**CS 586 Project Report**

**Jian Zhang**

**A20327380**

**5/1/2016**

Index

[**1.** **MDA-EFSM model** - 3 -](#_Toc449868580)

[**A.** **List of Events:** - 3 -](#_Toc449868581)

[**B.** **List of Actions:** - 4 -](#_Toc449868582)

[**C.** **MDA-EFSM** - 5 -](#_Toc449868583)

[**D.** **Input Operation of Account-1** - 6 -](#_Toc449868584)

[**E.** **Input Operation of Account-2** - 8 -](#_Toc449868585)

[**2.** **Class Diagram** - 10 -](#_Toc449868586)

[**3.** **General Description of Each Class** - 11 -](#_Toc449868587)

[**A.** **Accounts Package** - 11 -](#_Toc449868588)

[**B.** **Actions Package** - 12 -](#_Toc449868589)

[**C.** **Data Package** - 18 -](#_Toc449868590)

[**D.** **Factories Package** - 20 -](#_Toc449868591)

[**E.** **MDA-EFSM Package** - 22 -](#_Toc449868592)

[**F.** **States Package** - 24 -](#_Toc449868593)

[**G.** **Test Driver Class** - 28 -](#_Toc449868594)

[**4.** **Sequence Diagram** - 29 -](#_Toc449868595)

[**A.** **Sequence Diagram 1** - 29 -](#_Toc449868596)

[**B.** **Sequence Diagram 2** - 30 -](#_Toc449868597)

[**5.** **Source Code of Each Class** - 31 -](#_Toc449868598)

[**A. State Pattern** - 31 -](#_Toc449868599)

[**B. Strategy Pattern** - 46 -](#_Toc449868600)

[**C. Abstract Factory Pattern** - 59 -](#_Toc449868601)

[**D. Other Source Code** - 64 -](#_Toc449868602)

1. **MDA-EFSM model**
2. **List of Events:**
3. Open()
4. Login()
5. IncorrectLogin()
6. IncorrectPin(int max)
7. CorrectPinAboveMin()
8. CorrectPinBelowMin()
9. Logout()
10. Deposit()
11. DepositAboveMin()
12. DepositBelowMin()
13. Balance()
14. Withdraw()
15. WithdrawAboveMin()
16. WithdrawPenalty()
17. NoFound()
18. Lock()
19. IncorrectLock()
20. IncorrectUnlock()
21. UnlockAboveMin()
22. UnlockBelowMin()
23. Suspend()
24. Activate()
25. Close()
26. **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.

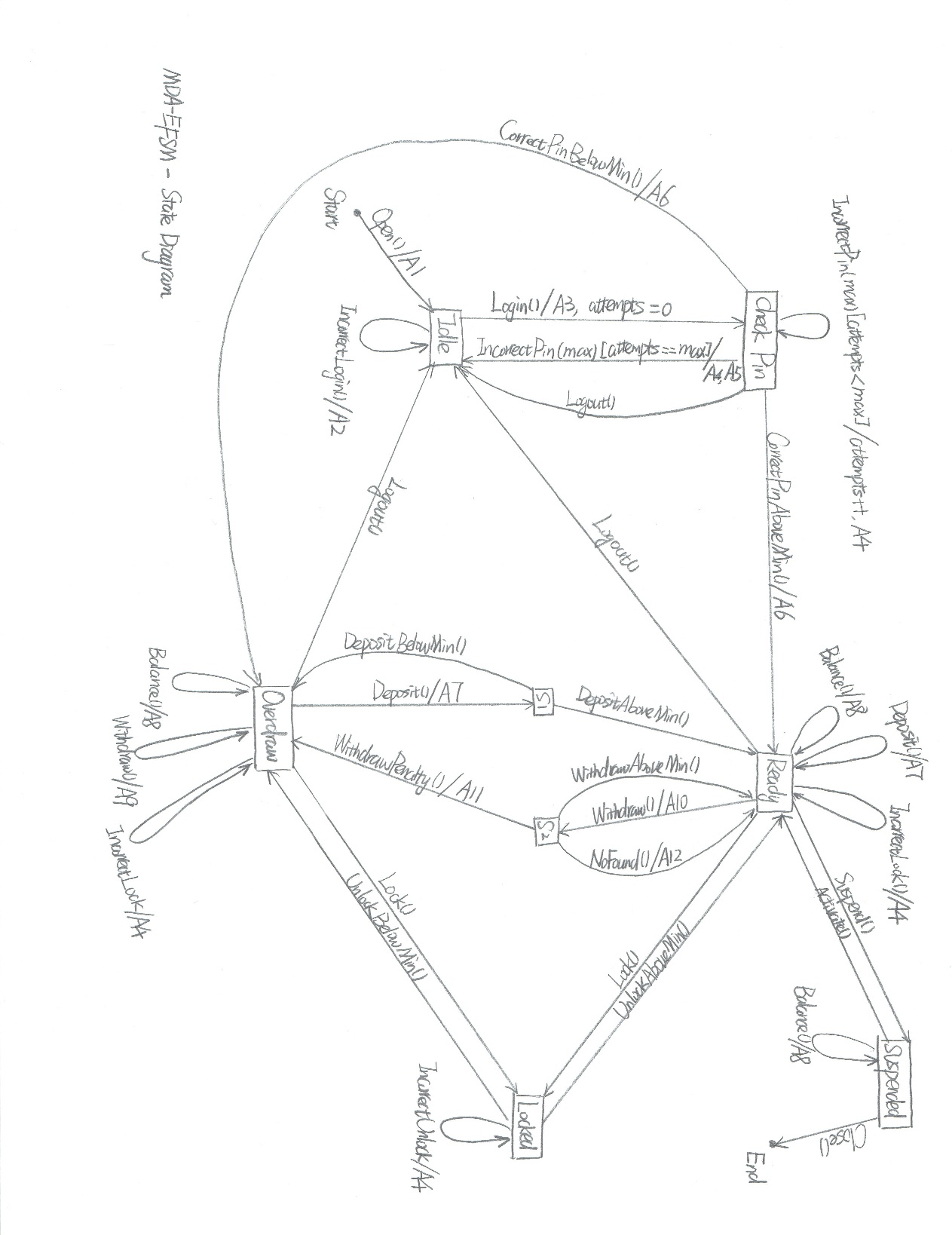
A11:

