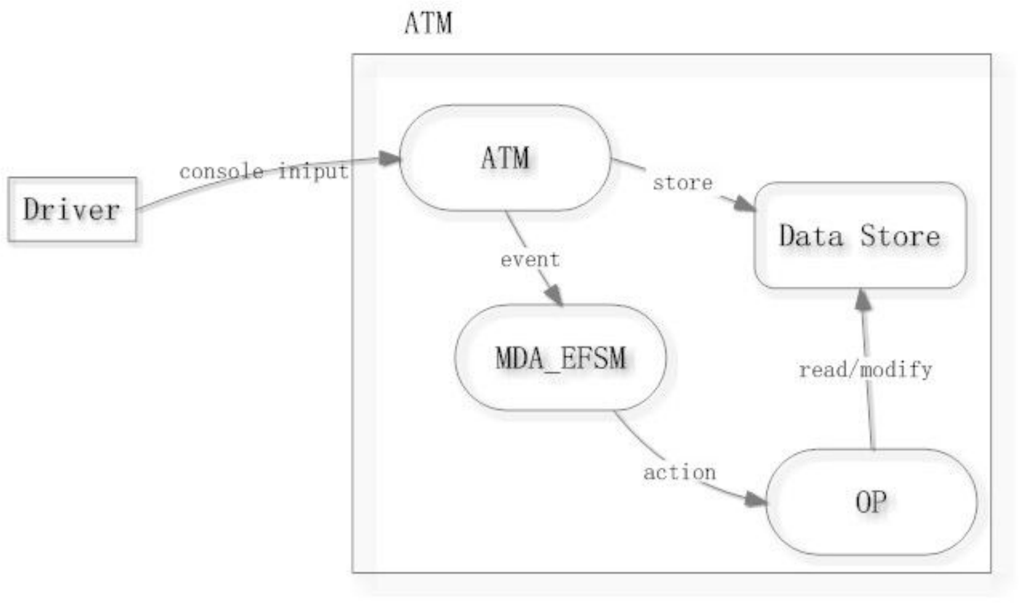
CS586 14F PROJECT REPORT

Feng Huang , A20281629

1. Model-Driven Architecture of the ATM components

a. A general MDA architecture of the ATM components

As the following figure illustrates, an ATM instance is consisted of the ATM input processor, MDA_EFSM state machine, Data Storage and Output processor. Besides, in order to easy the debugging procedure, a driver parser is provided, which takes input from console and parse user's options to perform ATM operations. The MDA_EFSM is platform independent because it does not know what data is, its only job is to perform state transaction for business logic.



b. MDA-EFSM model for the ATM components

i. A list of events for the MDA-EFSM

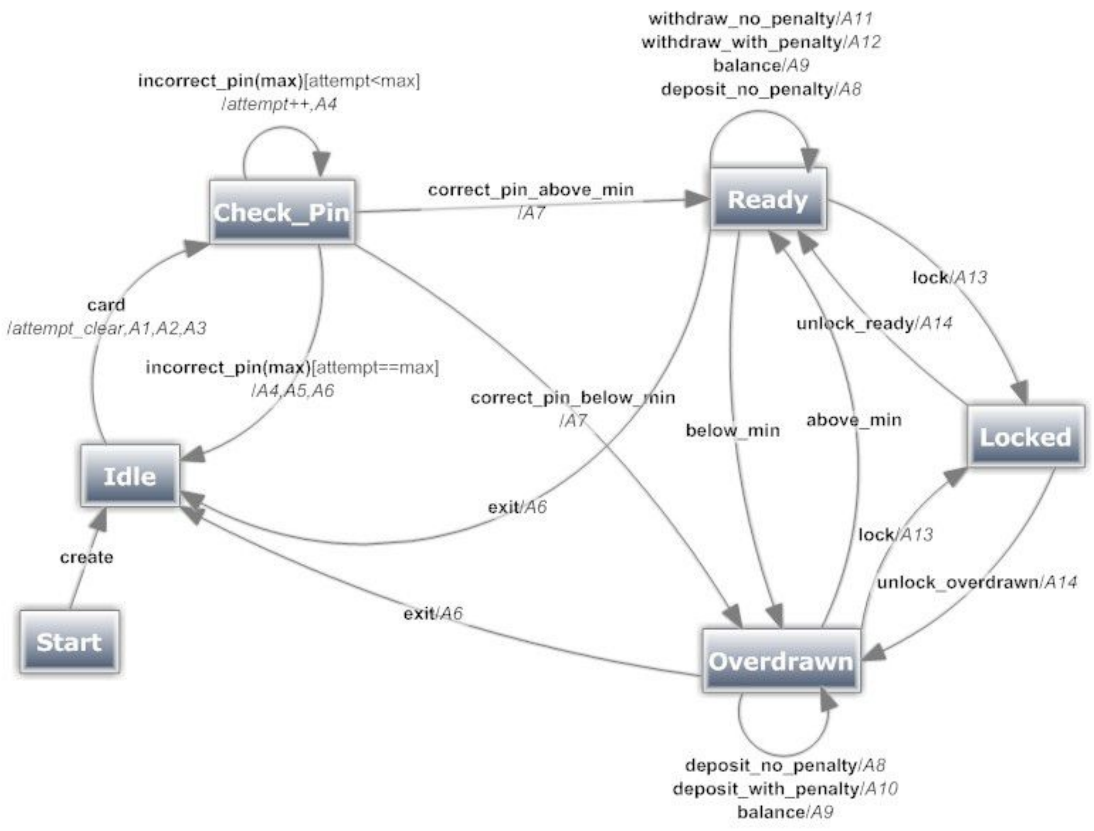
```
create();
card();
incorrect_pin(int max);
correct_pin_below_min();
correct_pin_above_min();
deposit_no_penalty();
deposit_with_penalty();
withdraw_no_penalty();
withdraw_with_penalty();
above_min();
below_min();
balance();
lock();
unlock_ready();
unlock_overdrawn();
exit();
```

ii. A list of actions for the MDA-EFSM with their descriptions

```
A1: store pin -- assign temp pin to pin in data store
A2: store balance -- assign temp balance to balance in data store
```

A3: prompt for pin -- *message*
A4: incorrect pin message -- *error message*
A5: too many attempts message -- *error message when attempts reach max*
A6: eject card -- *end operation*
A7: display menu -- *menu for deposit, balance, withdraw, exit, and lock if applicable*
A8: make deposit no penalty -- *deposit in ready state or if balance after deposit above min_balance in overdrawn state*
A9: display balance -- *show current balance in data store*
A10: make deposit with penalty -- *when in overdrawn state, after deposit the balance still below min_balance*
A11: withdraw no penalty -- *in ready state, after withdraw, balance still above min_balance*
A12: withdraw with penalty -- *in ready state, after withdraw, balance will below min_balance*
A13: lock -- *lock if applicable*
A14: unlock -- *unlock if applicable*

