Software System Architecture-Design Document

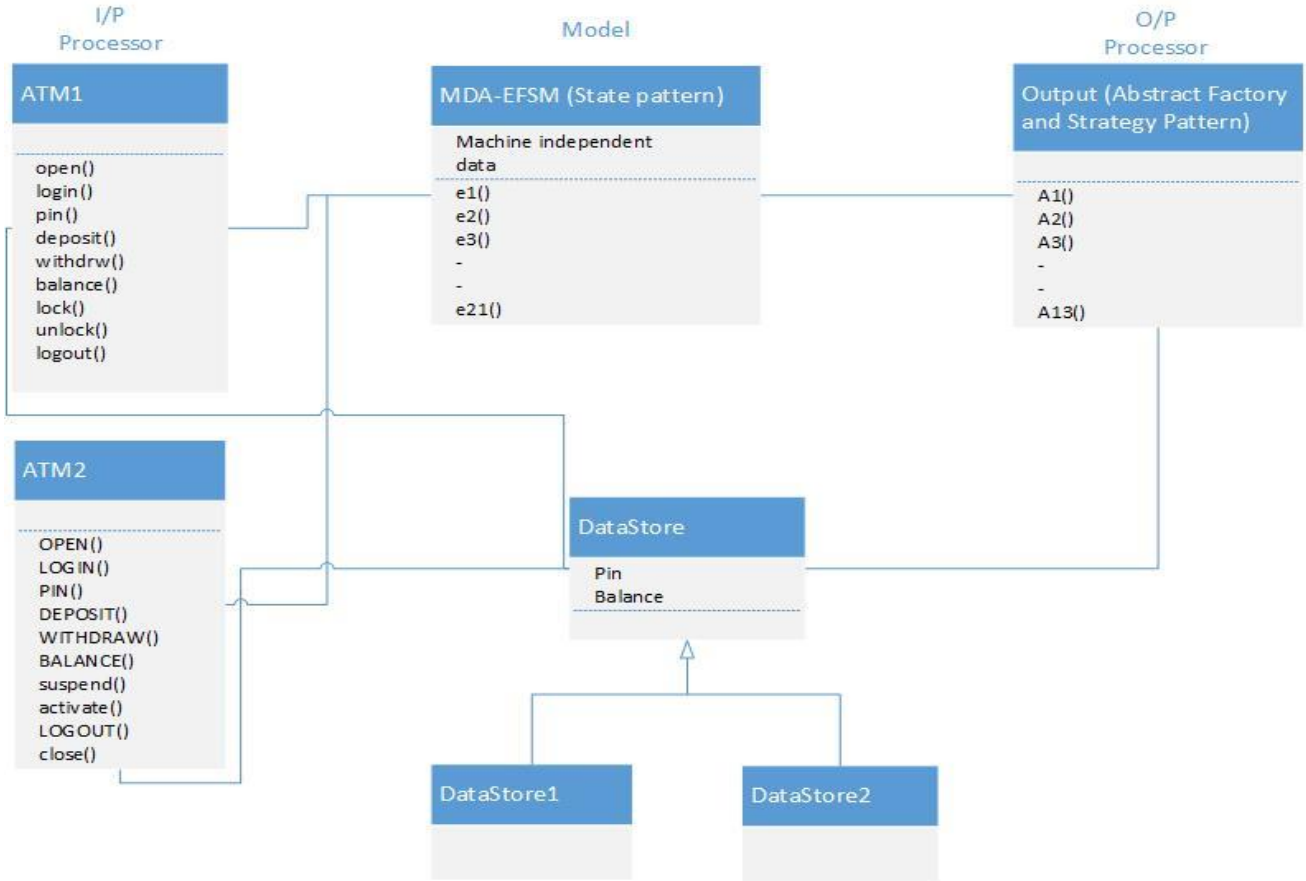
ATM SYSTEM

Aditya Kumar (A20331971, akumar61@hawk.iit.edu) | CS586 | May 2, 2016

1. Model Driven Architecture of the ATM Components

1.1. General MDA Architecture of the ATM Components:

This project implements two ATMs using Model Driven Architecture. The ATMs differ in their user interface specifications but share the same behaviors like depositing money, withdrawing money and checking the balance. The main goal of this project is to learn the art of separating the platform specific portion of the system from the platform independent portion. The general outline of the implementation is to use Model Driven Architecture (MDA) as the overall design and use Extended Finite State Machine (EFSM) to capture sates in which the model can be while performing some action or being in idle state. The project implements the Output processor using Strategy Pattern, Model processor using the State Pattern and MDA-EFSM Actions using Abstract Factory Method.



1.2. MDA-EFSM model for the ATM Components:

The purpose of MDA-EFSM is to capture the meta-behavior of all ATMs.

1.2.1. List of Events of the MDA-EFSM:

i. Open()
ii. Login()
iii. IncorrectLogin()
iv. IncorrectPin(int max)
v. CorrectPinBelowMin()
vi. CorrectPinAboveMin()
vii. Deposit()
viii. BelowMinBalance()
ix. AboveMinBalance()
x. Logout()
xi. Balance()
xii. Withdraw()
xiii. WithdrawBelowMinBalance()
xiv. NoFunds()
xv. Lock()
xvi. IncorrectLock()
xvii. Unlock()
xviii. IncorrectUnlock()
xix. Suspend()
xx. Activate()
xxi. Close()

1.2.2. List of Actions of 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: IncorrectLockMsg()
// displays incorrect lock message
A12: IncorrectUnlockMsg()
// displays incorrect unlock message
A13: NoFundsMsg()
// Displays no sufficient funds message

1.3. Pseudo Code of the ATM Classes:

ATM Machine Class

Purpose: User interface or input class

Responsibilities: Responsible for storing data into temporary data store and interface between user and system.

Collaborators: MDA-EFSM, Data Store

Operations of the Input Processor (ATM-1)

```
open(string p, string y, float a) {
        //Store p,y amd a into temporary 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();
}



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



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



login(string y) {
	if(y==ds->id){
		m->login();
	}
	else
	{
		m->IncorrectLogin();
	}
}



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



lock(string x) {
	if(ds->pin==x) m->Lock();
	else m->IncorrectLock();
}



unlock(string x) {
```

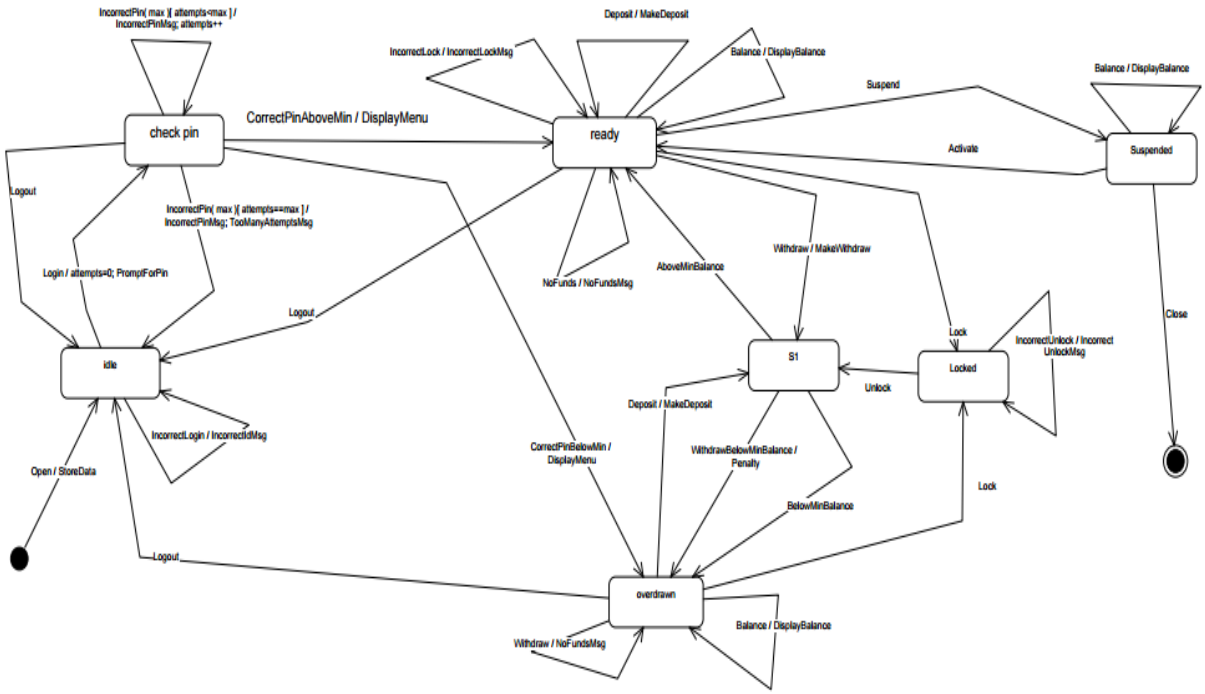
```
        if(x==ds->pin) {
                m->unlock();
                if(ds->balance>500)
                        m->AboveMinBalance();
                else m->BelowMinBalance();
        }
        else m->IncorrectUnlock();
}
```

Operations of the Input Processor (ATM-2)

```
OPEN(int p, int y, int a) {
//Store p,y and a into temporary 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();
}


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


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

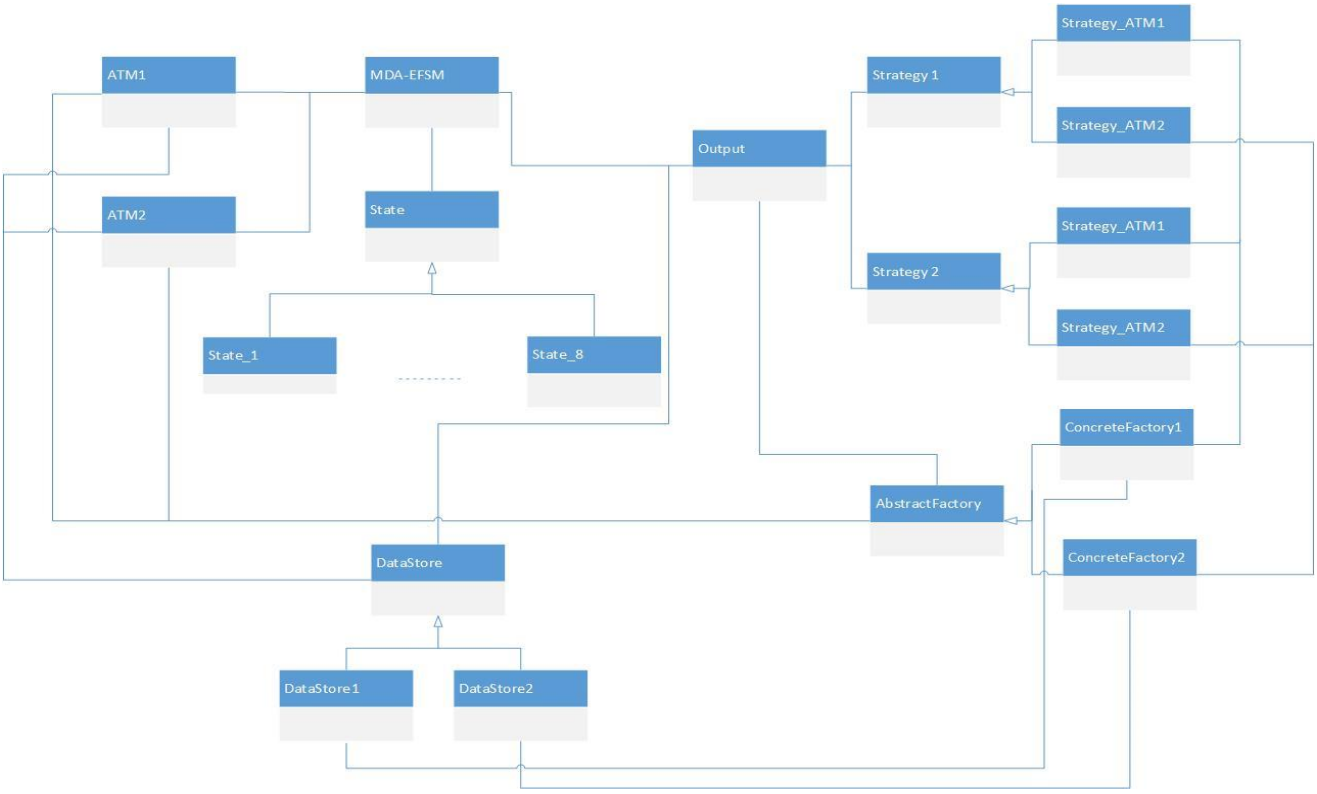
```
LOGIN(int y) {
	if(y==ds->id){
		m->Login();
	}
	else
	{
		m->IncorrectLogin();
	}
}



WITHDRAW(int w) {
	ds-> temp_w=w;
	if(ds->balance>0){
		m->withdraw();
	}
	else m->NoFunds ();
}



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



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



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

1.4. State Diagram of MDA-EFSM:



2. Class Diagram of the MDA-ATM Components

Figure 1: High-Level Class Diagram



This is a high level representation of the connections between the classes.

