CS-586 PROJECT

CHETHAN KUMAR APPAJI
[A20355296]
cappaji@hawk.iit.edu

CONTENTS

1. Introduction

2. MDA Architecture

3. Class Diagram

4. Sequence Diagram

5. Design Patterns

6. Source Code

INTRODUCTION

The Project is the simple design implementation of an Account System consisting of different Account components. Model Driven Architecture (MDA) has been used to the design the system. First step of the implementation was carried out by designing the MDA-EFSM model where the Meta-Events and Meta-Actions were identified based on the behavior of different Account components.

Account system contains two different account components ACCOUNT-1 and ACCOUNT-2 and these are state-based components and support three types of transactions: withdrawal, deposit, and balance inquiry. Along with the operations such as login, pin, logout, lock, unlock, activate, suspend and close.

The meta-model implementation of the Account unit adapts the following three OO design patterns:

1. State Pattern
   In State pattern a class behavior changes based on its state. The objects created represents various states and a context object behavior varies as its state object changes.

2. Strategy Pattern
   In Strategy pattern a class behavior or its algorithm can be changed at run time. The objects created represents various strategies and a context object behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

3. Abstract factory Pattern
   Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

MDA ARCHITECTURE

Two different Account components exhibit different functionalities, Aspects that vary between two ACCOUNT components are:

a. Maximum number of times incorrect pin can be entered
b. Minimum balance
c. Display menu(s)
d. Messages, e.g., error messages, etc.
e. Penalties
f. Operation names and signatures
g. Data types

MDA Architecture Model

The general MDA architecture model for Account system is as shown below.

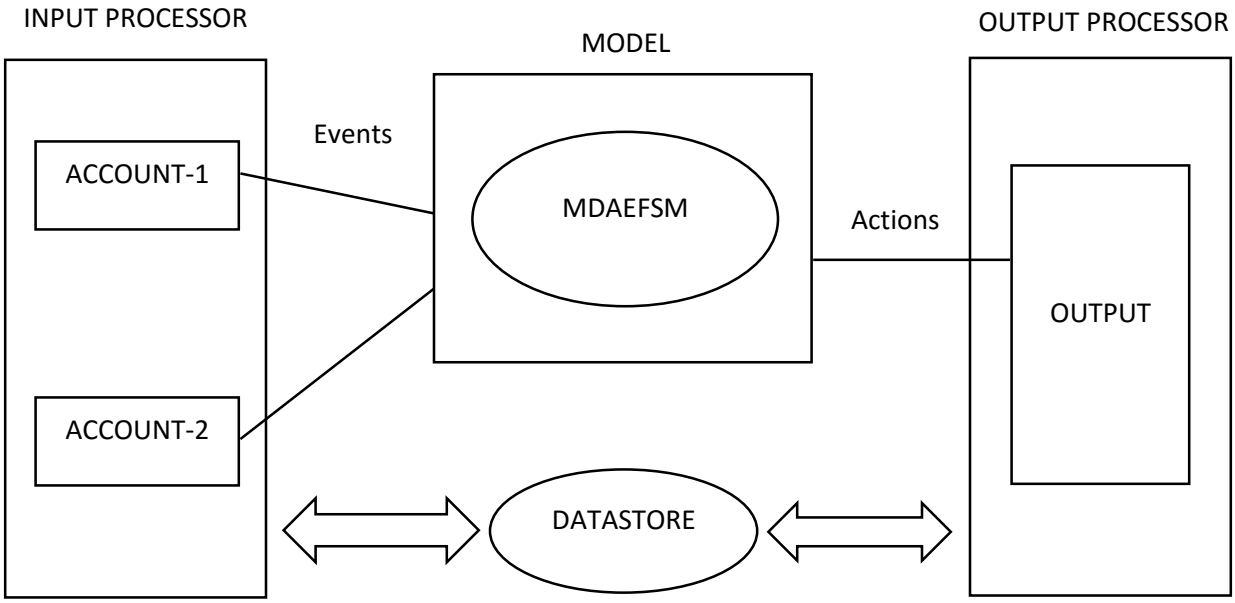


Figure 1: General MDA Architecture

Elements of the MDA architecture: The input processor has two accounts with distinguishing characters, along with a Data Store that is shared between the input and the output processor. While the MDA communicates between the input and the output.

Relationships between MDA elements

- Input processor, MDA and Output processor are related by aggregation.
- MDA elements are related using aggregation with the Output processor.
- The subclasses of classes above are related to them through inheritance.

MDA-EFSM Model

MDA-EFSM Events:

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

MDA-EFSM Actions:

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 msg
A12: IncorrectUnlock Msg() // displays incorrect unlock msg
A13: NoFundsMsg() // Displays no sufficient funds msg

Pseudo-code of all operations of Input Processors of Account 1 & 2

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();
}
```

Notice:
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

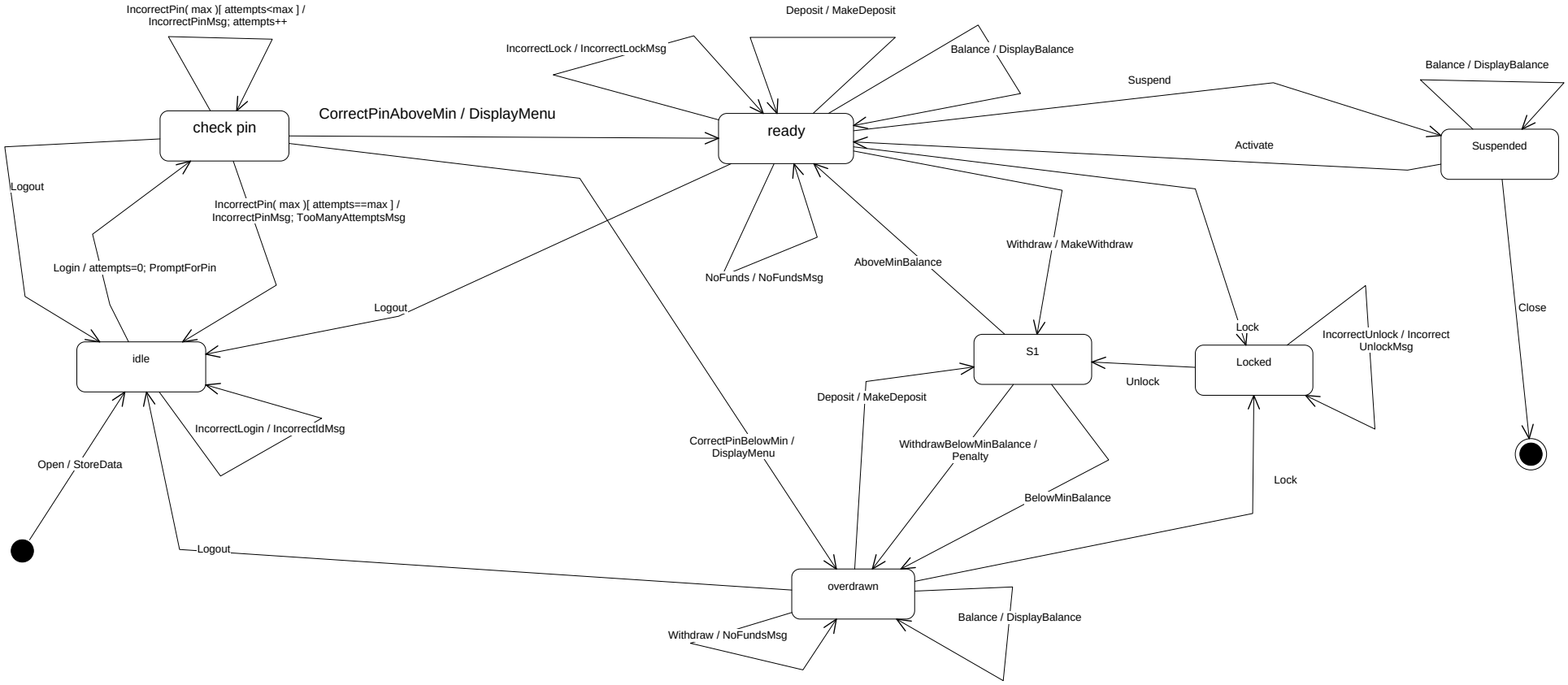
Operations of the Input Processor (ACCOUNT-2)

```
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();
}

LOGIN (int y) {
 if (y==ds->uid)
 m->Login();
 else m->IncorrectLogin();

 LOGOUT() {m->Logout();}

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

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

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

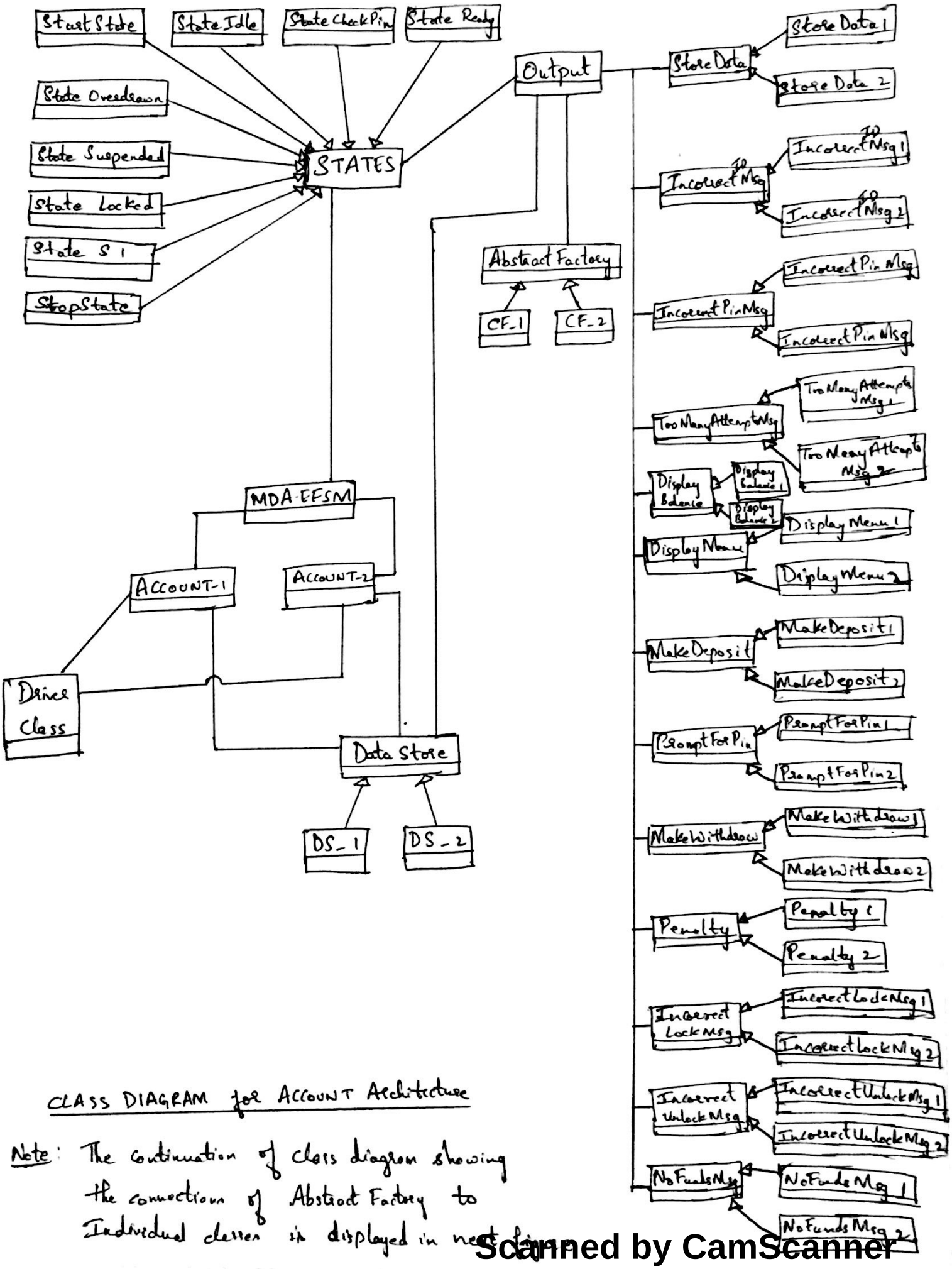
Notice:
m: is a pointer to the MDA-EFSM object
ds: is a pointer to the Data Store object which contains the following data items:

- 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

MDA-EFSM for ACCOUNT:

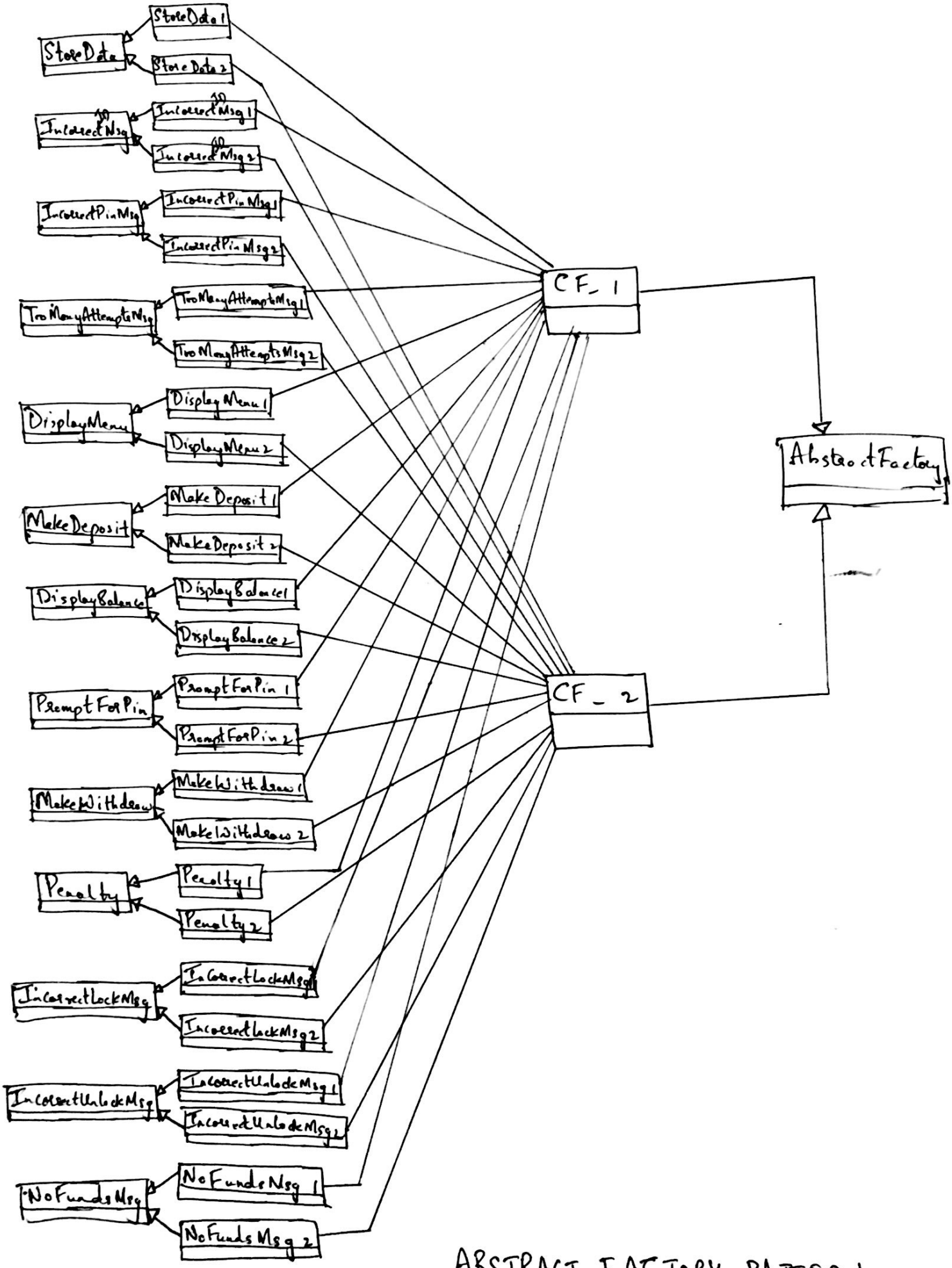

IncorrectPin(max)[attempts<max] /
IncorrectPinMsg; attempts++
Deposit / MakeDeposit
IncorrectLock / IncorrectLockMsg
Balance / DisplayBalance
Balance / DisplayBalance
Suspend
check pin
CorrectPinAboveMin / DisplayMenu
ready
Activate
Suspended
Logout
IncorrectPin(max)[attempts==max] /
IncorrectPinMsg; TooManyAttemptsMsg
Withdraw / MakeWithdraw
Login / attempts=0; PromptForPin
NoFunds / NoFundsMsg
AboveMinBalance
Close
Logout
Lock
IncorrectUnlock / Incorrect
UnlockMsg
idle
S1
Locked
Unlock
Deposit / MakeDeposit
IncorrectLogin / IncorrectIdMsg
CorrectPinBelowMin /
DisplayMenu
WithdrawBelowMinBalance /
Penalty
Open / StoreData
Lock
BelowMinBalance
Logout
overdrawn
Balance / DisplayBalance
Withdraw / NoFundsMsg

3. CLASS DIAGRAM



CLASS DIAGRAM for ACCOUNT Architecture

Note: The continuation of class diagram showing the connections of Abstract Factory to Individual classes is displayed in next figure.



