CS 586 Project Report

Jian Zhang

A20327380

5/1/2016

Index

1. MDA-EFSM model

A. List of Events:

1. Open()
2. Login()
3. IncorrectLogin()
4. IncorrectPin(int max)
5. CorrectPinAboveMin()
6. CorrectPinBelowMin()
7. Logout()
8. Deposit()
9. DepositAboveMin()
10. DepositBelowMin()
11. Balance()
12. Withdraw()
13. WithdrawAboveMin()
14. WithdrawPenalty()
15. NoFound()
16. Lock()
17. IncorrectLock()
18. IncorrectUnlock()
19. UnlockAboveMin()
20. UnlockBelowMin()
21. Suspend()
22. Activate()
23. Close()

B. List of Actions:

A1:
StoreData(): store the temp data.

A2:
IncorrectIDMsg(): Display the message for incorrect ID.

A3:
PromptPin(): Display the prompt PIN message.

A4:
IncorrectPinMsg(): Display the message for incorrect pin.

A5:
TooManyAttemptsMsg(): Display the message for too many attempts of entering pin.

A6:
DisplayMenu(): Display the menu.

A7:
MakeDeposit(): Making deposit with deposit amount.

A8:
DisplayBalance(): Display the value of current balance.

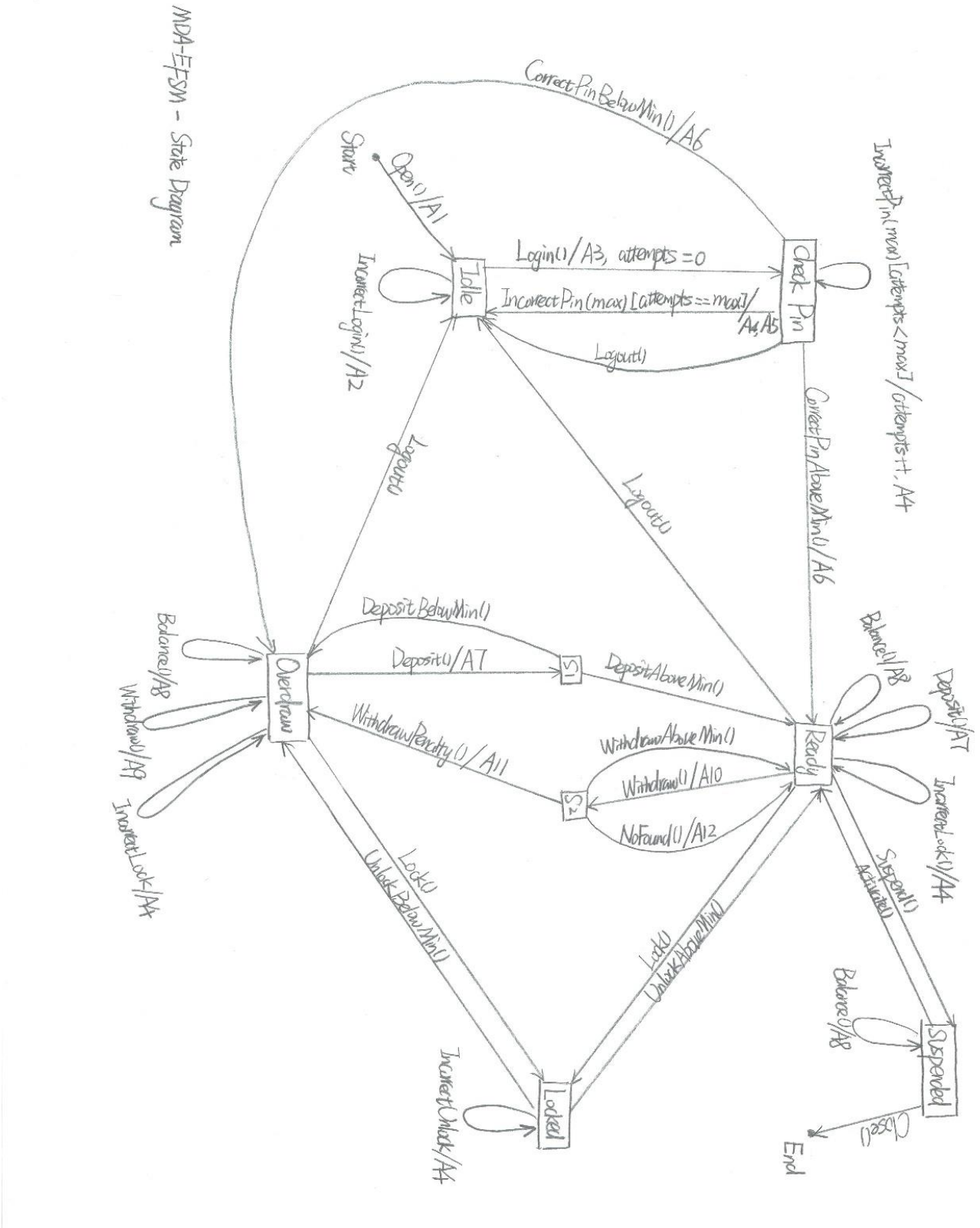
A9:
BelowMinMsg(): Display the message that the current balance is below the minimum required balance.

A10:
MakeWithdraw(): Making withdraw with withdraw amount.

A11:
BelowMinPenalty(): Apply the penalty for the balance witch is below the minimum.

A12:
NoFoundMsg(): Display the message that there is no found in the account.

C. MDA-EFSM


MDA-EFSM - State Diagram
Start
Open()/A1
Idle
IncorrectLogin()/A2
Login()/A3, attempts=0
Check Pin
IncorrectPin(max)[attempts<max]/attempts++, A4
IncorrectPin(max)[attempts==max]/A4,A5
Logout()
CorrectPinBelowMin()/A6
CorrectPinAboveMin()/A6
Logout()
Logout()
Overdraw
Balance()/A8
Withdraw()/A9
IncorrectLock()/A4
Deposit BelowMin()
Deposit()/A7
S1
DepositAboveMin()
WithdrawReady()/A11
S2
WithdrawAboveMin()
Withdraw()/A10
NoFound()/A12
Ready
Balance()/A8
Deposit()/A7
IncorrectLock()/A4
Lock()
UnlockBelowMin()
Lock()
UnlockAboveMin()
Suspend()
Activated
Locked
IncorrectUnlock/A4
Suspended
Balance()/A8
Close()
End

D. Input Operation of Account-1

```
open(string p, string y, float a){
    d.temp_p = p
    d.temp_y = y
    d.temp_a = a
    m.open()
}

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

deposit(float d){
    d.temp_d = d
    m.Deposit()
    if(d.balance > 500){
        m.DepositAboveMin()
    }else{
        m.DepositBelowMin()
    }
}

balance(){
    m.Balance()
}

login(string y){
    if(y == d.userID){
        m.Login()
    }else{
        m.IncorrectLogin()
    }
}
```

```
logout(){
    m.Logout()
}

withdraw(float w){
    d.temp_w = w
    d.penalty = 20
    m.Withdraw()
    if(d.balance > 500){
        m.WithdrawAboveMin()
    }else{
        m.WithdrawPenalty()
    }
}

lock(string x){
    if(x == d.pin){
        m.Lock()
    }else{
        m.IncorrectLock()
    }
}

unlock(string x){
    if(x == d.pin){
        if(d.balance > 500){
            m. UnlockAboveMin()
        }else{
            m. UnlockBelowMin()
        }
    }else{
        m.IncorrectUnlock()
    }
}
```

E. Input Operation of Account-2

```
OPEN(int p, int y, int a){
    d.temp_p = p
    d.temp_y = y
    d.temp_a = a
    m.open()
}

PIN(int x){
    if(x == d.pin){
        m.CorrectPinAboveMin()
    }else{
        m.IncorrectPin(2)
    }
}

DEPOSIT(int d){
    d.temp_d = d
    m.Deposit()
}

WITHDRAW(int w){
    d.temp_w = w
    m.Withdraw()
    if(d.balance > 0){
        m.WithdrawAboveMin()
    }else{
        m.NoFound()
    }
}

BALANCE(){
    m.Balance()
}

LOGIN(int y){
    if(y == d.userID){
        m.Login()
    }else{
        m.IncorrectLogin()
    }
}
```