Figure 2: State Pattern Class Diagram :: shows the associations of ATM1, ATM2, DataStore and MDA-EFSM and State

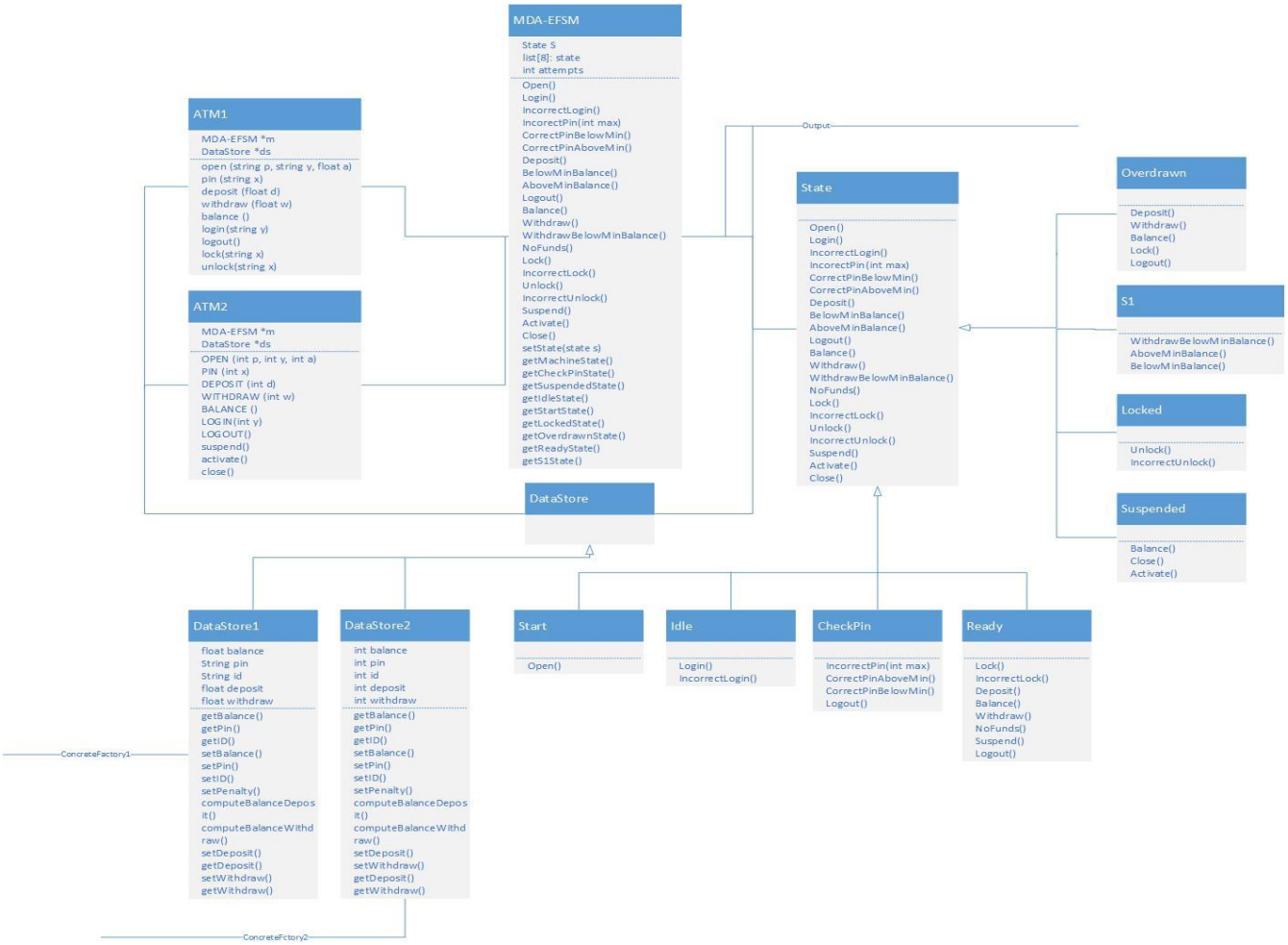


Figure 3: Abstract and Strategy Pattern Class Diagram :: shows the associations of Abstract and ConcreteFactory classes and Strategy classes (StoreData, IncorrectPinMsg, IncorrectIdMsg and DisplayMenu) and Output.

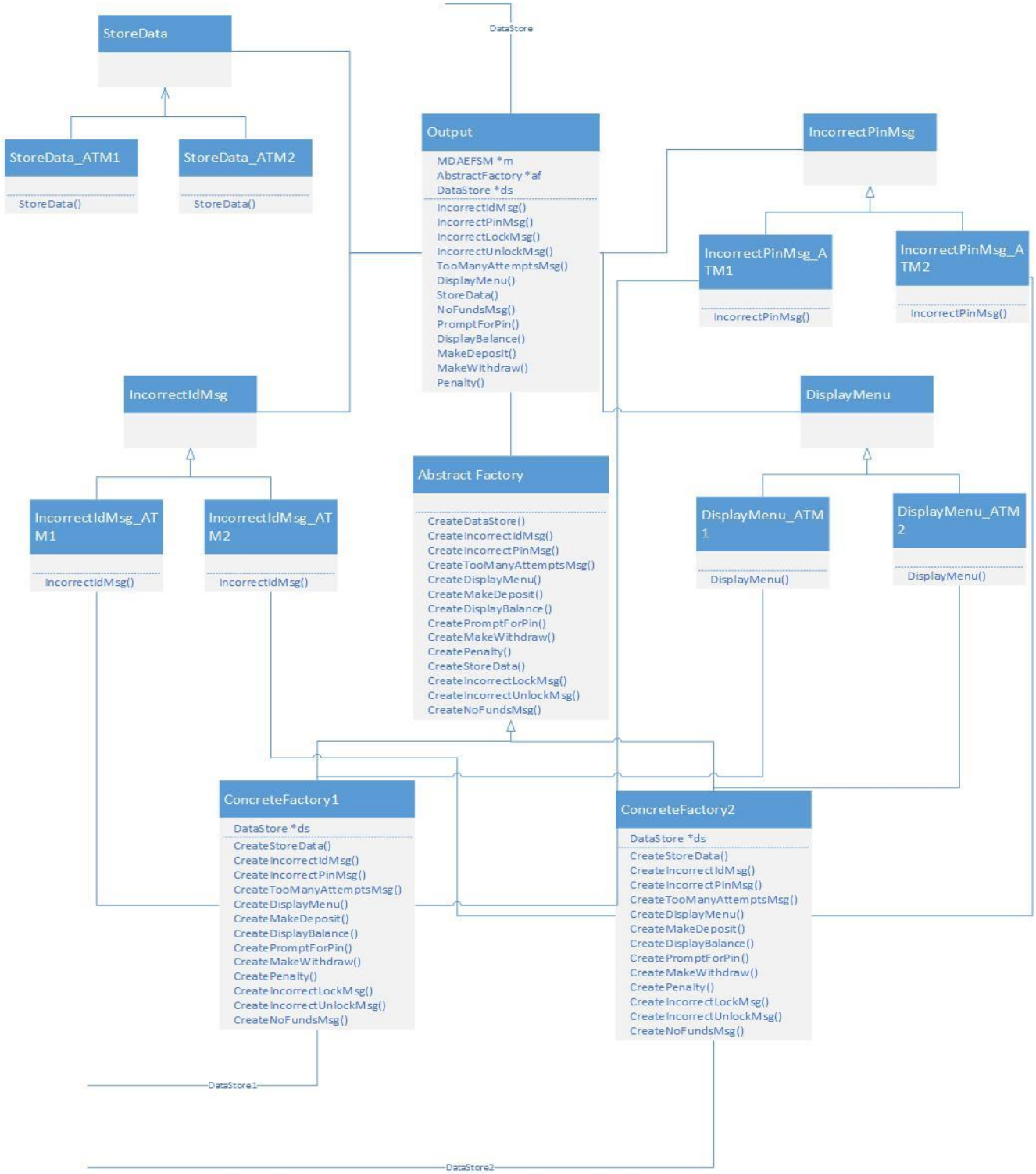


Figure 4: Abstract and Strategy Pattern Class Diagram :: shows the associations of Abstract and ConcreteFactory classes and Strategy classes (MakeWithdraw, MakeDeposit, TooManyAttempts and DisplayBalance) and Output.

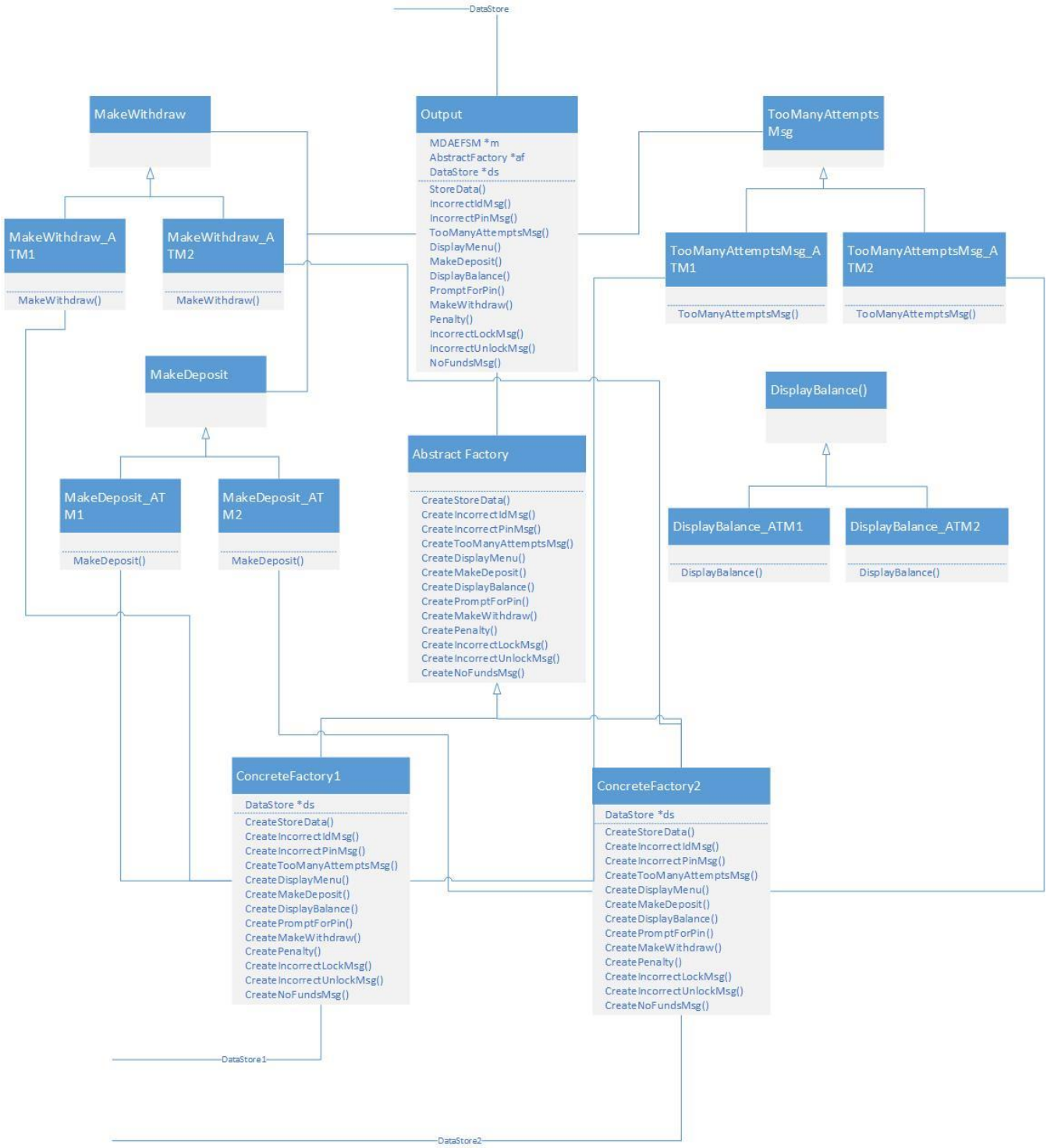


Figure 5: Abstract and Strategy Pattern Class Diagram :: shows the associations of Abstract and ConcreteFactory classes and Strategy classes (PromptForPin, Penalty and NoFundsMsg) and Output.