ABSTRACT FACTORY PATTERN

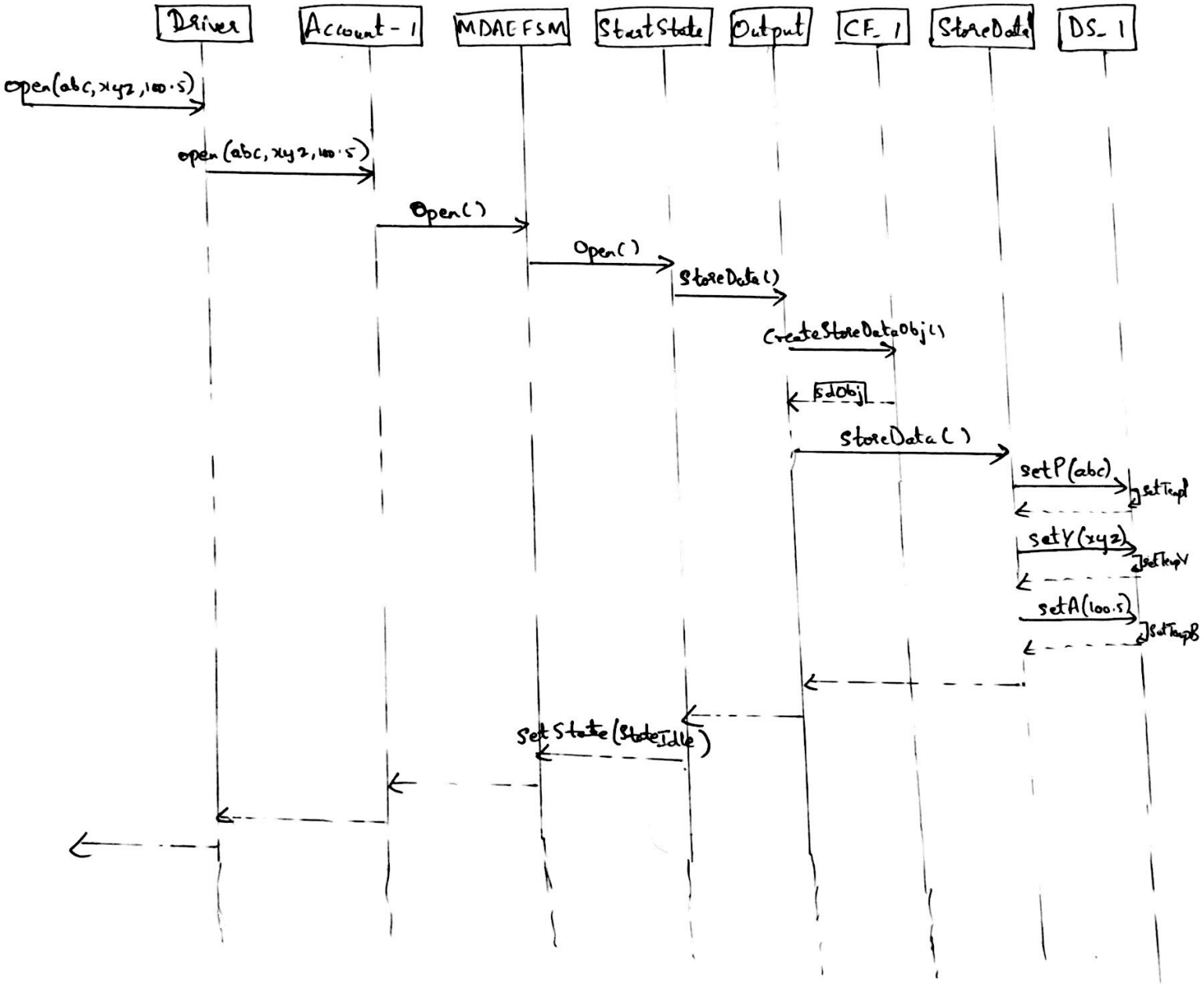
- Abstract Factory: This class consists of all overridden methods in Concrete factory
- CF_1 : Consists of all implementation methods of ACCOUNT-1
- CF_2: Consists of all implementation methods of ACCOUNT-2
- DS: This class has all variables value as abstract
- DS_1: This class has the variable and overridden methods pertaining to ACCOUNT-1
- DS_2: This class has the variable and overridden methods pertaining to ACCOUNT-2
- ACCOUNT-1: Implementation of ACCOUNT-1 functionalities and methods
- ACCOUNT-2: Implementation of ACCOUNT-2 functionalities and methods
- MDAEFSM: Implementation Next states its transition
- States: Interface States Consists of all the signature of implementation methods
- StartState: First State has its implementation of Open method
- StateIdle: StateIdle has the implementation of Login & IncorrectLogin methods
- StateCheckPin: StateCheckPin has the implementation of IncorrectPin, CorrectPinAboveMin, CorrectPinBelowMin & Logout methods
- StateReady: StateReady has the implementation of Deposit, Withdraw, Balance, NoFunds, Lock, IncorrectLock, Suspend, close & Logout methods
- StateOverdrawn: StateOverdrawn has the implementation of Deposit, WithdrawBelowMinBalance, Balance, Lock, IncorrectLock & Logout methods
- StateSuspended: StateSuspended has the implementation of Balance, Activate & Close methods
- StateLocked: StateLocked has the implementation of Unlock & IncorrectUnlock methods
- StateS1: StateS1 has the implementation of BelowMinBalance, AboveMinBalance & WithdrawBelowMinBalance methods
- StopState: Ends transaction.
- Output: OutPut Processor Consists link with Abstract Factory and DataStore(DS)
- StoreData: Stores Data onto the Datastore
- StoreData1
- StoreData2
- IncorrectIDMsg: Displays IncorrectID Message
- IncorrectIDMsg1
- IncorrectIDMsg2
- IncorrectPin: Displays IncorrectPin Message
- IncorrectPin1
- IncorrectPin2
- TooManyAttemptsMsg: Displays TooManyAttempts Message for maximum number of Pin try.
- TooManyAttemptsMsg1
- TooManyAttemptsMsg2
- DisplayBalance: Displays current Bamnce in the Account
- DisplayBalance1
- DisplayBalance2
- DisplayMenu: Displays menu option of transactions available for account
- DisplayMenu1
- DisplayMenu2
- MakeDeposit: Update the balance in the account by adding deposit amount
- MakeDeposit1

- MakeDeposit2
- MakeWithdraw: Update the balance in the account by deducting withdraw amount
- MakeWithdraw1
- MakeWithdraw2
- PromptForPin: Displays P message when login details is succesfully entered
- PromptForPin1
- PromptForPin2
- Penalty: Update the balance amount by deducting the penalty amount along with withdraw amount when withdraw is made on below minimum balance
- Penalty1
- Penalty2
- IncorrectLockMsg: Displays IncorrectLock Message
- IncorrectLockMsg1
- IncorrectLockMsg2
- IncorrectunlockMsg: Displays IncorrectUnlock Message
- IncorrectUnlockMsg1
- IncorrectUnlockMsg2
- NoFundsMsg: Displays NoFunds meassage when there is no balance available in the account
- NoFundsMsg1
- NoFundsMsg2

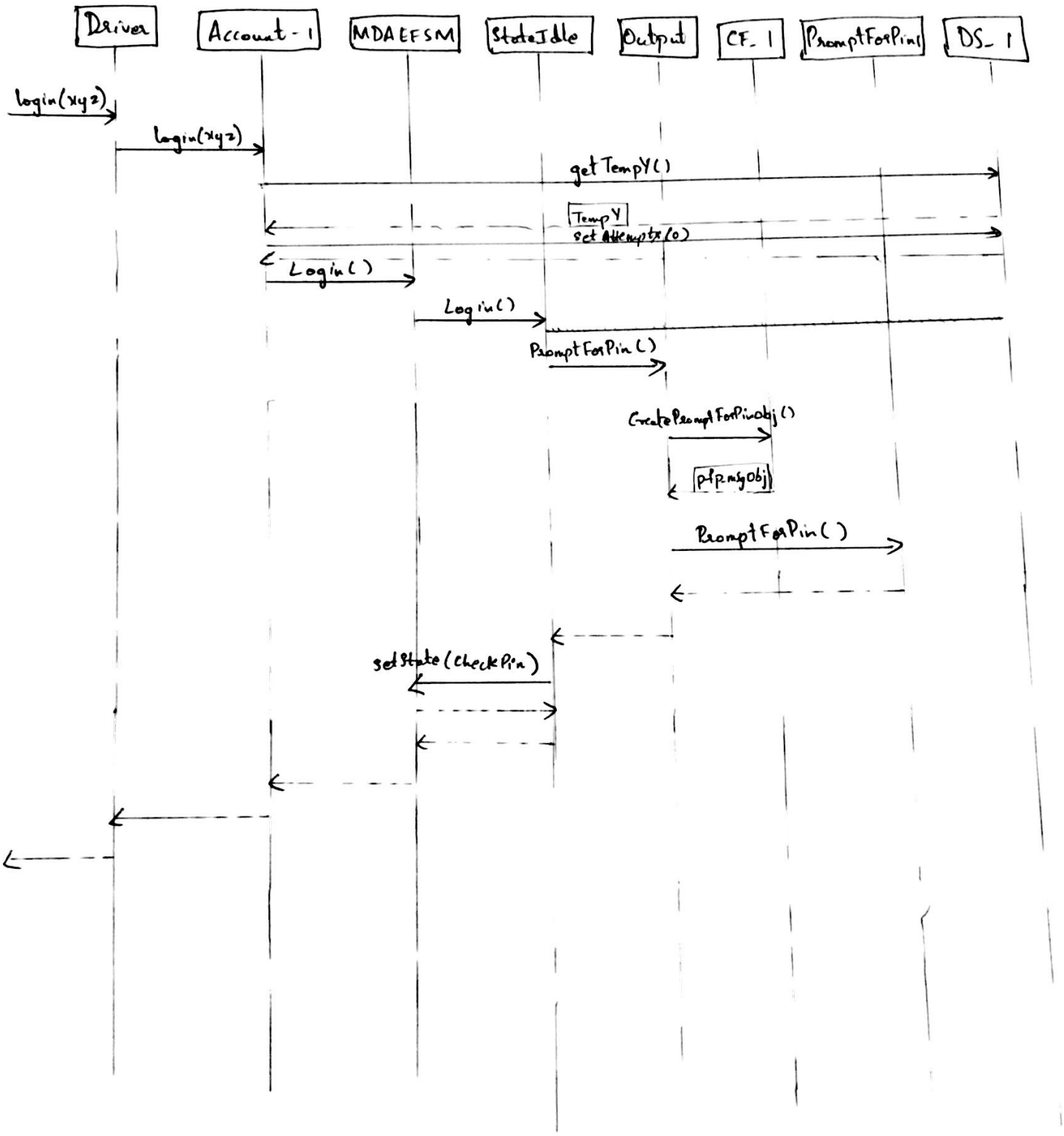
4. SEQUENCE DIAGRAM

a. Scenario - 1

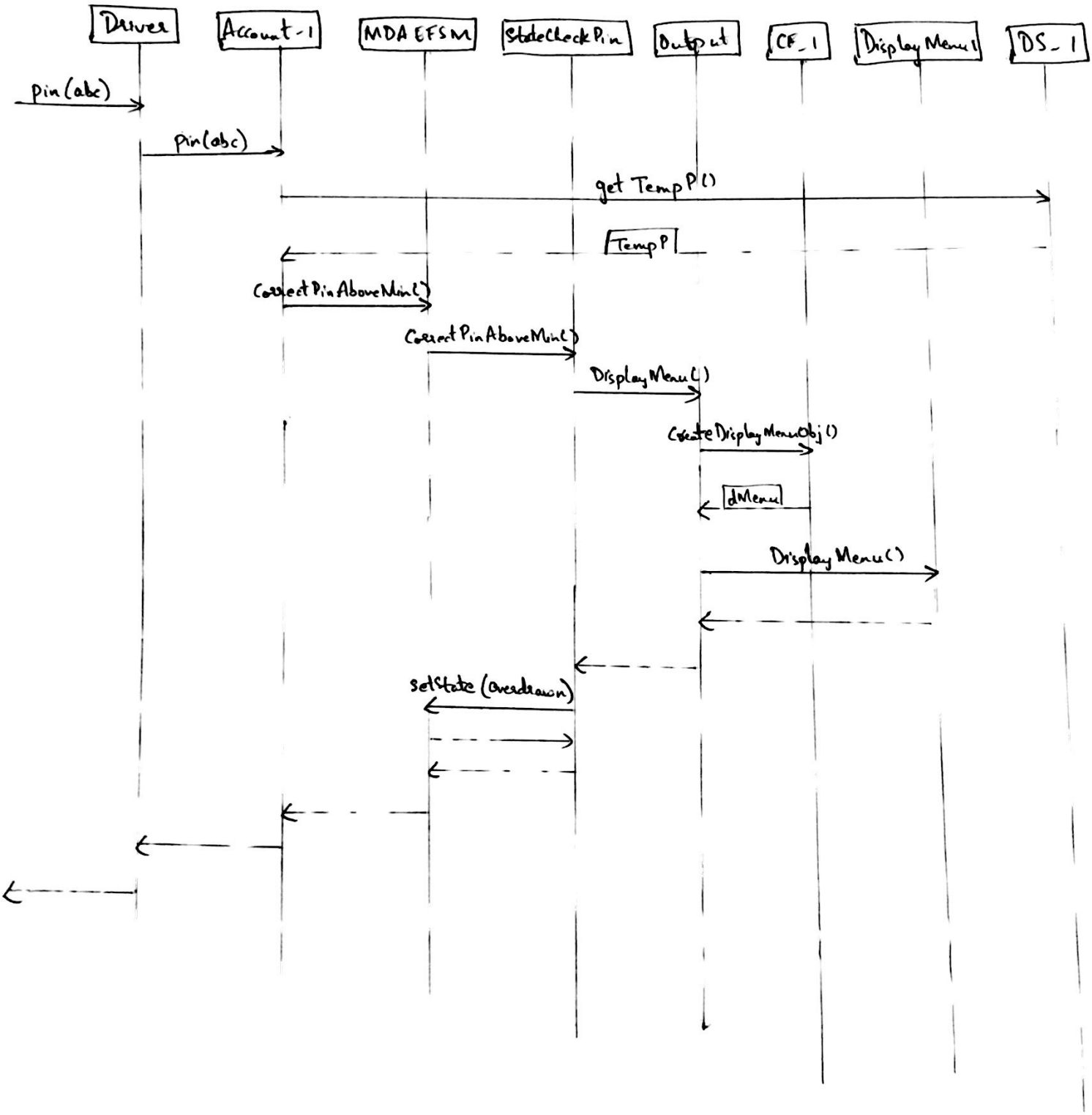
open(abc, xyz, 100.5)

