CS 586 - Project

Design and Implementation of ATMs using MDA

The goal of this project is to design two different ATM components using a Model-Driven Architecture (MDA) and then implements these ATM components.

Instructor: Dr. Bogdan Korel

Darshankumar Zala
Illinois Institute of Technology

Table of Contents

1. Introduction

The goal of this project is to design an ATM Based system involving two different ATM components using a Model-Driven Architecture (MDA) and then implement these ATM components based on this design.

1.1. General Description of the project

There are two ATM components: ATM-1 and ATM-2

The **ATM-1** component supports the following Output Processor rations:

```
card (int x, int y)     // ATM card is inserted where x is a balance and y is a pin #
pin (int x)             // provides pin #
deposit (int d);        // deposit amount d
withdraw (int w);       // withdraw amount w
balance ();             // display the current balance
lock(int x);            // lock the ATM, where x is a pin #
unlock(int x);          // unlock the ATM, where x is pin #
exit();                 // exit from the ATM
```

The **ATM-2 component** supports the following Output Processor rations:

```
CARD (float x, string y);   // ATM card is inserted where x is a balance and y is a pin #
PIN (string x);             // provides pin #
DEPOSIT (float d);          // deposit amount d
WITHDRAW (float w);         // withdraw amount w
BALANCE ();                 // display the current balance
EXIT();                     // exit from the ATM
```

Both ATM components are state-based components and support three types of transactions: withdrawal, deposit, and balance inquiry. Before any transaction can be performed, Output Processoreration *card*(*x*, *y*) (or *CARD*(*x*, *y*)) must be issued, where *x* is an initial balance in the account and *y* is a pin used to get permission to perform transactions. Before any transaction can be performed, Output Processoreration *pin(x)* (or *PIN(x)*) must be issued. The *pin(x)* (or *PIN(x)*) Output Processoreration must contain the valid pin # that must be the same as the pin # provided in *card*(*x*, *y*) (or *CARD*(*x*, *y*)) Output Processoreration.

There is a limit on the number of attempts with an invalid pin. The account can be overdrawn (below minimum balance), but a penalty applies. If the balance is below the minimum balance then the withdrawal transaction cannot be performed. In addition, ATM-1 component can be locked by issuing *lock(x)* Output Processoreration, where *x* is a pin #. The ATM-1 can be unlocked by *unlock(x)* Output Processoreration. The detailed behavior of both ATM components is specified using EFSM. Notice that there are several differences between both ATMs. Aspects that vary between two ATM components:

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

The goal of this project is to design two ATM components using a Model-Driven Architecture (MDA) covered in the course. An executable meta-model, referred to as MDA-EFSM, of ATM components should capture the "generic behavior" of both ATM components and should be decoupled from data and implementation details. In the design there should be **ONLY** one MDA-EFSM for both ATM components. In addition, in the Model-Driven Architecture coupling between components should be minimized and cohesion of components should be maximized (components with high cohesion and low coupling between components).

The metamodel (MDA-EFSM) used in the Model-Driven architecture should be expressed as an EFSM (Extended Finite State Machine) model. Notice that the EFSMs shown in Figures 1 and Figure 2 are **not acceptable** as a meta-model (MDA-EFSM) for this model driven architecture.
In the design make use the following OO design patterns:

- state pattern
- strategy pattern
- abstract factory pattern

1.2. Assumptions for the project

1. Architecture is going to evolve over a period of time.
2. The elements of MDA architecture should be decoupled as much as possible.
3. The MDA-EFSM mode is platform independent and it should not be affected by any changes or versions of i/p and o/p processor.
4. The MDA-EFSM model should be deterministic model.
5. The MDA-EFSM model should capture the "general behavior" of both the ATM components. Only one meta-model for both the ATM components.
6. Meta-model is executable and is not having any excess to platform dependent model.
7. It is assumed that if the user over withdraw money from his account then he'll be charge $10 for ATM-1 and $100 for ATM-2.
8. It is assumed that after each unsuccessful withdrawal; system will go to "overdrawn" state irrespective of its balance after transaction.

2. Model-Driven Architecture of the component

2.1. MDA-EFSM model for the ATM components

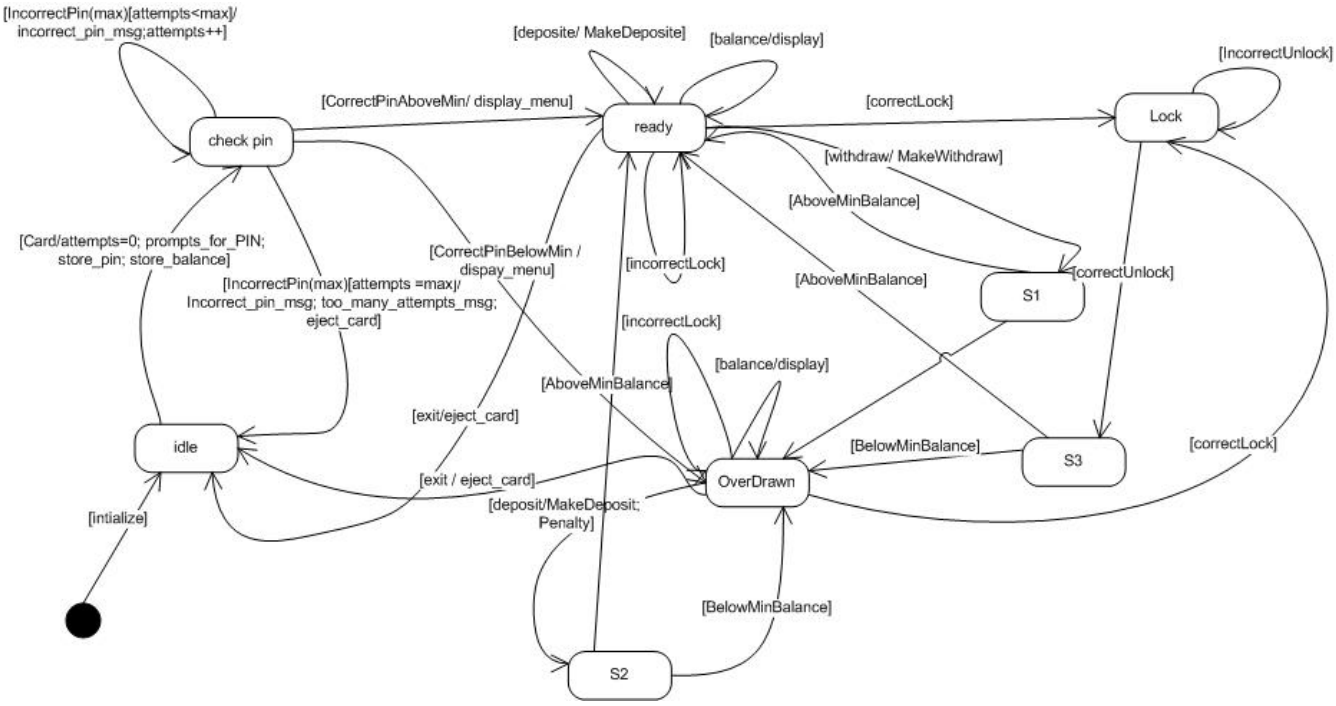
2.1.1. List of events for the MDA-EFSM

- initialize()
- card()
- deposit()
- withdraw()
- balance()
- incorrectPin(int max)
- correctPinBelowMin()
- correctPinAboveMin()
- belowMinBalance()
- aboveMinBalance()
- correctLock()
- incorrectLock()
- correctUnlock()
- incorrectUnlock()
- exit()

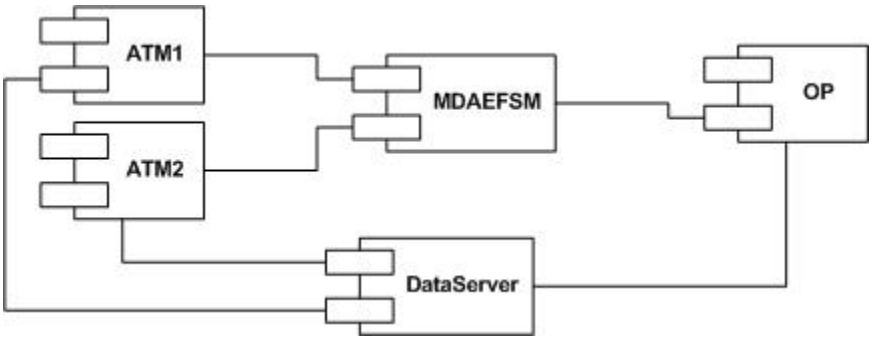
2.1.2. List of actions for the MDA-EFSM

- store_pin
- store_balance
- prompt_for_pin
- incorrect_pin_msg
- too_many_attempt_msg
- makeDepositemakeWithdraw
- aboveMinBalance
- belowMinBalanceMsg
- display_menu
- penalty
- eject_card
- lock
- unlock
- incorrectLockMsg

2.1.3. Diagram of the MDA-EFSM



2.2. MDA Architecture of the ATM Components



2.3. Description of elements of the MDA Architecture

2.3.1. Input Processor:

1. Input processor is consists of ATM-1 and ATM-2.
2. Input processor is responsible for taking the inputs from user and calling the appropriate methods of the metaModel(MDA-EFSM).
3. The output and behavior are not carried out in this component. It is responsible only for the input.

Collaborators: MetaModel (MDA-EFSM), DataServer and Param through the association relationship.

2.3.2. MDA-EFSM (MetaModel):

1. It is responsible for the internal behavior of the whole system.
2. MDA-EFSM does not have any access to the system dependent Data and does not return anything to the Input Processor.
3. It calls the methods of the Output Processor.
4. It composes its own data which stores the maximum number of attempts for the pin. This data is not part of the DataServer.

Collaborators: Output Processor, Input Processor (only for inputs) through the association relationship

2.3.3. Output Processor:

1. This component displays results on the console.
2. It has no direct relationship with the Input Processor. They are connected only via MetaModel.
3. It takes its inputs from the MetaModel and display results using the Data from DataServer.

Collaborators: MetaModel(only for inputs), DataServer

2.3.4. DataServer:

1. This component is responsible for storing the data of the system.
2. The Input Processor writes to the DataServer and the Output Processor fetches data from the DataServer for its operations.
3. It does not compose any behavior in the system.