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.

1. **MDA-EFSM**

****

1. **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()

}

}

1. **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()

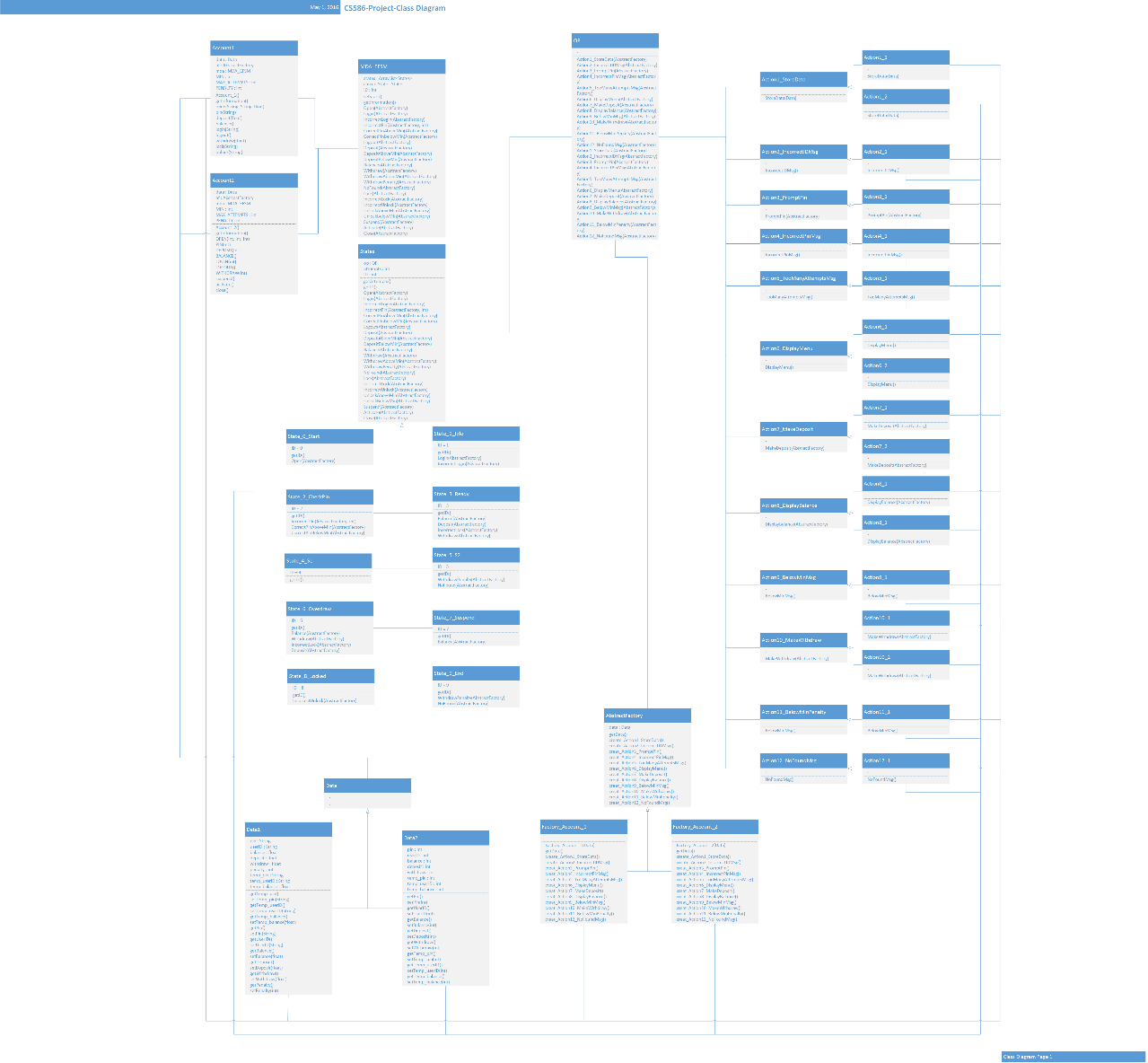
}

close(){

m.Close()

}

1. **Class Diagram**

****

1. **General Description of Each Class**

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

1. Accounts, this package contains the two account classes.
2. Actions, this package contains all the output operation and actions classes.
3. Data, this package contains all data classes.
4. Factories, this package contains all abstract factory classes.
5. MDA-EFSM, this package contains the EFSM class.
6. States, this package contains all the state classes.
7. Test Driver, this class is the test driver of the program.