iii. A state diagram of the MDA-EFSM



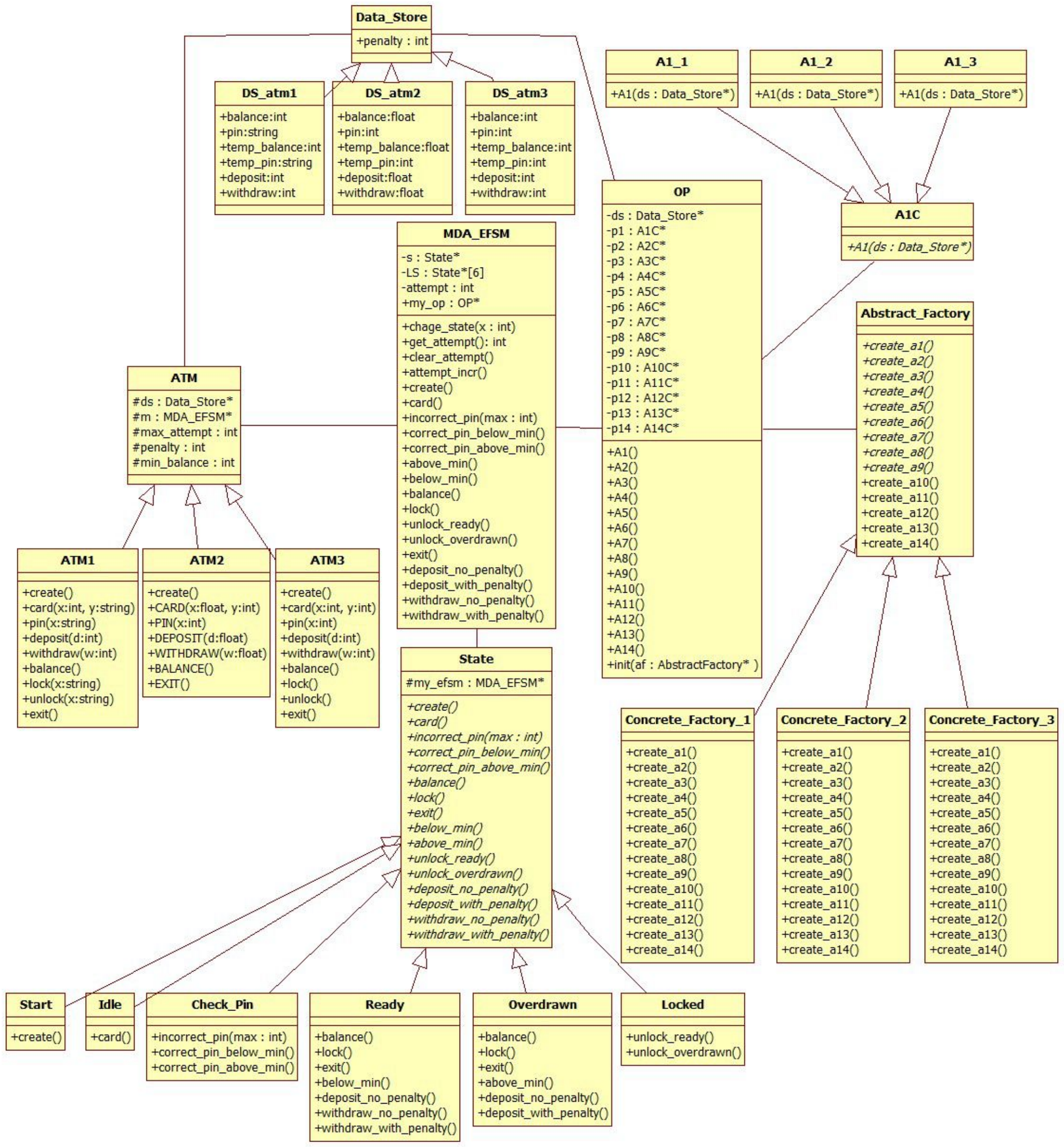
iv. Pseudo-code of all operations of Input Processors of ATM-1, ATM-2 and ATM-3

```
/* ATM1 */
create(){
    // init atm-dependent data
    max_attempt = 3;
    penalty = 10;
    min_balance = 1000;
    // create Data_Store object
    ds = new DS_atm1();
    ds->penalty = penalty;
    // create OP object and init it
    OP *op = new OP();
    op->ds = ds;
    op->init(new Concrete_Factory_1());
    // create MDA_EFSM and init it
    m = new MDA_EFSM();
    m->my_op = op;
    m->create();
```

```
}
card(int x, std::string y){
    ((DS_atm1 *)ds)->temp_balance = x;
    ((DS_atm1 *)ds)->temp_pin = y;
    m->card();
}
pin(std::string x){
    if (((DS_atm1 *)ds)->pin == x){
        if (((DS_atm1 *)ds)->balance < min_balance)
            m->correct_pin_below_min();
        else
            m->correct_pin_above_min();
    }
    else
        m->incorrect_pin(max_attempt);
}
deposit(int d){
    ((DS_atm1 *)ds)->deposit = d;
    if (((DS_atm1 *)ds)->balance + d > min_balance)
        m->deposit_no_penalty();
    else
        m->deposit_with_penalty();
    if (((DS_atm1 *)ds)->balance >= min_balance)
        m->above_min();
}
withdraw(int w){
    ((DS_atm1 *)ds)->withdraw = w;
    if (((DS_atm1 *)ds)->balance - w >= min_balance)
        m->withdraw_no_penalty();
    else if (((DS_atm1 *)ds)->balance - w > 0 ){
        m->withdraw_with_penalty();
        m->below_min();
    }
}
balance(){
    m->balance();
}
lock(std::string x){
    if (((DS_atm1 *)ds)->pin == x)
        m->lock();
}
unlock(std::string x){
    if (((DS_atm1 *)ds)->pin == x){
        if (((DS_atm1 *)ds)->balance >= min_balance)
            m->unlock_ready();
        else
            m->unlock_overdrawn();
    }
}
exit(){
    m->exit();
}
/* ATM2 */
create(){
    // init atm-dependent data
    max_attempt = 2;
    penalty = 20;
    min_balance = 500;
    // create Data_Store object
    ds = new DS_atm2();
    ds->penalty = penalty;
    // create OP object and init it
    OP *op = new OP();
    op->ds = ds;
    op->init(new Concrete_Factory_2());
    // create MDA_EFSM and init it
    m = new MDA_EFSM();
    m->my_op = op;
    m->create();
}
CARD(float x, int y){
    ((DS_atm2 *)ds)->temp_balance = x;
    ((DS_atm2 *)ds)->temp_pin = y;
    m->card();
}
PIN(int x){
    if (((DS_atm2 *)ds)->pin == x){
        if (((DS_atm2 *)ds)->balance < min_balance)
            m->correct_pin_below_min();
        else
            m->correct_pin_above_min();
    }
    else
        m->incorrect_pin(max_attempt);
}
DEPOSIT(float d){
    ((DS_atm2 *)ds)->deposit = d;
    if (((DS_atm2 *)ds)->balance + d > min_balance)
        m->deposit_no_penalty();
    else
        m->deposit_with_penalty();
    if (((DS_atm2 *)ds)->balance >= min_balance)
        m->above_min();
}
WITHDRAW(float w){
    ((DS_atm2 *)ds)->withdraw = w;
    if (((DS_atm2 *)ds)->balance - w >= min_balance)
        m->withdraw_no_penalty();
    else if (((DS_atm2 *)ds)->balance - w > 0 ){
        m->withdraw_with_penalty();
        m->below_min();
    }
}
BALANCE(){
    m->balance();
}
EXIT(){
    m->exit();
}
/* ATM3 */
create(){
// init atm-dependent data
    max_attempt = 1;
    penalty = 0;
    min_balance = 100;
    // create Data_Store object
    ds = new DS_atm3();
    ds->penalty = penalty;
    // create OP object and init it
    OP *op = new OP();
    op->ds = ds;
    op->init(new Concrete_Factory_3());
    // create MDA_EFSM and init it
    m = new MDA_EFSM();
    m->my_op = op;
    m->create();
}
card(int x, int y){
    ((DS_atm3 *)ds)->temp_balance = x;
    ((DS_atm3 *)ds)->temp_pin = y;
    m->card();
}
pin(int x){
    if (((DS_atm3 *)ds)->pin == x){
        if (((DS_atm3 *)ds)->balance < min_balance)
            m->correct_pin_below_min();
        else
            m->correct_pin_above_min();
    }
    else
        m->incorrect_pin(max_attempt);
}
deposit(int d){
    ((DS_atm3 *)ds)->deposit = d;
    if (((DS_atm3 *)ds)->balance + d > min_balance)
        m->deposit_no_penalty();
    else
        m->deposit_with_penalty();
    if (((DS_atm3 *)ds)->balance >= min_balance)
        m->above_min();
}
withdraw(int w){
    ((DS_atm3 *)ds)->withdraw = w;
    if (((DS_atm3 *)ds)->balance - w >= min_balance)
```

```
            m->withdraw_no_penalty();
        else if (((DS_atm3 *)ds)->balance - w > 0 ){
            m->withdraw_with_penalty();
            m->below_min();
        }
    }
    balance(){
        m->balance();
    }
    lock(){
        m->lock();
    }
    unlock(){
        if (((DS_atm1 *)ds)->balance >= min_balance)
            m->unlock_ready();
        else
            m->unlock_overdrawn();
    }
    exit(){
        m->exit();
    }
```

2. Class diagram(s) of the MDA of the ATM components.



a) ATM, ATM1, ATM2, ATM3 class:

This set of classes are input processors in MDA architecture. ATM1,2,3 are subclass which have operations like create, card, pin. These operations are interface of client program to call. Atm-specific data are located in super class ATM. Attributes are pointers to Data_Store, MDA_EFSM, and arguments like minimal balance, penalty value, and maximal attempt. During the create() call of ATM, these attributes will be initialized, Data_Store and MDA_EFSM instance will be created and pointed to by corresponding pointers. Besides the create(), other methods works as following steps:

1. Access ds(Data_Store*), put temporary value or check existed value, and decide next step.
2. According to checked value, send event to m(MDA_EFSM*) by call m->xxxx().
3. Access ds if necessary, call additional event if some condition is met.

Pseudo-codes: already given in first section.

b) Data_Store, DS_atm1, DS_atm2, DS_atm3 class:

Due to difference of data types of atms, data store only has penalty in super class, other data will be located in subclass. For each field like pin and balance, there is a temp value field, this is because input processor (ATM) can just read from data store and write to temporary fields, only output processor has right to write to non-temp field because output processor is driven by MDA_EFSM which has state transaction logic inside. So what every input processor (ATM) do to modify balance or pin is not guaranteed to be done.

Pseudo-codes: no functions in these classes.

c) MDA_EFSM, State, Start, Idle, Check_Pin, Ready, Overdrawn, Locked class:

This set of class represents State Pattern. State transition diagram is given in previous section. This state pattern work as the following:

1. ATM invokes event (call) in MDA_EFSM instance.
2. In the call of MDA_EFSM, it will call on each state's method for real implementation.
3. EFSM maintains a pointer to current state, which will provide operation for events.
4. Before call in state class return, it will change EFSM's current state by certain logic.
5. This design is a decentralized version of Stater Pattern.
6. The field my_op stores pointer to a OP instance, for output operation, this instance is create in ATM's create() and initialized after MDA_EFSM is created.

Pseudo-codes:

```
MDA_EFSM::MDA_EFSM(){
    LS[0] = new Start(this);
    LS[1] = new Idle(this);
    LS[2] = new Check_Pin(this);
    LS[3] = new Ready(this);
    LS[4] = new Overdrawn(this);
    LS[5] = new Locked(this);
    s = LS[0];
}
MDA_EFSM::create(){
    s->create();
}
MDA_EFSM::card(){
    s->card();
}
MDA_EFSM::incorrect_pin(int max){
    s->incorrect_pin(max);
}
MDA_EFSM::correct_pin_below_min(){
    s->correct_pin_below_min();
```

```
}
MDA_EFSM::correct_pin_above_min(){
    s->correct_pin_above_min();
}
MDA_EFSM::deposit_no_penalty(){
    s->deposit_no_penalty();
}
MDA_EFSM::deposit_with_penalty(){
    s->deposit_with_penalty();
}
MDA_EFSM::withdraw_no_penalty(){
    s->withdraw_no_penalty();
}
MDA_EFSM::withdraw_with_penalty(){
    s->withdraw_with_penalty();
}
MDA_EFSM::above_min(){
    s->above_min();
}
MDA_EFSM::below_min(){
    s->below_min();
}
MDA_EFSM::balance(){
    s->balance();
}
MDA_EFSM::lock(){
    s->lock();
}
MDA_EFSM::unlock_ready(){
    s->unlock_ready();
}
MDA_EFSM::unlock_overdrawn(){
    s->unlock_overdrawn();
}
MDA_EFSM::exit(){
    s->exit();
}
Start::create(){
    my_efsm->change_state(1);
}
Idle::card(){
    my_efsm->clear_attempt();
    my_efsm->my_op->A1();
    my_efsm->my_op->A2();
    my_efsm->my_op->A3();
    my_efsm->change_state(2);
}
Check_Pin::incorrect_pin(int max){
    if (my_efsm->get_attempt() < max){
        my_efsm->attempt_incr();
        my_efsm->my_op->A4();
    }
    else{
        my_efsm->my_op->A4();
        my_efsm->my_op->A5();
        my_efsm->my_op->A6();
        my_efsm->change_state(1);
    }
}
Check_Pin::correct_pin_below_min(){
    my_efsm->my_op->A7();
    my_efsm->change_state(4);
}
Check_Pin::correct_pin_above_min(){
    my_efsm->my_op->A7();
    my_efsm->change_state(3);
}
Ready::deposit_no_penalty(){
    my_efsm->my_op->A8();
}
Ready::withdraw_with_penalty(){
    my_efsm->my_op->A12();
}
Ready::withdraw_no_penalty(){
    my_efsm->my_op->A11();
}
Ready::below_min(){
    my_efsm->change_state(4);
}
Ready::balance(){
    my_efsm->my_op->A9();
}
Ready::lock(){
    my_efsm->change_state(5);
    my_efsm->my_op->A13();
}
Ready::exit(){
    my_efsm->my_op->A6();
    my_efsm->change_state(1);
}
Overdrawn::deposit_no_penalty(){
    my_efsm->my_op->A8();
}
Overdrawn::deposit_with_penalty(){
    my_efsm->my_op->A10();
}
Overdrawn::above_min(){
    my_efsm->change_state(3);
}
Overdrawn::balance(){
    my_efsm->my_op->A9();
}
Overdrawn::lock(){
    my_efsm->change_state(5);
    my_efsm->my_op->A13();
}
Overdrawn::exit(){
    my_efsm->my_op->A6();
    my_efsm->change_state(1);
}
Locked::unlock_ready(){
    my_efsm->change_state(3);
    my_efsm->my_op->A14();
}
Locked::unlock_overdrawn(){
    my_efsm->change_state(4);
    my_efsm->my_op->A14();
}
```