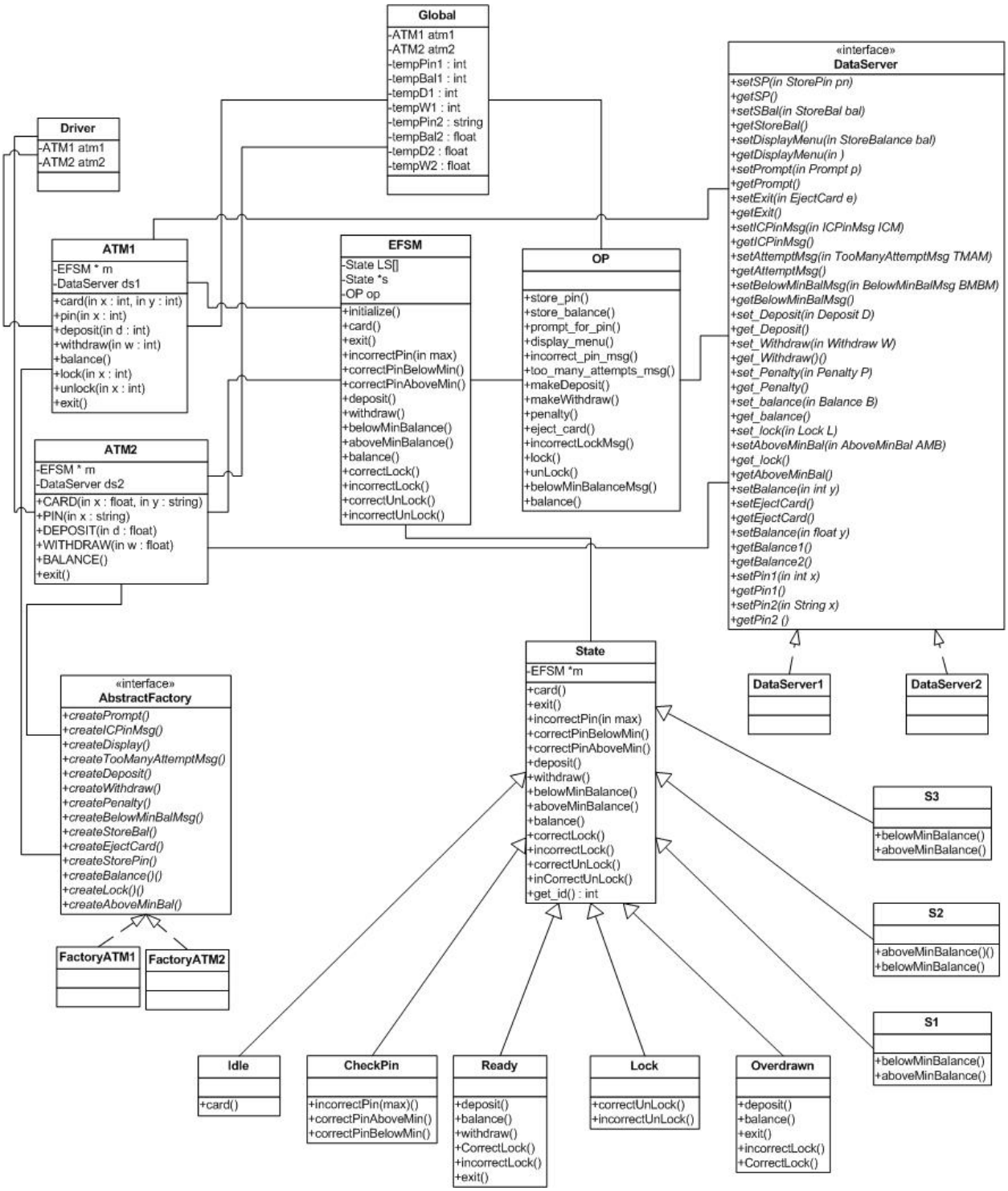
Collaborators: Input Processor (only for inputs), Output Processor.

2.4. Relationship between MDA elements

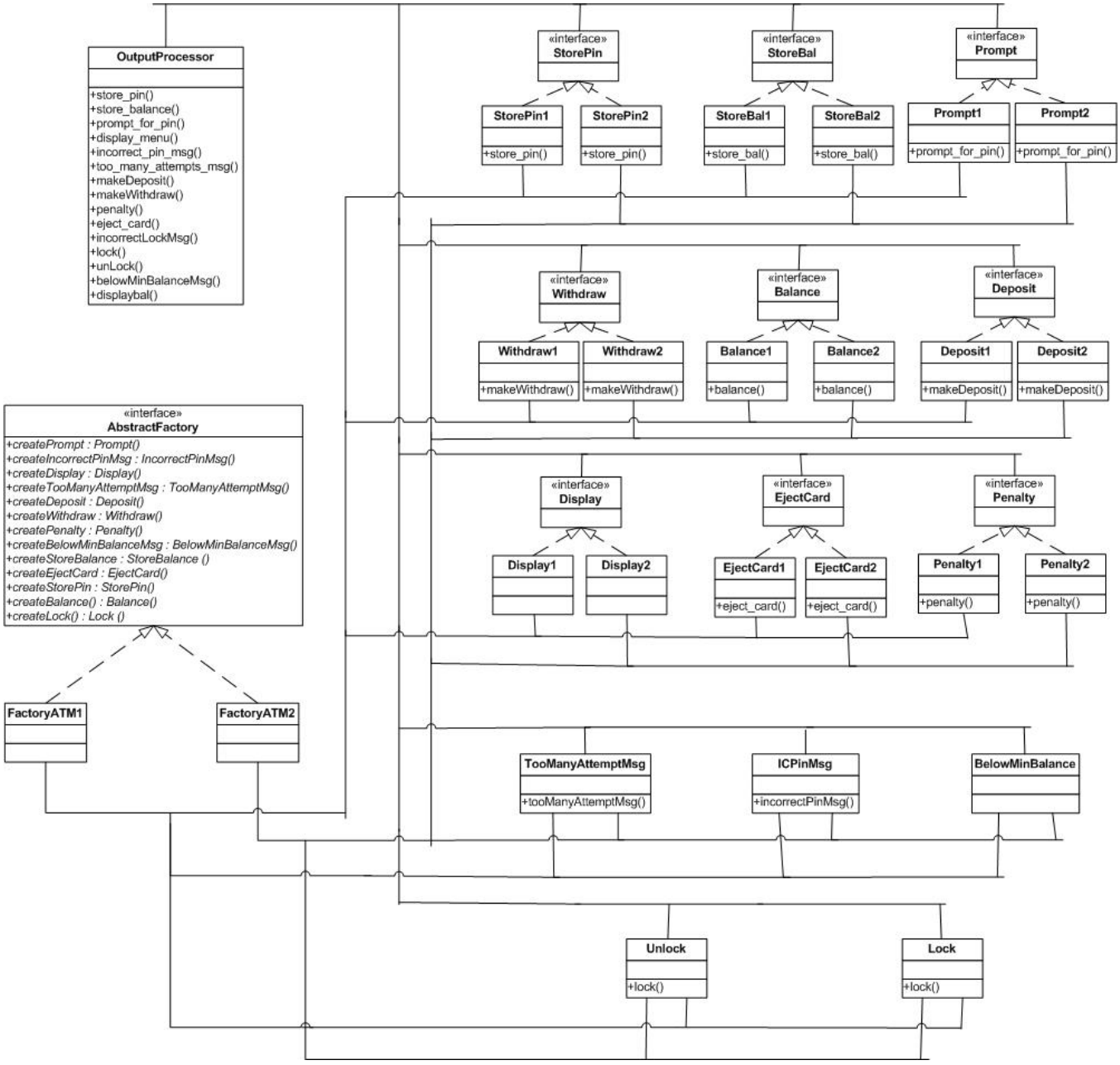
Element 1	Element 2	Relationship
InputProcessor (ATM1, ATM2)	MetaModel (EFSM)	InputProcessor call the MetaModel's Method
InputProcessor (ATM1, ATM2)	DataServer (DataServer, Param)	InputProcessor writes user data into DataServer
MetaModel (EFSM)	OutputProcessor (OP)	MetaModel calls the methods of Output Processor
DataServer (DataServer, Param)	OutputProcessor (OP)	OutputProcessor will get the data from DataServers and store the output in the DataServers
DataServer (DataServer, Param)	MetaModel (EFSM)	No relationship

3. Class Diagram of the MDA of the ATM Components

Class Diagram: Part I



Class Diagram: Part II



3.1. Purpose of the class

Class Name	Responsibility	Collaborator
ATM1	- Takes the input from the user for ATM1. - Call the methods of the MetaModel accordingly.	EFSM
ATM2	- Takes the input from the user for ATM2 - Call the methods of the MetaModel accordingly.	EFSM
EFSM	- Contains the behavior of the system. - It calls the appropriate methods of the output processor.	OP, State
State	- Implements the State Design Pattern. - Have various states like idle, ready etc.	Idle, CheckPin, Ready, Overdrawn, Lock, S1, S2, S3
Idle	- Implements the Idle state of the EFSM. - Responsible for handling the inputs in the **idle** state and calling the appropriate Output Processor method.	OP
CheckPin	- Implements the CheckPin state of the EFSM. - Responsible for handling the inputs in the Check Pin state and calling the appropriate Output Processor method.	OP
Ready	- Implements the Ready state of the EFSM. - It is responsible for handling the inputs in the Ready state and calling the appropriate Output Processor method.	OP
OverDrawn	- Implements the OverDrawn state of the EFSM. - Responsible for handling the inputs in the OverDrawn state and calling the appropriate OUTPUT PROCESSOR method.	OP

Class Name	Responsibility	Collaborator
Lock	- Implements the Locked state of the EFSM. - Responsible for handling the inputs in the Lock state and calling the appropriate Output Processor method.	OP
S1	- Implements S1 state of the MDA EFSM. - Responsible for checking the balance going into the Ready state or the OverDrawn state and calling the appropriate Output Processor method.	OP
OP	- Responsible for displaying the appropriate outputs on the console. - It displays separate messages for the ATM1 and ATM2 by calling the appropriate objects of the message classes.	Prompt, IncorrectPinMsg, Display, TooManyAttemptMsg, Deposit, Withdraw, Penalty, BelowMinBalanceMsg, Balance, Lock, Unlock, EjectCard
Prompt	Interface for prompt_for_pin message	OP
Prompt1	Implements prompt_for_pin for ATM1	Prompt
Prompt2	Implements prompt_for_pin for ATM2	Prompt
Display	Interface for displayMenu	OP
Display1	Implement displayMenu for ATM 1	Display
Display2	Implement displayMenu for ATM2	Display
TooManyAttemptsMsg	Interface for tooManyAttemptsMsg	OP
ICPinMsg	Interface for incorrectPinMsg	OP
Deposit	Interface for makeDeposit	OP
Deposit1	Implements Deposit for ATM1	Deposit
Deposit2	Implements Deposit for ATM2	Deposit
Withdraw	Interface for makeWithdraw	OP
Withdraw1	Implements Withdraw for ATM1	Withdraw
Withdraw2	Implements Withdraw for ATM2	Withdraw
Penalty	Interface for penalty	OP

Class Name	Responsibility	Collaborator
Penalty1	Implement Penalty for ATM1	Penalty
Penalty2	Implement Penalty for ATM2	Penalty
BelowMinBal	Interface for belowMinBalanceMsg	OP
AboveMinBal	Interface for AboveMinBal	OP
AboveMinBal1	Implements AboveMinBal for ATM1	AboveMinBal
AboveMinBal2	Implements AboveMinBal for ATM2	AboveMinBal
Balance	Interface for balance	OP
Balance1	Implements Balance for ATM1	Balance
Balance2	Implements Balance for ATM2	Balance
EjectCard	Interface for eject_card	OP
EjectCard1	Implements EjectCard for ATM1	EjectCard
EjectCard2	Implements EjectCard for ATM2	EjectCard
Lock	Lock message for ATM1	OP
Unlock	Unlock functionality for ATM1	OP
incorrectLockMsg	IncorrectLock Message for ATM1	OP
Abstract Factory	Interface responsible for implementing abstract factory design pattern.	Account_1
FactoryATM1	Implements AbstractFactory for ATM 1	AbstractFactory
FactoryATM2	Implements AbstractFactory for ATM 2	AbstractFactory
DataServer	Interface for the Database of the System	ATM1, ATM2, OP
DataServer1	The implementation of the database for ATM1	ATM1, OP
DataServer2	The implementation of the database for ATM2	ATM2, OP
Global	Responsible for the storing the global variables	ATM1, ATM2, OP

