CS 586 SOFTWARE SYSTEM ARCHITECTURE

Rutul Mehta (A20476293)

Table of Contents

1. MDA-EFSM model for Account Components (Part-1)

1.1 A list of meta events for the MDA-EFSM

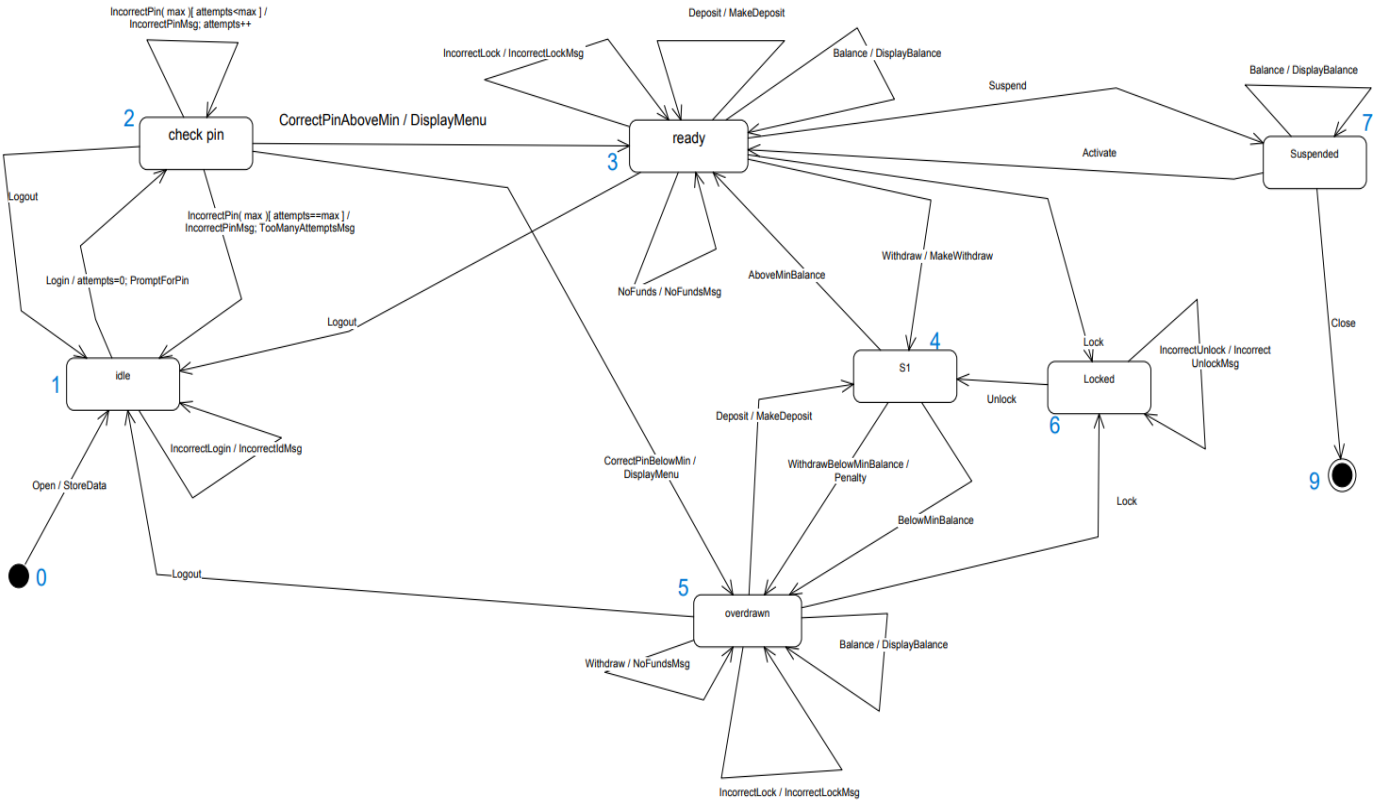
1.1.1 MDA-EFSM Events:

- Open()
- Login()
- IncorrectLogin()
- IncorectPin(int max)
- CorrectPinBelowMin()
- CorrectPinAboveMin()
- Deposit()
- BelowMinBalance()
- AboveMinBalance()
- Logout()
- Balance()
- Withdraw()
- WithdrawBelowMinBalance()
- NoFunds()
- Lock()
- IncorrectLock()
- Unlock()
- IncorrectUnlock()
- Suspend()
- Activate()
- Close()

1.1.2 MDA-EFSM Actions

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

1.1.3 A state diagram of the MDA-EFSM



1.1.4 Pseudo-code of all operations of Input Processors of Accounts: Account-1 and Account-2

Operations of the Input Processor (ACCOUNT-1)

```
open (int p, int y, int a) {
// store p, y and a in temp data store
        ds->temp_p=p;
        ds->temp_y=y;
        ds->temp_a=a;
        m->Open();
}

pin (int x) {
        if (x==ds->pin) {
            if (ds->balance > 100)
                m->CorrectPinAboveMin ();
            else m->CorrectPinBelowMin();
        }
        else m->IncorrectPin(1)
}


deposit (int d) {
        ds->temp_d=d;
        m->Deposit();
        if (ds->balance>100)
            m->AboveMinBalance();
        else m->BelowMinBalance();
}

withdraw (int w) {
        ds->temp_w=w;
        m->withdraw();
        if ((ds->balance>100)
             m->AboveMinBalance();
        else m->WithdrawBelowMinBalance();
}
```

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

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

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

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

unlock (int x) {
        if (x==ds->pin) {
            m->Unlock();
            if (ds->balance > 100)
                m->AboveMinBalance ();
            else m->BelowMinBalance();
        }
        else m->IncorrectUnlock();
}
```

Notice:
m: is a pointer to the MDA-EFSM object
ds: is a pointer to the Data Store DS-1 object
In the data store:
balance: contains the current balance
pin: contains the pin #
id: contains user id

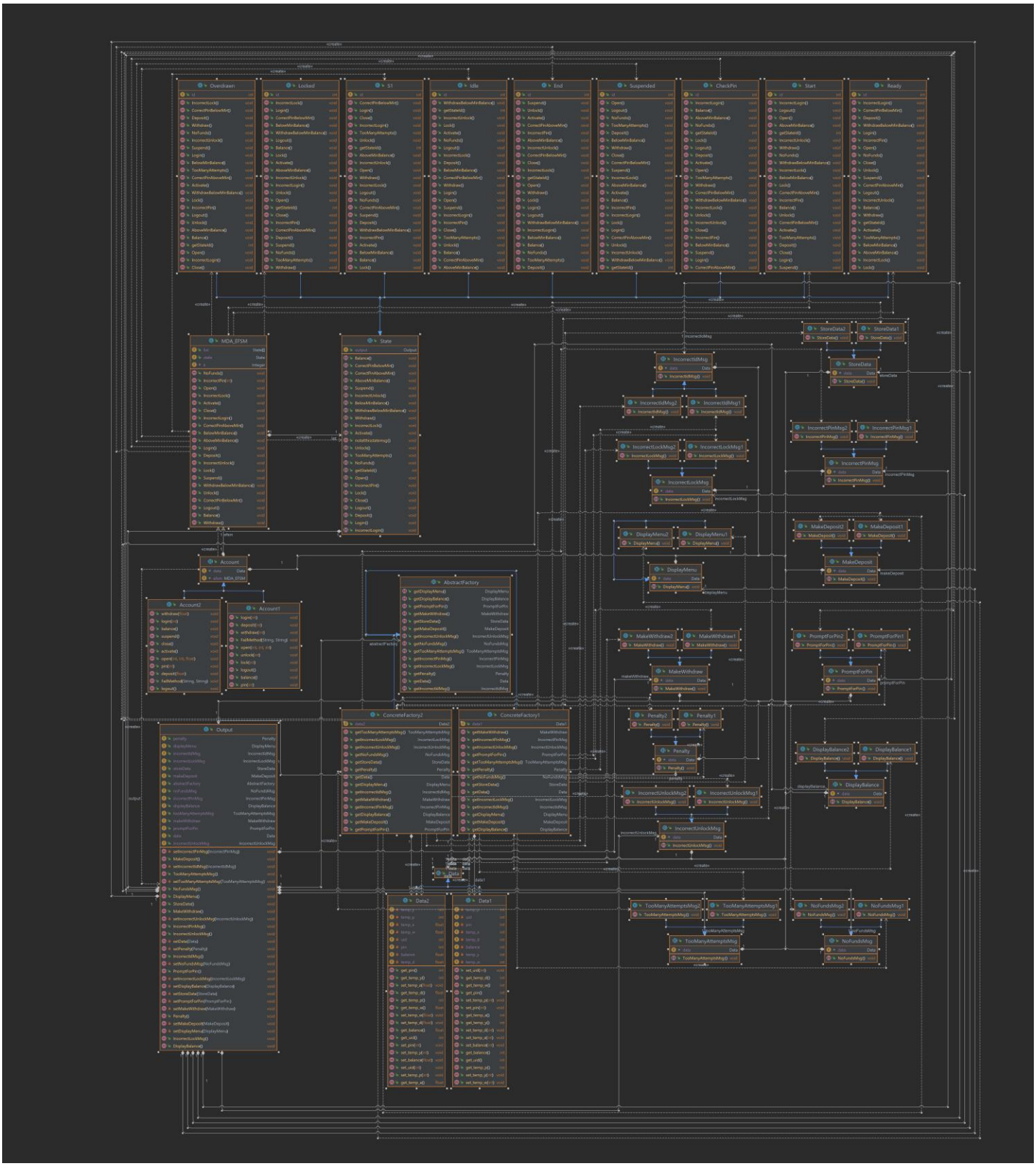
Operations of the Input Processor
(ACCOUNT-2)

```
OPEN (int p, int y, float a) {
// store p, y and a in temp data store
        ds->temp_p=p;
        ds->temp_y=y;
        ds->temp_a=a;
        m->Open();
}

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

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

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

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

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

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

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

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

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

Notice:
m: is a pointer to the MDA-EFSM object
ds: is a pointer to the Data Store DS-2 object
In the data store:
balance: contains the current balance
pin: contains the pin #
id: contains user id

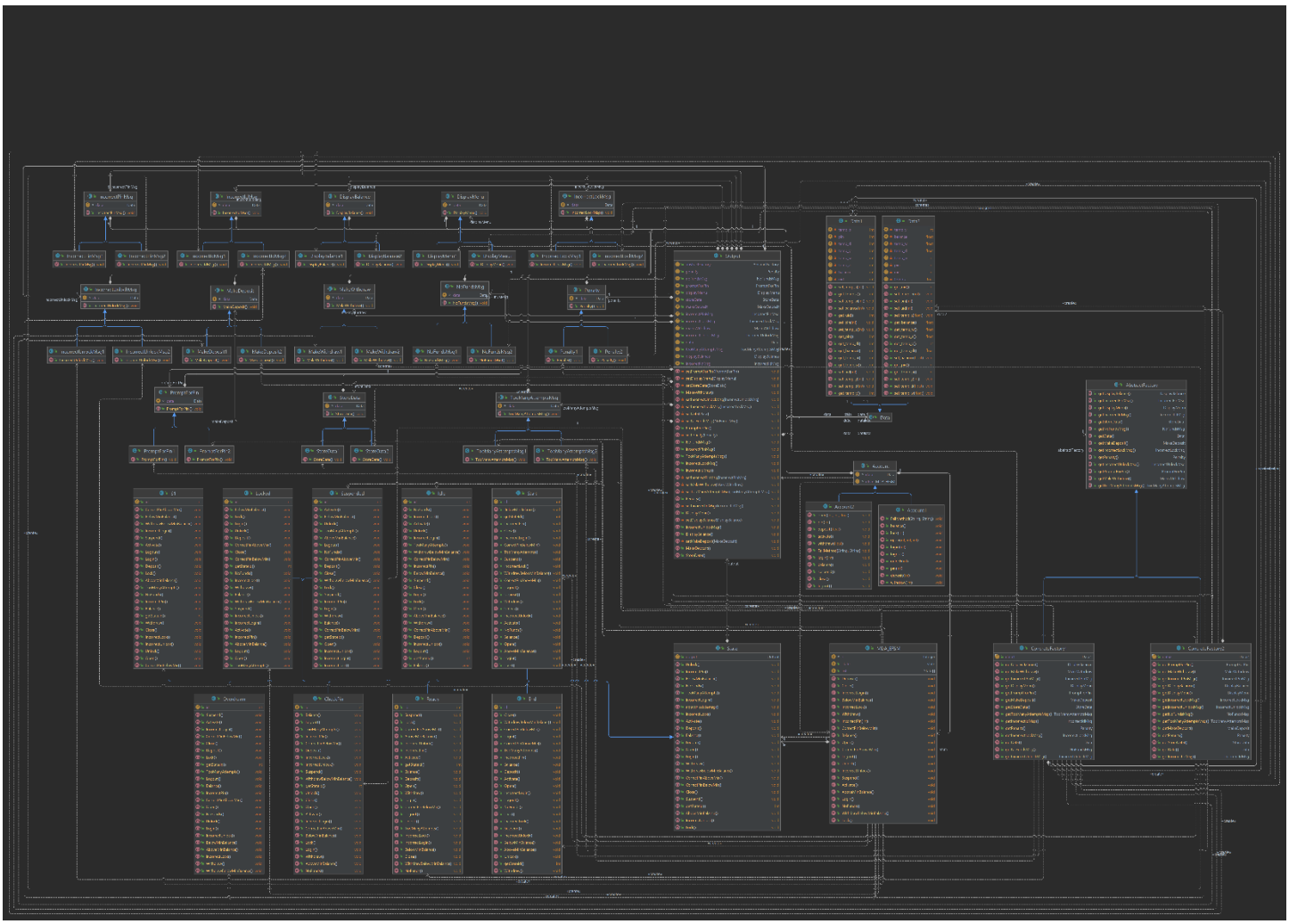
2. Class diagram of the MDA of the Account components

2.1 Class Diagram (Generated from the Code)

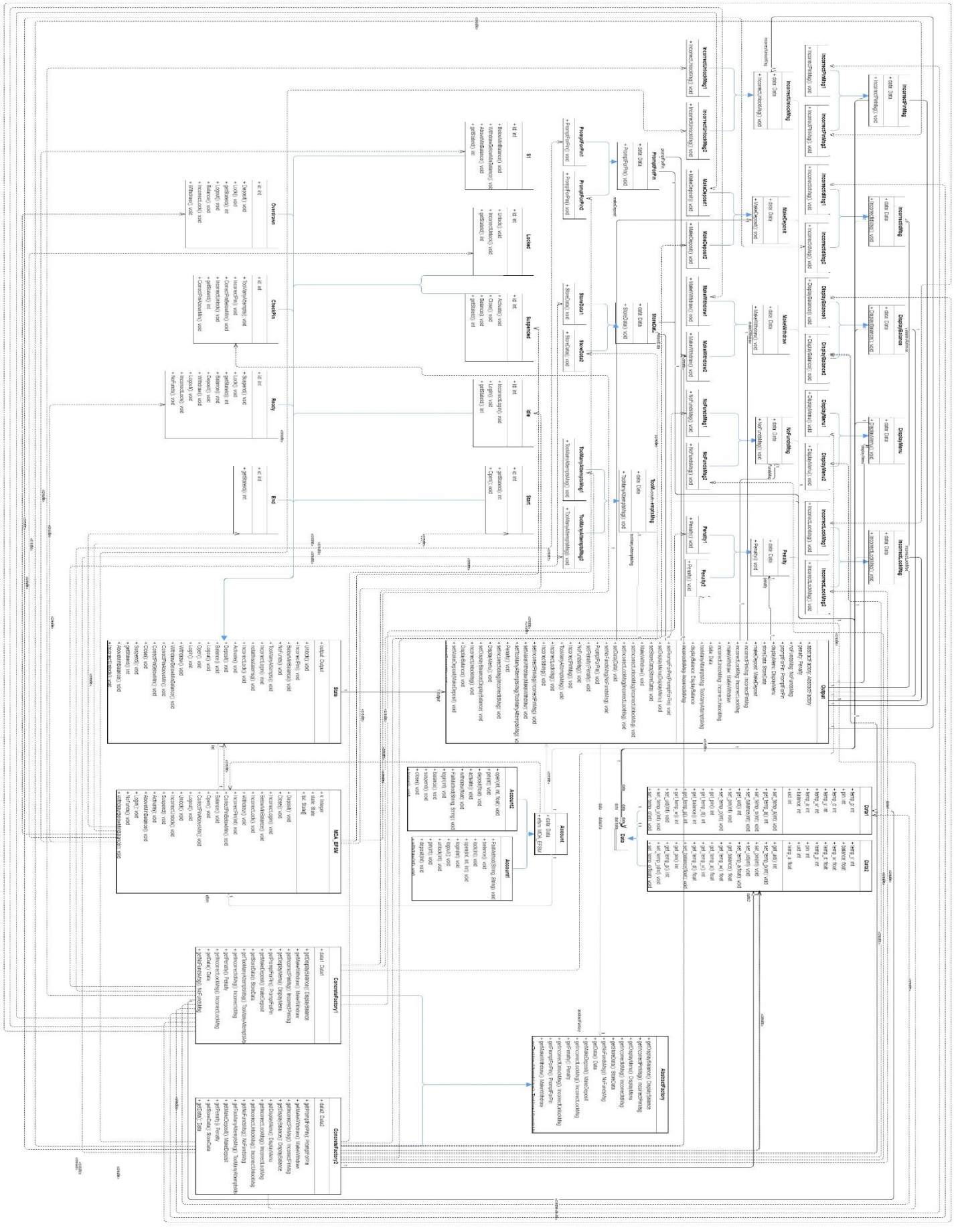


- As mentioned in the above figure, all the methods of the state class (e.g Idle, CheckPin, Ready, etc) mentioned in the above figure do not invoke methods from the output class. With a centralized approach, only a few methods are invoked but I call other methods just to display the message that a particular event can not be performed from the current state.