d) OP , AxC class:

This set of class are for Strategy Pattern. A ds attribute for data accessing, and AnC for replacing different strategies. Method init() which related to Abstract Factory Pattern will be explained in next section. An output operation is performed as following:

1. In state class, my_efsm->my_op->Ax() is called.
2. In the body of Ax(), px->Ax() is called, which real implementation resides.
3. In the class diagram only one set of action class is shown, actually there are 14 of them.

Pseudo codes: (only listed A1 for instance)

```
    OP::init(Abstract_Factory *af){
    p1 = af->create_a1();
    ... ...
    }
    OP::A1(){
        p1->A1(ds);
    }
    ... ...
    A1C *Concrete_Factory_1::create_a1(){
        return new A1_1();
    }
    ... ...
        // store pin
    A1_1::A1(Data_Store *ds){
        ((DS_atm1 *)ds)->pin = ((DS_atm1 *)ds)->temp_pin;
    }
```

e) Abstract_Factory, Concrete_Factory_x class:

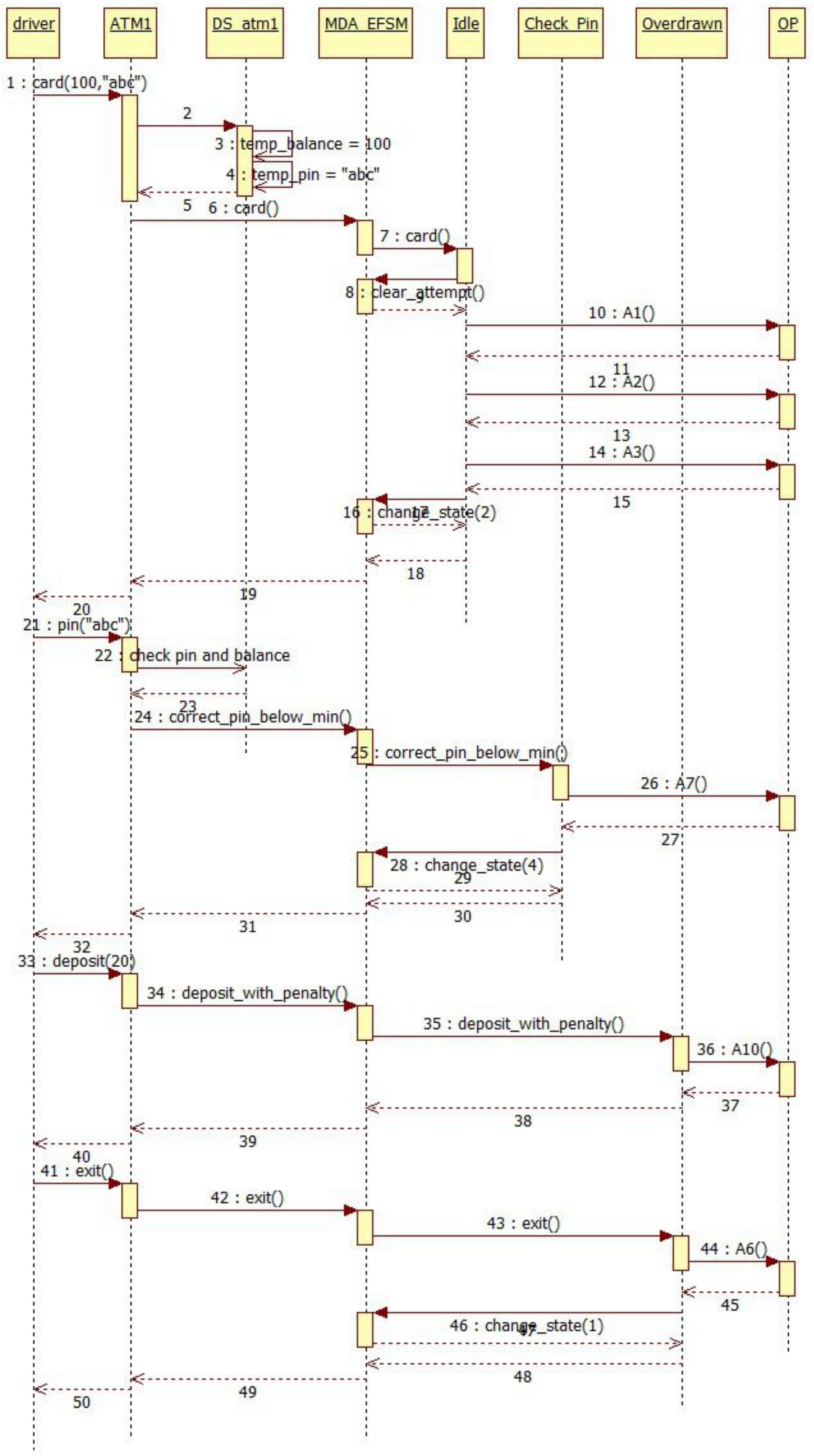
This is a typical Abstract Factory Pattern. In the pattern, AxC are the abstract products, and A1_1 and A1_2 etc. Classes are concrete products. Concrete factories are shown in diagram. When a OP instance is created in ATM's create(), the init() is called with concrete object passing in, this helps OP to initialize all px pointers and make those operation ready.

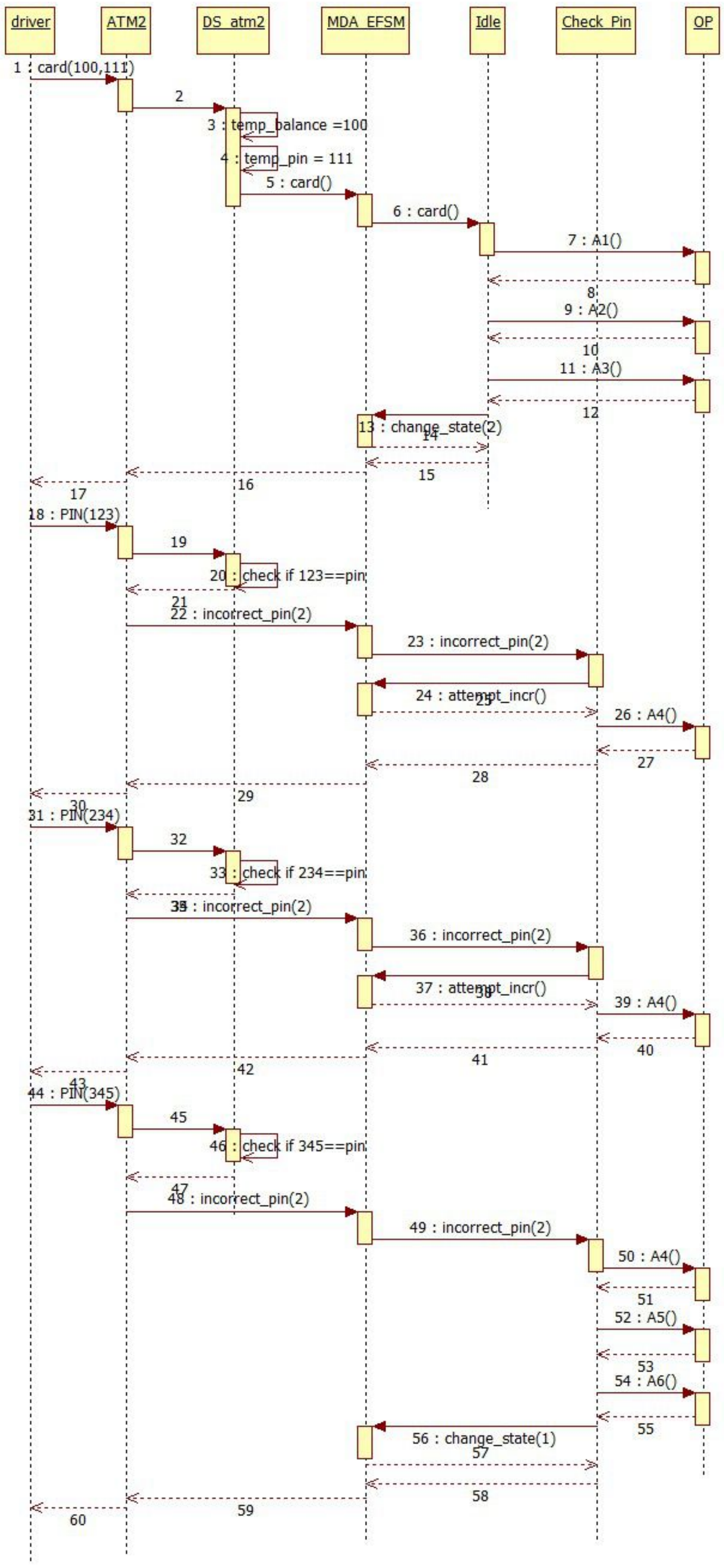
Pseudo codes : shared with section (d).

