System.out.println("State - READY");
            break;
            case 6: s = slist[5];
            System.out.println("State - Overdrawn");
            break;
            default:
                    System.out.println("Operation not permitted in this state");
                    break;

            }
    }
    public void Unlock()
    {
            // Calls Unlock to do action in states and performs state transition here
            s.Unlock();
            switch (s.get_id()) {
            case 7: s = slist[4];
            System.out.println("State changed to S1");
            break;
            default:
                    System.out.println("Operation not permitted in this state");
                    break;

            }
    }
    public void IncorrectUnlock()
```

```
{
        // Calls IncorrectUnlock to do action in states and performs state transition here
        s.IncorrectUnlock();
        switch (s.get_id()) {
        case 7: s = slist[6];
        System.out.println("State - LOCK");  // no change of state
        break;

        }
}
public void Suspend()
{
        // Calls Suspend to do action in states and performs state transition here
        s.Suspend();
        switch (s.get_id()) {
        case 4: s = slist[7];
        System.out.println("State changed to SUSPENDED");
        break;
        default:
                System.out.println("Operation not permitted in this state");
                break;
        }
}
public void Activate()
{
        // Calls Activate to do action in states and performs state transition here
        s.Activate();
        switch (s.get_id()) {
        case 8: s = slist[3];
        System.out.println("State changed to READY");
        break;
        default:
                System.out.println("Operation not permitted in this state");
                break;
        }
}
public void Close()
{
        // Calls Close to do action in states and performs state transition here
        s.Close();
        switch (s.get_id()) {
        case 8: s = slist[8];
        System.out.println("State changed to CLOSED");
        break;
        default:
                System.out.println("Operation not permitted in this state");
                break;
        }
}
public int get_attempts( )
{
```

```
		return attempts;
	}
	public void set_attempts(int attempts) {
		this.attempts = attempts;
	}
}
```

// StateSuper.java

```
package mda_States;

import outputProcessor.Output;
import mda_Efsm.Efsm;

public class StateSuper {

	protected static Output output; // initialize pointer to output object
	protected int id; // declares state id
	protected Efsm mda; // intialize pointer to efsm object


	public void initializeOutput(Output output){
		// initializes output processor
		this.output = output;
	}

	public int get_id()
	{
		// returns current state id
		return id;
	}
	public void Open()
	{
		// Do nothing
	}
	public void Login()
	{
		// Do nothing
	}
	public void IncorrectLogin()
	{
		// Do nothing
	}
	public void IncorectPin(int max)
	{
		// Do nothing
	}
	public void CorrectPinBelowMin()
	{
		// Do nothing
	}
```

```
public void CorrectPinAboveMin()
{
	// Do nothing
}
public void Deposit()
{
	// Do nothing
}
public void BelowMinBalance()
{
	// Do nothing
}
public void AboveMinBalance()
{
	// Do nothing
}
public void Logout()
{
	// Do nothing
}
public void Balance()
{
	// Do nothing
}
public void Withdraw()
{
	// Do nothing
}
public void WithdrawBelowMinBalance()
{
	// Do nothing
}
public void NoFunds()
{
	// Do nothing
}
public void Lock()
{
	// Do nothing
}
public void IncorrectLock(){
	// Do nothing
}
public void Unlock()
{
	// Do nothing
}
public void IncorrectUnlock()
{
	// Do nothing
}
```

```
	public void Suspend()
	{
		// Do nothing
	}
	public void Activate()
	{
		// Do nothing
	}
	public void Close()
	{
		// Do nothing
	}

}
```

// StartState.java

```
package mda_States;

import mda_Efsm.Efsm;
import outputProcessor.Output;

public class StartState extends StateSuper {

	Efsm mda =null;

   public StartState(Efsm mda)
   {
      this.mda =  mda; // pointer efsm object
      id = 1; // initializes state id
   }

	@Override
	public void Open()
	{
		// calls output action to store p, y and a in temp data store
		output.StoreData();
	}
	@Override
	public void Login()
	{
	 // Do Nothing
	}
	@Override
	public void IncorrectLogin()
	{
		// Do Nothing
	}
	@Override
	public void IncorectPin(int max)
	{
```

```
		// Do Nothing
}
@Override
public void CorrectPinBelowMin()
{
		// Do Nothing
}
@Override
public void CorrectPinAboveMin()
{
		// Do Nothing
}
@Override
public void Deposit()
{
		// Do Nothing
}
@Override
public void BelowMinBalance()
{
		// Do Nothing
}
@Override
public void AboveMinBalance()
{
		// Do Nothing
}
@Override
public void Logout()
{
		// Do Nothing
}
@Override
public void Balance()
{
		// Do Nothing
}
@Override
public void Withdraw()
{
		// Do Nothing
}
@Override
public void WithdrawBelowMinBalance()
{
		// Do Nothing
}
@Override
public void NoFunds()
{
		// Do Nothing
```

```
	}
	@Override
	public void Lock()
	{
		// Do Nothing
	}
	@Override
	public void IncorrectLock()
	{
		// Do Nothing
	}
	@Override
	public void Unlock()
	{
		// Do Nothing
	}
	@Override
	public void IncorrectUnlock()
	{
		// Do Nothing
	}
	@Override
	public void Suspend()
	{
		// Do Nothing
	}
	@Override
	public void Activate()
	{
		// Do Nothing
	}
	@Override
	public void Close()
	{
		// Do Nothing
	}

}
```

// IdleState.java

```
package mda_States;

import mda_Efsm.Efsm;
import outputProcessor.Output;

public class IdleState extends StateSuper {

	public IdleState(Efsm ef){
	this.mda =  ef;  // pointer efsm object
	id = 2; // initializes state id
```

```
}

@Override
public void Login()
{
	// intializes number of attempts to zero and calls output action PromptPin
	mda.set_attempts(0);
	output.PromptPin();

