Seungho Han A20360115

1. MDA-EFSM model for the ACCOUNT components

MDA- EFSM Events:

Open()
Login()
IncorrectLogin()
IncorrectPin(int max)
CorrectPinBelowMin()
CorrectPinAboveMin()
Deposit()
BelowMinBalance()
AboveMinBalance()
Logout()
Balance()
Withdraw()
WithdrawBelowMinBalance()
NoFunds()
Lock()
IncorrectUnlock()
Suspend()
Active()
Close()

MDA-EFSM Actions:

A1: StoreData() // stores pin from temporary data store to pin in data store
A2: IncorrectIdMsg() // displays incorrect ID message
A3: IncorrectPinMsg() // displays incorrect pin message
A4: TooManyAttemptsMsg() // display too many attempts message
A5: DisplayMenu() // display a menu with a list of transactions
A6: MakeDeposit() // makes deposit
A7: DisplayBalance() // displays the current value of the balance
A8: PromptForPin() // prompts to enter pin
A9: MakeWithdraw() // makes withdraw
A10: Penalty() // applies penalty (decreases balance by the amount of penalty)
A11: IncorrectLock Msg() // displays incorrect lock msg
A12: IncorrectUnlock Msg() // displays incorrect unlock msg
A13: NoFundsMsg() // Displays no sufficient funds msg

Pseudo-code of all operations of Input Processors

ACCOUNT-1

* m is a pointer to the MDA-EFSM object
* ds is a pointer to the DataStore object

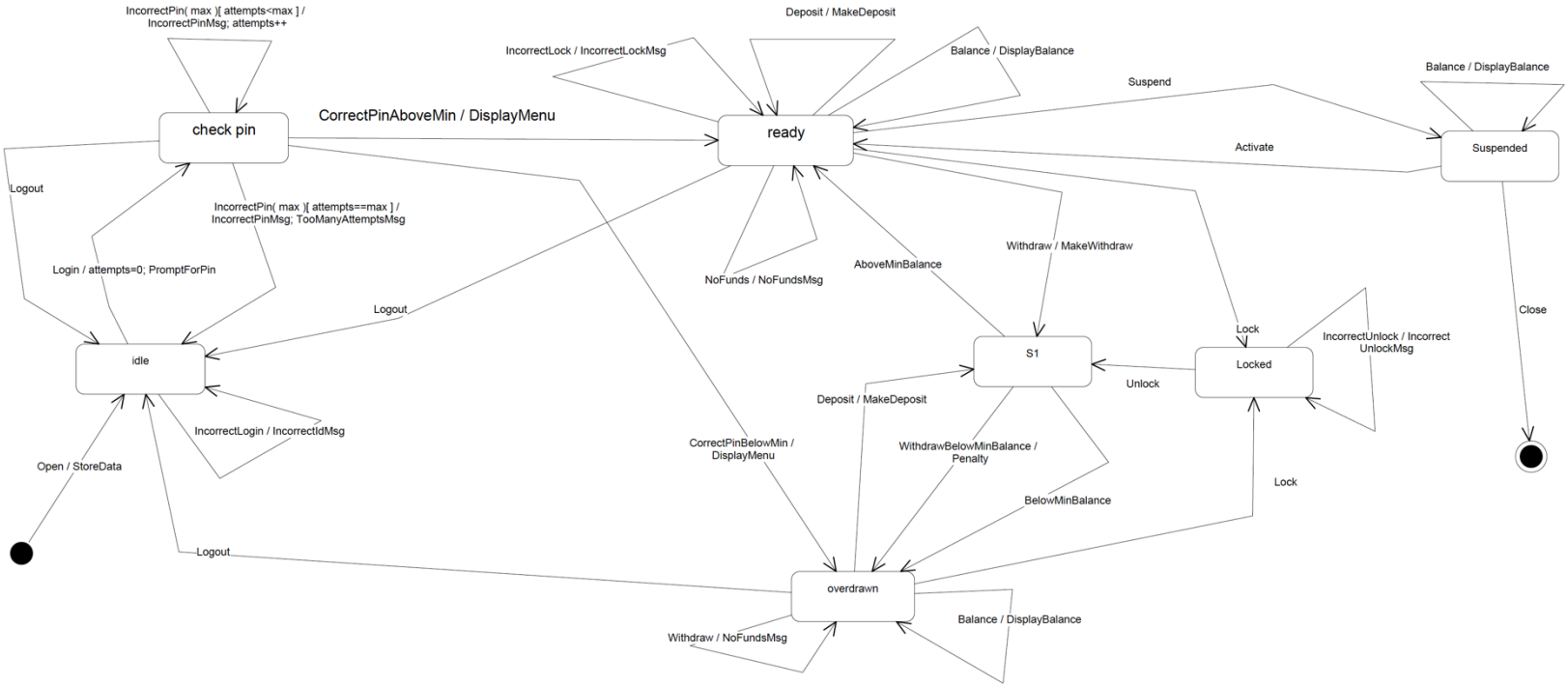
```
open (string p, string y, float a) {
        ds->temp_p=p;
        ds->temp_y=y;
        ds->temp_a=a;
        m->Open();
}

pin (string x) {
        if (x==ds->pin) {
                if (d->balance > 500)
                        m->CorrectPinAboveMin ();
                 else m->CorrectPinBelowMin();
        }
        else m->IncorrectPin(3)
}

deposit (float d) {
        ds->temp_d=d;
        m->Deposit();
        if (ds->balance>500)
                m->AboveMinBalance();
        else m->BelowMinBalance();
}

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

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

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

```
}

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

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

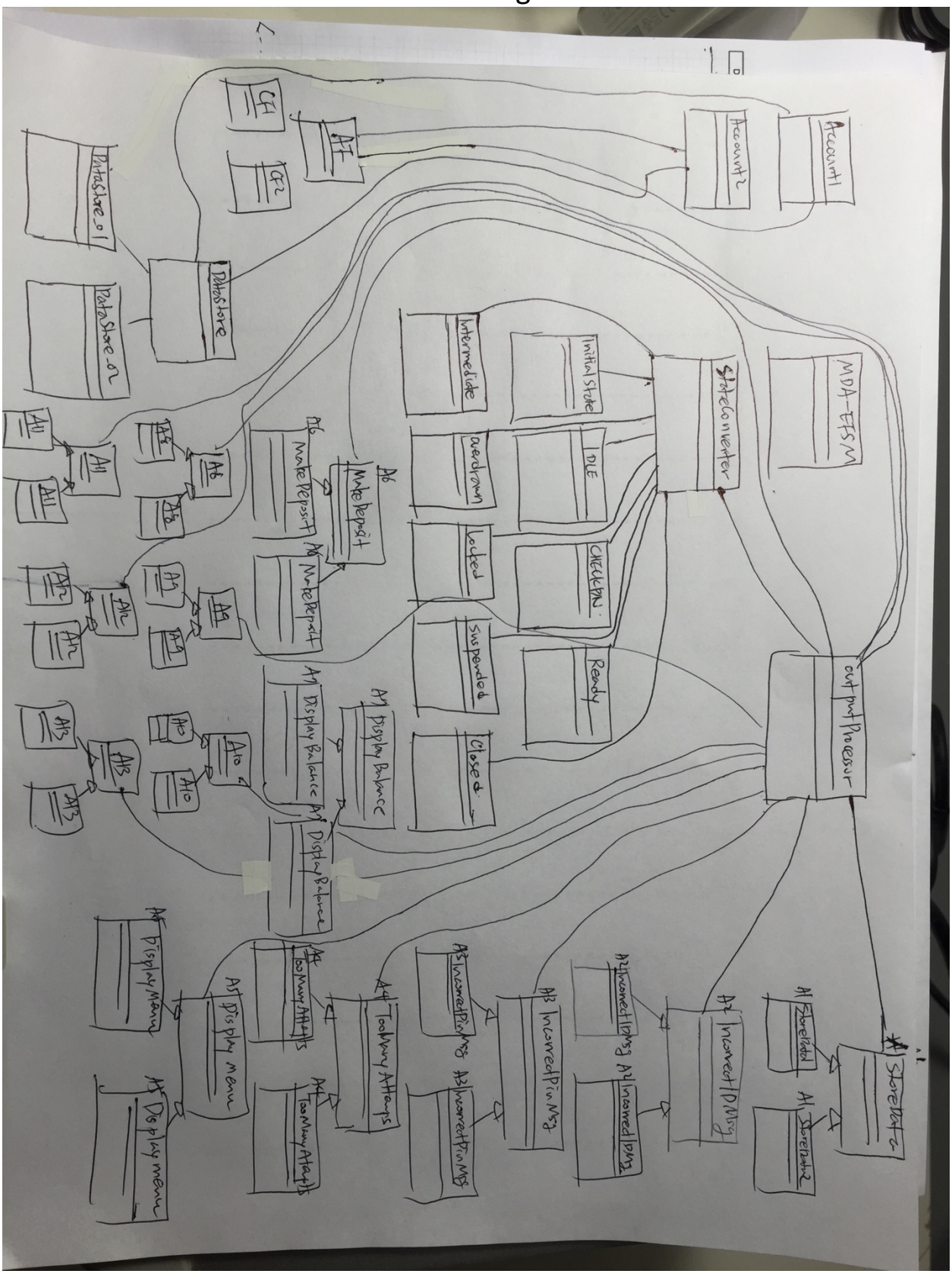
ACCOUNT-2

```
OPEN (int p, int y, int a) {
        ds->temp_p=p;
        ds->temp_y=y;
        ds->temp_a=a;
        m->Open();
}

PIN (int x) {
        if (x==ds->pin)
                m->CorrectPinAboveMin ();
        else m->IncorrectPin(2)
}

DEPOSIT (int d) {
        ds->temp_d=d;
        m->Deposit();
}

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

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

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

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

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

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

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


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

2. Class Diagram



3. Description of the class

class AbstractFactory

offers abstract data Store and declarations of output processors which should be extended in concrete factories and concrete data stores.

Attributes:

ds // Abstract Datastore

sda // Abstract StoreData
ima // Abstract IncorrectIdMsg
ipm // Abstract IncorrectPinMsg
tma // Abstract TooManyAttempts
dma // Abstract DisplayMenu
mda // Abstract MakeDeposit
dba // Abstract DisplayBalance
pfa // Abstract PromptForPin
mwa // Abstract MakeWithdraw
pna // Abstract Panelty
ila // Abstract IncorrectLockMsg
iua // Abstract IncorrectUnlockMsg
nfa // Abstract NoFundsMsg

Operations:
returns above instances to the OutputProcessor or the other classes which needs data store.

Class ConcreteFactory_01 & Class ConcreteFactory_02

Implements declarations from the AbastractFactory in the Constructor.

Operations:
Constructor

Class DataStore

This DataStore class offers abstract methods which should be implemented in Datastore_01 and DataStore_02.

Operations:

```
public Object getP(){ return null; }
public void setP(Object p) { }
public Object getTemp_p() {   return null; }
public void setTemp_p(Object temp_p) { }
public Object getY() { return null; }
public void setY(Object y) { }
public Object getTemp_y() { return null; }
public void setTemp_y(Object temp_y) { }
public Object getA() { return null; }
public void setA(Object a) { }
public Object getTemp_a() { return null;}
public void setTemp_a(Object temp_a) { }
public Object getD() { return null;}
public void setD(Object d) { }
public Object getTemp_d() { return null; }
public void setTemp_d(Object temp_d) { }
public Object getPin() { return null; }
public void setPin(Object pin) {          }
public Object getTemp_pin() { return null; }
public void setTemp_pin(Object temp_pin) { }
public Object getW() { return null; }
public void setW(Object w) { }
public Object getTemp_w() { return null;}
public void setTemp_w(Object temp_w) { }
public Object getBalance() { return null;        }
public void setBalance(Object balance) { }
public Object getUid() { return null; }
public void setUid(Object uid) { }
public Object getX() { return null;}
public void setX(String x) { }
```

Class DataStore_01 & Class DataStore_02

DataStore_01 and DataStore_02 inherits methods from super class(DataStore) to implement appropriate methods of types.

DataStore_01 sets or gets member variables with String and float type. And DataStore_02 sets or gets member variables with integer type.

Class Driver