3. **Dynamics. Provide sequence diagrams of two Scenarios:**

   A. ATM-1 : card(100,"abc"), pin("abc"), deposit(20), exit()

   B. ATM-2 : CARD(100,111), PIN(123), PIN(234), PIN(345)


driver
ATM1
DS_atm1
MDA_EFSM
Idle
Check_Pin
Overdrawn
OP
1 : card(100,"abc")
2
3 : temp_balance = 100
4 : temp_pin = "abc"
5
6 : card()
7 : card()
8 : clear_attempt()
9
10 : A1()
11
12 : A2()
13
14 : A3()
15
16 : change_state(2)
17
18
19
20
21 : pin("abc")
22 : check pin and balance
23
24 : correct_pin_below_min()
25 : correct_pin_below_min()
26 : A7()
27
28 : change_state(4)
29
30
31
32
33 : deposit(20)
34 : deposit_with_penalty()
35 : deposit_with_penalty()
36 : A10()
37
38
39
40
41 : exit()
42 : exit()
43 : exit()
44 : A6()
45
46 : change_state(1)
47
48
49
50


driver
ATM2
DS_atm2
MDA_EFSM
Idle
Check_Pin
OP
1 : card(100,111)
2
3 : temp_balance =100
4 : temp_pin = 111
5 : card()
6 : card()
7 : A1()
8
9 : A2()
10
11 : A3()
12
13 : change_state(2)
14
15
16
17
18 : PIN(123)
19
20 : check if 123==pin
21
22 : incorrect_pin(2)
23 : incorrect_pin(2)
24 : attempt_incr()
25
26 : A4()
27
28
29
30
31 : PIN(234)
32
33 : check if 234==pin
34
35 : incorrect_pin(2)
36 : incorrect_pin(2)
37 : attempt_incr()
38
39 : A4()
40
41
42
43
44 : PIN(345)
45
46 : check if 345==pin
47
48 : incorrect_pin(2)
49 : incorrect_pin(2)
50 : A4()
51
52 : A5()
53
54 : A6()
55
56 : change_state(1)
57
58
59
60

4. Source-code and patterns

- **state pattern** : MDA_EFSM.cpp/h, State.cpp/h : all classes
- **strategy pattern** : OP.cpp/h : Ax() in class OP
- **abstract factory pattern** : OP.cpp/h :
  class Abstract_Factory
  class Concrete_Factory_1
  class Concrete_Factory_2
  class Concrete_Factory_3
  void OP::init(Abstract_Factory *af)

5. Well documented (commented) source code.

Source code are constructed and listed in the following order:

MDA_EFSM.cpp/h	-- MDA_EFSM core
State.cpp/h	-- State Pattern core
Data_Store.h	-- platform independent data wrapper
ATM.cpp/h	-- input processor
OP.cpp/h	-- action implementation, Abstract Factory Pattern, Strategy Pattern
Parser.cpp/h	-- input parser & atm function invoker, also applied Strategy Pattern
main.cpp	-- program entrance

MDA_EFSM.cpp:

```
#include "MDA_EFSM.h"
void MDA_EFSM::change_state(int x){
    if (x>=0 && x<=5)
    s = LS[x];
}
int MDA_EFSM::get_attempt(){
    return attempt;
}
void MDA_EFSM::clear_attempt(){
    attempt = 0;
}
void MDA_EFSM::attempt_incr(){
    attempt++;
}
MDA_EFSM::MDA_EFSM(){
    LS[0] = new Start(this);
    LS[1] = new Idle(this);
    LS[2] = new Check_Pin(this);
    LS[3] = new Ready(this);
    LS[4] = new Overdrawn(this);
    LS[5] = new Locked(this);
    s = LS[0];
}
MDA_EFSM::~MDA_EFSM(){
    delete[] LS;
    delete my_op;
}
void MDA_EFSM::create(){
    s->create();
}
void MDA_EFSM::card(){
    s->card();
}
void MDA_EFSM::incorrect_pin(int max){
    s->incorrect_pin(max);
}
void MDA_EFSM::correct_pin_below_min(){
    s->correct_pin_below_min();
}
void MDA_EFSM::correct_pin_above_min(){
    s->correct_pin_above_min();
}
void MDA_EFSM::deposit_no_penalty(){
    s->deposit_no_penalty();
}
void MDA_EFSM::deposit_with_penalty(){
    s->deposit_with_penalty();
}
void MDA_EFSM::withdraw_no_penalty(){
    s->withdraw_no_penalty();
}
void MDA_EFSM::withdraw_with_penalty(){
    s->withdraw_with_penalty();
}
void MDA_EFSM::above_min(){
    s->above_min();
}
void MDA_EFSM::below_min(){
    s->below_min();
}
void MDA_EFSM::balance(){
    s->balance();
}
void MDA_EFSM::lock(){
    s->lock();
}
void MDA_EFSM::unlock_ready(){
    s->unlock_ready();
}
void MDA_EFSM::unlock_overdrawn(){
    s->unlock_overdrawn();
}
void MDA_EFSM::exit(){
    s->exit();
}
```
MDA_EFSM.h:
```
#ifndef _MDA_EFSM_H
#define _MDA_EFSM_H
#include "State.h"
#include "OP.h"
class State;
class MDA_EFSM{
private:
    State *s;
    State *LS[6];
    int attempt;
public:
    OP *my_op;
    void change_state(int x);
    int get_attempt();
    void clear_attempt();
    void attempt_incr();
    MDA_EFSM();
    ~MDA_EFSM();
    void create();
    void card();
    void incorrect_pin(int max);
    void correct_pin_below_min();
    void correct_pin_above_min();
    void deposit_no_penalty();
    void deposit_with_penalty();
    void withdraw_no_penalty();
    void withdraw_with_penalty();
    void above_min();
    void below_min();
    void balance();
    void lock();
    void unlock_ready();
    void unlock_overdrawn();
    void exit();
};
#endif
```
State.cpp:
```
#include "State.h"
void Start::create(){
    my_efsm->change_state(1);
}
void Idle::card(){
    my_efsm->clear_attempt();
    my_efsm->my_op->A1();
    my_efsm->my_op->A2();
    my_efsm->my_op->A3();
    my_efsm->change_state(2);
}
void Check_Pin::incorrect_pin(int max){
    if (my_efsm->get_attempt() < max){
        my_efsm->attempt_incr();
        my_efsm->my_op->A4();
    }
    else{
        my_efsm->my_op->A4();
        my_efsm->my_op->A5();
        my_efsm->my_op->A6();
        my_efsm->change_state(1);
    }
}
void Check_Pin::correct_pin_below_min(){
    my_efsm->my_op->A7();
    my_efsm->change_state(4);
}
void Check_Pin::correct_pin_above_min(){
    my_efsm->my_op->A7();
    my_efsm->change_state(3);
}
void Ready::deposit_no_penalty(){
    my_efsm->my_op->A8();
}
void Ready::withdraw_with_penalty(){
    my_efsm->my_op->A12();
}
void Ready::withdraw_no_penalty(){
    my_efsm->my_op->A11();
}
void Ready::below_min(){
    my_efsm->change_state(4);
}
void Ready::balance(){
    my_efsm->my_op->A9();
}
void Ready::lock(){
    my_efsm->change_state(5);
    my_efsm->my_op->A13();
}
void Ready::exit(){
    my_efsm->my_op->A6();
    my_efsm->change_state(1);
}
void Overdrawn::deposit_no_penalty(){
    my_efsm->my_op->A8();
}
void Overdrawn::deposit_with_penalty(){
    my_efsm->my_op->A10();
}
void Overdrawn::above_min(){
    my_efsm->change_state(3);
}
void Overdrawn::balance(){
    my_efsm->my_op->A9();
}
void Overdrawn::lock(){
    my_efsm->change_state(5);
    my_efsm->my_op->A13();
}
void Overdrawn::exit(){
    my_efsm->my_op->A6();
    my_efsm->change_state(1);
}
void Locked::unlock_ready(){
    my_efsm->change_state(3);
    my_efsm->my_op->A14();
}
void Locked::unlock_overdrawn(){
    my_efsm->change_state(4);
    my_efsm->my_op->A14();
}
```

State.h:
```
#ifndef _STATE_H
#define _STATE_H
#include "MDA_EFSM.h"
class MDA_EFSM;
// state pattern -- the decentralized version
class State{
protected:
    MDA_EFSM *my_efsm;
public:
    State(MDA_EFSM *e):my_efsm(e){}
    virtual void create(){}
    virtual void card(){}
    virtual void incorrect_pin(int max){}
    virtual void correct_pin_below_min(){}
    virtual void correct_pin_above_min(){}
    virtual void deposit_no_penalty(){}
    virtual void deposit_with_penalty(){}
    virtual void withdraw_with_penalty(){}
    virtual void withdraw_no_penalty(){}
    virtual void above_min(){}
    virtual void below_min(){}
    virtual void balance(){}
    virtual void lock(){}
    virtual void exit(){}
    virtual void unlock_ready(){}
    virtual void unlock_overdrawn(){}
};
class Start : public State{
public:
    Start(MDA_EFSM *e):State(e){}
    void create();
};
class Idle : public State{
public:
    Idle(MDA_EFSM *e):State(e){}
    void card();
};
class Check_Pin : public State{
public:
    Check_Pin(MDA_EFSM *e):State(e){}
    void incorrect_pin(int max);
    void correct_pin_below_min();
    void correct_pin_above_min();
};
class Ready : public State{
public:
    Ready(MDA_EFSM *e):State(e){}
    void deposit_no_penalty();
    void withdraw_with_penalty();
    void withdraw_no_penalty();
    void below_min();
    void balance();
    void lock();
    void exit();
};
class Overdrawn : public State{
public:
    Overdrawn(MDA_EFSM *e):State(e){}
    void deposit_no_penalty();
    void deposit_with_penalty();
    void above_min();
    void balance();
    void lock();
    void exit();
};
class Locked : public State{
public:
    Locked(MDA_EFSM *e):State(e){}
    void unlock_ready();
    void unlock_overdrawn();
};
#endif
```