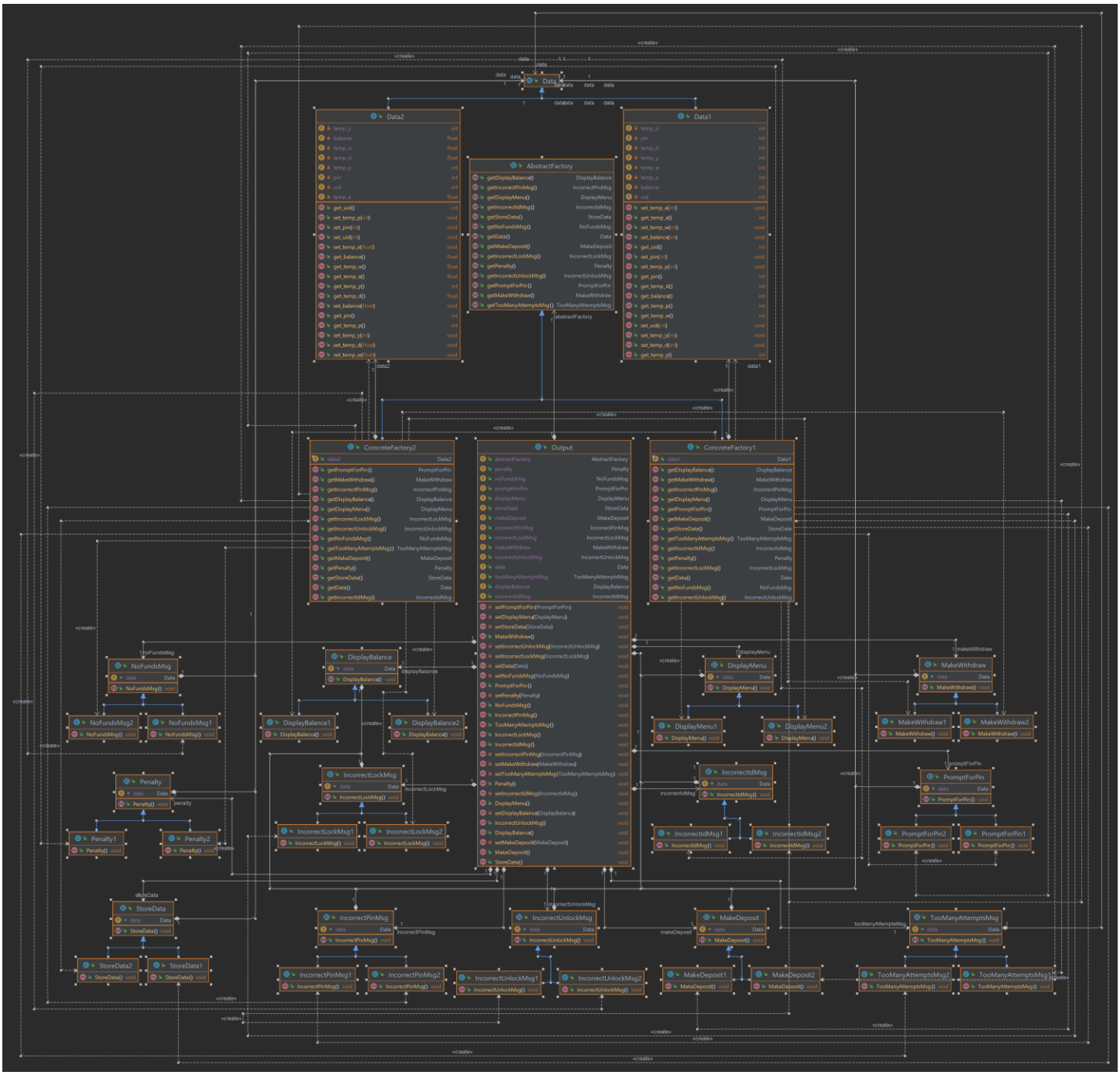
❖ Class Diagram in horizontal format.



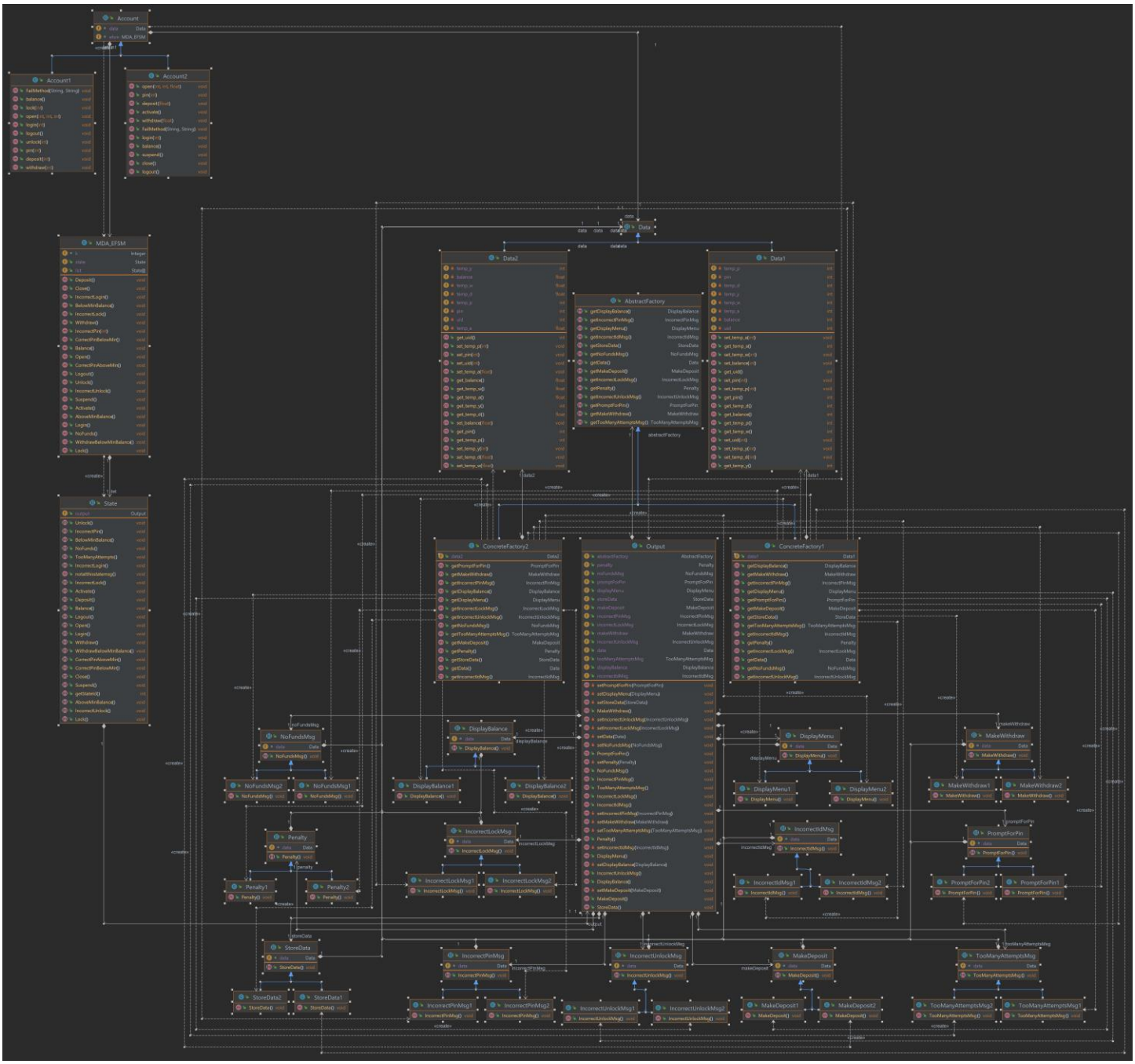
❖ The same Class diagram from draw.io



2.2 Strategy Pattern



2.3 Abstract Factory Pattern



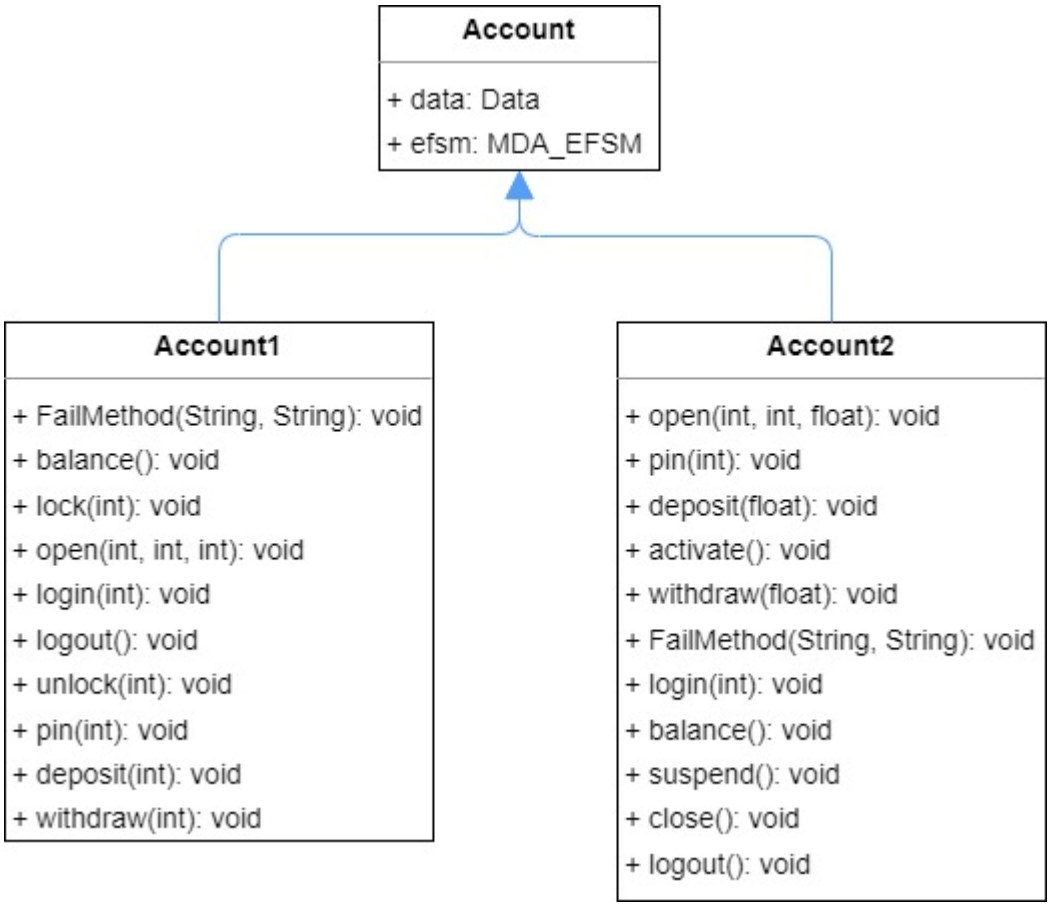
2.4 State Pattern

3. The purpose of the class and responsibilities of each operation.

3.1 AccountDriver Class

- Users can select whether they want to perform Account-1 operations or Account-2 operations, both Account-1 and Account-2 have a set of operations.
- After the selection of the Account, users can pass input data to perform operations related to that specific Account. Those data are stored in the Data class. For Account-1, Data1 class is used as a storage class whereas for Account-2 Data2 storage class is used.
- Next, I will explain the operations of the Account-1 and Account-2 Classes.

3.2 Account, Account1, and Account2 Class



3.2.1 Account Class

```
/**
 * Account abstract Class
 */
public abstract class Account {
    Data data;
    MDA_EFSM efsm;

    /**
     * Constructor of the Account class
     * @param abstractFactory
     */
    Account(AbstractFactory abstractFactory){
```

```
        this.data = abstractFactory.getData();
        this.efsm = new MDA_EFSM(new Output(abstractFactory));
    }
}
```

3.2.2 Account1 Class

```
/**
 * Account1 Class
 * All the methods of account1 class invoke members of MDA_EFSM class.
 */
public class Account1 extends Account{

    /**
     * Constructor of the account1 class.
     * @param abstractFactory
     */
    public Account1(AbstractFactory abstractFactory) {
        super(abstractFactory);
    }
```

3.2.2.1 Open(int p, int y, int a)

```
    /**
     * Open method is used to open an account
     * @param p - Pin
     * @param y - User Identification
     * @param a - Account balance
     */
    public void open(int p, int y, int a){
        Data1 data1 = (Data1)this.data;
        if(p>0 && y>0 && a>0){
            data1.set_temp_p(p);
            data1.set_temp_y(y);
            data1.set_temp_a(a);
            efsm.Open();
        }
        else{
            FailMethod("Open","Pin, UserId, Balance");
        }
    }
```

- This method is used to open an account by storing the pin, user identification, and balance in a temporary variable of the Data1 class.
- This method invokes the Open() method of the MDA_EFSM class if all three parameters are positive else call FailMethod().

3.2.2.2 Pin(int x)

```
    /**
     * Pin method check whether entered pin is correct or not.
     * @param x - Pin
     */
```

```
public void pin(int x){
    Data1 data1 = (Data1)this.data;
    if(x == data1.get_pin()){
        if(data1.get_balance() > 100){
            efsm.CorrectPinAboveMin();
        }
        else{
            efsm.CorrectPinBelowMin();
        }
    }
    else{
        efsm.IncorrectPin(1);
    }
}
```

- This method checks whether entered pin is correct or not.
- If user enters correct pin and account balance is above 100 then CorrectPinAboveMin() method is invoked else CorrectPinBelowMin() method is invoked from MDA_EFSM class.
- If a user enters an incorrect pin then IncorrectPin(1) method invokes.