3.2. Pseudo-code

3.2.1. Pseudo Code for ATM1

```
card (int x, int y)
 {
        Create object of AbstractFactory;
        Create object of ConcreteAMT1;
        Create data1 object;
        store x; store y;
        m->card();
}
pin (int x){
    if (x = = pn) {
          if (b<100)
             m->correctPinBelowMin();
           else
             m -> correctPinAboveMin();
     }
    else {
           m-> incorrectPin(2);
           if(j<2) {
           j++;
           pin(Param.k);
           }
     }
}
deposit (int d)
{
   store d;
   m->deposit();
   if (b<100)
     m->belowMinBalance();
  else
     m->aboveMinBalance();
}
withdraw (int w)
{
     store w;
     m->withdraw();
     if (b<100)
          m->belowMinBalance();
      else
          m->aboveMinBalance();
}
```

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

lock (int x){
    If (x == pn)
        m -> correctLock();
   else
        m -> incorrectLock();
}
unlock (int x){
    If (x == pn)
            m -> correctUnLock();
   else
            m -> incorrectUnLock();
}
lock (int x){
       If(x == pn)
              m-> correctLock();
       else
               m-> incorrectLock();
}
unlock (int x){
        If(x == pn)
           m->correctUnlock();
       else
          m->incorrectUnlock();
}
exit(){
       m -> exit();
}
```

3.2.2. Pseudo Code for ATM2

```
CARD (float x, String y) {
     store x;
     store y;
     m->card();
}

PIN (string x){
    if (x = = pn)
       {
          if(b<1000)
              m->belowMinBalance();
           else
```

```
                m -> aboveMinBalance();
        }
    else
        m-> incorrectPin(2);
}
DEPOSIT (float d) {
        store d;
        m-> deposit();
        if (b<1000)
            m->belowMinBalance();
         else
          m->aboveMinBalance();
}
WITHDRAW (float w){
         Store w;
         m-> withdraw();
         if (b<1000)
          m->belowMinBalance();
         else
             m->aboveMinBalance();
}
BALANCE(){
          m->balance();
}
EXIT(){
      m -> exit();
}
```

3.2.3. Pseudo Code for EFSM

```
card(){
       if (s->get_id() == 0)
         {
          s-> card ();
             s=LS[1];
          }
}
incorrectPin (int max){
       if(s->get_id()==1)
       {
          s->incorrectPin(max);
          if (v->attempts >=max)
                    s=LS[0];
          else
                    s = LS[1];
        }
}
```

```
correctPinBelowMin(){
    if(s->getId()==1) {
         s-> correctPinBelowMin ();
         s=LS[6];
       }
}
correctPinAboveMin(){
      if(s->getId()==1)
       {
          s-> correctPinAboveMin ();
          s=LS[2];
       }
}
deposit(){
     if(s->getId()==2) {
        s->deposit();
        s = LS[2];
       }
      if (s->getId()==6) {
          s->deposit();
          s = LS[7];
       }
}
withdraw (){
   If (s->getId()==2]) {
          s->withdraw();
           s = LS[3];
    }
belowMinBalance(){
If (s->getId() == 3 || s->getId () == 5 || s->getId () == 7)
         {
            s -> belowMinBalance();
            s = LS[6]
          }
}
aboveMinBalance(){
      If (s->getId() == 3 || s->getId () == 5 || s->getId () == 7)
        {
           s -> aboveMinBalance();
           s = LS [2]
         }}
correctLock (){
    If (s->getId () == 2 || s->getId () == 6){
           s -> correctLock();
           s = LS [4]
         }}
```

```
incorrectLock (){
    If (s->getId () == 2) {
            s->incorrectLock();
            s = LS [2]
          }
    If (s->getId () == 6) {
            s->incorrectLock();
            s = LS [6]
          }
 }
correctUnlock(){
    If (s->getId () == 4) {
            s-> correctUnlock ();
            s = LS [5]
          }
}
incorrectUnlock(){
    If (s->getId () == 4) {
            s-> incorrectUnlock ();
            s = LS [4]
          }
}
balance(){
   If (s->getId () == 2)
      {
          s->balance();
          s = LS[2];
      }
    if (s->getId()== 6)
     {
          s->balance();
          s = LS[6];
     }
}
exit(){
    If (s->getId () == 2 || s->getId () == 6) {
          s->exit();
          s = LS[0];
      }
}
```

3.2.4. Pseudo Code for Idle Class

```
card (){
 op -> store_pin();
 op -> store_balance();
 op -> prompt_for_pin();
}
```

3.2.5. Pseudo Code for CheckPin Class

```
incorrectPin (int max){
   If (v.attempts < max)     {
          op-> incorrectPinMsg ();
          v.attempts++;
         op->prompt_for_pin();
    }
   else    {
      op-> incorrectPinMsg ();
      op-> tooManyAttemptsMsg ();
    }
}
correctPinBelowMin(){
     op -> belowMinBalancMsg();
}
correctPinAboveMin(){
  Display menu;
}
```

3.2.6. Pseudo Code for Ready Class

```
deposit () {
   op->makeDeposit();
}
withdraw(){
   op->makeWithdraw();
}
correctLock(){
     op->lock();
     op->disp();
}
 incorrectLock(){
   op->incorrect_pin_msg();
   op->displayMenu();
}
balance(){
   op->balance();
   op->display_menu();
}
```

3.2.7. Pseudo Code for S1, S2, S3 Class

```
belowMinBalance(){
  op -> penalty();
   op->displayMenu();
}
```

```
aboveMinBalance(){
   op->displayMenu();
}
```

3.2.8. Pseudo Code for Overdrawn Class

```
deposit(){
   op->makeDeposit();
}
correctLock(){
     op -> lock();
     op-> disp();
}
 incorrectLock()
{
   op->incorrect_pin_msg();
   op->display_menu();
}
 balance(){
   op->balance();
   op->display_menu();
}
```

3.2.9. Pseudo Code for OP Class

```
store_pin(){
 sp -> store_pin();}
store_balance(){
 sb -> store_balance();}
prompt_for_pin (){
   ppp->prompt_for_pin();
}
incorrectPinMsg(){
   ipm->incorrectPinMsg();
}
tooManyAttemptsMsg(){
   tmam-> tooManyAttemptsMsg();
}
makeDeposit(){
   md->makeDeposit();
}
makeWithdraw(){
   mw->makeWithdraw();
}
penalty(){
   pen->penalty();
}
```

```
eject_card (){
    ec ->eject_card();
}
incorrectLockMsg(){
   ilm -> incorrectLockMsg();
}
lock(){
 l ->lock();
}
unlock(){
 u -> unlock();
}
displayMenu(){
    dm->displayMenu();
}
belowMinBalanceMsg(){
    bmbm-> belowMinBalanceMsg ();
}
balance(){
    bal->balance();
}
```

3.2.10. Pseudo Code for FactoryATM1 Class

```
createPrompt (){
        return new Prompt1();
}
createIncorrectPinMsg (){
        return new IncorrectPinMsg1();
}
createDisplay(){
        return new Display1();
 }
createTooManyAttemptsMsg (){
        return new TooManyAttemptsMsg1();
}
createDeposit (){
        return new Deposit1();
 }
createWithdraw (){
        return new Withdraw1();
    }
createPenalty(){
        return new Penalty1();
    }
createBelowMinBalanceMsg(){
        return new BelowMinBalanceMsg1();
 }
```

```
createBalance(){
    return new Balance1();
}
createLock(){
    return new lock();
}
createUnlock(){
    return new unlock();
}
```

3.2.11. Pseudo Code for FactoryATM2 Class

```
create Prompt (){
    return new Prompt2();
}
createIncorrectPinMsg(){
    return new IncorrectPinMsg2();
}
createDisplay(){
    return new Display2(); }
createTooManyAttemptsMsg(){
    return new TooManyAttemptsMsg2();
}
createDeposit(){
    return new Deposit2();
  }
createWithdraw(){
    return new Withdraw2();
}
createPenalty(){
    return new Penalty2();
}
createBelowMinBalanceMsg(){
    return new BelowMinBalanceMsg ();
}
createBalance(){
    return new Balance2();
}
```

3.2.12. Pseudo Code for Data1 Class

```
setPin(int pin){
   p=pin;
}
getPin(){
   return p;
}
setBalance (int balance){
   b = balance;
```

```
}
getBalance () {
     return b;
 }
setDeposit(int deposit){
   d = deposit;
 }
 getDeposit() {
     return d;
 }
setWithdraw(int withdraw){
  w = withdraw;
}
getWithdraw(){
  return w;
}
setMaxAttempt (int max){
    maxAttempts = max;
}
getMaxAttempts(){
 return maxAttempts;
}
setMinBalance(int mb){
  minBalance = mb;
}
getMinBalance(){
 Return minBalance;
}
setPenalty (int pen){
 penalty = pen;
}
getPenalty(){
 return penalty;
}
public void setStorePin(StorePin p){
 this.sp = p;
}
```

Similar methods for all the classes associated with ConcreteATM1

3.2.13. Pseudo Code for Data2 Class

```
setPin(string pin){
       p=pin;
}
getPin(){
       peturn pin;
}
setBalance (float balance){
```