Data_Store.h:
```
#ifndef _DATA_STORE_H
#define _DATA_STORE_H
#include <string>
```

```
class Data_Store{
public:
    int penalty;
};
class DS_atm1 : public Data_Store{
public:
    int balance;
    int temp_balance;
    std::string pin;
    std::string temp_pin;
    int deposit;
    int withdraw;
};
class DS_atm2 : public Data_Store{
public:
    float balance;
    float temp_balance;
    int pin;
    int temp_pin;
    float deposit;
    float withdraw;
};
class DS_atm3 : public Data_Store{
public:
    int balance;
    int temp_balance;
    int pin;
    int temp_pin;
    int deposit;
    int withdraw;
};
#endif
```

ATM.cpp:

```
#include "ATM.h"
/* ATM */
ATM::~ATM(){
    delete ds;
    delete m; // will delete op subsequentially
}
/* ATM1 */
void ATM1::create(){
    // init atm-dependent data
    max_attempt = 3;
    penalty = 10;
    min_balance = 1000;
    // create Data_Store object
    ds = new DS_atm1();
    ds->penalty = penalty;
    // create OP object and init it, op will be delete in MDA_EFSM
    OP *op = new OP();
    op->ds = ds;
    op->init(new Concrete_Factory_1());
    // create MDA_EFSM and init it
    m = new MDA_EFSM();
    m->my_op = op;
    m->create();
}
void ATM1::card(int x, std::string y){
    // store data to temp storage, for latter OP to use
    ((DS_atm1 *)ds)->temp_balance = x;
    ((DS_atm1 *)ds)->temp_pin = y;
    m->card();
}
void ATM1::pin(std::string x){
    if (((DS_atm1 *)ds)->pin == x){
        if (((DS_atm1 *)ds)->balance < min_balance)
            // go to overdrawn state
            m->correct_pin_below_min();
        else
            // go to ready state
            m->correct_pin_above_min();
    }
    else
        m->incorrect_pin(max_attempt);
}
void ATM1::deposit(int d){
    // store desposit number to temp storage
    ((DS_atm1 *)ds)->deposit = d;
    // decide if penalty involved
    if (((DS_atm1 *)ds)->balance + d > min_balance)
        m->deposit_no_penalty();
    else
        m->deposit_with_penalty();
    // decide whether to move state to ready
    if (((DS_atm1 *)ds)->balance >= min_balance)
        m->above_min();
}
void ATM1::withdraw(int w){
    // store withdraw value to temp storage
    ((DS_atm1 *)ds)->withdraw = w;
    // decide whether take the penalty
    if (((DS_atm1 *)ds)->balance - w >= min_balance)
        m->withdraw_no_penalty();
    else if (((DS_atm1 *)ds)->balance - w > 0 ){
        m->withdraw_with_penalty();
        // move to overdrawn state
        m->below_min();
    }
}
void ATM1::balance(){
    m->balance();
}
void ATM1::lock(std::string x){
    if (((DS_atm1 *)ds)->pin == x)
        m->lock();
}
void ATM1::unlock(std::string x){
    if (((DS_atm1 *)ds)->pin == x){
        // choose which state to go back according to balance
        if (((DS_atm1 *)ds)->balance >= min_balance)
            m->unlock_ready();
        else
            m->unlock_overdrawn();
    }
}
void ATM1::exit(){
    // back to Idle state
    m->exit();
}
/* ATM2 */
void ATM2::create(){
    // init atm-dependent data
    max_attempt = 2;
    penalty = 20;
    min_balance = 500;
    // create Data_Store object
    ds = new DS_atm2();
    ds->penalty = penalty;
    // create OP object and init it
    OP *op = new OP();
    op->ds = ds;
    op->init(new Concrete_Factory_2());
    // create MDA_EFSM and init it
    m = new MDA_EFSM();
    m->my_op = op;
    m->create();
}
void ATM2::CARD(float x, int y){
    // store data to temp storage, for latter OP to use
    ((DS_atm2 *)ds)->temp_balance = x;
    ((DS_atm2 *)ds)->temp_pin = y;
    m->card();
}
void ATM2::PIN(int x){
    if (((DS_atm2 *)ds)->pin == x){
        if (((DS_atm2 *)ds)->balance < min_balance)
            // go to overdrawn state
            m->correct_pin_below_min();
        else
            // go to ready state
            m->correct_pin_above_min();
    }
    else
        m->incorrect_pin(max_attempt);
```

```
}
void ATM2::DEPOSIT(float d){
    // store desposit number to temp storage
    ((DS_atm2 *)ds)->deposit = d;
    // decide if penalty involved
    if (((DS_atm2 *)ds)->balance + d > min_balance)
        m->deposit_no_penalty();
    else
        m->deposit_with_penalty();
    // decide whether to move state to ready
    if (((DS_atm2 *)ds)->balance >= min_balance)
        m->above_min();
}
void ATM2::WITHDRAW(float w){
    // store withdraw value to temp storage
    ((DS_atm2 *)ds)->withdraw = w;
    // decide whether take the penalty
    if (((DS_atm2 *)ds)->balance - w >= min_balance)
        m->withdraw_no_penalty();
    else if (((DS_atm2 *)ds)->balance - w > 0 ){
        m->withdraw_with_penalty();
        // move to overdrawn state
        m->below_min();
    }
}
void ATM2::BALANCE(){
    m->balance();
}
void ATM2::EXIT(){
    // back to Idle state
    m->exit();
}
/* ATM3 */
void ATM3::create(){
// init atm-dependent data
    max_attempt = 1;
    penalty = 0;
    min_balance = 100;
    // create Data_Store object
    ds = new DS_atm3();
    ds->penalty = penalty;
    // create OP object and init it
    OP *op = new OP();
    op->ds = ds;
    op->init(new Concrete_Factory_3());
    // create MDA_EFSM and init it
    m = new MDA_EFSM();
    m->my_op = op;
    m->create();
}
void ATM3::card(int x, int y){
    // store data to temp storage, for latter OP to use
    ((DS_atm3 *)ds)->temp_balance = x;
    ((DS_atm3 *)ds)->temp_pin = y;
    m->card();
}
void ATM3::pin(int x){
    if (((DS_atm3 *)ds)->pin == x){
        if (((DS_atm3 *)ds)->balance < min_balance)
            // go to overdrawn state
            m->correct_pin_below_min();
        else
            // go to ready state
            m->correct_pin_above_min();
    }
    else
        m->incorrect_pin(max_attempt);
}
void ATM3::deposit(int d){
    // store desposit number to temp storage
    ((DS_atm3 *)ds)->deposit = d;
    // decide if penalty involved
    if (((DS_atm3 *)ds)->balance + d > min_balance)
        m->deposit_no_penalty();
    else
        m->deposit_with_penalty();
    // decide whether to move state to ready
    if (((DS_atm3 *)ds)->balance >= min_balance)
        m->above_min();
}
void ATM3::withdraw(int w){
    // store withdraw value to temp storage
    ((DS_atm3 *)ds)->withdraw = w;
    // decide whether take the penalty
    if (((DS_atm3 *)ds)->balance - w >= min_balance)
        m->withdraw_no_penalty();
    else if (((DS_atm3 *)ds)->balance - w > 0 ){
        m->withdraw_with_penalty();
        // move to overdrawn state
        m->below_min();
    }
}
void ATM3::balance(){
    m->balance();
}
void ATM3::lock(){
    m->lock();
}
void ATM3::unlock(){
    // choose which state to go back according to balance
    if (((DS_atm1 *)ds)->balance >= min_balance)
        m->unlock_ready();
    else
        m->unlock_overdrawn();
}
void ATM3::exit(){
    // back to Idle state
    m->exit();
}
```
ATM.h:
```
#ifndef _ATM_H
#define _ATM_H
#include "Data_Store.h"
#include "MDA_EFSM.h"
#include "OP.h"
#include <string>
class ATM{
protected:
    MDA_EFSM *m;
    Data_Store *ds;
    int max_attempt;
    int penalty;
    int min_balance;
public:
    ~ATM();
};
class ATM1 : public ATM{
public:
    void create();
    void card(int x, std::string y);
    void pin(std::string x);
    void deposit(int d);
    void withdraw(int w);
    void balance();
    void lock(std::string x);
    void unlock(std::string x);
    void exit();
};
class ATM2 : public ATM{
public:
    void create();
    void CARD(float x, int y);
    void PIN(int x);
    void DEPOSIT(float d);
    void WITHDRAW(float w);
    void BALANCE();
    void EXIT();
};
class ATM3 : public ATM{
public:
    void create();
    void card(int x, int y);
    void pin(int x);
    void deposit(int d);
    void withdraw(int w);
    void balance();
    void lock();
    void unlock();
    void exit();
};
#endif
```