```
ds                  // data store
op                  // OutputProcessor

account1            // instance for account1
account2            // instance for account1

factory             // AbstractFactory
converter           // StateConverter

p1, y1, a1
x1, d1, w1          // local variables for account 1 input

p2, y2, a1          // local variables for account 2 input
x2, d2, w2


If (ACCOUNT-1) {
        account1 = new Account1 (factory, converter);
        display whole menu of operations

        switch ( c ) {
        case 0:                             // open
                account1.open(p1, y1, a1)
        case 1:                             // login
                account1.login(y1)
        case 2:                             // pin
                account1.pin(x1)
        case 3:                             // deposit
                account1.deposit(d1)
        case 4:                             // withdraw
                account1.withdraw(w1)
        case 5:                             // balance
                account1.balance()
        case 6:                             // logout
                account1.logout()
        case 7:                             // lock
                account1.lock()
        case 8:                             // unlock
                account1.unlock()
}

ELSE If (ACCOUNT-2) {
```

```
        account2 = new Account2 (factory, converter);
        display whole menu of operations

        switch ( c ) {
        case 0:                                 // open
                account2.OPEN(p1, y1, a1)
        case 1:                                 // login
                account2.LOGIN(y1)
        case 2:                                 // pin
                account2.PIN(x1)
        case 3:                                 // deposit
                account2.DEPOSIT(d1)
        case 4:                                 // withdraw
                account2.WITHDRAW(w1)
        case 5:                                 // balance
                account2.BALANCE()
        case 6:                                 // logout
                account2.LOGOUT()
        case 7:                                 // suspend
                account2.suspend()
        case 8:                                 // active from suspend
                account1.active()
        case 9:                                 // close account2
                account2.close()
}
```

Class Account1

Attributes:
ds // datastore
m // stateconverter

operations:

```
Account1(AbstractFactory af, StateConverter sc) {
        ds = af.getDS();
        m = sc;
}

open(p, y, a) {
        ds.setTemp
        ds.setTemp_p(p);            // set temporary data with account 1 input
        ds.setTemp_y(y);
        ds.setTemp_a(a);
        m.Open();                   // m is a pointer to the state converter
}


void pin(String x) {
        if (x == ds.getP()) {
                if ( ds.getA() > 500) {
                        m.CorrectPinAboveMin();
                } else {
                        m.CorrectPinBelowMin();
                }
        } else {
                m.IncorrectPin(3);          // set number of maximum attempts into 3
        }
}

void deposit(float d) {
        ds.setTemp_d(d);                    // set temporary data with account 1 input
        m.Deposit();
        if (ds.getA() > 500)    m.AboveMinBalance();
        else                    m.BelowMinBalance();
}

void withdraw(float w) {
        ds.setTemp_w(w);                    // set temporary data with account 1 input
        m.Withdraw();
```

```
        if (ds.getA() > 500)) {   m.AboveMinBalance();}
        else { m.WithdrawBelowMinBalance(); }
}

void balance() { m.Balance();  }          // request to print balance

void login(String y) {              // login with pin
        if (y==ds.getY()) {
                m.Login();
        } else {
                m.IncorrectLogin();
        }
}

void logout() {                     // make account1 logout
        m.Logout();
}

void lock(String x) {               // make account1 locked
        if (x == ds.getP()) {
                m.Lock();
        } else
                m.IncorrectLock();
}

void unlock(String x) {             // make account 1 unlocked when the pin is correct
        if (x == ds.getP() ){
                m.Unlock();
                if ((float) ds.getA() > 500) {
                        m.AboveMinBalance();
                } else {
                        m.BelowMinBalance();
                }
        } else {
                m.IncorrectUnlock();
        }
}
```

Class Account2

Attributes:
ds // datastore
m // stateconverter

operations:

```
Account1(AbstractFactory af, StateConverter sc) {
        ds = af.getDS();
        m = sc;
}

OPEN(int p, y, a) {
        ds.setTemp
        ds.setTemp_p(p);            // set temporary data with account 1 input
        ds.setTemp_y(y);
        ds.setTemp_a(a);
        m.Open();                   // m is a pointer to the state converter
}


void PIN(int x) {
        if (x == ds.getP()) {
                m.CorrectPinAboveMin();
        } else {
                m.IncorrectPin(2);          // set number of maximum attempts into 2
        }
}

void DEPOSIT(int d) {
        ds.setTemp_d(d);                    // set temporary data with account 1 input
        m.Deposit();
}

void WITHDRAW(int w) {
        ds.setTemp_w(w);                    // set temporary data with account 1 input

        if (ds.getA() > 0)) {
                m.Withdraw();
                m.AboveMinBalcen();
        } else {
                m.NoFunds();
        }
```

```
        }

        void BALANCE() { m.Balance();          }          // request to print balance

        void LOGIN(int y) {                 // login with pin
                if (y==ds.getY()) {
                        m.Login();
                } else {
                        m.IncorrectLogin();
                }
        }

        void LOGOUT() {                             // make account1 logout
                m.Logout();
        }

        void suspend() {
                m.Suspend();
        }

        void activate() {
                m.Activate();
        }

        void close () {
                 m.Close();
        }
}
```

Interface efsm-states

This interface offers the events of mda-efsm which should be implemented in all of actual states. Each of states have OutputProcessor instance to get functions from abstractFactory

Operations:
void Open();
void Login();
void Pin();
void IncorrectLogin();
void IncorrectPin(int max);
void CorrectPinBelowMin();
void CorrectPinAboveMin();
void Deposit();
void BelowMinBalance();
void AboveMinBalance();
void Logout();
void Balance();
void Withdraw();
void WithdrawBelowMinBalance();
void NoFunds();
void Lock();
void IncorrectLock();
void Unlock();
void IncorrectUnlock();
void Suspend();
void Active();
void Close();

class S0_InitialState implements

performs functions before the Idle state
: Open

Attributes:
OutputProcessor Action;

Operations:

```
public S0_InitialState(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {
        Action.StoreData();
}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() { }

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
```

```
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {}

@Override
public void Suspend() {}

@Override
public void Active() {}

@Override
public void Close() {}
```

class S1_Idle implements

Idle state performs the function related Login

Attributes:
OutputProcessor Action;

Operations:

```
public S1_Idle(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {

}

@Override
public void Login() {
        System.out.println("S1->Login");
        Action.PromptForPin();
}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {
        System.out.println("IncorrectLogin");
        Action.IncorrectIdMsg();
}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}
```

```
    @Override
    public void BelowMinBalance() {}

    @Override
    public void AboveMinBalance() {}

    @Override
    public void Logout() {}

    @Override
    public void Balance() {}

    @Override
    public void Withdraw() {}

    @Override
    public void WithdrawBelowMinBalance() {}

    @Override
    public void NoFunds() {}

    @Override
    public void Lock() {}

    @Override
    public void IncorrectLock() {}

    @Override
    public void Unlock() {}

    @Override
    public void IncorrectUnlock() {}

    @Override
    public void Suspend() {}

    @Override
    public void Active() {}

    @Override
    public void Close() {}
}
```

class S2_CheckPin

Check Pin state performs functions related the permission

Attributes:
OutputProcessor Action;

static int counter; // this counter is used in the IncorrectPin
                     // to print TooManyAttemptsMsg

operations:

```
public S2_CheckPin(OutputProcessor op) {
        Action = op;
        counter = 0;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {
        counter++;
        Action.IncorrectPinMsg();
        if (counter == max) {
                Action.TooManyAttemptsMsg();
        }
}

@Override
public void CorrectPinBelowMin() { Action.DisplayMenu(); }

@Override
public void CorrectPinAboveMin() { Action.DisplayMenu();  }

@Override
```

```
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {}

@Override
public void Suspend() {}

@Override
public void Active() {}

@Override
public void Close() {}
```

class S3_Ready

Attributes:
OutputProcessor Action;

Operations:

```
public S3_Ready(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {
        Action.MakeDeposit();
}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}
```

```
@Override
public void Balance() {
	Action.DispayBalance();
}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub
	Action.MakeWithdraw();
}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {
	Action.NoFundsMsg();
}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {
	Action.IncorrectLockMsg();
}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {}

@Override
public void Suspend() {}

@Override
public void Active() {}

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

class S4_IntermediateState

Intermediate State means S1 state which is located in the sample solution of MDA-EFSM

Attributes:
OutputProcessor Action;

Operations:

```
public S4_IntermediateState(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}
```

```
	@Override
	public void Logout() {}

	@Override
	public void Balance() {}

	@Override
	public void Withdraw() {}

	@Override
	public void WithdrawBelowMinBalance() {
		Action.Penalty();
	}

	@Override
	public void NoFunds() {}

	@Override
	public void Lock() {}

	@Override
	public void IncorrectLock() {}

	@Override
	public void Unlock() {}

	@Override
	public void IncorrectUnlock() {}

	@Override
	public void Suspend() {}

	@Override
	public void Active() {}

	@Override
	public void Close() {}
}
```

class S5_Overdrawn

Attributes:
OutputProcessor Action;

Operations:

```
public S5_Overdrawn(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {
        Action.MakeDeposit();}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}
```

```
	@Override
	public void Balance() {
		Action.DispayBalance();
	}

	@Override
	public void Withdraw() {
		Action.NoFundsMsg();

	}

	@Override
	public void WithdrawBelowMinBalance() {
		Action.Penalty();

	}

	@Override
	public void NoFunds() {}

	@Override
	public void Lock() {}

	@Override
	public void IncorrectLock() {}

	@Override
	public void Unlock() {}

	@Override
	public void IncorrectUnlock() {}

	@Override
	public void Suspend() {}

	@Override
	public void Active() {}

	@Override
	public void Close() {}

}
```

class S6_Locked

Only Account 1 can be included in this state

Attributes:
OutputProcessor Action;

Operations:

```
public S6_Locked(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}
```

```
	@Override
	public void Logout() {}

	@Override
	public void Balance() {}

	@Override
	public void Withdraw() {}

	@Override
	public void WithdrawBelowMinBalance() {}

	@Override
	public void NoFunds() {}

	@Override
	public void Lock() {}

	@Override
	public void IncorrectLock() {}

	@Override
	public void Unlock() {}

	@Override
	public void IncorrectUnlock() {
		Action.IncorrectUnlockMsg();
	}

	@Override
	public void Suspend() {}

	@Override
	public void Active() {}

	@Override
	public void Close() {}

}
```

class S7_Suspended

Only Account 2 can be included in this state

Attributes:
OutputProcessor Action;

Operations:

```
public S7_Suspended(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}
```

```
	@Override
	public void Balance() {
		System.out.println("S7->Balance");
		Action.DispayBalance();
	}

	@Override
	public void Withdraw() {}

	@Override
	public void WithdrawBelowMinBalance() {}

	@Override
	public void NoFunds() {}

	@Override
	public void Lock() {}

	@Override
	public void IncorrectLock() {}

	@Override
	public void Unlock() {}

	@Override
	public void IncorrectUnlock() {}

	@Override
	public void Suspend() {}

	@Override
	public void Active() {}

	@Override
	public void Close() {}

}
```

class S8_Closed

Only Account 2 can be included in this state

Attributes:
OutputProcessor Action;