3.2.2.3 Deposite(int d)

```
/**
 * Deposit method deposit amount in account
 * @param d - deposit amount
 */
public void deposit(int d){
    Data1 data1 = (Data1)this.data;
    if(d>0){
        data1.set_temp_d(d);
        efsm.Deposit();
        if(data1.get_balance() > 100){
            efsm.AboveMinBalance();
        }
        else{
            efsm.BelowMinBalance();
        }
    }
    else{
        FailMethod("Deposit","Deposit amount");
    }

}
```

- This method calls when the user wants to deposit amount to the Account.
- If account balance is above 100 then AboveMinBalance() method is invoked else BelowMinBalance() method is invoked from MDA_EFSM class.
- If a user enters a negative value then FailMethod() is called.

3.2.2.4 Withdraw(int w)

```
/**
 * Withdraw method withdraw amount from account
 * @param w - withdraw amount
 */
public void withdraw(int w){
    Data1 data1 = (Data1)this.data;
    if(w>0){
        data1.set_temp_w(w);
        efsm.Withdraw();
        if (data1.get_balance() > 100) {
            efsm.AboveMinBalance();
        } else {
            efsm.WithdrawBelowMinBalance();
        }
    }
    else{
        FailMethod("Withdraw", "Withdraw amount");
    }
}
```

- This method calls when the user wants to withdraw some amount from the Account.
- If the account balance is above 100 then AboveMinBalance() method is invoked else withdrawBelowMinBalance() method is invoked from the MDA_EFSM class.
- If a user enters a negative value then FailMethod() is called.

3.2.2.5 Balance()

```
/**
 * balance method check the balance of the account.
 */
public void balance(){
    efsm.Balance();
}
```

- This method invokes the Balance() method from the MDA_EFSM class to check the balance of the account.

3.2.2.6 Login(int y)

```
/**
 * login method check whether user entered correct UserID or not.
 * @param y - User Identification number
 */
public void login(int y){
    Data1 data1 = (Data1)this.data;
    if(y>0){
        if(y == data1.get_uid()){
            efsm.Login();
        }
        else{
            efsm.IncorrectLogin();
        }
    }
    else{
```

```
            FailMethod("Login", "User Identification");
        }
    }
```

- This method checks the user identification number to login into the system.
- If the UserId is correct then login() otherwise IncorrectLogin() method is invoked.
- If the user enters a negative value of the userid then FailMethod() calls.

3.2.2.7 Logout()

```
    /**
     * logout method use to logout from the current system.
     */
    public void logout(){
        efsm.Logout();
    }
```

- This method invokes the logout method of the MDA_EFSM class.

3.2.2.8 Lock(int x)

```
    /**
     * lock method use to lock account by entering correct pin number
     * @param x - PIN
     */
    public void lock(int x){
        Data1 data1 = (Data1)this.data;
        if(x>0){
            if(data1.get_pin() == x){
                efsm.Lock();
            }
            else{
                efsm.IncorrectLock();
            }
        }
        else{
            FailMethod("Lock", "Pin number");
        }
    }
```

- This method is used to lock the account by entering the correct pin.
- If user enter correct pin Lock() method else IncorrectLock() method is invoked.
- The user must enter a positive number as a pin.

3.2.2.9 Unlock(int x)

```
    /**
     * unlock method uses to unlock account by entering correct pin number
     * @param x - Pin number
     */
    public void unlock(int x){
        Data1 data1 = (Data1)this.data;
        if(x>0){
            if(x == data1.get_pin()){
                efsm.Unlock();
                if(data1.get_balance()>100){
```

```
                efsm.AboveMinBalance();
            }
            else{
                efsm.BelowMinBalance();
            }
        }
        else{
            efsm.IncorrectUnlock();
        }
    }
    else{
        FailMethod("Unlock", "Pin number");
    }
}
```

- This method is used to unlock the account by entering the correct pin.
- If the user enters the correct pin and the user account balance is above 100 then AboveMinBalance() else BelowMinBalance() method is invoked from the MDA_EFSM class.
- The user must enter a positive number as a pin.

3.2.2.10 FailMethod(string methodname, string varname)

```
/**
 * FailMethod call when user enters invalid data in any other method.
 * @param methodname - Method name
 * @param varname - Variable name
 */
public void FailMethod(String methodname, String varname){
    System.out.println(methodname+" Method Failed");
    System.out.println(varname+" should be greater than 0");
    System.out.println("\n");
}
```

- FailMethod() call when the user enters invalid data as an input.
- This method simply prints the message.

3.2.3 Account2 Class

3.2.3.1 Open(int p, int y, int a)

```
/**
 * Open method is used to open an account
 * @param p - Pin
 * @param y - User Identification
 * @param a - Account balancw
 */
public void open(int p, int y, float a){
    Data2 data2 = (Data2)this.data;
    if(p>0 && y>0 && a>0){
        data2.set_temp_p(p);
        data2.set_temp_y(y);
        data2.set_temp_a(a);
        efsm.Open();
    }
    else{
```

```
            FailMethod("Open","Pin, UserId, Balance");
        }
    }
```

- This method is used to open an account by storing the pin, user identification, and balance in a temporary variable of the Data1 class.
- This method invokes the Open() method of the MDA_EFSM class if all three parameters are positive else call FailMethod().

3.2.3.2 Pin(int x)

```
    /**
     * Pin method check whether entered pin is correct or not.
     * @param x - Pin
     */
    public void pin(int x){
        Data2 data2 = (Data2)this.data;
        if(x == data2.get_pin()){
            efsm.CorrectPinAboveMin();
        }
        else{
            efsm.IncorrectPin(2);
        }
    }
```

- This method checks whether entered pin is correct or not.
- If user enters correct pin and account balance is above 100 then CorrectPinAboveMin() method is invoked else CorrectPinBelowMin() method is invoked from MDA_EFSM class.
- If a user enters an incorrect pin then IncorrectPin(1) method invokes.

3.2.3.3 Deposit(float d)

```
    /**
     * Deposit method deposit amount in account
     * @param d - deposit amount
     */
    public void deposit(float d){
        Data2 data2 = (Data2)this.data;
        if(d>0){
            data2.set_temp_d(d);
            efsm.Deposit();
        }
        else{
            FailMethod("Deposit","Deposit amount");
        }
    }
```

- This method calls when the user wants to deposit amount to the Account.
- If account balance is above 100 then AboveMinBalance() method is invoked else BelowMinBalance() method is invoked from MDA_EFSM class.
- If a user enters a negative value then FailMethod() is called.

3.2.3.4 Withdraw(float w)

```
/**
 * Withdraw method withdraw amount from account
 * @param w - withdraw amount
 */
public void withdraw(float w){
    Data2 data2 = (Data2)this.data;
    if(w>0){
        data2.set_temp_w(w);
        if (data2.get_balance() > 0) {
            efsm.Withdraw();
            efsm.AboveMinBalance();
        } else {
            efsm.NoFunds();
        }
    }
    else{
        FailMethod("Withdraw", "Withdraw amount");
    }
}
```

- This method calls when the user wants to withdraw some amount from the Account.
- If the account balance is above 100 then AboveMinBalance() method is invoked else withdrawBelowMinBalance() method is invoked from the MDA_EFSM class.
- If a user enters a negative value then FailMethod() is called.

3.2.3.5 Balance()

```
/**
 * balance method check the balance of the account.
 */
public void balance(){
    efsm.Balance();
}
```

- This method invokes the Balance() method from the MDA_EFSM class to check the balance of the account.

3.2.3.6 Login(int y)

```
/**
 * login method check whether user entered correct UserID or not.
 * @param y - User Identification number
 */
public void login(int y){
    Data2 data2 = (Data2)this.data;
    if(y>0){
        if(y == data2.get_uid()){
            efsm.Login();
        }
        else{
            efsm.IncorrectLogin();
        }
    }
```

```
        else{
            FailMethod("Login", "User Identification");
        }
    }
```

- This method checks the user identification number to login into the system.
- If the UserId is correct then login() otherwise IncorrectLogin() method is invoked.
- If the user enters a negative value of the userid then FailMethod() calls.

3.2.3.7 Logout()

```
    /**
     * logout method use to logout from the current system.
     */
    public void logout(){
        efsm.Logout();
    }
```

- This method invokes the logout() method of the MDA_EFSM class.

3.2.3.8 Suspend()

```
    /**
     * suspend method call when user wants to suspend the current system.
     */
    public void suspend(){
        efsm.Suspend();
    }
```

- This method invokes the suspend() method of the MDA_EFSM class.

3.2.3.9 Activate()

```
    /**
     * activate method calls when a user activates an account from a suspended state.
     */
    public void activate(){
        efsm.Activate();
    }
```

- This method invokes the activate() method of the MDA_EFSM class.

3.2.3.10 Close()

```
    /**
     * close method calls when user wants to close the account.
     */
    public void close(){
        efsm.Close();
    }
```

- This method invokes the close method of the MDA_EFSM class.

3.2.3.11 FailMethod()

```
    /**
     * FailMethod call when user enters invalid data in any other method.
     * @param methodname - Method name
     * @param varname - Variable name
```

```
     */
    public void FailMethod(String methodname, String varname){
        System.out.println(methodname+" Method Failed");
        System.out.println(varname+" should be greater than 0");
        System.out.println("\n");
    }
}
```

- FailMethod() calls when the user enters invalid data as an input.
- This method simply prints the message.

3.3 AbstractFactory, ConcreteFactory1, ConcreteFactory2 Classes

3.3.1 AbstractFactory Class

```
/**
 * This is an abstract class.
 * All the declared methods are also abstract.
 * The implementation of the abstract methods is in the inherited concrete factory class.
 */
public abstract class AbstractFactory {
    public abstract Data getData();
    public abstract StoreData getStoreData();
    public abstract IncorrectIdMsg getIncorrectIdMsg();
    public abstract IncorrectPinMsg getIncorrectPinMsg();
    public abstract TooManyAttemptsMsg getTooManyAttemptsMsg();
    public abstract DisplayMenu getDisplayMenu();
    public abstract MakeDeposit getMakeDeposit();
    public abstract DisplayBalance getDisplayBalance();
    public abstract PromptForPin getPromptForPin();
    public abstract MakeWithdraw getMakeWithdraw();
    public abstract Penalty getPenalty();
    public abstract IncorrectLockMsg getIncorrectLockMsg();
    public abstract IncorrectUnlockMsg getIncorrectUnlockMsg();
    public abstract NoFundsMsg getNoFundsMsg();
}
```

3.3.2 ConcreteFactory1 Class

```
public class ConcreteFactory1 extends AbstractFactory{

    //Create an object of the Data1 class.
    public final Data1 data1;

    //Constructor of the ConcreteFactory1 class where Data1 class is Initialized.
    public ConcreteFactory1(){
        data1 = new Data1();
    }

    //return Data1.
    @Override
    public Data getData() {
        return data1;
    }
```

- Here are the methods of the ConcreteFactory1 class with the explanation.

```
    //return instance of the StoreData1.
    @Override
    public StoreData getStoreData() {
        return new StoreData1(this.data1);
    }
```

```java
    //return instance of the IncorrectIdMsg1
    @Override
    public IncorrectIdMsg getIncorrectIdMsg() {
        return new IncorrectIdMsg1(this.data1);
    }

    //return instance of the IncorrectPinMsg1
    @Override
    public IncorrectPinMsg getIncorrectPinMsg() {
        return new IncorrectPinMsg1(this.data1);
    }

    //return instance of the TooManyAttemptsMsg1
    @Override
    public TooManyAttemptsMsg getTooManyAttemptsMsg() {
        return new TooManyAttemptsMsg1(this.data1);
    }

    //return instance of the DisplayMenu1
    @Override
    public DisplayMenu getDisplayMenu() {
        return new DisplayMenu1(this.data1);
    }
    //return instance of the MakeDeposit1
    @Override
    public MakeDeposit getMakeDeposit() {
        return new MakeDeposit1(this.data1);
    }

    //return instance of the DisplayBalance1
    @Override
    public DisplayBalance getDisplayBalance() {
        return new DisplayBalance1(this.data1);
    }

    //return instance of the PromptForPin1
    @Override
    public PromptForPin getPromptForPin() {
        return new PromptForPin1(this.data1);
    }

    //return instance of the MakeWithdraw1
    @Override
    public MakeWithdraw getMakeWithdraw() {
        return new MakeWithdraw1(this.data1);
    }

    //return instance of the Penalty1
    @Override
    public Penalty getPenalty() {
```

```
        return new Penalty1(this.data1);
    }

    //return instance of the IncorrectLockMsg1
    @Override
    public IncorrectLockMsg getIncorrectLockMsg() {
        return new IncorrectLockMsg1(this.data1);
    }

    //return instance of the IncorrectUnlockMsg1
    @Override
    public IncorrectUnlockMsg getIncorrectUnlockMsg() {
        return new IncorrectUnlockMsg1(this.data1);
    }

    //return instance of the NoFundsMsg1
    @Override
    public NoFundsMsg getNoFundsMsg() {
        return new NoFundsMsg1(this.data1);
    }
}
```

3.3.3 ConcreteFactory2 Class

```
public class ConcreteFactory2 extends AbstractFactory{

    //Create object of the Data2 class.
    public final Data2 data2;

    //Constructor of the ConcreteFactory2 class where Data2 class is Initialized.
    public ConcreteFactory2(){
        data2 = new Data2();
    }

    //return Data2.
    @Override
    public Data getData() {
        return data2;
    }
```