Then, descript all classes by packages in general.

1. **Accounts Package**
   1. Class Account\_1
      1. Responsibility

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

* + 1. 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.
   * 1. Operations
7. Account\_1(): this is the constructor which initializes all the attributes.
8. getInformation(): return the current state and attempts number.
9. open(String, String, float): open an account with provided string pin, string user ID and float balance.
10. pin(String): provide string pin by client.
11. deposit(float): deposit with amount provided by client.
12. balance(): check the current balance.
13. login(String): login with string pin provided by client.
14. logout(): logout from the account 1.
15. withdraw(float): withdraw with amount provided by client.
16. lock(String): locks an account with string pin.
17. unlock(String): unlocks an account with string pin.
    1. Class Account\_2
       1. Responsibility

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

* + 1. 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.
   * 1. Operations
7. Account\_2(): this is the constructor which initializes all the attributes.
8. getInformation(): return the current state and attempts number.
9. OPEN(int, int, int) : open an account with provided int pin, int user ID and int balance.
10. PIN(int): provide int pin by client.
11. DEPOSIT(int): deposit with amount provided by client.
12. BALANCE(): check the current balance.
13. LOGIN(int): login with int pin provided by client.
14. LOGOUT(): logout from the account 2.
15. WITHDRAW(int): withdraw with amount provided by client.
16. suspend(): suspends an account.
17. activate(): activates a suspends account.
18. close(): an account is closed.
19. **Actions Package**
    1. Class OP
       1. Responsibility

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

* + 1. Main Attributes
    2. 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.
    1. Class Action1\_StoreData
       1. Responsibility

This is the abstract super class for all the action 1 classes, all action 1 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. StoreData(Data): this is the abstract method of action 1, store data.
   1. Class Action1\_1
      1. Responsibility

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

* + 1. Main Attributes
    2. Operations

1. StoreData(Data): store the temp data.
   1. Class Action1\_2
      1. Responsibility

This is the first concrete class of action 1 for the account 2

* + 1. Main Attributes
    2. Operations

1. StoreData(Data): store the temp data.
   1. Class Action2\_IncorrectIDMsg
      1. Responsibility

This is the abstract super class for all the action 2 classes, all action 2 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. IncorrectIDMsg(): this is the abstract method of action 2, display the incorrect ID message.
   1. Class Action2\_1
      1. Responsibility

This is the first concrete class of action 2.

* + 1. Main Attributes
    2. Operations

1. IncorrectIDMsg(): display the incorrect ID message.
   1. Class Action3\_ PromptPin
      1. Responsibility

This is the abstract super class for all the action 3 classes, all action 3 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. PromptPin(): this is the abstract method of action 3, display the prompt pin message.
   1. Class Action3\_1
      1. Responsibility

This is the first concrete class of action 3.

* + 1. Main Attributes
    2. Operations

1. PromptPin(): display the prompt pin message.
   1. Class Action4\_IncorrectPinMsg
      1. Responsibility

This is the abstract super class for all the action 4 classes, all action 4 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. IncorrectPinMsg(): this is the abstract method of action 4, display the incorrect pin message.
   1. Class Action4\_1
      1. Responsibility

This is the first concrete class of action 4.

* + 1. Main Attributes
    2. Operations

1. IncorrectPinMsg(): display the incorrect pin message.
   1. Class Action5\_TooManyAttemptsMsg
      1. Responsibility

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

* + 1. Main Attributes
    2. Operations

1. TooManyAttemptsMsg(): this is the abstract method of action 5, display the too many attempts of entering pin message.
   1. Class Action5\_1
      1. Responsibility

This is the first concrete class of action 5.

* + 1. Main Attributes
    2. Operations

1. TooManyAttemptsMsg(): display the too many attempts of entering pin message.
   1. Class Action6\_DisplayMenu
      1. Responsibility

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

* + 1. Main Attributes
    2. Operations

1. DisplayMenu(): this is the abstract method of action 6, display the account operation menu.
   1. Class Action6\_1
      1. Responsibility

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

* + 1. Main Attributes
    2. Operations

1. DisplayMenu(): display the account-1’s operation menu.
   1. Class Action6\_2
      1. Responsibility

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

* + 1. Main Attributes
    2. Operations

1. DisplayMenu(): display the account-2’s operation menu.
   1. Class Action7\_MakeDeposit
      1. Responsibility

This is the abstract super class for all the action 7 classes, all action 7 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. MakeDeposit(): this is the abstract method of action 7, making deposit.
   1. Class Action7\_1
      1. Responsibility

This is the first concrete class of action 7 for the account 1.