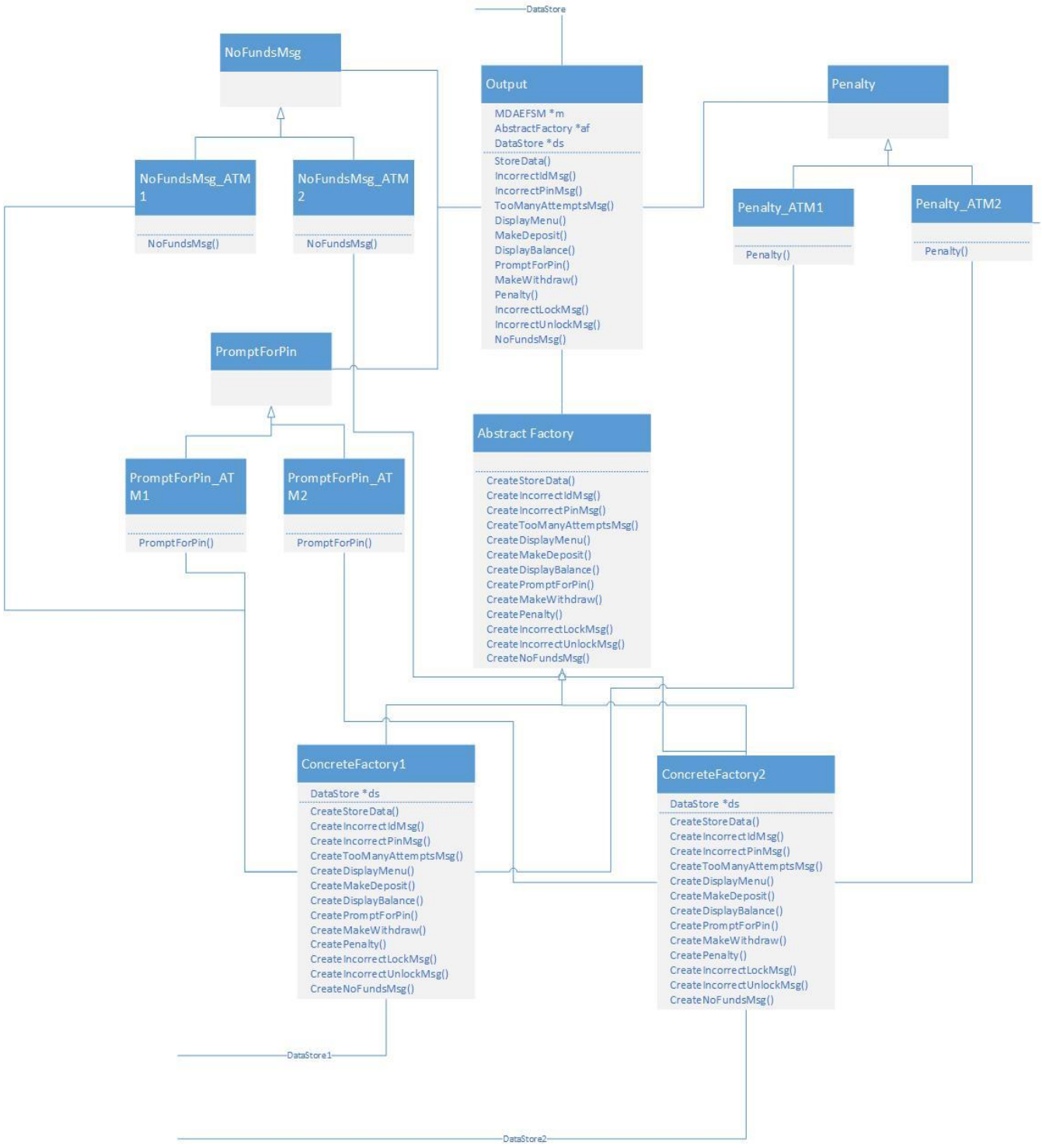
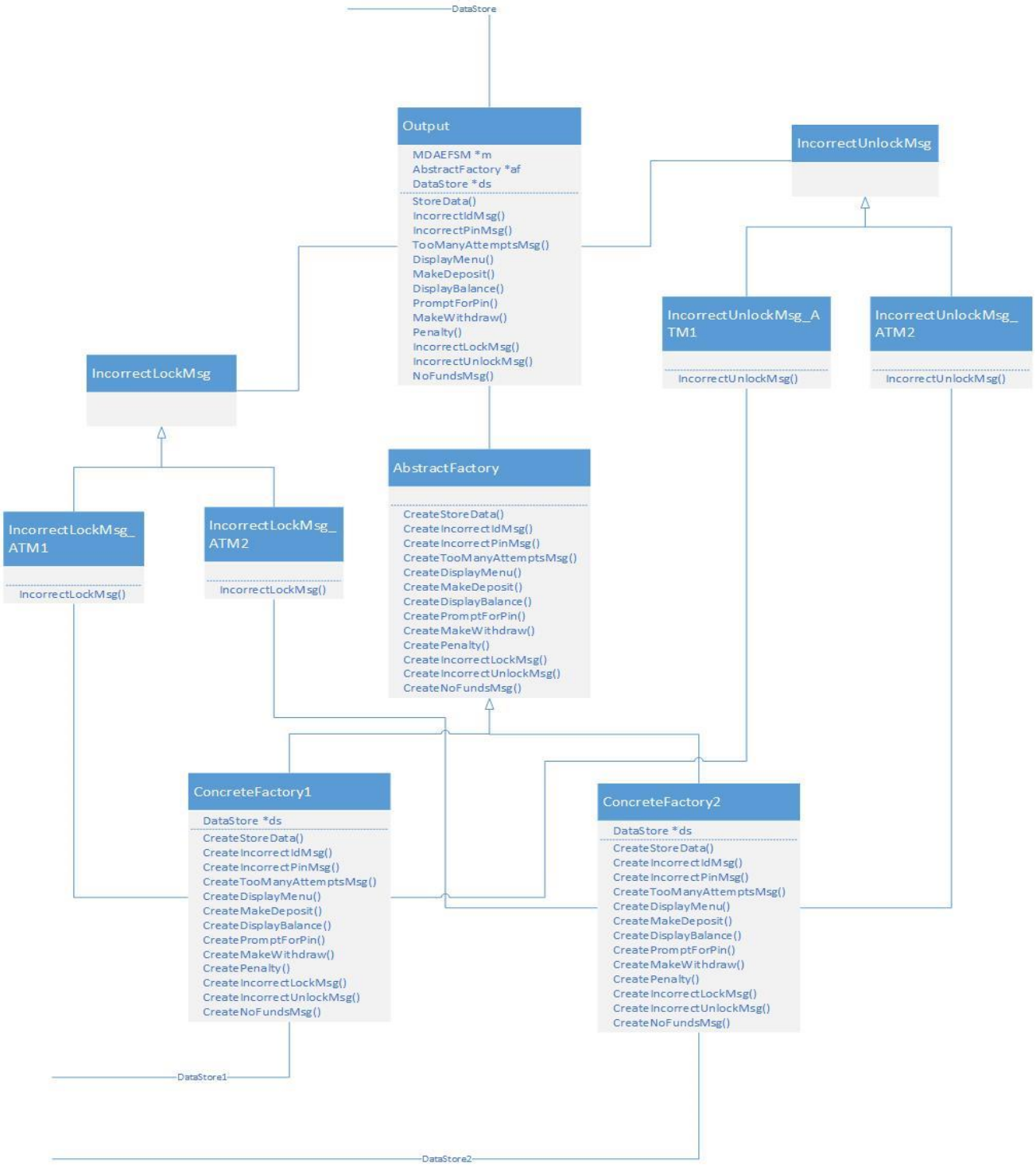


Figure 6: Abstract and Strategy Pattern Class Diagram :: shows the associations of Abstract and ConcreteFactory classes and Strategy classes (IncorrectLockMsg and IncorrectUnlockMsg) and Output.



3. Operations and Attributes:

3.1. DataStore Class

Purpose: The DataStore class is required by ATM for storing temporary data and Output class by AbstractFactory and Strategy Pattern for computing and storing data like id, pin and balance permanently.

Collaborators: ATM1, ATM2, Output, ConcreteFactory1, ConcreteFactory2 and Strategy Classes for implementing actions.

Attributes:

DataStore1

```
/* Temporary Storage Variables */
public float tempBalance;
public String tempPin;
public String tempID;
public float tempDeposit;
public float tempWithdraw;


/* Permanent Storage Variables */
public float balance;
public String pin;
public String id;
public float deposit;
public float withdraw;
```

DataStore2

```
/* Temporary Storage Variables */
public int tempBalance;
public int tempPin;
public int tempID;
public int tempDeposit;
public int tempWithdraw;


/* Permanent Storage Variables */
public int balance;
public int pin;
public int id;
public int deposit;
public int withdraw;
```

3.2. MDA-EFSM Class

Purpose: The MDA-EFSM class manages state machine for State pattern. MDA-EFSM is implemented in a way that it is independent of different types of atm machines.

Responsibilities: Captures platform independent behavior.

Collaborators: ATM1, ATM2 and Output.

Attributes:

```
State efsmState = null;

State List[8] ;
public int attempts;

AbstractFactory factory =null;
Output output = null;
```

3.3. State Class

Purpose: The State class manages states for the ATM Machine.

Responsibilities: Captures states possible.

Collaborators: ATM1, ATM2 and Output.

b1. StartState class – contains Open() action.
b2. IdleState class – contains Login() and IncorrectLogin() actions.
b3. CheckPinState class – contains IncorrectPin(), CorrectPinAboveMin(), CorrectPinBelowMin() and Logout() actions.
b4. ReadyState class – contains IncorrectLock(), Deposit(), Balance(), Withdraw(), NoFunds(), Suspend() and Logout() actions.
b5. OverdrawnState class – contains Deposit(), Balance(), Withdraw(), Lock() and Logout() actions.
b6. LockedState class – contains Unlock() and IncorrectUnlock() actions.
b7. S1State class – contains AboveMinBalance(), BelowMinBalance() and WithdrawBelowMinBalance() actions.
b8. SuspendedState class – contains Balance(), Close() and Activate() actions.

Attributes:

```
MDAEFSM *m;

Output *op;
```

3.4. Output Class

Purpose: The Output class calls actions of the MDA-EFSM.

Responsibilities: Captures states possible.

Collaborators: All strategy abstract classes, DataStore, Abstract Factory class.

Attributes:

```
AbstractFactory factory = null;

DataStore dataStore = null;
```

3.5. Abstract and Concrete Factory Class

Purpose: To group strategies for a particular ATM machine.

Responsibilities: Captures states possible.

Collaborators: All respective strategy classes needed by ATM1. ATM2 and DataStore needed by them.