Operations:

```
public S8_Closed(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
```

```
    public void Logout() {}

    @Override
    public void Balance() {}

    @Override
    public void Withdraw() {}

    @Override
    public void WithdrawBelowMinBalance() {}

    @Override
    public void NoFunds() {}

    @Override
    public void Lock() {}

    @Override
    public void IncorrectLock() {}

    @Override
    public void Unlock() {}

    @Override
    public void IncorrectUnlock() {}

    @Override
    public void Suspend() {}

    @Override
    public void Active() {}

    @Override
    public void Close() {}

}
```

```
class StateConverter {

    IS
    IL
    CP
    RD
    IT
    OD
    LK
    SP
    CD

    CurrentState                    // stores current store

    counter                         // counts number of incorrect pin attempts

    public StateConverter(OutputProcessor op) {     // Constructor

        IS      = new S0_InitialState(op);
        IL      = new S1_Idle(op);
        CP      = new S2_CheckPin(op);
        RD      = new S3_Ready(op);
        IT      = new S4_IntermediateState(op);
        OD      = new S5_Overdrawn(op);
        LK      = new S6_Locked(op);
        SP      = new S7_Suspended(op);
        CD      = new S8_Closed(op);

        CurrentState = IS ;

        counter = 0;

    }


    public void Open() {
        if (CurrentState == IS) {
            CurrentState.Open();
            CurrentState = IL;
        }
    }

    public void Login() {
        if (CurrentState == IL) {
```

```
                CurrentState.Login();
                CurrentState = CP;
        }
}

public void Pin() {
        if (CurrentState == CP) {
                CurrentState.Pin();
                CurrentState = RD;
        }
}

public void IncorrectLogin() {
        if (CurrentState == IL) {
                CurrentState.IncorrectLogin();
                CurrentState = IL;
        }
}

public void IncorrectPin(int max) {

        counter++;

        CurrentState.IncorrectPin(max);

        if(counter == max) {                    // if the number of invalid pin attempts is
                CurrentState = IL;              // maximam, then make state Idle

        }
}


public void CorrectPinAboveMin() {
        if (CurrentState == CP) {
                CurrentState.CorrectPinAboveMin();
                CurrentState = RD;
        }
}

public void CorrectPinBelowMin() {
        if (CurrentState == CP) {
                CurrentState.CorrectPinBelowMin();
                CurrentState = OD;
        }
```

```
}

public void Deposit() {                         // executes both in the Ready or Overdrawn
        if (CurrentState == RD) {
                CurrentState.Deposit();
                CurrentState = RD;
        } else if (CurrentState == OD) {
                CurrentState.Deposit();
                CurrentState = IT;
        }
}

public void AboveMinBalance() {
        if (CurrentState == IT) {
                CurrentState = RD;
        }
}

public void BelowMinBalance() {
        if (CurrentState == IT) {
                CurrentState = OD;
        }
}

public void Logout() {
        if (CurrentState == RD) {
                CurrentState = IL;
        }
}

public void Balance() {                 // executes in Ready, Overdrawn, Suspended
        if (CurrentState == RD) {
                CurrentState.Balance();
        } else if (CurrentState == OD) {
                CurrentState.Balance();
        } else if (CurrentState == SP) {
                CurrentState.Balance();
        }
}

public void Withdraw() {
        if (CurrentState == RD) {
                CurrentState.Withdraw();
                CurrentState = IT;
```

```
        } else if (CurrentState == OD) {
                CurrentState.Withdraw();
                CurrentState = OD;
        }
}

public void WithdrawBelowMinBalance() {
        if (CurrentState == IT) {
                CurrentState.WithdrawBelowMinBalance();
                CurrentState = OD;
        }
}

public void NoFunds() {
        if (CurrentState == RD) {
                CurrentState = RD;
        }
}

public void Lock() {
        if (CurrentState == RD) {
                CurrentState = LK;
        } else if (CurrentState == OD) {
                CurrentState = LK;
        }
}

public void IncorrectLock() {
        if (CurrentState == RD) {
                CurrentState = RD;
        }
}

public void Unlock() {
        if (CurrentState == LK) {
                CurrentState = IT;
        }
}

public void IncorrectUnlock() {
        if (CurrentState == LK) {
                CurrentState.IncorrectUnlock();
                CurrentState = LK;
        }
```

```
	}

	public void Suspend() {
		if (CurrentState == RD) {
			CurrentState = SP;
		}
	}

	public void Activate() {
		if (CurrentState == SP) {
			CurrentState = RD;
		}
	}

	public void Close() {
		if (CurrentState == SP) {
			CurrentState = CD;
		}
	}
}
```

abstract class A01_StoreData_Abs {

Attributes:
DataStore ds;

Operations:

```
public A01_StoreData_Abs(DataStore dataStore) {
        ds = dataStore;
}

        public void StoreData() {}       // abstract method
}
```

class A01_StoreData_AC1 extends A01_StoreData_Abs {

Operations

```
        public A01_StoreData_AC1(DataStore ds1) {
                super(ds1);
        }

        @Override
        public void StoreData() {
                ds.setP(ds.getTemp_p());
                ds.setY(ds.getTemp_y());
                ds.setA(ds.getTemp_a());
        }
}
```

class A01_StoreData_AC2 extends A01_StoreData_Abs {

Operations

```
        public A01_StoreData_AC1(DataStore ds2) {
                super(ds2);
        }
        @Override
        public void StoreData() {
                ds.setP(ds.getTemp_p());
                ds.setY(ds.getTemp_y());
                ds.setA(ds.getTemp_a());
        }
}
```

abstract class A02_IncorrectIdMsg_Abs {

Attributes:
DataStore ds;

Operations:

```
public A02_IncorrectIdMsg_Abs(DataStore dataStore) {
    ds = dataStore;
}

public void IncorrectIdMsg() {
    System.out.println("Incorrect ID");
}
```

}

class A02_IncorrectIdMsg_AC1 extends A02_IncorrectIdMsg_Abs {

Operations:

```
public A02_IncorrectIdMsg_AC1(DataStore ds1) {
    super(ds1);
}

public void IncorrectIdMsg() {
    System.out.println("IncorrectIdMsg");
    if(ds.getY() != ds.getTemp_y()) {
        System.out.println("Incorrect ID");
    }
}
```

}

class A02_IncorrectIdMsg_AC2 extends A02_IncorrectIdMsg_Abs {

Operations:

```
public A02_IncorrectIdMsg_AC1(DataStore ds1) {
    super(ds1);
}

public void IncorrectIdMsg() {
    System.out.println("IncorrectIdMsg");
    if(ds.getY() != ds.getTemp_y()) {
        System.out.println("Incorrect ID");
    }
}
```

}

```
abstract class A03_IncorrectPinMsg_Abs {

Operations:
        public void IncorrectPinMsg() {
                System.out.println("Incorrect Pin");
        }

class A03_IncorrectPinMsg_AC1 extends A03_IncorrectPinMsg_Abs {

operations:
        public void IncorrectPinMsg() {
                System.out.println("Incorrect Pin");
        }
}

class A03_IncorrectPinMsg_AC2 extends A03_IncorrectPinMsg_Abs {

operations:
        public void IncorrectPinMsg() {
                System.out.println("Incorrect Pin");
        }
}

abstract class A04_TooManyAttemptsMsg_Abs {
operations:
        public void TooManyAttemptsMsg() {
                System.out.println("Too Many Attempts");
        }
}

class A04_TooManyAttemptsMsg_AC1 extends A04_TooManyAttemptsMsg_Abs {
operations:
        public void TooManyAttemptsMsg() {
                System.out.println("Too Many Attempts");
        }
}

class A04_TooManyAttemptsMsg_AC2 extends A04_TooManyAttemptsMsg_Abs {
operations:
        public void TooManyAttemptsMsg() {
                System.out.println("Too Many Attempts");
        }
}
```

abstract class A05_DisplayMenu_Abs {

```
operations:
        public void DisplayMenu() {
                // abstract method
        }
}
```

class A05_DisplayMenu_AC1 extends A05_DisplayMenu_Abs {

```
operations:
        public void DisplayMenu() {
                System.out.println("TRANSACTION MENU");
                System.out.println("deposit");
                System.out.println("withdraw");
                System.out.println("logout");
                System.out.println("lock");
                System.out.println("unlock");
        }
}
```

class A05_DisplayMenu_AC2 extends A05_DisplayMenu_Abs {

```
operations:
        public void DisplayMenu() {
                System.out.println("TRANSACTION MENU");
                System.out.println("DEPOSIT");
                System.out.println("WITHDRAW");
                System.out.println("BALANCE");
                System.out.println("LOGOUT");
                System.out.println("suspend");
                System.out.println("active");
                System.out.println("close");
        }
}
```

abstract class A06_MakeDeposit_Abs {

Attributes:
DataStore ds;

Operations:

```
        public A06_MakeDeposit_Abs(DataStore dataStore) {
                ds = dataStore;
        }
```

```
        public void MakeDeposit() {
                ds.setA((float)ds.getTemp_d() + (float)ds.getA());
        }
}
```

class A06_MakeDeposit_AC1 extends A06_MakeDeposit_Abs {

```
Operations:
        public A06_MakeDeposit_AC1(DataStore ds1) {
                super(ds1);
        }

        public void MakeDeposit() {
                ds.setA((float)ds.getTemp_d() + (float)ds.getA());
        }
}
```

class A06_MakeDeposit_AC2 extends A06_MakeDeposit_Abs {

```
Operations:
        public A06_MakeDeposit_AC2(DataStore ds2) {
                super(ds2);
        }

        public void MakeDeposit() {
                ds.setA((int)ds.getTemp_d() + (int)ds.getA());
        }
}
```

abstract class A07_DisplayBalance_Abs {

```
Attributes:
DataStore ds;

Operations:
        public A07_DisplayBalance_Abs(DataStore dataStore) {
                ds = dataStore;
        }

        public void DisplayBalance() {

        }
}
```

class A07_DisplayBalance_AC1 extends A07_DisplayBalance_Abs {

```
Operations:
        public A07_DisplayBalance_AC1(DataStore ds1) {
                super(ds1);
        }

        public void DisplayBalance() {
                System.out.println("Current balance is " + ds.getA());
        }
}
```

class A07_DisplayBalance_AC2 extends A07_DisplayBalance_Abs {

```
Operations:
        public A07_DisplayBalance_AC1(DataStore ds1) {
                super(ds1);
        }

        public void DisplayBalance() {
                System.out.println("Current balance is " + ds.getA());
        }
}
```

abstract class A08_PromptForPin_Abs {

```
operations:
        public void PromptForPin() {
                System.out.println("Please enter your pin number");
        }
}
```

class A08_PromptForPin_AC1 extends A08_PromptForPin_Abs {

```
operations:
        public void PromptForPin() {
                System.out.println("Please enter your pin number");
        }
}
```

class A08_PromptForPin_AC2 extends A08_PromptForPin_Abs {

```
operations:
        public void PromptForPin() {
                System.out.println("Please enter your pin number");
        }
}
```

```
abstract class A09_MakeWithdraw_Abs {
Attributes:
DataStore ds;

Operations:
        public A09_MakeWithdraw_Abs(DataStore dataStore) {
                ds = dataStore;
        }
        public void MakeWithdraw() {

                ds.setA((float)ds.getA() - (float)ds.getTemp_w());
        }
}

class A09_MakeWithdraw_AC1 extends A09_MakeWithdraw_Abs {
Operations:
        public A09_MakeWithdraw_AC1(DataStore ds1) {
                super(ds1);
        }

        public void MakeWithdraw() {

                ds.setA((float)ds.getA()- (float)ds.getTemp_w());
        }
}

class A09_MakeWithdraw_AC2 extends A09_MakeWithdraw_Abs {
Operatinos:
        public A09_MakeWithdraw_AC2(DataStore ds2) {
                super(ds2);
        }

        public void MakeWithdraw() {
                ds.setA((int)ds.getA() - (int)ds.getTemp_w());
        }
}

abstract class A10_Panelty_Abs {
Attributes:
DataStore ds;

Operations:
        public A10_Panelty_Abs(DataStore dataStore) {
```

```
            ds = dataStore;
        }

        public void Penalty() {
            // Abstract Method
        }
}
```

class A10_Panelty_AC1 extends A10_Panelty_Abs {

```
operations:
        public A10_Panelty_AC1(DataStore ds1) {
            super(ds1);
        }

        public void Penalty() {
            ds.setA((float )ds.getA()-20);
        }

}
```

class A10_Panelty_AC2 extends A10_Panelty_Abs {

```
Operations:
        public A10_Panelty_AC2(DataStore ds2) {
            super(ds2);
        }

        public void Penalty() {
            ds.setA((int)ds.getA()-20);
        }
}
```

abstract class A11_IncorrectLockMsg_Abs {

```
operations:
        public void IncorrectIdMsg() {
            System.out.println("Incorrect Lock");
        }
}
```

class A11_IncorrectLockMsg_AC1 extends A11_IncorrectLockMsg_Abs{

```
operations:
        public void IncorrectLockMsg() {
            System.out.println("Incorret Lock");
        }
```

```
}

class A11_IncorrectLockMsg_AC2 extends A11_IncorrectLockMsg_Abs{
operations:
	public void IncorrectLockMsg() {
		System.out.println("Incorret Lock");
	}
}

abstract class A12_IncorrectUnlockMsg_Abs {
operations:
	public void IncorrectUnlockMsg() {
		System.out.println("Incorrect Unlock");
	}
}

class A12_IncorrectUnlockMsg_AC1 extends A12_IncorrectUnlockMsg_Abs {
operations:
	public void IncorrectUnlockMsg() {
		System.out.println("Incorrect Unlock");
	}

}

class A12_IncorrectUnlockMsg_AC2 extends A12_IncorrectUnlockMsg_Abs {
operations:
	public void IncorrectUnlockMsg() {
		System.out.println("Incorrect Unlock");
	}

}

abstract class A13_NoFundsMsg_Abs {
Operations:
	public void NoFundsMsg() {
		System.out.println("No Funds");
	}
}

class A13_NoFundsMsg_AC1 extends A13_NoFundsMsg_Abs {
operations
	public void NoFundsMsg() {
		System.out.println("No Funds");
```