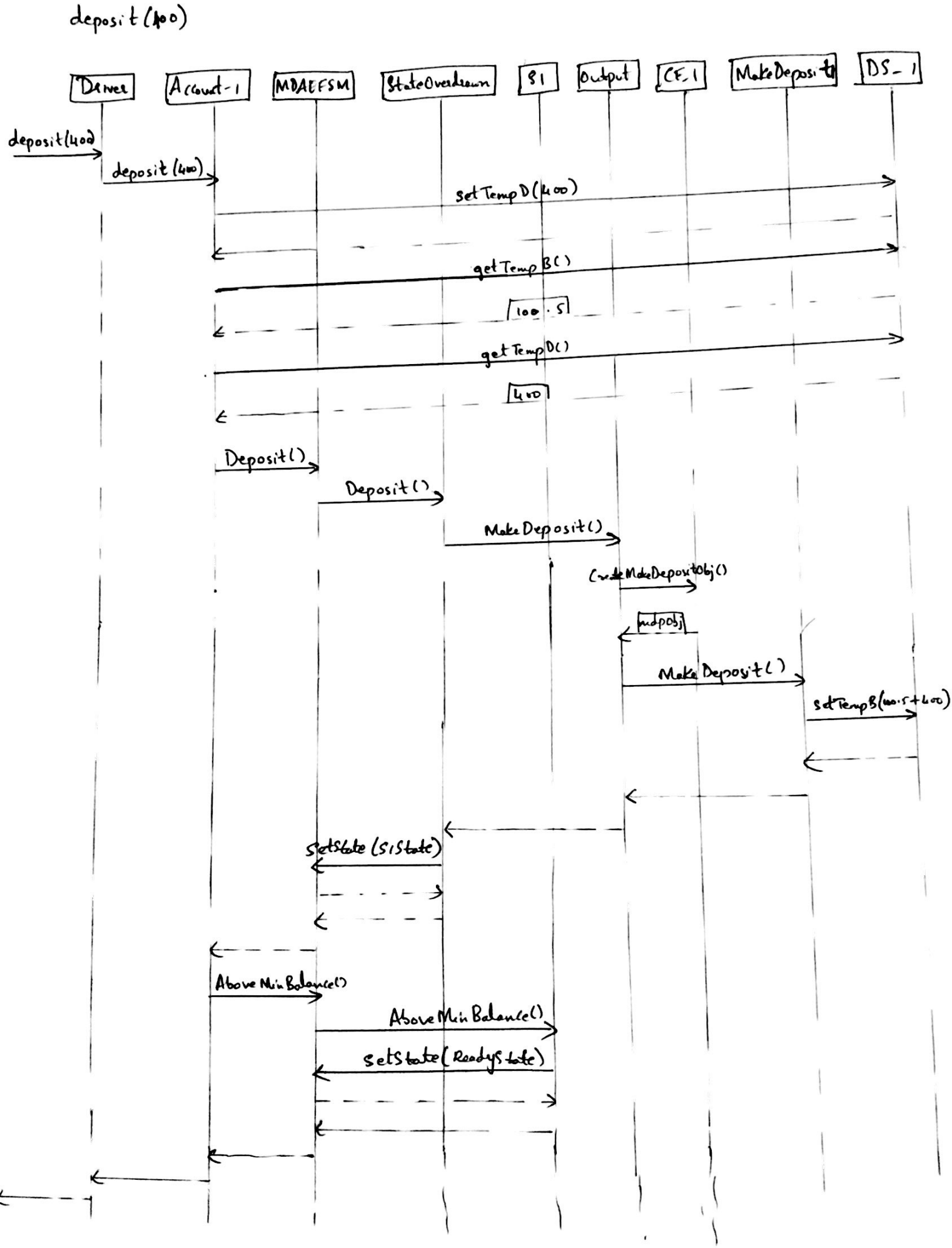


login (xyz)

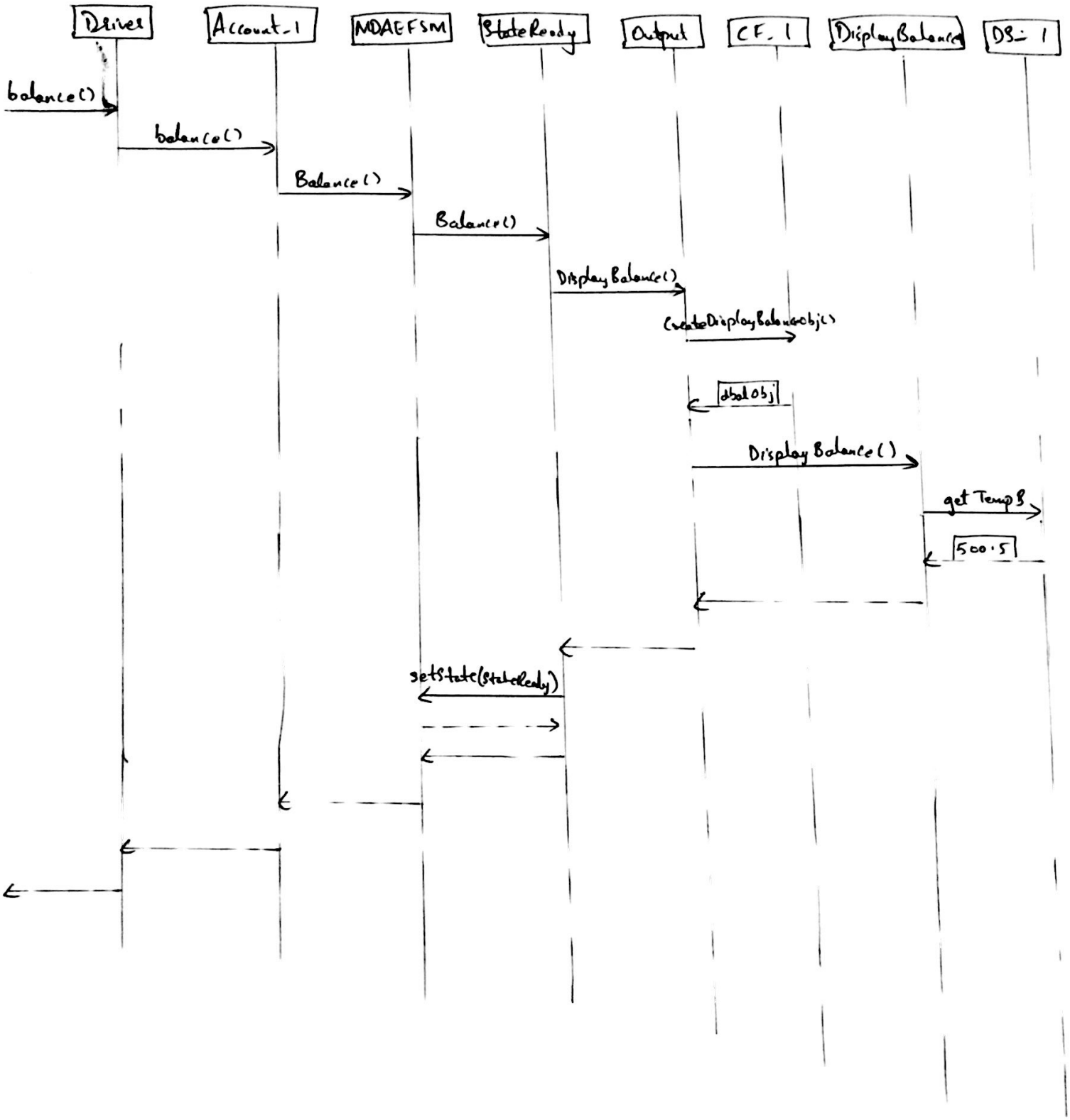


Pin (abc)

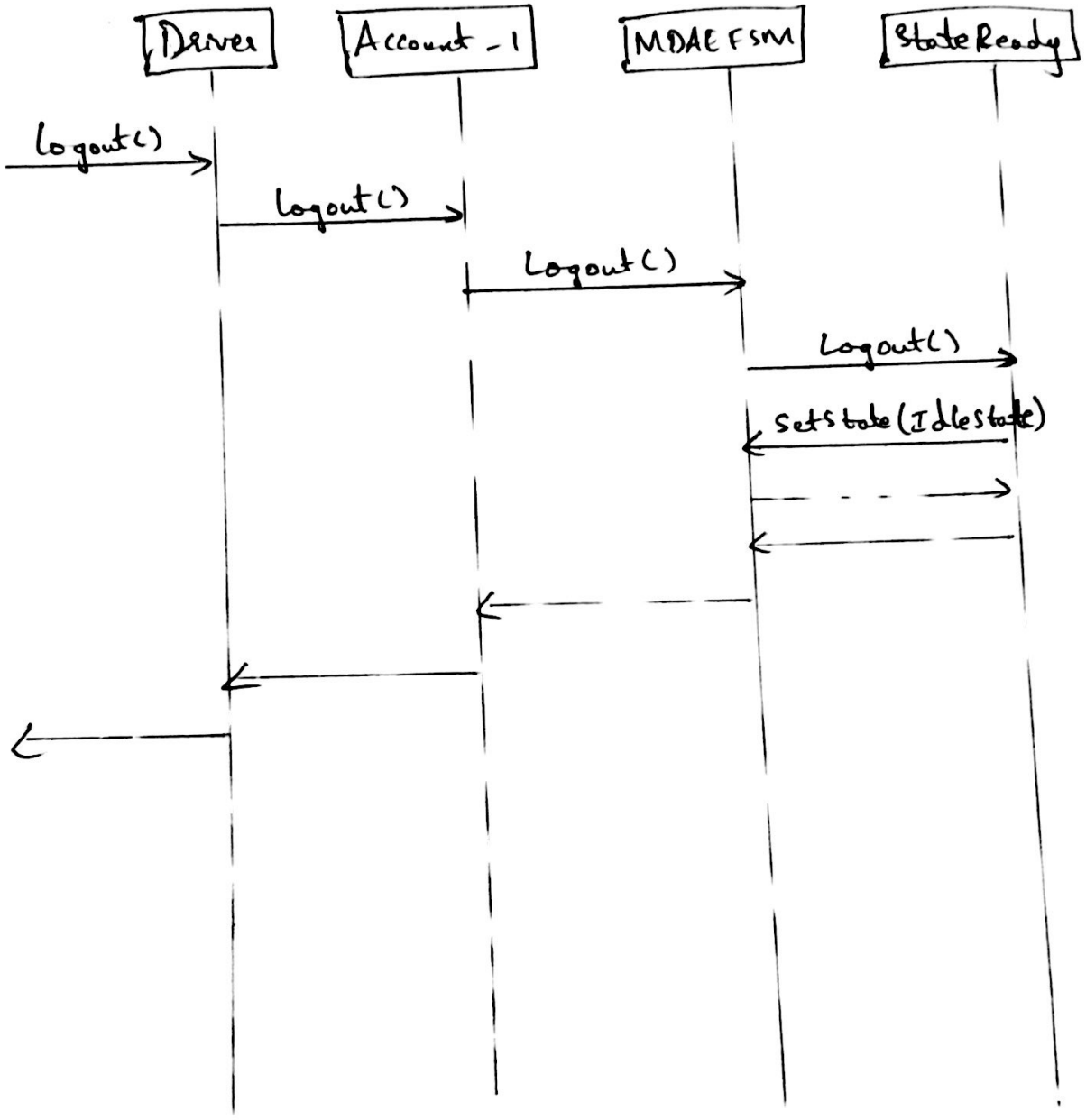




balance()