OP.cpp:

```
#include "OP.h"
    // store pin
void A1_1::A1(Data_Store *ds){
    ((DS_atm1 *)ds)->pin = ((DS_atm1 *)ds)->temp_pin;
}
void A1_2::A1(Data_Store *ds){
    ((DS_atm2 *)ds)->pin = ((DS_atm2 *)ds)->temp_pin;
}
void A1_3::A1(Data_Store *ds){
    ((DS_atm3 *)ds)->pin = ((DS_atm3 *)ds)->temp_pin;
}
    // store balance
void A2_1::A2(Data_Store *ds){
    ((DS_atm2 *)ds)->balance = ((DS_atm2 *)ds)->temp_balance;
}
void A2_2::A2(Data_Store *ds){
    ((DS_atm2 *)ds)->balance = ((DS_atm2 *)ds)->temp_balance;
}
void A2_3::A2(Data_Store *ds){
    ((DS_atm3 *)ds)->balance = ((DS_atm3 *)ds)->temp_balance;
}
    // prompt for pin
void A3_1::A3(Data_Store *ds){
    std::cout<<"ATM-1: please input pin ..."<<std::endl;
}
void A3_2::A3(Data_Store *ds){
    std::cout<<"ATM-2: please input pin ..."<<std::endl;
}
void A3_3::A3(Data_Store *ds){
    std::cout<<"ATM-3: please input pin ..."<<std::endl;
}
    // incorrect pin message
void A4_1::A4(Data_Store *ds){
    std::cout<<"ATM-1: incorrect pin ..."<<std::endl;
}
void A4_2::A4(Data_Store *ds){
    std::cout<<"ATM-2: incorrect pin ..."<<std::endl;
}
void A4_3::A4(Data_Store *ds){
    std::cout<<"ATM-3: incorrect pin ..."<<std::endl;
}
    // too many attempts message
void A5_1::A5(Data_Store *ds){
    std::cout<<"ATM-1: too many attempts ..."<<std::endl;
}
void A5_2::A5(Data_Store *ds){
    std::cout<<"ATM-2: too many attempts ..."<<std::endl;
}
void A5_3::A5(Data_Store *ds){
    std::cout<<"ATM-3: too many attempts ..."<<std::endl;
}
    // eject card
void A6_1::A6(Data_Store *ds){
    std::cout<<"ATM-1: card ejected ..."<<std::endl;
}
void A6_2::A6(Data_Store *ds){
    std::cout<<"ATM-2: card ejected ..."<<std::endl;
}
void A6_3::A6(Data_Store *ds){
    std::cout<<"ATM-3: card ejected ..."<<std::endl;
}
    // display menu
void A7_1::A7(Data_Store *ds){
    std::cout<<"ATM-1: pin correct, Welcome Back !"<<std::endl;
}
void A7_2::A7(Data_Store *ds){
    std::cout<<"ATM-2: pin correct, Welcome Back !"<<std::endl;
}
void A7_3::A7(Data_Store *ds){
    std::cout<<"ATM-3: pin correct, Welcome Back !"<<std::endl;
}
    // make deposit no penalty
void A8_1::A8(Data_Store *ds){
    std::cout<<"ATM-1: deposit "<<((DS_atm1 *)ds)->deposit<<" to balance "<<((DS_atm1
*)ds)->balance<<std::endl;
    ((DS_atm1 *)ds)->balance = ((DS_atm1 *)ds)->balance + ((DS_atm1 *)ds)->deposit;
}
void A8_2::A8(Data_Store *ds){
    std::cout<<"ATM-2: deposit "<<((DS_atm2 *)ds)->deposit<<" to balance "<<((DS_atm2
*)ds)->balance<<std::endl;
    ((DS_atm2 *)ds)->balance = ((DS_atm2 *)ds)->balance + ((DS_atm2 *)ds)->deposit;
}
void A8_3::A8(Data_Store *ds){
    std::cout<<"ATM-3: deposit "<<((DS_atm3 *)ds)->deposit<<" to balance "<<((DS_atm3
*)ds)->balance<<std::endl;
    ((DS_atm3 *)ds)->balance = ((DS_atm3 *)ds)->balance + ((DS_atm3 *)ds)->deposit;
}
    // display balance
void A9_1::A9(Data_Store *ds){
    std::cout<<"ATM-1: balance = "<<((DS_atm1 *)ds)->balance <<std::endl;
}
void A9_2::A9(Data_Store *ds){
    std::cout<<"ATM-2: balance = "<<((DS_atm2 *)ds)->balance <<std::endl;
}
void A9_3::A9(Data_Store *ds){
    std::cout<<"ATM-3: balance = "<<((DS_atm3 *)ds)->balance <<std::endl;
}
    //  make deposit with penalty
void A10_1::A10(Data_Store *ds){
    std::cout<<"ATM-1: deposit "<<((DS_atm1 *)ds)->deposit<<" to balance "<<((DS_atm1
*)ds)->balance<<" , penalty="<<ds->penalty<<std::endl;
    ((DS_atm1 *)ds)->balance = ((DS_atm1 *)ds)->balance + ((DS_atm1 *)ds)->deposit -
ds->penalty;
}
void A10_2::A10(Data_Store *ds){
    std::cout<<"ATM-2: deposit "<<((DS_atm2 *)ds)->deposit<<" to balance "<<((DS_atm2
*)ds)->balance<<" , penalty="<<ds->penalty<<std::endl;
    ((DS_atm2 *)ds)->balance = ((DS_atm2 *)ds)->balance + ((DS_atm2 *)ds)->deposit -
ds->penalty;
}
void A10_3::A10(Data_Store *ds){
    std::cout<<"ATM-3: deposit "<<((DS_atm3 *)ds)->deposit<<" to balance "<<((DS_atm3
*)ds)->balance<<" , penalty="<<ds->penalty<<std::endl;
    ((DS_atm3 *)ds)->balance = ((DS_atm3 *)ds)->balance + ((DS_atm3 *)ds)->deposit -
ds->penalty;
}
    // withdraw no penalty
void A11_1::A11(Data_Store *ds){
    std::cout<<"ATM-1: withdraw = "<<((DS_atm1 *)ds)->withdraw <<std::endl;
    ((DS_atm1 *)ds)->balance = ((DS_atm1 *)ds)->balance - ((DS_atm1 *)ds)->withdraw;
}
void A11_2::A11(Data_Store *ds){
    std::cout<<"ATM-2: withdraw = "<<((DS_atm2 *)ds)->withdraw <<std::endl;
    ((DS_atm2 *)ds)->balance = ((DS_atm2 *)ds)->balance - ((DS_atm2 *)ds)->withdraw;
}
void A11_3::A11(Data_Store *ds){
    std::cout<<"ATM-3: withdraw = "<<((DS_atm3 *)ds)->withdraw <<std::endl;
    ((DS_atm3 *)ds)->balance = ((DS_atm3 *)ds)->balance - ((DS_atm3 *)ds)->withdraw;
}
    // withdraw with penalty
void A12_1::A12(Data_Store *ds){
    std::cout<<"ATM-1:   withdraw   =   "<<((DS_atm1   *)ds)->withdraw   <<"   with
penalty="<<ds->penalty<<std::endl;
    ((DS_atm1 *)ds)->balance = ((DS_atm1 *)ds)->balance - ((DS_atm1 *)ds)->withdraw -
ds->penalty;
}
void A12_2::A12(Data_Store *ds){
    std::cout<<"ATM-2:   withdraw   =   "<<((DS_atm2   *)ds)->withdraw   <<"   with
penalty="<<ds->penalty<<std::endl;
    ((DS_atm2 *)ds)->balance = ((DS_atm2 *)ds)->balance - ((DS_atm2 *)ds)->withdraw -
ds->penalty;
}
void A12_3::A12(Data_Store *ds){
    std::cout<<"ATM-3:   withdraw   =   "<<((DS_atm3   *)ds)->withdraw   <<"   with
penalty="<<ds->penalty<<std::endl;
    ((DS_atm3 *)ds)->balance = ((DS_atm3 *)ds)->balance - ((DS_atm3 *)ds)->withdraw -
```

```
ds->penalty;
}
    // lock
void A13_1::A13(Data_Store *ds){
    std::cout<<"ATM-1: Locked"<<std::endl;
}
void A13_2::A13(Data_Store *ds){
    // left blank
}
void A13_3::A13(Data_Store *ds){
    std::cout<<"ATM-3: Locked"<<std::endl;
}
    // unlock
void A14_1::A14(Data_Store *ds){
    std::cout<<"ATM-1: unLocked"<<std::endl;
}
void A14_2::A14(Data_Store *ds){
    // left blank
}
void A14_3::A14(Data_Store *ds){
    std::cout<<"ATM-3: unLocked"<<std::endl;
}
A1C *Concrete_Factory_1::create_a1(){
    return new A1_1();
}
A2C *Concrete_Factory_1::create_a2(){
        return new A2_1();
}
A3C *Concrete_Factory_1::create_a3(){
    return new A3_1();
}
A4C *Concrete_Factory_1::create_a4(){
    return new A4_1();
}
A5C *Concrete_Factory_1::create_a5(){
    return new A5_1();
}
A6C *Concrete_Factory_1::create_a6(){
    return new A6_1();
}
A7C *Concrete_Factory_1::create_a7(){
    return new A7_1();
}
A8C *Concrete_Factory_1::create_a8(){
    return new A8_1();
}
A9C *Concrete_Factory_1::create_a9(){
    return new A9_1();
}
A10C *Concrete_Factory_1::create_a10(){
    return new A10_1();
}
A11C *Concrete_Factory_1::create_a11(){
    return new A11_1();
}
A12C *Concrete_Factory_1::create_a12(){
    return new A12_1();
}
A13C *Concrete_Factory_1::create_a13(){
    return new A13_1();
}
A14C *Concrete_Factory_1::create_a14(){
    return new A14_1();
}
A1C *Concrete_Factory_2::create_a1(){
    return new A1_2();
}
A2C *Concrete_Factory_2::create_a2(){
    return new A2_2();
}
A3C *Concrete_Factory_2::create_a3(){
    return new A3_2();
}
A4C *Concrete_Factory_2::create_a4(){
    return new A4_2();
}
A5C *Concrete_Factory_2::create_a5(){
    return new A5_2();
}
A6C *Concrete_Factory_2::create_a6(){
    return new A6_2();
}
A7C *Concrete_Factory_2::create_a7(){
    return new A7_2();
}
A8C *Concrete_Factory_2::create_a8(){
    return new A8_2();
}
A9C *Concrete_Factory_2::create_a9(){
    return new A9_2();
}
A10C *Concrete_Factory_2::create_a10(){
    return new A10_2();
}
A11C *Concrete_Factory_2::create_a11(){
    return new A11_2();
}
A12C *Concrete_Factory_2::create_a12(){
    return new A12_2();
}
A13C *Concrete_Factory_2::create_a13(){
    return new A13_2();
}
A14C *Concrete_Factory_2::create_a14(){
    return new A14_2();
}
A1C *Concrete_Factory_3::create_a1(){
    return new A1_3();
}
A2C *Concrete_Factory_3::create_a2(){
    return new A2_3();
}
A3C *Concrete_Factory_3::create_a3(){
    return new A3_3();
}
A4C *Concrete_Factory_3::create_a4(){
    return new A4_3();
}
A5C *Concrete_Factory_3::create_a5(){
    return new A5_3();
}
A6C *Concrete_Factory_3::create_a6(){
    return new A6_3();