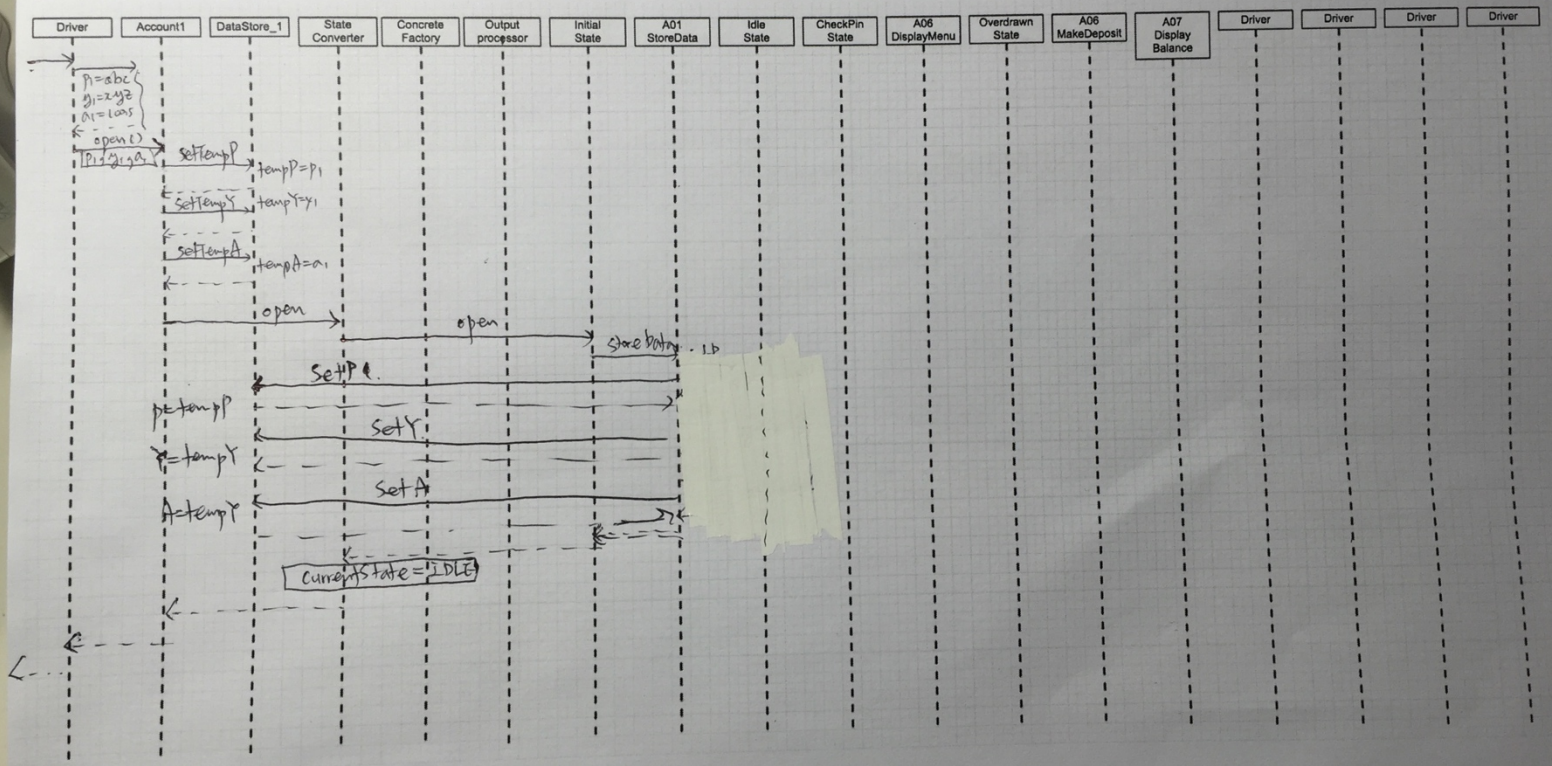
```
        }
}
```

class A13_NoFundsMsg_AC2 extends A13_NoFundsMsg_Abs {

```
operations
        public void NoFundsMsg() {
                System.out.println("No Funds");
        }
}
```

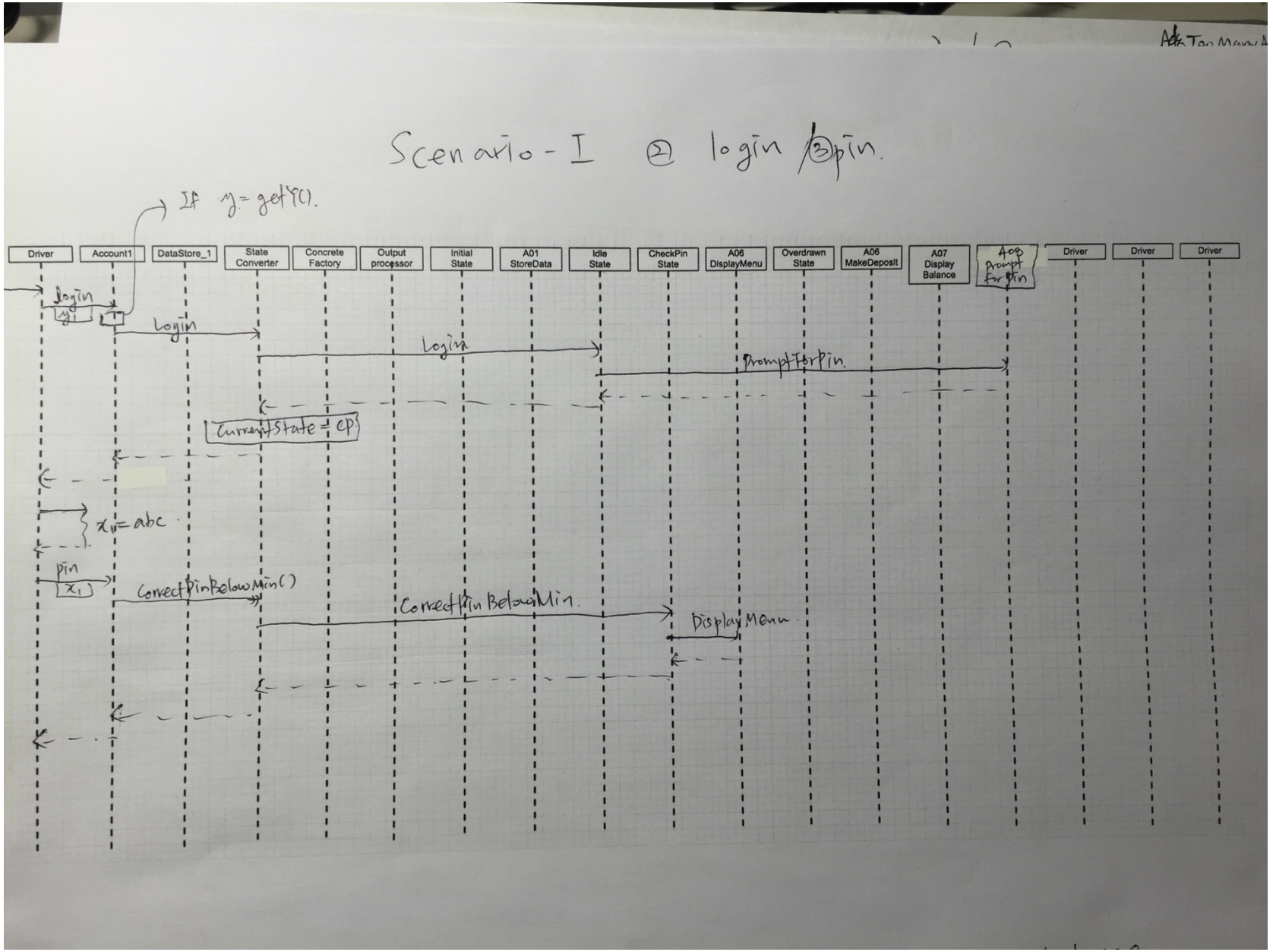
4. Sequence Diagram

Scenario-1 ① open(abc, xyz, 100.5)



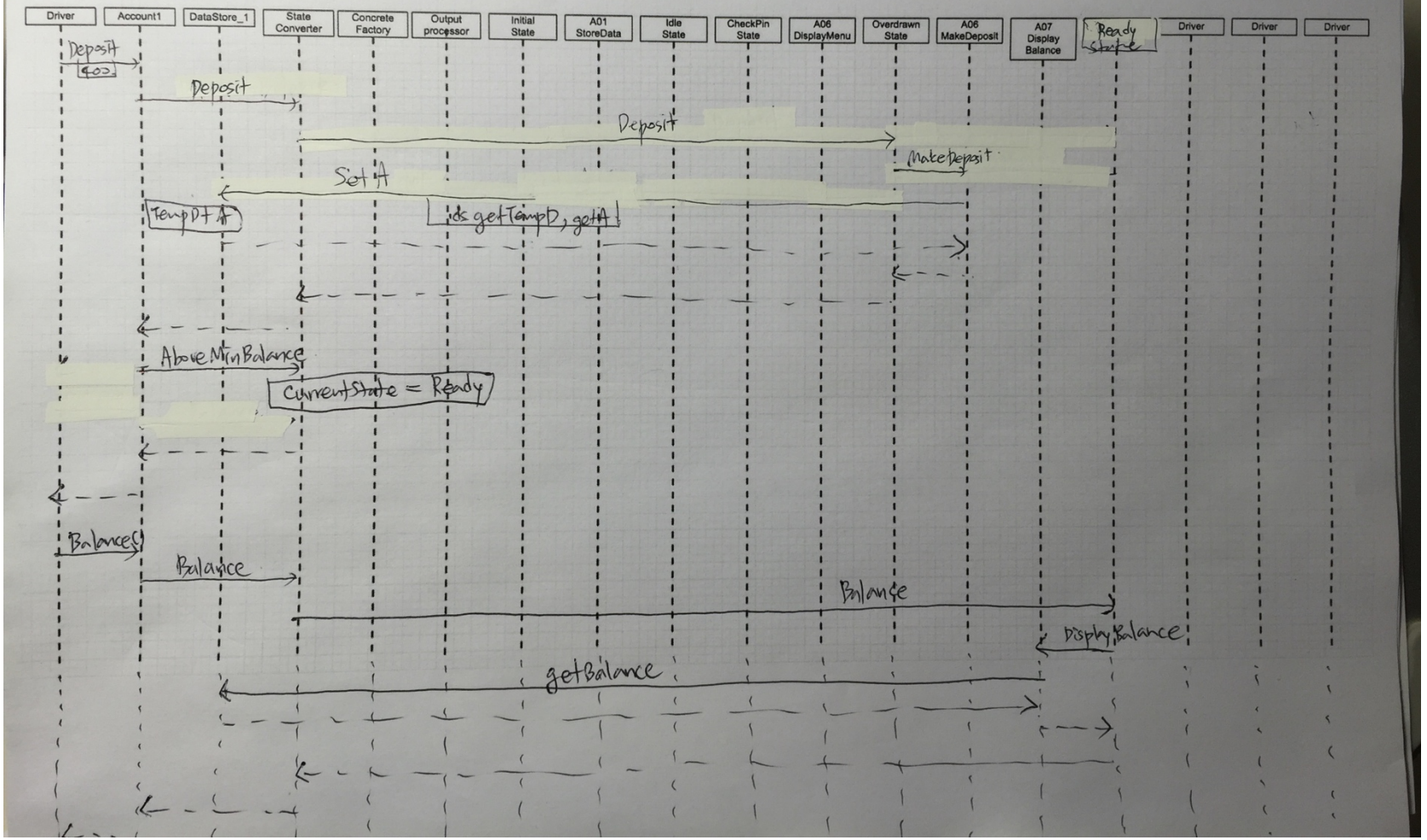
Next Page.