```
    b = balance;
}
getBalance () {
    return b;
 }
setDeposit(float deposit){
d = deposit;
 }
getDeposit() {
     return d;
 }
setWithdraw(float withdraw){
w = withdraw;
}
getWithdraw(){
          return w;
}
setMaxAttempt (int max){
    maxAttempts = max;
}
getMaxAttempts(){
 return maxAttempts;
}
setMinBalance(float mb){
  minBalance = mb;
}
getMinBalance(){
 Return minBalance;
}
setPenalty (float pen){
 penalty = pen;
}
getPenalty(){
 return penalty;
}
public void setStorePin(StorePin p){
 this.sp = p;
}
```

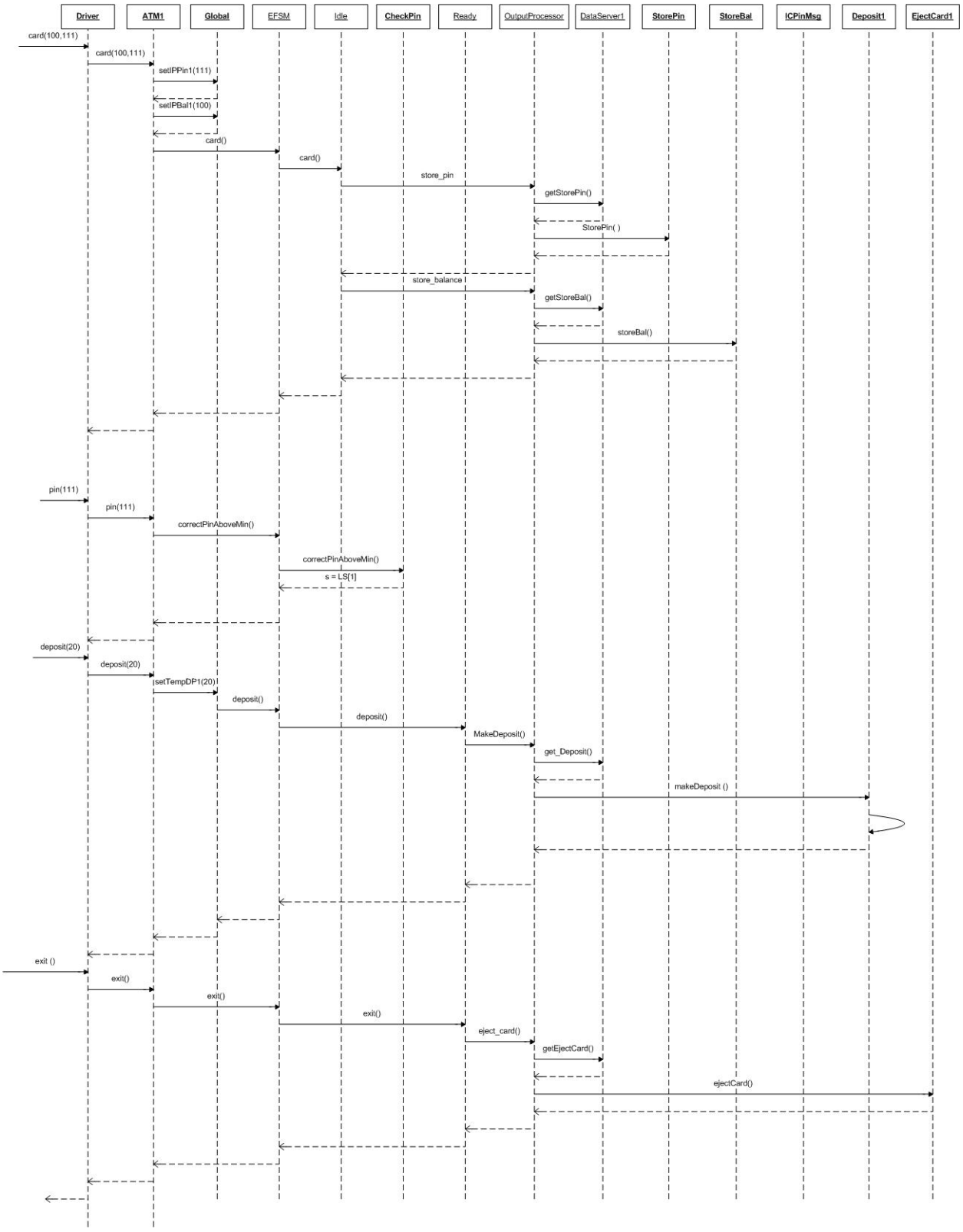
Similar methods for all the classes associated with ConcreteATM2

3.3. Attributes of the class

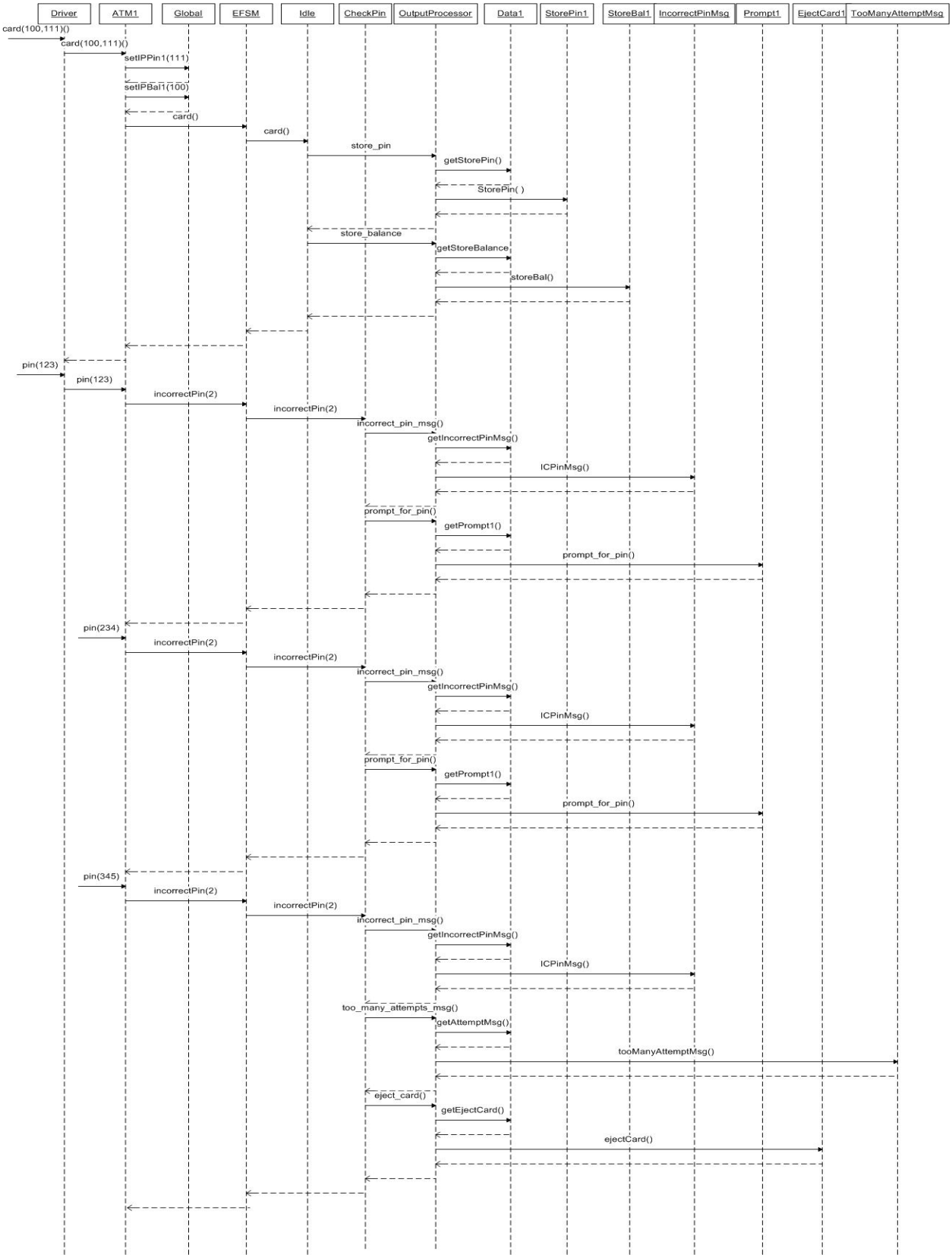
Class Name	Attributes
ATM1	EFSM m; DataServer ds1;
ATM2	EFSM m; DataServer ds2;
EFSM	State *s; State LS[]; OP op
State	OP op ,ME v
Idle	OP op ,ME v
CheckPin	OP op ,ME v
Ready	OP op ,ME v
OverDrawn	OP ,ME v
Lock	OP ,ME v
S1,S2,S3	OP ,ME v
DataServer1	References of the all the classes that are collaborated with OP
DataServer2	References of the all the classes that are collaborated with OP
Global	ATM1, ATM2.

4. Dynamics

4.1. Sequence Diagram: Scenario I



4.2. Sequence Diagram: Scenario II



5. Conclusions

The MDA architecture used in the ATM System separates the behavior of the system from the input and output. It is a very good way to design for cross platform systems. There are many benefits to using the MDA approach:

- It is more concentrated on application's business functionality and behavior which allows the investors to concentrate more on the aspects that critically affect core business processes.
- Makes it easier to integrate applications and facilities across middleware boundaries.
- Provides higher interoperability by always being available on a domain's preferred platform defined by the Domain facilities, and on multiple platforms whenever there is a need.

Thus MDA has a great scope in the future

6. Source Code

6.1. Driver

6.1.1. Driver.java

```
package Driver;
import java.io.*;

import ATM.*;
public class driver {

   // Display menu for ATM - 1
   public static void menu1() throws IOException
   {
         ATM1 atm1 = new ATM1();
         char menuChoice;
         System.out.println("\n\n WELCOME TO ATM-1");
         do
         {
         System.out.println("\nPlease select the operation: " +
                     "\n\t 1. card(int, int)" +
                     "\n\t 2. pin(int)" +
                     "\n\t 3. deposit(int)" +
                     "\n\t 4. withdraw(int)" +
                     "\n\t 5. balance()" +
                     "\n\t 6. exit()" +
                     "\n\t 7. lock(int)" +
                     "\n\t 8. unlock(int)" +
                     "\n\t q. Quit the program");

         BufferedReader buffStr = new BufferedReader (new
InputStreamReader (System.in));
         menuChoice = buffStr.readLine().charAt(0);
         int x, y, p, d, w;
         switch(menuChoice)
         {
         case '1': System.out.println("Operation: card(int, int)");
```

```
                    System.out.print("\t Balance:");
                    x = Integer.parseInt(buffStr.readLine());
                    System.out.print("\t Pin:");
                    y = Integer.parseInt(buffStr.readLine());
                    atm1.card(x, y);
                    break;
            case '2': System.out.println("Operation: pin(int)");
                    System.out.print("\t Pin:");
                    p = Integer.parseInt(buffStr.readLine());
                    atm1.pin(p);
                    break;
            case '3':System.out.println("Operation: deposit(int)");
                    System.out.print("\t Amount:");
                    d = Integer.parseInt(buffStr.readLine());
                    atm1.deposit(d);
                    break;
            case '4':System.out.println("Operation: Withdraw(int)");
                    System.out.print("\t Amount:");
                    w = Integer.parseInt(buffStr.readLine());
                    atm1.withdraw(w);
                    break;
            case '5':System.out.println("Operation: balance()");
                    atm1.balance();
                    break;
            case '6':System.out.println("Operation: exit()");
                    atm1.exit();
                    break;
            case '7':System.out.println("Operation: lock(int)");
                    System.out.print("\t Pin:");
                    p = Integer.parseInt(buffStr.readLine());
                    atm1.lock(p);
                    break;
            case '8':System.out.println("Operation: unlock(int)");
                    System.out.print("\t Pin:");
                    p = Integer.parseInt(buffStr.readLine());
                    atm1.unlock(p);
                    break;
            case 'q':System.out.println("Operation: Quit the program");
                    atm1.quit();
                    break;
            default: System.out.println("INVALID CHOICE");
            }
            }while(menuChoice != 'q');
    }

    // Display menu for ATM - 2
    public static void menu2() throws IOException
    {
            ATM2 atm2 = new ATM2();
            char menuChoice;
            System.out.println("\n\n WELCOME TO ATM-2");
            do
            {
            System.out.println("\nPlease select the operation: " +
                    "\n\t 1. CARD(float, String)" +
                    "\n\t 2. PIN(String)" +
                    "\n\t 3. DEPOSIT(float)" +
```

```
                        "\n\t 4. WITHDRAW(float)" +
                        "\n\t 5. BALANCE()" +
                        "\n\t 6. EXIT()" +
                        "\n\t q. Quit the program");
            BufferedReader buffStr = new BufferedReader (new
    InputStreamReader (System.in));
            menuChoice = buffStr.readLine().charAt(0);
            float x, d, w;
            String y, p;
            switch(menuChoice)
            {
            case '1': System.out.println("Operation: card(int, int)");
                        System.out.print("\t Balance:");
                        x = Float.parseFloat(buffStr.readLine());
                        System.out.print("\t Pin:");
                        y = buffStr.readLine();
                        atm2.CARD(x, y);
                        break;
            case '2': System.out.println("Operation: pin(int)");
                        System.out.print("\t Pin:");
                        p = buffStr.readLine();
                        atm2.PIN(p);
                        break;
            case '3':System.out.println("Operation: deposit(int)");
                        System.out.print("\t Amount:");
                        d = Float.parseFloat(buffStr.readLine());
                        atm2.DEPOSIT(d);
                        break;
            case '4':System.out.println("Operation: WITHDRAW(int)");
                        System.out.print("\t Amount:");
                        w = Float.parseFloat(buffStr.readLine());
                        atm2.WITHDRAW(w);
                        break;
            case '5':System.out.println("Operation: balance()");
                        atm2.BALANCE();
                        break;
            case '6':System.out.println("Operation: exit()");
                        atm2.EXIT();
                        break;
            case 'q':System.out.println("Operation: Quit the program");
                        atm2.QUIT();
                        break;
            default:System.out.println("INVALID CHOICE");
            }