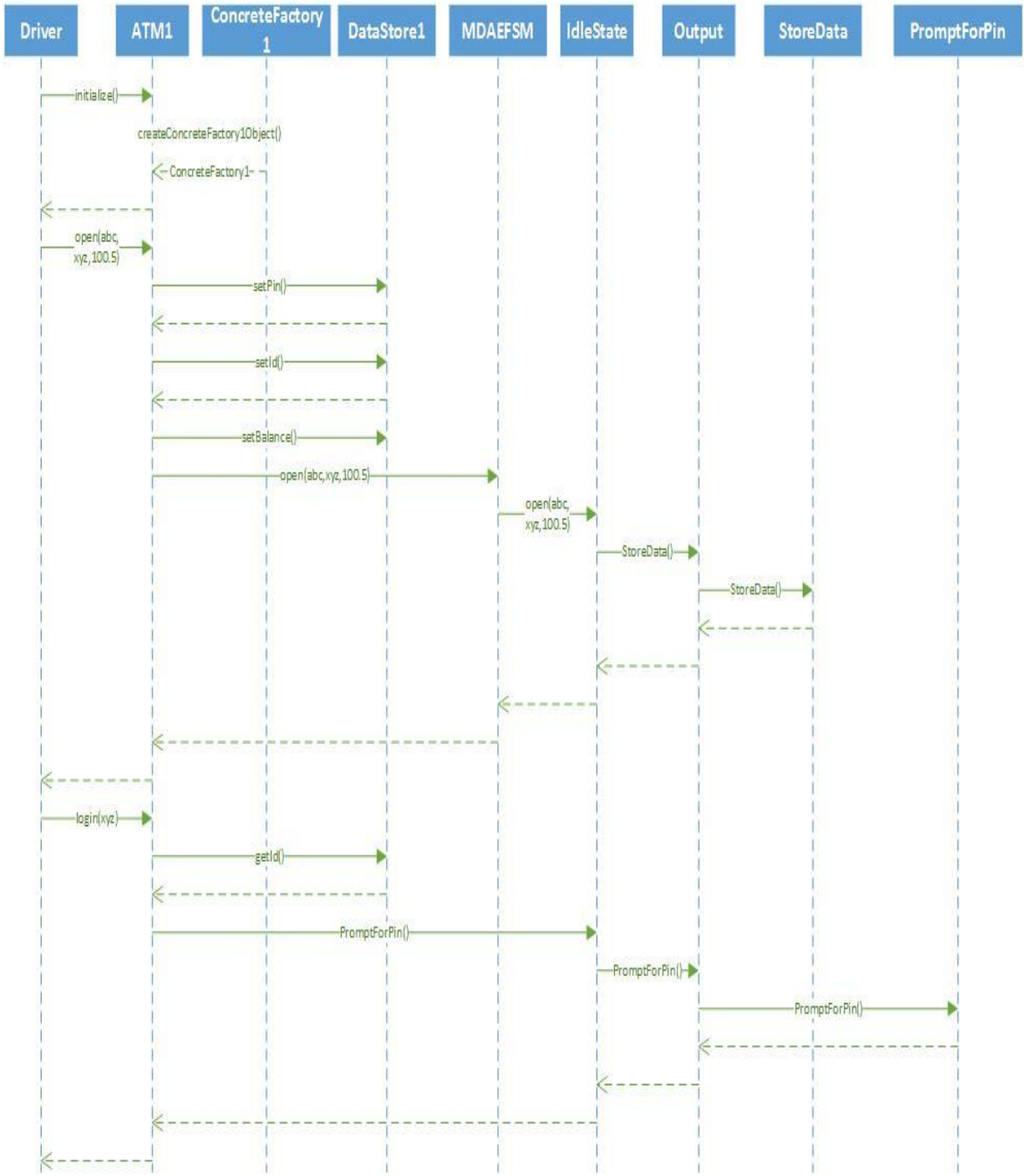
3.6. Strategy Class

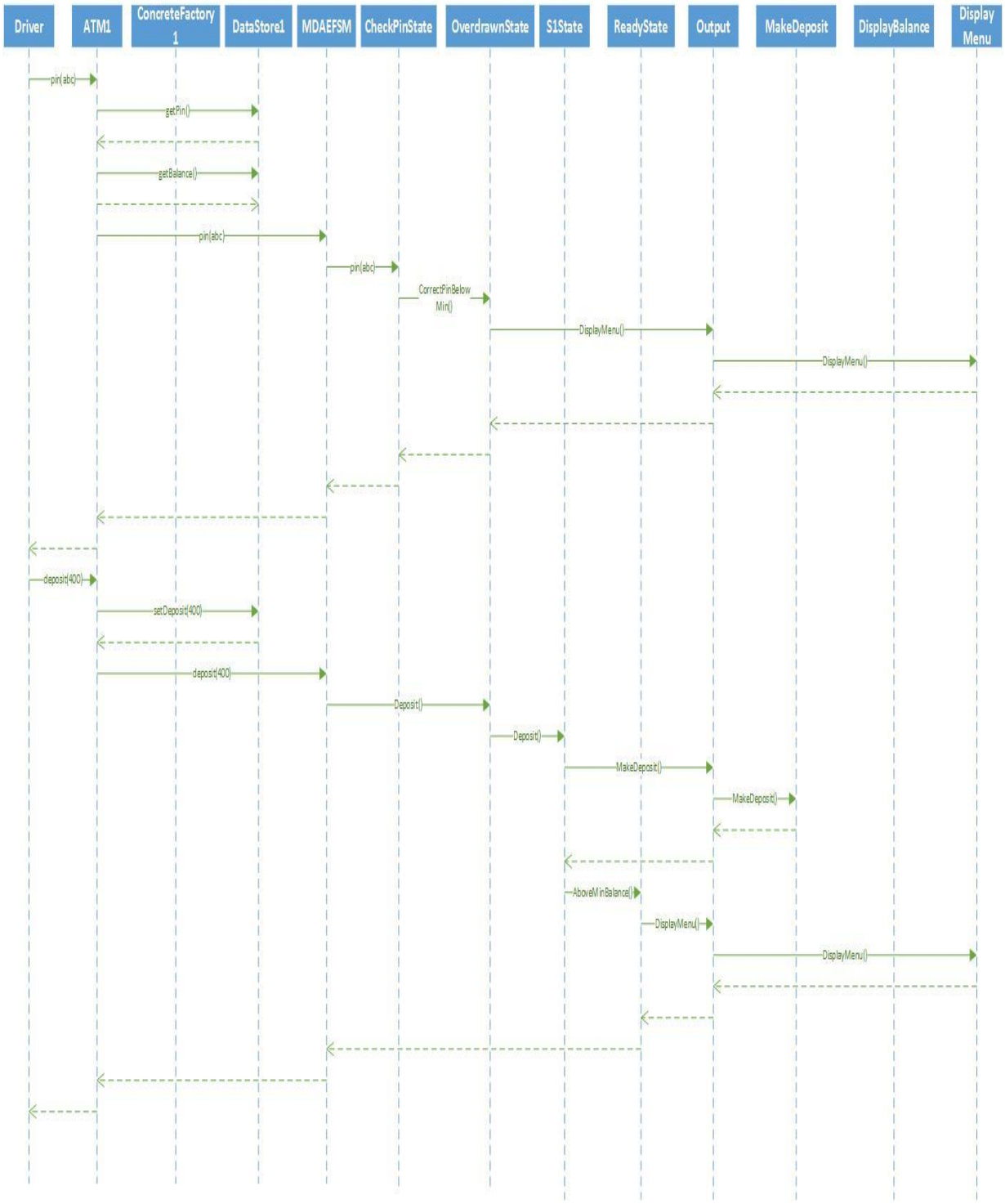
Purpose: Implement different functionalities through actions.

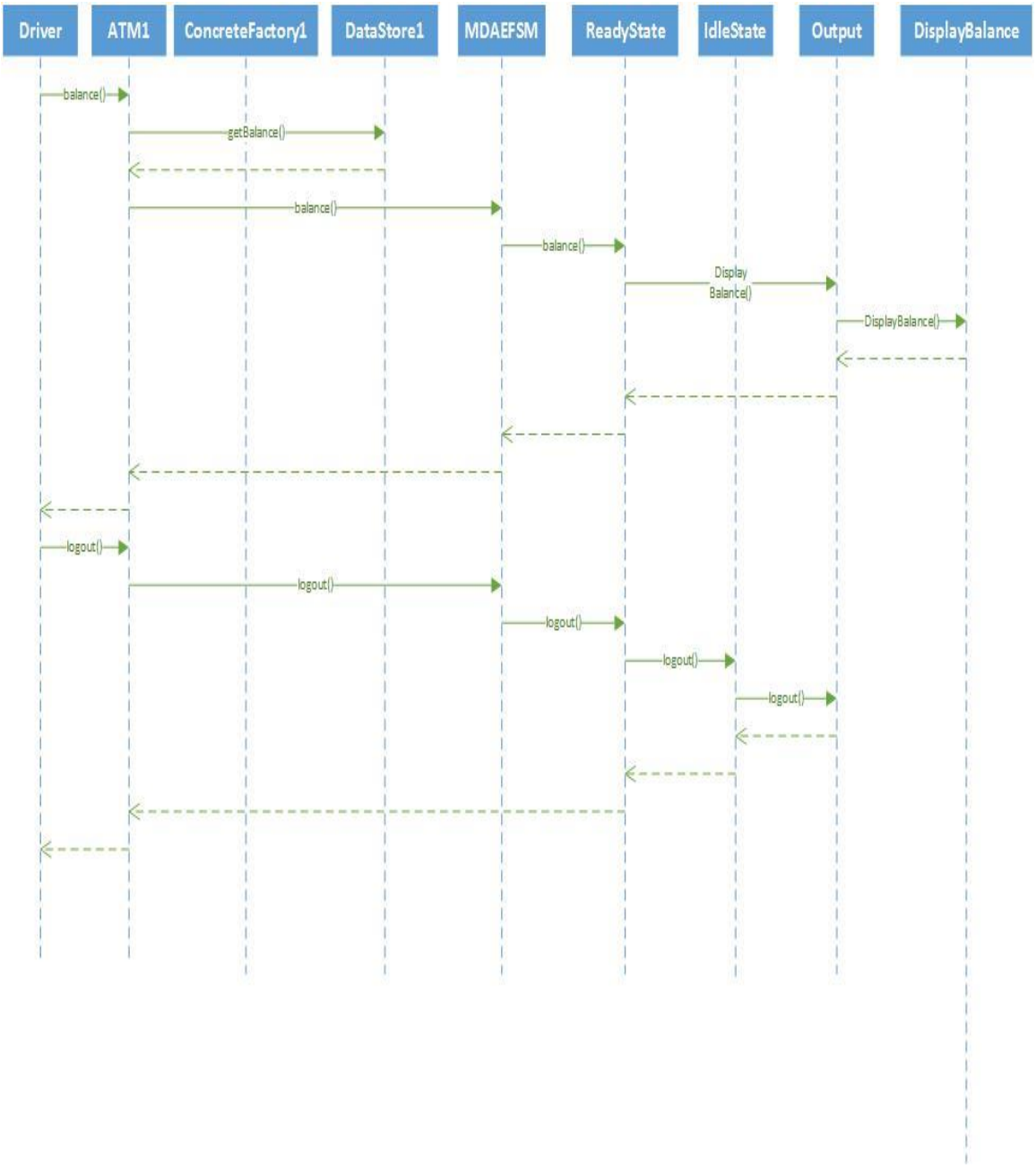
Collaborators: Concrete factory classes.

4. Dynamics:

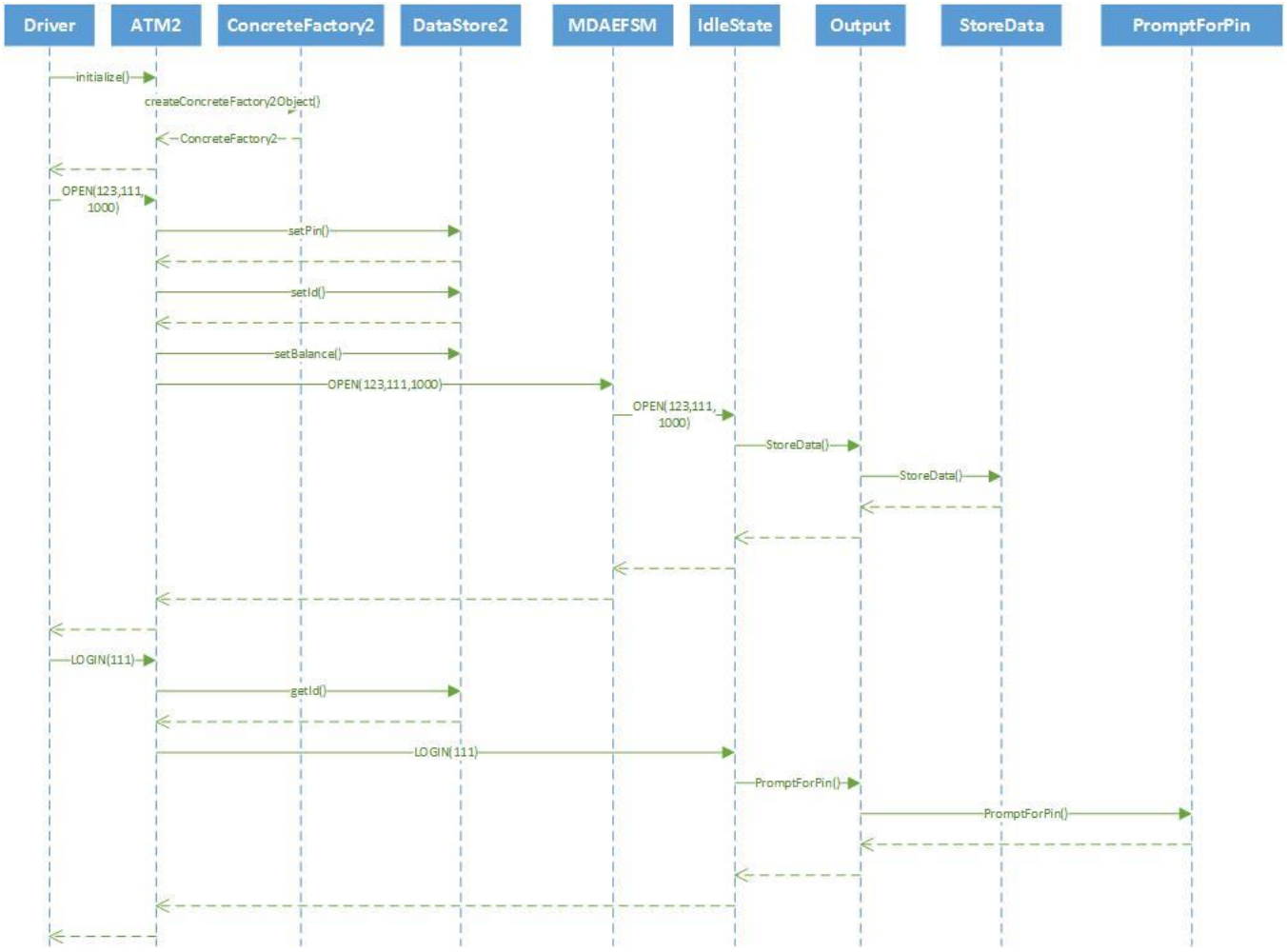
4.1. **Scenario-I**: How the deposit is made in the ACCOUNT-1 component, i.e., the following sequence of operations is issued: open(abc,xyz,100.5), login(xyz), pin(abc), deposit(400), balance(), logout()