Scenario - I ① login ③pin.



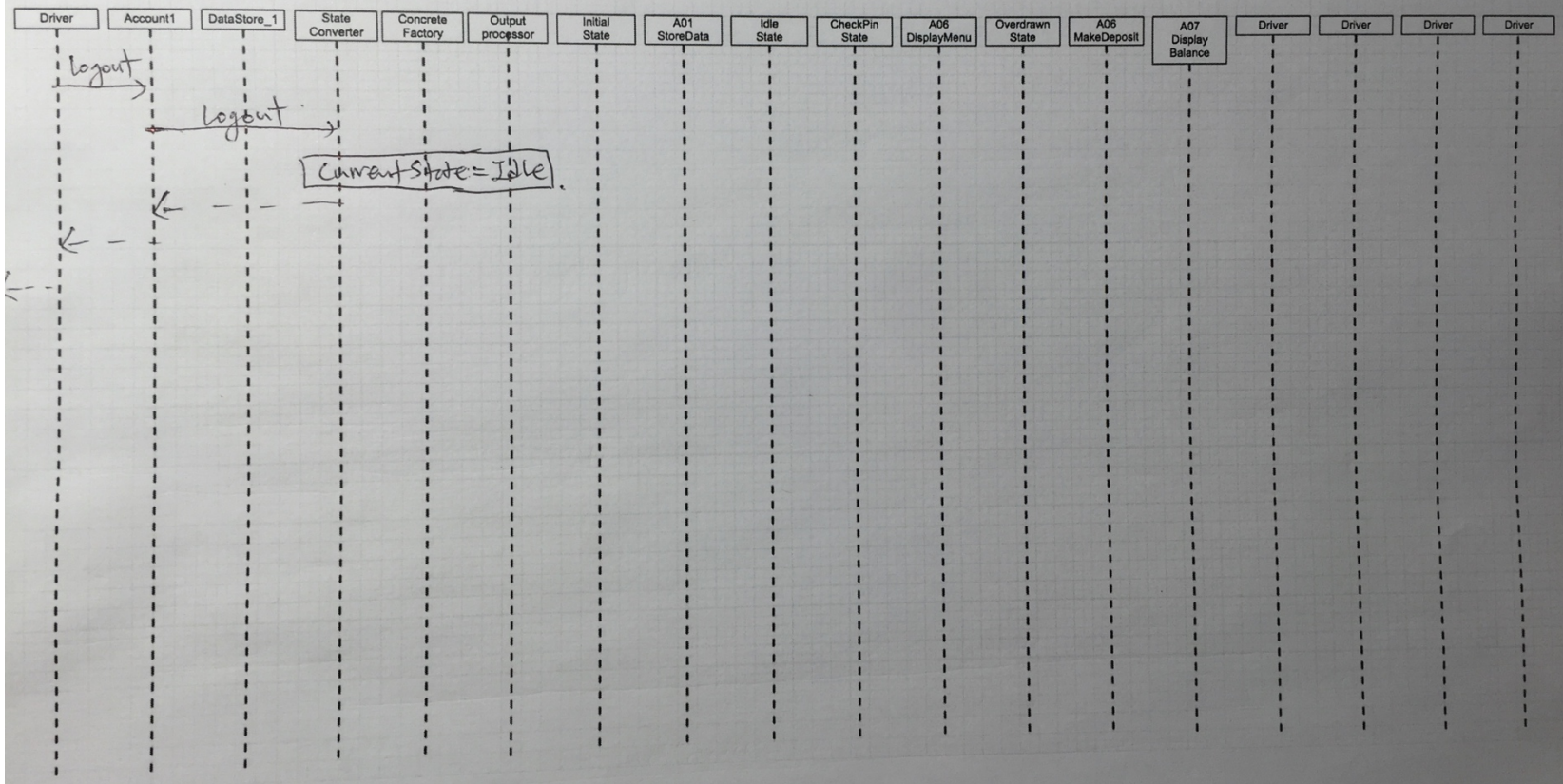
A06 Too Many Attempts

Scenario - I ④ Deposit / ⑤ balance

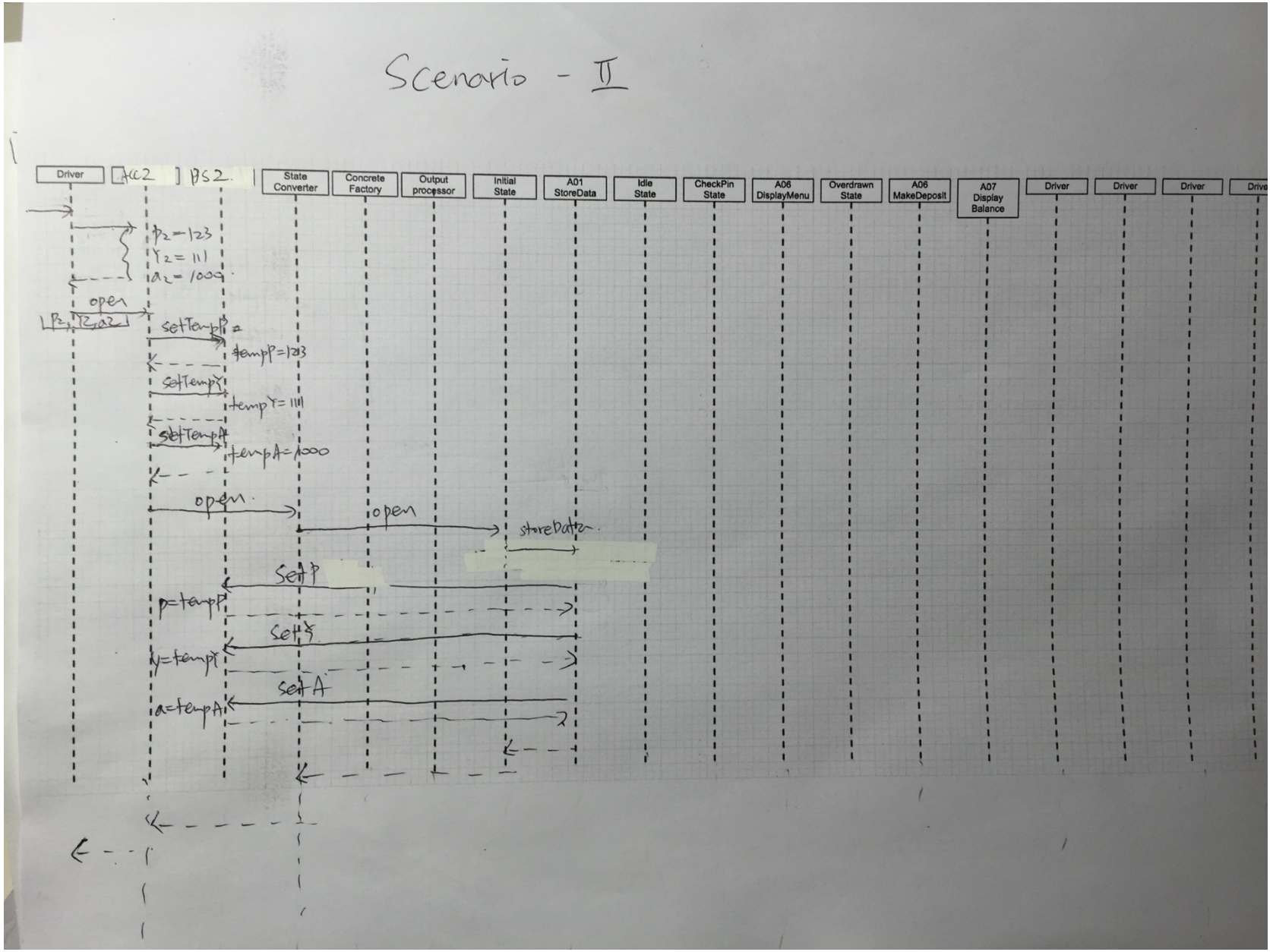


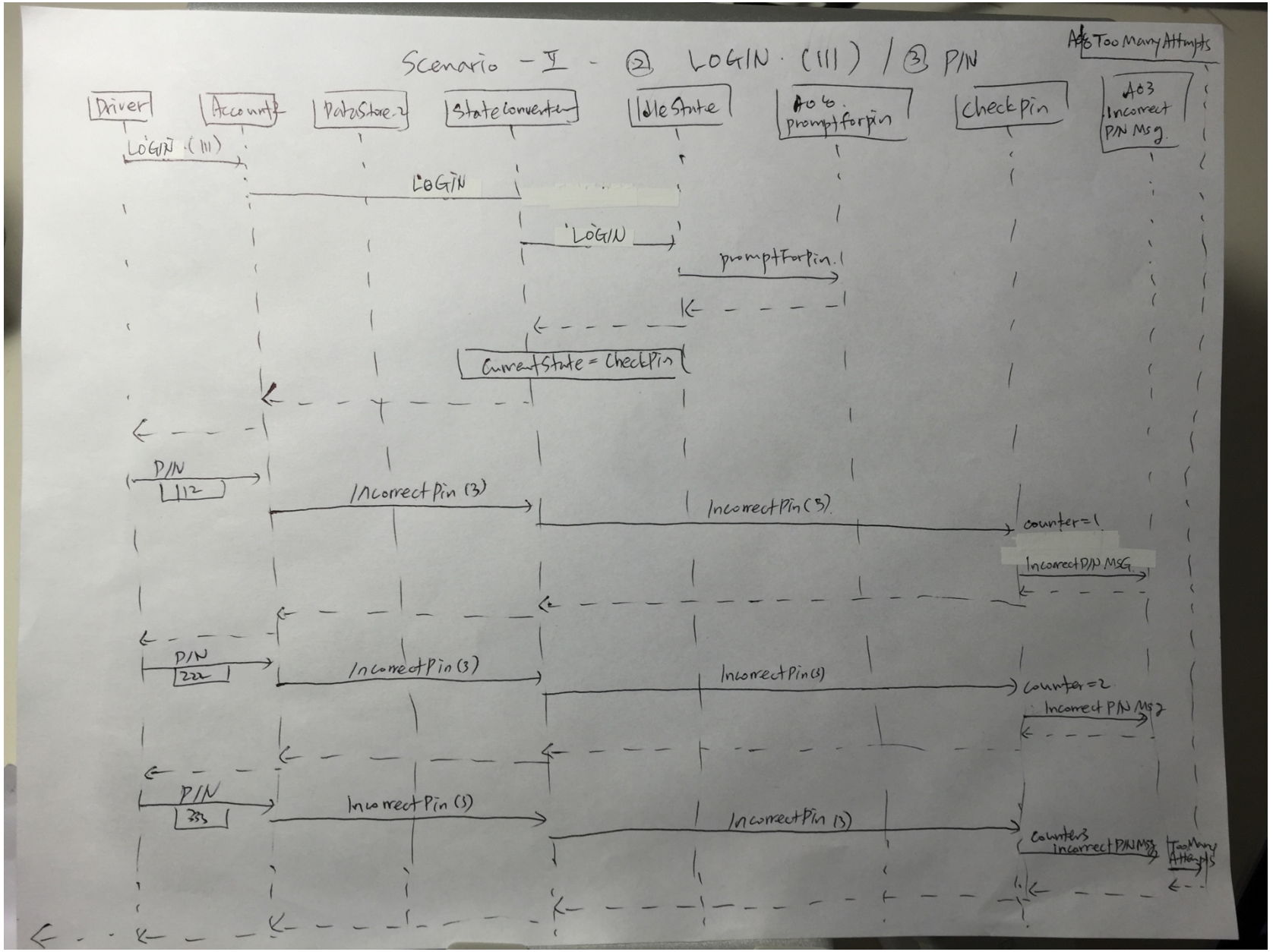
A06 Too Many Attmpts

Scenario - I ⑥ Logout.