* + 1. Main Attributes
    2. Operations

1. MakeDeposit(): making deposit for account-1.
   1. Class Action7\_2
      1. Responsibility

This is the first concrete class of action 7 for the account 2

* + 1. Main Attributes
    2. Operations

1. MakeDeposit(): making deposit for account-2.
   1. Class Action8\_DisplayBalance
      1. Responsibility

This is the abstract super class for all the action 8 classes, all action 8 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. DisplayBalance(): this is the abstract method of action 8, display the current balance.
   1. Class Action8\_1
      1. Responsibility

This is the first concrete class of action 8 for the account 1.

* + 1. Main Attributes
    2. Operations

1. DisplayBalance(): display current balance of account-1.
   1. Class Action8\_2
      1. Responsibility

This is the first concrete class of action 8 for the account 2

* + 1. Main Attributes
    2. Operations

1. DisplayBalance(): display current balance of account-2.
   1. Class Action9\_BelowMinMsg
      1. Responsibility

This is the abstract super class for all the action 9 classes, all action 9 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. BelowMinMsg(): this is the abstract method of action 9, display the below MIN message.
   1. Class Action9\_1
      1. Responsibility

This is the first concrete class of action 9.

* + 1. Main Attributes
    2. Operations

1. BelowMinMsg(): display the below MIN message.
   1. Class Action10\_MakeWithdraw
      1. Responsibility

This is the abstract super class for all the action 10 classes, all action 10 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. MakeWithdraw(): this is the abstract method of action 10, making withdraw.
   1. Class Action10\_1
      1. Responsibility

This is the first concrete class of action 10 for the account 1.

* + 1. Main Attributes
    2. Operations

1. MakeWithdraw(): making withdraw for account-1.
   1. Class Action10\_2
      1. Responsibility

This is the first concrete class of action 10 for the account 2

* + 1. Main Attributes
    2. Operations

1. MakeWithdraw(): making withdraw for account-2.
   1. Class Action11\_BelowMinPenalty
      1. Responsibility

This is the abstract super class for all the action 11 classes, all action 11 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. BelowMinPenalty(): this is the abstract method of action 11, when withdraw below the MIN, apply penalty on the balance.
   1. Class Action11\_1
      1. Responsibility

This is the first concrete class of action 11 for the account 1.

* + 1. Main Attributes
    2. Operations

1. BelowMinPenalty(): for account 1 when withdraw below the MIN, apply penalty on the balance.
   1. Class Action12\_NoFoundMsg
      1. Responsibility

This is the abstract super class for all the action 12 classes, all action 12 classes should implement it.

* + 1. Main Attributes
    2. Operations

1. NoFoundMsg(): this is the abstract method of action 10, display the no found in account message.
   1. Class Action12\_1
      1. Responsibility

This is the first concrete class of action 12.

* + 1. Main Attributes
    2. Operations

1. NoFoundMsg(): display the no found in account message.
2. **Data Package**
   1. Class Data
      1. Responsibility

This is the abstract super class for all data classes.

* + 1. Main Attributes
    2. Operations
  1. Class Data1
     1. Responsibility

This is the data class for account 1.

* + 1. 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.
   * 1. Operations
10. getTemp\_pin(): return temp pin value.
11. setTemp\_pin(String): set the value of temp pin.
12. getTemp\_userID(): return temp user ID value.
13. setTemp\_userID(String): set the value of temp user ID.
14. getTemp\_balance(): return temp balance value.
15. setTemp\_balance(float): set the value of temp balance.
16. getPin(): return pin value.
17. setPin(String): set the value of pin.
18. getUserID(): return user ID value.
19. setUserID(String): set the value of user ID
20. getBalance(): return balance value.
21. setBalance(float): set the value of balance.
22. getDeposit(): return deposit amount value.
23. setDeposit(float): set the value of deposit amount.
24. getWithdraw(): return withdraw amount value.
25. setWithdraw(float): set the value of withdraw amount.
26. getPenalty(): return penalty amount value.
27. setPenalty(int): set the value of penalty amount.
    1. Class Data2
       1. Responsibility

This is the data class for account 2.

* + 1. 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.
   * 1. Operations
9. getTemp\_pin(): return temp pin value.
10. setTemp\_pin(String): set the value of temp pin.
11. getTemp\_userID(): return temp user ID value.
12. setTemp\_userID(String): set the value of temp user ID.
13. getTemp\_balance(): return temp balance value.
14. setTemp\_balance(float): set the value of temp balance.
15. getPin(): return pin value.
16. setPin(String): set the value of pin.
17. getUserID(): return user ID value.
18. setUserID(String): set the value of user ID
19. getBalance(): return balance value.
20. setBalance(float): set the value of balance.
21. getDeposit(): return deposit amount value.
22. setDeposit(float): set the value of deposit amount.
23. getWithdraw(): return withdraw amount value.
24. setWithdraw(float): set the value of withdraw amount.
25. **Factories Package**
    1. Class AbstractFactory
       1. Responsibility