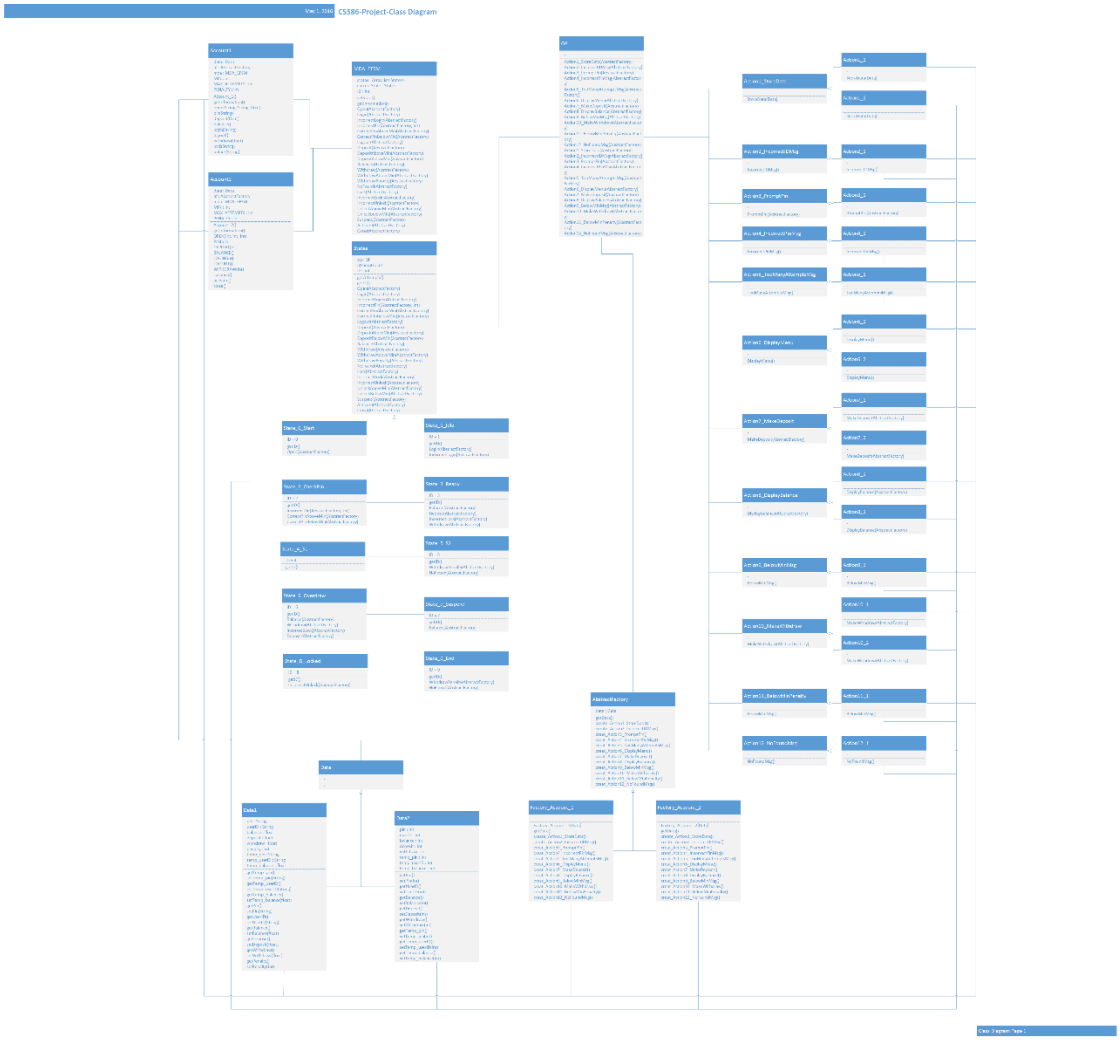
```
LOGOUT(){
    m.Logout()
}

suspend(){
    m.Suspend()
}

activate(){
    m.Activate()
}

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

2. Class Diagram



3. General Description of Each Class

There are 6 packages and 1 test driver in this program.

A. Accounts, this package contains the two account classes.
B. Actions, this package contains all the output operation and actions classes.
C. Data, this package contains all data classes.
D. Factories, this package contains all abstract factory classes.
E. MDA-EFSM, this package contains the EFSM class.
F. States, this package contains all the state classes.
G. Test Driver, this class is the test driver of the program.

Then, descript all classes by packages in general.

A. Accounts Package

1. Class Account_1

i. Responsibility
This is the class which will provide operations which belongs to account 1 for the clients.

ii. Main Attributes
1) data: this is the data object contains all the data for account 1.
2) af: this is the abstract factory object which create instances for account 1.
3) mda: this is the MDA_EFSM object which will control all the states pass.
4) MIN: this contain the minimum balance can withdraw.
5) MAX_ATTEMPTS: this is the maximum number of attempts.
6) PENALTY: this is the penalty which withdraw below MIN.

iii. Operations
1) Account_1(): this is the constructor which initializes all the attributes.
2) getInformation(): return the current state and attempts number.
3) open(String, String, float): open an account with provided string pin, string user ID and float balance.
4) pin(String): provide string pin by client.
5) deposit(float): deposit with amount provided by client.
6) balance(): check the current balance.
7) login(String): login with string pin provided by client.
8) logout(): logout from the account 1.
9) withdraw(float): withdraw with amount provided by client.
10) lock(String): locks an account with string pin.
11) unlock(String): unlocks an account with string pin.

2. Class Account_2

i. Responsibility
This is the class which will provide operations which belongs to account 2 for the clients.

ii. Main Attributes
1) data: this is the data object contains all the data for account 2.
2) af: this is the abstract factory object which create instances for account 2.
3) mda: this is the MDA_EFSM object which will control all the states pass.
4) MIN: this contain the minimum balance can withdraw.
5) MAX_ATTEMPTS: this is the maximum number of attempts.
6) PENALTY: this is the penalty which withdraw below MIN.

iii. Operations
1) Account_2(): this is the constructor which initializes all the attributes.
2) getInformation(): return the current state and attempts number.
3) OPEN(int, int, int) : open an account with provided int pin, int user ID and int balance.
4) PIN(int): provide int pin by client.
5) DEPOSIT(int): deposit with amount provided by client.
6) BALANCE(): check the current balance.
7) LOGIN(int): login with int pin provided by client.
8) LOGOUT(): logout from the account 2.
9) WITHDRAW(int): withdraw with amount provided by client.
10) suspend(): suspends an account.
11) activate(): activates a suspends account.
12) close(): an account is closed.

B. Actions Package

1. Class OP

i. Responsibility
This is the class which will provide all the actions in the MDA-EFSM diagram.

ii. Main Attributes

iii. Operations
1) Action1_StoreData(AbstractFactory): provide action1 of store data.
2) Action2_IncorrectIDMsg(AbstractFactory): provide action2 of generate the incorrect ID message.
3) Action3_PromptPin(AbstractFactory): provide action3 of generate the prompt pin message.
4) Action4_IncorrectPinMsg(AbstractFactory): provide action4

of generate the incorrect pin message.

5) Action5_TooManyAttemptsMsg(AbstractFactory): provide action5 of generate the too many attempts message.
6) Action6_DisplayMenu(AbstractFactory): provide action6 of generate the menu.
7) Action7_MakeDeposit(AbstractFactory): provide action7 of making deposit.
8) Action8_DisplayBalance(AbstractFactory): provide action8 of display the current balance.
9) Action9_BelowMinMsg(AbstractFactory): provide action9 of generate the below MIN message.
10) Action10_MakeWithdraw(AbstractFactory): provide action10 of making withdraw.
11) Action11_BelowMinPenalty(AbstractFactory): provide action11, imply the penalty.
12) Action12_NoFoundMsg(AbstractFactory): provide action12 of generate the no found message.

2. Class Action1_StoreData
   i. Responsibility
      This is the abstract super class for all the action 1 classes, all action 1 classes should implement it.
   ii. Main Attributes
   iii. Operations
      1) StoreData(Data): this is the abstract method of action 1, store data.

3. Class Action1_1
   i. Responsibility
      This is the first concrete class of action 1 for the account 1.
   ii. Main Attributes
   iii. Operations
      1) StoreData(Data): store the temp data.

4. Class Action1_2
   i. Responsibility
      This is the first concrete class of action 1 for the account 2
   ii. Main Attributes
   iii. Operations
      1) StoreData(Data): store the temp data.

5. Class Action2_IncorrectIDMsg

i. Responsibility
This is the abstract super class for all the action 2 classes, all action 2 classes should implement it.
ii. Main Attributes
iii. Operations
1) IncorrectIDMsg(): this is the abstract method of action 2, display the incorrect ID message.

6. Class Action2_1
i. Responsibility
This is the first concrete class of action 2.
ii. Main Attributes
iii. Operations
1) IncorrectIDMsg(): display the incorrect ID message.

7. Class Action3_ PromptPin
i. Responsibility
This is the abstract super class for all the action 3 classes, all action 3 classes should implement it.
ii. Main Attributes
iii. Operations
1) PromptPin(): this is the abstract method of action 3, display the prompt pin message.

8. Class Action3_1
i. Responsibility
This is the first concrete class of action 3.
ii. Main Attributes
iii. Operations
1) PromptPin(): display the prompt pin message.

9. Class Action4_IncorrectPinMsg
i. Responsibility
This is the abstract super class for all the action 4 classes, all action 4 classes should implement it.
ii. Main Attributes
iii. Operations
1) IncorrectPinMsg(): this is the abstract method of action 4, display the incorrect pin message.

10. Class Action4_1
i. Responsibility

This is the first concrete class of action 4.

ii. Main Attributes

iii. Operations

1) IncorrectPinMsg(): display the incorrect pin message.

11. Class Action5_TooManyAttemptsMsg

i. Responsibility

This is the abstract super class for all the action 5 classes, all action 5 classes should implement it.

ii. Main Attributes

iii. Operations

1) TooManyAttemptsMsg(): this is the abstract method of action 5, display the too many attempts of entering pin message.

12. Class Action5_1

i. Responsibility

This is the first concrete class of action 5.

ii. Main Attributes

iii. Operations

1) TooManyAttemptsMsg(): display the too many attempts of entering pin message.

13. Class Action6_DisplayMenu

i. Responsibility

This is the abstract super class for all the action 6 classes, all action 6 classes should implement it.

ii. Main Attributes

iii. Operations

1) DisplayMenu(): this is the abstract method of action 6, display the account operation menu.

14. Class Action6_1

i. Responsibility

This is the first concrete class of action 6 for the account 1.

ii. Main Attributes

iii. Operations

1) DisplayMenu(): display the account-1's operation menu.

15. Class Action6_2

i. Responsibility

This is the first concrete class of action 6 for the account 2

ii. Main Attributes

iii. Operations
1) DisplayMenu(): display the account-2's operation menu.

16. Class Action7_MakeDeposit
i. Responsibility
This is the abstract super class for all the action 7 classes, all action 7 classes should implement it.
ii. Main Attributes
iii. Operations
1) MakeDeposit(): this is the abstract method of action 7, making deposit.

17. Class Action7_1
i. Responsibility
This is the first concrete class of action 7 for the account 1.
ii. Main Attributes
iii. Operations
1) MakeDeposit(): making deposit for account-1.

18. Class Action7_2
i. Responsibility
This is the first concrete class of action 7 for the account 2
ii. Main Attributes
iii. Operations
1) MakeDeposit(): making deposit for account-2.

19. Class Action8_DisplayBalance
i. Responsibility
This is the abstract super class for all the action 8 classes, all action 8 classes should implement it.
ii. Main Attributes
iii. Operations
1) DisplayBalance(): this is the abstract method of action 8, display the current balance.

20. Class Action8_1
i. Responsibility
This is the first concrete class of action 8 for the account 1.
ii. Main Attributes
iii. Operations
1) DisplayBalance(): display current balance of account-1.

21. Class Action8_2
    i. Responsibility
       This is the first concrete class of action 8 for the account 2
    ii. Main Attributes
    iii. Operations
        1) DisplayBalance(): display current balance of account-2.

22. Class Action9_BelowMinMsg
    i. Responsibility
       This is the abstract super class for all the action 9 classes, all action 9 classes should implement it.
    ii. Main Attributes
    iii. Operations
        1) BelowMinMsg(): this is the abstract method of action 9, display the below MIN message.

23. Class Action9_1
    i. Responsibility
       This is the first concrete class of action 9.
    ii. Main Attributes
    iii. Operations
        1) BelowMinMsg(): display the below MIN message.
24. Class Action10_MakeWithdraw
    i. Responsibility
       This is the abstract super class for all the action 10 classes, all action 10 classes should implement it.
    ii. Main Attributes
    iii. Operations
        1) MakeWithdraw(): this is the abstract method of action 10, making withdraw.

25. Class Action10_1
    i. Responsibility