Scenario - II


Driver
Acc2
Bs2
State Converter
Concrete Factory
Output processor
Initial State
A01 StoreData
Idle State
CheckPin State
A06 DisplayMenu
Overdrawn State
A06 MakeDeposit
A07 Display Balance
Driver
Driver
Driver
p2=123
r2=111
a2=1000
open
setTempP
tempP=123
setTempR
tempR=111
setTempA
tempA=1000
open
open
storeData
SetP
p=tempP
SetR
r=tempR
SetA
a=tempA



5. Source-code and patters

For state pattern,

efsm_states offers the interfaces to implement the real states .
Each of the states (from S0_InitialState to S8_Closed) implement interfaces.
Finally StateConverter class control the states and store a pointer to the current states

For strategy pattern,

All of abstract classes in the outputProcessor package offers the functions which are different from each other (Account 1 and Account 2).
And concrete classes in the outputProcessor package extends the abstract methods to provide the appropriate functions to the both accounts.

For Abstract pattern,

AbstractFactory class has outputProcessor Objects and methods which provide functions by getter. ConcreteFactory_01 and ConcreteFactory_02 extends AbstractFactory and instansiate them.

6. Source Code

package abstractFactory

Class AbstractFactory

```
package abstractFactory;

import dataStore.DataStore_01;

import outputProcessor.A01_StoreData_AC1;
import outputProcessor.A02_IncorrectIdMsg_AC1;
import outputProcessor.A03_IncorrectPinMsg_AC1;
import outputProcessor.A04_TooManyAttemptsMsg_AC1;
import outputProcessor.A05_DisplayMenu_AC1;
import outputProcessor.A06_MakeDeposit_AC1;
import outputProcessor.A07_DisplayBalance_AC1;
import outputProcessor.A08_PromptForPin_AC1;
import outputProcessor.A09_MakeWithdraw_AC1;
import outputProcessor.A10_Panelty_AC1;
import outputProcessor.A11_IncorrectLockMsg_AC1;
import outputProcessor.A12_IncorrectUnlockMsg_AC1;
import outputProcessor.A13_NoFundsMsg_AC1;

public class ConcreteFactory_01 extends AbstractFactory{

	public ConcreteFactory_01() {

		super.ds = new DataStore_01();

		super.sda = new A01_StoreData_AC1(super.ds);
		super.ima = new A02_IncorrectIdMsg_AC1(super.ds);
		super.ipm = new A03_IncorrectPinMsg_AC1();
		super.tma = new A04_TooManyAttemptsMsg_AC1();
		super.dma = new A05_DisplayMenu_AC1();
		super.mda = new A06_MakeDeposit_AC1(super.ds);
		super.dba = new A07_DisplayBalance_AC1(super.ds);
		super.pfa = new A08_PromptForPin_AC1();
		super.mwa = new A09_MakeWithdraw_AC1(super.ds);
		super.pna = new A10_Panelty_AC1(super.ds);
		super.ila = new A11_IncorrectLockMsg_AC1();
		super.iua = new A12_IncorrectUnlockMsg_AC1();
```

```
            super.nfa = new A13_NoFundsMsg_AC1();

        }
}
```

Class ConcreteFactory_01

```
package abstractFactory;

import dataStore.DataStore_01;

import outputProcessor.A01_StoreData_AC1;
import outputProcessor.A02_IncorrectIdMsg_AC1;
import outputProcessor.A03_IncorrectPinMsg_AC1;
import outputProcessor.A04_TooManyAttemptsMsg_AC1;
import outputProcessor.A05_DisplayMenu_AC1;
import outputProcessor.A06_MakeDeposit_AC1;
import outputProcessor.A07_DisplayBalance_AC1;
import outputProcessor.A08_PromptForPin_AC1;
import outputProcessor.A09_MakeWithdraw_AC1;
import outputProcessor.A10_Panelty_AC1;
import outputProcessor.A11_IncorrectLockMsg_AC1;
import outputProcessor.A12_IncorrectUnlockMsg_AC1;
import outputProcessor.A13_NoFundsMsg_AC1;

public class ConcreteFactory_01 extends AbstractFactory{

        public ConcreteFactory_01() {

                super.ds = new DataStore_01();

                super.sda = new A01_StoreData_AC1(super.ds);
                super.ima = new A02_IncorrectIdMsg_AC1(super.ds);
                super.ipm = new A03_IncorrectPinMsg_AC1();
                super.tma = new A04_TooManyAttemptsMsg_AC1();
                super.dma = new A05_DisplayMenu_AC1();
                super.mda = new A06_MakeDeposit_AC1(super.ds);
                super.dba = new A07_DisplayBalance_AC1(super.ds);
                super.pfa = new A08_PromptForPin_AC1();
                super.mwa = new A09_MakeWithdraw_AC1(super.ds);
                super.pna = new A10_Panelty_AC1(super.ds);
                super.ila = new A11_IncorrectLockMsg_AC1();
                super.iua = new A12_IncorrectUnlockMsg_AC1();
                super.nfa = new A13_NoFundsMsg_AC1();
```

```
	}
}
```

Class ConcreteFactory_02

```
package abstractFactory;

import dataStore.DataStore_02;
import outputProcessor.A01_StoreData_AC2;
import outputProcessor.A02_IncorrectIdMsg_AC2;
import outputProcessor.A03_IncorrectPinMsg_AC2;
import outputProcessor.A04_TooManyAttemptsMsg_AC2;
import outputProcessor.A05_DisplayMenu_AC2;
import outputProcessor.A06_MakeDeposit_AC2;
import outputProcessor.A07_DisplayBalance_AC2;
import outputProcessor.A08_PromptForPin_AC2;
import outputProcessor.A09_MakeWithdraw_AC2;
import outputProcessor.A10_Panelty_AC2;
import outputProcessor.A11_IncorrectLockMsg_AC2;
import outputProcessor.A12_IncorrectUnlockMsg_AC2;
import outputProcessor.A13_NoFundsMsg_AC2;

public class ConcreteFactory_02 extends AbstractFactory {

	public ConcreteFactory_02() {
		super.ds = new DataStore_02();

		super.sda = new A01_StoreData_AC2(super.ds);
		super.ima = new A02_IncorrectIdMsg_AC2(super.ds);
		super.ipm = new A03_IncorrectPinMsg_AC2();
		super.tma = new A04_TooManyAttemptsMsg_AC2();
		super.dma = new A05_DisplayMenu_AC2();
		super.mda = new A06_MakeDeposit_AC2(super.ds);
		super.dba = new A07_DisplayBalance_AC2(super.ds);
		super.pfa = new A08_PromptForPin_AC2();
		super.mwa = new A09_MakeWithdraw_AC2(super.ds);
		super.pna = new A10_Panelty_AC2(super.ds);
		super.ila = new A11_IncorrectLockMsg_AC2();
		super.iua = new A12_IncorrectUnlockMsg_AC2();
		super.nfa = new A13_NoFundsMsg_AC2();
	}
}
```

Package DataStore

Class DataStore_01

```
package dataStore;

/**
 * @author Seungho Han A20360115
 *
 * DataStore_01 inherits methods from DataStore to implement
 * appropriate methods of types.
 *
 * In this code, A indicates both of initial Balance and
 * the Balance during the execution of program.
 */

public class DataStore_01 extends DataStore {

	Object p;
	Object temp_p;

	Object y;
	String temp_y;

	Object a;
	Object temp_a;

	Object d;
	Object temp_d;

	Object w;
	Object temp_w;

	Object x;

	@Override
	public Object getP() {
		return p;
	}

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

```
@Override
public Object getTemp_p() {
	return temp_p;
}

@Override
public void setTemp_p(Object temp_p) {
	this.temp_p = (String)temp_p;
}

@Override
public Object getY() {
	return y.toString();
}

@Override
public void setY(Object y) {
	this.y = (String)y;
}

@Override
public Object getTemp_y() {
	return temp_y;
}

@Override
public void setTemp_y(Object temp_y) {
	this.temp_y = (String)temp_y;
}

@Override
public Object getA() {
	return a;
}

@Override
public void setA(Object a) {
	this.a = (float)a;
}

@Override
public Object getTemp_a() {
	return temp_a;
```

```
}

@Override
public void setTemp_a(Object temp_a) {
	this.temp_a = (float)temp_a;
}

@Override
public Object getD() {
	return d;
}

@Override
public void setD(Object d) {
	this.d = (float)d;
}

@Override
public Object getTemp_d() {
	return temp_d;
}

@Override
public void setTemp_d(Object temp_d) {
	this.temp_d = (float)temp_d;
}

@Override
public Object getW() {
	return w;
}

@Override
public void setW(Object w) {
	this.w = (float)w;
}

@Override
public Object getTemp_w() {
	return temp_w;
}

@Override
public void setTemp_w(Object temp_w) {
```

```
		this.temp_w = (float)temp_w;
	}

	@Override
	public Object getX() {
		return x;
	}

	@Override
	public void setX(String x) {
		this.x = (String)x;
	}
}
```

Class DataStore_02

```
package dataStore;
/**
 * @author Seungho Han A20360115
 *
 * DataStore_02 inherits methods from DataStore to implement
 * appropriate methods of types.
 *
 * In this code, A indicates both of initial Balance and
 * the Balance during the execution of program.
 */
public class DataStore_02 extends DataStore {

	Object p;

	Object temp_p;

	Object y;
	Object temp_y;

	Object a;
	Object temp_a;

	Object x;

	Object d;
	Object temp_d;

	Object w;
```

```
Object temp_w;

@Override
public Object getP() {
	return p;
}

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

public Object getTemp_p() {
	return temp_p;
}

public void setTemp_p(Object temp_p) {
	this.temp_p = (int)temp_p;
}

public Object getY() {
	return y;
}

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

public Object getTemp_y() {
	return temp_y;
}

public void setTemp_y(Object temp_y) {
	this.temp_y = (int)temp_y;
}

public Object getA() {
	return (int)a;
}

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

public Object getTemp_a() {
```

```
        return temp_a;
}

public void setTemp_a(Object temp_a) {
        this.temp_a = (int)temp_a;
}

public Object getX() {
        return x;
}

public void setX(Object x) {
        this.x = (int)x;
}

public Object getD() {
        return d;
}

public void setD(Object d) {
        this.d = (int)d;
}

public Object getTemp_d() {
        return temp_d;
}

public void setTemp_d(Object temp_d) {
        this.temp_d = (int)temp_d;
}

public Object getW() {
        return w;
}

public void setW(Object w) {
        this.w = (int)w;
}

public Object getTemp_w() {
        return temp_w;
}

public void setTemp_w(Object temp_w) {
```

```
                this.temp_w = (int)temp_w;
        }
}
```

Abstract Class DataStore

```
public abstract class DataStore {

        public Object getP(){ return null; }

        public void setP(Object p) { }

        public Object getTemp_p() {   return null; }

        public void setTemp_p(Object temp_p) { }

        public Object getY() { return null; }

        public void setY(Object y) { }

        public Object getTemp_y() { return null; }

        public void setTemp_y(Object temp_y) { }

        public Object getA() { return null; }

        public void setA(Object a) { }

        public Object getTemp_a() { return null;        }

        public void setTemp_a(Object temp_a) { }

        public Object getD() { return null;}

        public void setD(Object d) { }

        public Object getTemp_d() { return null; }

        public void setTemp_d(Object temp_d) { }

        public Object getPin() { return null; }

        public void setPin(Object pin) {          }
```

```
	public Object getTemp_pin() { return null; }

	public void setTemp_pin(Object temp_pin) { }

	public Object getW() { return null; }

	public void setW(Object w) { }

	public Object getTemp_w() { return null;	}

	public void setTemp_w(Object temp_w) { }

	public Object getBalance() { return null;	}

	public void setBalance(Object balance) { }

	public Object getUid() { return null; }

	public void setUid(Object uid) { }

	public Object getX() { return null;}

	public void setX(String x) { }

}
```

package MDA_EFSM

interface efsm_states

```
package MDA_EFSM;