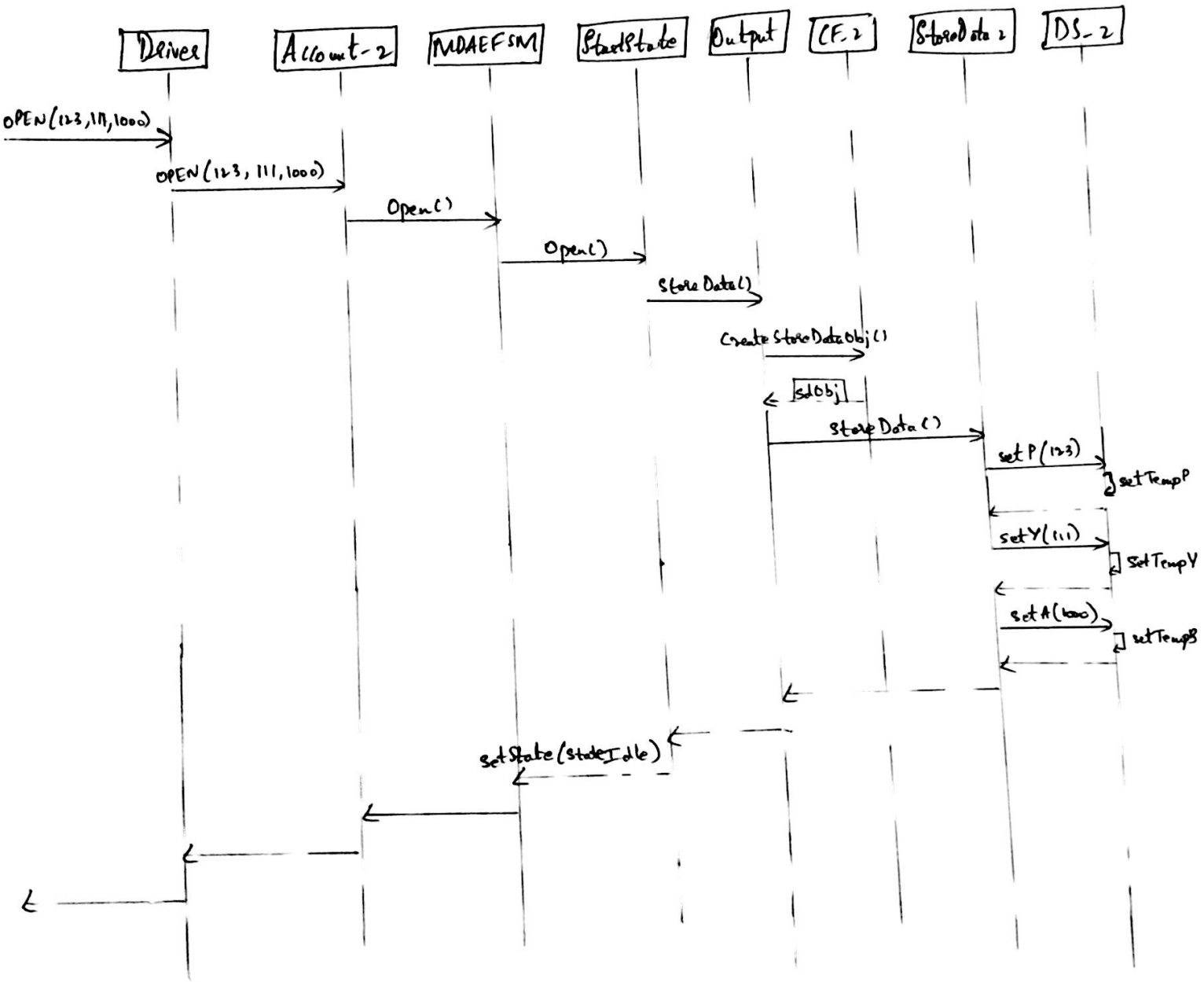


logout()

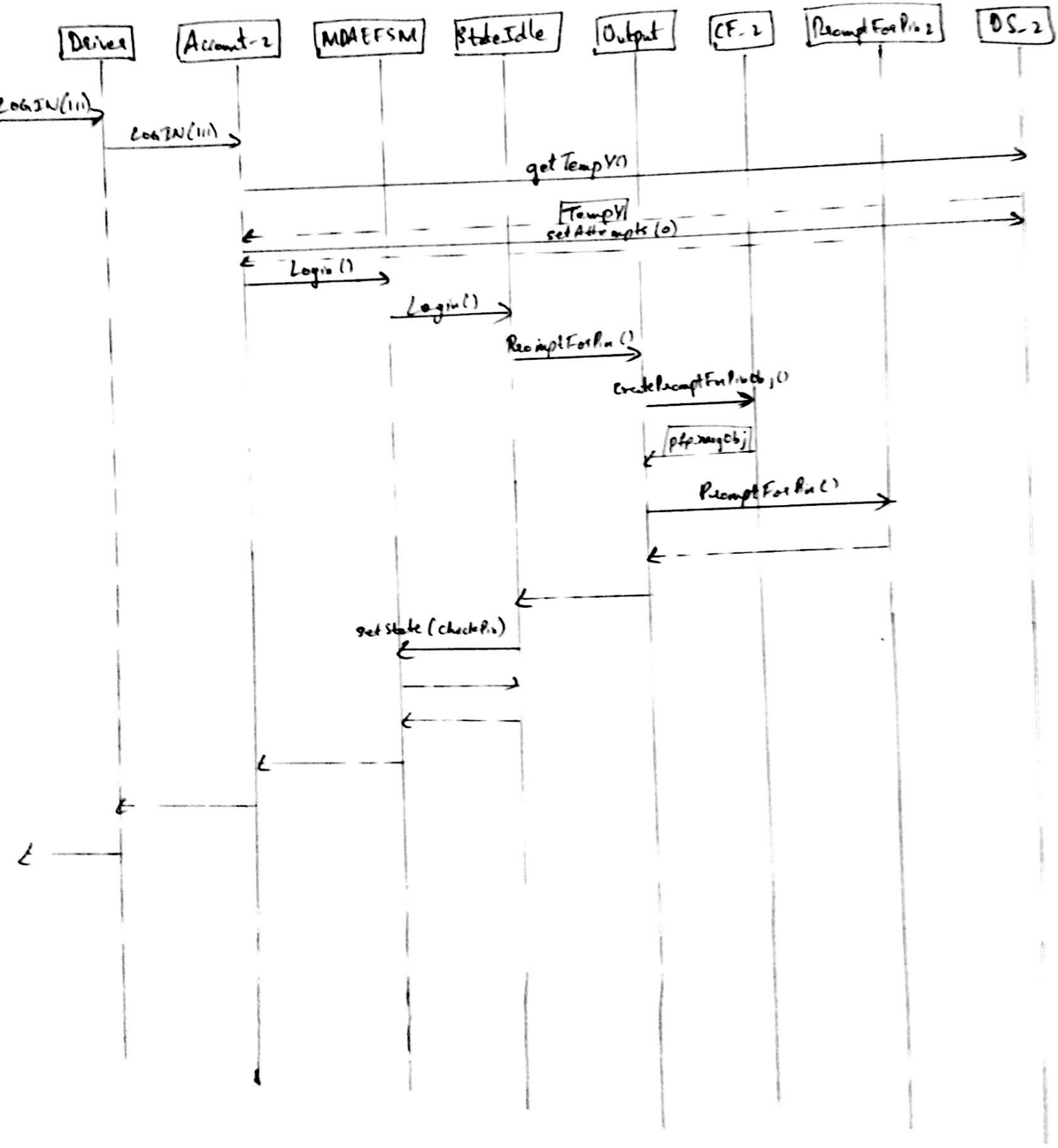


b. Scenario - 2

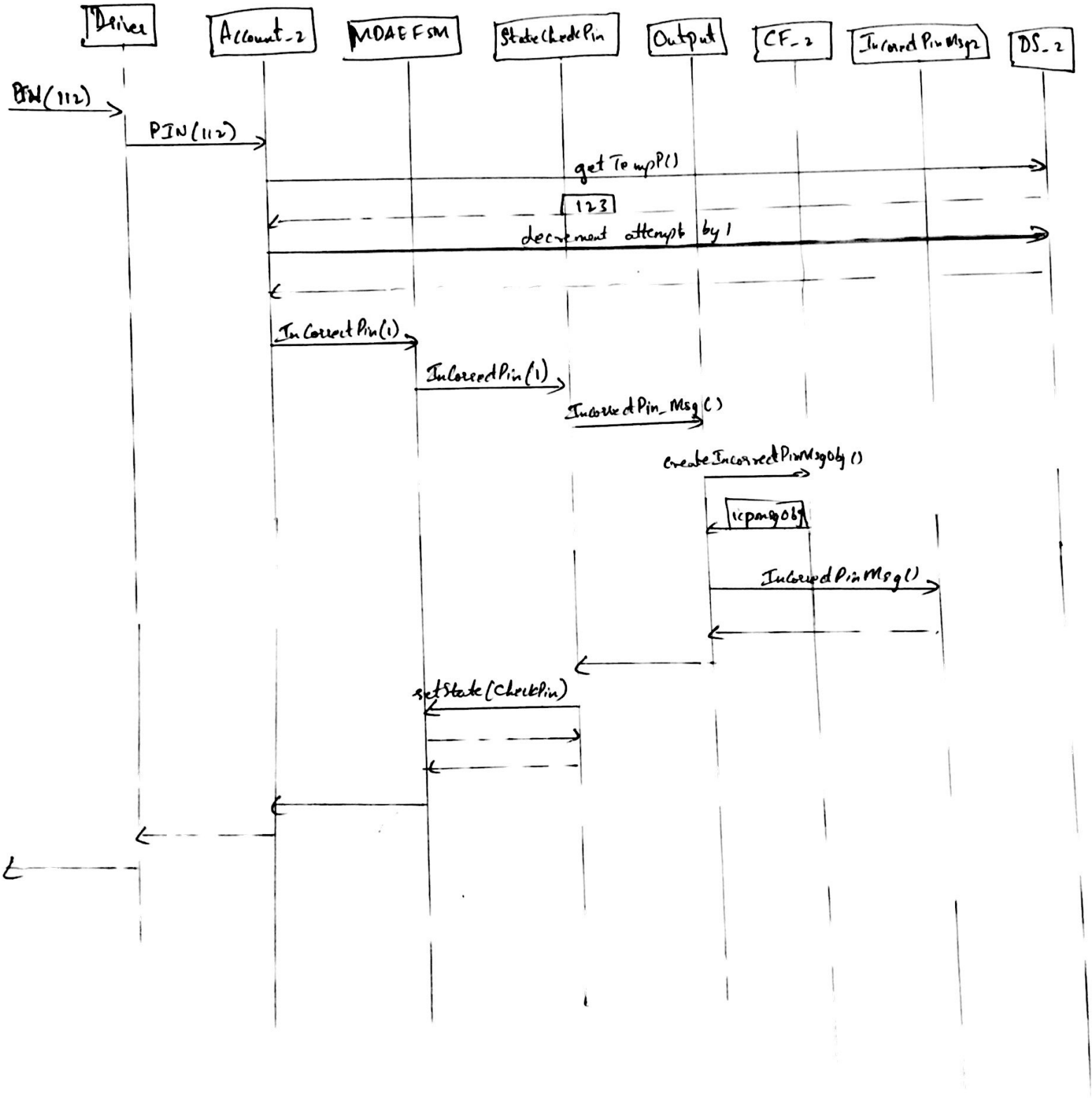
OPEN (123, 111, 1000)



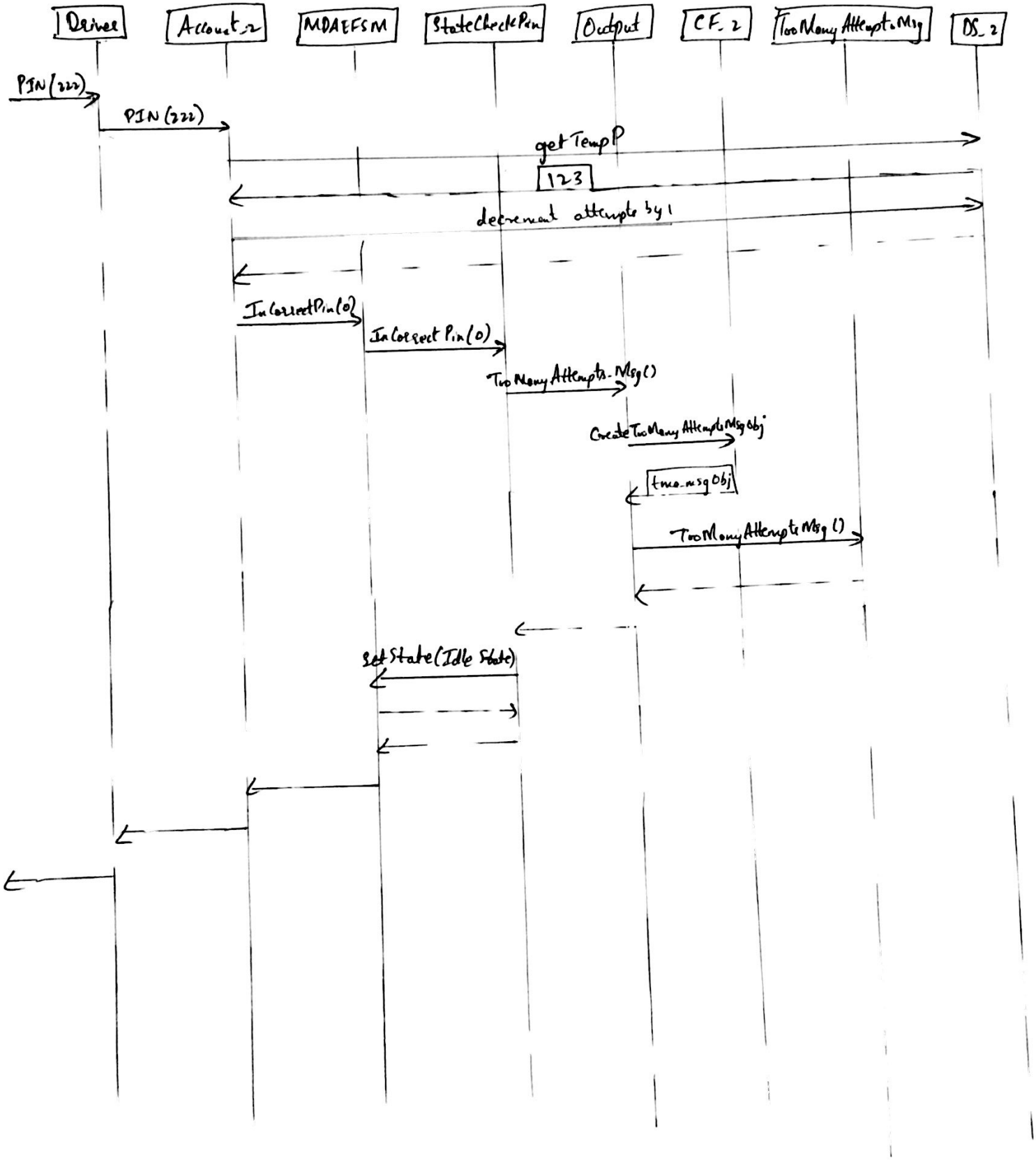
LOGIN(111)