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

* + 1. Main Attributes

1. Data data: store the data object.
   * 1. Operations
2. getData(): abstract method to return the data object.
3. create\_Action1\_StoreData(): abstract method to create Action 1 object.
4. create\_Action2\_IncorrectIDMsg(): abstract method to create Action 2 object.
5. creat\_Action3\_PromptPin(): abstract method to create Action 3 object.
6. creat\_Action4\_IncorrectPinMsg(): abstract method to create Action 4 object.
7. creat\_Action5\_TooManyAttemptsMsg(): abstract method to create Action 5 object.
8. creat\_Action6\_DisplayMenu(): abstract method to create Action 6 object.
9. creat\_Action7\_MakeDeposit(): abstract method to create Action 7 object.
10. creat\_Action8\_DisplayBalance(): abstract method to create Action 8 object.
11. creat\_Action9\_BelowMinMsg(): abstract method to create Action 9 object.
12. creat\_Action10\_MakeWithdraw(): abstract method to create Action 10 object.
13. creat\_Action11\_BelowMinPenalty(): abstract method to create Action 11 object.
14. creat\_Action12\_NoFoundMsg(): abstract method to create Action 12 object.
    1. Class Factory\_Account\_1
       1. Responsibility

This is the factory class that create objects for account 1.

* + 1. Main Attributes
    2. 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.
    1. Class Factory\_Account\_2
       1. Responsibility

This is the factory class that create objects for account 2.

* + 1. Main Attributes
    2. 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.
14. **MDA-EFSM Package**
    1. Class MDA\_EFSM
       1. Responsibility

This is the class of MDA-EFSM which will control all the state transfer based on the diagram and call the relation actions.

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

This is the abstract super class for all state classes.

* + 1. 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.
   * 1. Operations
4. getAttempts(): return the current attempt times.
5. getID(): return the state’s ID.
6. Open(AbstractFactory): super method of event open.
7. Login(AbstractFactory): super method of event login.
8. IncorrectLogin(AbstractFactory): super method of event incorrect login.
9. IncorrectPin(AbstractFactory, int): super method of event incorrect pin.
10. CorrectPinAboveMin(AbstractFactory): super method of event correct pin above min.
11. CorrectPinBelowMin(AbstractFactory): super method of event correct pin below min.
12. Logout(AbstractFactory): super method of event logout.
13. Deposit(AbstractFactory): super method of event deposit.
14. DepositAboveMin(AbstractFactory): super method of event deposit above min.
15. DepositBelowMin(AbstractFactory): super method of event deposit below min.
16. Balance(AbstractFactory): super method of event check balance.
17. Withdraw(AbstractFactory): super method of event making withdraw.
18. WithdrawAboveMin(AbstractFactory): super method of event making withdraw above min.
19. WithdrawPenalty(AbstractFactory): super method of event making withdraw and apply the penalty.
20. NoFound(AbstractFactory): super method of event no found in account.
21. Lock(AbstractFactory): super method of event locks the account.
22. IncorrectLock(AbstractFactory): super method of event incorrect lock.
23. IncorrectUnlock(AbstractFactory): super method of event incorrect unlock.
24. UnlockAboveMin(AbstractFactory): super method of event unlocks the account with balance above min.
25. UnlockBelowMin(AbstractFactory): super method of event unlocks the account with balance above min.
26. Suspend(AbstractFactory): super method of event suspends the account.
27. Activate(AbstractFactory): super method of event activates a suspended account.
28. Close(AbstractFactory): super method of event closes an account.
    1. Class State\_0\_Start
       1. Responsibility

This is the state 0, start state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 0.
   * 1. Operations
2. getID(): return the start state’s ID which is 0.
3. Open(AbstractFactory): override the super method in the State class, which will call action 1.
   1. Class State\_1\_Idle
      1. Responsibility

This is the state 1, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 1.
   * 1. Operations
2. getID(): return the start state’s ID which is 1.
3. Login(AbstractFactory): override the super method in the State class, which will call action 3 and reset the attempts to 0.
4. IncorrectLogin(AbstractFactory): override the super method in the State class, which will call action 2.
   1. Class State\_2\_CheckPin
      1. Responsibility

This is the state 2, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 2.
   * 1. Operations
2. getID(): return the start state’s ID which is 2.
3. 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.
4. CorrectPinAboveMin(AbstractFactory): override the super method in the State class, which will call action 6.
5. CorrectPinBelowMin(AbstractFactory): override the super method in the State class, which will call action 6.
   1. Class State\_3\_Ready
      1. Responsibility

This is the state 3, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 3.
   * 1. Operations
2. getID(): return the start state’s ID which is 3.
3. Balance(AbstractFactory): override the super method in the State class, which will call action 8.
4. Deposit(AbstractFactory): override the super method in the State class, which will call action 7.
5. IncorrectLock(AbstractFactory): override the super method in the State class, which will call action 4.
6. Withdraw(AbstractFactory): override the super method in the State class, which will call action 10.
   1. Class State\_4\_S1
      1. Responsibility

This is the state 4, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 4.
   * 1. Operations
2. getID(): return the start state’s ID which is 4.
   1. Class State\_5\_S2
      1. Responsibility

This is the state 5, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 5.
   * 1. Operations
2. getID(): return the start state’s ID which is 5.
3. WithdrawPenalty(AbstractFactory): override the super method in the State class, which will call action 11.
4. NoFound(AbstractFactory): override the super method in the State class, which will call action 12.
   1. Class State\_6\_Overdraw
      1. Responsibility