       This is the first concrete class of action 10 for the account 1.
    ii. Main Attributes
    iii. Operations
        1) MakeWithdraw(): making withdraw for account-1.

26. Class Action10_2
    i. Responsibility
       This is the first concrete class of action 10 for the account 2
    ii. Main Attributes

iii. Operations
1) MakeWithdraw(): making withdraw for account-2.

27. Class Action11_BelowMinPenalty
i. Responsibility
This is the abstract super class for all the action 11 classes, all action 11 classes should implement it.
ii. Main Attributes
iii. Operations
1) BelowMinPenalty(): this is the abstract method of action 11, when withdraw below the MIN, apply penalty on the balance.

28. Class Action11_1
i. Responsibility
This is the first concrete class of action 11 for the account 1.
ii. Main Attributes
iii. Operations
1) BelowMinPenalty(): for account 1 when withdraw below the MIN, apply penalty on the balance.

29. Class Action12_NoFoundMsg
i. Responsibility
This is the abstract super class for all the action 12 classes, all action 12 classes should implement it.
ii. Main Attributes
iii. Operations
1) NoFoundMsg(): this is the abstract method of action 10, display the no found in account message.

30. Class Action12_1
i. Responsibility
This is the first concrete class of action 12.
ii. Main Attributes
iii. Operations
1) NoFoundMsg(): display the no found in account message.

C. Data Package

1. Class Data
i. Responsibility
This is the abstract super class for all data classes.
ii. Main Attributes
iii. Operations

2. Class Data1
   i. Responsibility
      This is the data class for account 1.
   ii. Main Attributes
      1) String pin : the pin for account 1.
      2) String userID : the user ID for account 1.
      3) Float balance : the balance for account 1.
      4) Float deposit : the deposit amount for account 1.
      5) Float withdraw : the withdraw amount for account 1.
      6) Int penalty : the penalty amount for account 1.
      7) String temp_pin : the temp pin for account 1.
      8) String temp_userID : the temp user ID for account 1.
      9) Float temp_balance : the temp balance for account 1.
   iii. Operations
      1) getTemp_pin(): return temp pin value.
      2) setTemp_pin(String): set the value of temp pin.
      3) getTemp_userID(): return temp user ID value.
      4) setTemp_userID(String): set the value of temp user ID.
      5) getTemp_balance(): return temp balance value.
      6) setTemp_balance(float): set the value of temp balance.
      7) getPin(): return pin value.
      8) setPin(String): set the value of pin.
      9) getUserID(): return user ID value.
      10) setUserID(String): set the value of user ID
      11) getBalance(): return balance value.
      12) setBalance(float): set the value of balance.
      13) getDeposit(): return deposit amount value.
      14) setDeposit(float): set the value of deposit amount.
      15) getWithdraw(): return withdraw amount value.
      16) setWithdraw(float): set the value of withdraw amount.
      17) getPenalty(): return penalty amount value.
      18) setPenalty(int): set the value of penalty amount.

3. Class Data2
   i. Responsibility
      This is the data class for account 2.
   ii. Main Attributes
      1) Int pin : the pin for account 2.
      2) Int userID : the user ID for account 2.
      3) Int balance : the balance for account 2.
      4) Int deposit : the deposit amount for account 2.

5) Int withdraw : the withdraw amount for account 2.
6) Int temp_pin : the temp pin for account 2.
7) Int temp_userID : the temp user ID for account 2.
8) Int temp_balance : the temp balance for account 2.

iii. Operations

1) getTemp_pin(): return temp pin value.
2) setTemp_pin(String): set the value of temp pin.
3) getTemp_userID(): return temp user ID value.
4) setTemp_userID(String): set the value of temp user ID.
5) getTemp_balance(): return temp balance value.
6) setTemp_balance(float): set the value of temp balance.
7) getPin(): return pin value.
8) setPin(String): set the value of pin.
9) getUserID(): return user ID value.
10) setUserID(String): set the value of user ID
11) getBalance(): return balance value.
12) setBalance(float): set the value of balance.
13) getDeposit(): return deposit amount value.
14) setDeposit(float): set the value of deposit amount.
15) getWithdraw(): return withdraw amount value.
16) setWithdraw(float): set the value of withdraw amount.

D. Factories Package

1. Class AbstractFactory

i. Responsibility

This is the abstract class for all the object factory classes.

ii. Main Attributes

1) Data data: store the data object.

iii. Operations

1) getData(): abstract method to return the data object.
2) create_Action1_StoreData(): abstract method to create Action 1 object.
3) create_Action2_IncorrectIDMsg(): abstract method to create Action 2 object.
4) creat_Action3_PromptPin(): abstract method to create Action 3 object.
5) creat_Action4_IncorrectPinMsg(): abstract method to create Action 4 object.
6) creat_Action5_TooManyAttemptsMsg(): abstract method to create Action 5 object.
7) creat_Action6_DisplayMenu(): abstract method to create Action 6 object.

8) creat_Action7_MakeDeposit(): abstract method to create Action 7 object.
9) creat_Action8_DisplayBalance(): abstract method to create Action 8 object.
10) creat_Action9_BelowMinMsg(): abstract method to create Action 9 object.
11) creat_Action10_MakeWithdraw(): abstract method to create Action 10 object.
12) creat_Action11_BelowMinPenalty(): abstract method to create Action 11 object.
13) creat_Action12_NoFoundMsg(): abstract method to create Action 12 object.

2. Class Factory_Account_1
   i. Responsibility
   This is the factory class that create objects for account 1.
   ii. Main Attributes
   iii. Operations
   1) getData(): return the data object.
   2) create_Action1_StoreData(): create object of Action 1 for account 1 and return it.
   3) create_Action2_IncorrectIDMsg(): create object of Action 2 for account 1 and return it.
   4) creat_Action3_PromptPin(): create object of Action 3 for account 1 and return it.
   5) creat_Action4_IncorrectPinMsg(): create object of Action 4 for account 1 and return it.
   6) creat_Action5_TooManyAttemptsMsg(): create object of Action 5 for account 1 and return it.
   7) creat_Action6_DisplayMenu(): create object of Action 6 for account 1 and return it.
   8) creat_Action7_MakeDeposit(): create object of Action 7 for account 1 and return it.
   9) creat_Action8_DisplayBalance(): create object of Action 8 for account 1 and return it.
   10) creat_Action9_BelowMinMsg(): create object of Action 9 for account 1 and return it.
   11) creat_Action10_MakeWithdraw(): create object of Action 10 for account 1 and return it.
   12) creat_Action11_BelowMinPenalty(): create object of Action 11 for account 1 and return it.
   13) creat_Action12_NoFoundMsg(): create object of Action 12 for

account 1 and return it.

3. Class Factory_Account_2
   i. Responsibility
   This is the factory class that create objects for account 2.
   ii. Main Attributes
   iii. Operations
   1) getData(): return the data object.
   2) create_Action1_StoreData(): create object of Action 1 for account 2 and return it.
   3) create_Action2_IncorrectIDMsg(): create object of Action 2 for account 2 and return it.
   4) creat_Action3_PromptPin(): create object of Action 3 for account 2 and return it.
   5) creat_Action4_IncorrectPinMsg(): create object of Action 4 for account 2 and return it.
   6) creat_Action5_TooManyAttemptsMsg(): create object of Action 5 for account 2 and return it.
   7) creat_Action6_DisplayMenu(): create object of Action 6 for account 2 and return it.
   8) creat_Action7_MakeDeposit(): create object of Action 7 for account 2 and return it.
   9) creat_Action8_DisplayBalance(): create object of Action 8 for account 2 and return it.
   10) creat_Action9_BelowMinMsg(): create object of Action 9 for account 2 and return it.
   11) creat_Action10_MakeWithdraw(): create object of Action 10 for account 2 and return it.
   12) creat_Action11_BelowMinPenalty(): create object of Action 11 for account 2 and return it.
   13) creat_Action12_NoFoundMsg(): create object of Action 12 for account 2 and return it.

E. MDA-EFSM Package

1. Class MDA_EFSM
   i. Responsibility
   This is the class of MDA-EFSM which will control all the state transfer based on the diagram and call the relation actions.
   ii. Main Attributes
   1) states : ArrayList<States>: store all the 9 states.
   2) currentState : States: store the current state.
   3) ID : int: store the current state's ID.

iii. Operations

1) setStart(): initialize all the attributes.
2) getInformation(): return current state's ID
3) Open(AbstractFactory): method of event open.
4) Login(AbstractFactory): method of event login.
5) IncorrectLogin(AbstractFactory): method of event incorrect login.
6) IncorrectPin(AbstractFactory, int): method of event incorrect pin.
7) CorrectPinAboveMin(AbstractFactory): method of event correct pin above min.
8) CorrectPinBelowMin(AbstractFactory): method of event correct pin below min.
9) Logout(AbstractFactory): method of event logout.
10) Deposit(AbstractFactory): method of event deposit.
11) DepositAboveMin(AbstractFactory): method of event deposit above min.
12) DepositBelowMin(AbstractFactory): method of event deposit below min.
13) Balance(AbstractFactory): method of event check balance.
14) Withdraw(AbstractFactory): method of event making withdraw.
15) WithdrawAboveMin(AbstractFactory): method of event making withdraw above min.
16) WithdrawPenalty(AbstractFactory): method of event making withdraw and apply the penalty.
17) NoFound(AbstractFactory): method of event no found in account.
18) Lock(AbstractFactory): method of event locks the account.
19) IncorrectLock(AbstractFactory): method of event incorrect lock.
20) IncorrectUnlock(AbstractFactory): method of event incorrect unlock.
21) UnlockAboveMin(AbstractFactory): method of event unlocks the account with balance above min.