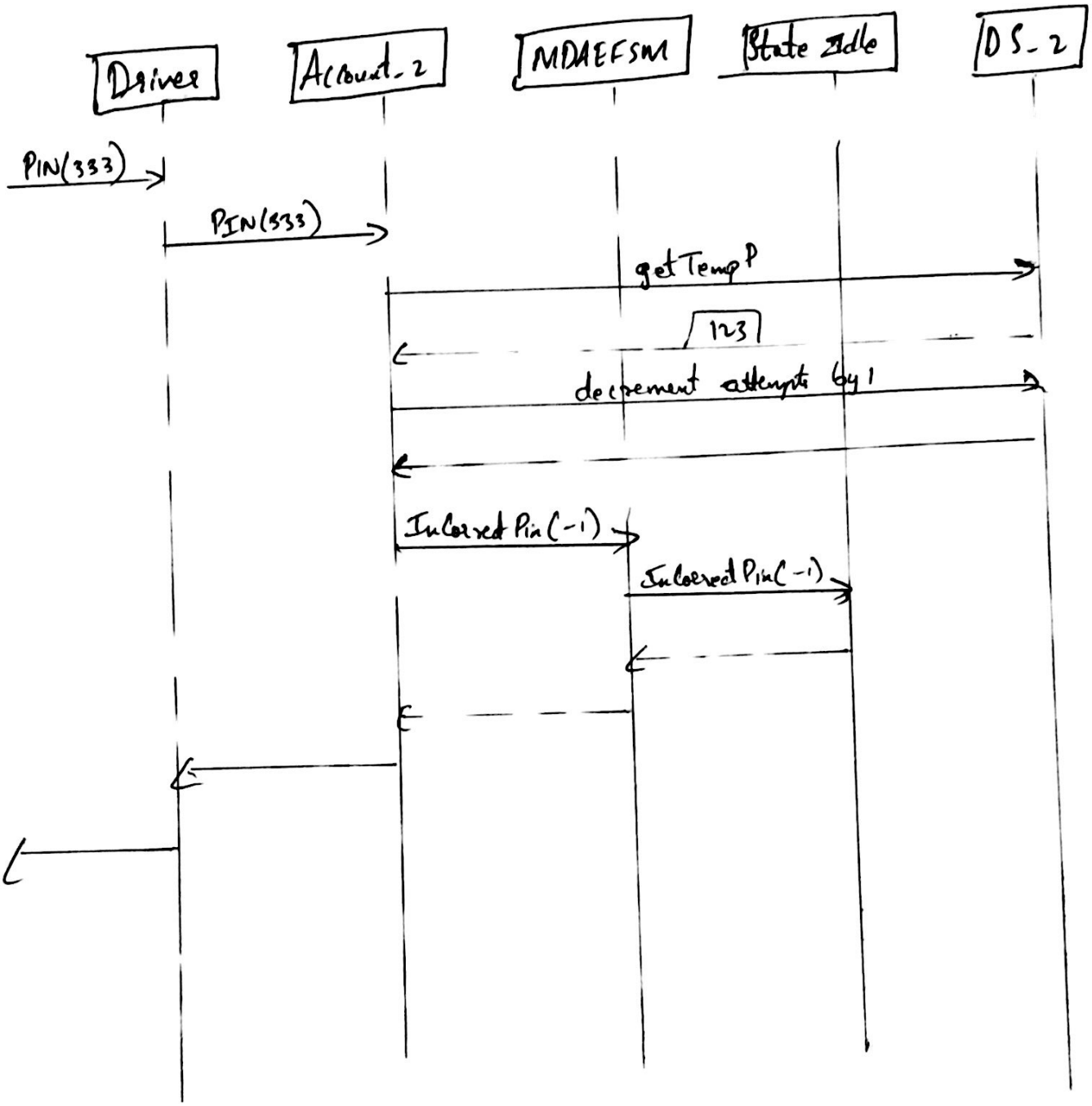
PIN (112)



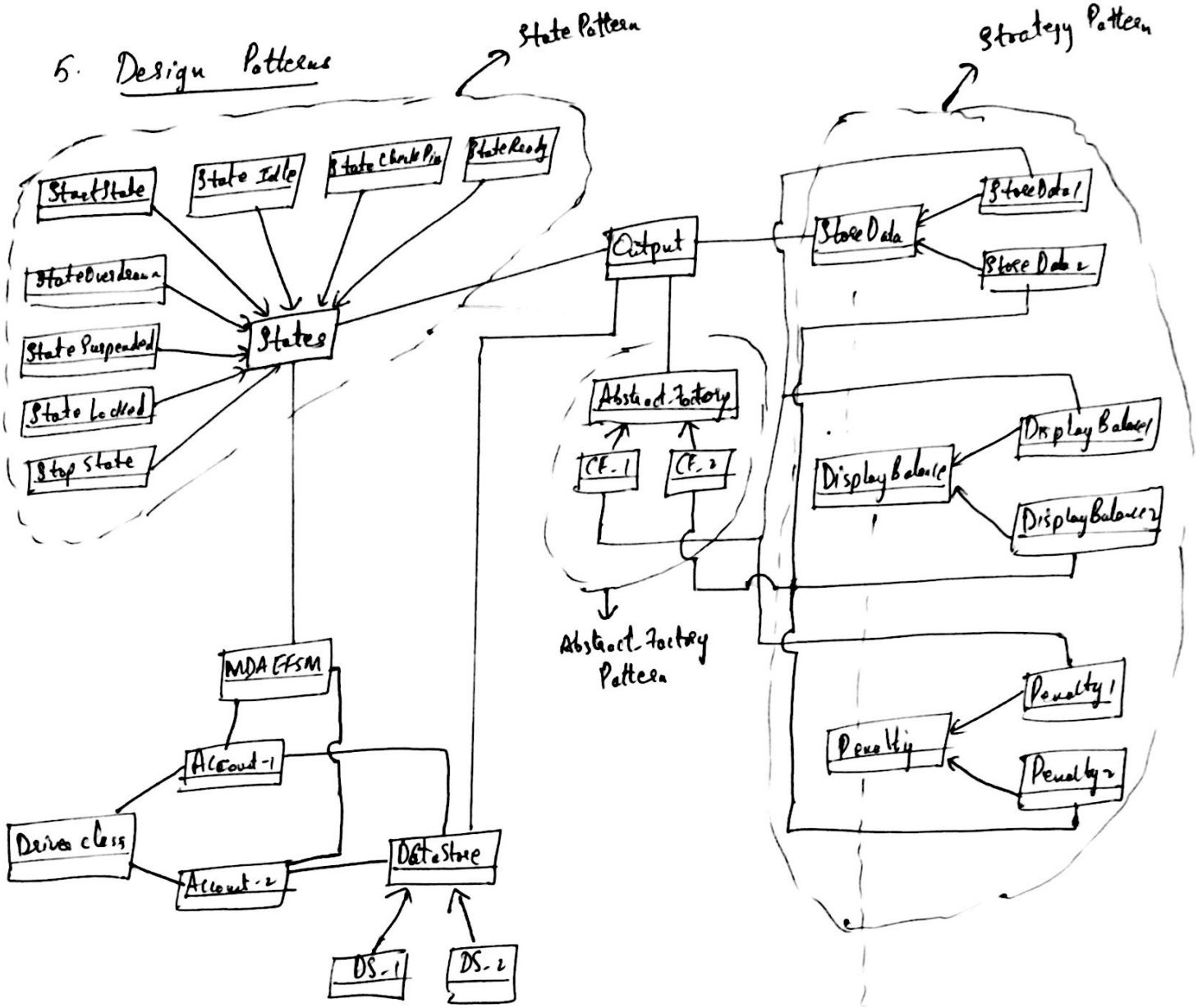
PIN(222)



PIN (333)



5. Design Patterns



State Pattern: Ensures that the Account state are maintained in execution process
De-centralized State Pattern is used where each state update the of context class after the specific event or action is performed
StartState, StateIdle, StateCheckPin ---- StopState & DataStore forms a State Pattern

Abstract factory Pattern: Encapsulates group of Individual factories that exhibit common theme without specifying concrete classes. Provides reference to specific operation for client, where client is unaware of generic interface implemented.

Strategy pattern: An interface common to all logic which provides ability to store a Reference to some code block & retrieve the same. An object is created which represent strategies & context object whose behavior varies as according to strategy object. The strategy object changes the executing block of the context object.

Example: StoreData, StoreData1 & StoreData2 || DisplayBalance, DisplayBalance1 & DisplayBalance2

SOURCE CODE

DRIVER CLASS

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import pkg_Abstract_Factory.CF_1;
import pkg_Abstract_Factory.CF_2;
import pkg_Account.Account_1;
import pkg_Account.Account_2;
import pkg_Mda_Efsm.MDAEFSM;
import pkg_Output.Output;

public class DriverClass {

	public static void main(String[] args) throws IOException
	{
		BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
	String input = null;
	int choice = 1 ;

	System.out.println(" *** CS 586 ***" );
	System.out.println(" *** Project ***" );
	System.out.println(" *** Demo ***" );
	System.out.println();
	System.out.println();
```

```
System.out.println();
System.out.println();

System.out.println("Press Enter key to continue");
input = bufferedReader.readLine();
System.out.println();
System.out.println();
System.out.println();
System.out.println();

for(;;){
        Return:{ System.out.println(" *** Select Account-1, Account-2 ***" );
        System.out.println("         1. Account-1" );
        System.out.println("         2. Account-2" );

        input = bufferedReader.readLine();
        if(input.equalsIgnoreCase("1"))
        {
                String p,y,x;
                float a,d,w;
        CF_1 factory = new CF_1();
        Output output = new Output(factory,factory.GetDataStore());
        MDAEFSM mdaefsm = new MDAEFSM(factory,output);
        Account_1 ac_1 = new Account_1(mdaefsm,factory.GetDataStore());

      System.out.println("*** Menu of Operations ***" );
      System.out.println("         0. open(string, string, float)" );
      System.out.println("         1. login(string)" );
```

```
System.out.println("         2. pin(string)" );
System.out.println("         3. deposit(float)" );
System.out.println("         4. withdraw(float)" );
System.out.println("         5. balance()" );
System.out.println("         6. logout()" );
System.out.println("         7. lock(string)" );
System.out.println("         8. unlock(string)" );
System.out.println("         q. Quit the demo program" );
System.out.println();
System.out.println("  Please make a note of these operations" );
System.out.println("          ACCOUNT-1 Execution" );


            while (true)
{

   System.out.println("  Select Operation: ");
   System.out.println("0-open,1-login,2-pin,3-deposit,4-withdraw,5-balance,6-
logout,7-lock,8-unlock,q-Quit");
   try{
                  input = bufferedReader.readLine();



   if(input.isEmpty()) continue;
   if(input.equalsIgnoreCase("q"))
            break;


   choice = Integer.parseInt(input);


   switch(choice)
```

```
{
 case 0:   //Opening Account
                        System.out.println("   Operation:   open(string p, string y, float a)\n");
                        System.out.println("  Enter value of the parameter p:");
                        p = bufferedReader.readLine();
                        System.out.println("  Enter value of the parameter y:");
                        y = bufferedReader.readLine();
                        System.out.println("  Enter value of the parameter a:");
                        input = bufferedReader.readLine();
                        a = Float.parseFloat(input);
                        ac_1.open(p,y,a);
                        break;

 case 1:   //Login
                        System.out.println("  Operation:  login(string y)");
                        System.out.println("  Enter value of the parameter y:");
                        y = bufferedReader.readLine();
                        ac_1.login(y);
                        break;

 case 2:  //Pin
                        System.out.println("  Operation: pin(string x)");
                        System.out.println("  Enter value of the parameter x:");
                        x = bufferedReader.readLine();
                        ac_1.pin(x);
                        break;

        case 3:  //Deposit
```

```
        System.out.println("  Operation: deposit(float d)");
                System.out.println("  Enter value of the parameter d:");
                input = bufferedReader.readLine();
                d = Float.parseFloat(input);
                ac_1.deposit(d);
                break;

case 4:  //Withdraw
                System.out.println("  Operation:  withdraw(float w)");
                System.out.println("  Enter value of the parameter w:");
                input = bufferedReader.readLine();
                w = Float.parseFloat(input);
                ac_1.withdraw(w);
                break;

case 5:  // Balance
                System.out.println("  Operation:  balance()");
                ac_1.balance();
                break;

case 6:  // logout
                System.out.println("  Operation:  logout()");
                ac_1.logout();
                break;

case 7:  // Lock
                System.out.println("  Operation:  lock(string x)");
                System.out.println("  Enter value of the parameter x:");
```

```
                    x = bufferedReader.readLine();
                    ac_1.lock(x);
                    break;

            case 8:  // Unlock
                    System.out.println("  Operation:  unlock(string x)");
                    System.out.println("  Enter value of the parameter x:");
                    x = bufferedReader.readLine();
                    ac_1.unlock(x);
                    break;

            default:
                    System.out.println("Invalid Choice!!! Enter a Valid Choice");
                    break;
        }
        }
        catch(NumberFormatException e){

            System.out.println("Enter correct choice or value of the parameter");
            System.out.println();

        }
        catch(NullPointerException e){
            System.out.println("Failed!!! NULL reference invoke open first");
            System.out.println();
        }

    }
```

```
            System.out.println("Thanks for using Account - 1" );
            break;

    }

    else if(input.equalsIgnoreCase("2"))
    {
            int p,y,a,x,d,w;
    CF_2 factory = new CF_2();
    Output output = new Output(factory,factory.GetDataStore());
    MDAEFSM mdaefsm = new MDAEFSM(factory,output);
    Account_2 ac_2 = new Account_2(mdaefsm,factory.GetDataStore());

System.out.println("*** Menu of Operations ***" );
System.out.println("        0. OPEN(int, int, int)" );
System.out.println("        1. LOGIN(int)" );
System.out.println("        2. PIN(int)" );
System.out.println("        3. DEPOSIT(int)" );
System.out.println("        4. WITHDRAW(int)" );
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("        q. Quit the demo program" );
System.out.println();
System.out.println("  Please make a note of these operations" );
System.out.println("          ACCOUNT-2 Execution" );
```

```
		while (true)
	{

	   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,q-Quit");
	   try{
			input = bufferedReader.readLine();

	   if(input.isEmpty()) continue;
	   if(input.equalsIgnoreCase("q"))
		break;

	   choice = Integer.parseInt(input);

	   switch(choice)
	   {
	    case 0:   //Opening Account
				System.out.println("   Operation:   OPEN(int p, int y, int
a)\n");
				System.out.println("  Enter value of the parameter p:");
				input = bufferedReader.readLine();
				p = Integer.parseInt(input);
				System.out.println("  Enter value of the parameter y:");
				input = bufferedReader.readLine();
				y = Integer.parseInt(input);
				System.out.println("  Enter value of the parameter a:");
				input = bufferedReader.readLine();
```

```
                    a = Integer.parseInt(input);
                    ac_2.OPEN(p,y,a);
                    break;

case 1:   //Login
                    System.out.println("  Operation:  LOGIN(int y)");
                    System.out.println("  Enter value of the parameter y:");
                    input = bufferedReader.readLine();
                    y = Integer.parseInt(input);
                    ac_2.LOGIN(y);
                    break;

case 2:  //Pin
                    System.out.println("  Operation: PIN(int x)");
                    System.out.println("  Enter value of the parameter x:");
                    input = bufferedReader.readLine();
                    x = Integer.parseInt(input);
                    ac_2.PIN(x);
                    break;

        case 3:  //Deposit
                System.out.println("  Operation: DEPOSIT(int d)");
                    System.out.println("  Enter value of the parameter d:");
                    input = bufferedReader.readLine();
                    d = Integer.parseInt(input);
                    ac_2.DEPOSIT(d);
                    break;
```

```
case 4:  //Withdraw
                System.out.println("  Operation:  WITHDRAW(int w)");
                System.out.println("  Enter value of the parameter w:");
                input = bufferedReader.readLine();
                w = Integer.parseInt(input);
                ac_2.WITHDRAW(w);
                break;

case 5:  // Balance
                System.out.println("  Operation:  BALANCE()");
                ac_2.BALANCE();
                break;

case 6:  // Logout
                System.out.println("  Operation:  LOGOUT()");
                ac_2.LOGOUT();
                break;

case 7:  // Suspend
                System.out.println("  Operation:  suspend()");
                ac_2.suspend();
                break;

case 8:  // Activate
                System.out.println("  Operation:  activate()");
                ac_2.activate();
                break;
```

```
                case 9:  // Close
                                System.out.println("  Operation:  close()");
                                ac_2.close();
                                break;

                default:
                                System.out.println("Invalid Choice!!! Enter a Valid Choice");
                                break;
            }
            }
            catch(NumberFormatException e){

                System.out.println("Enter correct choice or parameter");
                System.out.println();