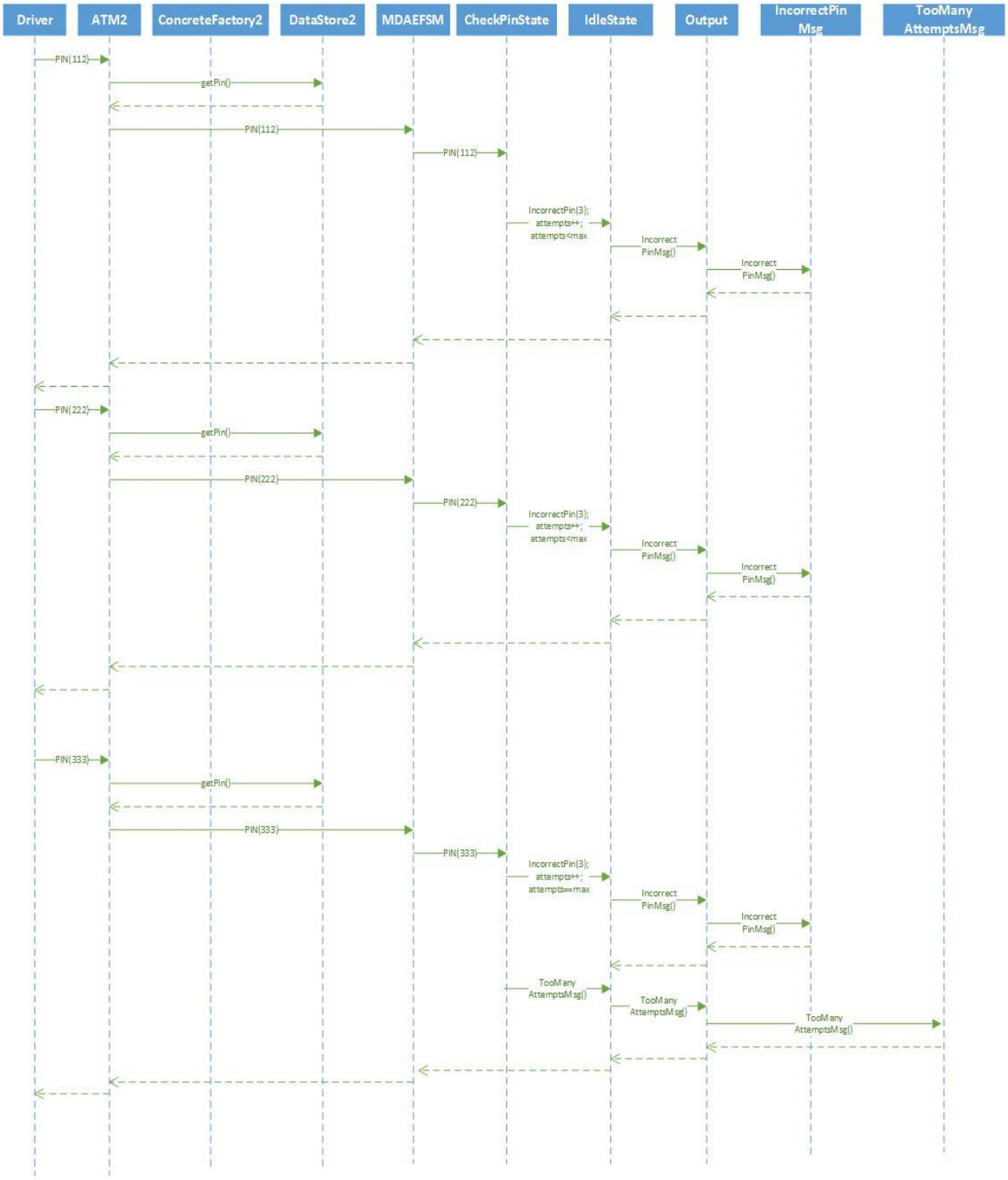



Driver
ATM1
ConcreteFactory 1
DataStore1
MDAEFSM
CheckPinState
OverdrawnState
S1State
ReadyState
Output
MakeDeposit
DisplayBalance
Display Menu
pin(abc)
getPin()
getBalance()
pin(abc)
pin(abc)
CorrectPinBelow Min()
DisplayMenu()
DisplayMenu()
deposit(400)
setDeposit(400)
deposit(400)
Deposit()
Deposit()
MakeDeposit()
MakeDeposit()
AboveMinBalance()
DisplayMenu()
DisplayMenu()


Driver
ATM1
ConcreteFactory1
DataStore1
MDAEFSM
ReadyState
IdleState
Output
DisplayBalance
balance()
getBalance()
balance()
balance()
Display Balance()
DisplayBalance()
logout()
logout()
logout()
logout()
logout()

4.2. **Scenario-II**: How an incorrect pin is entered three times in the ACCOUNT-2 component, i.e., the following sequence of operations is issued: OPEN(123,111,1000), LOGIN(111), PIN(112), PIN(222), PIN(333)




Driver
ATM2
ConcreteFactory2
DataStore2
MDAEFSM
CheckPinState
IdleState
Output
IncorrectPin Msg
TooMany AttemptsMsg
PIN(112)
getPin()
PIN(112)
PIN(112)
IncorrectPin(3); attempts++; attempts<max
Incorrect PinMsg()
Incorrect PinMsg()
PIN(222)
getPin()
PIN(222)
PIN(222)
IncorrectPin(3); attempts++; attempts<max
Incorrect PinMsg()
Incorrect PinMsg()
PIN(333)
getPin()
PIN(333)
PIN(333)
IncorrectPin(3); attempts++; attempts==max
Incorrect PinMsg()
Incorrect PinMsg()
TooMany AttemptsMsg()
TooMany AttemptsMsg()
TooMany AttemptsMsg()

5. Source Code And Patterns:

5.1 Driver.java

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

import MDA_EFSM.MDAEFSM;
import Output.Output;
import ATM_Machine.ATM1;
import ATM_Machine.ATM2;
import AbstractFactory.ConcreteFactory1;
import AbstractFactory.ConcreteFactory2;


/*
 * CLASS : Driver ( Main function Program)
 */
public class Driver
{
   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(" ******** Select ATM Machine to operate *******" );
               System.out.println("         1. ATM Machine - 1" );
               System.out.println("         2. ATM Machine - 2" );

               input = bufferedReader.readLine( );
               if(input.equalsIgnoreCase("1"))
               {

               ConcreteFactory1 factory = new ConcreteFactory1( );
               Output output = new Output(factory,factory.GetDataStore( ));
               MDAEFSM mdaefsm = new MDAEFSM(factory,output);
               ATM1 atm = new ATM1(mdaefsm,factory.GetDataStore( ));

               float balance,deposit,withdraw;
               String PIN, ID;


            while (true)
            {

               System.out.println("  Enter Operation from the list below: ");

               System.out.println("*************************************");
                     System.out.println(" ATM Machine-1 MENU of Operations" );
                     System.out.println("         1. open(String, String, float)" );
```

```
                System.out.println("        2. login(String)" );
                System.out.println("        3. pin(String)" );
                System.out.println("        4. deposit(deposit)" );
                System.out.println("        5. withdraw(withdraw)" );
                System.out.println("        6. balance(balance)" );
                System.out.println("        7. lock(PIN)" );
                System.out.println("        8. unlock(PIN)" );
                System.out.println("        9. logout( )" );
                System.out.println("****************************************");

                        input = bufferedReader.readLine( );

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

            choice = Integer.parseInt(input);

            switch(choice)
            {

                    case 1:   //open
                            System.out.println("\n  Operation:  open(String PIN, String
ID, float balance)");
                            System.out.println("   Enter value of the parameter balance:");
                            input = bufferedReader.readLine( );
                            balance = Float.parseFloat(input);

                            System.out.println("   Enter value of the parameter PIN:");
                            PIN = bufferedReader.readLine( );

        System.out.println("   Enter value of the parameter ID:");
        ID = bufferedReader.readLine( );


        atm.open(PIN, ID, balance);

                            break;

                    case 2:  //login
                            System.out.println("  Operation:  login(String y)");
                            System.out.println("  Enter value of ID:");
                            ID = bufferedReader.readLine( );
                            atm.login(ID);
                            break;

      case 3:  //pin
          System.out.println("  Operation:  pin(String x)");
          System.out.println("  Enter value of PIN:");
          PIN = bufferedReader.readLine( );
          atm.pin(PIN);
          break;

                    case 4:  //deposit
                            System.out.println("  Operation:  deposit(float d)");
```

```
                                        System.out.println("  Enter value of the parameter Deposit:");
                                        input = bufferedReader.readLine( );
                                        deposit = Float.parseFloat(input);
                                        atm.deposit(deposit);
                                        break;

                                case 5:  // withdraw
                                        System.out.println("  Operation:  withdraw(float w)");
                                        System.out.println("  Enter value of the parameter
Withdraw:");
                                        input = bufferedReader.readLine( );
                                        withdraw = Float.parseFloat(input);
                                        atm.withdraw(withdraw);
                                        break;

                                case 6:  // balance
                                        System.out.println("  Operation:  balance( )");
                                        atm.balance( );
                                        break;

                                case 7:  // lock
                                        System.out.println("  Operation:  lock(String PIN)");
                                        System.out.println("  Enter value of the parameter PIN:");
                                        PIN = bufferedReader.readLine( );
                                        atm.lock(PIN);
                                        break;

                                case 8:  // unlock
                                        System.out.println("  Operation:  unlock(String PIN)");
                                        System.out.println("  Enter value of the parameter PIN:");
                                        PIN = bufferedReader.readLine( );
                                        atm.unlock(PIN);
                                        break;

                                case 9:  // logout
                                        System.out.println("  Operation:  logout( )");
                                        atm.logout( );
                                        break;

                                    default:
                                        System.out.println("Invalid Choice");
                                        break;
                    }
                }
                System.out.println("Thanks for using ATM Machine - 1" );
                    }

                    else if(input.equalsIgnoreCase("2"))
                    {
                    ConcreteFactory2 factory = new ConcreteFactory2( );
                    Output output = new Output(factory,factory.GetDataStore( ));
                    MDAEFSM mdaefsm = new MDAEFSM(factory,output);
                    ATM2 atm = new ATM2(mdaefsm,factory.GetDataStore( ));
```

```
 System.out.println("ATM Machine-2" );

 int balance,deposit,withdraw,PIN,ID;

 while (true) {

System.out.println("  Enter Operation from the list below: ");

 System.out.println("***************************************");
 System.out.println("         ATM Machine-2 MENU of Operations" );
 System.out.println("         1. OPEN(int, int, int)" );
 System.out.println("         2. LOGIN(int)" );
 System.out.println("         3. PIN(int)" );
 System.out.println("         4. DEPOSIT(deposit)" );
 System.out.println("         5. WITHDRAW(withdraw)" );
 System.out.println("         6. BALANCE( )" );
 System.out.println("         7. suspend( )" );
 System.out.println("         8. activate( )" );
 System.out.println("         9. LOGOUT( )" );
 System.out.println("        10. close()" );
System.out.println("***************************************");


                        input = bufferedReader.readLine( );

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

    choice = Integer.parseInt(input);

    switch(choice)
    {
      case 1:   //open
                System.out.println("\n  Operation:  OPEN(int PIN, int ID, int balance)");
                System.out.println("   Enter value of the parameter balance:");
                input = bufferedReader.readLine( );
                balance = Integer.parseInt(input);

                System.out.println("   Enter value of the parameter PIN:");
                input = bufferedReader.readLine( );
                PIN = Integer.parseInt(input);

                System.out.println("   Enter value of the parameter ID:");
                input = bufferedReader.readLine( );
                ID = Integer.parseInt(input);


                atm.OPEN(PIN, ID, balance);

                break;

                  case 2:  //login
                                  System.out.println("  Operation:  LOGIN(int x)");
                                  System.out.println("  Enter value of LOGIN:");
```

```
                    input = bufferedReader.readLine( );
                    ID = Integer.parseInt(input);
                    atm.LOGIN(ID);
                    break;

                case 3:  //pin
                    System.out.println("  Operation:  PIN(int x)");
                    System.out.println("  Enter value of PIN:");
                    input = bufferedReader.readLine( );
                    PIN = Integer.parseInt(input);
                    atm.PIN(PIN);
                    break;

                case 4:  //deposit
                    System.out.println("  Operation:  DEPOSIT(int d)");
                    System.out.println("  Enter value of the parameter Deposit:");
                    input = bufferedReader.readLine( );
                    deposit = Integer.parseInt(input);
                    atm.DEPOSIT(deposit);
                    break;

                case 5:  // withdraw
                    System.out.println("  Operation:  WITHDRAW(int w)");
                    System.out.println("  Enter value of the parameter
Withdraw:");
                    input = bufferedReader.readLine( );
                    withdraw = Integer.parseInt(input);
                    atm.WITHDRAW(withdraw);
                    break;

                case 6:  // balance
                    System.out.println("  Operation:  BALANCE( )");
                    atm.BALANCE( );
                    break;

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

                case 8:  // activate
                    System.out.println("  Operation:  activate()");
                    atm.activate( );
                    break;

                case 9:  // logout
                    System.out.println("  Operation:  LOGOUT( )");
                    atm.LOGOUT( );
                    break;

                case 10:  // close
                    System.out.println("  Operation:  close( )");
                    atm.close( );
                    break;
```

```
                    default:
                                        System.out.println("Invalid Choice");
                                        break;
                }
            }
            System.out.println("Thanks for using ATM Machine - 2" );
                }

    }
}
```

5.2 AbstractFactory.java

```
package AbstractFactory;

import DataStore.*;
import Strategy.*;

/*
 * INTERFACE:: Abstract Factory PATTERN starts here
 */

public interface AbstractFactory
{
    public DataStore CreateDataStore( );
    public DisplayBalance CreateDisplayBalance( );
    public DisplayMenu CreateDisplayMenu( );
    public IncorrectIdMsg CreateIncorrectIdMsg( );
    public IncorrectLockMsg CreateIncorrectLockMsg( );
    public IncorrectPinMsg CreateIncorrectPinMsg( );
    public IncorrectUnlockMsg CreateIncorrectUnlockMsg( );
    public MakeDeposit CreateMakeDeposit( );
    public MakeWithdraw CreateMakeWithdraw( );
    public NoFundsMsg CreateNoFundsMsg( );
    public Penalty CreatePenalty( );
    public PromptForPin CreatePromptForPin( );
    public StoreData CreateStoreData( );
    public TooManyAttemptsMsg CreateTooManyAttemptsMsg( );
}
```

5.2.1 ConcreteFactory1.java