            }
            while(menuChoice != 'q');
    }

    public static void mainMenu() throws IOException
    {
            char accChoice;
            ATM1 atm1 = new ATM1();
            ATM2 atm2 = new ATM2();

            System.out.println("\n");
            System.out.println("Please select the choice : "+
```

```
                        "\n\t 1. ATM-1, "+
                        "\n\t 2. ATM-2, "+
                        "\n\t 3. Exit"+
                        "\n\t Enter Choice:");

          BufferedReader buffStr = new BufferedReader (new
InputStreamReader (System.in));
          accChoice = buffStr.readLine().charAt(0);

          switch(accChoice)
          {
          case '1':
                        atm1.initialize();
                        menu1();
                        break;
          case '2':
                        atm2.initialize();
                        menu2();
                        break;
          case '3':
                default:
          }
    }

    public static void main(String[] args) throws IOException{
          // TODO Auto-generated method stub
          mainMenu();

    }

}
```

6.2. ATM Package

6.2.1. ATM1.java

```
/**
 * CLASS        : ATM 1
 * OPERATION    : Input model for MDA - EFSM, holds set of input methods
for ATM 1
 * AUTHOR             : Darshankumar Zala
 */

package ATM;
import java.io.IOException;

import DataServer.*;
import EFSM.*;
import Param.*;
import Driver.driver;

public class ATM1{

    EFSM m = new EFSM();
```

```
    AbstractFactory AF = new FactoryATM1();
    DataServer1 ds1 = new DataServer1();

    /**
     * Function : initialize()
     *                : Initialize global data structure and Abstract
Factory methods.
     */

    public void initialize()
    {
          Global.ds = ds1;
          Global.ds.setSP(AF.createStorePin());
          Global.ds.setStoreBal(AF.createStoreBal());
          Global.ds.setDisplayMenu(AF.createDisplay());
          Global.ds.setExit(AF.createEjectCard());
          Global.ds.setICPinMsg(AF.createICPinMsg());
          Global.ds.setPrompt(AF.createPrompt());
          Global.ds.setAttemptMsg(AF.createTooManyAttemptMsg());
          Global.ds.setBelowMinBalMsg(AF.createBelowMinBalMsg());
          Global.ds.set_Deposit(AF.createDeposit());
          Global.ds.set_Withdraw(AF.createWithdraw());
          Global.ds.set_Penalty(AF.createPenalty());
          Global.ds.set_balance(AF.createBalance());
          Global.ds.set_lock(AF.createLock());
          Global.ds.setAboveMinBal(AF.createAboveMinBal());
          Global.ds.setEjectCard(AF.createEjectCard());
    }

    /**
     * Function : Card(int x, int y)
     *                : Set input values for balance and pin
     */
    public void card(int x, int y)
    {
          Global.setIPPin1(y);
          Global.setIPBal1(x);

          m.card();
    }

    /**
     * Function : pin(int x)
     *                : Validate pin entered by user.
     */
    int j = 0;
    public void pin(int x)
    {
          int p = Global.ds.getPin1();
          if(p == x)
          {
                if(Global.ds.getBalance1() < 100)
                   m.correctPinBelowMin();
                else
                      m.correctPinAboveMin();
          }
          else
```

```
            {
                  m.incorrectPin(2);
                  if(j<2)
                  {
                   j++;
                   pin(Global.k);
                  }
                  else
                        m.exit();
            }
      }

      /**
       * Function : deposit(int d)
       *                : Perform deposit operation and set the state after
deposit according to balance.
       */
      public void deposit(int d)
      {
            Global.setTempDP1(d);
            m.deposit();

            if(Global.ds.getBalance1() < 100)
            {

                  m.belowMinBalance();
            }
            else
            {
                  m.aboveMinBalance();
            }

      }

      /**
       * Function : withdraw(int w)
       *                : Perform withdraw operation and apply penalty if
applicable.
       */
      public void withdraw(int w)
      {

            Global.setTempWD1(w);
            m.withdraw();
            if(Global.getTempWD1() < 100)
            {
                  m.belowMinBalance();
            }
            else
            {
                  m.aboveMinBalance();

            }
      }

      /**
       * Function : balance()
```

```
	 *                : Display current balance
	 */
	public void balance()
	{
		m.balance();
	}

	/*
	 * Function : lock(int x)
	 *                : Lock the account when user enters valid pin
	 */
	public void lock(int x)
	{
		System.out.println("Inside lock");
	    if(x == Global.ds.getPin1())
		  {
		System.out.println("Inside correct lock");
			m.correctLock();
		  }
	    else
	    {
		System.out.println("Inside incorrect lock");
		  m.incorrectLock();
	    }
	}

	/**
	 * Function : unlock(int x)
	 *                : Unlock the locked account for normal operations
when user enters valid pin
	 */
	public void unlock(int x)
	{
		System.out.println("Inside unlock");
		if(x == Global.ds.getPin1())
		{
			System.out.println("Inside correct unlock");
			m.correctUnlock();
			if(Global.ds.getBalance1() < 100)
			{
				m.belowMinBalance();
			}
			else
			{
				m.aboveMinBalance();
			}
		}
		else
			{
			m.incorrectUnlock();
			}
	}

	/**
	 * Function : exit()
	 *                : Eject card and reset the ATM to initial stage.
	 */
```

```
    public void exit()
    {
          m.exit();
    }

    /**
     * Function : quit()
     *                : Finish the operation of the ATM 1 and go back to
main menu.
     */
    public void quit() throws IOException
    {
          System.out.println("Thank you for using ATM 1");
          driver.mainMenu();
    }
}
```

6.2.2. ATM2

```
/*
 * CLASS        : ATM 2
 * OPERATION    : Input model for MDA - EFSM, holds set of input methods
for ATM 2
 * AUTHOR             : Darshankumar Zala
 */


package ATM;
import java.io.IOException;

import DataServer.*;
import Driver.driver;
import EFSM.EFSM;
import Param.AbstractFactory;
import Param.FactoryATM2;

public class ATM2{

    EFSM m = new EFSM();

    DataServer2 ds2 = new DataServer2();
    AbstractFactory AF = new FactoryATM2();
    int j = 0;

    /**
     * Function : initialize()
     *                : Initialize global data structure and Abstract
Factory methods.
     */
    public void initialize()
    {
          Global.ds =ds2;

          Global.ds.setSP(AF.createStorePin());
          Global.ds.setStoreBal(AF.createStoreBal());
          Global.ds.setDisplayMenu(AF.createDisplay());
          Global.ds.setExit(AF.createEjectCard());
```

```
        Global.ds.setICPinMsg(AF.createICPinMsg());
        Global.ds.setPrompt(AF.createPrompt());
        Global.ds.setAttemptMsg(AF.createTooManyAttemptMsg());
        Global.ds.setBelowMinBalMsg(AF.createBelowMinBalMsg());
        Global.ds.set_Deposit(AF.createDeposit());
        Global.ds.set_Withdraw(AF.createWithdraw());
        Global.ds.set_Penalty(AF.createPenalty());
        Global.ds.set_balance(AF.createBalance());
        Global.ds.setAboveMinBal(AF.createAboveMinBal());
        Global.ds.setEjectCard(AF.createEjectCard());
    }

    /**
     * Function : CARD(float x, String y)
     *                : Set input values for balance and pin
     */
    public void CARD(float x, String y)
    {
        Global.setIPBal2(x);
        Global.setIPPin2(y);
        Global.ds.setPrompt(AF.createPrompt());

        m.card();
    }

    /**
     * Function : PIN(String x)
     *                : Validate pin entered by user.
     */
    public void PIN(String x)
    {
        String p = Global.ds.getPin2();

        if(p.equals(x))
        {

            if(Global.ds.getBalance2() < 1000.00)
               m.correctPinBelowMin();
            else
                m.correctPinAboveMin();
        }
        else
        {
            m.incorrectPin(3);
            if(j<3)
            {
             j++;
             PIN(Global.k1);
            }
            else
                m.exit();
        }
    }

    /**
     * Function : DEPOSIT(float d)
```

```
     *                : Perform deposit operation and set the state after
deposit according to balance.
     */
    public void DEPOSIT(float d)
    {
          Global.setTempDeposit2(d);
          m.deposit();

          if(Global.ds.getBalance2() < 1000)
          {

                m.belowMinBalance();
          }
          else
          {
                m.aboveMinBalance();
          }
    }

    /**
     * Function : WITHDRAW(float w)
     *                : Perform withdraw operation and apply penalty if
applicable.
     */
    public void WITHDRAW(float w)
    {
          Global.setTempWithdraw2(w);
          m.withdraw();
          if(Global.getTempWithdraw2() < 1000)
          {
                m.belowMinBalance();
          }
          else
          {
                m.aboveMinBalance();
          }
    }

    /**
     * Function : BALANCE()
     *                : Display current balance
     */
    public void BALANCE()
    {
          m.balance();
    }

    /**
     * Function : EXIT()
     *                : Eject card and reset the ATM to initial stage.
     */
    public void EXIT()
    {
          m.exit();
    }

    /**
```

```
     * Function : QUIT()
     *                : Finish the operation of the ATM 1 and go back to
main menu.
     */
    public void QUIT() throws IOException
    {
          System.out.println("Thank you for using ATM 2");
          driver.mainMenu();
    }

}
```

6.3. DataServer Package

6.3.1. DataServer.java

```
/*
 * INTERFACE    : DataServer
 * OPERATION    : Provide interface for the DataServers of ATM1 and ATM2
 * AUTHOR             : Darshankumar Zala
 */

package DataServer;

import OP.*;

public interface DataServer {

   public int amt = 0;

// Abstract Factory Implementations

// Get and Set methods for Store Pin
   public void setSP(StorePin pn);
   public StorePin getSP();

// Get and Set methods for Store Pin
   public void setStoreBal(StoreBal bal);
   public StoreBal getStoreBal();

// Get and Set methods for Display Menu
   public void setDisplayMenu(Display d);
   public Display getDisplayMenu();

// Get and Set methods for Prompt
   public void setPrompt(Prompt p);
   public Prompt getPrompt();

// Get and Set methods for Exit
   public void setExit(EjectCard e);
   public EjectCard getExit();

// Get and Set methods for Incorrect Pin Message
   public void setICPinMsg(ICPinMsg IPM);
   public ICPinMsg getICPinMsg();
```

```
// Get and Set methods for Too Many Attempts Message
   public void setAttemptMsg (TooManyAttemptMsg TMAM);
   public TooManyAttemptMsg getAttemptMsg();

// Get and Set methods for Below Minimum Balance Message
   public void setBelowMinBalMsg (BelowMinBalMsg BMBM);
   public BelowMinBalMsg getBelowMinBalMsg();

// Get and Set methods for Deposit
   public void set_Deposit (Deposit D);
   public Deposit get_Deposit();

// Get and Set methods for Withdraw
   public void set_Withdraw (Withdraw W);
   public Withdraw get_Withdraw();

// Get and Set methods for Penalty
   public void set_Penalty(Penalty P);
   public Penalty get_Penalty();