}

@Override
public void IncorrectLogin()
{
	// Calls output action IncorrectID Message
	output.IncorrectIdmsg();
}
@Override
public void Open()
{
	// Do Nothing
}
@Override
public void IncorectPin(int max)
{
	// Do Nothing
}
@Override
public void CorrectPinBelowMin()
{
	// Do Nothing
}
@Override
public void CorrectPinAboveMin()
{
	// Do Nothing
}
@Override
public void Deposit()
{
	// Do Nothing
}
@Override
public void BelowMinBalance()
{
	// Do Nothing
}
@Override
public void AboveMinBalance()
{
	// Do Nothing
```

```
}
@Override
public void Logout()
{
	// Do Nothing
}
@Override
public void Balance()
{
	// Do Nothing
}
@Override
public void Withdraw()
{
	// Do Nothing
}
@Override
public void WithdrawBelowMinBalance()
{
	// Do Nothing
}
@Override
public void NoFunds()
{
	// Do Nothing
}
@Override
public void Lock()
{
	// Do Nothing
}
@Override
public void IncorrectLock(){
	// Do Nothing
}
@Override
public void Unlock()
{
	// Do Nothing
}
@Override
public void IncorrectUnlock()
{
	// Do Nothing
}
@Override
public void Suspend()
{
	// Do Nothing
}
@Override
```

```
	public void Activate()
	{
		// Do Nothing
	}
	@Override
	public void Close()
	{
		// Do Nothing
	}

}
```

// CheckpinState.java

```
package mda_States;

import mda_Efsm.Efsm;
import outputProcessor.Output;

public class CheckpinState extends StateSuper {
	//Efsm mda =null;

	public CheckpinState(Efsm ef) {
		mda =  ef; // pointer to efsm object
		id =3; //intializes state id
	}
	@Override
	public void Open()
	{

	}
	@Override
	public void Login()
	{

	}
	@Override
	public void IncorrectLogin()
	{

	}
	@Override
	public void IncorectPin(int max)
	{
		/* calls output action incorrectPin msg and calls action TooManyAttempts if number of
attempts
		 * exceeds maximum
		 */

		if ( mda.get_attempts() < max)
		{
```

```
                output.IncorrectPinmsg();
                mda.set_attempts(mda.get_attempts()+1);
                //System.out.println("inside if"+ max+ mda.get_attempts());
        }
        else if( max == mda.get_attempts())
        {
                output.IncorrectPinmsg();
                output.TooManyattempts();
                mda.set_attempts(mda.get_attempts()+1);
        }

}
@Override
public void CorrectPinBelowMin()
{
        // Calls output action Display Menu to display the Menu details
        output.DisplayMenu();
}
@Override
public void CorrectPinAboveMin()
{
        // Calls output action Display Menu to display the Menu details
        output.DisplayMenu();
}
@Override
public void Deposit()
{
        // Do nothing
}
@Override
public void BelowMinBalance()
{
        // Do nothing
}
@Override
public void AboveMinBalance()
{
        // Do nothing
}
@Override
public void Logout()
{
        // Do nothing
}
@Override
public void Balance()
{
        // Do nothing
}
@Override
```

```
	public void Withdraw()
	{
		// Do nothing
	}
	@Override
	public void WithdrawBelowMinBalance()
	{
		// Do nothing
	}
	@Override
	public void NoFunds()
	{
		// Do nothing
	}
	@Override
	public void Lock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectLock(){
		// Do nothing
	}
	@Override
	public void Unlock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectUnlock()
	{
		// Do nothing
	}
	@Override
	public void Suspend()
	{
		// Do nothing
	}
	@Override
	public void Activate()
	{
		// Do nothing
	}
	@Override
	public void Close()
	{
		// Do nothing
	}

}
```

// LockedState.java

```
package mda_States;

import mda_Efsm.Efsm;

public class LockedState extends StateSuper {

	public LockedState( Efsm ef){
		this.mda =ef; // pointer to efsm object
		id = 7; // initializes state id
	}
	@Override
	public void Open()
	{
		// Do Nothing
	}
	@Override
	public void Login()
	{
		// Do Nothing
	}
	@Override
	public void IncorrectLogin()
	{
		// Do Nothing
	}
	@Override
	public void IncorectPin(int max)
	{
		// Do Nothing
	}
	@Override
	public void CorrectPinBelowMin()
	{
		// Do Nothing
	}
	@Override
	public void CorrectPinAboveMin()
	{
		// Do Nothing
	}
	@Override
	public void Deposit()
	{
		// Do Nothing
	}
	@Override
	public void BelowMinBalance()
	{
		// Do Nothing
```

```
}
@Override
public void AboveMinBalance()
{
	// Do Nothing
}
@Override
public void Logout()
{
	// Do Nothing
}
@Override
public void Balance()
{
	// Do Nothing
}
@Override
public void Withdraw()
{
	// Do Nothing
}
@Override
public void WithdrawBelowMinBalance()
{
	// Do Nothing
}
@Override
public void NoFunds()
{
	// Do Nothing
}
@Override
public void Lock()
{
	// Do Nothing
}
@Override
public void IncorrectLock(){
	// Do Nothing
}
@Override
public void Unlock()
{
	// Do Nothing
}
@Override
public void IncorrectUnlock()
{
	// Calls Output action IncorrectUnlock message
	output.IncorrectUnlockmsg();
}
```

```
	@Override
	public void Suspend()
	{
		// Do Nothing
	}
	@Override
	public void Activate()
	{
		// Do Nothing
	}
	@Override
	public void Close()
	{
		// Do Nothing
	}

}
```

// OverdrawnState.java

```
package mda_States;

import mda_Efsm.Efsm;

public class OverdrawnState extends StateSuper {

	public OverdrawnState(Efsm ef){
		mda =  ef; // pointer to efsm object
		id =6; // initializes to state id
	}
	@Override
	public void Open()
	{
		// Do nothing
	}
	@Override
	public void Login()
	{
		// Do nothing
	}
	@Override
	public void IncorrectLogin()
	{
		// Do nothing
	}
	@Override
	public void IncorectPin(int max)
	{
		// Do nothing
	}
	@Override
```