22) UnlockBelowMin(AbstractFactory): method of event unlocks the account with balance above min.
23) Suspend(AbstractFactory): method of event suspends the account.
24) Activate(AbstractFactory): method of event activates a suspended account.
25) Close(AbstractFactory): method of event closes an account.

F. States Package

1. Class States

i. Responsibility

This is the abstract super class for all state classes.

ii. Main Attributes

1) op : OP: this is the object of op class, control the actions.
2) attempts : int: this is the current attempt times.
3) ID : int: this is the state's ID.

iii. Operations

1) getAttempts(): return the current attempt times.
2) getID(): return the state's ID.
3) Open(AbstractFactory): super method of event open.
4) Login(AbstractFactory): super method of event login.
5) IncorrectLogin(AbstractFactory): super method of event incorrect login.
6) IncorrectPin(AbstractFactory, int): super method of event incorrect pin.
7) CorrectPinAboveMin(AbstractFactory): super method of event correct pin above min.
8) CorrectPinBelowMin(AbstractFactory): super method of event correct pin below min.
9) Logout(AbstractFactory): super method of event logout.
10) Deposit(AbstractFactory): super method of event deposit.
11) DepositAboveMin(AbstractFactory): super method of event deposit above min.
12) DepositBelowMin(AbstractFactory): super method of event deposit below min.
13) Balance(AbstractFactory): super method of event check balance.
14) Withdraw(AbstractFactory): super method of event making withdraw.
15) WithdrawAboveMin(AbstractFactory): super method of event making withdraw above min.
16) WithdrawPenalty(AbstractFactory): super method of event making withdraw and apply the penalty.
17) NoFound(AbstractFactory): super method of event no found in account.
18) Lock(AbstractFactory): super method of event locks the account.
19) IncorrectLock(AbstractFactory): super method of event incorrect lock.
20) IncorrectUnlock(AbstractFactory): super method of event

incorrect unlock.

21) UnlockAboveMin(AbstractFactory): super method of event unlocks the account with balance above min.
22) UnlockBelowMin(AbstractFactory): super method of event unlocks the account with balance above min.
23) Suspend(AbstractFactory): super method of event suspends the account.
24) Activate(AbstractFactory): super method of event activates a suspended account.
25) Close(AbstractFactory): super method of event closes an account.

2. Class State_0_Start
   i. Responsibility
   This is the state 0, start state in the MDA-EFSM diagram.
   ii. Main Attributes
   1) ID : int: this is the start state's ID which is 0.
   iii. Operations
   1) getID(): return the start state's ID which is 0.
   2) Open(AbstractFactory): override the super method in the State class, which will call action 1.

3. Class State_1_Idle
   i. Responsibility
   This is the state 1, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
   1) ID : int: this is the start state's ID which is 1.
   iii. Operations
   1) getID(): return the start state's ID which is 1.
   2) Login(AbstractFactory): override the super method in the State class, which will call action 3 and reset the attempts to 0.
   3) IncorrectLogin(AbstractFactory): override the super method in the State class, which will call action 2.

4. Class State_2_CheckPin
   i. Responsibility
   This is the state 2, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
   1) ID : int: this is the start state's ID which is 2.
   iii. Operations
   1) getID(): return the start state's ID which is 2.
   2) IncorrectPin(AbstractFactory, int): override the super method in the State class, which will call action 4 and 5 if attempts

equals to max attempts, or only call action 4 if below the max attempts.
3) CorrectPinAboveMin(AbstractFactory): override the super method in the State class, which will call action 6.
4) CorrectPinBelowMin(AbstractFactory): override the super method in the State class, which will call action 6.

5. Class State_3_Ready
   i. Responsibility
   This is the state 3, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
   1) ID : int: this is the start state's ID which is 3.
   iii. Operations
   1) getID(): return the start state's ID which is 3.
   2) Balance(AbstractFactory): override the super method in the State class, which will call action 8.
   3) Deposit(AbstractFactory): override the super method in the State class, which will call action 7.
   4) IncorrectLock(AbstractFactory): override the super method in the State class, which will call action 4.
   5) Withdraw(AbstractFactory): override the super method in the State class, which will call action 10.

6. Class State_4_S1
   i. Responsibility
   This is the state 4, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
   1) ID : int: this is the start state's ID which is 4.
   iii. Operations
   1) getID(): return the start state's ID which is 4.

7. Class State_5_S2
   i. Responsibility
   This is the state 5, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
   1) ID : int: this is the start state's ID which is 5.
   iii. Operations
   1) getID(): return the start state's ID which is 5.
   2) WithdrawPenalty(AbstractFactory): override the super method in the State class, which will call action 11.
   3) NoFound(AbstractFactory): override the super method in the State class, which will call action 12.

8. Class State_6_Overdraw
   i. Responsibility
      This is the state 6, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
      1) ID : int: this is the start state's ID which is 6.
   iii. Operations
      1) getID(): return the start state's ID which is 6.
      2) Balance(AbstractFactory): override the super method in the State class, which will call action 8.
      3) Withdraw(AbstractFactory): override the super method in the State class, which will call action 9.
      4) IncorrectLock(AbstractFactory): override the super method in the State class, which will call action 4.
      5) Deposit(AbstractFactory): override the super method in the State class, which will call action 7.
9. Class State_7_Suspend
   i. Responsibility
      This is the state 7, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
      1) ID : int: this is the start state's ID which is 7.
   iii. Operations
      1) getID(): return the start state's ID which is 7.
      2) Balance(AbstractFactory): override the super method in the State class, which will call action 8.

10. Class State_8_Locked
   i. Responsibility
      This is the state 8, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
      1) ID : int: this is the start state's ID which is 8.
   iii. Operations
      1) getID(): return the start state's ID which is 8.
      2) IncorrectUnlock(AbstractFactory): override the super method in the State class, which will call action 4.

11. Class State_9_End
   i. Responsibility
      This is the state 9, idle state in the MDA-EFSM diagram.
   ii. Main Attributes
      2) ID : int: this is the start state's ID which is 9.
   iii. Operations

3) getID(): return the start state's ID which is 9.

G. Test Driver Class

1. Class Test Driver

i. Responsibility

This is the client class, which will test all the operations that provided by the account 1 and account 2.

ii. Main Attributes

iii. Operations

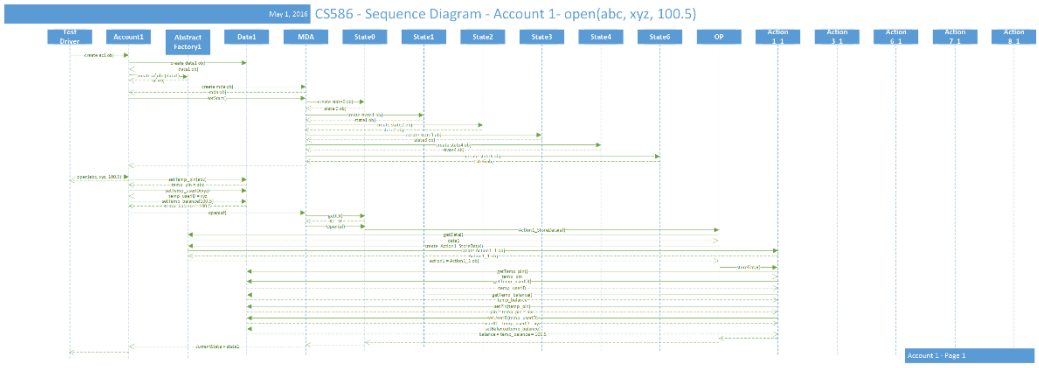
1) printTitle(): print the title of this class.
2) printAccountSelection(): print the operation of account selection.
3) printAccount_1_Menu(): print the operations provided by the account 1.
4) printAccount_1_Selection(): print the selected operation of account 1.
5) printAccount_2_Menu(): print the operations provided by the account 2.
6) printAccount_2_Selection(): print the selected operation of account 2
7) Account_1_Operations(Account_1, String): call the operations that selected by the user in the account 1.
8) Account_2_Operations(Account_2, String) : call the operations that selected by the user in the account 2.
9) main(String[]): get the user inputs and display all the information.

4. Sequence Diagram

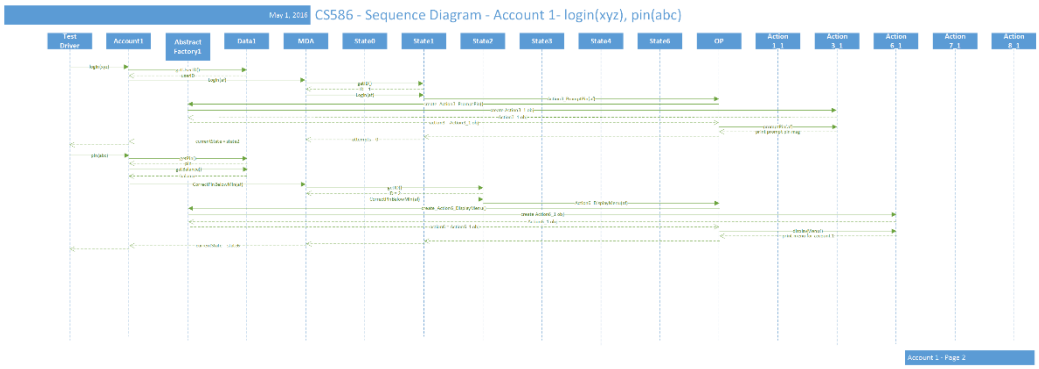
A. Sequence Diagram 1

This sequence diagram shows how to make deposit in account 1 component. The following sequence of operations is issued: open(abc, xyz, 100.5), login(xyz), pin(abc), deposit(400), balance(), logout()

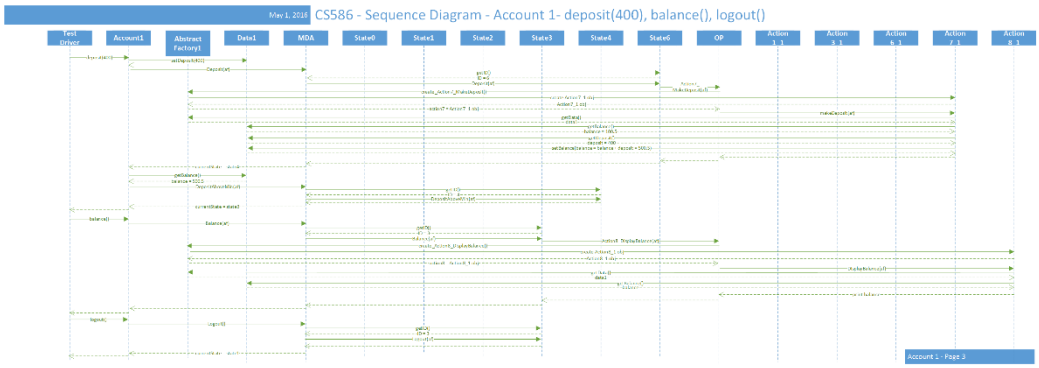
Sequence Diagram 1-1: open(abc, xyz, 100.5)



Sequence Diagram 1-2: login(xyz), pin(abc)



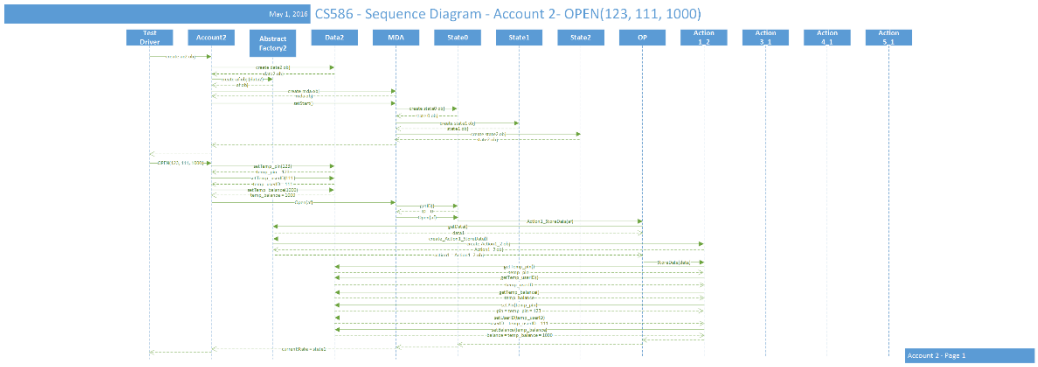
Sequence Diagram 1-3: deposit(400), balance(), logout()



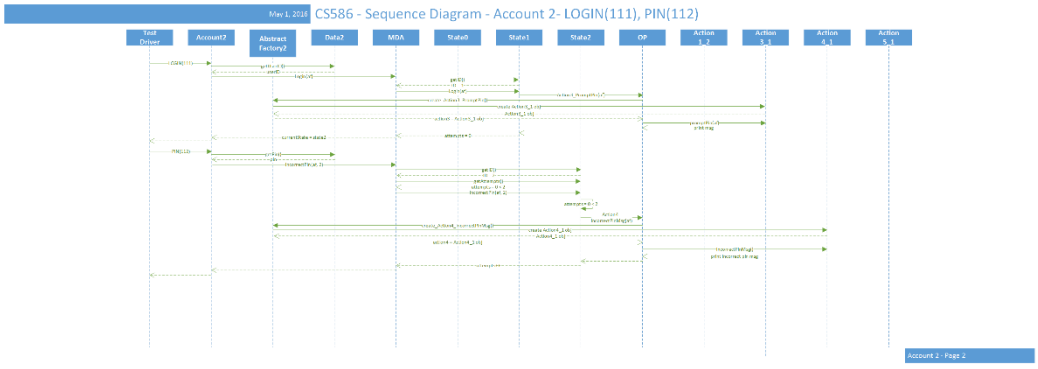
B. Sequence Diagram 2

This sequence diagram shows how an incorrect pin is entered three times in the account 2 component. The following sequence of operations is issued: OPEN(123, 111, 1000), LOGIN(111), PIN(112), PIN(222), PIN(333)

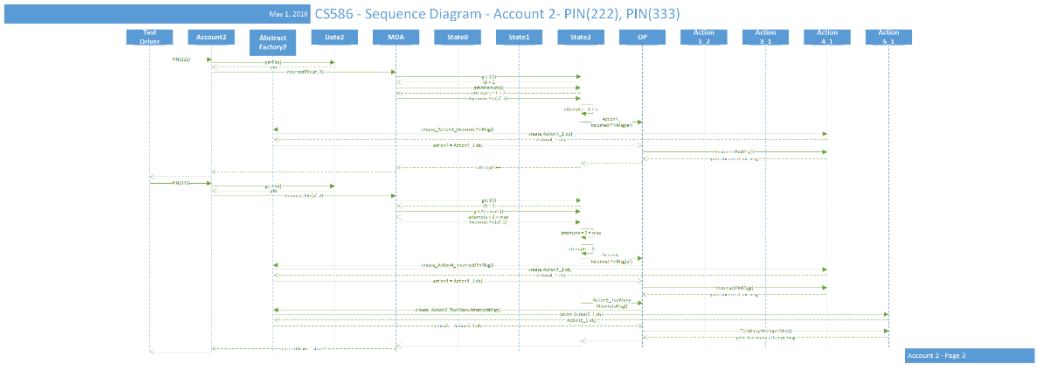
Sequence Diagram 2-1: OPEN(123, 111, 1000)



Sequence Diagram 2-2: LOGIN(111), PIN(112)



Sequence Diagram 2-3: PIN(222), PIN(333)



5. Source Code of Each Class

There are 6 packages and 1 test driver in this program.
Accounts, this package contains the two account classes.
Actions, this package contains all the output operation and actions classes.
Data, this package contains all data classes.
Factories, this package contains all abstract factory classes.
MDA-EFSM, this package contains the EFSM class.
States, this package contains all the state classes.
Test Driver, this class is the test driver of the program.
And some of them imply the three design patterns, which are state pattern, strategy pattern and abstract factory pattern.

A. State Pattern

The state pattern is implemented in the states package and MDA-EFSM package, here we use the centralized version, the MDA-EFSM will control all the state transfers and the states will call the related actions based on the MDA-EFSM diagram.

1. MDA-EFSM Package

1. Class MDA_EFSM