This is the state 6, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 6.
   * 1. Operations
2. getID(): return the start state’s ID which is 6.
3. Balance(AbstractFactory): override the super method in the State class, which will call action 8.
4. Withdraw(AbstractFactory): override the super method in the State class, which will call action 9.
5. IncorrectLock(AbstractFactory): override the super method in the State class, which will call action 4.
6. Deposit(AbstractFactory): override the super method in the State class, which will call action 7.
   1. Class State\_7\_Suspend
      1. Responsibility

This is the state 7, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 7.
   * 1. Operations
2. getID(): return the start state’s ID which is 7.
3. Balance(AbstractFactory): override the super method in the State class, which will call action 8.
   1. Class State\_8\_Locked
      1. Responsibility

This is the state 8, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 8.
   * 1. Operations
2. getID(): return the start state’s ID which is 8.
3. IncorrectUnlock(AbstractFactory): override the super method in the State class, which will call action 4.
   1. Class State\_9\_End
      1. Responsibility

This is the state 9, idle state in the MDA-EFSM diagram.

* + 1. Main Attributes

1. ID : int: this is the start state’s ID which is 9.
   * 1. Operations
2. getID(): return the start state’s ID which is 9.
3. **Test Driver Class**
   1. Class Test Driver
      1. Responsibility

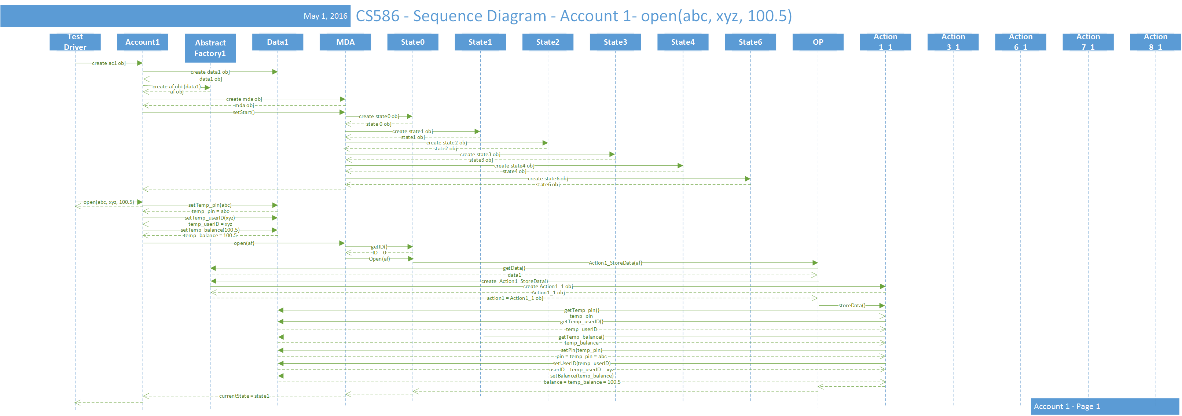
This is the client class, which will test all the operations that provided by the account 1 and account 2.

* + 1. Main Attributes
    2. Operations

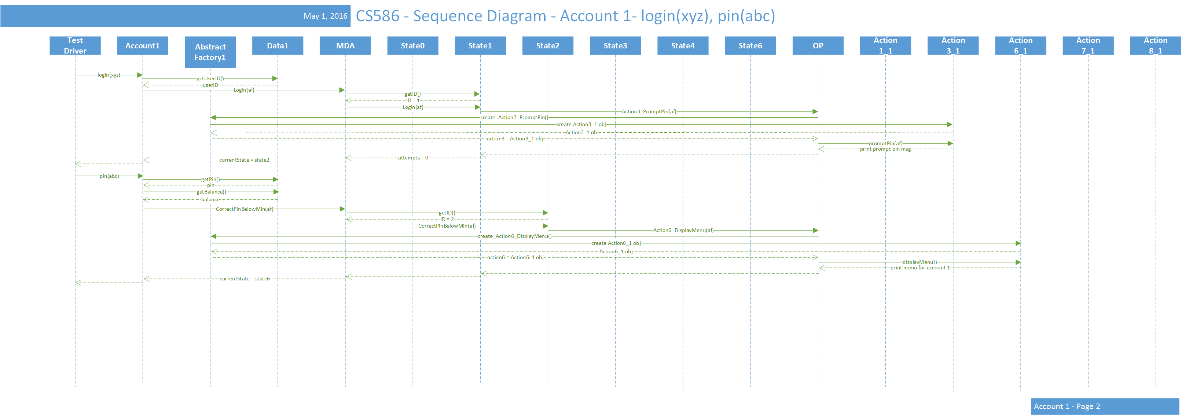
1. printTitle(): print the title of this class.
2. printAccountSelection(): print the operation of account selection.
3. printAccount\_1\_Menu(): print the operations provided by the account 1.
4. printAccount\_1\_Selection(): print the selected operation of account 1.
5. printAccount\_2\_Menu(): print 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.
10. **Sequence Diagram**
11. **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()