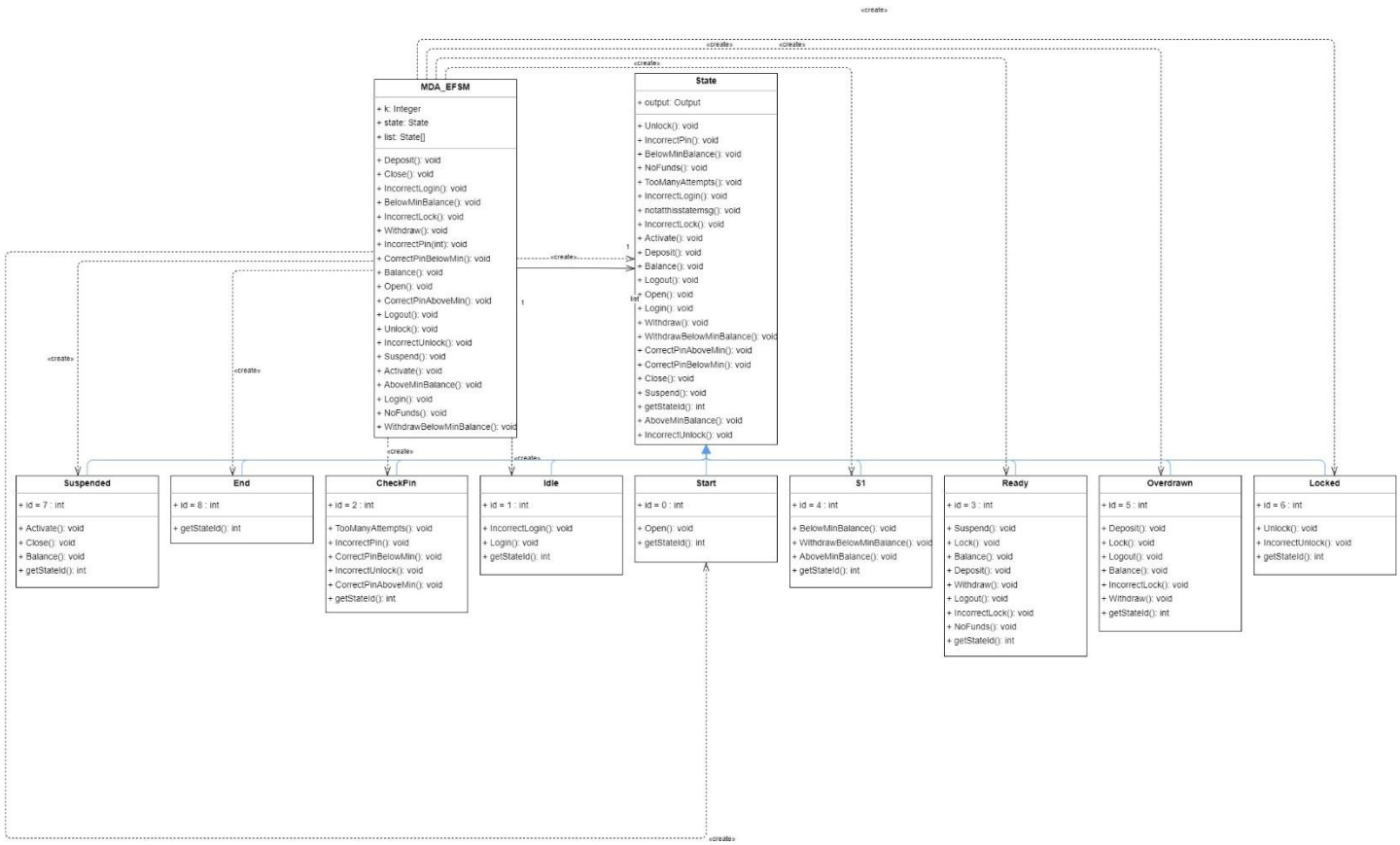
- Here are the methods of the ConcreteFactory1 class with the explanation.

```
    //return StoreData2.
    @Override
    public StoreData getStoreData() {
        return new StoreData2(this.data2);
    }
```

```java
    //return IncorrectIdMsg2
    @Override
    public IncorrectIdMsg getIncorrectIdMsg() {
        return new IncorrectIdMsg2(this.data2);
    }

    //return IncorrectPinMsg2
    @Override
    public IncorrectPinMsg getIncorrectPinMsg() {
        return new IncorrectPinMsg2(this.data2);
    }

    //return TooManyAttemptsMsg2
    @Override
    public TooManyAttemptsMsg getTooManyAttemptsMsg() {
        return new TooManyAttemptsMsg2(this.data2);
    }

    //return DisplayMenu2
    @Override
    public DisplayMenu getDisplayMenu() {
        return new DisplayMenu2(this.data2);
    }
    //return MakeDeposit2
    @Override
    public MakeDeposit getMakeDeposit() {
        return new MakeDeposit2(this.data2);
    }

    //return DisplayBalance2
    @Override
    public DisplayBalance getDisplayBalance() {
        return new DisplayBalance2(this.data2);
    }

    //return PromptForPin2
    @Override
    public PromptForPin getPromptForPin() {
        return new PromptForPin2(this.data2);
    }

    //return MakeWithdraw2
    @Override
    public MakeWithdraw getMakeWithdraw() {
        return new MakeWithdraw2(this.data2);
    }

    //return Penalty2
    @Override
    public Penalty getPenalty() {
```

```
        return new Penalty2(this.data2);
    }

    //return IncorrectLockMsg2
    @Override
    public IncorrectLockMsg getIncorrectLockMsg() {
        return new IncorrectLockMsg2(this.data2);
    }

    //return IncorrectUnlockMsg2
    @Override
    public IncorrectUnlockMsg getIncorrectUnlockMsg() {
        return new IncorrectUnlockMsg2(this.data2);
    }

    //return NoFundsMsg2
    @Override
    public NoFundsMsg getNoFundsMsg() {
        return new NoFundsMsg2(this.data2);
    }
}
```

3.4 MDA_EFSM Class



```
public class MDA_EFSM {

    public State list[];
    public State state;
    Integer k = 0;

    /**
     * Constructor of the MDA_EFSM class.
     * @param output - object of the output class
     */
    public MDA_EFSM(Output output){
        list = new State[9];
        list[0] = new Start(output);
        list[1] = new Idle(output);
        list[2] = new CheckPin(output);
        list[3] = new Ready(output);
        list[4] = new S1(output);
        list[5] = new Overdrawn(output);
        list[6] = new Locked(output);
        list[7] = new Suspended(output);
        list[8] = new End(output);
        state = list[0];
    }
```

- As specified in the above MDA_EFSM constructor, the different states are identified with unique IDs.
- 0 – Start state
  1 – Idle state
  2 – CheckPin state
  3 – Ready state
  4 – S1 state
  5 – Overdrawn state
  6 – Locked state
  7 – Suspended state
  8 – End state

3.4.1 Open()

```
    /**
     * This method calls open method from the state class.
     * State is changed from start to Idle.
     */
    public void Open(){
        state.Open();
        int stateid = state.getStateId();
        if(stateid == 0){
            state = list[1];
        }
    }
```

3.4.2 Login()

```
    /**
     * This method calls login method from the state class.
     * State is changed from Idle to CheckPin.
     */
    public void Login(){
        state.Login();
        int stateid = state.getStateId();
        if(stateid == 1){
            state = list[2];
        }
    }
```

3.4.3 IncorrectLogin()

```
    /**
     * This method calls IncorrectLogin method from the state class.
     */
    public void IncorrectLogin(){
        state.IncorrectLogin();
    }
```

3.4.4 IncorrectPin(int max)

```
    /**
     * This method calls IncorrectPin or TooManyAttempts methods from state class based on
the limit of maximum attempts.
     * If the limit to enter an incorrect pin is exceeded State is changed from CheckPin to
Idle.
     * @param Max - Maximum attempts possible for an incorrect pin.
     */
    public void IncorrectPin(int Max){
        if(k < Max){
            state.IncorrectPin();
            k=k+1;
        }
        else if(k == Max){
            state.TooManyAttempts();
            int stateid = state.getStateId();
            if(stateid == 2){
                state = list[1];
            }
        }
    }
```

3.4.5 CorrectPinBelowMin()

```
    /**
     * This method calls CorrectPinBelowMin method from state class.
     * State is changed from CheckPin to Overdrawn.
     */
```

```
    public void CorrectPinBelowMin(){
        state.CorrectPinBelowMin();
        int stateid = state.getStateId();
        if(stateid == 2){
            state = list[5];
        }
    }
```

3.4.6 CorrectPinAboveMin()

```
    /**
     * This method calls CorrectPinAboveMin method from state class.
     * State is changed from CheckPin to Ready
     */
    public void CorrectPinAboveMin(){
        state.CorrectPinAboveMin();
        int stateid = state.getStateId();
        if(stateid == 2){
            state = list[3];
        }
    }
```

3.4.7 Deposit()

```
    /**
     * This method calls Deposit method from state class.
     * State is changed from Overdrawn to S1.
     */
    public void Deposit(){
        state.Deposit();
        int stateid = state.getStateId();
        if(stateid == 5){
            state = list[4];
        }
    }
```

3.4.8 BelowMinBalance()

```
    /**
     * This method calls BelowMinBalance method from state class.
     * State is changed from S1 to Overdrawn.
     */
    public void BelowMinBalance(){
        state.BelowMinBalance();
        int stateid = state.getStateId();
        if(stateid == 4){
            state = list[5];
        }
    }
```

3.4.9 AboveMinBalance()

```
    /**
     * This method calls AboveMinBalance method from state class.
     * State is changed from S1 to Ready.
     */
    public void AboveMinBalance(){
        state.AboveMinBalance();
        int stateid = state.getStateId();
        if(stateid == 4){
            state = list[3];
        }
    }
```

3.4.10 Logout()

```
    /**
     * This method calls Logout method from state class.
     * State is changed from CheckPin, Ready, and Overdrawn to Idle.
     */
    public void Logout(){
        state.Logout();
        int stateid = state.getStateId();
        if(stateid == 2){
            state = list[1];
        }
        else if(stateid == 5){
            state = list[1];
        }
        else if(stateid == 3){
            state = list[1];
        }
    }
```

3.4.11 Balance()

```
    /**
     * This method calls Balance method from state class.
     */
    public void Balance(){
        state.Balance();
    }
```

3.4.12 Withdraw()

```
    /**
     * This method calls Withdraw method from state class.
     * State is changed from Ready to S1.
     */
    public void Withdraw(){
        state.Withdraw();
        int stateid = state.getStateId();
        if(stateid == 3){
```

```
            state = list[4];
        }
    }
```

3.4.13 WithdrawBelowMinBalance()

```
    /**
     * This method calls WithdrawBelowMinBalance method from state class.
     * State is changed from S1 to Overdrawn.
     */
    public void WithdrawBelowMinBalance(){
        state.WithdrawBelowMinBalance();
        int stateid = state.getStateId();
        if(stateid == 4){
            state = list[5];
        }
    }
```

3.4.14 NoFunds()

```
    /**
     * This method calls NoFunds method from state class.
     */
    public void NoFunds(){
        state.NoFunds();
    }
```

3.4.15 Lock()

```
    /**
     * This method calls Lock method from state class
     * State is changed from Ready, Overdrawn to Lock
     */
    public void Lock(){
        state.Lock();
        int stateid = state.getStateId();
        if(stateid == 3){
            state = list[6];
        }
        else if(stateid == 5){
            state = list[6];
        }
    }
```

3.4.16 IncorrectLock()

```
    /**
     * This method calls IncorrectLock method from state class.
     */
    public void IncorrectLock(){
        state.IncorrectLock();
    }
```

3.4.17 Unlock()

```
    /**
     * This method calls Unlock method from state class.
     * State is changed from Lock to S1.
     */
    public void Unlock(){
        state.Unlock();
        int stateid = state.getStateId();
        if(stateid == 6){
            state = list[4];
        }
    }
```

3.4.18 IncorrectUnlock()

```
    /**
     * This method calls IncorrectUnlock method from state class.
     */
    public void IncorrectUnlock(){
        state.IncorrectUnlock();
    }
```

3.4.19 Suspend()

```
    /**
     * This method calls Suspend method from state class.
     * State is changed from Ready to Suspended.
     */
    public void Suspend(){
        state.Suspend();
        int stateid = state.getStateId();
        if(stateid == 3){
            state = list[7];
        }
    }
```

3.4.20 Activate()

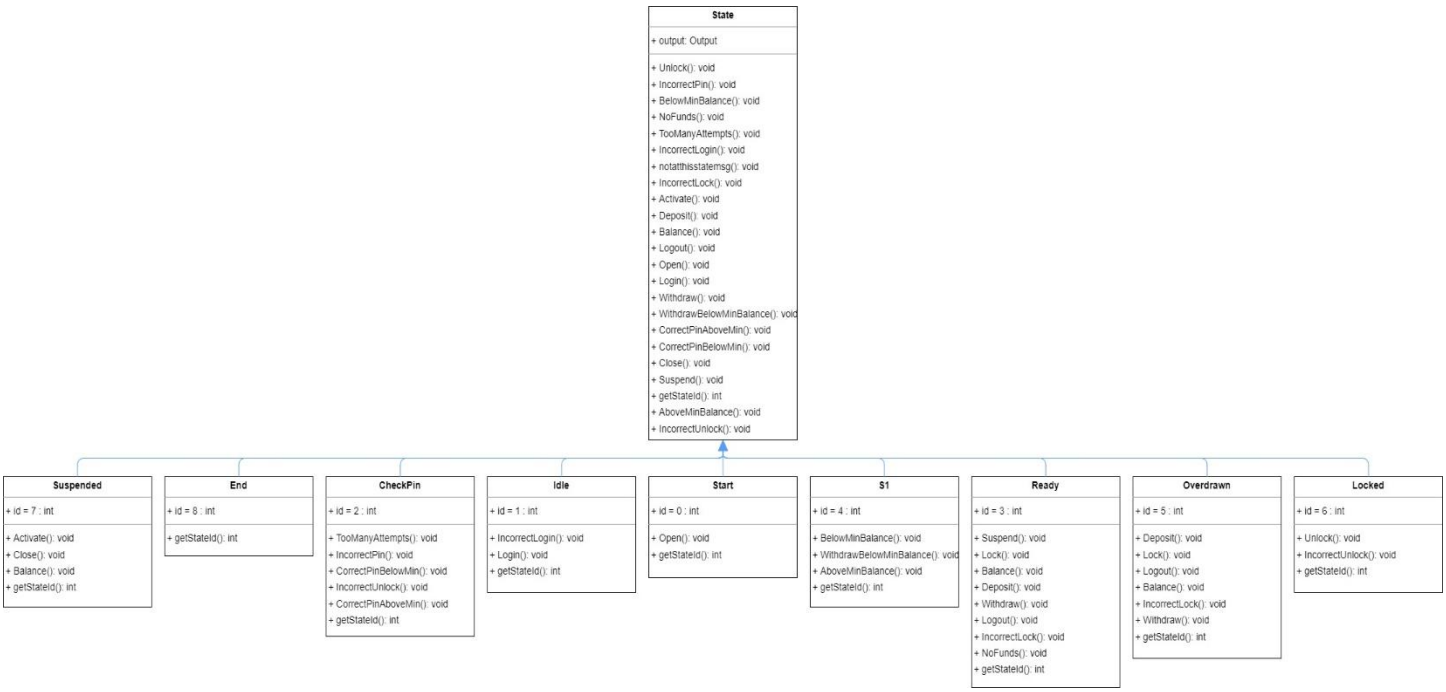
```
    /**
     * This method calls Activate method from state class.
     * State is changed from Suspended to Ready.
     */
    public void Activate(){
        state.Activate();
        int stateid = state.getStateId();
        if(stateid == 7){
            state = list[3];
        }
    }
```

3.4.21 Close()

```
    /**
```

```
     * This method calls Close method from state class.
     * State is changed from Suspended to End.
     */
    public void Close(){
        state.Close();
        int stateid = state.getStateId();
        if(stateid == 7){
            state = list[8];
        }
    }
}
```

3.5 State Classes



3.5.1 State Class

```
public State(Output output){
        this.output = output;
    }

    public abstract void Open();
    public abstract void Login();
    public abstract void IncorrectLogin();
    public abstract void IncorrectPin();
    public abstract void CorrectPinBelowMin();
    public abstract void CorrectPinAboveMin();
    public abstract void Deposit();
    public abstract void BelowMinBalance();
    public abstract void AboveMinBalance();
```

```
    public abstract void Logout();
    public abstract void Balance();
    public abstract void Withdraw();
    public abstract void WithdrawBelowMinBalance();
    public abstract void NoFunds();
    public abstract void Lock();
    public abstract void IncorrectLock();
    public abstract void Unlock();
    public abstract void IncorrectUnlock();
    public abstract void Suspend();
    public abstract void Activate();
    public abstract void Close();

    public abstract int getStateId();

    public abstract void TooManyAttempts();

    //This method calls when the user tries to perform the wrong action from a particular
state.
    public void notatthisstatemsg() {
        System.out.println("");
        System.out.println("WRONG ACTION !!! Cannot Perform From This Present State ");
    }
```

- As mentioned above, apart from the several abstract methods. notatthisstatemsg() method calls when a user tries to perform incorrect action from a particular state.

3.5.2 Start state Class

- ID = 0
- Open(): Invokes StoreData from output class.
- getStateId(): Return current state id.

```
/**
 * Start State Class
 * From this State Open() is Invoked.
 */
public class Start extends State{
    public int id = 0;

    public Start(Output output){
        super(output);
    }

    //StoreData() is invoked from the Output Class.
    @Override
    public void Open() {
        this.output.StoreData();
    }

    //return current state ID.
```

```
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.3 Idle state Class

- ID = 1
- Login(): Invokes PromptForPin() from output class.
- IncorrectLogin(): Invokes IncorrectIdMsg() from output class.
- getStateId(): Return current state Id.