}
A7C *Concrete_Factory_3::create_a7(){
    return new A7_3();
}
A8C *Concrete_Factory_3::create_a8(){
    return new A8_3();
}
A9C *Concrete_Factory_3::create_a9(){
    return new A9_3();
}
A10C *Concrete_Factory_3::create_a10(){
    return new A10_3();
}
A11C *Concrete_Factory_3::create_a11(){
    return new A11_3();
}
A12C *Concrete_Factory_3::create_a12(){
    return new A12_3();
}
A13C *Concrete_Factory_3::create_a13(){
    return new A13_3();
}
A14C *Concrete_Factory_3::create_a14(){
    return new A14_3();
}
void OP::A1(){
    p1->A1(ds);
}
void OP::A2(){
    p2->A2(ds);
}
```

```
void OP::A3(){
    p3->A3(ds);
}
void OP::A4(){
    p4->A4(ds);
}
void OP::A5(){
    p5->A5(ds);
}
void OP::A6(){
    p6->A6(ds);
}
void OP::A7(){
    p7->A7(ds);
}
void OP::A8(){
    p8->A8(ds);
}
void OP::A9(){
    p9->A9(ds);
}
void OP::A10(){
    p10->A10(ds);
}
void OP::A11(){
    p11->A11(ds);
}
void OP::A12(){
    p12->A12(ds);
}
void OP::A13(){
    p13->A13(ds);
}
void OP::A14(){
    p14->A14(ds);
}
// init the OP attributes according to concrete factory passed in
void OP::init(Abstract_Factory *af){
    p1 = af->create_a1();
    p2 = af->create_a2();
    p3 = af->create_a3();
    p4 = af->create_a4();
    p5 = af->create_a5();
    p6 = af->create_a6();
    p7 = af->create_a7();
    p8 = af->create_a8();
    p9 = af->create_a9();
    p10 = af->create_a10();
    p11 = af->create_a11();
    p12 = af->create_a12();
    p13 = af->create_a13();
    p14 = af->create_a14();
    delete af;
}
```
OP.h:
```
#ifndef _OP_H
#define _OP_H
#include "Data_Store.h"
#include <iostream>
/* A1 */
class A1C{
public:
    // store pin
    virtual void A1(Data_Store *ds)=0;
};
class A1_1 : public A1C{
public:
    void A1(Data_Store *ds);
};
class A1_2 : public A1C{
public:
    void A1(Data_Store *ds);
};
class A1_3 : public A1C{
public:
    void A1(Data_Store *ds);
};
```

```
/* A2 */
class A2C{
public:
    // store balance
    virtual void A2(Data_Store *ds)=0;
};
class A2_1 : public A2C{
public:
    void A2(Data_Store *ds);
};
class A2_2 : public A2C{
public:
    void A2(Data_Store *ds);
};
class A2_3 : public A2C{
public:
    void A2(Data_Store *ds);
};
/* A3 */
class A3C{
public:
    // prompt for pin
    virtual void A3(Data_Store *ds)=0;
};
class A3_1 : public A3C{
public:
    void A3(Data_Store *ds);
};
class A3_2 : public A3C{
public:
    void A3(Data_Store *ds);
};
class A3_3 : public A3C{
public:
    void A3(Data_Store *ds);
};
/* A4 */
class A4C{
public:
    // incorrect pin message
    virtual void A4(Data_Store *ds)=0;
};
class A4_1 : public A4C{
public:
    void A4(Data_Store *ds);
};
class A4_2 : public A4C{
public:
    void A4(Data_Store *ds);
};
class A4_3 : public A4C{
public:
    void A4(Data_Store *ds);
};
/* A5 */
class A5C{
public:
    // too many attempts message
    virtual void A5(Data_Store *ds)=0;
};
class A5_1 : public A5C{
public:
    void A5(Data_Store *ds);
};
class A5_2 : public A5C{
public:
    void A5(Data_Store *ds);
};
class A5_3 : public A5C{
public:
    void A5(Data_Store *ds);
};
/* A6 */
class A6C{
public:
    // eject card
    virtual void A6(Data_Store *ds)=0;
};
class A6_1 : public A6C{
public:
    void A6(Data_Store *ds);
};
class A6_2 : public A6C{
public:
    void A6(Data_Store *ds);
};
class A6_3 : public A6C{
public:
    void A6(Data_Store *ds);
};
/* A7 */
class A7C{
public:
    // display menu
    virtual void A7(Data_Store *ds)=0;
};
class A7_1 : public A7C{
public:
    void A7(Data_Store *ds);
};
class A7_2 : public A7C{
public:
    void A7(Data_Store *ds);
};
class A7_3 : public A7C{
public:
    void A7(Data_Store *ds);
};
/* A8 */
class A8C{
public:
    // make deposit no penalty
    virtual void A8(Data_Store *ds)=0;
};
class A8_1 : public A8C{
public:
    void A8(Data_Store *ds);
};
class A8_2 : public A8C{
public:
    void A8(Data_Store *ds);
};
class A8_3 : public A8C{
public:
    void A8(Data_Store *ds);
};
/* A9 */
class A9C{
public:
    // display balance
    virtual void A9(Data_Store *ds)=0;
};
class A9_1 : public A9C{
public:
    void A9(Data_Store *ds);
};
class A9_2 : public A9C{
public:
    void A9(Data_Store *ds);
};
class A9_3 : public A9C{
public:
    void A9(Data_Store *ds);
};
/* A10 */
class A10C{
public:
    // make deposit with penalty
    virtual void A10(Data_Store *ds)=0;
};
class A10_1 : public A10C{
public:
    void A10(Data_Store *ds);
};
```

```
class A10_2 : public A10C{
public:
    void A10(Data_Store *ds);
};
class A10_3 : public A10C{
public:
    void A10(Data_Store *ds);
};
/* A11 */
class A11C{
public:
    // withdraw no penalty
    virtual void A11(Data_Store *ds)=0;
};
class A11_1 : public A11C{
public:
    void A11(Data_Store *ds);
};
class A11_2 : public A11C{
public:
    void A11(Data_Store *ds);
};
class A11_3 : public A11C{
public:
    void A11(Data_Store *ds);
};
/* A12 */
class A12C{
public:
    // withdraw with penalty
    virtual void A12(Data_Store *ds)=0;
};
class A12_1 : public A12C{
public:
    void A12(Data_Store *ds);
};
class A12_2 : public A12C{
public:
    void A12(Data_Store *ds);
};
class A12_3 : public A12C{
public:
    void A12(Data_Store *ds);
};
/* A13 */
class A13C{
public:
    // lock
    virtual void A13(Data_Store *ds)=0;
};
class A13_1 : public A13C{
public:
    void A13(Data_Store *ds);
};
class A13_2 : public A13C{
public:
    void A13(Data_Store *ds);
};
class A13_3 : public A13C{
public:
    void A13(Data_Store *ds);
};
/* A14 */
class A14C{
public:
    // unlock
    virtual void A14(Data_Store *ds)=0;
};
class A14_1 : public A14C{
public:
    void A14(Data_Store *ds);
};
class A14_2 : public A14C{
public:
    void A14(Data_Store *ds);
};
class A14_3 : public A14C{
```

```
public:
    void A14(Data_Store *ds);
};
// abstract factory pattern -- attribute for operations defined by concrete factory
class Abstract_Factory{
public:
    virtual A1C *create_a1()=0;
    virtual A2C *create_a2()=0;
    virtual A3C *create_a3()=0;
    virtual A4C *create_a4()=0;
    virtual A5C *create_a5()=0;
    virtual A6C *create_a6()=0;
    virtual A7C *create_a7()=0;
    virtual A8C *create_a8()=0;
    virtual A9C *create_a9()=0;
    virtual A10C *create_a10()=0;
    virtual A11C *create_a11()=0;
    virtual A12C *create_a12()=0;
    virtual A13C *create_a13()=0;
    virtual A14C *create_a14()=0;
};
class Concrete_Factory_1 : public Abstract_Factory{
public:
     A1C *create_a1();
     A2C *create_a2();
     A3C *create_a3();
     A4C *create_a4();
     A5C *create_a5();
     A6C *create_a6();
     A7C *create_a7();
     A8C *create_a8();
     A9C *create_a9();
     A10C *create_a10();
     A11C *create_a11();
     A12C *create_a12();
     A13C *create_a13();
     A14C *create_a14();
};
class Concrete_Factory_2 : public Abstract_Factory{
public:
     A1C *create_a1();
     A2C *create_a2();
     A3C *create_a3();
     A4C *create_a4();
     A5C *create_a5();
     A6C *create_a6();
     A7C *create_a7();
     A8C *create_a8();
     A9C *create_a9();
     A10C *create_a10();
     A11C *create_a11();
     A12C *create_a12();
     A13C *create_a13();
     A14C *create_a14();
};
class Concrete_Factory_3 : public Abstract_Factory{
public:
     A1C *create_a1();
     A2C *create_a2();
     A3C *create_a3();
     A4C *create_a4();
     A5C *create_a5();
     A6C *create_a6();
     A7C *create_a7();
     A8C *create_a8();
     A9C *create_a9();
   A10C *create_a10();
     A11C *create_a11();
     A12C *create_a12();
     A13C *create_a13();
     A14C *create_a14();
};
// stragegy pattern -- operations replacable by changing attribute
class OP{
public:
    Data_Store *ds;
    A1C *p1;
    A2C *p2;
    A3C *p3;
    A4C *p4;
    A5C *p5;
    A6C *p6;
    A7C *p7;
    A8C *p8;
    A9C *p9;
    A10C *p10;
    A11C *p11;
    A12C *p12;
    A13C *p13;
    A14C *p14;
    void A1();
    void A2();
    void A3();
    void A4();
    void A5();
    void A6();
    void A7();
    void A8();
    void A9();
    void A10();
    void A11();
    void A12();
    void A13();
    void A14();
    void init(Abstract_Factory *af);
};

#endif
```