```
public void CorrectPinBelowMin()
{
	// Do nothing
}
@Override
public void CorrectPinAboveMin()
{
	// Do nothing
}
@Override
public void Deposit()
{
	// Calls output action MakeDeposit
	output.MakeDeposit();
}
@Override
public void BelowMinBalance()
{
	// Do nothing
}
@Override
public void AboveMinBalance()
{
	// Do nothing
}
@Override
public void Logout()
{
	// Do nothing
}
@Override
public void Balance()
{
	// Calls output action Display Balance
	output.DisplayBalance();
}
@Override
public void Withdraw()
{
	// Calls Output Action Withdraw
	output.NoFundsmsg(); //withdraw action
}
@Override
public void WithdrawBelowMinBalance()
{
	// Do nothing
}
@Override
public void NoFunds()
{
	// Do nothing
```

```
	}
	@Override
	public void Lock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectLock(){

		// calls ouput action incorrectlock to display incorrect lock message
		output.IncorrectLockmsg();
	}
	@Override
	public void Unlock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectUnlock()
	{
		// Do nothing
	}
	@Override
	public void Suspend()
	{
		// Do nothing
	}
	@Override
	public void Activate()
	{
		// Do nothing
	}
	@Override
	public void Close()
	{
		// Do nothing
	}

}
```

// ReadyState.java

```
package mda_States;

import mda_Efsm.Efsm;
import outputProcessor.Output;

public class ReadyState extends StateSuper {

	public ReadyState(Efsm ef){
		this.mda =  ef; // pointer to efsm object
```

```
        id = 4; // initializes state id
}

@Override
public void Open()
{
        // Do nothing
}
@Override
public void Login()
{
        // Do nothing
}
@Override
public void IncorrectLogin()
{
        // Do nothing
}
@Override
public void IncorectPin(int max)
{
        // Do nothing
}
@Override
public void CorrectPinBelowMin()
{
        // Do nothing
}
@Override
public void CorrectPinAboveMin()
{
        // Do nothing
}
@Override
public void Deposit()
{
        // Calls MakeDeposit output action
        output.MakeDeposit();
}
@Override
public void BelowMinBalance()
{
        // Do nothing
}
@Override
public void AboveMinBalance()
{
        // Do nothing
}
@Override
public void Logout()
```

```
{
	// Do nothing
}
@Override
public void Balance()
{
	// Calls DisplayBalance to do output action
	output.DisplayBalance(); // Display balance
}
@Override
public void Withdraw()
{
	// Calls MakeWithdraw to do output action
	output.MakeWithdraw(); //withdraw action
}
@Override
public void WithdrawBelowMinBalance()
{
	// Do nothing
}
@Override
public void NoFunds()
{
	// Calls NoFundsmsg to do respective output action
	output.NoFundsmsg(); //displays the no funds message
}
@Override
public void Lock()
{
	// Do nothing
}
@Override
public void IncorrectLock(){
	// Calls IncorrectLockmsg to do output action
	output.IncorrectLockmsg();
}
@Override
public void Unlock()
{
	// Do nothing
}
@Override
public void IncorrectUnlock()
{
	// Do nothing
}
@Override
public void Suspend()
{
	// Do nothing
}
```

```
	@Override
	public void Activate()
	{
		// Do nothing
	}
	@Override
	public void Close()
	{
		// Do nothing
	}
}
```

// S1State.java

```
package mda_States;

import mda_Efsm.Efsm;

public class S1State extends StateSuper{

	public S1State(Efsm ef) {

		this.mda =  ef; // pointer to efsm object
		id = 5; // initializes state id
	}
	@Override
	public void Open()
	{
		// Do nothing
	}
	@Override
	public void Login()
	{
		// Do nothing
	}
	@Override
	public void IncorrectLogin()
	{
		// Do nothing
	}
	@Override
	public void IncorectPin(int max)
	{
		// Do nothing
	}
	@Override
	public void CorrectPinBelowMin()
	{
		// Do nothing
	}
	@Override
```

```
public void CorrectPinAboveMin()
{
	// Do nothing
}
@Override
public void Deposit()
{
	// Do nothing
}
@Override
public void BelowMinBalance()
{
	// Do nothing
}
@Override
public void AboveMinBalance()
{
	// Do nothing
}
@Override
public void Logout()
{
	// Do nothing
}
@Override
public void Balance()
{
	// Do nothing
}
@Override
public void Withdraw()
{
	// Do nothing
}
@Override
public void WithdrawBelowMinBalance()
{
	// Calls Penalty to do output action with respect to Penalty
	output.Penalty();
}
@Override
public void NoFunds()
{
	// Do nothing
}
@Override
public void Lock()
{
	// Do nothing
}
@Override
```

```
	public void IncorrectLock(){
		// Do nothing
	}
	@Override
	public void Unlock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectUnlock()
	{
		// Do nothing
	}
	@Override
	public void Suspend()
	{
		// Do nothing
	}
	@Override
	public void Activate()
	{
		// Do nothing
	}
	@Override
	public void Close()
	{
		// Do nothing
	}
}
```

// StopState.java

```
package mda_States;

import mda_Efsm.Efsm;

public class StopState extends StateSuper{

	public StopState(Efsm ef)
	{
		this.mda = ef; // pointer to efsm object
		id =9; // initializes state id
	}
	@Override
	public void Open()
	{
		// Do nothing
	}
	@Override
	public void Login()
	{
```

```
		// Do nothing
}
@Override
public void IncorrectLogin()
{
		// Do nothing
}
@Override
public void IncorectPin(int max)
{
		// Do nothing
}
@Override
public void CorrectPinBelowMin()
{
		// Do nothing
}
@Override
public void CorrectPinAboveMin()
{
		// Do nothing
}
@Override
public void Deposit()
{
		// Do nothing
}
@Override
public void BelowMinBalance()
{
		// Do nothing
}
@Override
public void AboveMinBalance()
{
		// Do nothing
}
@Override
public void Logout()
{
		// Do nothing
}
@Override
public void Balance()
{
		// Do nothing
}
@Override
public void Withdraw()
{
		// Do nothing
```