```
package Jian.MDA_EFSM;

import java.util.ArrayList;

import Jian.Factories.AbstractFactory;
import Jian.States.State_0_Start;
import Jian.States.State_1_Idle;
import Jian.States.State_2_CheckPin;
import Jian.States.State_3_Ready;
import Jian.States.State_4_S1;
import Jian.States.State_5_S2;
import Jian.States.State_6_Overdraw;
import Jian.States.State_7_Suspend;
import Jian.States.State_8_Locked;
import Jian.States.State_9_End;
import Jian.States.States;
```

```
public class MDA_EFSM {
    ArrayList<States> states;
    States currentState;
    int ID = 0;

    public void setStart() {
        states = new ArrayList<States>();
        states.add(new State_0_Start());
        states.add(new State_1_Idle());
        states.add(new State_2_CheckPin());
        states.add(new State_3_Ready());
        states.add(new State_4_S1());
        states.add(new State_5_S2());
        states.add(new State_6_Overdraw());
        states.add(new State_7_Suspend());
        states.add(new State_8_Locked());
        states.add(new State_9_End());
        currentState = states.get(0);
        ID = 0;
    }

    public void getInformation() {
        ID = currentState.getID();
        System.out.println("Current state : " + ID);
        int a = currentState.getAttempts();
        System.out.println("Attempts = " + a);
    }

    public void Open(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 0) {
            currentState.Open(af);
            currentState = states.get(1);
        }
    }

    public void Login(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 1) {
            currentState.Login(af);
            currentState = states.get(2);
        }
```

```
}

public void IncorrectLogin(AbstractFactory af) {
    ID = currentState.getID();
    if (ID == 1) {
        currentState.IncorrectLogin(af);
    }
}

public void IncorrectPin(AbstractFactory af, int max) {
    ID = currentState.getID();
    if (ID == 2) {
        int attempts = currentState.getAttempts();
        if (attempts == max) {
            currentState.IncorrectPin(af, max);
            currentState = states.get(1);
        } else if (attempts < max) {
            currentState.IncorrectPin(af, max);
        }
    }
}

public void CorrectPinAboveMin(AbstractFactory af) {
    ID = currentState.getID();
    if (ID == 2) {
        currentState.CorrectPinAboveMin(af);
        currentState = states.get(3);
    }
}

public void CorrectPinBelowMin(AbstractFactory af) {
    ID = currentState.getID();
    if (ID == 2) {
        currentState.CorrectPinBelowMin(af);
        currentState = states.get(6);
    }
}

public void Logout(AbstractFactory af) {
    ID = currentState.getID();
    if (ID == 2 || ID == 3 || ID == 6) {
        currentState.Logout(af);
```

```
            currentState = states.get(1);
        }

    }

    public void Deposit(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 6) {
            currentState.Deposit(af);
            currentState = states.get(4);
        } else if (ID == 3) {
            currentState.Deposit(af);
        }

    }

    public void DepositAboveMin(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 4) {
            currentState.DepositAboveMin(af);
            currentState = states.get(3);
        }
    }

    public void DepositBelowMin(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 4) {
            currentState.DepositBelowMin(af);
            currentState = states.get(6);
        }
    }

    public void Balance(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 3 || ID == 6 || ID == 7) {
            currentState.Balance(af);
        }
    }

    public void Withdraw(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 3) {
```

```
            currentState.Withdraw(af);
            currentState = states.get(5);
        } else if (ID == 6) {
            currentState.Withdraw(af);
        }
    }

    public void WithdrawAboveMin(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 5) {
            currentState.WithdrawAboveMin(af);
            currentState = states.get(3);
        }
    }

    public void WithdrawPenalty(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 5) {
            currentState.WithdrawPenalty(af);
            currentState = states.get(6);
        }
    }

    public void NoFound(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 5) {
            currentState.NoFound(af);
            currentState = states.get(3);
        }
    }

    public void Lock(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 3 || ID == 6) {
            currentState.Lock(af);
            currentState = states.get(8);
        }
    }

    public void IncorrectLock(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 3 || ID == 6) {
```

```
            currentState.IncorrectLock(af);
        }
    }

    public void IncorrectUnlock(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 8) {
            currentState.IncorrectUnlock(af);
        }
    }

    public void UnlockAboveMin(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 8) {
            currentState.UnlockAboveMin(af);
            currentState = states.get(3);
        }
    }

    public void UnlockBelowMin(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 8) {
            currentState.UnlockBelowMin(af);
            currentState = states.get(6);
        }
    }

    public void Suspend(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 3) {
            currentState.Suspend(af);
            currentState = states.get(7);
        }
    }

    public void Activate(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 7) {
            currentState.Activate(af);
            currentState = states.get(3);
        }
    }
```

```
    public void Close(AbstractFactory af) {
        ID = currentState.getID();
        if (ID == 7) {
            currentState.Close(af);
            currentState = states.get(9);
        }
    }
}
```

2. States Package

1. Class States

```
package Jian.States;

import Jian.Actions.OP;
import Jian.Factories.AbstractFactory;

public abstract class States {
	OP op = new OP();
	int attempts = 0;
	int ID = 0;

	public int getAttempts() {
		return attempts;
	}

	public int getID() {
		return ID;
	}

	public void Open(AbstractFactory af) {
	}

	public void Login(AbstractFactory af) {
	}

	public void IncorrectLogin(AbstractFactory af) {
	}

	public void IncorrectPin(AbstractFactory af, int max) {
	}

	public void CorrectPinAboveMin(AbstractFactory af) {
	}

	public void CorrectPinBelowMin(AbstractFactory af) {
	}

	public void Logout(AbstractFactory af) {
	}
```

```
public void Deposit(AbstractFactory af) {
}

public void DepositAboveMin(AbstractFactory af) {
}

public void DepositBelowMin(AbstractFactory af) {
}

public void Balance(AbstractFactory af) {
}

public void Withdraw(AbstractFactory af) {
}

public void WithdrawAboveMin(AbstractFactory af) {
}

public void WithdrawPenalty(AbstractFactory af) {
}

public void NoFound(AbstractFactory af) {
}

public void Lock(AbstractFactory af) {
}

public void IncorrectLock(AbstractFactory af) {
}

public void IncorrectUnlock(AbstractFactory af) {
}

public void UnlockAboveMin(AbstractFactory af) {
}

public void UnlockBelowMin(AbstractFactory af) {
}

public void Suspend(AbstractFactory af) {
}
```

```
    public void Activate(AbstractFactory af) {
    }

    public void Close(AbstractFactory af) {
    }
}
```

2. Class State_0_Start

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_0_Start extends States{
    @Override
    public int getID() {
        ID = 0;
        return ID;
    }

    @Override
    public void Open(AbstractFactory af) {
        op.Action1_StoreData(af);
    }
}
```

3. Class State_1_Idle

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_1_Idle extends States{
    @Override
    public int getID() {
        ID = 1;
        return ID;
    }

    @Override
```

```
    public void Login(AbstractFactory af) {
        op.Action3_PromptPin(af);
        attempts = 0;
    }

    @Override
    public void IncorrectLogin(AbstractFactory af) {
        op.Action2_IncorrectIDMsg(af);
    }
}
```

4. Class State_2_CheckPin

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_2_CheckPin extends States {
    @Override
    public int getID() {
        ID = 2;
        return ID;
    }

    @Override
    public void IncorrectPin(AbstractFactory af, int max) {
        if (attempts == max) {
            attempts = 0;
            op.Action4_IncorrectPinMsg(af);
            op.Action5_TooManyAttemptsMsg(af);
        } else if (attempts < max) {
            op.Action4_IncorrectPinMsg(af);
            attempts++;
        }
    }

    @Override
    public void CorrectPinAboveMin(AbstractFactory af) {
        op.Action6_DisplayMenu(af);
    }
```

```
    @Override
    public void CorrectPinBelowMin(AbstractFactory af) {
        op.Action6_DisplayMenu(af);
    }
}
```

5. Class State_3_Ready

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_3_Ready extends States {
    @Override
    public int getID() {
        ID = 3;
        return ID;
    }

    @Override
    public void Balance(AbstractFactory af) {
        op.Action8_DisplayBalance(af);
    }

    @Override
    public void Deposit(AbstractFactory af) {
        op.Action7_MakeDeposit(af);
    }

    @Override
    public void IncorrectLock(AbstractFactory af) {
        op.Action4_IncorrectPinMsg(af);
    }

    @Override
    public void Withdraw(AbstractFactory af) {
        op.Action10_MakeWithdraw(af);
    }
}
```

6. Class State_4_S1

```
package Jian.States;

public class State_4_S1 extends States {
    @Override
    public int getID() {
        ID = 4;
        return ID;
    }
}
```

7. Class State_5_S2

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_5_S2 extends States {
    @Override
    public int getID() {
        ID = 5;
        return ID;
    }

    @Override
    public void WithdrawPenalty(AbstractFactory af) {
        op.Action11_BelowMinPenalty(af);
    }

    @Override
    public void NoFound(AbstractFactory af) {
        op.Action12_NoFoundMsg(af);
    }

}
```

8. Class State_6_Overdraw

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_6_Overdraw extends States {
    @Override
    public int getID() {
        ID = 6;
        return ID;
    }

    @Override
    public void Balance(AbstractFactory af) {
        op.Action8_DisplayBalance(af);
    }

    @Override
    public void Withdraw(AbstractFactory af) {
        op.Action9_BelowMinMsg(af);
    }

    @Override
    public void IncorrectLock(AbstractFactory af) {
        op.Action4_IncorrectPinMsg(af);
    }

    @Override
    public void Deposit(AbstractFactory af) {
        op.Action7_MakeDeposit(af);
    }
}
```

9. Class State_7_Suspend

```
package Jian.States;
import Jian.Factories.AbstractFactory;
public class State_7_Suspend extends States {
    @Override
    public int getID() {
```

```
            ID = 7;
            return ID;
        }

        @Override
        public void Balance(AbstractFactory af) {
            op.Action8_DisplayBalance(af);
        }
}
```

10. Class State_8_Locked

```
package Jian.States;

import Jian.Factories.AbstractFactory;

public class State_8_Locked extends States {
        @Override
        public int getID() {
            ID = 8;
            return ID;
        }
        @Override
        public void IncorrectUnlock(AbstractFactory af) {
            op.Action4_IncorrectPinMsg(af);
        }
}
```

11. Class State_9_End