/**
 * @author Seungho Han A20360115
 * efsm_states interface offers interfaces which should be
 * implemented at each of states.
 * And each of states have a OutputProcessor instance, which
 * gets AbstractFactory.
 */

public interface efsm_states {
	void Open();
```

```
        void Login();
        void Pin();
        void IncorrectLogin();
        void IncorrectPin(int max);
        void CorrectPinBelowMin();
        void CorrectPinAboveMin();
        void Deposit();
        void BelowMinBalance();
        void AboveMinBalance();
        void Logout();
        void Balance();
        void Withdraw();
        void WithdrawBelowMinBalance();
        void NoFunds();
        void Lock();
        void IncorrectLock();
        void Unlock();
        void IncorrectUnlock();
        void Suspend();
        void Active();
        void Close();
}
```

class S0_InitialState implements

performs functions before the Idle state
: Open

Attributes:
OutputProcessor Action;

Operations:

```
        public S0_InitialState(OutputProcessor op) {
                Action = op;
        }

        @Override
        public void Open() {
                Action.StoreData();
        }

        @Override
        public void Login() {}
```

```
@Override
public void Pin() {}

@Override
public void IncorrectLogin() { }

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
```

```
public void IncorrectLock() {}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {}

@Override
public void Suspend() {}

@Override
public void Active() {}

@Override
public void Close() {}
```

class S1_Idle implements

Idle state performs the function related Login

Attributes:
OutputProcessor Action;

Operations:

```
public S1_Idle(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {

}

@Override
public void Login() {
        System.out.println("S1->Login");
        Action.PromptForPin();
}

@Override
public void Pin() {}
```

```
@Override
public void IncorrectLogin() {
	System.out.println("IncorrectLogin");
	Action.IncorrectIdMsg();
}

@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
```

```
        public void IncorrectLock() {}

        @Override
        public void Unlock() {}

        @Override
        public void IncorrectUnlock() {}

        @Override
        public void Suspend() {}

        @Override
        public void Active() {}

        @Override
        public void Close() {}
}
```

class S2_CheckPin

Check Pin state performs functions related the permission

Attributes:
OutputProcessor Action;

```
static int counter;             // this counter is used in the IncorrectPin
                                // to print TooManyAttemptsMsg
```

operations:

```
        public S2_CheckPin(OutputProcessor op) {
                Action = op;
                counter = 0;
        }

        @Override
        public void Open() {}

        @Override
        public void Login() {}

        @Override
        public void Pin() {}

        @Override
```

```
public void IncorrectLogin() {}

@Override
public void IncorrectPin(int max) {
	counter++;
	Action.IncorrectPinMsg();
	if (counter == max) {
		Action.TooManyAttemptsMsg();
	}
}

@Override
public void CorrectPinBelowMin() { Action.DisplayMenu(); }

@Override
public void CorrectPinAboveMin() { Action.DisplayMenu();  }

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}
```

```
    @Override
    public void IncorrectLock() {}

    @Override
    public void Unlock() {}

    @Override
    public void IncorrectUnlock() {}

    @Override
    public void Suspend() {}

    @Override
    public void Active() {}

    @Override
    public void Close() {}
```

class S3_Ready

Attributes:
OutputProcessor Action;

Operations:

```
    public S3_Ready(OutputProcessor op) {
        Action = op;
    }

    @Override
    public void Open() {}

    @Override
    public void Login() {}

    @Override
    public void Pin() {}

    @Override
    public void IncorrectLogin() {}

    @Override
    public void IncorrectPin(int max) {}

    @Override
```

```
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {
	Action.MakeDeposit();
}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {
	Action.DispayBalance();
}

@Override
public void Withdraw() {
	// TODO Auto-generated method stub
	Action.MakeWithdraw();
}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {
	Action.NoFundsMsg();
}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {
	Action.IncorrectLockMsg();
```

```
}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {}

@Override
public void Suspend() {}

@Override
public void Active() {}

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

class S4_IntermediateState

Intermediate State means S1 state which is located in the sample solution of MDA-EFSM

Attributes:
OutputProcessor Action;

Operations:

```
public S4_IntermediateState(OutputProcessor op) {
        Action = op;
}

@Override
public void Open() {}

@Override
public void Login() {}

@Override
public void Pin() {}

@Override
public void IncorrectLogin() {}
```

```
@Override
public void IncorrectPin(int max) {}

@Override
public void CorrectPinBelowMin() {}

@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {
        Action.Penalty();
}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {}

@Override
public void Unlock() {}
```

```
        @Override
        public void IncorrectUnlock() {}

        @Override
        public void Suspend() {}

        @Override
        public void Active() {}

        @Override
        public void Close() {}
}
```

class S5_Overdrawn

Attributes:
OutputProcessor Action;

Operations:

```
        public S5_Overdrawn(OutputProcessor op) {
                Action = op;
        }

        @Override
        public void Open() {}

        @Override
        public void Login() {}

        @Override
        public void Pin() {}

        @Override
        public void IncorrectLogin() {}

        @Override
        public void IncorrectPin(int max) {}

        @Override
        public void CorrectPinBelowMin() {}

        @Override
        public void CorrectPinAboveMin() {}
```

```
@Override
public void Deposit() {
	Action.MakeDeposit();}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {
	Action.DispayBalance();
}

@Override
public void Withdraw() {
	Action.NoFundsMsg();

}

@Override
public void WithdrawBelowMinBalance() {
	Action.Penalty();

}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {}

@Override
public void Unlock() {}

@Override
```

```
    public void IncorrectUnlock() {}

    @Override
    public void Suspend() {}

    @Override
    public void Active() {}

    @Override
    public void Close() {}

}
```

class S6_Locked

Only Account 1 can be included in this state

Attributes:
OutputProcessor Action;

Operations:

```
    public S6_Locked(OutputProcessor op) {
        Action = op;
    }

    @Override
    public void Open() {}

    @Override
    public void Login() {}

    @Override
    public void Pin() {}

    @Override
    public void IncorrectLogin() {}

    @Override
    public void IncorrectPin(int max) {}

    @Override
    public void CorrectPinBelowMin() {}
```

```
@Override
public void CorrectPinAboveMin() {}

@Override
public void Deposit() {}

@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {
        Action.IncorrectUnlockMsg();
}

@Override
public void Suspend() {}
```

```
	@Override
	public void Active() {}

	@Override
	public void Close() {}

}
```

class S7_Suspended

Only Account 2 can be included in this state

Attributes:
OutputProcessor Action;

Operations:

```
	public S7_Suspended(OutputProcessor op) {
		Action = op;
	}

	@Override
	public void Open() {}

	@Override
	public void Login() {}

	@Override
	public void Pin() {}

	@Override
	public void IncorrectLogin() {}

	@Override
	public void IncorrectPin(int max) {}

	@Override
	public void CorrectPinBelowMin() {}

	@Override
	public void CorrectPinAboveMin() {}

	@Override
	public void Deposit() {}
```

```
@Override
public void BelowMinBalance() {}

@Override
public void AboveMinBalance() {}

@Override
public void Logout() {}

@Override
public void Balance() {
        System.out.println("S7->Balance");
        Action.DispayBalance();
}

@Override
public void Withdraw() {}

@Override
public void WithdrawBelowMinBalance() {}

@Override
public void NoFunds() {}

@Override
public void Lock() {}

@Override
public void IncorrectLock() {}

@Override
public void Unlock() {}

@Override
public void IncorrectUnlock() {}

@Override
public void Suspend() {}

@Override
public void Active() {}

@Override
```

```
	public void Close() {}

}
```

class S8_Closed

Only Account 2 can be included in this state

Attributes:
OutputProcessor Action;

Operations:

```
	public S8_Closed(OutputProcessor op) {
		Action = op;
	}

	@Override
	public void Open() {}

	@Override
	public void Login() {}

	@Override
	public void Pin() {}

	@Override
	public void IncorrectLogin() {}

	@Override
	public void IncorrectPin(int max) {}

	@Override
	public void CorrectPinBelowMin() {}

	@Override
	public void CorrectPinAboveMin() {}

	@Override
	public void Deposit() {}

	@Override
	public void BelowMinBalance() {}
```

```
    @Override
    public void AboveMinBalance() {}

    @Override
    public void Logout() {}

    @Override
    public void Balance() {}

    @Override
    public void Withdraw() {}

    @Override
    public void WithdrawBelowMinBalance() {}

    @Override
    public void NoFunds() {}

    @Override
    public void Lock() {}

    @Override
    public void IncorrectLock() {}

    @Override
    public void Unlock() {}

    @Override
    public void IncorrectUnlock() {}

    @Override
    public void Suspend() {}

    @Override
    public void Active() {}

    @Override
    public void Close() {}

}
```

class StateConverter {

```
    private efsm_states IS;
```

```
private efsm_states IL;
private efsm_states CP;
private efsm_states RD;
private efsm_states IT;
private efsm_states OD;
private efsm_states LK;
private efsm_states SP;
private efsm_states CD;

private efsm_states CurrentState;

static int counter;

public StateConverter(OutputProcessor op) {          // Constructor

        IS      = new S0_InitialState(op);
        IL      = new S1_Idle(op);
        CP      = new S2_CheckPin(op);
        RD      = new S3_Ready(op);
        IT      = new S4_IntermediateState(op);
        OD      = new S5_Overdrawn(op);
        LK      = new S6_Locked(op);
        SP      = new S7_Suspended(op);
        CD      = new S8_Closed(op);

        CurrentState = IS ;

        counter = 0;

}


public void Open() {
        if (CurrentState == IS) {
                CurrentState.Open();
                CurrentState = IL;
        }
}

public void Login() {
        if (CurrentState == IL) {
                CurrentState.Login();
                CurrentState = CP;
        }
```

```
}

public void Pin() {
	if (CurrentState == CP) {
		CurrentState.Pin();
		CurrentState = RD;
	}
}

public void IncorrectLogin() {
	if (CurrentState == IL) {
		CurrentState.IncorrectLogin();
		CurrentState = IL;
	}
}

public void IncorrectPin(int max) {

	counter++;

	CurrentState.IncorrectPin(max);

	if(counter == max) {			// if the number of invalid pin attempts is
		CurrentState = IL;		// maximam, then make state Idle

	}
}


public void CorrectPinAboveMin() {
	if (CurrentState == CP) {
		CurrentState.CorrectPinAboveMin();
		CurrentState = RD;
	}
}

public void CorrectPinBelowMin() {
	if (CurrentState == CP) {
		CurrentState.CorrectPinBelowMin();
		CurrentState = OD;
	}
}

public void Deposit() {				// executes both in the Ready or Overdrawn
```

```
        if (CurrentState == RD) {
                CurrentState.Deposit();
                CurrentState = RD;
        } else if (CurrentState == OD) {
                CurrentState.Deposit();
                CurrentState = IT;
        }
}

public void AboveMinBalance() {
        if (CurrentState == IT) {
                CurrentState = RD;
        }
}

public void BelowMinBalance() {
        if (CurrentState == IT) {
                CurrentState = OD;
        }
}

public void Logout() {
        if (CurrentState == RD) {
                CurrentState = IL;
        }
}

public void Balance() {                 // executes in Ready, Overdrawn, Suspended
        if (CurrentState == RD) {
                CurrentState.Balance();
        } else if (CurrentState == OD) {
                CurrentState.Balance();
        } else if (CurrentState == SP) {
                CurrentState.Balance();
        }
}

public void Withdraw() {
        if (CurrentState == RD) {
                CurrentState.Withdraw();
                CurrentState = IT;
        } else if (CurrentState == OD) {
                CurrentState.Withdraw();
                CurrentState = OD;
```

```
        }
}

public void WithdrawBelowMinBalance() {
        if (CurrentState == IT) {
                CurrentState.WithdrawBelowMinBalance();
                CurrentState = OD;
        }
}

public void NoFunds() {
        if (CurrentState == RD) {
                CurrentState = RD;
        }
}

public void Lock() {
        if (CurrentState == RD) {
                CurrentState = LK;
        } else if (CurrentState == OD) {
                CurrentState = LK;
        }
}

public void IncorrectLock() {
        if (CurrentState == RD) {
                CurrentState = RD;
        }
}

public void Unlock() {
        if (CurrentState == LK) {
                CurrentState = IT;
        }
}

public void IncorrectUnlock() {
        if (CurrentState == LK) {
                CurrentState.IncorrectUnlock();
                CurrentState = LK;
        }
}

public void Suspend() {
```

```
		if (CurrentState == RD) {
			CurrentState = SP;
		}
	}

	public void Activate() {
		if (CurrentState == SP) {
			CurrentState = RD;
		}
	}

	public void Close() {
		if (CurrentState == SP) {
			CurrentState = CD;
		}
	}
}
```