```
	}
	@Override
	public void WithdrawBelowMinBalance()
	{
		// Do nothing
	}
	@Override
	public void NoFunds()
	{
		// Do nothing
	}
	@Override
	public void Lock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectLock(){
		// Do nothing
	}
	@Override
	public void Unlock()
	{
		// Do nothing
	}
	@Override
	public void IncorrectUnlock()
	{
		// Do nothing
	}
	@Override
	public void Suspend()
	{
		// Do nothing
	}
	@Override
	public void Activate()
	{
		// Do nothing
	}
	@Override
	public void Close()
	{
		// Do nothing
	}
}
```

// SuspendedState.java

```
package mda_States;

import mda_Efsm.Efsm;

public class SuspendedState extends StateSuper {

	public SuspendedState(Efsm ef){
		this.mda = ef;
		id = 8;
	}

	@Override
	public void Open()
	{
		// Do nothing
	}
	@Override
	public void Login()
	{
		// Do nothing
	}
	@Override
	public void IncorrectLogin()
	{
		// Do nothing
	}
	@Override
	public void IncorectPin(int max)
	{
		// Do nothing
	}
	@Override
	public void CorrectPinBelowMin()
	{
		// Do nothing
	}
	@Override
	public void CorrectPinAboveMin()
	{

	}
	@Override
	public void Deposit()
	{

	}
	@Override
	public void BelowMinBalance()
	{

	}
```

```
@Override
public void AboveMinBalance()
{
	// Do nothing
}
@Override
public void Logout()
{
	// Do nothing
}
@Override
public void Balance()
{
	// Calls DisplayBalance to do output action
	output.DisplayBalance(); // display available balance
}
@Override
public void Withdraw()
{
	// Do nothing
}
@Override
public void WithdrawBelowMinBalance()
{
	// Do nothing
}
@Override
public void NoFunds()
{
	// Do nothing
}
@Override
public void Lock()
{
	// Do nothing
}
@Override
public void IncorrectLock(){
	// Do nothing
}
@Override
public void Unlock()
{
	// Do nothing
}
@Override
public void IncorrectUnlock()
{
	// Do nothing
}
@Override
```

```
	public void Suspend()
	{
		// Do nothing
	}
	@Override
	public void Activate()
	{
		// Do nothing
	}
	@Override
	public void Close()
	{
		// Do nothing
	}


}
```

STRATEGY PATTERN

// **Output.java**

```
package outputProcessor;

import abstractFactory.*;
import dataStore.DataStore;
import strategy.*;

public class Output {

	private DataStore data;
	private StoreData storedata;
	private IncorrectIdmsg incorrectid;
	private IncorrectPinmsg incorrectpin;
	private TooManyattempts tooattempts;
	private DisplayMenu dmenu;
	private MakeDeposit mdeposit;
	private DisplayBalance dbalance;
	private PromptPin promptpin;
	private MakeWithdraw mwithdraw;
	private Penalty penalty;
	private IncorrectLockmsg lock;
	private IncorrectUnlockmsg unlock;
	private NoFundsmsg nofunds;
	AbstractSuper factory =null;
	DataStore dataStore = null;

	public Output(AbstractSuper factory,DataStore dataStore)
	{
		// Initializes respective factory and datastore of ACCOUNT Components
    this.factory = factory;
```

```
    this.dataStore = dataStore;
}
        public void StoreData(){
                // Initializes StoreData object and Calls output action StoreData
                storedata = factory.getStoredata();
                storedata.StoreData(dataStore);
        }
        public void PromptPin(){
                // Initializes PromptPin object and Calls output action PromptPin
                promptpin = factory.getPromptpin();
                promptpin.PromptPin();
        }
        public void DisplayMenu(){
                // Initializes DisplayMenu object and Calls output action DisplayMenu
                dmenu = factory.getDmenu();
                dmenu.DisplayMenu();
        }
        public void IncorrectIdmsg() {
                // Initializes IncorrectIdmsg object and Calls output action IncorrectIdmsg
                incorrectid = factory.getIncorrectid();
                incorrectid.IncorrectIdmsg();
        }
        public void IncorrectPinmsg() {
                // Initializes IncorrectPinmsg object and Calls output action IncorrectPinmsg
                incorrectpin = factory.getIncorrectpin();
                incorrectpin.IncorrectPinmsg();
        }
        public void TooManyattempts() {
                // Initializes TooManyattempts object and Calls output action TooManyattempts
                tooattempts = factory.getTooattempts();
                tooattempts.TooManyattempts();
        }
        public void IncorrectLockmsg(){
                // Initializes IncorrectLockmsg object and Calls output action IncorrectLockmsg
                lock = factory.getLock();
                lock.IncorrectLockmsg();

        }
        public void IncorrectUnlockmsg(){
                // Initializes IncorrectLockmsg object and Calls output action IncorrectLockmsg
                unlock = factory.getUnlock();
                unlock.IncorrectUnlockmsg();
        }
        public void DisplayBalance(){
                // Initializes DisplayBalance object and Calls output action DisplayBalance
                dbalance = factory.getDbalance();
                dbalance.DisplayBalance(dataStore);
        }
        public void Penalty(){
                // Initializes Penalty object and Calls output action Penalty
                penalty = factory.getPenalty();
```

```
                penalty.Penalty(dataStore);
        }
        public void NoFundsmsg(){
                // Initializes NoFundsmsg object and Calls output action NoFundsmsg
                nofunds = factory.getNofunds();
                nofunds.NoFundsmsg();
        }
        public void MakeDeposit()
        {
                // Initializes MakeDeposit object and Calls output action MakeDeposit
                mdeposit = factory.getMdeposit();
                mdeposit.MakeDeposit(dataStore);
        }
        public void MakeWithdraw()
        {
                // Initializes MakeWithdraw object and Calls output action MakeWithdraw
                mwithdraw = factory.getMwithdraw();
                mwithdraw.MakeWithdraw(dataStore);
        }
}
```

// DisplayBalance.java