// Get and Set methods for Balance
   public void set_balance(Balance B);
   public Balance get_balance();

// Get and Set methods for Lock
   public void set_lock(Lock L);
   public Lock get_lock();

// Get and Set methods for Above Minimum Balance
   public void setAboveMinBal(AboveMinBal AMB);
   public AboveMinBal getAboveMinBal();

// Get and Set methods for Eject Card
   public void setEjectCard(EjectCard EC);
   public EjectCard getEjectCard();

// Get and Set methods for storing Pin and Balance
   public void setBalance(int y);
   public void setBalance(float y);
   public int getBalance1();
   public float getBalance2();
   public void setPin1(int x);
   public int getPin1();
   public void setPin2(String x);
   public String getPin2();

}
```

6.3.2. DataServer1.java

```
package DataServer;

import OP.*;

public class DataServer1 implements DataServer{

private int b = 0;                    // Balance
```

```
private int p = 0;                    // Pin
private int d = 0;                    // Deposit
private int w = 0;                    // Withdraw
private int maxAttempt = 2;       // Maximum number of pin attempt
private int minBalance= 100;      // Minimum bank balance
private int penalty = 10;         // Penalty amount on less than minimum
balance
private StorePin sp;
private Display dm;
private Prompt pm;
private StoreBal sb;
private EjectCard ej;
private ICPinMsg ipm;
private TooManyAttemptMsg tmam;
private BelowMinBalMsg bmbm;
private Withdraw wt;
private Deposit dp;
private Penalty pp;
private Balance BB;
private Lock LL;
private AboveMinBal abm;
private EjectCard ec;

public void setPin1(int x) {
this.p = x;
}

public int getPin1() {
return this.p;
}

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

public void setBalance(float y) {}

public int getBalance1() {
return b;
}

public float getBalance2() {
return 0;
}

public void setSP(StorePin p){
this.sp = p;
}

public StorePin getSP(){
return this.sp;
}

public void setStoreBal(StoreBal b)
{
this.sb = b;
}
```

```
public StoreBal getStoreBal()
{
return this.sb;
}

public void setPrompt(Prompt p) {
this.pm = p;
}

public Prompt getPrompt() {
return this.pm;
}

public void setDisplayMenu(Display d)
{
this.dm = d;
}

public Display getDisplayMenu()
{
return this.dm;
}

public void setICPinMsg(ICPinMsg IPM) {
this.ipm = IPM;
}

public ICPinMsg getICPinMsg() {
return this.ipm;
}

public void setAttemptMsg(TooManyAttemptMsg TMAM) {
this.tmam = TMAM;
}

public TooManyAttemptMsg getAttemptMsg() {
return this.tmam;
}

public void setBelowMinBalMsg(BelowMinBalMsg BMBM) {
this.bmbm = BMBM;
}

public BelowMinBalMsg getBelowMinBalMsg() {
return this.bmbm;
}

public void set_Deposit(Deposit D) {
this.dp = D;
}

public Deposit get_Deposit() {
return this.dp;
}

public void set_Withdraw(Withdraw W) {
```

```
this.wt = W;
}

public Withdraw get_Withdraw() {
return this.wt;
}

public void set_Penalty(Penalty P)
{
this.pp = P;
}
public Penalty get_Penalty()
{
return this.pp;
}

public void set_balance(Balance B)
{
this.BB = B;
}
public Balance get_balance()
{
return this.BB;
}


// Get deposit amount
public int getDeposit()
{
return this.d;
}
// Set deposit amount
public void setDeposit(int amt)
{
this.d = amt;
}
// Get withdraw amount
public int getWithdraw()
{
return this.w;
}
// Set withdraw amount
public void setWithdraw(int amt)
{
this.w = amt;
}

// Get Maximum attempt
public int getMaxAttempt()
{
return this.maxAttempt;
}
// Set withdraw amount
public void setMaxAttempt(int a)
{
this.maxAttempt = a;
}
```

```
// Get Minimum balance amount
public int getMinBalance()
{
return this.minBalance;
}
// Set Minimum balance amount
public void setMinBalance(int amt)
{
this.minBalance = amt;
}
// Get Penalty amount
public int getPenalty()
{
return this.penalty;
}
// Set Penalty amount
public void setPenalty(int amt)
{
this.penalty = amt;
}

public void setExit(EjectCard e)
{
this.ej = e;
}
public EjectCard getExit()
{
return this.ej;
}

public void set_lock(Lock L)
{
this.LL = L;
}
public Lock get_lock()
{
return this.LL;
}

@Override
public String getPin2() {
return null;
}

@Override
public void setPin2(String x) {

}

@Override
public void setAboveMinBal(AboveMinBal AMB) {
this.abm = AMB;
}

@Override
public AboveMinBal getAboveMinBal() {
return this.abm;
```

```
}

public void setEjectCard(EjectCard EC) {
this.ec = EC;
}

public EjectCard getEjectCard() {
return this.ec;
}
}
```

6.3.3. DataServer2.java

```
package DataServer;

import OP.AboveMinBal;
import OP.Balance;
import OP.BelowMinBalMsg;
import OP.Deposit;
import OP.Display;
import OP.EjectCard;
import OP.ICPinMsg;
import OP.Lock;
import OP.Penalty;
import OP.Prompt;
import OP.StoreBal;
import OP.StorePin;
import OP.TooManyAttemptMsg;
import OP.Withdraw;

public class DataServer2 implements DataServer{

   private float b = 0; // Balance
   private String p = ""; // Pin
   private float d = 0; // Deposit
   private float w = 0; // Withdraw
   private int maxAttempt = 2; // Maximum number of pin attempt
   private int minBalance = 100; // Minimum bank balance
   private float penalty = 2;    // Panelty amount on less than minimum
balance
   private StorePin sp;
   private Display dm;
   private Prompt pm;
   private StoreBal sb;
   private EjectCard ej;
   private ICPinMsg ipm;
   private TooManyAttemptMsg tmam;
   private BelowMinBalMsg bmbm;
   private Withdraw wt;
   private Deposit dp;
   private Penalty pp;
   private Balance BB;
   private Lock LL;
   private AboveMinBal abm;
   private EjectCard ec;
```

```
//private Unlock UL;

// Get Pin
public String getPin2()
{
	return p;
}
// Set Pin
public void setPin2(String pin)
{
	this.p = pin;
}
// Get Balance
public float getBalance2()
{
	return b;
}
public int getBalance1()
{
	return 0;
}
// Set balance
public void setBalance(float bal)
{
	this.b = bal;
}
// Get deposit amount
public float getDeposit()
{
	return this.d;
}
// Set deposit amount
public void setDeposit(float amt)
{
	this.d = amt;
}
// Get withdraw amount
public float getWithdraw()
{
	return this.w;
}
// Set withdraw amount
public void setWithdraw(float amt)
{
	this.w = amt;
}

// Get Maximum attempt
public int getMaxAttempt()
{
	return this.maxAttempt;
}
// Set withdraw amount
public void setMaxAttempt(int a)
{
	this.maxAttempt = a;
}
```

```
// Get Minimum balance amount
public int getMinBalance()
{
	return this.minBalance;
}
// Set Minimum balance amount
public void setMinBalance(int amt)
{
	this.minBalance = amt;
}
// Get Penalty amount
public float getPenalty()
{
	return this.penalty;
}
// Set Penalty amount
public void setPenalty(float amt)
{
	this.penalty = amt;
}

public void setSP(StorePin p)
{
	this.sp = p;
}

public StorePin getSP()
{
	return this.sp;
}

public void setStoreBal(StoreBal b)
{
	this.sb = b;
}

public StoreBal getStoreBal()
{
	return this.sb;
}

public void setPrompt(Prompt p) {
	this.pm = p;
}

public Prompt getPrompt() {
	return this.pm;
}

public void setDisplayMenu(Display d)
{
	this.dm = d;
}

public Display getDisplayMenu()
{
	return this.dm;
```

```
}
public void setExit(EjectCard e)
{
	this.ej = e;
}
public EjectCard getExit()
{
return this.ej;
}

public void setICPinMsg(ICPinMsg IPM) {
	this.ipm = IPM;
}

public ICPinMsg getICPinMsg() {
	return this.ipm;
}

public void setAttemptMsg(TooManyAttemptMsg TMAM) {
	this.tmam = TMAM;
}

public TooManyAttemptMsg getAttemptMsg() {
	return this.tmam;
}

public void setBelowMinBalMsg(BelowMinBalMsg BMBM) {
	this.bmbm = BMBM;
}

public BelowMinBalMsg getBelowMinBalMsg() {
	return this.bmbm;
}

public void set_Deposit(Deposit D) {
	this.dp = D;
}

public Deposit get_Deposit() {
	return this.dp;
}

public void set_Withdraw(Withdraw W) {
	this.wt = W;
}

public Withdraw get_Withdraw() {
	return this.wt;
}
public void set_Penalty(Penalty P)
{
	this.pp = P;
}

public Penalty get_Penalty()
{
	return this.pp;
```

```
	}

	public void set_balance(Balance B)
	{
		this.BB = B;
	}
	public Balance get_balance()
	{
		return this.BB;
	}
	public void set_lock(Lock L)
	{
		this.LL = L;
	}
	public Lock get_lock()
	{
		return this.LL;
	}

	@Override
	public void setAboveMinBal(AboveMinBal AMB) {
		this.abm = AMB;
	}

	@Override
	public AboveMinBal getAboveMinBal() {
		return this.abm;
	}

	public void setEjectCard(EjectCard EC) {
		this.ec = EC;
	}

	public EjectCard getEjectCard() {
		return this.ec;
	}


	public void setBalance(int x) {
		// TODO Auto-generated method stub

	}
	@Override
	public void setPin1(int x) {
		// TODO Auto-generated method stub

	}
	@Override
	public int getPin1() {
		// TODO Auto-generated method stub
		return 0;
	}
}
```

6.3.4. Global.java