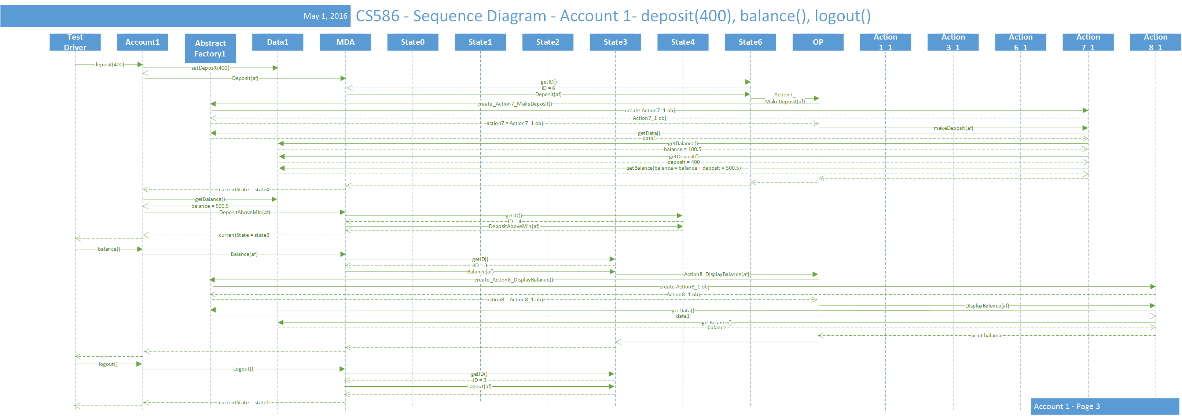
Sequence Diagram 1-1: open(abc, xyz, 100.5)



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



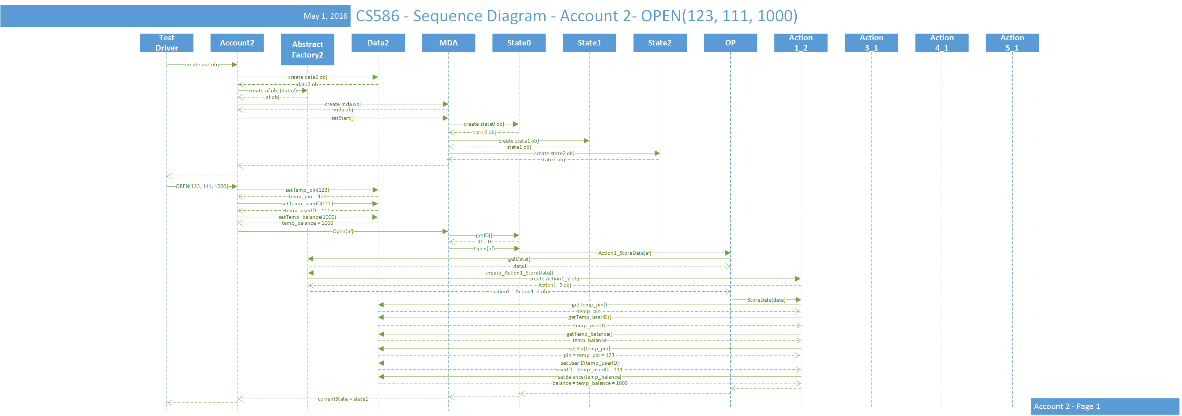
Sequence Diagram 1-3: deposit(400), balance(), logout()



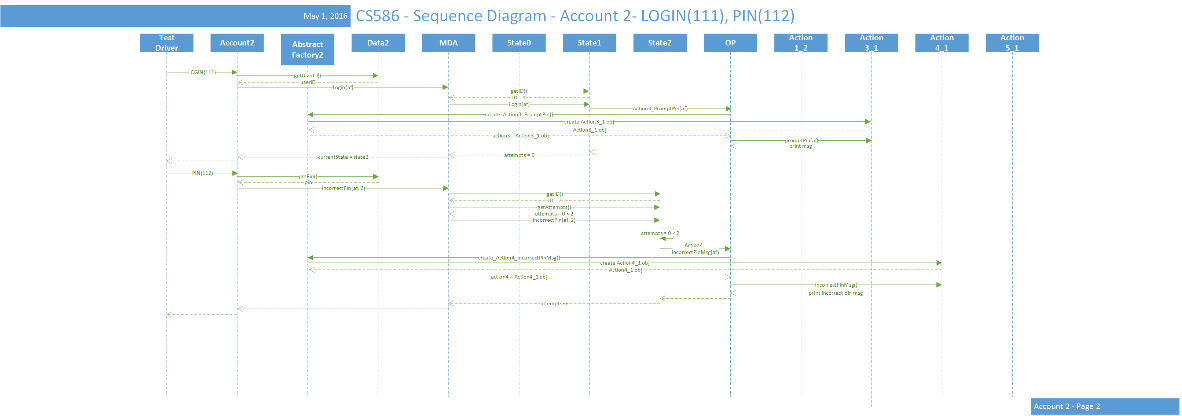
1. **Sequence Diagram 2**

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)