            }
            catch(NullPointerException e){
                System.out.println("Failed!! Null reference Invoke OPEN first");
                System.out.println();
            }

        }
                System.out.println("Thanks for using Account - 2" );
                break;

            }
            else
            {
```

```
                        System.out.println("Enter either 1 or 2");
                        break Return;
                }
        }
        }
        }
}
```

ABSTRACT FACTORY

```
package pkg_Abstract_Factory;

import pkg_Data_Store.DS;
import pkg_Strategy.*;

public interface AbstractFactory
{
        public DS CreateDataStoreObj();
        public StoreData CreateStoreDataObj();
        public DisplayBalance CreateDisplayBalanceObj();
        public DisplayMenu CreateDisplayMenuObj();
        public IncorrectLockMsg CreateIncorrectLockMsgObj();
        public IncorrectUnlockMsg CreateIncorrectUnlockMsgObj();
        public IncorrectIdMsg CreateIncorrectIdMsgObj();
        public IncorrectPinMsg CreateIncorrectPinMsgObj();
        public MakeDeposit CreateMakeDepositObj();
        public MakeWithdraw CreateMakeWithdrawObj();
```

```
	public NoFundsMsg CreateNoFundsMsgObj();
	public Penalty CreatePenaltyObj();
	public PromptForPin CreatePromptForPinObj();
	public TooManyAttemptsMsg CreateTooManyAttemptsMsgObj();
}
```

CONCRETE FACTORY 1

```
package pkg_Abstract_Factory;

import pkg_Data_Store.DS;
import pkg_Data_Store.DS_1;
import pkg_Strategy.*;



/*
 * Concrete Factory 1 that consists of implementations of the ACCOUNT -1
 */
public class CF_1 implements AbstractFactory
{
	DS obj_ds = new DS_1();
	StoreData obj_sdt = new StoreData1();
	DisplayBalance obj_bal = new DisplayBalance1();
	DisplayMenu obj_dme = new DisplayMenu1();
	IncorrectLockMsg obj_ilm = new IncorrectLockMsg1();
	IncorrectUnlockMsg obj_iulm = new IncorrectUnlockMsg1();
	IncorrectIdMsg obj_idm = new IncorrectIdMsg1();
	IncorrectPinMsg obj_ipm = new IncorrectPinMsg1();
	MakeDeposit obj_mdt = new MakeDeposit1();
```

```
MakeWithdraw obj_mwt = new MakeWithdraw1();
NoFundsMsg obj_nfm = new NoFundsMsg1();
Penalty obj_pty = new Penalty1();
PromptForPin obj_pfp = new PromptForPin1();
TooManyAttemptsMsg obj_tmam = new TooManyAttemptsMsg1();


@Override
public DS CreateDataStoreObj() {
	return this.obj_ds;
}

public DS GetDataStore(){
	return this.obj_ds;
}

@Override
public StoreData CreateStoreDataObj() {
	return this.obj_sdt;
}

@Override
public DisplayBalance CreateDisplayBalanceObj() {
	return this.obj_bal;
}

@Override
public DisplayMenu CreateDisplayMenuObj() {
```

```
        return this.obj_dme;
}

@Override
public IncorrectLockMsg CreateIncorrectLockMsgObj() {
        return this.obj_ilm;
}

@Override
public IncorrectUnlockMsg CreateIncorrectUnlockMsgObj() {
        return this.obj_iulm;
}
@Override
public IncorrectIdMsg CreateIncorrectIdMsgObj() {
        return this.obj_idm;
}

@Override
public IncorrectPinMsg CreateIncorrectPinMsgObj() {
        return this.obj_ipm;
}

@Override
public MakeDeposit CreateMakeDepositObj() {
        return this.obj_mdt;
}

@Override
```

```
	public MakeWithdraw CreateMakeWithdrawObj() {
		return this.obj_mwt;
	}

	@Override
	public NoFundsMsg CreateNoFundsMsgObj() {
		return this.obj_nfm;
	}

	@Override
	public Penalty CreatePenaltyObj() {
		return this.obj_pty;	}
	@Override
	public PromptForPin CreatePromptForPinObj() {
		return this.obj_pfp;
	}

	@Override
	public TooManyAttemptsMsg CreateTooManyAttemptsMsgObj() {
		return this.obj_tmam;
	}

}
```

CONCRETE FACTORY 2

```
package pkg_Abstract_Factory;

import pkg_Data_Store.DS;
```

```
import pkg_Data_Store.DS_2;
import pkg_Strategy.*;

public class CF_2 implements AbstractFactory
{
	DS obj_ds = new DS_2();
	StoreData obj_sdt = new StoreData2();
	DisplayBalance obj_bal = new DisplayBalance2();
	DisplayMenu obj_dme = new DisplayMenu2();
	IncorrectLockMsg obj_ilm = new IncorrectLockMsg2();
	IncorrectUnlockMsg obj_iulm = new IncorrectUnlockMsg2();
	IncorrectIdMsg obj_idm = new IncorrectIdMsg2();
	IncorrectPinMsg obj_ipm = new IncorrectPinMsg2();
	MakeDeposit obj_mdt = new MakeDeposit2();
	MakeWithdraw obj_mwt = new MakeWithdraw2();
	NoFundsMsg obj_nfm = new NoFundsMsg2();
	Penalty obj_pty = new Penalty2();
	PromptForPin obj_pfp = new PromptForPin2();
	TooManyAttemptsMsg obj_tmam = new TooManyAttemptsMsg2();


	@Override
	public DS CreateDataStoreObj() {
		return this.obj_ds;
	}

	public DS GetDataStore(){
		return this.obj_ds;
```

```
}

@Override
public StoreData CreateStoreDataObj() {
        return this.obj_sdt;
}

@Override
public DisplayBalance CreateDisplayBalanceObj() {
        return this.obj_bal;
}

@Override
public DisplayMenu CreateDisplayMenuObj() {
        return this.obj_dme;
}

@Override
public IncorrectLockMsg CreateIncorrectLockMsgObj() {
        return null;
}

@Override
public IncorrectUnlockMsg CreateIncorrectUnlockMsgObj() {
        return null;
}

@Override
```

```
public IncorrectIdMsg CreateIncorrectIdMsgObj() {
        return this.obj_idm;
}

@Override
public IncorrectPinMsg CreateIncorrectPinMsgObj() {
        return this.obj_ipm;
}

@Override
public MakeDeposit CreateMakeDepositObj() {
        return this.obj_mdt;
}

@Override
public MakeWithdraw CreateMakeWithdrawObj() {
        return this.obj_mwt;
}

@Override
public NoFundsMsg CreateNoFundsMsgObj() {
        return this.obj_nfm;
}

@Override
public Penalty CreatePenaltyObj() {
        return this.obj_pty;
}
```

```
	@Override
	public PromptForPin CreatePromptForPinObj() {
		return this.obj_pfp;
	}

	@Override
	public TooManyAttemptsMsg CreateTooManyAttemptsMsgObj() {
		return this.obj_tmam;
	}
}
```

ACCOUNT – 1

```
package pkg_Account;

import pkg_Data_Store.*;
import pkg_Mda_Efsm.MDAEFSM;

public class Account_1 {

	MDAEFSM mdaefsm =null;
	DS datastore = null;

	public Account_1(MDAEFSM mdaefsm, DS datastore)
	{
		this.mdaefsm = mdaefsm;
		this.datastore = datastore;
		//this.open(((DS_1)datastore).tempP, ((DS_1)datastore).tempY, ((DS_1)datastore).tempB);
	}

	public void open(String p, String y, float a)
	{
		((DS_1)datastore).tempP= p;
		((DS_1)datastore).tempY=y;
		((DS_1)datastore).tempB=a;
		mdaefsm.Open();
```

```
	}

	public void login(String y)
	{
		if ((((DS_1)datastore).tempY).equals(y))
		{
			((DS_1)datastore).settemp_attempts(3);
			mdaefsm.Login();
		}
		else
		{
			mdaefsm.IncorrectLogin();
		}
	}

	public void pin(String x)
	{
			if((((DS_1)datastore).tempP).equals(x)  &&  (((DS_1)datastore).tempB  >
500))
			{
				mdaefsm.CorrectPinAboveMin();
			}
			else if((((DS_1)datastore).tempP).equals(x) && (((DS_1)datastore).tempB
<= 500))
			{
				mdaefsm.CorrectPinBelowMin();
			}
			else		if(!(((DS_1)datastore).tempP).equals(x)		&&
(((DS_1)datastore).temp_attempts > 0))
			{
				((DS_1)datastore).temp_attempts		=
((DS_1)datastore).temp_attempts - 1;
				mdaefsm.IncorrectPin(((DS_1)datastore).temp_attempts);
			}

	}

	public void deposit(float d)
	{
		((DS_1)datastore).tempD = d;
		if (((((DS_1)datastore).tempB) + d) > 500)
		{
			mdaefsm.Deposit();
			mdaefsm.AboveMinBalance();
		}
		else if (((((DS_1)datastore).tempB) + d)  <= 500)
```

```
		{
			mdaefsm.Deposit();
			mdaefsm.BelowMinBalance();
		}
		else
		{
			mdaefsm.Deposit();
		}
}

public void withdraw(float w)
{
		((DS_1)datastore).tempW = w;
		if (((((DS_1)datastore).tempB) - w) > 500)
		{
			mdaefsm.Withdraw();
			mdaefsm.AboveMinBalance();
		}
		else if (((((DS_1)datastore).tempB) - w)  <= 500)
		{
			mdaefsm.Withdraw();
			mdaefsm.WithdrawBelowMinBalance();
		}
		else
		{
			mdaefsm.BelowMinBalance();
		}
}

public void balance()
{
		mdaefsm.Balance();
}

public void logout()
{
		mdaefsm.Logout();
}

public void lock(String x)
{
		if ((((DS_1)datastore).tempP).equals(x))
		{
			mdaefsm.Lock();
		}
		else
```

```
		{
			mdaefsm.IncorrectLock();
		}
	}

	public void unlock(String x)
	{
		if ((((DS_1)datastore).tempP).equals(x) && (((DS_1)datastore).tempB > 500) )
		{
			mdaefsm.Unlock();
			mdaefsm.AboveMinBalance();
		}
		else if ((((DS_1)datastore).tempP).equals(x) && (((DS_1)datastore).tempB <=
500))
		{
			mdaefsm.Unlock();
			mdaefsm.BelowMinBalance();
		}
		else if (!(((DS_1)datastore).tempP).equals(x))
		{
			mdaefsm.IncorrectUnlock();
		}
	}

}
```

ACCOUNT – 2

```
package pkg_Account;

import pkg_Data_Store.*;

import pkg_Mda_Efsm.MDAEFSM;

public class Account_2
{
	MDAEFSM mdaefsm =null;
	DS datastore = null;

	public Account_2(MDAEFSM mdaefsm, DS datastore)
	{
		this.mdaefsm = mdaefsm;
		this.datastore = datastore;
		//this.OPEN(((DS_2)datastore).p, ((DS_2)datastore).y, ((DS_2)datastore).a);
```

```
	}



	public void OPEN(int p, int y, int a)
	{
		((DS_2)datastore).tempP = p;
		((DS_2)datastore).tempY = y;
		((DS_2)datastore).tempB = a;
		mdaefsm.Open();
	}

	public void LOGIN(int y)
	{
		if (y==((DS_2)datastore).tempY)
		{
			((DS_2)datastore).settemp_attempts(2);
			mdaefsm.Login();
		}
		else if(y!=((DS_2)datastore).tempY)
		{
			mdaefsm.IncorrectLogin();
		}
	}

	public void PIN(int x)
	{
			if((x==((DS_2)datastore).tempP))
			{
				mdaefsm.CorrectPinAboveMin();
			}

			else if((x!=((DS_2)datastore).tempP) &&
(((DS_2)datastore).temp_attempts > 0))
			{
				((DS_2)datastore).temp_attempts =
((DS_2)datastore).temp_attempts - 1;
				mdaefsm.IncorrectPin(((DS_2)datastore).temp_attempts);
			}

	}

	public void DEPOSIT(int d)
	{
		((DS_2)datastore).tempD = d;
		mdaefsm.Deposit();
```

```
	}

	public void WITHDRAW(int w)
	{
		((DS_2)datastore).tempW = w;
		if ((((DS_2)datastore).tempB) <= 0)
		{
			mdaefsm.NoFunds();
		}
		else if ((((DS_2)datastore).tempB)> 0)
		{
			mdaefsm.Withdraw();
			mdaefsm.AboveMinBalance();
		}
	}

	public void BALANCE()
	{
		mdaefsm.Balance();
	}

	public void LOGOUT()
	{
		mdaefsm.Logout();
	}

	public void activate()
	{
		mdaefsm.Activate();
	}

	public void suspend()
	{
		mdaefsm.Suspend();
	}

	public void close()
	{
		mdaefsm.Close();
	}

}
```

DATASTORE (DS)

```
package pkg_Data_Store;
```

```java
/*
 * Data Store(DS) is a Abstract Class and acts as Database which stores all the variable values
 */
public abstract class DS {

}
```

DS_1

```java
package pkg_Data_Store;
/*
 * DS_1 is the database for the ACCOUNT-1
 */
public class DS_1 extends DS
{

	public String p= tempP;
	public String y = tempY;
	public float a = tempB;

	//public float b = temp_a;
	public static float tempB;
	public static String tempY;
	public static String tempP;
	public float tempD;
	public float tempW;
	public int temp_attempts;
	public int attempts;
	public int flag;

	public DS_1() {
		super();
	}

	public DS_1(String p, String y,float a){
		super();
		this.p = p;
		this.y = y;
		this.a = a;
	}

	public String getP()
	{
		return p;
	}
```

```
public void setP(String p)
{
	this.p = p;
}

public String getY() {
	return y;
}
public void setY(String y) {
	this.y = y;
}


public float getA() {
	return a;
}
public void setA(float a) {
	this.a = a;
}
public float getTempB() {
	return tempB;
}
public void setTempB(float tempB) {
	DS_1.tempB = tempB;
}
public String getTempY() {
	return tempY;
}
public void setTempY(String tempY) {
	DS_1.tempY = tempY;
}
public String getTempP() {
	return tempP;
}
public void setTempP(String tempP) {
	DS_1.tempP = tempP;
}
public float getTempD() {
	return tempD;
}
public void setTempD(float tempD) {
	this.tempD = tempD;
}
public float getTempW() {
	return tempW;
}
```

```
	public void setTempW(float tempW) {
		this.tempW = tempW;
	}

	public int gettemp_attempts()
	{
		return temp_attempts;
	}

	public void settemp_attempts(int t_at)
	{
		temp_attempts=t_at;
	}

	public int getattempts()
	{
		return attempts;
	}

	public void setattempts()
	{
		attempts = attempts+1;
	}
}
```

DS_2

```
package pkg_Data_Store;
/*
 * DS_2 is the database for the ACCOUNT-2
 */
public class DS_2 extends DS
{

	public int p = tempP;
	public int y = tempY;
	public int a = tempB;

	//public float b = temp_a;
	public static int tempB;
	public static int tempY;
	public static int tempP;
	public int tempD;
	public int tempW;
	public int temp_attempts;
	public int attempts;
```