```
package Jian.States;
public class State_9_End extends States{
        @Override
        public int getID() {
            ID = 9;
            return ID;
        }
}
```

B. Strategy Pattern

The strategy pattern is implemented in the actions package, all actions are invoked by the OP class, and each action has a super abstract class and one or two concrete class of the implementation.

1. Actions Package

1. Class OP

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Factories.AbstractFactory;

/**
 * This is the class which will provide all the actions in the MDA-EFSM diagram.
 */

public class OP {

    public void Action1_StoreData(AbstractFactory af) {
        Data data = af.getData();
        Action1_StoreData action1 = af.create_Action1_StoreData();
        action1.StoreData(data);
    }

    public void Action2_IncorrectIDMsg(AbstractFactory af) {
        Action2_IncorrectIDMsg action2 = af.create_Action2_IncorrectIDMsg();
        action2.IncorrectIDMsg();
    }

    public void Action3_PromptPin(AbstractFactory af) {
        Action3_PromptPin action3 = af.creat_Action3_PromptPin();
        action3.PromptPin(af);
    }

    public void Action4_IncorrectPinMsg(AbstractFactory af) {
        Action4_IncorrectPinMsg action4 = af.creat_Action4_IncorrectPinMsg();
        action4.IncorrectPinMsg();
    }
```

```
    public void Action5_TooManyAttemptsMsg(AbstractFactory af) {
        Action5_TooManyAttemptsMsg action5 = af.creat_Action5_TooManyAttemptsMsg();
        action5.TooManyAttemptsMsg();
    }

    public void Action6_DisplayMenu(AbstractFactory af) {
        Action6_DisplayMenu action6 = af.creat_Action6_DisplayMenu();
        action6.DisplayMenu();
    }

    public void Action7_MakeDeposit(AbstractFactory af) {
        Action7_MakeDeposit action7 = af.creat_Action7_MakeDeposit();
        action7.MakeDeposit(af);
    }

    public void Action8_DisplayBalance(AbstractFactory af) {
        Action8_DisplayBalance action8 = af.creat_Action8_DisplayBalance();
        action8.DisplayBalance(af);
    }

    public void Action9_BelowMinMsg(AbstractFactory af) {
        Action9_BelowMinMsg action9 = af.creat_Action9_BelowMinMsg();
        action9.BelowMinMsg();
    }

    public void Action10_MakeWithdraw(AbstractFactory af) {
        Action10_MakeWithdraw action10 = af.creat_Action10_MakeWithdraw();
        action10.MakeWithdraw(af);
    }

    public void Action11_BelowMinPenalty(AbstractFactory af) {
        Action11_BelowMinPenalty action11 = af.creat_Action11_BelowMinPenalty();
        action11.BelowMinPenalty(af);
    }

    public void Action12_NoFoundMsg(AbstractFactory af) {
        Action12_NoFoundMsg action12 = af.creat_Action12_NoFoundMsg();
        action12.NoFoundMsg();
    }
}
```

2. Class Action1_StoreData

```
package Jian.Actions;

import Jian.Data.Data;

public interface Action1_StoreData {
    public void StoreData(Data data);
}
```

3. Class Action1_1

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data1;

public class Action1_1 implements Action1_StoreData{
    @Override
    public void StoreData(Data data) {
        String temp_pin = ((Data1)data).getTemp_pin();
        String temp_userID = ((Data1)data).getTemp_userID();
        float temp_balance = ((Data1)data).getTemp_balance();

        ((Data1)data).setPin(temp_pin);
        ((Data1)data).setUserID(temp_userID);
        ((Data1)data).setBalance(temp_balance);

        System.out.println("Action 1 Finish!");
    }
}
```

4. Class Action1_2

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data2;

public class Action1_2 implements Action1_StoreData{
```

```
    @Override
    public void StoreData(Data data) {
        int temp_pin = ((Data2)data).getTemp_pin();
        int temp_userID = ((Data2)data).getTemp_userID();
        int temp_balance = ((Data2)data).getTemp_balance();

        ((Data2)data).setPin(temp_pin);
        ((Data2)data).setUserID(temp_userID);
        ((Data2)data).setBalance(temp_balance);

        System.out.println("Action 1 Finish!");
    }
}
```

5. Class Action2_IncorrectIDMsg

```
package Jian.Actions;

public interface Action2_IncorrectIDMsg {
    public void IncorrectIDMsg();
}
```

6. Class Action2_1

```
package Jian.Actions;

public class Action2_1 implements Action2_IncorrectIDMsg{
    @Override
    public void IncorrectIDMsg() {
        System.out.println("INCORRECT ID MSG");

        System.out.println("Action 2 Finish!");
    }
}
```

7. Class Action3_PromptPin

```
package Jian.Actions;
```

```
import Jian.Factories.AbstractFactory;

public interface Action3_PromptPin {
    public void PromptPin(AbstractFactory af);
}
```

8. Class Action3_1

```
package Jian.Actions;

import Jian.Factories.AbstractFactory;

public class Action3_1 implements Action3_PromptPin {

    @Override
    public void PromptPin(AbstractFactory af) {
        System.out.println("PROMPT TO ENTER PIN");

        System.out.println("Action 3 Finish!");
    }
}
```

9. Class Action4_IncorrectPinMsg

```
package Jian.Actions;

public interface Action4_IncorrectPinMsg {
    public void IncorrectPinMsg();
}
```

10. Class Action4_1

```
package Jian.Actions;

public class Action4_1 implements Action4_IncorrectPinMsg {

    @Override
    public void IncorrectPinMsg() {
```

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

            System.out.println("Action 4 Finish!");
        }
```

11. Class Action5_TooManyAttemptsMsg

```
package Jian.Actions;

public interface Action5_TooManyAttemptsMsg {
    public void TooManyAttemptsMsg();
}
```

12. Class Action5_1

```
package Jian.Actions;

public class Action5_1 implements Action5_TooManyAttemptsMsg {

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

        System.out.println("Action 5 Finish!");
    }
}
```

13. Class Action6_DisplayMenu

```
package Jian.Actions;

public interface Action6_DisplayMenu {
    public void DisplayMenu();
}
```

14. Class Action6_1

```
package Jian.Actions;

public class Action6_1 implements Action6_DisplayMenu {

    @Override
    public void DisplayMenu() {
        System.out.println("DISPLAY MENU OF ACCOUNT 1");
        System.out.println("\t balance");
        System.out.println("\t deposit");
        System.out.println("\t withdraw");
        System.out.println("\t lock");
        System.out.println("\t unlock");

        System.out.println("Action 6 Finish!");
    }
}
```

15. Class Action6_2

```
package Jian.Actions;

public class Action6_2 implements Action6_DisplayMenu {

    @Override
    public void DisplayMenu() {
        System.out.println("DISPLAY MENU OF ACCOUNT 2");
        System.out.println("\t BALANCE");
        System.out.println("\t DEPOSIT");
        System.out.println("\t WITHDRAW");
        System.out.println("\t suspend");
        System.out.println("\t activate");
        System.out.println("\t close");

        System.out.println("Action 6 Finish!");
    }

}
```

16. Class Action7_MakeDeposit

```
package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action7_MakeDeposit {
    public void MakeDeposit(AbstractFactory af);
}
```

17. Class Action7_1

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data1;
import Jian.Factories.AbstractFactory;

public class Action7_1 implements Action7_MakeDeposit {

    @Override
    public void MakeDeposit(AbstractFactory af) {
        Data data = af.getData();
        float balance = ((Data1)data).getBalance();
        float deposit = ((Data1)data).getDeposit();
        balance = balance + deposit;
        ((Data1)data).setBalance(balance);

        System.out.println("Action 7 Finish!");
    }

}
```

18. Class Action7_2

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data2;
import Jian.Factories.AbstractFactory;
```

```
public class Action7_2 implements Action7_MakeDeposit {

    @Override
    public void MakeDeposit(AbstractFactory af) {
        Data data = af.getData();
        int balance = ((Data2)data).getBalance();
        int deposit = ((Data2)data).getDeposit();
        balance = balance + deposit;
        ((Data2)data).setBalance(balance);

        System.out.println("Action 7 Finish!");
    }

}
```

19. Class Action8_DisplayBalance

```
package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action8_DisplayBalance {
    public void DisplayBalance(AbstractFactory af);
}
```

20. Class Action8_1

```
package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action8_DisplayBalance {
    public void DisplayBalance(AbstractFactory af);
}
```

21. Class Action8_2

```
package Jian.Actions;
```

```
import Jian.Data.Data;
import Jian.Data.Data2;
import Jian.Factories.AbstractFactory;

public class Action8_2 implements Action8_DisplayBalance {

    @Override
    public void DisplayBalance(AbstractFactory af) {
        Data data = af.getData();
        int balance = ((Data2)data).getBalance();

        System.out.println("DISPLAY BALANCE : " + balance);

        System.out.println("Action 8 Finish!");

    }

}
```

22. Class Action9_BelowMinMsg

```
package Jian.Actions;

public interface Action9_BelowMinMsg {
    public void BelowMinMsg();
}
```

23. Class Action9_1

```
package Jian.Actions;

public class Action9_1 implements Action9_BelowMinMsg {
    @Override
    public void BelowMinMsg() {
        System.out.println("BELOW MIN MSG");

        System.out.println("Action 9 Finish!");
    }
}
```

24. Class Action10_MakeWithdraw

```
package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action10_MakeWithdraw {
    public void MakeWithdraw(AbstractFactory af);
}
```

25. Class Action10_1

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data1;
import Jian.Factories.AbstractFactory;

public class Action10_1 implements Action10_MakeWithdraw{

    @Override
    public void MakeWithdraw(AbstractFactory af) {
        Data data = af.getData();
        float balance = ((Data1)data).getBalance();
        float withdraw = ((Data1)data).getWithdraw();
        balance = balance - withdraw;
        ((Data1)data).setBalance(balance);

        System.out.println("Action 10 Finish!");
    }

}
```

26. Class Action10_2

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data2;
import Jian.Factories.AbstractFactory;
```

```
public class Action10_2 implements Action10_MakeWithdraw{

    @Override
    public void MakeWithdraw(AbstractFactory af) {
        Data data = af.getData();
        int balance = ((Data2)data).getBalance();
        int withdraw = ((Data2)data).getWithdraw();
        balance = balance - withdraw;
        ((Data2)data).setBalance(balance);

        System.out.println("Action 10 Finish!");
    }

}
```

27. Class Action11_BelowMinPenalty

```
package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action11_BelowMinPenalty {
    public void BelowMinPenalty(AbstractFactory af);
}
```

28. Class Action11_1

```
package Jian.Actions;

import Jian.Data.Data;
import Jian.Data.Data1;
import Jian.Factories.AbstractFactory;

public class Action11_1 implements Action11_BelowMinPenalty {

    @Override
    public void BelowMinPenalty(AbstractFactory af) {
        Data data = af.getData();
        float balance = ((Data1)data).getBalance();
        int penalty = ((Data1)data).getPenalty();
```

```
            balance = balance - penalty;
            ((Data1)data).setBalance(balance);

            System.out.println("Action 11 Finish!");
        }
}
```

29. Class Action12_NoFoundMsg

```
package Jian.Actions;

public interface Action12_NoFoundMsg {
    public void NoFoundMsg();
}
```

30. Class Action12_1

```
package Jian.Actions;

public class Action12_1 implements Action12_NoFoundMsg {

    @Override
    public void NoFoundMsg() {
        System.out.println("NO FOUND MSG");

        System.out.println("Action 12 Finish!");
    }

}
```

C. Abstract Factory Pattern

The abstract factory pattern is implemented in the factories package, the concrete factory 1 and factory 2 are responsible for creating the right objects for account 1 and account 2.

1. Factories Package

1. Class AbstractFactory

```
package Jian.Factories;

import Jian.Actions.*;
import Jian.Data.Data;

public abstract class AbstractFactory {
    Data data;

    public abstract Data getData();

    public abstract Action1_StoreData create_Action1_StoreData();

    public abstract Action2_IncorrectIDMsg create_Action2_IncorrectIDMsg();

    public abstract Action3_PromptPin creat_Action3_PromptPin();

    public abstract Action4_IncorrectPinMsg creat_Action4_IncorrectPinMsg();

    public abstract Action5_TooManyAttemptsMsg
    creat_Action5_TooManyAttemptsMsg();

    public abstract Action6_DisplayMenu creat_Action6_DisplayMenu();

    public abstract Action7_MakeDeposit creat_Action7_MakeDeposit();

    public abstract Action8_DisplayBalance creat_Action8_DisplayBalance();

    public abstract Action9_BelowMinMsg creat_Action9_BelowMinMsg();

    public abstract Action10_MakeWithdraw creat_Action10_MakeWithdraw();

    public abstract Action11_BelowMinPenalty creat_Action11_BelowMinPenalty();
```

```
    public abstract Action12_NoFoundMsg creat_Action12_NoFoundMsg();