```
/**
 * Idle State Class
 * From this State Login(), IncorrectLogin() are Invoked.
 */
public class Idle extends State{

    public int id = 1;

    public Idle(Output output) {
        super(output);
    }

    //PromptForPin() is invoked from Output class.
    @Override
    public void Login() {
        this.output.PromptForPin();
    }

    //IncorrectIdMsg() is invoked from Output class.
    @Override
    public void IncorrectLogin() {
        this.output.IncorrectIdMsg();
    }

    //return current State ID.
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.4 CheckPin State

- ID = 2
- IncorrectPin(): Invokes IncorrectPinMsg() from the output class.
- CorrectPinBelowMin(): Invokes DisplayMenu() from the output class.
- CorrectPinAboveMin(): Invokes DisplayMenu() from the output class.
- TooManyAttempts(): Invokes TooManyAttemptsMsg() from the output class.

- Logout(): No Action perform.
- getStateId(): Return current state Id

```
/**
 * Check Pin State Class
 * From this State IncorrectPin(), CorrectPinBelowMin(), CorrectPinAboveMin(),
TooManyAttempts(), Logout() are Invoked.
 */
public class CheckPin extends State{

    public int id = 2;

    public CheckPin(Output output) {
        super(output);
    }

    // IncorrectPinMsg() is invoked from Output class.
    @Override
    public void IncorrectPin() {
        this.output.IncorrectPinMsg();
    }

    // DisplayMenu() is invoked from Output class.
    @Override
    public void CorrectPinBelowMin() {
        this.output.DisplayMenu();
    }

    // DisplayMenu() is invoked from Output class.
    @Override
    public void CorrectPinAboveMin() {
        this.output.DisplayMenu();
    }

    @Override
    public void Logout() {
        //notatthisstatemsg();
    }

    //return current state ID.
    @Override
    public int getStateId() {
        return this.id;
    }

    //TooManyAttemptsMsg() is invoked from Output class.
    @Override
    public void TooManyAttempts() {
        this.output.TooManyAttemptsMsg();
    }
```

```
}
```

3.5.5 Ready State Class

- ID = 3
- Deposit(): Invokes MakeDeposit() from the output class.
- Balance(): Invokes DisplayBalance() from the output class.
- Withdraw(): Invokes MakeWithdraw() from the output class.
- NoFunds(): Invokes NoFundsMsg() from the output class.
- Logout(): No action perform.
- IncorrectLock(): Invokes IncorrectLockMsg() from the output class.
- Lock(): No action perform.
- Suspend(): No action perform.
- getStateId(): Return current state Id.

```
/**
 * Ready State Class
 * From this State Deposit(), Balance(), Withdraw(), NoFunds(), Logout(), IncorrectLock(),
Lock(), Suspend() are Invoked.
 */
public class Ready extends State{

    public Ready(Output output) {
        super(output);
    }

    public int id = 3;

    // MakeDeposit() is invoked from Output class.
    @Override
    public void Deposit() {
        this.output.MakeDeposit();
    }

    @Override
    public void Logout() {
        //notatthisstatemsg();
    }

    // DisplayBalance() is invoked from Output class.
    @Override
    public void Balance() {
        this.output.DisplayBalance();
    }

    // MakeWithdraw() is invoked from Output class.
    @Override
    public void Withdraw() {
        this.output.MakeWithdraw();
    }
```

```
    // NoFundsMsg() is invoked from Output class.
    @Override
    public void NoFunds() {
        this.output.NoFundsMsg();
    }

    @Override
    public void Lock() {
        //notatthisstatemsg();
    }

    // IncorrectLockMsg() is invoked from Output class.
    @Override
    public void IncorrectLock() {
        this.output.IncorrectLockMsg();
    }

    @Override
    public void Suspend() {
        //notatthisstatemsg();
    }

    //return current state ID.
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.6 S1 State Class

- ID = 4
- BelowMinBalance(): No Action perform.
- AboveMinBalance(): No Action perform.
- WithdrawBelowMinBalance(): Invokes Penalty() from output class.
- getStateId(): Return current state Id.

```
/**
 * S1 State Class
 * From this State BelowMinBalance(), AboveMinBalance(), WithdrawBelowMinBalance() are
Invoked.
 */
public class S1 extends State{

    public int id = 4;

    public S1(Output output) {
        super(output);
    }
```

```
    @Override
    public void BelowMinBalance() {
        //notatthisstatemsg();
    }

    @Override
    public void AboveMinBalance() {
        //notatthisstatemsg();
    }

    //Penalty() is invoked from Output class.
    @Override
    public void WithdrawBelowMinBalance() {
        this.output.Penalty();
    }

    //return current state ID.
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.7 Overdrawn State Class

- ID = 5
- Deposit(): Invokes MakeDeposit() from output class.
- Balance(): Invokes DisplayBalance() from output class.
- Withdraw(): Invokes MakeWithdraw() from the output class.
- IncorrectLock(): Invokes IncorrectLockMsg() from output class.
- Lock(): No Action Perform.
- Logout(): No Action Perform.
- getStateId(): Return current state Id.

```
/**
 * Overdrawn State Class
 * From this State Deposit(), Balance(), Withdraw(), IncorrectLock(), Lock(), Logout() are
Invoked.
 */
public class Overdrawn extends State{

    public int id = 5;

    public Overdrawn(Output output) {
        super(output);
    }

    // MakeDeposit() is invoked from Output class.
    @Override
    public void Deposit() {
        this.output.MakeDeposit();
```

```
    }

    @Override
    public void Logout() {
        //notatthisstatemsg();
    }

    // DisplayBalance() is invoked from Output class.
    @Override
    public void Balance() {
        this.output.DisplayBalance();
    }

    // NoFundsMsg() is invoked from Output class.
    @Override
    public void Withdraw() {
        this.output.NoFundsMsg();
    }

    @Override
    public void Lock() {
        //notatthisstatemsg();
    }

    // IncorrectLockMsg() is invoked from Output class.
    @Override
    public void IncorrectLock() {
        this.output.IncorrectLockMsg();
    }

    //return current state ID.
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.8 Locked State Class

- ID = 6
- IncorrectUnlock(): Invokes IncorrectUnlockMsg() from output class.
- Unlock(): No Action perform.
- getStateId(): Return current state Id.

```
/**
 * Locked State Class
 * From this State IncorrectUnlock(), Unlock() are Invoked.
 */
public class Locked extends State{

    public int id = 6;
```

```
    public Locked(Output output) {
        super(output);
    }

    @Override
    public void Unlock() {
        //notatthisstatemsg();
    }

    //IncorrectUnlockMsg() is invoked from Output class.
    @Override
    public void IncorrectUnlock() {
        this.output.IncorrectUnlockMsg();
    }

    //return current state ID.
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.9 Suspended State Class

- ID = 7
- Balance(): Invokes DisplayBalance() from output class.
- Activate(): No Action perform.
- Close(): No Action perform.
- getStateId(): Return current state Id.

```
/**
 * Suspended State Class
 * From this State Balance(), Activate(), Close() are Invoked.
 */
public class Suspended extends State{

    public int id = 7;

    public Suspended(Output output) {
        super(output);
    }

    //DisplayBalance is invoked from Output Class.
    @Override
    public void Balance() {
        this.output.DisplayBalance();
    }

    @Override
    public void Activate() {
```

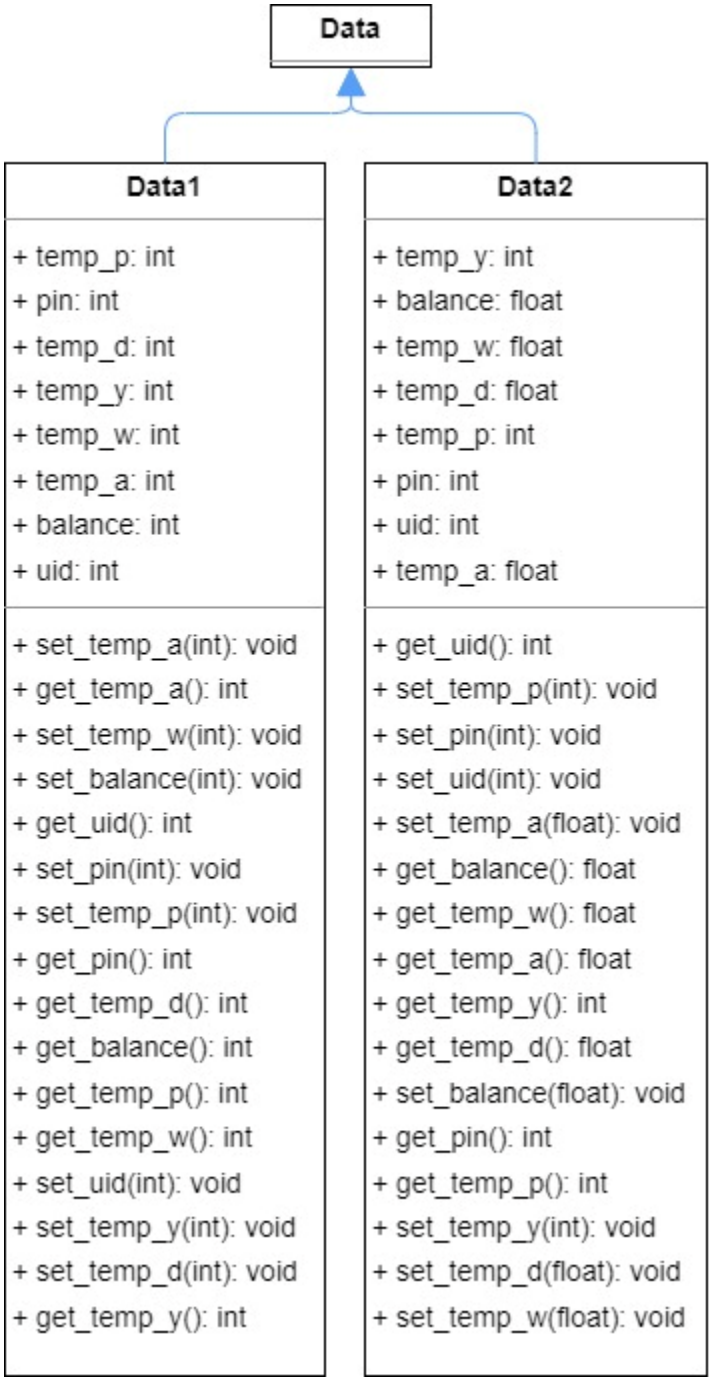
```
        //notatthisstatemsg();
    }

    @Override
    public void Close() {
        //notatthisstatemsg();
    }

    //return current state ID.
    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.5.10 End State Class

- ID = 8
- Nothing is invoked from the current state.
- getStateId(): Return current state Id.

```
/**
 * Check Pin State Class
 * From this State, no other method is Invoked.
 */
public class End extends State{

    public int id = 8;
    public End(Output output) {
        super(output);
    }

    @Override
    public int getStateId() {
        return this.id;
    }
}
```

3.6 Data Classes



- These storage classes contain getter and setter methods for data.

3.6.1 Data class

```
/**
 * Data Abstract class
 */
public abstract class Data {

}
```

3.6.2 Data1 class

```
/**
 * Data1 is a child class of the Data class.
 * Data1 class is used by Account1.
 * Data1 has getter setter methods utilized in Account1 and Output class.
 */
public class Data1 extends Data{
    private int pin;
    private int balance;
    private int uid;

    private int temp_p;
    private int temp_y;
    private int temp_a;
    private int temp_d;
    private int temp_w;

    //get pin number
    public int get_pin(){
        return pin;
    }

    //set pin number
    public void set_pin(int pin){
        this.pin = pin;
    }

    //get account balance
    public int get_balance(){
        return balance;
    }

    //set balance
    public void set_balance(int balance){
        this.balance = balance;
    }

    //get user identification number
    public int get_uid(){
        return uid;
    }

    //set user identification number
    public void set_uid(int uid){
        this.uid = uid;
    }

    //get value of temporary variable temp_p (for pin).
    public int get_temp_p(){
        return temp_p;
```

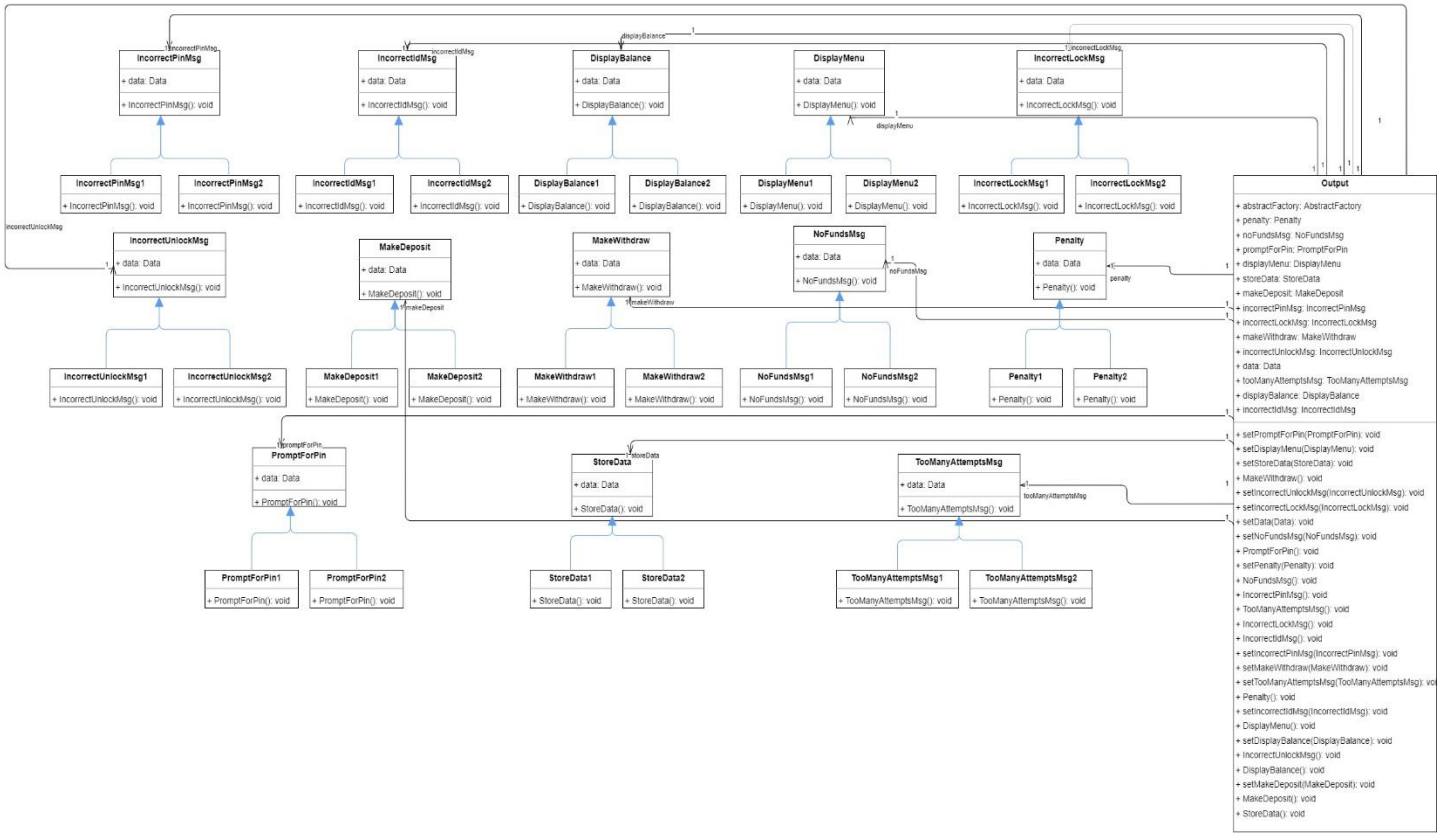
```
    }

    //set value of temporary variable temp_p (for pin).
    public void set_temp_p(int temp_p){
        this.temp_p = temp_p;
    }

    //get value of temporary variable temp_y (for userId).
    public int get_temp_y(){
        return temp_y;
    }

    //set value of temporary variable temp_y (for userId).
    public void set_temp_y(int temp_y){
        this.temp_y = temp_y;
    }

    //get value of temporary variable temp_a (for balance)
    public int get_temp_a(){
        return temp_a;
    }

    //set value of temporary variable temp_a (for balance)
    public void set_temp_a(int temp_a){
        this.temp_a = temp_a;
    }

    //get value of temporary variable temp_d (for deposit amount)
    public int get_temp_d(){
        return temp_d;
    }

    //set value of temporary variable temp_d (for deposit amount)
    public void set_temp_d(int temp_d){
        this.temp_d = temp_d;
    }

    //get value of temporary variable temp_w (for withdraw amount)
    public int get_temp_w(){
        return temp_w;
    }

    //set value of temporary variable temp_w (for withdraw amount)
    public void set_temp_w(int temp_w){
        this.temp_w = temp_w;
    }
}
```

3.6.3 Data2 class.