```
public int flag;

public DS_2() {
	super();
}

public DS_2(int p, int y, int a){
	super();
	this.p = p;
	this.y = y;
	this.a = a;
}
public int getP() {
	return p;
}

public void setP(int p) {
	this.p = p;
}

public int getY() {
	return y;
}

public void setY(int y) {
	this.y = y;
}

public int getA() {
	return a;
}

public void setA(int a) {
	this.a = a;
}

public int getTempB() {
	return tempB;
}

public void setTempB(int tempB) {
	DS_2.tempB = tempB;
}

public int getTempY() {
	return tempY;
```

```
}

public void setTempY(int tempY) {
	DS_2.tempY = tempY;
}

public int getTempP() {
	return tempP;
}

public void setTempP(int tempP) {
	DS_2.tempP = tempP;
}

public int getTempD() {
	return tempD;
}

public void setTempD(int tempD) {
	this.tempD = tempD;
}

public int getTempW() {
	return tempW;
}

public void setTempW(int tempW) {
	this.tempW = tempW;
}


public int gettemp_attempts()
{
	return temp_attempts;
}

public void settemp_attempts(int t_at)
{
	temp_attempts=t_at;
}

public int getattempts()
{
	return attempts;
}
```

```
	public void setattempts()
	{
		attempts = attempts+1;
	}

}
```

MDAEFSM

```
package pkg_Mda_Efsm;

import pkg_Abstract_Factory.AbstractFactory;
import pkg_Output.Output;

public class MDAEFSM
{
	States start_state = new StartState(this);
	States stop_state = new StopState(this);
	//Objects for all states
	States statechkpn = new StateCheckPin(this);
	States stateidle = new StateIdle(this);
	States statelkd = new StateLocked(this);
	States stateovdr = new StateOverdrawn(this);
	States staterdy = new StateReady(this);
	States statespd = new StateSuspended(this);
	States states1 = new StateS1(this);

	Output output = null;
	States efsmState = null;
	AbstractFactory factory =null;

	//Constructor to set the object values
	public MDAEFSM (AbstractFactory fact, Output out)
	{
		efsmState = start_state;
      this.factory = fact;
      this.output = out;
	}

	/*
	 * Methods implementation to the next state
	 */
	public void Open()
	{
		efsmState.Open();
```

```
        printCurrentState();
}

public void Login ()
{
        efsmState.Login();
        printCurrentState();
}

public void IncorrectLogin()
{
        efsmState.IncorrectLogin();
        printCurrentState();
}

public void IncorrectPin(int max)
{
        efsmState.IncorrectPin(max);
        printCurrentState();
}

public void CorrectPinBelowMin()
{
        efsmState.CorrectPinBelowMin();
        printCurrentState();
}

public void CorrectPinAboveMin()
{
        efsmState.CorrectPinAboveMin();
        printCurrentState();
}

public void Deposit()
{
        efsmState.Deposit();
        printCurrentState();
}

public void BelowMinBalance()
{
        efsmState.BelowMinBalance();
        printCurrentState();
}

public void AboveMinBalance()
```

```
{
        efsmState.AboveMinBalance();
        printCurrentState();
}

public void Logout()
{
        efsmState.Logout();
        printCurrentState();
}

public void Balance()
{
        efsmState.Balance();
        printCurrentState();
}

public void Withdraw()
{
        efsmState.Withdraw();
        printCurrentState();
}

public void WithdrawBelowMinBalance()
{
        efsmState.WithdrawBelowMinBalance();
        printCurrentState();
}

public void NoFunds()
{
        efsmState.NoFunds();
        printCurrentState();
}

public void Lock()
{
        efsmState.Lock();
        printCurrentState();
}

public void IncorrectLock()
{
        efsmState.IncorrectLock();
        printCurrentState();
}
```

```
        public void Unlock()
        {
                efsmState.Unlock();
                printCurrentState();
        }

        public void IncorrectUnlock()
        {
                efsmState.IncorrectUnlock();
                printCurrentState();
        }

        public void Suspend()
        {
                efsmState.Suspend();
                printCurrentState();
        }

        public void Activate()
        {
                efsmState.Activate();
                printCurrentState();
        }

        public void Close()
        {
                efsmState.Close();
                printCurrentState();
        }


public void setState(States efsmState)
{
   this.efsmState = efsmState;
}
public States getAccountState()
{
   return efsmState;
}

public States getCheckPinState()
{
   return statechkpn;
}
public States getIdleState()
```

```
    {
        return stateidle;
    }
    public States getLockedState()
    {
        return statelkd;
    }

    public States getOverdrawnState()
    {
        return stateovdr;
    }

    public States getReadyState()
    {
        return staterdy;
    }

    public States getSuspendedState()
    {
        return statespd;
    }

    public States getS1State()
    {
        return states1;
    }
        public void printCurrentState()
        {
                System.out.println("The Current State is :"+ efsmState.getClass().getName());
        }

        public States getStopstate() {
                return stop_state;
        }

}
```

START STATE

```
package pkg_Mda_Efsm;

public class StartState implements States
{

        MDAEFSM mdaefsm = null;
```

```
public StartState(MDAEFSM mdaefsm) {
        this.mdaefsm= mdaefsm;

}
@Override
public void Open() {
        mdaefsm.output.StoreData();
        mdaefsm.setState(mdaefsm.getIdleState());
}

@Override
public void Login() {

}

@Override
public void IncorrectLogin() {

}

@Override
public void IncorrectPin(int max) {

}

@Override
public void CorrectPinBelowMin() {

}

@Override
public void CorrectPinAboveMin() {

}

@Override
public void Deposit() {

}

@Override
public void BelowMinBalance() {

}
```

```
@Override
public void AboveMinBalance() {

}

@Override
public void Logout() {

}

@Override
public void Balance() {

}

@Override
public void Withdraw() {

}

@Override
public void WithdrawBelowMinBalance() {

}

@Override
public void NoFunds() {

}

@Override
public void Lock() {

}

@Override
public void IncorrectLock() {

}

@Override
public void Unlock() {

}

@Override
```

```
	public void IncorrectUnlock() {

	}

	@Override
	public void Suspend() {

	}

	@Override
	public void Activate() {

	}

	@Override
	public void Close() {

	}

}
```

CHECKPIN STATE

```
package pkg_Mda_Efsm;

public class StateCheckPin implements States
{
	MDAEFSM mdaefsm = null;

	public StateCheckPin(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
	public void Open() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Login() {
		// TODO Auto-generated method stub

	}

	@Override
```

```
public void IncorrectLogin() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectPin(int max) {
	if (max > 0){
		mdaefsm.output.IncorrectPin_Msg();
		System.out.println("ENTER THE PIN AGAIN");
		mdaefsm.setState(mdaefsm.getCheckPinState());
	}

	else if (max == 0){
		mdaefsm.output.TooManyAttempts_Msg();
		mdaefsm.setState(mdaefsm.getIdleState());
	}

}

@Override
public void CorrectPinBelowMin() {
	mdaefsm.output.DisplayMenu();
	mdaefsm.setState(mdaefsm.getOverdrawnState());
}

@Override
public void CorrectPinAboveMin() {
	mdaefsm.output.DisplayMenu();
	mdaefsm.setState(mdaefsm.getReadyState());
}

@Override
public void Deposit() {
	// TODO Auto-generated method stub

}

@Override
public void BelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void AboveMinBalance() {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void Logout() {
		mdaefsm.setState(mdaefsm.getIdleState());
	}

	@Override
	public void Balance() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Withdraw() {
		// TODO Auto-generated method stub

	}

	@Override
	public void WithdrawBelowMinBalance() {
		// TODO Auto-generated method stub

	}

	@Override
	public void NoFunds() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Lock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectLock() {
		// TODO Auto-generated method stub

	}

	@Override
```

```
	public void Unlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectUnlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Suspend() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Close() {
		// TODO Auto-generated method stub

	}

}
```

IDLE STATE

```
package pkg_Mda_Efsm;

public class StateIdle implements States
{
	MDAEFSM mdaefsm = null;

	public StateIdle(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
	public void Open() {
		// TODO Auto-generated method stub
```

```
}

@Override
public void Login() {
	mdaefsm.output.PromptForPin();
	mdaefsm.setState(mdaefsm.getCheckPinState());
}

@Override
public void IncorrectLogin() {
	mdaefsm.output.IncorrectId_Msg();
	mdaefsm.setState(mdaefsm.getIdleState());
}

@Override
public void IncorrectPin(int max) {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {
	// TODO Auto-generated method stub

}

@Override
public void BelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
```

```
public void AboveMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void Logout() {
	// TODO Auto-generated method stub

}

@Override
public void Balance() {
	// TODO Auto-generated method stub

}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub

}

@Override
public void WithdrawBelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void NoFunds() {
	// TODO Auto-generated method stub

}

@Override
public void Lock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLock() {
	// TODO Auto-generated method stub

}
```

```
	@Override
	public void Unlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectUnlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Suspend() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Close() {
		// TODO Auto-generated method stub

	}

}
```

LOCKED STATE

```
package pkg_Mda_Efsm;

public class StateLocked implements States
{
	MDAEFSM mdaefsm = null;

	public StateLocked(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
```

```
public void Open() {
	// TODO Auto-generated method stub

}

@Override
public void Login() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLogin() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectPin(int max) {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {
	// TODO Auto-generated method stub

}

@Override
public void BelowMinBalance() {
	// TODO Auto-generated method stub

}
```

```
@Override
public void AboveMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void Logout() {
	// TODO Auto-generated method stub

}

@Override
public void Balance() {
	// TODO Auto-generated method stub

}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub

}

@Override
public void WithdrawBelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void NoFunds() {
	// TODO Auto-generated method stub

}

@Override
public void Lock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLock() {
	// TODO Auto-generated method stub
```

```
	}

	@Override
	public void Unlock() {
		mdaefsm.setState(mdaefsm.getS1State());
	}

	@Override
	public void IncorrectUnlock() {
		mdaefsm.output.IncorrectUnlock_Msg();
		mdaefsm.setState(mdaefsm.getLockedState());
	}

	@Override
	public void Suspend() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Close() {
		// TODO Auto-generated method stub

	}

}
```

OVERDRAWN STATE

```
package pkg_Mda_Efsm;

public class StateOverdrawn implements States
{
	MDAEFSM mdaefsm = null;

	public StateOverdrawn(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}
```

```
@Override
public void Open() {
	// TODO Auto-generated method stub

}

@Override
public void Login() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLogin() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectPin(int max) {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {
	mdaefsm.output.MakeDeposit();
	mdaefsm.setState(mdaefsm.getS1State());
}

@Override
public void BelowMinBalance() {
	// TODO Auto-generated method stub
```

```
}

@Override
public void AboveMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void Logout() {
	mdaefsm.setState(mdaefsm.getIdleState());
}

@Override
public void Balance() {
	mdaefsm.output.DisplayBalance();
	mdaefsm.setState(mdaefsm.getOverdrawnState());
}

@Override
public void Withdraw() {

}

@Override
public void WithdrawBelowMinBalance() {
	mdaefsm.output.NoFunds_Msg();
	System.out.println("Below minimum balance");
	mdaefsm.setState(mdaefsm.getOverdrawnState());
}

@Override
public void NoFunds() {
	// TODO Auto-generated method stub

}

@Override
public void Lock() {
	mdaefsm.setState(mdaefsm.getLockedState());
}

@Override
public void IncorrectLock() {
	mdaefsm.output.IncorrectLock_Msg();
}
```

```
	@Override
	public void Unlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectUnlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Suspend() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Close() {
		// TODO Auto-generated method stub

	}

}
```

READY STATE

```
package pkg_Mda_Efsm;

public class StateReady implements States
{
	MDAEFSM mdaefsm = null;

	public StateReady(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
```

```
public void Open() {
	// TODO Auto-generated method stub

}

@Override
public void Login() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLogin() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectPin(int max) {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {
	mdaefsm.output.MakeDeposit();
	mdaefsm.setState(mdaefsm.getReadyState());
}

@Override
public void BelowMinBalance() {

}
```

```
@Override
public void AboveMinBalance() {

}

@Override
public void Logout() {
        mdaefsm.setState(mdaefsm.getIdleState());
}

@Override
public void Balance() {
        mdaefsm.output.DisplayBalance();
        mdaefsm.setState(mdaefsm.getReadyState());
}

@Override
public void Withdraw() {
        mdaefsm.output.MakeWithdraw();
        mdaefsm.setState(mdaefsm.getS1State());
}

@Override
public void WithdrawBelowMinBalance() {

}

@Override
public void NoFunds() {
        mdaefsm.output.NoFunds_Msg();
        mdaefsm.setState(mdaefsm.getReadyState());
}

@Override
public void Lock() {
        mdaefsm.setState(mdaefsm.getLockedState());
}

@Override
public void IncorrectLock() {
        mdaefsm.output.IncorrectLock_Msg();
        mdaefsm.setState(mdaefsm.getReadyState());
}

@Override
public void Unlock() {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectUnlock() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Suspend() {
		mdaefsm.setState(mdaefsm.getSuspendedState());
	}

	@Override
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Close() {
		mdaefsm.setState(mdaefsm.getStopstate());
	}

}
```

STATES INTERFACE

```
package pkg_Mda_Efsm;

public interface States
{
	public void Open();
	public void Login();
	public void IncorrectLogin();
	public void IncorrectPin(int max);
	public void CorrectPinBelowMin();
	public void CorrectPinAboveMin();
	public void Deposit();
	public void BelowMinBalance();
	public void AboveMinBalance();
	public void Logout();
	public void Balance();
	public void Withdraw();
```

```
	public void WithdrawBelowMinBalance();
	public void NoFunds();
	public void Lock();
	public void IncorrectLock();
	public void Unlock();
	public void IncorrectUnlock();
	public void Suspend();
	public void Activate();
	public void Close();

}
```

S1 STATE

```
package pkg_Mda_Efsm;

public class StateS1 implements States
{
	MDAEFSM mdaefsm = null;

	public StateS1(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
	public void Open() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Login() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectLogin() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectPin(int max) {
		// TODO Auto-generated method stub
```

```
}

@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {

}

@Override
public void BelowMinBalance() {
	mdaefsm.setState(mdaefsm.getOverdrawnState());
}

@Override
public void AboveMinBalance() {
	mdaefsm.setState(mdaefsm.getReadyState());
}

@Override
public void Logout() {
	// TODO Auto-generated method stub

}

@Override
public void Balance() {
	// TODO Auto-generated method stub

}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub

}
```

```
@Override
public void WithdrawBelowMinBalance() {
	mdaefsm.output.Penalty();
	mdaefsm.setState(mdaefsm.getOverdrawnState());
}

@Override
public void NoFunds() {
	// TODO Auto-generated method stub

}

@Override
public void Lock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLock() {
	// TODO Auto-generated method stub

}

@Override
public void Unlock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectUnlock() {
	// TODO Auto-generated method stub

}

@Override
public void Suspend() {
	// TODO Auto-generated method stub

}

@Override
public void Activate() {
	// TODO Auto-generated method stub
```

```
	}

	@Override
	public void Close() {
		// TODO Auto-generated method stub

	}

}
```

SUSPENDED STATE

```
package pkg_Mda_Efsm;

public class StateSuspended implements States
{
	MDAEFSM mdaefsm = null;

	public StateSuspended(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
	public void Open() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Login() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectLogin() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectPin(int max) {
		// TODO Auto-generated method stub