}
```

2. Class Factory_Account_1

```
package Jian.Factories;
import Jian.Actions.*;
import Jian.Data.Data;

public class Factory_Account_1 extends AbstractFactory{

    public Factory_Account_1(Data data) {
        this.data = data;
    }

    @Override
    public Data getData() {
        return data;
    }

    @Override
    public Action1_StoreData create_Action1_StoreData() {
        return new Action1_1();
    }

    @Override
    public Action2_IncorrectIDMsg create_Action2_IncorrectIDMsg() {
        return new Action2_1();
    }

    @Override
    public Action3_PromptPin creat_Action3_PromptPin() {
        return new Action3_1();
    }

    @Override
    public Action4_IncorrectPinMsg creat_Action4_IncorrectPinMsg() {
        return new Action4_1();
    }
```

```
    @Override
    public Action5_TooManyAttemptsMsg creat_Action5_TooManyAttemptsMsg() {
        return new Action5_1();
    }

    @Override
    public Action6_DisplayMenu creat_Action6_DisplayMenu() {
        return new Action6_1();
    }

    @Override
    public Action7_MakeDeposit creat_Action7_MakeDeposit() {
        return new Action7_1();
    }

    @Override
    public Action8_DisplayBalance creat_Action8_DisplayBalance() {
        return new Action8_1();
    }

    @Override
    public Action9_BelowMinMsg creat_Action9_BelowMinMsg() {
        return new Action9_1();
    }

    @Override
    public Action10_MakeWithdraw creat_Action10_MakeWithdraw() {
        return new Action10_1();
    }

    @Override
    public Action11_BelowMinPenalty creat_Action11_BelowMinPenalty() {
        return new Action11_1();
    }

    @Override
    public Action12_NoFoundMsg creat_Action12_NoFoundMsg() {
        return new Action12_1();
    }
}
```

3. Class Factory_Account_2

```
package Jian.Factories;
import Jian.Actions.*;
import Jian.Data.Data;

public class Factory_Account_2 extends AbstractFactory{

    public Factory_Account_2(Data data) {
        this.data = data;
    }

    @Override
    public Data getData() {
        return data;
    }

    @Override
    public Action1_StoreData create_Action1_StoreData() {
        return new Action1_2();
    }

    @Override
    public Action2_IncorrectIDMsg create_Action2_IncorrectIDMsg() {
        return new Action2_1();
    }

    @Override
    public Action3_PromptPin creat_Action3_PromptPin() {
        return new Action3_1();
    }

    @Override
    public Action4_IncorrectPinMsg creat_Action4_IncorrectPinMsg() {
        return new Action4_1();
    }

    @Override
    public Action5_TooManyAttemptsMsg creat_Action5_TooManyAttemptsMsg() {
        return new Action5_1();
    }
```

```
    @Override
    public Action6_DisplayMenu creat_Action6_DisplayMenu() {
        return new Action6_2();
    }

    @Override
    public Action7_MakeDeposit creat_Action7_MakeDeposit() {
        return new Action7_2();
    }

    @Override
    public Action8_DisplayBalance creat_Action8_DisplayBalance() {
        return new Action8_2();
    }

    @Override
    public Action9_BelowMinMsg creat_Action9_BelowMinMsg() {
        return new Action9_1();
    }

    @Override
    public Action10_MakeWithdraw creat_Action10_MakeWithdraw() {
        return new Action10_2();
    }

    @Override
    public Action11_BelowMinPenalty creat_Action11_BelowMinPenalty() {
        return new Action11_1();
    }

    @Override
    public Action12_NoFoundMsg creat_Action12_NoFoundMsg() {
        return new Action12_1();
    }
}
```

D. Other Source Code

Here are the remaining classes' source code.

1. Account Package

1. Class Account_1

```
package Jian.Accounts;

import Jian.Data.Data;
import Jian.Data.Data1;
import Jian.Factories.AbstractFactory;
import Jian.Factories.Factory_Account_1;
import Jian.MDA_EFSM.MDA_EFSM;;

/**
 * This is the class which will provide operations which belongs to account 1
 * for the clients.
 */
public class Account_1 {
	Data data;
	AbstractFactory af;
	MDA_EFSM mda;

	final int MIN = 500;
	final int MAX_ATTEMPTS = 3;
	final int PENALTY = 20;

	// This is the constructor.
	// Initialize data, abstract factory and MDA-EFSM class.
	public Account_1() {
		data = new Data1();
		af = new Factory_Account_1(data);
		mda = new MDA_EFSM();
		mda.setStart();
	}

	// return the current state and attempts number.
	public void getInformation() {
		mda.getInformation();
	}
```

```
// Open an account-1
// p is pin, y is user ID, a is balance
public void open(String p, String y, float a) {
    ((Data1) data).setTemp_pin(p);
    ((Data1) data).setTemp_userID(y);
    ((Data1) data).setTemp_balance(a);

    mda.Open(af);
}

// Provide pin number
public void pin(String x) {
    String pin = ((Data1) data).getPin();
    float balance = ((Data1) data).getBalance();

    if (x.equals(pin)) {
        if (balance > MIN) {
            mda.CorrectPinAboveMin(af);
        } else {
            mda.CorrectPinBelowMin(af);
        }
    } else {
        mda.IncorrectPin(af, MAX_ATTEMPTS);
    }
}

// Deposit amount d
public void deposit(float d) {
    ((Data1) data).setDeposit(d);

    mda.Deposit(af);

    float balance = ((Data1) data).getBalance();

    if (balance > MIN) {
        mda.DepositAboveMin(af);
    } else {
        mda.DepositBelowMin(af);
    }
}
```

```
// Display the current balance
public void balance() {
    mda.Balance(af);
}

// Login where y is an user ID
public void login(String y) {
    String userID = ((Data1) data).getUserID();

    if (y.equals(userID)) {
        mda.Login(af);
    } else {
        mda.IncorrectLogin(af);
    }
}

// Logout from account-1
public void logout() {
    mda.Logout(af);
}

// Withdraw with amount w
public void withdraw(float w) {
    ((Data1) data).setWithdraw(w);
    ((Data1) data).setPenalty(PENALTY);

    mda.Withdraw(af);

    float balance = ((Data1) data).getBalance();

    if (balance > MIN) {
        mda.WithdrawAboveMin(af);
    } else {
        mda.WithdrawPenalty(af);
    }
}

// Locks account, x is a pin
public void lock(String x) {
    String pin = ((Data1) data).getPin();

    if (x.equals(pin)) {
```

```
                mda.Lock(af);
            } else {
                mda.IncorrectLock(af);
            }
        }

        // Unlock an account, x is a pin
        public void unlock(String x) {
            String pin = ((Data1) data).getPin();
            float balance = ((Data1) data).getBalance();

            if (x.equals(pin)) {
                if (balance > MIN) {
                    mda.UnlockAboveMin(af);
                } else {
                    mda.UnlockBelowMin(af);
                }
            } else {
                mda.IncorrectUnlock(af);
            }
        }
}
```

2. Class Account_2

```
package Jian.Accounts;

import Jian.Data.Data;
import Jian.Data.Data2;
import Jian.Factories.AbstractFactory;
import Jian.Factories.Factory_Account_2;
import Jian.MDA_EFSM.MDA_EFSM;;

/**
 * This is the class which will provide operations which belongs to account 2
 * for the clients.
 */

public class Account_2 {
    Data data;
    AbstractFactory af;
    MDA_EFSM mda;
```

```
final int MIN = 500;
final int MAX_ATTEMPTS = 2;
final int PENALTY = 20;

// This is the constructor.
// Initialize data, abstract factory and MDA-EFSM class.
public Account_2() {
	data = new Data2();
	af = new Factory_Account_2(data);
	mda = new MDA_EFSM();
	mda.setStart();
}

// return the current state and attempts number.
public void getInformation() {
	mda.getInformation();
}

// Open an account-2
// p is pin, y is user ID, a is balance
public void OPEN(int p, int y, int a) {
	((Data2) data).setTemp_pin(p);
	((Data2) data).setTemp_userID(y);
	((Data2) data).setTemp_balance(a);

	mda.Open(af);
}

// Provide pin number
public void PIN(int x) {
	int pin = ((Data2) data).getPin();

	if (x == pin) {
		mda.CorrectPinAboveMin(af);

	} else {
		mda.IncorrectPin(af, MAX_ATTEMPTS);
	}
}

// Deposit amount d
```

```
public void DEPOSIT(int d) {
    ((Data2) data).setDeposit(d);

    mda.Deposit(af);
}

// Display the current balance
public void BALANCE() {
    mda.Balance(af);
}

// Login where y is an user ID
public void LOGIN(int y) {
    int userID = ((Data2) data).getUserID();

    if (userID == y) {
        mda.Login(af);
    } else {
        mda.IncorrectLogin(af);
    }
}

// Logout from account-1
public void LOGOUT() {
    mda.Logout(af);
}

// Withdraw with amount w
public void WITHDRAW(int w) {
    ((Data2) data).setWithdraw(w);

    mda.Withdraw(af);

    int balance = ((Data2) data).getBalance();

    if (balance > 0) {
        mda.WithdrawAboveMin(af);
    } else {
        mda.NoFound(af);
    }
}
```

```
    // suspends an account.
    public void suspend() {
        mda.Suspend(af);
    }

    // activates a suspends account
    public void activate() {
        mda.Activate(af);
    }

    // an account is closed.
    public void close() {
        mda.Close(af);
    }
}
```

3. Data Package

1. Class Data

```
package Jian.Data;

public abstract class Data {
}
```

2. Class Data1

```
package Jian.Data;

public class Data1 extends Data{
	String pin;
	String userID;
	float balance;
	float deposit;
	float withdraw;
	int penalty;

	String temp_pin;
	String temp_userID;
	float temp_balance;


	public String getTemp_pin() {
		return temp_pin;
	}
	public void setTemp_pin(String temp_pin) {
		this.temp_pin = temp_pin;
	}

	public String getTemp_userID() {
		return temp_userID;
	}
	public void setTemp_userID(String temp_userID) {
		this.temp_userID = temp_userID;
	}
	public float getTemp_balance() {
		return temp_balance;
```

```
    }
    public void setTemp_balance(float temp_balance) {
        this.temp_balance = temp_balance;
    }
    public String getPin() {
        return pin;
    }
    public void setPin(String pin) {
        this.pin = pin;
    }
    public String getUserID() {
        return userID;
    }
    public void setUserID(String userID) {
        this.userID = userID;
    }
    public float getBalance() {
        return balance;
    }
    public void setBalance(float balance) {
        this.balance = balance;
    }
    public float getDeposit() {
        return deposit;
    }
    public void setDeposit(float deposit) {
        this.deposit = deposit;
    }
    public float getWithdraw() {
        return withdraw;
    }
    public void setWithdraw(float withdraw) {
        this.withdraw = withdraw;
    }
    public int getPenalty() {
        return penalty;
    }
    public void setPenalty(int penalty) {
        this.penalty = penalty;
    }
}
```

3. Class Data2

```
package Jian.Data;

public class Data2 extends Data{
	int pin;
	int userID;
	int balance;
	int deposit;
	int withdraw;
	int temp_pin;
	int temp_userID;
	int temp_balance;

	public int getPin() {
		return pin;
	}
	public void setPin(int pin) {
		this.pin = pin;
	}
	public int getUserID() {
		return userID;
	}
	public void setUserID(int userID) {
		this.userID = userID;
	}
	public int getBalance() {
		return balance;
	}
	public void setBalance(int balance) {
		this.balance = balance;
	}
	public int getDeposit() {
		return deposit;
	}
	public void setDeposit(int deposit) {
		this.deposit = deposit;
	}
	public int getWithdraw() {
		return withdraw;
	}
	public void setWithdraw(int withdraw) {
```