```
package AbstractFactory;

import DataStore.DataStore;
import DataStore.DataStore1;
import Strategy.*;


public class ConcreteFactory1 implements AbstractFactory
{
    DataStore ds = new DataStore1( );
```

```
DisplayBalance displayBalance = new DisplayBalance_ATM1( );
DisplayMenu displayMenu = new DisplayMenu_ATM1( );
IncorrectIdMsg incorrectIdMsg = new IncorrectIdMsg_ATM1( );
IncorrectLockMsg incorrectLockMsg = new IncorrectLockMsg_ATM1( );
IncorrectPinMsg incorrectPinMsg = new IncorrectPinMsg_ATM1( );
IncorrectUnlockMsg incorrectUnlockMsg = new IncorrectUnlockMsg_ATM1( );
MakeDeposit makeDeposit = new MakeDeposit_ATM1( );
MakeWithdraw makeWithdraw = new MakeWithdraw_ATM1( );
NoFundsMsg noFundsMsg = new NoFundsMsg_ATM1( );
Penalty penalty = new Penalty_ATM1( );
PromptForPin promptForPin = new PromptForPin_ATM1( );
StoreData storeData = new StoreData_ATM1( );
TooManyAttemptsMsg tooManyAttemptsMsg = new TooManyAttemptsMsg_ATM1( );



        public void ConcreteFactory( )
        {

        }

public DataStore CreateDataStore( )
{
   return(this.ds);
}

public DataStore GetDataStore( )
{
   return this.ds;
}

public DisplayBalance CreateDisplayBalance( )
{
   return this.displayBalance;
}

public DisplayMenu CreateDisplayMenu( )
{
   return this.displayMenu;
}

public IncorrectIdMsg CreateIncorrectIdMsg( )
{
   return this.incorrectIdMsg;
}

public IncorrectLockMsg CreateIncorrectLockMsg( )
{
   return this.incorrectLockMsg;
}

public IncorrectPinMsg CreateIncorrectPinMsg( )
{
   return this.incorrectPinMsg;
}
```

```
    public IncorrectUnlockMsg CreateIncorrectUnlockMsg( )
    {
        return this.incorrectUnlockMsg;
    }

    public MakeDeposit CreateMakeDeposit( )
    {
        return this.makeDeposit;
    }

    public MakeWithdraw CreateMakeWithdraw( )
    {
        return this.makeWithdraw;
    }

    public NoFundsMsg CreateNoFundsMsg( )
    {
        return this.noFundsMsg;
    }

    public Penalty CreatePenalty( )
    {
        return this.penalty;
    }

    public PromptForPin CreatePromptForPin( )
    {
        return this.promptForPin;
    }

    public StoreData CreateStoreData( )
    {
        return this.storeData;
    }

    public TooManyAttemptsMsg CreateTooManyAttemptsMsg( )
    {
        return this.tooManyAttemptsMsg;
    }
}
```

5.2.2 ConcreteFactory2. Java

```
package AbstractFactory;

import DataStore.DataStore;
import DataStore.DataStore1;
import DataStore.DataStore2;
import Strategy.*;

public class ConcreteFactory2 implements AbstractFactory
{
    DataStore ds = new DataStore2( );
```

```
DisplayBalance displayBalance = new DisplayBalance_ATM2( );
DisplayMenu displayMenu = new DisplayMenu_ATM2( );
IncorrectIdMsg incorrectIdMsg = new IncorrectIdMsg_ATM2( );
IncorrectLockMsg incorrectLockMsg = new IncorrectLockMsg_ATM2( );
IncorrectPinMsg incorrectPinMsg = new IncorrectPinMsg_ATM2( );
IncorrectUnlockMsg incorrectUnlockMsg = new IncorrectUnlockMsg_ATM2( );
MakeDeposit makeDeposit = new MakeDeposit_ATM2( );
MakeWithdraw makeWithdraw = new MakeWithdraw_ATM2( );
NoFundsMsg noFundsMsg = new NoFundsMsg_ATM2( );
Penalty penalty = new Penalty_ATM2( );
PromptForPin promptForPin = new PromptForPin_ATM2( );
StoreData storeData = new StoreData_ATM2( );
TooManyAttemptsMsg tooManyAttemptsMsg = new TooManyAttemptsMsg_ATM2( );



public void ConcreteFactory( )
{

}

public DataStore CreateDataStore( )
{
   return(this.ds);
}

public DataStore GetDataStore( )
{
   return this.ds;
}

public DisplayBalance CreateDisplayBalance( )
{
   return this.displayBalance;
}

public DisplayMenu CreateDisplayMenu( )
{
   return this.displayMenu;
}

public IncorrectIdMsg CreateIncorrectIdMsg( )
{
   return this.incorrectIdMsg;
}

public IncorrectLockMsg CreateIncorrectLockMsg( )
{
   return this.incorrectLockMsg;
}

public IncorrectPinMsg CreateIncorrectPinMsg( )
{
   return this.incorrectPinMsg;
}
```

```
    public IncorrectUnlockMsg CreateIncorrectUnlockMsg( )
    {
        return this.incorrectUnlockMsg;
    }

    public MakeDeposit CreateMakeDeposit( )
    {
        return this.makeDeposit;
    }

    public MakeWithdraw CreateMakeWithdraw( )
    {
        return this.makeWithdraw;
    }

    public NoFundsMsg CreateNoFundsMsg( )
    {
        return this.noFundsMsg;
    }

    public Penalty CreatePenalty( )
    {
        return this.penalty;
    }

    public PromptForPin CreatePromptForPin( )
    {
        return this.promptForPin;
    }

    public StoreData CreateStoreData( )
    {
        return this.storeData;
    }

    public TooManyAttemptsMsg CreateTooManyAttemptsMsg( )
    {
        return this.tooManyAttemptsMsg;
    }

}
```

5.3 DataStore.java

```
package DataStore;
//
//  DataStore2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
```

```
public abstract class DataStore{

}
```

5.3.1 DataStore1.java

```
package DataStore;
//
//  DataStore1.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class DataStore1 extends DataStore
{

   /* Temporary Storage Variables */
   public float tempBalance;
   public String tempPin;
   public String tempID;
   public float tempDeposit;
   public float tempWithdraw;


   /* Permanent Storage Variables */
   public float balance;
   public String pin;
   public String id;
   public float deposit;
   public float withdraw;

   public float getBalance( )
   {
      return this.balance;
   }

   public String getPin( )
   {
      return this.pin;
   }

   public String getID( )
   {
      return this.id;
   }

   public float setBalance( )
   {
      return this.balance = this.tempBalance;
   }
```

```
    public String setPin( )
    {
        return this.pin = this.tempPin;
    }

    public String setID( )
    {
        return this.id = this.tempID;
    }

    public float setPenalty( )
    {
        this.balance = this.balance - 20;
        this.tempBalance = balance;
        return this.balance;
    }

    public void computeBalanceDeposit( )
    {
        this.balance = this.balance + this.deposit;
        this.tempBalance = this.balance;
    }
    public void computeBalanceWithdraw( )
    {
        this.balance = this.balance - this.withdraw;
        this.tempBalance = this.balance;
    }

    public void setDeposit( )
    {
        this.deposit = this.tempDeposit;
    }
    public void setWithdraw( )
    {
        this.withdraw = this.tempWithdraw;
    }
    public float getDeposit( )
    {
        return this.deposit;
    }
    public float getWithdraw( )
    {
        return this.withdraw;
    }
}
```

5.3.2 DataStore2.java

```
package DataStore;
//
//  DataStore2.java
//
```

```
//
//  Created by Aditya on 04/24/16.
//
//

public class DataStore2 extends DataStore
{

   /* Temporary Storage Variables */
   public int tempBalance;
   public int tempPin;
   public int tempID;
   public int tempDeposit;
   public int tempWithdraw;


   /* Permanent Storage Variables */
   public int balance;
   public int pin;
   public int id;
   public int deposit;
   public int withdraw;

   public int getBalance( )
   {
      return this.balance;
   }

   public int getPin( )
   {
      return this.pin;
   }

   public int getID( )
   {
      return this.id;
```

```
}

public int setBalance( )
{
   return this.balance = this.tempBalance;
}

public int setPin( )
{
   return this.pin = this.tempPin;
}

public int setID( )
{
   return this.id = this.tempID;
}

public int setPenalty( )
{
   this.balance = this.balance - 0;
   this.tempBalance = balance;
   return this.balance;
}

public void computeBalanceDeposit( )
{
   this.balance = this.balance + this.deposit;
   this.tempBalance = this.balance;
}
public void computeBalanceWithdraw( )
{
   this.balance = this.balance - this.withdraw;
   this.tempBalance = this.balance;
}

public void setDeposit( )
```

```
    {
        this.deposit = this.tempDeposit;
    }
    public void setWithdraw( )
    {
        this.withdraw = this.tempWithdraw;
    }
    public int getDeposit( )
    {
        return this.deposit;
    }
    public int getWithdraw( )
    {
        return this.withdraw;
    }
}
```

5.4.1 ATM1.java

```
package ATM_Machine;

import MDA_EFSM.MDAEFSM;
import DataStore.DataStore;
import DataStore.DataStore1;

/*
 * CLASS : ATM1 Machine Implementation for collecting parameters
 * from the UI through Driver.java and invoking right event in MDA-EFSM
 */

public class ATM1
{
        /* Pointer to MDA-EFSM */
        MDAEFSM mdaefsm = null;
        /* Pointer to DataStore */
        DataStore dataStore = null;

        public ATM1(MDAEFSM mdaefsm,DataStore dataStore)
        {
    this.mdaefsm = mdaefsm;
    this.dataStore = dataStore;
    }

        public void open(String p, String y, float a)
        {
```

```
                    if(a > 0)
                    {
                ((DataStore1)dataStore).tempPin = p;
                ((DataStore1)dataStore).tempBalance = a;
                ((DataStore1)dataStore).tempID = y;
                mdaefsm.Open( );
                    }
                    else
                    {
                            System.out.println("Amount can't be in negative");
                    }
}

public void pin(String x)
{
        if(x.equals(((DataStore1)dataStore).tempPin))
        {
                if(((DataStore1)dataStore).tempBalance > 500.00)
                        mdaefsm.CorrectPinAboveMin( );
                else
         mdaefsm.CorrectPinBelowMin( );
        }
        else
        {
                mdaefsm.IncorrectPin(3);
        }
}

public void deposit(float d)
{
        if(d > 0)
        {
                ((DataStore1)dataStore).tempDeposit = d;
        mdaefsm.Deposit( );

        if(((DataStore1)dataStore).tempBalance > 500.00)
        {
                mdaefsm.AboveMinBalance( );
        }
        else
        {
                mdaefsm.BelowMinBalance( );
        }
        }

        else
        {
                System.out.println("Amount can't be in negative");
        }
}

public void withdraw(float w)
{
        if(w>0)
        {
```

```
                ((DataStore1)dataStore).tempWithdraw = w;
        mdaefsm.Withdraw( );

        if(((DataStore1)dataStore).tempBalance > 500.00)
        {
                mdaefsm.AboveMinBalance( );
        }
        else
        {
                mdaefsm.WithdrawBelowMinBalance( );
        }
        }

        else
        {
                System.out.println("Amount can't be in negative");
        }



}

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

public void login(String y){
   if(y.equals(((DataStore1)dataStore).tempID))
      mdaefsm.Login( );
   else
      mdaefsm.IncorrectLogin( );
}

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

public void lock(String x)
{
        if(x.equals(((DataStore1)dataStore).tempPin))
      mdaefsm.Lock( );
        else
      mdaefsm.IncorrectLock( );
}

public void unlock(String x)
{
        if(x.equals(((DataStore1)dataStore).tempPin))
        {
                mdaefsm.Unlock( );

                if(((DataStore1)dataStore).balance > 500.00)
                {
```

```
                        mdaefsm.AboveMinBalance( );
                }
                else
                {
                        mdaefsm.BelowMinBalance( );
                }
        }
        else
                mdaefsm.IncorrectUnlock( );

   }
}
```

5.4.2 ATM2.java

```
package ATM_Machine;

import MDA_EFSM.MDAEFSM;
import DataStore.DataStore;
import DataStore.DataStore1;
import DataStore.DataStore2;

/*
 * CLASS : ATM2 Machine Implementation for collecting parameters
 * from the UI through Driver.java and invoking right event in MDA-EFSm
 */

public class ATM2
{

        /* Pointer to MDA-EFSM */
        MDAEFSM mdaefsm = null;
        /* Pointer to DataStore */
        DataStore dataStore = null;

        public ATM2(MDAEFSM mdaefsm,DataStore dataStore)
        {
      this.mdaefsm = mdaefsm;
      this.dataStore = dataStore;
   }

   public void OPEN(int p, int y, int a)
   {
        if(a > 0)
                {
            ((DataStore2)dataStore).tempPin = p;
            ((DataStore2)dataStore).tempBalance = a;
            ((DataStore2)dataStore).tempID = y;
            mdaefsm.Open( );
                }
                else
                {
                        System.out.println("Amount can't be in negative");
                }
```