```
/**
 * Data2 is child class of the Data class.
 * Data2 class is used by the Account1.
 * Data2 has getter setter methods utilized in Account1 and Output class.
 */
public class Data2 extends Data{
    private int pin;
    private float balance;
    private int uid;

    private int temp_p;
    private int temp_y;
    private float temp_a;
    private float temp_d;
    private float temp_w;

    //get pin number
    public int get_pin(){
        return pin;
    }

    //set pin number
    public void set_pin(int pin){
        this.pin = pin;
    }

    //get account balance
    public float get_balance(){
        return balance;
    }

    //set balance
    public void set_balance(float balance){
        this.balance = balance;
    }

    //get user identification number
    public int get_uid(){
        return uid;
    }

    //set user identification number
    public void set_uid(int uid){
        this.uid = uid;
    }

    //get value of temporary variable temp_p (for pin).
    public int get_temp_p(){
        return temp_p;
```

```
    }

    //set value of temporary variable temp_p (for pin).
    public void set_temp_p(int temp_p){
        this.temp_p = temp_p;
    }

    //get value of temporary variable temp_y (for userId).
    public int get_temp_y(){
        return temp_y;
    }

    //set value of temporary variable temp_y (for userId).
    public void set_temp_y(int temp_y){
        this.temp_y = temp_y;
    }

    //get value of temporary variable temp_a (for balance)
    public float get_temp_a(){
        return temp_a;
    }

    //set value of temporary variable temp_a (for balance)
    public void set_temp_a(float temp_a){
        this.temp_a = temp_a;
    }

    //get value of temporary variable temp_d (for deposit amount)
    public float get_temp_d(){
        return temp_d;
    }

    //set value of temporary variable temp_d (for deposit amount)
    public void set_temp_d(float temp_d){
        this.temp_d = temp_d;
    }

    //get value of temporary variable temp_w (for withdraw amount)
    public float get_temp_w(){
        return temp_w;
    }

    //set value of temporary variable temp_w (for withdraw amount)
    public void set_temp_w(float temp_w){
        this.temp_w = temp_w;
    }
}
```

3.7 Strategy Classes



3.7.1 Output class

- Output class implements all the above-mentioned methods, it gets the object from the ConcreteFactory1 and ConcreteFactory2 and performs operations separately for Account1 and Account2 classes.
- The variable and methods used by the output class are mentioned in the above image.

```
public class Output {
    public AbstractFactory abstractFactory;

    public Data data;
    public StoreData storeData;
    public IncorrectIdMsg incorrectIdMsg;
    public IncorrectPinMsg incorrectPinMsg;
    public TooManyAttemptsMsg tooManyAttemptsMsg;
    public DisplayMenu displayMenu;
    public MakeDeposit makeDeposit;
    public DisplayBalance displayBalance;
    public PromptForPin promptForPin;
    public MakeWithdraw makeWithdraw;
    public Penalty penalty;
    public IncorrectLockMsg incorrectLockMsg;
    public IncorrectUnlockMsg incorrectUnlockMsg;
    public NoFundsMsg noFundsMsg;

    public Output(AbstractFactory abstractFactory){
        this.abstractFactory = abstractFactory;
```

```
        /*
         * Here Get method gets the Object and Sets it using set method
         */
        setData(abstractFactory.getData());
        setStoreData(abstractFactory.getStoreData());
        setIncorrectIdMsg(abstractFactory.getIncorrectIdMsg());
        setIncorrectPinMsg(abstractFactory.getIncorrectPinMsg());
        setTooManyAttemptsMsg(abstractFactory.getTooManyAttemptsMsg());
        setDisplayMenu(abstractFactory.getDisplayMenu());
        setMakeDeposit(abstractFactory.getMakeDeposit());
        setDisplayBalance(abstractFactory.getDisplayBalance());
        setPromptForPin(abstractFactory.getPromptForPin());
        setMakeWithdraw(abstractFactory.getMakeWithdraw());
        setPenalty(abstractFactory.getPenalty());
        setIncorrectLockMsg(abstractFactory.getIncorrectLockMsg());
        setIncorrectUnlockMsg(abstractFactory.getIncorrectUnlockMsg());
        setNoFundsMsg(abstractFactory.getNoFundsMsg());
    }

    private void setData(Data data) {
    }

    //Store data
    public void StoreData(){
        this.storeData.StoreData();
    }

    private void setStoreData(StoreData storeData) {
        this.storeData = storeData;
    }

    //IncorrectIdMsg
    public void IncorrectIdMsg(){
        this.incorrectIdMsg.IncorrectIdMsg();
    }

    private void setIncorrectIdMsg(IncorrectIdMsg incorrectIdMsg) {
        this.incorrectIdMsg = incorrectIdMsg;
    }

    //IncorrectPinMsg
    public void IncorrectPinMsg(){
        this.incorrectPinMsg.IncorrectPinMsg();
    }

    private void setIncorrectPinMsg(IncorrectPinMsg incorrectPinMsg) {
        this.incorrectPinMsg = incorrectPinMsg;
    }

    //TooManyAttemptsMsg
```

```
    public void TooManyAttemptsMsg(){
        this.tooManyAttemptsMsg.TooManyAttemptsMsg();
    }

    private void setTooManyAttemptsMsg(TooManyAttemptsMsg tooManyAttemptsMsg) {
        this.tooManyAttemptsMsg = tooManyAttemptsMsg;
    }

    //DisplayMenu
    public void DisplayMenu(){
        this.displayMenu.DisplayMenu();
    }

    private void setDisplayMenu(DisplayMenu displayMenu) {
        this.displayMenu = displayMenu;
    }

    //MakeDeposit
    public void MakeDeposit(){
        this.makeDeposit.MakeDeposit();
    }

    private void setMakeDeposit(MakeDeposit makeDeposit) {
        this.makeDeposit = makeDeposit;
    }

    //DisplayBalance
    public void DisplayBalance(){
        this.displayBalance.DisplayBalance();
    }

    private void setDisplayBalance(DisplayBalance displayBalance) {
        this.displayBalance = displayBalance;
    }

    //PromptForPin
    public void PromptForPin(){
        this.promptForPin.PromptForPin();
    }

    private void setPromptForPin(PromptForPin promptForPin) {
        this.promptForPin = promptForPin;
    }

    //MakeWithdraw
    public void MakeWithdraw(){
        this.makeWithdraw.MakeWithdraw();
    }

    private void setMakeWithdraw(MakeWithdraw makeWithdraw) {
```

```
        this.makeWithdraw = makeWithdraw;
    }

    //Penalty
    public void Penalty(){
        this.penalty.Penalty();
    }

    private void setPenalty(Penalty penalty) {
        this.penalty = penalty;
    }

    //IncorrectLockMsg
    public void IncorrectLockMsg(){
        this.incorrectLockMsg.IncorrectLockMsg();
    }

    private void setIncorrectLockMsg(IncorrectLockMsg incorrectLockMsg) {
        this.incorrectLockMsg=incorrectLockMsg;
    }

    //IncorrectUnlockMsg
    public void IncorrectUnlockMsg(){
        this.incorrectUnlockMsg.IncorrectUnlockMsg();
    }

    private void setIncorrectUnlockMsg(IncorrectUnlockMsg incorrectUnlockMsg) {
        this.incorrectUnlockMsg = incorrectUnlockMsg;
    }

    //NoFundsMsg
    public void NoFundsMsg(){
        this.noFundsMsg.NoFundsMsg();
    }

    private void setNoFundsMsg(NoFundsMsg noFundsMsg) {
        this.noFundsMsg = noFundsMsg;
    }
}
```

3.7.2 DisplayBalance, DisplayBalance1, DisplayBalance2 Classes

```
//Abstract Class DisplayBalance
public abstract class DisplayBalance {
    Data data;

    public DisplayBalance(Data data){
        this.data = data;
    }
```

```
    public abstract void DisplayBalance();
}
```

```
//DisplayBalance1 class belongs to Account-2
public class DisplayBalance1 extends DisplayBalance{

    public DisplayBalance1(Data data) {
        super(data);
    }
    /**
     * This method display balance for Account-1.
     */
    @Override
    public void DisplayBalance() {
        Data1 data1 = (Data1)data;
        System.out.println("");
        System.out.println(">>>> Your Account Balance is: "+data1.get_balance());
        System.out.println("");
    }
}
```

```
//DisplayBalance2 class belongs to Account-2
public class DisplayBalance2 extends DisplayBalance{

    public DisplayBalance2(Data data) {
        super(data);
    }

    /**
     * This method display balance for Account-2.
     */
    @Override
    public void DisplayBalance() {
        Data2 data2 = (Data2)data;
        System.out.println("");
        System.out.println(">>>> Your Account Balance is: "+data2.get_balance());
        System.out.println("");
    }
}
```

3.7.3 DisplayMenu, DisplayMenu1, DisplayMenu2 Classes

```
public abstract class DisplayMenu {
    Data data;

    public DisplayMenu(Data data){
        this.data = data;
    }
```

```
    public abstract void DisplayMenu();
}
```

```
public class DisplayMenu1 extends DisplayMenu{

    public DisplayMenu1(Data data) {
        super(data);
    }

    @Override
    public void DisplayMenu() {
        System.out.println(">>>> Available Options: ");
        System.out.print("\n");
        System.out.print("     2. deposit(int)");
        System.out.print("\n");
        System.out.print("     3. withdraw(int)");
        System.out.print("\n");
        System.out.print("     4. balance()");
        System.out.print("\n");
        System.out.print("     6. logout()");
        System.out.print("\n");
        System.out.print("     7. lock(int)");
        System.out.print("\n");
    }
}
```

```
public class DisplayMenu2 extends DisplayMenu{

    public DisplayMenu2(Data data) {
        super(data);
    }

    @Override
    public void DisplayMenu() {
        System.out.println(">>>> Available Options: ");
        System.out.print("\n");
        System.out.print("     2. deposit(float)");
        System.out.print("\n");
        System.out.print("     3. withdraw(float)");
        System.out.print("\n");
        System.out.print("     4. balance()");
        System.out.print("\n");
        System.out.print("     6. logout()");
        System.out.print("\n");
        System.out.print("     7. suspend()");
        System.out.print("\n");
    }
}
```

3.7.4 IncorrectIdMsg, IncorrectIdMsg1, IncorrectIdMsg2 Classes

```
//IncorrectIdMsg Abstract class
public abstract class IncorrectIdMsg {
    Data data;
    public IncorrectIdMsg(Data data){
        this.data = data;
    }

    public abstract void IncorrectIdMsg();
}
```

```
//IncorrectIdMsg1 class belongs to Account-1
public class IncorrectIdMsg1 extends IncorrectIdMsg{

    public IncorrectIdMsg1(Data data) {
        super(data);
    }

    //This method display incorrect ID message for Account-1
    @Override
    public void IncorrectIdMsg() {
        System.out.println("");
        System.out.println(">>>> InCorrect UserId !!! Try Again");
        System.out.println("");
    }

}
```

```
//IncorrectIdMsg2 class belongs to Account-2
public class IncorrectIdMsg2 extends IncorrectIdMsg{

    public IncorrectIdMsg2(Data data) {
        super(data);
    }

    /**
     * This method display incorrect ID message for Account-2
     */
    @Override
    public void IncorrectIdMsg() {
        System.out.println("");
        System.out.println(">>>> InCorrect UserId !!! Try Again");
        System.out.println("");
    }

}
```

3.7.5 IncorrectLockMsg, IncorrectLockMsg1, IncorrectLockMsg2 Classes

```
//IncorrectLockMsg Abstract class
public abstract class IncorrectLockMsg {
    Data data;

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

    public abstract void IncorrectLockMsg();
}
```

```
//IncorrectLockMsg1 class belongs to Accounut-1
public class IncorrectLockMsg1 extends IncorrectLockMsg{

    public IncorrectLockMsg1(Data data) {
        super(data);
    }

    //This method incorrect lock message for Account-1
    @Override
    public void IncorrectLockMsg() {
        System.out.println("");
        System.out.println(">>>> Account is not locked due to InCorrect Pin !!! Try Again");
        System.out.println("");
    }
}
```

```
//IncorrectLockMsg2 class belongs to Account-2
public class IncorrectLockMsg2 extends IncorrectLockMsg{

    public IncorrectLockMsg2(Data data) {
        super(data);
    }

    //No operation is performed to show Incorrect Lock Message for Account-2.
    @Override
    public void IncorrectLockMsg() {

    }
}
```

3.7.6 IncorrectPinMsg, IncorrectPinMsg1, IncorrectPinMsg2 Classes

```
//IncorrectPinMsg Abstract class.
public abstract class IncorrectPinMsg{
    Data data;
```

```
    public IncorrectPinMsg(Data data){
        this.data = data;
    }

    public abstract void IncorrectPinMsg();
}
```

```
public class IncorrectPinMsg1 extends IncorrectPinMsg{

    public IncorrectPinMsg1(Data data) {
        super(data);
    }

    //This method display incorrect Pin message for Account-1
    @Override
    public void IncorrectPinMsg() {
        System.out.println("");
        System.out.println(">>>> InCorrect Pin !!! Try Again");
        System.out.println("");
    }
}
```

```
public class IncorrectPinMsg2 extends IncorrectPinMsg{

    public IncorrectPinMsg2(Data data) {
        super(data);
    }

    //This method display incorrect Pin message for Account-2.
    @Override
    public void IncorrectPinMsg() {
        System.out.println("");
        System.out.println(">>>> InCorrect Pin !!! Try Again");
        System.out.println("");
    }
}
```

3.7.7 IncorrectUnlockMsg, IncorrectUnlockMsg1, IncorrectUnlockMsg2 Classes

```
//IncorrectUnlockMsg Abstract class.
public abstract class IncorrectUnlockMsg {
    Data data;

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

    public abstract void IncorrectUnlockMsg();
}
```

```
public class IncorrectUnlockMsg1 extends IncorrectUnlockMsg{

    public IncorrectUnlockMsg1(Data data) {
        super(data);
    }

    //This method display incorrect unlock message for Account-1.
    @Override
    public void IncorrectUnlockMsg() {
        System.out.println("");
        System.out.println(">>>> Account unlock is not possible due to InCorrect Pin !!! Try
Again");
        System.out.println("");
    }
}
```

```
public class IncorrectUnlockMsg2 extends IncorrectUnlockMsg{

    public IncorrectUnlockMsg2(Data data) {
        super(data);
    }

    //No operation is performed to Show Incorrect unlock Message for Account-2.
    @Override
    public void IncorrectUnlockMsg() {