```
package DataServer;

public class Global {
    public static DataServer ds = null;

    private static int tempPin1;
    private static int tempBal1;
    private static String tempPin2;
    private static float tempBal2;
    private static int tempD1;
    private static float tempD2;
    private static int tempW1;
    private static float tempW2;

    public static int k;
    public static int pin;
    public static String k1;
    // ATM 1
    public static void setIPPin1(int p){     tempPin1 = p;}

   public static int getIPPin1() {return tempPin1;}

   public static void setIPBal1(int b){      tempBal1 = b;}

   public static int getIPBal1(){return tempBal1;}

   public static void setTempDP1(int d){tempD1 = d;}

   public static int getTempDP1(){return tempD1;}

   public static void setTempWD1(int w){tempW1 = w;}

   public static int getTempWD1(){return tempW1;}

   // ATM 2
    public static void setIPPin2(String p){  tempPin2 = p;}

   public static String getIPPin2() {return tempPin2;}

   public static void setIPBal2(float b){    tempBal2 = b;}

   public static float getIPBal2(){return tempBal2;}

   public static void setTempDeposit2(float d){tempD2 = d;}

   public static float getTempDeposit2(){return tempD2;}

   public static void setTempWithdraw2(float w){tempW2 = w;}

   public static float getTempWithdraw2(){return tempW2;}

}
```

6.4. EFSM Package

6.4.1. EFSM.java

```
package EFSM;

import OP.*;
import Param.*;

public class EFSM {

   private State LS[] = null;
   private State S = null;
   private OutputProcessor op;
   ME v = new ME();
   public EFSM()
   {
         LS = new State[10];
         LS[0] = new Idle(this, op, IConstants.IDLE_STATE_ID);
         LS[1] = new CheckPin(this, op, IConstants.CHECK_PIN_STATE_ID);
         LS[2] = new Ready(this, op, IConstants.READY_STATE_ID);
         LS[3] = new S1(this, op, IConstants.S1_STATE_ID);
         LS[4] = new Lock(this, op, IConstants.LOCK_STATE_ID);
         LS[5] = new S3(this, op, IConstants.S3_STATE_ID);
         LS[6] = new Overdrawn(this, op, IConstants.OVERDRAWN_STATE_ID);
         LS[7] = new S2(this, op, IConstants.S2_STATE_ID);
         S = LS[ME.id];
   }


   public void initialize()
   {}

   public void card()
   {
         if(S.get_id()==0)
         {
               ME.attempts = 0;
               S.card();
               ME.id = 1;
               S = LS[ME.id];
         }
   }

   public void exit()
   {
         System.out.println("State: " + S.get_id());
         if(S.get_id()==1||S.get_id()==2 || S.get_id()==6)
         {
               S.exit();
               ME.attempts=0;
               ME.id = 0;
               S = LS[ME.id];
         }
   }

   public void incorrectPin(int max)
```

```
{
	if(S.get_id()==1)
	{
			S.incorrectPin(max);
			S = LS[ME.id = 1];
	}
}

public void correctPinBelowMin()
{
	if(S.get_id()==1)
	{
		S.correctPinBelowMin();
		ME.id = 6;
		S = LS[ME.id];
	}
}

public void correctPinAboveMin()
{
	if(S.get_id()==1)
	{
		S.correctPinAboveMin();
		ME.id = 2;
		S = LS[ME.id];
	}
}

public void deposit()
{
	if(S.get_id()== 2)
	{
		S.deposit();
		ME.id=2;
		S = LS[ME.id];
	}
	if (S.get_id()==6)
	{
		S.deposit();
		ME.id=7;
		S = LS[ME.id];
	 }
}

public void withdraw()
{
    if(S.get_id()== 2)
      {
      S.withdraw();
      ME.id = 3;
		S = LS[ME.id];
    }
 }


public void belowMinBalance()
{
```

```
		if(S.get_id() == 3)
		{
			S.belowMinBalance();
			ME.id = 6;
			S = LS[ME.id];
		}

		if(S.get_id() == 7)
		{
			S.belowMinBalance();
			ME.id = 6;
			S = LS[ME.id];
		}
		if(S.get_id() == 5)
		{
			S.belowMinBalance();
			ME.id = 6;
			S = LS[ME.id];
		}
	}

	public void aboveMinBalance()
	{
		if(S.get_id() == 3)
		{
			S.aboveMinBalance();
			ME.id = 2;
			S = LS[ME.id];
		}
		if(S.get_id() == 7)
		{
			S.aboveMinBalance();
			ME.id = 2;
			S = LS[ME.id];
		}
		if(S.get_id() == 5)
		{
			S.aboveMinBalance();
			ME.id = 2;
			S = LS[ME.id];
		}
	}
	public void balance()
	{
		if(S.get_id() == 2)
		{
			S.balance();
			ME.id = 2;
			  S = LS[ME.id];
		}
		if(S.get_id() == 6)
		{
			S.balance();
			ME.id = 6;
			  S = LS[ME.id];
		}
	}
```

```
    public void correctLock()
    {
          if(S.get_id() == 2)
          {
                S.correctLock();
                ME.id = 4;
                  S = LS[ME.id];
          }
          if(S.get_id() == 6)
          {
                S.correctLock();
                ME.id = 4;
                  S = LS[ME.id];
          }
    }

    public void incorrectLock()
    {
          if(S.get_id() == 2)
          {
            S.incorrectLock();
          ME.id = 2;
            S = LS[ME.id];
          }

          if(S.get_id() == 6)
          {
            S.incorrectLock();
          ME.id = 6;
            S = LS[ME.id];
          }
    }

    public void correctUnlock()
    {
          if(S.get_id() == 4)
          {
            S.correctUnLock();
          ME.id = 5;
            S = LS[ME.id];
          }
    }

    public void incorrectUnlock()
    {
          if(S.get_id() == 4)
          {
          ME.id = 4;
            S = LS[ME.id];
          }
    }
}
```

6.4.2. State.java

```
package EFSM;
```

```
import OP.*;

public class State {


    EFSM m = null;
    OutputProcessor op = null;
    int id;
    public State(EFSM m, OutputProcessor op, int id)
    {
          this.m = m;
          this.op = op;
          this.id = id;
    }
    public int get_id()
    {
          return id;
    }

    public void set_id(int id)
    {
          this.id= id;
    }

    public void initialize()
    {

    }

    public void card()
    {};

    public void exit()
    {

    }
    public void incorrectPin(int max)
    {

    }

    public void correctPinBelowMin()
    {

    }

    public void correctPinAboveMin()
    {

    }

    public void deposit()
    {

    }
    public void withdraw()
    {
```

```
    }

    public void belowMinBalance()
    {

    }
    public void aboveMinBalance()
    {

    }
    public void balance()
    {

    }
    public void correctLock()
    {

    }
    public void incorrectLock()
    {

    }
    public void correctUnlock()
    {

    }
    public void incorrectUnlock()
    {

    }
    public void correctUnLock() {


    }
    public void incorrectUnLock() {


    }
}
```

6.4.3. Idle.java

```
package EFSM;
import OP.*;

public class Idle extends State{
   OutputProcessor op = new OutputProcessor();
   public Idle(EFSM m, OutputProcessor opp, int id)
   {
         super(m, opp, id);
   }

   public int get_id()
   {
         return 0;
```

```
    }

    public void card()
    {
          System.out.println("Inside card method of Idle");

          op.store_pin();
          op.store_balance();
    }
}
```

6.4.4. CheckPin.java

```
package EFSM;
import OP.*;
public class CheckPin extends State{
    OutputProcessor op = new OutputProcessor();
    ME v = new ME();
    public CheckPin(EFSM m, OutputProcessor op, int id)
    {
          super(m, op, id);
    }

    public void correctPinBelowMin()
    {
          op.belowMinBalMsg();
    }

    public void correctPinAboveMin()
    {
    }

    public void incorrectPin(int max)
    {
          if(ME.attempts<max){
            op.incorrect_pin_msg();
            ME.attempts++;
            op.prompt_for_pin();
        }
        else{
            op.incorrect_pin_msg();
            op.too_many_attempts_msg();
            op.eject_card();
        }
    }

    public void exit()
    {
          op.eject_card();
    }

    public int get_id()
    {
          return 1;
    }

}
```

6.4.5. Ready.java

```
package EFSM;

import OP.OutputProcessor;

public class Ready extends State{

   OutputProcessor op = new OutputProcessor();
   public Ready(EFSM m, OutputProcessor op, int id)
   {
         super(m, op, id);
   }

   public int get_id()
   {
         return 2;
   }

   public void deposit()
   {
         op.MakeDeposit();
   }
   public void withdraw()
   {
         op.MakeWithdraw();
   }
   public void balance()
    {
    op.displayBalance();
    }

   public void correctLock()
   {
         op.lock();
   }

   public void incorrectLock()
   {
         op.incorrectLockMsg();
   }

   public void exit()
   {
         op.eject_card();
   }
}
```

6.4.6. Overdrawn.java

```
package EFSM;

import OP.OutputProcessor;

public class Overdrawn extends State{

   OutputProcessor op = new OutputProcessor();
   public Overdrawn(EFSM m, OutputProcessor op, int id)
```

```
    {
          super(m, op, id);
    }

    public int get_id()
    {
          return 6;
    }

    public void balance()
     {
            op.displayBalance();
     }

    public void deposit()
    {
          op.MakeDeposit();
    }
    public void correctLock()
    {
          op.lock();
    }

    public void incorrectLock()
    {
          op.incorrectLockMsg();
    }

    public void exit()
    {
          op.eject_card();
    }
}
```

6.4.7. Lock.java

```
package EFSM;

import OP.OutputProcessor;

public class Lock extends State {
    OutputProcessor op = new OutputProcessor();

    public Lock(EFSM m, OutputProcessor op, int id)
    {
          super(m, op, id);
    }

    public int get_id()
    {
          return 4;
    }

    public void correctUnLock()
    {
    }

    public void incorrectUnLock()
```

```
    {
        op.prompt_for_pin();
    }
}
```

6.4.8. S1.java

```
package EFSM;

import OP.OutputProcessor;

public class S1 extends State{
    OutputProcessor op = new OutputProcessor();
    public S1(EFSM m, OutputProcessor op, int id)
    {
        super(m, op, id);
    }
    public int get_id()
    {
        return 3;
    }

    public void belowMinBalance()
    {
        op.belowMinBalMsg();
        op.penalty();
    }
    public void aboveMinBalance()
    {
        op.aboveMinBal();
        System.out.println("Ready");
    }
}
```

6.4.9. S2.java

```
package EFSM;

import OP.*;

public class S2 extends State{

    OutputProcessor op = new OutputProcessor();

    public S2(EFSM m, OutputProcessor op, int id)
    {
        super(m, op, id);
    }
    public int get_id()
    {
        return 7;
    }

    public void belowMinBalance()
    {
        op.belowMinBalMsg();
```

```
    }
    public void aboveMinBalance()
    {
          System.out.println("Ready");
    }
}
```

6.4.10. S3.java

```
package EFSM;

import OP.OutputProcessor;

public class S3 extends State{

    OutputProcessor op = new OutputProcessor();

    public S3(EFSM m, OutputProcessor op, int id)
    {
          super(m, op, id);
    }

    public int get_id()
    {
          return 5;
    }

    public void aboveMinBalance()
    {
          System.out.println("Ready");
    }

    public void belowMinBalance()
    {
          op.belowMinBalMsg();
    }
}
```

6.5. OP Package (Output Processor)

6.5.1. OutputProcessor.java

```
package OP;
import DataServer.*;
public class OutputProcessor {

    public void store_pin()
    {
          Global.ds.getSP().storePin();
    }

    public void store_balance()
    {
          Global.ds.getStoreBal().storeBal();
    }

    public void prompt_for_pin()
```

```
{
	Global.ds.getPrompt().prompt_for_pin();
}

public void display_menu()
{
	Global.ds.getDisplayMenu().displayMenu();
}

public void incorrect_pin_msg()
{
	Global.ds.getICPinMsg().incorrect_pin_msg();
}

public void too_many_attempts_msg()
{
	Global.ds.getAttemptMsg().tooManyAttemptMsg();
}
public void MakeDeposit()
{
	Global.ds.get_Deposit().makeDeposit();
}

public void MakeWithdraw()
{
	Global.ds.get_Withdraw().makeWithdraw();
}

public void belowMinBalMsg()
{
	Global.ds.getBelowMinBalMsg().belowMinBalMsg();
}
public void penalty()
{
	Global.ds.get_Penalty().penalty();
}

public void eject_card()
{
	Global.ds.getEjectCard().ejectCard();
}

public void incorrectLockMsg()
{
	Global.ds.getICPinMsg().incorrect_pin_msg();
}

public void lock()
{
	Global.ds.get_lock().lock();
}

public void unlock()
{}

public void displayBalance()
{
```

```
            Global.ds.get_balance().displayBal();
    }

    public void aboveMinBal()
    {
            Global.ds.getAboveMinBal().aboveMinBal();
    }
}
```

6.5.2. StorePin.java