```
package strategy;
import dataStore.DataStore;

abstract public class DisplayBalance {
        public void DisplayBalance(DataStore ds){
        }
}
```

// DisplayBalance1.java

```
package strategy;

import dataStore.DataStore;

public class DisplayBalance1 extends DisplayBalance {
        public void DisplayBalance(DataStore ds){
                // Displays the current Balance of Account1
                System.out.println("ACCOUNT-1 CURRENT BALANCE IS "+ds.getA1bal());
        }
}
```

// DisplayBalance2.java

```
package strategy;

import dataStore.DataStore;

public class DisplayBalance2 extends DisplayBalance {
```

```
	public void DisplayBalance(DataStore ds){
		// Displays the current Balance of Account2
		System.out.println("ACCOUNT-2 CURRENT BALANCE IS "+ds.getA2bal());
	}
}
```

// DisplayMenu.java

```
package strategy;

abstract public class DisplayMenu {

	public void DisplayMenu()
	{
	}
}
```

// DisplayMenu1.java

```
package strategy;

public class DisplayMenu1 extends DisplayMenu{

	public void DisplayMenu(){
		System.out.println("TRANSCATION MENU");
		System.out.println("DEPOSIT");
		System.out.println("WITHDRAW");
		System.out.println("BALANCE");
		System.out.println("LOCK");

	}
}
```

// DisplayMenu2.java

```
package strategy;

public class DisplayMenu2 extends DisplayMenu {

	public void DisplayMenu(){
		System.out.println("TRANSCATION MENU");
		System.out.println("DEPOSIT");
		System.out.println("WITHDRAW");
		System.out.println("BALANCE");
		System.out.println("SUSPENDED");

	}
}
```

// IncorrectIdmsg.java

```
package strategy;

abstract public class IncorrectIdmsg {

	public void IncorrectIdmsg(){
```

```
        }
}
```
// IncorrectIdmsg1.java
```
package strategy;

public class IncorrectIdmsg1 extends IncorrectIdmsg {

public void IncorrectIdmsg(){
                //displays Incorrect User ID Message
                System.out.println("INCORRECT USERID MESSAGE");
        }
}
```
// IncorrectIdmsg2.java
```
package strategy;

public class IncorrectIdmsg2 extends IncorrectIdmsg {

public void IncorrectIdmsg(){
        //displays Incorrect User ID Message
                System.out.println("INCORRECT USERID MESSAGE");
        }

}
```
// IncorrectLockmsg.java
```
package strategy;

abstract public class IncorrectLockmsg {
        public void IncorrectLockmsg(){

        }
}
```
// IncorrectLockmsg1.java
```
package strategy;

public class IncorrectLockmsg1 extends IncorrectLockmsg {
        public void IncorrectLockmsg(){
                // displays Incorrect Lock Message
                System.out.println("Incorrect Lock Attempted");

        }
}
```
// Incorrectpinmsg.java
```
package strategy;

abstract public class IncorrectPinmsg {

        public void IncorrectPinmsg(){

        }
```

```
}
```
// Incorrectpinmsg1.java
```
package strategy;

public class IncorrectPinmsg1 extends IncorrectPinmsg{

	public void IncorrectPinmsg(){
		// displays Incorrect Pin Message
		System.out.println("INCORRECT PIN MESSAGE");
	}
}
```
// Incorrectpinmsg2.java
```
package strategy;

public class IncorrectPinmsg2 extends IncorrectPinmsg {

public void IncorrectPinmsg(){
	// displays Incorrect Pin Message
		System.out.println("INCORRECT PIN MESSAGE");
	}
}
```
// IncorrectUnlockmsg.java
```
package strategy;

abstract public class IncorrectUnlockmsg {
	public void IncorrectUnlockmsg(){

	}
}
```
// IncorrectUnlockmsg1.java
```
package strategy;

public class IncorrectUnlockmsg1 extends IncorrectUnlockmsg {
	public void IncorrectUnlockmsg(){
		// displays Incorrect Unlock Message
		System.out.println("Incorrect UnLock Attempted");

	}
}
```
// MakeDeposit.java
```
package strategy;

import dataStore.DataStore;

abstract public class MakeDeposit {
	public void MakeDeposit(DataStore ds)
	{

	}
```

```
}
```
// MakeDeposit1.java
```
package strategy;

import dataStore.DataStore;

public class MakeDeposit1 extends MakeDeposit{
	public void MakeDeposit(DataStore ds)
	{
		// adds deposit amount to the current balance and sets current balance again in DataStore1
		ds.setA1bal(ds.getA1bal()+ds.getA1temp_deposit());
	}
}
```
// MakeDeposit2.java
```
package strategy;

import dataStore.DataStore;

public class MakeDeposit2 extends MakeDeposit{
	public void MakeDeposit(DataStore ds)
	{
		// adds deposit amount to the current balance and sets current balance again in DataStore2
		ds.setA2bal(ds.getA2bal()+ ds.getA2temp_deposit());
	}
}
```
// MakeWithdraw.java
```
package strategy;

import dataStore.DataStore;

abstract public class MakeWithdraw {

	public void MakeWithdraw(DataStore ds)
	{

	}
}
```
// MakeWithdraw1.java
```
package strategy;

import dataStore.DataStore;

public class MakeWithdraw1 extends MakeWithdraw{

	public void MakeWithdraw(DataStore ds)
	{
		// deducts withdraw amount from the current balance and sets current balance again in
DataStore1
		ds.setA1bal(ds.getA1bal() - ds.getA1temp_withdraw());
	}
}
```

// MakeWithdraw2.java