	}
```

```
@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {
	// TODO Auto-generated method stub

}

@Override
public void BelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void AboveMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void Logout() {
	// TODO Auto-generated method stub

}

@Override
public void Balance() {
	mdaefsm.output.DisplayBalance();
	mdaefsm.setState(mdaefsm.getSuspendedState());
}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub
```

```
}

@Override
public void WithdrawBelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void NoFunds() {
	// TODO Auto-generated method stub

}

@Override
public void Lock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLock() {
	// TODO Auto-generated method stub

}

@Override
public void Unlock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectUnlock() {
	// TODO Auto-generated method stub

}

@Override
public void Suspend() {
	// TODO Auto-generated method stub

}

@Override
```

```
	public void Activate() {
		mdaefsm.setState(mdaefsm.getReadyState());
	}

	@Override
	public void Close() {
		mdaefsm.setState(mdaefsm.getStopstate());
	}

}
```

STOP STATE

```
package pkg_Mda_Efsm;

public class StopState implements States
{
	MDAEFSM mdaefsm = null;

	public StopState(MDAEFSM mdaefsm) {
		this.mdaefsm= mdaefsm;
	}

	@Override
	public void Open() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Login() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectLogin() {
		// TODO Auto-generated method stub

	}

	@Override
	public void IncorrectPin(int max) {
		// TODO Auto-generated method stub

	}
```

```
@Override
public void CorrectPinBelowMin() {
	// TODO Auto-generated method stub

}

@Override
public void CorrectPinAboveMin() {
	// TODO Auto-generated method stub

}

@Override
public void Deposit() {
	// TODO Auto-generated method stub

}

@Override
public void BelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void AboveMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void Logout() {
	// TODO Auto-generated method stub

}

@Override
public void Balance() {
	// TODO Auto-generated method stub

}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub
```

```
}

@Override
public void WithdrawBelowMinBalance() {
	// TODO Auto-generated method stub

}

@Override
public void NoFunds() {
	// TODO Auto-generated method stub

}

@Override
public void Lock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectLock() {
	// TODO Auto-generated method stub

}

@Override
public void Unlock() {
	// TODO Auto-generated method stub

}

@Override
public void IncorrectUnlock() {
	// TODO Auto-generated method stub

}

@Override
public void Suspend() {
	// TODO Auto-generated method stub

}

@Override
```

```
	public void Activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void Close() {
		// TODO Auto-generated method stub

	}

}
```

OUTPUT

```
package pkg_Output;


import pkg_Abstract_Factory.AbstractFactory;
import pkg_Data_Store.*;
import pkg_Strategy.*;



public class Output
{
	AbstractFactory factory =null;
	DS data_store_obj = null;


	//Constructor for setting the object values
	public Output(AbstractFactory factory,DS dataStore)
	{
      this.factory = factory;
      this.data_store_obj = dataStore;
   }
```

```
//Action that stores data values
public void StoreData()
{
	//String temp_s = data_store_obj.getClass().getName();

	System.out.println(" OUTPUT-> Action StoreData");
	StoreData sd = factory.CreateStoreDataObj();

		sd.StoreData(data_store_obj, data_store_obj, data_store_obj);

}

public void IncorrectId_Msg()
{
	System.out.println(" OUTPUT-> Action IncorrectId_Msg");
	IncorrectIdMsg icd_msg = factory.CreateIncorrectIdMsgObj();
	icd_msg.IncorrectIdMsg();
}

public void IncorrectPin_Msg()
{
	System.out.println(" OUTPUT-> Action IncorrectPin_Msg");
	IncorrectPinMsg icp_msg = factory.CreateIncorrectPinMsgObj();
	icp_msg.IncorrectPinMsg();
}

public void TooManyAttempts_Msg()
{
```

```
        System.out.println(" OUTPUT-> Action TooManyAttempts_Msg");
        TooManyAttemptsMsg tma_msg = factory.CreateTooManyAttemptsMsgObj();
        tma_msg.TooManyAttemptsMsg();
}

public void DisplayBalance()
{
        System.out.println(" OUTPUT-> Action Balance");
        DisplayBalance dbal = factory.CreateDisplayBalanceObj();
        dbal.DisplayBalance(data_store_obj);
}

public void DisplayMenu()
{
        System.out.println(" OUTPUT-> Action Menu");
        DisplayMenu dmenu = factory.CreateDisplayMenuObj();
        dmenu.DisplayMenu();
}

public void MakeDeposit()
{
        System.out.println(" OUTPUT-> Action Deposit");
        MakeDeposit mdp = factory.CreateMakeDepositObj();
        mdp.MakeDeposit(data_store_obj);
}

public void MakeWithdraw()
{
```

```
        System.out.println(" OUTPUT-> Action MakeWithdraw");
        MakeWithdraw mwdr = factory.CreateMakeWithdrawObj();
        mwdr.MakeWithdraw(data_store_obj);
}

public void Penalty()
{
        System.out.println(" OUTPUT-> Action Penalty");
        Penalty plt = factory.CreatePenaltyObj();
        plt.Penalty(data_store_obj);
}

public void IncorrectLock_Msg()
{
        System.out.println(" OUTPUT-> Action IncorrectLock_Msg");
        IncorrectLockMsg ilk_msg = factory.CreateIncorrectLockMsgObj();
        ilk_msg.IncorrectLockMsg();
}

public void IncorrectUnlock_Msg()
{
        System.out.println(" OUTPUT-> Action IncorrectUnlock_Msg");
        IncorrectUnlockMsg iulk_msg = factory.CreateIncorrectUnlockMsgObj();
        iulk_msg.IncorrectUnlockMsg();
}

public void NoFunds_Msg()
{
```

```
                System.out.println(" OUTPUT-> Action NoFunds_Msg");
                NoFundsMsg nfd_msg = factory.CreateNoFundsMsgObj();
                nfd_msg.NoFundsMsg();
        }

        public void PromptForPin()
        {
                System.out.println(" OUTPUT-> Action PromptForPin_Msg");
                PromptForPin pfp_msg = factory.CreatePromptForPinObj();
                pfp_msg.PromptForPin();
        }
}
```

DISPLAY BALANCE

```
package pkg_Strategy;

import pkg_Data_Store.DS;

public abstract class DisplayBalance {
        public abstract void DisplayBalance(DS ds);

}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;

public class DisplayBalance1 extends DisplayBalance {

        @Override
        public void DisplayBalance(DS ds) {
                System.out.println("CURRENT BALANCE IN THE ACCOUNT : "+
((DS_1)ds).getTempB());
        }
}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;

public class DisplayBalance2 extends DisplayBalance {

	@Override
	public void DisplayBalance(DS ds) {
		System.out.println("CURRENT BALANCE IN THE ACCOUNT : "+
((DS_2)ds).getTempB());

	}

}
```

DISPLAY MENU

```
package pkg_Strategy;

public abstract class DisplayMenu {
	public abstract void DisplayMenu();

}
```

```
package pkg_Strategy;

public class DisplayMenu1 extends DisplayMenu{

	@Override
	public void DisplayMenu() {

		System.out.println("Deposit");
		System.out.println("Withdraw");
		System.out.println("Balance");
		System.out.println("Lock");
		System.out.println("Unlock");

	}

}
```

```
package pkg_Strategy;


public class DisplayMenu2 extends DisplayMenu{
```

```
	@Override
	public void DisplayMenu() {
		System.out.println("Deposit");
		System.out.println("Withdraw");
		System.out.println("Balance");
		System.out.println("suspend");
		System.out.println("activate");
		System.out.println("close");
	}

}
```

INCORRECT ID MESSAGE

```
package pkg_Strategy;

public abstract class IncorrectIdMsg {
	public abstract void IncorrectIdMsg();

}
```

```
package pkg_Strategy;

public class IncorrectIdMsg1 extends IncorrectIdMsg{

	@Override
	public void IncorrectIdMsg() {
		System.out.println("INCORRECT LOGIN ID");
	}

}
```

```
package pkg_Strategy;

public class IncorrectIdMsg2 extends IncorrectIdMsg{

	@Override
	public void IncorrectIdMsg() {
		System.out.println("INCORRECT LOGIN ID");
	}

}
```

INCORRECT LOCK MESSAGE

```
package pkg_Strategy;

public abstract class IncorrectLockMsg {
	public abstract void IncorrectLockMsg();

}
```

```
package pkg_Strategy;

public class IncorrectLockMsg1 extends IncorrectLockMsg {

	@Override
	public void IncorrectLockMsg() {
		System.out.println("INCORRECT LOCK PIN");
	}


}
```

```
package pkg_Strategy;

public class IncorrectLockMsg2 extends IncorrectLockMsg{

	@Override
	public void IncorrectLockMsg() {
		System.out.println("INCORRECT LOCK PIN");
	}

}
```

INCORRECT PIN MESSAGE

```
package pkg_Strategy;

public abstract class IncorrectPinMsg {
	public abstract void IncorrectPinMsg();

}
```

```
package pkg_Strategy;

public class IncorrectPinMsg1 extends IncorrectPinMsg {

	@Override
	public void IncorrectPinMsg() {
```

```
		System.out.println("INCORRECT PIN");
	}

}
```

```
package pkg_Strategy;

public class IncorrectPinMsg2 extends IncorrectPinMsg {

	@Override
	public void IncorrectPinMsg() {
		System.out.println("INCORRECT PIN");
	}

}
```

INCORRECT UNLOCK MESSAGE

```
package pkg_Strategy;

public abstract class IncorrectUnlockMsg {
	public abstract void IncorrectUnlockMsg();

}
```

```
package pkg_Strategy;

public class IncorrectUnlockMsg1 extends IncorrectUnlockMsg{

	@Override
	public void IncorrectUnlockMsg() {
		System.out.println("INCORRECT UNLOCK PIN");
	}

}
```

```
package pkg_Strategy;

public class IncorrectUnlockMsg2 extends IncorrectUnlockMsg{

	@Override
	public void IncorrectUnlockMsg() {
		System.out.println("INCORRECT UNLOCK PIN");
	}
```

```
}
```

MAKE DEPOSIT

```
package pkg_Strategy;

public class IncorrectUnlockMsg2 extends IncorrectUnlockMsg{

	@Override
	public void IncorrectUnlockMsg() {
		System.out.println("INCORRECT UNLOCK PIN");
	}

}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;

public class MakeDeposit1 extends MakeDeposit{

	@Override
	public void MakeDeposit(DS ds) {
		((DS_1)ds).setTempB((((DS_1)ds).getTempB()) + (((DS_1) ds).getTempD()));
	}

}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;


public class MakeDeposit2 extends MakeDeposit{

	@Override
	public void MakeDeposit(DS ds) {
		((DS_2)ds).setTempB((((DS_2)ds).getTempB()) + (((DS_2) ds).getTempD()));
	}

}
```

MAKE WITHDRAW

```
package pkg_Strategy;

import pkg_Data_Store.DS;

public abstract class MakeWithdraw {
	public abstract void MakeWithdraw(DS ds);

}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;

public class MakeWithdraw1 extends MakeWithdraw{

	@Override
	public void MakeWithdraw(DS ds) {
		((DS_1)ds).setTempB((((DS_1)ds).getTempB()) - (((DS_1) ds).getTempW()));
	}

}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;


public class MakeWithdraw2 extends MakeWithdraw{

	@Override
	public void MakeWithdraw(DS ds) {
		((DS_2)ds).setTempB((((DS_2)ds).getTempB()) - (((DS_2) ds).getTempW()));
	}

}
```

NO FUNDS MESSAGE

```
package pkg_Strategy;

public abstract class NoFundsMsg {
	public abstract void NoFundsMsg();

}
```

```
package pkg_Strategy;

public class NoFundsMsg1 extends NoFundsMsg{

	@Override
	public void NoFundsMsg() {
		System.out.println("NO FUNDS");
	}

}
```

```
package pkg_Strategy;

public class NoFundsMsg2 extends NoFundsMsg{

	@Override
	public void NoFundsMsg() {
		System.out.println("NO FUNDS");
	}

}
```

PENALTY

```
package pkg_Strategy;

import pkg_Data_Store.DS;

public abstract class Penalty {
	public abstract void Penalty(DS ds);

}
```

```
package pkg_Strategy;


import pkg_Data_Store.DS;

import pkg_Data_Store.DS_1;


public class Penalty1 extends Penalty{
```

```
	@Override
	public void Penalty(DS ds) {
		((DS_1)ds).setTempB((((DS_1) ds).getTempB()) - 20);
	}

}
```

```
package pkg_Strategy;

import pkg_Data_Store.DS;
import pkg_Data_Store.DS_2;

public class Penalty2 extends Penalty{

	@Override
	public void Penalty(DS ds) {
		((DS_2)ds).setTempB((((DS_2)ds).getTempB()) - (((DS_2) ds).getTempW())-
20);
	}

}
```

PROMPT FOR PIN

```
package pkg_Strategy;

public abstract class PromptForPin {
	public abstract void PromptForPin();

}
```

```
package pkg_Strategy;
```

```
public class PromptForPin1 extends PromptForPin {

	@Override
	public void PromptForPin() {
		System.out.println("ENTER PIN TO CONTINUE");
	}

}
```

```
package pkg_Strategy;

public class PromptForPin2 extends PromptForPin{

	@Override
	public void PromptForPin() {
		System.out.println("ENTER PIN TO CONTINUE");
	}

}
```

STORE DATA

```
package pkg_Strategy;

import pkg_Data_Store.DS;

public abstract class StoreData
{
	public abstract void StoreData(DS ds1, DS ds2, DS ds3);
}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;

public class StoreData1 extends StoreData
{
	public void StoreData(DS ds1, DS ds2, DS ds3) {
		((DS_1)ds1).setP(((DS_1)ds1).tempP);
		((DS_1)ds2).setY(((DS_1)ds2).tempY);
		((DS_1)ds3).setA(((DS_1)ds3).tempB);

	}
}
```

```
package pkg_Strategy;

import pkg_Data_Store.*;

public class StoreData2 extends StoreData
{
	public void StoreData(DS ds1, DS ds2, DS ds3) {
		((DS_2)ds1).setP(((DS_2)ds1).tempP);
		((DS_2)ds2).setY(((DS_2)ds2).tempY);
		((DS_2)ds2).setA(((DS_2)ds2).tempB);

	}
}
```

TOO MANY ATTEMPTS

```
package pkg_Strategy;

public abstract class TooManyAttemptsMsg {
	public abstract void TooManyAttemptsMsg();

}
```

```
package pkg_Strategy;

public class TooManyAttemptsMsg1 extends TooManyAttemptsMsg{

	@Override
	public void TooManyAttemptsMsg() {
		System.out.println("TOO MANY ATTEMPTS");
	}

}
```

```
package pkg_Strategy;

public class TooManyAttemptsMsg2 extends TooManyAttemptsMsg{

	@Override
	public void TooManyAttemptsMsg() {
		System.out.println("TOO MANY ATTEMPTS");
	}
```

}