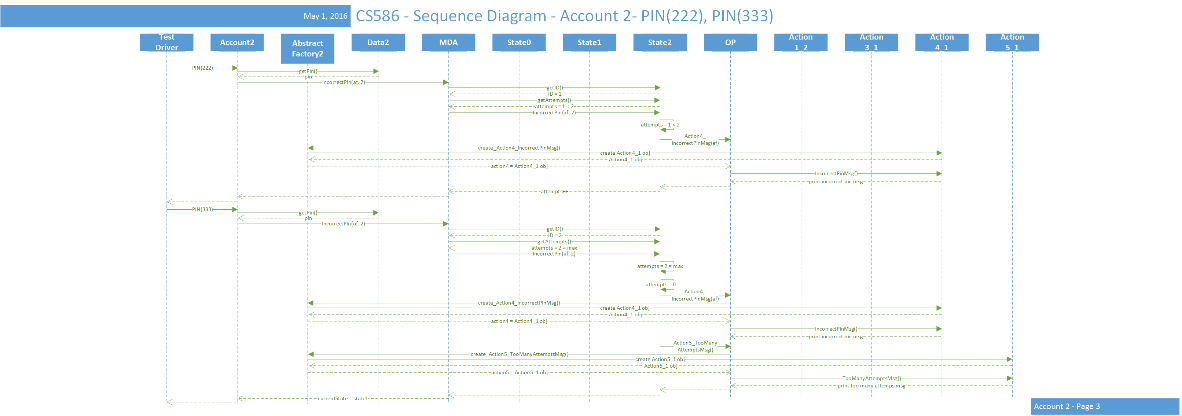
Sequence Diagram 2-1: OPEN(123, 111, 1000)



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



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



1. **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.

## 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**

#### 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);

}

}

}

1. **States Package**

#### 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) {

}

}

#### 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);

}

}

#### 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);

}

}

#### 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);

}

}

#### 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);

}

}

#### Class State\_4\_S1

package Jian.States;

public class State\_4\_S1 extends States {

@Override

public int getID() {

ID = 4;

return ID;

}

}

#### 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);

}

}

#### 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);

}

}

#### 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);

}

}

#### 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);

}

}

#### Class State\_9\_End

package Jian.States;

public class State\_9\_End extends States{

@Override

public int getID() {

ID = 9;

return ID;

}

}

## 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**

#### 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();

}

}

#### Class Action1\_StoreData

package Jian.Actions;

import Jian.Data.Data;

public interface Action1\_StoreData {

public void StoreData(Data data);

}

#### 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!");

}

}

#### 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!");

}

}

#### Class Action2\_IncorrectIDMsg

package Jian.Actions;

public interface Action2\_IncorrectIDMsg {

public void IncorrectIDMsg();

}

#### 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!");

}

}

#### Class Action3\_PromptPin

package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action3\_PromptPin {

public void PromptPin(AbstractFactory af);

}

#### 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!");

}

}

#### Class Action4\_IncorrectPinMsg

package Jian.Actions;

public interface Action4\_IncorrectPinMsg {

public void IncorrectPinMsg();

}

#### 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!");

}

#### Class Action5\_TooManyAttemptsMsg

package Jian.Actions;

public interface Action5\_TooManyAttemptsMsg {

public void TooManyAttemptsMsg();

}

#### 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!");

}

}

#### Class Action6\_DisplayMenu

package Jian.Actions;

public interface Action6\_DisplayMenu {

public void DisplayMenu();

}

#### 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!");

}

}

#### 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!");

}

}

#### Class Action7\_MakeDeposit

package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action7\_MakeDeposit {

public void MakeDeposit(AbstractFactory af);

}

#### 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!");

}

}

#### 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!");

}

}

#### Class Action8\_DisplayBalance

package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action8\_DisplayBalance {

public void DisplayBalance(AbstractFactory af);

}

#### Class Action8\_1

package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action8\_DisplayBalance {

public void DisplayBalance(AbstractFactory af);

}

#### 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!");

}

}

#### Class Action9\_BelowMinMsg

package Jian.Actions;

public interface Action9\_BelowMinMsg {

public void BelowMinMsg();

}

#### 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!");

}

}

#### Class Action10\_MakeWithdraw

package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action10\_MakeWithdraw {

public void MakeWithdraw(AbstractFactory af);

}

#### 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!");

}

}

#### 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!");

}

}

#### Class Action11\_BelowMinPenalty

package Jian.Actions;

import Jian.Factories.AbstractFactory;

public interface Action11\_BelowMinPenalty {

public void BelowMinPenalty(AbstractFactory af);

}

#### 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!");

}

}

#### Class Action12\_NoFoundMsg

package Jian.Actions;

public interface Action12\_NoFoundMsg {

public void NoFoundMsg();

}

#### 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!");

}

}

## 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**

#### 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();

}

#### 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();

}

}

#### 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();

}

}

## Other Source Code

Here are the remaining classes’ source code.

1. **Account Package**

#### 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);

}

}

}

#### 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);

}

}

1. **Data Package**

#### Class Data

package Jian.Data;

public abstract class Data {

}

#### 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;

}

}

#### 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;

}

}

1. **Test Driver Class**

#### 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");

}

}

}

}