```
}

public void PIN(int x)
{
        if(x == ((DataStore2)dataStore).tempPin)
        {
      mdaefsm.CorrectPinAboveMin( );
        }
        else
        {
                mdaefsm.IncorrectPin(2);
        }
}

public void DEPOSIT(int d)
{
        if(d > 0)
        {
        ((DataStore2)dataStore).tempDeposit = d;
        mdaefsm.Deposit( );
        }
        else
                {
                        System.out.println("Wrong Input");
                }
}

public void WITHDRAW(int w)
{
        if(w > 0)
        {
                ((DataStore2)dataStore).tempWithdraw = w;
        if( ((DataStore2)dataStore).tempBalance > 0)
        {
         mdaefsm.Withdraw( );
        }
        else
        {
                mdaefsm.NoFunds( );
        }
        }
        else
                {
                        System.out.println("Amount can't be in negative");
                }

}

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

public void LOGIN(int y){
   if(y == (((DataStore2)dataStore).tempID))
```

```
            mdaefsm.Login( );
        else
            mdaefsm.IncorrectLogin( );
    }

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

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

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

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

5.5 MDA-EFSM (STATE PATTERN)

5.5.1 MDAEFSM.java

```
package MDA_EFSM;

import AbstractFactory.AbstractFactory;
import Output.Output;

/*
 *  CLASS : MDAEFSM ( STATE PATTERN )
 */

public class MDAEFSM {

    State checkpinState = new CheckPinState(this);
    State idleState = new IdleState(this);
    State lockedState = new LockedState(this);
    State overdrawnState = new OverdrawnState(this);
```

```
State readyState = new ReadyState(this);
State s1State = new S1State(this);
State startState = new StartState(this);
State suspendedState = new SuspendedState(this);

State efsmState = null;

//State List[8] ;
        public int attempts;

        AbstractFactory factory =null;
        Output output = null;

public MDAEFSM(AbstractFactory factory,Output output) {
   efsmState = startState;
   attempts = 0;
   this.factory = factory;
   this.output = output;
}

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

public void Login( )
{
   efsmState.Login( );
   attempts = 0;
   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(State efsmState)
{
    this.efsmState = efsmState;
}
public State getMachineState( ) {
    return efsmState;
}

public State getCheckPinState( ) {
    return checkpinState;
}

public State getSuspendedState( ) {
    return suspendedState;
}

public State getIdleState( ) {
    return idleState;
}

public State getStartState( ) {
    return startState;
}

public State getLockedState( ) {
    return lockedState;
}

public State getOverdrawnState( ) {
    return overdrawnState;
}

public State getReadyState( ) {
```

```
        return readyState;
    }

    public State getS1State( ) {
        return s1State;
    }

    public void printCurrentState( ){
            System.out.println("*** Current State : "+ efsmState.getClass( ).getName( )+"***");
    }

}
```

5.5.2 State.java

```
package MDA_EFSM;
//
//  State.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public interface State
{
    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( );
}
```

5.5.3 StartState.java

```
package MDA_EFSM;
//
// StartState.java
//
//
/  Created by Aditya on 04/24/16.
//
//

public class StartState implements State
{
   MDAEFSM mdaefsm =null;

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

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

   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( )
    {

    }}
```

5.5.4 IdleState.java

```
package MDA_EFSM;
//
//  IdleState.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class IdleState implements State
{
    MDAEFSM mdaefsm =null;

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

    public void Open( )
    {

    }

    public void Login( )
    {
        mdaefsm.attempts = 0;
        mdaefsm.output.PromptForPin( );
        mdaefsm.setState(mdaefsm.getCheckPinState( ));

    }
```

```
public void IncorrectLogin( )
{
   mdaefsm.output.IncorrectIdMsg( );
}

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( )
    {

    }
}
```

5.5.5 CheckPinState.java

```
package MDA_EFSM;
//
//  CheckPinState.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class CheckPinState implements State
{
    MDAEFSM mdaefsm =null;
```

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

public void Open( )
{

}

public void Login( )
{

}

public void IncorrectLogin( )
{

}

public void IncorrectPin(int max)
{
        if( mdaefsm.attempts < max )
        {
                mdaefsm.attempts++;
                mdaefsm.output.IncorrectPinMsg();
        }
        else if( mdaefsm.attempts == max )
        {
                mdaefsm.output.IncorrectPinMsg( );
                mdaefsm.output.TooManyAttemptsMsg( );
                mdaefsm.setState(mdaefsm.getIdleState( ));
        }

}

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

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

public void Deposit( )
{

}

public void BelowMinBalance( )
```

```
{

}

public void AboveMinBalance( )
{

}

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

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( )
    {

    }
}
```

5.5.6 ReadyState.java

```
package MDA_EFSM;

//
//  ReadyState.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class ReadyState implements State
{
    MDAEFSM mdaefsm =null;

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

    public void Open( )
    {

    }

    public void Login( )
    {

    }

    public void IncorrectLogin( )
    {

    }

    public void IncorrectPin(int max)
    {
```

```
}

public void CorrectPinBelowMin( )
{

}

public void CorrectPinAboveMin( )
{

}

public void Deposit( )
{
   mdaefsm.output.MakeDeposit( );
}

public void BelowMinBalance( )
{

}

public void AboveMinBalance( )
{

}

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

public void Balance( )
{
   mdaefsm.output.DisplayBalance( );
}

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

public void WithdrawBelowMinBalance( )
{

}

public void NoFunds( )
{
   mdaefsm.output.NoFundsMsg( );
}

public void Lock( )
{
```

```
        mdaefsm.setState(mdaefsm.getLockedState( ));
    }

    public void IncorrectLock( )
    {
        mdaefsm.output.IncorrectLockMsg( );
    }

    public void Unlock( )
    {

    }

    public void IncorrectUnlock( )
    {

    }

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

    public void Activate( )
    {

    }

    public void Close( )
    {

    }
}
```

5.5.7 OverdrawnState.java

```
package MDA_EFSM;
//
//  OverdrawnState.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class OverdrawnState implements State
{
    MDAEFSM mdaefsm =null;

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

```
public void Open( )
{

}

public void Login( )
{

}

public void IncorrectLogin( )
{

}

public void IncorrectPin(int max)
{

}

public void CorrectPinBelowMin( )
{

}

public void CorrectPinAboveMin( )
{

}

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

public void BelowMinBalance( )
{

}

public void AboveMinBalance( )
{

}

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

public void Balance( )
{
   mdaefsm.output.DisplayBalance( );
}
```

```
    public void Withdraw( )
    {
       mdaefsm.output.NoFundsMsg( );
    }

    public void WithdrawBelowMinBalance( )
    {

    }

    public void NoFunds( )
    {

    }

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

    public void IncorrectLock( )
    {

    }

    public void Unlock( )
    {

    }

    public void IncorrectUnlock( )
    {

    }

    public void Suspend( )
    {

    }

    public void Activate( )
    {

    }

    public void Close( )
    {

    }
}
```

5.5.8 LockedState.java

```
package MDA_EFSM;
//
//  LockedState.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class LockedState implements State
{
   MDAEFSM mdaefsm =null;

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

   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( )
{
   mdaefsm.setState(mdaefsm.getS1State( ));
}

public void IncorrectUnlock( )
{
   mdaefsm.output.IncorrectUnlockMsg( );
}

public void Suspend( )
```

```
    {

    }

    public void Activate( )
    {

    }

    public void Close( )
    {

    }
}
```

5.5.9 S1State.java

```
package MDA_EFSM;

//
//  S1State.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class S1State implements State
{
    MDAEFSM mdaefsm =null;

    public S1State(MDAEFSM mdaefsm)
    {
        this.mdaefsm =  mdaefsm;
    }

    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( )
{
   mdaefsm.setState(mdaefsm.getOverdrawnState( ));
}

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

public void Logout( )
{

}

public void Balance( )
{

}

public void Withdraw( )
{

}

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

public void NoFunds( )
{

}
```

```
    public void Lock( )
    {

    }

    public void IncorrectLock( )
    {

    }

    public void Unlock( )
    {

    }

    public void IncorrectUnlock( )
    {

    }

    public void Suspend( )
    {

    }

    public void Activate( )
    {

    }

    public void Close( )
    {

    }
}
```

5.5.10 SuspendedState.java

```
package MDA_EFSM;
//
// SuspendedState.java
//
//
//  Created by Aditya on 04/24/16.
//
//

public class SuspendedState implements State
{
    MDAEFSM mdaefsm =null;

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

```
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( )
{
   mdaefsm.output.DisplayBalance( );
```

```
    }

    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( )
    {
       mdaefsm.setState(mdaefsm.getReadyState());
    }

    public void Close( )
    {
            mdaefsm.setState(mdaefsm.getStartState());
    }
}
```

5.6 Output.java

```
package Output;

import DataStore.DataStore;
import AbstractFactory.*;
import Strategy.DisplayBalance;
import Strategy.DisplayMenu;
import Strategy.IncorrectIdMsg;
import Strategy.IncorrectLockMsg;
import Strategy.IncorrectPinMsg;
import Strategy.IncorrectUnlockMsg;
import Strategy.MakeDeposit;
import Strategy.MakeWithdraw;
import Strategy.NoFundsMsg;
import Strategy.Penalty;
import Strategy.PromptForPin;
import Strategy.StoreData;
import Strategy.TooManyAttemptsMsg;

/*
 *  CLASS : Output
 */
public class Output
{
        AbstractFactory factory =null;
        DataStore dataStore = null;

        public Output(AbstractFactory factory,DataStore dataStore)
        {
      this.factory = factory;
      this.dataStore = dataStore;
   }

   public void IncorrectIdMsg( )
   {
      System.out.println("\n OUTPUT:: Action IncorrectIdMsg");
      IncorrectIdMsg incorrectID = factory.CreateIncorrectIdMsg( );
      incorrectID.IncorrectIdMsg( );
   }


   public void IncorrectPinMsg( )
        {
                System.out.println("\n OUTPUT:: Action IncorrectPinMsg");
                IncorrectPinMsg incorrectPin = factory.CreateIncorrectPinMsg( );
                incorrectPin.IncorrectPinMsg( );
        }

   public void IncorrectLockMsg( )
   {
      System.out.println("\n OUTPUT:: Action IncorrectLockMsg");
      IncorrectLockMsg incorrectLock = factory.CreateIncorrectLockMsg();
```

```
    incorrectLock.IncorrectLockMsg();
}

public void IncorrectUnlockMsg( )
{
    System.out.println("\n OUTPUT:: Action IncorrectUnlockMsg");
    IncorrectUnlockMsg incorrectUnlock = factory.CreateIncorrectUnlockMsg( );
    incorrectUnlock.IncorrectUnlockMsg( );
}

        public void TooManyAttemptsMsg( )
        {
                System.out.println("\n OUTPUT:: Action TooManyAttemptsMsg");
                TooManyAttemptsMsg tooManyAttempts = factory.CreateTooManyAttemptsMsg( );
                tooManyAttempts.TooManyAttemptsMsg( );
        }

        public void DisplayMenu( )
        {
                System.out.println("\n OUTPUT:: Action DisplayMenu");
                DisplayMenu displayMenu = factory.CreateDisplayMenu( );
                displayMenu.DisplayMenu( );
        }

        public void StoreData( )
        {
                System.out.println("\n OUTPUT:: Action StoreData");
                StoreData storeData = factory.CreateStoreData( );
                storeData.StoreData(dataStore);
        }

        public void NoFundsMsg( )
        {
                System.out.println("\n OUTPUT:: Action NoFundsMsg");
                NoFundsMsg noFunds = factory.CreateNoFundsMsg( );
                noFunds.NoFundsMsg( );
        }

        public void PromptForPin( )
        {
                System.out.println("\n OUTPUT:: Action PromptForPin ");
                PromptForPin promptForPin = factory.CreatePromptForPin( );
                promptForPin.PromptForPin( );
        }

        public void DisplayBalance( )
        {
                System.out.println("\n OUTPUT:: Action DisplayBalance ");
                DisplayBalance displayBalance = factory.CreateDisplayBalance( );
                displayBalance.DisplayBalance(dataStore);
        }

        public void MakeDeposit( )
        {
                System.out.println("\n OUTPUT:: Action MakeDeposit ");
```

```
                MakeDeposit makeDeposit = factory.CreateMakeDeposit( );
                makeDeposit.MakeDeposit(dataStore);
        }

        public void MakeWithdraw( )
        {
                System.out.println("\n OUTPUT:: Action MakeWithdraw  ");
                MakeWithdraw makeWithdraw = factory.CreateMakeWithdraw( );
                makeWithdraw.MakeWithdraw(dataStore);
        }

        public void Penalty( )
        {
                System.out.println("\n OUTPUT:: Action Penalty  ");
                Penalty penalty = factory.CreatePenalty( );
                penalty.Penalty(dataStore);
        }

}
```

5.7 STRATEGY PATTERN

5.7.1 DisplayBalance.java

```
package Strategy;
//
//  DisplayBalance.java
//
//
//  Created by Aditya on 04/24/16.
//
//

import DataStore.*;