Parser.cpp:

```
#include "Parser.h"
using namespace std;
void Parser_1::start_atm(){
    ATM1 *atm1 = new ATM1();
    atm1->create();
    cout << "ATM-1 created"<<endl;

    bool quit_flag = true; // quit when false, otherwise loop for input
    int x;
    int temp_int;
    string temp_str;
    do{
        cout << "\tMenu of Operations\n\
    1.card(int,string)\n\
    2.pin(string)\n\
    3.deposit(int)\n\
    4.withdraw(int)\n\
    5.balance()\n\
    6.lock(string)\n\
    7.unlock(string)\n\
    8.exit()\n\
    9.quit the demo\n";
        cout <<"Select Operation : \n>";
        cin >> x;
        switch(x){
        case 1:
            cout<<"selected : card(int x,string y)\ndefine balance x :\n>";
            cin.clear();
     cin.ignore(numeric_limits<streamsize>::max(), '\n');
            while (!(cin >> temp_int)){
                cout << " input int wrong"<<endl;
                cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout<<"define balance x :\n>";
            };
            cout << "define pin y:\n>";
            cin.clear();
     cin.ignore(numeric_limits<streamsize>::max(), '\n');
            getline(cin,temp_str);
            atm1->card(temp_int,temp_str);
            quit_flag = true;
            break;
        case 2:
            cout<<"selected : pin(string x)\ninput pin x :\n>";
            cin.clear();
```

```
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
              getline(cin,temp_str);
              atm1->pin(temp_str);
              quit_flag = true;
              break;
         case 3:
              cout<<"selected : deposit(int d)\ndefine amount d :\n>";
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
      while (!(cin >> temp_int)){
                  cout << " input int wrong"<<endl;
                  cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         cout<<"define amount d :\n>";
              };
      atm1->deposit(temp_int);
              quit_flag = true;
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
              break;
         case 4:
              cout<<"selected : withdraw(int w)\ndefine amount w :\n>";
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
      while (!(cin >> temp_int)){
                  cout << " input int wrong"<<endl;
                  cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         cout<<"define amount w :\n>";
              };
      atm1->withdraw(temp_int);
              quit_flag = true;
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
              break;
         case 5:
              cout<<"selected : balance()\n";
              atm1->balance();
              quit_flag = true;
              break;
         case 6:
              cout<<"selected : lock(string x)\nprovide pin :\n>";
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
              getline(cin,temp_str);
              atm1->lock(temp_str);
              quit_flag = true;
              break;
         case 7:
              cout<<"selected : unlock(string x)\nprovide pin :\n>";
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
              getline(cin,temp_str);
              atm1->unlock(temp_str);
              quit_flag = true;
              break;
         case 8:
              cout<<"selected : exit()\n>";
              atm1->exit();
              quit_flag = true;
              break;
         case 9:
              cout << "quit demo"<<endl;
              quit_flag = false;
              break;
         default :
              cout << "input error"<<endl;
              quit_flag = true;
              cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
              break;
         }
     }while(quit_flag);
}
void Parser_2::start_atm(){
    ATM2 *atm2 = new ATM2();
    atm2->create();
```

```
cout << "ATM-2 created"<<endl;

bool quit_flag = true; // quit when false, otherwise loop for input
int x;
int temp_int;
float temp_flt;
do{
    cout << "\tMenu of Operations\n\
1.CARD(float,int)\n\
2.PIN(int)\n\
3.DEPOSIT(float)\n\
4.WITHDRAW(float)\n\
5.BALANCE()\n\
6.EXIT()\n\
7.quit the demo\n";
    cout <<"Select Operation : \n>";
    cin >> x;
    switch(x){
    case 1:
        cout<<"selected : CARD(float x,int y)\ndefine balance x :\n>";
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
 while (!(cin >> temp_flt)){
            cout << " input float wrong"<<endl;
            cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout<<"define balance x :\n>";
        };
        cout << "define pin y:\n>";
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
 while (!(cin >> temp_int)){
            cout << " input int wrong"<<endl;
            cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout<<"define pin y :\n>";
        };
        atm2->CARD(temp_flt,temp_int);
        quit_flag = true;
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
        break;
    case 2:
        cout<<"selected : PIN(int x)\ninput pin x :\n>";
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
 while (!(cin >> temp_int)){
            cout << " input int wrong"<<endl;
            cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout<<"define pin y :\n>";
        };
        atm2->PIN(temp_int);
        quit_flag = true;
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
        break;
    case 3:
        cout<<"selected : DEPOSIT(float d)\ndefine amount d :\n>";
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
 while (!(cin >> temp_flt)){
            cout << " input float wrong"<<endl;
            cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout<<"define pin y :\n>";
        };
 atm2->DEPOSIT(temp_flt);
        quit_flag = true;
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
        break;
    case 4:
        cout<<"selected : WITHDRAW(float w)\ndefine amount w :\n>";
        cin.clear();
 cin.ignore(numeric_limits<streamsize>::max(), '\n');
 while (!(cin >> temp_flt)){
```

```
                cout << " input float wrong"<<endl;
                cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"define pin y :\n>";
                };
         atm2->WITHDRAW(temp_flt);
                quit_flag = true;
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            case 5:
                cout<<"selected : BALANCE()\n";
                atm2->BALANCE();
                quit_flag = true;
                break;
            case 6:
                cout<<"selected : EXIT()\n>";
                atm2->EXIT();
                quit_flag = true;
                break;
            case 7:
                cout << "quit demo"<<endl;
                quit_flag = false;
                break;
            default :
                cout << "input error"<<endl;
                quit_flag = true;
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            }
        }while(quit_flag);
}
void Parser_3::start_atm(){
        ATM3 *atm3 = new ATM3();
        atm3->create();
        cout << "ATM-3 created"<<endl;

        bool quit_flag = true; // quit when false, otherwise loop for input
        int x;
        int temp_int1;
        int temp_int2;
        do{
            cout << "\tMenu of Operations\n\
        1.card(int,int)\n\
        2.pin(int)\n\
        3.deposit(int)\n\
        4.withdraw(int)\n\
        5.balance()\n\
        6.lock(int)\n\
        7.unlock(int)\n\
        8.exit()\n\
        9.quit the demo\n";
            cout <<"Select Operation : \n>";
            cin >> x;
            switch(x){
            case 1:
                cout<<"selected : card(int x,int y)\ndefine balance x :\n>";
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
                while (!(cin >> temp_int1)){
                    cout << " input int wrong"<<endl;
                    cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"define balance x :\n>";
                };
                cout << "define pin y:\n>";
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         while (!(cin >> temp_int2)){
                    cout << " input int wrong"<<endl;
                    cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"define pin y :\n>";
                };
                atm3->card(temp_int1,temp_int2);
                quit_flag = true;
                cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            case 2:
                cout<<"selected : pin(int x)\ninput pin x :\n>";
                cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
         while (!(cin >> temp_int1)){
                    cout << " input int wrong"<<endl;
                    cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"input pin x :\n>";
                };
                atm3->pin(temp_int1);
                quit_flag = true;
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            case 3:
                cout<<"selected : deposit(int d)\ndefine amount d :\n>";
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         while (!(cin >> temp_int1)){
                    cout << " input int wrong"<<endl;
                    cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"define amount d :\n>";
                };
         atm3->deposit(temp_int1);
                quit_flag = true;
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            case 4:
                cout<<"selected : withdraw(int w)\ndefine amount w :\n>";
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         while (!(cin >> temp_int1)){
                    cout << " input int wrong"<<endl;
                    cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"define amount w :\n>";
                };
         atm3->withdraw(temp_int1);
                quit_flag = true;
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
                break;
            case 5:
                cout<<"selected : balance()\n";
                atm3->balance();
                quit_flag = true;
                break;
            case 6:
                cout<<"selected : lock()\n";
                atm3->lock();
                quit_flag = true;
                break;
            case 7:
                cout<<"selected : unlock()\n";
                atm3->unlock();
                quit_flag = true;
                break;
            case 8:
                cout<<"selected : exit()\n>";
                atm3->exit();
                quit_flag = true;
                break;
            case 9:
                cout << "quit demo"<<endl;
                quit_flag = false;
                break;
            default :
                cout << "input error"<<endl;
                quit_flag = true;
                cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
            break;
        }
    }while(quit_flag);
}
```
Parser.h:
```
#ifndef _PARSER_H
#define _PARSER_H
#include <iostream>
#include <limits>
#include "ATM.h"
class Parser{
public:
    virtual void start_atm()=0;
};
class Parser_1 : public Parser{
    void start_atm();
};
class Parser_2 : public Parser{
    void start_atm();
};
class Parser_3 : public Parser{
    void start_atm();
};
#endif
```
main.cpp:
```
#include <iostream>
#include <stdio.h>
#include <limits>
#include "Parser.h"

using namespace std;
/* run this program using the console pauser or add your own getch, system("pause") or
input loop */
void welcome_prompt();
int main(int argc, char** argv) {
    welcome_prompt();
    return 0;
}
void welcome_prompt(){
    cout << "     CS586 14F Project"<<endl;
    cout << "       MDA-EFSM-ATM  "<<endl;
    cout << "-----------------------------\n\n\n\n"<<endl;
    int i;
    bool retry_flag = false;
    Parser *parser;
    do{
        cout << "select ATM-1, ATM-2, or ATM-3"<<endl;
        cout << " 1 = ATM-1\n 2 = ATM-2\n 3 = ATM-3"<<endl;
        cout << ">";
        i = 0;
        cin >> i;
        switch(i){
        case 1 :
            cout << "you selected ATM-1"<<endl;
            retry_flag = false;
            parser = new Parser_1();
            break;
        case 2 :
            cout << "you selected ATM-2"<<endl;
            retry_flag = false;
            parser = new Parser_2();
            break;
        case 3 :
            cout << "you selected ATM-3"<<endl;
            retry_flag = false;
            parser = new Parser_3();
            break;
        default :
            cout << "input error"<<endl;
            retry_flag = true;
            cin.clear();
     cin.ignore(numeric_limits<streamsize>::max(), '\n');
            break;
        }
    }while(retry_flag);
    parser->start_atm();
}
```