```
package OP;

public interface StorePin {

    public void storePin();

}
```

6.5.3. StorePin1.java

```
package OP;

import DataServer.Global;

public class StorePin1 implements StorePin{

    public void storePin() {
            Global.ds.setPin1(Global.getIPPin1());
    }
}
```

6.5.4. StorePin2.java

```
package OP;

import DataServer.Global;

public class StorePin2 implements StorePin{

    public void storePin() {
            Global.ds.setPin2(Global.getIPPin2());
    }
}
```

6.5.5. StoreBal.java

```
package OP;

public interface StoreBal {

    public void storeBal();
}
```

6.5.6. StoreBal1.java

```
package OP;

import DataServer.Global;
```

```
public class StoreBal1 implements StoreBal{

   public void storeBal()
   {
         Global.ds.setBalance(Global.getIPBal1());
         System.out.println(" New Balance = " + Global.ds.getBalance1());
   }
}
```

6.5.7. StoreBal2.java

```
package OP;

import DataServer.Global;

public class StoreBal2 implements StoreBal{

   public void storeBal()
   {
         Global.ds.setBalance(Global.getIPBal2());
         System.out.println(" New Balance = " + Global.ds.getBalance2());
   }
}
```

6.5.8. ICPinMsg.java

```
package OP;

public class ICPinMsg {

   public void incorrect_pin_msg()
   {
         System.out.println("Incorrect Pin entered");
   }
}
```

6.5.9. Prompt.java

```
package OP;
public interface Prompt {
   public void prompt_for_pin();
}
```

6.5.10. Prompt1.java

```
package OP;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import DataServer.*;

public class Prompt1 implements Prompt {
```

```
   int a;
         BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
         String s= new String();

         public void prompt_for_pin()
             {
                 System.out.println("Enter Pin :");

                 try{s=in.readLine();}
                 catch(Exception e){}

                 try{Global.k = Integer.parseInt(s); }
                 catch(Exception e){}
           }
}
```

6.5.11. Prompt2.java

```
package OP;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import DataServer.Global;

public class Prompt2 implements Prompt{

   int a;
   BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
   String s= new String();

   public void prompt_for_pin()
       {
             System.out.println("Enter Pin");
          try{s=in.readLine();}
          catch(Exception e){}
          try{Global.k1 = s; }
          catch(Exception e){}
        }
}
```

6.5.12. TooManyAttemptMsg.java

```
package OP;

public class TooManyAttemptMsg {

   public void tooManyAttemptMsg()
   {
         System.out.println("Too many attempts performed");
   }
}
```

6.5.13. AboveMinBal.java

```
package OP;

public interface AboveMinBal {
   public void aboveMinBal();

}
```

6.5.14. AboveMinBal1.java

```
package OP;

import DataServer.Global;

public class AboveMinBal1 implements AboveMinBal {
   public void aboveMinBal() {
         Global.ds.setBalance(Global.getTempWD1());
   }
}
```

6.5.15. AboveMinBal2.java

```
package OP;

import DataServer.Global;

public class AboveMinBal2 implements AboveMinBal {
   public void aboveMinBal() {
         Global.ds.setBalance(Global.getTempWithdraw2());
   }
}
```

6.5.16. BelowMinBalMsg.java

```
package OP;

public class BelowMinBalMsg {

   public void belowMinBalMsg()
   {
         System.out.println("Balance is below minimum");
   }
}
```

6.5.17. Balance.java

```
package OP;

public interface Balance {

   public void displayBal();

}
```

6.5.18. Balance1.java

```
package OP;
```

```
import DataServer.Global;

public class Balance1 implements Balance{
   public void displayBal()
   {
         System.out.println("Your balance is "+Global.ds.getBalance1());
   }

}
```

6.5.19. Balance2.java

```
package OP;

import DataServer.Global;

public class Balance2 implements Balance{
   public void displayBal()
   {
         System.out.println("Your balance is "+Global.ds.getBalance2());
   }
}
```

6.5.20. Deposit.java

```
package OP;

public interface Deposit {
   public void makeDeposit();
}
```

6.5.21. Deposit1.java

```
package OP;

import DataServer.Global;

public class Deposit1 implements Deposit{

   public void makeDeposit() {
         int bal = Global.ds.getBalance1();
         int amt = Global.getTempDP1();

         bal = bal + amt;
         Global.ds.setBalance(bal);
   }
}
```

6.5.22. Deposit2.java

```
package OP;

import DataServer.Global;
```

```
public class Deposit2 implements Deposit {

   public void makeDeposit() {

               float bal = Global.ds.getBalance2();
               float amt = Global.getTempDeposit2();
               bal = bal + amt;
               Global.ds.setBalance(bal);
   }
}
```

6.5.23. Withdraw.java

```
package OP;

public interface Withdraw {

   public void makeWithdraw();
}
```

6.5.24. Withdraw1.java

```
package OP;

import DataServer.Global;

public class Withdraw1 implements Withdraw{

   public void makeWithdraw() {
         int bal = Global.ds.getBalance1(); // b
         int amt = Global.getTempWD1();

         Global.setTempWD1(bal-amt);
   }
}
```

6.5.25. Withdraw2.java

```
package OP;

import DataServer.Global;

public class Withdraw2 implements Withdraw{

   public void makeWithdraw() {

                     float bal = Global.ds.getBalance2(); // b
                     float amt = Global.getTempWithdraw2();

                     Global.setTempWithdraw2(bal-amt);
               }
         }
```

6.5.26. Penalty.java

```
package OP;

public interface Penalty
{
   public void penalty();

}
```

6.5.27. Penalty1.java

```
package OP;
import DataServer.Global;

public class Penalty1 implements Penalty
{
   public void penalty()
   {
         int penalty = 10;
         Global.ds.setBalance(Global.ds.getBalance1() - penalty);
         System.out.println("Penalty deducted from your balance");
   }
}
```

6.5.28. Penalty2.java

```
package OP;

import DataServer.Global;

public class Penalty2 implements Penalty{
   public void penalty()
   {
         float penalty = 100;
         Global.ds.setBalance(Global.ds.getBalance2() - penalty);
         System.out.println("Penalty deducted from your balance");
   }
}
```

6.5.29. Lock.java

```
package OP;

public interface Lock {
     public void lock();
     public void disp_menu();
}
```

6.5.30. Lock1.java

```
package OP;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import DataServer.Global;
```

```
public class Lock1 implements Lock {
   int a;
   BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
   String s= new String();

   public void disp_menu()
   {
         System.out.println("Enter Pin to unlock:");
         System.out.println("Enter Pin");
       try{s=in.readLine();}
          catch(Exception e){}
       try{Global.pin = Integer.parseInt(s); }
           catch(Exception e){}
   }

   public void lock()
   {
         System.out.println(" You are in locked state");
   }
}
```

6.5.31. Unlock.java

```
package OP;

public class Unlock {

}
```

6.5.32. EjectCard.java

```
package OP;

public interface EjectCard {
   public void ejectCard();
}
```

6.5.33. EjectCard1.java

```
package OP;

import DataServer.Global;

public class EjectCard1 implements EjectCard{

   public void ejectCard() {
         Global.ds.setBalance(0);
         Global.ds.setPin1(0);
   }
}
```

6.5.34. EjectCard2.java

```
package OP;

import DataServer.Global;

public class EjectCard2 implements EjectCard{
```

```
    public void ejectCard() {
          Global.ds.setBalance(0);
          Global.ds.setPin2("");
    }
}
```

6.6. Param Package

6.6.1. AbstractFactory.java

```
package Param;
import OP.*;

public interface AbstractFactory
{
    public Prompt createPrompt();

    public ICPinMsg createICPinMsg();

    public Display createDisplay();

    public TooManyAttemptMsg createTooManyAttemptMsg();

    public Deposit createDeposit();

    public Withdraw createWithdraw();

    public Penalty createPenalty();

    public BelowMinBalMsg createBelowMinBalMsg();

    public StoreBal createStoreBal();

    public EjectCard createEjectCard();

    public StorePin createStorePin();

    public Balance createBalance();

    public Lock createLock();

    public AboveMinBal createAboveMinBal();
}
```

6.6.2. FactoryATM1.java

```
package Param;

import OP.*;

public class FactoryATM1 implements AbstractFactory{
    public Prompt createPrompt(){
         return new Prompt1();
     }
    public ICPinMsg createICPinMsg(){
         return new ICPinMsg();
```

```
     }

    public Display createDisplay() {
          return new Display1();
    }

    public TooManyAttemptMsg createTooManyAttemptMsg() {
          return new TooManyAttemptMsg();
    }

    public Deposit createDeposit() {
          return new Deposit1();
    }

    public Withdraw createWithdraw() {
          return new Withdraw1();
    }

    public Penalty createPenalty() {
          return new Penalty1();
    }

     public BelowMinBalMsg createBelowMinBalMsg() {
          return new BelowMinBalMsg();
    }

    public StoreBal createStoreBal() {
          return new StoreBal1();
    }

    public EjectCard createEjectCard() {
          return new EjectCard1();
    }

    public StorePin createStorePin() {
          return new StorePin1();
    }

    public Lock createLock(){
          return new Lock1();
    }

    public Unlock createUnlock(){
          return new Unlock();
    }

    public Balance createBalance() {
          return new Balance1();
    }

    public AboveMinBal createAboveMinBal() {
          return new AboveMinBal1();
    }
}
```

6.6.3. FactoryATM2.java

```
package Param;
```

```
import OP.*;

public class FactoryATM2 implements AbstractFactory{

   public Prompt createPrompt(){
        return new Prompt2();
    }

   public ICPinMsg createICPinMsg(){
        return new ICPinMsg();
    }

   public Lock createLock(){
         return null;
   }

   public Display createDisplay() {
         return new Display2();
   }

   public TooManyAttemptMsg createTooManyAttemptMsg() {
         return new TooManyAttemptMsg();
   }

   public Deposit createDeposit() {
         return new Deposit2();
   }

   public Withdraw createWithdraw() {
         return new Withdraw2();
   }

   public Penalty createPenalty() {
         return new Penalty2();
   }

   public BelowMinBalMsg createBelowMinBalMsg() {
         return new BelowMinBalMsg();
   }

   public StoreBal createStoreBal() {
         return new StoreBal2();
   }

   public EjectCard createEjectCard() {
         return new EjectCard2();
   }

   public StorePin createStorePin() {
         return new StorePin2();
   }

   public Balance createBalance() {
         return new Balance2();
   }
```

```
    public AboveMinBal createAboveMinBal() {
          return new AboveMinBal2();
    }
}
```

6.6.4. IConstants.java

```
package Param;

public interface IConstants {

    public final static int pn = 0;
    public final static int b = 0;
    public final static float bal = 0;
    public final static String pin = "";


    /**
     * Idle State ID.
     */
    public final static int IDLE_STATE_ID = 0;

    /**
     * Check Pin State ID.
     */
    public final static int CHECK_PIN_STATE_ID = 1;


    /**
     * Ready State ID.
     */
    public final static int READY_STATE_ID = 2;

    /**
     * Interim State S1.
     */
    public final static int S1_STATE_ID = 3;

    /**
     * Lock state id.
     */
    public final static int LOCK_STATE_ID = 4;

    /**
     * S3 state id.
     */
    public final static int S3_STATE_ID = 5;

    /**
     * Overdrawn state id.
     */
    public final static int OVERDRAWN_STATE_ID = 6;

    /**
     * S2 state id.
     */
    public final static int S2_STATE_ID = 7;
}
```