    }

}
```

3.7.8 MakeDeposit, MakeDeposit1, MakeDeposit2 Classes

```
//MakeDeposit Abstract Class
public abstract class MakeDeposit {
    Data data;

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

    public abstract void MakeDeposit();
}
```

```
public class MakeDeposit1 extends MakeDeposit {

    public MakeDeposit1(Data data) {
```

```
        super(data);
    }

    //This method performs deposit operation for Account-1.
    @Override
    public void MakeDeposit() {
        Data1 data1 = (Data1)data;
        int deposit = data1.get_temp_d();
        data1.set_balance(data1.get_balance()+deposit);
        System.out.println("");
        System.out.println(">>>> Amount is deposited successfully !!");
        System.out.println("");
    }
}
```

```
public class MakeDeposit2 extends MakeDeposit{

    public MakeDeposit2(Data data) {
        super(data);
    }

    //This method performs deposit operation for Account-2.
    @Override
    public void MakeDeposit() {
        Data2 data2 = (Data2)data;
        float deposit = data2.get_temp_d();
        data2.set_balance(data2.get_balance()+deposit);
        System.out.println("");
        System.out.println(">>>> Amount is deposited successfully !!");
        System.out.println("");
    }
}
```

3.7.9 MakeWithdraw, MakeWithdraw1, MakeWithdraw2 Classes

```
//MakeWithdraw Abstract Class.
public abstract class MakeWithdraw {
    Data data;

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

    public abstract void MakeWithdraw();
}
```

```
public class MakeWithdraw1 extends MakeWithdraw{

    public MakeWithdraw1(Data data) {
```

```
        super(data);
    }

    //This method performs withdraw operation for Account-1.
    @Override
    public void MakeWithdraw() {
        Data1 data1 = (Data1)data;
        int withdraw = data1.get_temp_w();
        data1.set_balance(data1.get_balance()-withdraw);
        System.out.println("");
        System.out.println(">>>> Amount is withdrawed successfully !!");
        System.out.println("");
    }

}
```

```
public class MakeWithdraw2 extends MakeWithdraw{

    public MakeWithdraw2(Data data) {
        super(data);
    }

    //This method performs withdraw operation for Account-2.
    @Override
    public void MakeWithdraw() {
        Data2 data2 = (Data2)data;
        float withdraw = data2.get_temp_w();
        data2.set_balance(data2.get_balance()-withdraw);
        System.out.println("");
        System.out.println(">>>> Amount is withdrawed successfully !!");
        System.out.println("");
    }
}
```

3.7.10 NoFundsMsg, NoFundsMsg1, NoFundsMsg2 Classes

```
//NoFundsMsg Abstract Class.
public abstract class NoFundsMsg {
    Data data;

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

    public abstract void NoFundsMsg();
}
```

```
public class NoFundsMsg1 extends NoFundsMsg{
```

```
    public NoFundsMsg1(Data data) {
        super(data);
    }

    //No operation is performed to show No fund message for Account-1.
    @Override
    public void NoFundsMsg() {

    }

}
```

```
public class NoFundsMsg2 extends NoFundsMsg{

    public NoFundsMsg2(Data data) {
        super(data);
    }

    //This method display no fund alert message for Account-2.
    @Override
    public void NoFundsMsg() {
        System.out.println("");
        System.out.println(">>>> No fund Available in the Account !!!");
        System.out.println("");
    }

}
```

3.7.11 Penalty, Penalty1, Penalty2 Classes

```
//Penalty Abstract Class.
public abstract class Penalty {
    Data data;

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

    public abstract void Penalty();
}
```

```
public class Penalty1 extends Penalty{

    public Penalty1(Data data) {
        super(data);
    }

    //$15 penalty is applied when user's account balance is below $100 for Account-1.
    @Override
```

```
    public void Penalty() {
        Data1 data1 = (Data1)data;
        int balance = data1.get_balance();
        if(balance <=100){
            data1.set_balance(balance-15);
        }
        else{
            data1.set_balance(balance);
        }
        System.out.println("");
        System.out.println(">>>> $15 Penalty is applied !!");
        System.out.println("");
    }

}
```

```
public class Penalty2 extends Penalty{

    public Penalty2(Data data) {
        super(data);
    }

    //No penalty is applied to Account-2.
    @Override
    public void Penalty() {

    }

}
```

3.7.12 PromptForPin, PromptForPin1, PromptForPin2 Classes

```
//PromptForPin Abstract Class.
public abstract class PromptForPin {
    Data data;

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

    public abstract void PromptForPin();
}
```

```
public class PromptForPin1 extends PromptForPin{

    public PromptForPin1(Data data) {
        super(data);
    }
```

```
    //Prompt message to enter pin is appeared after successful login for Account-1.
    @Override
    public void PromptForPin() {
        System.out.println(">>>> Available Options: ");
        System.out.print("\n");
        System.out.print("1. Pin(int)");
        System.out.print("\n");
    }
}
```

```
public class PromptForPin2 extends PromptForPin{

    public PromptForPin2(Data data) {
        super(data);
    }

    ////Prompt message to enter pin is appeared after successful login for Account-2.
    @Override
    public void PromptForPin() {
        System.out.println(">>>> Available Options: ");
        System.out.print("\n");
        System.out.print("1. Pin(int)");
        System.out.print("\n");
    }

}
```

3.7.13 StoreData, StoreData1, StoreData2 Classes

```
//StoreData Abstract Class.
public abstract class StoreData {
    Data data;

    public StoreData(Data data){
        this.data=data;
    }
    public abstract void StoreData();
}
```

```
public class StoreData1 extends StoreData{

    public StoreData1(Data data) {
        super(data);
    }

    //This method store temporary variable for pin, user Id, and balance for Account-1.
    @Override
    public void StoreData() {
        Data1 data1 = (Data1)data;
```

```
        data1.set_pin(data1.get_temp_p());
        data1.set_balance(data1.get_temp_a());
        data1.set_uid(data1.get_temp_y());
    }
}
```

```
public class StoreData2 extends StoreData{

    public StoreData2(Data data) {
        super(data);
    }

    //This method store temporary variable for pin, user Id, and balance for Account-2.
    @Override
    public void StoreData() {
        Data2 data2 = (Data2)data;
        data2.set_pin(data2.get_temp_p());
        data2.set_balance(data2.get_temp_a());
        data2.set_uid(data2.get_temp_y());
    }
}
```

3.7.14 TooManyAttemptsMsg, TooManyAttemptsMsg1, TooManyAttemptsMsg2 Classes

```
//TooManyAttemptsMsg Abstract Class.
public abstract class TooManyAttemptsMsg {
    Data data;

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

    public abstract void TooManyAttemptsMsg();
}
```

```
public class TooManyAttemptsMsg1 extends TooManyAttemptsMsg{

    public TooManyAttemptsMsg1(Data data) {
        super(data);
    }

    //This method display message when Maximum limit for Incorrect pin is exceeded for
Account-1.
    @Override
    public void TooManyAttemptsMsg() {
        System.out.println("");
        System.out.println(">>>> Maximum limit for Incorrect pin is exceeded !!");
        System.out.println("");
    }
```

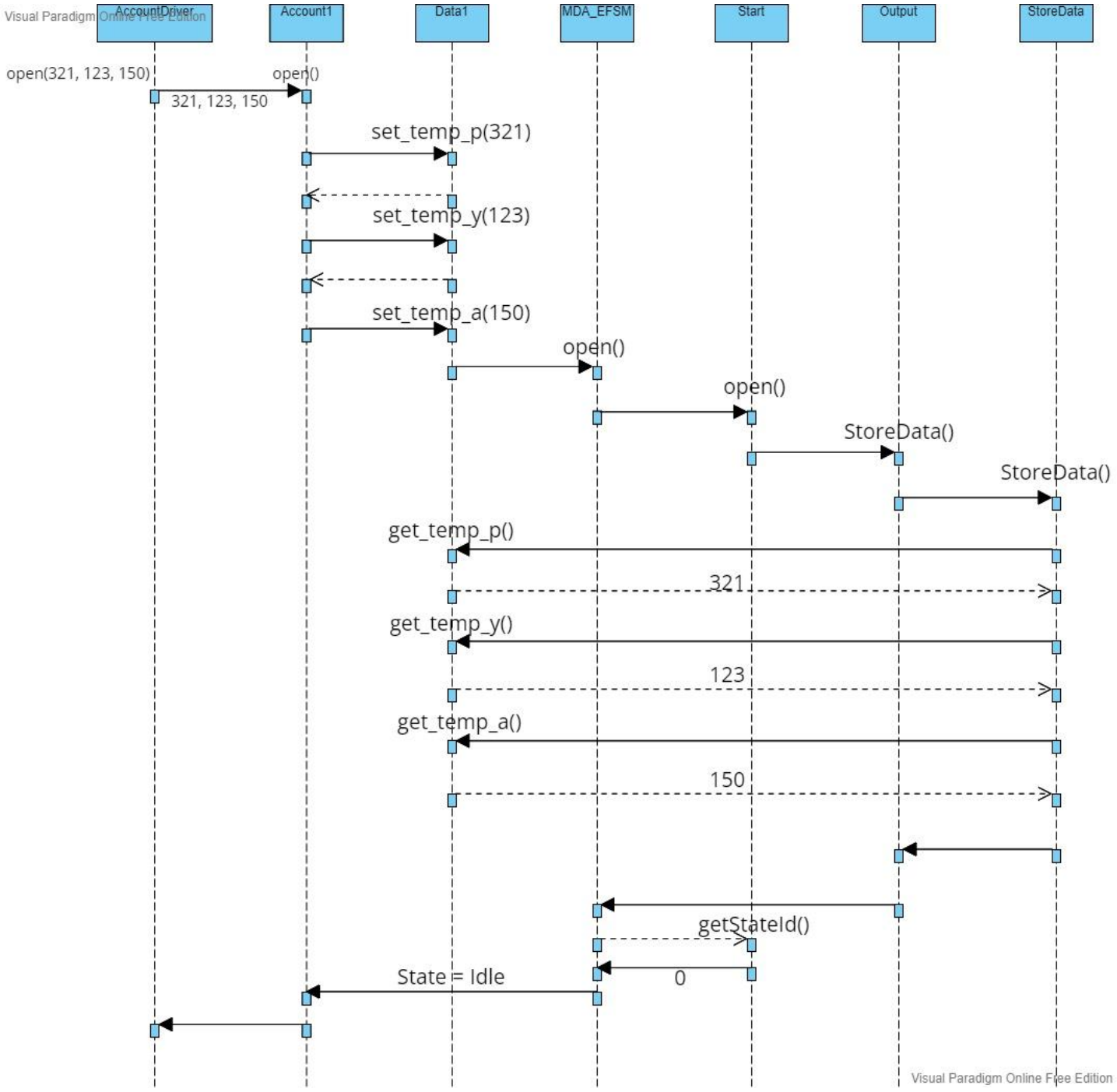
```
}
```

```
public class TooManyAttemptsMsg2 extends TooManyAttemptsMsg{

    public TooManyAttemptsMsg2(Data data) {
        super(data);
    }

    //This method display message when Maximum limit for Incorrect pin is exceeded for
Account-2.
    @Override
    public void TooManyAttemptsMsg() {
        System.out.println("");
        System.out.println(">>>> Maximum limit for Incorrect pin is exceeded !!");
        System.out.println("");
    }