package outputProcessor

```
package outputProcessor;

import dataStore.DataStore;

public abstract class A01_StoreData_Abs {

	DataStore ds;

	public A01_StoreData_Abs(DataStore dataStore) {
		ds = dataStore;
	}

	public void StoreData() {}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A01_StoreData_AC1 extends A01_StoreData_Abs {

	public A01_StoreData_AC1(DataStore ds1) {
		super(ds1);
```

```
	}

	@Override
	public void StoreData() {
		ds.setP(ds.getTemp_p());
		ds.setY(ds.getTemp_y());
		ds.setA(ds.getTemp_a());
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A01_StoreData_AC2 extends A01_StoreData_Abs {

	public A01_StoreData_AC2(DataStore _ds2) {
		super(_ds2);
	}

	public void StoreData() {
		ds.setP(ds.getTemp_p());
		ds.setY(ds.getTemp_y());
		ds.setA(ds.getTemp_a());
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public abstract class A02_IncorrectIdMsg_Abs {

	DataStore ds;

	public A02_IncorrectIdMsg_Abs(DataStore dataStore) {
		ds = dataStore;
	}

	public void IncorrectIdMsg() {
		System.out.println("Incorrect ID");
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A02_IncorrectIdMsg_AC1 extends A02_IncorrectIdMsg_Abs {

	public A02_IncorrectIdMsg_AC1(DataStore ds1) {
		super(ds1);
	}

	public void IncorrectIdMsg() {
		System.out.println("IncorrectIdMsg");
		if(ds.getY() != ds.getTemp_y()) {
			System.out.println("Incorrect ID");
		}
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A02_IncorrectIdMsg_AC2 extends A02_IncorrectIdMsg_Abs {

	public A02_IncorrectIdMsg_AC2(DataStore ds2) {
		super(ds2);
		// TODO Auto-generated constructor stub
	}

	public void IncorrectIdMsg() {
		System.out.println("Incorrect ID");
	}
}
```

```
package outputProcessor;

public abstract class A03_IncorrectPinMsg_Abs {

	public void IncorrectPinMsg() {
		System.out.println("Incorrect Pin");
```

```
	}
}
```

```
package outputProcessor;

public class A03_IncorrectPinMsg_AC1 extends A03_IncorrectPinMsg_Abs {

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

```
package outputProcessor;

public class A03_IncorrectPinMsg_AC2 extends A03_IncorrectPinMsg_Abs {


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

```
package outputProcessor;

public abstract class A04_TooManyAttemptsMsg_Abs {

	public void TooManyAttemptsMsg() {
		System.out.println("Too Many Attempts");
	}
}
```

```
package outputProcessor;

public class A04_TooManyAttemptsMsg_AC1 extends A04_TooManyAttemptsMsg_Abs {

	public void TooManyAttemptsMsg() {
		System.out.println("Too Many Attempts");
	}
}
```

```
package outputProcessor;

public class A04_TooManyAttemptsMsg_AC2 extends A04_TooManyAttemptsMsg_Abs {
```

```
	public void TooManyAttemptsMsg() {
		System.out.println("Too Many Attempts");
	}
}
```

```
package outputProcessor;

public abstract class A05_DisplayMenu_Abs {
	public void DisplayMenu() {

	}
}
```

```
package outputProcessor;

public class A05_DisplayMenu_AC1 extends A05_DisplayMenu_Abs {
	public void DisplayMenu() {
		System.out.println("TRANSACTION MENU");
		System.out.println("deposit");
		System.out.println("withdraw");
		System.out.println("logout");
		System.out.println("lock");
		System.out.println("unlock");
	}
}
```

```
package outputProcessor;

public class A05_DisplayMenu_AC2 extends A05_DisplayMenu_Abs {
	public void DisplayMenu() {
		System.out.println("TRANSACTION MENU");
		System.out.println("DEPOSIT");
		System.out.println("WITHDRAW");
		System.out.println("BALANCE");
		System.out.println("LOGOUT");
		System.out.println("suspend");
		System.out.println("active");
		System.out.println("close");
	}
}
```

```
package outputProcessor;
```

```
import dataStore.DataStore;

public abstract class A06_MakeDeposit_Abs {

	DataStore ds;

	public A06_MakeDeposit_Abs(DataStore dataStore) {
		ds = dataStore;
	}

	public void MakeDeposit() {
		ds.setA((float)ds.getTemp_d() + (float)ds.getA());
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A06_MakeDeposit_AC1 extends A06_MakeDeposit_Abs {

	public A06_MakeDeposit_AC1(DataStore ds1) {
		super(ds1);
	}

	public void MakeDeposit() {
		ds.setA((float)ds.getTemp_d() + (float)ds.getA());
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A06_MakeDeposit_AC2 extends A06_MakeDeposit_Abs {

	public A06_MakeDeposit_AC2(DataStore ds2) {
		super(ds2);
	}

	public void MakeDeposit() {
		ds.setA((int)ds.getTemp_d() + (int)ds.getA());
```

```
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public abstract class A07_DisplayBalance_Abs {

	DataStore ds;

	public A07_DisplayBalance_Abs(DataStore dataStore) {
		ds = dataStore;
	}

	public void DisplayBalance() {

	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A07_DisplayBalance_AC1 extends A07_DisplayBalance_Abs {

	public A07_DisplayBalance_AC1(DataStore ds1) {
		super(ds1);
	}

	public void DisplayBalance() {
		System.out.println("Current balance is " + ds.getA());
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A07_DisplayBalance_AC2 extends A07_DisplayBalance_Abs {

	public A07_DisplayBalance_AC2(DataStore ds2) {
		super(ds2);
```

```
	}

	public void DisplayBalance() {
		System.out.println("Current balance is " + ds.getA());
	}
}
```

```
package outputProcessor;

public abstract class A08_PromptForPin_Abs {

	public void PromptForPin() {
		System.out.println("Please enter your pin number");
	}
}
```

```
package outputProcessor;

public class A08_PromptForPin_AC1 extends A08_PromptForPin_Abs {

	public void PromptForPin() {
		System.out.println("Please enter your pin number");
	}
}
```

```
package outputProcessor;

public class A08_PromptForPin_AC2 extends A08_PromptForPin_Abs {

	public void PromptForPin() {
		System.out.println("Please enter your pin number");
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public abstract class A09_MakeWithdraw_Abs {

	DataStore ds;

	public A09_MakeWithdraw_Abs(DataStore dataStore) {
```

```
		ds = dataStore;
	}

	public void MakeWithdraw() {

		ds.setA((float)ds.getA() - (float)ds.getTemp_w());
	}

}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A09_MakeWithdraw_AC1 extends A09_MakeWithdraw_Abs {

	public A09_MakeWithdraw_AC1(DataStore ds1) {
		super(ds1);
	}

	public void MakeWithdraw() {

		ds.setA((float)ds.getA()- (float)ds.getTemp_w());
	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A09_MakeWithdraw_AC2 extends A09_MakeWithdraw_Abs {

	public A09_MakeWithdraw_AC2(DataStore ds2) {
		super(ds2);
	}

	public void MakeWithdraw() {
		ds.setA((int)ds.getA() - (int)ds.getTemp_w());
	}
}
```

```
package outputProcessor;
```

```
import dataStore.DataStore;

public abstract class A10_Panelty_Abs {
	DataStore ds;

	public A10_Panelty_Abs(DataStore dataStore) {
		ds = dataStore;
	}

	public void Penalty() {

	}
}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A10_Panelty_AC1 extends A10_Panelty_Abs {

	public A10_Panelty_AC1(DataStore ds1) {
		super(ds1);
	}

	public void Penalty() {
		ds.setA((float )ds.getA()-20);
	}

}
```

```
package outputProcessor;

import dataStore.DataStore;

public class A10_Panelty_AC2 extends A10_Panelty_Abs {
	public A10_Panelty_AC2(DataStore ds2) {
		super(ds2);
	}

	public void Penalty() {
		ds.setA((int)ds.getA()-20);
	}
}
```

```
package outputProcessor;

public abstract class A11_IncorrectLockMsg_Abs {

	public void IncorrectIdMsg() {
		System.out.println("Incorrect Lock");
	}
}
```

```
package outputProcessor;

public class A11_IncorrectLockMsg_AC1 extends A11_IncorrectLockMsg_Abs{

	public void IncorrectLockMsg() {
		System.out.println("Incorret Lock");
	}
}
```

```
package outputProcessor;

public class A11_IncorrectLockMsg_AC2 extends A11_IncorrectLockMsg_Abs {

	public void IncorrectLockMsg() {
		System.out.println("Incorret Lock");
	}
}
```

```
package outputProcessor;

public abstract class A12_IncorrectUnlockMsg_Abs {
	public void IncorrectUnlockMsg() {
		System.out.println("Incorrect Unlock");
	}
}
```

```
package outputProcessor;

public class A12_IncorrectUnlockMsg_AC1 extends A12_IncorrectUnlockMsg_Abs {

	public void IncorrectUnlockMsg() {
		System.out.println("Incorrect Unlock");
```

```
	}

}
```

```
package outputProcessor;

public class A12_IncorrectUnlockMsg_AC2 extends A12_IncorrectUnlockMsg_Abs {

	public void IncorrectUnlockMsg() {
		System.out.println("Incorrect Unlock");
	}
}
```

```
package outputProcessor;

public abstract class A13_NoFundsMsg_Abs {


	public void NoFundsMsg() {
		System.out.println("No Funds");
	}
}
```

```
package outputProcessor;

public class A13_NoFundsMsg_AC1 extends A13_NoFundsMsg_Abs {

	public void NoFundsMsg() {
		System.out.println("No Funds");
	}
}
```

```
package outputProcessor;

public class A13_NoFundsMsg_AC2 extends A13_NoFundsMsg_Abs {

	public void NoFundsMsg() {
		System.out.println("No Funds");
	}
}
```

```
package outputProcessor;
```

```
import abstractFactory.AbstractFactory;
import dataStore.DataStore;

public class OutputProcessor {
	public DataStore ds;

	public A01_StoreData_Abs sda;
	public A02_IncorrectIdMsg_Abs ima;
	public A03_IncorrectPinMsg_Abs ipm;
	public A04_TooManyAttemptsMsg_Abs tma;
	public A05_DisplayMenu_Abs dma;
	public A06_MakeDeposit_Abs mda;
	public A07_DisplayBalance_Abs dba;
	public A08_PromptForPin_Abs pfa;
	public A09_MakeWithdraw_Abs mwa;
	public A10_Panelty_Abs pna;
	public A11_IncorrectLockMsg_Abs ila;
	public A12_IncorrectUnlockMsg_Abs iua;
	public A13_NoFundsMsg_Abs nfa;

	public OutputProcessor (AbstractFactory af) {
		ds = af.getDS();
		sda = af.getA01();
		ima = af.getA02();
		ipm = af.getA03();
		tma = af.getA04();
		dma = af.getA05();
		mda = af.getA06();
		dba = af.getA07();
		pfa = af.getA08();
		mwa = af.getA09();
		pna = af.getA10();
		ila = af.getA11();
		iua = af.getA12();
		nfa = af.getA13();
	}

	public void StoreData() {
		System.out.println("outputProcessor -> StoreData");
		sda.StoreData();
	}

	public void IncorrectIdMsg() {
		ima.IncorrectIdMsg();
```

```
}

public void IncorrectPinMsg() {
        ipm.IncorrectPinMsg();
}

public void TooManyAttemptsMsg() {
        tma.TooManyAttemptsMsg();
}

public void DisplayMenu() {
        dma.DisplayMenu();
}

public void MakeDeposit() {
        mda.MakeDeposit();
}

public void DispayBalance() {
        dba.DisplayBalance();
}

public void PromptForPin() {
        pfa.PromptForPin();
}

public void MakeWithdraw() {
        mwa.MakeWithdraw();
}

public void Penalty() {
        pna.Penalty();
}

public void IncorrectLockMsg() {
        ila.IncorrectIdMsg();
}

public void IncorrectUnlockMsg() {
        iua.IncorrectUnlockMsg();
}

public void NoFundsMsg() {
        nfa.NoFundsMsg();
```

```
        }
}
```