```
        this.withdraw = withdraw;
    }
    public int getTemp_pin() {
        return temp_pin;
    }
    public void setTemp_pin(int temp_pin) {
        this.temp_pin = temp_pin;
    }
    public int getTemp_userID() {
        return temp_userID;
    }
    public void setTemp_userID(int temp_userID) {
        this.temp_userID = temp_userID;
    }
    public int getTemp_balance() {
        return temp_balance;
    }
    public void setTemp_balance(int temp_balance) {
        this.temp_balance = temp_balance;
    }

}
```

4. Test Driver Class

1. Class TestDriver

```
package Jian;

import java.util.Scanner;

import Jian.Accounts.Account_1;
import Jian.Accounts.Account_2;

public class TestDriver {
    public static void printTitle() {
        final String TITLE = "\tCS 586 PROJECT\n\t" + "Test Driver\n\t" + "JIAN ZHANG\n\t" + "A20327380\n\n" + "Enter Anything to Continue";
        System.out.println(TITLE);
    }

    public static void printAccountSelection() {
        final String AC_SELECTION = "Please select account number\n" + "1: Account-1\n" + "2: Account-2\n\n" + "Enter 1 or 2 : ";
        System.out.print(AC_SELECTION);
    }

    public static void printAccount_1_Menu() {
        final String AC_1_MENU = "\n\t\t\tAccount-1\n\t\t" + "Menu of Operations\n\t" + "0. open(string p, string y, float a)\n\t" + "1. login(string y)\n\t" + "2. pin(string x)\n\t" + "3. deposit(float d)\n\t" + "4. withdraw(float w)\n\t" + "5. balance()\n\t" + "6. logout()\n\t" + "7. lock(string x)\n\t" + "8. unlock(string x)\n\t" + "i. information\n\t" + "q. quit\n\n";
        System.out.println(AC_1_MENU);
    }

    public static void printAccount_1_Selection() {
        final String AC_1_SELECTION = "\n\t\tACCOUNT-1 EXECUTION\n\n\t" + "Select Operations:\n" + "0: open,  1: login,  2: pin,  3: deposit, 4: withdraw,\n" + "5: balance, 6: logout, 7: lock, 8: unlock,  i: info,  q. quit\n" + "Please Select : ";
        System.out.print(AC_1_SELECTION);
    }

    public static void printAccount_2_Menu() {
```

```
        final String AC_2_MENU = "\n\t\t\tAccount-2\n\t\t" + "Menu of
        Operations\n\t" + "0. OPEN(int p, int y, int a)\n\t" + "1. LOGIN(int y)\n\t" +
        "2. PIN(int x)\n\t" + "3. DEPOSIT(int d)\n\t" + "4. WITHDRAW(int w)\n\t" +
        "5. BALANCE()\n\t" + "6. LOGOUT()\n\t" + "7. suspend()\n\t" + "8.
        activate()\n\t" + "9. close()\n\t" + "i. information\n\t" + "q. quit\n\n\n";
        System.out.println(AC_2_MENU);
    }

    public static void printAccount_2_Selection() {
        final String AC_2_SELECTION = "\n\t\tACCOUNT-2 EXECUTION\n\n\t" +
        "Select Operations:\n" + "0: OPEN,  1: LOGIN,  2: PIN,  3:
        DEPOSIT,  4: WITHDRAW, 5: BALANCE, \n" + "6: LOGOUT, 7: suspend,
        8: activate, 9: close,  i: info,  q. quit\n" + "Please Select : ";
        System.out.print(AC_2_SELECTION);
    }

    public static void Account_1_Operations(Account_1 ac1, String op) {
        try {
            @SuppressWarnings("resource")
            Scanner scanner = new Scanner(System.in);
            switch (op) {
            case "0":
                System.out.println("\n\t\t open operation start");
                System.out.println("This operation will open an account where p is a
                pin, y is an user ID, " + "and a is a balance.");
                System.out.println("Please enter p (String): ");
                String p = scanner.next();
                System.out.println("Please enter y (String): ");
                String y = scanner.next();
                System.out.println("Please enter a (float): ");
                float a = scanner.nextFloat();
                ac1.open(p, y, a);
                System.out.println("\n\t\t open operation finish");
                break;

            case "1":
                System.out.println("\n\t\t login operation start");
                System.out.println("This operation will login where y is an user ID");
                System.out.println("Please enter y (String): ");
                String y1 = scanner.next();
                ac1.login(y1);
                System.out.println("\n\t\t login operation finish");
```

```
                break;

            case "2":
                System.out.println("\n\t\t pin operation start");
                System.out.println("This operation will provide pin x.");
                System.out.println("Please enter x (String): ");
                String x = scanner.next();
                System.out.println("\n\t\t pin operation finish");
                ac1.pin(x);
                break;

            case "3":
                System.out.println("\n\t\t deposit operation start");
                System.out.println("This operation will provide deposit amount d.");
                System.out.println("Please enter d (float): ");
                float d = scanner.nextFloat();
                ac1.deposit(d);
                System.out.println("\n\t\t deposit operation finish");
                break;

            case "4":
                System.out.println("\n\t\t withdraw operation start");
                System.out.println("This operation will provide withdraw amount
w.");
                System.out.println("Please enter w (float): ");
                float w = scanner.nextFloat();
                ac1.withdraw(w);
                System.out.println("\n\t\t withdraw operation finish");
                break;

            case "5":
                System.out.println("\n\t\t balance operation start");
               System.out.println("This operation will display the current balance.");
                ac1.balance();
                System.out.println("\n\t\t balance operation finish");
                break;

            case "6":
                System.out.println("\n\t\t logout operation start");
                System.out.println("This operation will logout from account.");
                ac1.logout();
                System.out.println("\n\t\t logout operation finish");
```

```
                    break;

                case "7":
                    System.out.println("\n\t\t lock operation start");
                    System.out.println("This operation will locks an account where x is a
pin.");
                    System.out.println("Please enter x (String): ");
                    String x1 = scanner.next();
                    ac1.lock(x1);
                    System.out.println("\n\t\t lock operation finish");
                    break;
                case "8":
                    System.out.println("\n\t\t unlock operation start");
                    System.out.println("This operation will unlock an account where x is
a pin.");
                    System.out.println("Please enter x (String): ");
                    String x2 = scanner.next();
                    ac1.unlock(x2);
                    System.out.println("\n\t\t unlock operation finish");
                    break;

                case "i":
                    System.out.println("\n\t\t information operation start");
                    System.out.println("This operation will provide the information.");
                    ac1.getInformation();
                    System.out.println("\n\t\t information operation finish");
                    break;

                case "q":
                    System.out.println("\n\t\t quit operation start");
                    System.out.println("This operation will quit the program.");
                    System.exit(0);
                    System.out.println("\n\t\t quit operation finish");
                    break;

                default:
                    System.out.println("Please enter correct operation number.");
                    break;

                }
            } catch (Exception e) {
                System.out.println("Please check format.");
```

```
    }

}

public static void Account_2_Operations(Account_2 ac, String op) {
    try {
        @SuppressWarnings("resource")
        Scanner scanner = new Scanner(System.in);
        switch (op) {
        case "0":
            System.out.println("\n\t\t OPEN operation start");
            System.out.println("This operation will open an account where p is a
            pin, y is an user ID, " + "and a is a balance.");
            System.out.println("Please enter p (int): ");
            int p = scanner.nextInt();
            System.out.println("Please enter y (int): ");
            int y = scanner.nextInt();
            System.out.println("Please enter a (int): ");
            int a = scanner.nextInt();
            ac.OPEN(p, y, a);
            System.out.println("\n\t\t OPEN operation finish");
            break;

        case "1":
            System.out.println("\n\t\t LOGIN operation start");
            System.out.println("This operation will login where y is an user ID");
            System.out.println("Please enter y (int): ");
            int y1 = scanner.nextInt();
            ac.LOGIN(y1);
            System.out.println("\n\t\t LOGIN operation finish");
            break;

        case "2":
            System.out.println("\n\t\t PIN operation start");
            System.out.println("This operation will provide pin x.");
            System.out.println("Please enter x (int): ");
            int x = scanner.nextInt();
            ac.PIN(x);
            System.out.println("\n\t\t PIN operation finish");
            break;

        case "3":
```

```
                System.out.println("\n\t\t DEPOSIT operation start");
                System.out.println("This operation will provide deposit amount d.");
                System.out.println("Please enter d (int): ");
                int d = scanner.nextInt();
                ac.DEPOSIT(d);
                System.out.println("\n\t\t DEPOSIT operation finish");
                break;

            case "4":
                System.out.println("\n\t\t WITHDRAW operation start");
                System.out.println("This operation will provide withdraw amount
w.");
                System.out.println("Please enter w (int): ");
                int w = scanner.nextInt();
                ac.WITHDRAW(w);
                System.out.println("\n\t\t WITHDRAW operation finish");
                break;

            case "5":
                System.out.println("\n\t\t BALANCE operation start");
               System.out.println("This operation will display the current balance.");
                ac.BALANCE();
                System.out.println("\n\t\t BALANCE operation finish");
                break;

            case "6":
                System.out.println("\n\t\t LOGOUT operation start");
                System.out.println("This operation will logout from account.");
                ac.LOGOUT();
                System.out.println("\n\t\t LOGOUT operation finish");
                break;

            case "7":
                System.out.println("\n\t\t suspend operation start");
                System.out.println("This operation will suspend an account.");
                ac.suspend();
                System.out.println("\n\t\t suspend operation finish");
                break;

            case "8":
                System.out.println("\n\t\t activate operation start");
                System.out.println("This operation will activate an account.");
```

```
                ac.activate();
                System.out.println("\n\t\t activate operation finish");
                break;

            case "9":
                System.out.println("\n\t\t close operation start");
                System.out.println("This operation will close an account.");
                ac.close();
                System.out.println("\n\t\t close operation finish");
                break;

            case "i":
                System.out.println("\n\t\t information operation start");
                System.out.println("provide the information.");
                ac.getInformation();
                System.out.println("\n\t\t information operation finish");
                break;

            case "q":
                System.out.println("\n\t\t quit operation start");
                System.out.println("quit the program.");
                System.exit(0);
                System.out.println("\n\t\t quit operation start");
                break;

            default:
                System.out.println("Please enter correct operation number.");
                break;
            }
        } catch (Exception e) {
            System.out.println("Please check format.");
        }

    }

    public static void main(String[] args) {
        while (true) {
            try {
                @SuppressWarnings("resource")
                Scanner scanner = new Scanner(System.in);
                printTitle();
                scanner.next();
```

```
                printAccountSelection();
                int num = scanner.nextInt();
                if (num == 1) {
                    Account_1 account_1 = new Account_1();
                    printAccount_1_Menu();
                    while (true) {
                        printAccount_1_Selection();
                        String n = scanner.next();
                        Account_1_Operations(account_1, n);
                    }
                }

                if (num == 2) {
                    Account_2 account_2 = new Account_2();
                    printAccount_2_Menu();
                    while (true) {
                        printAccount_2_Selection();
                        String n = scanner.next();
                        Account_2_Operations(account_2, n);
                    }
                }
                System.out.println("\nPlese enter 1 or 2.\n\n");
            } catch (Exception e) {
                System.out.println("\nPlese check your entering.\n\n");
            }
        }
    }
}
```