```
package strategy;

import dataStore.DataStore;

public class MakeWithdraw2 extends MakeWithdraw {
	public void MakeWithdraw(DataStore ds)
	{
		// deducts withdraw amount from the current balance and sets current balance again in
DataStore2
		ds.setA2bal(ds.getA2bal() - ds.getA2temp_withdraw());
	}
}
```

// NoFundsmsg.java

```
package strategy;

abstract public class NoFundsmsg {
		public void NoFundsmsg(){

		}
}
```

// NoFundsmsg1.java

```
package strategy;

public class NoFundsmsg1 extends NoFundsmsg {
	public void NoFundsmsg(){
		System.out.println("NO FUNDS IN YOUR ACCOUNT");
	}
}
```

// Penalty.java

```
package strategy;

import dataStore.DataStore;

abstract public class Penalty {
	public void Penalty(DataStore ds){

	}

}
```

// Penalty1.java

```
package strategy;
```

```
import dataStore.DataStore;

public class Penalty1 extends Penalty {
	public void Penalty(DataStore ds){
		// set Penalty if withdrawal amount is lesser than minimum balance

		System.out.println(" Minimum required balance is $500. So Penalty is applied.");
		float bal = ds.getA1bal();
		float afterpenalty = bal - 20;
		ds.setA1bal(afterpenalty);
		System.out.println(" After a Penalty of 20$, Balance is "
		+ ds.getA1bal() );

	}

}
```

// PromptPin.java

```
package strategy;

import dataStore.DataStore;

abstract public class PromptPin {
	public void PromptPin(){

	}
}
```

// PromptPin1.java

```
package strategy;

public class PromptPin1 extends PromptPin {
	public void PromptPin()
	{
		// displays PromptPin Message for account1
		System.out.println("Enter the Pin:" );
	}

}
```

// PromptPin2.java

```
package strategy;

public class PromptPin2 extends PromptPin {
	public void PromptPin()
	{
		// displays PromptPin Message for account2
		System.out.println("Enter the Pin: " );
	}

}
```

// StoreData.java

```
package strategy;

import dataStore.DataStore;
import dataStore.DataStore.*;

abstract public class StoreData {
	public void StoreData(DataStore ds){

	}
}
```

// StoreData1.java

```
package strategy;

import dataStore.DataStore;
import dataStore.DataStore.*;

public class StoreData1 extends StoreData{

	@Override
	public void StoreData(DataStore ds){
		// sets pin,userid, balance from temporary to permanent variables in DataStore1
		ds.setA1pin(ds.getA1temp_pin());
		ds.setA1userid(ds.getA1temp_userid());
		ds.setA1bal(ds.getA1temp_bal());
	}

}
```

// StoreData2.java

```
package strategy;

import dataStore.DataStore;

public class StoreData2 extends StoreData {

	@Override
	public void StoreData(DataStore ds){
		// sets pin,userid, balance from temporary to permanent variables in DataStore2
		ds.setA2pin(ds.getA2temp_pin());
		ds.setA2userid(ds.getA2temp_userid());
		ds.setA2bal(ds.getA2temp_bal());
		//String user = ds.getUserid();
	}
}
```

// TooManyattempts.java

```
package strategy;
```

```
abstract public class TooManyattempts {
	public void TooManyattempts(){

	}
}
```
// TooManyattempts1.java
```
package strategy;

public class TooManyattempts1 extends TooManyattempts{
	public void TooManyattempts(){
		// displays  TooManyattempts Message
		System.out.println("Too Many times Attempted");
	}

}
```
// TooManyattempts2.java
```
package strategy;

public class TooManyattempts2 extends TooManyattempts{
	public void TooManyattempts(){
		// displays  TooManyattempts Message
		System.out.println("Too Many times Attempted");
	}

}
```
INPUT PROCESSOR:

// ACCOUNT-1.java
```
package inputProcessor;
import mda_Efsm.Efsm;
import dataStore.DataStore;
import dataStore.DataStore1;
import abstractFactory.AbstractSuper;
import abstractFactory.ConcreteFactory1;

public class Account_1 {

	private AbstractSuper af;
	private Efsm mda;
	private DataStore ds;

	public Account_1(Efsm mda , AbstractSuper factory , DataStore ds)
	{
		// Creates pointer to objects of Efsm, AbstractSuper , DataStore
		this.af = factory;
		this.mda = mda;
		this.ds = ds ;
	}
```

```
	public void open (String pin , String user_id , float bal)
	{
		// store p, y and a in temp data store

		ds.setA1temp_pin(pin);
		ds.setA1temp_userid(user_id);
		ds.setA1temp_bal(bal);
		mda.Open();
	}

	public void pin(String x)
	{
		// checks the pin value and calls events based on minimum balance criterion
		 String value = ds.getA1pin();
		 if (x.equals(value))
		{
		 if (ds.getA1bal() > 500)
			mda.CorrectPinAboveMin ();
		 else
			mda.CorrectPinBelowMin();
		}
		 else
		 {
			mda.IncorectPin(3);
		 }
	}


	public void deposit (float d)
	{
		// Stores the deposit user input in DataStore and calls events based on min balance
criterion
		 ds.setA1temp_deposit(d);
		 mda.Deposit();
		 if (ds.getA1bal()>500)
			mda.AboveMinBalance();
		 else
			mda.BelowMinBalance();
	}
	public void withdraw (float w)
	{
		// stores withdraw amt in Datastore and calls respective events
		ds.setA1temp_withdraw(w);
		mda.Withdraw();
		 if ((ds.getA1bal()>500))
				mda.AboveMinBalance();
		 else
			mda.WithdrawBelowMinBalance();
	}
	public void balance()
	{
```