}
```

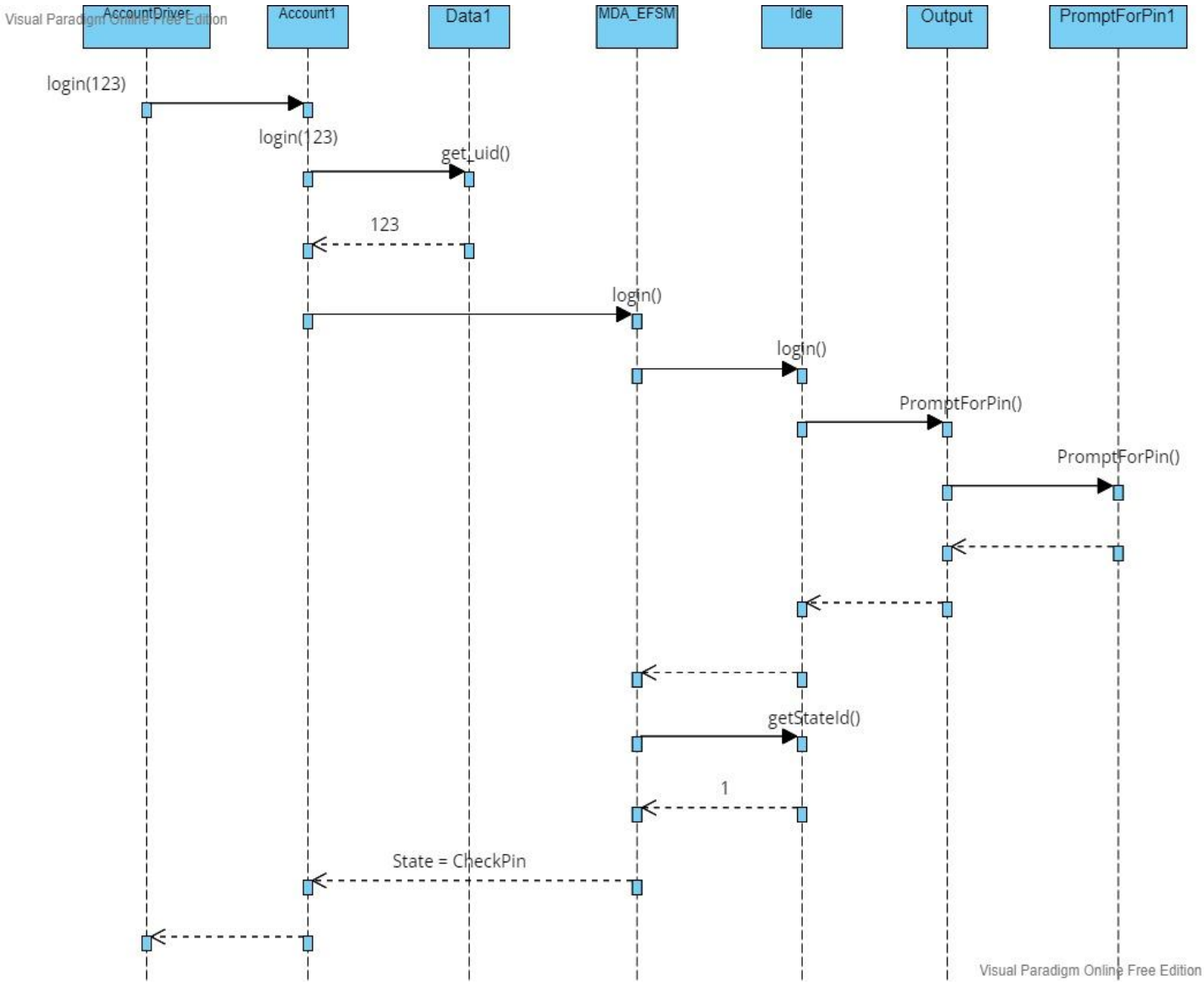
3.8 Sequence Diagram

- Dynamics. Provide a sequence diagrams for ACCOUNT-1 for the following sequence of operations:
- open(321,123,150), login(123), pin(321), withdraw(70), balance(), logout()

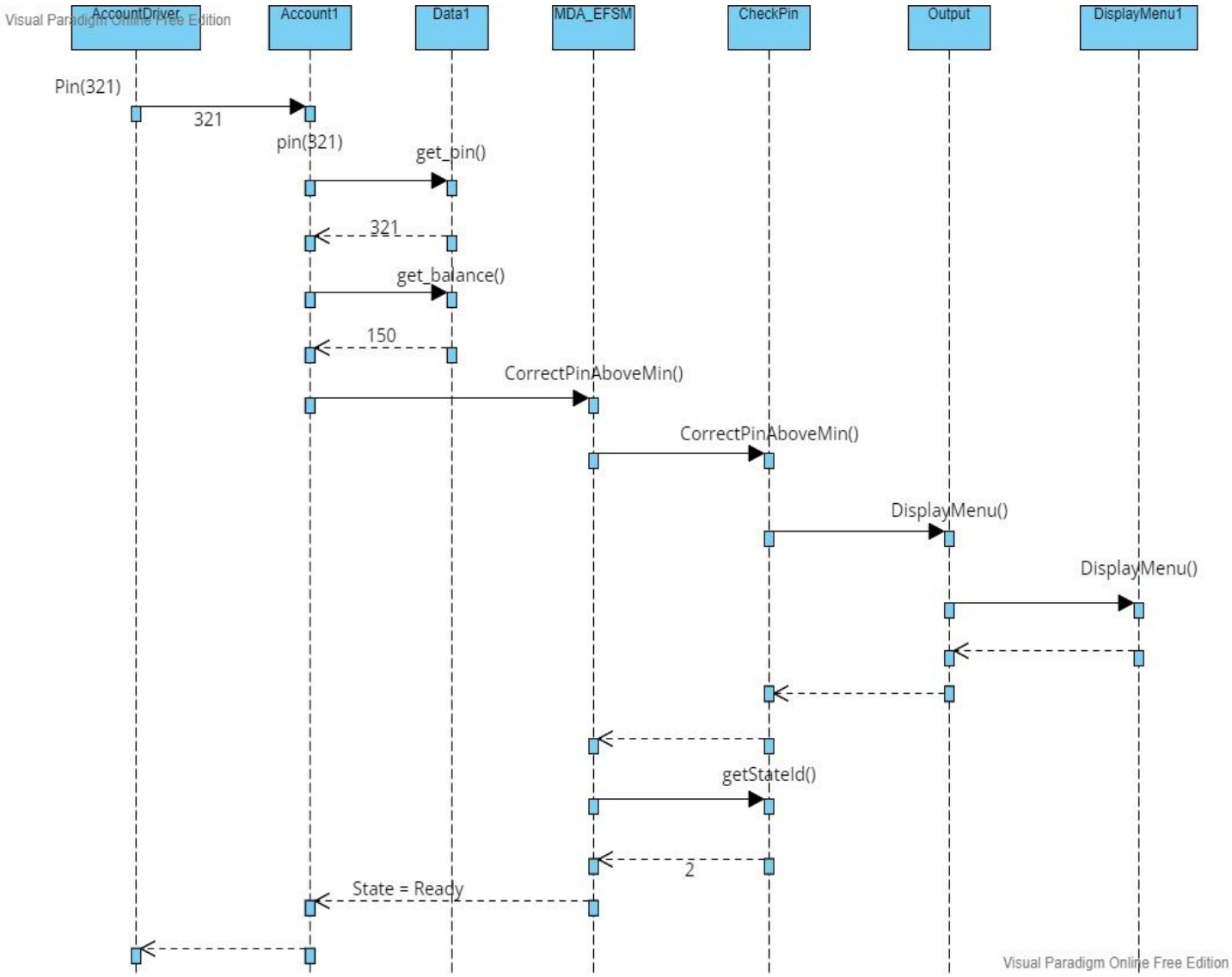
3.8.1 open(321,123,150)



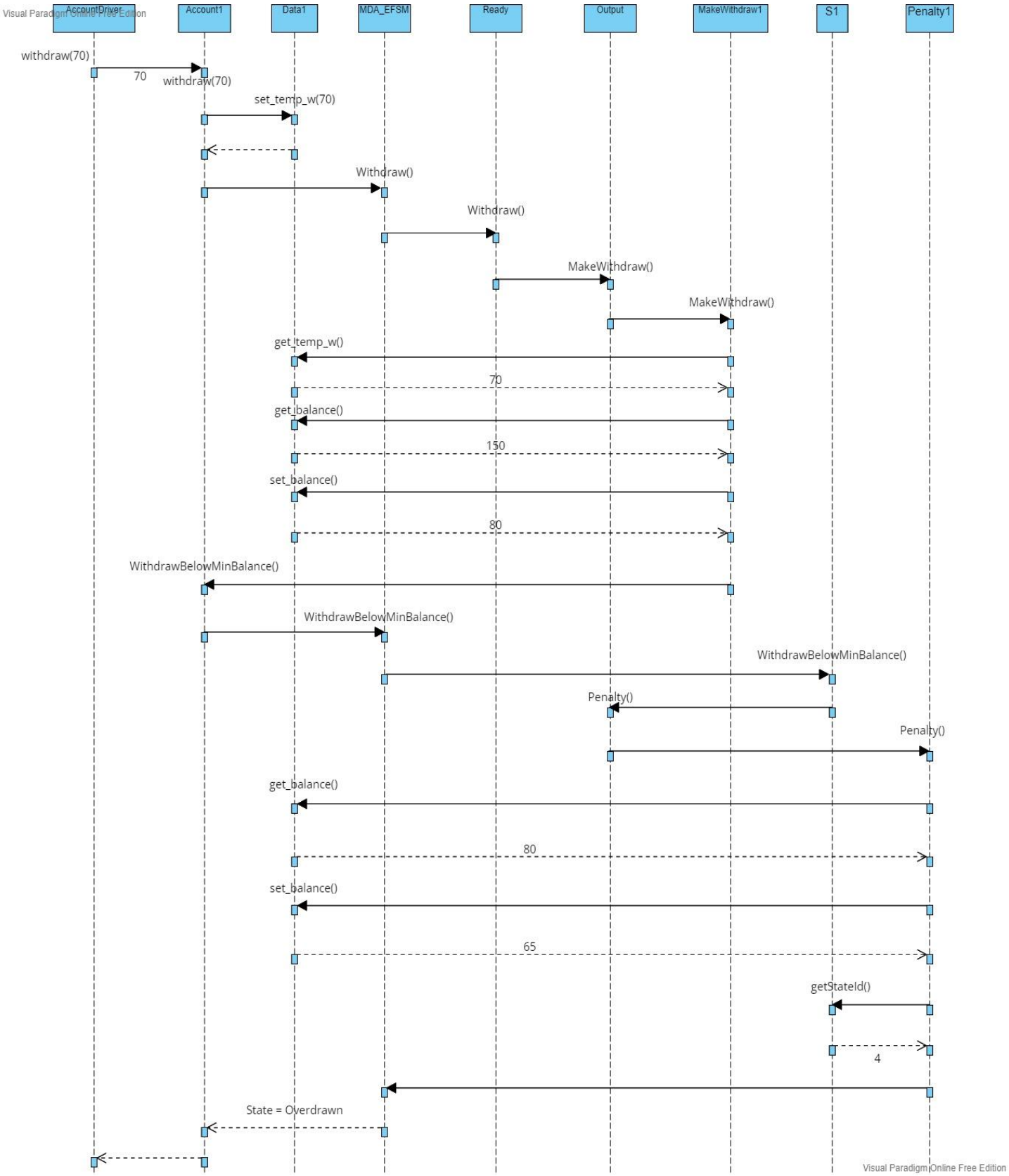
3.8.2 Login(123)



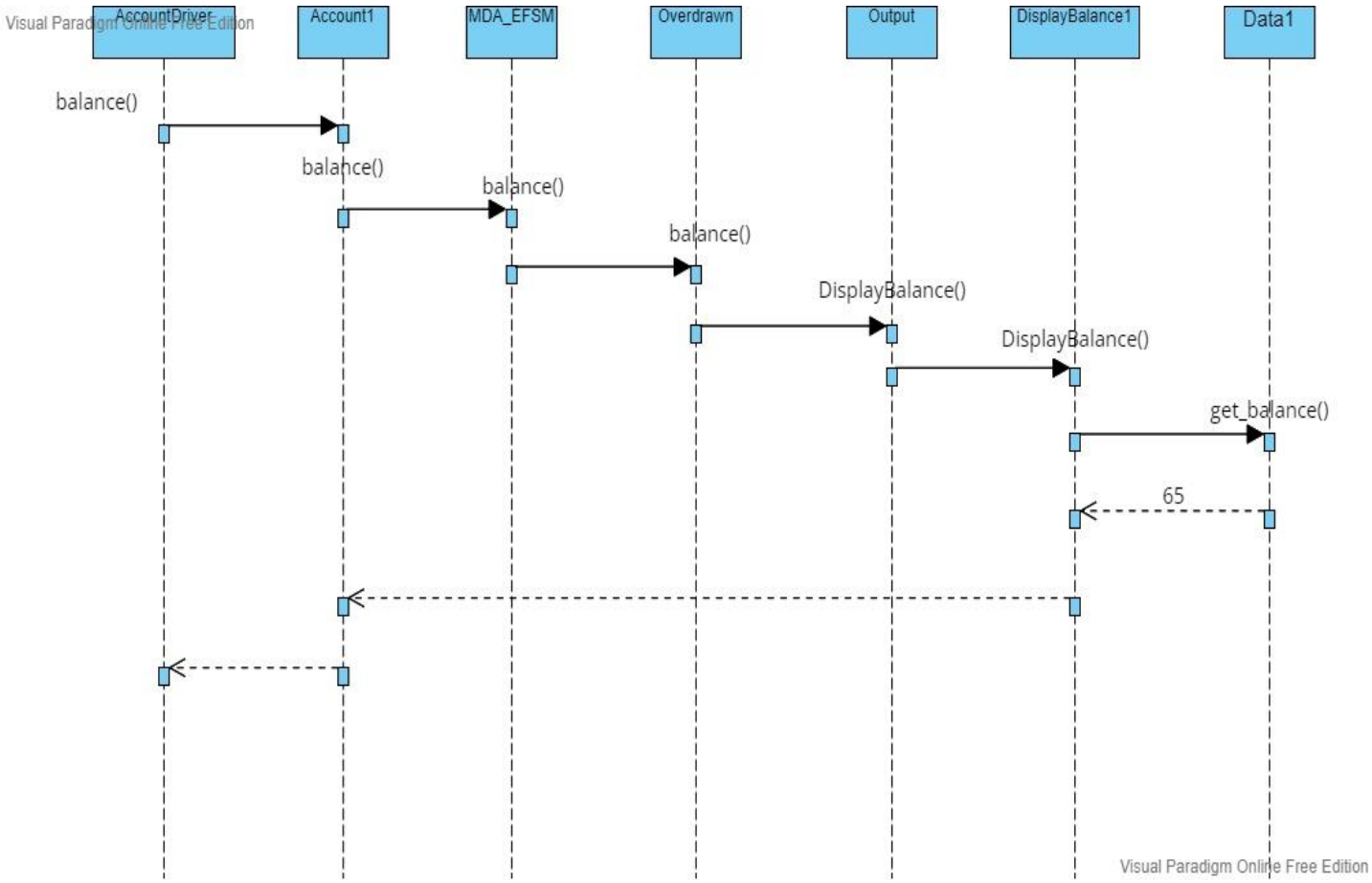
3.8.3 Pin(321)


Visual Paradigm Online Free Edition
AccountDriver
Account1
Data1
MDA_EFSM
CheckPin
Output
DisplayMenu1
Pin(321)
321
pin(321)
get_pin()
321
get_balance()
150
CorrectPinAboveMin()
CorrectPinAboveMin()
DisplayMenu()
DisplayMenu()
getStateId()
2
State = Ready
Visual Paradigm Online Free Edition

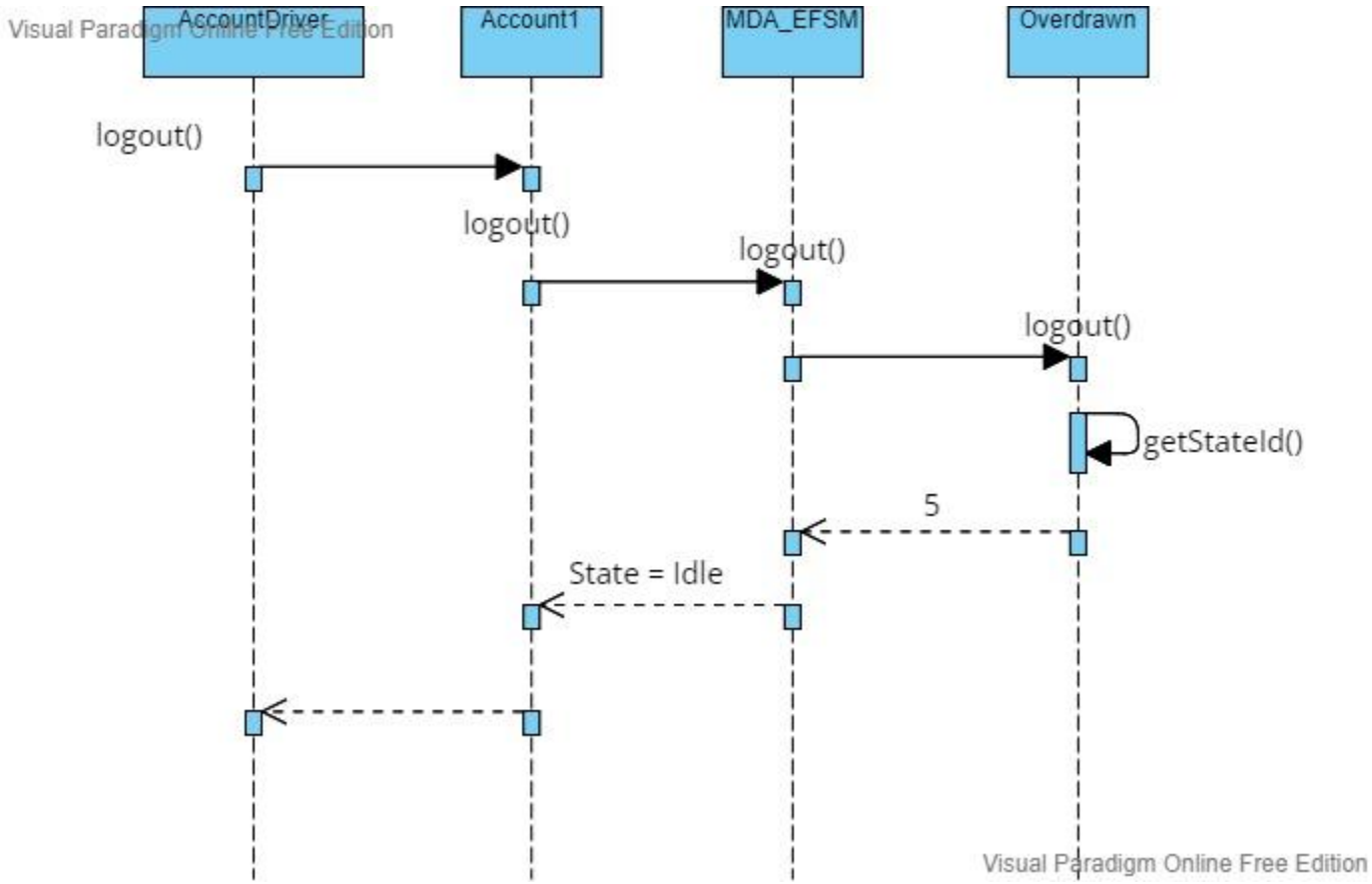
3.8.4 Withdraw(70)



3.8.5 Balance()


Visual Paradigm Online Free Edition
AccountDriver
Account1
MDA_EFSM
Overdrawn
Output
DisplayBalance1
Data1
balance()
balance()
balance()
balance()
DisplayBalance()
DisplayBalance()
get_balance()
65
Visual Paradigm Online Free Edition

3.8.6 Logout()


AccountDriver
Account1
MDA_EFSM
Overdrawn
logout()
logout()
logout()
logout()
getStateId()
5
State = Idle
Visual Paradigm Online Free Edition