/*
 *  ABSTRACT CLASS : DisplayBalance (STRAGTEGY PATTERN)
 */

public abstract class DisplayBalance
{
        public abstract void DisplayBalance(DataStore dataStore);
}
```

5.7.2 DisplayBalance_ATM1.java

```
package Strategy;
//
//  DisplayBalance_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
```

```
import DataStore.*;

public class DisplayBalance_ATM1 extends DisplayBalance
{
        public void DisplayBalance(DataStore dataStore)
        {
                System.out.println("ATM Machine 1:: Account Balance is: " +
((DataStore1)dataStore).getBalance( ) );
        }
}
```

5.7.3 DisplayBalance_ATM2.java

```
package Strategy;
//
//  DisplayBalance_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//

import DataStore.*;

public class DisplayBalance_ATM2 extends DisplayBalance
{
        public void DisplayBalance(DataStore dataStore)
        {
                System.out.println("ATM Machine 2:: Balance is: " +
((DataStore2)dataStore).getBalance( ) );
        }
}
```

5.7.4 DisplayMenu.java

```
package Strategy;
//
//  DisplayMenu.java
//
//
//  Created by Aditya on 04/24/16.
//
//

import DataStore.*;

/*
 *  ABSTRACT CLASS : DisplayMenu (STRAGTEGY PATTERN)
 */

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

```
}
```

5.7.5 DisplayMenu_ATM1.java

```
package Strategy;
//
//  DisplayMenu_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//

import DataStore.*;

public class DisplayMenu_ATM1 extends DisplayMenu
{
        public void DisplayMenu( )
        {
                System.out.println("ATM Machine 1:: Transaction Menu: ");
                System.out.println("   1:: Balance " );
                System.out.println("   2:: Deposit " );
                System.out.println("   3:: Withdraw " );
        }
}
```

5.7.6 DisplayMenu_ATM2.java

```
package Strategy;
//
//  DisplayMenu_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//

import DataStore.*;

public class DisplayMenu_ATM2 extends DisplayMenu
{
        public void DisplayMenu( )
        {
                System.out.println("ATM Machine 2:: Transaction Menu: ");
                System.out.println("   1:: Balance " );
                System.out.println("   2:: Deposit " );
                System.out.println("   3:: Withdraw " );
        }
}
```

5.7.7 IncorrectIdMsg.java

```
package Strategy;
//
//  IncorrectIdMsg.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : IncorrectIdMsg (STRAGTEGY PATTERN)
 */

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

5.7.8 IncorrectIdMsg_ATM1.java

```
package Strategy;
//
//  IncorrectIdMsg_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectIdMsg_ATM1 extends IncorrectIdMsg
{
        public void IncorrectIdMsg( )
        {
                System.out.println("ATM Machine 1:: Incorrect Id Entered ");
        }
}
```

5.7.9 IncorrectIdMsg_ATM2.java

```
package Strategy;
//
//  IncorrectIdMsg_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectIdMsg_ATM2 extends IncorrectIdMsg
{
```

```
        public void IncorrectIdMsg( )
        {
                System.out.println("ATM Machine 2:: Incorrect Id: ");
        }
}
```

5.7.10 IncorrectLockMsg.java

```
package Strategy;
//
//  IncorrectLockMsg.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : IncorrectLockMsg (STRAGTEGY PATTERN)
 */

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

5.7.11 IncorrectLockMsg_ATM1.java

```
package Strategy;
//
//  IncorrectLockMsg_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectLockMsg_ATM1 extends IncorrectLockMsg
{
        public void IncorrectLockMsg( )
        {
                System.out.println("ATM Machine 1:: Incorrect Lock Attempted");
        }
}
```

5.7.12 IncorrectLockMsg_ATM2.java

```
package Strategy;
```

```
//
//  IncorrectLockMsg_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectLockMsg_ATM2 extends IncorrectLockMsg
{
        public void IncorrectLockMsg( )
        {
                System.out.println("ATM Machine 2:: Incorrect Lock ");
        }
}
```

5.7.13 IncorrectPinMsg.java

```
package Strategy;
//
//  IncorrectPinMsg.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : IncorrectPinMsg (STRAGTEGY PATTERN)
 */

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

5.7.14 IncorrectPinMsg_ATM1.java

```
package Strategy;
//
//  IncorrectPinMsg_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectPinMsg_ATM1 extends IncorrectPinMsg
{
        public void IncorrectPinMsg( )
        {
```

```
                System.out.println("ATM Machine 1:: Incorrect Pin Entered ");
        }
}
```

5.7.15 IncorrectPinMsg_ATM2.java

```
package Strategy;
//
//  IncorrectPinMsg_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectPinMsg_ATM2 extends IncorrectPinMsg
{
        public void IncorrectPinMsg( )
        {
                System.out.println("ATM Machine 2:: Incorrect Pin: ");
        }
}
```

5.7.16 IncorrectUnlockMsg.java

```
package Strategy;
//
//  IncorrectUnlockMsg.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : IncorrectUnlockMsg (STRAGTEGY PATTERN)
 */

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

5.7.17 IncorrectUnlockMsg_ATM1.java

```
package Strategy;
//
//  IncorrectUnlockMsg_ATM1.java
//
//
```

```
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectUnlockMsg_ATM1 extends IncorrectUnlockMsg
{
        public void IncorrectUnlockMsg( )
        {
                System.out.println("ATM Machine 1:: Incorrect Unlock Attempted");
        }
}
```

5.7.18 IncorrectUnlockMsg_ATM2.java

```
package Strategy;
//
//  IncorrectUnlockMsg_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class IncorrectUnlockMsg_ATM2 extends IncorrectUnlockMsg
{
        public void IncorrectUnlockMsg( )
        {
                System.out.println("ATM Machine 2:: Incorrect Unlock ");
        }
}
```

5.7.19 MakeDeposit.java

```
package Strategy;
//
//  MakeDeposit.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : MakeDeposit (STRAGTEGY PATTERN)
 */

public abstract class MakeDeposit
{
        public abstract void MakeDeposit(DataStore dataStore);
}
```

5.7.20 MakeDeposit_ATM1.java

```
package Strategy;
//
//  MakeDeposit_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class MakeDeposit_ATM1 extends MakeDeposit
{
        public void MakeDeposit(DataStore dataStore)
        {
                ((DataStore1)dataStore).setDeposit( );
                ((DataStore1)dataStore).computeBalanceDeposit( );
                System.out.println("ATM Machine 1:: After Deposit, Balance is " +
((DataStore1)dataStore).getBalance( ) );
        }
}
```

5.7.21 MakeDeposit_ATM2.java

```
package Strategy;
//
//  MakeDeposit_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class MakeDeposit_ATM2 extends MakeDeposit
{
        public void MakeDeposit(DataStore dataStore)
        {
                ((DataStore2)dataStore).setDeposit( );
                ((DataStore2)dataStore).computeBalanceDeposit( );
                System.out.println("ATM Machine 2:: After Deposit, Balance is " +
((DataStore2)dataStore).getBalance( ) );
        }
}
```

5.7.22 MakeWithdraw.java

```
package Strategy;
//
//  MakeWithdraw.java
//
```

```
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : MakeWithdraw (STRAGTEGY PATTERN)
 */

public abstract class MakeWithdraw
{
        public abstract void MakeWithdraw(DataStore dataStore);
}
```

5.7.23 MakeWithdraw_ATM1.java

```
package Strategy;
//
//  MakeWithdraw_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class MakeWithdraw_ATM1 extends MakeWithdraw
{
        public void MakeWithdraw(DataStore dataStore)
        {
                ((DataStore1)dataStore).setWithdraw( );
                ((DataStore1)dataStore).computeBalanceWithdraw( );
                System.out.println("ATM Machine 1:: After Withdrawal, Balance is " +
((DataStore1)dataStore).getBalance( ) );
        }
}
```

5.7.24 MakeWithdraw_ATM2.java

```
package Strategy;
//
//  MakeDeposit_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;
```

```
public class MakeWithdraw_ATM2 extends MakeWithdraw
{
	public void MakeWithdraw(DataStore dataStore)
	{
		((DataStore2)dataStore).setWithdraw( );
		((DataStore2)dataStore).computeBalanceWithdraw( );
		System.out.println("ATM Machine 2:: After Withdraw, Balance is " +
((DataStore2)dataStore).getBalance( ) );
	}
}
```

5.7.25 NoFundsMsg.java

```
package Strategy;
//
//  NoFundsMsg.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : NoFundsMsg (STRAGTEGY PATTERN)
 */

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

5.7.26 NoFundsMsg_ATM1.java

```
package Strategy;
//
//  NoFundsMsg_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class NoFundsMsg_ATM1 extends NoFundsMsg
{
	public void NoFundsMsg( )
	{
		System.out.println("ATM Machine 1:: No Funds Available");
	}
}
```

5.7.27 NoFundsMsg_ATM2.java

```
package Strategy;
//
//  NoFundsMsg_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class NoFundsMsg_ATM2 extends NoFundsMsg
{
        public void NoFundsMsg( )
        {
                System.out.println("ATM Machine 2:: No Funds ");
        }
}
```

5.7.28 Penalty.java

```
package Strategy;
//
//  Penalty.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : Penalty (STRAGTEGY PATTERN)
 */

public abstract class Penalty
{
        public abstract void Penalty(DataStore dataStore);
}
```

5.7.29 Penalty_ATM1.java

```
package Strategy;
//
//  Penalty_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;
```

```
public class Penalty_ATM1 extends Penalty
{
	public void Penalty(DataStore dataStore)
	{
		((DataStore1)dataStore).setPenalty( );
		System.out.println("ATM Machine 1:: Minimum required balance is $500. So Penalty is
applied.");
		System.out.println("ATM Machine 1:: After a Penalty of 20$, Balance is " +
((DataStore1)dataStore).balance );
	}
}
```

5.7.30 Penalty_ATM2.java

```
package Strategy;
//
//  Penalty_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class Penalty_ATM2 extends Penalty
{
	public void Penalty(DataStore dataStore)
	{

	}
}
```

5.7.31 PromptForPin.java

```
package Strategy;
//
//  PromptForPin.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : PromptForPin (STRAGTEGY PATTERN)
 */

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

5.7.32 PromptForPin_ATM1.java

```
package Strategy;
//
//  PromptForPin_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class PromptForPin_ATM1 extends PromptForPin
{
        public void PromptForPin( )
        {
                System.out.println("ATM Machine 1:: Enter the Pin: " );
        }
}
```

5.7.33 PromptForPin_ATM2.java

```
package Strategy;
//
//  PromptForPin_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class PromptForPin_ATM2 extends PromptForPin
{
        public void PromptForPin( )
        {
                System.out.println("ATM Machine 2:: Enter the Pin:: " );
        }
}
```

5.7.34 StoreData.java

```
package Strategy;
//
//  StoreData.java
//
//
//  Created by Aditya on 04/24/16.
//
//
```

```
import DataStore.*;

/*
 *  ABSTRACT CLASS : StoreData (STRAGTEGY PATTERN)
 */

public abstract class StoreData
{
        public abstract void StoreData(DataStore datastore);
}
```

5.7.35 StoreData_ATM1.java

```
package Strategy;
//
//  StoreData_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class StoreData_ATM1 extends StoreData
{
        public void StoreData(DataStore dataStore)
        {
                ((DataStore1)dataStore).setID( );
                ((DataStore1)dataStore).setPin( );
    ((DataStore1)dataStore).setBalance( );
                System.out.println("ATM Machine 1:: After storing the ID, pin and balance, ID is" +
((DataStore1)dataStore).id + " PIN is " + ((DataStore1)dataStore).pin + " and Balance is " +
((DataStore1)dataStore).balance);
        }
}
```

5.7.36 StoreData_ATM2.java

```
package Strategy;
//
//  StoreData_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class StoreData_ATM2 extends StoreData
{
        public void StoreData(DataStore dataStore)
        {
    ((DataStore2)dataStore).setPin( );
```

```
        ((DataStore2)dataStore).setBalance( );
        System.out.println("ATM Machine 2:: After storing the pin and balance, PIN is " +
((DataStore2)dataStore).pin + "and Balance is " + ((DataStore2)dataStore).balance);
            }
}
```

5.7.37 TooManyAttemptsMsg.java

```
package Strategy;
//
//  TooManyAttemptsMsg.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

/*
 *  ABSTRACT CLASS : TooManyAttemptsMsg (STRAGTEGY PATTERN)
 */

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

5.7.38 TooManyAttemptsMsg_ATM1.java

```
package Strategy;
//
//  TooManyAttemptsMsg_ATM1.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class TooManyAttemptsMsg_ATM1 extends TooManyAttemptsMsg
{
            public void TooManyAttemptsMsg( )
            {
                        System.out.println("ATM Machine 1:: Too Many Attempts Made ");
            }
}
```

5.7.39 TooManyAttemptsMsg_ATM2.java

```
package Strategy;
//
```

```
//  TooManyAttemptsMsg_ATM2.java
//
//
//  Created by Aditya on 04/24/16.
//
//
import DataStore.*;

public class TooManyAttemptsMsg_ATM2 extends TooManyAttemptsMsg
{
        public void TooManyAttemptsMsg( )
        {
                System.out.println("ATM Machine 2:: Too Many Attempts: ");
        }
}
```