```
                // calls Balance event
                mda.Balance();
        }

        public void login (String y)
        {
                // gets permanent userid and checks it with user input. Calls respective events on
condition
                 if (y.equals(ds.getA1userid())){
                        mda.Login();}
                 else
                        mda.IncorrectLogin();
        }
        public void logout()
        {
                // Calls Logout event
                mda.Logout();
        }
        public void lock (String x)
        {
                // Checks the entered pin with permanent pin in DataStore and calls events.
                 if (ds.getA1pin().equals(x))
                        mda.Lock();
                 else
                        mda.IncorrectLock();
        }
        public void unlock (String x)
        {
                // Checks the entered pin with permanent pin in DataStore and calls events.
                 if (x.equals(ds.getA1pin()))
                 {
                        mda.Unlock();
                 if (ds.getA1bal() > 500)
                        mda.AboveMinBalance ();
                 else
                        mda.BelowMinBalance();
                 }
                 else mda.IncorrectUnlock();
        }

}
```

// ACCOUNT-2.java

```
package inputProcessor;

import abstractFactory.AbstractSuper;
import dataStore.DataStore;
import dataStore.DataStore2;
import mda_Efsm.Efsm;

public class Account_2 extends AbstractSuper{
```

```
	private AbstractSuper af;
	private Efsm mda;
	private DataStore ds;

	public Account_2(Efsm mda , AbstractSuper factory , DataStore ds2)
	{
		this.af = factory;
		this.mda = mda;
		this.ds = ds2 ;
	}
	public void OPEN (int p, int y, int a) {
		// store p, y and a in temp data store
		ds.setA2temp_pin(p);
		ds.setA2temp_userid(y);
		ds.setA2temp_bal(a);
		mda.Open();
		}
	public void PIN (int x)
	{
		// checks the pin value and calls events based on minimum balance criterion
		 if (x == ds.getA2pin())
			mda.CorrectPinAboveMin ();
		 else mda.IncorectPin(2);
	}
	public void DEPOSIT (int d)
	{
		// Stores the deposit user input in DataStore and calls events based on min balance
criterion
		 ds.setA2temp_deposit(d);
		 mda.Deposit();
	}
	public void WITHDRAW (int w)
	{
		// stores withdraw amt in Datastore and calls respective events
		 ds.setA2temp_withdraw(w);
		 if (ds.getA2bal() > 0){
			mda.Withdraw();
			mda.AboveMinBalance();}
		 else
			mda.NoFunds();
	}
	public void BALANCE()
	{
		// calls Balance event
		mda.Balance();
	}
	public void LOGIN (int y)
	{
		// gets permanent userid and checks it with user input. Calls respective events on
condition
```

```
                if (y == ds.getA2userid())
                        mda.Login();
                else
                        mda.IncorrectLogin();
        }
        public void LOGOUT()
        {
                mda.Logout();
        }
        public void suspend ()
        {
                mda.Suspend();
        }
        public void       activate ()
        {
                mda.Activate();
        }
        public void close ()
        {
                mda.Close();
        }
}
```

MAIN DRIVER

// Driver.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Scanner;

import abstractFactory.ConcreteFactory1;
import abstractFactory.ConcreteFactory2;
import inputProcessor.Account_1;
import inputProcessor.Account_2;
import mda_Efsm.Efsm;
import mda_States.StateSuper;
import outputProcessor.Output;

public class Driver {

        public static void main(String[] args){
                        int input;
                        System.out.println("Model Driven Architecture");
                                Scanner in = new Scanner(System.in);
                                System.out.println("Select the Account Components");
                                System.out.println("1. ACCOUNT-1");
                                System.out.println("2. ACCOUNT-2");
                                input = in.nextInt();
                                if( input == 1){
                                        System.out.println("You selected ACCOUNT -1");
```

```
                                        ConcreteFactory1 factory = new ConcreteFactory1();
                                        Output output = new Output(factory,factory.GetDataStore());
                                        StateSuper states = new StateSuper();
                                        states.initializeOutput(output);
                                        Efsm mda = new Efsm( );
                                        Account_1 a1 = new Account_1(mda,
factory,factory.GetDataStore());
                                        account1_operations(a1);
                                }
                                else if(input == 2){
                                        System.out.println("You selected ACCOUNT -2");
                                        ConcreteFactory2 factory = new ConcreteFactory2();
                                        Output output = new Output(factory,factory.GetDataStore());
                                        StateSuper states = new StateSuper();
                                        states.initializeOutput(output);
                                        Efsm mda = new Efsm( );
                                        Account_2 a2 = new Account_2(mda,
factory,factory.GetDataStore());
                                        account2_operations(a2);
                                }
                                else{
                                        System.out.println("Enter valid option");
                                }
                }

                public static void account1_operations(Account_1 a1){
                        String inputPin , inputUserid;
                        float inputBal, inputdeposit, inputwithdraw;
                        int userInput =0 ;
                        System.out.println("\n");
                        System.out.println("ACCOUNT-1");
                        System.out.println("MENU OF OPERATIONS");
                        System.out.println("1. open(String p,string y , float a)");
                        System.out.println("2. login (string y)");
                        System.out.println("3. pin(string x)");
                        System.out.println("4. deposit(float d)");
                        System.out.println("5. withdraw (float w)");
                        System.out.println("6. balance()");
                        System.out.println("7. lock(string x)");
                        System.out.println("8. unlock(string x)");
                        System.out.println("9. logout( )");
                        System.out.println(" 10. Quit the demo program" );
                        System.out.println(" Please make a note of these operations" );
                        System.out.println("ACCOUNT-1 Execution");
                        do{
                                System.out.println("Select Operation:");
                                System.out.println(" 1-open , 2-login, 3-pin, 4-deposit, 5-withdraw, 6-
balance, 7-lock"
                                                + " 8-unlock, 9-logout, 10-Quit");
                                Scanner sc1 = new Scanner(System.in);
                                userInput=sc1.nextInt();
```

```
                    if(userInput <= 10 )
                    {
                    Scanner sc2 = new Scanner(System.in);
                    Scanner sc3 = new Scanner(System.in);
                    Scanner sc4 = new Scanner(System.in);
                    switch (userInput) {
                    case 1: System.out.println("OPERATION open(String, String, float):");
                    System.out.println("Enter the value of parameter p (pin)");
                    inputPin = sc2.nextLine();
                    System.out.println("Enter the value of parameter y (user_id)");
                    inputUserid = sc3.nextLine();
                    System.out.println("Enter the value of parameter a (balance)");
                    inputBal = sc3.nextFloat();
                    a1.open(inputPin, inputUserid, inputBal);
                    break;
                    case 2: System.out.println("OPERATION login(String):");
                    System.out.println("Enter the value of parameter y (user_id)");
                    inputUserid = sc3.nextLine();
                    a1.login(inputUserid);
                    break;
                    case 3: System.out.println("OPERATION pin(String):");
                    System.out.println("Enter the value of parameter x (pin)");
                    inputPin = sc3.nextLine();
                    a1.pin(inputPin);
                    break;
                    case 4: System.out.println("OPERATION deposit(float):");
                    System.out.println("Enter the value of parameter d (deposit amount)");
                    inputdeposit = sc3.nextFloat();
                    a1.deposit(inputdeposit);
                    break;
                    case 5: System.out.println("OPERATION withdraw(float):");
                    System.out.println("Enter the value of parameter w (withdrawal
amount)");
                    inputwithdraw = sc3.nextFloat();
                    a1.withdraw(inputwithdraw);
                    break;
                    case 6: System.out.println("OPERATION balance():");
                    a1.balance();
                    break;
                    case 7: System.out.println("OPERATION lock(String):");
                    System.out.println("Enter the value of parameter x (pin)");
                    inputPin = sc3.nextLine();
                    a1.lock(inputPin);
                    break;
                    case 8: System.out.println("OPERATION unlock(String):");
                    System.out.println("Enter the value of parameter x (pin)");
                    inputPin = sc3.nextLine();
                    a1.unlock(inputPin);
                    break;
                    case 9: System.out.println("OPERATION logout():");
                    a1.logout();
```

```
                                        break;
                                        case 10: System.out.println("OPERATION quit()");
                                        System.exit(0);
                                        default: System.out.println("Operation not permitted in this state");
                                        break;
                                        }
                                        }
                                        else
                                        {
                                                System.out.println("ENTER VALID INPUT OPERTION");
                                                account1_operations(a1);
                                        }
                        }while(( userInput > 0 )&&( userInput < 11 ));
        }
                        public static void account2_operations(Account_2 a2){
                                int inputPin , inputUserid, inputBal, inputdeposit, inputwithdraw;
                                int userInput =0 ;
                                System.out.println("\n");
                                System.out.println("ACCOUNT-2");
                                System.out.println("MENU OF OPERATIONS");
                                System.out.println("0. OPEN(int p,int y , int a)");
                                System.out.println("1. LOGIN(int y)");
                                System.out.println("2. PIN (int x)");
                                System.out.println("3. DEPOSIT(int d)");
                                System.out.println("4. WITHDRAW (int w)");
                                System.out.println("5. BALANCE()");
                                System.out.println("6. LOGOUT()");
                                System.out.println("7. suspend()");
                                System.out.println("8. activate() " );
                                System.out.println("9. close() " );
                                System.out.println("10. Quit the Program");
                                System.out.println(" Please make a note of these operations" );
                                System.out.println("ACCOUNT-2 Execution");
                                do{
                                        System.out.println("Select Operation:");
                                        System.out.println(" 0-OPEN , 1-LOGIN, 2-PIN, 3-DEPOSIT, 4-
WITHDRAW, 5-BALANCE, 6-LOGOUT, 7-suspend"
                                                        + " 8-activate, 9-close, 10-Quit");
                                        Scanner sc1 = new Scanner(System.in);
                                        userInput=sc1.nextInt();
                                        if(userInput <= 10 )
                                        {
                                        Scanner sc2 = new Scanner(System.in);
                                        Scanner sc3 = new Scanner(System.in);
                                        Scanner sc4 = new Scanner(System.in);
                                        switch (userInput) {
                                        case 0: System.out.println("OPERATION OPEN(int, int, int):");
                                        System.out.println("Enter the value of parameter p (pin)");
                                        inputPin = sc2.nextInt();
                                        System.out.println("Enter the value of parameter y (user_id)");
                                        inputUserid = sc3.nextInt();
```

```
System.out.println("Enter the value of parameter a (balance)");
inputBal = sc3.nextInt();
a2.OPEN(inputPin, inputUserid, inputBal);
break;
case 1: System.out.println("OPERATION LOGIN(int):");
System.out.println("Enter the value of parameter y (user_id)");
inputUserid = sc3.nextInt();
a2.LOGIN(inputUserid);
break;
case 2: System.out.println("OPERATION PIN(int):");
System.out.println("Enter the value of parameter x (pin)");
inputPin = sc3.nextInt();
a2.PIN(inputPin);
break;
case 3: System.out.println("OPERATION DEPOSIT(int):");
System.out.println("Enter the value of parameter d (deposit amount)");
inputdeposit = sc3.nextInt();
a2.DEPOSIT(inputdeposit);
break;
case 4: System.out.println("OPERATION WITHDRAW(INT):");
System.out.println("Enter the value of parameter w (withdrawal
amount)");
inputwithdraw = sc3.nextInt();
a2.WITHDRAW(inputwithdraw);
break;
case 5: System.out.println("OPERATION BALANCE():");
a2.BALANCE();
break;
case 6: System.out.println("OPERATION LOGOUT():");
a2.LOGOUT();
break;
case 7: System.out.println("OPERATION suspend():");
a2.suspend();
break;
case 8: System.out.println("OPERATION activate():");
a2.activate();
break;
case 9: System.out.println("OPERATION close():");
a2.close();
break;
case 10: System.out.println("OPERATION quit()");
System.exit(0);
default: System.out.println("Enter Valid Input Operation");
break;
}
}
else
{
        System.out.println("ENTER VALID INPUT OPERTION");
        account2_operations(a2);
}
```

```
        }while(( userInput >= 0 )&&( userInput < 11 ));
    }
}
```